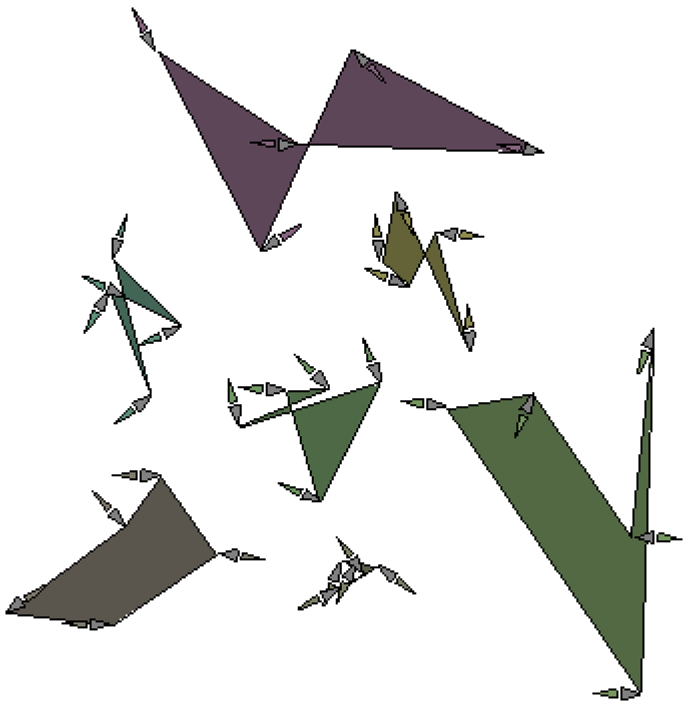
Evolutionary design of game vehicles and their controllers



Samuel Roberts

A thesis submitted for the degree of Doctor of Philosophy

Department of Computer Science and Electronic Engineering

University of Essex

September 2015

Abstract

Procedural content generation (PCG) is a growing field of interest in the domain of computational intelligence as it relates to games. There are ever increasing examples and applications of PCG that have been studied in academic contexts. Player expectations of the amount of content in games increase as computers and video game consoles are capable of using more content, and automation of content creation becomes more desirable. While many means of procedural content generation using some form of search algorithm have been tried and tested, we examine evolutionary algorithms as a means to generate content, where it has not frequently been used before.

We examine the generation of vehicles, specifically spaceships, within two dimensional game simulations. These simulations are based upon a simple Newtonian physics system with different physical rules, representing games such as *Lunar Lander* or *Asteroids*, and evolve linear vectors of real numbers that act as vehicle genotypes by encoding placement of components to a vehicle point mass, with a form defined by the placement of each component. We use simple 1-ply lookahead controllers, simple rule-based controllers, and MCTS-based controllers as means to test and therefore indirectly guide the evolution of vehicle designs.

We are able to demonstrate that evolutionary algorithms can be used to generate effective vehicle designs, suitable for use by the same controller as used for testing, for simple tasks without much issue. We also show that there are some factors of a problem environment that impact the demands and the conditions affecting vehicle design evolution more than others, such as velocity loss factors and the topology of the game world used. It is also evident that the use of different controllers to test vehicles causes different designs to emerge based on the strengths of said controllers.

Acknowledgements

The author would like to thank Simon Lucas, for being an invaluable and patient advisor, as well as a source of insights and guidance. Thanks also to the members of the Game Intelligence Group over the years, for their contributions to the author's understanding and their support and suggestions throughout the entirety of research.

The author would also like to thank his friends. With all of your incredible advice, thought-provoking questions, and simply being there to talk to, you've all helped in the process of putting this work together in many different ways. Thank you to all of you. It has been a privilege to meet you all, no matter where in the world. There are too many names for me to list here, but I appreciate you all.

Last but not least, the author would like to thank his family for their support over the past years. A heartfelt thank you to all of you, your kind words, your patience and your guidance that all helped me to come this far.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Games are a fascinating test-bed for artificial intelligence research, as they provide an easily controlled simulated environment and a goal for agents within to achieve.

Evolutionary algorithms are an effective way to provide answers to problems of numerical optimisation, and many problems that do not directly relate to numerical optimisation can be rephrased in a way that can be optimised. While design can be a highly subjective process at times, effective designs in respect to a specific, objective goal can be shown to be effective.

Through the course of research discussed, the feasibility of evolutionary algorithms as a means to design vehicles within a game context is examined. In addition, the differences between the evolution of a design alone to co-evolved design and controller parameters are presented and discussed.

The problem of generating entities within a game and the problem of generating controllers for entities within a game are frequently examined as separate problems. However, the simultaneous generation of both together suggest a potential for more unique behaviours emerging from the interaction between the physical composition of an entity and its capacity for action and reaction within the game context.

Focusing on the generation of both interesting vehicle designs and interesting ship behaviours, adaptation is required for both designs and controllers to work together to produce interesting behaviours to the extent of which a vehicle is capable within a simulation context.

By evolving in tandem, the capacity exists for the design of a vehicle and its controller to evolve to fit one another, with the potential for the strengths of both to compensate for any weaknesses, such as a smarter controller taking advantage of an otherwise less capable design, or a very well designed vehicle being easier to control.

The objective of this research is to examine the generation of two-dimensional game vehicles, and how control strategies and environmental parameters affect the resulting designs.

1.1.1 Motivation

Why evolutionary?

Evolutionary methods have been thoroughly demonstrated to be highly effective at solving a wide variety of optimisation problems. The problem of design in many situations falls to one of finding optimal parameters for some base solution. Evolutionary algorithms provide an excellent means of tuning and tweaking values to improve upon this base solution, and this 'base solution' needs to be nothing more than a framework of values that can be changed and adapted to a problem context.

The strength of evolutionary algorithm approaches to problems of numerical optimisation is such that this iterative strategy of solving a problem does not need to have contextual awareness of the problem being considered. The only inputs and outputs required by the solving strategy are the parameters generated by the evolutionary process, and a value relating to the suitability of a solution to the problem it is applied to – its evolutionary fitness.

By reducing all characteristics of a problem to a single value, certain aspects of problem solving become much simpler, while others, such as conflicting and competing goals, become more troublesome. Through the course of research, there were situations where an approach of using a single fitness value to solve design-based problems was advantageous, and there were also situations where the complexity of a design problem lead to an unclear fitness landscape.

Previous use of evolutionary methods for design problems similar to the problems explored within this research have tended more towards neuroevolutionary methods such as composite pattern producing networks. Evolutionary algorithms see less use for procedural content generation of this nature and are more frequently used for applications such as for level design and generation. They have not seen any use for the generation of game agents and their controllers. While this may be simply because other methods are more suited, it is worth understanding whether optimisation methods are intrinsically ill-suited for the problem of content generation or if they are simply underutilised.

Why design?

Procedural content generation is a field of growing importance and relevance within the video games industry. As technology improves, there becomes a greater capacity for content within games, and as this capacity grows, the expectation of more content from video game consumers scales with it. The increasing demand for content leads to increasing and costly requirement for more staff, more hours, and the logistical burdens that come with both of these. As a result, PCG is a means of offloading some of this demand, through offline tools used to create content for games during development and online means to adjust and create content on the fly tailored to individual players.

Within the field of procedural content generation, many efforts have been made to generate a wide variety of content within different contexts and different varities of games. Static content such as level designs, generative artwork and sound and other such resources have been investigated thoroughly with a strong diversity of results and approaches, both in academic and commercial circles.

The behaviour of non-player entities within a game can also be considered a form of content. Various methods to imbue the entities within a game with interesting behaviours exist, from simple and endlessly repeating patterns, to nested state machines and variations thereof as well as goal-based planning systems. These different control strategies, or controllers, are suited entirely based on the nature of the agents and their desired capabilities within games. More sophisticated controllers can potentially lead to more pleasantly surprising and generally interesting behaviours in comparison to game entities that make repetitive or completely random actions. However, more sophisticated controllers tend to require more design.

Techniques such as composition pattern-producing neural networks have been used within the context of procedural content generation to good result, and ample literature exists showcasing how CPPNs can be used to generate content for a game as well as controllers for said content. It could be argued that pattern-producing algorithms such as these are a more natural fit in some situations based on the type of content being generated.

Why games?

Games have frequently been a useful testbed for artificial intelligence, as their very nature demands the existence of a goal and interesting obstacles that prevent a player or agent from accomplishing that goal. The ease to which a game's environment can be modified and controlled in comparison to real world testing environments also make them a useful context to study, and this ability to modify and control these artificial environments also becomes a factor worth investigating in itself.

Additionally, as previously mentioned, games have a commercial relevance that is hard to ignore as a growing industry. Techniques that can be tested in reference to games can also be applied to games as well, and methods of automated generation of content and behaviours are a boon as previously mentioned. A means by which to both generate and test agents within a game by evaluating their performance could also be considered a useful asset given the requirements and amount of time and cost to testing game elements.

Why vehicles?

The concept of arranging predefined components into a connected structure allows for an easier design task that allows intuitive dimensions of change for said components, such as position and orientiation for the components and the size and mass of a vehicle being determined by how distant these components are from the centre. These vehicles could also be considered spaceships in the game contexts, except for those situations where the game context does not accurately model the physical properties of a vacuum. Some contexts make use of a quasi-friction property referred to as a velocity loss factor, for example.

As an aside, the design of vehicles in modular structures is, at time of writing, an ever-increasingly popular feature of games, with recent examples such as *Scraps*[1] and *TerraTech*[2]. The concept in particular of modular spaceship construction is much older and has been a feature of multiple space-themed strategy games, such as in *Master of Orion*[3]. While it isn't claimed that the vehicle composition focused on in this research is similar to any existing commercial examples, this does demonstrate that the choice of vehicles as entities to design is relevant to existing games.

Why controllers?

While evolving designs alone with reference to a fixed controller that does not change based on design is part of the research presented, the evolution of controllers as an additional feature of evolutionary design was a fascinating topic within its own right and led itself naturally to inclusion as the course of experimentation continued. One of the aims of the research was to evolve the designs along with their controllers, to ideally lead to a whole agent greater than the sum of its evolved composition within the game physics and the evolved weighting of its simple lookahead controller.

In addition, while the problems being examined were perhaps not ideal test problems to use as a means of benchmarking the control algorithms used, the comparison of simpler and faster controllers to more computationally expensive and slower controllers both in terms of effectiveness and in terms of diversity and interestingness of behaviour was also an interesting aspect to consider as a direct consequence of comparing evolutionarily refined controllers to general control methods.

[1]Moment Studio, 2015
[2]Payload Studios, 2015
[3]Microprose, 1993

1.1.2 Vehicles and Spaceships

Throughout this thesis, the terms *spaceship*, *ship* and *vehicle* are used interchangeably. As the focus of the thesis is on the evolution of two-dimensional game vehicles within an *Asteroids* styled game context, it only becomes natural to refer to the vehicles being evolved as spaceships or ships. While the actual entities are a more abstract configuration of components in a two-dimensional context, and are more accurately described as vehicles, understand both these terms to relate to the exact same thing.

1.2 Research Objectives

The objectives of this research are as follows.

- To test whether evolutionary strategies can be used to generate effective game vehicle designs within game-related contexts

The primary aim of the research is simply to establish whether the chosen technique of generating designs is in fact suitable for the nature of the problem. There are issues inherent with compressing the multidimensional aspect of design problems into a single formula producing a single value which only increase in scope and complexity as further dimensions are considered.

The effectiveness of evolutionary algorithms as a problem solving method increases with better phrasings of fitness functions and can decrease as dimensions of problems increase or additional conflicting concerns arise. Theoretically, there comes a point where this particular evolutionary approach begins to fail due to the complexity of the problems being considered, and part of this research consists of understanding where this situation emerges and the possible reasons for its emergence.

As discussed, game contexts are a useful simulation context to consider as they naturally lead to a specific goal to accomplish which can be reinforced with additional secondary goals if required. The approaches taken for this problem are specifically only effective for game vehicles due to the physics involved. While the physics examined in the context simulate a reasonably intuitive two dimensional physics in all examined contexts, they are far from realistic outside of game contexts.

- To investigate whether evolving the parameters of reactive controllers in conjunction with the parameters of a design leads to improved behaviours

During the course of research, certain reactive controllers were developed that used weightings based on a number of conditions relevant to the context of the experiment to influence chosen actions and resulting behaviours. More details are given in chapters 4 to 6, but these underlying principles are similar across all implementations of the reactive controllers used within those chapters. In addition to the composition of a vehicle, the weightings of these controllers are bundled into the same chromosome and iteratively improved along with the physical representation of the vehicle agent.

Theoretically, this merging of the representation of both design and controller should allow for the emergence of designs and controllers evolved to work together in tandem to accomplish goals more effectively than unrelated generic controllers. Of course, whether this actually holds true is something to be explored in later chapters.

- To explore how different controllers of varying levels of sophistication and complexity can affect the designs and behaviours produced by evolutionary design of game vehicles.

The sophistication of different controllers in particular can range from simple reactive controllers that do not use any form of planning, to one-ply lookahead controllers that explore only the most immediate potential actions within the state-space of a problem, to controllers that look ahead further using random rollouts to sample potential outcomes and futures of actions.

Different approaches to problem solving are taken by different control methods for self-motivated game entities. While only reactive controllers have been considered for the evolutionary aspects of controller improvement, the state-space search algorithms and heuristics used for other methods of control grant these alternate methods increasing capacity to control a wider variety of designs to greater effect. It stands to reason that different controllers with different approaches to problem solving will lead to different behaviours, and examples of this can be more clearly observed in chapter 6, but the reasons why different controllers act in different ways themselves are another facet of this course of research.

- To see if *interesting behaviours* can be observed as a consequence of the interaction between controller and vehicle.

While different controllers for the same designs would be expected to have different levels of effectiveness with respect to solving a problem, it is less simple to predict the uniqueness or novelty of interactions between controllers and designs.

Interesting behaviours in this context is defined to mean unexpected but not maladaptive behaviours that have some level of novelty to them, and the exact nature of this quality is difficult to adequately define. In some respects, this objective is exploring the emergence of new unexpected behaviours as a result of the combination of a vehicle's controller, environment and goals in such a way that it would be worth commenting on said behaviours.

1.3 Contributions

This thesis focuses on the generation of agents through evolutionary algorithms in a method that has not been attempted previously, and especially not with the additional evolution of controller parameters. The evolution of both of these features within a single chromosome allows for the reinforcement of controller behaviour to vehicle composition and capability, and the strengths and weaknesses of this approach are deeply explored through the course of this work.

While evolutionary algorithms have been used to generate other forms of content, they have not been used to generate both the design and controller of self-motivated game vehicles in this manner. Rephrasing a problem of design variables to a problem based upon the performance of a design within a context is not a new idea, but this particular problem of agent generation in this context has not been attempted before with evolutionary algorithms.

As evolutionary algorithms are known to be strong solutions to the problem of numerical optimisation, the interesting part of this lies in how this design problem can be rephrased. Using a novel chromosomal representation of values that are scaled in different ways, explained in detail in chapter 3 and onwards, this problem of design has been adapted to fit known good approaches to optimisation problems.

Additionally, the relationship between the evolution of a controller and a design and the resulting impact on game vehicle behaviours is also examined, as well as the relationship of both of these to the surrounding environment based upon the parameters of the environment. These combinations of factors, as well as what is permitted and what is not permitted by the constraints of the design problem for consideration, lead to interesting behaviours that are both expected and unexpected. These novel behaviours could lead to a more entertaining experience for the player of a hypothetical game similar in composition to the game contexts considered.

1.4 Publications

This thesis covers work that has been previously covered in existing publications co-written by the same author.

Chapter 3 covers work that has been detailed in the paper *Evolving spaceship designs for optimal control and the emergence of interesting behaviour* [68] relating to the use of evolutionary algorithms to generate game vehicles for navigational tasks, and explores the concepts discussed in the paper in a little further detail than what has been published.

Chapter 4 discusses experimental outcomes to problem contexts based upon the game *Lunar Lander*, and these findings were previously reported in the paper *Measuring interestingness of continuous game problems* [69]. Both this paper and the previous paper have been presented by the author at the IEEE Computational Intelligence in Games conference, for the 2012 and 2013 conferences respectively.

The game physics and scenario established within chapter 4 were also used as the problem context for the paper *Rolling Horizon methods for Games with Continuous States and Actions* [71], of which the author of this thesis co-authored the paper as it related to this scenario. The primary investigation and findings were not contributed by this author.

1.5 Organization

A brief overview of the contents of the thesis is as follows, with each chapter summarised in regards to its content and how it relates to the overarching problem described previously. The chapters are organised in a chronological progression as each new finding from one set of experiments directly informed the structure and objectives of the succeeding research.

1.5.1 Literature Review

Chapter 2 does not discuss direct experimental evidence but is instead a brief review of relevant literature. Summaries of procedural content generation and relevant evolutionary computation methods are presented to give context to this work and of how it relates, directly and indirectly, to what has been accomplished previously.

In particular, chapter 2 examines specific techniques for generating content procedurally, such as through means like genetic programming, evolution of neural networks, and less frequent but equally valid techniques relating to more specific types of content such as answer-set programming and noise-based algorithms. Previous examples of generated agents are examined and compared to the approaches taken here. Previous applications of similar evolutionary approaches to problems are considered to place these findings in context in a broader scope. Finally, some consideration of control strategies and problem solving methods for agents within a game context is given.

1.5.2 Vehicle Generation for Navigational Tasks

Initial experiments and initial proof of concepts are presented and discussed in chapter 3, along with the encouraging preliminary results and findings resulting from them. These experiments investigate the evolution of designs within an *Asteroids* inspired environment, and do not focus on the evolution of controllers. The initial experiments are shown to have promising results as vehicles are generated for simple navigational problems with success. The results of this chapter lead to the foundation of the chromosomal representation of vehicles that, although modified in subsequent experiments, remains the base for all future experiments presented.

1.5.3 Vehicle Generation Under Varying Problem Definition

Chapter 4 focuses on subsequent work investigating a scenario inspired by *Lunar Lander*, and the successes and unexpected difficulties that came about as a direct result. Moving to a more complex scenario and physics after the successes of previous experiments, it is discovered that what was assumed to be a relatively trivial scenario is deceptively complex due to the addition of increasing demands and pressures from the game physics.

After little success is had in evolving effective designs and a new approach to evolution of controllers via evolution of the actions themselves, the lens of the problem is adjusted to focus from the vehicles to the environments themselves. The evolution of the environmental parameters themselves give some indication as to which parameters of the environment have the greatest impacts on the problem difficulty, including properties such as the gravity of the simulation and values that affect the judgement of how successful a vehicle is at the end of the simulation.

1.5.4 Vehicle Competition, Co-evolution, and Controller Adjustments

Efforts to explore a more co-evolutionary scenario are detailed in chapter 5, as well as further examples of success and further difficulties due to an increasingly complicated situation. After the thorough examination of designs generated in a competition against environment, the question of how designs would fare with other designs as an evolutionary pressure arose.

Additional components are considered in addition to component types previously examined, allowing for vehicles to attack each other in a combat scenario using the simpler physics from the initial vehicle generation experiments. In an attempt to introduce new features of interest and evolutionary pressures, items are added to the scenario to benefit vehicles and to make the game context less of a toy example and more like a game that could potentially exist. After this change is made, the problem appears to become too complex for the design and control methods being used, and this is investigated through regression to a simpler scenario and comparison of the results to the behaviours seen in the more complex combat scenario. Some of the more interesting movement behaviours observed during these experiments are discussed.

1.5.5 Predator and Prey Scenario

Reviewing the difficulties faced in increasingly complicated scenarios, chapter 6 showcases a much simpler predator-prey scenario, allowing for competitive co-evolutionary development as intended previously but at a much more manageable level. Additionally, the use of a more sophisticated MCTS-based controller is trialled and compared to the simplistic but evolutionary methods used to control vehicles previously.

The issues and findings relating to the rephrasing of the problem as a coevolutionary problem are discussed here, as well as the differences between different coevolutionary approaches used. It is demonstrated that vehicles evolved for specialist goals perform better at their specialist roles than vehicles evolved to be generalists, and that the unequal playing field between a predator and prey lead to different evolutionary pressures and differences in designs and by extension behaviours as a result.

1.5.6 Conclusion

Concluding in chapter 7 are the overall thoughts and summary results of the course of research as a whole, and the findings relating to the relationship between controller and design, or "brain" and "body" of generated vehicles. Results are discussed with reference to all previous work as a whole, and the practicalities and problems encountered through the work as a whole are considered in light of the bigger picture. Future work that extends upon what has been discovered and accomplished is also discussed that could answer some of the questions raised and not answered by the experiments themselves.

Chapter 2

Literature Review

2.1 Overview

This thesis deals with the automated design and optimisation of vehicles for use in 2D space games. This is naturally a form of procedural content generation (PCG). However, there are many types of PCG and the particular focus in this thesis will be on search-based PCG. This is necessary since the design objectives are most easily stated in the form of an objective function.

Optimisation algorithms may then be used to find good solutions i.e. vehicles which satisfy the design goals for the problem at hand. Although there are many possible types of optimisation algorithm, we only briefly review the literature in this area since optimisation is not the main contribution of the thesis.

Many of the fitness functions developed are based on the "controllability" of a vehicle. Although some aspects of a vehicle's dynamics can be characterised by analysing it's Newtonian mechanics i.e. its ability to accelerate in a number of directions or rotate, it is not obvious what the optimal "Newtonian" profile would be for a given situation. Therefore, we also complement this analysis with the observed fitness when the vehicles is used by an agent or set of agents to perform a given set of tasks.

This naturally leads on to computational intelligence in games, or Game AI, albeit in a rather specific way. How can we best design agents to control vehicles in order to drive them in particular ways to achieve the given set of tasks? In particular within Game AI we look at MCTS as an effective means of controlling a vehicle, and within the context of this thesis in comparison to simpler methods of control.

The rest of this chapter is structured as follows. First, an overview of procedural content generation, and with in particular special consideration given to the methods of content generation used. Reference will be made to what content is produced, but of more relevance is the way in which this content is created, as there are clear emerging patterns over time as to the more frequently studied means of procedural content generation, particularly concerning neuroevolutionary techniques.

As specific forms of evolutionary computation have been used for the work conducted in this thesis, it is also key to examine these as well. CMA-ES is discussed, as well as coevolution of populations, previous applications, and the benefits that coevolution bring as well as some of the difficulties it can introduce.

Finally, the use of different controller strategies in games will be briefly discussed, with particular focus on MCTS, an effective general control algorithm for agents in games that has seen increasing modern usage.

2.2 Procedural Content Generation

Procedural content generation is listed as a notable key area of computational intelligence as applied to games by Yannakakis et al. [97], and with increasing research efforts into this field it can be successfully argued as a worthy field of investigation with increasing relevance both to modern academia and professional interests. Hendrikx et al. [30] further examine the relevance and scope of procedural content generation, examining previous games that make intensive use of procedural content generation techniques such as the space flight and trading game *Elite*, as well as the experimental game *.kkreiger*. Hendrikx et al. draw distinctions between different types of content that can be generated for games, such as static assets including images and sound, level designs for the space of the game, the scenario (or rules and constraints) of the game and, more relevant to the work in this thesis, the behaviours of entities within a game. Smelik et al. [78] examine the use of procedural generation as a tool to assist an artist or designer in creating terrain in the context of a virtual world. In their work, a designer is able to specify a coarse high level overview of terrain, with features such as mountains, rivers, roads and a city plan. Through procedural content generation, extra detail is added to these features such as the exact surface topology of the region, bridges over the rivers, and buildings placed in the planned city layout. Higher level examples such as *ANGELINA* [14] [15], software that can create entire games from simple concepts, further extend the space of what procedural content generation is capable of.

Togelius et al. [89] further discuss and overview different categories of search-based procedural content generation in particular. These categories are based on opposed pairs of traits, and of which evolutionary search is but one such trait. Examples of non-search-based procedural content generation are known to exist, such as generation of content based on answer set programming method [79]. However, it is clear from examining the literature that the main focus in recent years has been into search-based procedural content generation, typically evolutionary, focusing on the optimisation of a specified goal or combination of goals.

Of the categories mentioned by Togelius et al., the research focus of this thesis has been completely focused on offline generation, but typically on a timescale that could be used to produce content rapidly and that could be adapted for online generation. Vectors of numerical parameters are used in the stead of random seeds for the vehicle designs, although random seeds are used themselves to generate environments, an interesting distinction based upon desiring a more finely refinable spaceship design compared to the testing environment used.

Yannakakis and Togelius [96] discuss also, in addition to different forms of procedural content generation, the capacity to extend existing PCG methods through feedback from the player or user of the content generated, and label this as *Experience-Driven Procedural Content Generation*. They discuss means of gathering player opinion and modelling player experience and categorise it into four areas, with the fourth being a hybrid combination of the other three approaches. These areas are subjective, involving direct questions and surveys of player opinion, objective, involving recording player experience through direct means such as physiological or expressive changes in the player, and gameplay-based, involving gathering metrics of play.

Shaker et al. [76] use features of platform game levels as defined criteria to analyse for player reception and enjoyment. Feature analysis of possible levels is carried out through the use of a neural network, trained via genetic algorithm with a fitness measure based upon player preference. The focus is less on generation of content so much as analysis of existing content, although it ties into the experience driven aspect of EDPCG. This is an interesting use of techniques that could be argued to be neuroevolutionary, but requires manual tuning of the topology of the network, unlike examples of work that evolve the topology as well as the weights and thresholds of a neural network. Pedersen et al. [57] also examine features of levels, but for the purpose of generating them based upon similar concepts of EDPCG. Fitness for the levels generated in this work, for example, are based upon analysis of features associated with player emotion, such as fun, challenge, and frustration.

2.2.1 Genetic Algorithms

Although there are some fundamental differences between genetic algorithm techniques and genetic programming techniques, the two are very related in both being specific forms of evolutionary computation and share common terminology and behaviour.

Genetic algorithms work to adapt a linear genotype, typically a string of bits or a vector of real numbers, through operators such as point mutation and crossover. These operators affect the genotype at a specific location and can involve swapping around strings of values or modifying individual values.

Genetic programming techniques tend to have similar operators, but where genetic algorithms use operators that work on simple values, genetic programming operators can work on individual code elements of a genetic program or perhaps even block-level elements such as subroutines and complicated logic trees.

Genetic programming allows for more complex representations and therefore a more direct encoding of content that can be produced at the cost of an increasingly large search space and issues such as solution bloat, where solutions can become inefficient and contain effectively useless features.

There are many examples of genetic algorithms being used to generate content, especially for level generation. Liapis et al. [40] generate levels for use in a real time strategy game, with fitness based upon both the navigation between player bases with a preference for tactical areas such as choke points and initial territory size, placement of resources close to player bases and in strategically important locations, and visual impressiveness derived from symmetry and the balance of impassible areas to passable areas. This fitness is initially given as a weighted sum of subgoals, a characteristic that several works share [41] [69] [43] [40] [18] [46] [7] [75] [80]. While there are means for approaching multiple objective optimisation problems within a genetic algorithm context, it is frequently the case that multiple goals are condensed into a single fitness measure.

McGuinness and Ashlock [48] use genetic algorithms to both evolve two-dimensional tiles and their connections to one another. Tiles are evolved as chunks of passable and impassable terrain, with specific adjacency properties that allow connection to other tiles, and are evolved to meet these adjacency properties. Connection plans that assemble these tiles into interesting structures based on some desired criteria. These criteria can include ease of traversal, prioritisation of cul-de-sac structures, or symmetry. This multi-layered approach to generation is not unique, having been conceptually core to the generation of environments [68] or generators [34] that define the emergence of some desired content in an indirect manner. Later work by the same authors [2] would go on to examine the evolution of finite state automata for terrain creation. The generated terrain approximated the output of three dimensional functions; in particular hill, crater and parabolic ridge shaped functions. The rulesets for these automata are evolved through a simple genetic algorithm, as it is easy to express simple automata rulesets as a linear chromosome.

Future work by the same authors [3] would instead consider evolution of cell placement within a grid directly, for the purpose of two-dimensional ‘dungeon’ style levels, as opposed to indirectly through the use of cellular automata. Genetic algorithms are used in this later work on a two dimensional array, and as before with the substitution between runs of subtly different fitness measures to promote different level types. Linden et al. [92] examine level generation of this type, and survey and categorise various different methods, comparing cellular automata and genetic algorithm methods of designing levels. While it is not clear that any technique is inherently better, the results generated by different methods tend to produce qualitatively distinct results.

In an example of level generation that are not simply two-dimensional static maps, Ferreira and Toledo [16] use genetic algorithms to generate levels for the physics-based game *Angry Birds*. Levels are represented within the algorithm as an array of columns, with each element in a column a discrete element of a game level or a predefined compound block of such elements. Levels are optimised towards having stable two-dimensional physically modelled structures that a player can then attempt to dismantle in the course of play.

Meanwhile, Lanzi et al. [37] represent three-dimensional level maps in a first person perspective game as a vector of arenas and corridors, with arenas defining wide open spaces and a size and corridors defining connecting paths with some specific length. Level maps are evaluated based upon the collection of statistics of bot player matches taking place within these levels with the aim being to optimise balance. Levels with both bot players of equal skill having equal score are assumed to be more balanced and therefore fair and well constructed.

These methods of generation can be used also to generate level generators themselves [34] through use of an interactive genetic algorithm that receives direct input from a user, honing generators to desired characteristics with new generators produced through mutation and crossover. Work such as this would directly inspire later experiments with *Lunar Lander* [69] as covered in this thesis.

2.2.2 Genetic Programming

As mentioned, in addition to genetic algorithms, evolution of genetic programs is also used for procedural content generation. Most uses of genetic programming seem to pivot around the use of specialised grammars for the problem domain, which is typically either level design or the definition of rulesets for a game.

Frade et al. [18] examine the evolution of terrain generating programs for game terrain. They use genetic programming principles and a fitness measure based upon weighted subgoals relating to the accessibility and obstacle edge length of the generated terrain. The resulting generators produce a mathematical function defining a surface. This surface is the generated terrain to be used.

Browne and Maire [7] define a system named Ludi, which through use of a custom game description language is able to represent a variety of discrete grid-based turn-based games. A general game player using alpha-beta search with iterative deepening is used to test the games through playing them against itself. Game quality is evaluated on qualitative measures such as depth, clarity, drama (the ability for a losing player to recover) and decisiveness. These criteria are represented as a weighted sum, and evolved through a standard genetic algorithm approach. Hom and Marks [46] had earlier demonstrated a rule based board game generation system very similar to Ludi, where games are evaluated and evolved based upon a weighted sum of two goals, game balance and diversity - a form of novelty selection [38]. Both groups use very similar approaches, such as a design-grammar based generation system, although the later Ludi allows for different board cell adjacencies, such as allowing for grids of hexagonal cells.

Togelius and Schmidhuber [88] also look at creation of simple games from simple rulesets through evolution of rules and neural network controllers, trained with evolutionary algorithms and responding to visual fields around a game agent. The fitness measures for the controllers are based upon their agent's performance within a game, while the fitness measures for rules are instead based upon the performance of the controllers in an interesting case of coevolution. If controllers perform too well, it is assumed the game is too easy and the ruleset is penalised. If controllers perform poorly, it is assumed the game is too hard. Similar methodology was used in work discussed in this thesis [69] for determining the 'interestingness' of *Lunar Lander* game variants.

Shaker et al. [75] uses genetic programs phrased in a specific grammar to generate levels for the game *Super Mario Bros*, which are evaluated by the structures and patterns of levels that the generators produce. In future work [74] Shaker et al. examine level generation for a physics based game similar to *Cut the Rope*, based upon a design grammar. Fitness is based upon placement of elements to some specific constraints and effectively generates levels in a component based manner, a highly familiar approach at this juncture. There is also discussion of the capacity to use level generators such as this as tools for designers, especially the use of AI agents for testing and evaluating levels. This is similar to the testing of levels with pre-existing bots examined in work by Lanzi et al. [37].

2.2.3 Neuroevolutionary PCG

Of special note are related techniques all underneath the umbrella of evolutionary adapted neural networks, as these techniques have been investigated to a considerable extent. These are among the most popular methods of content generations due to the flexibility of representation and the enforced preservation of novelty inherent to the methods used.

Stanley and Miikkulainen [85] describe NEAT, or NeuroEvolution of Augmenting Topologies, a technique for evolving neural networks. This technique allows for a more sophisticated capacity for neural networks to evolve as it allows the topology of connections between neurons to evolve as well as weights, as opposed to using a topology where all neurons in each layer are connected to all neurons in the next layer, and simply evolving the weighting to prioritise some connections over others. The specific technique described brings some advantages over previous topological evolution techniques, such as starting with a minimal network to avoid unnecessary network bloat as early as possible, growth of the network to allow for development of a suitable network from a minimal starting point, marking the history of genetic changes to identify when networks with different encodings represent the same network phenotype, and through enforced speciation to prevent unwanted convergence towards local optima, as well as to protect novelty and innovation from being lost later into the run. This enforced speciation ensures that individuals of similar fitness compete among each other in individual niches as opposed to the greater population, thus preventing potential novel solutions from being lost later into the run and avoiding premature convergence to undesired local optima.

Stanley et al. [83] describe rtNEAT, “realtime NEAT”, as a variation of NEAT for problems where offline generation is less applicable. Instead of discrete generations, a process is repeated at regular time intervals within the simulation. During this process, individual fitnesses are evaluated and adjusted based on the number of existing individuals within the population, and the worst individual is replaced by the offspring of two individuals selected based on fitness proportion. While the introduction of rtNEAT focuses on its use as the AI for individual units within a real time strategy game, Olesen et al. [53] investigate its use for AI of the higher order players and opponents of a real time strategy game. rtNEAT is examined as a means to balance the level of challenge of an AI player to its opponent based on a challenge rating. This challenge rating itself is based on a number of game specific metrics.

Stanley [82] further describes CPPN, which several papers examined [82] [11] [12] [39] [43] [42] [27] [26] [41] use in conjunction with with NEAT to produce content. CPPNs, or Compositional Pattern Producing Networks, are neural networks that can encode certain patterns through use of gradient functions assigned to neurons that transform their inputs through their assigned function. Within the context of this paper the example provided and investigated is two dimensional imagery formed by NEAT-evolved networks, although as stated before other works use CPPN to generate content through this pattern generation based network representation.

There are numerous works that use CPPN-NEAT to generate content. Several examples use CPPN-NEAT in order to generate images [39] [41] [43], with the functions used in the neurons provided by CPPN being particularly suited to generation of imagery. The images generated are also interactively evolved with player input [96] based on preference and explicit subjective choice. Clune and Lipson [12] use CPPN-NEAT to interactively evolve content in a manner similar to the previous works, although for 3D geometry as opposed to 2D images. User input is used to determine fitness, and designs are evolved based on user preference. Also examined is the evolution of 3D objects to mimic an existing given input, to allow for multiple variations on a single thematic item. The ability to take existing content and synthesise similar but not identical variations is a particular strength of procedural content generation.

Hastings et al. [26] also describe an extension to NEAT described as cgNEAT, short for "content-generating NEAT". Following on from earlier research [28] of defining particle systems using NEAT, cgNEAT similarly defines particle systems for use in a game context. cgNEAT utilises CPPNs to represent items of content, and defines the fitness of an item based on player use of the item, making it a variety of interactive content generation where fitness is derived based on player input. Relating to discussion of experience driven procedural content generation [96], this would be an example of gameplay-based interactive fitness; metrics of play are used as a direct basis for the measure of a content item's fitness. While this is used in work by the same authors and refined in order to generate interesting and useful particle systems for weapons in a space-themed game [26] [27] [29], Pantaleev [56] discusses generation of skills in a role playing game, and how interactive PCG using metrics of use as a fitness measure can allow for skillsets that break away from traditional structures of skill sets common to role playing games.

HyperNEAT is also described by Stanley et al. [84] as an extension to NEAT, which uses the spatial patterns produced by CPPN as a means to indirectly encode an arbitrary amount of connections between an arbitary amount of neurons. In this way regularities in the physical space of the problem can be exploited by mapping the network to the qualities of a given problem. Clune et al. [11] use HyperNEAT to evolve gaits for quadruped creatures in work reminiscent of Sims' [77] work on evolving creatures with gaits, except with a fixed design and leg amount. The focus is not on the morphology of the agent but on the neural network controlling leg operation. Comparison is made between the performance of HyperNEAT evolution versus the evolution of a neural network with a fixed topology, and this work demonstrates that in the context of this gait evolution problem HyperNEAT is far more effective at evolving a satisfactory neural network than fixed topology evolution is. This work is further expanded upon [13] and described in additional detail, with comparisons to regular NEAT made as well as the fixed topology variant described. Still applying to the problem of gait control, HyperNEAT is shown to converge to a more effective solution within significantly fewer generations than regular NEAT, which in turn converges faster than NEAT with fixed topology. This only makes sense; by constraining NEAT to a fixed topology, the majority of benefits that NEAT allows are lost.

2.2.4 Other Methods

Methods other than some variant of neuroevolution, genetic algorithms or genetic programming have also been examined for their capacity to generate interesting content for game contexts. For example, Pachet [55] shows that music can be generated for games using multiple layered Markov models as a basis for musical phrases, with appropriate representation and biasing to allow for interactive generation of music. Conversely, Jordan et al. [33] use musical analysis transforms to generate levels for games.

Browne [6] shows that UCT, which will be briefly explained later when discussing MCTS, can be directly used for procedural content generation as well. It is demonstrated that a genetic algorithm given an arbitrary string of bits is likely to have a skewed distribution of possible bits visited early into its operation. When it is not known which bits are the most significant, this can present a problem. A variant of UCT described as UCG, or Upper Confidence for Graphs, is shown to explore all bits with uniform probability. The concept behind UCG is based upon representing bit-strings as graphs that can be traversed and nodes expanded, with values back-propagated. UCG is shown to match and exceed a standard bit-string genetic algorithm approach to a problem relating to arrangement of pentominoes in a finite space.

Cook and Colton [15] demonstrate that it is feasible to create entire games given theme/key words, through combination use of several data mining services and banks of premade assets freely available on the internet. While earlier versions of the software were designed to generate simple arcade games [14], the latest versions are capable of more complicated three-dimensional games. Their software, *ANGELINA*, creates levels with these selected assets starting with a critical path for the player from start to finish through the level and selecting for various criteria such as increased connectedness and sparser distribution for levels, and more complex rulesets that allow for more interesting game behaviours. The coevolution is cooperative in nature - four separate populations are used for each of the fitness metrics with a final solution derived from these populations. This is not the only work to use coevolutionary methods, and more examples will be discussed in the context of coevolution as a whole in the next section of this chapter. Similar software with similar goals and different methodology exists. Nelson et al. [50] demonstrate the generation of simple games with simple goals, also using external services to parse and comprehend given key words, and produce appropriate games based on a limited set of pre-existing game rules determined to be the most suitable for the verb supplied.

Non-Search-Based Procedural Content Generation

Answer-Set Programming In less directly search-based methods of generating content, work conducted by Smith and Mateas [79] uses answer set programming for content generation. A full definition of answer set programming is outside the scope of this chapter, but it can be described as a form of logic programming to produce satisfactory answers that meet certain specified logical conditions. Using a defined space of possible games and rules in a simple gridlike world with simple colored rectangular agents, game systems can be solved by tools that can provide answers to answer set programming problems. Notable is the absence of a fitness measure - if particular qualities are desired, specific high-level concepts such as winning a game through some specific action can be made part of the requirements for a valid solution.

Computational Geometry Olsen [54] demonstrates a means of generating terrain as used in a commercial game *Tribal Trouble*, which is based upon first combining the output of noise functions and Voronoi diagrams to form a heightmap and then running deterministic simulations of erosion upon the result to form natural appearing terrain. The initial stochastic elements are the noise functions and Voronoi diagrams used, and this is a very indirect form of content generation. While some parameters such as number of Voronoi regions can be made explicit, the majority of the process is affected mostly by a single seed number. Uriarte and Ontanon [90] also use Voronoi diagrams to generate terrain and level layouts for a real-time strategy game, generating regions for one quarter of the map and reflecting it to create the other three quarters. Generated maps are analysed using fitness functions, but are not evolved using these functions. These functions were used to measure how balanced the map is considered to be for its application, but comparison to statistics of games using generated maps showed no correlation between win rates and reported balance metrics. This outcome is similar to something that was encountered during the research for this thesis. Attempting to fit prescriptive measures to better generate content did not seem to produce results that were as effective as the metrics suggested.

Computational Narrative Grey and Bryson [21] examine designing agent behaviour through assigning memory to agents, and having these memories inform the decisions taken by agents based on previously established interactions and known qualities of game objects or other agents. In particular, a simple emotional model is used to influence agent interactions. As well as agent behaviours, in-game objectives can emerge organically from the interactions between agents and their own goals and opinions. Valls-Vargas et al. [91] also use narrative-based techniques to generate content, similar to other work [25]. Valls-Vargas et al. generate causal narratives to games, and while the exact nature of this narrative creation process is perhaps out of scope for consideration, the generated plot points are used to form a topology and then a level layout for a generated level with an inherent narrative or list of goals to accomplish for the player. Other narrative based approaches to content generation are numerous and vast, and largely beyond the scope of this review.

2.2.5 Generation of Dynamic Agents and NPCs

While generation of behaviour has been briefly mentioned [58], many efforts to generate dynamic agents can be found. The work of Sims [77] examines the evolution of component-based vehicles in a physical context, and encouraging behaviour from both the evolutionary design of a vehicle/agent and its controller to fit some aim, a key idea used in the research discussed in this thesis. Krčah [36] builds upon the work of Sims, focusing on reimplementation of the previous creatures to the previous designs and then on developing creatures for more complex behaviours. In the process, however, Krčah introduces new features unintentionally as a result of this reimplementation, allowing for unforeseen but beneficial features, an effect noticed in experiments with spaceship predator-prey scenarios as well as scenarios prior to the predator-prey experiments.

Remaining with the concept of generating agents from components with emergent behaviour from these components, Joachimczak et al. [32] examine the coevolution of both body and mind in the context of spring-based soft-body organisms in a 2D fluid environment. The implementation and simulation is a lot more biologically inspired, with a linear genome converted into a virtual organism by the modelling of cell division. Control is implicit by varying lengths of the springs between cells, as opposed to an evolved neural network or other means of control, in a manner similar to *Sodaconstructor*. In a commercial context, the idea of dynamic agent behaviour based upon the interaction between components is taken to a much deeper extreme in the game *Creatures* [20], featuring an extensively simulated biochemistry in addition to neural networks to drive agent behaviours.

While Mahlmann et al. [45] do not generate agents from components, they do attempt to generate sets of units for use in a strategy game using MCTS as a controller for the units. A noted unexpected consequence of the parameters allowed for units and the nature of the game lead to units that, rather than moving towards other units in the game environment, would evolve such that they had the ability to attack at a significantly larger and larger range, and would remain stationary and do nothing but attack. This is a problem that was also encountered in some of the scenarios covered in this thesis, and which will be discussed in more detail in the appropriate chapters.

Spaceship Agents

However, far more relevant to the work completed are examples of spaceship design and approaches that have been taken. Liapis et al. [42] have conducted work that focuses on the generation of spaceships through evolution of neural networks, discarding any unsuitable ships based on aesthetic considerations. The shape of a spaceship is constructed using neural networks to permute a circular set of points into a polygon, which is then horizontally mirrored. Ships are evaluated based on a number of criteria. As mentioned before, Liapis et al.'s work involves a representation that enforces a sense of 'front of ship' and 'back of ship'; any components are thrusters if at the back or turrets if at the front. This means there are ship designs that are reasonably effective in my work that cannot be represented in this system, and vice versa. The ships produced within this work are initially more aesthetically 'spaceships', however. Criteria such as polygon self-intersection, permitted in experiments conducted for this thesis, is explicitly not allowed in this work, while criteria such as components having a minimum distance between them are shared between both approaches.

It is worth noting that this work progresses in a divergent matter to the aims of generating spaceships with specific behaviours. Later work by Liapis et al. [43] focuses upon generating aesthetically pleasing spaceship designs. as opposed to ones designed to work effectively in a game scenario. The fitness measures used all select for aesthetic criteria, and user input also imposes weighting on the fitness of designs (causing some aesthetic although not very spaceship looking designs to emerge). Further work [41] focuses more on attempting to produce recognisably spaceship-like designs.

While not directly related to the generation of spaceship agents, there exist also efforts to improve or define behaviours of operation for vehicles in much the same way [65] [59], which will be focused on later.

2.2.6 Representation of Solution

Particular note must be made of the representation used to encode a solution. McGuinness [47] shows that representation choice in content generation has a significant impact on the nature of the content generated, using level generation as an example. The different representations examined allow for different spaces of content to be generated while simultaneously blocking certain types of content from being created within that representation. While the context is different in terms of the content being produced, both the nature of the genotype of a solution and the transformation or mapping process between genotype to phenotype can heavily impact any type of content being generated in such a manner.

As discussed before when discussing the grammars and sets of operations used in genetic programming, as well as the representations used for NEAT and its variants, the exact nature of content generated and solutions found to content generation problems can be the difference between a feasible approach or an infeasible one. Methods such as cgNEAT depend entirely on representation, while methods such as HyperNEAT attempt to directly exploit representation as a means to solving problems. Genetic programming approaches will frequently define a set of operators or grammar that is highly problem specific [7] [75].

As an example for variants of NEAT, work by Liapis et al. [42], the choice of representation within procedural content generation leads to certain spaces of designs being more frequent, or becoming much harder to express if not totally inexpressible. Within the context of Liapis et al.'s work, all evolved vehicles absolutely must have thrusters on one side and turrets on the other, imposing an unavoidable limitation on the space of spaceship designs that can be produced using their methods. This is in contrast to the representation used within this thesis, which allows for more flexibility of representation, although admittedly at the cost of expanding the problem space. Adrian et al. [1] also discuss the impact of representation within their work, as narrowing the space of representation in the context of level designs for their genetic algorithm is claimed to produce more interesting and enjoyable levels as a result.

Nygren et al. [52] examine the specific case of platform level generation as well, represent levels as directed graphs with each node representative of a feature of a platform game level, which is an effort to use the strength of feature-based content generation, as examined in other work [76] [75] [57], and then map the graph to a grid and select features for the grid based on connection and dimensions of discrete grid entities. This choice of graph representation, also used in other work [6], is similar to how cgNEAT and HyperNEAT operate as well, and graph representations of genotypes, while not as common as linear strings of bits or real numbers, are a reoccurring representation in the examined literature.

2.3 Controller Strategies

In order for vehicle designs generated throughout the many experiments this thesis covers to be tested, it is necessary to use some form of control strategy or algorithm to do so. In addition, it can be seen what forms of controller for these vehicles can provide better results. As has already been mentioned, the interaction between controller and vehicle design can be a very self-reinforcing one over the course of generations. Testing these vehicle agents allows for far more controlled and rapidly completed testing of the vehicle designs than any manual testing could accomplish.

By their very nature, agents require agency, and agency requires some form of decision making, planning, or other form of control process. Examining agents as a concept in themselves, there are some interesting examples particularly relevant to the problem of controlling vehicles in a physical situation.

In order to effectively control physical vehicles it is necessary to understand effective means of controlling entities that have been used in previous research. Reynolds [66] famously demonstrates that very simple rules can lead to highly effective behaviours for the purposes of vehicles that must move convincingly or in a physical simulation. These core ideas were used in multiple basic controllers in my work to allow for steering based on multiple simple conditions. Mingliang et al. [95] show that pursuing a moving target can be made more effective by pathfinding to where the target is expected to be based on its predicted trajectory. This idea, although not implementation, due to the difference in game topology, is used in state-based controllers for predator-prey scenario experiments, which will be discussed later.

As discussed earlier, there exist previous examples of efforts to generate controllers in a game context [58]. Watson et al. [93] also examine generation of controllers for agents in a game context, attempting to generate agent behaviours where all agents can perform more or less the same behaviours without much issue. The only changing factor is context (i.e. availability of nearby agents and their states) and the controllers themselves, being finite state machines evolved through standard operators (mutation, crossover). Previous work also examined the evolution of weighting of subgoals as part of a singular value function for a macro-action based greedy 1-ply controller [69].

Michel [49] has in previous work used neural networks to define brains for artificial life agents, but these neural networks are themselves defined by chromosomes that undergo a genetic algorithm and fitness evaluation to produce better controllers - better brains. These controllers are used to pilot a simulated robot in a simulated physical environment, a problem not unlike other problems of controlling vehicles within physical simulations [42] [43] [41] [60] [65] [59] [62]. Behaviours developed within the simulation were then transferred to an actual physical robot with only minor discrepancies between the behaviour of the simulated agent and the physical robot.

The challenges faced in transferring systems designed in simulation to real-world robots are similar challenges faced in application of theoretical approaches to practial problems in general. No simulation can be entirely accurate down to every facet of an intended domain environment, and the noise inherent in sensors and other physical components adds further complexity to adapting a system designed in simulation to a system designed for real-world application.

Principally, any assumptions of a forward model will likely not cover every feasible case of real-world application, as, at the time of writing, there does not exist a forward model for every feasible state of reality, making planning-based approaches more complex.

However, as demonstrated, these challenges are not insurmountable, and system designs that do not use a perfect forward model and operate in a noisy simulation environment can be designed with the robustness and error tolerance required in physical reality. Using genetic algorithms to iteratively improve neural networks is, as mentioned, one such approach to providing some level of error correction and noise tolerance.

Tan et al. [87] examine an interesting case requiring a generated controller for a vehicle agent. They use a genetic algorithm based controller can be used to tune behaviour towards some desired level of proficiency in realtime, not for optimal behaviour but instead competitive behaviour with a difficulty tailored to be challenging for the player. The context used is a simple tag game between a player and an agent controlled via evolved weighting of game-specific behaviours. Different behaviours are used in combination with one another with precedence controlled through evolution. Two approaches are taken, involving use of a single chromosome, and retaining of a more effective and less effective chromosome for behaviour adaptation.

2.3.1 MCTS

MCTS, or Monte Carlo Tree Search, is an algorithm that combines the precision of tree search with stochastic simulation to investigate combinatorial search spaces that deterministic tree search methods may not be able to satisfactorily explore within reasonable time frames. The general operation of the basic algorithm is to, for a given game state, create a search tree based on what actions are possible from that state and future states. Child nodes representing future states are selected through some strategy, expanded into nodes relating to possible future actions, and then simulated playouts assuming random action are projected from these possible future actions until a result is achieved. The information about this random simulation is then back-propagated through the search tree, updating the values of the nodes of the tree and updating what action from the root node, the current state, is the superior choice.

This method can be interrupted at any iteration and the best action as perceived at the time used, which means the balance between effectiveness and running time is simple to tune. The nature of this method also means that an explicit heuristic is not required for the problem it is applied to, simply knowledge of legal moves and end conditions, and with a suitable selection strategy the topology of the tree will adjust to explore and exploit more beneficial actions with less time and processing wasted in areas of the search space that provide less reward.

2.3.2 Selection in MCTS

Kocsis and Szepesvári [35] define UCT, Upper Confidence Bounds for Trees which is a selection strategy, a means of deciding what actions to take at a given state, used in many MCTS implementations and the selection strategy of choice for this thesis. The function of UCT is to apply UCB1 in a format suitable for tree search. UCB1, Upper Confidence Bounds 1, is an algorithm designed to investigate possible actions that have some random payoff associated with them, in a class of problems known as bandit problems. The probability distribution of the payoff for each action is unknown, and UCB1 acts to sample the actions, exploring all actions and exploiting particularly favourable actions with the least observed regret in payoff.

Without defining the algorithm entirely, it can be summarised that UCT adapts UCB1 to treat each state in a tree search algorithm as a bandit problem, where each action has an unknown payoff distribution.

Other selection strategies are known of, but have not been tested thoroughly within this thesis, as MCTS using UCT is effective in many circumstances with little tuning. Lucas et al. [44] detail a selection strategy for MCTS with feature-based evolutionary biasing, referred to as fast evolutionary MCTS, and show that it can be more effective in some test problems than MCTS using standard UCT for selection of nodes. By analysing the specific features of the problem as aspects to guide selection, it is possible to make better selection of nodes within the search space, at the cost of requiring domain specific knowledge for feature analysis. The exact importance of the features is part of what fast evolutionary MCTS will optimise, but the initial existence of the weighting must be specified. The evolutionary approach described is a simple hill-climbing approach, and there is commentary that this may be all that is required given the adaptability of the MCTS algorithm itself. This is the form in which testing was conducted in the course of this thesis, but it was ultimately passed over in favour of more familiar MCTS methods.

Perez et al. [61] examine an extension to fast evolutionary MCTS, and seem to address one of the shortcomings of the previously described method, which is the requirement for domain specific features. In this work, Perez et al. attempt to determine useful features in the context of general game playing, where features are not only variable between games but variable between game steps within a single game. Given there are significantly more games where important entities may exist or not exist at an arbitrary moment than games with fixed entities, it is not difficult to see why this might be the case. Interestingly, this extended method is compared to the more common MCTS with unmodified UCT selection, to the previous fast evolutionary method, and to a method of MCTS with feature-based biasing without the evolutionary aspect, and knowledge-based fast evolutionary MCTS seems to perform better than the other three variants in most scenarios. Unfortunately, there was not enough time to include this method as a controller in the experiments discussed in this thesis, but the possibility for future research would seem self-evident.

In particular, other than the simpler controllers used as described in previous work [68], the other main controller of interest used in this research is a Monte Carlo Tree Search based controller with minimal adjustment to the standard algorithm [8].

2.3.3 Use of MCTS in Prior Work

Chaslot et al. [8] showcase the basic MCTS algorithm and discuss how the technique can be useful for computational intelligence within game contexts, such as board games and video games. It is especially adaptive to situations where a heuristic function is absent or where the search space can branch quite elaborately, situations where search methods such as A* and minimax can struggle, and where greedy search and other such 1-ply lookahead controllers do not function well. Samothrakis et al. [72] show that MCTS can be used for real-time games with multiple opponents through limiting the depth of the search tree to effective result. The use of MCTS in experiments for this thesis is also depth-limited to enable faster operation.

Multiple works [65] [60] [59] show that the physical travelling salesman problem, a problem that includes the path following problem examined in earlier research as one of a set of problems [68], can be handled using MCTS with macro-actions. This discovery is the basis for the MCTS controller used in my own research for later problems involving MCTS, although the heuristic route planning is not used to compute routes. Future research [64] examines multi-objective approaches to the problems, although multi-objective means were not fully examined in the course of experiments that this thesis examines.

Perez et al. [60] examine the physical travelling salesman problem, and demonstrate the advantages of a long-term planning strategy within the context of this problem. While MCTS without significant modification is suitable for short term planning in such a scenario, PTSP involves physics wherein the shortest path is not a simple case of minimising distance, but of reducing directional changes. A long-term plan is formed using A* pathfinding to approach waypoints within this scenario, with MCTS used as the the short-term plan for actually navigating to each waypoint. This combined approach performs better than MCTS alone in this problem context.

Continuing on the theme of MCTS variations and adaptations, Baier and Winands [4] look at using minimax as a substitution or alternate strategy within different phases of MCTS. Minimax is examined as an alternative to random playouts for the rollout phase and as an alternative to UCT for selection and expansion, and back-propagation. Comparing variants of MCTS with minimax being used for these different phases shows markedly different impact in different game scenarios tested, and seem to perform worse with increasing depth allowed to the minimax algorithm, suggesting that perhaps the additional time cost of minimax works against the improvements of a theoretically smarter approach to rollouts or expansion. In addition, Tak et al. [86] show that MCTS can be adapted for use in games where players move simultaneously, testing several games with several variant techniques to varying effectiveness.

MCTS can be adapted to several different games without the need for significant modification, however, and can in several cases perform well in such situations. Nijssen et al. [51] look at the use of MCTS as a control method in the game *Scotland Yard*, a discrete turn based board game with hidden information. Comparisons of MCTS to other forms of tree search, variants of minimax adapted to cope with uncertainty and player coalitions, showed that MCTS players were much more challenging opponents to face. In a less discrete simulation, Sanselone et al. [73] demonstrate a case where MCTS was used to control several different agents in a learning simulation based around an operating room. The agents were required to act in specific roles as assistants to the player, with the ability to adapt their chosen actions in response to the player's actions. Even when the player deviates from the expected actions the learning simulation advises the player to take, MCTS allows the agents to act in response to any complications the player may create in said deviation, showcasing the adaptability that MCTS can provide.

2.4 Use of Evolution

As well established at this point, a common means of search within search-based procedural content generation [89] is the use of evolutionary computation. Several works in particular [42] [41] [26] [39] use specific techniques to evolve a neural network for content generation as discussed previously, mostly NEAT derivatives.

The key factor of change for content is typically based on a given fitness function to determine the suitability of the content for the problem domain being considered. In many cases [96] [12] [56] [26] [29], some means of player/user feedback is also used as an additional measure to an existing fitness measure to drive content generation to be more in line with modelled preferences.

Evolutionary computation has long been a widely-accepted means of problem solving, and the field of artificial life can be seen as a particularly focused form of evolutionary computation. In brief summary, artificial life is a relatively wide field concerning the modelling of aspects of biology in some form. Typically, artificial life simulations focus on the simulation and/or evolution of artificial organisms in a simulated physical environment. *Creatures*, a notable artificial life simulation produced commercially [20], simulates entire artificial biochemistries and heavily focuses on the interaction between an artificial organism's brain, body and environment. Evolutionary computation can be considered either a subset or extension from artificial life concepts, but instead of modelling an individual's capacity to survive in a simulated environment directly an individual is instead awarded fitness based on its suitability to solve some problem.

Particularly relevant approaches to the total generation of dynamic agents, both in design and in controller [77] [36], use evolutionary methods to produce interesting designs and behaviours. While coevolution for procedural content generation has been briefly mentioned before, it is also worth discussing by itself in a scope greater than just content generation, and the benefits and drawbacks of coevolution should be discussed.

2.4.1 Coevolution

Coevolution is a process wherein multiple populations or several distinct species within a single population evolve in such a way that the fitness landscape is defined by their interactions. Coevolution can be competitive, as with predator and prey species attempting to capture and evade each other, or as with parasites and hosts. It can also be cooperative, with multiple species evolving in isolation as components of a greater solution.

Cooperative coevolution is of less direct relevance to this thesis, but as an example Potter and de Jong [63] use subpopulations that evolve independently, as separate subcomponents to a greater final solution that is assembled from the best individuals from each subpopulation/species, as mentioned. Cook and Colton [15] also demonstrate the value of coevolution by evolving separate species of subcomponents to define various aspects and requirements of a game, rather than evolving the entire specification for a game in one population of overly complex individuals.

Competitive coevolution is more directly relevant to the thesis, especially focusing on the concept of predator prey dynamics, but before these can be considered it is prudent to consider that, while coevolution brings new advantages from new evolutionary pressures, coevolution can also lead to some difficulties due to the fitness landscape changing with the evolving population.

Floreano et al. [17] discuss how changing fitness landscape can lead to a stalling effect on the objective fitness of coevolutionary populations of agents. This has been described as a ‘Red Queen’ effect, in reference to the Lewis Carroll character who could move nowhere as the landscape was moving with her. This is a frequently observed behaviour in competitive coevolutionary problems, where fitness landscapes change and individuals can only ever stay in the ‘same place’ within the fitness landscape. Floreano et al. show the need for a metric to determine a population’s fitness that is independent to the same evolutionary pressures as the population.

Cliff and Miller [10] discuss the Red Queen effect in more detail, noting how direct measurement of fitness values of individuals in a coevolutionary context is seldom useful due to the changing fitness landscape coevolution causes. Proposed are means to evaluate the fitness of a population to compensate for this, such as contests between a current population and earlier ancestors of that population, and measures of genetic distance between the same. Genetic distance is less of a comparison of adequacy of solutions and more intended to be a measure that there has been any significant evolutionary progress or change whatsoever, as opposed to mild changes to adapt to a slightly changed fitness landscape.

Samothrakis et al. [70] work with evolving players in the game of Othello, and the problems of fitness measure that coevolution introduces. They focus on intransivities, where some evolved players have misleading reported effectiveness due to being able to exploit/specialise against particular other players. Recommendations to resolve this include round robin tournaments, comparison against earlier snapshots of the population of solvers, performance in relation to a fixed static player, and distance to an optimal Nash player. In the work in this thesis on the predator-prey scenario involving spaceships, tournaments are used, although knockout instead of round robin due to time constraints, as well as comparison to a fixed design. In contrast to the work by Samothrakis et al., however, Ballinger et al. [5] attempt to specifically evolve strategies to counter specific opponents in a simulated RTS context by incorporating and evolving human strategies that have worked against said opponents previously. It is interesting to note this effort to not just recognise but capitalise on intransivities in order to produce more selectively specialised opponents.

Predator and Prey

While coevolution has its downsides, the nature of competitive coevolution is one wherein innovation is required to simply survive in a changing fitness landscape. A good competitive coevolution scenario should promote rapid advancement, some form of evolutionary arms race, and in particular predator and prey scenarios are particularly suited to providing this sort of innate conflict.

Predator prey scenarios require specialist behaviours and can lead to interesting behavioural examples. Reynolds [67] demonstrates that genetic programming can be used to define programs for vehicles that effectively play tag with each other, in a predator-prey scenario, and which change behaviour upon successfully tagging each other. This work shows the capacity for fixed design ships to act both as predators and prey, but while including the capacity to branch into different behaviour while predator or prey. This need to delineate the separate behaviours shows the difference in the required strategies to succeed as predator or prey. Cliff et al. [9] have also looked at the predator prey scenario and coevolution to produce specialised predator agents and prey agents in a 2D simulation similar to the spaceship scenario examined in the course of research. The agents are modelled as circular bodies with an associated neural network, where each neuron has a physical presence and acts as an input (sensor) or output (motor) based on placement with the body. Somewhat unexpected is the level of difficulty in getting the expected behaviour of predator-that-catches-prey discussed in this paper. Other approaches also show that teamwork can be evolved between groups of predators, such as the work of Wittkamp et al. [94] which shows that such teamwork can be evolved even in an environmental context that includes noise. Unfortunately, there was not sufficient time to examine team behaviour of agents to as much detail as desired for this thesis.

2.4.2 CMA-ES

Evolution strategies themselves are a part of a greater set of evolutionary computation methods [81], and are an effective "generate-and-test" method of content generation.

While a full definition of CMA-ES [24] is beyond the scope of this review, it is an evolution strategy based upon shaping the distribution of a population towards the optimal distribution.

CMA-ES is based upon adjusting a mean vector and covariance matrix of a population made up of vectors of random scalar variables. Unlike other forms of genetic algorithm, which maintain a distinct population of specific individuals, CMA-ES simply tracks the mean vector and covariance matrix without reference to individuals, as these features describe the distribution of the population without the requirement for specific instances from it.

For each evolutionary iteration, individuals are sampled from the random distribution described by the covariance matrix, and the fitnesses of these individuals are evaluated. The mean of the distribution is adjusted based on the samples that produce higher fitnesses, with the probability of selecting samples that produced higher fitness values before increasing in the process, and the covariance matrix is adjusted also so that the steps taken to reach the most successful individual also become likelier. In this way, CMA-ES follows the gradient of the fitness landscape stochastically, with the stochastic element allowing for resiliency and adaptability.

It is innately a minimisation method that can be used on problems of up to 80 dimensions with effectiveness [24], which makes it suitable for the high dimensionality of the problems examined within this thesis. Another one of CMA-ES's strengths lies in its ability to function without a large degree of tuning [22], performing well with provided defaults and able to handle the ill-conditioned problems encountered in the course of this research. Ill-conditioned problem functions are functions where small changes to the inputs can result in very large changes to the outputs.

Evolution strategies were considered in general as a problem-solving approach as opposed to methods such as neuroevolution as, while plenty of existing literature demonstrates the effectiveness of neuroevolutionary or similar types of iterative search for similar problems of content and behaviour generation, there appeared to be a deficit of work using evolutionary strategies to do so. Using evolutionary strategies for a problem of this nature appeared to be a novel approach and a more beneficial contribution to the field as a result.

Preliminary work indicated that evolution strategies and CMA-ES were effective at finding good solutions to basic forms of the problems examined, and as CMA-ES was shown to be effective and known to be a resilient algorithm, it was the algorithm selected to attempt to find solutions to problems for the evolutionary algorithm component of the problem-solving approach taken.

It has also been shown [23] that CMA-ES exhibits better performance for ambiguous problems in comparison to other algorithms also operating within the class of estimation of population distributions.

Previous work has been documented [68] to demonstrate that CMA-ES can be used with considerable effectiveness to evolve spaceship designs for various tasks. The earlier success of CMA-ES in previous experiments lead it to be used for the course of research as an effective means of generation solutions. It is also feasible to use CMA-ES for multi-objective optimisation [31], although this is beyond the scope of the thesis, which instead examined problems where sub-goals or multiple objectives were phrased as weighted components of a single function to optimise. This is not an unusual choice when compared to other examples of content generation through evolutionary means [18] [7] [46] [40]. In the context of entity generation, CMA-ES has also seen some successful use in the evolution of muscle co-ordination in predefined creature structures, leading to interesting behaviours that react to adverse scenarios without the need to necessarily perceive what is happening [19].

While evolution strategies such as CMA-ES are less represented within the context of procedural content generation, there are some examples of prior work. Perez et al. [62] describe using CMA-ES as a means for content generation through optimisation of constraints for a level in the physical travelling salesman problem, a scenario based upon the classic travelling salesman problem with the addition of a Newtonian physics model and a vehicle that must travel to each point once. As in other work using CMA-ES for PCG [68], the levels are encoded as a vector of real number values, and are evaluated by estimation of the optimal route to traverse the levels created for this game, favouring levels with shorter or faster estimated routes.

As can be seen in this review, there has been significant effort into studying means of content generation that uses simple genetic algorithms, some variation of genetic programming, or one of several highly specialised neuroevolutionary techniques that readily lend themselves to the task of procedural content generation. It is noted, however, that content generation through the use of evolutionary strategies such as CMA-ES is less represented within the literature. This thesis does not investigate CMA-ES as a means of pure optimisation for a problem, as content generation can be seen as a highly creative endeavour and thus highly subjective and qualitative, but instead explores the use of CMA-ES as an effective means of attempting optimisation for increasingly complex tasks. While direct comparisons between neuroevolutionary methods and the CMA-ES methods used within this thesis cannot be backed by direct empirical evidence due to time constraints, it is interesting to note some of the differences in outcome caused by the different methods of problem solving requiring different means of solution encoding.

2.5 Summary

So, as has been determined through analysis of the literature, while there exist many, many examples of search-based procedural content generation, there is especially heavy favouring of certain techniques over others. This may be because, such as in the case of CPPN-NEAT [82], diversity of solutions is preserved, but it is also clear that CMA-ES [24] is under-represented. As there exist successful applications of CMA-ES in the generation of content [68] [62], it is possible that the more popular methods, such as genetic algorithms or neuroevolutionary approaches, are simply more visible within the literature, or that their wide report of successes is what leads to their selection over other options. Either way, it appears there is room for examining CMA-ES as a means of content generation.

It is also clear that MCTS is an adaptive control strategy for agents within game simulations to use, and that its use as an example of a more complicated but more effective control system (compared to simpler approaches such as 1-ply lookahead controllers) seems well justified for the type of scenario being evaluated, with adaptations such as the use of macro-actions [65] [59] to reduce branching complexity for the simulation search space.

Chapter 3

Vehicle Generation for Navigational Tasks

3.1 Problem Summary

3.1.1 The Need for Basic Problems

Within this chapter we will examine the basics of vehicle generation and evolution. In order to generate vehicle designs it is first necessary to examine a suitable means for doing so.

The basic problem that is examined within this chapter consists of generating spaceships for very simple problems; moving in a direction from a fixed starting point, following a path, and meeting some design objectives. These design objectives in particular are to produce a ship with a high potential linear thrust, which can also be considered as a high "speed", while also producing a ship that can produce high torque to be able to change its direction effectively. These three basic problems are formulated to define a purpose for the overarching problem of designing spaceships to meet some goal. This constitutes preliminary work to ensure that the methods chosen for spaceship generation are suitable for the purposes considered. All three problems assume a similar problem environment, that of an infinite two-dimensional plane, which will be discussed in more detail in the next section.

These three problems are examined in order to provide a constructive base to understand how effective the chosen means of generation, including the representation of spaceship designs, and evolutionary method are at fulfilling the task of generating interesting designs in a simpler context with less overhead from problem complexity. In particular, the basic problem of moving to the right is a trivial problem, and any choice of generative algorithm that could not meet such a task would be noticeably unsuitable to the problem being considered.

3.1.2 Relevance to Game Context

Given the basic problems mentioned previously, it is key to understand what makes these problems worth investigating and why they are an effective means of indicating and prototyping the means for vehicle generation that later work expands and builds upon.

The ability for a vehicle to move in a straight line is paramount for any sort of game that contains agents that interact with a physical environment, and the ability to follow a path is also particularly relevant in several pre-existing games with a physical environment..

The number, position, and orientation of a set of rocket motors (thrusters) on a spaceship are evolved to produce designs capable of satisfying the demands of the basic problems mentioned. Due to the simple but accurate (given the limitations of what was chosen to model) implementation of a physics model, this leads to some interesting behaviours emerging, which already could provide some interesting enemy ships for an *Asteroids*-style game.

Each thruster is either on or off for each time-step, and in our opinion this adds interest to the behaviour compared to ships that move using unseen and perhaps unrealistic forces. The possibility also arises for novel (or at least, rarely used) game mechanics where ships can be partially disabled in a physically realistic way by shooting out particular thrusters. For example, if a ship has a single thruster acting as a brake, then shooting out that thruster may cause more havoc behind enemy lines than destroying the entire ship. While this particular example of a described behaviour is not examined in detail in this thesis, the architecture makes such a thing possible.

For now, the extent to which effective ship designs can be evolved given the experimental setup is examined.

3.2 Methodology

In this section, it shall be discussed how the experiments were set up in more detail. The actual simulation code for the experiments as described was written in Java, using the publicly available Hansen Java implementation of CMA-ES[1] without modification, and using the default parameter settings.

3.2.1 Environmental Physics

The environment that these spaceships were simulated within was a two dimensional flat plane. There were no limits or edges to this plane, other than the numerical limits that arise from hardware. There was a velocity loss factor, reducing velocity similarly to how drag and friction

[1] `http://www.lri.fr/~hansen/cmaes_inmatlab.html`

forces would reduce speeds, although this would be more properly considered a convenience property of a game environment than actual space. This was calculated as a multiplication per simulation step of both linear and angular velocities by a constant factor. This applies to all spaceships. Though non-intuitive given the scenario of spaceships in an empty space, this friction presented both an obstacle of sorts for spaceships to overcome and a feature to exploit. This friction could also be set to zero if desired. It should be noted that this necessarily increases the difficulty of the problem, as active action is required to brake as opposed to passive inaction.

3.2.2 Spaceship Construction and Composition

Physical Properties of a Spaceship

A spaceship has a number of physical properties. At its most basic, a spaceship is modelled as a point, and for both these experiments and all future experiments discussed this holds true for the purposes of physical calculations. This point possesses a mass, m, a centre of mass, $\mathbf{R}$, and a moment of inertia, I. These values begin identical for all spaceships at the constant core values of m_c and I_c. A spaceship has a position vector, $\mathbf{s}$, a velocity vector, $\mathbf{v}$, a rotation, θ, and an angular velocity, ω. A spaceship can also have an arbitrary number of components attached to it, which are also modelled as points with their own masses, m_t, and turning moments, I_t. A component is attached to the ship based on a relative position offset, $\mathbf{s_t}$, and a relative angle offset, θ_t.

The values specified for these are assumed to be when the spaceship is not rotated and its local axes match the world axes. The component attachment is rigid and cannot be broken for any reason.

For these experiments, the attachment of a component to the spaceship increases the overall mass of the spaceship through the modelling of an added "strut" mass. A strut affects the mass and turning moment of the overall ship depending on how long it is. The length of a strut, l is exactly equal to the distance from the point mass of the component to the spaceship's origin point. This is not the same as the spaceship's centre of mass, which is recalculated for each new addition to the ship. As the position offset of the component is relative to the origin point, this therefore means that $l = |\mathbf{s}_t|$. The mass of a strut is proportional to its length, or $m_s = k_m l$, where k_m is set to two. As the strut is rectangular, its centre of mass, $\mathbf{R_s}$, is exactly equal to its midpoint, which is $\frac{1}{2}\mathbf{s}_t$. Its moment of inertia is equal to its mass multiplied by an arbitrary constant, k_I. It is not modelled as a physical rectangle, however, and is instead modelled as an additional point mass halfway between the origin of the ship and the position of the thruster. All of these changes are in addition to the direct changes caused by adding the component itself to the ship.

In summary, the most basic physical properties of a given spaceship are defined as follows. Given that, for the ith thruster:

$$m_s^i = k_m |\mathbf{s}_t^i|$$

$$\mathbf{s}_s^i = \frac{1}{2}\mathbf{s}_t^i$$

$$I_s^i = k_I m_s^i$$

The properties of a ship with N thrusters are:

$$m = m_c + m_t \sum_{i=1}^{N} m_s^i$$

$$\mathbf{R} = \frac{1}{m}\sum_{i=1}^{N}(m_t \mathbf{s}_t^i + m_s^i \mathbf{s}_s^i)$$

$$I = I_c + \sum_{i=1}^{N}((I_t|\mathbf{s}_t^i - \mathbf{R}|^2) + (I_s^i|\mathbf{s}_s^i - \mathbf{R}|^2))$$

While reference is made to components as a generic collective term, within these experiments the only components that were examined were thrusters. The set of components would be added to in later experiments, but this restriction was made for the purposes of simplifying the problems examined. All thrusters had and have the same mass, moment of inertia and thrusting power. Thrusters can either be on, and providing a fixed thrusting force, or off, providing no force. The front of each thruster is defined to be its attachment point, and the resultant direction of thrust is in line from the back to the front of the thruster. Depending on the attachment position and rotation relative to the spaceship's centre of mass, thrusters can provide linear acceleration, angular acceleration, or in most cases both.

The shape of the spaceship's hull is defined as a polygon, with each attached component defining a vertex of the polygon. The shape of the spaceship does not affect the physics of these simulations, however, as the spaceship is modelled as a connected group of point masses. The model includes a set of struts linking each thruster to the centre of mass of the ship, as explained earlier.

In some of the much later experiments covered in future chapters, the shape of the spaceship hull would influence collision detection, but in these experiments there is no reason to detect collision, and the hull polygon exists as a purely cosmetic feature for spaceships within the context of these simpler problems.

3.2.3 Spaceship Representation

As the core mass of each spaceship is constant, the only variation between spaceships lies in the arrangement of components attached to a spaceship. As a result, all that must be modelled to evolve a spaceship are the components, and how they are arranged. As the only existing components are thrusters, this reduces the requirements of the representation to simply specify the properties for the thrusters.

Each thruster has two properties (three numbers in total), as thrust is fixed at a constant value for all thrusters. These properties are a vector representing the position of the thruster and a real number representing the rotation of the thruster. Both values are relative to the origin point of the spaceship the thruster is attached to, providing the spaceship is aligned with the world axes. The vector representing position contains two elements, an x co-ordinate and a y co-ordinate, and this means in total three real numbers can be used to position and rotate a thruster.

As a result, a spaceship is represented as a vector of real numbers of size $3n$, where n is the maximum number of thrusters a spaceship can have. This may not always be equal to the number of thrusters a spaceship can possess. When constructing a spaceship from its representation, if a thruster would be placed within a certain distance of another thruster, it is silently discarded and not added to the spaceship. Thrusters occurring earlier in the vector have higher precedence in placement than thrusters occurring later, and are more likely to be considered valid.

Due to invalid thrusters not being placed, part of the representation of a spaceship can become unused data that does not directly affect the spaceship's performance. Despite the data for these impossible thrusters seeming useless, it can, as in nature, become useful if the data is altered significantly. It can also become useful if the thrusters blocking a thruster from being viable are moved away, freeing the space for the thruster to be added.

It is key to mention that this representation used in these preliminary experiments would not remain the final representation used in later experiments, and that this was also not the first choice of representation used. Previous representations will now be briefly discussed, both to explain what was tried, and to also explain the reasons that these representations were not used.

Previous Representations

Object Representation In the earliest iterations of these experiments, ships did not have an actual genetic representation, and were instead treated as an array of components. The available operations for creating new ships from existing ships were crossover, mutation, and cloning. Crossover would occur with a 70% chance, mutation with a 20% chance and cloning with a 10% chance.

If a ship was selected for crossover, the second ship would be selected through a basic tournament selection. During crossover, a copy of the first ship's components list would be made, and at a randomly selected point every component following this selected point would be discarded and replaced with a partial copy of the second ship's components list, starting from the equivalent point. The resulting ship would be copied into the next generation's population.

If a ship was selected for mutation, the relative position and rotation of a randomly selected component would be modified by a small amount, and this mutated ship would be copied into the next generation's population. If a ship was selected for cloning, it was simply copied directly into the population of the next generation.

This was a very ineffective and slow method of representing ships, and was both highly inflexible and did not allow for changes to the implementation of the vehicles - what should be the phenotype to the genotype - without also requiring changes to the genetic operators. This was very quickly replaced with a real-number vector based representation, which would also allow for evolution strategies such as CMA-ES to be used to improve these number-string based chromosomes much more effectively.

Attachment Point Representation The first version of the representation used was more compact than the three-value-per-component variant described, requiring only a single value per thruster. This was achieved by modelling components as elements attached to a predefined hull at given points, with the index of a value corresponding to an attachment point on the spaceship and the value itself corresponding to a rotation in radians for the component to have, still relative to the main spaceship rotation.

This, however, required a fixed pre-designed hull structure and could not allow for any variation on component placement. The configurations of thrusters produced were also not particularly interesting or effective, and so the current representation allowing for free placement of components, with the body of the spaceship defined by the placement of components, was the next one designed. It proved to be a flexible base for both these experiments and future experiments with mild modifications as future experiments became more complex.

3.2.4 Simulation Properties

The simulation used for testing and evaluating spaceship designs operated in terms of discrete time units, referred to as timesteps. At the beginning of a simulation, a randomly seeded population of spaceships would be iterated over once per discrete time unit, with each spaceship's physical properties (such as velocity, acceleration, angular velocity etc.) recalculated based on the amount of time passed. In effect, the simulation acted as a simple numerical integrator with regards to properties such as acceleration. The environmental physics as previously discussed also factored into how each spaceship was updated per iteration of the loop. Each time unit had a value of 0.2 seconds for the tests considered, and this value would not change for the remainder of the experimental work. While the potential for numerical integration of resultant velocities and positions to be more accurate with a smaller timestep was considered, this was considered beyond the scope of the problem being investigated.

Within the context of this simulation, values such as acceleration for the ships would remain at a constant zero and produce no action without the direct action of a controller. The purpose of a controller within the context of the simulation was to control the thrusters of a ship, determining which of the thrusters to turn on or turn off for a time unit. Based on the placement of these thrusters, the acceleration created would change, meaning that ultimately the combination of both thruster placement and controller decision together made up the behaviour and effectiveness of a design for a given problem context.

After a set number of time units, the experiment would end, and the fitnesses for each individual in the spaceship population would be calculated based on the fitness function for the problem being considered.

3.2.5 Spaceship Controller

When being simulated, a spaceship must be controlled effectively in order to demonstrate that its design is useful for the simulation it exists in.

For each step of a simulation, a controller is given a target position in the two dimensional space to move the ship towards. For simple linear movement, the target is the current position of the ship, plus an offset to the right. This unattainable target drives the ship constantly to the right, which is sufficient for the purposes of that particular experiment. For following a path, however, the target is whichever way-point the ship has to visit next along the path.

As the design objectives problem is not simulation based, there is no need to consider a controller in its context.

Lookup Table Controller

The primary controller used for spaceships during simulation is a simple algorithm. It is not the same as a one-step look-ahead controller that would examine every possible state after every possible action a ship can take, although such a controller was used prior to the development of this controller. Because this controller does not examine states, it is considerably faster than a one-step look-ahead controller. This is because no simulation is required to determine states after actions have been made, which lowers the required amount of simulation in an experiment significantly.

During initialisation, a lookup-table based controller is assigned to each ship. Upon being assigned to a ship, the controller calculates a table of actions and the results that arise from taking these actions. As mentioned before, an action is a combination of states for the thrusters of a ship, as shown in Fig. 3.3. The controller is capable of deciding what thrusters should be active and inactive for each step of the simulation, and knowing in advance what each combination does allows for faster decisions later. For each action the ship can possibly take, the controller stores the resulting changes in linear and angular velocity.

Given a target, the controller will first attempt to look up the action it can take that will give it the most appropriate magnitude of linear thrust towards the target. The most appropriate magnitude of the vector of acceleration is determined to be that of the magnitude of the distance. This is a rough estimate of the amount of thrust required and not a precise requirement, but this heuristic value aids the controller in selecting an appropriate amount of thrust to use to move to a target. During this selection process, only the magnitude is important, as the direction is considered in the next step. The action that produces a thrust vector with the closest (smallest absolute difference) magnitude is selected.

Once it is known which action will produce the most appropriate magnitude of thrust, the direction heading of the target is compared to the direction this action will take the spaceship. If the angular difference between the direction to the target and the direction of thrust is within a certain tolerance, the controller will commit to accelerating the ship using the previously chosen thrust action.

If the angular difference is not within this tolerance, and the controller is not already committed to moving in a straight line, the controller will next consider the angular difference. It will look for an action with as suitable a torque force as possible, with a suitable torque being something close to the angular difference. The controller will then choose this action for the ship to take. If, while accelerating linearly, the angular difference becomes larger than some tolerance, the controller will cease accelerating linearly and attempt to correct its orientation.

The behaviour that results from this is perhaps not the best means of piloting a spaceship, but it is reliable and consistent. Spaceships will turn until they are in a suitable orientation to move rapidly towards a target, occasionally correcting their course.

It is worth noting that the controller has been known to struggle with certain ship designs (see Fig 3.4). The designs which tend to provoke this erroneous behaviour are not typically designs which would be useful or desired for the specific situation. However, they might also be designs that human players could control where the artificial controller could not. These are rarer occurrences, however.

Greedy Controller

Prior to the use of this lookup-table based controller, however, there was a basic one step look ahead greedy search controller used for very early testing of the simulation. A greedy search controller is a controller that, when presented with all available actions for the next state in the simulation, naively chooses the action with the highest expected value with no further consideration or planning. It is an inherently reactive controller highly susceptible to local minima, but is simple to understand and implement.

This controller did use partial simulation, for exactly one frame, for each possible configuration of thrusters being either on or off, and would use the same fitness function for the problem as a heuristic measure of the suitability of an action. Within the context of the moving right problem, the greedy controller would simply favour any action that moved it furthest to the right, and within the context of following a path the greedy controller would favour any action that moved it closer to the next waypoint for consideration.

While it should be mentioned that this controller was used as an initial test, it is mentioned more as a historical note and explanation for the nature of the controller used primarily for the results presented in this chapter, as the designs produced using this controller almost all fall into the same design category and behaviour. While this is worth noting and is elaborated on in subsection 3.3.4, there is otherwise little to note regarding this particular instance of a 1-ply controller being used to test spaceships.

3.2.6 Generation Experiments

As stated before, different problems were examined, and these different problems were examined to discover and compare the impact of specific requirements on the resulting spaceship designs. One such problem is an attempt to evolve a population to meet specific constraints or design objectives likely to promote good ship design. Attempting to fulfil design objectives to produce ships that may be effective is, however, not identical to producing ships that are effective in practice, especially given the limitations of the lookup table controller used. Two further fitness measures are used to produce ship designs, and both of these are empirical, relying on the outcome of fixed length simulations. As the implementation of CMA-ES minimises (rather than maximises), by default the fitness scores are converted from maximisation problems to minimisation by a simple transformation $F_t = C - F_a$, where F_t is the transformed fitness, F_a is the original fitness and C is a suitably large constant.On the graphs in the results section we

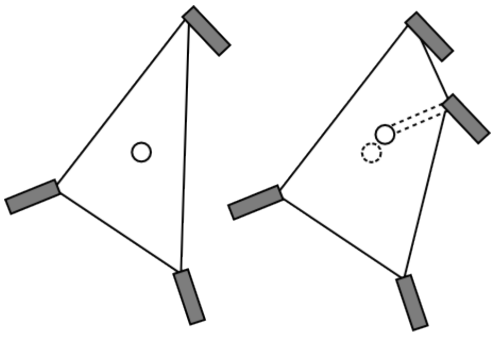


Figure 3.1: When a component is added to a spaceship, the centre of mass, among other properties, changes. Further properties are affected by the addition of an invisible “strut” between the thruster and the centre of mass. This ensures larger ships have a larger mass.

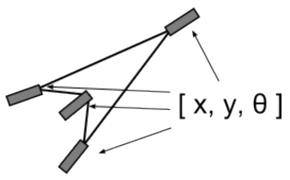


Figure 3.2: Every thruster on a ship can be represented as a three values. The first two values relate to the position vector, and the third value relates to the rotation in radians. A string of sets of three values can represent a set of thrusters. As all other properties are constant, a set of thrusters can define a spaceship design.

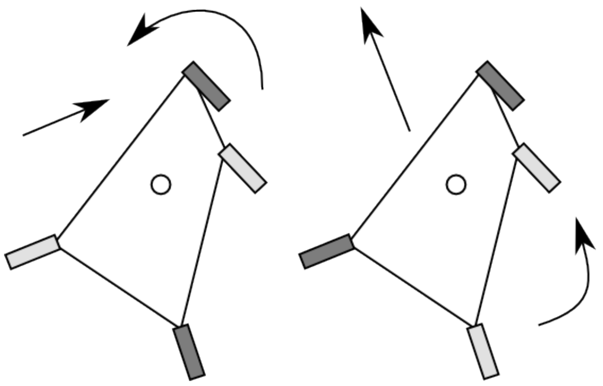


Figure 3.3: In this diagram, lighter coloured thrusters are active. The curved arrows are representative of how much torque is produced and in which direction. The straight arrows are representative of how much thrust is produced and which direction the ship will move in.

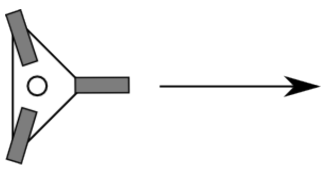


Figure 3.4: This ship will never move to the right. None of its potential thrust vectors can be rotated to point in the desired direction, as all of its thrusters are directly aligned with the centre of mass. The ship cannot turn, and so this ship will never move.

show the raw fitness values.

All problems discussed here used specific parameters and constraints, shown in Table 3.1, that did not change during the course of the experiment, giving a context to the values mentioned in the following descriptions of problems and fitness functions. The values chosen for these initial experiments were decided based on fiat decisions.

Design Objectives Problem

The simplest of these fitness measures rewards spaceship designs with the capacity for high movement and high turning speed. These were calculated by examining every possible action a spaceship can take. An action is defined as a set of states for each of the thrusters belonging to a ship. As each thruster can be only on or off, the only means of controlling a spaceship is to set thrusters to be firing or not firing. For every possible action a ship can take, the resulting linear velocity and angular velocity are compared with the largest known linear and angular velocities, and the largest known values for each were updated respectively. Spaceships capable of higher linear acceleration and higher angular acceleration were judged to be more capable of moving and manoeuvring than ships without such properties, and were scored more highly. The final calculated fitness of a ship was:

$$F_a = w_1 v_{max} + w_2 \omega_{max}$$

where v_{max} is a scalar indicating the largest magnitude of linear thrust the ship is capable of, ω_{max} is a scalar indicating the largest absolute amount of angular acceleration the ship is capable of producing and both w_1 and w_2 are weightings for both of these components.

Linear Movement Problem

One of the major shortcomings of the design objectives problem is that the ship designs are scored based on assumed principles and are never actually tested in a meaningful way beyond possible outputs. The linear movement problem is a simple simulation-based scenario with a goal of moving simply as far to the right from a starting point as possible. This promotes ship designs that are capable of quickly orienting themselves to a preferred direction of travel, as well as quick movement in that direction of travel, as in Fig. 3.5. Ships were scored based on the horizontal distance travelled within a fixed amount of simulation steps, and penalised for travelling vertically. This places emphasis on travel in a straight line. A high horizontal distance implies that the spaceship was able to turn and move quickly, and so was scored more highly. A high vertical distance implies the spaceship may not have altered its initial orientation correctly. The ship is simulated a number of times in specific rotations, and its final score is the mean of the scores achieved in all of these runs. For n trials, the initial starting angle for a ship on the ith repeat was $i\frac{2\pi}{n}$ radians. The fitness calculated during one of these trials was

$$F_i = w_x d_x - w_y |d_y|$$

with d_x as the horizontal distance travelled, d_y as the vertical distance travelled, and both w_x and w_y as weights. The final fitness was the mean of F_i over all trials.

The need for a series of specific starting rotations is to avoid random noise affecting the progress of the ship designs. Introducing random orientations can mean that poorly designed ships unable to steer correctly can be scored with a much greater fitness than they otherwise should have, while better ship designs could be scored far lower than they should be for similar reasons. Additionally, it is preferable to have a number of trials rather than a single fixed direction of rotation. If all ships were to be scored when pointing a fixed direction, it could produce designs that were only effective in moving in a single straight line. Although this experiment favours straight line speed, the fact that for some orientations it is necessary to turn first means that the fit designs need at least some turning ability.

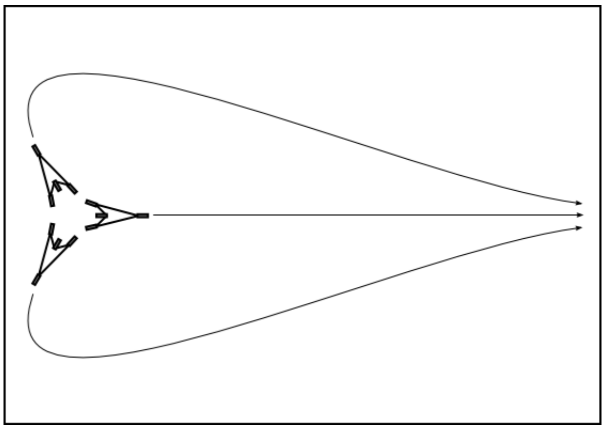


Figure 3.5: Initial starting rotations of a ship in the linear movement experiment and expected trajectories.

Path Following Problem

While assessing ships moving to the right does allow for spaceship designs that can turn and move, it does not promote designs that are capable of finer manoeuvring, such as turning to stay on a path. A path following experiment, along with fitness measure, was used to promote the development of ships capable of following a path consisting of a number of nodes or way-points, such as in Fig. 3.6. This path was kept constant throughout the course of the experiment, and the challenge posed to the ships was to reach as many waypoints within a fixed amount of simulation steps. This path is created through generating a fixed number of points using a fixed seed for the random number generator used.

Ships were scored based on the amount of way-points successfully reached, multiplied by a constant, and were then penalised by the distance to the immediate next way-point along the path that the ship did not reach. This allows for a finer degree of granularity in scoring ship designs than simple measurement of way-points reached alone. As with linear movement, the ship was simulated a number of times in different starting rotations, with its final score equal to the mean of the scores achieved over all trials. The starting rotation for each trial was, as with the linear movement experiment, $i\frac{2\pi}{n}$ radians, where n is the number of trials and i is the current trial. The fitness score for this trial was:

$$F_i = w_1 \cdot (N_{max} - N_j) + w_2 s$$

N_j is the index of the most recent way-point visited, N_{max} is the number of way-points on the path, and s is the magnitude of the displacement from the ship's ending position towards way-point N_{j+1}. w_1 and w_2 are weights used to adjust the significance of factors in the fitness measure. As with the linear movement experiment, the overall score was simply the mean of F_i over all trials.

Where the other functions are phrased as functions to be maximised, and then converted to a more appropriate form for CMA-ES, this particular function produces lower values for higher success. This function does not need to be adjusted for use by CMA-ES.

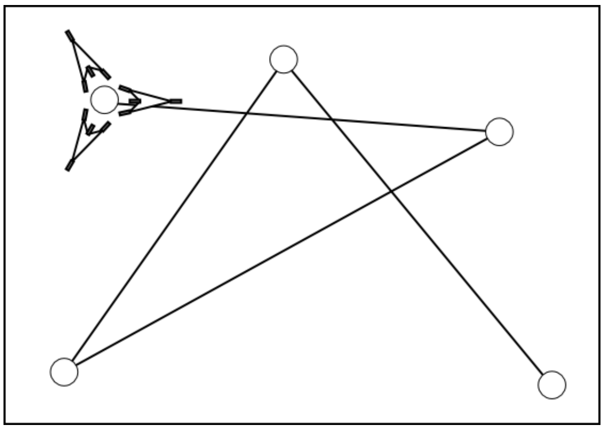


Figure 3.6: Initial starting rotations of a ship in the path following experiment, and an example path that must be followed. Ships must move from one point to the next.

3.3 Initial Results

The results of evolving spaceship designs to meet the three problems can be described for each problem examined. As can be seen in all of these experiments, as in Fig. 3.7, 3.8 and 3.11,

Table 3.1: Parameters and Constants

Table 3.2: *

Experiment	Symbol
Design Objective	d
Linear Movement	l
Path Following	p

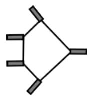
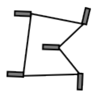
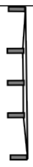
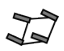
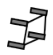
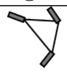
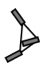
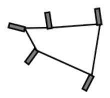
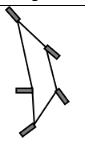
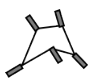
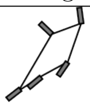
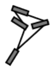
Parameter	Experiment	Value
Maximum generations	all	10000
Initial values	all	0
Initial standard deviation	all	20
Velocity loss factor per step	all	0.99
Spaceship initial mass m_s	all	100
Spaceship initial moment of inertia I_s	all	1
No. of ship thrusters	all	5
Thruster mass m_i	all	12
Thruster moment I_i	all	0.5
Thruster force	all	200
Strut distance mass factor k_m	all	2
Strut distance moment factor k_I	all	0.1
Angular tolerance to start moving (controller)	all	$\frac{\pi}{32}$
Angular tolerance to stop moving (controller)	all	$\frac{\pi}{8}$
Weighting of maximum velocity w_1	d	1
Weighting of maximum torque w_2	d	1
Number of starting rotations n	l, p	3
Number of steps in a simulation	l, p	200
Weighting of horizontal distance travelled w_x	l	1
Weighting of vertical distance travelled w_y	l	0.5
Way-points reached weighting w_1	p	100
Distance remaining weighting w_2	p	1
Number of way-points on path N_{max}	p	6
Radius of way-point	p	20

the best fitness in each generation becomes a limit of the population fitness as a whole quite rapidly. After beginning with a high level of fluctuation as various solutions are found, the general population begins to converge to a single design. When this occurs, the best fitness measure found so far is not as prone to change as much as the rest of the population does over time. There is, of course, a danger of this being simply convergence to some particular local optima, but it is hard to know this for sure. While the problems are still fairly simple, the solution space is incredibly vast.

It is also worth nothing that, outside of the given problem environment these ships were designed for, the ships that were designed for more complex problems generally ended up performing better in simpler scenarios than ships that were designed for simpler scenarios and presented with complex problems to solve. This seems to follow intuitively, although it is still clear that a given ship design will perform the best in the original problem it was generated as a solution for.

Some example designs and their fitnesses in different experiments can be found in Table 3.3. As mentioned previously, lower scores indicate better behaviour.

Table 3.3: Example Ship Designs, with Fitness Scores

Fitness	Off-line Designs		
			
Design Objective	7.99	8.27	8.51
Linear Movement	1774	1866	2114
Path Following	738	785	864
Fitness	**Design Objective Designs**		
			
Design Objective	6.53	6.52	7.85
Linear Movement	1855	1590	1934
Path Following	786	845	911
Fitness	**Linear Movement Designs**		
			
Design Objective	7.47	8.59	8.75
Linear Movement	1508	1646	1737
Path Following	739	777	755
Fitness	**Path Following Designs**		
			
Design Objective	8.52	8.50	8.64
Linear Movement	1744	1795	1728
Path Following	575	671	638

The three offline designs were designed by hand without any evolutionary improvement to be used as a baseline. The other examples given in the table were selected from experimental results themselves selected purely at random. They represent the best design at the final generation for the given experiment. The designs are divided up based on their problem of origin.

As can be seen, designs evolved specifically to excel at a given problem domain outperform offline human designs, although are specialised to the specific problem they were evolved to solve. Interestingly enough, the third manual design performs with much less effectiveness at the task of moving to the right than would otherwise be assumed, due to the need to initially reorient the ship at the beginning of the task to face to the right. This initial reorientation with a tight time budget requires effectiveness in turning.

3.3.1 Design Objectives Problem

Spaceship designs produced in this context were almost always small. This was perhaps a result of smaller ships possessing smaller mass, and therefore greater acceleration potential.

This greater acceleration potential resulted in spaceships which, when compared against similar configurations of thrusters for larger ships, accelerated at a greater rate than larger ships. It is interesting to note that the designs produced follow certain zig-zag patterns. Results of a typical evolutionary run are shown in Figure 3.7.

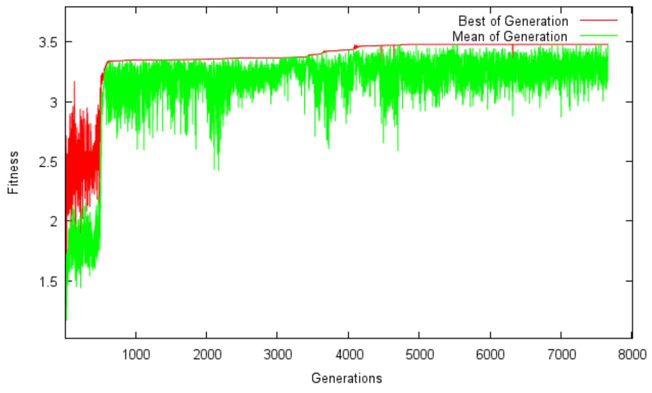


Figure 3.7: An example of a swiftly converging design objective run, chosen at random. Due to lack of improvement near the end of the run, it was terminated early. It is common for design objective runs to converge to a maximum without advancing much further. The fitness shown is raw fitness that has not been transformed for CMA-ES.

3.3.2 Linear Movement Problem

Spaceship designs that arose here tended to have multiple thrusters concentrated on a single side of the ship in order to provide a greater thrust in a single direction. The thruster positions also allowed for turning, in order for the ship to orient itself in the direction that would allow it to move the fastest. It was clear, though, that linear acceleration is prioritised above all else, and the capacity to slow down quickly was often lacking. The spaceships produced in this context did not perform as well when tested in the path following experiment.

A typical sample run is shown in Figure 3.8. This particular experiment takes a long time to improve for reasons that will be discussed later again, but can be briefly summarised here. Where CMA-ES does not immediately find a useful local minima to exploit due to initial population distribution, the tendency to explore by increasing the covariance of the population distribution leads to larger ships directly, which do not tend to perform as well as smaller ships due to bulkier masses. As can be seen in this graph, improvement stagnates for many generations, before some local minima is found that acts as a transitive step to better solutions.

In addition to the standard linear movement experiment, Figure 3.9 demonstrates how adversely random starting orientations affected the progress of a population. As can be seen, there is no improvement whatsoever, as the random factor has far outweighed every other aspect of the experiment, causing extremely unfit designs to be considered fit. This can lead to arbitrary designs, or it can lead to ships that cannot move or turn in a reasonable manner at all. Over time, this causes the fitness of the population as a whole to either reach zero, with minor fluctuations as further ships are erroneously considered fit, or to display no trend other than uniform random values.

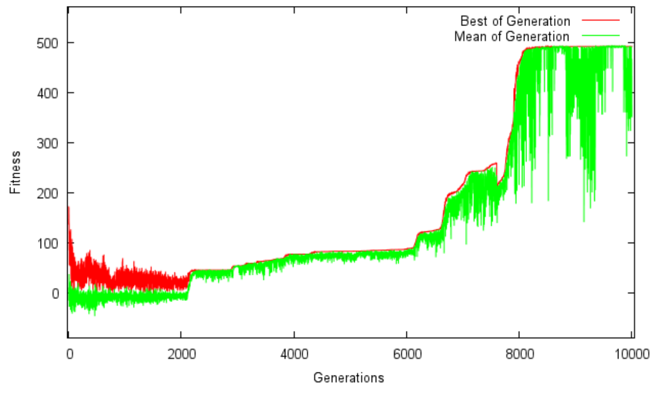


Figure 3.8: The typical progress of a spaceship population attempting to move to the right, shown here over 10,000 generations. This experiment was selected at random. Overall population convergence to an effective design is achieved by generation 8000. The fitness shown here is raw fitness that has not been transformed for CMA-ES.

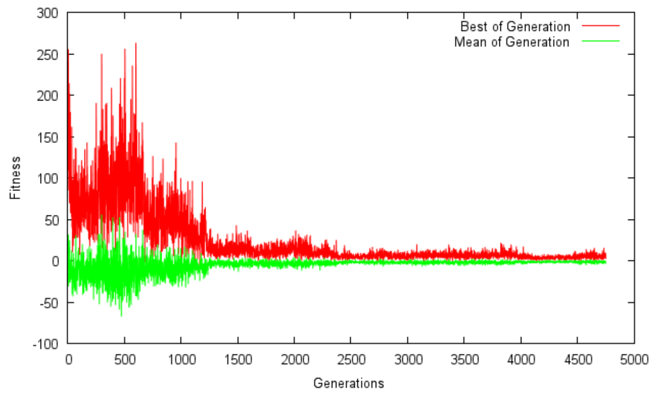


Figure 3.9: An interrupted run of the linear movement experiment, with sub-run starting orientations set to random angles. This run was selected due to particularly anomalous performance. As can be seen, the population does not improve in a consistent manner and the run has been cancelled due to inability to meaningfully improve the population.

3.3.3 Path Following Problem

The path following experiment was a slightly more difficult scenario to produce effective spaceship designs through evolutionary means. The produced ships needed to be able to move quickly along the straight lines between way-points on the path, as well as turn to the next way-point as quickly as could have been managed. Selection pressure favoured ship designs that were capable of moving and turning very fast, and ships that succeeded in this particular scenario tended to be small, with thruster positions allowing for a higher turning moment over a prolonged acceleration in one direction.

Due to the nature of the fixed path tested, it was common to find ships that exploited the shape of the path. This was possibly over-fitting. In particular, given a path where a circular turn is the best course of action for the way-points closest to the start, ship designs emerged that proved adequate at turning a circle, but struggled to proceed onwards from the circular turn.

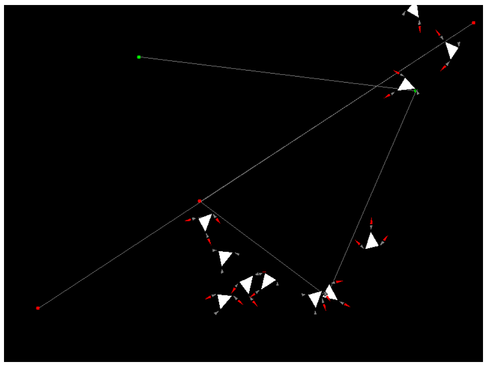


Figure 3.10: Visualisation of a group of ships, 3000 generations into an example run, attempting to follow a path. Red nodes have not yet been visited by any ships, while green nodes have been visited by at least one ship. It is apparent here that the population has converged towards a specific design.

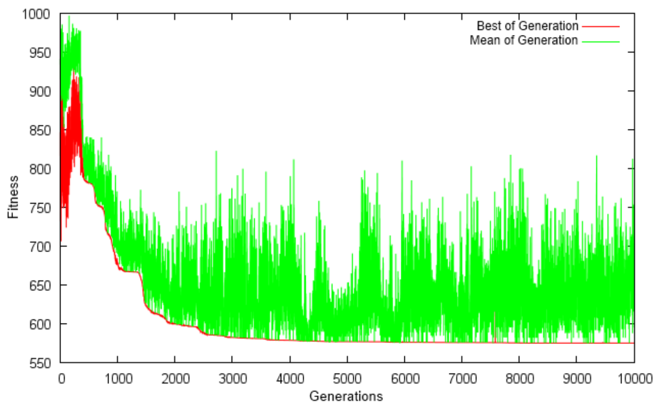


Figure 3.11: Displayed here is an example of a path following run showing high convergence to an effective design. Little improvement is seen once the population becomes saturated with minor variations on a single effective design.

3.3.4 Ship Shapes

There were interesting features observed within ship designs evolved during both the linear movement and the path following experiment. During early generations the ships tend to be small. This was a consequence of the chosen standard deviation of the initial population and how it related to the representation of ship components selected. Over time, they evolved into large bloated ships that were able to orient themselves, but moved slowly. These were typically observable after 500 generations. Then, usually by 2000 generations smaller and fitter ships evolved that were able to use all or most thrusters effectively, but without the excessive mass due to the long struts needed for larger ships. Some examples of the shapes evolved during these experiments can be seen in an online video[2], and an example is also shown in Figure 3.12.

The ships would begin small, as stated, as a consequence due to the initial standard deviation of the population being relatively small. As the representation of the positions of the components of a spaceship were relative co-ordinates in reference to a central anchor point, a low standard deviation for the parameters being used for co-ordinates would naturally correspond to a low standard deviation of potential position offsets from the centre of the ship, and a low magnitude of offsets due to the distribution being centred around zero. Higher standard deviations would also mean a potential for higher magnitudes of offsets, as the range of positions the components could be placed would expand, and as CMA-ES explored the fitness landscape, it would become clear that large, bulky ships were not actually advantageous. The way to reduce the size of the ships would be to reduce the standard deviation for the dimensions of the chromosome relating to ship component positions, and this would lead to the shrinking observed after the initial growth.

However, it is worth noting that, before switching to the lookup-table based controller, the previous 1-ply lookahead controller produced interesting ship designs and behaviours. Spaceships were designed such that they were able to rotate incredibly rapidly and react on a frame by frame basis to whatever action was determined to be the most effective. As the ship rotated, some thrusters would be in the correct alignment to provide an impulse that would move the ship in a desired direction. This proved to be a very effective strategy from the perspective of a 1-ply greedy search controller that aimed to maximise the very next frame's worth of action selection, but was not particularly effective overall. After switching to a more sensible control method, these spinning ships and their associated behaviour were no longer seen, as the lookup-table base controller worked in such a way that constant spinning was not seen as advantageous.

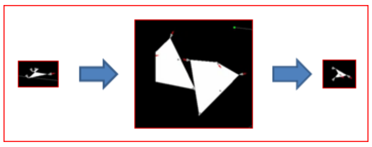


Figure 3.12: Frequently observed evolution of ship shapes over time. In the early generations (left image) ships are small with poorly coordinated thrusters, or missing thrusters due to overlap. After around 500 generations ships are often bloated in size (two sample ships shown in middle image). Then after several thousand generations (right image shows a ships after 5000 generations) small ships are found that are able to use most or all thrusters in effective combinations.

Table 3.4: Comparison of Evolved Fitness to Offline Fitnesses

	Best Offline Fitness	Runs Tested	Runs with Improvement	(%)
Design Objective	7.99	22	22	100%
Linear Movement	1774	46	32	70%
Path Following	738	50	40	80%

3.3.5 Overall Success Rates

Table 3.4 shows the general success of experiments in terms of outperforming the best scores for offline, human-made designs as shown in Table 3.3. For each experimental result, it was considered to be superior to handmade ship designs if the best fitness of the final generation was lower than the given fitness in the table.

It is least surprising that the task most directly tied to numerical optimisation, the design objectives problem, had greater success when solved via evolutionary methods rather than manually. Every run examined produced better designs for the problem than any of the handmade ships. Unique to this problem is the fact that no lengthy simulation or controller is involved, leading to a much closer connection to the input parameter vector and output fitness. This may also be why this problem was so successful.

The best handmade design for the linear movement problem outperformed 30% of runs, which could either indicate that the offline designs represented a better solution attempt for this problem than for others, or that the problem was slightly more difficult than expected for there to be inferior experimental outcomes. A 70% success rate suggests that the approach taken to ship designing through evolutionary progress still had considerable merit.

For the path following problem, it is possible that the reliance upon a single path led to overfitting in the designs, allowing the evolutionary process to produce ships designs far more suited for the specific given path than the more general offline designs. However, ultimately, these results showed that the best designs produced for these problems were generally performing much better than a human designed ship.

3.4 Preliminary Conclusions

This completes the examination of the basic problem of designing spaceships based upon the three experiments detailed earlier. A video of some of the results of the experiments discussed in this chapter is available online, as has been mentioned previously.

CMA-ES was found to be an effective algorithm for evolving spaceship designs in accordance with the specified fitness functions, using a vector of real numbers as the means for encoding spaceship components.

[2]http://www.youtube.com/watch?v=3eAEbizMVPw

However, it was necessary to use a fixed set of initial spaceship orientations, since random orientations meant that the signal to noise ratio was too low when measuring fitness. The resulting spaceships could potentially be used directly within an *Asteroids*-style video game, providing a rich variety of enemies or collaborators who can be shot at and partially disabled in interesting ways, but only the basic movement functionality of such spaceship designs was observed in any meaningful capacity within this work.

In particular, variations of behaviour given the problem environment, the representation used and the control method used for testing were all shown to have dramatic and varied impacts on the nature of designs that arose, as well as the behaviours of the spaceships being tested. While the representation seemed sufficient, it was not entirely clear if the control method used, the lookup-table based controller, was necessarily sufficient for more complex tasks. While it was clear that the designs evolved for spaceships could compensate for weaknesses or complement strengths of the controller used, it was not readily apparent that the controller could adapt to more complicated problems, and so more difficult problems were examined, and will be discussed in the following chapters.

From these initial results, however, it was clear that it was both feasible to generate spaceships for simpler problems using CMA-ES, and from this base further experiments were performed, starting with an increase in the complexity of the problem being examined. The question to answer could essentially be phrased as this; given it was possible to generate designs for spaceships in this problem, what more would be required for generating spaceships in a more complicated scenario?

Chapter 4

Vehicle Generation Under Varying Problem Definition

4.1 Lunar Lander Scenario

4.1.1 Continuing from Simpler Scenarios

In this chapter, we continue the previous work on evolving spaceships for simpler problems by extending the scope and complexity of the problem that the ship designs must be able to solve. While the previous work focused on very limited mobility problems that were almost trivial to solve, within this chapter we examine a more complicated problem based upon Atari's *Lunar Lander* game. Within this scenario, vehicles exist within a two-dimensional physical space as in earlier experiments. This space has a vertical downwards gravity force, much like *Lunar Lander*, as well as a basic 'ground' surface constructed of multiple line segments. It is the purpose of the vehicles to navigate successfully from a starting point to a flat landing pad, and to land as slowly as possible onto this landing pad.

4.1.2 Why Lunar Lander?

Initially, the *Lunar Lander* scenario was selected as an example of what seemed to be a simple extension of the earlier movement based experiments. As the purpose of the game is based entirely on movement, and the generated designs were selected and improved for optimisation of movement and no other quality, it seemed a logical progression to move from a problem of simple motion to a problem of more intricate motion.

However, as work progressed, it became clear that the *Lunar Lander* scenario was far more complex than it had at first appeared. The focus remained vehicle generation for the initial efforts to study this problem, but it soon became clear that the difficult nature of the scenario itself was worth examining in its own right. During this chapter, the initial efforts to generate vehicles for this problem will be discussed, and then the efforts to generate movement paths for vehicles of fixed design. Finally, both vehicle generation and fixed behaviour generation are examined together with the properties of the scenario itself, as a complete investigation into what makes the scenario itself so difficult.

4.2 Environmental Properties

The basic environment of the *Lunar Lander* scenario consists mostly of a two-dimensional space and a landscape. Vehicles that moved out of horizontal bounds were not moved to the opposite horizontal edge of the space, as in some *Lunar Lander* implementations, for the simple reason of allowing for a more linear measure of ship proximities further ahead. The co-ordinate space used within this environment defined the origin to be at the top-left of the world, in the same way as defined for screen display spaces.

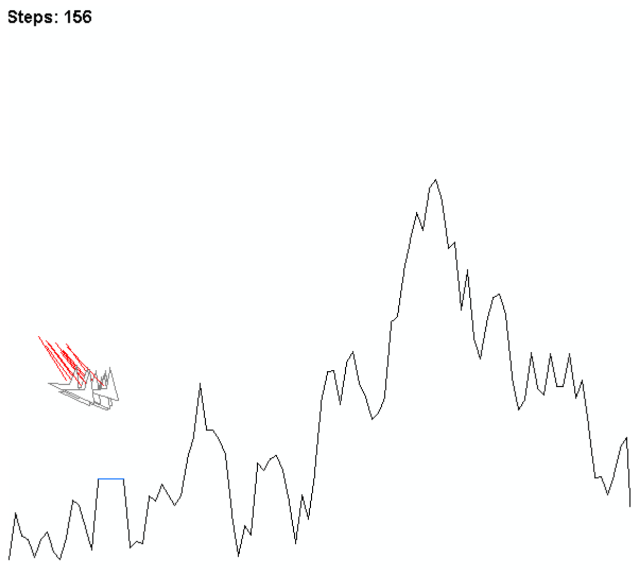


Figure 4.1: A view of ships within this environment.

4.2.1 Physics of the Experiment

The original game provided players with a spaceship that was capable of rotating clockwise, rotating counter-clockwise and of adjusting the level of thrust used to move the ship in the direction it currently faced. The ships in the experiment do not have the capability to directly alter their own linear and angular velocities, and must instead toggle and adjust the power of a symmetrical collection of thrusters mounted on them. All thrusters can affect the linear velocity of the ship in some manner, based on relative position and orientation from the ship's centre of mass. Thrusters offset from the centre of mass for the ship that are not directly pointed at or away from the centre of mass can also produce angular impulses.

4.2.2 Generation of Terrain

The landscape was generated through a fixed random seed that is determined during experimental set-up. Terrain was generated as a series of vertices placed randomly along a specific range within a lower section of the screen. Each vertex was horizontally equidistant, which meant that, given an index within an ordered data structure such as an array, only the y co-ordinates themselves needed to be stored, with the x co-ordinates implicit based on the order of y co-ordinates and the index of a given y co-ordinate. The terrain itself was handled as a series of line segments between the specified landscape vertices for collision calculations and graphical display, and considered to be solid at any point beneath the jagged line created by these line segments.

The vertical component of the first vertex was defined to be a fraction of the world height, c. Each subsequent vertex had a vertical component based on a random number taken from a scaled normal distribution, added to the vertical component of the previous vertex's vertical component. For a number of points, n, and a world width w and height h, the initial starting vertex $\boldsymbol{p}^0$ was defined as:

$$\boldsymbol{p}_x^0 = 0 \tag{4.1}$$

$$\boldsymbol{p}_y^0 = c \tag{4.2}$$

The arbitrary point $\boldsymbol{p}_i$, where $i \neq 0$, and σ was the scale factor, was defined as:

$$\boldsymbol{p}_x^i = \frac{iw}{n} \tag{4.3}$$

$$\boldsymbol{p}_y^i = \boldsymbol{p}_y^{i-1} + \sigma\mathcal{N}(0,1) \tag{4.4}$$

This process was repeated until the entire vector of y values was populated, forming the landscape. Once this had been created, the landing pads are generated through a similar procedure, but were instead represented as pairs of values; the initial index or scaled x component of the point where the landing point begins, and the length in points of the landing pad. These initial indices were selected randomly, with the lengths predefined during the experiment setup, and were created simply by setting the vertical component of all landing pad points to be equal to the initial landing pad point, effectively flattening them. In the event a landing pad would begin at the right of the landscape and be too long to fit on the landscape, it wrapped around to the other side of the landscape.

In order to allow the collision of vehicles with the landscape, it was necessary to be able to determine whether an arbitrary co-ordinate fell within the lunar landscape. Firstly, it was determined whether or not the vehicle was currently present within the horizontal range of the landscape, which was a simple boundary check on the horizontal component of the vehicle's position. After confirming the vehicle was within the horizontal boundaries of the landscape, the next condition to check for collision was whether the lowest vertical point of the spaceship's bounding circle happened to be inside the landscape or not.

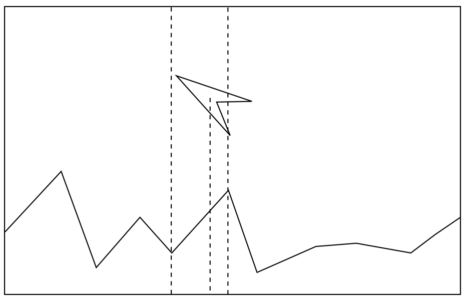


Figure 4.2: The point closest to the spaceship on the landscape lies in between two of the defined vertices of the landscape, and requires interpolation to calculate the y co-ordinate from the x co-ordinate of the ship.

This could be achieved by determining the closest leftmost discrete point of the landscape to the point being queried, and interpolating the y component of the closest landscape point from there. For a given query point of $\boldsymbol{q}$, the equivalent index i in the vector of points of length n could be determined simply through $floor(\frac{\boldsymbol{q}_x}{n})$. The interpolation value, v, was equal to $\frac{\boldsymbol{q}_x}{n} - i$. From there, the landscape point $\boldsymbol{l}$ that corresponds was:

$$\boldsymbol{l}_x = \boldsymbol{q}_x \tag{4.5}$$

$$\boldsymbol{l}_y = \boldsymbol{l}_y^i + v\boldsymbol{l}_y^{(i+1) \bmod n} \tag{4.6}$$

In addition to the physical properties of the landscape, the environment could also act upon entities within it with a gravity force and a velocity loss factor, both of which were defined before the experiment began. Gravity was simply an addition to the vertical component of any entity capable of movement each step, which included all vehicles, while the loss factor was any value n where $0 \leq n \leq 1$. Both the linear velocity and angular velocity of any entity in the environment was multiplied each frame by this loss factor, allowing for modelling a resistance factor of the environment, such as an 'air resistance', or no resistance at all. While it is acknowledged that the lunar landscape is not known for its atmospheric resistance, such a feature can be considered a gameplay construct.

4.3 Vehicle Physics and Composition

Within this environment, a vehicle has the same physical properties as those described in the previous chapter. At the beginning of a simulation, every single one of a ship's physical values will be set to a constant default, allowing for entirely deterministic outcomes given the same deterministic controller. The vehicle is capable of both turning through angular impulses and moving through linear impulses, as before.

4.3.1 Collision Detection

In the original Lunar Lander, the collision detection is extremely accurate between the spaceship and the terrain. There is very little leeway given, as if the ship collides with the terrain at all but the lowest velocities possible, the ship is destroyed upon contact. The shape of the spaceship is simple enough to allow for a high level of accurate collision.

For the purposes of the experiment, spaceships were instead given a bounding circle around their central origin, which is not the same as the centre of mass for the majority of component based ships. This bounding circle was not the circle that enclosed all components but a circle with a radius equal to the mean of the distances between each component and the origin of the spaceship. When this bounding circle intersected with the landscape, it was treated as a collision. If the ship was moving too fast, it was destroyed.

The vehicle collided with the landscape based upon the lowest point of its bounding circle colliding with the highest point, $\boldsymbol{l}$ of the landscape on the same horizontal axis. Collision occurred if the following condition was true:

$$\boldsymbol{s}_y + r \geq \boldsymbol{l}_y \tag{4.7}$$

If this condition is true, the ship has collided with the landscape, and has either landed or crashed based on the survivable velocity threshold determined for the experiment. A simulation where the ship had landed or crashed was immediately halted and scored. A successful landing for this problem requires landing on a landing pad, and the scoring rewarded landings on landing pads alone.

Each vehicle is supplied with an initial amount of fuel, which can vary based on experimental set-up. This fuel is consumed each and every step of the simulation based upon the amount of *forward* thrust applied to the vehicle. Rotational impulses do not reduce fuel, as in *Lunar*

Lander. For a given thrust amount, the amount of fuel consumed per step is exactly equal to the thrust, factoring in the thrust limit.

4.3.2 Composition of Vehicles

A vehicle is composed of a central body and a number of components, which can vary but in this and previous experiments have simply been thrusters. The central body has a mass and moment of inertia, as do its individual components. As a component is added, an amount of mass is added in the form of a "strut" between the local origin of the vehicle and the component. This leads to vehicles with components that are spaced out to end up more massive than vehicles with tightly clustered components. The thrusters have a minimum thrust value that cannot be reduced below and a maximum thrust value that cannot be exceeded. A thruster will produce thrust equal to whatever thrust value it has been assigned by a controller.

4.3.3 Chromosomal Representation

The chromosomes for the vehicles were slightly altered for this experiment. In an effort to compensate for increasing the complexity of the problem significantly, the chromosome of the ship was now considered to represent only half of the vehicle components and their position and rotation relative to their parent vehicle. The phenotype corresponding to a given genotype would mirror the creation and attachment of each component along the ship's local Y axis, creating a horizontally symmetrical ship with enforced symmetry. In effect, the chromosomal structure for the vehicles remained identical to previous experiments, with the exception that now only half as much data was required to produce a vehicle with an arbitrary even number of components.

As before, a thruster is defined with three values. The first two values are a local x- and y-co-ordinate relative to the vehicle's local origin or geometric centre. The third is the fixed local rotation relative of the thruster to the vehicle's own rotation.

4.3.4 Fitness Measure

The reward given to a ship for each generation was totalled after a number of runs. At the start of an individual ship simulation, a ship was given a finite amount of fuel which is expended based upon the active thrusters and the intensity of their thrust. Once the ship runs out of fuel its simulation is halted.

Ships were highly rewarded for colliding with one of the two landing pads as slowly as possible. If a ship collided with the landscape, it was still rewarded but not as highly. Reward increased as velocity approached zero. If the ship collided with the landscape at higher than a survivable velocity, the ship was destroyed and was punished in proportion to how high its velocity was above a survivable amount.

The fitness measure was soon improved to compensate for flaws, with the main error discovered being that ships were penalised for crashing but not for never landing. As a result, this comparatively better outcome was selected for and promoted by the measure. To correct this, ships were given a small reward for landing at all, even erroneously, to promote designs that would land to begin with. This kept roughly the same dynamic as the punishment-reward measure from before but with a better mark of progress being made.

4.4 Controller Details

4.4.1 Calculating Actions

The controller for lunar lander ships was kept constant across all ships, as with the initial experiments described in the previous chapter. The controller created a look-up table of sets of thruster states corresponding to specific actions.

As previously mentioned, thrusters were initially created in a symmetrical fashion for the ship. This look-up table was created by taking every possible combination of thrusters being active or inactive, with a constant thrusting force identical to all thrusters, and determining what the resultant velocity and torque for the ship would be for the next time-step of the simulation.

The combination of thrusters that provided the highest amount of clockwise and counter-clockwise torque were stored as the thrusters to activate when the ship controller attempted to turn clockwise or counter-clockwise. Likewise, the combination of thrusters that provided the highest amount of upwards (negative y) velocity with the ship in its initial orientation were tracked as the thrusters responsible for providing forward/upward motion.

In practice, these thrusters would be largely kept active throughout most of the run, but the controller was capable of adjusting the thrust power of these thrusters in fixed amounts either way. Reducing the thrust to zero would have the same resultant effect as deactivating the thruster. Thrust could not be reduced below zero.

Each step of the simulation, the controller could select to perform an action out of several available actions. These actions were to do nothing, to turn clockwise, to turn counter-clockwise, to increase linear thrust amount or to decrease linear thrust amount.

4.4.2 Macro-Actions

Due to the complexity of the problem and the problem of combinatorial explosion faced when dealing with a comparatively analogue problem space, with positions stored as co-ordinates of real numbers and not as, for example, discrete grid cells, examining the problem with controllers taking one action per simulation step would not lead to results particularly quickly or effectively.

Instead, the concept of a "macro-action" was used. A macro-action is, most simply defined for the context of this experiment and future experiments within this thesis, an action that is repeated over multiple time-steps. The function this serves is to greatly reduce the size of the problem space, as the number of decision points during the simulation will fall at the expense of fine granularity.

A macro-action was composed of three values, set up as shown in Figure 4.3. These values are the linear impulse applied to the ship each frame the action is active, the rotational impulse applied, and the number of timesteps the macro-action was active for. Once the macro-action has been executed for a number of frames either equal or greater to the number of timesteps it should be active for, the controller would use the next macro-action in the set of macro-actions provided to it through evolutionary means.

The number of time-steps can either be a parameter of the action or it can be assumed to be kept constant per macro-action. In this thesis, both approaches were used, as both have their strengths and disadvantages. Allowing the time to vary for each macro-action allows for finer resolution if required, but it also adds additional complexity to the problem itself as yet another problem dimension to be resolved. Keeping the length of macro-actions uniform resolves this problem, but can theoretically mean fewer solutions that are effective for the problem.

In the context of this experiment, the time taken per macro-action was varied per action. Later experiments would move to uniform-length macro-actions, in efforts to reduce experimental complexity.

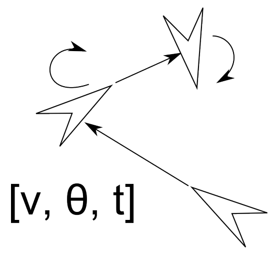


Figure 4.3: Each set of three doubles determines the linear thrust, the angular impulse and the amount of steps to repeat the action for. Upon finishing the action, the next action is used. When the last action is used, the first action is used afterwards and the process repeats.

These vectors of macro-actions constituted the solutions evolved within the constraints of the outer problem, which in turn had its parameters set through evolution. There were, in effect, two nested evolutionary processes at work during any given experiment. An outer evolutionary process generated problem candidates while an inner evolutionary process attempted to provide successful action sequences for ships within the context of said problem candidates.

4.4.3 Heuristic

The original heuristic simply attempted to minimise the distance between a ship and the closest landing point, and to minimise velocity as distance decreased. This did not lead to effective ship behaviour, however, as the heuristic had insufficient data and could not determine whether a ship would, in a given state, be about to crash into terrain horizontally or whether the ship had crashed at all. It was replaced with a better, although still flawed, heuristic, that produced behaviour better able to achieve the goal of landing safely.

When not within a short distance of the landing pad, the improved heuristic rewarded states for being close to the nearest landing pad, but also discouraged close horizontal distance to the landscape and close vertical distance to the landscape. This was achieved through the use of a logical grid placed over the field of the experiment. Grid cells that contained any terrain were considered unsafe, and both horizontal and vertical distances from the closest unsafe grid cells on both axes were summed to emphasise that both must be kept as large as possible.

When within proximity of the landing pad, the focus of the ship was to reduce its velocity and attempt to right itself, as this would grant it greater control over its thrust vector.

4.5 Initial Results and New Approaches

4.5.1 Experimental Setup

The parameters used for the physics and environment for initial experiments are given in Table 4.1, and many of these values were later used as defaults for future experiments detailed in this chapter. Macro-actions were represented as a chromosome of 12 values, for 4 macro-actions of 3 components each, as previously explained and shown in Figure 4.3.

Two fitness measures were used to evaluate the overall success of a candidate solution to the problem, expressed as a set of macro-actions executed in looping sequence. When the end of the sequence was reached, the sequence would begin again, and this would continue until landing or the end of the alloted simulation time.

The *velocity measure* simply measured the velocity of the ship with respect to a target limit, given as the survivable velocity threshold in the table of parameters, and, for a magnitude of velocity $|\boldsymbol{v}|$ and a survivable threshold v_t, the velocity score f_v was calculated as

$$f_v = \max(0, |\boldsymbol{v}| - v_t) \tag{4.8}$$

where the fitness f was simply calculated as $f = f_v$.

The *proximity measure* was not used in isolation, and was instead added to the total fitness score along with the velocity measure for these early experiments as an earlier attempt to minimise both conditions. The proximity measure was used to measure the proximity of the ship to the nearest landing pad, to reward landing in the intended spot. A vector was found from the vehicle's origin point, $\boldsymbol{s}$, to the centre of nearest landing pad, $\boldsymbol{l}^p$. This measure, f_p, was calculated as the horizontal distance between $\boldsymbol{s}$ and $\boldsymbol{l}^p$, and can be written as

$$f_p = |\boldsymbol{s}_x - \boldsymbol{l}^p_x| \tag{4.9}$$

where the total fitness for using both the velocity and proximity measures together was

$$f = f_v + f_p \tag{4.10}$$

As the goal was to approach *Lunar Lander* solution behaviour, simply using the proximity measure alone was not considered as there was no consideration for slowing down inherent to the reward function.

Table 4.1: Initial Problem Constants

Constant	Value
Number of Landscape Vertices	100
Number of Macro-Actions	4
Maximum Timesteps Per Run	400
Maximum Function Evals	1000
Maximum Generations	50
Population Size	20
Initial Solution Parameter σ	10
Thrust Limit	20
Gravity	20
Velocity Loss Factor	1.0
Landscape Seed	104353
Number of Landing Pads	1
Landing Pad Width (Vertices)	5
Surivable Velocity Threshold	15

4.5.2 Initial Heuristic

The original controller heuristic lead to poor performance of each individual ship, as most ships would attempt to simply fall towards the landing pad and were unable to provide satisfactory responses to horizontal distance from the closest landing pad.

As can be seen in Figure 4.4, the original heuristic proved to result in ship behaviour so ill-performing that there were no successes from which to derive improved populations over time. The fitness of the population moved around erratically, and most other runs of the experiment failed to produce any meaningful data for more than three generations or, in some case, any data at all. The original fitness function also penalised ships for crashing but did not penalise

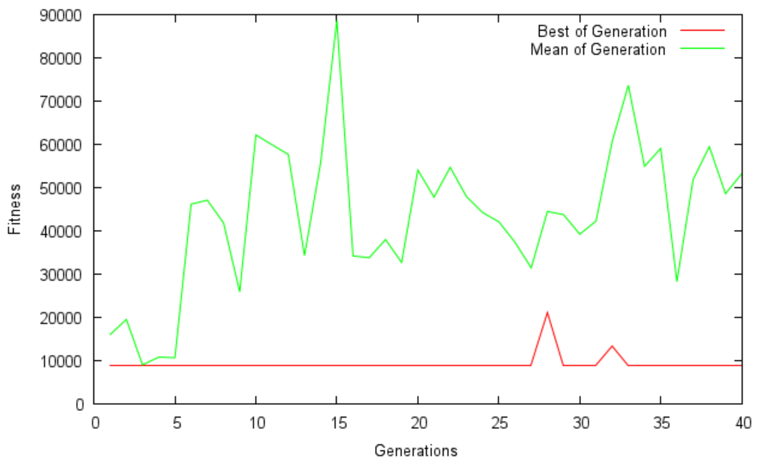


Figure 4.4: Results from original heuristic and fitness function, where lower score is better. The experiment these results come from was selected as a sample randomly. As is visible, efforts to explore the available search space lead to the mean fitness of the population actually decreasing over time due to wider exploration of the search space, with no improvement for the best ship fitness.

ships for failing to land at all, resulting in even less progress when coupled with the original heuristic.

4.5.3 Improved Heuristic

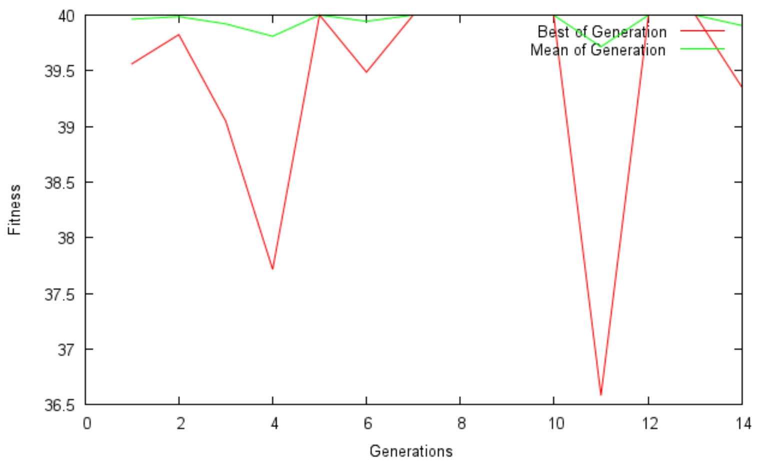


Figure 4.5: Results from changed heuristic and fitness function, where lower score is better. This is a randomly selected sample run. The addition of more heavy penalties for undesired behaviour made the problem far more difficult to resolve and ended up in many CMA-ES runs being abandoned early due to lack of apparent improvements through evolution, as shown here.

Unfortunately, despite the improvements and modifications to the code, there still seemed to be little to no impact on the effect of the fitness of ships over time. The changed heuristic appeared insufficient to produce any immediately noticeable improvements to the results of the experiment.

4.6 Change of Focus

It was around this point that a change of focus was made regarding what the problem to be solved was. While there remained ill success in attempting to find vehicle design solutions to the *Lunar Lander* scenario, it was hoped and expected that focusing on finding specific movement paths with regards to a fixed design would firstly demonstrate that the problem was solvable within the defined environment and secondly indicate what the characteristics of movement that solved the problem were.

4.6.1 Changes to Vehicles

As the focus became that of evolving a set of movement actions, and not the ship itself, there were some minor changes to the way vehicles worked within the simulation. Every ship was now modelled as a point mass as before, but with no components attached to it.

All linear forces applied to the ship were applied as a single value, causing a linear acceleration based on a current heading or direction of the ship. This thrust value, a_T, was restricted according to two rules. Firstly, linear force applied to the ship could not be negative, and a thrust value below zero was corrected to zero. Secondly, the thrust value could not exceed the thrust limit, a_{max}, as defined in the experiment. Any thrust value that would exceed this value was instead capped to this value. The thrust value therefore always met the condition of $0 \leq a_T \leq a_{max}$.

The use of fuel was also affected by this. An amount of thrust that was already capped by the thrust limit would consume an amount of fuel equal to the thrust cap. There was no possibility of wasting fuel by any actions that would result in attempting to exceed the thrust cap.

4.6.2 The New Problem

As mentioned before, this experiment hinged around a *Lunar Lander* inspired scenario, involving ships navigating a two-dimensional space with a gravitational force. The main focus was not simply to land on a landing pad, but to do so while also minimising landing velocity and avoiding landing (or crashing) elsewhere. For each generation or iteration of the experiment, a population of vehicles were simulated starting from a fixed starting point, and scored based on a fitness measure upon either landing or the simulation reaching its maximum allowed length. A secondary fitness measure was used in all contexts if a vehicle does not land, scoring the proximity of the ship to the landscape in an especially penalised manner. The fitness measure was formulated such that the reward for never landing was far reduced in comparison to the reward for landing poorly, thus making the focus of the experiment on optimal landings.

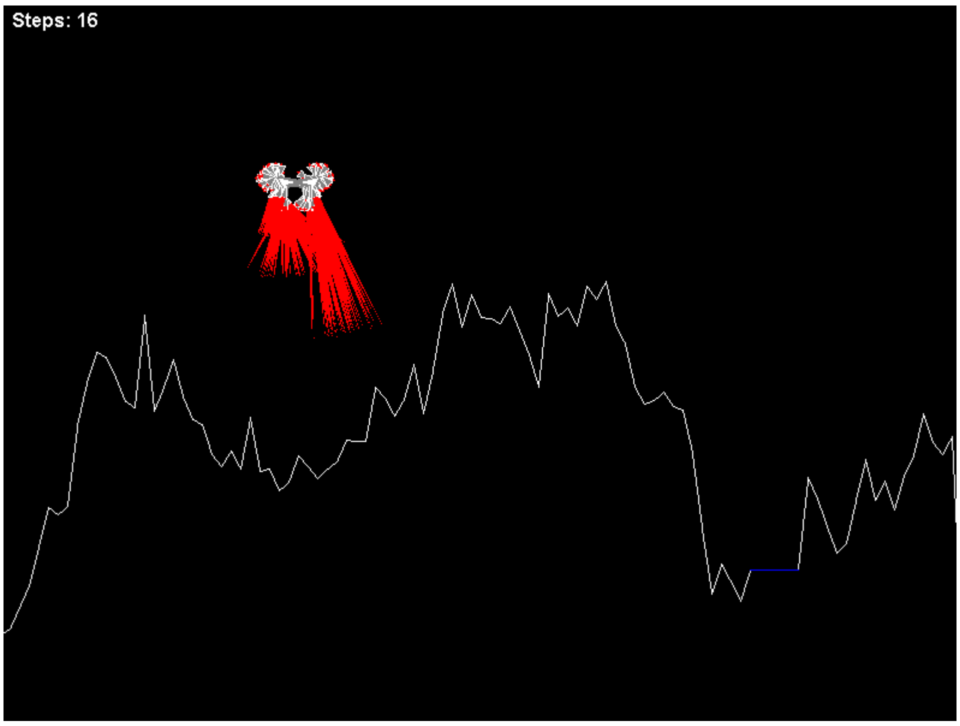


Figure 4.6: The landscape used for the experiments, with initial starting point and blue landing pad visible.

4.6.3 Fitness Function Variation

Two significant fitness measures were tested for this experiment. The first was a cut-down version of the second fitness function, which only rewarded vehicles for landing with a velocity as close to zero as could be possibly achieved. The second fitness measure, in addition to rewarding for a minimal velocity, rewarded for a minimal distance to the landing pad. Due to the wrapping nature of the simulation space, the lowest of the two possible distances was the one selected as the distance from the vehicle on landing to the landing pad.

These two fitness measures cannot be fairly compared directly against each other, as the second fitness measure would produce higher error due to the dual conditions being optimised for.

A more complicated fitness measure involving the above conditions as well as minimising fuel consumption was tested, but did not lead to any results of note, with problem strategies being unable to improve on the worst possible result for multiple iterations. The addition of more conditions to the fitness measure appeared to cause the experiment to break down, and the exact tipping point for this failure was one of the things investigated in later research, appearing later in this chapter.

4.6.4 Controller Variation

The main focus for controllers in these experiments were incredibly simple sets of macro-actions. Three sets were trialled. It is worth noting that although the evolved values include negative numbers, in practice they equate to effectively switching off the thruster, as the force of a thruster is set to 0 if it is ever set to a negative value.

Of these three sets, the first set was a handmade set of totally arbitrary thruster values for each step. The values were chosen based on observed trends in previous sets of generated macro-actions and were not intended to be particularly successful. As predicted, vehicles controlled by this set of macro-actions performed poorly. The second set was generated through an experiment evolving the set of macro-actions to control a vehicle that had been hand designed, which lead to better performance. The best performing set of macro-actions was the third set, which had been generated through an identical experiment with a previously successful vehicle design used as the base.

In further experiments, the fixed sequence of macro-actions to be used was evolved in addition to the representation of the vehicle design. As the values of thrust tended to be two orders of magnitude greater than typical successful ship designs, in these experiments the representation used for the ship components was made two orders of magnitude larger than normal, with simple scaling applied upon constructing the ship for simulation.

Table 4.2: Results from a number of runs, using different fitness measures to evaluate suitability of solutions to the lunar lander problem.

Controller	Fitness Measure	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
Static macro-actions	Vel.	12	38	12	10	23	19
Static macro-actions	Vel., Prox.	330	371	330	330	350	342
Evolved from handmade	Vel.	0	0	143	0	0	29
Evolved from handmade	Vel., Prox.	144	24	24	144	24	72
Evolved from evolved	Vel.	0	0	0	0	0	0
Evolved from evolved	Vel., Prox.	1	25	25	25	25	20

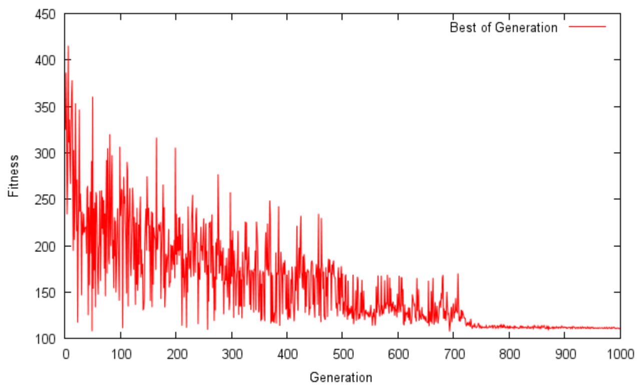


Figure 4.7: Fitness over generations for coevolution, where values closer to zero indicate more effective solutions. Randomly selected sample run.

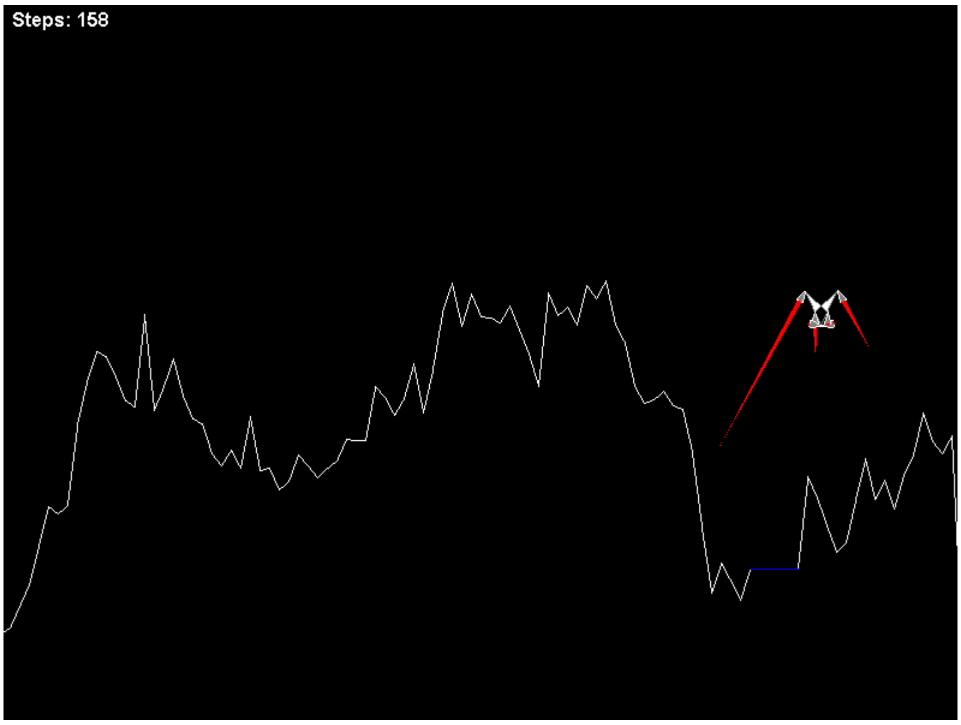


Figure 4.8: Example successful ship designs in mid-flight towards landing pad.

4.6.5 Success of Control Sequence Evolution

While initial efforts to produce ship designs that would naturally lend themselves to the task of landing safely may have not led to any satisfactory conclusions, the efforts to produce any working solutions to the problem of landing a ship in this scenario were successful. The evolution of sequences of macro-actions allowed for ships to start at an arbitrary position and land on the landscape while meeting the criteria for a successful landing.

Table 4.2 displays the outcome of some of these experiments. Static macro-actions were designed by hand based on an expected potential solution to the problem. The outcome of this macro-action series is given in the table as "static macro-actions".

These actions were used to determine the initial population of a population of candidate solutions. This population would still be adjusted by an initial uniform randomness across the dimensions of the macro-actions, but it would serve effectively as a multi-dimensional region of the fitness landscape to begin exploring and exploiting. The results for this experiment are given as "evolved from handmade".

This process was repeated with the best of the evolved solutions becoming the basis for a new set of experiments, with the results given as "evolved from evolved". As can be seen in the table, as more evolutionary progress is made, more suitable series of macro-actions are produced.

As seen in this table of results, the existence of any runs with an error of 0 indicated a perfect success for the fitness measure being used, meaning at the very least the minimisation of velocity to below an acceptable level was feasible. The iterative process of evolving multiple macro-action sequences from each other demonstrated that an initial error of 342 for the combined velocity and proximity problem could be reduced to 20, a considerable improvement even in a small number of runs.

This demonstrated that this approach had not yet found perfect solutions, but that it was able to reduce error through evolutionary iteration. For this reason, the approach was considered successful enough to be used as the internal basis for focusing on a new problem to determine how the success rate of this approach could be improved.

4.7 Investigating Problem Environments

Having determined that the problem was solvable by approaching it from a different, simpler perspective, the next step to take seemed to be an investigation into what exactly made the problem so difficult. Initial thoughts suggested that certain aspects of the problem environment and its physics might be related to what made this particular problem so difficult to solve with the approaches used, and these parameters and constraints of the environmental physics became the next focus of research.

This investigation, this error is determined by the fulfilment of certain sub-conditions. These sub-conditions, and their relation to the performance of the generated solutions, in turn affect the difficulty and interest of the problem, as do the features of the problem environment. As the elements of a problem are investigated, it becomes increasingly clear that the combination and weighting of elements of a suitability measure have a drastic impact on the difficulty of a problem and the effectiveness of the solutions.

To investigate the difficulty of the problem at hand, a measure of 'interestingness' was considered as a way of measuring problem difficulty by means of the rate of convergence of an evolutionary algorithm attempting to solve it. More details are given later in defining the measure itself, but understand 'interestingness' to refer to this specific idea of solution progress for a given problem.

4.7.1 Vehicle Control and Solutions

The solutions provided to the problem consisted of a vector representative of a set of macro-actions. The controller for the vehicle simply used the set of macro-actions provided through evolution to control the vehicle. The success of the vehicle in landing was entirely dependent on the suitability of the evolved set of macro-actions as a means of controlling the vehicle.

A set of macro-actions was represented as a vector, where every set of three values in the vector represented one action, as shown in Figure 4.3. Once the last set of three values was executed, the controller would wrap around to the beginning of the vector and continuing executing the macro-actions from the first defined macro-action. This would repeat until the vehicle had landed or the simulation had terminated after reaching the maximum allowed number of timesteps.

4.7.2 Solution Suitability Measure

The suitability measure used had a small but significant set of sub-conditions with associated weights. These measures tally the error of the current situation relative to a hypothetical perfect solution, and produced values intended to be minimised as opposed to maximised.

Velocity

The first of these sub-conditions measured the velocity of the vehicle upon landing. The magnitude of the velocity, $|\boldsymbol{v}|$, penalising all velocity above the survivable velocity threshold, v_t. Given the weighting w_v, the contribution, c_v to the final score was

$$c_v = \max(0, w_v(|\boldsymbol{v}| - v_t)) \tag{4.11}$$

As the desired behaviour is to minimise the landing velocity, no transforms need to be made for this equation.

Proximity

Another sub-condition measured the proximity of the vehicle to the nearest landing pad upon landing. The priority for this sub-condition was to minimise the horizontal distance between the vehicle and the nearest landing pad. The distance to the nearest landing pad's centre was calculated as $\boldsymbol{l}^p$. Given the weighting w_d, the contribution, c_d to the final score was

$$c_d = w_d|\boldsymbol{s}_x - \boldsymbol{l}^p_x| \tag{4.12}$$

Landing Angle

This sub-condition measured how close the landing angle of the vehicle was to the ideal angle of being perfectly perpendicular to the flat landing pad, with a heading facing vertically upwards. To calculate this, a unit vector representative of the ideal landing heading, $\boldsymbol{h}$, was rotated by the ship's rotation θ, to give the rotated heading vector $\boldsymbol{h}^\theta$. Given the weighting w_a, the contribution, c_a to the final score was

$$c_a = w_a(\arccos(\boldsymbol{h} \cdot \boldsymbol{h}^{\theta})) \tag{4.13}$$

Fuel Consumption

This last sub-condition measured the efficiency of the fuel usage in a simulation. The priority for this was lowering fuel usage. It was simple to calculate given the fuel consumed, f, and the initial fuel amount, f_0. Given the weighting w_f, the contribution, c_f to the final score was

$$c_f = w_f(f_0 - f) \tag{4.14}$$

Total Effectiveness

Given these component objectives, the final measure of the effectiveness of a solution, f, was

$$f = c_v + c_d + c_a + c_f \tag{4.15}$$

Alternate Measure

In the event that a solution did not result in a spaceship colliding with the landscape, an alternate measure was used. As the aim was to minimise the error, this value was enlarged to encourage any solutions that resulted in landings instead of aimless movement in the space above the landscape for the full length of allotted timesteps. As such, the effectiveness of a solution f, when the ship had failed to land, was defined based on the absolute distance of the y co-ordinate of the ship, $\boldsymbol{s}_y$, to the y co-ordinate of the landscape, $\boldsymbol{l}_y$, on the same vertical line of the ship. Hence, f became

$$f = 100|\boldsymbol{s}_y - \boldsymbol{l}_y| \tag{4.16}$$

By multiplying by this factor, even the worst results of landing with the landscape became an improvement to the case where spaceships did not even land. This encouraged solutions which land, and hopefully encouraged better landings as a direct result.

4.7.3 Problem Representation

Given the above measures all applied to the inner *Lunar Lander* inspired problem, the outer problem constituted finding a set of environmental parameters that led to worthwhile evolutionary progress in solving the inner problem. These environmental parameters were represented as a vector of doubles that corresponded to specific parameters, based on their ordering. These doubles were then transformed into values more appropriate for their corresponding parameters. There were thirteen parameters in total that varied and were adjusted in the course of evolution. The parameters that were varied are those included in Table 4.4, as well as the weightings on the four sub-conditions discussed earlier. The weightings for these conditions defaulted to a value of 1 for all sub-conditions, save for the fuel conservation sub-condition, which defaulted to a value of 0.

4.7.4 Problem Interestingness Measure

A candidate problem was evaluated through simple statistical analysis of the progress of evolution that took place according to the parameters of said candidate problem. Firstly, an initial baseline population of 1000 random individuals was generated and evaluated with the environmental parameters as specified for the problem candidate. From the fitness scores of these individuals, a mean fitness for this random population was calculated, $f_{\bar{u}}$, which approximated the general performance of unimproved solutions to the problem. Also calculated was the initial standard deviation of the population fitness, σ.

Given this baseline fitness measure, CMA-ES was used to improve a population of individuals in the standard manner, attempting to find as optimal a solution as could be found to the problem within a predetermined number of fitness measure evaluations, as specified in Table 4.3. Upon completion, the best discovered fitness, f_b, was stored. The interestingness of the problem could be calculated with these values based on the number of standard deviations of improvement from the baseline performance fitness. This simple measure of rate of solution progress within a problem, f_p, was

$$f_p = \frac{f_b - f_{\bar{u}}}{\sigma} \tag{4.17}$$

This measure was used as it was a simple and intuitive measure of the progress made in improving solutions to the problem. A higher number of standard deviations of improvement implied a higher level of improvement. If the numerical value of the standard deviations was low, this implied that the problem was incredibly difficult to solve through random generation, as opposed to through a guided process such as evolutionary search. A high amount of improvement suggested that there was a high level of improvement that could be made with a more guided attempt to solve the problem, making it an interesting problem to investigate, as well as indicating that it was a problem wherein noteworthy progress could be made.

This measure, f_p, is referred to as the “interestingness” of a problem candidate. The name “interestingness” is, by nature, incredibly subjective as well as perhaps vague. However, this name is chosen to refer to specifically this particular context; problem candidates that score highly according to this measure are considered interesting problems in terms of capacity for improvement of solutions during the course of evolution. In a vacuum, this name is of course less meaningful, but for the sake of a useful shorthand, and for the rest of this chapter, this measure is referred to as the interestingness of a problem. It should be clarified that this is not necessarily a measure of how interesting a problem is outside of this context!

For each problem candidate, the final measure of the candidate's interestingness was averaged over three runs to prevent random chance producing misleading outlier results.

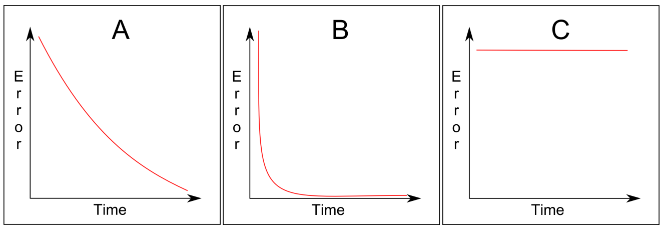


Figure 4.9: An interesting problem should produce a curve like graph A. A trivial problem will produce a curve like graph B, and will not score highly on the interestingness measure as the initial random fitness will likely be much higher, making the relative progress smaller. Too difficult a problem will produce a curve like graph C, and will not score highly as there will have been little to no improvement in fitness at all.

4.7.5 Outer Problem Experimental Setup

In order to improve the interestingness of problems, a select number of parameters known to affect the physics of the simulation and values of the fitness for the solutions to the problem were chosen for consideration, shown in Table 4.4 and discussed in the next subsection.

In initial investigation, problems were evolved with only a single dimension of change. All other parameters were set to their default values. Each of the individual parameters, as listed previously, became the individual focus of specific evolutionary runs. This was carried out to investigate which of these parameters had the most dramatic impact on the ending level of interestingness for the problem.

These dimensions of change were examined in two ways. The first of these methods involved evolving individual parameters of the problem environment, with all other parameters being kept to their defaults. The second of these involved modifying every parameter of the environment, thus making the general task of optimisation a lot harder but theoretically allowing for a problem with a higher measure of interestingness.

In both cases, a two-layered evolutionary approach was taken. For each given problem candidate, a chromosome listing either the value for the individual parameter or all values for each of the variable parameters was used to set up a problem environment. Within this environment, an evolutionary run attempting to find a successful set of macro-actions within the problem was carried out, with the standard deviation of the initial starting fitness and the best ending fitness at the last generation being the outputs of this process used to analyse the fitness of the problem candidate, using the interestingness measure previously defined.

4.7.6 Experimental Parameters

The parameters used for these experiments are given in Tables 4.3 and 4.4.

Table 4.3 refers to values that were kept constant throughout all experiments and not considered as dimensions of change to vary. These values were excluded due to a consideration that varying them would either not affect the experiment or affect it in unpredictable ways. In some cases, this was a measure of safety. Allowing evolution to control variables such as the length of simulations and the maximum function evaluations would upset some of the fundamental expectations of problem candidates being tested, and keeping these things constant would allow a fairer direct comparison between different problem candidates.

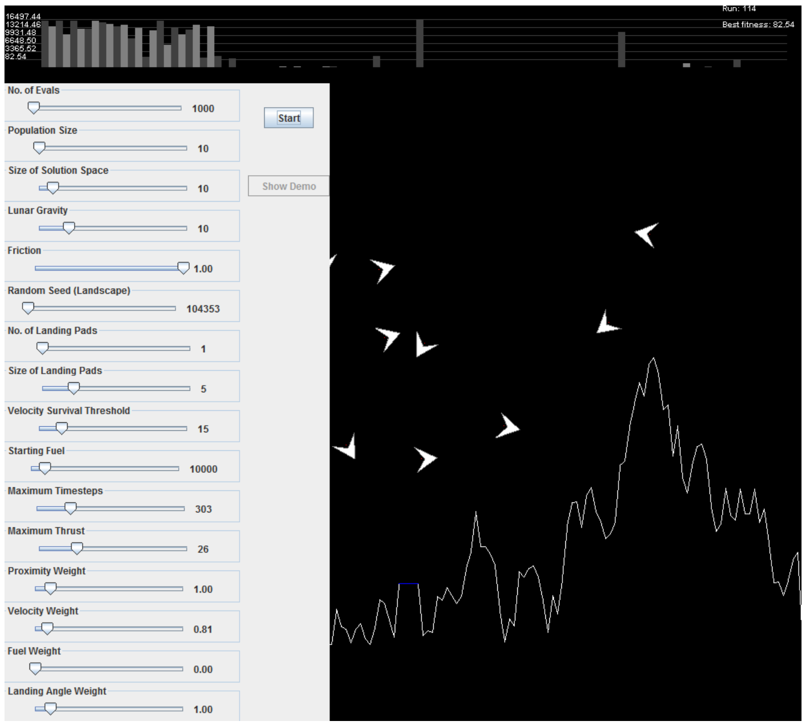
Table 4.4 lists the parameters that were permitted to vary through evolution. The values given are the defaults that are used for each parameter if the particular parameter was not being considered for evolution, as well as the lower and upper bounds of the values. These defaults and bounds were chosen mostly arbitrarily, based on loose observations of the spaceship physics in earlier experiments and the need to have some form of starting value to improve upon.

Table 4.3: Constant Values

Constant	Value
Number of Landscape Vertices	100
Number of Macro-Actions	4
Maximum Timesteps Per Run	400
Maximum Function Evals	10000
Population Size	50

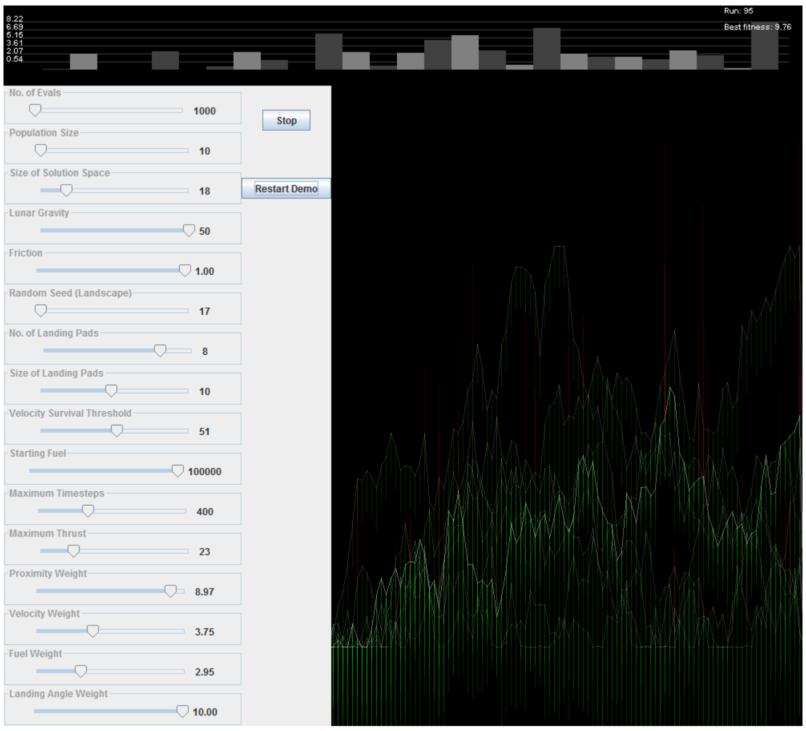
Table 4.4: Variable Parameters

Parameters	Default	Lower Bound	Upper Bound
Initial Solution Parameter σ	10	1	100
Thrust Limit	20	1	100
Gravity	20	0	50
Velocity Loss Factor	1.0	0.0	1.0
Landscape Seed	104353	0	2147483647
Number of Landing Pads	1	1	10
Landing Pad Width (Vertices)	5	1	20
Survivable Velocity Threshold	15	0	100
Initial Fuel	10000	1000	100000



No. of Evals	1000
Population Size	10
Size of Solution Space	10
Lunar Gravity	10
Friction	1.00
Random Seed (Landscape)	104353
No. of Landing Pads	1
Size of Landing Pads	5
Velocity Survival Threshold	15
Starting Fuel	10000
Maximum Timesteps	303
Maximum Thrust	26
Proximity Weight	1.00
Velocity Weight	0.81
Fuel Weight	0.00
Landing Angle Weight	1.00

Figure 4.10: A visualisation of multiple solutions within the context of one problem environment. The parameters of the problem environment are visible and can be adjusted using the sliders to the left. A fitness graph of the evolutionary progress of the solution population is at the top.



No. of Evals	1000
Population Size	10
Size of Solution Space	18
Lunar Gravity	50
Friction	1.00
Random Seed (Landscape)	17
No. of Landing Pads	8
Size of Landing Pads	10
Velocity Survival Threshold	51
Starting Fuel	100000
Maximum Timesteps	400
Maximum Thrust	23
Proximity Weight	8.97
Velocity Weight	3.75
Fuel Weight	2.95
Landing Angle Weight	10.00

Figure 4.11: A visualisation of multiple problem environments overlaid upon each other, using the same software as seen in Figure 4.10. Green vertical lines indicate the strength of gravity for a landscape, while upwards red lines indicate the allowed maximum safe landing velocity for a landing pad.

4.8 Results

4.8.1 Evolving Individual Parameters

Table 4.5: Varying Individual Parameters Alone

Parameter	Best Value	Best f_p
Initial Solution Parameter σ	0.28	0.63
Thrust Limit	155	0.33
Gravity	3.13	0.50
Velocity Loss Factor	0.01	6
Landscape Seed	1	147
Number of Landing Pads	9	0.37
Landing Pad Width (Vertices)	5	0.37
Survivable Velocity Threshold	89	0.38
Initial Fuel	9837	0.37

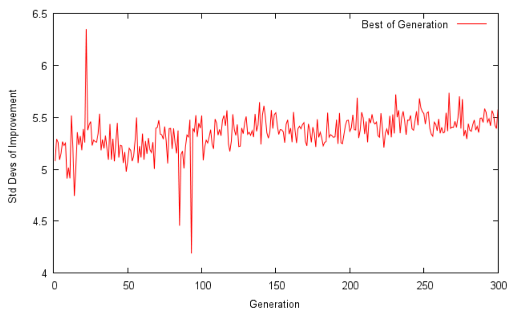


Figure 4.12: The progress of a problem population evolving to produce interesting problems, with the only parameter varying being the velocity loss factor. Barely any improvement is made at all as a result of varying only this one parameter.

The results of these experiments across multiple runs can be seen in Table 4.5, where the best value and related interestingness for each dimension across all runs are listed. Interestingly enough, it became apparent that the majority of individual parameters did not greatly impact the interestingness of the problem. One of the parameters which did have much of a noticeable effect, the velocity loss factor, did not do much to improve the interestingness of the problem over the course of an evolutionary run.

A reason for the velocity loss factor having much of a trend and impact on the interestingness of the problem was likely related to affecting the difficulty of the problem. With the physics as defined earlier, the inertia of a ship is one of the obstacles to be challenged and overcome. However, with a velocity loss factor between 0 and 1 exclusive, both the angular and linear inertia of the ship become less and less of an obstacle to overcome. Therefore, as the velocity loss factor approached zero, the problem became easier, and as the problem became easier, the capacity for improved progress of an evolutionary run became greater also.

Most interestingly, parameters that did little to affect the progression of problem evolution by themselves produced values within the same range of roughly 0.3 standard deviations of improvement. Even parameters that would seem to have an impact on the progress of solutions individually did not always when they were the only variable to change.

Curiously, only adjusting the landscape seed for the random generation of terrain lead to a wildly random evolutionary course of progress with no clear trends. However, given the landscape seed only affects the terrain generation, this was more useful as a measure of the highest albeit entirely chance-based improvement that could be achieved with the default problem parameters. Comparing the result of 147 standard deviations to the result of 179124 standard deviations of improvement from a later example of a problem candidate, as depicted in Table 4.6, indicates that there could likely be considerable improvement to be made on these default parameters. This improvement existing at all, however, suggests that the physical landscape shape can completely change the progress that could be made in this type of problem.

4.8.2 Evolving All Parameters

Table 4.6: Example Evolved Parameter Set

Parameter	Value
Initial Solution Parameter σ	68.26
Thrust Limit	0.00007
Gravity	9.92
Velocity Loss Factor	0.34
Landscape Seed	0.00
Number of Landing Pads	1
Landing Pad Width (Vertices)	4
Survivable Velocity Threshold	1.08
Initial Fuel	55654
Proximity Weight	1.82
Velocity Weight	7.31
Fuel Weight	5.07
Angle Weight	5.14
Interestingness	179124 σ

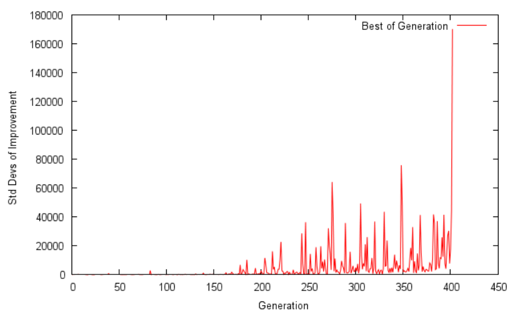


Figure 4.13: The progress of a problem population evolving to produce interesting problems, with all parameters of the problem free to change. Improvement is erratic for multiple generations until a single highly effective set of parameters is found. These parameters are listed in Table 4.6.

Interestingly, as can be seen in Figure 4.13, the course of evolution for many problems tended to be very erratic for multiple generations, with improvements being made and then discarded, until eventually a theoretically ground-breaking result was found. This tended to coincide with early discovery of a problem candidate with a very low initial standard deviation for generating the initial set of solutions, or a very low thrust limit for solutions to use.

Table 4.6 shows the parameters for an example problem candidate involving the modification of all legal parameters. As mentioned previously, the improvement, or interestingness, was defined in terms of initial standard deviations of the best ending result from the initial population mean.

However, the excessively large value that lead to this outcome indicated a serious inadequacy in this definition of improvement. This sudden increase did not directly indicate whether the progress being made was significant or not, but it did indicate that a means of artificially inflating the value produced by the measure had been discovered. This suggested that perhaps the measure of progress was somewhat naive, if it could be manipulated in this manner.

Table 4.7 shows the average interestingness accomplished over several runs, and demonstrates just how volatile the interestingness measure could be. Figure 4.13 demonstrates the evolutionary progress of interestingness resulting in the candidate parameters detailed in Table 4.6, and showcases a reoccuring pattern for many of the interestingness experiments, where one critical generation appears to suddenly cause a large spike in the fitness landscape, whether for better or worse.

This suggests a highly chaotic fitness landscape where tiny changes could cause considerable generation to generation changes, and the fact that these spikes occurred at the end of a run is not a surprising one. Faced with an extreme improvement, the criteria used for CMA-ES explicitly tells the algorithm to cease, and faced with an extreme deterioration, the algorithm ceases in this case as well.

Table 4.7: Interestingness Across All Runs

Number of Runs	39
Mean of Runs	82660
Std. Dev. of Runs	988497
Worst Interestingness	-1657942
Best Interestingness	4116275

Many of the most effective solutions had managed to meet the simple interestingness measure's implied demands either by beginning with a low initial standard deviation for generation, or by enforcing a minimal thrust limit to lead to a low initial diversity of solutions. The latter can be seen as one of the example best problem parameter sets as given in Table 4.6. In effect, what appeared to happen is the initial random population, when tested, all measured very similarly. This implied a problem that could not be solved through random unguided generation of solutions, but for which improved solutions could be.

When combined together, the parameters that appeared to affect a problem's score the most were parameters such as the velocity loss factor, the size and number of landing pads, the amount of initial fuel and the survivable velocity threshold. These factors directly impacted on the difficulty of a problem. Such variances determined whether or not a problem was simpler to solve, but given that the inner problem was an error minimisation problem, there was a limit to how much improvement could be made when presented with an easier problem. As a result, simple or trivial problems were not rewarded as highly for problems that begin with incredibly difficult solutions that could be solved effectively through evolutionary means. The end result was a compromise, where the size and number of landing pads did not stray far from their original defaults, but where often more fuel was given than default. This suggested that the default arbitrary fuel amount was too low for practical problems within the class of problems. The velocity loss factor has already been discussed, but its impact on problems is still as apparent when it is not the only parameter being varied.

4.8.3 Simultaneous Evolution of Parameters and Weights

One of the more interesting consequences of evolving both the parameters and weights of the problem was the way the two reinforce each other to produce a better environment for solution progress. As can be seen in the example set of evolved problem parameters in Table 4.6, the fuel weighting was much higher than the default, and the initial amount of fuel given to the ships in the inner problem was significantly higher than the default value. As can also be seen, the weighting for landing as slowly as possible under a threshold had become the highest weighted sub-condition in this example problem. The thrust limit was also very small, as was the velocity loss factor, to reduce inertia as much as possible, and the survivable velocity threshold was very small to strongly promote ships that could land within this limit.

However, because this threshold was very small, it would mean a random solution had much less of a chance to solve the problem well, based on how biased the scoring was towards ships that landed with a lower ending velocity. This created a situation where the progress of evolution within this problem would be significantly higher with a guided approach than a purely random approach, which was an interesting situation to examine, as opposed to a problem where pure random chance had a good probability of solving the problem to begin with.

Another interesting example to look at concerns evolved problems that placed a higher emphasis on fuel weighting. This meant that quantitatively better problems in the adjacent space of possible problems required higher initial fuel, and the two parameters reinforce each other as the course of evolution continues. In cases where fuel weighting was close to zero, the fuel value does not undergo much directed change, and instead drifts randomly as other conditions become more highly weighted. This split of focus either fuel measures or other measures was common across generated problem candidates, and suggested that the fuel sub-condition was in competition with the other sub-conditions. This makes sense, as the other three sub-conditions all required the expenditure of fuel to reduce the speed, adjust the position and adjust the orientation of the ship to as close to an optimal velocity, position and orientation as possible.

Despite this, there existed examples where both the velocity and fuel sub-conditions both have an emphasis, as in Table 4.6. Of the sub-conditions most affected, the velocity sub-condition was usually the most directly affected, as the fuel required to slow down and approach the landing pad far exceeded that for adjusting position and orientation.

The weighting of the proximity sub-condition and the size and availability of landing pads also appeared to impact each other. With a higher availability of landing pads, the proximity sub-condition became easier to fulfil and so lead to an improvement in the fitness of the population of solutions. With improvement in the progress of solutions, the problem candidate itself scored more highly, and as a result problems with more and larger landing pads dominate the population for the run in question.

4.9 Lunar Lander Conclusions

During this chapter, we have evolved candidates for problems within a class of related problems based upon landing a simplistic spaceship vehicle onto a jagged landscape much as in the game of *Lunar Lander*, while being mindful of one or many conditions of this landing.

These problem candidates have been evolved in accordance with a simple statistical measure of improvement. This measure is based upon multiples of the standard deviation of initial fitnesses. These initial fitnesses have been calculated from an initial random population of solutions within the environment of the problem candidate. This measure has been effective in leading to the generation of interesting problems within the space of problems relating to the *Lunar Lander* scenario as described at the beginning of this chapter.

The interaction and co-evolution between the problem and solution spaces examined have lead to interesting environments with unexpected changes to the physics of the problem. These environments have lead to interesting solutions that exploit these emergent characteristics of these environments. These solutions then affect the environment itself as they improve over the course of generations. Problems that were impossible to solve do not share this characteristic, and were discarded as less interesting than their counterparts that allow for effective solutions. Problem environments that encouraged good solutions have emerged from these requirements, which lead to a game space similar to but subtly distinct from the original *Lunar Lander*.

It was entirely possible that these results were heavily impacted by the particular choice of encoding for macro-actions, and that a different representation may have resulted in different results. However, there were only so many possible actions within this environment, and the definition of the measure of 'interestingness' used is based on the general success of solutions and not their actual content. It may be likely a different encoding for actions or a different controller would have produced similar results with only minor differences.

As the 'interestingness' measure ties into how well a computer controller can play the game, there are necessarily differences between how a human player might perceive the interestingness of a particular scenario and the value produced by the interestingness measure. It is perhaps safe to assume that there would be some connection, as an impossible game is not fun for a human player, and neither is a trivial game. However, the capacity for the computer controller to perform manoeuvres that a human player could not, through frame-perfect course changes, means it can easily be the case that a human player could struggle with a set of game parameters rated high in interestingness. Likewise, human ingenuity can be more than capable of solving some games that the software cannot.

Ultimately, the measure used for 'interestingness' is not without some issues. The example result of a run given produces far too high a value to indicate anything very useful, especially as it is the result of manipulating conditions such that the denominator of the function used, the standard deviation of a random population's fitness, is as low as possible. While there exist such flaws in this simple statistical measure of improvement, it does not appear to be without merit altogether. There exists plenty of scope for improvement upon this measure. Further examination in different contexts and case studies would also be a useful effort. What defines a problem as "interesting" to solve will always be a highly subjective matter, but it is considered that this attempt to define a measure of problem interestingness that is simple and fast to compute, as well as intuitive, is computationally beneficial in guiding the evolution of more interesting and solvable problems belonging to wider problem classes.

Interestingly, the 'best' landscapes produced according to the earlier given measure of fitness tended towards making the problem far easier, by introducing several landing pads. This had two immediately apparent effects. Firstly, it allowed for a greater chance of a safer landing by providing more landscape that would count as valid landing terrain. However, this also meant that a significant amount of the jagged landscape would be flattened closer to a flat uniform surface, due to the manner in how landing pads were added to the terrain during generation. Both of these factors would allow for a simpler environment for the problem.

In retrospect, one reason for this scenario being more complex than initially thought may have been the innate driving force of the simulation towards a failure state. If the problem is examined as a simulation of infinite length, then it can be safely assumed that any movements upwards serve to prolong the simulation and will never lead to a terminal state. This happened frequently in earlier experiments due to the harsh penalty of a failed landing appearing a far worse outcome than to never land in the first place. The fitness measure was adjusted to reward landing at all, with higher rewards for landings that were not also crashes, which prevented this degenerative and undesired behaviour.

In situations where fuel is a limited parameter of the experiment, eventually a vehicle will exhaust its fuel supply and the gravity force of the simulation will cause the vehicle to travel towards the landscape. As the amount of landscape where a landing is considered a crash is far greater than the areas where success is possible, this also means there are many likely paths of movement that result in a failure state.

Of the remaining possible movement paths that terminate on the landing pad of the landscape, only the ones that ultimately terminate beneath a certain speed of movement are considered a success. As there is a constant downwards acceleration, without any input at all from a controller it is almost certain that in most scenarios a vehicle will arrive at the landscape with too high a velocity to be considered a successful landing. A successful landing requires both downward movement that must also be well counterbalanced with an upward speed, but in order to produce this upward velocity there is necessarily less control over the lateral movement of the vehicle.

Having examined this *Lunar Lander* scenario, the next course of action was to investigate the interactions between spaceships that had been generated in a step closer to the original intent of generating vehicles for use in a game as potential antagonists or allies for the player. Until this point, vehicles had been evaluated individually, with other vehicles existing completely independently to any others being trialled, evaluated or demonstrated. The next scenario to investigate would be a complete change to this paradigm, and the initial stages of examining the competitive co-evolution of vehicle designs in a shared simulation space.

A video demonstrating some of the work discussed in this chapter is available online. [1]

[1] https://www.youtube.com/watch?v=7oK4RTYJc8A

Chapter 5

Vehicle Competition, Co-evolution, and Controller Adjustments

5.1 Problem Summary

5.1.1 Premise

The general premise of the experiments thus far have been an extension to the very first experiments relating to procedural design of vehicles within the context of a 2D physics-based game. A key common element to all of the experiments investigated thus far is that all vehicles within a population operate in total isolation to one another, having no effect on the effectiveness of other members during simulation.

While the last notable experiments relating to actual vehicle composition focused on simple movement, a new scenario was devised in order to focus on more complex and interactive actions between multiple vehicles.

This scenario consisted of placing two teams of generated vehicles in opposition to one another in the same space. With evolved movement capabilities and the ability to now contain and fire weapons at other vehicles, the expected results were to find designs exploiting weaknesses within the other team's designs. It was expected that some form of evolutionary arms race would emerge as a result of these changes.

5.1.2 Properties of the Combat Scenario

Vehicles were controlled by simple algorithmic controllers, based upon aligning their velocity heading to some desired direction affected by weighted conditions and inputs. The intent of the experiment was such that an effective design and set of controller weights would reinforce each other, and thus be the best end result of an evolutionary algorithm trial. New additions in these experiments consisted of adding more parameters to the vehicle chromosome that would allow for the intended combat scenario, while also allowing for more interesting behaviours to emerge relating to self-preservation and aggression.

Every vehicle was given three finite quantities directly impacting what the vehicle could do within the simulation. The first of these was a health, or hull, value, that determined how much damage the vehicle could take from enemy weapon fire before being destroyed.

The second was a fuel amount, similar to the use of fuel in the *Lunar Lander* scenario. Use of the vehicle's thrusters would deplete this fuel amount, and upon exhaustion the vehicle would no longer be able to use its thrusters to move.

The third and final amount was the amount of projectiles left available to the vehicle, as an integer amount. Use of weapons would deplete this value, and exhaustion would mean loss of the capacity to use weapons.

5.2 Combat Scenario

5.2.1 Vehicle Composition Changes

Vehicles were composed of a point mass connected to a series of individual point components, which in turn defined the vertices of a polygon that defined the shape of the vehicle, as in previous experiments and chapters.

In addition to the thruster components from previous experiments, turret components were introduced. These components remained fixed at a relative position and rotation to the centre point of the vehicle, in the same manner as thruster components, and could be in one of two binary states; active or inactive. While active, the turret would repeatedly fire projectiles in the direction it was facing, limited by a rate of fire which was made constant for all turret components across all vehicles. For each bullet fired, a turret would consume a unit of ammunition.

When a turret was predicted to hit an enemy target, it would be activated and then deactivated once predicted to hit nothing. However, this was based on a simple line trace, with no form of leading, resulting in vehicles that were much better at evading than aiming.

While not a direct change to the composition of a vehicle, it should be noted that a significant change to the collision detection for vehicles was made at this time. This change was to use the polygonal hull of the vehicle, produced by treating each vehicle component as a vertex, as the means of collision detection. This meant that the detection of collisions between vehicles and other physical entities within the simulation, such as bullets and pickups, were more accurately registered. This meant that the shape of a vehicle could determine whether a vehicle was more or less likely to evade threats, making the exact position of components even more important.

5.2.2 Vehicle Components

The components that a vehicle could be made of were thrusters or turrets. Each of these components had in common the properties of mass and moment of inertia. When a vehicle was constructed, it was constructed such that each component was added sequentially. If a component was too close to where an existing component would be, it was skipped and not added. Adding a component increased the mass of the vehicle by the mass of the component, as well as the mass of an invisible "strut" from the centre of the vehicle to the component, which itself had a mass dependent on its length. As such, components being placed further away from the geometric origin of the vehicle had a greater impact on the vehicle's total mass. The moment of inertia of the vehicle was similarly affected by component placement, including the strut, but not as significantly.

After all of the components were added, they were translated such that the geometrical centre and centre of mass for the vehicle would be the same point, to make future physics calculations simpler.

Thruster

Thrusters were, as before, components that provide physical impulses of force to their parent vehicles, based upon their local position and rotation in relation to the vehicle's centre of mass. They consumed one unit of fuel from its parent vehicle's supply for each unit of force exerted, while activated, and would not operate if the parent vehicle's fuel supply had been totally depleted. More thrusters active at a time would mean more fuel consumption.

Turret

Turrets were components that fired bullets with a trajectory dependant on the position and rotation of the turret. Bullets were consumed every time a turret fired, and every turret component had an inherent "cool-down" period where no bullet could be fired immediately following firing a bullet. A turret could only fire a bullet when its cool-down period had elapsed. Toggling the turret to active would lead to it firing as often as it could while ammunition supplies lasted. If the parent vehicle had no more bullets, a turret would not fire, active or inactive.

5.2.3 Combat Simulation Operation

In order to evaluate the fitnesses of the two teams, a single combat simulation was run per generation, with every vehicle given an individual score based on their own individual performance within this simulation.

The simulated combat began with all members of the population being placed based on their chromosome values for their initial position and rotation. The simulation consisted of vehicles moving around and firing at other vehicles based on the controller output, within a finite bounded rectangular environment. Vehicles that attempted to leave the confines of the environment instead bounced off of the edge.

A vehicle was rewarded for remaining hull, or health, points, very lightly penalised on ammunition expended to encourage efficiency, and flatly penalised for being destroyed. With the result of one combat simulation complete, four more simulations were run with each vehicle receiving the mean of their performance scores as their fitness score.

When vehicles were being artificially punished for being too large, their fitnesses were reduced if the components of the vehicle were greater than 50 pixels away from the origin of the vehicle, with the reduction becoming more severe the further over this limit the components were. Vehicles that were too large to fit into the game world were heavily penalised further.

Projectiles

Projectiles were fired by turrets and have a finite time of existence within the game world. Collision with the edge of the world destroyed them, depleting an individual time-to-live counter destroyed them and collision with a vehicle that was not their vehicle of origin destroyed them, as well as inflicting damage upon the other vehicle. Collision detection between vehicles and bullets was based upon testing the point in space the bullet occupied against the polygon of the vehicle. Before this test, a bounding circle collision test was used to determine whether or not more finer grained detection was required.

Pickups

Placed within the game space, at identical locations for each combat simulation determined by a starting random seed, were pickups.

These circular pickups collide based as a point intersecting the polygon of vehicles that overlap with them, same as with bullets. Four types existed in total, with three existing for each of the finite resources possessed by a vehicle. Fuel pickups granted the vehicle more fuel, ammo pickups granted the vehicle more ammunition, and hull pickups granted the vehicle more hull integrity. No pickups could increase the amount of these quantities beyond the initial maximum universal to every vehicle.

A fourth type of pickup, *mines*, would instead decrease hull integrity, and was intended to provide a reason for vehicles to be agile but not to simply constantly move blindly.

Mines act as static bullets that can damage any ship that comes into contact with them, removing hull value from the vehicle. There is otherwise nothing differentiating them from other pickups. They can be considered as negative hull pickups, but are not considered as valuable for the controller to move the vehicle towards, and are considered a hazard.

Upon collision with a pickup, the appropriate addition to vehicle resources is made (or, in the case of mines, subtraction) and the pickup is removed from the simulation. pickups are not replaced during a simulation, meaning that finite resources can only be temporarily replenished.

5.3 Changes to Representation

5.3.1 Structure of Chromosome

The vehicle chromosome used for these experiments was a 39-dimension vector of double-precision numbers, each interpreted and converted to appropriate values for vehicle construction outside of the main CMA-ES loop. As individual dimensions related to different elements that worked more effectively in different ranges, these doubles were scaled up and down as appropriate for the purpose of the represented value.

Game World Placement

The first three values in the chromosome array indicated the initial position and rotation of the vehicle. The first two values indicated the X and Y co-ordinates of the vehicle in the game space. The co-ordinate provided is relative to the centre of the game world, but still using the same axes of game space as in previous experiments, where X increases in a rightwards and Y increases in a downwards direction. The third value indicated the orientation of the vehicle as an angle in radians. These values were not scaled.

Controller Weights

The next four values, chromosome values 3 to 6, related to controller weightings. Each of these values were scaled by a factor of 0.01, and were used as weightings for the controller's sub-objectives, in a manner similar to the later experiments examined in the previous chapter. These weightings are discussed in more detail in section 5.4.

Component Specifications

After this were five sets of four values, a set of values for each of the five vehicle components used in these experiments. The first of these values was a new addition to the component value sets, and was used to determine whether a component was to be a turret or a thruster. A simple check for the sign of the value was used to determine if a component should be added as a turret or a thruster. The remaining three values determined an attachment position offset and rotation relative to the vehicle's own position and rotation, as in previous experiments. These physical properties are scaled by a factor of 0.1 from the real values held in the chromosome.

5.3.2 CMA-ES Changes

CMA-ES was used as the main evolutionary algorithm for improving vehicles, with an initial mean X of 0 and an initial standard deviation σ of 100. The standard deviation affects spatial dimensions the most directly, as the values used for spatial dimensions were not scaled.

5.4 Controller Operation

The algorithmic controller for each vehicle was rewritten so that it no longer attempted to reach a specific target point in space. Instead of attempting to reach a target point, the controller attempted to ultimately steer a vehicle to align its velocity with a non-normalised steering vector.

This desired steering vector was calculated based on vectors pointing to various different points of interest relative to the vehicle in question. These vectors would be normalised themselves and scaled based on the earlier defined "controller weightings" in the chromosome mentioned earlier, allowing for some of these different directional vectors to have a much stronger bias than others, and allowing the evolutionary process to hook into elements of the behaviour of a vehicle.

Once all direction vectors were calculated, normalised and weighted, they were summed to a final direction vector that is normalised, and the most appropriate thrust action that would move the vehicle in this direction is the thrust action taken.

These steering directions were based upon a variety points of interest, but none of the individual points themselves were ever directly selected as a point to approach. The combination of approaching all of these points, weighted by the appropriate chromosome values, would determine the final direction to steer towards.

5.4.1 Calculating Directions

Attack Target Direction

Targets to attack were acquired based on the closest vehicle legal to target. A vehicle was legal to target if it had a hull integrity above zero and was on a team different to that of the attacking vehicle. Additionally, vehicles were also only targeted if it seemed possible that a shot would collide with the target vehicle.

This hit prediction was based on a finite length hit-scan from all turrets and assumed the target did not move. If no vehicles were likely to be hit, the target defaulted to the closest legal target. The direction to the target was calculated and could be used alongside a chromosome weighting to determine whether a vehicle would approach or flee from its target and with what priority.

Mean Direction To All Enemies

Additionally, a direction was calculated to allow for the approach towards or retreat away from the mean position of all enemies. This was more meaningful for the team scenario than for the free-for-all scenario.

Nearest Ally Direction

The direction to the nearest ally, a vehicle on the same team, was also calculated and could be weighted via chromosome value to encourage grouping or spreading out. It did not seem that in practice this direction was ever put to much practical use. During the later free-for-all scenarios, this direction vector was not taken into account.

Bullet Avoidance Direction

Bullet avoidance took into consideration the direction of all projectiles that seemed likely to collide with the vehicle, as well as static mines, individually weighting them inversely based on distance. The closer a bullet or mine was to a vehicle, the more important it became. All of these weighted directions were normalised into a single bullet avoidance direction vector.

The final avoidance vector itself was weighted by a vehicle's chromosome. Higher positive numbers would lead to the vehicle placing more priority on dodging bullets, but there also existed the capacity for a vehicle that would move towards bullets. Few vehicle chromosomes persisted beyond the initial 20 generations with this pathological behaviour.

Pickup Collection Direction

Pickup collection worked almost identically to bullet avoidance but in reverse, with closer pickups having higher priority over distant pickups. As it was also weighted by a weight that could be positive or negative, vehicle controllers could greatly prioritise collection of pickups or even avoid them altogether.

5.4.2 Weighting Directions

For each of the previous directions given, a dimension of the chromosome in the controller weightings block was allocated to it. Each direction vector would be normalised and then scaled by the value given in the chromosome, with potentially no bound on the value of the scaling coefficient taken from the chromosome. With negative weighting, some of these directions could potentially become avoidance directions, or, in the case of the bullet avoidance direction, even become a tendency to move actively into danger. With low magnitude weightings approaching zero, these directions had the potential to be ignored entirely, all through evolutionary iteration.

5.4.3 Executing Actions

On initialisation of the controller, the thrust vectors resulting from every combination of active and inactive thrusters were calculated and stored. Movement was based upon two stages.

The first stage involved turning on all thrusters that would move the vehicle in the desired direction of movement. After checking the thrust vector associated with every move action, the move action producing the thrust vector most matching the desired thrust is used. All thrusters associated with the move action were activated for the next simulation step.

The second stage was only used if there was an enemy target within range. If there was, the second stage involved activating any additional thrusters that could help with spinning a turret in range of a target. All turrets were checked for their direction of fire and this direction was compared to the direction vector from the vehicle to the target. The smallest angular difference was used to pick the turret closest to the correct alignment. All move actions were then checked again for actions that would individually produce the best torque for the next step as desired for the angular difference. The thrusters involved in the move action were then activated in addition to the thrusters selected for the first stage.

Once movement was handled, all turrets were checked to see if firing would collide with a stationary hostile vehicle, using a very naive assumption of the target vehicle staying totally still. If they would collide, the turret was fired.

5.5 Combat Scenario Experiment

5.5.1 Team Scenario

For each experimental run of the team based combat scenario, the simulation was initialised with two populations of vehicle designs, unlike the one population used in earlier experiments. These vehicles were placed randomly either in the left or right side of the playing field, depending on which population they were a member of, and assigned a team based on the population they were a part of. Rather than being evaluated individually, an entire simulation was run with every individual ship existing in the same environment, with fitness scores calculated during the simulation for each ship and after the end of the simulation.

As with previous experiments, the experiment was given a finite number of timesteps to finish, but was also terminated upon the destruction of all vehicles of one team. Destruction of a vehicle was defined as the complete depletion of a vehicle's hull value, depleted by bullets fired by vehicles of the other team. Destruction of a vehicle would lead to an immediate removal of the vehicle from the simulation environment.

The pickups as described were not included in this experiment, but they were included in subsequent experiments.

The vehicles were given an initial hull and fuel value, although in practice the fuel value was sufficiently high that no effects were observed relating to fuel. Fuel worked as in previous experiments, with some fuel depleted for active thrusters per timestep the thrusters were active. The values for constants of the team based experiment are given in Table 5.1.

Table 5.1: Constants for Team Based Scenario

Initial Hull	200
Initial Fuel	100000
Bullet Damage	10
Max. Free Component Distance, d_{free}	50
Component Distance Penalty Weighting, w_{dp}	100
Number of Timesteps	300
Ship Hit Score, s_h	10
Ship Destroyed Score, s_d	100
Hull Score Weighting, w_h	200
Fired Bullets Penalty Weighting, w_b	1
Destruction Penalty, p_d	1000
Team Score Bonus Weighting, w_t	0.1

Fitnesses for ships were calculated in two stages. The first stage was based upon the achievements of a ship during the simulation, and a secondary stage added after the simulation.

Fitness for ships inside the simulation was based upon the number of bullets fired that successfully hit an enemy ship , n_h, and number of enemy ships destroyed by the ship, n_d. The score values for these were s_h and s_d respectively. The resulting ship fitness component calculated from actions taken over the simulation, or f_s, was therefore calculated as

$$f_s = s_h n_h + s_d n_d \tag{5.1}$$

At the end of the simulation, the score for an entire team, f_t, was calculated as the sum of all simulation fitnesses for the team. For n_{team} ships on a team, and where the term f_s^i represents the fitness f_s for the ith ship, the team fitness was calculated as

$$f_t = \sum_{i=1}^{n_{team}} f_s^i \tag{5.2}$$

Ships were given a static penalty for designs with components with a distance magnitude from the ship centre, $\boldsymbol{s_c}$, greater than d_{free} (as defined in Table 5.1) to discourage the evolution of larger ships without expressly forbidding it. This design penalty, p_{design} was calculated as

$$p_{design} = w_{dp} \max(|\boldsymbol{s_c}| - d_{free}) \tag{5.3}$$

With the ship's individual simulation score and design penalty, along with the team score, the final fitness, f, for a ship was calculated using these previous values with further considerations.

A ship was rewarded for how much remaining ship hull, h_r, it had at the end of the simulation, with w_h used as a weighting for the hull remaining. It was also penalised for the number of bullets fired, n_b, weighted by w_b, to discourage incessant bullet firing. Finally, the ship's fitness was given a bonus based on the team score, weighted by w_t, and a flat penalty p_d was applied for the ship exiting the simulation in a destroyed state. With all of this in consideration, the ship fitness f was

$$f = \begin{cases} f_s + w_h h_r + w_t f_t - w_b n_b - p_{design} & \text{if ship undestroyed} \\ f_s + w_h h_r + w_t f_t - w_b n_b - p_{design} - p_d & \text{otherwise} \end{cases} \tag{5.4}$$

This fitness score was then negated for use by CMA-ES as a minimisation problem. As discussed in results later in his chapter, this complex fitness function was a very difficult one to minimise.

5.5.2 Teamless Scenario

A comparatively simpler scenario was devised without consideration for teams, and where the team based parameters of the controller were still available but only applied to the individual ships themselves. Angular momentum was removed in this experiment to make the problem of ship design and control marginally easier, removing the need to apply a counter torque force to remain facing the desired direction of travel.

The simulation was populated with a single population, as with previous experiments, with ships placed based on positional parameters in their chromosome instead of at random. If ships were too large to fit on the playing field, they were immediately removed from the simulation and given the worst possible fitness score. The constants for these experiments are shown in Table 5.2, with values different from the team based scenario shown in bold.

Table 5.2: Constants for Teamless Combat Scenario

Initial Hull	200
Initial Fuel	100000
Bullet Damage	**5**
Max. Free Component Distance, d_{free}	**20**
Component Distance Penalty Weighting, w_{dp}	100
Number of Timesteps	300
Ship Hit Score, s_h	**20**
Ship Destroyed Score, s_d	100
Hull Score Weighting, w_h	**100**
Fired Bullets Penalty Weighting, w_b	1
Destruction Penalty, p_d	**500**
Bounce Penalty, p_b	**5**
Travel Bonus Weighting, w_t	**0.1**
Pickup Collect Score, s_p	**50**

Fitness was calculated as with the previous experiment, with the following modifications. The team score no longer made sense and was removed from the fitness calculation. However, the simulation fitness was adjusted also to include a penalty for bouncing off of the edges of the playing field, p_b, and rewards for moving to encourage movement within the playing field, equal to the velocity per timestep weighted by w_t. Additionally, with the addition of pickups to this experiment, a bonus for pickup collection, s_p was also added to the fitness metric. Collision with mines provided no pickup benefit.

Given the number of bounces in a simulation as n_b, the number of successful pickup collections as n_p, and the sum of all magnitudes for every timestep during simulation as $|\boldsymbol{v}|$, the new simulation fitness was calculated as

$$f_s = s_h n_h + s_d n_d + s_p n_p - p_b n_b \tag{5.5}$$

with the final fitness f calculated for a ship as

$$f = \begin{cases} f_s + w_h h_r - w_b n_b - p_{design} & \text{if ship undestroyed} \\ f_s + w_h h_r - w_b n_b - p_{design} - p_d & \text{otherwise} \end{cases} \tag{5.6}$$

with this final fitness negated for CMA-ES, as before.

5.6 Combat Scenario Results

While the results of both the team-based and non-team-based variant of the scenario were not as effective as hoped, the reasons for the lack of effective results were clear. The complexity of the scenario coupled with the use of a controller for all vehicles that could no longer adequately cope with the presented scenario both indicated that a simpler scenario and a more effective controller were changes required to see behaviours more in line with the initial expectations.

The steering-based controller, designed to be simple and quick to evaluate, was not able to make effective use of the vehicle designs presented in any suitable fashion, thus negating any benefit of observing the relationship between the controller brain and vehicle body.

5.6.1 Problems

Unfortunately, none of the expected behaviours seemed to occur. Initially, the two teams were treated as separate populations with evolutionary selection and propagation executed in isolation, and so did not actually have much capacity to respond to the other team designs, as there was no useful transfer of information between population.

The scenario was redesigned to remove the team-based element from it. Instead of having teams of co-operating and competing agents, the scenario was redesigned so that every individual agent within the simulation competed with one another for the resources within the simulation as well directly attacking one another.

This also did not work, as all individuals were being evaluated based on their performance in the combat simulation. Save for pickup and mine placement, all obstacles presented to the individuals were themselves other individuals being evaluated and improved. There was no clear way to improve objectively as given in the fitness measure. All fitness scores were relative, and did not result in any improvement more direct than between generation to generation. There was no objective reward function considered that would indicate whether a given vehicle performed effectively on an objective, absolute scale.

Due to the fact all fitnesses were relative, it meant that objectively poor vehicle designs, such as very large vehicles that could barely move and were very easy targets to hit, would become viable if other vehicles had similar designs, and could easily occur due to an interesting consequence of the representation of the vehicle chromosomes interacting with CMA-ES. As improvement did not happen at the expected rate, the usual reaction from the CMA-ES algorithm was to increase the standard deviation of the values of the vehicle chromosomes. As the placement of components was specified as pairs of real numbers relative to a central offset point for a vehicle, this increased standard deviation for the values had the direct consequence of components being placed further and further apart. In turn, this caused the vehicles to expand in size, which, leading to maladaptive designs, would lead to vehicles that performed even worse, thus requiring an increase of the search space via increasing standard deviation again.

These large vehicles could be seen to perform less effectively than designs from earlier generations, from simple inspection of the playback of simulations as a human observer. However, they were still judged relative to all other vehicles, which due to identical conditions would also perform poorly. This was completely counter to the initial assumption that more dominant vehicle designs would dominate a population and shape the landscape as a result. When a population began to perform badly, it seemed to be a reoccurring phenomenon that both populations of vehicle designs would begin to gravitate towards designs that were only effective because of identical degenerate evolution occurring within the opposing population.

There was one positive to the increasing vehicle size, however. Turret components would begin in a state in the simulation not just in range of but intersecting a target vehicle. With no capacity to evade such a situation, targets would be rapidly removed from the simulation, leading to an arms race of larger and larger vehicles exploiting this strategy and thus becoming more and more vulnerable to it.

While this could be seen as a flaw with the representation of vehicles, it was also clear that before this point of failure, vehicles with smaller designs were not behaving in an interesting or sophisticated way, indicating a problem with the algorithmic controller approach used up to this point. It was also indicative of the flaws of the fitness measure not providing a consistent and objective means of comparing fitnesses on anything more than a generation to generation basis. The fitness measure also tended to be very discrete due to the way vehicles were awarded points during the vehicle combat simulation, leading to less than ideal comparisons where fitness scores were identical despite differences in design and behaviour.

Table 5.3: Team Based Scenario Scores

Number of Runs	10
Mean of Runs	14956
Std. Dev. of Runs	19071
Worst Fitness	-3000
Best Fitness	40826

Shown in Table 5.3 are the scores for the team-based scenario across a number of runs, shown negated and where a larger positive result indicates greater fitness. The results indicate that the vehicles were meeting the fitness function as specified, but in largely uninteresting albeit efficient ways, such as the larger vehicles mentioned previously. This was believed to be indicative of a fitness function that rewarded behaviours unintended for the purposes of designing agile, novel ships. The high standard deviation again indicated that this problem was perhaps not very well defined, or that the numerous interactions between dimensions of the chromosome parameters made this a difficult problem to solve reliably.

Table 5.4: Teamless Scenario Scores

Number of Runs	10
Mean of Runs	-2
Std. Dev. of Runs	6
Worst Fitness	-2×10^8
Best Fitness	799

Table 5.4 shows the statistics for a number of experiments based on the single population, teamless, free-for-all experiment. As with the previous table, here positive values indicate fitness. Despite the problems inherent with the team based scenario, the results here appeared to be even worse, perhaps as a direct result of the design penalty function being made much more punishing to designs that attempt to exceed the smaller "safe radius" explicit to the fitness function design of the experiment. The increasing burden of all of the conflicting parameters and variables used were the motivation to try a simpler scenario, and the design penalty function was one of the first things to be removed.

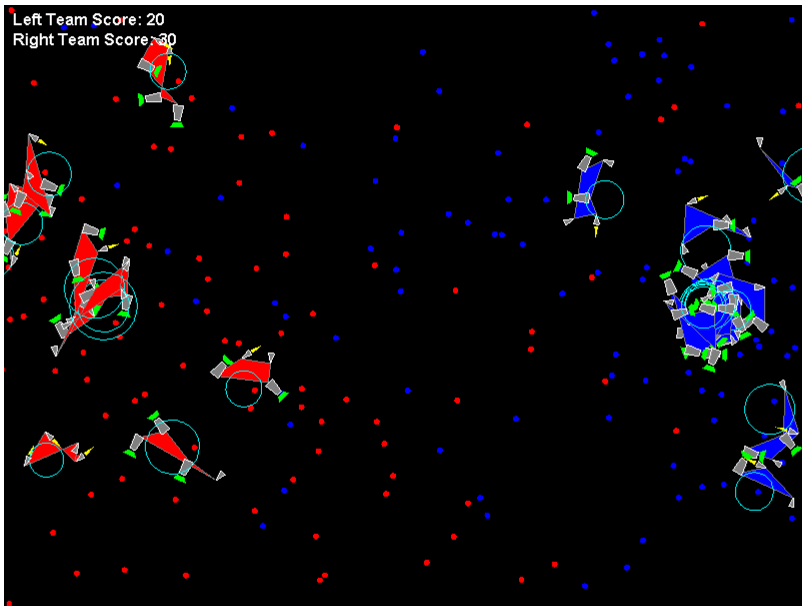


Figure 5.1: Example of team-based combat scenario, where two competing teams of vehicles attempt to evade and attack the other team. This is an image of an earlier version of the software, before adjustments were made to make the centre of mass and geometric centre equal, as can be seen by the visualisation of the bounding circle for the vehicles. The coloured smaller circles are projectiles.

5.7 Pickup Collection Scenario

Determining that the problem had perhaps become too complex at this stage, a simpler scenario was devised. The simulation was redesigned to only allow for single vehicles operating in

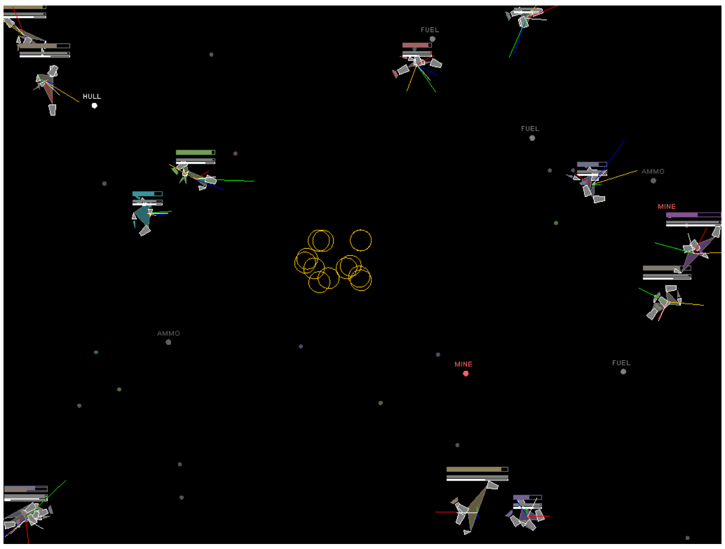


Figure 5.2: Example of a free for all combat scenario, where the previously described features apply. Visualised are the various steering factors used by the controller to decide the best approach vector to orient the ship towards.

isolation. The new goal was for a vehicle to collect all pickups placed within the playing field, while avoiding all mines. To further reduce complexity for the problem, rotational inertia as a physical property was removed. Rotation was still possible, but the need to counter-spin in order to handle rotational inertia was still no longer a factor, simplifying the movement physics of the simulation.

5.7.1 Experimental Setup

The experiments for the pickup collection scenario did not require a combined simulation of every vehicle at the same time, and as a result vehicles were simulated independently from one another in separate simulations, as with previous experiments.

As with the previous teamless experiment, at the beginning of a simulation, if a vehicle was too large to exist within the playing field, evaluation ended and the worst possible fitness value was returned. Vehicles were still penalised for bouncing on the screen, and the reward function was adjusted to reflect a scenario wherein pickups became much more important. The design penalty present in previous experiments was removed. The parameters used for this experiment are given in Table 5.5. Many of these values were chosen arbitrarily, with the benefit of pickups chosen to outweigh the negatives of mines in fitness score.

During a simulation, a ship could bounce on the edges of the playing field n_b times, collect n_p non-mine pickups and collect n_m mines. If all pickups were collected, the simulation would end with t remaining timesteps. The remaining hull of the ship, h, at the end of the simulation, given n_{ph} hull pickups collected, can be calculated as

Table 5.5: Constants for Pickup Collection Scenario

Initial Hull, $h_{initial}$	100
Number of Timesteps	300
Number of Pickups Total	10
Hull Score Weighting, w_h	100
Bounce Penalty, p_b	100
Pickup Collect Score, s_p	1000
Hull Pickup Restoration, h_p	30
Mine Pickup Damage, h_m	50
Mine Collect Penalty, s_p	500
Remaining Time Weighting, w_t	10

$$h = \max(\min(h_{initial} + n_{ph}h_p - n_m h_m, h_{initial}), 0) \tag{5.7}$$

The fitness for a ship at the end of a simulation, f, was calculated as

$$f = \begin{cases} w_h h + n_p s_p - n_m s_p - n_b p_b + w_t t & \text{all non-mine pickups collected} \\ w_h h + n_p s_p - n_m s_p - n_b p_b & \text{otherwise} \end{cases} \tag{5.8}$$

As with previous experiments, f was negated to rephrase the problem as a minimisation problem.

5.7.2 Results

The initial fitness measure used for this scenario was to give a fixed reward for collecting each pickup, and a time bonus based on remaining timesteps if all were to be collected. However, this turned out to be too discrete, much as the fitness measure used in the previous experiment, as it was evidently unlikely for any vehicle to collect all pickups within the given time budget.

Table 5.6: Pickup Collection Scenario Scores

Number of Runs	10
Mean of Runs	7070
Std. Dev. of Runs	1622
Worst Fitness	3400
Best Fitness	8800

The results of these pickup experiments are shown as a whole in Table 5.6, negated from the values negated for CMA-ES, and where positive values indicate a better result. While the results here were not as dire as the results were for the teamless combat scenario, it should be considered that the definition of the fitness function for the scenario suggested a score greater than 8000 for managing to collect all non-mine pickups, and while in some cases this was accomplished, on average most runs did not collect every pickup.

Increases to the time budget did not make the problem any simpler to solve, and visualisation of the paths taken by the steering-based controller in attempting to pilot a given vehicle design only further highlighted the flaws in the controller. It was clear that the previous steering-based controller was completely insufficient for the problem and that a different approach was required.

5.8 MCTS Controller

5.8.1 Motivation and Initial Challenges

The need for a better approach to collecting pickups was apparent. Oscillating behaviour was commonly observed where a vehicle would be close to a target item but overcompensate in trying to move towards it, only to end up targeting a different pickup in the game world in doing so, as seen in Figure 5.3. In attempting to collect this new pickup, it would frequently overshoot and then target the original pickup, leading to an endless oscillation.

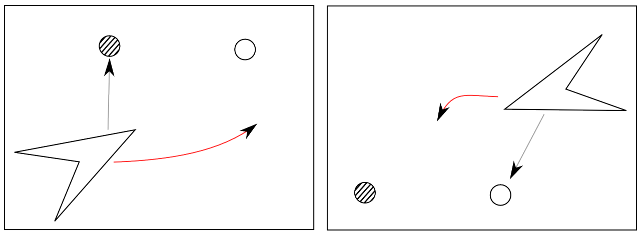


Figure 5.3: Oscillation of target selection from overcompensating movement. The red arrow gives an idea of vehicle velocity, while the grey arrow indicates the target the vehicle is attempting to reach.

The controller was also 1-ply and contained no inherent ability to plan, making it ill-suited for anything further than reaction to the state of the game world at the very moment in time.

When it became clear that the algorithmic controller was no longer suitable for a problem of this complexity, a Monte Carlo Tree Search based controller was considered instead. However, with an action space of 2^n actions per game time step, where n was the number of components in a vehicle, it became clear that realtime operation of MCTS for such an action space would be nigh impossible without the use of macro actions.

Macro actions were defined in the same way as actions were. Where an action was a combination of component on/off states, a macro action was the same, but to be repeated for 15 timesteps per macro action. Enforcing this repetition reduced the number of required decisions for MCTS to take during a simulation, meaning rollouts could be performed much faster and theoretically with a reduced search space.

Using macro actions made this approach possible, but the initial tests did not produce behaviour that surpassed or even matched the behaviours produced by the algorithmic controller. Where the algorithmic controller would collect two or three pickups within a set amount of timesteps, the MCTS controller would not collect a single pickup in most tests.

Limiting the action space by using a fixed vehicle, where the only permitted actions were to activate one of four thrusters (see Figure 5.4), also did not do much to improve the performance of standard MCTS. This may have been due to poor rewards given for terminal states, dependant only on the number of pickups collected as no rollout would ever successfully collect all pickups available.

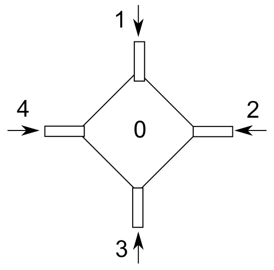


Figure 5.4: Simple five action vehicle, with an idle action (0), and four other actions corresponding to activating a specific thruster.

The experiments involved in testing this special case vehicle with an MCTS controller did not vary significantly from the previous pickup collection problems, save for the specialised vehicle design and use of very specific actions. All other elements of the problem remained the same, save for a minor modification to the fitness function for the purposes of evaluating terminal states.

This terminal state measure was adjusted to provide for more continuous information in the case where the vehicle had not managed to succeed in collecting a pickup.

This adjustment consisted of using an alternate metric to factor in cases where the vehicle had not succeeded in collecting in a pickup. In this case, the value of a given terminal state at the end of a simulation was, given $|\boldsymbol{s_p}|$ as the distance between the vehicle and the nearest pickup and $f_{original}$ as the previous fitness function, as follows:

$$f = f_{original} + \frac{1000}{\boldsymbol{s_p}} \tag{5.9}$$

This had the net effect of making the terminal state scoring function less discrete, but alone did not cause much of an improvement in the capability of the controller to collect pickups. However, there was still space for improvement.

5.8.2 Use of a Heuristic

A heuristic was tested, based upon an influence map populated of cells, each with an associated weighting, designed such that the influence upon these cells was generated by pickups within the game world. Originally, pickups would contribute positively to this map, and mines would contribute negatively, but this lead to bad interactions in the case of mines being near pickups. It was more effective to simply have mines have no impact on the influence map.

The value v for each cell in the influence map was more specifically calculated as follows, where d_i is the scalar distance between the cell and a pickup p_i for p_t total pickups, and D as the maximum possible distance in the bounded game world.

$$v = \sum_{i=1}^{p_t} 50((D - d_i)/D) \tag{5.10}$$

The way that states were evaluated by this heuristic was simply to report the value of the cell the vehicle happened to be in for that state.

The heuristic itself was initially used in MCTS by tracking the heuristic values for all states reached in a rollout and initially using a sum of the values in addition to the terminal state for the rollout, but this sum would tend to dominate the value given for the terminal state given the length of rollouts. The value for the rollout was changed to be calculated by taking the maximum value reached with the heuristic in addition to the terminal state score, which would mean the heuristic information could have an influence on the value of a rollout but without overwhelming the terminal state scoring.

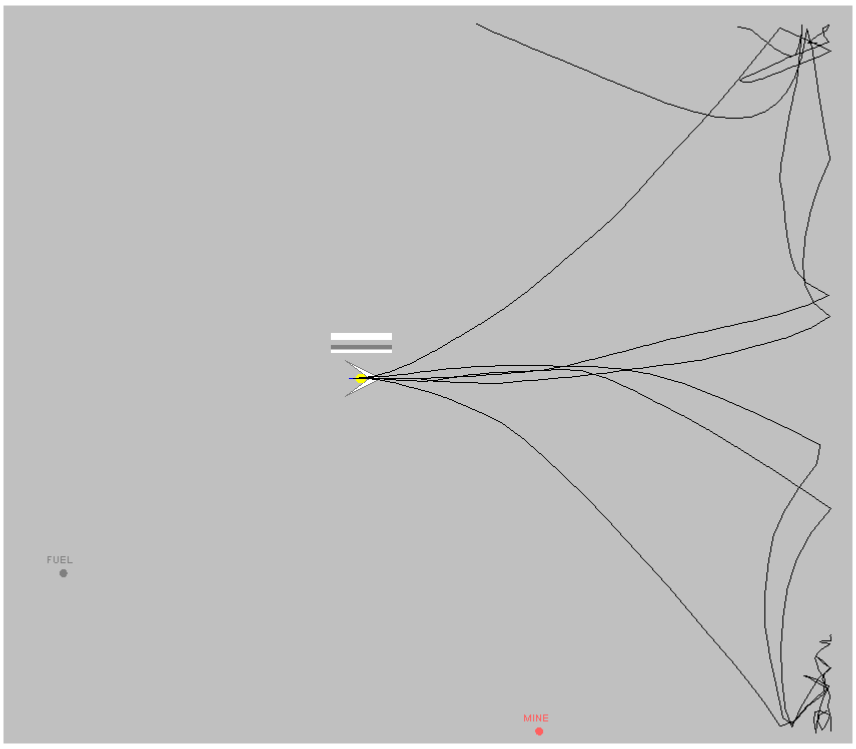


Figure 5.5: Very deep rollouts did not necessarily indicate whether an action was going to be successful, given the wildly divergent random paths taken. Shown in this image are all deep rollouts produced for a given starting position in a single example simulation.

As can be seen in Figure 5.5, the rollouts tended to involve a great deal of bouncing and very little direction, mostly exploring actions with no real exploitation of any actions that would serve a more effective purpose. This random behaviour indicated that the way the heuristic had been implemented was having little to no impact in its current state and from visualising the rollouts it became clear as to why this could be. With such high rollout depths and such a fine granularity to the macro action length, assigning the best heuristic value for a rollout was almost meaningless. The rollout could cover potentially any sort of path and would be assigned the heuristic value of the most useful cell it happened to cross on the random walk. It was also clear that the macro action needed more length to encourage a more coherent path without being so long as to remove the ability to control the vehicle properly.

Given these symptoms, one of the first things tried was to penalise bouncing. Originally, the total number of bounces made by the vehicle would be weighted against the vehicle in the terminal state of the game, but this did little to dissuade the vehicle in general from bouncing as it would result in a set penalty for the remainder of the game that could not be avoided or undone from that moment on, ceasing to become a meaningful statistic.

The heuristic was adjusted so that only the number of bounces within a single step were significant. In addition, to combat the random walk issues encountered earlier, the heuristic returned to a sum of the values of each cell of the influence map, but significantly scaled down.

The main and most effective changes were to reduce the rollout depth to a much smaller amount, thus meaning that rollouts reflected more accurately on the outcomes of actions, and to find a suitable length for macro actions. This length had to be a compromise between allowing the controller enough granularity of control while not overwhelming it.

With these changes to the way the heuristic impacted the rollout values and the limits on the rollout depths, rollout values became a much more useful signifier of the benefit of an action, and the vehicle began to collect the pickup more frequently in each simulation than before.

Other changes examined to view their impact included calculating the influence map based on logarithmic falloff as opposed to linear, by assigning the value to each cell as follows:

$$v = \sum_{i=1}^{n} 50(5 - \ln d_i) \tag{5.11}$$

However, this influence map did not seem to work as well as the linear falloff map. It may be the case that using the best value reached instead of a sum of values for the heuristic would have been more suitable for this case, but using a sum of values tended to result in very heavy penalties for straying away from the optimal path towards the pickup. Given the stochastic nature of MCTS, this was almost guaranteed to happen for at least one action, and veering into high penalty territory did not often allow for recovery.

It can be argued that, by putting in many specific conditions, subconditions and a variety of

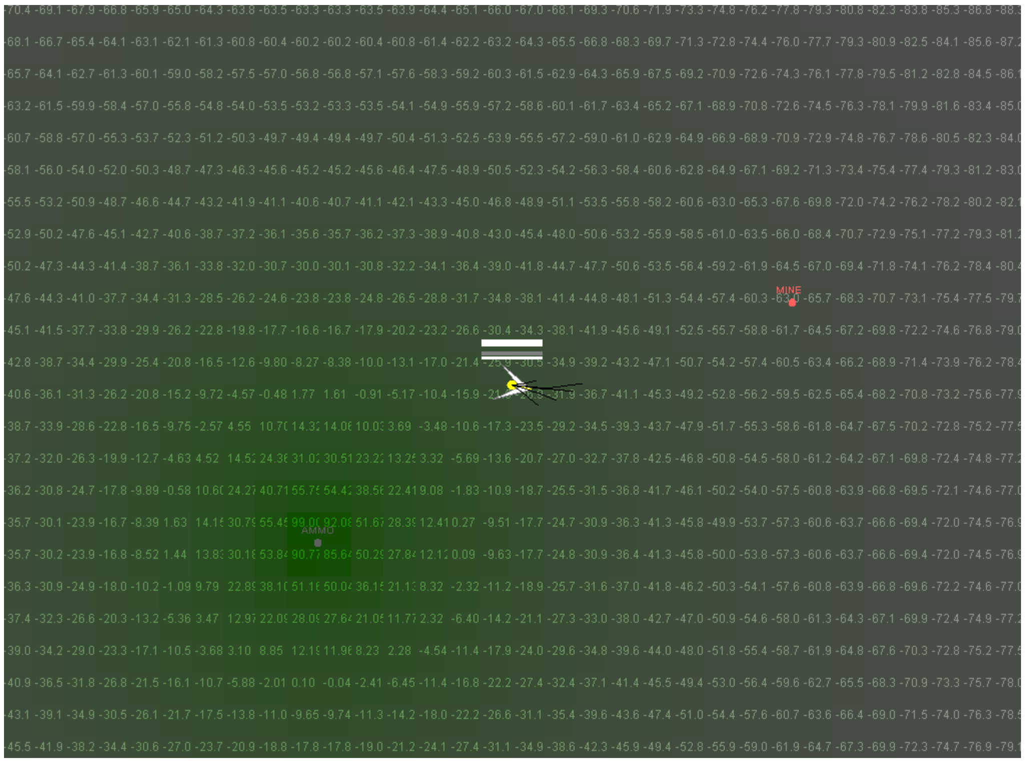


Figure 5.6: Logarithmic influence map, with heavy penalties for moving away from the pickup.

arbitrary terms and formulae into the fitness function to deal with very specific problems, the system no longer acted as an AI system. The constant readjustment of the fitness function to avoid specific pitfalls could be considered as increased injection of domain knowledge in a very direct sense, such as the design penalty for larger ships used in earlier versions of the combat scenario.

These fitness function adjustments also could be considered as a continual shifting of the goal posts of what was desired, as when earlier designs were developed that met the conditions of the fitness function were shown, the results were declared as disappointing.

However, it can also be argued that evolutionary problems always require a tailored fitness landscape in order to improve. A flat fitness landscape or one full of unexpected pitfalls does not lead to success in evolution or many forms of iterative search, and these various adjustments were made in order to promote better success in directions more desired for the aim of generating interesting game AI behaviours.

Ultimately, despite all of these adjustments, there still remained room for surprising emergent behaviours, as will be discussed in the next chapter. The AI techniques used, such as CMA-ES and MCTS, allowed for beginning from uniformly random solutions and improvement upon them, without being faced with an intractable problem due to an incredibly complex and ill-conditioned search space.

5.8.3 MCTS Test Results

The controller was now in a functioning state with a specific selection of values for the roll-out depth, macro action length, value and priority given to the exploration term in the UCT calculations and the use of a heuristic based on an influence map.

Presented are some results based on 30 runs each with these parameters assigned different values. In each of these cases, the exact same scenario was presented, with a mine and a pickup placed into the simulation at the same positions each time. The vehicle was granted 500 timesteps in order to collect this pickup, and then evaluated based on the following metrics. For the number of collected pickups, p_c, collected mines, m_c, total pickups, p_t, elapsed timesteps, t, and proximity of the vehicle to the closest pickup point, s_p, the score at the terminal state of the game, s_t, was as follows:

$$s_t = \begin{cases} (500 - t) - 10m_c, & \text{if } p_c = p_t \\ \frac{1000}{s_p} + 10p_c - 10m_c, & \text{otherwise} \end{cases} \tag{5.12}$$

In order to test each of these parameters, the other parameters were set to default values based on the most recent values known to produce successful results, using the scenario of collecting a single pickup while avoiding a single mine. A linear-based influence map was used, with the value of each cell being based on inverse linear distance to the pickup. Note that these defaults have not necessarily been selected as a result of being the best possible values, but of simply being the values that first lead to some successful result.

Table 5.7: Default Values

Parameter	Value
Influence Map	Linear falloff
Rollout Depth	50
Macro Action Length	15
Exploration Term k	1.4

With the values in Table 5.7 selected as defaults, it was simple to vary the values to see how performance was affected.

Table 5.8: Rollout Depth Scores

Rollout Depth	Mean Score	Std Dev
20	48	56
50	72	86
100	32	39
200	3	5

From the results in Table 5.8, it is interesting to observe the mean result for the chosen rollout depth default to be higher than others. However, this matches earlier observations where a rollout depth that was too short did not get a great deal of information from the heuristics to differentiate that rollout from others, whereas rollouts with larger depths gathered too much information and were more susceptible to misleading noise from rollout values.

Table 5.9: Influence Map Scores

Influence Map	Mean Score	Std Dev
Linear Falloff	72	86
Logarithmic Falloff	56	68
No Map	58	81

Initial observations, shown in Table 5.9, showed that the influence map worked better when given weightings based on linear proximity to the pickup as opposed to a sharp logarithmic falloff. The results in Table 5.9 seem to support this find, and that using influence mapping in the heuristic produces better results than no influence mapping at all. It is however notable that some rare successful pickup collections were observed without the influence map. It may have been the case that the changes made in addition to adding the influence map have made it easier to find working solutions, such as the heuristic adjustment to avoid bouncing.

Table 5.10: Macro Action Length Scores

Macro Action Length	Mean Score	Std Dev
15	72	86
30	119	100
50	48	63
100	21	38

From the results of testing macro actions, seen in Table 5.10 it is clearer that a macro action length of 30 is likely more preferable than 15. To keep the results consistent, the value of the macro action length has been kept at 15 for all tests other than the macro action length test, but it has been noted that better results come from this value.

Table 5.11: Exploration Term k Scores

Exploration Term k Value	Mean Score	Std Dev
0.3	40	56
0.5	38	53
1	72	90
1.4	72	86
2	40	62

Table 5.11 shows the results of testing different values for k when calculating the UCT value for a tree node within MCTS, with the 1.4 value used as a default from its initial success in being used on a simple trial and error basis. There is not much distinction in using 1.4 instead of 1, but these values are still more suited than values above and below, suggesting that perhaps more exploration priority may be required than the values 0.3 and 0.5 would give, but that too much exploration is detrimental overall.

5.8.4 Interesting Movement Examples

To briefly mention some examples of interesting movement seen before, a significant consequence of the addition of a bounce penalty to the heuristic is the vehicle no longer accelerating into the edge of the world, but instead ceasing to accelerate, sometimes slowing down near the edge of the world and then speeding up after either turning away from the edge of the world or after the unavoidable bounce has occurred (see Figure 5.7).

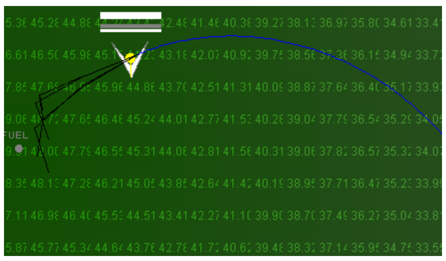


Figure 5.7: The vehicle ceases acceleration as it approaches the edge. The rollouts being tinted black indicate that every foreseeable action will involve a bounce on the wall. After this bounce is completed, the vehicle collects the pickup almost immediately afterwards.

Mostly, the vehicle will tend to curve in a sensible path towards the pickup without much deviation, as shown in Figure 5.8.

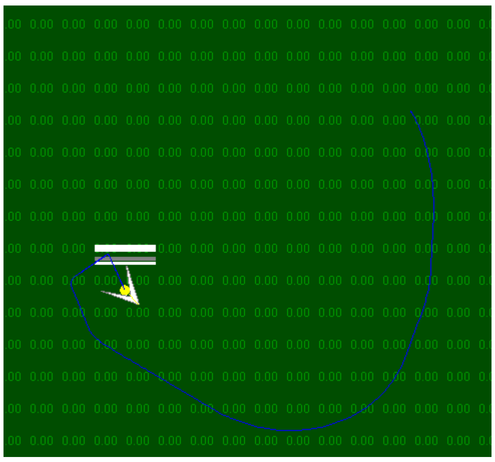


Figure 5.8: The vehicle curves towards the pickup, slightly overshoots, slows down and turns more precisely directly into it. The vehicle is at the position where the pickup was.

However, an odd re-occurring defect seen in the cases where the vehicles were not successful tended to be orbital movement, perhaps relating back to the issues faced with the steering-based controller used before MCTS, or perhaps also based upon some form of local minima in the influence map.

It most frequently occurred near a pickup, which could be explained by some combination of factors. These factors included an inability to control momentum and compensate for it due to rollouts being too short, as well as the local minima reason of becoming trapped in trying to approach the best available influence, which would naturally lead to a orbit, as seen in Figure 5.9.

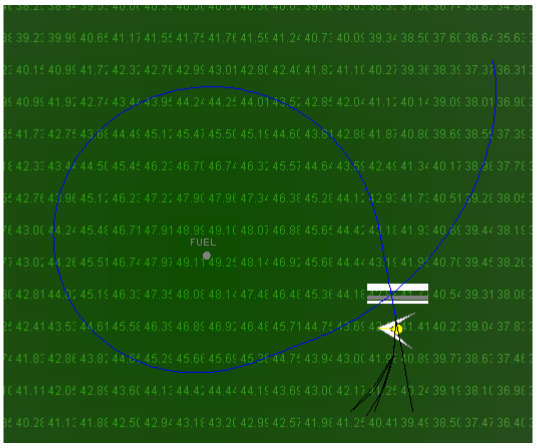


Figure 5.9: The vehicle travels around the pickup, never to collect it.

5.9 Chapter Conclusions

Having once again fallen into a trap of attempting to analyse a situation with too much complexity to it to be sure of what had been the primary issue of fault, the original aims of investigating vehicle interactions were set aside. The issue of the limitations produced by using a steering-based reactive controller became the primary focus of research and investigation. The original combat scenario was replaced by a simpler scenario for individual vehicles to solve, and, similarly to the *Lunar Lander* experiments in the previous chapter, involved something that appeared superficially trivial until it became evident that the approaches used to this point were failing to produce solutions of any appreciable effectiveness.

After switching to a more resilient, adaptable method of controlling vehicles by using MCTS to control them, the original intent of investigating the impact on designs and behaviours for vehicles caused by direct competition became a viable topic to investigate again. However, rather than rescale everything back to the original overly complicated scenario, with teams of vehicles attacking each other with pickups available in the environment, a similarly scaled down scenario was constructed to investigate vehicle interaction in a more focused manner.

This new scenario was a simple chase scenario. A vehicle was chosen to be a predator and another vehicle chosen to be prey, with each vehicle having mutually incompatible and unique goals to fulfil during a simulation of fixed duration. Using the new and improved MCTS controller framework as a base to tune and refine would allow for the behaviours expected from these previous vehicle combat and pickup collection experiments to appear.

In the next chapter, this predator-prey scenario will be further explained and the findings from this more reasonably scaled scenario discussed and displayed. Given that a vehicle could collect a pickup and avoid sources of danger by the end of the pickup collection experiments, could a vehicle also approach a moving target and flee from a moving source of danger in an effective, sophisticated manner?

Chapter 6

Predator and Prey Scenario

6.1 Problem Summary

6.1.1 Overview

The predator and prey scenario shares many conceptual elements with the previous combat and pickup collection scenarios. A vehicle must pursue a single quarry, which is itself another vehicle attempting to evade the pursuing vehicle. However, this scenario is simpler than the combat scenario while remaining non-trivial and including interaction between vehicles as part of the criteria for design that evolution must satisfy.

The aim of this scenario was to be as simple as possible while allowing for competition and interaction between two agents. However, once a controller was functioning as expected and the two agents were acting in a more or less optimal manner, it became clear that the problem had become too simple, and there were no particularly interesting behaviours observed as a result. The game topography was changed to promote more interesting behaviour, becoming an enclosed circle. To provide a more challenging scenario to overcome, slowly moving asteroids were added to serve a similar function to the static mines in the previous chapter.

6.1.2 Specialist Goals

Within this scenario there exist two vehicles as agents with conflicting goals. The predator vehicle must attempt to intercept and collide with a prey vehicle, which must attempt to not allow this to happen for as long as the finite simulation time demands. As these goals are both conflicting, different characteristics of design and behaviour were theorised to be required, as it can be seen in nature that the needs and effectiveness of designs for predator and prey organisms differ based on their specialised roles.

6.1.3 Varied Approaches

While there was some progress made in evolving vehicles versus an initially fixed design opponent using the same controller as the ones provided to the evolving vehicles, an initial situation of evolving a vehicle against an unresponsive, static design was not exactly what had been intended to be tested in this scenario. After the experiment was shown to be functional using this initial approach, two other approaches were tested to investigate coevolution of vehicle designs based on interaction between different vehicles.

The first of these coevolutionary approaches was to use a single population and test vehicles both in their performance as predator vehicles and as prey vehicles, taking competing pairs and testing their roles by treating each vehicle as first one role and then the other. The second of these approaches involved two populations of dedicated specialists, one population of predators, and one population of preys.

6.2 Methodology

6.2.1 Spaceship Competition

As it was shown that evolving designs versus an opponent of a fixed, static design could produce interesting results, coevolutionary variations of the scenario were next considered.

Two coevolutionary experiments were used. In the first, all vehicles within the population were assigned a fitness based on their mean performance as both predator and prey. In the second, two separate populations of specialist ships were used, where one population consisted solely of predator vehicles, and the other of prey vehicles. In both coevolutionary experiments, a tournament was used to determine which vehicles should be tested against which other vehicles, with the mean performance of a vehicle over its matches used as the fitness.

While common practice when using a tournament in an evolutionary context is to assign fitness to the population members based on their ranking within the tournament, the tournament selection was used only as an alternative to other means of assigning vehicles to each other as competitors, such as a round-robin tournament, and not as a means of evaluating the fitness of the vehicles.

This method was used to reduce computation time for the experiments, and admittedly may not have been the most optimal selection method used for evaluating vehicles against one another. It was also decided that to include tournament ranking as part of the fitness measure for coevolutionary experiments would produce results that could not be fairly compared against results from non-coevolutionary experiments. For easier comparison, tournament status was not incorporated into the fitness measure for these reasons.

Vehicles would receive an assigned fitness based upon the mean of their performance in all rounds they qualified to compete in. While this meant that vehicles would effectively be tested until they failed, the intention was the averaging out of unexpected fluke victories.

It took more time than originally anticipated to gather the data from these experiments due to the number of variables being tested. While the original aim was that more scenarios would be tested in a similar way to this scenario, or that the previous complex scenarios could be revisited, the decision was made to reduce the scope to keep the experiments within manageable time constraints.

6.2.2 Use of Different Controllers

For the purposes of these experiments, four controllers were used. All controllers controlled their associated vehicle by providing an action indicating whether to accelerate and/or turn, and if to turn, whether to turn clockwise or counter clockwise.

These controllers were as follows. The first controller tested can be described as a 'condition-action' or a very simple rule-based controller. The second controller was perhaps the simplest in concept and execution, being a simple one-ply greedy search controller. A flat or uniform Monte Carlo search based controller was used as well, which uniformly selected actions to simulate rollouts from its given state. Finally a Monte Carlo tree search controller was used to examine how the simpler controllers fared in comparison.

Evolutionary Condition-Action Controller

The condition-action controller used scenario-specific pairs of conditions and actions.

A condition consisted of a check relating to some specified scenario value, such as distance to the nearest asteroid or distance to a ship, and a comparison for whether or not it was below an evolved threshold.

If the value being considered fell below this evolved threshold, the action associated with this condition was used by the controller. The conditions were specified in a priority based order such that later conditions to be evaluated had an intrinsically higher priority, overriding any conditions that preceded them.

This incredibly simple reactive evolutionary controller was much faster to execute and repeatedly simulate than an MCTS controller, but it didn't produce especially interesting, compelling or even effective behaviour.

The condition-action controller used was a very simple controller working on three rules, working in a similar manner to previous reactive controllers that had no capacity for planning but with the hopes that a more specialised ruleset would bring greater success. Of the three defined rules, the first rule was always active, while the other two rules were of higher precedence but were based upon proximity to some specific element of the game environment.

These two proximity based rules were based upon explicit thresholds defined by a vehicle design's chromosome in a similar manner to the weights used in the spaceship combat scenario.

The first rule of three affected movement, either towards or away from the other vehicle, dependent on the vehicle's role. Predator vehicles would aim to move towards the prey vehicle, and prey vehicles would aim to move away from the predator vehicle.

The second rule had priority over the first, and determined whether or not the vehicle should avoid collision with the edge of the world. For the wrapping or toroidal world topology, this rule was ignored. If the vehicle was within a specific range of the edge, it would attempt to decelerate and change course away from the edge. The direction the controller would attempt to match in its trajectory was a perpendicular direction to the world edge at the nearest point to the vehicle.

The third rule had the highest priority of all three rules, and was trigged by proximity to asteroids. Much as with edge avoidance, the vehicle would attempt to decelerate and move in a direction opposite of the direction from the vehicle to the asteroid. This was to avoid impacts with asteroids.

The weightings used by the condition-action controller are not used by the other controllers and are unique to the condiction-action controller alone. Other controllers would not use these parts of the chromosome and they were instead retained for compatibility reasons.

Greedy Search Controller

The greedy search controller simply examined, for each possible action combination out of movement, turning, or taking no action, the value for the state reached by each action. The best value calculated was used to determine which action to take. As with previous experiments, this greedy one-ply lookahead strategy for attempting to solve a slightly more complicated scenario led to behaviour that only worked when circumstances were straightforward and very little complications were involved.

This controller frequently led to vehicles that would crash into asteroids and return a fitness score of zero, stalling evolutionary progress, simply due to an inability to react to asteroids due to not even considering their presence until one macro-action step too late.

Uniform Monte Carlo Controller

The uniform Monte Carlo controller was almost a halfway point in execution between the short-sighted but fast execution of the greedy search controller and the rollout-based sampling of potential consequences for actions granted by MCTS. For each possible action, the uniform Monte Carlo controller performed a fixed number of rollouts, and treated the mean value of all rollouts from a state arrived at by a specific action as the value for the action in question. The action with the best value after this fixed amount of sampling would be selected. In a sense, this controller was a naive approach to the x-armed bandit scenario with an exploration strategy of simply trying n options $\frac{t}{n}$ times, where t would be the total number of samples or rollouts allowed.

Not having anything more sophisticated than this, the Monte Carlo controller was incapable of exploring or exploiting the search space any further or deeper than the one-ply shore granted by considering the very immediate next states its next macro-action would arrive at, but the use of rollouts to sample potential consequences for actions made this controller more robust in situations that would lead to failure states for the greedy search controller.

Use of MCTS

Much of what was discussed regarding the use of MCTS in the previous chapter applies to the experiments conducted within this chapter too, although there was greater success in applying MCTS to this simpler problem with fewer active controllers in the simulation at a time.

There were initially multiple setbacks in getting a Monte Carlo Tree Search controller working and operational. Many simpler controllers were developed for testing and comparative purposes in the process of developing a functional MCTS controller, such as the aforementioned greedy 1-ply lookahead controller for this scenario, and a simpler uniform Monte Carlo controller which examined all possible actions with equal probability, but to lesser effectiveness.

The MCTS controller was implemented with a selection strategy of UCT, and was conducted as an open loop search. In a closed loop search, every node of the constructed search tree corresponds to a specific state of the simulation reliably. The open loop search used meant that every node instead contained the number of visits and total accumulated value of rollouts achieved while selecting and expanding nodes. This statistical data was used rather than raw state value for each node of the search tree, helping to build a probabilistic view of of the search space.

While the problem examined was deterministic, implying a closed loop approach may have been just as successful, when it came to competition amongst different controllers neither had a very strong model of the opponent's actions taken during their own rollouts, simulating them as if they had taken no action at all. While the simulation itself remained deterministic, true prediction of the other controller's actions when using a stochastic approach such as MCTS would cause the problem, if not the simulation, to be stochastic in nature.

The MCTS controller also used depth-limited rollouts, as rolling out every decision point until the end of the simulation proved to result in noisier and less useful values for nodes. Limiting the rollouts to a specific depth required a focus on immediacy and promoting more active behaviours. For all MCTS usage in this chapter, the depth was limited to 100 timesteps. Other values used, such as the number of rollouts per decision point, are given in Table 6.1.

The MCTS controller had only one significant downside in relation to other controllers. As a more complex controller, it required significantly higher processing time than the much simpler reactive evolutionary or 1-ply lookahead controllers. However, the MCTS controller did result in noticeably better performance of vehicles than the simpler, faster controllers.

In an attempt to make the MCTS controller more effective for the task predator vehicle control, an MCTS controller controlling a predator vehicle would track not only the current position of the prey vehicle, but its predicted position based upon its current velocity if the prey vehicle was not likely to change it. This allowed for some smarter course changes causing interception of the prey vehicle, but this was frequently not seen, as the prey vehicle's trajectory was rarely in a direction where such interception would be feasible.

Figure 6.1 shows an example of using MCTS rollouts for this experiment, using fixed vehicle designs as an initial test of the implementation of the MCTS controller.

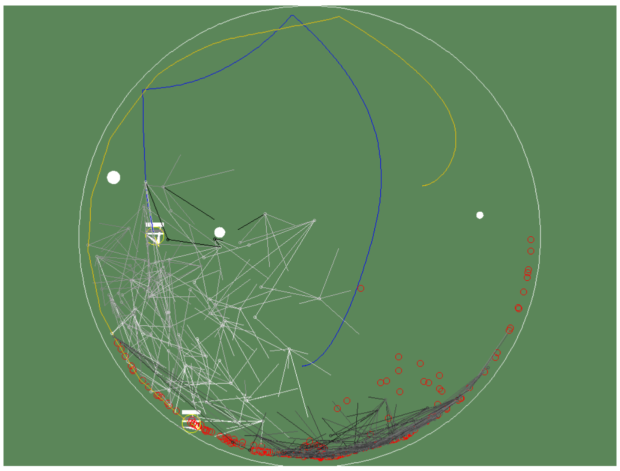


Figure 6.1: Example of the predator prey scenario in action with fixed design ships, both using MCTS based controllers. The blue trail belongs to the predator, and the orange trail belongs to the prey. Red circles are predicted positions of the prey ship within predator rollouts. The trees are shaded based on relative value - most prey state values are lower value, and most predator states are higher value.

6.2.3 Environmental Variation

A set of the environmental constraints used for the problem were altered to specific values, with the focus being upon one value at any given time in relation to a default set of parameters, as given in Table 6.1.

The ranges used for the parameters are given in Table 6.2. Following is a description of what each parameter meant and affected. Some of the parameters chosen for analysis were parameters that had had drastic impact on the *Lunar Lander* scenario previously investigated, while others were chosen for variation as they were new to this experiment, such as the asteroid parameters. The thruster speed was not known to produce any significant differences in behaviour, but it was not known if this would remain true for this scenario as well.

Table 6.1: Default Parameters

World Type	CIRCULAR
Friction	0.99
Asteroid Count	4
Asteroid Size	5 - 15
Asteroid Speed Range	20
Thruster Speed	1000
MC/MCTS Iterations	4000
Function evaluations	300
Max timesteps per sim	700

Table 6.2: Parameter Values. Values in bold mark the default values used when a parameter was kept constant.

World Type	BOUNDED	WRAPPING	**CIRCULAR**
Friction	1	**0.99**	0.8
Asteroid Count	0	**4**	8
Asteroid Size	1 - 5	**5 - 15**	15 - 25
Asteroid Speed Range	0	**20**	40
Thruster Speed	500	**1000**	2000

World Type The topology of the world could be one of three distinct types, *bounded*, *circular* or *wrapping*. With *bounded* topology, the playing field was rectangular with explicitly defined edges that vehicles could not cross. All objects that collided with the edges of the playing field would have their velocity reflected, similar to billiard balls bouncing off of the edges of a table. The *circular* topology also included an impassable boundary, but defined as a circle of an explicit radius as opposed to a rectangle. As a result of this change in the shape of the boundary, there were no corners that a predator ship could chase a prey ship towards. The *wrapping* topology involved a rectangular playing field, as with the *bounded* topology, but with the change that objects moving off of one edge would reappear at the opposite edge. The environment created by this would be closer conceptually to objects lying on the surface of a torus than the flat rectangle or circle of the other topologies used.

Velocity Loss This parameter determined a velocity loss coefficient that was applied to all vehicle velocities within the simulation per simulation frame, or tick, as in the previous scenarios. The velocity loss property tends to have an immediate effect of causing inertia control to be less difficult. This is no different within the context of this problem.

Asteroid Count This parameter determined how many asteroids were present within the simulation. There would always be exactly this many asteroids. While the next two parameters were random in nature, it is important to note that all random generation functions were fixed with a universal seed across every experiment carried out.

Asteroid Size This parameter determined the upper and lower bounds of a uniform size distribution. Asteroids generated would have sizes according to this distribution.

Asteroid Speed Range This parameter determined the upper bound of a uniformly random speed asteroids could possess. Asteroids could possess a lower speed than was listed as the speed range.

Thruster Speed This parameter determined how much force each of the thruster components of a vehicle could produce. All thrusters could be either on or off, and would either produce this amount of thrust or no thrust depending on this parameter. The unit of the thrust is similar to Newtons, although given in terms of pixel acceleration instead of metre acceleration.

6.2.4 Ship Scoring

Each variant of the predator prey experiment, whether vehicles competing against a fixed opponent, competing against vehicles in the same generalist population, or evolving in separate specialist populations, was tested with all four controllers as detailed earlier, with environmental parameters varying as listed.

Fitnesses were calculated based on the terminal state of a simulation. The simulation reached a terminal state either when the number of timesteps elapsed reached a maximum, or if the predator vehicle 'caught' the prey vehicle. Upon reaching this terminal state, the simulation would halt and the fitness calculation would be made.

A predator was considered to have 'caught' a prey vehicle if it had come into collision with it. Collision was defined as an intersection between the hull polygon of the predator and prey vehicles.

Predator fitness scores were defined as follows, for fitness f, where d indicates the magnitude of the distance between ships, t the number of timesteps, d_{max} the maximum possible distance between the ships and t_{max} the maximum permitted timesteps within the simulation:

$$f = \begin{cases} 1 - \frac{d_{max}-d}{2d_{max}} & \text{if prey uncaught} \\ 1 - (\frac{t_{max}-t}{t_{max}} + \frac{1}{2}) & \text{if prey caught} \end{cases} \tag{6.1}$$

Prey scores were defined as follows:

$$f = \begin{cases} 1 - \frac{d_{max}-d}{d_{max}} & \text{if prey uncaught} \\ 1 & \text{if prey caught} \end{cases} \tag{6.2}$$

These equations were designed to produce lower values to indicate greater fitness, in order to keep the problem as one of minimisation, and normalised to provide a scale indicating at a glance how effective a design was. A vehicle with a fitness value of zero would be hypothetically perfect in this scenario against the opponent, with the implication of the opposite if the vehicle had a fitness of one.

If either a predator or a prey vehicle came into contact with an asteroid, the score for the colliding vehicle was set to one.

The fitness functions used here are not zero-sum, and in retrospect this may have been a fairer metric to use. These fitness functions were designed primarily with a focus on predator behaviour and promoting catching ships, as prey fitness was more difficult to quantify due to markedly different expectations.

However, by evaluating behaviours independently of one another, the measures used allow for the improvement of both predator and prey without running into evolutionary stagnation due to a mandatory zero-sum fitness, where neither side is able to improve.

Given these fitness metrics, the task for the prey of eluding a predator is considerably harder to achieve than the task for the predator catching the prey. It also means that performance of predators and preys cannot be directly compared to each other due to the different scoring metrics used. Only performances of specific roles can be compared to each other, such as a predator from one experiment being compared to a predator from a different experiment.

It also means it is harder to come to a conclusion as to the fairness of the metrics when compared directly against each other. The prey metric did not allow for good scores nearly as easily as the predator metric did, but prey vehicles were also being given a more challenging role to perform, which was to survive for an amount of time, rather than attempt to catch a target.

6.3 Results

All results presented are the mean of the best fitnesses at the end of experimental runs with the given environmental parameters. Best results are in bold.

6.3.1 Unexpected Adaptations

It was noteworthy that a reoccurring defense mechanism of the prey populations exploited a particular oversight of the simulation code. The mechanism for determining if a predator had caught a prey was based upon collision detection between the polygonal structures of the vehicles formed by the positioning of their components, as opposed to the simple bounding circle collision detection used in previous scenarios.

The evolutionary process would exploit this for prey vehicle designs, leading to designs which made the intersection detection less reliable. Prey vehicles would have strange, winding shapes with thrusters placed far away from their centre of mass to produce long strips that would be less likely to register as colliding with anything. Components would also be distributed in such a way that the vehicle's physical centre of mass was displaced from the ship's geometric centre, meaning predator vehicles, which would use the distance between their geometric centre to the prey vehicle's geometric centre, would have inaccurate detection of where the prey ship was. All four controllers used distance to the prey vehicle geometric centre as a metric to guide them closer to the prey vehicle, thus making this a useful behaviour to develop for evasion purposes.

This same exploitation of the underlying physics model was adapted by predator vehicle as well, as seen in Figures 6.3 and 6.5, as it would mean surviving collisions with asteroids that would otherwise destroy them. However, as seen in Figure 6.4, this reduction of collision hull presence could also be to the detriment of the predator at times.

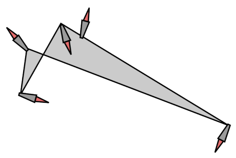


Figure 6.2: An example specialised prey ship design. Not shown on this depiction is the geometric centre of the ship, which sits more to the upper left of the design as shown than might be expected. The winding of the ship polygon makes intersection detection less likely.

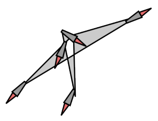


Figure 6.3: An example specialised predator ship design. It too has interesting winding of its ship polygon, sacrificing better collision detection with the prey in order to avoid colliding with asteroids, and appears to be designed for quick bursts in a specific direction.

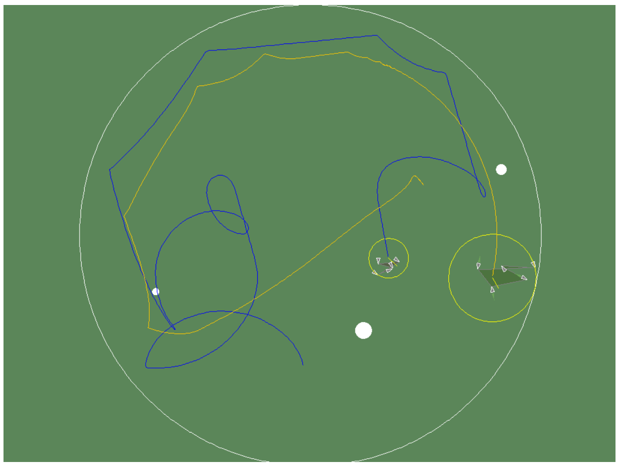


Figure 6.4: Example of the trajectory of an incredibly agile predator, made less effective due to its collision hull being so small it barely registers as in contact with a prey ship or an asteroid in most cases.

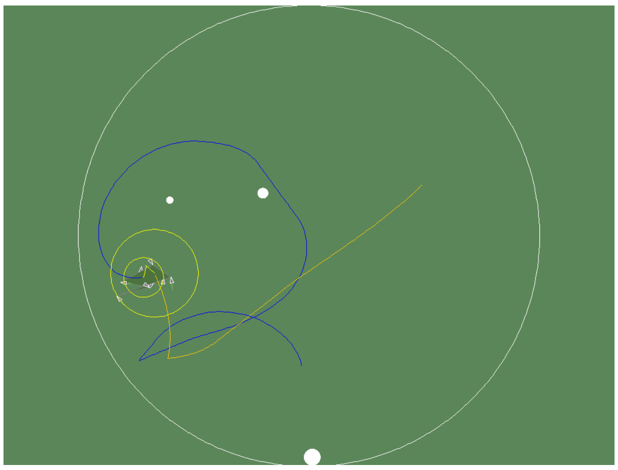


Figure 6.5: The previous agile predator in a case where it successfully catches the prey ship.

6.3.2 Effect of Different Approaches

Static Opponent

This was considered to be the base state of the experiment and a means of comparison. While statistical progress was made by evolving populations versus a static opponent, a static opponent did not require a significant amount of behavioural or design adaptation. Vehicle designs that could catch or elude the static opponent would simply continue to do so with little pressure to improve.

Same Population

While it may have been due to the nature of the tournament selection process used to select vehicles to compete against each other, the general impact of coevolution within the same population did not appear to produce noticably better or worse results in fitness.

This may have been due to the nature of the fitness landscape changing when the basis of comparison was itself changing in addition to the population being tested, as both the test population and the competition selected for the test population were one and the same.

It may also have been the case that coevolution of a single population would not confer an advantage noticable within the simulation length and number of simulations and generations used, and that longer experiments would produce more noticable results.

Shown in Table 6.3 are the means across all runs for each controller using the static opponent approach. Comparing to Table 6.4, which shows the same but for the same population coevolution approach, the average fitness appears to be better (lower) in some cases for some controllers but not for others, with no clear superior approach and very similar results as indicated by a low standard deviation across both approaches.

While these two approaches can be compared side by side, it is difficult to do so for the specialist populations approach due to a different fitness metric.

Table 6.3: Fixed Opponent Controller Comparison

	Condition-Action	Uniform MC	Greedy Search	MCTS
Number of Runs	37	37	37	37
Average Fitness	0.3890	0.4219	0.4148	0.4457
Std. Dev. Fitness	0.0560	0.0666	0.0517	0.0431
Best Fitness	0.2682	0.2714	0.2487	0.3707
Worst Fitness	0.5075	0.5470	0.5119	0.5561

Specialist Populations

It was intuitively apparent, and backed by Table 6.12, that specialising a population for a specific purpose would lead to a population better fitted for that purpose than a population

Table 6.4: Single Population Controller Comparison

	Condition-Action	Uniform MC	Greedy Search	MCTS
Number of Runs	37	37	37	37
Average Fitness	0.4184	0.4041	0.4179	0.4146
Std. Dev. Fitness	0.0510	0.0501	0.0405	0.0446
Best Fitness	0.2977	0.1999	0.3064	0.2449
Worst Fitness	0.5148	0.4776	0.4883	0.4733

with pressures to create a more generalised design. However, as can be seen in Table 6.13, it was also noteworthy that the sheer contrast between fitness values reported for predator and prey ships may have highlighted a shortcoming of the asymmetric fitness measures used.

Table 6.5 shows the results across many runs for the predator specialist population, and Table 6.6 shows the same for the prey population. As already mentioned, these tables highlight the discrepancy in the fitness functions used for the predator and the prey, but from the predator results alone indicate that better performance was achieved via this separation into specific roles. It is worth noting that, given a result of 1 for a prey vehicle being caught, none of the runs examined finished with a prey vehicle that was not capable of evading the predator for the duration of the simulation.

Table 6.5: Specialist Predators Controller Comparison

	Condition-Action	Uniform MC	Greedy Search	MCTS
Number of Runs	37	37	37	37
Average Fitness	0.1251	0.1354	0.1580	0.1099
Std. Dev. Fitness	0.0649	0.0790	0.0971	0.0655
Best Fitness	0.0104	0.0271	0.0494	0.0185
Worst Fitness	0.2817	0.3812	0.4278	0.2755

Table 6.6: Specialist Prey Controller Comparison

	Condition-Action	Uniform MC	Greedy Search	MCTS
Number of Runs	37	37	37	37
Average Fitness	0.6346	0.5676	0.6165	0.6036
Std. Dev. Fitness	0.0745	0.1461	0.0997	0.1579
Best Fitness	0.5112	0.0697	0.3070	0.0174
Worst Fitness	0.8947	0.7853	0.9102	0.8862

6.3.3 Effect of Different Controllers

Strangely enough, it seems as though the controller used did not appear to have a significant bearing in the performance of vehicles.

However, the choice of controller did appear to affect the fixed opponent scenario, as can be seen in Table 6.10 showing the mean of 3 runs across each parameter and in Table 6.3, but only barely. In this particular scenario, the condition-action controller appeared to perform better than other controllers, which would make a certain amount of sense.

For each of these experiments, the same controller was used to control both the evolved design and the static design. The weights of the condition-action controller were also modified by evolution along with the vehicle design, to be a better fit for its specific design. Mapping this evolved controller weighting to a different vehicle design would not necessarily cause worse performance, but it would not perform as well as it would for the design it was originally improved for. Comparatively, other controllers would perform in a fairer fashion for the fixed opponent compared to the evolved design.

This is shown by the lack of a dominant controller in Table 6.11, showing the results of the coevolution of designs in competition with one another. When both vehicles are controlled by a condition-action controller with weighting unique to their own designs, this advantage from the static opponent scenario vanishes completely.

As can also be seen in the previous Tables 6.4, 6.5 and 6.6, the different controllers used in experiments other than the fixed opponent experiment did not seem to indicate substantially different progress from one another. Each controller produced relatively similar mean average fitnesses and little deviation among best solution candidates across different experiments.

This could be due to the constraints placed upon the number of generations and length of simulations, selected to allow for experimental progress within a reasonable timescale. It was not expected that the more rule-based controller would perform generally better than other controllers in any experiment, but given its capacity for evolutionary improvement and tailoring to the ship design, it is not a completely surprising outcome.

It was unclear as to why more sophisticated techniques such as Monte Carlo based searches would perform less effectively than more reactive, short-sighted strategies, but it is also the case that the said Monte Carlo strategies used for this problem were those that were designed to solve problems on a more general level. The condition-action controller was highly limited but also highly specialised to the problem being examined. This difference may explain why, despite its apparent simplicity, it outperformed Monte Carlo methods in certain specific contexts.

6.3.4 Effect of Environmental Variation

It seemed clear, given the results seen across Tables 6.10, 6.11, 6.12 and 6.13 that the two environmental parameters making the largest difference to the performance and score of vehicle designs were the type of world used in the simulation and the velocity loss co-efficient, applied each simulation step to all vehicles as a sort of universal friction.

The means across all controllers for each parameter type are presented in Table 6.7, showcasing the rough effect each parameter had as a whole. Within this table, specialist populations are given based on the type of specialist, predator or prey. The best values in each parameter group are shown in bold. As can be seen, there is little trend between the different experiments as to which parameters allowed for greater success, although there is an understandable trend to better progress being made with fewer asteroids in existence, and for easier prey evasion when there are no borders in the world topology to be blocked by.

Table 6.7: Environmental Variation Effects

Parameters		Fixed Opponent	Coevolution	Predators	Prey
World Type	BOUNDED	0.4397	0.3961	0.2166	0.5472
	WRAPPING	**0.3474**	**0.3753**	0.1924	**0.4832**
	CIRCULAR	0.4220	0.4462	**0.1097**	0.6582
Friction	1	**0.4147**	0.4502	0.1663	0.6607
	0.99	0.4220	0.4462	**0.1097**	0.6582
	0.8	0.4749	**0.4460**	0.1682	**0.6378**
Asteroid Count	0	**0.4073**	0.4315	0.1220	**0.6277**
	4	0.4220	0.4462	**0.1097**	0.6582
	8	0.4153	**0.4215**	0.1355	0.6304
Asteroid Size	1 - 5	**0.4147**	**0.4263**	0.1523	**0.6542**
	5 - 15	0.4220	0.4462	**0.1097**	0.6582
	15 - 25	0.4408	0.4345	0.1208	0.6772
Asteroid Speed Range	0	**0.4050**	0.4273	0.1355	**0.6021**
	20	0.4220	0.4462	**0.1097**	0.6582
	40	0.4063	**0.4137**	0.1196	0.6554
Thruster Speed	500	**0.4161**	0.4392	0.1234	0.6338
	1000	0.4220	0.4462	**0.1097**	0.6582
	2000	0.4428	**0.4099**	0.1149	**0.6333**

The world topology greatly affecting vehicle performance was not surprising, as the world structure changes the nature of the problem entirely. While a circular playing field did not allow a prey vehicle to become trapped in a corner, the wrapped playing field allowed for far easier evasion as it inherently resulted in absolutely no barriers for prey to surpass while evading a predator. Unsurprisingly, when looking at specialist populations, specialised predator vehicles tended to perform worse when the world was topologically toroid, while prey ships tended to perform a lot better.

The absence of a friction or drag factor in space, while closer to the reality of physics within a vacuum, made the problem of effective movement more difficult. Vehicles must be designed in a way to be able to reduce or alter their inertia in a manner that cannot allow for simple inaction to provide the desired outcome. This additional complexity of the problem resulted in vehicle designs that did not perform as well within these environmental constraints as vehicles in more forgiving environments.

Other than the world topology and the velocity loss factor of the vehicles in the environment, it did not appear that other parameters of the environment had as dramatic an effect. This is itself interesting, as it was hypothesised that factors such as increasing the number or speed of asteroids would create a more hostile environment. Given that asteroid impacts affect all vehicles to the same detriment, it is likely that the threat of asteroids was perhaps not as much of a factor in the simulation as first thought.

Adjusting thruster speed for all vehicles did not seem to make a consistent change across different configurations of controller and environment. As both predator and prey used the same thruster speed, it is likely that any advantage conferred by higher thruster speed to either predator or prey is also conferred to the opponent vehicle, and the same with any disadvantages. This would necessarily result in a negation and lack of any advantage or disadvantage to either vehicle.

6.4 Evolving Predators Against A Fixed, Evolved Design

As the previous experiments evolving against a fixed spaceship proved to be less than informative, further experiments involving evolving populations of predators against a previously evolved prey ship design were carried out.

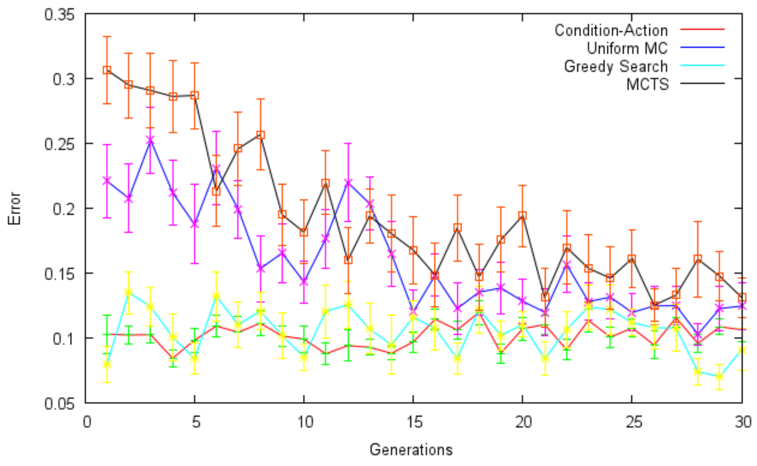


Figure 6.6: Performance of different evolved populations of ships with the same conditions over 18 runs. All ships were tested against a fixed ship design that been evolved to use an MCTS controller. The standard error of the performance error across varied populations is also depicted.

These additional experiments can confirm the flexibility of a more sophisticated controller in its ability to promote more effective design evolution. This experiment was nearly identical to the previous experiments but focusing only on evolving predator ships, with one of the best prey ships of the previous population used as the basis for comparison and evaluation. As can be observed in Figure 6.6, the ability of different controllers may initially seem to conflict with the previous discoveries and assumptions, as the simpler controllers start off performing simply better and the more complicated controllers perform worse.

While it may appear that MCTS performs worse than every other trialled controller, it should be noted that the trialled ship used as a fixed means of comparison was one of the best coevolved prey ships from a population using MCTS as the means of ship control. What this meant as a result was that the initial random population of ships generated during the MCTS run would all be less adapted for use by MCTS than the initial prey, meaning the initial prey begins the experiment with an advantage in the case where the predator and prey ships are being controlled with MCTS. The initial prey ship and its original fitness can be seen in Figure 6.7.

However, as evolution takes place, the predator designs are able to adapt to the behaviour of the fixed prey ship where the fixed prey ship specifically cannot adapt, as the controller tuning remains the same for both the predator ships and the fixed prey ship.

As ship designs are tested fairly with the same controller for both, the initial random population of predator ships begin at something of a disadvantage proportional to how effective the controller is at using the fixed prey ship to evade the predators. This explains why the initial performance for each controller is worse depending on the general effectiveness of the controller overall. A more effective controller means a prey ship that begins more readily equipped to evade. However, as seen with the more basic controllers, less improvement is seen the less sophisticated the controller is, as the evasiveness and ability to manoeuvre well of the prey ship is a motivator for evolution. When a prey ship's behaviour is so simple that a straight line will catch the prey, there is little room for improvement on designs and especially little room for improvement of the weightings for the condition-action controller.

With this reference to evolutionary progress against a fixed design, it becomes clear that a more sophisticated controller can lead to initially worse behaviour comparatively due to the equal playing field established during these experiments, but that this initial imbalance begins to rapidly dwindle as a smarter controller allows for more effectiveness in improvements.

It should be noted that the results observed here differ from the evolution against a fixed ship as the test case for the fixed ship was a specific spaceship opponent with very basic capabilities, and not a design from the end of the coevolution of a population of predator and prey ships. It is likely that, if the experiment had been conducted for more generations than shown, the lack of evolutionary pressures could have resulted in the same stagnation witnessed for the much simpler fixed prey ship experiment. Based upon Figure 6.6, it is incredibly likely that this stagnation has already become the case for the simpler controllers. There is little room for improvement in

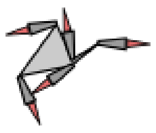


Figure 6.7: Fixed prey design used for testing evolution and controller comparisons. Evolved from a coevolutionary MCTS run where both predator and prey were controlled by MCTS in default environmental conditions. This ship scored a prey error of 0.442, where 0 would indicate a theoretical maximum distance from the predator and full survival for the entire simulation.

the behaviour through design of simpler controllers against an equally intelligent fixed design.

6.4.1 Controller Comparisons

To examine the underlying assumption that identical controllers would initially fare worse against each other based on complexity, a further experiment tested the produced evolved population of ships from the previous experiment against the same fixed prey ship, but with different controllers. The initial random distribution of ships for each controller was tested, with the results shown in Table 6.8, and the final evolved population of ships was tested similarly, with results shown in Table 6.9.

Figure 6.8 shows an example of the simulation where two vehicles used two different controllers versus one another.

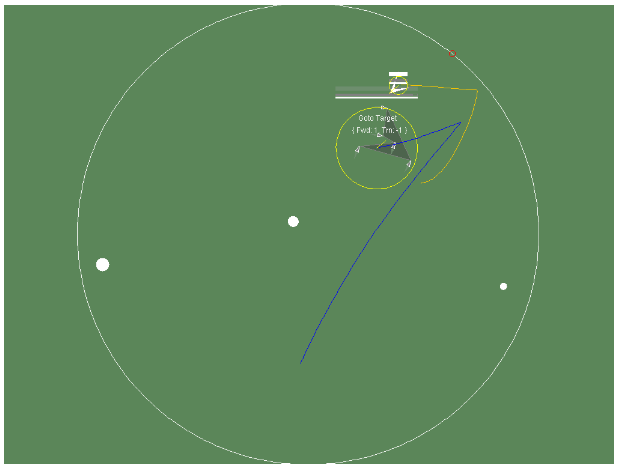


Figure 6.8: Example of the predator prey scenario with an evolved predator ship, using a condition-action controller. The prey is using a uniform Monte Carlo rollout based controller.

As can be seen, more sophisticated controllers were more effective against less sophisticated controllers, but interestingly enough, effectiveness for evolved ships at the end of a run actually decreased compared to the initial random distribution. This is likely due to over-fitting, as the ships being examined at the end of their evolutionary run were specifically evolved to compete against a ship sharing the same controller as themselves. In this different context, it is not entirely surprising that these adapted ships perform less effectively against other controllers.

While the comparison of low level reactive controllers to high level planning controllers can be considered unfair, it should be noted that one of the initial research problems posed was to examine whether controllers evolved alongside the design of a vehicle would be, by evolving alongside a given vehicle design, capable of behaviour that could match or perhaps even exceed the performance of vehicles with unmodified higher level controllers.

However, it is clear to see that this is not the case here, and that high level controllers remain more capable than low level reactive controllers. The greedy search controller is presented as an example to compare the performance of the condiction-action controller to as a fairer comparison, and it is evident that the greedy search controller also outperforms the evolved condiction-action controller.

One interesting outcome of this experiment appears to be the indication that the capacity for evolved weightings does not allow the condition-action controller to outperform any other controller, thus suggesting it was perhaps too simplistic in its formulation.

Table 6.8: Performance of ships against a fixed evolved prey ship, with initial randomised designs. Easiest opponents for each controller are bolded.

Predator Controller	Fixed Prey Controller	Mean Error	Std Err of Error
Condition-Action	**Greedy Search**	**0.2974**	**0.0318**
	Uniform MC	0.4285	0.0268
	MCTS	0.3972	0.0262
Greedy Search	**Condition-Action**	**0.1587**	**0.0266**
	Uniform MC	0.3613	0.0330
	MCTS	0.4008	0.0171
Uniform MC	**Condition-Action**	**0.1971**	**0.0303**
	Greedy Search	0.3107	0.0352
	MCTS	0.3960	0.0238
MCTS	**Condition-Action**	**0.1509**	**0.0232**
	Greedy Search	0.3300	0.0338
	Uniform MC	0.3745	0.0291

Table 6.9: Performance of ships against a fixed evolved prey ship, with final evolved designs. Easiest opponents for each controller are bolded.

Predator Controller	Fixed Prey Controller	Mean Error	Std Err of Error
Condition-Action	**Greedy Search**	**0.3442**	**0.0347**
	Uniform MC	0.4331	0.0324
	MCTS	0.4275	0.0295
Greedy Search	**Condition-Action**	**0.2242**	**0.0286**
	Uniform MC	0.4243	0.0225
	MCTS	0.4237	0.0232
Uniform MC	**Condition-Action**	**0.1833**	**0.0291**
	Greedy Search	0.3153	0.0335
	MCTS	0.4472	0.0162
MCTS	**Condition-Action**	**0.2698**	**0.0319**
	Greedy Search	0.4074	0.0263
	Uniform MC	0.3808	0.0281

6.5 Chapter Conclusions

After getting more complicated controllers to work in a scenario where vehicles directly affected each others' fitness, it became clear that this coevolutionary scenario did lead to something of the intended arms race between vehicles and their roles and behaviours as desired. While evolving vehicles against static designs did not cause much pressure for improvement, evolving specialist populations dedicated to specific roles led to vehicle designs more competent at either pursuing or evading a given target, with adaptations that were not entirely predicted but that made consistent sense emerging from the evolutionary process as well.

The desired agility of vehicles and interesting movement behaviours that had not shown up in previous experiments surfaced in the circumstances created by this experiment, and the evolutionary pressures from other vehicles helped to accelerate this process.

One shortcoming of this experiment in particular was an increasing focus only on evolving the predator designs, with comparatively less time given to efforts to evolve better prey vehicles. The fitness measure for the predator received more attention and care than the prey measure may have, further leading to greater asymmetry between the designs though not as intended.

Altogether, given the fitness of the prey ship designs being tested, all experiments within this chapter indicated that the scenario was intrinsically biased against the prey. This partly is due to the asymmetrical fitness requirements, where predators simply need to touch a prey ship, while prey ships are required to survive for the entire duration. It is also related to the different behavioural requirements. The prey ship must therefore, in order to survive, pose more of an evasive target to the prey ship and not simply move in an easily countered manner, such as directly into a wall. Against an inflexible or otherwise unfit prey, a predator does not need to do much more than move in a straight line. Some of the behaviour of these controllers with direct reference to two fixed designs can be accessed online in the form of short animations.[1]

In the next chapter, the overall observations and findings of how the controller and the design of a vehicle interact with each other to produce behaviours dependent upon both will be discussed and compared across scenarios where possible. While it has been seen from the beginning that simple controllers can provide simple behaviours and complex controllers allow for more effective, less reactive and more flexible behaviours, in the next chapter the ways in how this manifests in different scenarios will be summarised and contrasted.

Table 6.10: Evolving against a Fixed Initial Design

Parameter Type		Condition Action	Greedy Search	Uniform Monte Carlo	Monte Carlo Tree Search
Default Parameters		0.4396	**0.3971**	0.4385	0.4129
World Type	BOUNDED	**0.3792**	0.4530	0.4692	0.4575
	WRAPPING	0.3888	**0.3034**	0.3330	0.3642
Velocity Loss	1	0.4079	**0.3837**	0.4576	0.4096
	0.8	**0.4378**	0.4827	0.4884	0.4908
Asteroid Count	0	0.4083	0.4152	0.4143	**0.3915**
	8	**0.3804**	0.4580	0.3986	0.4241
Asteroid Size	1 - 5	**0.3852**	0.4051	0.4359	0.4327
	15 - 25	**0.4049**	0.4240	0.4440	0.4902
Asteroid Speed Range	0	**0.3492**	0.4266	0.4126	0.4315
	40	**0.3487**	0.4662	0.4046	0.4059
Thruster Speed	500	**0.3891**	0.4333	0.4253	0.4167
	2000	**0.4034**	0.4110	0.4611	0.4957

[1]http://imgur.com/a/NPXct

Table 6.11: Competitive Single-Population Coevolution of Ship Designs

Parameter Type		Condition Action	Greedy Search	Uniform Monte Carlo	Monte Carlo Tree Search
Default Parameters		0.4644	**0.4348**	0.4360	0.4495
World Type	BOUNDED	0.3883	**0.3881**	0.3933	0.4147
	WRAPPING	0.3660	**0.3641**	0.3897	0.3816
Velocity Loss	1	0.4889	0.4457	**0.4062**	0.4602
	0.8	**0.4303**	0.4547	0.4582	0.4409
Asteroid Count	0	0.4559	0.4203	**0.4124**	0.4372
	8	**0.3857**	0.4219	0.4300	0.4482
Asteroid Size	1 - 5	**0.3866**	0.4425	0.4366	0.4393
	15 - 25	0.4342	**0.4013**	0.4525	0.4502
Asteroid Speed Range	0	0.4229	**0.4036**	0.4426	0.4402
	40	0.3934	**0.3827**	0.4274	0.4513
Thruster Speed	500	0.4291	0.4358	**0.4281**	0.4639
	2000	0.4266	0.3988	**0.3941**	0.4202

Table 6.12: Competitive Coevolution of Specialist Population - Predator Designs

Parameter Type		Condition Action	Greedy Search	Uniform Monte Carlo	Monte Carlo Tree Search
Default Parameters		0.0828	0.0835	0.2046	**0.0678**
World Type	BOUNDED	**0.1512**	0.2454	0.2170	0.2530
	WRAPPING	0.2310	**0.1047**	0.1956	0.2383
Velocity Loss	1	**0.1247**	0.2494	0.1342	0.1570
	0.8	0.2171	**0.1081**	0.1585	0.1890
Asteroid Count	0	**0.0865**	0.1751	0.1176	0.1088
	8	0.1291	**0.0984**	0.1944	0.1200
Asteroid Size	1 - 5	0.1303	0.1661	**0.1212**	0.1916
	15 - 25	0.1021	**0.0860**	0.1709	0.1240
Asteroid Speed Range	0	**0.1209**	0.1651	0.1275	0.1284
	40	0.1193	**0.0775**	0.1910	0.0905
Thruster Speed	500	0.1378	0.1051	**0.1029**	0.1477
	2000	**0.0991**	0.1051	0.1126	0.1427

Table 6.13: Competitive Coevolution of Specialist Population - Prey Designs

Parameter Type		Condition Action	Greedy Search	Uniform Monte Carlo	Monte Carlo Tree Search
Default Parameters		0.7486	0.6524	**0.5994**	0.6324
World Type	BOUNDED	0.5709	0.5574	0.5860	**0.4747**
	WRAPPING	0.4393	0.6945	0.5207	**0.2782**
Velocity Loss	1	0.7668	0.6409	**0.6146**	0.6205
	0.8	0.6896	**0.5839**	0.6286	0.6489
Asteroid Count	0	0.6363	**0.5870**	0.6265	0.6611
	8	0.6230	0.6210	**0.5703**	0.7073
Asteroid Size	1 - 5	0.6549	**0.6470**	0.6535	0.6616
	15 - 25	0.7320	**0.6485**	0.6587	0.6694
Asteroid Speed Range	0	0.5982	**0.5340**	0.6199	0.6563
	40	0.7235	0.6446	**0.5928**	0.6607
Thruster Speed	500	0.6720	**0.5640**	0.6504	0.6485
	2000	0.6330	**0.5640**	0.5951	0.7411

Chapter 7

Conclusion

7.1 Overview

The initial questions that were to be answered in beginning this research related to what forms and behaviours vehicles would take and exhibit based upon a very simple 1-ply look-ahead controller using simple heuristics tailored to individual problems. However, as research progressed, it became clear over time that in trying to answer these initial questions, further questions arose. The impact of the environment on the design, both directly and indirectly via the environment's effect on the controller used, was also something that warranted further investigation. In addition, the impact of vehicles directly competing and co-operating with each other drastically changed what defined an effective vehicle and lead to interesting arms races and exploitation of the underlying simulation physics as vehicles evolved to gain more of an advantage over their competitors.

The expectations were that controllers and designs would affect each other based on the strengths and weaknesses of both. It was expected that smarter controllers would allow for better designs with the capability to control worse designs with more adaptability and precision than less smart controllers, and that less smart controllers would by necessity lead to specific types of designs that could be more easily used by them. Evolution of the parameters of reactive controllers did not support these expectations, however, and it seemed that the contexts that these controllers were involved in were too complex for the reactive approaches used.

Evolutionary algorithms have been shown to be a feasible means of generating effective game vehicle designs within the simplest of the contexts used, with variable results for more complicated contexts and scenarios. This evolutionary approach seems to become less effective as more problem dimensions and aspects are introduced, and conflicting and competing goals further make the problem of design appear almost intractable when equal consideration is given.

Different controllers also had different levels of effectiveness across different designs, a relationship that was originally predicted and a hypothesis that held to be true through the course of research. Unexpected emergent behaviours were also seen during the course of investigating different controllers and problem contexts.

7.2 Results Discussion

The initial experiments lead to interesting behavioural patterns, the most frequently observed being "spinner" ships that, while not fast at moving from one point to another, were easily able to change direction on a frame-to-frame basis, allowing for agile manoeuvrability at the cost of linear speed. Tweaks to the heuristics ceased this behaviour as it seemed to be the result of a glitch, but this sort of behaviour was only really prevented by stopping the controllers from having such a high fidelity of frame to frame control over vehicles.

Overall, the initial experiments showed great promise and allowed for the selection of a chromosomal representation that allowed for a wide degree of freedom in design space. This allowance of potential designs would later be shown to be more of a problem than a benefit.

It was originally hypothesised that the simplistic nature of some controllers would by necessity create an evolutionary pressure where designs would arise to overcome the shortcomings and limitations of these controllers for their respective problems. However, it became readily apparent that, outside of simple problems, these simpler controllers were too limited to pilot their corresponding vehicles in a way that promoted any design as a valid direction for evolutionary progress, thus leading to stalling and lack of improvement for fitness and very little result.

This was especially seen in the beginning of the *Lunar Lander* research, where the complexity of the problem meant that controllers that had worked effectively in previous applications failed to produce any sort of gradual solution to the problem. This complexity likely arose from the nature of the problem space, with most paths through the state space of the game leading to undesired terminal states that could be delayed but not easily avoided.

While the combined task of evolving a ship design and a controller or set of behaviours seemed to be largely intractable with the approaches taken, simplyifing the problem to one purely of evolving a set of behaviours to solve the problem produced much better results. The use of evolved macro-actions to guide a vehicle with a fixed design from a starting point to a reasonable ending state proved to be an effective one. From the positive results seen in simply adjusting the behaviour, the question of what caused the problem to be so complex was able to be analysed at all, knowing that the problem was solvable by certain methods.

The conflicting nature of the subgoals relating to fuel conservation and slow landing also were a factor, as it was demonstrated during later experiments that removing the fuel conservation goal altogether lead to more improvement of the population of vehicles. Easier problems with more adverse opening scenarios were also shown to produce more progress than complex problems that remained complex, as these initially difficult problems would be highly unlikely to solve with an initial random population of designs but would rapidly become solved by evolution-guided problem solving methods.

The spaceship combat scenario did not lead to many particularly effective behaviours due to the complexity of the scenario, as while the capacity for recognising the distance to allies was provided it did not seem to be used in any of the tests observed. One idiosyncratic behaviour noticed within this scenario was the 'turret ship' behaviour, where ships would evolve to become very large and begin with a turret component inside the enemy team's starting location. Both sides would attempt to do this in a very rapid highly convergent arms race, as this would mean defeating the opposing ships within a very short window of time simply by firing very soon after the problem was initialised. However, when it came to introducing incentives such as pickups, many ships seemed to be incapable of moving to a static target, oscillating and orbiting it instead due to the nature of the steering based controller attempting to compensate for error to an overcorrective extent that only introduced further instability.

The target oscillation issue persisted into the simplified pickup collection scenario, thus demonstrating that the evolved reactive or steering based controllers were no longer sufficient for the complexity of the problems being considered. During transition work between the pickup collection scenario and the predator and prey scenario, more sophisticated controllers were implemented to greater success at the cost of taking more time to run and of having no evolutionary relationship with the designs they were assigned to.

The predator and prey scenario did not produce as interesting behaviours as might have been expected, nor did it provide much space for evolutionary progress. As both predator and prey were given the same properties such as number of thrusters, neither had any significant advantages such as speed in order to evade, and both populations grew to roughly the same agility. As both predator and prey were tested with the same controller for fairness, there were no advantages in terms of 'intellect' for either party, and with both parties so evenly matched many runs ended due to stagnation and lack of fitness improvement.

Interesting behaviours were observed common to both predator and prey, however. Exploiting the manner in which the hull polygons were constructed, both predator and prey evolved to have shapes that would be less likely to collide with the hazards introduced into the environment, although this came at a cost to the predator of being less able to catch prey and requiring more agility for additional chances to catch prey.

Additionally, while not all observed behaviours were as interesting as expected, there were examples as shown in the chapter where surprising and agile manoeuvres were observed by smaller predators, exploiting their ability to rapidly change direction to loop around a fleeing prey vehicle and collide with the prey vehicle, all before the prey vehicle could successfully decelerate or turn to move away. Despite the general sluggish behaviours shown, the capacity for truly interesting behaviours existed within the system as designed.

The results of coevolution applied to the predator and prey scenario are less clear, as little evidence exists to suggest that the coevolutionary approach lead to any increased or decreased performance relative to vehicle populations that evolved with respect to a fixed opponent.

As controller comparisons with reference to fixed and previously evolved designs indicated during the work on the predator prey scenario, MCTS was a very effective controller. However, surprisingly, the simple 1-ply lookahead greedy search controller was also reasonably effective against the MCTS controller despite having no capacity to plan and being entirely deterministic. While the MCTS controller outperformed the greedy search controller on all counts, it was still surprising to see the final evolved predator ships, controlled by MCTS, perform less effectively against the fixed prey ship when it was controlled by greedy search than uniform Monte Carlo rollouts. This anomaly aside, the performance of controllers relative to each other indicated that the efforts to design a simple but evolvable controller for the predator-prey scenario were not very effective, and suggests that the capacity of a controller to evolve does not excuse it from overly simplistic design overall.

Generally, using MCTS as a controller produced far better results, as MCTS as a control algorithm allowed for more flexible, responsive and better planned behaviour without the need for explicit planning considerations. While a heuristic was used to allow for better action selection in a shorter timeframe, it was capable of reasonable but inefficient action without further guidance. The use of macro-actions helped to reduce the state space that required traversal and exploration, allowing for better behaviour. Additionally, the flexibility and adaptability of this controller approach allowed for more agile ships and behaviours, most noticeably as seen in the predator prey scenario, with predator ships able to tightly curve back on their own trajectories. The predator and prey ships both tended towards smaller designs as well, as this allowed for smaller mass and therefore superior manoeuvrability when attempting to evade or chase the target ship.

Conversely, bulky ships were produced by the simpler controllers, a byproduct of the representation used for components. As simpler controllers would usually fare poorly within scenarios, the reaction from the implementation of CMA-ES used would be to increase the step size used for varying the parameters of the vector of real values encoding a vehicle design. This would have a direct effect of spreading the components out physically farther from the geometric centre of the vehicle, making the ship bulkier. However, for controllers already struggling with smaller, nimbler ships, this would only lead to decreased behaviour, and the problem would eventually become effectively unsolvable, with these doomed runs frequently being halted early due to the population fitness becoming irrecoverable. In some cases, with evolved controllers, even the controllers themselves could become host to disastrous conditions, such as fleeing from vital resources needed to keep functioning, or to negatively weight conditions vital for a higher fitness.

7.3 Summary

In conclusion, it has been shown that evolutionary algorithms are sufficient for problems of design, with some caveats. The formulation of a design problem into a problem of performance within a simulation tied to a measure of effectiveness is non-trivial for many cases, and the exact nature of a fitness measure used can make or break the capacity of evolutionary iteration to produce designs that are improved instead of drifting through design space.

The addition of extra values to the chromosome over time and subsequent experiments only complicated the design space being examined by CMA-ES. Increasing numbers of values meant greater problem dimensions, especially when compared to the initial setup used where controller weightings and component types were not represented as part of the chromosome due to such properties not existing.

The choice of CMA-ES as a means for evolutionary progress had a strong influence on all experiments conducted in this thesis. All problems had a necessary requirement for solutions to in some way relate to a vector of real numbers, the chromosome. This chromosome, the genotype of the solution, did not always impact the representation of the solution in the context of the simulation, or its phenotype, very intuitively.

As an example, changes to values such as the rotation of components compared to changes in the values of the location of components can alone demonstrate some of the problems with treating everything as abstract, disconnected values. While the space of Cartesian co-ordinates used for locations had a simple mapping in all circumstances for chromosomes including values for positional data, the value for an angle of rotation does not have so simple a mapping. For any given rotation, adding or subtracting some multiple of 2π would result in functionally identical behaviour, making a component's rotation's dimension of fitness topologically different in the fitness landscape to, as stated, the dimensions for the x and y co-ordinates of a component's offset.

Additionally, due to this requirement to have a fixed, well-defined vector of real numbers, the number of components a vehicle could have in any experiment was by necessity limited to some number, reducing the capacity for a variety in number of components permitted per vehicle. Allowing for more components would require adding at least three to four new dimensions to the solution vector, as each new component would require a pair of co-ordinates and a rotation, and in certain experiments, a component type as well.

The requirement for all values to be specifically real numbers and not integers also caused some difficulties; certain parameters could not be anything other than integers, such as number of landing pads in the *Lunar Lander* scenario, and the logic for enforcing this had to happen at the level of solution evaluation. As a result, the fitness values for any of these obligatory integer parameters would have a very discrete fitness landscape for this dimension, where any adjustments made below a magnitude of 1 would simply appear to cause no change.

In retrospect, while the use of CMA-ES allowed for fast improvements in simpler problems and fast convergence to some local optima in problems with a more continuous fitness function, the non-intuitive relationship between the numerical chromosome and the simulated objects within the problem space made the use of CMA-ES less suited as the experiments grew in complexity. In some cases the chromosome design itself could have been improved, but due to the aforementioned problems with multiple types of parameter needing to conform to a vector of real numbers, there may be some upper limit on the usefulness of treating all problems as directly connected to a real number vector.

Many difficulties were encountered as discussed in the previous sections, and the typical reaction to these discoveries in earlier stages of the research were to try and alter the heuristics used by controllers, to alter properties of the environment, or to alter the fitness measures and goals to be met by the vehicles. However, as research progressed, it became clearer that a more reasonable approach was to improve the controllers while examining which of the environmental parameters may have been causing the issues, and to see what faults were within the controllers that produced degenerate behaviour when provided with handmade vehicle designs with reasonable capabilities.

There is a relationship between the capacity of the controller and the effectiveness of the design, and it is almost always that a more sophisticated controller will control or pilot a vehicle design with far better effectiveness than any other controller. The evolution of controller parameters along with vehicle design parameters does not appear to have any net benefit compared to an adequately 'smarter' controller. Conversely, some designs are simply so ineffectual that the controller used will never make any significant impact on the behaviour of the vehicle. Consider, for example, a vehicle with all thrusters perfectly aligned with its centre of mass. This vehicle will not be particularly manoeuvrable as it will never be able to turn and must counter-thrust in any situation it wishes to reverse its direction.

While the exact nature of this relationship between brain and body is subject to interpretation, it can be shown that reasonably effective vehicles and their controllers can be designed for each other using evolutionary methods to produce interesting behaviours and designs in an automated fashion without the need for direct human input. Characteristic designs and behaviours such as the aforementioned spinning ships and giant turret ships emerged as a direct result of evolutionary pressures and controller capacities, and modifications to the environment to reward or punish these behaviours reinforced or dissuaded them as expected.

The primary objective throughout the course of these experiments was to design game vehicles through evolutionary means and to examine the usability of these game vehicles, and while the results have not been as effective as expected, the reasons for the complexities and difficulties faced in different problem contexts have been unexpected and surprising at times. One of the implicit objectives based upon the motivation was to evolve ships as opponents or other procedurally generated NPCs in a game context. While there have not been very many examples of highly effective ships for more complicated tasks, interesting behaviour with a mixture of predictable and unpredictable elements was observed, thus meaning that these designs combined with their controllers may in fact be hypothetically useful to a two dimensional game similar to the problem scenarios considered.

Throughout the course of this research, the experimental methods examined have strayed into the purposes of encouraging and demonstrating very specific behaviours that are “intelligent”, with intrusive modifications being made to the environment and fitness evaluation of the agents under consideration.

Experimental methods focusing on learning for practical applications tend towards a minimal amount of domain knowledge with the expectation or desire for a learning system to acquire the necessary knowledge within the learning process. The way this learning is carried out is typically of less importance as the resiliency of the system produced for its target environment and domain, as well as the system’s capacity to produce good results.

The experimental methods focusing on designing for demonstrating “artificial intelligence” used within this thesis, tended towards including a lot more domain knowledge and specific adjustments to value functions to promote certain desired behaviours, with the way the learning was carried out being tweaked and adjusted constantly throughout experimental runs. While this may not have produced truly resilient, robust actors with designs suitable for a variety of applications, this does not rule the approaches used in this thesis out from consideration, but does suggest future work where fitness measures and value functions used are locked into place.

Whether or not the approaches and results shown within this thesis can be argued to be acting as computational intelligence may be a point of contention, but to dismiss them as based upon nothing more than indirect designs would be to undermine the results seen in later chapters of the work, as well as the initial, simpler designs seen for problems where the fitness functions were not so heavily adjusted. It is, however, entirely possible that the results may have been entirely different with different definitions of success in mind during the research.

7.4 Future Work

It is worth revisiting the base assumptions that went into the design of the vehicle chromosome for all vehicle experiments. It is not necessarily the best choice for the task at hand, and while allowing a large amount of freedom and a wide space of vehicles, it has also proved to make already difficult problems much harder by providing a wide amount of solution space that does not lead to any progress.

The sheer scope of the design space allowed by the choice of chromosomal representation led to undirected and fruitless evolutionary runs. While this was not true in every case, it was a reoccurring trend in the more advanced problem contexts. Future work would do well to avoid such issues by placing restrictions on the designs allowed, thus requiring fewer dimensions to define a vehicle design and simplifying a relatively open-ended problem considerably. One such architecture would be to have predefined designs with spaces or slots for components to be placed, thus making the only variables the type of component assigned to a predefined slot, and perhaps its orientation relative to the vehicle.

Returning to the concept of evolving a controller strategy along with the vehicle, it might also be interesting to evolve controller-based structures in the limited slots available for the vehicle, such as distance sensors filtered on specific types of game entity, as opposed to abstract weightings for a specific controller architecture to make use of.

It is also worth revisiting the original spaceship combat scenario, but with a fixed environment, and a simpler scenario, perhaps with teams of fewer vehicles and with the smarter control techniques established for the predator-prey scenario. Control algorithms making use of planning techniques, no matter how simple, seemed to be required for the level of complexity exhibited in the scenario as presented, and it would be worth investigating a context where both planning controllers and reactive controllers could demonstrate some success.

Game tokens such as pick-ups and mines could be reduced to make the scenario less complicated and finite resources tweaked or removed to make the scenario simpler, and the scenario could be tuned in this way to allow for a simpler or more complex situation in order to investigate the effect of this complexity on the nature of designs.

Coevolutionary design of vehicles did not seem to progress as desired and retesting the evolved vehicles against a fixed design did not show a clear improvement. The reason for this might have been due to a population that did not necessarily objectively improve, but instead evolved against the available opponents, with some predator ships scoring especially highly against other prey ships more so than perhaps better predator ships would have scored against all prey ships. The method of testing predator and prey ships used would promote ships that could be extremely effective against some ships even if not very effective against all ships. They would need to be especially effective against these specific prey ships, but the boost in score granted by being so effective could compensate for less effective performance in other contests.

The measure used during coevolution as perhaps to blame for the observed results. As previously mentioned, predators were tested against preys, and both ships were assigned the mean score of the outcome of their competitions, with losing ships taken out of the running. Victorious ships would be tested again and again, but a consequence of this would be that some predators would be tested less frequently than others with certain prey ships as their opponents. The consequences of this would be incredibly difficult to predict due to the stochastic nature of the evolutionary algorithm and the ordering of the population. This would lead to intransitively effective ship designs being presented as potentially the best ship design of the generation. While a flawed ship design would not last across generations, it would have a direct consequence for the immediate next generation and bias evolutionary progression as a result. A measure involving testing every single ship design against every other ship design would perhaps avoid the worst of the intransitively successful ship design propagation into future generations, but would also have taken a combinatorially longer time to complete experimental runs.

Future work could examine the potential presence of such intransitivites and whether certain paired matchups of predator and prey ships would lead to misleading scores by testing predators against preys in an asymmetrical, round robin style of tournament, with each predator competed against each prey. Taking the already existing evolved designs and using such testing would be an interesting means of determining if this was truly the case.

Another aspect of future work would be the usability of the generated ship designs by human players, or sampling the opinions of human players versus the evolved designs and their controllers as potential opponents. It would be useful to investigate the relationship between the effectiveness of a design in testing and how enjoyable an opponent the combined design and controller would be. After all, vehicles that are too effective may end up being less fun or interesting to play against, and designs that are not perfect opponents that exhibit strange behaviours with a mixture of predictable and unpredictable elements may be more interesting.

Bibliography

[1] D.H. Adrian and S.C.A. Luisa. An approach to level design using procedural content generation and difficulty curves. *Proceedings of IEEE Conference on Computational Intelligence and AI in Games (CIG)*, 2013.

[2] D. Ashlock and C. McGuinness. Landscape automata for search based procedural content generation. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[3] D. Ashlock and C. McGuinness. Automatic generation of fantasy role-playing modules. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.

[4] H. Baier and M.H.M. Winands. Monte-Carlo tree search and minimax hybrids. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[5] C. Ballinger and S. Louis. Finding robust strategies to defeat specific opponents using case-injected coevolution. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[6] C. Browne. UCT for PCG. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[7] C. Browne. Evolutionary game design: Automated game design comes of age. *ACM Special Interest Group on Genetic and Evolutionary Computation (SIGEVOlution)*, 6(2):3–16, 2014.

[8] G. Chaslot, E. Bakkes, I. Szita, and P. Spronck. Monte-Carlo Tree Search: a new framework for game AI. In *Proceedings of Artificial Intelligence in Digital Entertainment (AIIDE)*, pages 216–217, 2008.

[9] D. Cliff and G.F. Miller. Co-evolution of pursuit and evasion II: simulation methods and results. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 506–515. MIT Press, 1995.

[10] D. Cliff and G.F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Proceedings of the Third European Conference on Advances in Artificial Life*, 1995.

[11] J. Clune, B.E. Beckmann, C. Ofria, and R.T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation (CEC)*, pages 2764–2771, 2009.

[12] J. Clune and H. Lipson. Evolving 3D objects with a generative encoding inspired by developmental biology. *ACM Special Interest Group on Genetic and Evolutionary Computation (SIGEVOlution)*, 5(4):2–12, November 2011.

[13] J. Clune, K.O. Stanley, R.T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367, June 2011.

[14] M. Cook and S. Colton. Multi-faceted evolution of simple arcade games. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 289–296, Aug 2011.

[15] M. Cook and S. Colton. Ludus Ex Machina: building a 3D game designer that competes alongside humans. *Proceedings of the Fifth International Conference on Computational Creativity*, 2014.

[16] L. Ferreira and C. Toledo. A search-based approach for generating Angry Birds levels. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.

[17] D. Floreano and S. Nolfi. God save the Red Queen! competition in co-evolutionary robotics. In *Proceedings of the Second Annual Conference on Genetic Programming*, pages 398–406, 1997.

[18] M. Frade, F.F. de Vega, and C. Cotta. Aesthetic terrain programs database for creativity assessment. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 350–354. IEEE, 2012.

[19] T. Geijtenbeek, M. van de Panne, and A.F. van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6), 2013.

[20] S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. In *Proceedings of the First International Conference on Autonomous Agents*, pages 22–29, New York, NY, USA, 1997. ACM.

[21] J. Grey and J. Bryson. Procedural quests: A focus for agent interaction in role-playing-games. In *Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB) 2011 Symposium: AI & Games*, pages 3–10, 2011.

[22] N. Hansen. The CMA evolution strategy: A tutorial. `http://www.lri.fr/~hansen/cmatutorial.pdf`, 2005.

[23] N. Hansen. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192, pages 75–102. Springer Berlin Heidelberg, 2006.

[24] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 282–291. Springer Berlin Heidelberg, 2004.

[25] K. Hartsook, A. Zook, S. Das, and M.O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 297–304, 2011.

[26] E. Hastings, R. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 1, 2009.

[27] E. Hastings, R. Guha, and K. Stanley. Evolving content in the Galactic Arms Race video game. *Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 241–248, 2009.

[28] E. Hastings, R. Guha, and K.O. Stanley. Neat particles: Design, representation, and animation of particle system effects. In *IEEE Conference on Computational Intelligence and Games (CIG)*, 2007.

[29] E.J. Hastings and K.O. Stanley. Interactive genetic engineering of evolved video game content. In *Proceedings of the Foundations of Digital Games (FDG) Workshop on Procedural Content Generation*, 2010.

[30] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1:1–1:22, February 2013.

[31] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation (MIT)*, 15(1):1–28, 2007.

[32] M. Joachimczak and B. Wróbel. Co-evolution of morphology and control of soft-bodied multicellular animats. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation (GECCO)*, pages 561–568. ACM, 2012.

[33] A. Jordan, D. Scheftelowitsch, J. Lahni, J. Hartwecker, M. Kuchem, M. Walter-Huber, N. Vortmeier, T. Delbrügger, Ü. Guüler, I. Vatolkin, and M. Preuss. BeatTheBeat music-based procedural content generation in a mobile game. *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*, pages 320–327, 2012.

[34] M. Kerssemakers, J. Tuxen, J. Togelius, and G.N. Yannakakis. A procedural procedural level generator generator. In *IEEE Conference on Computational Intelligence and AI in Games (CIG)*, pages 335–341, Sept 2012.

[35] L. Kocsis and C. Szepesvri. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282–293, 2006.

[36] P. Krčah. Evolving virtual creatures revisited. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2007.

[37] P.L. Lanzi, D. Loiacono, and R. Stucchi. Evolving maps for match balancing in first person shooters. *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.

[38] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*, Cambridge, MA, 2008. MIT Press.

[39] J. Lehman and K. O. Stanley. Beyond open-endedness: Quantifying impressiveness. In *Proceedings of the Thirteenth International Conference on Artificial Life (ALIFE XIII.* MIT Press, 2012.

[40] A. Liapis, H.P. Martnez, J. Togelius, and G.N. Yannakakis. Adaptive game level creation through rank-based interactive evolution, 2013.

[41] A. Liapis, G. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 4:213–228, 2012.

[42] A. Liapis, G. N. Yannakakis, and J. Togelius. Neuroevolutionary constrained optimization for content creation. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 71–78, Aug 2011.

[43] A. Liapis, G.N. Yannakakis, and J. Togelius. Optimizing visual properties of game content through neuroevolution. In *Proceedings of Artificial Intelligence in Digital Entertainment (AIIDE)*. The AAAI Press, 2011.

[44] S.M. Lucas, S. Samothrakis, and D. Perez. Fast evolutionary adaptation for Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.

[45] T. Mahlmann, J. Togelius, and G. N. Yannakakis. Towards procedural strategy game generation: Evolving complementary unit types. In *Proceedings of the International Conference on Applications of Evolutionary Computation*, pages 93–102, 2011.

[46] J. Marks and V. Hom. Automatic design of balanced board games. In *Proceedings of Artificial Intelligence in Digital Entertainment (AIIDE)*, pages 25–30. The AAAI Press, 2007.

[47] C. McGuinness. Statistical analyses of representation choice in level generation. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 312–319, Sept 2012.

[48] C. McGuinness and D. Ashlock. Incorporating required structure into tiles. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 16–23, Aug 2011.

[49] O. Michel. An artificial life approach for the synthesis of autonomous agents. In *Artificial Evolution*, volume 1063, pages 220–231. Springer Berlin Heidelberg, 1996.

[50] M.J. Nelson and M. Mateas. Towards automated game design. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 626–637, 2007.

[51] P. Nijssen and M.H.M. Winands. Monte Carlo tree search for the hide-and-seek game Scotland Yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):282–294, Dec 2012.

[52] N. Nygren, J. Denzinger, B. Stephenson, and J. Aycock. User-preference-based automated level generation for platform games. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 55–62, Aug 2011.

[53] J.K. Olesen, G.N. Yannakakis, and J. Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *2008 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 87–94, Dec 2008.

[54] J. Olsen. Realtime procedural terrain generation - realtime synthesis of eroded fractal terrain for use in computer games, 2004.

[55] F. Pachet. Beyond the cybernetic jam fantasy: The continuator. *IEEE Computer Graphics and Applications*, 24(1):31–35, 2004.

[56] A. Pantaleev. In search of patterns: Disrupting RPG classes through procedural content generation. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, pages 4:1–4:5, 2012.

[57] C. Pedersen, J. Togelius, and G.N. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2(1):54–67, March 2010.

[58] L. Pena, S. Ossowski, J.-M. Pena, and S.M. Lucas. Learning and evolving combat game controllers. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 195–202, Sept 2012.

[59] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P.I Cowling, and S.M. Lucas. Solving the physical traveling salesman problem: Tree search and macro actions. *IEEE Transactions on Computational Intelligence in Games (CIG)*, 6(1):31–45, March 2014.

[60] D. Perez, P. Rohlfshagen, and S.M. Lucas. Monte Carlo tree search: Long-term versus short-term planning. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 219–226, Sept 2012.

[61] D. Perez, S. Samothrakis, and S.M. Lucas. Knowledge-based fast evolutionary mcts for general video game playing. *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games*, pages 68–75, 2014.

[62] D. Perez, J. Togelius, S. Samothrakis, P. Rohlfshagen, and S.M. Lucas. Automated map generation for the physical traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 18(5):708–720, Oct 2014.

[63] M.A. Potter and K.A.D. Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the International Conference on Evolutionary Computation/The Third Conference on Parallel Problem Solving from Nature*, pages 249–257, 1994.

[64] E. Powley, D. Whitehouse, and P. Cowling. Monte Carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In

Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG), Aug 2013.

[65] E.J. Powley, D. Whitehouse, and P.I. Cowling. Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 234–241. IEEE, 2012.

[66] C. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, 1999.

[67] C.W. Reynolds. Competition, coevolution and the game of tag. In *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 59–69. MIT Press, 1994.

[68] S. Roberts and S. Lucas. Evolving spaceship designs for optimal control and the emergence of interesting behaviour. *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*, pages 342–349, 2012.

[69] S. Roberts and S. Lucas. Measuring interestingness of continuous game problems. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013.

[70] S. Samothrakis, S. Lucas, T.P. Runarsson, and D. Robles. Coevolving game-playing agents: Measuring performance and intransitivities. *IEEE Transactions on Evolutionary Computation*, 17(2):213–226, April 2013.

[71] S. Samothrakis, S. Roberts, D. Perez, and S. Lucas. Rolling horizon methods for games with continuous states and actions. In *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2014.

[72] S. Samothrakis, D. Robles, and S. Lucas. Fast approximate max-n Monte Carlo tree search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 3:142–154, 2011.

[73] M. Sanselone, S. Sanchez, C. Sanza, D. Panzoli, and Y. Duthen. Constrained control of non-playing characters using Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.

[74] M. Shaker, M.H. Sarhan, O.A. Naameh, N. Shaker, and J. Togelius. Automatic generation and analysis of physics-based puzzle games. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013.

[75] N. Shaker, M. Nicolau, G.N. Yannakakis, J. Togelius, and M. O'Neill. Evolving levels for Super Mario Bros using grammatical evolution. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 304–311, Sept 2012.

[76] N. Shaker, G.N. Yannakakis, and J. Togelius. Feature analysis for modeling game content quality. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 126–133, Aug 2011.

[77] K. Sims. Evolving virtual creatures. In *Proceedings of SIGGRAPH 94, Annual Conference Series*, pages 15–22, 1994.

[78] R. Smelik, T. Tutenel, K.J. de Kraker, and R. Bidarra. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 2:1–2:8, 2010.

[79] A.M. Smith and M. Mateas. Variations Forever: flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 273–280, Aug 2010.

[80] W. Sombat, P. Rohlfshagen, and S.M. Lucas. Evaluating the enjoyability of the ghosts in Ms Pac-Man. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 379–387, Sept 2012.

[81] W.M. Spears, K.A. De Jong, T. Bck, T. Ba, D.B. Fogel, and H.D. Garis. An overview of evolutionary computation. In *Machine Learning: ECML-93*, volume 667, pages 442–459, 1993.

[82] K.O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, June 2007.

[83] K.O. Stanley, B.D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, Dec 2005.

[84] K.O. Stanley, D.B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, April 2009.

[85] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation (MIT)*, 10(2):99–127, June 2002.

[86] M.J.W. Tak, M. Lanctot, and M.H.M. Winands. Monte Carlo tree search variants for simultaneous move games. *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games*, pages 68–75, 2014.

[87] C.H. Tan, K.C. Tan, and A Tay. Dynamic game difficulty scaling using adaptive behavior-based AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4):289–301, Dec 2011.

[88] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *2008 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 111–118, Dec 2008.

[89] J. Togelius, G.N. Yannakakis, K.O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 3(3):172–186, Sept 2011.

[90] A Uriarte and S. Ontanon. PSMAGE: balanced map generation for StarCraft. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[91] J. Valls-Vargas, S. Ontanon, and J. Zhu. Towards story-based content generation: From plot-points to maps. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Aug 2013.

[92] R. van der Linden, R. Lopes, and R. Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 6(1):78–89, March 2014.

[93] S. Watson, W. Banzhaf, and A. Vardy. Automated design for playability in computer game agents. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.

[94] M. Wittkamp, L. Barone, P. Hingston, and L. While. Noise tolerance for real-time evolutionary learning of cooperative predator-prey strategies. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 25–32, Sept 2012.

[95] M. Xu, Z. Pan, H. Lu, Y. Ye, P. Lv, and A E. Rhalibi. Moving-target pursuit algorithm using improved tracking strategy. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2(1):27–39, March 2010.

[96] G.N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, July 2011.

[97] G.N. Yannakakis and J. Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games (CIG)*, 2014.