Optimising Agent Behaviours and Game Parameters to Meet Designer's Objectives

W. Sombat

A thesis submitted for the degree ofDoctor of Philosophy

School of Computer Science and Electronic Engineering

University of Essex

Date of submission September 2016

I would like to dedicate this thesis to my loving family without whom I would have not survived the life of academic years. To my wife, Saowanee Sombat, who stood by for all those years supporting, caring, and understanding. To my older son, Pichai Sombat, whose achievements kept me proud and that I had not been lost and was doing the right thing. My younger son, Amornthep Sombat, who took me into pleasant journeys beyond imagining every night before bed after I came home wearied and depressed.

For my father, Suay Sombat, whose role model I took from. Whose motivation and concerns kept me in the path to this day.

To my colleagues and co-workers and my boss at Ubon Ratchathani University, thank you for supporting, understanding and for those extra work you had to cover for me during my academic years.

To my scholarship provider, Ministry of Science and Technology of Thailand, without the financial support I would not be able to raise my family here.

Acknowledgements

I cannot express enough sincere thanks to my supervisory board for their support and encouragement: Professor Massimo Poesio, supervisory board chair, Professor Richard Bartle, Professor Simon M. Lucas. I would like to give special thank to my supervisor, Professor Simon M. Lucas, for whose understanding, supporting, and insight knowledge helped keep me on track time after time.

My experiment could not have been accomplished without the support from my colleagues under the same supervision. Their tips and guidelines are priceless.

Abstract

The game industry is one of the biggest economic sector in the entertainment business whose product rely heavily on the quality of the interactivity to stay relevant. Non-Player Character (NPC) is the main mechanic used for this purpose and it has to be optimised for its designated behaviour. The development process iteratively circulates the results among game designers, game AI developers, and game testers. Automatic optimisation of NPCs to designer's objective will increase the speed of each iteration, and reduce the overall production time.

Previous attempts used entropy evaluation metrics which are difficult to translate the terms to the optimising game and a slight misinterpretation often leads to incorrect measurement. This thesis proposes an alternative method which evaluates generated game data with reference result from the testers. The thesis first presents a reliable way to extract information for NPCs classification called Relative Region Feature (RRF). RRF provides an excellent data compression method, a way to effectively classify, and a way to optimise objective-oriented adaptive NPCs. The formalised optimisation is also proved to work on classifying player skill with the reference hall-of-fame scores.

The demonstration are done on the on-line competition version of Ms PacMan. The generated games from participating entries provide challenging optimising problems for various evolutionary optimisers. The thesis developed modified version of CMA-ES and PSO to effectively tackle the problems. It also demonstrates the adaptivity of MCTS NPC which uses the evaluation method. This NPC performs reasonably well given adequate resources and no reference NPC is required.

Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Statement

In video games, controlling Non-Player Characters (NPCs) to deliver the required dynamics is essential to provide a satisfactory experience to the player. Big range of possible methods for implementing NPC AI, from hand-coding, finite state machine, through to a behavioural tree and neural networks. NPCs may vary greatly in how they adapt to the action of the player, and the intelligence it exhibits. Many of these have parameters that can be tuned. This thesis explores several ways of optimising the NPC AI. The optimiser ensures optimal settings for the NPCs to provide good user experience. The optimal settings drive the NPC to follow its designate mechanics. This thesis outlines the optimisation process and propose the methodology for generating, evaluating and optimising NPCs.

1.2 Motivation

Video games constitute major part of the entertainment industry and most popular video games rely on NPCs to entertain the players. Therefore, it is essential that the NPCs' behaviour matches the intention of the designer. However, the NPCs have to provide a good experience for as many players as possible. Player may have different skills and preferences. The game designers are responsible for balancing internal mechanics of the game. Game balancing is the fine-tuning phase in which a functioning game is adjusted to be deep, fair, and interesting [Jaffe et al., 2012]. Groups of researchers have contributed to the field, most notably the procedural content generation group [Togelius et al., 2011b]. Especially, on hamlet game engine where they control the flow experience by adjusting item properties.

Important questions of the field are (1) how to measure balance and (2) how reliable the measurement is. Important research in the field focuses on quantifying various aspects of the games. This includes work on quantifying properties of game levels [Liapis et al., 2013] and entertainment measurement [Yannakakis, 2005]. However, most measurement metrics are not guaranteed to work across all genre of games, and might also have differed result on particular group of players [Sombat et al., 2012b].

Along with a quantification system and an evaluation framework to generate systematic gameplay, optimisation is of equal importance. Once the evaluation system has been agreed upon and the feedback from the players have been received, optimisation should guarantee suitable gameplay.

1.3 Goals and Scope

The goal of this thesis is to study the process of generating NPCs to match a designated goal. This includes investigation of a suitable test-bed game, the measurement method, and the optimisation process. The thesis aims to provide a formalised methodology to optimise NPCs given a reference objectives.

The resulting NPCs should closely follow the intended high level behaviour of the reference objectives. Its result should be evident on the test-bed game; Ms Pac-Man, and the process should be thorough.

1.4 Structure of The Thesis

The rest of the thesis will guide the reader through the process of generating adaptive NPCs and classifying their behaviour.

Next chapter provides the necessary background on the subject. It describes the game development process with an emphasis on game AI. This should provide adequate knowledge for creating NPCs for game. The chapter continues on optimisation background from mathematical optimisation to evolutionary optimisation. After explaining the advantages of evolutionary optimisation, it will provide important optimisation techniques in the field. These optimisation techniques will be applied in subsequent chapters. The chapter should prove useful in understanding the modified versions proposed later on. These optimisation techniques includes genetic algorithm, evolutionary strategies, Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES), Particle Swarm Optimisation (PSO), and Monte Carlo Tree Search (MCTS).

Once the background has been established, the chapter moves on to analysing the evaluation metrics suggested by previous work. This includes the definition and formulae used to calculate the translated entropy for Ms PacMan game. An experiment is initiated to measure the performance of the evaluation metrics. Detail of the experiment is also reviewed along with the inconclusive result. The chapter, then, explains an alternative evaluation method for the prey-predator game. It proposes the relative region feature (RRF) extraction technique whose generated data is used in later chapters.

Chapter 4 describes how to create adaptive NPC by altering actions among selected agents. Given, the agents and the user preference ranking from the online competition, the chapter explains how an adaptive NPC could be generated. It also gives the formal procedure to create a user experience ranker when the number of preference ranking levels is lower than the number of participating agents. A ranker could be created from many configurations holding one reference player agent. A Thorough evaluation on the rankers is done to find an optimal ranker. The ranker is capable of ranking game data from unknown agents with high correlation to the preference ranks. An optimal configuration is selected to generate adaptive NPCs. The adaptability is demonstrated by the last experiment in the chapter.

Chapter 5 repeats the procedure established previously on the player hall-of-fame scores. This hall-of-fame list ranks players' score from highest to lowest which are then grouped into skill levels. Players' agents with higher scores are assumed to have higher skill. The created ranker also shows results highly correlated to the skill ranking levels.

Chapter 6 deals mostly with optimisation. The chapter presents two sets of

optimisation problem with known optimal solutions. These problems are finding optimal rankers for user experience ranking and finding optimal rankers for player skill ranking from the previous two chapters. They provide challenges to the optimisation techniques mentioned in the background chapter. The chapter proposes the modification algorithm and compares the performance results.In player skill level ranking problem, a modified version of PSO is developed and shown to outperform the others by a significant value.

Later in the chapter, we introduce a way to create adaptive NPC by utilising MCTS. This approach has an advantage over the agent-switching NPCs as it requires no agent in the implementation. MCTS is used to find appropriate response using the ranker as the evaluator. The experiment starts by analysing game data for the decision statistics to be used in optimising MCTS parameters. CMA-ES optimises MCTS parameters for real time constrained by the average decision time limit. The comparison shows better correlation value for game data generated by this MCTS-based adaptive NPC.

1.5 Contribution

The thesis provides a reliable way of optimising NPCs to fit user experience and player skill criteria. It proposes a game data extraction technique which can be used to create the user experience rankers and the player skill rankers. The optimal rankers can reliably rank game data from unknown agents. A Re-calibration calculation is proposed to improve the ranker's scoring system. The thesis, also, proposes modified version of CMA-ES and PSO for finding the optimal rankers. User experience rankers and player skill rankers are generated to evaluate an

adaptive NPCs; agent-blending NPCs. This NPC adapts by stochasticly select reference NPCs to response. For game starting out with a limit number of NPCs or no NPC, we also propose an adaptive MCTS NPC which performs equally well to the agent-blending NPC without the requirement.

Chapter 2

Background and Related Work

This chapter contains the necessary background material to understand the proposed system from game design to optimisation.

The first section reviews game design elements and discusses an attempt to quantify some of the cognitive terms while the second discusses the game development process. Introduction to optimisation is in the third section. The remaining sections detail advanced techniques used to accomplish the goal when the mathematical functions need to be approximated.

2.1 Game Design

Games consists of four main elements: mechanics, story, aesthetics, and technology [Schell, 2008]. Understanding each element in the game is important in order to create a successful game. These elements can be described as followed:

- **Mechanics** consists of the rules, the procedures, and the goals.

- **Story** defines the sequence of events in the games along with the message and information giving to the player.
- **Aesthetics** defines the looks and feels of the game including the sound and music.
- **Technology** are the necessary tools that the player need to play game, e.g., input devices, display devices, or hand-held devices.

An attempt to standardise the tool used to analyse video games is called Mechanics-Dynamics-Aesthetics framework (MDA) [Hunicke et al., 2004]. The framework formalises the terms as follows:

- Aesthetics is the appeal of the game, including but not limited to the following taxonomy
  - Sensation - game as sense-pleasure
  - Fantasy - game as make-believe
  - Narrative - game as drama
  - Challenge - game as obstacle course
  - Fellowship - game as social framework
  - Discovery - game as uncharted territory
  - Expression - game as self-discovery
  - Submission - game as pastime
- Dynamics work to create aesthetic experiences for example, challenge is created through time pressure and opponent play.

- Mechanics are the various actions, behaviour and control mechanisms in the game.

The framework works nicely as a bridge to the gap between game design and development, game criticism, and technical game research. For large game project, the game development flows smoothly among the teams. However, there is still a gap to be bridged between the designer's objectives and game AI development team. The framework has no detail specification on how designers' objectives could be achieved at the implementation level.

One view of game design from Koster and Wright [2004], is that games are made out of smaller games. The smallest level of a game is called a game atom. These game atoms consist of input, model, feedback and mastery to characterise the following:

- **Input** - a player does something.
- **Model** - the opponent or NPCs calculates a response.
- **Feedback** - the player get feedback.
- **Mastery** - the player learns from this feedback, and gets to do something again.

In this scenario, the opponent or NPCs constitutes most of the aesthetic of the game. Therefore, well-designed games with clear objectives for NPCs have better chance of success. Controlling or optimising NPCs behaviour to meet the designer's objectives is just as important.

2.2 Game Development

A video game is a software product therefore its development is also a software development [Bethke, 2003]. Just as any software development, game developers iteratively improve their product on each production cycle. A production cycle consists of four phases; pre-production, production, testing, and wrap-up [Chandler, 2009]. Pre-production is the planing and designing phase where at least the game concept and the development plan must be realised. Production is where the coding and asset building begins. The time frame between these two phases may be overlapped. Some tasks in the production phase can start parallel to the pre-production phase. The testing phase is a critical phase in game development [Chandler, 2009]. It includes plan validation and code release. Post-production is when the product is actually completed and the teams need to take notes for future project.

In some cases, the production cycle resolves to; concept, pre-production, production, and post-production where post-production includes testing and releasing game. In either case, the connections are clear between the designing team, the production team, and the testing team.

The focus of this thesis is on the AI developers team who are directly responsible for creating the designed NPC AI. The work is related to Procedural Content Generation (PCG) because its definition is given as the algorithmical creation of game content with limited or indirect user input [Togelius et al., 2011a]. Examples of PCG are software tools that create game maps, systems that create new weapons, programs that generate balanced board games, game engines that can populate a game world, and map editors [Togelius et al., 2015].

The next section gives an overview of the field of optimisation which will be used to automate the NPC generation process. The content covers mathematical optimisation techniques as well as evolutionary approaches.

2.3 Optimisation

Optimisation is the process of finding the best solution from all feasible solutions [Boyd and Vandenberghe, 2004]. Optimisation can be classified as either convex optimisation or non-convex optimisation (e.g., non-linear optimisation).

2.3.1 Convex Optimisation

Convex optimisation guarantees to solve the problem reliably and efficiently given the problem is well-formula and conformed to convex properties. Convex optimisation problem can be generalised as:

$$\text{minimize } f_0(x)$$

$$\text{subject to } f_i(x) \leq b_i, i = 1, \cdots, m$$

Where each function f_i must be convex (e.g., has the following property).

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

$$\text{if } \alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$$

It is worth noting that both least-squares problems and linear programs are special cases of convex optimisation problem.

2.3.2 Non-convex Optimisation

Traditional techniques for general non-convex problems usually involve problem decomposition and solving the convex sub-problems [Boyd and Vandenberghe, 2004]. The common techniques are:

- **Local optimisation methods** which uses non-linear programming techniques to approach the task.
  - find a point that minimises f_0 among feasible points near it.
  - fast, can handle large problems
  - require initial guess
  - provide no information about distance to (global) optimum.
- **Global optimisation methods** with the following characteristics.
  - find the (global) solution
  - problems might not be well-defined or too complex to be modelled [Weise, 2008].
  - worst-case complexity grows exponentially with problem size
  - advance techniques from many fields: machine learning, reinforcement learning, evolutionary optimisation, preference learning results in the field called meta-heuristic optimisation [Luke, 2013]

2.4 Evolutionary Optimisation

Evolutionary algorithms are an umbrella term used to describe computer-based problem solving systems which use computational models of some known mech-

anisms of evolution as key elements in their design and implementation [Spears et al., 1993]. These algorithms share a common conceptual base of simulating the evolution of individual structures via process of selection, mutation and reproduction. The processes depend on the perceived performance of the individual structures as defined by an environment.

In other words, evolutionary algorithms maintain a population of structures, that evolve according to rules of selection and other operators, that are referred to as "genetic operators", such as recombination and mutation. Each individual in the population receives a measure of its fitness in the environment. Recombination operation signifies exploration whereas mutation signifies exploitation.

Evolutionary algorithms consists of the following three main categories [Back et al., 1996].

- Genetic Algorithms - commonly used to solve optimisation problems by searching feasible solution space using string of numbers with common operators such as recombination and mutation.

- Evolution Strategies - optimisation techniques that searches for solution using real-value vector by iteratively evolving a population from an initial pool of candidates. The techniques use natural problem-dependent representations where the primary search operations are mutation and selection. Mutation is normally done by adding a random value to each vector component. Individual step sizes are either governed by self-adaptation or by covariance matrix adaptation (CMA-ES [Hansen and Ostermeier, 1996]).

- Genetic Programming - stochastically transforms populations of programs into new population to perform a user-defined task [Poli et al., 2008].

2.5 Preference Learning

Preference learning is about inducing predictive preference models from empirical data using utility functions and preference relations [Fürnkranz and Hüllermeier, 2010]. From a machine learning point of view, these two approaches pose two learning problems: learning utility functions and learning preference relations. Learning preference relations deviates from conventional problems like classification and regression, as it involves the prediction of complex structures, such as rankings or partial order relations, rather than single values. Moreover, training input in preference learning will not be offered in the form of complete examples but may comprise more general types of information, such as relative preferences or different kinds of indirect feedback and implicit preference information [FÜRNKRANZ and HÜLLERMEIER, 2003]. Preference learning has three types of rankings problems as shown in Figure 2.1, 2.2, and 2.3.

Figure 2.1: Label Ranking

Given:

1. a set of training instances $\{x_l | l = 1, 2, \ldots, n\} \in \mathcal{X}$
2. a set of labels $\mathcal{Y} = \{y_i | i = 1, 2, \ldots, k\}$
3. for each training instance x_l: a set of pairwise preferences of the form $y_i \succ_{x_l} y_j$
4. for each training example e_k:

Find:

- a ranking function that maps any $x \in \mathcal{X}$ to a ranking $\succ_x$ of $\mathcal{Y}$ (permutation $\pi_x \in S_k$)

Figure 2.2: Instance Ranking

Given:

1. a set of training instances $\{x_l | l = 1, 2, \ldots, n\} \in \mathcal{X}$
2. a set of labels $\mathcal{Y} = \{y_i | i = 1, 2, \ldots, k\}$
3. for each training instance x_l and associated label y_l

Find:

- a ranking function that allows one to order a new set of instances $\{x_j\}_{j=1}^{t}$ according to their (unknown) preference degrees.

Figure 2.3: Object Ranking

Given:

1. a set of training instances $\{x_l | l = 1, 2, \ldots, n\} \in \mathcal{X}$
2. a set of labels $\mathcal{Y} = \{y_i | i = 1, 2, \ldots, k\}$
3. for each training instance x_l: a set of pairwise preferences of the form $y_i \succ_{x_l} y_j$
4. for each training example e_k:

Find:

- a ranking function that maps any $x \in \mathcal{X}$ to a ranking $\succ_x$ of $\mathcal{Y}$ (permutation $\pi_x \in S_k$)

2.5.1 Related Research on Preference Learning

In recent research, pair-wise preference learning is used to rank the preferred ghost teams from the Ms. Pac-Man competition [Sombat et al., 2012a] using on-line evaluation from real players. This paper used preference learning in conjunction with classification tools to verify that the reliability in specifying the ghost team from game replays. Classification and preference learning can also be used in

place of heuristic evaluation with learning algorithm. This technique usually outperforms the latter in the case where the learning agent tries to imitate human player from game replays where high similarity of actions selected might not be optimal strategy [Wistuba et al., 2012]. Preference learning is also used to predict move in Othello game [Lucas and Runarsson] when combining with board inversion provides the best result beating many other methods.

2.6 MCTS

In 1940s Fermi, Ulam, von Neumann, Metroplois and others began to use random numbers to solve different problems in physics from a stochastic perspective [Landau and Binder, 2005]. Since then Monte Carlo methods have been applied widely even though much of the work were unpublished. Researchers now acknowledge that MCTS originated in statistical physics where they have been used to obtain approximations to intractable integrals. They have since been used in a wide array of domains including games research. Monte Carlo approaches in which the actions of a given state are uniformly sampled are described as flat Monte Carlo which is used to achieve world champion level play in Bridge and Scrabble [Ginsberg, 2001; Sheppard, 2002].

MCTS is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results. This has great impact on computational intelligence especially in games where states can be represented as trees of decisions [Cameron Browne, 2012] .

MCTS assumes that the true value of an action may be approximated using random simulation; and the values may be used efficiently to adjust the policy

towards a best-first strategy. The algorithm progressively builds a partial game tree, guided by the results of previous exploration of the leaf nodes. The tree will assemble that of the actual game tree and presumably more accurate as the tree is built.

2.6.1 General MCTS Algorithm

The basic algorithm involves iteratively building a search tree until some predefined computational budget typically a time, memory or iteration constraint is reached, at which point the search is halted and the best-performing root action returned. Each node in the search tree represents a state of the domain, and directed links to child nodes represent actions leading to subsequent states. The iteration process of the algorithm is presented in 2.4.

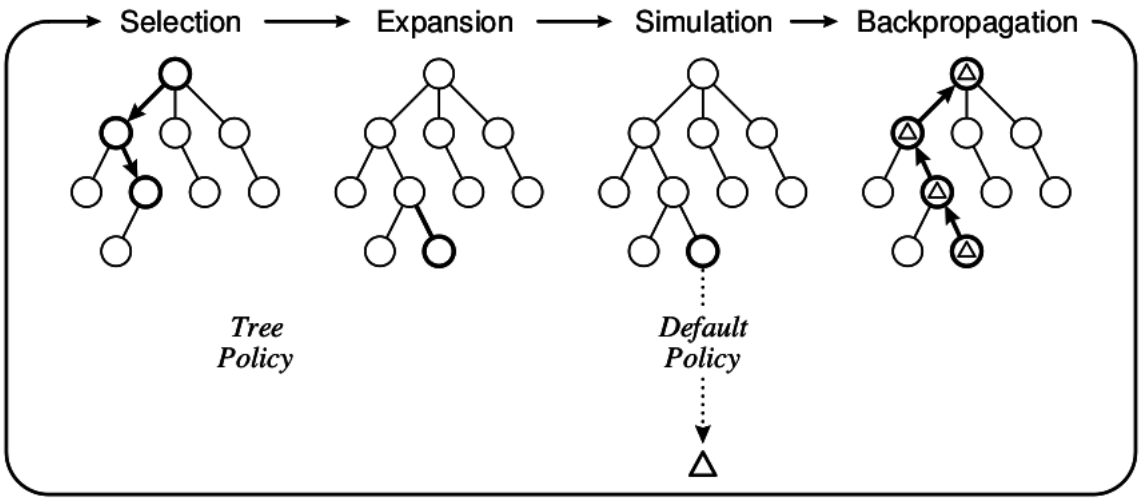

Figure 2.4: MCTS Iteration Process [Chaslot et al., 2008]

Four steps are applied per search iteration:

1. Selection: Starting at the root node, a child selection policy is recursively

applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a non-terminal state and has unvisited (i.e. unexpanded) children.

2. Expansion: One (or more) child nodes are added to expand the tree, according to the available actions.

3. Simulation: A simulation is run from the new node(s) according to the default policy to produce an outcome.

4. Back-propagation: The simulation result is backed up (i.e. back-propagated) through the selected nodes to update their statistics.

These may be grouped into two distinct policies:

1. Tree Policy: Select or create a leaf node from the nodes already contained within the search tree (selection and expansion).

2. Default Policy: Play out the domain from a given non-terminal state to produce a value estimate (simulation).

The back-propagation step does not use a policy itself, but updates node statistics that inform future tree policy decisions as illustrate in Algorithm 1.

create root node v_0 with state s_0;
while *within computational budget* **do**
  $v_l \leftarrow TreePolicy(v_0)$;
  $\Delta \leftarrow DefaultPolicy(s(v_l))$;
  $Backup(v_l, \Delta)$;
end
return $\alpha(BestChild(v_0, 0))$

Algorithm 1: General MCTS Approach [Cameron Browne, 2012]

Input: v
choose ain untried actions from $A(s(v))$;
add a new child v' to v;
with $s(v') = f(s(v), a)$;
and $a(v') = a$;
return v'

Algorithm 2: EXPAND node expansion procedure [Cameron Browne, 2012]

Input: v, c
Output: child with best UCT value
return $\underset{v' \in children(v)}{\text{argmax}} \frac{Q(v_i)}{N(v')} + c\sqrt{\frac{2lnN(v)}{N(v')}}$;

Algorithm 3: BESTCHILD finds best child using UCT [Cameron Browne, 2012]

Input: v, Δ
while v *is not null* **do**
  $N(v) \leftarrow N(v) + 1$;
  $Q(v) \leftarrow Q(v) + \Delta(v, p)$;
  $v \leftarrow$ parent of v;
end

Algorithm 4: BACKUP rollouts result back propagation [Cameron Browne, 2012]

2.6.2 Upper Confidence Bounds for Tree (UCT)

Kocsis and Szepesvári proposed the use of UCB1 as tree policy which value a child node with the expected reward approximated by the Monte Carlo simulations [de Mesmay et al., 2009]. Every time a node is to be selected within the existing tree, the choice may be modelled as an independent multi-armed bandit problem. A child node j is selected to maximise:

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2ln(n)}{n_j}}$$

where n is the number of times the current (parent) node has been visited, n_j the number of times child j has been visited and $C_p > 0$ is a constant. If more than one child node has the same maximal value, the tie is usually broken randomly. The values of $X_{i,t}$ and thus of X_j are understood to be within $[0, 1]$.

Input: s_0
create root node v_0 with state s_0;
while *within computational budget* **do**
  $v_l \leftarrow TreePolicy(v_0)$;
  $\Delta \leftarrow DefaultPolicy(s(v_l))$;
  $Backup(v_l, \Delta)$;
end
return $\alpha(BestChild(v_0, 0))$

Algorithm 5: UCT [Cameron Browne, 2012]

2.6.3 MCTS for Ms Pac-Man

Ms Pac-Man has enormous game tree due the size of nodes in the mazes and the possibility that the path could repeat itself even with limit number of children a node can have. Monte Carlo sampling approaches have been proposed to tackle

Input: v_0
$v \leftarrow v_0$;
while v *is nonterminal* **do**
  if v *not fully expanded* **then return** *Expand*(v) ;
  else $v \leftarrow BestChild(v, C_p)$;
end
return v

Algorithm 6: Tree Policy Function [Cameron Browne, 2012]

Input: s
while s *is non-terminal* **do**
  choose $a \in A(s)$ uniformly at random;
  $s \leftarrow f(s, a)$;
end
return *reward for state* s

Algorithm 7: Default Policy Function [Cameron Browne, 2012]

Input: v, Δ
while v *is not null* **do**
  $N(v) \leftarrow N(v) + 1$;
  $Q(v) \leftarrow Q(v) + \Delta(v, p)$;
  $v \leftarrow$ parent of v;
  $\Delta \leftarrow -\Delta$;
end

Algorithm 8: BackupNegamax [Cameron Browne, 2012]

this including finding optimal routes in real-time [Pepels and Winands, 2012]. Robles and Lucas [2009] used a route-tree based on possible moves that Ms Pac-Man can take. Flat Monte Carlo approach for the endgame strategy is also used to improved the agent's score by 20% with some basic assumptions regarding the character's movements [Bruce Kwong-Bun Tong, 2011]. Samothrakis et al. [2011] used MCTS with a 5-player max-n game tree, in which each ghost is treated as an individual player. Other applications of MCTS on Ms Pac-Man are avoiding trapped moves, move planning [Nguyen and Thawonmas, 2011], and in combination with heuristics learned from game-play to create better agent.

2.7 CMA-ES

CMA-ES stands for Covariance Matrix Adaptation Evolution Strategy. Evolution strategies (ES) are stochastic, derivative-free methods for numerical optimization of non-linear or non-convex continuous optimization problems. This uses an adaptation scheme for adapting arbitrary normal mutation distributions [Hansen and Ostermeier, 1996].

2.7.1 Principles

2.7.1.1 Maximum-likelihood

This principle is based on the idea to increase the probability of successful candidate solutions and search steps. The mean of the distribution is updated such that the likelihood of previously successful candidate solutions is maximized. The covariance matrix of the distribution is updated (incrementally) such that

the likelihood of previously successful search steps is increased [Hansen et al., 1995] Both updates can be interpreted as a natural gradient descent. Also, in consequence, CMA-ES conducts an iterated principal components analysis of successful search steps while retaining all principal axes. Estimation of distribution algorithms and the cross-entropy method are based on very similar ideas, but estimate (non-incrementally) the covariance matrix by maximizing the likelihood of successful solution points instead of successful search steps.

2.7.1.2 Search/Evolution Path

Two paths of the time evolution of the distribution mean of the strategy are recorded, called search or evolution paths. These paths contain significant information about the correlation between consecutive steps. Specifically, if consecutive steps are taken in a similar direction, the evolution paths become lone. The evolution paths are exploited in two ways. One path is used for the covariance matrix adaptation procedure in place of single successful search steps and facilitates a possibly much faster variance increase of favorable directions. The other path is used to conduct an additional step-size control. This step-size control aims to make consecutive movements of the distribution mean orthogonal in expectation. The step-size control effectively prevents premature convergence yet allowing fast convergence to an optimum.

2.7.2 Algorithm

In the following the most commonly used $(\mu/\mu_w, \lambda)$-CMA-ES is outlined, where in each iteration step a weighted combination of the μ best out of λ new candidate solutions is used to update the distribution parameters. The main loop consists

of three main parts:

- sampling of new solutions
- re-ordering of the sampled solutions based on their fitness
- update of the internal state variables based on the re-ordered samples

2.7.2.1 pseudo-code

set λ;

initialize $m, \sigma, C = I, p_\sigma = 0, p_c = 0$;

while *not terminate* **do**

  for $i \leftarrow 1$ ***to*** λ **do**

    $x_i = \text{sample\_multivariate\_normal}(m, \text{covariance\_matrix}{=}\sigma^2 C)$;

    $f_i = \text{fitness}(x_i)$;

  end

  $x_{1..\lambda} \leftarrow x_{s(1)..s(\lambda)}$ with $s(i) = \text{argsort}(f_{1..\lambda}, i)$;

  $m' = m$;

  $m \leftarrow \text{update\_m}(x_1, .., x_\lambda)$;

  $p_\sigma \leftarrow \text{update\_ps}(p_\sigma, \sigma^{-1} C^{-1/2}(m - m'))$;

  $p_c \leftarrow \text{update\_pc}(p_c, \sigma^{-1}(m - m'), ||p_\sigma||)$;

  $C \leftarrow \text{update\_C}(C, p_c, (x_1 - m')/\sigma, .., (x_\lambda - m')/\sigma)$;

  $\sigma \leftarrow \text{update\_sigma}(\sigma, ||p_\sigma||)$;

end

return m *or* x_1

Algorithm 9: CMA-ES Algorithm [Hansen, 2011]

2.8 PSO

PSO stands for particle swarm optimization. It is an evolutionary optimisation developed by Kennedy and Eberhart [1995] in 1995. It was inspired by the social behaviour of bird flocking and fish schooling [Eberhart and Kennedy, 1995]. It works by guiding a group of particles through problem space by manipulating their velocities. The velocity of each particle is stochastically adjusted with the influence of its best known position and the population best position. The term swarm comes from the irregular movements of the particles in the problem space, similar to a swarm of mosquitoes [Eberhart, 2001]. PSO has advantages over other optimisation techniques because it is not largely affected by the size and non-linearity of the problem [Del Valle et al., 2008]. In general, PSO has the following properties:

- Straightforward to implement.
- Few parameters to configure.
- Manages memory efficiently by keeping track of particle best position and population best position.
- More efficient in maintaining the diversity of the swarm as oppose to using selection for new population generation in which worst parent is most likely to be discarded. This property is especially valuable when optimizing problems that contain many local minima [Van den Bergh and Engelbrecht, 2006].

2.8.1 Standard PSO

Standard PSO was created to solve continuous problem space. Its algorithm is given in algorithm 10 and the update equation for the velocity and the position of a particle are given in equation 2.1.

```
Data: s, r_i, φ_p, φ_g, f(), MAXGEN
Result: best particle g
swarm = { g, p_i };
for i = 1, .., s do
    p_i.x = U(r_i);
    p_i.v = U(−r_i, r_i);
    p_i.b = p_i.x;
end
g = best p_i;
if g = global optimum then
    return g;
end
while g not optimal and not MAXGEN do
    updateParticle(p_i.v, p_i.x);
    evaluate(p_i.x);
    for i = 1, .., s do
        if f(p_i.x) > f(p_i.b) then
            p_i.b = p_i.x;
            if f(p_i.b) > f(g) then
                g = p_i.b
            end
        end
    end
    if g = global optimum then
        return g;
    end
end
```

Algorithm 10: Adapted from Standard PSO: SPSO

$$p.v_j \mathrel{+}= \phi_p u(0,1)(p.b_j - p.x_j) + \phi_g u(0,1)(g_j - p.x_j)$$
$$p.x_j \mathrel{+}= p.v_j \tag{2.1}$$

2.8.2 Discrete PSO

The first discrete version of the optimiser is the binary PSO proposed by Kennedy and Eberhart [1997]. The binary uses the same equation to update particle's velocity while relies on equation 2.2 to alter solution value between 0 and 1.

$$p.x_j = \begin{cases} 1 & \text{if } u(0,1) < \frac{1}{1+e^{-p.v_j}} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Laskari et al. [2002] has suggested rounding off the continuous optimum values to the nearest integer for solving discrete problem space. Pan et al. [2008] has suggested cross-over operation when particle's best position required updating. When the integer solution is assumed to be sampled from a single universe, the solution can be obtained using Set-Based PSO by Langeveld and Engelbrecht. This PSO version discretised velocity and used set operations in the original velocity equation.

The sigmoid function in equation 2.2 is used as a switch function in the binary version. It specifies whether or not to ignore the variable. The proposing PSO algorithm in later chapter uses this function to decide whether to move up or down the rank.

CMA-ES also requires modification to select only from valid integral candidates and the candidates only use the sign of the different vector when adjustments are required.

Chapter 3

Characterising NPC Behaviour

This chapter starts with the discussion of the related work either having common goals or using the same techniques. The followed section details an attempt to formulate player enjoyment metrics and discuss the challenges it proposed. The next section provides a methodology for gathering human player preference data through on-line questionnaires, followed by the analysis section. The last section describes the process of ghosts team ranking and classification in search for the corresponding features responsible for higher preference ranking by on-line players.

3.1 Ms Pac-Man

Ms Pac-Man is an arcade video game produced by Midway in 1981. The game is classified in the Maze genre as the original Pac-Man from Namco in 1980 [Lucas, 2007]. Pac-Man is a one-player game where the player controls the character to gather points by eating dots. The player moves the character around a maze to

clear the dots while avoiding the four ghosts. Player loses a life when contact with one of the ghosts. However, the ghosts turns edible for a brief period of time when the character eat a power dot. There are fewer power dots in a maze than the normal ones. Both power dot and edible ghost have higher scores than the normal dots.

Related differences between Ms Pac-Man and the original Pac-Man are:

- Gender of the character. Pac-Man represents male while Ms Pac-Man represents female.
- Number of mazes. Pac-Man has one maze while Ms Pac-Man consists of four mazes.
- Number of tunnels. The maze in the original Pac-Man game has only 1 tunnel. In Ms Pac-Man, one maze has 1 tunnel while the other three mazes has 2 tunnels.
- Number of dots. There are 240 dots and 4 power dots in the original Pac-Man. In Ms Pac-Man, the number of normal dots in the four mazes are 220, 240, 238, and 234, repectively when the number of power dots are the same as that of the original game.

The game consists of four mazes in total labelled A, B, C and D and cycle throughout the game with maximum number 16 mazes to clear. The player starts in maze A with three lives; an additional life is awarded at 10000 points. Each pill eaten scores 10 points, each power pill is worth 50 points. The NPCs are the four ghosts: Blinky (red), Pinky (pink), Inky (green) and Sue (brown). When a power pill is eaten the ghosts reverse the directions and turn them blue.

The score for eating each blue ghost in succession immediately after a power pill has been consumed starts at 200 points and doubles each time, for a total of 200+400+800+1600=3000 additional points.

3.2 Related Work

Controlling NPC behaviour has been the major aim of game AI research for a long time. Different techniques have been studied in variety of context. Some work focuses on controlling NPCs on a single game while the others aims for multiple games [Bjrnsson and Finnsson, 2009; Mhat and Cazenave, 2010]. The research can also be categorized into controlling NPCs in real time (on-line) or ahead of time (off-line). Common NPC controlling techniques are reinforcement learning [McPartland and Gallagher, 2011; Wang et al., 2010], neural networks [Parker and Bryant, 2012], evolutionary strategies [Recio et al., 2012], MCTS [Maes et al., 2012; Nguyen and Thawonmas, 2013; Samothrakis et al., 2011].

3.3 Game Entertainment Evaluation

On some classic games such as chess, checker and Othello, computer can plays human at any level with the exception of Go. But with the use of Monte Carlo Tree Search [Browne et al., 2012], computer opponents in Go are improving results. Strong AI components is not the only area of research for video games as highlighted by Laird and V. [2000]. And since then there has been significant research in the area including - designing AI for NPCs, game content creation [Shaker et al., 2010], and player entertainment/satisfaction. AI are now also used

to provide entertaining and engaging NPCs for the human players, since the game industry already has acceptable AI for NPCs for most purposes according to N. [2012]. On the other hand, Lucas et al. [2012] argues that there is great potential in making game AI better, and that when the bots are smarter new possibilities for interesting game play will naturally emerge.

One trend is to design game agents that are more interesting and fun to play against. The holy grail of this research is to have reliable quantitative measures of what makes a game fun. Each individual player has their own idea of what makes a game enjoyable, and different players are looking for different things. Theoretical approaches to define fun in computer games are based on the well-known theory of flow [Csikszentmihalyi, 1991] which results in a model for evaluating player enjoyment called GameFlow [P. Sweetser and P. Wyeth, 2005]. Quantitative approaches came later with an attempt to capture the entertainment value of a game. The works of Vorderer et al. [2003], Malone [1981] and N. et al. [2006] agree that the level of challenge significantly impacts player satisfaction, especially when the challenge of the task matches the player's abilities. Yannakakis [2005] developed some measures that attempted to quantify fun in prey-predator games such as Pac-Man. He developed an "interest function" consisting of three distinct factors: challenge, behavioural diversity and spatial diversity. Although the measures are a useful first step, it was not clear to us how well they would work in practice for our reasonably faithful implementation of Ms Pac-Man, since they were developed in the context of simpler examples and designed to apply to a general class of games. The measures therefore omit a great deal of game-specific information that can be used to better understand the player experience. The formula are listed below for reference.

3.3.1 Level of Challenge (C)

This concept is based on how long the ghosts take to capture the player: the longer the capturing time the easier the game, as expressed by Equation 3.1.

$$C = \left[1 - \left(\frac{E\{t_k\}}{\max\{t_k\}}\right)\right]^{p_1} \tag{3.1}$$

where t_k is the number of game ticks the ghosts take to capture Ms Pac-Man the k-th time. $E\{t_k\}$ is the expected number of game ticks for a player to lose a life, $\max\{t_k\}$ the maximum game ticks taken over N games and p_1 is a weighting parameter.

3.3.2 Level of Behaviour Diversity (B)

This measure is based on the idea that behavioural diversity can be measured by variations in the score obtained by a player over a series of games. Since the Level of Challenge is based on the number of game ticks, the level of Behaviour Diversity is defined using the standard deviation of the duration a player manages to survive:

$$B = \left(\frac{\sigma_{t_k}}{\sigma_{\max}}\right)^{p_2} \tag{3.2}$$

where

$$\sigma_{\max} = \frac{1}{2}\sqrt{\frac{N}{(N-1)}}(t_{\max} - t_{\min}) \tag{3.3}$$

and where σ_{t_k} is the standard deviation of t_k over N games, p_2 a weighting parameter and $t_{\min} \leq t_k$.

3.3.3 Level of Spatial Diversity (S)

Yannakakis used the following idea to define the concept of spatial diversity: to make the game more enjoyable, the ghosts must behave aggressively and exploratory to capture the player unexpectedly at times. The level of spatial diversity is formulated using number of nodes in the graph and number of visits to the nodes. Presumably, more exploratory ghosts cover all nodes more uniformly.

The level of spatial diversity is defined to be the average of the distribution value on different maze levels:

$$S = E\{H_n\} \tag{3.4}$$

where

$$H_n = \left[-\frac{1}{logVn} \sum \frac{v_{in}}{Vn} log\left(\frac{v_{in}}{Vn}\right) \right]^{p_3} \tag{3.5}$$

and where v_{in} is the number of visits to graph node i in maze n, $V_n = \sum_i v_{in}$ the total number of visits in maze n and v_{in} is the number of visits to cell i in maze n.

3.3.4 Interest Function

The overall Interest Function is then defined to be a weighted sum of the three individual measures outlined above:

$$I = \frac{\gamma C + \delta B + \varepsilon S}{\gamma + \delta + \varepsilon} \tag{3.6}$$

This measure may subsequently be used to assign a scalar value to a ghost team that indicates its perceived level of entertainment. Later in the chapter, we will test how it work in Ms PacMan.

3.4 The Ms Pac-Man vs Ghosts Competition

3.4.1 Ms Pac-Man

Ms Pac-Man is an arcade video game produced by Midway in 1981. The game is classified in the Maze genre as the original Pac-Man from Namco in 1980 Lucas [2007]. The test-bed implementation maintain compatibility to the original game. The player controls the agent to gather points by eating dots and avoiding ghosts. Player loses a life when contact with one of the ghosts. The ghosts turns edible for a while when player eat a power dot. Power dots and edible ghosts have higher scores than the normal dots.

The game consists of four mazes in total labelled A, B, C and D and cycle throughout the game with maximum number 16 mazes to clear. The player starts in maze A with three lives; an additional life is awarded at 10000 points. Each pill eaten scores 10 points, each power pill is worth 50 points. The non-player 4character (NPC) are the four ghosts: Blinky (red), Pinky (pink), Inky (green) and Sue (brown). When a power pill is eaten the ghosts reverse the directions and turn them blue. The score for eating each blue ghost in succession immediately after a power pill has been consumed starts at 200 points and doubles each time, for a total of 200+400+800+1600=3000 additional points.

The arcade game Ms Pac-Man is the most popular successor to the classic

Pac-Man, one of the most successful arcade games ever made. The player takes control of Ms Pac-Man using a 4-way joystick and needs to navigate her across a series of mazes. Ms Pac-Man scores points by eating the pills that are scattered around the maze but is chased by four ghosts at the same time. Whenever a ghost gets too close to Ms Pac-Man she loses a life. However, there are also four power pills in each maze which, when eaten, turn the ghosts edible for a short period of time, allowing Ms Pac-Man to chase and eat them instead. The first ghost eaten awards 200 points and this reward doubles with each ghost eaten in succession.

The game consists of four mazes which are played in order: whenever a maze is cleared (i.e., all pills have been eaten), the game moves on to the next maze until the game is over. Each maze contains a different layout with pills and power pills placed at specific locations. Each pill eaten scores 10 points, each power pill is worth 50 points. Ms Pac-Man starts the game with three lives; an additional life is awarded at 10,000 points. At the start of each level, the ghosts start in the lair in the middle of the maze and, after some idle time, enter the maze in their pursuit of Ms Pac-Man.

3.4.2 Ms Pac-Man vs Ghosts

The Ms Pac-Man vs Ghosts Competition is currently in its third iteration, having built on the success of the Ms Pac-Man Screen-Capture Competitions: competitors are asked to write controllers for either or both Ms Pac-Man and the ghosts and all entries compete with one another in a round-robin tournament to establish the best controllers. Ms Pac-Man controllers attempt to maximise the score of

the game while the ghosts strive to minimise the score. There are no restrictions regarding the techniques or algorithms used to create the logic for either side but controllers have only 40ms per game step to compute a move. Each game lasts a maximum of 16 levels and each level is limited to 3000 time steps to avoid infinite games that do not progress. Whenever the time limit of a level has been reached, the game moves on to the next level, awarding the points associated with the remaining pills to Ms Pac-Man; this is to encourage more aggressive behaviour of the ghosts, and avoids the ghosts spoiling a game by grouping together and circling a few remaining pills.

3.5 Classification of Ghost Teams

Each ghost team is designed and implemented with different strategies. Individual ghost in a ghost team follows a specific rule governed by the overall strategy. Each strategy orchestrates the ghosts differently and more sophisticated strategies are exhibited by the ghost teams with high scores. This section studies the movement of the ghosts and the overlapping decisions among the ghost teams.

3.5.1 Measuring Decision Overlap

We are interested to see how distinct the ghost teams are from each other so we designed an experiment to measure deviations in the action space. Each ghost team was asked to return actions for $2,000$ unique game states that were generated from games played by the starter controllers; only game states where three or four ghosts need to make a decision were considered (ghosts are not allowed to reverse so they only make decisions at junctions). The actions returned are integers

in the range $[0, 4]$ and any invalid directions are converted to "neutral" prior evaluation. The value 5 is used to signify if a ghost was not required to take an action. The response of each ghost team thus consists of a 4-digit string specifying the actions for Blinky, Inky, Pinky, and Sue sequentially. We can then calculate the percentages of overlapping actions between the different ghost teams in identical situations, ignoring actions from ghosts that are not required to take an action. This data is shown in Table 3.2 using equation 3.7 such that each entry in the table shows the percentage of similar actions made by ghost team i and ghost team j; the data is also visualised in Figure 3.1.

$$P_{ij} = 100 \times \frac{b_{ij} + i_{ij} + p_{ij} + s_{ij}}{B + I + P + S} \tag{3.7}$$

where b_{ij} (i_{ij}, p_{ij}, s_{ij}) is number of the identical actions made by Blinky (Inky, Pinky, Sue) for ghost teams i and j and B (I, P, S) is the total number of actions Blinky (Inky, Pinky, Sue) is required to take.

Table 3.1 shows the controller entries for the CIG11 competition.

Name	ID	PacMan ID	Ghosts ID	Vote Rank
NearestPillPacMan	20	20	-	-
Legacy	24	-	24	1
Legacy2TheReckoning	25	-	25	17
xsl11	27	27	27	9
PhantomMenace	28	28	28	14
brucetong	60	60	60	15
mcharles	64	64	64	7
GLaDOS	66	-	66	16
Ant_Bot	67	67	-	-
num01	71	-	71	13
Nostalgia	73	-	73	2
kveykva	74	-	74	11
Zekna_	76	76	-	
hacklash	78	78	78	8
jackhftang	79	-	79	6
Spooks	80	80	80	10
ICEgUCT_CIG11	81	-	81	5
ICEpAmbush_CIG11	82	82	-	
rcpinto	83	83	83	12
KaiserKyle	86	-	86	4
Scintillants	87	-	87	3
schrum2	88	88	-	
CERRLA	89	89	-	
emgallar	90	90	-	
Random	91	91	91	18
garner	92	92	-	
	26	16	18	

Table 3.1: Controller Entries for the CIG11 Competition

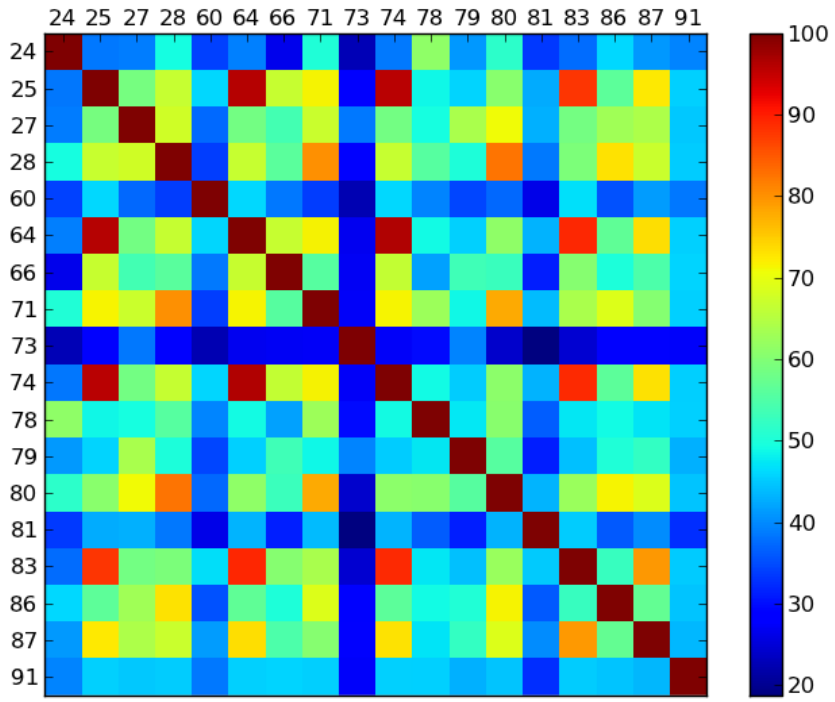

Figure 3.1: Confusion matrix of the percentages of similar decision made by the ghost teams.

	24	25	27	28	60	64	66	71	73	74	78	79	80	81	83	86	87	91
24		38	38	49	34	39	26	50	22	38	61	40	51	33	37	46	40	39
25	38		58	66	46	95	66	71	28	95	48	45	60	42	88	56	72	45
27	38	58		67	37	58	53	67	38	58	49	63	70	43	58	63	64	44
28	49	66	67		33	66	56	80	28	66	55	50	82	38	59	73	67	45
60	34	46	37	33		46	38	33	22	46	39	34	37	26	47	35	41	38
64	39	95	58	66	46		66	71	26	96	49	45	61	43	89	56	73	45
66	26	66	53	56	38	66		55	27	66	41	53	52	31	60	50	54	46
71	50	71	67	80	33	71	55		27	71	62	48	77	43	63	69	60	45
73	22	28	38	28	22	26	27	27		27	29	39	24	18	24	28	28	27
74	38	95	58	66	46	96	66	71	27		48	45	61	43	89	56	73	45
78	61	48	49	55	39	49	41	62	29	48		47	60	36	47	49	47	45
79	40	45	63	50	34	45	53	48	39	45	47		55	31	44	50	52	42
80	51	60	70	82	37	61	52	77	24	61	60	55		43	62	71	68	44
81	33	42	43	38	26	43	31	43	18	43	36	31	43		45	36	40	32
83	37	88	58	59	47	89	60	63	24	89	47	44	62	45		52	79	45
86	46	56	63	73	35	56	50	69	28	56	49	50	71	36	52		57	44
87	40	72	64	67	41	73	54	60	28	73	47	52	68	40	79	57		43
91	39	45	44	45	38	45	46	45	27	45	45	42	44	32	45	44	43	

Table 3.2: Confusion matrix shows number of time the ghost teams made the same decision in percentages.

3.5.2 Analysis of Ghost Decision

The $2,000$ game states used require a total of $6,009$ decisions to be made: Blinky is required take 1,806 decision, Inky is required to take 1,877 decisions, Pinky is required to take 440 decisions and Sue is required to take 1,886 decisions. The percentage of entries that makes the same decisions more than 50% of the time is 47% while the percentage of entries that make the same decisions more than

80% of the time is 10%. There are a few entries that shows high percentages of similarity all of which are rule-based entries with conditionally using the same rule to make decision at the implementation level.

3.5.3 Experimental Setup For Ranking and Classification

In this experiment, 18 ghost teams and 15 Ms Pac-Man controllers are pitted against one another and games are recorded. The process begins by selecting one ghost team and one Ms Pac-Man controller from the pool to play 20 matches. Each match is run normally until the game is over. During the match important game information is saved at each time step for replays and analysis:

- total time, level time, score, maze, level
- action, location and direction of Ms Pac-Man
- number of lives remaining
- statuses of all pills and power pills (eaten or not)
- location, direction, edible time, lair time of each ghost

Even though the size of game state is fixed, the size of a match may vary depending on how long the match takes. All 5,400 matches were played and recorded sequentially.

3.5.4 Ghost Teams Ranking with Interest Function

To obtain the interest value mentioned in Section 3.3 we ran the following procedure through all 300 matches: the game states are read and the duration Ms

Pac-Man survived is recorded by counting the number of game states passed to produce all t_k value from which the average and maximum is easily obtained (3.1). At this point we can also calculate equation 3.2 by finding the standard deviation of t_k get the maximum and minimum to feed to equation 3.3.

For the spatial diversity equation 3.4, because the we need to calculate number of visits to each cell (node), this needs to be calculated separately depending on which of the four mazes the game state is in. This can be done in one of the following two ways: (1) by evaluating match one by one and average the value if the match played on more than one maze and (2) by keeping tracks of all visiting counts for 4 mazes, then all 300 matches can be read, and calculate once all the reading is done. There is minor value differences between the two method. In this experiment we used the first approach since it can be done incrementally.

In final step, we calculate the interest value of the ghost team by calculating equation 3.6 with suggesting weight for parameters from the original author using: $p_1 = 0.5$, $p_2 = 1$, $p_3 = 4$, $\gamma = 1$, $\delta = 2$, $\varepsilon = 3$

The interest values for all ghost teams is presented in rank order of this measure of interest in table IV. This bears no relationship to the rank order of preferences expressed by human players in table 3.4, and actually ranks the Random team highest, which human players found least interesting to play against.

Name	100*C	100*B	100*S	100*I	Rank	Vote Rank
Random	93.00	32.65	31.46	42.11	1	18
jackhftang	93.17	22.63	32.85	39.50	2	6
GLaDOS	96.78	25.68	28.24	38.81	3	16
Spooks	95.49	22.59	28.53	37.71	4	10
xsl11	94.41	19.14	31.10	37.67	5	9
num01	94.56	27.85	25.20	37.64	6	13
Nostalgia	97.14	20.70	28.89	37.53	7	2
PhantomMenace	95.70	19.68	29.52	37.27	8	14
Legacy	97.10	19.46	29.03	37.19	9	1
ICEgUCT_CIG11	97.47	16.25	30.74	37.03	10	5
KaiserKyle	97.72	17.53	29.33	36.80	11	4
Scintillants	96.95	16.90	29.59	36.59	12	3
kveykva	95.70	27.00	22.28	36.09	13	11
Legacy2TheRec.	96.23	24.43	23.27	35.82	14	17
hacklash	97.97	17.24	26.93	35.54	15	8
brucetong	98.62	13.89	28.53	35.33	16	15
mcharles	98.79	14.06	28.15	35.22	17	7
rcpinto	96.12	25.22	20.43	34.64	18	12

Table 3.3: Results from the analysis of games using proposed measurement.

Rank	Name	Elo	+	-	games	score	oppo.	draws
1	Legacy	108	88	83	53	62%	4	0%
2	Nostalgia	76	86	82	55	60%	-9	0%
3	Scintillants	72	94	91	45	58%	4	0%
4	KaiserKylets	67	80	77	60	58%	-4	0%
5	ICEgUCT_CIG11	51	74	72	71	56%	-5	0%
6	jackhftangts	32	80	79	59	54%	0	0%
7	mcharles	27	84	83	53	53%	4	0%
8	hacklash	26	86	85	52	54%	1	0%
9	xsl11	21	79	78	61	52%	5	0%
10	Spooks	15	76	76	65	52%	1	0%
11	kveykva	-14	80	81	59	47%	6	0%
12	rcpinto	-46	83	86	52	44%	-4	0%
13	num01	-57	76	78	64	42%	6	0%
14	PhantomMenace	-58	80	82	58	43%	5	0%
15	brucetong	-60	85	88	52	42%	-2	0%
16	GLaDOS	-63	79	81	59	44%	-16	0%
17	Legacy2TheReckoning	-91	77	80	62	40%	-5	0%
18	RandomGhosts	-108	85	90	52	37%	1	0%

Table 3.4: Results of Bayes Elo Analysis From On-line User Preference Sombat et al. [2012b]

3.5.5 Relative Region Feature: RRF

Results in Section 3.5.2 show that ghost teams can be distinguished from each other by the decisions they make given a set of game states. However, measuring each decision offers a microscopic view of behaviour, and does not lead directly to any useful analysis of what might make a game fun. In pursuit of this goal,

we designed a feature space that should be able to classify game logs as belong to a particular ghost team, and also be useful in estimating fun.

The original Ms Pac-Man ghosts are fun to play against, and the rules controlling their behaviour ensure that they come at Pac-Man from different directions, and are sometimes close by and sometimes far away. Hence, we developed relative features that would account for the distances and directions of each individual ghost to the Pac-Man. This is depicted in Figure 3.2 which labels the regions relative to the position of the Pac-Man.

We further clarify this by plotting ghosts positions relative to the Pac-Man, and found that the density of the relative ghosts positions exhibits differences. This leads to region separations as to whether the ghosts likely to be in the left-right-up-down position to location of Ms Pac-Man. Figure 3.2 shows regions numbering where Ms Pac-Man is at the centre of the diagram on the left. The picture in the right hand side in Figure 3.2 is a game state of a match at game tick 530 with score 1,180 in level 1. Mapping the region for the ghost at that game state would result in Blinky at region number 2, Inky is in region number 3, Pinky is in region number 0, and Sue is in region number 6.

3.5.6 Ghost Team Classification

The first step of the classification is to turns all the matches into region data. This is done match by match. One match file turns into one region file. The converter programs will turn each game state in the match one-by-one to region data, with each game state mapping to a single region string. For example, the game state on the right of Figure 3.2 is turns to 4-digit region string '2306'. The

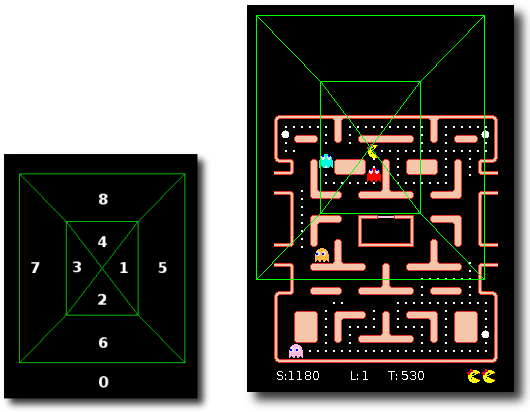

Figure 3.2: Region numbering (left) and overlay of regions relative to the position of Ms Pac-Man (right).

region string varies on the size of the regions chosen, but is always of length 4. In this experiment, three regions sizes are set-up. Small-size regions is the same size as the maze. Large-size regions covers twice the size of the maze to keep the ghosts in the range of region number 1 to 8. Medium-size regions is the middle size between the two size.

The region data of a match is essentially a text file where each line is a region string converted from the game state where the game tick is the same as the line number. The data is then organised for the classification by grouping them using the ghost teams identification irrespective of which Ms Pac-Man team it is playing. The files are organised into 18 directories corresponding to the ghost teams where each directory contains 300 region data files. These traces alter significantly with depending on ghost team behaviour.

The text classifiers includes the step of preprocessing and transforming which would help us discern the noisy data in developing custom classifier. For example the term frequency inverse document frequency will take care of our high fre-

quency value for region string *0000* and scale the dimensions of the feature vector for us [Joachims, 1998]. In this experiment we apply popular text classifier from scikit-learn [Pedregosa et al., 2011]. The selected classifiers are Ridge classifier (RidgeC), k-Nearest Neighbour Classifier (KNNC) [Dasarathy, 1991], Support Vector Machine classifier [Joachims, 1998] using LIBLINEAR [Fan et al., 2008] (SVMC), Stochastic Gradient Descent classifier (SGDC) Yin and Kushner [2003], and Bernoulli Naive Bayes classifier (BNBC) Rish [2001]. In addition to the five classifiers, we have created a custom classification pipeline. This pipeline consists of a count vectorizer for feature extraction, TF-IDF for vector transformer, and SVMC as the classifier (SVMC Pipeline).

All classifiers are trained with 3,510 region data files with 195 files from each ghost teams. The remaining 105 region data files for each ghost team are used for testing and validation. The classifiers scores are then evaluated with 1,890 region data files. Table 3.5 shows the F1-scores for all the classifiers, with the overall best result being the SVMC Pipeline using the small region features. The classifiers perform better with small region RRF dataset. Small region RRF dataset outperforms medium size by 2.43% and it outperforms large size dataset by 5.97%. The table also shows that the best classifier outperforms the second best by 3.13%.

Figure 3.3 shows the confusion matrices for the SVMC classifier based on small, medium and large regions respectively. The F1-score is the harmonic mean of precision and recall, see equation 3.8.

$$F_1 = 2 * \left(\frac{precision * recall}{precision + recall} \right) \tag{3.8}$$

$$precision = \frac{tp}{tp + fp} \tag{3.9}$$

$$recall = \frac{tp}{tp + fn} \tag{3.10}$$

where tp (true positive) is the number of matches the ghost team played and correctly classified, fp (false negative) the number of matches other ghost team played but incorrectly classified and fn (false negative) the number of matches other ghost team played and classified as not belong to the ghost team.

	Small	Medium	Large
RidgeC	0.74	0.71	0.68
KNNC (n= 5)	0.65	0.63	0.61
KNNC (n=10)	0.60	0.60	0.57
KNNC (n=20)	0.54	0.57	0.54
SVMC	0.74	0.72	0.69
SGDC	0.74	0.73	0.70
BNBC	0.57	0.53	0.53
SVMC Pipeline	0.78	0.74	0.72

Table 3.5: Classifiers F1 Scores.

3.6 Ghost Team Ranking With Classifier

As shown in previous section, reliable classifiers can be generated using region-base movement for Ms Pac-Man game. This section demonstrates that with appropriate Pac-Man agent new ghosts controller can be ranked and rated. Table

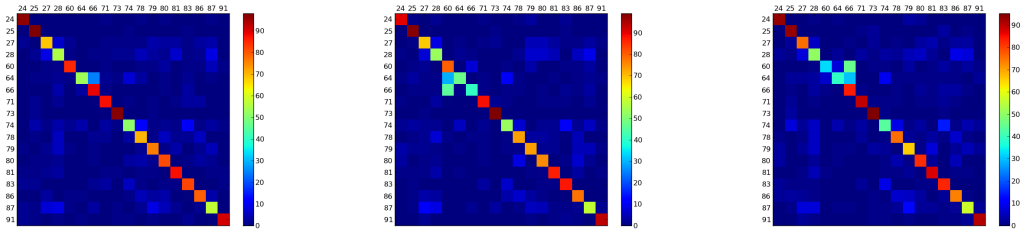

Figure 3.3: Confusion matrices for different region sizes (small, medium, and large; left to right) with SVMC Pipeline.

3.6 lists all entries for the experiment.

Name	ID	PacMan ID	Ghosts ID
NearestPillPacMan	20	20	-
Legacy	24	-	24
Legacy2TheReckoning	25	-	25
xsl11	27	27	27
PhantomMenace	28	28	28
brucetong	60	60	60
mcharles	64	64	64
GLaDOS	66	-	66
Ant_Bot	67	-	67
num01	71	-	71
Nostalgia	73	-	73
kveykva	74	-	74
Zekna_	76	-	76
hacklash	78	78	78
jackhftang	79	-	79
Spooks	80	80	80
ICEgUCT_CIG11	81	-	81
ICEpAmbush_CIG11	82	82	-
rcpinto	83	83	83
KaiserKyle	86	-	86
Scintillants	87	-	87
schrum2	88	88	-
CERRLA	89	89	-
emgallar	90	90	-
Random	91	91	91
garner	92	92	-
	26	17	18

Table 3.6: CIG11 Entries Used In The Experiment

In order to create reliable ghost ranking classifier, reliable pacman agents need to be identified. The next experiment is setup to find most reliable pacman entries to use as classification. is required as Some pacman entries especially the entries that

3.6.1 Classifiers Evaluation

Comparing modern text classifications: SVC, MultinomialNB, and SGD. Set up for evaluation:

- select based pacman controller, NearestPill pacman.
- Generate 400 games against each ghosts team total of 18*400 = 7,200 samples
- feature vectorizer - CountVectorizer
- classifiers - SVC, MultinomialNB, SGD
- classifier evaluation using StratifiedKFold - folds = 4

3.6.2 PacMan Selection

Because pacman entries implemented differently, some pacman should be more reliable than the others when used as evaluating pacman agent in the classifier. Rule-based entries should yield more reliable classifier than those with random decision making.

Match data is generated from round-robin tournament of all pacman entries versus all ghost entries. Each of the 17 pacman entries will have 18 ghosts team to

play against. There are 17*18 or 306 possible matches. Each match will generate 500 games and converted to region-based data where 400 of those are used as a training dataset and the remaining 100 games as the testing dataset.

ID	Name	SVC	SGD	MuiltinomailNB
20	NearestPill	0.69	0.68	0.68
27	xsl11	0.85	0.83	0.80
28	PhantomMenace	0.85	0.84	0.83
60	brucetong	0.69	0.65	0.61
64	mcharles	0.83	0.82	0.75
67	Ant_Bot	0.54	0.52	0.54
76	Zekna	0.78	0.72	0.90
78	hacklash	0.53	0.53	0.51
80	**Spooks**	**0.92**	**0.89**	**0.91**
82	ICEpAmbush_CIG11	0.57	0.55	0.54
83	rcpinto	0.87	0.86	0.85
88	schrum2	0.67	0.66	0.64
89	CERRLA	0.83	0.82	0.81
90	emgallar	0.63	0.58	0.62
91	RandomNonRev	0.36	0.35	0.35
92	garner	0.71	0.68	0.65

Table 3.7: classifiers performance based on pacman entries

The SVMC classifiers are generated from the training dataset corresponding to each pacman entries. Model evaluation are performed on Each classifier f1-score report and confusion matrix is inspected. The table 3.7 reports f1-score on all classifiers.

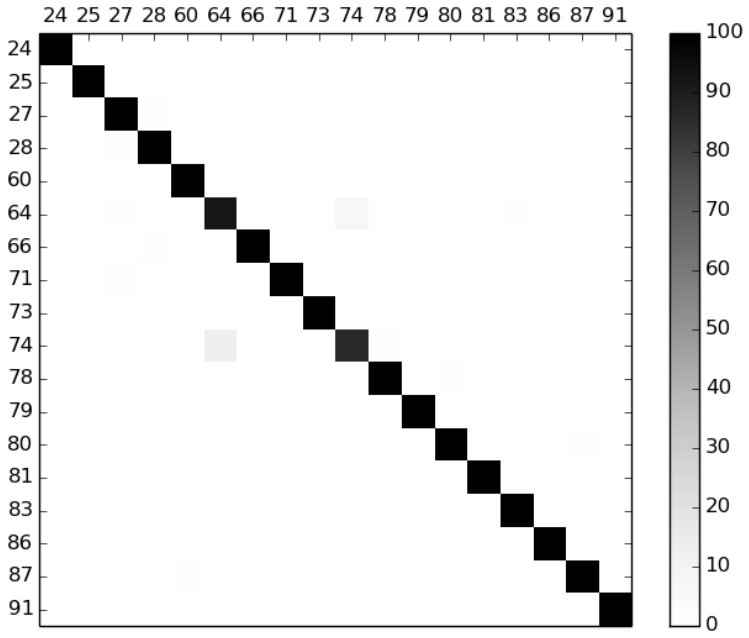

Figure 3.4: Confusion Matrix for Classifier built with Spooks pacman.

The highlighted classifier build from Spooks pacman has the highest f1-score as show in table 4.2 and with Figure 3.4. Figure 3.5 shows the confusion matrix for nearest pill pacman.

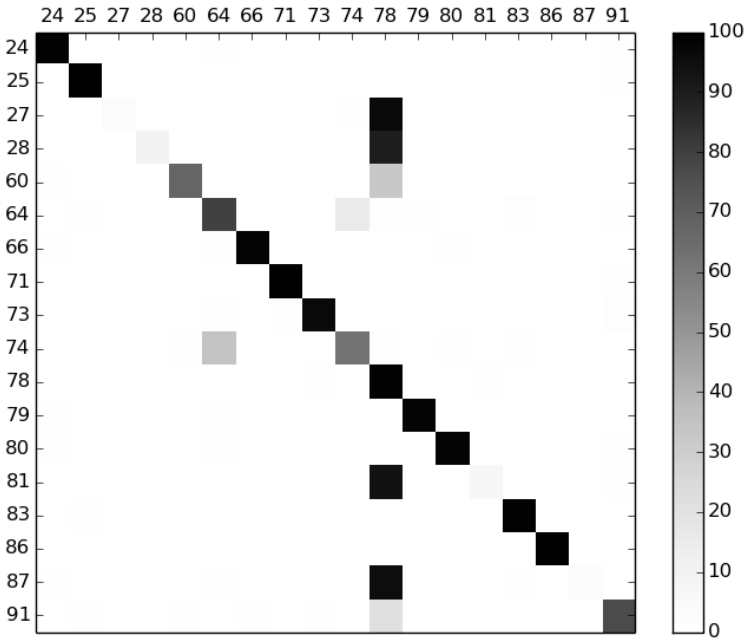

Figure 3.5: Confusion Matrix for Classifier built with NearestPill pacman.

3.6.3 Ghosts Team Evaluation

Overall classifiers performance based on ghosts entries.

ID	Name	SVC	SGD	MuiltinomailNB
24	**Legacy**	**0.85**	**0.85**	**0.80**
25	Legacy2TheReckoning	**0.86**	0.80	**0.84**
27	xsl11	0.73	0.72	0.63
28	PhantomMenace	0.61	0.58	0.56
60	brucetong	0.75	0.77	0.75
64	mcharles	0.32	0.35	0.32
66	GLaDOS	0.66	0.64	0.69
71	num01	0.84	0.81	0.79
73	Nostalgia	0.83	**0.82**	0.81
74	kveykva	0.27	0.07	0.30
78	hacklash	0.72	0.72	0.68
79	jackhftang	0.77	0.76	0.75
80	Spooks	0.81	0.76	0.76
81	ICEgUCT_CIG11	0.78	0.75	0.79
83	rcpinto	0.78	0.77	0.83
86	KaiserKyle	0.77	0.79	0.76
87	Scintillants	0.63	0.67	0.58
91	Random	0.74	0.74	0.71

Table 3.8: classifiers performance based on ghosts entries

3.7 Conclusions

Creating AI for game NPCs to match player preferences is possible given adequate implementation of NPCs implementation. In the experimental study that directly measures human preferences in the game of Ms Pac-Man using a set of ghost teams from a recent Ms Pac-Man versus Ghosts Competition. The competition

not only allowed us access to numerous distinct ghost teams but also gave us a good idea of the playing strengths of these teams. To make the most of the noisy preference data we used the Bayes Elo tool to optimally fit a Bradley-Terry model and found that some teams were significantly preferred to other teams. The Yannakakis model of interest [Yannakakis, 2005] was found to not produce useful estimates. However, we developed a relative region approach that is more directly applicable to the game of Pac-Man, and found that text classification algorithms were able to classify ghost teams with reasonable accuracy. The idea of using classification to evaluate automated game-play based on user preference data can be extended to other type of games. This study demonstrates how to extract movement traces from Ms Pac-Man which is equally applicable to any other predator-prey game where similar behaviours are prominent. This approach can also be used in platform games where movement traces such as 'jumping on' and 'jumping over' enemies and objects (e.g., Super Mario) can be used as an indication of the gamer enjoying the game. This approach may also be generalised to other types of games especially those where replays are widely available (as is often the case with real-time strategy games used in gaming competitions).

Chapter 4

Player Experience Levels

This chapter presents a systematic method for creating NPCs with ability to adapt to player experience levels in Ms PacMan game. The user experience levels uses on-line user preference data as reference resource.

The research uses the RRF 3.5.5 technique in search for a way to correctly rank player experience levels of NPCs. The methodology should be applicable to other criteria as well such as difficulty levels based on NPCs scores.

4.1 Experiment setting

The experiment uses 15 pacmans entries and 18 ghosts teams entries from the CIG11 pacman-vs-ghosts on-line contest. In addition to previous evaluation of fun evaluation of the ghosts team entries, this experiment adds additional a ruled-base pacman controller called, NearestPillPacMan. The pacman controller aims to collect as many pills as possible by selecting the shortest path to the closest pill. The total number of pacman controllers used is 16. There are 16×18 possible

matches and 200 unique games are generated for each match. The total of $57,600$ games are used in this experiment. Table 4.1 shows all of the entries.

Name	ID	PacMan ID	Ghosts ID	Vote Rank
NearestPillPacMan	20	20	-	-
Legacy	24	-	24	1
Legacy2TheReckoning	25	-	25	17
xsl11	27	27	27	9
PhantomMenace	28	28	28	14
brucetong	60	60	60	15
mcharles	64	64	64	7
GLaDOS	66	-	66	16
Ant_Bot	67	67	-	-
num01	71	-	71	13
Nostalgia	73	-	73	2
kveykva	74	-	74	11
Zekna_	76	76	-	
hacklash	78	78	78	8
jackhftang	79	-	79	6
Spooks	80	80	80	10
ICEgUCT_CIG11	81	-	81	5
ICEpAmbush_CIG11	82	82	-	
rcpinto	83	83	83	12
KaiserKyle	86	-	86	4
Scintillants	87	-	87	3
schrum2	88	88	-	
CERRLA	89	89	-	
emgallar	90	90	-	
Random	91	91	91	18
garner	92	92	-	
	26	16	18	

Table 4.1: Controller Entries for CIG11 Competition

4.2 PacMan entry selection

Sixteen classifiers is built based on 16 pacman entries. Each classifier is corresponding to a pacman entry. Each classifier is trained on 2,700 games taking from 150 games from each of the 18 ghosts teams entries. The remaining 900 games are testing games drawing 50 games from each ghosts team entry. Table 4.2 displays the performance of all the classifiers. Each row shows the precision, recall and the F_1 score on the corresponding classifier with the title pacman entry. The precision is the positive predictive value calculated with precision. The recall is the sensitivity or true positive rate calculated with recall. The F_1 score is the harmonic mean of precision and recall calculated with F_1. TP, FN, and FP are shorten for true positive, false negative, and false positive respectively. Table 4.2 also hilights pacman entry with highest accurate score.

	PacMan Entries	Precision	Recall	F1 Score
20	NearestPill	0.81	0.80	0.80
27	xsl11	0.82	0.80	0.81
28	PhantomMenace	0.81	0.81	0.80
60	brucetong	0.59	0.57	0.56
64	mcharles	0.80	0.80	0.79
67	Ant_Bot	0.47	0.47	0.46
76	Zekna	0.86	0.81	0.80
78	hacklash	0.58	0.54	0.53
80	**Spooks**	**0.92**	**0.92**	**0.92**
82	ICEpAmbush_CIG11	0.63	0.56	0.55
83	rcpinto	0.73	0.73	0.73
88	schrum2	0.53	0.50	0.50
89	CERRLA	0.80	0.78	0.78
90	emgallar	0.55	0.54	0.54
91	RandomNonRev	0.31	0.30	0.29
92	garner	0.58	0.58	0.57
		0.67	0.66	0.65

Table 4.2: Performance of classifiers built with different PacMan entries

4.3 Classifier result

As hilighted in table 4.2, classifier trained with Spooks pacman entry has the highest F_1 score of 0.915. The classified result of the classifier can be visualised with confusion matrix in figure 4.1. The confusion matrix shows that the ghosts team entries 64 and 74 have negative effect on the accuracy. These are mcharles

and kveykva ghosts teams. In fact, these two ghosts teams has identical implementation. They are the startup ghosts controller provided by the competition. Further experiment will exclude ghosts team 64, and will regard ghosts team 74 as a valid entry. The rank differences should not affect the classification or the ranking order. The reason is that the classification treats the rank value as label not ordinal while the ranking order depends only on the voted rank. Table 4.3 shows the detail accurate scores before excluding ghosts team 64.

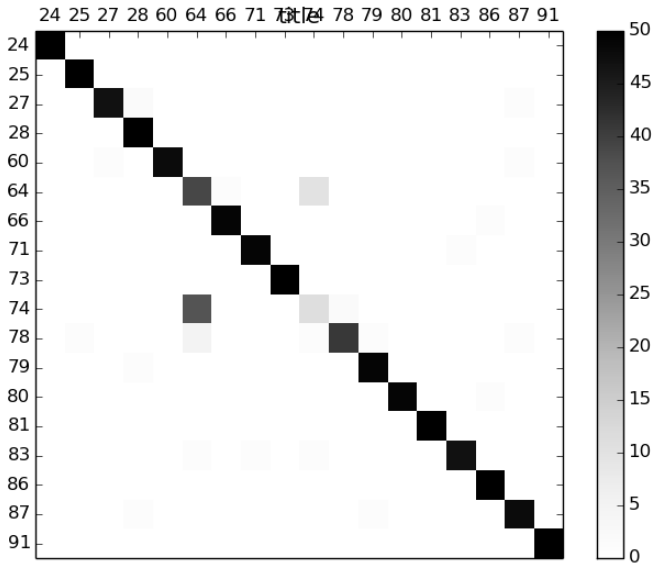

Figure 4.1: Confusion Matrix for Classifier Trained with SpooksPacman Games

ID	Ghosts Team	Precision	Recall	F1-Score
24	Legacy	1.00	1.00	1.00
25	Legacy2TheReckoning	0.98	1.00	0.99
27	xsl11	0.98	0.94	0.96
28	PhantomMenace	0.93	1.00	0.96
60	brucetong	1.00	0.96	0.98
64	mcharles	0.48	0.78	0.59
66	GLaDOS	0.98	0.98	0.98
71	num01	0.98	0.98	0.98
73	Nostalgia	1.00	1.00	1.00
74	kveykva	0.48	0.22	0.30
78	hacklash	0.95	0.82	0.88
79	jackhftang	0.96	0.98	0.97
80	Spooks	1.00	0.98	0.99
81	ICEgUCT_CIG11	1.00	1.00	1.00
83	rcpinto	0.98	0.94	0.96
86	KaiserKyle	0.96	1.00	0.98
87	Scintillants	0.94	0.96	0.95
91	Random	1.00	1.00	1.00
		0.92	0.92	0.92

Table 4.3: Accuracy Score Report for Classifier Trained with SpooksPacman Games

Figure 4.2 shows confusion matrix of the classifier trained with Spooks pacman games excluding games from ghosts team 64. Table 4.4 shows the actual accurate scores.

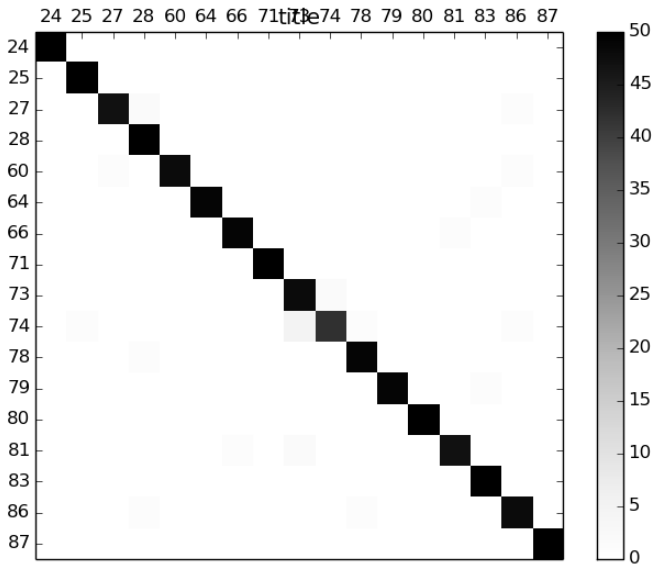

Figure 4.2: Confusion Matrix for Classifier Trained with SpooksPacman Games After Removing Duplicate Entry

ID	Ghosts Team	Precision	Recall	F1-Score
24	Legacy	1.00	1.00	1.00
25	Legacy2TheReckoning	0.98	1.00	0.99
27	xsl11	0.98	0.94	0.96
28	PhantomMenace	0.93	1.00	0.96
60	brucetong	1.00	0.96	0.98
66	GLaDOS	1.00	0.98	0.99
71	num01	0.98	0.98	0.98
73	Nostalgia	1.00	1.00	1.00
74	kveykva	0.87	0.96	0.91
78	hacklash	0.96	0.84	0.89
79	jackhftang	0.96	0.98	0.97
80	Spooks	1.00	0.98	0.99
81	ICEgUCT_CIG11	1.00	1.00	1.00
83	rcpinto	0.98	0.94	0.96
86	KaiserKyle	0.96	1.00	0.98
87	Scintillants	0.94	0.96	0.95
91	Random	1.00	1.00	1.00
		0.97	0.97	0.97

Table 4.4: Accuracy Score Report for Classifier Trained with SpooksPacman Games

4.4 Update result for selecting pacman entry

The experiment re-evaluates the classifiers for each pacman entries after removing the duplicate ghosts team entries. Table 4.5 shows the performance result where the classifier trained with SpooksPacman still remains the highest accurate clas-

sifier.

	PacMan Entries	Precision	Recall	F1 Score
20	NearestPill	0.849	0.846	0.845
27	xsl11	0.863	0.839	0.844
28	PhantomMenace	0.866	0.859	0.861
60	brucetong	0.609	0.589	0.585
64	mcharles	0.855	0.848	0.849
67	Ant_Bot	0.504	0.494	0.492
76	Zekna	0.914	0.853	0.847
78	hacklash	0.615	0.572	0.565
80	**Spooks**	**0.973**	**0.972**	**0.972**
82	ICEpAmbush_CIG11	0.677	0.588	0.597
83	rcpinto	0.795	0.779	0.780
88	schrum2	0.574	0.535	0.535
89	CERRLA	0.837	0.819	0.821
90	emgallar	0.576	0.562	0.563
91	RandomNonRev	0.332	0.319	0.311
92	garner	0.619	0.618	0.609
		0.716	0.693	0.692

Table 4.5: Classifiers Comparison After Removing Duplicate Ghosts Team Entries

4.5 Using the classifier as ranker

Base on the performance shown in table 4.5, the most promising classifier is the one trained using SpooksPacman. The experiment uses this classifiers as a ranker to match that of voted rank by human players.

To rank a set of games, the experiment adds extra step after the games pre-

diction - rank score calculation. Two type of rank score calculations are used in the experiment: weighted rank and max rank. The equation for weighted rank is $r = \sum_{i=1}^{17} c_i v_i$ where i indicates the index of ghosts team entry. c_i is the number of games classified as belonging to the i^{th} ghosts team entry. v_i is the voted rank for the i^{th} ghosts team entry. Max rank simply uses the voted rank of the most predicted ghosts team entry. The experiment deploys two ranking correlation coefficient to evaluate the predicted rank to the voted rank: Spearman's ρ, and Kendall's τ. Both Spearman's ρ and Kendall's τ output value in the range -1 to 1. Value closes to 1 indicates strong correlation, where value closes to -1 indicates negative association. Spearman's ρ concerns only the rank distances in its equation $\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$. Kendall's τ equation for this experiment is $\tau = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$. $n_0 = \frac{n(n-1)}{2}$

$n_1 = \sum_i \frac{t_i(t_i-1)}{2}$

$n_2 = \sum_j \frac{u_j(u_j-1)}{2}$

$n_c =$ number of concordant pairs

$n_d =$ number of discordant pairs

$t_i =$ number of tied values in the i^{th}group of ties for the first quantity

$u_j =$ number of tied values in the j^{th}group of ties for the second quantity

Kendall's τ is preferable when predicted rank contains ties. The experiment also reports the p-value from significance test from each calculation. The p-value, based on 10% significance level, can be interpreted against null hypothesis as followed:

- $p \leq 0.01$ very strong presumption.

- $0.01 < p \leq 0.05$ strong presumption.
- $0.05 < p \leq 0.1$ low presumption.
- $0.1 < p$ no presumption.

Table 4.6 shows ranking result of the classifier training on all 17 ghosts team entries. The ranking result shows a perfect ranking systems when equally trained on two thirds of the games from all 17 ghosts teams leaving one third of the portion for testing.

Ghosts	24	25	27	28	60	66	71	73	74	78	79	80	81	83	86	87	91	Voted	Weighted	Max
24	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1.00	1
25	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	17.00	17
27	0	0	47	2	0	0	0	0	0	0	0	0	0	0	0	1	0	9	9.08	9
28	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14.00	14
60	0	0	1	0	48	0	0	0	0	0	0	0	0	0	0	1	0	15	14.64	15
66	0	0	0	0	0	49	0	0	0	0	0	0	0	0	1	0	0	16	15.76	16
71	0	0	0	0	0	0	49	0	0	0	0	0	0	1	0	0	0	13	12.98	13
73	0	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	2	2.00	2
74	0	0	0	0	0	0	0	0	48	2	0	0	0	0	0	0	0	11	10.88	11
78	0	1	0	0	0	0	0	0	5	42	1	0	0	0	0	1	0	8	8.34	8
79	0	0	0	1	0	0	0	0	0	0	49	0	0	0	0	0	0	6	6.16	6
80	0	0	0	0	0	0	0	0	0	0	0	49	0	0	1	0	0	10	9.88	10
81	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0	5	5.00	5
83	0	0	0	0	0	0	1	0	2	0	0	0	0	47	0	0	0	12	11.98	12
86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0	4	4.00	4
87	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	48	0	3	3.28	3
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	18	18.00	18
ρ																		1.00	1.00	1.00
p value																		0.00	0.00	0.00
τ																		1.00	1.00	1.00
p value																		0.00	0.00	0.00

Table 4.6: Ranking results from classifier trained with 17*150 games tested with 17*50 games.

4.6 Ranking Result With Leave-One-Out

The experiment applies leave-one-out validation to measure the ranking performance on untrained games presumably from unknown ghosts team. Seventeen rankers are generated each of which is trained by omitting games from one ghosts

team entry. The i^{th} ranker omits the games from ghosts team entry index i. It will train on 16×150 games and test on 100 games from ghosts team entry index i. Omitting ghosts team serve as unknown ghosts team entry. Leave-one-out is a common technique to prevent overfitting problem and alternately evaluate how well the ranker will works on unseen samples drawing from the population. Table 4.7 shows the result of the rankers and the overall correlation of the weighted rank and max rank. Even though, both coefficients show positive correlation to the voted rank, the values are not significant enough to draw the conclusion. However, the p-value is close to the targeting 0.1 on significance level of 10%. Future research could fine tune the classifiers to get a better ranking result. Hyperparameter techniques or gridsearch should lead to a better result.

Ghosts	24	25	27	28	60	66	71	73	74	78	79	80	81	83	86	87	91	Vote	Weight	Max
24	-	0	0	0	1	10	0	0	1	28	5	0	0	0	5	0	0	1	9.20	8
25	0	-	0	0	0	2	0	0	28	18	0	0	0	2	0	0	0	17	10.16	11
27	0	0	-	21	0	0	0	0	0	0	9	0	0	0	0	20	0	9	8.16	14
28	0	0	2	-	1	17	1	0	0	0	0	5	0	0	0	24	0	14	8.80	3
60	0	5	2	10	-	27	0	0	1	0	0	0	0	2	0	2	1	15	14.68	16
66	0	0	0	19	6	-	0	0	10	1	0	14	0	0	0	0	0	16	12.28	14
71	0	0	0	0	1	0	-	0	12	0	0	0	0	37	0	0	0	13	11.82	12
73	0	8	0	22	0	8	2	-	1	4	0	0	0	0	0	0	5	2	14.62	14
74	0	1	0	0	0	3	1	0	-	34	0	0	0	11	0	0	0	11	9.64	8
78	0	5	0	1	0	2	0	0	39	-	0	2	0	0	1	0	0	8	11.68	11
79	0	0	2	15	2	3	0	0	0	0	-	20	0	0	4	4	0	6	10.68	10
80	0	0	0	9	0	8	0	0	2	8	15	-	0	0	8	0	0	10	9.24	6
81	0	0	0	0	4	5	0	0	0	0	38	0	-	0	0	0	3	5	8.44	6
83	0	0	0	0	0	0	21	0	28	0	0	1	0	-	0	0	0	12	11.82	11
86	0	0	0	1	0	0	0	0	0	28	5	15	0	0	-	1	0	4	8.42	8
87	0	0	3	46	1	0	0	0	0	0	0	0	0	0	0	-	0	3	13.72	14
91	0	0	0	1	27	12	0	8	0	0	0	0	0	2	0	0	-	18	13.02	15
ρ																		1.00	0.24	0.28
p																		0.00	0.35	0.27
τ																		1.00	0.21	0.21
p																		0.00	0.23	0.24

Table 4.7: Leave-One-Out Ranking Result

4.6.1 Ranking by grouping

Generally a video games has a small number difficulty levels for player to choose from. Exceptions are games with subjective difficulty which intention to offer challenges appropriate to players of different skill levels. Most video games set AI difficulty levels to a small number ranging from 4 to 8 levels. The original Ms. PacMan prompts player with four difficulty levels and challenges the player

with speed-up where higher levels incur faster NPCs movements. The modified version of Ms. PacMan for the competition, however, has removed the speed-up factor. Hence, the difficulty levels relies solely on the challenging levels of the ghosts team.

To achieve lower number of fun levels, the ghosts team entries are ranked in descending order and divided into the desired number of groups. Therefore the game data generated from the ghosts entries in the same group belongs to the same label or have the same rank. The group number indicates the rank. The lower number group has higher preference to the ones with higher number.

The experiment trains and verifies the classifiers on the group-wise game data. The verification process uses a common cross-validation technique called leave-one-out where each iteration reports the rank of the unseen game data using weighted rank algorithm 11.

Table 4.8, 4.9, and 4.10 shows the result when the number of groups are 4, 5, and 6 respectively.

Group	1	2	3	4	Voted	Weighted	Max
1	-	116	10	74	1	2.79	2
2	76	-	46	78	2	2.63	4
3	16	58	-	126	3	3.18	4
4	48	68	134	-	4	2.34	3
ρ					1.00	-0.40	0.32
p value					0.00	0.60	0.68
τ					1.00	-0.33	0.18
p value					0.04	0.50	0.71

Table 4.8: Ranking result when separating ghosts teams into 4 levels

Group	1	2	3	4	5	Voted	Weighted	Max
1	-	23	61	20	46	1	3.59	3
2	38	-	68	7	37	2	3.03	3
3	39	33	-	51	27	3	2.96	4
4	32	0	58	-	110	4	3.78	5
5	32	3	26	139	-	5	3.36	4
ρ						1.00	0.10	0.79
p value						0.00	0.87	0.11
τ						1.00	0.00	0.67
p value						0.01	1.00	0.10

Table 4.9: Ranking result when separating ghosts teams into 5 levels

Group	1	2	3	4	5	6	Voted	Weighted	Max
1	-	20	41	0	35	54	1	4.41	6
2	36	-	67	0	44	3	2	3.17	3
3	25	30	-	31	41	23	3	3.68	5
4	0	0	48	-	4	98	4	5.01	6
5	34	3	13	2	-	98	5	4.50	6
6	4	0	18	26	102	-	6	4.48	5
ρ							1.00	0.60	0.06
p value							0.00	0.21	0.91
τ							1.00	0.33	0.08
p value							0.00	0.35	0.83

Table 4.10: Ranking result when separating ghosts teams into 6 levels

4.6.2 Remarks on using weighted ranking score

Table 4.10 demonstrates the drawback of using rank calculated with the weighted ranking score when classified games from group number 6. Majority of the games are classified as the lowest rank possible. Intuitively, the rank should be close to 6 instead the calculation yield 4.48 resulting in $\rho = 0.608$. The ranking correlation will increase to $\rho = 0.771$ with p-value $= 0.072$ if the rank of 5.1 is used. Meaningful conclusion could be drawn.

4.6.3 Fixing the weighted ranking score

The following algorithms is used to solve the weighted rank scoring problem. Table 4.11 shows the result of the recalibration algorithm. The new rank correlation increases to 0.771 with significance level of 0.072. The result is an improvement over the original weighted rank technique. The original result is:

Data: prediction(p), rank(r)
Result: weighted rank(w)
$f(i)$ = frequency of rank i in prediction p;
$w = \sum_{i=1} i \times f(i)$;

Algorithm 11: Weighted Rank

Data: prediction(p), rank(r), leaveoutindex(e)
Result: recalibrated rank(w)
$r = [r_{min}, r_{max}]$
remove $e$from $r$and sequentialise element without skipping values
$r' =$ recalibrate$(r - e)$
$r' = [r'_{min}, r'_{max}]$

$$w = \frac{(x - r'_{min})(r_{max} - r_{min})}{(r'_{max} - r'_{min})} + r_{min}$$

Algorithm 12: Recalibrate Weighted Rank

Group	1	2	3	4	5	6	Voted	Weighted	Recalibrate	Max
1	-	20	41	0	35	54	1	4.41	3.02	6
2	36	-	67	0	44	3	2	3.17	1.76	3
3	25	30	-	31	41	23	3	3.68	2.56	5
4	0	0	48	-	4	98	4	5.01	4.17	6
5	34	3	13	2	-	98	5	4.50	3.56	6
6	4	0	18	26	102	-	6	4.48	4.35	5
ρ							1.00	0.60	0.77	0.06
p value							0.00	0.21	0.07	0.91
τ							1.00	0.33	0.60	0.08
p value							0.01	0.35	0.09	0.83

Table 4.11: Leave-one-out validation result of the ranker when group the ghosts team into 6 groups.

4.7 User Experience Ranking

When the labels for the classification is small, leave-one-out technique performs badly as demonstrated in previous section. This prompts for a better way to evaluate the classifier. The simple approach is to treat each member of the group separately not as a whole. Since each member of a group is a ghosts team entry, leaving any of the member out as a validation sample is reasonable. Moreover, the skipped ghosts teams could be considered unknown to the ranker because no data from ghosts team entry are available during the training process.

In actual event of ranking a new ghosts team, the game data must be generated. This is done by playing out the game using a reference pacman entry.

The reference pacman entry should provides best support for the ranker. Hence, it is commonly the most accurate pacman entry in the pool. Without loss of generality, this experiment uses Spooks 80 as the reference. The dataset are from the reference pacman entry.

4.7.1 Ranked Groups as User Experience Levels

Previous chapter has analysed the on-line user preference data and presented the user preference ranking of the ghosts teams. The users prefer to play against the ghosts teams on the top of the rank to the ones at the bottom of the table. These ghosts teams offer better experience to majority of the users. The preference order gives natural order for categorizing the levels of user experience from best to worst. Rather than trying to quantify the preference, the experiment focuses on maintaining equal group size. There are two main argument for this. Firstly, equal group size will most likely maintain equal unique features exhibited by each ghosts team member. Secondly, each group will have the fair amount samples for both training and testing.

Table 4.12 shows the user experience levels when categorised into 6 levels.

User Experience Level	ID	Ghosts Name	Vote Rank
1	24	Legacy	1
	73	Nostalgia	2
	87	Scintillants	3
2	86	KaiserKyle	4
	81	ICEgUCT_CIG11	5
	79	jackhftang	6
3	78	hacklash	8
	27	xsl11	9
	80	Spooks	10
4	74	kveykva	11
	83	rcpinto	12
	71	num01	13
5	28	PhantomMenace	14
	60	brucetong	15
	66	GLaDOS	16
6	25	Legacy2TheReckoning	17
	91	Random	18

Table 4.12: Ghosts Team Ranking with 6 User Experience Levels

4.7.2 Ranker for User Experience Levels

Given an unknown ghosts team data, the final ranker should reliably identify the user experience level or the preference level.

To ensure the ranker performance, the test samples should equally fall into

any of the user experience level. One way to simulate such process is to reserve one member from each level as the test sample. Then, the ranker trains on the remaining sample. A ranker is uniquely govern by its reserving list hence there are 486 rankers.

One such instance is the ranker with the reserving list $87-79-80-71-66-91$. The experiment generates this ranker by a training SVM classifier with 2,200 samples. The training samples are from the remaining 11 ghosts teams each of which provides 200 samples. The ranker classifies the testing ghosts team one by one. Each time it reports the distribution of 100 classified samples among the levels along with the **weighted** value and the level with **max**imum frequency. Finally the score for the ranker is calculated using Spearman's rank correlation between the actual rank and the weighted rank. Table 4.13 is the score report of this ranker.

Group	1	2	3	4	5	6	Rank	Weighted	Max
1	0	0	12	0	38	0	1	4.52	5
2	1	21	5	0	23	0	2	3.46	5
3	1	8	34	2	5	0	3	3.04	3
4	0	0	0	48	2	0	4	4.04	4
5	0	1	6	2	41	0	5	4.66	5
6	3	0	0	0	47	0	6	4.76	5
ρ							1.00	0.60	0.03
p value							0.00	0.21	0.95
τ							1.00	0.47	0.09
p value							0.00	0.19	0.81

Table 4.13: Ghosts Team Ranking with 6 User Experience Levels

4.8 Optimal User Experience Ranker

Further study of the experience rankers shows that there are diversity of performances. The correlation scores range from -0.71 to 0.94. There are 20 instances with the 0.94 while there is only 1 ranker scores -0.71. The mean for all 486 rankers is 0.32 with 0.37 standard deviation. The histogram of the study is in figure 4.3.

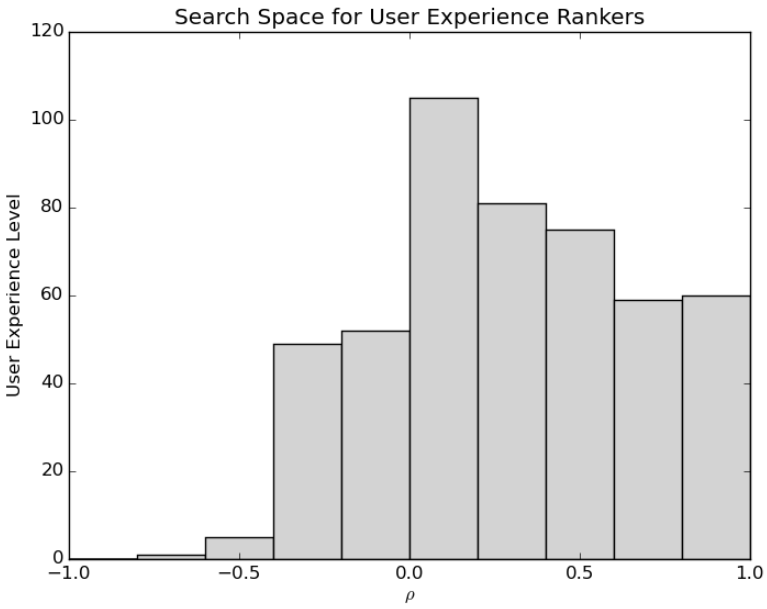

Figure 4.3: Histogram of 486 User Experience Rankers

Table 4.14 shows an optimal experience ranker using Spooks as pacman reference.

Group	1	2	3	4	5	6	Rank	Weighted	Max
1	0	15	30	0	5	0	1	2.90	3
2	0	0	49	0	1	0	2	3.04	3
3	2	1	12	32	3	0	3	3.66	4
4	0	0	3	45	2	0	4	3.98	4
5	0	0	8	0	42	0	5	4.68	5
6	1	0	6	22	21	0	6	4.24	4
ρ							1.00	0.94	0.80
p value							0.0	0.00	0.05
τ							1.00	0.87	0.70
p value							0.00	0.01	0.05

Table 4.14: An Optimal User Experience Ranker $24-86-78-74-66-25-200$

4.9 Blending Ghosts Team

In this experiment, an evolvable ghosts team is implemented using the optimal user experience ranker from previous section. The ranker reports that game data from the reserving list $24-86-78-74-66-25$ gives the most relevant ranks. The ranker uses Spooks as the reference pacman entry.

4.9.1 Implementation

The BlendingGhosts will alter its responding actions by switching the index through the reserving list. The index value directly maps to user experience level minus one. When the calculated index is 0, the BlendingGhosts responds with the Legacy, 24, decision. When the calculated index is 1, the BlendingGhosts

responds with the KaiserKyle, 86, decision, and so forth. The ghosts changes by varying the weight variable. It uses algorithm 13 to find the index value from the given weight in the action selection process.

Data: weight: w, ranks: n

Result: ghosts actions

$r = (n - 1) * w$;

$i = \lfloor r \rfloor$;

if $u(0, 1) > r - i$ **then**

  $i = i + 1$;

end

actions = ghosts[i].getAction();

Algorithm 13: actions selection for blending ghosts team

4.9.2 Weight Variation and Result

To show the effect of weight variation, the experiment uses the ranker to rank games generated using SpooksPacMan and BlendingGhosts. Weight assigned to the BlendingGhosts varies from 0 to 1 with the step size of 0.05. For each corresponding weight value, the experiment generates 100 games as the testing data for the ranker. The total number of training games for the ranker is 2,200 games. The games are from 11 ghosts teams play against SpooksPacMan. These are all the ghosts teams from the entries except for the 6 ghosts teams used in the BlendingGhosts. Table 4.15 show the result of different weight values and the calculated rank.

Weight%	1	2	3	4	5	6	Weight	Weighted	Max
0	0	41	41	1	17	0	0.00	2.94	2.50
5	1	30	63	0	6	0	0.05	2.80	3
10	1	21	71	0	7	0	0.10	2.91	3
15	2	16	79	0	3	0	0.15	2.86	3
20	3	2	88	3	4	0	0.20	3.03	3
25	5	6	69	8	12	0	0.25	3.16	3
30	6	4	75	3	12	0	0.30	3.11	3
35	3	5	70	2	20	0	0.35	3.31	3
40	3	3	53	14	27	0	0.40	3.59	3
45	0	3	42	26	29	0	0.45	3.81	3
50	2	1	54	18	25	0	0.50	3.63	3
55	0	0	43	26	31	0	0.55	3.88	3
60	1	1	22	64	12	0	0.60	3.85	4
65	0	8	45	5	42	0	0.65	3.81	3
70	1	14	38	0	47	0	0.70	3.78	5
75	0	11	45	0	44	0	0.75	3.77	3
80	1	3	66	0	30	0	0.80	3.55	3
85	3	3	32	0	62	0	0.85	4.15	5
90	2	2	28	0	68	0	0.90	4.30	5
95	0	0	40	2	58	0	0.95	4.18	5
100	1	0	26	7	66	0	1.00	4.37	5
ρ							1.00	0.90	0.76
p value							0.00	0.00	0.00
τ							1.00	0.74	0.65
p value							0.00	0.00	0.00

Table 4.15: user experience levels corresponding to various weight of Blending-Ghosts

4.10 Conclusions

This chapter gives details on to classify and rank the ghosts team entries for Ms PacMan. It shows that RRF 3.5.5 data can be used to classify and rank the ghosts team entries with high accuracy. The ghosts teams can be grouped and ranked according to the user on-line preferences ranking.

The experiment also propose a recalibrating calculation 12 for weighted rank in each iteration of leave-one-out.

Despite the success, the technique describe in this chapter requires user on-line evaluation as well as the consistency of the extracted RRF data. RRF data of some games might not represent the game state as well as Ms. PacMan. However the ranking technique should be applicable on games whose state could be extracted in much the same way as the RRF.

Chapter 5

Player Skill Levels

This chapter will evaluate the performance of using the RRF 3.5.5 technique to classify Ms PacMan difficulty levels. First section describes the generated data for the classification. Next section gives details on the ranking problem and how to group the pacman entries into fewer levels.

5.1 Experiment data

The experiment for this chapter uses only the entries from the CIG11 3.4 competition. This consists of 15 pacman entries and 17 unique ghosts team entries. When the competition is over, the competition reports the hall-of-fame for all of the entries. Pacman entries are listed by the average score of all of possible matches. The hall-of-fame list is shown in table 5.1 in descending order of their average scores. The strongest pacman entry is at the top of the table while the weakest one is on the bottom. There is no game data available from the competition therefore the experiment has to generate the games first before the analysis.

The hall-of-fame score list is the only reference resource used in this experiment.

Rank	ID	Name	Affiliation	Avg. Score
1	80	Spooks	Private	41,447
2	28	PhantomMenace	Private	32,108
3	82	ICEpAmbush_CIG11	Ritsumeikan University	20,009
4	67	Ant_Bot	University Carlos III de Madrid	17,301
5	60	brucetong	City University of Hong Kong	15,316
6	76	Zekna_	Private	13,386
7	78	hacklash	Brigham Young University	12,825
8	90	emgallar	University Carlos III de Madrid	12,148
9	27	xsl11	Private	8265
10	92	garner	Private	7762
11	83	rcpinto	UFRGS	6504
12	64	mcharles	University of California	5755
13	89	CERRLA	The University of Waikato	4277
14	88	schrum2	University of Texas at Austin	3796
15	22	RandomNonRevPacMan	University of Essex	1197

Table 5.1: Pacman Hall-of-fame List

5.1.1 Generating dataset

A game data is a complete game generated with one pacman entry versus one ghosts team entry. Therefore, a game data consists of a list of game states from initial game state to the last game state. The game data collection contains 51,000 unique game data generated from 15 pacman entries and 17 ghosts entries. That is 200 unique game data for each possible match of a pacman entry versus a ghosts team entry.

The experiment extracts RRF 3.5.5 dataset from the game data collection for

the next process.

5.2 Predictability of the dataset

This section describes the process to find out how predictable each pacman entry is. The experiment trains the selected classifier on two thirds of the dataset. The remaining one third of the dataset is used to evaluate the model.

5.2.1 Data preparation

The dataset is labelled with the corresponding ID of the pacman entry. There are 3,400 RRF data for each label from 200 game data for each ghost entry. The training dataset draws 150 RRF data randomly from the 200 RRF game data for each pacman-ghosts match. This leaves 50 RRF data from the 200 RRF data as the testing dataset. Overall, the training dataset contains 38,250 RRF data and the testing dataset contains 12,750 RRF data.

SVM classifier is selected for this experiment as the result of the classifier study in section 3.5.

5.2.2 Predictability Result

Table 5.2 shows the classifier report. This shows that most pacman entries are highly predictable in a one-vs-all scheme.

ID	Pacman Entries	Precision	Recall	F1 Score
27	xsl11	0.94	0.92	0.93
28	PhantomMenace	0.88	0.90	0.89
60	brucetong	0.44	0.45	0.45
64	mcharles	0.90	0.95	0.93
67	Ant_Bot	0.68	0.70	0.69
76	Zekna	0.82	0.97	0.89
78	hacklash	0.81	0.90	0.85
80	Spooks	0.94	0.96	0.95
82	ICEpAmbush_CIG11	0.80	0.67	0.73
83	rcpinto	0.87	0.94	0.90
88	schrum2	0.95	0.97	0.96
89	CERRLA	0.84	0.92	0.88
90	emgallar	0.66	0.48	0.56
91	RandomNonRev	0.72	0.54	0.62
92	garner	0.48	0.51	0.50
		0.78	0.79	0.78

Table 5.2: Pacman Predictability Table

Using the RRF data, the classifier can identify the corresponding pacman entry with overall accuracy of 78%. Moreover, if the game data belongs to five of the 15 pacman entries it can be identified with more than 90% certainty. However, the opposite is true for the game data from brucetong, emgallar, and garner where the prediction is equal to random. It is important to note that high accuracy

does not automatically imply a rule-based pacman entry or vice versa. Ant_Bot and RandomNonRev are good example of this as Ant_Bot employs evolutionary strategy and RandomNonRev makes random actions. Intuitively, the game data from these two entries should have predictability around 50%. In the case of RandomNonRev, it can be indirectly conclude that RRF 3.5.5 extracts more information than just the pacman actions.

5.3 Reference Ghosts Team Selection

The finalise player skill ranker will be used to rank an unknown pacman entry. In return, the ranker will give the skill level of the pacman entry. It is intuitive to use one reference ghosts team to train the ranker. If new pacman entry needs evaluation, the ranker will rank the game data from this pacman and the reference ghosts team.

This section provides the detail on how select the reference ghosts team. The reference ghosts team is the ghosts team which will boost the performance of the ranker. Therefore, It is the ghosts team entry whose game data gives the highest predictability.

5.3.1 Data Preparation

The RRF dataset is labelled with the IDs of the ghosts team entries. Each label has 3,000 samples. There are 17 labels in the dataset. Altogether, there are 38,250 training data and 12,750 testing data.

5.3.2 Classification Result

Table 5.3 shows the accuracy score report for all the labels. There are ghosts team entries with prediction score more than 80%. While most ghosts team entries has more than 60% chance of correctness, there are two ghosts team entries which has worse prediction than guessing. They are Scintillants and kveykva. The surprisingly high prediction score of Random ghosts team demonstrates again that RRF data contains more information than just the response actions.

ID	Ghosts Entries	Precision	Recall	F1 Score
24	Legacy	0.63	0.79	0.70
25	Legacy2TheReckoning	0.68	0.75	0.71
27	xsl11	0.60	0.69	0.64
28	PhantomMenace	0.54	0.52	0.53
60	brucetong	0.58	0.49	0.53
66	GLaDOS	0.66	0.50	0.57
71	num01	0.79	0.66	0.72
73	Nostalgia	0.67	0.68	0.67
74	kveykva	0.52	0.33	0.40
78	hacklash	0.56	0.60	0.58
79	jackhftang	0.65	0.59	0.62
80	Spooks	0.63	0.58	0.60
81	ICEgUCT_CIG11	0.83	0.75	0.79
83	rcpinto	0.49	0.77	0.60
86	KaiserKyle	0.63	0.59	0.61
87	Scintillants	0.45	0.53	0.48
91	Random	0.68	0.63	0.65
		0.62	0.61	0.61

Table 5.3: Reference Ghosts Teams Classification Result

Comparing to table 5.3 to table 5.2, the predictability of the ghosts team entries is much lower than that of the pacman entries. In other word, it is easier to identify the pacman entry than to identify the ghosts team entry from a given game data. This mainly dues to the fact that there are far fewer action space for a pacman entry than that of the ghosts team entry. Pacman has at most 4 possible actions in a particular game state where ghosts team has at most

$4^4 = 256$ possible actions.

The most reliable ghosts team is ICEgUCT_CIG11 with predictability of 79%. Therefore, it is used as the reference ghosts team in the next section.

5.4 Player Skill Ranking

The player skill levels are presumably a small number. This experiment assumes that there are 5 skill levels. The next section gives the detail of the group division as well as the dataset preparation.

5.4.1 Data Preparation

The experiment divides the pacman entries into 5 groups according to their average score. Each group contains three pacman entries. Group label 1 consists of the top three pacman entries. All game data by the member of the group will be labelled with the group or the rank number. Group with lower number has higher average score, i.e., stronger pacman entries.

Table 5.4 shows all the groups and theirs corresponding pacman entries members.

Group Number (Skill Level)	ID	Pacman Name	Rank	Avg. Score
1	80	Spooks	1	41,447
	28	PhantomMenace	2	32,108
	82	ICEpAmbush_CIG11	3	20,009
2	67	Ant_Bot	4	17,301
	60	brucetong	5	15,316
	76	Zekna_	6	13,386
3	78	hacklash	7	12,825
	90	emgallar	8	12,148
	27	xsl11	9	8265
4	92	garner	10	7762
	83	rcpinto	11	6504
	64	mcharles	12	5755
5	89	CERRLA	13	4277
	88	schrum2	14	3796
	22	RandomNonRevPacMan	15	1197

Table 5.4: Pacman Groups Table

The final dataset consists of 5 labels. Each label has 10,200 RRF data. All 51,000 samples are used in the ranking and evaluation in the next section.

5.4.2 Ranking

The ranker uses SVM classifier with weighted rank method described in Algorithm 11. To assert that the ranker will be able to classify game data of an unknown pacman entry, all game data from 5 selected pacman entries must be omitted in the training process. These omitted game data are later used in the evaluation

process to verify the ranker. The list of the 5 omitted pacman entries is called the testing list. This simulates ranking game data from unseen pacman entry.

Figure 5.1 illustrates the omitting process. The numbers in the stacks are the identification numbers of the pacman entries. The numbers below the stacks are the group labels where each x_i is the IDs of the testing pacman entries. There are 3^5 possible ways to train and test the ranker each of which will give different accuracy score.

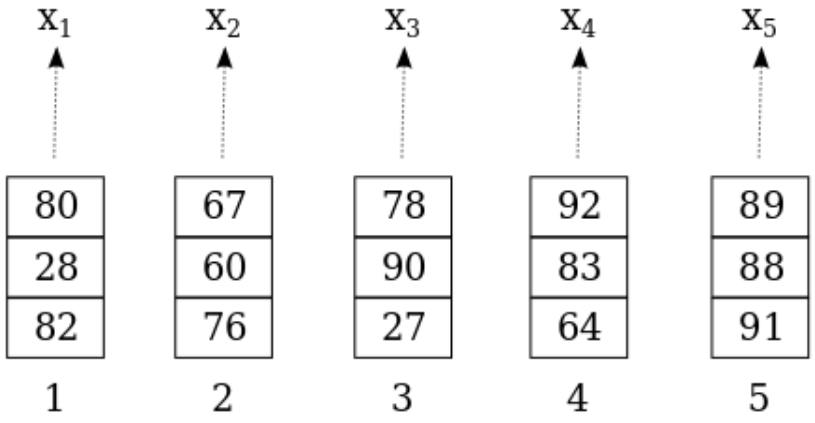

Figure 5.1: Selection of Pacman Entries for Evaluation

String label of the IDs in the testing list is used to differentiate the ranker. For example, ranker with label $80-67-78-92-89$ will be created using the following steps.

- The dataset for the ranker is collection of all the game data from the reference ghosts team.
- The number of samples for training is 2,000. Each group provides 400 training samples from the remaining pacman entries, e.g., 200 samples from

each of its' member. For instance, group 1 randomly selects 200 samples from the game data of the reference ghosts team and pacman entry ID 82. It also does the same for pacman entry ID 28.

- For each of the pacman entries in the testing list:
  randomly sample 50 game data
  rank - how many of these game classified as group 1, 2, 3, 4, and 5
  calculate the weighted rank - **no recalibration is required because no rank is skipped.**

- Find the correlation between the group rank and the weighted rank.

Table 5.5 show the result of ranker $80-67-78-92-89$. The p value shows that no conclusion can be drawn with confidential about the relation between the group rank and the weighted rank. Despite the high value of rho, the ranker $80-67-70-92-89$ is not usable.

Group	1	2	3	4	5	Rank	Weighted	Max
1	50	0	0	0	0	1	1.00	1
2	2	44	3	0	1	2	2.08	2
3	22	2	16	4	6	3	2.40	1
4	0	49	1	0	0	4	2.02	2
5	0	14	10	1	25	5	3.74	5
ρ						1.00	0.70	0.74
p value						0.00	0.19	0.15
τ						1.00	0.60	0.67
p value						0.01	0.14	0.10

Table 5.5: Result of Ranker $80 - 67 - 78 - 92 - 89$

5.5 Optimal Player Skill Ranker

As mentioned in previous section, there are $3^5 = 243$ ways to build the player skill rankers for a reference ghosts team entry. All 243 rankers are investigated in this experiment. Every ranker are generated and tested using the procedure describe in section 5.4.2. Some of the rankers are inconclusive with no correlation at all. A number of the rankers are highly usable with high correlation and minimal p-value. Table 5.6 shows one example of the 26 optimal player skill rankers.

Group	1	2	3	4	5	Rank	Weighted	Max
1	27	23	0	0	0	1	1	1
2	1	47	1	0	1	2	2	2
3	1	33	8	3	5	3	2	2
4	3	4	15	27	1	4	3	4
5	0	10	0	0	40	5	4	5
ρ						1.00	0.97	0.97
p value						0.00e+00	0.00	0.00
τ						1.00	0.95	0.95
p value						0.01	0.02	0.02

Table 5.6: An Optimal Player Skill Ranker $80 - 67 - 90 - 83 - 89$

Figure 5.2 shows the count of rankers across the range of the Spearman's ρ value $[-1, 1]$. The figure shows that there are 38 rankers whose ρ values are 0.83 or more. There are 26 optimal player skill rankers with maximum ρ value of 1.0. Unfortunately, majority of these rankers can not be used to reliably rank an unknown pacman entry.

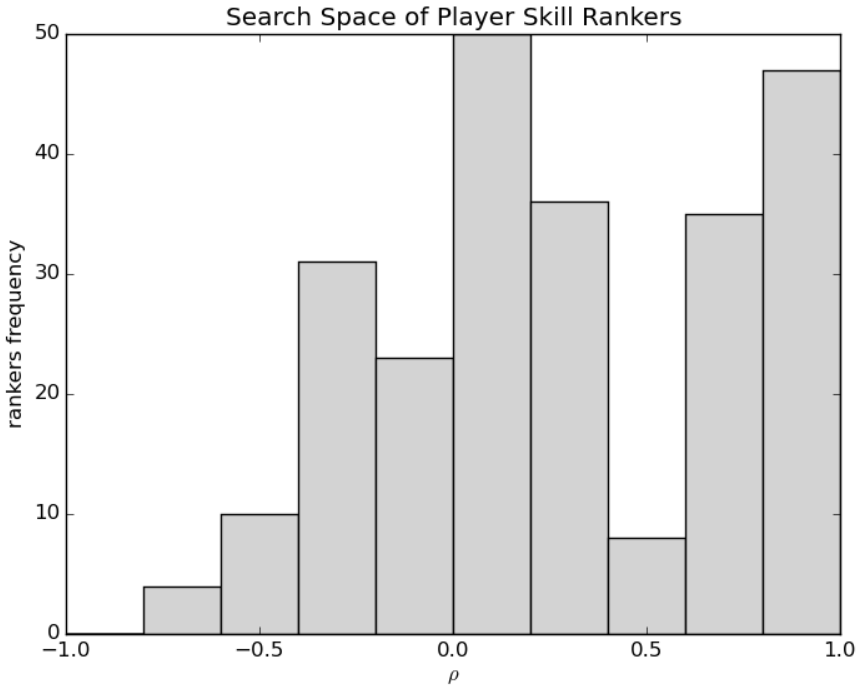

Figure 5.2: Histogram of 243 Player Skill Rankers With Spearman's ρ Values

5.6 Blending PacMan

In this experiment, a new pacman entry is implemented using the testing list from an optimal ranker from previous section. The experiment will show that it is possible to adjust the pacman entry to any skill levels. The new pacman entry is called BlendingPacMan. BlendingPacMan is adjustable through a double variable weight with the range from 0.0 to 1.0. Varying weight from 0.0 to 1.0 should gradually increase the skill level from 1 to 5. The action selection algorithm is the same as that of BlendingGhosts described in algorithm 13.

5.6.1 Data generation

The weight is set to vary from 0.0 to 1.0 with the step size of 5. For every value of weight, BlendingPacMan plays 100 games against the reference ghosts team, ICEgUCT_CIG11 whose ID is 81. The experiment carries out extra step to ensure that the 100 games for each weight value are unique.

There are 2,100 game data from all 21 weight values. The experiment uses RRF to extract these game data and uses as the testing dataset.

5.6.2 Ranking

The ranker $80 - 67 - 90 - 83 - 89$ from table 5.6 is selected to rank the testing dataset. For each weight value, the ranker classify the corresponding 100 RRF data and report the weighted rank. The weighted result for all of the weight values are in table 5.7 along with the ρ correlation to the weight values.

Weight%	1	2	3	4	5	Weight	Weighted	Max
0	89	0	0	1	10	0.00	1.43	1
5	8	88	0	0	4	0.05	2.04	2
10	7	89	2	0	2	0.10	2.01	2
15	6	83	10	0	1	0.15	2.07	2
20	4	79	16	0	1	0.20	2.15	2
25	10	85	1	0	4	0.25	2.03	2
30	3	93	3	0	1	0.30	2.03	2
35	7	83	4	0	6	0.35	2.15	2
40	6	81	1	0	12	0.40	2.31	2
45	11	79	1	2	7	0.45	2.15	2
50	8	70	12	2	8	0.50	2.32	2
55	12	64	15	5	4	0.55	2.25	2
60	11	59	14	7	9	0.60	2.44	2
65	6	60	16	10	8	0.65	2.54	2
70	7	27	43	17	6	0.70	2.88	3
75	38	0	9	53	0	0.75	2.77	4
80	18	17	35	18	12	0.80	2.89	3
85	6	24	25	28	17	0.85	3.26	4
90	6	28	13	19	34	0.90	3.47	5
95	8	27	2	14	49	0.95	3.69	5
100	1	18	0	2	79	1.00	4.40	5
ρ						1.00	0.96	0.87
p value						0.00	0.00	0.00
τ						1.00	0.88	0.77
p value						0.00	0.00	0.00

Table 5.7: Result of Tuning BlendingPacMan To Different Skill Levels

Figure 5.3 shows the gradual upward slope of the skill level as the weight in-

crease when calculated using the weighted rank 11. Even though the Spearman's ρ correlation is very high for the curve using max rank, the method might not be suitable for actual skill level ranking. This is mainly because of the abrupt decrement of skill level from 4 to 3 when the weight value is around 0.8.

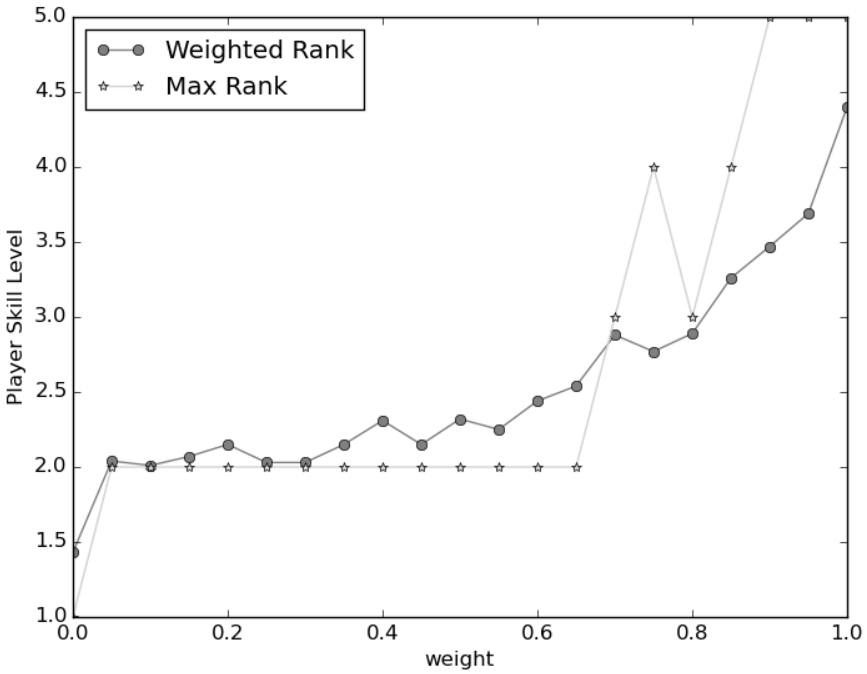

Figure 5.3: Plotting of Weight Variation and Skill Level

5.7 Conclusion

This chapter shows that the RRF 3.5.5 technique is a suitable feature extraction technique for generating player skill ranker in Ms PacMan. Player skill ranker can be created with the straight forward procedure 5.4.2. The resulting optimised player skill ranker can reliably rank an unknown pacman entry based on their

average scores. Once an optimal ranker is found, it is possible to create new pacman entry which can be tuned to any possible skill levels. Some of the experiment in this chapter also strongly suggest that the extracted data encapsulates more information than the pacman actions.

Chapter 6

Optimisation

This chapter presents the result of applying various optimisation techniques to the problems related to designing NPCs in this thesis. The first problem is the search for global optimal rankers for ranking user experience. Section 6.1 states the problems, methodology, and the result. The second optimising problem concerns with the optimal rankers for ranking player skills.

6.1 Optimising User Experience Rankers

The user experience rankers is described in section 4.7. Section 4.8 presents an optimal experience rankers with a specific reference pacman entry, namely, SpooksPacMan. These optimal experience rankers are local optimal in the search space of size 486 samples. They are constrained to smaller pool of rankers. The experiment conducted in this section, however, aims to find overall optimal experience rankers across 16 pools of search spaces. These overall optimal experience rankers will later be called global optimal experience rankers.

The global optimal experience rankers will have the highest ranking score from the original game dataset of 54,400 samples. The search space size for the global optimal experience rankers is 7,776. The search space is small enough for the brute-force method to successfully find the solution within a limited resources. The brute-force method will provide a good reference comparison for other optimisation techniques.

Figure 6.1 show the search space for global user experience rankers.

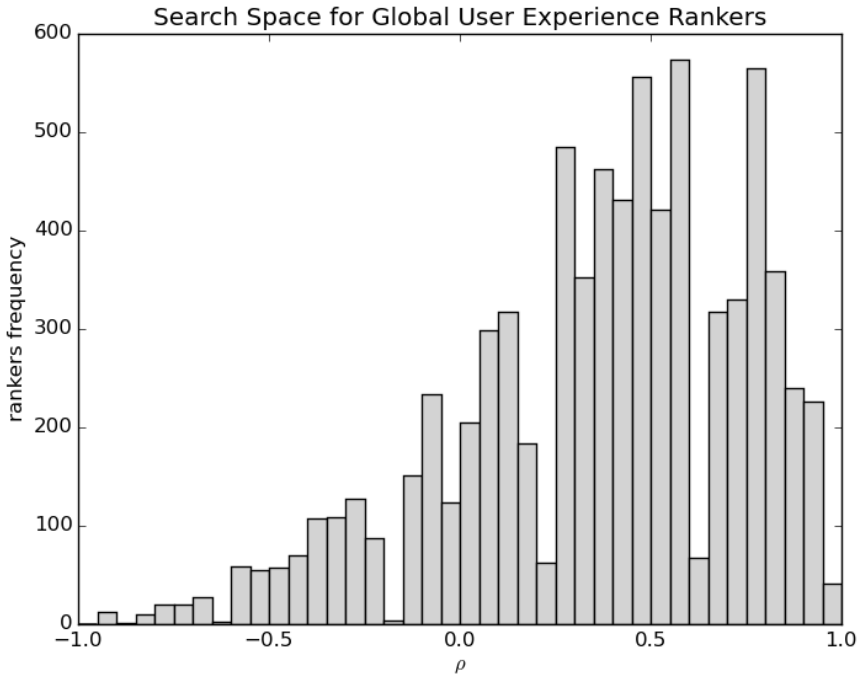

Figure 6.1: Search Space of 7,776 User Experience Rankers

The rankers ρ values range from -0.94 to 1.0. The average ρ value is 0.37 with standard deviation of 0.38. There are 32 global optimal rankers with ρ equals to 1.0 while there are 9 rankers with the minimum value.

6.1.1 Individual Encoding

A ranking solution is commonly called **individual** in evolutionary computing. In this case, the individual is an array where the value of the variables on each index are drawn from different sets. The individual is denoted by x and is defined in equation 6.1.

$$x = (p, g_1, g_2, g_3, g_4, g_5, g_6) \tag{6.1}$$

where

$p \in \{20, 27, 28, 60, 64, 67, 76, 78, 80, 82, 83, 88, 89, 90, 91, 92\}$ all possible values of for the Pac-Man agent.

$g_1 \in \{24, 73, 87\}$ all possible ghost team values that belong to preference level 1.

$g_2 \in \{86, 81, 87\}$ all possible ghost team values that belong to preference level 2.

$g_3 \in \{78, 27, 80\}$ all possible ghost team values that belong to preference level 3.

$g_4 \in \{74, 83, 71\}$ all possible ghost team values that belong to preference level 4.

$g_5 \in \{28, 60, 66\}$ all possible ghost team values that belong to preference level 5.

$g_6 \in \{25, 91\}$ all possible ghost team values that belong to preference level 6.

6.1.2 Algorithms

The testing algorithms are implemented using Distribution Evolutionary Algorithms in Python: DEAP [Fortin et al., 2012], module. These are the algorithms being tested: simple genetic algorithm (GA), simple evolutionary algorithm (sEA) [Baeck et al., 2000], $\mu+\lambda$ evolutionary strategy ($\mu+\lambda$ES), μ, λ evolutionary strategy ($\mu+\lambda$ES), evolutionary algorithm using ask-tell model (ask-tell-ES) [Collette et al., 2010], covariance matrix adaptation evolutionary strategy 2.7 (CMA-ES) [Ostermeier et al., 1994], and a variation of particle swarm optimisation (RDPSO)

in the next section 6.1.2.1.

6.1.2.1 Rolling Discrete PSO: RDPSO

All of the optimisers conform to the originals except for the PSO and CMA-ES. Both algorithms require modification in order to work with discrete variables with sets of possible values. It is worth noting that each corresponding set of values for the variables are in the rank order.

A modified version of PSO is presented here to solve the issue. The algorithm uses the logistic function of the particle velocity as the probability to change the value of the variables. Because, it uses the probability to roll the values to a better rank or down the rank, the algorithm is called Rolling Discete PSO or RDPSO. The variable rolls to the better rank by lowering its value. It does the opposite otherwise. RDPSO follows the conventional binary PSO algorithm [Kennedy and Eberhart, 1995] with variation on the updating function. The detail of RDPSO is given in the algorithm 14. The formulae for updating each particle's velocity $(p.v)$ and $p.x$ are in equation 6.2.

$$
\begin{aligned}
p.v_j &= \omega p.v_j + \phi_p u(0,1)(p.b_j - p.x_j) + \phi_g u(0,1)(g_j - p.x_j) \\
p.x_j &= \begin{cases} roll(p.x_j, -1) & \text{if } u(0,1) < \frac{1}{1+e^{-p.v_j}} \\ roll(p.x_j, +1) & \text{otherwise} \end{cases}
\end{aligned} \tag{6.2}
$$

The same idea applies to CMA-ES in the updating step. The statistics influencing the population's movement uses real number. However, when interpolating between two discrete values, the result has to be converted to integer and wrap around.

6.1.3 Evaluation Method

Random sampling is used the reference benchmark for performance evaluation. Each optimiser run 5,000 times. Each run records the number of evaluations and statistics of the final population. The run completes when a global optimal solution is found or the maximum generation of 1000 is reached. The expected mean of random sampling 32 solutions from 7,776 should be 243. The experiment compares the percentage of runs with lower evaluations calls. These runs are referred to as better runs. The result of random sampling on the problem is in figure 6.2. The random sampling reports 3,177 better runs with the mean of 243.45, and standard deviation of 243.07. This experiment uses line plot of the histogram to visualise the result of the optimisers.

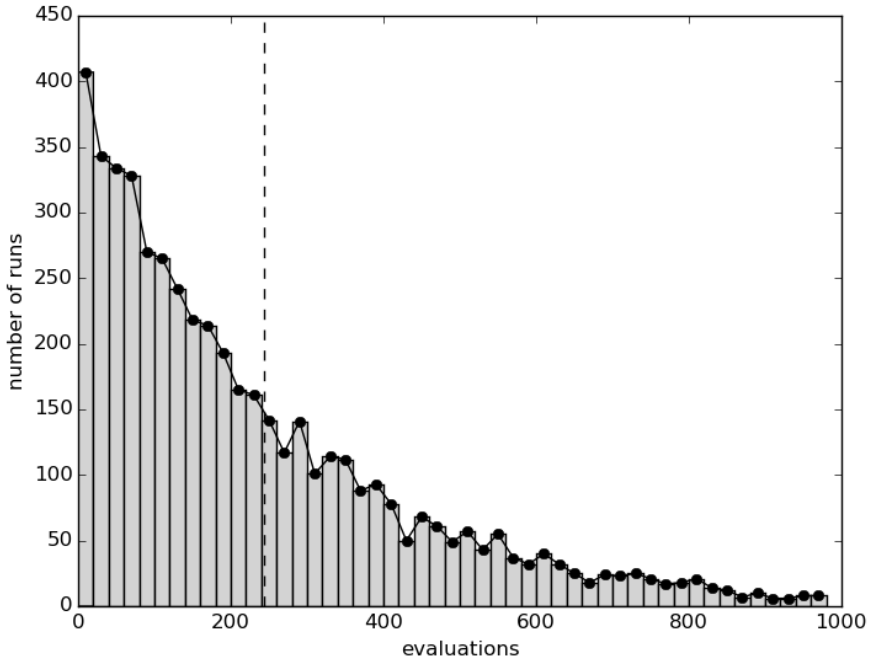

Figure 6.2: Number of Evaluations Calls by Random Sampling)

6.1.4 Results

Most optimisers have successes on most of the runs. Majority of the runs used less number of evaluation calls than the expected mean. One of such example is shown in figure 6.3.

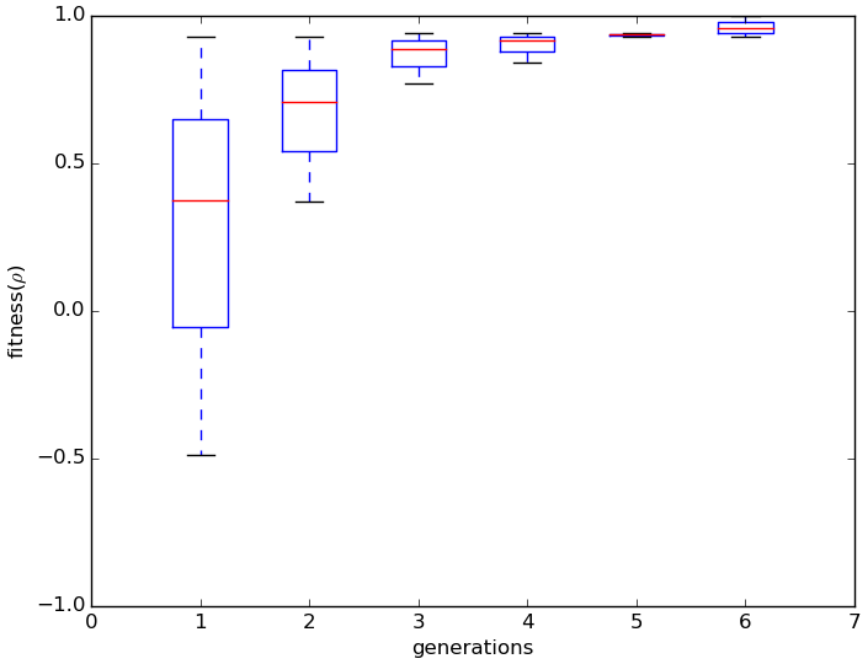

Figure 6.3: An Optimising Result with $(\mu + \lambda)$ ES

The run used 6 generations with 180 evaluation calls. In the final generation, the optimiser found the global optimal user experience ranker $27 - 87 - 79 - 27 - 71 - 60 - 91$.

However, some runs took much more generations than expected. In turn, it explored most of the search space area. Figure 6.4 illustrated those runs as it explore the terrain.

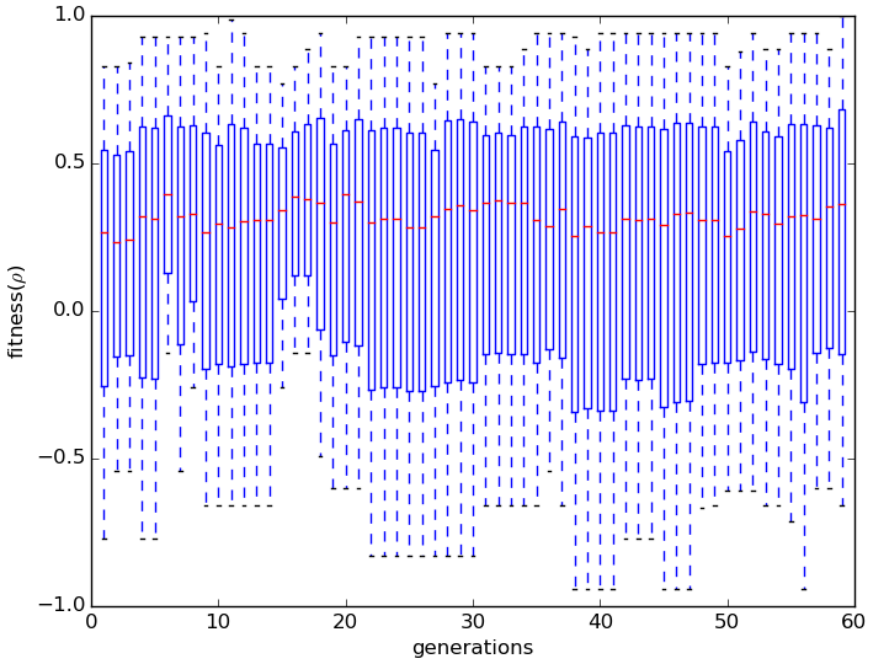

Figure 6.4: A Exploring Run with GA

Figure 6.5 show the histogram plots of some of the optimisers.

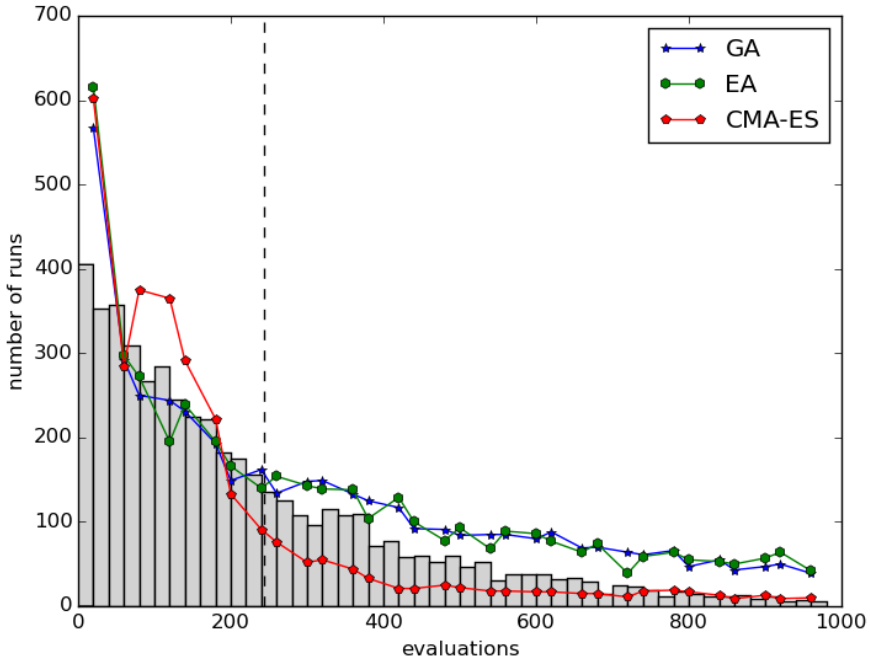

Figure 6.5: Optimisers Result On User Experience Ranking Problem

Table 6.1 summarises the experiment result of the optimisers. The statistics are not directly comparable as 4 of the optimisers contains some unsuccessful runs. Those contains the number of unsuccessful runs as followed; 148, 161, 218, and 430 respectively.

Optimisers	Better Runs (243.00)	Minimum	Mean	Maximum	Standard Deviation
Random	63.98%	0	237.70	1,000	228.22
GA	41.76%	30	544.91	4,830	599.88
sEA	42.40%	30	540.28	5,580	606.14
$\mu + \lambda$	56.32%	30	3,224.36	30,030	6,850.49
μ, λ	53.36%	30	3,502.39	30,030	7,145.70
CMA-ES	47.28%	30	4,371.68	30,030	7,817.17
RSPSO	43.28%	30	2,873.53	30,030	8,333.21

Table 6.1: Optimisers Performance Comparison on User Experience Ranker

6.2 Optimising Player Skill Rankers

The player skill rankers is described in section 5.4 where optimal rankers found in section 5.5 are local. The problem space is illustrated in figure 6.6. It consists of 4,131 unique individuals with 67 global optima. The value for these optima is 1.0. The ρ values of the population ranges from -1 to 1 with the mean of 0.33 and the standard deviation of 0.45.

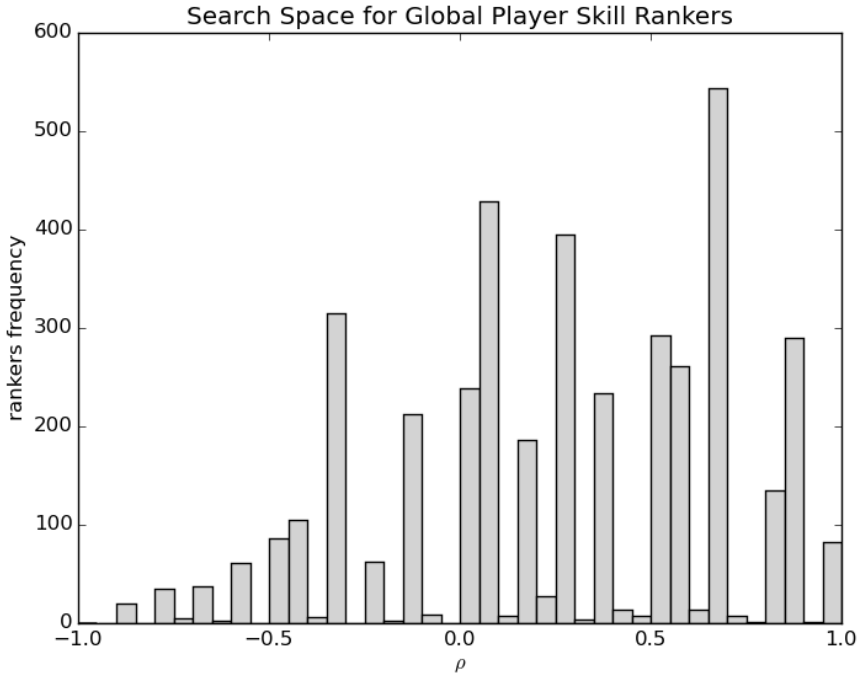

Figure 6.6: Search Space of 4,131 Player Skill Rankers

The random sampling should have the expected of 61.66 as show in figure 6.7.

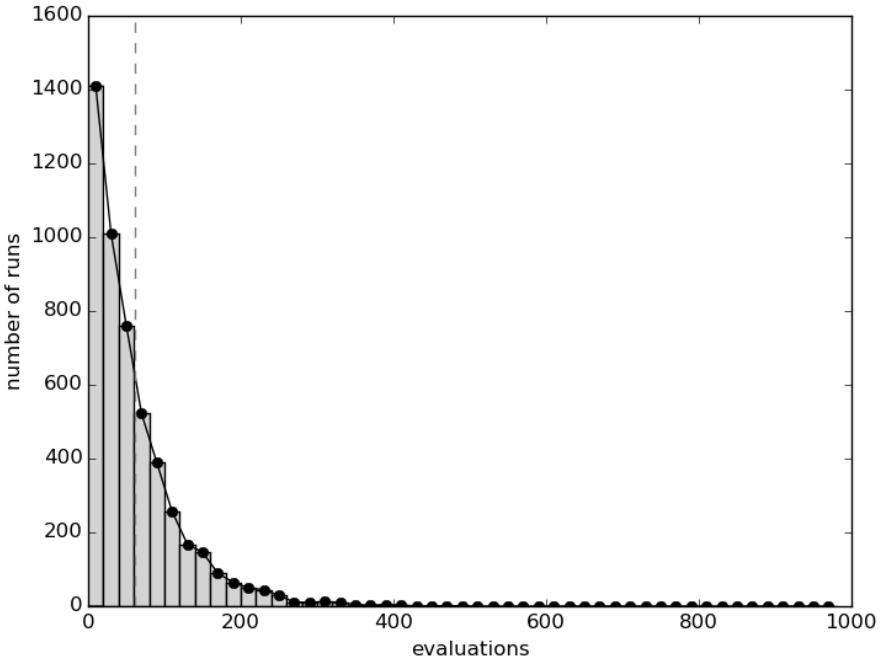

Figure 6.7: Reference Random Sampling for Player Skill Optimisation

6.2.1 Individual Encoding

The encoding of individual consists of 6 numbers. the first number identify the reference ghosts team id. The remaining refers to pacman id corresponding to the group number as shown in equation 6.3.

$$x = (g, p_1, p_2, p_3, p_4, p_5) \tag{6.3}$$

where

$g \in \{24, 25, 27, 28, 60, 66, 71, 73, 74, 78, 79, 80, 81, 83, 86, 87, 91\}$

$p_1 \in \{80, 28, 82\}$

$p_2 \in \{67, 60, 76\}$

$p_3 \in \{78, 90, 27\}$

$p_4 \in \{92, 83, 64\}$

$p_5 \in \{89, 88, 91\}$

6.2.2 Result

The search space for this problem is much smaller. Random sampling seems to work better than the optimisers. Random sampling with the mean of 60.48 outperforms all testing optimiser except for RDPSO. All runs in RDPSO are successful with less than 10 generations. Most of the runs are similar to figure 6.8.

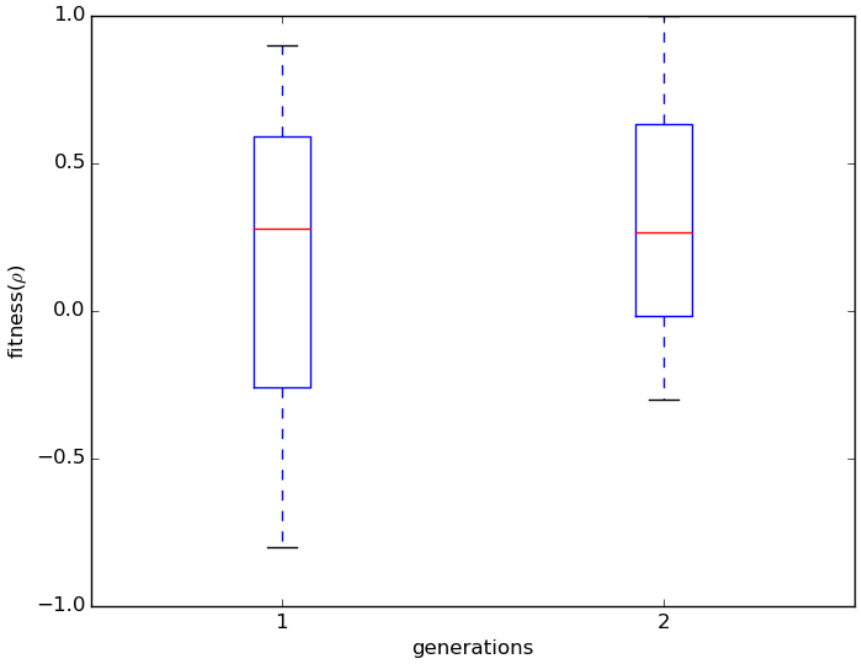

Figure 6.8: A Success Run with RDPSO

After 5,000 runs, the random sampling method results in 30.1% for the better runs with the average of 535 evaluations.

The comparison The reference random sampling result is shown in figure 6.9.

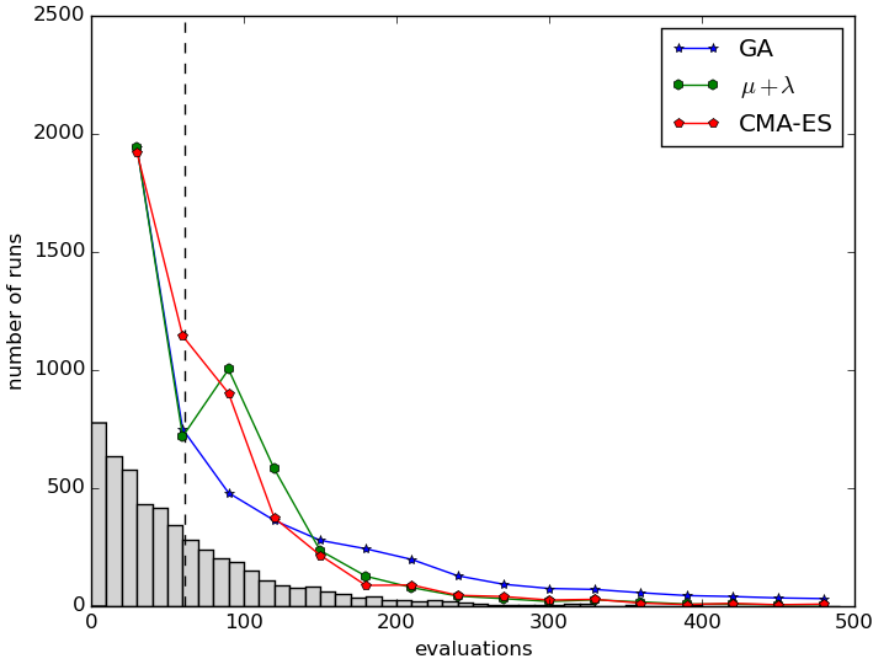

Figure 6.9: Performance Comparison on Player Skill Optimisation

Table 6.2 shows the comparison of the optimisers on player skill rankers.

Optimisers	Better Runs (61.66)	Minimum	Mean	Maximum	Standard Deviation
Random	64.18%	0	60.48	517	62.67
GA	53.86%	30	124.18	1,590	148.72
sEA	53.18%	30	125.27	1,470	148.91
$\mu + \lambda$	53.24%	30	151.11	30,030	843.62
μ, λ	54.56%	30	153.26	29,460	871.83
CMA-ES	61.34%	30	89.84	2,700	137.00
RDPSO	79.04%	30	57.64	270	29.65

Table 6.2: Optimisers Performance Comparison on Player Skill Rankers

6.3 Adaptive Tic-Tac-Toe NPCs using MCTS

Even though the accuracy of the NPC from previous two chapter show high correlation score, it has one drawback. It requires ghosts team entries to make response action.

This section provides alternative way to create adaptive NPC without relying on other ghosts team entries. When responding action is required, this NPC will select best responding action using MCTS. MCTS relies on the accuracy of the rollout result. If rollouts fully explore the game tree, the best responding action is guaranteed. The following experiment demonstrates the MCTS NPCs on Tic-Tac-Toe game.

6.3.1 Statistics

An experiment has been done to confirm statistical data on Tic-tac-toe game. The entire game tree has been explored to starting from an empty game state. The statistics results on Table 6.3 and Table 6.4 has confirmed Tic-tac-toe statistics from the reading resource.

Status	Non-unique Games	Unique Games
x wins	131,184	91
o wins	77,904	44
draw	46,080	3
Total	255,168	138

Table 6.3: Tic-tac-toe Winning Statistics

Tree Level	Nodes	Terminals	Unique
level0	1	0	0
level1	9	0	0
level2	72	0	0
level3	504	0	0
level4	3,024	0	0
level5	15,120	1,440	21
level6	54,720	5,328	21
level7	148,176	47,952	58
level8	200,448	72,576	23
Total	549,946	255,168	138

Table 6.4: Tic-tac-toe Game Tree Statistics

6.3.2 NPC Objectives

To make distinctive preference other than winning, the following types of winning position preferences are tested. **Preference**:

1. player type **R**: prefer to win with 3 in a row - w_0
2. player type **C**: prefer to win with 3 in a column - w_1
3. player type **D**: prefer to win with 3 diagonally - w_2

6.3.3 Implementation

The agent uses the MCTS search function as described in algorithm 15. The heuristic function is in equation 6.4 where $w_0, w_1, w_2, w_3 \in [0.0, 1.0]$.

$$h(s) = w_0\text{rowWin}(s) + w_1\text{columnWin}(s) + w_2\text{diagonalWin}(s) + w_3\text{win}(s) \quad (6.4)$$

$$\text{rowWin}(s) = \begin{cases} 1.0 & \text{if 'X' wins in state } s \text{ by 3 in a row} \\ 0.0 & \text{otherwise} \end{cases}$$

$$\text{columnWin}(s) = \begin{cases} 1.0 & \text{if 'X' wins in state } s \text{ by 3 in a column} \\ 0.0 & \text{otherwise} \end{cases}$$

$$\text{diagonalWin}(s) = \begin{cases} 1.0 & \text{if 'X' wins in state } s \text{ by having 3 diagonally} \\ 0.0 & \text{otherwise} \end{cases}$$

6.3.4 Results

For this experiment, the maximum simulation, S_x, is 1,000 without limitation on maximum rollouts; $R_x = \infty$. The MCTS NPC plays as 'X' against the random player 'O'. The experiments test w_0, w_1, w_2 independently where w_3 is fixed to 1. For each configuration values, MCTS NPC plays 1,000 with random player.

6.3.4.1 R - prefers to win by row

w_0 varies from 0 to 1 in 0.001 intervals while keeping $w_1 = 0, w_2 = 0, w_3 = 1$. For each value of w_0 the following statistics is recorded:

- number of games end with draw - **draws**
- number of games won by 'X' - **xwins**
- number of games won by 'O' - **owins**
- number of games 'X' wins with row - **rowwins**
- number of games 'X' wins with column - **columnwins**
- number of games 'X' wins diagonally - **diagonalwins**

The result of MCTS NPC adapting to preference **R** is shown in figure 6.10. When $w_0 = 0$, the percentage of wins by row is about 35% about 5% percentage less than wins by diagonal. Number of wins by row increases as weight rises from 0 to 0.2 and stays 75% after that. The number of games ends in draw has the mean of 1.12% overall with standard deviation of 0.34 while there is no game won by the random player from 1,000,000 games.

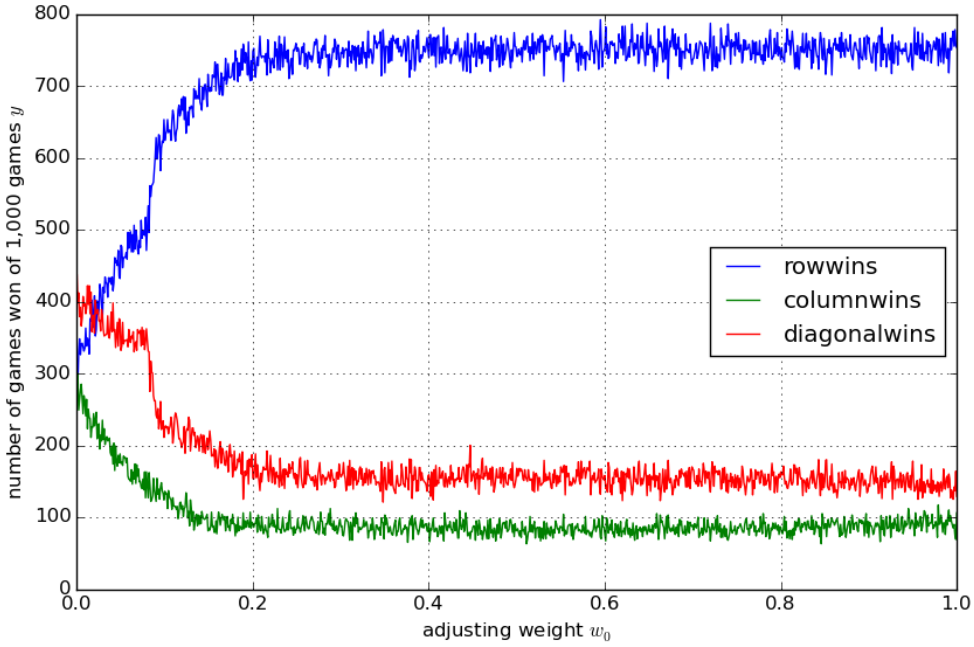

Figure 6.10: Result when vary row-win weight w_0

6.3.4.2 C - prefers to win by column

When adapting for wins by column, w_0 and w_2 are fixed at 0 and $w_3 = 1.0$ The result of varying w_1 from 0 to 1.0 with interval of 0.001 is shown in figure 6.11. Number of games won by MCTS NPC is 98.88% with 1.12% draws.

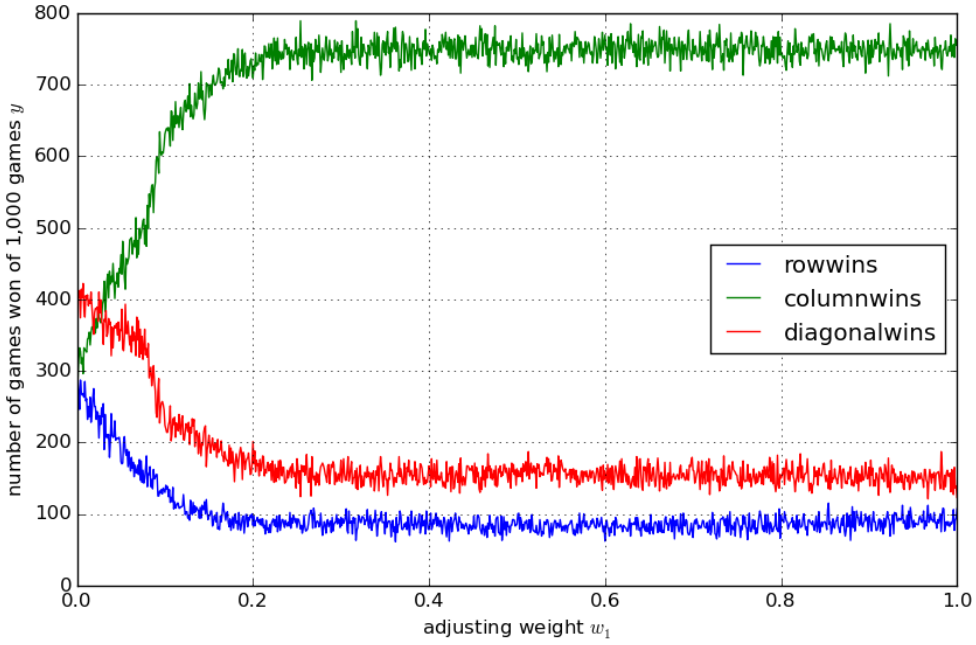

Figure 6.11: Result when vary column-win weight w_1

6.3.4.3 D - prefers to win by diagonal

The result of adapting to preference **D** is shown in figure 6.12. $w_0 and w_1$ are fixed at 0.0 and w_3 is 1.0. The statistics of wins and draws are the same as above but the adapting speed seems to be slower than the other two.

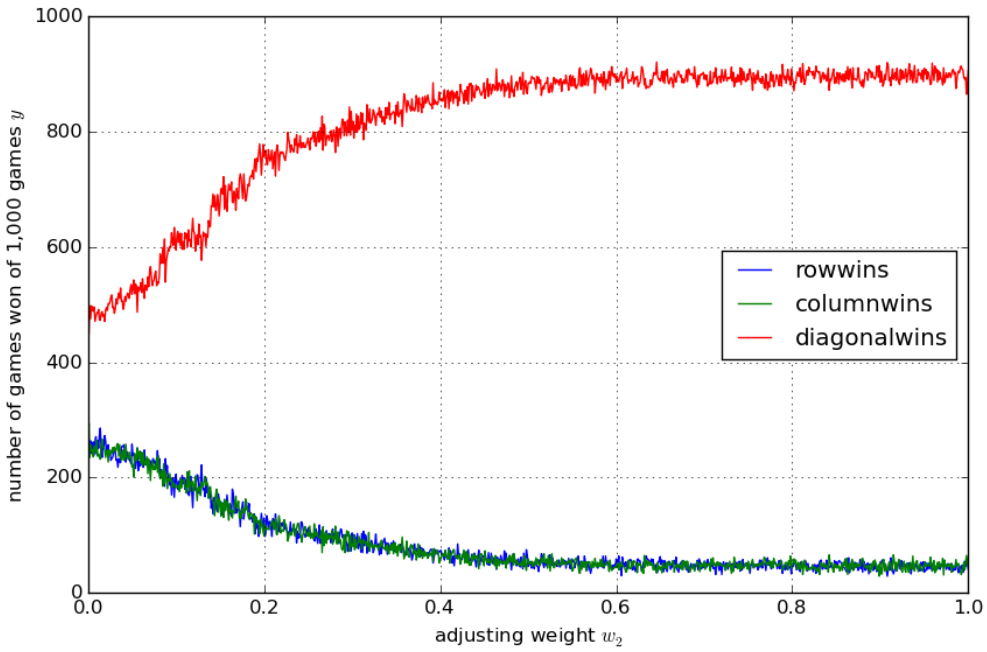

Figure 6.12: Result when vary diagonal-win weight w_2

6.4 Adapting MCTS NPC for Ms PacMan

This section applies the methodology in previous section to create adaptive NPC for Ms PacMan. The new NPC will be compared to the BlendingGhosts from chapter which relies on existing NPC models to give responding action.

Both ghosts teams uses a selected optimal user experience ranker 20-87-86-80-83-66-25. This means that the reference PacMan controller is NearestPillPacMan with entry ID 20. The remaining numbers represent ghosts team entry ID from each user experience level sequentially.

The NPCs are modified to accept the target level as an attribute. There are 6 possible user experience levels from 1 to 6. For each number of the levels, the number is assigned to the NPC's target level and the NPC runs against the reference PacMan controller to generate 20 distinct games.

Each NPC has 120 game data; 20 game data per level. RRF data are extracted from these games and evaluated using the selected ranker. The result of the BlendingGhosts is shown in table 6.5.

Group	1	2	3	4	5	6	Rank	Weighted	Max
1	0	14	0	2	3	1	1	2.85	2
2	0	15	1	0	3	1	2	2.70	2
3	0	3	1	9	6	1	3	4.05	4
4	0	8	1	7	2	2	4	3.45	2
5	0	4	3	1	8	4	5	4.25	5
6	0	2	1	4	1	12	6	5.00	6
ρ							1.00	0.89	0.82
p value							0.00	0.02	0.05
τ							1.00	0.73	0.75
p value							0.00	0.04	0.04

Table 6.5: User Experience Ranking Performance using BlendingGhosts

6.4.1 State Evaluation

The MCTS NPC for Ms PacMan uses algorithm 15 to search for best response action. RRF vector, x, is calculated during the simulation process. At the end of each simulation, the algorithm uses user experience ranker to calculate the rank for the vector before assigning the game state value. The formula is equation 6.5.

$$v = 1 - \frac{|t - t_e|}{6} \tag{6.5}$$

t is the target rank and t_e is the rank evaluated by the ranker. The selected ranker is the improved [Keerthi et al., 2001] sequential minimal optimization (SMO) [Platt, 1998] algorithm for training a support vector classifier. The algorithm uses pairwise coupling classification [Hastie and Tibshirani, 1998] internally on multi-class problems.

6.4.2 Decision Time Constraint

The pacman-vs-ghosts competition allows 40 milliseconds for entries to response with an action on each frame. However, the ghosts team entry does not have to search for the action every frame as there are often more frames with only one possible action to choose from. The game only expects response from the ghost team if any of the ghosts is at cross-section or a random threshold is reached. Therefore, the ghost team can reserve the time slot to plan ahead of time.

Statistics on the game dataset are shown in table 6.6. This shows that appropriate decision time is 284 milliseconds if the ghosts team would utilize the time slot efficiently. The game dataset is the same one from chapter 4 where there are 54,400 games from 16 pacman entries and 17 ghosts team entries.

PacMan Entries	ID	Score	FPG	R-FPG	Time
NearestPill	20	1,779.45	548.82	53.97	406.74
xsl11	27	5,135.57	1,619.79	207.30	312.55
PhantomMenace	28	14,865.58	4,602.89	717.30	256.68
brucetong	60	789.35	587.22	67.61	347.40
mcharles	64	3,080.09	780.32	89.90	347.20
Ant_Bot	67	490.19	305.33	38.25	319.31
Zekna	76	4,963.05	4,076.73	510.28	319.57
hacklash	78	373.11	244.80	25.16	389.14
Spooks	80	16,477.57	7,857.15	1,299.02	241.94
ICEpAmbush_CIG11	82	597.61	557.00	65.96	337.76
rcpinto	83	3,769.83	1,393.61	171.30	325.42
schrum2	88	460.36	255.08	27.12	376.28
CERRLA	89	2,298.24	724.70	76.67	378.09
emgallar	90	511.74	281.04	35.01	321.11
RandomNonRev	91	694.50	381.68	39.31	388.42
garner	92	630.68	530.03	58.64	361.54
		3,557.31	1,546.64	217.68	284.21

Table 6.6: Statistics of game dataset; FPG (frames per game), R-FGP (require response frames per game), Time (planning time in milliseconds)

The decision time constraint pushes maximum simulations S_x and maximum rollouts R_x down to minimal. The possible combination is $S_x = 50$ and $R_x = 7$ with average planning time 273.32 ms. The value of maximum rollout gives about 50 frames look-ahead.

6.5 Result

The performance result of the MCTS NPC is in table 6.7. The correlation is weaker when compare to the result of BlendingGhosts from table 6.5. This mainly due to the limitation of the decision time which severely reduces the exploration and exploitation of the game tree. With low number of simulations and low number of rollouts, the MCTS algorithms performance should be close to random.

However, the result in table 6.7 shows some correlation to the target ranking with reasonable number of true positives. The main cause of this problem is the game implementation for this thesis. While the game on the competition provides pre-calculated path distances for fast lookup, the game version used in this thesis is not. The path distance is calculated every time it is required. The reference pacman entry in MCTS, NearestPillPacMan, calls multiple times on one game frame.

Group	1	2	3	4	5	6	Rank	Weighted	Max
1	3	4	4	0	7	2	1	3.50	5
2	3	1	0	4	10	2	2	4.15	5
3	1	3	7	4	2	3	3	3.60	3
4	5	0	4	4	1	6	4	3.70	6
5	0	2	3	6	5	4	5	4.30	4
6	4	5	1	0	1	9	6	3.80	6
ρ							1.00	0.54	0.29
p value							0.00	0.26	0.57
τ							1.00	0.46	0.21
p value							0.00	0.18	0.54

Table 6.7: User Experience Ranking Performance using BlendingGhosts

6.6 Conclusion

The first two optimising problems in the first two sections have small search space. They post challenges to optimising algorithms as random search usually performs well and the problems have multiple local optima. However, most optimisers are found to perform reasonably well on both problems.

It should be interesting to see the result in other games with much bigger search space. Video games with many players and a number of NPCs implementation should also provide a great testing platform.

Finally, the study in this chapter shows that evolutionary strategy optimisers can help in the process of creating NPCs even when no reference NPCs are available. Alternately, the optimisation can assists in finding optimal condition for NPCs.

```
Data: s, r_i, ω, φ_p, φ_g, f(), MAXGEN
Result: best particle g
swarm = { g, p_i };
for i = 1, .., s do
    p_i.x = U(r_i);
    p_i.v = U(−r_i, r_i);
    p_i.b = p_i.x;
end
g = best p_i;
record generation z = 1;
if g = global optimum then
    return g;
end
while g not optimal and not MAXGEN do
    updateParticle(p_i.v, p_i.x);
    evaluate(p_i.x);
    for i = 1, .., s do
        if f(p_i.x) > f(p_i.b) then
            p_i.b = p_i.x;
            if f(p_i.b) > f(g) then
                g = p_i.b
            end
        end
    end
    record generation ++ z;
    if g = global optimum then
        return g;
    end
end
```

Algorithm 14: Rolling Discrete PSO: RDPSO

Data: s - game state, S_x - maximum simulation, R_x - maximum rollouts
Result: best action
root = Node(s);
while $i < S_x$ **do**
    node = select(root, 'UCT');
    expan(node);
    value = simulate(node.s, R_x);
    backpropagate(node, value);
end
return *selectBestChild(root).action*

Algorithm 15: Adapted MCTS Algorithm - search()

Chapter 7

Conclusion

The thesis focuses on creating adaptive NPCs to partially automate the task in responsibility of the AI developer team. The research assumes clear NPC objective and reference resources obtained from testers or on-line players. These are part of the game production cycle.

This thesis discusses the methodology for creating adaptive NPC using available NPC entries. The primary game for the study is Ms PacMan with public resources from the pacmans versus ghosts competition. The study requires reliable source of information to draw practical conclusion. This chapter summarises the contribution of the thesis as well as discussing the limitation and the applications.

The contribution of this thesis expands from the study of relative region data on CIG11. The data is found to be reliable for agent identification. The thesis further analyses how this technique could be used to identify the NPC agents as well as the agent for the player. Thorough investigation on the technique is done by first generating extensive amount of game data from the testing agents. The

research contributes the extraction technique along with the developments and analyse techniques for the specific data type.

7.1 RRF

RRF is an acronym for Relative Region Feature. It is an information extraction technique mentioned previously. The thesis uses RRF to refer to both the technique and RRF data to refers to the extracted data using RRF. The detail of RRF is in 3.5.5.

The extracted data abstracts a game state using only the locations of the game NPCs relative to the location of the player. For Ms PacMan, the technique reduces the size of the game data by 98.18%.

This technique applies directly to predator-prey games where proximity of the agents play important role in the game. Because RRF acts as game data compression, it should be attractive to complex games with multiple NPCs. The RRF dataset extract from those complex games should be at a magnitude smaller and easier to analyse.

It is important to note that the technique required pre-validate when applying to new games. Another drawback for the technique is that size determination to create RRF data. Games with huge arena are required to determine appropriate size of region. Region size determination could limit the resources available for further process.

7.2 Ranking

Chapter 3 shows that the RRF data can be used to create accurate classifier. The formal procedure for creating the classifier is given in chapter 4. The chapter gives analysis of the player experience as well as the development process to create the automatic ranker from the RRF data. The ranker consists of a classifier and a ranking scoring system. The evaluation of suitable classifiers are given in chapter 3. This chapter also analyse the effect of size of the relative regions. It has found that the small size produces more accurate classifiers overall. The most accurate classifier is SVM classifier. On RRF dataset, SVM classifiers outperforms ridge, k-nearest neighbours, stochastic gradient, and naive Bayes classifiers.

7.3 Evaluation

In addition to formalise the ranker generation, chapter 4 also formalises the procedure of creating adaptable NPCs using the result of optimal rankers. This form the basis for analysing player skill levels on chapter 5. The chapter employs both leave-one-out max and weighted ranking as the scoring system. It has found the drawback when scoring rankers with leave-one-out. The thesis proposes two solution to the problem; the re-calibration scoring system and the grouping skip scoring system. The re-calibration scoring system provides better scoring system for the ranker when using leave-one-out technique. However, re-calibration is not requires with the grouping skip scoring system. The grouping skip scoring system reserves one member of each group for evaluation. Unfortunately, the grouping has to be done manually and it requires a known ranking order. Each group must

contain two or more members. If there is a group with only one member, the scoring system should fall back to the re-calibration.

As shown in chapter 4 and chapter 5, the methodology should provide a way to systematically create efficient rankers to user experience and player skill. The methodology will work with any ranking list, however efficiency depends entirely on the extracted data. RRF dataset, which encapsulate the game data well, should also provide reliable rankers.

7.4 Optimisation

Chapter 6 presents the study of evolutionary algorithm's efficiency on finding global optimal ranker. Modified version of PSO and CMA-ES are also developed for this chapter for the rank variable. The discretised algorithms are RD-PSO and RD-CMA-ES. It involves rolling variable value up or down the ranking order. Therefore, the names are prepended with rolling discrete (RD).

RDPSO outperforms other testing algorithms including CMA-ES, sEA, $\mu + \lambda$ ES, μ, λ ES, and GA. RDPSO shows the promising result on finding optimal player skill rankers where it found the solution in every run with minimal amount of evaluation function calls. Conventional optimisers works on real number which are not directly applicable to discrete variables nor ranking orders. This thesis proposes the discretised version of the well-known optimisers to directly deal with such problem.

7.5 Future Work

The study of this thesis provides a good foundation for promising future work. Regarding how effective RRF data is in creating user experience rankers and creating reliable adaptive NPCs. The focus was to create effective data extraction method that can reliably identify both off-line and on-line optimisation. Currently, RRF data only contains the number of relative region numbers. However, it should generally to be more intuitive if RRF data contains more information. For example, physics-based simulation game might also consider enclose the vectors from player to NPCs. This includes other prey-predator games which can be viewed top-down.

RRF technique is also extensible to 3D game by using enclosing cube in place of the rectangular regions to cube regions. Each surrounding cube is assigned a number and RRF is a sequence of the cube numbers.

It should be interesting to see RRF technique performance on other prey-predator games. Furthermore, commercial games with high count of NPCs should benefit from RRF effectiveness in NPCs identification with minimal memory trace.

RDPSO is a promising optimiser specialised for problem whose variables' values are in preference-ordered set. Mathematical proof of convergence should be provided. More performance comparison should prove to popularise the technique.

Adaptive MCTS NPC performed really well when given enough resources. On limited resources, variation of MCTS to enhance to performance should be applicable including macro-action and history-lookup.

7.6 Summary

This thesis has presented an efficient methodology for creating adaptive NPCs in Ms PacMan game. The contribution includes the game data extraction technique, ranker creation from referencing ranking order, ranker scoring system, evaluation methodology, ordered-set discretised optimisers. Two type of objective-oriented adaptive NPCs are also proposed; agent-blending NPC and MCTS NPC.

Bibliography

Thomas Back, Hans paul Schwefel, and Fachbereich Informatik. Evolutionary computation: An overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29. IEEE Press, 1996. 13

T. Baeck, D.B. Fogel, and Z. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. Basic algorithms and operators. Taylor & Francis, 2000. ISBN 9780750306645. URL http://books.google.co.uk/books?id=4HMYCq9US78C. 106

E. Bethke. *Game Development and Production*. Wordware game developer's library. Wordware Pub., 2003. ISBN 9781556229510. URL http://books.google.co.uk/books?id=G7IknwEACAAJ. 10

Yngvi Bjrnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Trans. Comput. Intellig. and AI in Games*, 1(1):4–15, 2009. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig1.html#BjornssonF09. 30

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787. 11, 12

C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, march 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2186810. 30

Chi Wan Sung Bruce Kwong-Bun Tong, Chun Man Ma. A monte-carlo approach for the endgame of ms. pac-man. In *CIG*, pages 9–15, 2011. 22

Daniel Whitehouse et al. Cameron Browne, Edward J. Powley. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4 (1):1–43, 2012. 16, 18, 19, 20, 21

H.M. Chandler. *The Game Production Handbook*. Computer science series. Infinity Science Press, 2009. ISBN 9781934015407. URL http://books.google.co.uk/books?id=laiOw5WkdEcC. 10

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008. x, 17

Y. Collette, N. Hansen, G. Pujol, D. Salazar Aponte, and R. Le Riche. On object-oriented programming of optimizers – examples in scilab. In P. Breitkopf and R. F. Coelho, editors, *Multidisciplinary Design Optimization in Computational Mechanics*, chapter 14, pages 527–565. Wiley, 2010. in print. 106

M. Csikszentmihalyi. *Flow: The psychology of optimal experience*. Harper Perennial, 1991. 31

Belur V. Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *Mc Graw-Hill Computer Science Series*, pages 217–224, 1991. 47

Frédéric de Mesmay, Arpad Rimmel, Yevgen Voronenko, and Markus Püschel. Bandit-based optimization on graphs with application to library performance tuning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 729–736, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. URL http://doi.acm.org/10.1145/1553374.1553468. 20

Yamille Del Valle, Ganesh K Venayagamoorthy, Salman Mohagheghi, J-C Hernandez, and Ronald G Harley. Particle swarm optimization: basic concepts, variants and applications in power systems. *Evolutionary Computation, IEEE Transactions on*, 12(2):171–195, 2008. 25

RC Eberhart. *Swarm Intelligence*. _Morgan Kaufmann Publishers, 2001. 25

Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proc. 6th Int. Symp. Micro Machine and Human Science (MHS)*. Proc. 6th Int. Symp. Micro Machine and Human Science (MHS), October 1995. 25

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. 47

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012. 106

J. FÜRNKRANZ and E. HÜLLERMEIER. Pairwise preference learning and ranking. *Lecture notes in computer science*, pages 145–156, 2003. 14

J. Fürnkranz and E. Hüllermeier. Preference learning: An introduction. *Preference Learning*, 1, 2010. 14

Matthew L. Ginsberg. Gib: imperfect information in a computationally challenging game. *J. Artif. Int. Res.*, 14(1):303–358, 2001. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622394.1622405. 16

N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317, 1996. doi: 10.1109/ICEC.1996.542381. 13, 22

Nikolaus Hansen. The cma evolution strategy: A tutorial. 2011. 24

Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-370-0. URL http://dl.acm.org/citation.cfm?id=645514.657936. 23

Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998. 127

Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to

game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, pages 04–04, 2004. 8

Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. 2012. 2

Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998. 47

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3): 637–649, 2001. 127

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968. 25, 107

J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 5, pages 4104–4108 vol.5, Oct 1997. doi: 10.1109/ICSMC.1997.637339. 27

Raph Koster and Will Wright. *A Theory of Fun for Game Design*. Paraglyph Press, 2004. ISBN 1932111972. 9

J. E. Laird and Lent M. V. Human-level ai's killer application. In *Interactive computer games*, pages 1171–1178, 2000. 30

David Landau and Kurt Binder. *A Guide to Monte Carlo Simulations in Statis-*

tical Physics. Cambridge University Press, New York, NY, USA, 2005. ISBN 0521842387. 16

Joost Langeveld and Andries P Engelbrecht. A generic set-based particle swarm optimization algorithm. 27

Elena C Laskari, Konstantinos E Parsopoulos, and Michael N Vrahatis. Particle swarm optimization for integer programming. In *Computational Intelligence, Proceedings of the World on Congress on*, volume 2, pages 1582–1587. IEEE, 2002. 27

Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. Towards a generic method of evaluating game levels. 2013. 2

S Lucas and T Runarsson. Preference learning for move prediction and evaluation function approximation in othello. 16

Simon M. Lucas. Ms pac-man competition. *SIGEVOlution*, 2(4):37–38, December 2007. ISSN 1931-8499. URL http://doi.acm.org/10.1145/1399962.1399969. 28, 34

S.M. Lucas, P. Rohlfshagen, and D. Perez. Towards more intelligent adaptive video game agents: A computational intelligence perspective. In *A Computational Intelligence Perspective*, Cagliari, Italy, 5 2012. 31

Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. 12

Francis Maes, David Lupien St-Pierre, and Damien Ernst. Monte carlo search algorithm discovery for one player games. *CoRR*, abs/1208.4692,

2012. URL http://dblp.uni-trier.de/db/journals/corr/corr1208.html#abs-1208-4692. 30

T. W. Malone. What makes computer games fun. *Byte*, 6:258–277, 1981. 31

M. McPartland and M. Gallagher. Reinforcement learning in first person shooter games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3 (1):43–56, March 2011. ISSN 1943-068X. doi: 10.1109/TCIAIG.2010.2100395. 30

Jean Mhat and Tristan Cazenave. Combining uct and nested monte carlo search for single-player general game playing. *IEEE Trans. Comput. Intellig. and AI in Games*, 2(4):271–277, 2010. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig2.html#MehatC10. 30

Yannakakis G. N. Game ai revisited. In *Game AI Revisited*, Cagliari, Italy, 5 2012. 31

Yannakakis G. N., Lun H. H., and Hallam J. Modeling children's entertainment in the playwre playground. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 134–141, Reno, USA, May 2006. 31

Kien Quang Nguyen and Ruck Thawonmas. Applying monte-carlo tree search to collaboratively controlling of a ghost team in ms pac-man. *International Games Innovation Conference*, 0:8–11, 2011. 22

Kien Quang Nguyen and Ruck Thawonmas. Monte carlo tree search for collaboration control of ghosts in ms. pac-man. *IEEE Trans. Comput. Intellig.*

and AI in Games, 5(1):57–68, 2013. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig5.html#NguyenT13. 30

Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994. 106

P. Sweetser and P. Wyeth. GameFlow: a model for evaluating player enjoyment in games. *ACM Computers in Entertainment*, 3(3), July 2005. 31

Quan-Ke Pan, M Fatih Tasgetiren, and Yun-Chia Liang. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35(9):2807–2839, 2008. 27

Matt Parker and Bobby D. Bryant. Neurovisual control in the quake ii environment. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):44–54, 2012. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig4.html#ParkerB12. 30

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12: 2825–2830, 2011. 47

Tom Pepels and Mark HM Winands. Enhancements for monte-carlo tree search in ms pac-man. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 265–272. IEEE, 2012. 22

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. URL http://research.microsoft.com/~jplatt/smo.html. 127

Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. URL http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza). 13

Gustavo Recio, Emilio Martn, Csar Estbanez, and Yago Sez. Antbot: Ant colonies for video games. *IEEE Trans. Comput. Intellig. and AI in Games*, 4 (4):295–308, 2012. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig4.html#RecioMES12. 30

Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001. 47

David Robles and Simon M. Lucas. A simple tree search method for playing ms. pac-man. In *Proceedings of the 5th international conference on Computational Intelligence and Games*, CIG'09, pages 249–255, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-4814-2. URL http://dl.acm.org/citation.cfm?id=1719293.1719338. 22

Spyridon Samothrakis, David Robles, and Simon M. Lucas. Fast approximate max-n monte carlo tree search for ms pac-man. *IEEE Trans. Comput. Intellig. and AI in Games*, 3(2):142–154, 2011. 22, 30

Jesse Schell. *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0-12-369496-5. 7

N. Shaker, G. N Yannakakis, and J. Togelius. Towards automatic personalized content generation for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press*, 2010. 30

Brian Sheppard. World-championship-caliber scrabble. *Artif. Intell.*, 134(1-2):241–275, 2002. ISSN 0004-3702. URL http://dx.doi.org/10.1016/S0004-3702(01)00166-7. 16

W. Sombat, P. Rohlfshagen, and S.M. Lucas. Evaluating the enjoyability of the ghosts in ms pac-man. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 379–387, Sept 2012a. doi: 10.1109/CIG.2012.6374180. 15

Wichit Sombat, Philipp Rohlfshagen, and Simon M Lucas. Evaluating the enjoyability of the ghosts in ms pac-man. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 379–387. IEEE, 2012b. 2, 44

W. M. Spears, K.a De Jong, T. Back, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Machine Learning: ECML-93 - Proc. of the European Conference on Machine Learning*, pages 442–459. Springer, Berlin, Heidelberg, 1993. 13

Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis. What is procedural content generation?: Mario on the borderline. In *Proceedings of*

the 2nd International Workshop on Procedural Content Generation in Games, page 3. ACM, 2011a. 10

Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3 (3):172–186, 2011b. 2

Julian Togelius, Noor Shaker, and Mark J. Nelson. Introduction. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015. 10

Frans Van den Bergh and Andries Petrus Engelbrecht. A study of particle swarm optimization particle trajectories. *Information sciences*, 176(8):937–971, 2006. 25

P. Vorderer, T. Hartmann, and C Klimmt. Explaining the enjoyment of playing video games: the role of competition. In D. Marinelli, editor, *ICEC conference proceedings*, Pittsburgh, 2003. Carnegie Mellon University Press. 31

Hao Wang, Yang Gao 0001, and Xingguo Chen. Rl-dot: A reinforcement learning npc team for playing domination games. *IEEE Trans. Comput. Intellig. and AI in Games*, 2(1):17–26, 2010. URL http://dblp.uni-trier.de/db/journals/tciaig/tciaig2.html#WangGC10. 30

Thomas Weise. Global optimization algorithms theory and application , 2008. 12

M. Wistuba, L. Schaefers, and M. Platzner. Comparison of bayesian move prediction systems for computer go. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 91–99, Sept 2012. doi: 10.1109/CIG.2012.6374143. 16

G. N Yannakakis. *AI in computer games: generating interesting interactive opponents by the use of evolutionary computation.* PhD thesis, University of Edinburgh, 2005. 2, 31, 56

George G Yin and HJ Kushner. *Stochastic approximation and recursive algorithms and applications.* Springer, 2003. 47