Human-like Computer Agents using NeuroEvolution with Statistical Penalties

By Luong Huu Phuc

A project paper submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Kokolo Ikeda

March, 2017

Human-like Computer Agents using NeuroEvolution with Statistical Penalties

By Luong Huu Phuc (1510060)

A project paper submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Kokolo Ikeda

and approved by
Professor Hiroyuki Iida
Associate Professor Shinobu Hasegawa

February, 2017 (Submitted)

Abstract

Recently, many of things in modern life are inspired by the nature. Humans tried to create AI, which refers to things such as machines with intellect, to solve real life problems, and AI is currently attracting many people in the world. AI is important to modern life because it can be used not only to help humans doing tough, complex, repeated works, but also to entertain humans.

Typically, the term AI often appears in the game domain because games provides simple, well-defined and easily evaluating environment, so approaches such as supervised/reinforcement learning, optimization, tree search, reasoning for making computer agents are usually proposed through game environment.

Recently, there are some common trends in making game AI agent. Developing a strong AI computer agent is one of them. In 1997, the first computer champion in chess game won against human champion. In 2016, a method, which utilizes deep neural network, successfully created a strongest computer player in a very complex game of Go. Therefore, the ending of this trend is approaching.

Other than making strong AI computer agent, developing human-like AI computer agents is also interesting. Various approaches are used to develop human-like computer players and many competitions are held in order to find the most human-like computer agent. Developing a human-like computer agent, especially in games, is essential for many reasons. One of a reason is the human-like behavior of a computer agent can make game more engaging and interesting. A computer agent, which is able to behave in a human way, can be used to teach human players to play a game by guiding and supporting. Moreover, a human-like computer agent can be employed to simulate how a human player play a game for suitable game level evaluation.

NeuroEvolution is an training method for AI, and it has a potential to create human-like computer players for not only one particular game. Thus, this research focuses on developing human-like computer agents using NeuroEvolution, and statistical penalties are introduced to improve the human-likeness of NeuroEvolution. The term *Penalty* represents the differences in behaviors of human players and computer players. Penalty calculation is applied to limit mechanical actions, so the human-likeness of the computer agent might increase.

The goal of this research is creating human-like computer agent which can behave as intermediate human players. In order to create human-like computer players, two objectives are tackled. The first one is that a computer agent should be at the intermediate level of human players, and the other is that the computer agents behavior should be enough to pass the Turing Test, which is the test for human-likeness evaluation. To sum up, our proposed method employs NeuroEvolution to address the first objective, and optimizing statistical penalties is for the second objective.

Training computer players to produce human-like actions requires human data, so an analysis of human players' characteristics in the game domain was conducted. Rather

than directly used human data for training, our proposed method requires only statistical data from analysis. The performance of the computer agent was tested on a two-dimensional action game, and the human-like behavior was evaluated by judgments from human subject.

In this research, a modified version of the Super Mario Bros. game is used as a testbed. Super Mario Bros. is a famous two-dimension action game. In this game, players control Mario character to move right from the left of levels to reach a goal. The game is made into a benchmark for competitions and research purposes. Many times the benchmark of this game is put into competition to find the best controller, in term of strength. Since Super Mario Bros. is a deterministic game, the A* search algorithm has shown its outstanding performance. Besides, competition on the most human-like controller is also held, but the winner controller, in term of human-likeness, is not at the level of the least human-like human players.

The proposed method in this research is based on NeuroEvolution approach because this approach is proved to be the most human-like controller in Super Mario Bros. game. Even though A* search algorithm with some biological constraints can be successfully introduce with the human-likeness as human expert players, we do not employ A* search algorithm because NeuroEvolution has a potential to be extended to another problem domains. Thus, our proposed approach is not limited to only 2D action games.

In order to developing human-like computer agent in a 2D action game, or Super Mario Bros. in particular, we first do analysis on human player characteristics by giving instruction to human player and asking them to play some levels provided by the benchmark of Super Mario Bros. game. Our analysis has shown that even human players play in various styles given different instructions. Therefore, statistical information of human behavior is used in our method to minimize the difference in behavior between human players and computer players.

After gathering statistical data of human characteristics, some features, which describe the behavior of players, are used for penalties calculation. The penalties are the differences of features, for example, the average number of pressed button and frequency of changing actions, in the behavior of human players and computer players. If the behavior of a computer agent is significantly different from the behavior of human players, then a penalty is given to the computer agent. Consequently, by introducing the objective of minimizing penalties along with the main objective of the game, NeuroEvolution is utilized to train the computer agent so that the computer agent can solve the game and behave like human players.

The trained computer player is evaluated through two tests: a performance test and an extended version of the Turing Test. The performance test evaluates the strength of the computer agent and its behaviors. The strength is measured by the average score obtained from the game environment, and the behavior of the computer player is statistically compared to analyzed data of human players. For human-likeness, Turing Test is conducted by asking human subjects to rate the behavior of the computer player.

The experimental results shows that the human-likeness of agents created by using NeuroEvolution can be improved, and the computer agent is able to achieve the

number of actions partly similar to human players. The average human-likeness value of NeuroEvolution computer agent is 2.38. Our proposed method scores 3.02. However, human player score was 1.08 points higher than our proposed method.

Our method successfully enhances the human-likeness of a computer player. Further, if better features and training strategy are used, the human-likeness of computer players is expected to be higher.

Contents

Acknowledgment

Above all, I would like to express deepest gratitude to my advisor Professor Kokolo Ikeda of the School of Information Science at Japan Advanced Institute of Science and Technology. During the time I study and research under his guidance, I have received many supports, encouragements, and many valuable comments from him. Even though I have made many mistake, he was always patient and steered me out with his experience and immense knowledge.

I would like to acknowledge Professor Hiroyuki Iida for reading and giving insightful comments on my thesis. Besides, I would like to thank to Professor Atsuo Yoshitaka for his advice and comments on my minor research project.

I also would like to say thanks to all other members of my lab for helping me so much since the first time I came.

Finally, I must express my very profound gratitude to my parents for always be by my side supporting me in everything. Thank you.

Author

Luong Huu Phuc

Chapter 1

Introduction

Thousand years ago, humans tried to understand every single mechanism exhibiting in the nature. We even attempted to interpret the human body, brain, and also behavior. Recently, many things in modern life are inspired by the nature. In the viewpoint of researcher, the science and engineering of creating intelligent machines is called Artificial Intelligence (AI). AI is one of the newest sciences, which is currently attracting many people. AI is important to modern life because it helps doing tough, complex, and repeated works, to say nothing of its entertaining. The application of AI varies in different fields such as face detection for auto-focus in camera in image processing, speech recognition for translation in natural language processing. All of them aim at making intelligent things or machines be able to solve a problem.

Since games are simple, well-defined and easy to evaluate, many AI methods such as supervised/reinforcement learning, optimization, tree search, reasoning are proposed to solve problems in this domain. In the field of game AI, researchers tried to make intelligent machines which are able to beat human champions, to entertain human players, to automatically generate suitable game stages, or even to teach human players to play games. One common trend in making a game AI agent is developing a strong AI, says an AI champion - which is able to beat human experts. This kind of making a strong AI has drawn the attention of many people around the world including developers and researchers. In 1997, the first chess machine, called Deep Blue [10], won against human champion in chess game. Contemporary research on deep neural network, AlphaGo [14], is the first computer player to defeat a human champion in a very complex game of Go. Since one of the most complex game is solved already, so now is the time for other directions.

Another trend is developing human-like agents. The purpose of making human-like agent differs from that of a strong AI agent. Strong AI agents' behaviors tend to be somehow impossible to human players such as very precise movement or immediately responses. In contrast, human-like AI agents try to react more naturally as possible. In the past, judging the intelligence of a machine was done by Turing Test, proposed by Alan Turing [17] in 1950. In the original test, if a judge cannot clearly say which of the generated conversation text samples is from a machine or a human, it is considered to have passed the test. After being introduced, many researchers used extended versions of

the Turing Test to evaluate the human-likeness of their AI agents. However, the gap in human-likeness between recent computer agents and humans is still big and difficult to achieve.

Human-like game AI agent is necessary because sometimes human players think of whom they are playing with. Taken AI behavior as an example, AI agents act similarly in same situations of a game, but experiencing same things is not so interesting. Human players will not be surprised with the situation that they had already faced in the past. New and unexpected events in a game might be more amusing for human players. Thus, the objective is to develop non-human computer agents, which are able to behave in human ways, and many approaches such as supervised learning, biological constraints are proposed to fulfill this objective.

This research mainly focus on developing human-like game AI agents. Because of the variety in type of computer games, only 1-player action game type is taken into account. The target of this research is to create AI agents which can behave more like human by using some penalties to punish mechanical, non-human behavior as well as unnatural behavior so that more human-like and natural behavior can be obtained.

A testbed, which is used in this research, is Mario AI benchmark 0.1.9 [16]. This testbed had been already used in competitions in 2009 and 2010, and it is a modified version of Super Mario Bros. by Nintendo, one of the most famous two-dimension action game in the world. Strong AI agent in this game is already achieved by using A* search algorithm [4], and the most human-like AI is developed using NeuroEvolution algorithm [2]. However, the ideal human-likeness level of AI agents is not yet reached. Comparing to human players, the behavior of AI agents is still very unnatural. Therefore, if a computer can learn how to play, then the questions are "Is it possible to make a computer learn to play like a human?" and "How to make computer players be more like human players?" This research aims at enhancing the human-likeness of AI agents in Super Mario Bros., but it is not limited to only action games.

The structure of this research contains five chapters. The first chapter is the introduction. The second chapter is about Human-like computer agent, previous methods and its achievements. Chapter 3 describes the architect and implementation of the proposed method. Afterward, Chapter 4 shows the benchmark and how the proposed method is used. Next, Chapter 5 discusses evaluation methods and experiments results. Finally, Chapter 6 provides the summary of this research and future works.

Chapter 2

Human-like Computer Agent

Computers are very popular in recent days. The interaction between humans and computers becomes very dense. In the field of game AI research, the primary purpose of games is entertainment, so variety of strategies and techniques intended to amuse human players [1]. Game AI researchers tried increase the engagement of games by not only developing strong AI but also more human-like AI. The strength of AI computer agents can be determined by comparing to human champions. Unfortunately, measuring the human-likeness of computer agents is a difficult task. In this research, only human-like behavior is considered. Since the behavior of players depends on the type of game and the role of players such as opponent/teammate roles, the human-like behavior, which is considered in this research, is the behavior of the main character only. It is the problem of a computer player tries to play a single player game like a human player.

Human-like computer agents refers to machines that can actually behave in the way similar to how a human reacts to events. A human-like computer agent should be able to sense real world information, think, and response rationally. The ways it resolve a problem must be as much similar to human's behavior as possible. Furthermore, the differences in a human's decision and a computer agent's decision should be small. A machine is called a human-like computer agent if it can resolve problems or react to outside world in human ways.

The human-likeness of a computer agent is expected to make games more interesting and engaging due to the fact that an AI player can be a partner/rival or a guide to interact with human players in a game. Since computer players occasionally seem to be unfair and unreasonable in some situation, so playing with other human-like computer players may be more realistic, or natural in other words, and engaging than playing with consistent computer players. To measure the human-likeness of a computer agent, Alan Turing proposed the Turing Test [17] in 1950. This is a classical method for evaluating the human-likeness of a computer agent, and it is used widely in most of research on human-like machines.

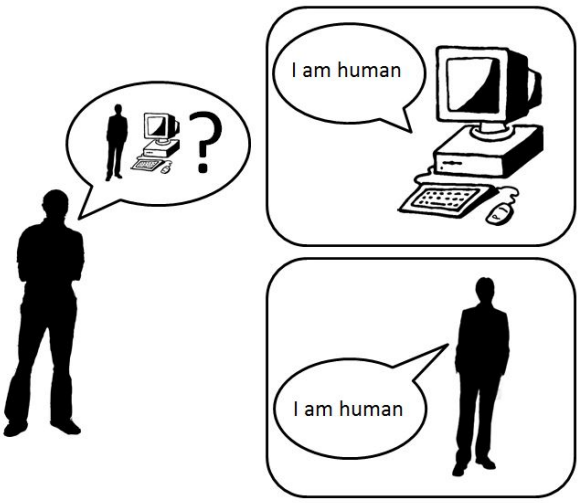

Figure 2.1: Turing Test example

2.1 Evaluation of Human-likeness by Turing Test

In artificial intelligence, the Turing Test is a method for judging whether or not a computer can think like a human. It was named after Alan Turing, an English computer scientist and mathematician who is the first to use and apply artificial intelligence during the 1940s and 1950s. In 1950, Alan Turing raised a question "Can machine think?" and described a problem called "Imitation Game". In his problem, there are three people, a man (A), a woman (B), and an interrogator (C) who may be of either gender. The interrogator need to point out the gender of the other two by asking them some questions. Whereas the role of A is to try and cause C to misjudge, B has the role of proving to C that B is the woman. Later on, he change the problem by replace A with a machine to investigate if it is capable of thinking like a human, and it soon became a test for computers. According to this test, a computer is designed to mimic some human behaviors, and it is considered to have passed the test if human subjects are unable to determine which one is a computer and which one is a human.

To simplify the problem addressed by the original Turing Test, the test is conducted by asking human subjects to distinguish which of the two conversations is from the real person as shown in figure 2.1. Therefore, the human-likeness can be measured by using this test. For that reason, an extended version of the Turing Test is considered as a test to judge if a computer can be a human.

Let's think about Turing Test for game AIs. There are reasons that Turing Test is promising for computer players in games. In computer games, Turing Test is similarly performed with participants: an interrogator, a human, and a computer program as shown in figure 2.1. the computer program in order to pass the test has to deceive the interrogator into thinking it is a human. This kind of Turing Test is used in the competition 2K BotPrize Contest, and its results is presented and analyzed afterward [5].

Furthermore, not only human-likeness is judged in the competition, but many comments and feedback on the skills, tactics, aggression and mistakes in behaviors of players are also given.

In 2012, another Turing Test competition was held in Mario AI Championship [13]. This competition is inspired by the 2K BotPrize Contest. While 2K BotPrize Contest uses first-person shooter game as the domain, Mario AI Championship focuses on a 2-dimension action game. Julian Togelius, the organizer of this Turing Test track in Mario AI Championship, employed "believability" as the measurement factor for the human-likeness of computer behavior. It means both believability and human-likeness is just the assessment for the behavior of computer player in the Turing Test. In the test, human subjects are asked to watch two game plays, one by a human player, and another one by a computer player. They are also asked to fill in a questionnaire.

According to the results of the Turing Test and information gathered from the questionnaires, we can figure out not only which algorithm is the most human-like, but also the reasons and factors that affects the believability, or the human-likeness in other words, of computer agents in games. That is one of extended versions of Turing Test which can be used for human-likeness evaluation, especially in game.

2.2 Human-like Game AI Agent Approaches

In last few years, numerous competitions were held to find either strongest AI agents or human-like agents in games. An earlier Turing Test competition, called BotPrize [5], was held to find human-like computer in the game Unreal Tournament 2004, which is a first-person shooter game in 3D arenas. Many approahces such as "NeuroEvolution" were employed, but their human-likeness were not at the level of the least human-like human players. In 2.2.1, the architecture of an agent using NeuroEvolution will be introduced.

The Mario AI Competition in 2009 [16], is for finding strong AI player in Mario, and it was shown that A* algorithm is very strong for this purpose. The Mario Ai Competition in 2010 [7] includes not only stregth-oriented tract but also three other tracks: *Learning track*, *Turing Test track*, and *Level Generation track*. The *Learning track* was for making AI agent which can learn how to play. The *Turing Test track* aimed at human-like behavior computer agent, and the *Level Generation track* used for making engaging level generator. Information about the Mario AI Competition is introduced in 2.2.2.

2.2.1 Architecture of the UT^2 Computer Agent

The BotPrize competition employed the computer game Unreal Tournament 2004 (UT2004) as a testbed. In this game, players are able to move around a three dimension virtual world in a first person's viewpoint. There are items such as weapons and health packs randomly respawn for player to pickup. A player uses weapon to damage other players, and a player die if enough damages are taken. The competition conducted the Turing Test to test the ability of imitating human players of computer players.

The result of the competition shown that the UT^2 bot ranked the 2^{nd} place with humanness rating of 27.3%. The humanness equals the number of human judgments on total judgments in percentage. Comparing to human player, even the least human-like player achieved 35.5% of human-likeness, so the human-likeness of the UT^2 bot is still not at the level of human players.

Architecture

This computer player learns combat behavior by NeuroEvolution - details are shown in 3.1, and UT^2 also uses human trace database to help it get unstuck. Additionally, an imitation technique is also used, to modify the observation as a human.

The architecture of UT^2 bot contains three modules triggered by logic cycles as shown in figure 2.2. The leftmost column is the control module. In the middle, some associated controller are used, and all available actions for the bot in this game are in the right column. The author provided this architecture to make it easier to programming, troubleshooting and to see the overall behavior.

Since UT^2 bot are trained by NeuroEvolution with multiobjective to learn combat behavior, a set of three objectives are defined. Based on the given objectives, the NeuroEvolution executes the process inspired by the nature to find solutions (details on

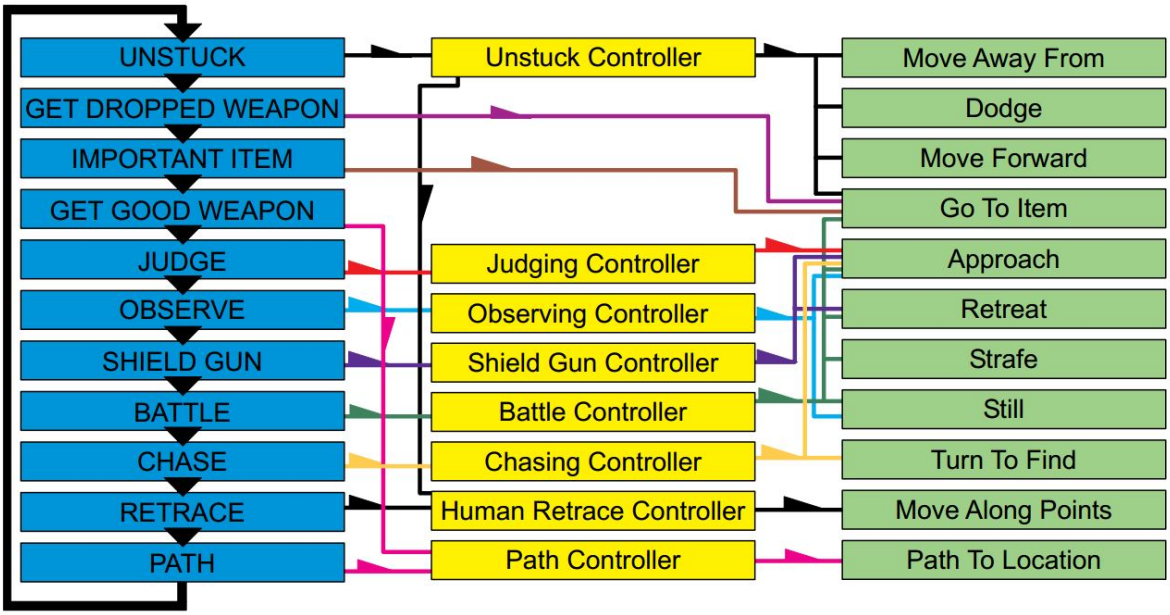

Figure 2.2: The UTˆ2 bot architecture. [11]

NeuroEvolution are shown in the next Chapter). The first objective is to maximize the dealt damage. And the others are to minimize both the received damage and the number of collision events with level geometry. Detailed architecture and learning process as well as multiobjecttive optimization are shown in [12]

Human Trace Replay

Training a computer player to play like a human player usually requires data collected from human players. Using human trace data is a natural way, but it may not be the best way. The UTˆ2 bot is trained this way.

A set of human trace replays are gathered in order to help UTˆ2 bot in navigation. These data are mainly used in two controller modules such as *UNSTUCK* and *RETRACE*, which are shown in the architecture of this computer agent.

The necessary human data that need to be collect are usually sequences of agent locations along with the time when the player was at that locations. This kind of information can be used to investigate how the human player move through a level, and the information is used in the UTˆ2 bot as way points.

The controller module *UNSTUCK* in UTˆ2 bot chooses actions by scripted response, similar to rule-based script. The current employed version of this controller module by the author sometimes fails to navigate in locations where target points are too far. In that case, human traces are used. Neither there are navigated points nor human traces, random actions are taken into account to solve stuck situations.

If there is no human data for training the computer player, it might be difficult to make the computer player be more human-like. Moreover, a huge number of human

data requires effort for collecting and training, and too few human data is not efficient for directly training.

2.2.2 Mario Competition 2012 Human-like Computer Agent

The winner of the Turing Test track which was held in the 2012 Mario Championship competition [13] is named VLS bot. After that, there have been several approaches for producing human-like behaviors introduced and compared [8], such as Supervised Learning, NeuroEvolution and Dynamic Scripting accordingly. In this section, important approaches are briefly introduced.

VLS Bot

The VLS bot has its name of its contributors Vinay, Likith, and Stefan. It is inspired by a technique in robotics field, namely Artificial Potential Field. This technique utilizes influence map in defining number of things in potential fields. The author described four types of field which are used in the agent to find solutions to situations in Super Mario Bros. game.

Four potential fields that the author used are: (1) The field of progression, (2) the field of rewards, (3) the field of opponents, and (4) the field of terrain. The first field, says progression field, makes Mario to move right rather than left. The field of rewards make coins, mushrooms, blocks, and flowers attractive so that Mario will change his attention to these rewards. Next, the field of opponents provides dangerous positions for Mario to keep distance from enemies or to kill them. Finally, terrain field ensure Mario to avoid gaps, dead ends, and search for correct path ways. By using these potential fields, the computer will choose the action that will take it to the most attractive position. Moreover, tuning parameters for each fields are also used, and the appropriated values are obtained by executing Turing Test.

The evaluation of human-likeness in the competition is calculated based on the number of human votes from human subjects. The result of the competition showed that VLS bot acquired 25.79% of score in the Turing Test track, while expert human players score 23.21% and 30.95% is the score of novice human players. An explanation for the human-likeness of the VLS bot is that VLS bot spends time moving left, collecting most items, and interactive with enemies other than just running in the right direction.

NeuroEvolution Bot

Artificial neural network (ANN) is commonly used in AI with supervised, unsupervised, or reinforcement learning method. However, the ANN can be trained by using Evolution Algorithm (EA), so the definition of Neuro-Evolution is came from this combination. Similar to Direct Policy Search, which is used in weightlifting robot [9] and later on as a framework for robot model [6], NeuroEvolution defines an ANN as a solution in the population. The mechanism of NeuroEvolution is described more detailed in the next section.

The proposed Mario computer agent using NeuroEvolution is basically trained by adjusting weights of the ANN. The purpose is to make an AI agent which produce the same trajectory or positions of Mario compared to a human player. In order to calculated the mean squared error, collected human data is required. The human data for training this agent is collected by asking 10 different human players to play 40 levels of Super Mario Bros. game. The game state with the corresponding action as well as Mario traces are recorded in a log file. For the training, 10 agents are trained using data from each human player, and the number of reset times, whenever the distance between AI and human positions exceeds a threshold, is used to calculate fitness score. The average human-likeness of ten computer players corresponded to ten human data is computed for evaluation.

The results of the Turing Test for the NeuroEvolution Bot [8] showed that NeuroEvolution had 89 times of human judgments in total. Supervised learning method, which uses ANN Back Propagation, achieved 21 times. Comparing to human players, human players received 110 times of human judgments. Thus, NeuroEvolution is better than ANN with supervised learning, but it is still not be able to achieve the human-likeness as human players The reason why NeuroEvolution in the result of this competition is better might due to the initialized structure and weights of the ANN in NeuroEvolution is the trained ANN in the supervised learning agent.

A* Agent with Biological Constraints

According to the result of the competition, A* is the strongest computer agent to solve the game Super Mario Bros. in particular. Unfortunately, the behavior of computer agent using A* search algorithm looks very mechanical, and the performance of A* is too efficient so that it might surpass even human expert.

To limit the unnatural actions and increase the human-likeness of A* algorithm, Fujii et al. [3] introduced the intentional error and artificial delay in actions of an AI agent. Biological constraints are applied by including noise into the input information of the agent, using information of the past for the current actions, and physical fatigue (by limiting the number of pressing keys). This approach made A* algorithm to be more human-like in Mario game than human expert players, but it may be less human-like in other game domains.

In the case of 2D action games, such as Super Mario Bros., there were currently three human-like approaches, which are VLS, NeuroEvolution, and A* with biological constraints. VLS can be implemented by using hard coding. Hence, it is simple and easy to implement, but its inflexibility is a drawback. It is said that NeuroEvolution is able to mimic one trajectory of a playing, but the cost for comparing the positions of the character in the game between computer agent and human players is expensive. Also, if different levels are given, the cost for collecting more human data raises. In addition, human traces data is also required for the performance of this agent. Meanwhile, A* provide strong AI as expert human players, and biological constraints contribute to its human-likeness. The disadvantage of this method is the calculation time since it requires

	Training method	Advantages	Disadvantages
UT^2 (*NeuroEvolution*)	NEAT[15] & Control modules	- Modularized	- Require good human traces - Have to defined actions for each module
VLS	Rule-based & Potential fields	- Simple and easy to implement	- Inflexible - Have to define potential fields
Mario bot (*NeuroEvolution*)	ANN + EA	- Can mimic a playing trajectory	- Require good human traces and time for comparing to human trajectory
A* + Biological constraints	Search algorithm & Influenced map & Rule-based	- Strong as expert humans - Less mechanical actions	- Require long calculation time
Proposed method	NeuroEvolution & Statistical Penalties	- Not require directly training using human data - Generality	- Require human statistical data

Table 2.1: Summary of previous methods for Human-like AI

searching for solution every single frame. The more time it has, the better performance it achieve.

Although many methods has been proposed to solve the problem of human-like computer agents, they are still not enough to have big achievement as in the case of strong AI players. Previous approaches have their own strengths and weaknesses which are summarized in the table 2.1. The proposed method is also included in table, the detail will be described from section 3.2.

Chapter 3

Human-like Computer Players based on Statistical Penalties of Features

The proposed method is described in this chapter. This approach uses NeuroEvolution with an additional calculation of penalties for deciding better solutions.

3.1 NeuroEvolution

NeuroEvolution is a combination of an artificial neural network (ANN) and an Evolutionary Algorithm (EA). In games, NeuroEvolution is used for decision making. The ANN in NeuroEvolution has the role of an input-output function. Its input is a game state (*orsituation*), and output of the ANN is usually an action. In NeuroEvolution, the ANN is trained by using EA to adjust the weights instead of directly training methods for ANN such as backpropagation using gradient descent. Typically, the ANN in NeuroEvolution is usually a fully connected network, and only its weights change. The objective of NeuroEvolution is to maximize/minimize a fitness value corresponding to how much a problem is solved.

In our proposed method, we do not use a fully connected neural network because of its performance and inflexible structure. Instead, we employ an approach, which is similar to the NeuroEvolution of Augmenting Topologies (NEAT) [15] proposed by Stanley and Miikkulainen in 2002, to apply in our problem. NEAT uses a direct encoding to store information of an ANN, for example, nodes, connections, etc. The difference between NEAT and normal NeuroEvolution approach is that the ANN in the NEAT contains no connection and hidden nodes at the beginning. It lately grows in number of hidden nodes and connection each generation in the process of the genetic algorithm.

3.1.1 Artificial Neural Network

Artificial Neural Network (ANN) is an approach which is inspired by the human brain. In ANN, there are three major parts, a node, a link or connection, and an activation

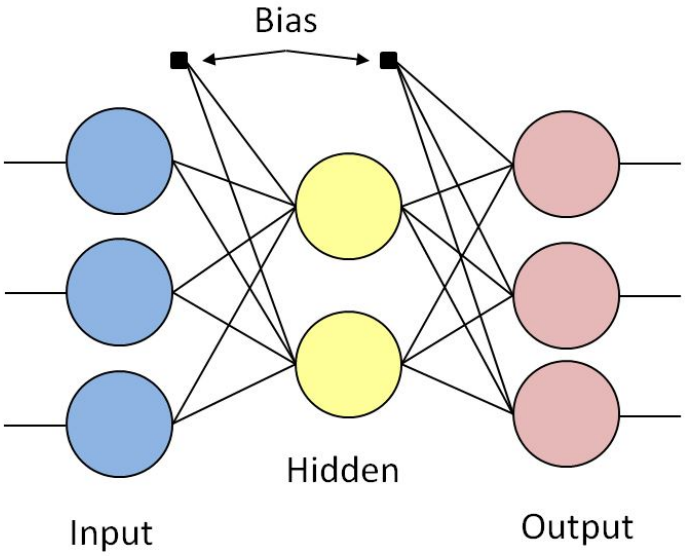

Figure 3.1: An example of a fully-connected ANN.

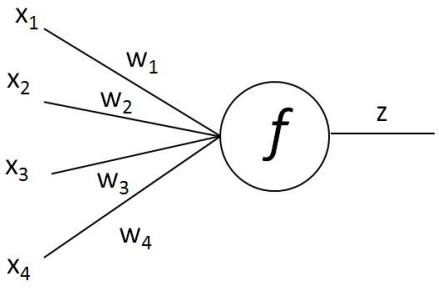

Figure 3.2: An example of a perceptron.

function. An ANN can have numerous nodes and connections, and it has a potential to simulate the human brain.

The smallest unit in the ANN is a node or a perceptron. Each perceptron has its own activation functions, but the activation functions are usually the same for every node in a particular problem. Then, grouping a set of perceptrons creates a layer. Typically, a network, which consists of three layers, are commonly used in many problems. These layers are categorized into three main types, for example, input, hidden and output type of a layer. Figure 3.1 presents the structure of an ANN. Each input layer and output layer contains only one layer, while hidden layer can be a set of multiple layers. Some simpler problems need only just one layer in the hidden layer such as XOR gate, and more complicated problems, in which image processing is adopted, require many layers for the hidden layer.

The perceptrons in an artificial neural network communicate with each other through connections or links. Connections transfer outputs of previous perceptrons and

feed into perceptrons of the next layer. Every connection has a weight value so that the input value of a perceptron is calculated using the following equations 3.1.

$$a = \sum_{i=0}^{i<n} x_i w_i \tag{3.1}$$

$$z = f(a) \tag{3.2}$$

In equation 3.1, a is the sum of all inputs multiplied by its weights. After the input sum is calculated, a function f is applied to correspond the output activation value. The activation function is used to define the output of a unit in the network; for example, a function outputs '1' for an activated status, and '0' for a deactivated status. This activation function should be chosen carefully, and it should also be appropriated for the objective of the training phase. Widely used activation functions includes *step*, *tanh*, *sigmoid*, *softmax*, etc.

Backpropagation is a usual way to train an ANN. The process of backpropagation method is shown as below:

1. Feed an input vector into the network, then compute output of each unit in the hidden layer and the output units.
2. Evaluate error of all output units.
3. Backpropagate the error of output units to compute error of each hidden unit.
4. Update weights of the network.
5. Return to the first step until reaching a specific number of iterations or an error threshold.

The application of artificial neural network is to simulate the way humans solve problems. Comparing to human brain, this mechanism is tremendously less complicated, but it is able to employ mathematical calculation for problem solving.

3.1.2 Evolutionary Algorithm

Evolutionary Algorithm (EA) is a stochastic search and optimization heuristics. The idea comes from the natural theory of evolution. Basically, EA utilizes the process of natural selection and reproduction. To illustrate, given a population of specific organisms. A survival organism, which adapts flexibly the environment and be able to survive, is supposed to be the suitable organism for that environment.

As an optimization method, an organism represents a solution for problem. Many candidate solutions are initialized, evaluated and selected according to the criterion of the problem. After the selection, new solution are reproduced by merging and modifying

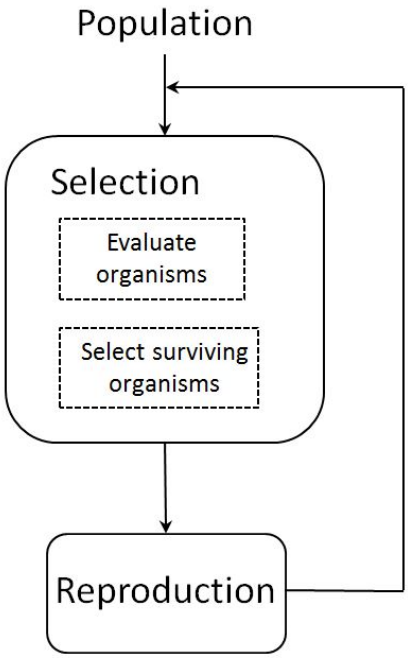

Figure 3.3: Basic procedure of Evolutionary Algorithm

existing solutions. By repeating this cycle, it is expected that better solution are found, like as better living are evolved.

An Evolutionary Algorithm starts with a set of initialized solutions called population. Figure 3.3 presents essential steps in EA. The first step in EA is evaluating and selecting best solutions in the population. A good solution is a solution which meets the requirements, and bad solutions will be removed from the population. Afterward, reproduction step create new solutions based on remained solutions in the population, and new solutions are put back into the population. These steps make a cycle, so new solutions with high adaptability are created. After some iterations, the best solution for a problem can be found.

3.1.3 Training in NeuroEvolution

An artificial neural network is usually trained by back propagation, but NeuroEvolution optimizes the weights of the ANN by doing the procedure of an Evolutionary Algorithm.

By definition, NeuroEvolution begins with a population, in which an organism is an ANN, and one organism is called a genome. The structure of initialized ANNs are randomly chosen, and it normally has no connection and nodes at the beginning. In case a fully-connected ANN is used, initialized weights are randomly chosen, and they differ for each genome.

Similar to the procedure in Evolutionary Algorithm, all genomes in the population will be evaluated by being put into an environment as shown in figure 3.4. Only

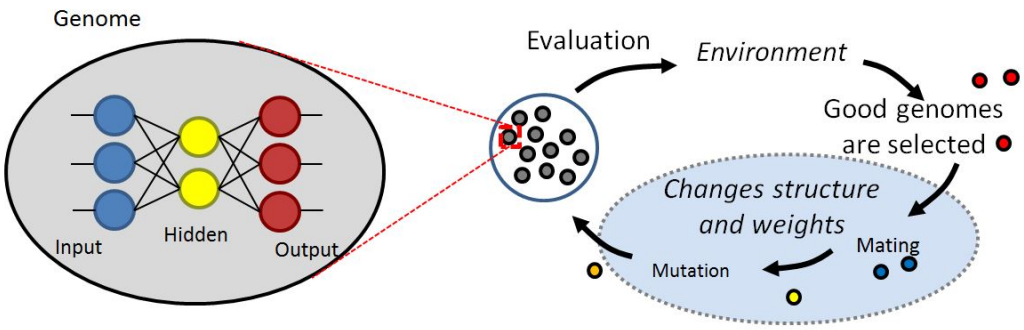

Figure 3.4: NeuroEvolution training process

good genomes are selected for the reproduction step to replace bad genomes. In reproduction step, either the structure or connections' weights of a genome can be changed to create a new genome. There are several ways to create new genomes, and one of them is combining the structure of two genomes. This is called mating in the nature or crossover in the domain of EA. Finally, new genomes are put back into the population as the next generation to continue evaluating. Throughout this process, high-quality solutions for optimization and search problems might be generated.

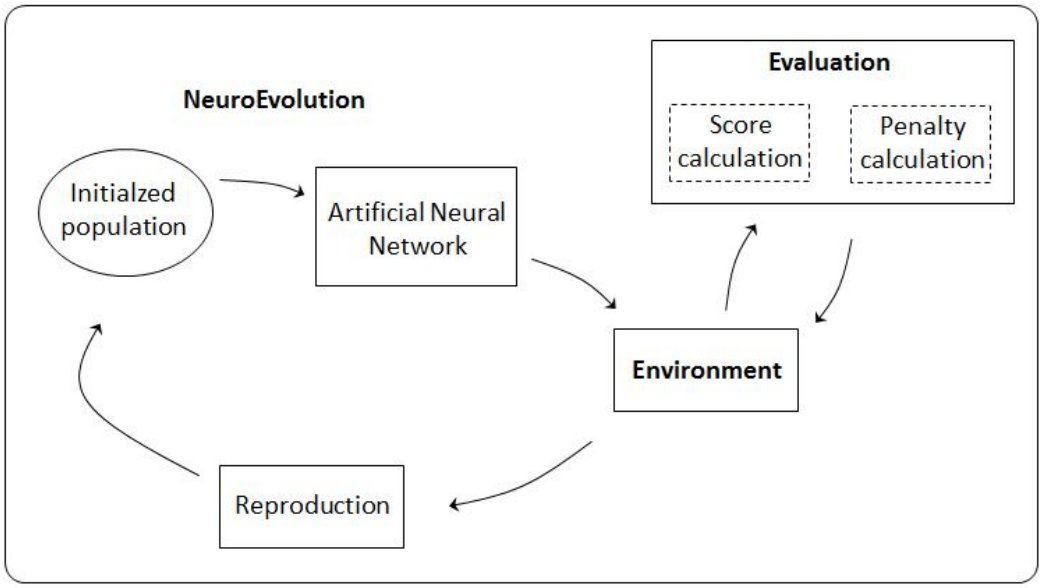

Figure 3.5: Proposed method architecture

3.2 Proposed architecture

Previous works had shown that NeuroEvolution has the potential to imitate human players. Despite more a human-like agent in games such as A* with biological constraints in Mario was shown, but it is only for 2D action games. The fact that A* is efficient for path finding if and only if there is enough time for calculation. More complex game might increase the calculation time of A*, so more general approach is needed.

Additionally, The NeuroEvolution agent using resetting times and human data [8] outperforms among ANN with Backpropagation and Dynamic Scripting, but it is still not at the human-likeness level of human players. Also, direct human trajectory matching is a costly calculation and not efficient since large human data is required to cover all situations. Therefore, we proposed a method, in which penalties is used for evaluating the behavior of computer agent. We introduce an additional term "Penalty" meaning the difference of behaviors between AI and human players in order to reduce unnatural behaviors. The architect of our method is depicted in figure 3.5

The architecture consists of three components: (1) *NeuroEvolution*, (2) *Environment*, (3) *Evaluation*. An approach similar to NEAT [15] is used to develop the first component *NeuroEvolution*. The *Environment* is the targeted game. In this research, Super Mario Bros. is the targeted game, and the benchmark of this game is used as a testbed. The *Evaluation* component is designed based on the *Environment*.

$$Fitness = Score - TotalPenalty \tag{3.3}$$

The process of this architecture consists steps similar to NeuroEvolution. At first, a set of artificial neural networks are initialized as the population. The structure

of ANNs contains no hidden node and connection. Then, they will be put into the environment for evaluation. Unlike the normal evaluation step in NeuroEvolution, an additional calculation of penalties is done because best solutions for the problem is not only the main target. For example, human players seem to change their action not so frequently, so a penalty value should be given to the computer player in case it changes actions too frequently. The best solution, in term of solving game, must satisfy some requirements to additionally be the most human-like solutions. That is why penlaty calculation is introduced. The fitness function for evaluating solution is computed by using equation 3.3. In the fitness function, *score* is the evaluation on the main target, and *totalpenalty* is calculated based on feature extracted from human play data. The calculation of total penalty is shown in the next section.

After being evaluated, the reproduction step handles the job of creating new solutions by mutating current solutions and replacing worse solutions in the population. Each individual ANNs is assumed as a solution, so mutation creates new solution by altering the ANN's structure. An evolutionary strategy, called $(1 + \lambda)$-ES, is used in this step for the mutation. More generally, λ mutants can be generated and compete with the parent, and the parent is replaced only if there is a better mutant.

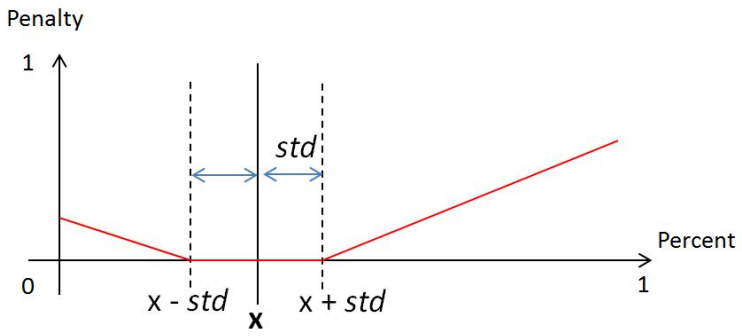

Figure 3.6: A feature penalty example

3.3 Definition of penalty calculation

Generally, penalty is a punishment which is given to a specific behavior to show that bad behaviors are not allowed. For computer agent in games, things such as number of pressed buttons, number of normal or illegal actions can be considered as features for punishment. Features can be defined differently according to the game. Hence, penalty can be used to punish mechanical and non-human actions by introducing which features should be punished.

To increase the human-likeness of computer agent, features, which describe the behavior, such as the number of actions, the number of times that actions are changed, etc. might be included. For instance, "Left button" is almost not used in A* agent, but human players may use for moving backward to collect more coins. Thus, the comparison of the feature is valuable. The idea is that if these features of a computer player and human players are significantly different, the punishment can be given to the computer player. As a consequence, a computer player with more human-like behavior, says it has less penalty or punishment, might be obtained by training with additional penalties beside score.

Figure 3.6 shows an example of a feature penalty calculation. Before calculating a penalty, human data is collected, and the analysis of human data should be done so that the considered feature average value and its standard deviation can be obtained. Suppose that the considered feature is the number of an action A, the average percentage of action A in human players' behavior is x percent, and the standard deviation is *std*. Next, let a computer agent to play the game, and its behavior is obtained. In addition to the score given by the environment, the percentage of action A of the computer player, $pActionA$, is compared to the average percentage of human players. The penalty is calculated using the following formula.

$$Penalty_{actionA} = \begin{cases} ((x - std) - pActionA)^2 & if \quad pActionA < x - std \\ (pActionA - (x + std))^2 & if \quad pActionA > x + std \\ 0 & \text{Otherwise} \end{cases} \tag{3.4}$$

Using equation 3.4 for penalties calculation in NeuroEvolution changes the prob-

lem into multi-objective optimization. Standard deviation is used in this case because not all human players play the same style. The variance in behavior of human players exists, so some human players seem to be more human than the others. For that reason, it is better to use standard deviation to declare the range of value which penalty is not given.

$$TotalPenalty = \sum w_i * Penalty_X \tag{3.5}$$

The total penalty is calculated using equation 3.5. The sum of penalties is calculated as a summation of the squares differences from the mean. Since different features may be used, the weight of each penalty for each feature are needed to be balanced. If weights are not balanced, some penalties may not be effectively used to limit unnatural behavior. To decide weights for features, the values such as average numbers of action of human players should be approximated. For instance, if the average of 'RIGHT' action is around 700 and the average of 'LEFT' action is 150, the weight of penalty of 'LEFT' action should be 4. Besides, if a feature is more important than others, its weight can be set higher.

Later on, the total penalty along with the score obtained from the game environment are used to defined a fitness function 3.3 for optimization in NeuroEvolution. The score returned from the environment of the game is to measure how far a computer agent is able to clear levels, while penalty calculation is to limit the behavior of the computer agent not to exceeds human players' limitations. Thus, the objectives are maximizing score and minimizing total penalty. The fitness value is calculated by taking the score minus to the total penalty. Note that the score and total penalty can also lead to underestimating solutions. If the value of score extremely big comparing to total penalty, the total penalty will have no effect at all, and vice versa. For example, assume that $S\left(score, penalty\right)$ is a solution associated with its evaluation, if there are two solutions $S_1\left(100, 1\right)$ and $S_2\left(90, 5\right)$, then S_2 is more human-like than S_1, but it is removed because the total fitness of S_1 is 101 larger than 95. Thus, we also need to balanced the weights of score and total penalty.

Chapter 4

Environment and human data analysis

This chapter introduces the environment which is used in this research. Brief information of human data analysis is also introduced followed by preliminary experiments.

4.1 Mario AI Benchmark 0.1.9

The testbed game used for the study presented in this research is a modified version of Markus Perssons Infinite Mario Bros [7], namely Mario AI Benchmark, which is a public domain clone of the Nintendos classic 2D action game Super Mario Bros. Super Mario Bros. is a very popular and famous game, so it was made into a benchmark and used in many competitions and international academic conferences. There are four tracks in the competition held in 2010, so participants of the competition and researchers can develop an AI as they prefer. These tracks are: Gameplay track, Learning track, Level Generation track, and Turing test track. The Gameplay track seeks for strongest controllers which are able to solve as much level as possible without any training or learning, while Learning track gives the maximum of 10000 times for a controller to play one level before it is going to be evaluated. Level Generation track, instead of looking for the best AI player, focuses on making generators of engaging and suitable levels for the game and human players. The last one, Turing test track, gives a competition on human-like controllers as they are going to be judged by human spectators.

The gameplay implemented in this benchmark is similar to the gameplay of original Mario games, but it is comparatively simpler than the original version. The main objective in the gameplay is to navigate the Mario to reach the end of a level. Possible actions for navigating Mario are: left, right, up, down, jump, run/shoot.

During a level, Mario can interact with varied objects and items, for example, coins, mushrooms, fire flowers, normal and hidden blocks. Collecting coins, and killing enemies give additional scores to the player. Mario can be in one of three state: Fire, Big, and Small. Besides, there are certain types of enemies depending on the "difficulty setting".

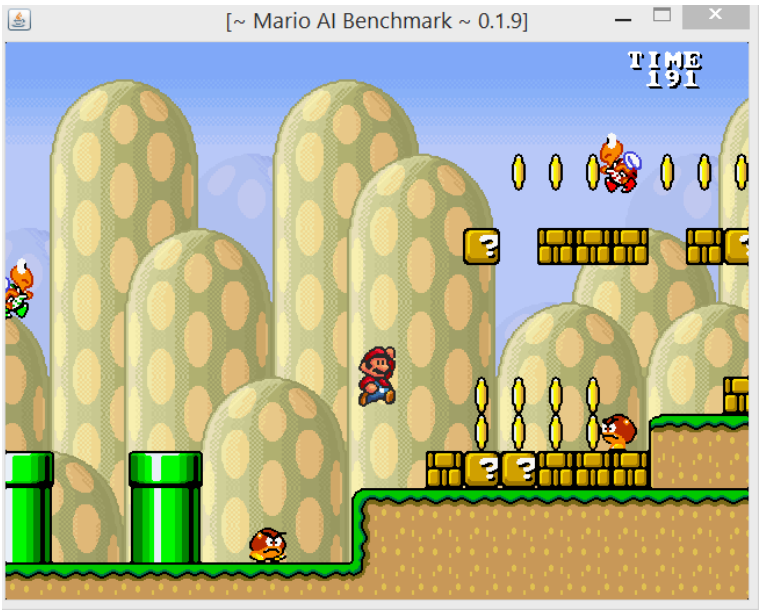

Figure 4.1: A screenshot of Mario AI benchmark

Difficulty can be set from 0 to 2. If difficulty increases, the number of enemies and gap (the abyss between two cliffs) are increased, and also more types of enemies appear. Mario will get hurt if he collides with an enemy; it means that if the current state of Mario is Fire (can shoot fireballs), and he gets hit by an enemy, his state degrades from Fire to Big (able to destroy breakable block). His state continues to degrades until he is Small, and he loses his life if getting hit by an enemy while in Small state. In addition, Mario will die if he falls into a gap regardless of his state.

The Mario AI Benchmark provides information such as observation of level, observation of enemies, Mario status, and enemy information for developing a computer agent. The future simulator is not provided by the benchmark. The observation is a 2 dimensions array with the size not larger than 22 by 22 cells (each cell has 16 by 16 pixels). Mario status contains Mario mode $(Fire, Big, Small)$, and information of all enemies appeared in the observation.

	Human			AI
Instruction	Free	Coin	Speed	A*
STAND	0.081 ± 0.057	0.192 ± 0.114	0.033 ± 0.029	0.000 ± 0.000
DOWN	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
RIGHT	0.046 ± 0.048	0.087 ± 0.045	0.020 ± 0.016	0.008 ± 0.005
LEFT	0.007 ± 0.006	0.042 ± 0.031	0.003 ± 0.003	0.006 ± 0.010
JUMP	0.011 ± 0.009	0.037 ± 0.045	0.003 ± 0.006	0.001 ± 0.001
SPEED/SHOOT	0.048 ± 0.047	0.076 ± 0.087	0.014 ± 0.011	0.000 ± 0.000
RIGHT-ALL	0.142 ± 0.093	0.220 ± 0.086	0.111 ± 0.057	0.141 ± 0.024
LEFT-ALL	0.016 ± 0.012	0.089 ± 0.038	0.007 ± 0.005	0.011 ± 0.027
JUMP-ALL	0.072 ± 0.047	0.153 ± 0.083	0.061 ± 0.037	0.024 ± 0.009
SPEED-ALL	0.130 ± 0.112	0.220 ± 0.187	0.104 ± 0.064	0.137 ± 0.033
OnGround	0.216 ± 0.105	0.425 ± 0.129	0.108 ± 0.058	0.045 ± 0.016
ChangeAction	0.099 ± 0.042	0.204 ± 0.052	0.061 ± 0.031	0.027 ± 0.020
JumpTimes	0.018 ± 0.008	0.037 ± 0.009	0.014 ± 0.007	0.011 ± 0.004
TimeSpent	0.503 ± 0.124	0.843 ± 0.152	0.259 ± 0.107	0.151 ± 0.051
Coins	0.352 ± 0.060	0.480 ± 0.060	0.270 ± 0.080	0.260 ± 0.063
Kills	0.404 ± 0.191	0.453 ± 0.189	0.222 ± 0.100	0.350 ± 0.097

Table 4.1: Human data characteristics

4.2 Human data analysis

We have done an analysis on human behavior in this game to obtain statistical data for our training our agent. Collecting human data takes time, and it is very costly. If we collect data from many human players, we need to categorize them by experience and level. Thus, we ask five human players to participate in because it is enough to see the differences between behaviors of human player and computer players. Five human players are given five levels in the Mario AI Benchmark with difficulty 0. Even with difficulty is set to 1, normal human players feel it hard to play, and they fail many levels. Moreover, if more levels with the same difficulty are given to human players, they will get bored so that their behavior may be different. Three instructions are given to human players to see if the human behavior changes for each objective. These instructions are: Free to play (*Free*), try to collect coins as many as possible (*Coin*), and run as fast as possible (*Speed*). The characteristics of human players are shown in table 4.1.

Only characteristics of human plays with difficulty 0 is shown in table 4.1. With difficulty is set to 1, even human players feel it is hard to clear a level, especially when Mario is not in Fire mode. Hence, with difficulty 0, human players are able to solve almost all levels.

In table 4.1, the average percentage and standard deviation of actions and some features are calculated. The *RIGHT-ALL* in the table means all action in which *RIGHT* button is pressed such as *RIGHT and JUMP*. Similar representation is used for the case of *LEFT-ALL*, *JUMP-ALL*, and *SPEED-ALL*. As mentioned in the previous section,

RUN or *SPEED* action is also *SHOOT* action to fire a fireball if Mario is in Fire mode. Features related directly to actions have the percentage value, which is the number of actions of the player in one level divided by the maximum number of actions allowed for each level, rounded to 4 number after the decimal point for more precise analysis because the maximum number of actions in a game play is 3000.

- Typically, in 2D action games, a character should move right to clear a level, so the average percentage of *RIGHT* action is larger than *LEFT* action.
- It is easily to see that the average percentage of *SPEED/SHOOT* action, in case of Free instruction and Coin instruction are given, is higher than in Speed instruction. It is possible to think that human players tend to be more careful so that they try to shoot and kill many enemies since the beginning state of Mario is *Fire*.
- Human players with Coin instruction pressed LEFT button the highest times comparing to the other two instructions. The reason might be collecting more coins requires Mario to go backward because some coins may be left while Mario traversing through a level.
- If human players are asked to play as fast as possible, they try to navigate the Mario to run and jump just to pass over obstacles and enemies. It results in the number of *RIGHT-ALL*, in which combinations of RIGHT, JUMP, and SPEED/SHOOT are taken into account, is significantly high, while only *RIGHT* action is low.
- Most of the time human players with Speed instruction use actions where *RIGHT* and *SPEED* are pressed at the same time. By excluding the action *SPEED/SHOT*, the remaining values of three instruction are 0.0818 (Free), 0.1444 (Coin), and 0.0902 (Speed). Then, dividing by the *TimeSpent*, the ratio of them are 0.24, 0.17, 0.34 accordingly, and the ratio between *RIGHT-ALL* and *LEFT-ALL* are 8.78, 2.46, 16.87. Thus, we can conclude that *SPEED* and *RIGHT* actions are used the most in Speed instruction.

Other than the number of actions, features such as how long Mario is on the ground, how frequently an action is changed, the number of jump times are also essential features. As mentioned above, Speed instruction may cause human players navigate Mario to jump over obstacles and enemies, so the duration of time Mario is on the ground is smallest, only 41% of the a time given to each level in average.

In general, human players change their actions not so frequently, around 23% to 25% in average. Besides, it is needless to say that the average number of collected coins is higher in case of Free and Coin instructions, and the average time human players spend for a level is shortest in case of Speed instruction.

The analysis on human behavior has showed that human players play in different styles based on a given instruction. Based on these features, a computer player is expected

to behave more like a human player if their average percentage of actions or features is similar or even almost the same as human players'.

Chapter 5

Evaluation and discussion

This chapter describes experiments of the proposed method and discuss the experimental results. To train our agent, we use the evolving neural network, which is is able to alter both weights and structure, because its performance is better than the fully-connected neural network as shown in the appendix A.1.

The setting used in the experiment is setting (b) in the appendix A.1. The number of input units is 15, and output units is 6 corresponded to actions in the Mario AI Benchmark. Maximum number of hidden unit is 20. Mutation chance is set to 0.5, no crossover is used. The population size is 50, and all solutions are paired every iteration. For each pair, $(2 + 10)$-ES strategy is used. By some preliminary experiments, as shown in the appendix, we figure out that this setting is currently the best setting to train at this time.

5.1 Performance test

We calculated gameplay statistic of human players, NeuroEvolution computer player without penalties, and the player used our approach. Human players are asked to play 5 different levels with difficulty 0 (the lowest difficulty). Hence, difficulty 0 is used for precise and valuable gameplay analysis.

The training phase of computer agents used the same 5 levels which human players are given. The training curve of computer players using only NeuroEvolution is shown in figure 5.1. Within 100 iteration, the computer agent is able to solve 5 training levels, and it is expected to solve more than just 5 levels which it is trained on. Nine penalties are used to calculate the total penalty. These features and its penalty weights are: $RIGHT\,(1)$, $LEFT\,(10)$, $RIGHT+SPEED\,(1)$, $OnGround\,(1)$, $ChangeAction\,(5)$, $JumpTimes\,(10)$, $Coins\,(1)$, $Kills\,(1)$, $IllegalAction\,(10)$.

Comparing NeuroEvolution and the proposed method, the best agent produced by NeuroEvolution without penalties is able to solve 5 trained levels within 100 iterations. The best agent in our proposed method is able to solve trained levels after 200 iterations. In addition, the total penalty continues to decrease as shown in figure 5.2 that means the proposed method is optimized to find solution, which has the percentages of feature

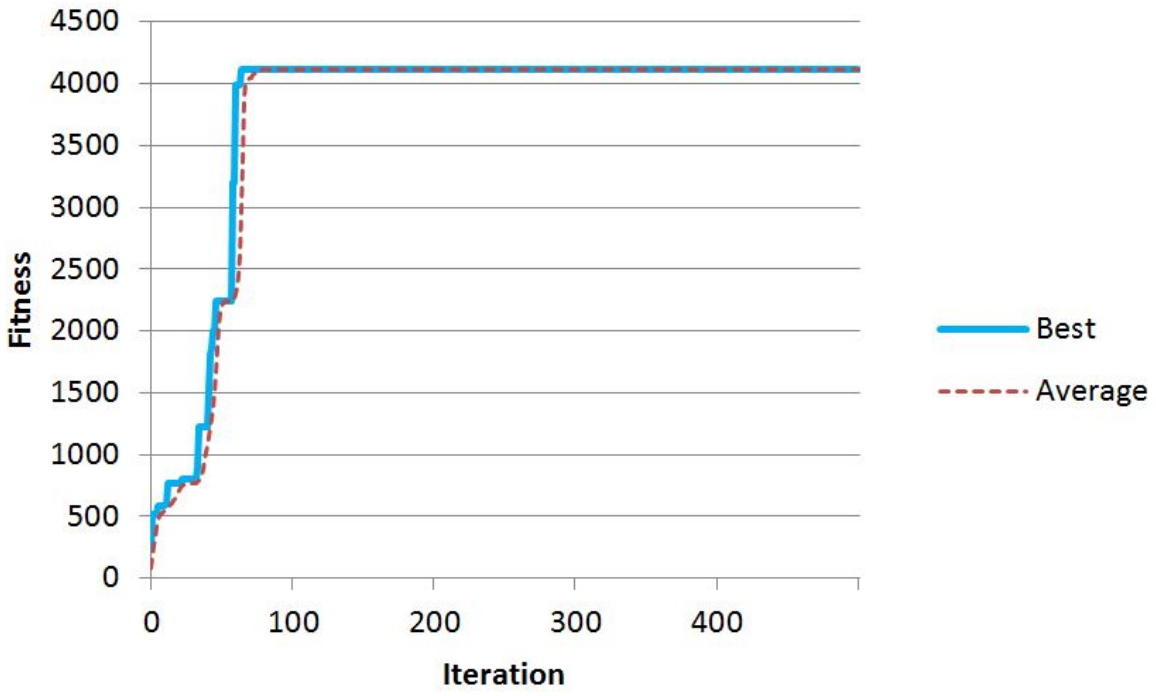

Figure 5.1: Fitness of the best candidate in the population while training NeuroEvolution computer agent.

similar to the given average percentage of human players.

For NeuroEvolution with Penalties, the results are shown in figure 5.2. According to the fitness calculation of the proposed method, three lines are presented in the figure. The light-blue light indicate the final fitness value after adding all penalty values. Specifically, the green dash line show the only the score of the agent, and the red dotted line is the total penalty value. According to the benchmark, if the agent can achieve 4096 in score, the level is cleared. Other than fitness, the total penalty maximum value depends on the number of chosen features for calculation. The ideal value for total penalty of the trained agent is 0, or maximum fitness value in other words.

The comparison between the average and standard deviation values of features is shown in figure 5.3. We let two agents, NeuroEvolution without penalties and our proposed method, play 60 levels. In case of score, the average score of our proposed agent is 2956.83, and NeuroEvolution agent is 3385.12. Therefore, our agent is not as strong as the agent, which employs NeuroEvolution without penalties, in case of average score.

For behavior, without penalty calculation, the computer agent seems to produce many illegal actions, and maybe only R+J+S action are used as in figure 5.3. It also changes actions too frequently.

By applying penalty into the fitness calculation, illegal actions are removed, the frequency of changing actions also decreases. The percentage of RIGHT button almost achieves as the average percentage of human players. Percentages of *Coins* and *Totalkills*

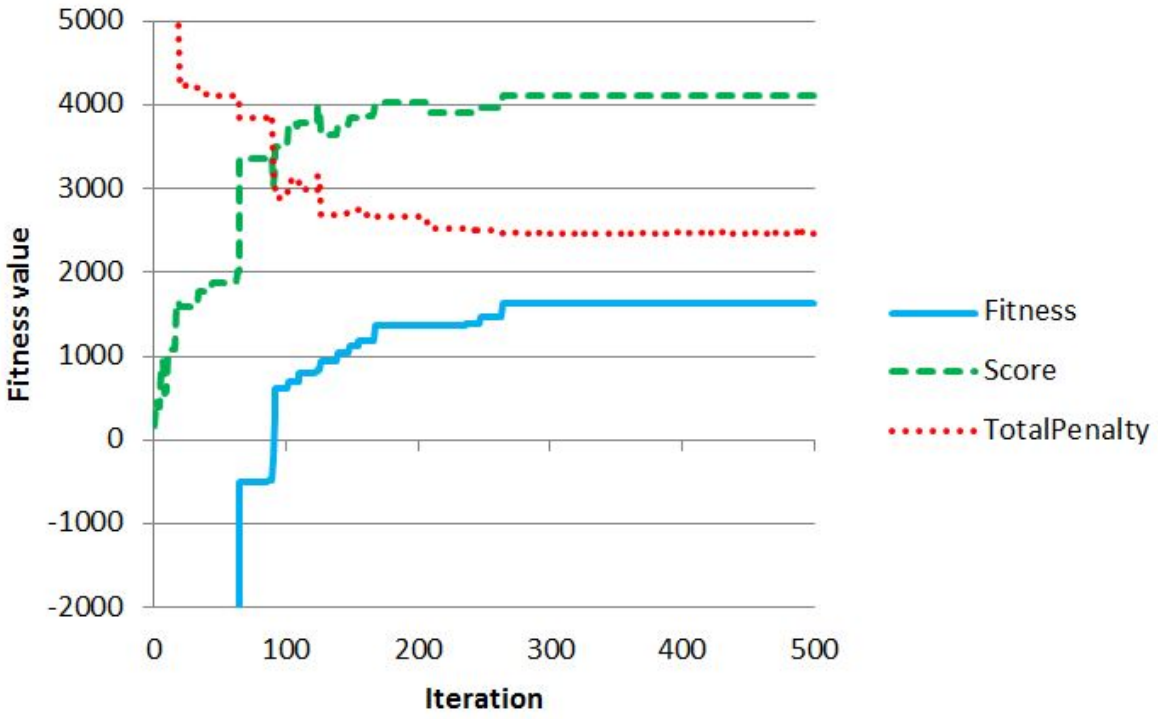

Figure 5.2: Fitness of the best genome while training NeuroEvolution + Penalties computer agent.

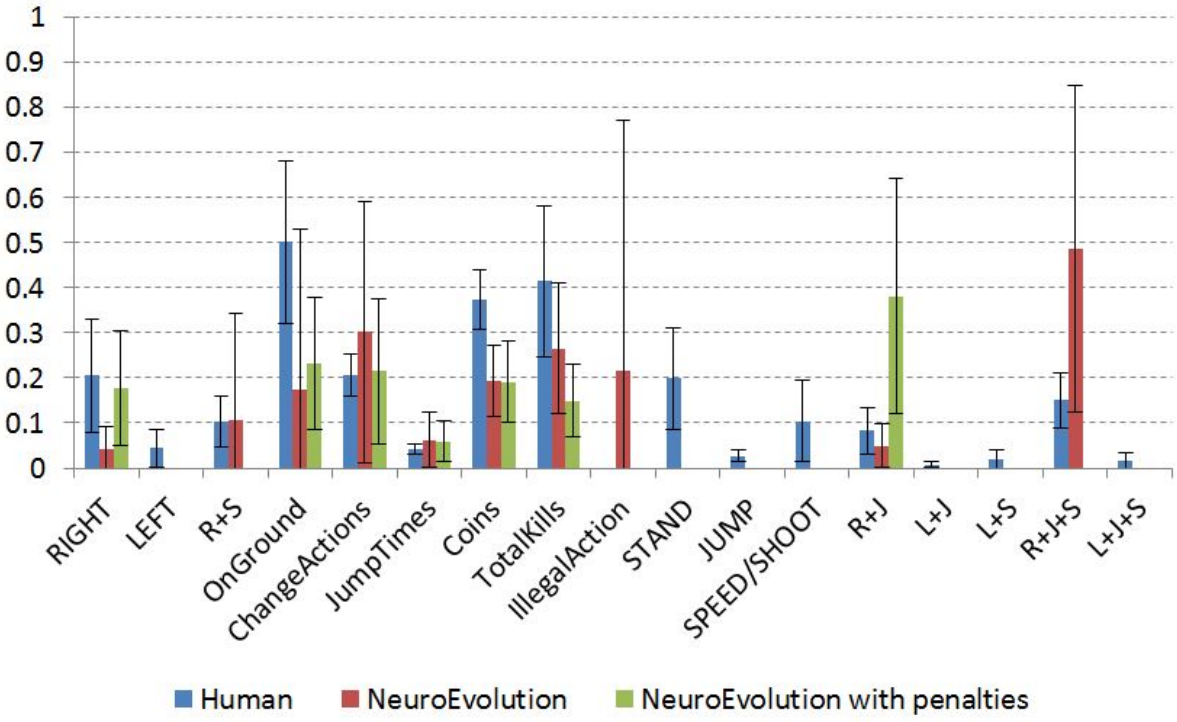

Figure 5.3: Comparison of average and standard deviation values of statistical features between human, NeuroEvolution agent, and our proposed method

can not achieve due to the weights of their penalties and the total number of coins and enemies in considered levels. As a result, percentages of features, which are considered in the penalties calculation, seems to approach human average percentages.

The average percentage of R+J button is extremely high, and there is an unbalance between other actions. However, the penalties of these features are not considered in the setting of the experiment. This problem can be avoided by including their penalties into the calculation.

There might be three problems in this result. One is the required time for optimizing total penalty is not enough. The second reason might be poor input features and penalty features, and the last one may come from the weight balancing between penalties.

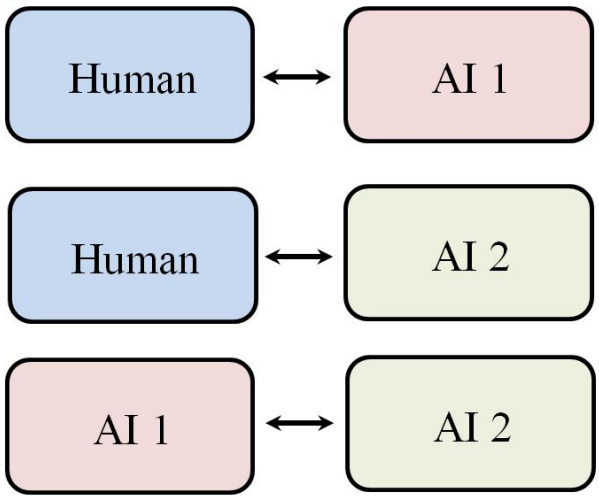

Figure 5.4: Matching agents for comparison

5.2 Human-likeness evaluation

To evaluate the human-likeness of our proposed method, we asked 16 human subjects to watch pairs of two videos. Each video contains a play of either a human player or a computer agent. An evaluation sheet is given for each human subject so that they can judge each video based on the questions in the sheet. By this, it is much easier for humans to evaluate the plays.

We recorded videos of three agents playing Super Mario Bros. game provided in the environment of the Mario AI Benchmark. Five random levels are chosen, and we let 5 human players play these level. Then, we let computer agent, which using only NeuroEvolution, play these five levels. Similarly, the computer agent, which is developed using our approach, also plays the same five levels.

The recorded videos are paired by type. We match a video of a human play to a video of the first AI ($AI1$) and the second AI ($AI2$) individually. We also compare AI1 and AI2 by match their videos together as shown in figure 5.4. We arrange this kind of video pairs in order to evaluate equally. We do not need to compare meaningless pairs such as Human-Human, AI1-AI1, or AI2-AI2. Because of the variance in judgments of human subject, the order of pairs differ in each video set, and also the order of the first video and the second video in a pair is randomly changed.

Each human subject is given a set of 5 pairs of videos. Every set includes 3 pairs of Human-AI videos and 2 pairs of AI1-AI2 videos. We have balanced the pairs of Human-AI videos so that the numbers of videos of AI1 and AI2 have the same probability to appear.

We also gave human subject an evaluation sheet. In the evaluation sheet, three questions are asked, and participants have to choose the option corresponded to their answers for each questions. The first question is about the skillfulness of the player in a

conducted video. The second question asks human subject to evaluate the human-likeness of the player. Five answer choices are given, from 1 to 5, for rating a player. Choice 1 indicates the lowest in considered ability, and choice 5 indicate the highest. In other words, if a skillfulness of a player is ranked 1, this player is considered to have poor skill, or to be a beginner. In contrast, ranked 5 player in skillfulness means this player is an expert. The similar rating is used for human-likeness.

In addition to two questions about skillfulness and human-likeness, a multiple choices about players' behavior are given. If human subjects have any comments or similar idea, they can tick those choices. The choices and results are shown in the appendix A.3. They can also give their own feedback or comments if they want.

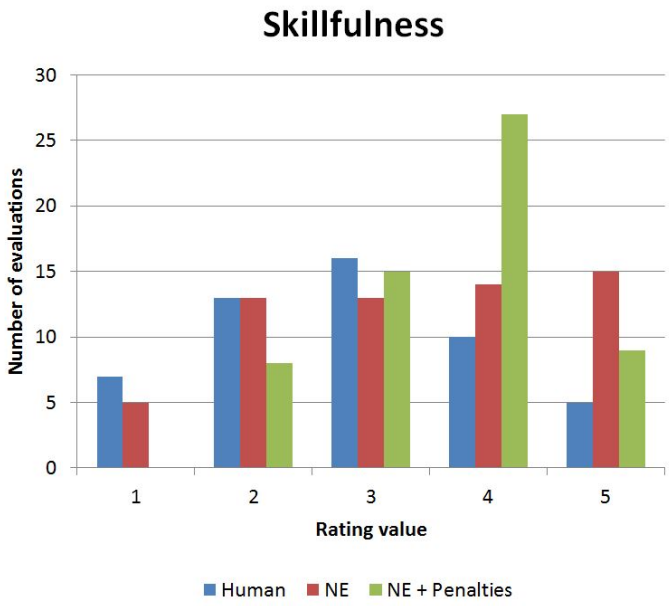

Figure 5.5: Skillfulness comparison

5.3 Experimental results

According to the evaluations from human subjects, we have summarized all the answers and comments. The comparison results are shown in figure 5.5, and 5.6.

Figure 5.5 shows the number of rating for each value from 1 to 5 corresponded to the skillful of each player. The blue column is the human players' performance, red columns is for the computer agent, which using only NeuroEvolution approach, and the green column indicates the performance of our proposed method.

The skillfulness of human players seems to have a Gaussian distribution. Human subjects ranked 3 for most of the game playing of human players, and 5 has lowest number of evaluations. Meanwhile, many plays from NeuroEvolution without penalties are ranked 5. Thus, the computer agent, which uses the NeuroEvolution without penalties, is the most skillful player, and human agent is the least skillful player. Comparing to proposed method, our approach is slightly less skillful than the NeuroEvolution computer agent, but more skillful than human players. Most of human subjects rated 4 for the skillfulness of our approach. It means that even our agent is still too strong for human players in general.

In average, the skillfulness of human players is 2.86. The computer agent, which was trained by NeuroEvolution without penalties, scores 3.35, and our proposed method NeuroEvolution with penalties scores 3.63. According to the evaluation from human subjects, the best solution in our proposed method was more skillful than the best solution in NeuroEvolution without penalties. Because of the penalties calculation, the behavior of the computer agent is less mechanical so that human subjects think that it is clever.

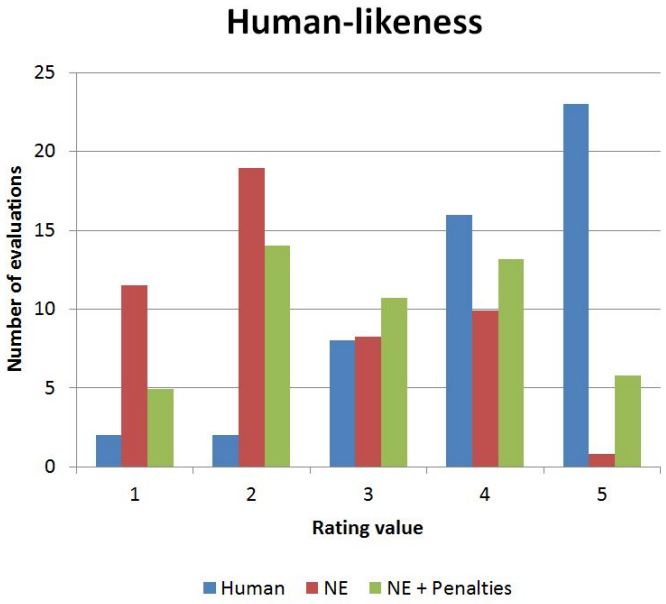

Figure 5.6: Human-likeness comparison

The next thing to be compared is the human-likeness. Our target to improve the human-likeness of computer agents. Figure 5.6 shows the human-likeness of human players, NeuroEvolution computer player, and our proposed computer player. It is easy to see that human players is the most human-like in this case. Human players are ranked 5 in most of the evaluations. The least human-like player is NeuroEvolution. Our proposed method has improved the human-like of NeuroEvolution a little bit, but it is still not as human-like as human players.

The average human-likeness value of NeuroEvolution computer agent is 2.38. Our proposed method scores 3.02. To think that the proposed method has improved the human-likeness of NeuroEvolution by 0.64 points in average. Unfortunately, the gap between humans' score and our agent's is still big, about 1.08 points. Thus, our method with current setting is not enough to achieved the human-likeness as human players.

The proposed method showed that the behavior of computer agents trained using NeuroEvolution is affected by the feature penalties. However, the human-likeness is not enough. The reason maybe the input features are not enough, so more detailed/richer information is needed. Besides, some human subjects raised a question "Why this player ignores the mushroom?" during the Turing Test (Some comments are shown in the appendix A.3). Therefore, if the computer player is able to recognize things, it might be better.

Chapter 6

Conclusion

6.1 Summaries

The goal of this research is creating human-like computer agent which can behave as intermediate human players. In order to create human-like computer players, two objectives are addressed. The first one is computer agents should be at the intermediate level of human players, and the other is that the human-likeness should be enough to pass the Turing Test.

The term *Penalty* is given to describe the differences between human players' behavior and a computer player's behavior. Penalty calculation is applied to limit mechanical actions, so the human-likeness of the computer agent might increase. Since making computer players to generate similar actions to human players requires human data. An analysis of human players' characteristics in the domain game should be done before training the computer agent. Unlike previous methods that human data are directly used for computation, our proposed method requires only statistical data from human players. Therefore, the calculation time significantly reduced. The experimental results shows that the proposed method is able to increase the human-likeness of NeuroEvolution, and the computer agent created by this method is able achieve the number of actions close to human players.

The experimental results showed that our method successfully enhances the human-likeness of a NeuroEvolution computer player. Moreover, the computer agent is able achieve the number of actions partly similar to human players, and average human-likeness is improved by 12.6723 %. Still, the human-likeness of the proposed method is not at the human players' level. Hence, if better features are defined with good training strategy and the training time is enough, the expected behavior of computer players may be even more human-like.

6.2 Future works

The proposed method shows its effect on the behavior of computer agents. To improve the efficiency of the training and search process, richer information and information of the past are required. Moreover, biological constraints and influenced map of dangerous areas may also need to be considered. Future works will focus on pre-processing of input features, and different structures of neural network, which are able to represent past information, will be further investigated.

Bibliography

[1] Michele D Dickey. "Engaging by design: How engagement strategies in popular computer and video games can inform instructional design". In: *Educational Technology Research and Development* 53.2 (2005), pp. 67–83.

[2] Dario Floreano, Peter Dürr, and Claudio Mattiussi. "Neuroevolution: from architectures to learning". In: *Evolutionary Intelligence* 1.1 (2008), pp. 47–62.

[3] Nobuto Fujii et al. "Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints". In: *Advances in Computer Entertainment*. Springer, 2013, pp. 61–76.

[4] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[5] Philip Hingston. "A turing test for computer game bots". In: *IEEE Transactions on Computational Intelligence and AI in Games* 1.3 (2009), pp. 169–186.

[6] Kokolo Ikeda. "Exemplar-based direct policy search with evolutionary optimization". In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. Vol. 3. IEEE. 2005, pp. 2357–2364.

[7] Sergey Karakovskiy and Julian Togelius. "The mario ai benchmark and competitions". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 55–67.

[8] Juan Ortega et al. "Imitating human playing styles in super mario bros". In: *Entertainment Computing* 4.2 (2013), pp. 93–104.

[9] Michael T Rosenstein and Andrew G Barto. "Robot weightlifting by direct policy search". In: *International Joint Conference on Artificial Intelligence*. Vol. 17. 1. Citeseer. 2001, pp. 839–846.

[10] Jonathan Schaeffer and Aske Plaat. "Kasparov versus deep blue: The re-match". In: *ICCA Journal* 20.2 (1997), pp. 95–101.

[11] Jacob Schrum, Igor V Karpov, and Risto Miikkulainen. "UT^2: Human-like behavior via neuroevolution of combat behavior and replay of human traces". In: *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE. 2011, pp. 329–336.

[12] Jacob Schrum, Igor V Karpov, and Risto Miikkulainen. "Human-like combat behaviour via multiobjective neuroevolution". In: *Believable bots*. Springer, 2013, pp. 119–150.

[13] Noor Shaker et al. "The turing test track of the 2012 mario ai championship: entries and evaluation". In: *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE. 2013, pp. 1–8.

[14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[15] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[16] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. "The 2009 mario ai competition". In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE. 2010, pp. 1–8.

[17] Alan M Turing. "Computing machinery and intelligence". In: *Mind* 59.236 (1950), pp. 433–460.

Appendices: Some preliminary experiments to find a good setting

To verify NeuroEvolution approach in the problem of human-like computer player in platform games, we have trained a computer agent to enable it to clear levels in Super Mario Bros. as human an intermediate player.

A.1 Fully-connected vs Evolving neural networks

First of all, a fully-connected neural network architect is used for tuning parameters for NeuroEvolution algorithm. The Mario AI Benchmark is used, so the number of input units depends on the setting of the observation and some additional information. In the first trial, 5x5 observation is used, and additional inputs are status information of Mario, which are Mario is on the ground, and is able to jump or not. For that setting, the number of inputs units is 27. Output units are fixed to 6 units corresponded to 6 individual buttons. One training level and 500 iterations are used. The setting of parameters in NeuroEvolution is: mutation chance is set to 0.5, pertubation chance is 0.9, and the step size for perturbation is 0.01.

Figure 1 show the performance of computer agent. The blue line indicates the average performance of all the population, and red line represents the performance of the best organism in the population. Since the fitness function is only the score return from the benchmark, a fitness value, which is larger than 8000, means a computer player is able to clear a level. The results in the figure show that, within 500 iterations, the computer agent is able to clear a level. However, if population is set to 300 and only 5 hidden units are used ($pop300hid5$), the performance is lower than the setting, which 10 hidden units are taken into account, but it can still clear a level. On the contrary, if population size is set to 100, the computer agent is not be able to clear a level within 500 iterations. The problem is that 100 population increase the chance the solution fall into the local optimum.

Next, we apply evolution strategy, says $(2 + 10)$-ES, to avoid local optimum. In the previous settings, mutants of better solutions replaced worse solutions, so new solutions have similar structure to some good solutions in the current iteration. As a result, diversity of population decreases, and the chance of finding global optimum decreases. By using evolution strategy, this problem can be avoid since new mutants will replace their parents if their performance is better. In this setting, a pair of 2 solution is randomly cho-

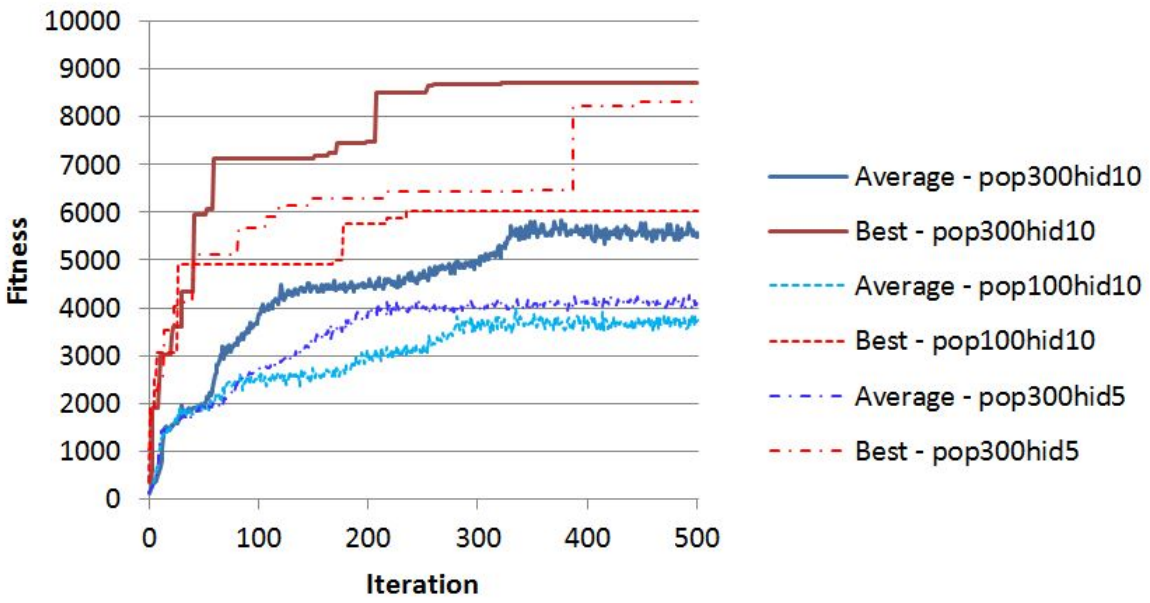

Figure 1: Performance of Fully-connected network

sen, and 10 mutants are generated for each solution. These new solutions are evaluated, and the best solution will replace its parent.

Figure 2 show the training curve of 3 trials. The training time of 2500 iterations for each trial is around 1.5 days. Yet, the computer agent is not able to clear a level.

Fortunately, the evolutionary strategy works well in case of evolving neural network, in which the number of nodes and connection can change. Figure 3 shows the performance of evolving neural network. The computer agent can clear a level within 500 iterations. Moreover, since the evolving ANN has no connection and node from the beginning, it does not require checking every connections for perturbation. Thus, the training time decreases to the average of 8 hours.

A.2 Input settings

Another test is on input features. There is a little change in the fitness function of this setting. Fitness function now uses only traversed distance from the beginning of a level, so the maximum fitness score is 4096. The input features are also different in the following tests. One test uses inputs from 7x7 observation, Mario status $(isAbleToJump, isAbleToShoot, isOnGround, MarioMode)$, and information of 2 nearest enemies $(type, distance, angletoeachenemy)$. The other setting use 3 areas, as shown in figure 4 , and information from 2 nearest enemies. Three areas are: (1) 4 cells in front of Mario, (2) spaces between Mario and the ground, and (3) a couple of column in front of Mario to the bottom of the observation. The first area is to check whether or not there is any obstacle that blocks Mario to process forward. The second area is to measure the

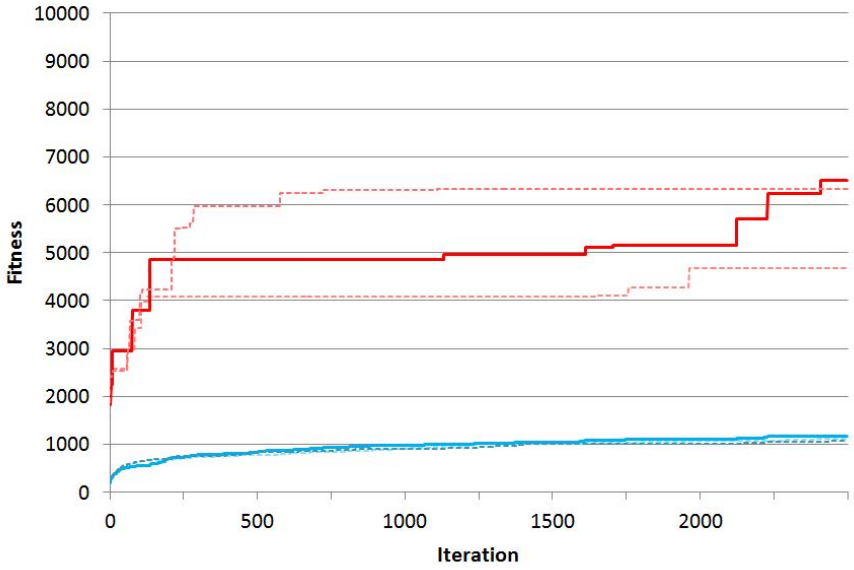

Figure 2: Performance of Fully-connected network using $(2 + 10)$-ES; population size 100.

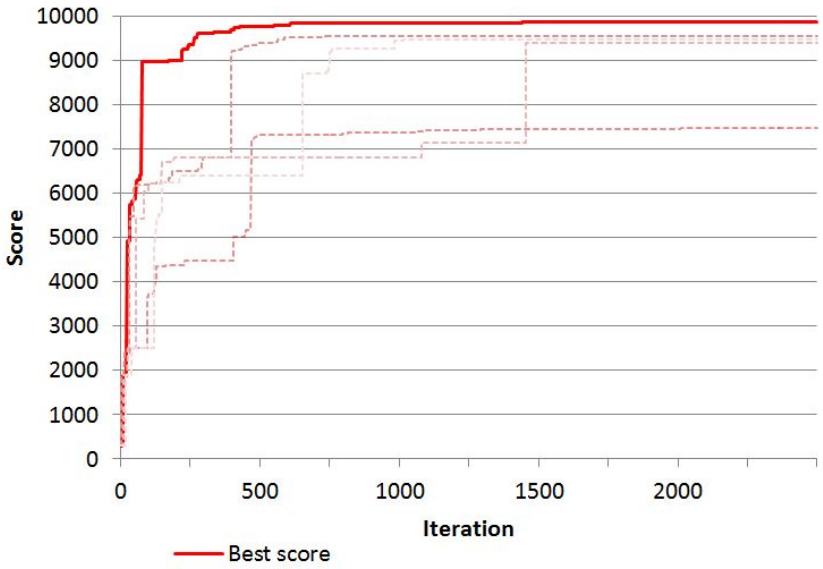

Figure 3: Performance of evolving network using $(2 + 10)$-ES; population size 100.

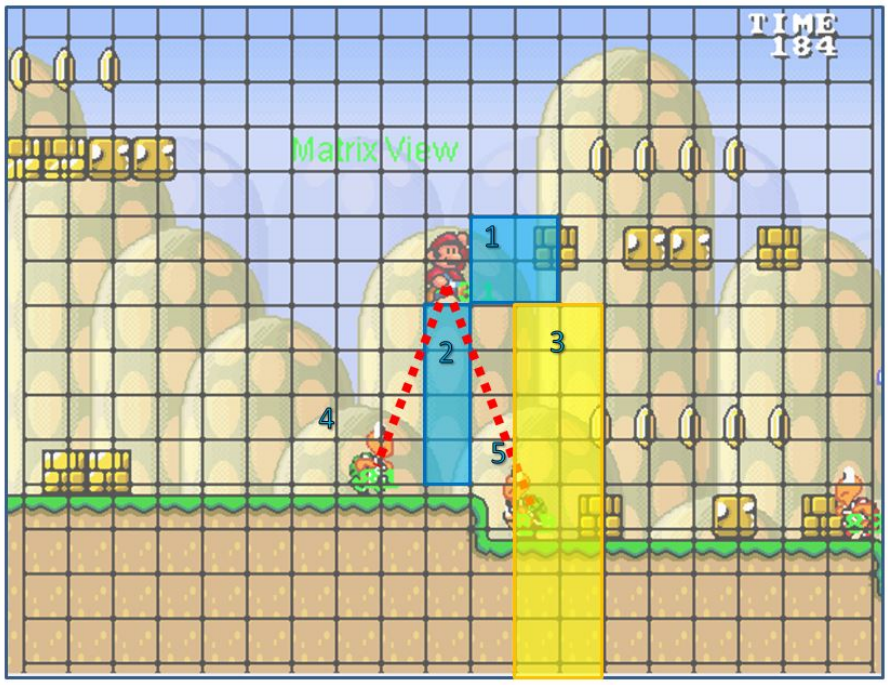

Figure 4: New input feature setting

distance of current position of Mario to the ground, and the third area is to check if there is any gap in front of Mario. These 3 areas return the value $\{true, false\}$ accordingly, so the search space that NeuroEvolution needs to explore dramatically decreases.

Figure 5shows the performance of two settings mentioned above. The numbers of input units are 59 and 15 corresponding to setting (a) and (b). In setting (a), 7x7 observation, Mario status, and 2 nearest enemies are used. In setting (b), inputs obtained from three areas, Mario status, Mario positions (x, y), and 2 nearest enemies are used. The position information of Mario are not needed in setting (a) because it is included in the observation. To make the training phase becomes fair, we included information of Mario's position in setting (b).

The results show that using input obtained from three introduced areas significantly improve the performance of training. The computer agent is able to clear a level just after 30 iterations. There is a drawback that it lack of richer information from the observation. At least, an intermediate computer player is achieved using this setting. Therefore, we fit this setting and apply penalty calculation to the experiments for evaluating the proposed method.

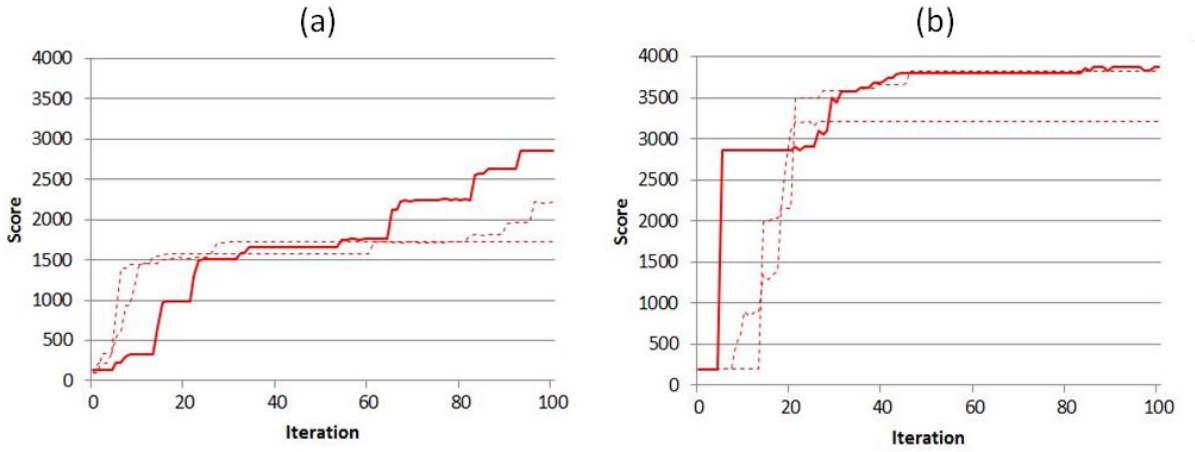

Figure 5: Performance comparison between old and new input feature

A.3 Comments from human subjects

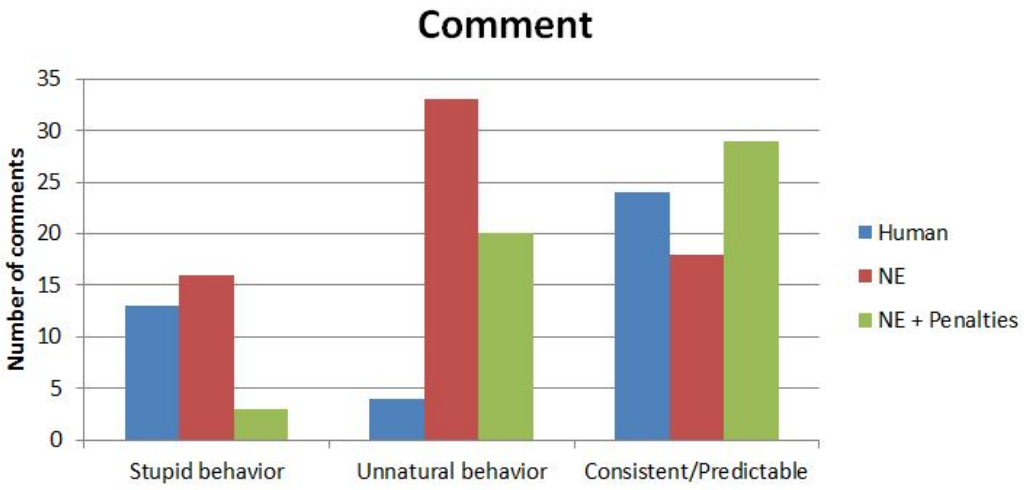

Figure 6: Comments received from human subjects after the Turing Test