

ABSTRACT

This dissertation proposes a method to achieve an adaptive fighting game AI by recognize the opponent and adapt itself to the opponent. Our proposed AI will first recognize the opponent with the use of the expert system and the machine learning technique classification, and then adapt itself to the opponent by load the solution which pre-optimized by Genetic Algorithm for that specific opponent.

FightingICE platform is selected as our testbed, serval experiments are held to test GA AIs and our proposed method in terms of winning rate and average score. The results prove that parameterized rule-based system bot tuned by GA is effective for against rule-based system AI while it is less effective for real-time decision-making AI. In addition, the experiment result also shows that our proposed AI has a variety of strategies.

Keywords: Genetic Algorithms, Fighting Games, Player Recognition

ACKNOWLEDGMENT

I gratefully acknowledge the suggestions provided by my supervisor Andy Holyer.

I also owe a big thank you to my parents for the financial support.

Furthermore, I would like to thank all developers at FightingICE for providing this platform to test my proposed method and giving so many suggestions. In addition, I would also thank for Wesley Tansey for the Gaussian mutation code.

Table of Contents

Abstract	2
Acknowledgment.....	3
Introduction.....	8
1.1 Introduction.....	8
1.2 Aim and Objectives	8
1.3 Related work	8
1.3.1 Adaptive AI for Fighting Games.....	8
1.3.2 Player Recognition	9
1.3.3 Genetic Algorithm	9
Background Theories	10
2.1 Expert system.....	10
2.2 Data mining: Classification	10
2.2.1 How Classification Work	10
2.2.2 Evaluation Method for Classification.....	10
2.3 Genetic Algorithm.	11
2.3.1 Search Space	11
2.3.2 Fitness and Fitness Function	12
2.3.3 Crossover.....	12
2.3.4 Mutation.....	13
2.3.5 Selection	14
Specification & Design	16
3.1 Platform Choice.....	16
3.1.1 Basic information of FightingICE	16
3.1.2 FightingICE competition rules.....	16
3.1.3 Character Zen	17
3.2 Recognition System Design	17
3.2.1 Classification Tool Selection	17
3.2.2 Tuple Design for Strategy Extraction	18
3.2.3 Features Extraction	19
3.2.3.1 Tuple Design for Feature Extraction	19
3.2.3.2 Expert System Design for Feature Extraction	19
3.2.4 Data collection Design for Strategy and Features Extraction.....	19
3.2.5 How our System Recognize the Opponent.....	19
3.3 Genetic Algorithm	20
3.3.1 Parameter Selection for Rule-based AI	20
3.3.2 Encoding	20
3.3.3 Population Size	21
3.3.4 Fitness Function.....	21
3.3.5 Selection Method	21
3.3.6 Crossover method	22
3.3.7 Mutation method	22
Implementation	23
4.1 Recognition.....	23
4.1.1 Strategy Extraction Evaluation.....	23

4.1.2	Features Extraction Evaluation.....	25
4.1.2.1	Number of Actions Through Period.....	25
4.1.2.2	Expert System Parameters Setting.....	26
4.1.3	Combined Recognition System	28
4.1.3.1	Recognition Performance at The End of The Round	28
4.1.3.2	Recognition Performance Through Game Timeline	29
4.1.3.3	Recognition Performance At 47 Seconds	30
4.2	Genetic Algorithm	30
4.2.1	Experiment on Mutation Method.....	31
4.2.2	Experiment on Crossover Method.....	32
4.2.3	Experiment on GA V.S. Random Generation	32
Experiment Result.....		34
5.1	Choice of Opponent AI.....	34
5.2	Initial Validation of GA.....	34
5.2.1	Fitness	34
5.2.2	Time cost to build Good AI	37
5.2.3	Strategy Variety.....	37
5.3	Recognition Result Based on Different Format	39
5.4	REGA and GA bot Test Against Last Year Competition AI	40
Conclusion & future work.....		43
6.1	Conclusion	43
6.2	Future Works	43
References		44
Appendices A: Bitbucket Commit Evidence.....		47
Appendices B: Creation Time for Snorkel and KYDB		49
Appendices C: Source code of Batch File and Shell File		50

Nomenclature

AI: Artificial Intelligence.

KYDB: KeepYourDistanceBot which is one of the last year competition AIs.

GA: Genetic Algorithm.

GA bot: Genetic Algorithm bot which load specific solution that optimized by Genetic Algorithm.

REGA bot: Recognition Enabled Genetic Algorithm bot, which use a combined system to recognize the opponent, and then load solution that optimized by Genetic Algorithm.

List of Figures

Figure 1: A Classic GA Pseudocode	11
Figure 2: An Example of Search Space in GA	12
Figure 3: One Point Crossover	12
Figure 4: N Point Crossover (example shows N = 2)	13
Figure 5: Uniform Crossover	13
Figure 6: Bit Flip Mutation	13
Figure 7: Roulette Wheel Selection	15
Figure 8: Screenshot of basic format from Thunder01	20
Figure 9: Best Classifier Result for SE Tuple 1-5, Evaluated by Training data set	23
Figure 10: The Number of Correct Recognition using Weka	24
Figure 11: Screen Shot of Opponent Statistics Analysis Using Weka GUI	26
Figure 12: Recognition Accuracy through Game Timeline	29
Figure 13: Fitness Score of GA Train Bot Against KYDB	35
Figure 14: Fitness Score of GA Train Bot Against Snorkel	35
Figure 15: Fitness Score of GA Train Bot Against Triumph	36
Figure 16: Fitness Score of GA Train Bot Against IchibanChan	36
Figure 17: A, B and C are chromosomes that generated by the GA against IchibanChan, Triumph and Snorkel	38
Figure 18: A, B and C are Chromosomes that Generated by The GA Against KYDB	39
Figure 19: Average score of The Tested AI in A Match	40
Figure 20: Bitbuck KCLAIPProject Commit Evidence part 1	47
Figure 21: Bitbuck KCLAIPProject Commit Evidence part 2	47
Figure 22: Bitbuck KCLAIPProject Commit Evidence part 3	47
Figure 23: Bitbuck KCLAIPProject Commit Evidence part 4	48
Figure 24: Bitbuck randomgeneration Commit Evidence	48
Figure 25: Creation Time for Snorkel	49
Figure 26: Creation Time for KYDB	49

List of Tables

Table 1: Tuple design for Strategy Extraction	18
Table 2: Tuple design for Features Extraction	19
Table 3: Basic Layout of Skill Chromosome, Each Parameter Was Represented in String Value.....	21
Table 4: Basic Layout of Probability Chromosome, Each Parameter Was Represented in Double Value	21
Table 5: 10 Cross-Validation result for SE Tuple 5	24
Table 6: 10 Cross-Validation Result for FE Tuple 1-5.....	25
Table 7: Weka Recognition Result Based on FE Tuple 1-5.....	26
Table 8: Result of Player Preference Attack and its Corresponding Percentage	27
Table 9: Recognition Accuracy data for Expert System and Weka Random Tree Classifier	28
Table 10: Recognition Accuracy of Combined Recognition System at the End of Each Round	29
Table 11: Recognition Accuracy at 47 Seconds of The Round	30
Table 12: GA Parameters Setting Used in All Chapter 4.....	31
Table 13: Results of GA Mutation Method Performance	31
Table 14: Results of GA Crossover Method Performance	32
Table 15: Result of GA and Random Search Performance	32
Table 16: Win rank of 2016 Fighting Game AI Competition.....	34
Table 17: Time Cost for GA to Generate a Good GA Bot	37
Table 18: Recognition Result Based on New Format Compare with the Result for Data Collection Bot	39
Table 19: Details of Experiment Result for Random Action AI, REGA bot and GA bot Against KYDB, Snorkel, Triumph and IchibanChan.	41

CHAPTER 1

Introduction

1.1 Introduction

The fighting game such as Street Fighter and King of Fighters have been the favorites video games which mainly designed for human versus human but still allow the human player play against AI. However, those AIs are typically designed manually by experts with a fixed difficulty level. There is a limit amount of studies on adaptive fighting game AI until the release of the Fighting Game AI Competition that hosted by IEEE CIG, around 20 papers are published by researcher based on this competition, methods such as Monte-Carlo Tree Search (Yoshida et al., 2016), K-nearest neighbor (Yamamoto et al., 2014) and fuzzy control (Chu and Thawonmas, 2015), etc. are already tested on this platform, nevertheless, the studies that try to achieve adaptive AI by recognize opponent and then adapt itself to the opponent are even more rarely.

1.2 Aim and Objectives

The aim of this study is to evaluate a new approach for adaptive AI that first identify the opponent by using an expert system and classification, then adapt itself with the solution optimized by Genetic Algorithm that specifically designed for that predicted opponent. Our proposed method perfectly suit the definition of a better AI given by Ricciardi and Thill (2008) "A better AI would have to be able to adapt to the player and model his or her actions, in order to predict and react to them."

To achieve the aim of this project, several steps are required:

1. Understand the use of expert system in recognition, design and implement the rules.
2. Feature extraction and strategy extraction will be taken to design datasets, experiments will hold to test which tuple for which classifier have highest accuracy, precision and recall.
3. Experiments will hold to test at which second this recognition system have highest accuracy to identify the opponent player during the game.
4. The parameters that we want to tune for our fighting game AI should be encoded to chromosome.
5. Experiments on the Genetic Algorithm's operators such as selection method, crossover operator and crossover rate, mutation operator and mutation rate will be carried out and results analysed.
6. Running the GA to optimize the solution against AIs that joined the last year Fighting Game AI Competition.

1.3 Related work

1.3.1 Adaptive AI for Fighting Games

- Sato et al. (2015) proposed an adaptive AI combined 3 rule-based AIs with a SW-UCB algorithm, the SW-UCB algorithm is used to switch the most suitable rule-based AI during the game. The evaluation performance shows that this method outperformed each of the rule-based AI that combined in its AI, and slightly improved the performance against an online learning player.
- Yoshida et al. (2016) applied Monte-Carlo Tree Search a fighting game AI in FightingICE, they conclude that MCTS is an effective mechanism for game AI.
- Kanetsuki et al (2015) developed a combination of Evolution Strategy and Fuzzy Control to improve the performance of Dynamic Scripting, the proposed AI outperforms the original AI.

- Nakagawa et al. (2015) demonstrated an AI that combine the k-nearest neighbour algorithm and a substring tree structure, experiment result proves that such method improves the prediction accuracy and performance of fighting game AI.
- Yamamoto et al. (2014) presented an AI that use K-nearest neighbour algorithm to predict opponent's attack action and a game simulator to deduce a countermeasure action, result shows the proposed AI works effectively against the top three AI in 2013 Fighting Game AI competition.
- Chu and Thawonmas (2015) proposed a fuzzy control to overcome the "cold start" of Yamamoto et al. (2014) kNN and game simulator AI. Result shows it effectively overcome the "cold start".
- Nakagawa et al. (2014) tested a method for online adjustment based on k-nearest neighbour algorithm to predict opponent's action and use a game simulator to determine an action, they reach the conclusion that this method can adjust the AI's strength but care must be taken in choosing the value of τ .
- Asayama et al. (2015) added a linear extrapolation and k-nearest neighbour algorithm to a rule-based AI enable it to predict opponent's position and action, it shows that the test result is better than the original rule-based AI.
- Park and Kim (2014) proposed an automatic policy learning method for fighting game AI, it use UCB1 to select highest reward action in history data, result shows this data0driven learning AI has a better performance than other learning-based AIs.
- Arellano et al. (2016) proposed a character generation approach for the M.U.G.E.N fighting game engine by Genetic Programming. The result shows GP can create a wide diversity of players with different strategies skills and no prior knowledge is needed to code the strategies for the AI character.

1.3.2 Player Recognition

In the literature, there are few examples of player recognition, only one studies form Ricciardi and Thill (2008) that investigated the player recognition in fighting games, they using standard MDP algorithm for player recognition, the result shows positive for static AIs, but negative for human players. What's more, there is only two other studies research on player recognition, Cho et al. (2013) using the algorithm form Weka API to predict the strategy of players and decide the change of build orders. The result on the public reply file shows that it can predict opponent's strategy accurately and increases the chance of winning in the game. Ballinger et al. (2015) use J48 decision tree in Weka to identify player and predict its next action, the results show it can identify a specific player out of 41 players with 75% of the time and can identify a player within 9 minutes of the game, it can predict what opponent will choose to do next 81% of the time.

1.3.3 Genetic Algorithm

There are serval studies use GA to create variety strategies AI with a good difficulty level. Byrne et al. (2010) demonstrated the Genetic Algorithm to optimize the attack behavior for the bot in a game called Toribash where the results prove that the GA was capable of evolving behavior for the AI. Ikeda et al. (2012) use GA to create AI have suitable strength for the human player while having various strategies and avoiding unnatural moves. Their result shows there are no unnatural moves, strength control is successfully but not sufficient for weaker players. The strategies used by AIs are variety.

GA is also tested on the other field such as first person shooter game, Cole et al. (2004) present an efficient method for using Genetic Algorithm to evolve asset of parameters First-Person Shooter Bots, their result shows the bots tuned by GA play as well as the bots tuned by a human expert.

CHAPTER 2

Background Theories

2.1 Expert system.

Grosan and Abraham (2011) states that “Rule-based systems (also known as expert systems) are the simplest form of artificial intelligence. it uses rules as the knowledge representation for knowledge coded into the system”. The implementation of the expert system is as simple as IF-Then-ELSE rules where the success of any expert system depends on the quality, completeness and the accuracy of the information contains in the rules. (Tutorialspoint, 2017)

There are two methods that are popular implemented to recommend a solution. The first one is forward chaining, which extracts data by a set of rules until a conclusion is reached. For example, it can be used to predict a name based on a set of certain attributes, thus, Forward chaining is used for our REGA bot to conclude opponent’s name. Where the second method backward chaining is the reverse work of the forward chaining, which tries to find out which rules trigger the conclusion in the past.

2.2 Data mining: Classification

The data mining technique classification (supervised learning) is a data analysis task, where a model or classifier is constructed based on a training dataset to predict class labels where prediction is to predict values for a known class label instance. Before classification, a set of tuples need to collect to train the classifier, where tuples (samples, instances, example) can have many attributes, each tuple is belonging to a predefined class label. A set of tuples is called training data.

2.2.1 How Classification Work

Han et al. (2012) pointed out that there are two steps in data classification, consisting of a learning step and classification step. A classifier is built describing a predetermined set of data classes in the learning step, where a classification algorithm builds the classifier by analyzing or “learning from” a training set made up of database tuples and their associated class labels. In the second step, the model is used for classification, a test dataset is used to estimate the accuracy of the classifier, if its accuracy is acceptable then the model is used to classify new data.

2.2.2 Evaluation Method for Classification

- Training set: Training set evaluation means a classifier is built by the full data set and apply the same dataset as the test set for evaluation, which always returns a high accuracy while it may not perform well when evaluated by different test sets.
- Percentage split: “Percentage split, is to randomly split the observations into two sets, a normally larger training set and a normally smaller test sets.” (Chou, 2013) Result for this one is normally worse than training set evaluation as randomly split may leading test data set not representative. For instance, some classes might be represented with very few instance or even no instances at all.
- k-fold Cross-Validation: Han et al. (2012) states that in k-fold Cross-validation the data are randomly partitioned into k folds subset, and then for subset S1, it again

divided to k-fold, the first subset of S1 will use as test data the remaining is built as training data, for subset S2, the second subset will be used as test data, for subset Sk, the kth subset of Sk will be used as test data. Finally, the average of the n evaluation results is the overall performance using the cross-validation method. This method overcomes the lack of the class representative that Percentage split has, perform a more reliable result than training set evaluation. Therefore, k-fold cross-validation will be used in our case where k is equal to 10 as Han et al. (2012) recommend to use 10-fold Cross-Validation.

- Precision and Recall: This is widely used in classification to measure the quality of the classifier, Han et al. (2012) noted that Precision is a measure of exactness, whereas Recall is a measure of completeness. The Precision and Recall are calculated by the following formula:

$$\text{Precision} = \frac{TP}{TP+FP} \quad (1)$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{TP}{P} \quad (2)$$

2.3 Genetic Algorithm.

Sivanandam and Deepa (2008) notes that since Holland first built the Genetic Algorithm in 1975, GA is now regarded as a powerful tool for solving search and optimization problems. Kramer (2017) affirms that the representation of each potential solution, often called chromosome, plays an important role to determine the choice of the genetic operators, it can be a list of real value genes or binary alphabet genes depending on the problem. Figure 1 shows the pseudocode of a classic Genetic Algorithm. At the beginning, a set of a population which are potential solutions is initialized, Kramer (2017) suggests that those initialization solutions should randomly cover the whole solutions space or to model and incorporate expert knowledge. Figure 1 shows an example of a classic GA.

```

Initialize Population
Repeat
    Repeat
        Crossover
        Mutation
        Evaluate Fitness
    Until Population Complete
    Select Parent
Until Termination Condition

```

Figure 1: A Classic GA Pseudocode

2.3.1 Search Space

Sivanandam and Deepa (2008) mentions that the space of all feasible solutions is called search space. Each point in the search space represents one possible solution. Figure 2 shows an example of a classic search space problem, each point in the search space represents a possible solution. The goal of the GA is to find the global optimal solution. However, a limitation listed by Sivanandam and Deepa (2008) for GA is "GA is not good at identifying a local optimum". For instance, GA may think the local maximum shows in Figure 2 is the global maximum when the global maximum is somewhere else.

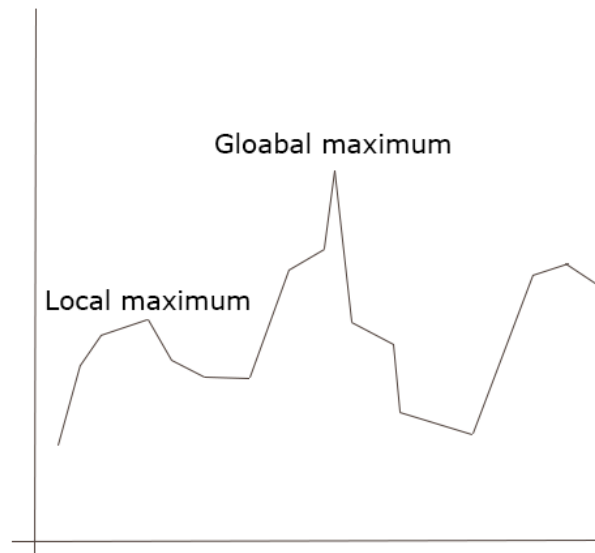


Figure 2: An Example of Search Space in GA

2.3.2 Fitness and Fitness Function

A fitness function is specific to the problem being solved, it evaluates the fitness of individual that generated by the GA, through either maximizing or minimizing the fitness values generation by generation, the individual with the global optimum could be found. Kramer (2017) highlights that the design of the fitness function is part of the modelling process of the whole optimization approach where the fitness function could have an influence on results and guide the search.

2.3.3 Crossover

Melanie (1999) stated that crossover is a main distinguishing feature of a GA, it describes the use of crossover is “randomly chooses a locus and exchanges the subsequence before and after that locus between two chromosomes to create two offspring.”

There are several crossover techniques, we list four of the popular one shows as follow:

- One point: This method splits up two solutions at 1 position that randomly selected by the algorithm and then alternately combine them to a new solution. The figure 3 shows how one point crossover works.

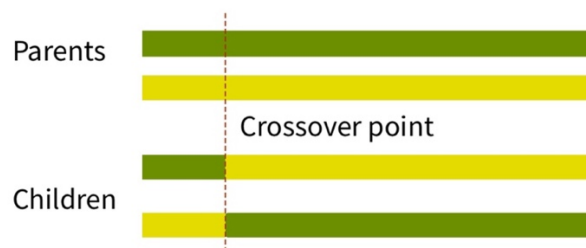


Figure 3: One Point Crossover

- N point: N point crossover is similar with one point crossover, the difference is the number of the crossover point is N rather than 1, it splits up two solutions randomly at

n positions and then combines them to a new solution. The Figure 4 shows how N point crossover works where N is equal to 2.

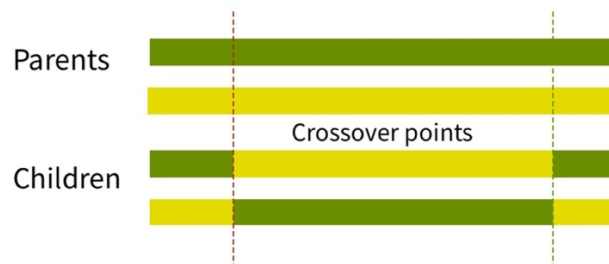


Figure 4: N Point Crossover (example shows N = 2)

- Arithmetic crossover: This method computes the arithmetic mean of two or more parents. For instance, if two parents are (4, 6, 8) and (6, 4, 6) the child will be (5, 5, 7).
- Uniform crossover: Uniform crossover uses a fixed mixing ratio such as 0.5 to randomly choose a bit from either of the parents, unlike N point crossover, each point between two genes have a predefined probability to become a crossover point at this generation. The example of uniform crossover shows below.

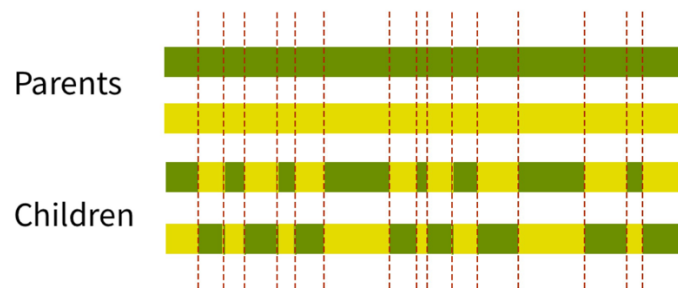


Figure 5: Uniform Crossover

2.3.4 Mutation

Kramer (2017) point out that mutation operators change a solution by disturbing them. The strength of disturbance is called mutation rate. Kramer (2017) also lists three main requirements for mutation operators. The first is reachability. There must be a minimum chance for each solution to reach every part of the search space to avoid stuck in a local optimal. The second is unbiasedness which means mutation operator should not design to search for a direction. The third is scalability. Each mutation operator should offer the degree of freedom that its strength is adaptable.

Four useful mutation techniques that are used in binary and value encoded chromosome is listed:

- Bit flip mutation: This mutation is commonly used in binary encoded GA, Figure 6 shows an example of bit flip mutation, it mutation one bit from 0 to 1, if the original bit is 1, it will become 0 if bit flip occurs on that bit.

```

101000110
  ↓
101010110

```

Figure 6: Bit Flip Mutation

- Swap Mutation: Swap mutation randomly select two genes and swap the positions of the gene, which will be used in our GA as the index of the actions in skill chromosome will lead to a strategy, this mutation can help to build different strategies for our GA individuals.
- Gaussian mutation: Kramer (2017) states that most processes in Gaussian mutation follow a Gaussian distribution, which can be calculated by formula:

$$X' = X + \sigma \times N(0,1) \quad (1)$$
 Where X is the solution that has been generated with crossover, N (0, 1) as a notation for a vector of Gaussian based noise. Variable σ is the mutation rate that scales the strength of the noise added. It simply adjusts a value slightly or no change between lower bound and upper bound. Kramer (2017) state the advantage of the Gaussian mutation is this mutation satisfied all mutation requirements.
- Uniform Mutation: Uniform mutation unlike Gaussian mutation only change slightly or make no change for gene value, it chooses a value between a predefined lower bound and upper bound randomly.

2.3.5 Selection

The third genetic operator is called selection; this process is based on the fitness values in the population. Four selection methods are listed which are commonly used in Genetic Algorithm.

- Elitist selection: This selection keeps the best individuals in each generation unchanged to the next generation, therefore, the best individuals will not be destroyed and the fitness level of next generation is at least remaining the same. The advantage of Elitism selection pointed out by Sharma (2014) is Elitism generates acceptable fitness population and that is why this selection is selected in our project.
- Roulette wheel: Holland (1992) developed the Schema Theorem that assumes individuals are selected based on their probabilities that are proportional to their fitness value. As the Figure 7 illustrates that all solutions form a wheel where the size of the slice depends on the fitness of the solution, a fixed point is given as the selection point, after wheel rotated, the slice that selected by the fixed point is chosen as the parent. The advantage of this selection method is all individuals have a chance to be selected, thus it is selected to be tested in our implementation section. Sharma (2014) claims that disadvantage of this selection is the lack of selection pressure. Further, Rahman et al. (2016) concludes that this method cannot handle minimization problem directly also negative fitness value which both are not in our case as we are a maximization problem and all fitness at least equal or higher than 0.

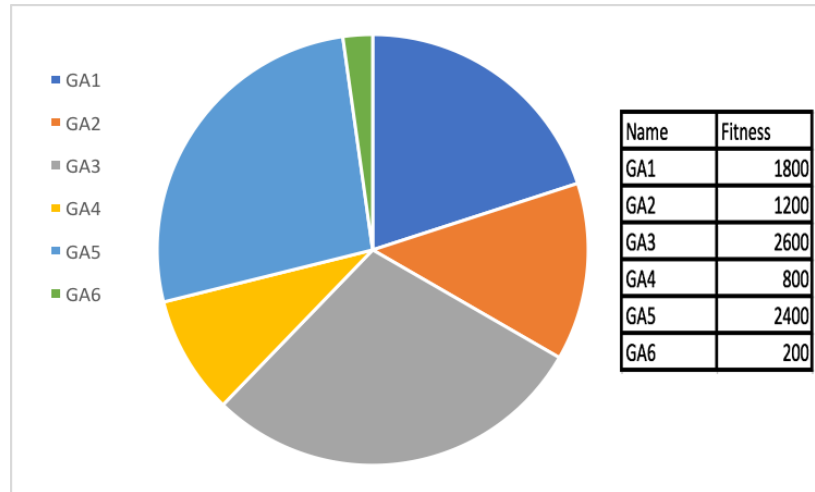


Figure 7: Roulette Wheel Selection

- Tournament selection: The tournament selection offers each solution a chance to survival even if that solution is less fit than other solutions, it selects N solutions out of all solutions, and choose the highest fitness solution within this N solutions as one parent. As Alabsi and Naoum (2012) indicate that the tournament selection does not guarantee to a reproduction of the best solution, thus, our GA will not use tournament selection as selection operator.
- Rank Selection: Obitko (1998) underlines that rank selection first ranks the population based on original fitness, and then assign fitness to chromosome based on the ranking, for instance, the worst chromosome will have fitness 1, then the second worst chromosome will have fitness 2 etc. and the best chromosome will have fitness N where N is the number of individuals in population.

CHAPTER 3

Specification & Design

Previous chapter gave a technical background in details. This chapter describes the platform choice, how to use expert system and Weka in opponent recognize, finally the parameters design of our GA in our project.

3.1 Platform Choice

FightingICE, a 2D fighting game, was chosen as the development platform as it is used for an international Fighting Game AI Competition, which hosted by IEEE Conference on Computational Intelligence in Games (IEEE CIG). In addition, as mentioned in the introduction, several AI techniques are already implemented based on this platform, thus, we believe Genetic Algorithm should be easy to implement by using this platform.

3.1.1 Basic information of FightingICE

A match in FightingICE consists of three 60-second rounds and 1 frame is set to 1/60 seconds where each AI must decide and input an action in 1 frame. "The platform has been designed to send delayed game information to the AIs which will use the same interface as that used by a human player to input their commands (ICE, 2017)" thus an AI in FightingICE will have same reaction time as human players.

The coordinate system is described by Asayama et al. (2015) "The position is determined in orthogonal coordinates where the origin is the top-left corner of the game window, i.e., the x-axis is the horizontally rightward direction, and the y-axis is the vertically downward direction."

The programming language Java and the Eclipse Software Development tool are selected since the competition launched, ICE (2017) provides a full detail of the get started guide which contains information including competition rules, install guide, AI classes information, etc, most importantly, the result and AI from last year's competition are available to download on the website.

3.1.2 FightingICE competition rules

The AI competition rules for the AI creation and Zen standard league are composed of those used in the Ms Pac-Man vs Ghosts League 2012 Competition, all rules list as follow:

1. The initial value of HP is 400 for both sides, and the HP will decrease when the character is hit.
2. After 60 seconds, the game will proceed to the next round, and the HP of both AIs will be reset to 400, the energy of both AI will be reset to 0.
3. The score for each round will be calculated by the formula:

$$Score = 1000 \times \frac{myHP}{myHP + oppHP} \quad (1)$$

myHP and oppHP are the HP of the player and the opponent, respectively. The maximum score of each round is 1000 if one player's HP reaches 0, another player whose HP is greater than 0 will give score 1000 and stop as a win.

4. The winner of a game is determined by the total score of the three rounds.
5. Memory usage is limited to 1GB, for those AI who exceed this limit, it will be disqualified.
6. Must not use multithread.
7. File reading/writing usage is limit to 50MB.

8. Each round has a 5 second initialization time and up to 60-second fighting time. Character's positions, energy and HPs will reset at end of each round.
9. Any conduct deemed fraudulent is prohibited, competitor should not doing anything listed as follow:
 - Disturbing opponent's controller.
 - Circumvent the competition's security framework.
 - Memory scanning.
 - Corrupt the file system.
 - Disrupt the ongoing competition.
 - Intentionally losing games.
 - Creating specific game states on purpose.

3.1.3 Character Zen

To make the experiments result fairly and equally, all experiments we hold and our AI and opponent AI only use the Character called Zen, it is one of 3 official characters that will be tested in 2017 FightingICE Competition.

There are 5 types of actions for each character:

- 1) Base: There are 4 base actions which are used in the normal status, such as stand, crouch, air and down.
- 2) Move: There are 6 movement actions to change the position of the character which is only allowed to perform while the character is on the ground.
- 3) Guard: There are 3 guard actions, which depending on the base the character is in, all those guard actions can be used to protect the character and reducing damage from the opponent.
- 4) Recovery: There are 12 recovery actions, those actions cannot be called, it will arise when the character is hit or performing a landing.
- 5) Skill: There are 31 skill actions, each skill action is an attack action which could be a simple punch, or an attack objects such as fire ball. The character must cancel current action to perform the next skill action.

3.2 Recognition System Design

This section describes the classification tool that is selected in our project. It also represents a strategy extraction and a feature extraction approaches that try to maximize the capture of game information and aspects that expose player-specific traits and strategy, so we can identify each player with a higher accuracy.

3.2.1 Classification Tool Selection

The classification algorithm used in our project will provide by Weka. Weka is a free, open source and powerful machine learning software from the University of Waikato, it contains statistical analysis tools, data pre-processing and regression techniques, many classification and clustering algorithms. Most importantly, the Weka and our AI are both written in Java, our AI can directly deploy and execute Weka functions without requiring additional compilation overhead.

In our implementation, all classification algorithms should be tested, only those algorithms that can build model within FightingICE initialization time and a reasonable good accuracy, precision, recall rate would be considered.

3.2.2 Tuple Design for Strategy Extraction

A strategy used by an AI can be considered as what decision an AI would choose in a certain situation. For instance, when the distance is short, choose heavy punch, when the distance is far and AI has high energy, it could generate a fireball.

To make our recognition more fairly, only data visible to the human player are collected, those including the following attributes:

- Distance X: A positive value shows the distance difference between our AI and opponent AI on x coordinate.

$$Distance\ X = |myPositionX - oppPositionX| \quad (1)$$

- Distance Y: A positive value shows the distance difference between our AI and opponent AI on y coordinate.

$$Distance\ Y = |myPositionY - oppPositionY| \quad (1)$$

- Distance X (\pm): A value shows the distance difference between our AI and opponent AI on x coordinate, as the game (0,0) starts at the left-hand side, it will have negative value if our AI on the left-hand side of opponent AI.

$$Distance\ X = myPositionX - oppPositionX \quad (1)$$

- Distance Y (\pm): A value shows the distance difference between our AI and opponent AI on Y coordinate, as the game (0,0) starts at the left-hand side, it will have negative value if our AI on the ground while opponent AI on the air.

$$Distance\ Y = myPositionY - oppPositionY \quad (1)$$

- My State type 1: Crouch, Air and Stand.
- Opponent State type 1: Crouch, Air and Stand.
- My State type 2: Air and Ground. Crouch and Stand is classed as Ground.
- Opponent State type 2: Air and Ground. Crouch and Stand is regarded as Ground.
- Opponent Action: All actions that Zen can perform.
- Opponent Skill Action: All skill actions that Zen can perform.
- Opponent name: The name of the opponent AI, this is a predefined value at each data collection stage, the opponent name is the class label for our database.

Above attributes are formed to tuples shows in the below Table

	Attribute 1	Attribute 2	Attribute 3	Attribute 4	Attribute 5	Class Name
SE Tuple 1	Distance X	Distance Y	Opponent Action	Not used	Not used	Opponent name
SE Tuple 2	Distance X	Distance Y	Opponent Skill Action	Not used	Not used	Opponent name
SE Tuple 3	Distance X (\pm)	Distance Y (\pm)	Opponent Attack Action	Not used	Not used	Opponent name
SE Tuple 4	Distance X (\pm)	Distance Y (\pm)	Opponent Attack Action	My State 2	Opponent State 2	Opponent name
SE Tuple 5	Distance X (\pm)	Distance Y (\pm)	Opponent Attack Action	My State 1	Opponent State 1	Opponent name

Table 1: Tuple design for Strategy Extraction

Five SE data sets corresponding to above five tuples will be collected and implemented in the next chapter, the best data set will be used in our recognition system.

3.2.3 Features Extraction

Features extraction is related to AIs' playing statics, it is designed to expose player-specific traits, so we can identify each player from other AIs based on their unique characteristics. There are two kinds of data we are interested, one is the number of attack actions AI made in a period, and another one is what actions would AI most likely to choose.

3.2.3.1 Tuple Design for Feature Extraction

The tuple designed for feature extraction shows from Table 2, by using those FE Tuples, Weka could make a class label prediction based on the corresponding attribute.

Name	Attribute 1	Class Name
FE Tuple 1	Opponent actions per one second	Opponent name
FE Tuple 2	Opponent actions per two seconds	Opponent name
FE Tuple 3	Opponent actions per three seconds	Opponent name
FE Tuple 4	Opponent actions per four seconds	Opponent name
FE Tuple 5	Opponent actions per five seconds	Opponent name

Table 2: Tuple design for Features Extraction

3.2.3.2 Expert System Design for Feature Extraction

Like human player has its own favorite action, most AIs, specially rule-based AI also has its own 'favorite' action, it can be regard as the action that AI most likely to perform, for instance, action A is performed 90 times out of 100 times of total actions the AI performed, or a group of actions it prefer to use than the rest of actions, for instance, action A, B and C performed about 70 times out of 100 times of total actions, those information form a AIs unique characteristics.

After the data collection session, any above two conditions will be used in our expert system, the actions name and corresponding occurred percentage are made up of our rules to determine the opponent's name.

3.2.4 Data collection Design for Strategy and Features Extraction

Due to the update of FightingICE 3.1, 2 of last year's competition AIs cannot run properly under certain conditions, to avoid error, we only collect the remaining 12 AIs.

Official provided "Random Action AI" is selected to collect those 12 AIs data. The Strategy Extraction Tuples and Features Extraction Tuples are collected for each opponent 100 matches (300 rounds) by using an ultimate HP mode (99999 HP).

As Weka GUI can show the action preference based on Strategy Extraction Tuples, no addition experiment is held to collect action preference data.

3.2.5 How our System Recognize the Opponent

Player recognition in terms of classification both for SE and FE tuples are the same where the designed recognition AI will make a prediction for the opponent name based on each frame's real-time instance data's attributes and update the corresponding name with its appeared number in this round, when system informed to conclude the decision, our recognition AI will regard the most appeared name as final recognition decision.

While the recognition decision made by the Expert system is easier, our expert system will assume the output fact as true if the face only contains one opponent name, otherwise regarded the face as a fault. However, by doing this would leading no opponent is

currently recognized, thus, we provide a combined system which let expert system to make the first decision, if it concludes only one name, that result will regard as our recognition decision, otherwise, the result conclude by the classification will used as our final decision, which is the opponent name.

3.3 Genetic Algorithm

All decisions taken to design the use of Genetic Algorithm for FightingICE and the implementation details of GA setting are shown in this section.

3.3.1 Parameter Selection for Rule-based AI

The two sets of parameters we try to optimize were: (1) Skill actions parameters and (2) Skill action active probability parameters. Since there are no movement actions are selected to optimize, a format shows from Figure 8 is chosen, which is the most commonly used format in last year competition's AIs, it will make a forward jump to get close to opponent, and the rules based shown in the Figure 8 deal with particular conditions, instead of calling function `mctsProcess` to select the best action, we will call GA solution to select the action we optimized.

```
if ((opp.energy >= 300) && ((my.hp - opp.hp) <= 300))
    cc.commandCall("FOR_JUMP_B B B");
else if (!my.state.equals(State.AIR) && !my.state.equals(State.DOWN)) {
    if ((distance > 150)) {
        cc.commandCall("FOR_JUMP");
    }
    else if (energy >= 300)
        cc.commandCall("STAND_D_DF_FC");
    else if ((distance > 100) && (energy >= 50))
        cc.commandCall("STAND_D_DB_BB");
    else if (opp.state.equals(State.AIR))
        cc.commandCall("STAND_F_D_DFA");
    else if (distance > 100)
        this.mctsProcessing();
    else
        this.mctsProcessing();
}
else if ((distance <= 150) && (my.state.equals(State.AIR) || my.state.equals(State.DOWN))
    && (((gameData.getStageXMax() - my.x) >= 200) || (xDifference > 0))
    && ((my.x >= 200) || xDifference < 0)) {
    if (energy >= 5)
        this.mctsProcessing();
    else
        this.mctsProcessing();
}
else
    this.mctsProcessing();
```

Figure 8: Screenshot of basic format from Thunder01

3.3.2 Encoding

Command library provided by FightingICE shows that an attack can be directly called by its name and the probability to call an action could be a double value, as such a real value like string or a double number is hard to encoded to binary, the value encoding recommended by Jaggi et al. (2013) is used for our chromosome.

There are two types of chromosomes, one is a string array containing 31 skills actions, and another one is a double array with size of 31 including the probability to occur the corresponding skill action.

As the cost power attack commonly cost more damage than a no-cost energy attack, those 14 cost energy attack skills are settled in the first half of the chromosome, then the

other 17 non-cost energy attack skills are in the second half of the chromosome. The layout of our designed chromosomes is shown below.

14 bits	17 bits
Cost Energy Skill Action Name	No-Cost Energy Skill Action Name

Table 3: Basic Layout of Skill Chromosome, Each Parameter Was Represented in String Value

14 bits	17 bits
Probability of Occur the Corresponding Cost Energy Skill Action	Probability of Occur the Corresponding No-Cost Energy Skill Action

Table 4: Basic Layout of Probability Chromosome, Each Parameter Was Represented in Double Value

3.3.3 Population Size

Mathworks (2017) state the large population size lead GA search the search space more thoroughly so that it has less chance stuck into the local optimal, but it also cost more time to run the GA. Due to the running time of the game engine, one match about 1 min in fast mode, population size is set as 10, the number of matches evaluates by the engine is set as 2 to return a more stable score result. The limitation of this population size design is it may not cover all possible solutions and could lead to a local optimal.

3.3.4 Fitness Function

As the original goal of this REGA bot is to join the 2017 FightingICE competition, the official score formula which used in the competition is selected as our fitness function to judge the fitness of the AI. The formula is shows follow:

$$Score = 1000 \times \frac{myHP}{myHP + oppHP} \quad (1)$$

However, the above formula has various limitations which may cause difficulty for our selection function. For instance, GA 1 win the match with 400 HP and KYDB lost with HP 0 while GA 2 win a match with HP 1 left and KYDB lost with 0 HP, by this formula, GA 1 and GA 2 will have the same score and regard as same strength, but it can be seen clearly that GA 1 perform a better strategy than GA 2, and GA 1 also have a better defense than GA 2. Another limitation is when all individuals in a generation have only 0 fitness value. This fitness function will be useless.

3.3.5 Selection Method

Elitism and Roulette Wheel Selection are used as selection method for our GA.

Due to the limitation described in the previous subsection, Elitism is selected to keep the top 2 individuals into next generation by considering three parameters: Fitness score, sum of my HP, sum of opponent HP. At first, Elitism will compare the fitness score, if the fitness score of two or more AIs are same, we consider the second parameter – AI's own HP, the more HP we have the better, and then ranked the AI by descending order, if the second parameter of two or more AIs are the same, we rank those AIs by comparing the opponent's HP, the lower the ranked higher. The parents' individuals will keep into the chromosome if none of the next generation's individual is better than these individuals, individual will changed immediately once next generation's individual is better.

Then the rest of individuals' parent will be selected by Roulette Wheel Selection. First, the sum of individuals' fitness is calculated, and then use a random generator to

randomly generate a number from minimal number up to the sum of all fitness score, and we add individual's fitness score one by one, once the score over the randomly generated number, we stop adding the score and the last individual we used to add the score will be selected as one parent, same process repeat to select another parent.

3.3.6 Crossover method

Uniform crossover is selected to be our crossover method as it can both work for string chromosome and double value chromosome. It is easy to implement the uniform operator in GA, simply use a random generator to generate a value within $[0, 1]$ at each index of gene on chromosome, if the value is less than 0.5, copy corresponding index of gene from parent A to the child chromosome, otherwise, that corresponding index of gene on parent B is cloned to the child chromosome.

Regarding crossover rate, Kumar and Gill (2010) examined that when the probability of crossover increase the execution time for job scheduling is decrease. Also, the diversity of the population increase, which means a higher crossover rate not only can increase the performance but also may lead AI out of the local optimal. As there is no recommended crossover rate, the crossover rate from 0.6 to 0.9 with step size 0.15 will be investigated in our implementation experiments to choose the best crossover rate for our GA.

3.3.7 Mutation method

The swap mutation is selected for skill action chromosome to deal with the string values, as the Java implement code by a top-down fashion, action is implemented sequentially, the first gene, which is the first skill action, typically implemented first, and once one action is activated, it will be input to the character and the program won't continue reading the following gene, as Arellano (2016) states that the skill action located at the end of the array would be less likely to be checked. Reducing the probability that skill action will be performed. This means the order of the action would be important, it could somehow create a strategy for the AI.

Uniform and Gaussian will be tested as mutation operator for our probability chromosome, the one with better performance will be selected in the final version GA. The implementation of both mutations is simple, for Uniform mutation, we randomly generate a probability value between $[0, 1]$ and keep as 0.001. In terms of Gaussian, we set the value for probability chromosome with the range between $[0, 1]$ also keep it as 0.001, the Variable σ is set as $1/6$ (Tansey, 2010). In Gaussian mutation, we randomly choose 1 or -1 to multiple the Variable value and then add it to the original probability value, if this new probability value is greater than 1, it will become to 1 and if this new probability value is less than 0, it will become to 0.

Unlike the binary encoding AI, Wright (1999) proves that "mutation rate for real-coded algorithm should be considerably higher than for binary-code algorithm", Haupt (2000) suggests that "the best mutation rate for Gas lies between 5 and 20%" while Obitko (1998) suggests that the binary encoding mutation rate always lies between 0.5% to 1%.

As our GA is a real-coded algorithm, we are testing the Gaussian and Uniform mutation with mutation rate range from 0.5 to 0.2 and step size 0.5, to test the performance of each setting.

CHAPTER 4

Implementation

The previous chapter described the recognition systems and the Genetic Algorithm design in our project. This section will describe the development process of the recognition system and Genetic Algorithm, the problem encountered during the development and steps taken to overcome them.

4.1 Recognition

Before to evaluate the dataset, the performance of ZeroR classifier will regard as our baseline, it simply predicts the majority class. Sayad (2017) recommend that zeroR is useful for determining a baseline performance as a benchmark for other classification methods. In addition, all classifiers that used by Weka means all classifiers that could build the model based on our data set, not every classifier that provided by Weka.

All the following experiments were performed on 2.7 GHz Intel Core i7 Processor, the model build time may differ for the different processor.

4.1.1 Strategy Extraction Evaluation

The first experiment is held to select the best data set out of five datasets by comparing the highest accuracy, which is evaluated by training data set with the use of all classifiers that provided by Weka. It can be discovered from Figure 9 that classifier build based on tuple 5 has identification performance as high as 78.69% which much higher than the baseline performance which only shows 13.19% accuracy, thus tuple 5 is selected for further use while others tuples are discarded.

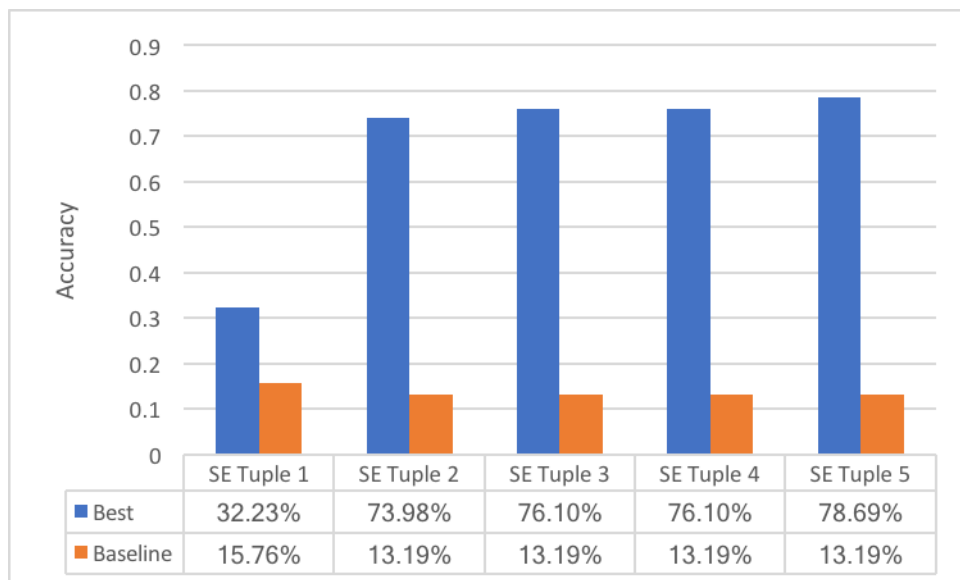


Figure 9: Best Classifier Result for SE Tuple 1-5, Evaluated by Training data set

All working classifiers with default parameter setting provided by Weka are used to build the model based on SE Tuple 5 data set and then evaluate by 10 Cross-Validation. Only Classifiers which can build the model within FightingICE initialization time with top 3

performance regarding accuracy, precision and recall rate are shown together with ZeroR as below Table:

Classifier Name	Model Build Time	Accuracy	Precision	Recall
Random Tree	1.77 seconds	48.3%	47.5%	48.3%
Hoeffding Tree	3.34 seconds	43.8%	42.9%	43.8%
OneR	0.12 seconds	40%	35.9%	40%
ZeroR	0.01 seconds	13.2%	1.7%	13.2%

Table 5: 10 Cross-Validation result for SE Tuple 5

The result shows from Table 5 present that Random Tree is the best classifier in terms of accuracy, precision and recall while it achieves a better performance in all measurements than Hoeffding Tree and OneR. All classifiers are better than ZeroR, which only has a base line 13.2% accuracy to identify a player out of 12 players.

Finally, 50 matches are tested for our recognition AI to against each of from last year competition's AI with Tuple 5 database to train Random Tree classifier at the beginning of the round, predicting and store the names and corresponding appearing times for each instance data at each frame and output a name at the end of the round.

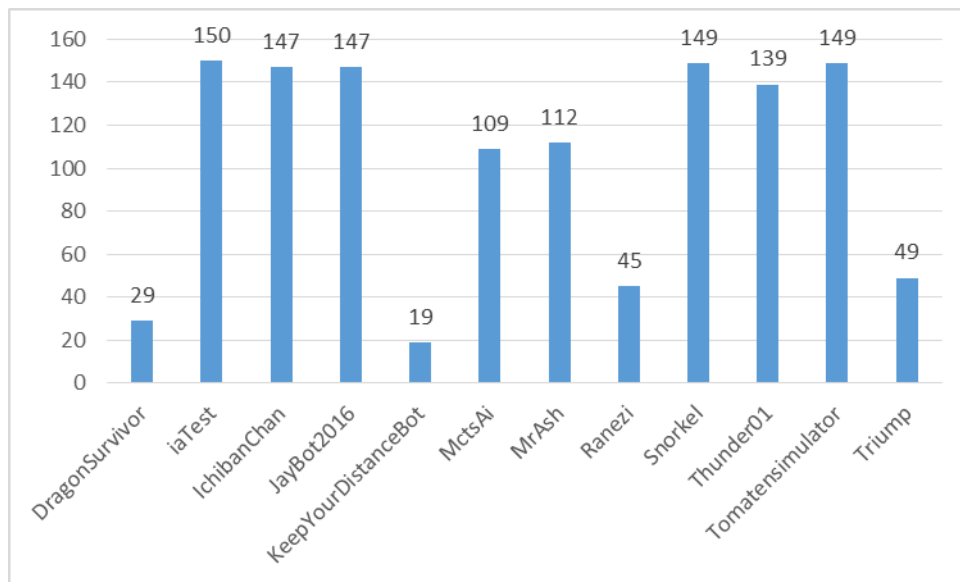


Figure 10: The Number of Correct Recognition using Weka

Figure 10 reports positive result that our Recognition AI achieved an overall 69.1% successful recognition rate, nearly 100% successfully recognition rate for real-time decision making AI such as Tomanter simulator, iaTest and IchibanChan. However, it shows negative for rule-based AI such as KYDB, DragonSurvivor, Ranezi and Triumph.

4.1.2 Features Extraction Evaluation

In this subsection, we will first implement the recognition accuracy of the number of actions through a period, then define the rules for expert system and evaluate it.

4.1.2.1 Number of Actions Through Period

Unlike the strategy extraction, feature extraction is a low-frequency analysis and more related to player's playing statistics, there is no need to select the best tuple, all FE tuples could present a positive result, so all five tuples are used for 10 cross validation and real-time data evaluation to test the performance.

ZeroR is tested before any other classifiers and it provides the baseline performance such accuracy with 8.33%, 0.7% precision and 8.33% for all five FE Tuples. Then all working classifiers with default parameter setting provided by Weka are used to build the model based on all data sets and then evaluate by 10 Cross-Validation. Only Classifiers which can build the model within FightingICE initialization time with top 3 performance regarding accuracy, precision and recall rate are shown:

Tuple Name	Classifier Name	Model Build Time	Accuracy	Precision	Recall
FE Tuple 1	Random Tree	0.35 seconds	23.8%	20.9%	23.8%
	J48	1.94 seconds	23.8%	20.9%	23.8%
	HoeffdingTree	0.94 seconds	23.8%	20.7%	23.8%
FE Tuple 2	Random Tree	0.15 seconds	25.5%	26.2%	25.5%
	OneR	0.03 seconds	25.5%	26.2%	25.5%
	Random Committee	1.73 seconds	25.5%	26.2%	25.5%
FE Tuple 3	J48	0.51 seconds	25.7%	26.7%	25.7%
	Bagging	0.98 seconds	25.6%	27.4%	25.7%
	AttributeSelected Classifier	0.86 seconds	25.7%	26.7%	25.7%
FE Tuple 4	Random Tree	0.06 seconds	25.5%	26.4%	25.5%
	Random Committee	0.62 seconds	25.5%	26.4%	25.5%
	Bagging	0.65 seconds	25.5%	27.4%	25.5%
FE Tuple 5	Random Tree	0.04 seconds	26.1%	27%	26.1%
	RandomSubSpace	0.58 seconds	26.1%	26.6%	26.1%
	Random Committee	0.46 second	26.1%	27%	26.1%

Table 6: 10 Cross-Validation Result for FE Tuple 1-5

It can be observed from Table 6 that none of one classifier is in the top 3 lists for all tuples, Random Tree and Random Committee are the only two classifiers that appear in top 3 lists for 4 times. Comparing those two classifiers, accuracy, precision and recall are the same for each case, but Random Tree can build model much faster than Random Committee. Thus, Random Tree is selected.

600 matches tested for all 12 AIs (50 per AI) against our recognition AI, which uses Random Tree classifier based on all five FE data sets with the same recognition decision rules applied in strategy extraction evaluation to identify targeted AI. Details of the results are listed in Table 7. Due to the positive performance for iaTest and DragonSurvivor, the average recognition correct rate reach above 35% while it only has about 25% if we remove the results for iaTest and DragonSurvivor. Thus, FE tuples are discarded and will not be used in our final recognition system.

Name	FE Tuple 1	FE Tuple 2	FE Tuple 3	FE Tuple 4	FE Tuple 5
DragonSurvivor	134	130	128	128	131
iaTest	150	150	150	149	149
IchibanChan	1	7	8	5	8
JayBot2016	64	99	24	61	44
KYDB	77	79	81	85	90
MctsAi	1	5	78	95	68
MrAsh	35	53	14	17	9
Ranezi	118	34	59	59	77
Snorkel	2	15	15	6	4
Thunder01	93	87	53	43	56
Tomatensimulator	0	14	49	61	61
Triump	0	0	4	17	7
Recognition correct rate per round	37.5%	37.4%	36.8%	40.3%	39.1%

Table 7: Weka Recognition Result Based on FE Tuple 1-5

4.1.2.2 Expert System Parameters Setting

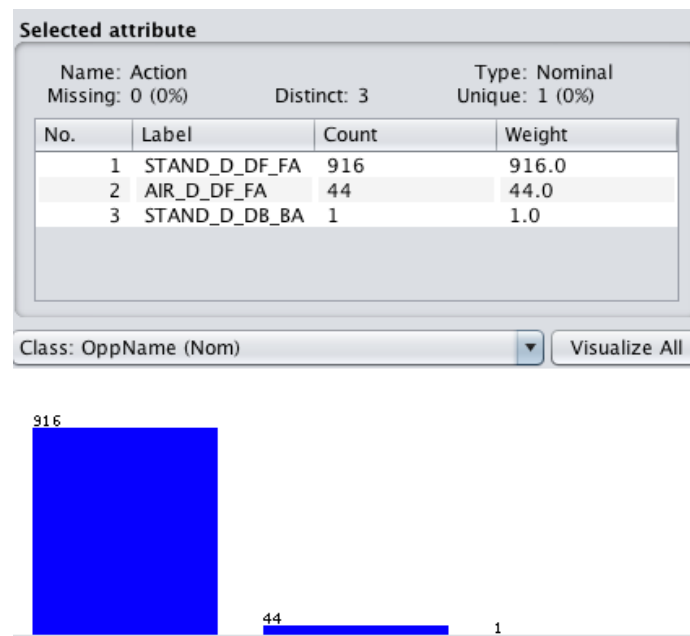


Figure 11: Screen Shot of Opponent Statistics Analysis Using Weka GUI

Weka GUI is applied to analysis the opponent statistics regarding skill actions, which shows as Figure 11, the higher column means that skill action is more likely to be chosen by the AI. It can be seen from Table 8 that each AI has difference preferable skill actions and the corresponding percentage is also different to each other.

All those attributes are used to form a set of rules to build our expert system to predict the name of the opponent, for instance, if opponent performs STAND_D_DF_FA out of all actions up to 96%, our expert system will conclude the decision that the opponent is KYDB. Values under the third column of Table 8 are manually decreased 5% to 10% when built our rules as all those values are average numbers.

AI Name	Preference Attack Actions	Performed Percentage for Corresponding Attack
KYDB	STAND_D_DF_FA	95.3%
DragonSurvivor	STAND_F_D_DFA STAND_D_DF_FA	36.2% 24.7%
laTest	THROW_A AIR_DB	30.7% 37.8%
IchibanChan	AIR_B STAND_B	23.3% 30.1%
JanBot2016	STAND_D_DB_BA	34.1%
MctsAi	STAND_FB STAND_D_DF_FA STAND_D_DF_FB STAND_D_DB_BA	11.7% 13.0% 9.2% 10.9%
MrAsh	AIR_B STAND_B	42.2% 22.0%
Ranezi	STAND_FB STAND_D_DF_FA STAND_D_DB_BA STAND_F_D_DFA	14.6% 10.3% 9.8% 9.3%
Snorkel	AIR_D_DF_FA	31.0%
Thunder01	STAND_F_D_DFA AIR_UB	13.0% 10.3%
Tomatensimulator	STAND_B STAND_FB STAND_D_DF_FB	21.0% 25.4% 20.6%
Triump	STAND_D_DF_FB CROUCH_B STAND_FB STAND_D_DF_FA	35.5% 12.0% 12.4% 12.5%

Table 8: Result of Player Preference Attack and its Corresponding Percentage

In this experiment, we test our expert system for each opponent (Table 9) 50 matches with predefined recognition rules that, expert system should return a true when the system only return the targeted opponent's name or a false for any other cases at the end of each round of the match.

Name	Expert System	Weka
DragonSurvivor	125	34
iaTest	150	150
IchibanChan	98	147
JayBot2016	0	82
KYDB	146	11
MctsAi	150	111
MrAsh	84	80
Ranezi	82	65
Snorkel	123	146
Thunder01	68	30
Tomatensimulator	90	148
Triump	70	46
Correct Recognition	1186	1050
Recognition Accuracy	66%	58%

Table 9: Recognition Accuracy data for Expert System and Weka Random Tree Classifier

It can be observed from Table 9 that for a specific player our expert system identification accuracy is 66%, comparing the performance with Weka using random tree classifier, it shows a slightly higher recognition accuracy in terms of Triump and Ranezi and a significant improvement for KYDB and DragonSurvivor.

4.1.3 Combined Recognition System

In this subsection, the combined recognition system is first evaluated to show the overall performance compared with our classification and expert system results, then evaluated to output the performance during the game timeline. Finally, if the result shows the timeline that the best performance is not at the end of the round, one additional will hold to test that the recognition performance at that specific second.

4.1.3.1 Recognition Performance at The End of The Round

The combined recognition system is tested to against each of last year competition's AI 50 matches and using the recognition rule described in the design section to output a recognized name at the end of each round.

We note from Table 10 that this combined recognition system could recognize an AI out of 12 AIs with 82.89% accuracy for each round which better than the expert system 69%, and much better than a random guess accuracy 8.3%. Therefore, a combined system is used to further test.

Name	Combine Recognition System
DragonSurvivor	130
iaTest	150
IchibanChan	149
JayBot2016	82
KeepYourDistanceBot	146
MctsAi	150
MrAsh	120
Ranezi	106
Snorkel	147
Thunder01	76
Tomatensimulator	148
Triump	88
Correct Recognition	1492
Recognition Accuracy	82.89%

Table 10: Recognition Accuracy of Combined Recognition System at the End of Each Round

4.1.3.2 Recognition Performance Through Game Timeline

Cho et al. (2013) present a research on strategy prediction performance during the game, it shows the prediction accuracy is variety through the game timeline, thus we applied an experiment to test the accuracy through the game timeline for our combined recognition system.

Our data collection bot is conducted in a round-robin fashion with five games for any other 12 AIs switching P1 and P2, following recognition rules defined in the previous subsection.

The Figure below shows that all three recognition systems reach peak accuracy at 47 seconds when it could successfully identify a player out of 12 players with an 83.3% accuracy for our combined recognition system, which is 5% higher than the accuracy at end of the round.

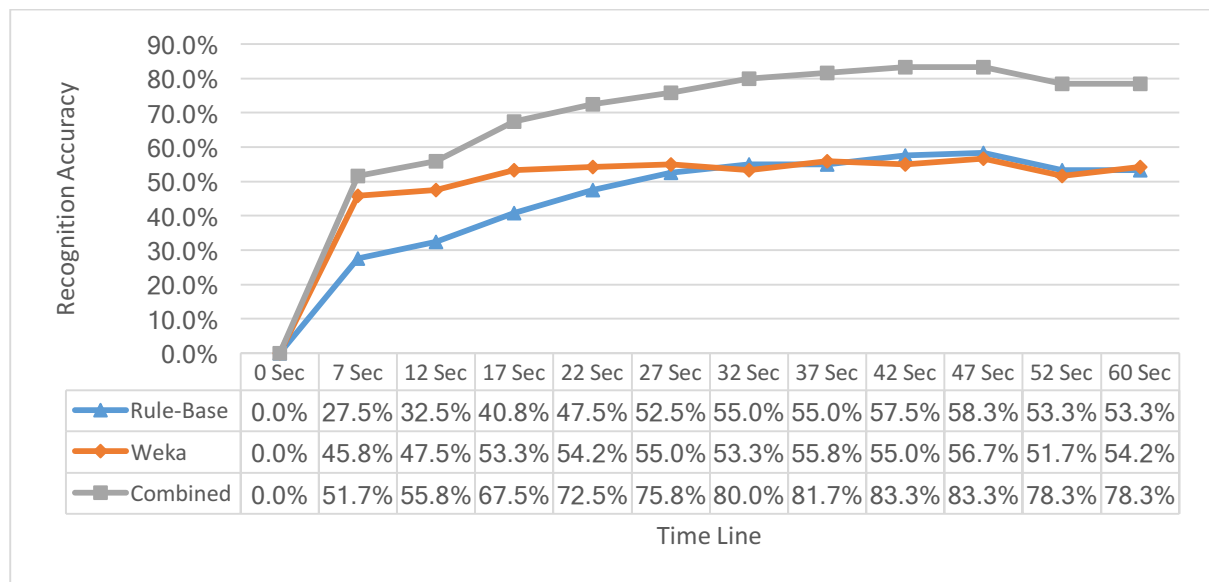


Figure 12: Recognition Accuracy through Game Timeline

4.1.3.3 Recognition Performance At 47 Seconds

To prove our recognition systems has a better result at 47 seconds than at the end of each round, one last recognition experiment is held to against 50 matches per AI that listed at Table 11 before implement the GA.

Name	47 seconds	End of the round
DragonSurvivor	131	130
iaTest	150	150
IchibanChan	143	149
JayBot2016	74	82
KYDB	146	146
MctsAi	150	150
MrAsh	135	120
Ranezi	117	106
Snorkel	148	147
Thunder01	95	76
Tomatensimulator	149	148
Triump	95	88
Recognition Accuracy	85.17%	82.89%

Table 11: Recognition Accuracy at 47 Seconds of The Round

Table 11 again prove that recognition performance at 47 seconds had a better result than at the end of the round.

We choose to let our combined recognition system to predict result at 47 seconds of each round with the following reasons.

- Higher accuracy: 85.17%>82.89%
- An official competition match only assigns 400 HP for each AI and once one AI with HP lower or equal than 0, that round will regard a loss and end. There is a better chance that AI predict a name at 47 seconds of the match than at the end of the match.
- The earlier we adapt our AI to the solution that optimized by GA, the higher points we could score, and a higher chance that we will win that match.

In conclusion, combined recognition system will be used for our REGA bot and it will make a recognition decision at 47 seconds of the first round.

4.2 Genetic Algorithm

Serval tests have been made to find the best Genetic algorithm parameters, all those GA tests follow the rules from the FightingICE competition:

- 60 second each round
- HP 400 for both AI
- $Score = 1000 \times \frac{myHP}{myHP + oppHP}$
- Once a player HP reach 0, another player's score regard as 1000

To make the GA perform faster, the fast mode provided by the FightingICE official is applied, "in this mode, the frame speed is not fixed to 60 FPS; the game proceeds to a next frame once the system gets inputs from both AIs. This mode might help you train your AI faster, but it will be not being used in the competition (ICE, 2017)".

All the GA experiments are using the GA setting from Table 12, as crossover rate, selection method and mutation method for probability chromosome and its mutation rate are tested in

those experiments, all those values are subject to change and will declare at the beginning of each experiment.

Generation	35
Score requirement	2000 points per match
Individuals	10
Matches attend for individuals for fitness function	2
Selection method	Elitism for top 2 AI, Roulette wheel selection for the rest individuals
Crossover method	Uniform
Mutation method for skill chromosome	Swap
Opponent tested	KYDB

Table 12: GA Parameters Setting Used in All Chapter 4

4.2.1 Experiment on Mutation Method

In this subsection, the crossover rate is set as 0.75, other GA parameters setting shows from Table 12.

It can be observed from Table 13 that all sets of GA setting complete the task successfully. What's more, the result also proves that Uniform mutation operator with mutation rate 0.15 is the best GA setting within all settings, as the number of generations costs to generate a good AI and the number of no improvement generations are both smallest, which means the GA keep improve nearly in each generation and can find a good solution with shorter time. Therefore, uniform mutation and mutation rate set as 0.15 will be used in chapter 5 to optimize the best GA solution for GA bot.

The second-best result is the Gaussian operator with mutation rate 0.05 which perform slightly worse results than Uniform with 0.15 mutation rate, except Run 1 it cost over 10 generation to generate a good GA AI, other 2 runs are both less than 5 generations to create the good AIs.

	Run 1		Run 2		Run 3	
Mutation Operator (mutation rate)	Total generation	No improvement generation	Total generation	No improvement generation	Total generation	No improvement generation
Uniform (0.05)	14	13	18	13	25	22
Uniform (0.1)	28	21	11	6	1	0
Uniform (0.15)	3	2	1	0	6	3
Uniform (0.2)	24	22	4	3	9	7
Gaussian (0.05)	10	6	3	2	2	1
Gaussian (0.1)	4	3	16	10	6	3
Gaussian (0.15)	4	2	34	32	2	1
Gaussian (0.2)	23	14	2	1	7	3

Table 13: Results of GA Mutation Method Performance

4.2.2 Experiment on Crossover Method

In addition to the GA parameters setting shows from Table 12, the mutation rate used for experiments on crossover method is set as 0.15.

Table 14 illustrates that crossover rate 75% is the most effective crossover rate out of three tested crossover rates, with this crossover rate, GA achieved to create a good GA AI against KYDB within 5 generations in all three runs while other two crossover rates shows an unstable performance. Consequently, crossover rate 75% is selected to use in chapter 5 to build best GA solutions.

Crossover Rate	Run 1		Run 2		Run 3	
	Total generation	No improvement generation	Total generation	No improvement generation	Total generation	No improvement generation
60%	2	1	6	3	32	28
75%	3	2	1	0	2	0
90%	15	11	2	0	4	2

Table 14: Results of GA Crossover Method Performance

4.2.3 Experiment on GA V.S. Random Generation

	Run 1	Run 2	Run 3
Mutation Operator (mutation rate)	Total generation	Total generation	Total generation
Uniform (0.05)	14	18	25
Uniform (0.1)	28	11	1
Uniform (0.15)	3	1	6
Uniform (0.2)	24	4	9
Gaussian (0.05)	10	3	2
Gaussian (0.1)	4	16	6
Gaussian (0.15)	4	34	2
Gaussian (0.2)	23	2	7
Random Generator	21	22	31

Table 15: Result of GA and Random Search Performance

As Sivanandam and Deepa (2008) points out that “Building a genetic algorithm, which performs no more than a random search happens more often than we can expect” the last experiment is held to check whether our GA perform better than a random search.

In addition to the GA setting shows from Table 12, we set the crossover rate as 75%. For equal comparison, same GA bots are used in random search, meanwhile the population size of random search is also set as 10, instead of using GA to generate the skill and probability chromosome, the random generator generates skill chromosome randomly select from the lists of skill actions and randomly generate a value from [0, 1] as probability value.

The result shows from Table 15 proves that our GA perform a better search than a random search for all GA settings. However, only the GA setting with Uniform mutation with 0.15 mutation rate and the Gaussian mutation with 0.05 mutation performed a stable search, both can search a good solution within 10 rounds while other GA settings are shown an unstable result.

CHAPTER 5

Experiment Result

5.1 Choice of Opponent AI

Game simulator is widely used in last year's competition AIs, which can simulate the progression of a fight for up to 1 second from the current time for the AI, it executing specified action for both players based on current game information, and return the outcome of the simulation either a minus value or superior value. In conclusion, this simulator class could analysis what would happen up to next one second, and help the AI to decide the action. Those AI from the rank 1 to rank 8, shows from Table 16, are using the simulator class while most of them also use machine learning algorithm; other AIs from rank 9 to the bottom are not using the game simulator class and most of them simply use the rule-based system. Due to the update of the FightingICE platform, the AI called Poring and BANZAI will not select in our experiment as it contains error when running the AI.

As our REGA Bot does not include a simulator class, GA will mainly select those top AIs without use of simulator class and a bottom AI that use the simulator class, details of those selected AIs are shown as follow:

- IchibanChan: uses rule-based strategy with the use of the game simulator class, the rule-based strategy based on distance difference on coordinate X between two players, distance to the most left and most right game stage, checking whether the enemy is in the air and energy.
- KeepYourDistanceBot: uses a rule-based system to keep the distance from the opponent and never engage in close combat, in addition, this AI only use the fireball to attack.
- Snorkel: Using UCB1 include reward algorithm to select an action that has highest UCB1 value, it assists with a rule-based system with defined rules to check the distance from the opponent and the energy of itself.
- Triumph: uses rule-based strategy considering variables such as distance and energy to define an attack or defence action. Use For_JUMP to get out of the corner and when opponent come closer.

Name	Win	Total match	Name	Win	Total Match
Thunder01	70	78	IchibanChan	39.5	78
MctsAi	63	78	KeepYourDistanceBot	26.5	78
iaTest	55	78	Poring	25.5	78
MrAsh	53	78	Snorkel	25	78
JayBot2016	47.5	78	Triumph	21.5	78
Tomatensimulator	45.5	78	DragonSurvivor	21	78
Ranezi	45	78	BANZAI	8	78

Table 16: Win rank of 2016 Fighting Game AI Competition

5.2 Initial Validation of GA

5.2.1 Fitness

Figure 13, 14, 15 and 16 shows the results of the experiments obtained with the best parameter that we discovered in the implementation stage (crossover rate 0.75 and the uniform operator with mutation rate 0.15). The number of the generation is set as unlimited

while the score requirement is set 2800 out of 3000, due to the limit time of project duration, we stop running the GA for Snorkel and IchibanChan when it shows no improvement in 15 generations.

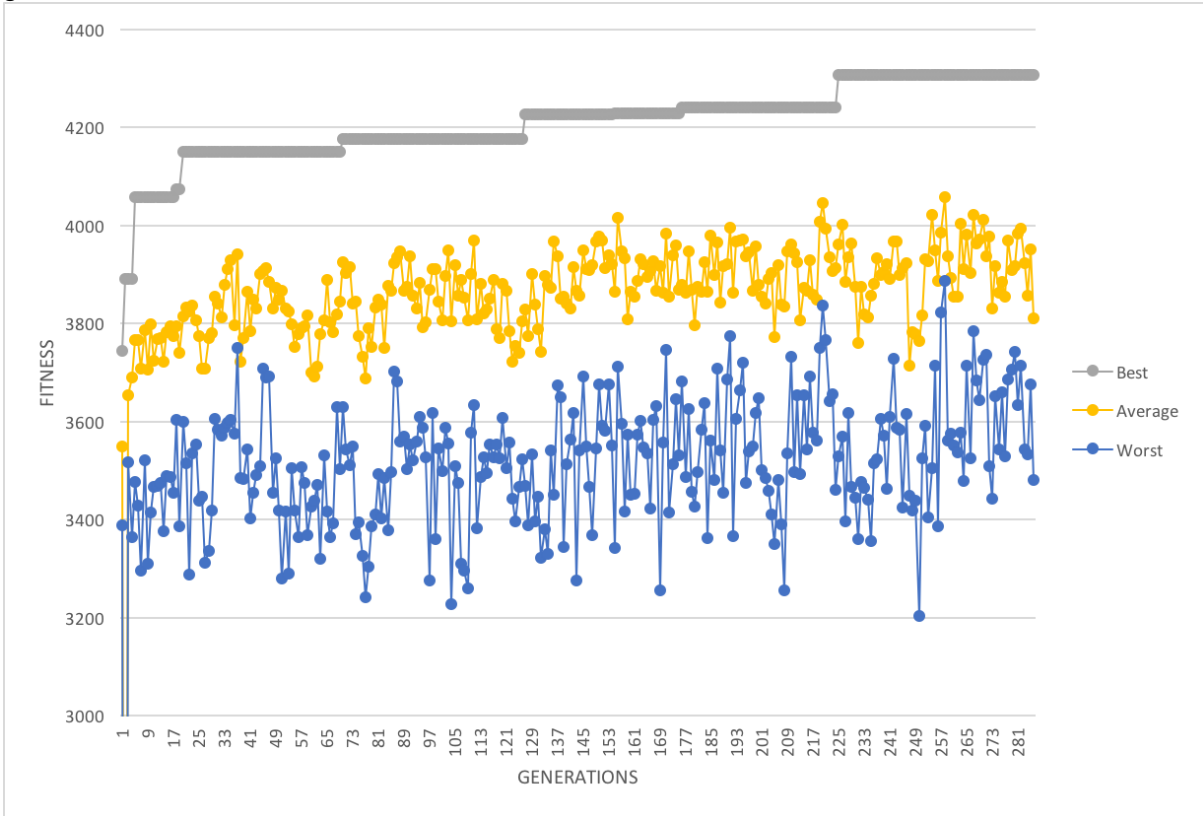


Figure 13: Fitness Score of GA Train Bot Against KYDB

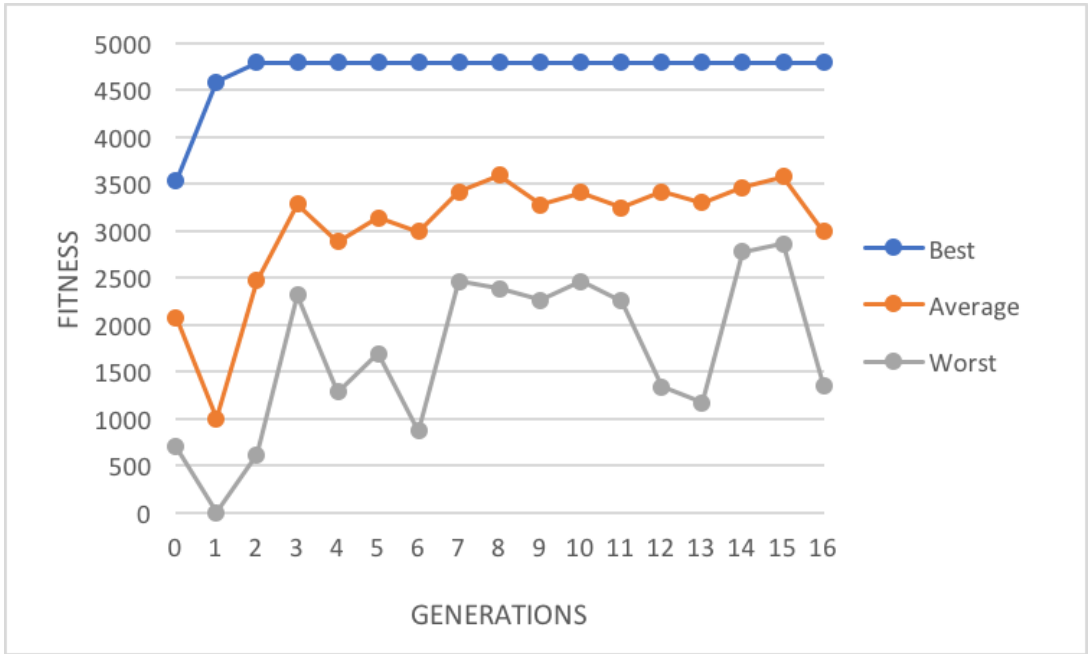


Figure 14: Fitness Score of GA Train Bot Against Snorkel

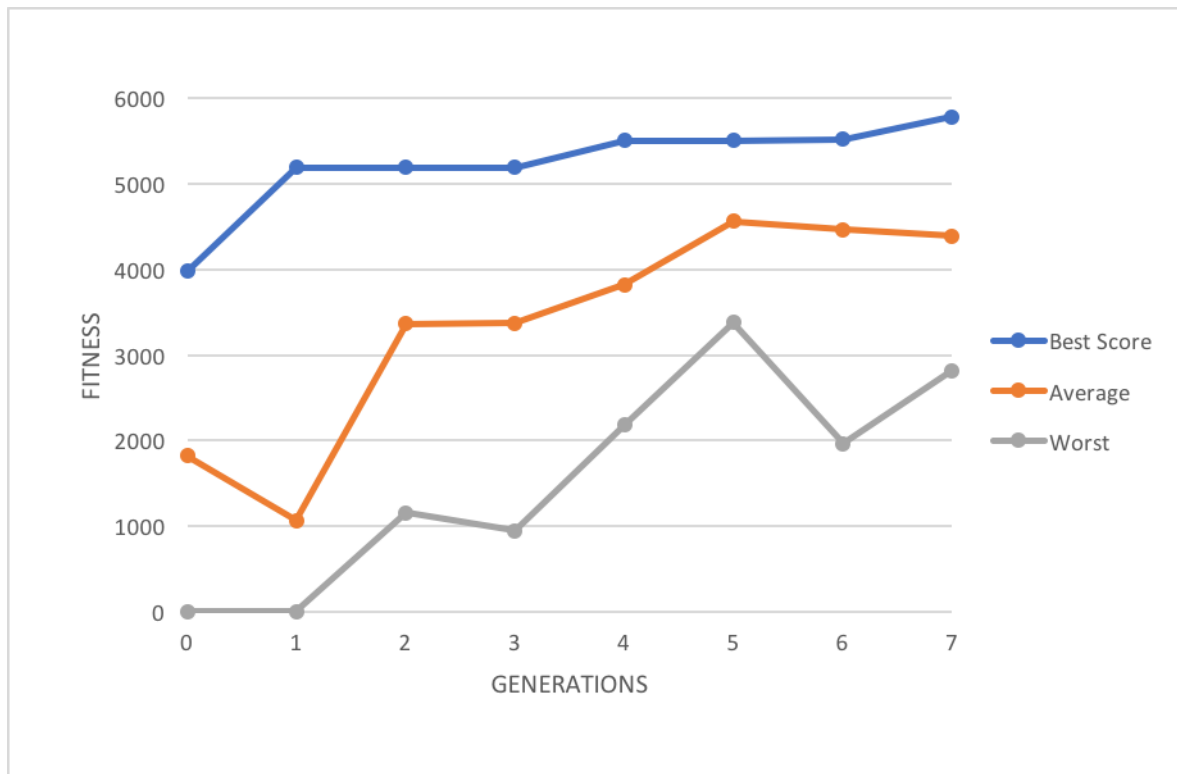


Figure 15: Fitness Score of GA Train Bot Against Triumph

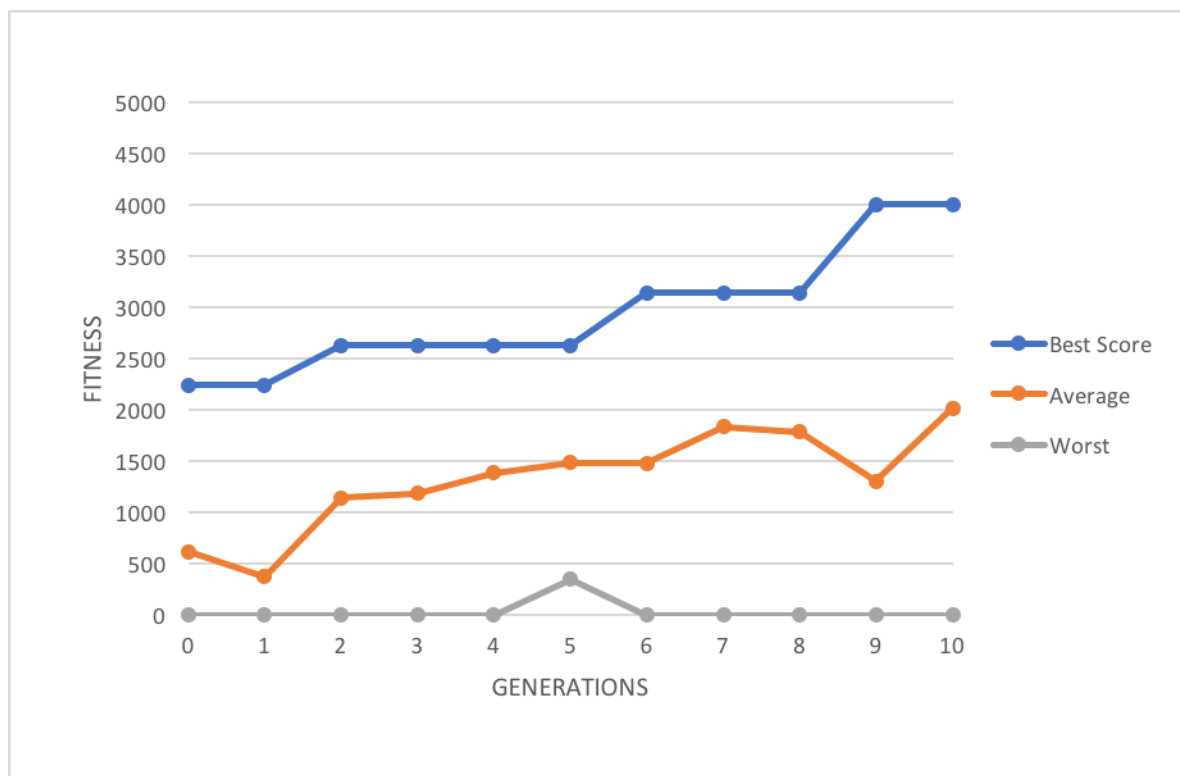


Figure 16: Fitness Score of GA Train Bot Against IchibanChan

From Figure 13, 14 and 15 we can note that when GA that trains GA Bot against AI not using simulator class, there is an immediate increase of the fitness score at the beginning of the generation, and then the improved speed of fitness score tends to be stable. Meanwhile, it can be seen in Figure 16 that there is a completely different score improvement trend

when training GA bot against IchibanChan, instead of significant improvement at the early stage, the fitness score improves stably and then increase dramatically at generation 9.

Figure 13 indicates that our GA may get stuck into local maximum which exactly proves one of the limitations that GA has which is not good at identifying a local optima and premature convergence occurs (Sivanandam and Deepa, 2008). However, it could be other reasons such as the limitation of our chromosome design as our chromosome only including the skill actions which may lead the current solution that we discovered is the global optimal solution and time waste in getting close to opponent as this opponent AI is designed never to perform a close attack and always move away from our AI when our AI getting close.

Finally, those best individuals show from figure 13, 14, 15 and 16 are collected to be used for our REGA Bot.

5.2.2 Time cost to build Good AI

The Experiments were performed on Inter Core i7-4700MQ CPU, each generation time for IchibanChan, KeepYourDistance and Triumph on this machine with the use of fastmode was 16 minutes while about 0.5 hours for Snorkel as Snorkel cost more initiation time per round. We observe from Table 17 that the generation time taken to build a good AI bot by our GA is fast while it only took 1.5 hours for our GA to build a good GA bot for Snorkel and KYDB, 2 hours taken to build a good GA bot for Triumph and about 2.5 hours used to build for IchibanChan.

Opponent Name	Generation time (hour)
GA for Triumph	2
GA for Snorkel	1.5
GA for KYDB	1.5
GA for IchibanChan	2.5

Table 17: Time Cost for GA to Generate a Good GA Bot

5.2.3 Strategy Variety

Figure 17 and 18 present that GA can always create different solutions that working for different opponents or a specific opponent. As we declared in the previous section, the location of the skill action and its probability could lead a different strategy, this proves that GA can develop several strategies, which means it can keep strategy variety for the opponent.

STAND_B	0.971	STAND_B	0.72	CROUCH_FB	0.97
THROW_A	0.651	THROW_A	0.837	STAND_FA	0.155
THROW_B	0.432	THROW_B	0.33	STAND_FB	0.894
STAND_FA	0.678	STAND_FA	0.609	CROUCH_FA	0.028
STAND_FB	0.411	STAND_FB	0.679	THROW_A	0.32
STAND_F_D_DFA	0.632	CROUCH_FA	0.84	THROW_B	0.072
CROUCH_FB	0.728	CROUCH_FB	0.427	STAND_A	0.809
STAND_D_DF_FC	0.049	STAND_D_DF_FA	0.311	STAND_B	0.843
STAND_D_DF_FB	0.521	STAND_D_DF_FB	0.504	CROUCH_A	0.473
STAND_D_DF_FA	0.518	STAND_D_DF_FC	0.068	STAND_F_D_DFA	0.341
CROUCH_FA	0.238	STAND_F_D_DFA	0.293	STAND_F_D_DFB	0.611
STAND_F_D_DFB	0.702	STAND_F_D_DFB	0.408	STAND_D_DF_FA	0.684
CROUCH_A	0.342	CROUCH_A	0.879	STAND_D_DF_FB	0.75
AIR_A	0.333	AIR_DA	0.739	AIR_DB	0.889
AIR_B	0.694	STAND_A	0.334	AIR_FA	0.123
AIR_DA	0.555	STAND_D_DB_BA	0.118	AIR_FB	0.614
AIR_DB	0.056	STAND_D_DB_BB	0.723	AIR_UA	0.92
STAND_A	0.44	AIR_UB	0.865	AIR_UB	0.096
STAND_D_DB_BA	0.749	AIR_B	0.592	CROUCH_B	0.692
STAND_D_DB_BB	0.763	AIR_FB	0.162	STAND_D_DB_BA	0.084
AIR_F_D_DFA	0.711	AIR_D_DF_FB	0.657	STAND_D_DB_BB	0.147
AIR_F_D_DFB	0.646	AIR_F_D_DFA	0.589	AIR_A	0.101
AIR_D_DB_BA	0.899	AIR_F_D_DFB	0.531	AIR_B	0.833
AIR_D_DB_BB	0.844	AIR_D_DB_BA	0.128	AIR_DA	0.805
CROUCH_B	0.808	AIR_D_DB_BB	0.51	AIR_F_D_DFB	0.127
AIR_FA	0.464	CROUCH_B	0.773	AIR_D_DB_BA	0.771
AIR_D_DF_FA	0.246	AIR_DB	0.127	AIR_D_DB_BB	0.957
AIR_UA	0.225	AIR_FA	0.704	STAND_D_DF_FC	0.624
AIR_UB	0.342	AIR_D_DF_FA	0.025	AIR_D_DF_FA	0.568
AIR_FB	0.215	AIR_UA	0.063	AIR_D_DF_FB	0.606
AIR_D_DF_FB	0.027	AIR_A	0.268	AIR_F_D_DFA	0.441

A)

B)

C)

Figure 17: A, B and C are chromosomes that generated by the GA against IchibanChan, Triump and Snorkel

STAND_D_DF_FC	0.572	THROW_B	0.02	STAND_D_DF_FA	0.202
STAND_B	0.521	STAND_D_DF_FC	0.428	CROUCH_B	0.861
CROUCH_A	0.763	STAND_B	0.584	STAND_FA	0.505
CROUCH_B	0.4	STAND_D_DF_FB	0.86	STAND_FB	0.794
STAND_FA	0.342	CROUCH_B	0.915	CROUCH_FA	0.417
STAND_FB	0.618	STAND_A	0.548	CROUCH_FB	0.156
STAND_D_DF_FB	0.355	STAND_F_D_DFB	0.131	THROW_A	0.597
CROUCH_FB	0.227	CROUCH_FA	0.071	THROW_B	0.066
STAND_D_DF_FA	0.75	CROUCH_FB	0.745	STAND_A	0.304
CROUCH_FA	0.371	STAND_D_DF_FA	0.818	STAND_B	0.295
THROW_A	0.165	CROUCH_A	0.733	STAND_D_DF_FC	0.126
THROW_B	0.049	STAND_FA	0.71	STAND_F_D_DFA	0.57
STAND_F_D_DFB	0.285	STAND_F_D_DFA	0.316	STAND_F_D_DFB	0.867
STAND_D_DB_BB	0.78	STAND_FB	0.81	STAND_D_DB_BB	0.085
AIR_A	0.002	STAND_D_DB_BB	0.521	AIR_A	0.596
AIR_B	0.335	STAND_D_DB_BA	0.631	AIR_B	0.668
STAND_A	0.089	THROW_A	0.771	CROUCH_A	0.327
STAND_D_DB_BA	0.063	AIR_A	0.959	STAND_D_DB_BA	0.366
AIR_UA	0.04	AIR_B	0.755	AIR_UA	0.536
AIR_UB	0.725	AIR_DA	0.794	AIR_UB	0.481
AIR_FA	0.6	AIR_UA	0.747	AIR_D_DF_FA	0.048
AIR_D_DF_FB	0.833	AIR_FA	0.597	AIR_D_DF_FB	0.579
AIR_F_D_DFA	0.652	AIR_FB	0.072	AIR_F_D_DFA	0.668
AIR_F_D_DFB	0.514	AIR_D_DF_FB	0.569	AIR_F_D_DFB	0.168
AIR_D_DB_BA	0.607	AIR_UB	0.574	AIR_D_DB_BA	0.167
AIR_D_DB_BB	0.643	AIR_D_DF_FA	0.461	AIR_D_DB_BB	0.518
STAND_F_D_DFA	0.777	AIR_DB	0.704	STAND_D_DF_FB	0.666
AIR_DA	0.414	AIR_F_D_DFA	0.678	AIR_DA	0.317
AIR_DB	0.261	AIR_F_D_DFB	0.386	AIR_DB	0.3
AIR_D_DF_FA	0.541	AIR_D_DB_BA	0.67	AIR_FA	0.97
AIR_FB	0.499	AIR_D_DB_BB	0.0	AIR_FB	0.642

Figure 18: A, B and C are Chromosomes that Generated by The GA Against KYDB

5.3 Recognition Result Based on Different Format

Before taking experiment for REGA bot, a quick experiment is held to compare the recognition accuracy between a new testing bot, which combines the Random Action AI with the Thunder01 AI format, and the data collection bot, which is the Random Action AI.

Name	Recognition Accuracy for Random Action with Thunder01 format	Recognition Accuracy for Random Action AI
KYDB	98%	97.3%
Snorkel	92%	98.7%
Triump	28%	63.3%
IchibanChan	98%	95.3%

Table 18: Recognition Result Based on New Format Compare with the Result for Data Collection Bot

Table 18 demonstrate that the testing bot performs worse than the data collection bot. Data collection bot has higher accuracy than the testing bot, where both bots can achieve an overall 95% recognition accuracy for KYDB, Snorkel and IchibanChan, but the recognition accuracy for Triump was poor for testing bot as the recognition accuracy is only 28%, which

even less than the half of the recognition accuracy that data collection bot did. Thus, the original data bot will be used to recognize opponent at the first round of the match for our REGA bot.

5.4 REGA and GA bot Test Against Last Year Competition AI

To test the performance of our proposed method, three AIs are used to test against those AI from last year competition. The details of three AI show as follow:

- Random Action AI: this AI is provided by official FightingICE competition that randomly performs actions without any predefined rules.
- The GA Bot: This AI will load a predefined solution that designed for the specific opponent at the beginning of the match.
- The REGA Bot: Instead of known which opponent is currently testing against, this AI will start as Random Action AI to recognize opponent at 47 seconds of the first round, in our experiment, a correct recognition would successfully load solution that specifically trained for that AI while a wrong recognition name would lead this AI to keep fighting as Random Action AI for that match.

All three AIs run 50 matches against the KYDB, Snorkel, Triumph and IchibanChan. The game setting and rules are followed the 2017 FightingICE competition as described in the early section of this dissertation.

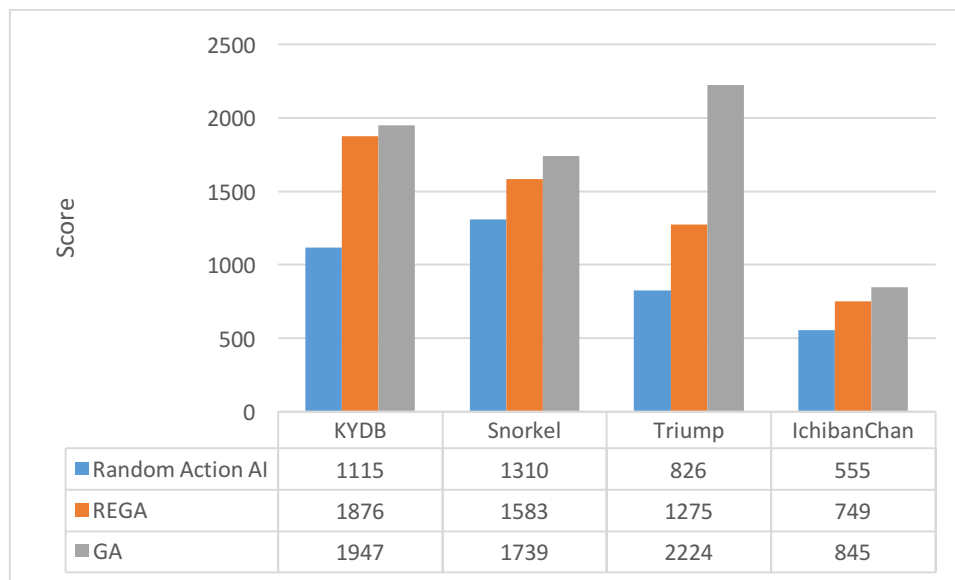


Figure 19: Average score of The Tested AI in A Match

Figure 19 shows that our GA is strong enough to test against those AIs that not using the game simulator class where all our GA achieve a score higher than the opponent. However, those scores are less than previous GA optimization section where they scored 2800, 2300 and 2400 when it against Triumph, KYDB and Snorkel. The potential reason for this could be the number of matches that set to be tested during the GA optimization section is small, which leads to this unstable result.

The result of GA tested against Snokel, which using the game simulator class, is extremely unacceptable (Figure 19). It only scores an average 845 points out of 3000 when the average score at GA generation section it achieves an average 2000 points per match, one of the reasons of this is the number of testing match is short as we described in the last paragraph, another reason could be the simulator class as mentioned in the beginning of this chapter, the real-time actions analysis provided by this simulator class prevent our GA solution from being effective. As our GA only won 2 matches out of 50 shows Table 19,

which proves that a simulator class is an extremely powerful tool when it used correctly, it can hugely improve the performance of the predefined rule-based system.

In terms of REGA bot against AI without using the simulator class, the average score is acceptable when the opponent is KYDB or Snorkel but unacceptable when the opponent is Triumph as the average score for REGA bot is lower than Triumph, details of this will describe in the next section. The result for REGA bot against IchibanChan is as unacceptable as GA against IchibanChan due to the optimal solution is not strong enough.

Opponent AI	Name	ROUND 1	ROUND 2	ROUND 3	Match	Win
KYDB	Random Action AI	370	380	365	50	1
	REGA with Wrong Recognition	576	500	500	1	1
	REGA with Correct Recognition	578	669	667	49	49
	GA	646 (506~684)	650 (500~696)	651 (506~687)	50	50
Snorkel	Random Action AI	421	423	466	50	9
	REGA with Wrong Recognition	425	460	452	1	0
	REGA with Correct Recognition	430	594	565	49	29
	GA	576 (113~780)	579 (124~1000)	584 (0~895)	50	40
Triumph	Random Action AI	288	272	266	50	0
	REGA with Wrong Recognition	274	263	193	23	0
	REGA with Correct Recognition	316	830	589	27	20
	GA	777 (0~1000)	746 (0~1000)	701 (0~1000)	50	48
IchibanChan	Random Action AI	194	183	178	50	0
	REGA with Wrong Recognition	46	497	244	1	0
	REGA with Correct Recognition	202	277	279	49	3
	GA	396 (0~1000)	202 (0~1000)	247 (0~1000)	50	2

Table 19: Details of Experiment Result for Random Action AI, REGA bot and GA bot Against KYDB, Snorkel, Triumph and IchibanChan.

It can be observed from Tables 19 that both GA and REGA shows a better result than Random Action AI, the solution that we trained by GA for each player is working for those AI without the use of simulator class, as those GA individuals won 138 matches out of 150.

In terms of REGA bot, the recognition performance from Table 19 shows that REGA bot made correct recognition 49 times for Snorkel, IchibanChan and Snorkel and identify the Triumph correctly 27 times achieving an overall 87% recognition accuracy which is an equivalent result as previous recognition testing section (Table 11). While the test performance will be evaluated in two cases.

The first case is wrong recognition, as we mention early, when REGA bot make a wrong recognition, we treat it to keep perform as the Random Action AI, the Table 19 shows that there is no significant difference in Random Action AI and REGA bot in wrong recognition. The only difference is the lucky win in one match for REGA bot against KYDB as Random Action AI only won one match out of 50 matches when against KYDB.

Another case is correct recognition. We observe from Table 19 that once REGA bot identified the opponent is KYDB, it has 100% winning ratio while it only won 29 matches out of 49 matches after it successfully identified the opponent is Snorkel. A closer look at the data indicates that the reason is REGA bot could average score 578 points at the first round when against KYDB, which increased from 370, nearly 200 points improvement, but the average score at first round only increased 9 points from 421 to 430 in terms of Snorkel, even REGA bot could score over 550 points at second and third rounds of the match, this slight improvement leading REGA to a below expectation result.

The overall score 1275 (Figure 19) for REGA bot is much lower than GA's score of 2274 regarding Triumph. There are two reasons can be discovered from Table 19, the first one is same as REGA bot against Snorkel, the improvement of the score is limited, which only improved around 30 from 288 to 316. The second reason is the REGA only recognized Triumph 27 times, with reference to those 27 times, it scored over 1700 points per match with a 74% winning rate. Thus, the result for REGA bot against Triumph shows negative.

CHAPTER 6

Conclusion & future work

6.1 Conclusion

This dissertation presented a method to achieve adaptive game AI by recognizing the opponent with the use of the expert system and machine learning technique, and then adapt itself to the recognized opponent by using the solution which pre-optimized by Genetic Algorithm for that specific opponent. At first, we apply a strategy extraction approach to select the best tuple and build the dataset, which is used to train the Random Tree classifier for classification, then a feature extraction approach is used to define the rules for expert system which is designed for player recognition, finally Genetic Algorithm is used to tune the parameterized rules for fighting game bot.

Testing those GA bots against AIs without using game simulator proved that the bots developed by the evolutionary process have a significantly better performance than then the AIs that without using the game simulator, achieving a 92% winning rate. However, our study was unsuccessful in proving the tuned rule-based system is better than real-time decision-making class, as it shows negative that the GA bot could only win AI that using game simulator 2 matches out of 50. In sum, we believe that 1) parameterized rule-based system bot tuned by GA is effective for against rule-based system AI while it is less effective for real-time decision-making AI, 2) GA can keep the minimal work to generate good AI as the time cost to build such AI is short, and 3) GA can keep variety of strategies for AI.

Our experiment result on player recognition shows that our REGA bot can recognize the opponent with a recognition rate 85.17% at 47 seconds of the first round. A further experiment on REGA bot against AIs without using game simulator could achieve a 66% winning ratio and 76% correct recognition rate while it also shows that the late adaptation after correct recognition or wrong recognition would lead this approach to a failure. In conclusion, our work on REGA bot against rule-based system AI is acceptable while we believe if we could shorter the recognition process, improve recognition accuracy for specific AIs or using a better format during recognition process, the result of our REGA bot would improve dramatically.

Due to the limit number of experiments, the findings of our study on GA operators cannot prove the following things: 1) Uniform mutation is better than Gaussian, 2) Uniform crossover with crossover rate 75% is better than 60% and 90%. 3) The performance of uniform mutation with 15% mutation rate is better than 5%, 10% and 20%.

6.2 Future Works

To further our research, we plan to do following things for GA:

- Instead of encoding skill actions and its probability into chromosomes, all actions would include in chromosomes.
- A new fitness function would be considered as current fitness function could generate an unstable solution.
- Game simulator would be considered at our next stage to make real-time decision.
- More crossover techniques and corresponding probabilities will investigate, and more mutation techniques and corresponding mutation rate would be tested.

Future work for recognition approach would concentrate on:

- Finding a better tuple, including more information, such as opponent energy.
- Recognizing the opponent earlier with a higher accuracy.
- Opponent action prediction.

CHAPTER 7

References

1. Yoshida, S., Ishihara, M., Miyazaki, T., Nakagawa, Y., Harada., and Thawonmas, R. (2016). 'Application of Monte-Carlo Tree Search in a Fighting Game AI', 5th Global Conference on consumer Electronics. Kyoto, Japan, 11-14 Oct. DOI: 10.1109/GCCE.2016.7800536.
2. Yamamoto, K., Mizuno, S., Chu, C.Y. and Thawonmas, R. (2014). 'Deduction of Fighting-Game Countermeasures Using the K-Nearest Neighbor Algorithm and a Game Simulator', 2014 IEEE Conference on Computational Intelligence and Games (CIG). Dortmund, Germany, 26-29 Aug. DOI: 10.1109/CIG.2014.6932915.
3. Chu, C.Y. and Thawonmas, R. (2015). 'Applying Fuzzy Control in Fighting Game AI', the 77th National Convention of IPSJ. Japan, 18 March. Vol.2, pp 101-102.
4. Ricciardi, A. and Thill, P. (2008). "Adaptive AI for fighting Games." Retrieved 23 Feb, 2017, from <http://cs229.stanford.edu/proj2008/RicciardiThill-AdaptiveAIForFightingGames.pdf>.
5. Sato, N., Temsiririrkkul, S., Sone, S. and Ikeda, K. (2015). 'Adaptive Fighting Game Computer Player by Switching Multiple Rule-based Controllers', 3rd International Conference on Applied Computing and Information Technology. Okayama, Japan, 12-16 July. DOI: 10.1109/ACIT-CSI.2015.18.
6. Kanetsuki, Y., Thawonmas, R. and Nakata, S. (2015). 'Optimization and Simplification of Dynamic Scripting with Evolution Strategy and Fuzzy Control in a Fighting Game AI', 4th Global Conference on Consumer Electronics (GCCE). Osaka, Japan, 27-30 Oct. DOI: 10.1109/GCCE.2015.7398536.
7. Nakagawa, Y., Yamamoto, K., Yin, C.C., Harada, T. and Thawonmas, R. (2015). 'Predicting the Opponent's Action Using the K-Nearest Neighbor Algorithm and a Substring Tree Structure', 2015 IEEE 4th Global Conference on Consumer Electronics. Kyoto, Japan, 27-30 Oct. DOI: 10.1109/GCCE.2015.7398673.
8. Nakagawa, Y., Yamamoto, K. and Thawonmas, R. (2014). 'Online adjustment of the AI's strength in a fighting game using the k-nearest neighbor algorithm and a game simulator', 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE). Tokyo, Japan, 07-10 Oct. DOI: 10.1109/GCCE.2014.7031274.
9. Asayama, K., Moriyama, K., Fukui, K. and Numao, Masayuki. (2015). 'Prediction as Faster Perception in a Real-time Fighting Video Game', 2015 IEEE Conference on Computational Intelligence and Games (CIG). Tainan, Taiwan, 31 Aug - 2 Sep. DOI: 10.1109/CIG.2015.7317672.
10. Park, H. and Kim, K-J. (2014). 'Learning to play fighting game using massive play data', 2014 IEEE Conference on Computational Intelligence and Games (CIG). Dortmund, Germany, 26-29 Aug. DOI: 10.1109/CIG.2014.6932921.
11. Arellano, G.M., Cant, R. and Woods, D. (2016). 'Creating AI Characters for Fighting Games using Genetic Programming', 2016 IEEE Transactions on Computational Intelligence and AI in Games. DOI:10.1109/TCIAIG.2016.2642158.
12. Cho, H., Kim, K. and Cho, S. (2013). 'Replay-based Strategy Prediction and Build Order Adaptation for StarCraft AI Bots', 2013 IEEE Conference on Computational Intelligence and Games (CIG). ON, Canada, 11-13 Aug. DOI: 10.1109/CIG.2013.6633666.
13. Ballinger, C., Liu, S. and Louis, S. (2015). 'Identifying Players and Predicting Actions from RTS Game Replays', 28th International Conference on Computer Applications in Industry and Engineering (CAINE). California, USA, 12-14 October.
14. Byrne, J., O'Neill, M. and Brabazon, A. (2010). 'Optimising offensive moves in toribash using a genetic algorithm', 16th International Conference on Soft Computing. Brno, Czech Republic, 23-25 June.

15. Ikeda, K., Tanaka, Y., Viennot, S., Quoc, N and Ueda, Y. (2012). 'Adaption of Game Ais using Genetic Algorithm: Keeping Variety and Suitable Strength', 2012 IEEE on the 6th International Conference on Soft Computing and Intelligent Systems, and the 13th International Symposium on Advanced Intelligence Systems. Kobe, Japan, 20-24 Nov. DOI: 10.1109/SCIS-ISIS.2012.6505249.
16. Cole, N., Louis, S.J. and Miles, C. (2004). 'Using a Genetic Algorithm to Tune First-Person Shooter Bots'. 2004 Congress on Evolutionary Computation, Portland, OR, USA, 19-23 June. DOI: 10.1109/CEC.2004.1330849.
17. Grosan, C. and Abraham, A. (2011). Rule-Based Expert Systems. In Intelligent Systems A Modern Approach. New York, Springer.
18. Tutorialspoint. (2017). "Artificial Intelligence - Expert Systems." Retrieved 16 June 2017, from https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_expert_systems.htm.
19. Han, J., Kamber, M. and Pei, J. (2012). Data Mining Concepts and Techniques. 3rd Ed. Burlington, Morgan Kaufmann.
20. Chou, C.H. (2013). Using Tic-Tac-Toe for Learning Data Mining Classifications and Evaluations. Int. J. Information and Education Technology. 3(4): 437-441.
21. Sivanandam, S.N. and Deepa, S.N. (2008). Introduction to Genetic Algorithms. New York, Springer.
22. Kramer, O. (2017). Genetic Algorithm Essentials. United States, Springer International Publishing.
23. Melanie, M. (1999). An Introduction to Genetic Algorithm. London, MIT Press.
24. Sharma, P., Wadhwa, A. and Komal. (2014). Analysis of Selection Schemes for Solving an Optimization Problem in Genetic Algorithm. Int. J. Computer Applications. 93(11): 1-3.
25. Holland, J.H. (1992). Adaptation in natural and artificial systems. London, MIT Press.
26. Rahman, R., Ramli, R., Jamari, Z. and Ku-Mahamud, K.R. (2016). Evolutionary Algorithm with Roulette-Tournament Selection for Solving Aquaculture Diet Formulation. Mathematical Problems in Engineering. 2016: 1-10. DOI: 10.1155/2016/3672758.
27. Alabsi, F. and Naoum, R. (2012). Comparison of selection methods and crossover operations using steady state genetic based intrusion detection system. J. Emerging Trends in Computing and Information Sciences. 3(7):1053-1058.
28. Obitko, M. (1998). "Introduction to Genetic Algorithms. XIII. Recommendations" Retrieved 27 March, 2017, from <http://www.obitko.com/tutorials/genetic-algorithms/recommendations.php>.
29. ICE. (2017). "Welcome to Fighting Game AI Competition" Retrieved 18 Dec, 2016, from <http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-2h.html>.
30. Jaggi, S., Bhushan, R. and Malhotra, D. (2013). Guidelines to Decide the Encoding Scheme Used For G.A.. Int. J. Advanced Research in Computer Science and Software Engineering. 3:1436-1440.
31. MathWorks. (2017). "Genetic Algorithm Options." Retrieved 3 July, 2017, from <https://uk.mathworks.com/help/gads/genetic-algorithm-options.html?requestedDomain=uk.mathworks.com>.
32. Kumar, E.R, Gill, E.S and Kaushik, E.A (2000). A Crossover Probability Distribution in Genetic Algorithm Under process Problem. Int. J. Advanced Research in Computer Science. 1(4): 504-508.
33. Tansey, W. (2010). "Evolutionary Algorithms: The Little Things (part 1)". Retrieved 13 July, 2017, from <http://www.nashcoding.com/2010/07/07/evolutionary-algorithms-the-little-things-you-d-never-guess-part-1/>.
34. Wright, A.H. (1999). 'Genetic Algorithms for Real Parameter Optimization', Foundation of Genetic Algorithms, Vol 1. P205-218. DOI: 10.1016/B978-0-08-050684-5.50016-1.

35. Haupt, R.L. (2000). 'Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors', IEEE Antennas and Propagation Society International Symposium. Salt Lake City, UT, USA, 16-21 July. DOI: 10.1109/APS.2000.875398.
36. Sayad, S. (2017). "ZeroR" Retrieved 4 June, 2017, from <http://www.saedsayad.com/zeror.htm>

CHAPTER 8

Appendices A: Bitbucket Commit Evidence

Author	Commit	Message	Date	Builds
 Wenger95	a1d3cd3	README.md edited online with Bitbucket	2017-02-06	
 Wenger95	d9beafc	AI Project setup	2017-02-06	

Figure 20: Bitbuck KCLAIProject Commit Evidence part 1































Author	Commit	Message	Date	Builds
 Wenger95	4132cb5	Can run game on windows bug fixed. Now ready to test whether make a judgement in first 5 seconds, 10 seconds, 15 seconds based on all actions and attack actions opponent m...	2017-03-03	
 Wenger95	62a9f61	14 AI in 2016 Competition included, error fixed. Ready for data collection session 2.	2017-03-02	
 Wenger95	22b588b	Update Runnable AI. Version 3. 10 runnable AI.	2017-03-01	
 Wenger95	a79d1cc	Update runnable opponent AI. Version 2. Finished collect 8 runnable characters' data	2017-02-28	
 Wenger95	f3f795e	Update runnable opponent AI. Version 1	2017-02-27	
 Wenger95	6963cc7	All running version AI that I've tested. Only bug AI left, will contact with their developer to know how to fix it and run it.	2017-02-27	
 Wenger95	d89683d	HP400, Nine types, Four types, AttPosNeg, Attack action, all action.	2017-02-24	
 Wenger95	43bf41a	Testing finished, This data collection version is for final dissertation. Later on may add Actions per second, or Actions per five seconds	2017-02-20	
 Wenger95	f29b25a	Fix bug of previous data collection version	2017-02-20	
 Wenger95	e89be5a	Data collection version. Output 8 files in total. 1: Only positive Dis X,Y. All actions from player. 2: Only positive Dis X,Y. Only attack from player. 3: Positive and Nega Dis X,Y. All a...	2017-02-20	
 Wenger95	f920884	Data Collection mode, collect 6 files. 1)All types Dis Neg. attack and all actions. *2 2) dis neg and posti. attack and all actions. *2 3) G-G, A - A, A-G, G-A.	2017-02-20	
 Wenger95	89cebbe	Dis Pos and Dis pos with Neg output in one time.	2017-02-19	
 Wenger95	c186141	Adjust txt file, previous collecting data won't tell you which character is this one, this case will also output the name of the character, so easy for identify character from database.	2017-02-18	
 Wenger95	c58ab32	Four types, G-G, A-G, G-A, A-A. Recording all rounds.	2017-02-18	
 Wenger95	3461ea8	Data collection version. Three types: Air-Air, Ground - Air, Air - Ground. This version only record the first round of each match	2017-02-18	
 Wenger95	c47946b	Only first round data. All type. Dis +,-.	2017-02-17	
 Wenger95	f11d8cc	Correct version of only record different action (pre and this action). Recollect the all type data from now.	2017-02-17	
 Wenger95	338ba81	This version only record attack actions	2017-02-16	
 Wenger95	24874b3	Disable Windows mode, combine with fastmode, making the game faster. Avoiding some actions that continue appeared to write in to the file.	2017-02-16	
 Wenger95	c93c7a6	Fixed batch file, now can quick start to train our AI character, this version limit hp as 100.	2017-02-16	
 Wenger95	12e424c	This version can output two files, one contain all actions, another one contain only attack (not sure, will check it later). Testing against with 2013 winner KNN AI character, cluster...	2017-02-16	
 Wenger95	51719a7	File bug fixed, now we can store data. Batch file haven't complete, still can't run it by click the batch file. Need help here.	2017-02-15	
 Wenger95	988a181	File Successfully created. However process function seems just run once. Next step: why only run once, why I can't get distance between my character and opponent.	2017-02-15	
 Wenger95	a942514	Fixed last version no official AI bug. This version: Wenger AI doesn't move at all, the file also not create.	2017-02-15	
 Wenger95	25e7f8f	Created three official baseline AI jar, ready to test and collect data	2017-02-15	
 Wenger95	22496ff	Write distance X (opponent character to my character) an distance Y (opponent character to my character), and opponent action to file. Not working this version.	2017-02-13	
 Wenger95	79c2089	Get frameData update, which is to collect the game data, when finish whold fd, we can then start to collect data both our character and opponent. However in this step, we only n...	2017-02-12	
 Wenger95	04cc503	FightingICE setup Finished Successfully. Now can start to test the AI. (Seems sound file is too large to commit, shows error, need manually backup the whole project)	2017-02-06	
 Wenger95	6648ccf	SampleAI, ready to test the AI code	2017-02-06	
 Wenger95	3fc3c38	SetUp	2017-02-06	

Figure 21: Bitbuck KCLAIProject Commit Evidence part 2

























Author	Commit	Message	Date	Builds
 Wenger95	44848ac	Add distance to WengerGA AI, remove score calculate from WengerGA AI.	2017-06-23	
 Wenger95	aee3781	Add WengerGA read actions chromosome and prob chromosome code	2017-06-23	
 Wenger95	dfc0457	Add actions chromosome and it's prob chromosome file output, making it possible for WengerGA AI file to read and run the AI.	2017-06-23	
 Wenger95	df13615	Fix selection bug. add selection function in main function.	2017-06-23	
 Wenger95	32c38d2	Finish the initSelection function, add the parents selection for this stage. Call it in mainGA	2017-06-22	
 Wenger95	47f14c1	initSelection added, afterwards selection would nearly based on this one, simply change the popsze, as only 8 chromosome need to be check the fitness.	2017-06-22	
 Wenger95	ca5438f	Adjust fitness function. Add uniform mutation.	2017-06-22	
 Wenger95	f6ae98a	Fitness function added, selection based on score. Tested in single java file, it works.	2017-06-22	
 Wenger95	a612bca	Finish the code for crossover, nearly finish the code for mutation.	2017-06-21	
 Wenger95	3be2764	6.20 fightingICE update	2017-06-20	
 Wenger95	a3a613c	Adjust WengerGA code, make it able to read from the output of the chromosome.	2017-06-20	
 Wenger95	e1fe5fb	Add WengerGA which is the real GA AI, Genetic Algorithm is the one to teain the AI, WengerGA is the one used to test against opponent	2017-06-20	
 Wenger95	98d79c5	Adjust the GA fitness bug	2017-06-20	
 Wenger95	c7a0bc59	Adding FightingICE score calculation code, which will be part of my fitness fuction	2017-06-18	
 Wenger95	29fbc16	Final version of recognize data collection.	2017-06-15	
 Wenger95	c1bd33f	Recognize AI, add new output (output all names in one file, easy for calculate percentage of all time guess.)	2017-06-14	
 Wenger95	3831e39	Setup for genetic algorithm	2017-06-13	
 Wenger95	18508d1	Test the arff file and auto create data file for recognize, (1,2,3,4,5) sec files	2017-06-13	
 Wenger95	e8bd666	Bug on recongize opponent's name and bug on how to weigh the opponent fixed. However, recongize rate still not as high as I wanted.	2017-06-09	
 Wenger95	c419bd2	try to initial real time instance which is same format of the database, use this instance to check the weighted of this instance of all instances, check which name have the highest ...	2017-06-08	
 Wenger95	0786f51	Batch can't load Arff file bug fixed.	2017-06-05	
 Wenger95	a2633ca	Add Load Arff function to load Arff data file and build random tree classifier. Can successfully read it inside the program, but can't run it on batch file.	2017-06-05	
 Wenger95	91f1e16	Backup, ready to add Weka code.	2017-06-02	
 Wenger95	b819b11	Adjust collection data. No need for flart 5,10,15 seconds actions number as too less instance for future use. Instead of this, record every 1,2,3,4,5 seconds attack actions.	2017-03-23	
 Wenger95	eba3f7e	Data collection session two, 10 left	2017-03-19	
 Wenger95	d6386c3	Ready to run the Triumph data collection.	2017-03-14	
 Wenger95	8e0bf35	Update the FightingICE platform and AIToolkit at Latest version 3.1	2017-03-14	
 Wenger95	dfe309e	Ready to collect Poring no dead version	2017-03-12	
 Wenger95	01f8d95	Three left, collect Poring	2017-03-06	
 Wenger95	8af678d	Uncomment the code that make a judgement in first 5 seconds, 10 seconds, 15 seconds based on all actions and attack actions opponent made can help us to recongize opponent	2017-03-04	

Figure 22: Bitbuck KCLAIProject Commit Evidence part 3

Author	Commit	Message	Date	Builds
Wenger95	4582fb0	Increase the default short distance of our GA AI to trigger a attack	2017-08-05	
Wenger95	d487e2c	Update the WengerGA.java file. Fix rule base distance bug	2017-08-05	
Wenger95	de54049	Update the output function	2017-08-04	
Wenger95	7282fe9	Update selection function	2017-08-04	
Wenger95	6c82a5e	Fixed Gaussian mutation rate bug.	2017-07-28	
Wenger95	133c647	Add Gaussian mutation code.	2017-07-13	
Wenger95	65f52a1	Fix WengerGA not get close bug.	2017-07-07	
Wenger95	74eea41	Replace for loop by while, if score*noMatch over requirement, we stop generation.	2017-07-06	
Wenger95	908d218	Fix generation condition bug.	2017-07-04	
Wenger95	f1372a9	Fix code that can stop generation when fitness reach required level	2017-07-04	
Wenger95	c3a7479	Delete walk and forward jump as This does not really help. Adjust the WengerGA value.	2017-07-03	
Wenger95	781a24b	Add Walk, for_jump jump etc movement action into our AI.	2017-06-30	
Wenger95	2d8caab	Increase the mutation rate, update the WengerGA AI structure, based on Thunder01	2017-06-30	
Wenger95	c3e2582	Delete averageno use original data instead	2017-06-29	
Wenger95	c723d40	Making AI more aggressive.	2017-06-29	
Wenger95	d7134f5	Fix score calculation bug.	2017-06-29	
Wenger95	a1871ff	Add --mute in batch file mute the sounds and image for FightingICE engine.	2017-06-27	
Wenger95	5bbaa8f	Fix score output bug. Fix best score is the sum of the score bug. Add output the chromosome to the file code.	2017-06-27	
Wenger95	83bb226	Output the best chromosome at the end of the generation.	2017-06-26	
Wenger95	f766fcb	Ready to go version, but generation test just for one round seems not stable enough for next selection. Not always better children chromosome	2017-06-26	
Wenger95	6f17e4b	Fix first generation prob files output wrong bug.	2017-06-25	
Wenger95	9a8cd5c	Fix bug in generation file reading, fix bug in indexoutofbound in generation selection.	2017-06-25	
Wenger95	26a108f	Move GA code under FightingICE folder, now batch file is working, program will wait until batch close to continue	2017-06-25	
Wenger95	f9c5b4a	Backup, will move Genetic Algorithm code under FightingICE document. The only way to run batch file successfully in the code.	2017-06-25	
Wenger95	885a4d2	Fix WengerGA class not found bug. Fix WengerGA file read not correct bug. Fix Genetic Algorithm run batch file bug. Fix Genetic Algorithm not creating action and prob for init ...	2017-06-24	
Wenger95	5d46493	Fixed second part action chromosome exchange index bug.	2017-06-24	
Wenger95	a11bf9d	Adjust value, use variable in the loop not the real value, make it easier to change. Ready to test	2017-06-24	
Wenger95	26c935e	Add additional WengerGA 8,9 and all corresponding WengerGA0-9 batch file for Genetic Algorithm to run. Add code to delete the thisgeneration.txt file at each iteration.	2017-06-24	
Wenger95	8888c6c	Add corresponding WengerGA 0-9. Add get close to opponent code for those AI	2017-06-24	
Wenger95	47bbf99	WengerGA: Add distance y for each action, is all default, and not acquire, would do it if have more time	2017-06-24	

Figure 23: Bitbuck KCLAIPProject Commit Evidence part 4

Author	Commit	Message	Date	Builds
Wenger95	e616e9c	Update the AI format	2017-08-13	
Wenger95	c474924	Adjust value to 2000*noMatch, while loops works. Ready to test.	2017-07-05	
Wenger95	c8ee15b	Replace multi conditions for loop by while loop. Test with 200 score to make sure it will stop when score over 200*noMatch	2017-07-05	
Wenger95	974266e	Fix for condition, if score greater than requirement, then program stop.	2017-07-05	
Wenger95	771f6f1	Fix conclude array index bug at selection.	2017-07-05	
Wenger95	624885a	Adjust bestscore array size same as conclude size in selection. Adjust the cmd to run the fightingice engine.	2017-07-04	
Wenger95	6ecf03d	Merge branch 'master' of https://bitbucket.org/Wenger95/randomgeneration	2017-07-04	
Wenger95	1833d87	Change Xover and mutation to completely random process, random select action and get prob for AI. Same structure as WengerGA	2017-07-04	
Wenger95	38d1876	README.md edited online with Bitbucket	2017-07-04	
Wenger95	48c8548	Update the fitness function, update the random process, delete the crossover, delete the mutation.	2017-07-04	
Wenger95	dea88bd	Update as a new repo. Next step create whole random chosen AI	2017-07-04	

Figure 24: Bitbuck randomgeneration Commit Evidence

Appendices B: Creation Time for Snorkel and KYDB

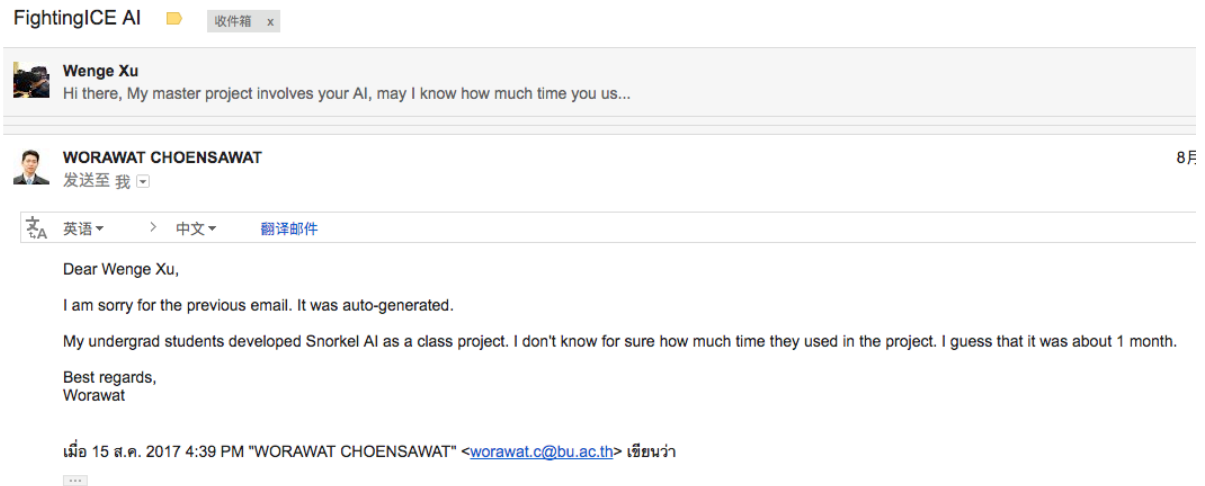


Figure 25: Creation Time for Snorkel

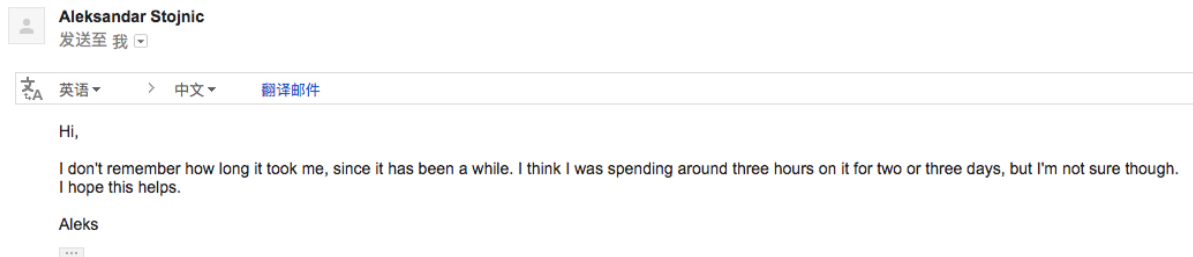


Figure 26: Creation Time for KYDB

Appendices C: Source code of Batch File and Shell File

Batch File

```
-----
setlocal ENABLEDELAYEDEXPANSION

set PATH=%PATH%;./lib/native/windows
java -cp FightingICE.jar;./lib/lwjgl.jar;./lib/lwjgl_util.jar;./lib/gameLib.jar;./lib/fileLib.jar;./lib/jinput.jar;./lib/commons_csv.jar;./lib/javatuples-1.2.jar;./lib/py4j0.10.4.jar Main -n 2 --disable-window --mute --fastmode --limithp 400 400 --c1 ZEN --c2 ZEN --a1 WengerGA0 --a2 DragonSurvivor

endlocal
exit
-----
```

Shell

```
#!/bin/bash
java -cp
FightingICE.jar:lib/gameLib.jar:lib/lwjgl.jar:lib/lwjgl_util.jar:lib/javatuples-
1.2.jar:lib/commons_csv.jar:lib/jinput.jar:lib/fileLib.jar:./lib/py4j0.10.4.jar -
Djava.library.path="lib/native/macosx" Main -n 2 --disable-window --mute --fastmode --
limithp 400 400 --c1 ZEN --c2 ZEN --a1 WengerGA0 --a2 KeepYourDistanceBot
```