Infinite Mario Bross AI using Genetic Algorithm

Ng Chee Hou, Niew Soon Hong, Chin Kim On, and Jason Teo
Evolutionary Computing Laboratory
School of Engineering and Information Technology
Universiti Malaysia Sabah, Jalan UMS, 88400, Kota Kinabalu, Sabah, Malaysia
kimonchin@ums.edu.my, jtwteo@ums.edu.my

Abstract—**Evolutionary Algorithm (EA) is commonly used to generate optimal Artificial Intelligence (AI) controller. It is a technique used to enhance the performance of generated controller. EA enables the system to evolve, to adapt and learn to give a better output. The implementation of EA into 2D game is not something new. Researchers used gaming platforms to test their own ideology or proposed algorithms. In this paper, a finite state machine which suitable to be used for Infinite Mario Bros game is proposed. The Genetic Algorithm (GA) is used along with the proposed finite state machine to evolve an AI agent that is capable to pass some levels of the game. The experimentation results showed that the finite state machine evolved with GA is able to create a competitive game bot that can pass through at least 3 levels of different game maps. The generated AI controller can guarantee to accomplish the tasks for some levels.**

Keywords-component; Artificial Intelligence (AI); Finite State Machine (FSM); Genetic Algorithm (GA); Infinite Mario Bros; Evolutionary Algorithms (EAs)

I. INTRODUCTION

The games play in Super Mario Bros consists of moving the player-controlled character "Mario", through two-dimensional levels, which are viewed sideways [1][2][3]. Mario can move forward, backward, duck, jump, optionally "go faster", "shoot fireballs" and all of these actions are depending on which state Mario is in. Gravity feature acts on Mario, making it necessary to jump over holes to get past them. Mario can be in one of the three states: "small state" (at the beginning of a game), "big state" (can crush some objects by jumping into them from below), and "fire state" (can shoot fireballs). The main goal of each level is to get to the end of the level, which means traversing it from left to right. Auxiliary goals include collecting as many as possible of the coins that are scattered around the level, clearing the level as fast as possible, and collecting the highest score, which in part depends on number of collected coins and killed enemies. When a human player plays the game, the view side is only small part of the current level from the side, with the screen centered on Mario. Still, this view is not easy for human player to predict the next appropriate step because sometimes human player is hard to adjust the movement of Mario and difficult to catch up the correct timing because of nervous or some other reasons.

This is the difference between human players and AI agent because the controller can visualize well by percept the surrounding environment and receive different inputs from sensors and able to perform perfect movement by using computation. One of the good reasons to use Evolutionary Algorithms (EAs) to play the game is to test the Artificial Intelligent (AI) agent whether the proposed learning algorithms and function representation that used are capable of. A related reason is that researchers like to compare the performance and effectiveness of different learning algorithms and function representations. There are a large number of AI algorithms available but their relative effectiveness differs widely. Some researchers proofed their algorithm worked well in some games. However, every game requires different skills to play, and poses different learning challenge. Hence, these reasons have motivated researchers to apply different algorithms in different game genres.

The application of EAs into gaming is not something new [4][5][6]. Typically, EAs attempts to replicate, in software, portions of the biological phenomenon of evolution. EA is just evolution in code by start with a population of chromosomes, use natural selection to choose the best ones, mate, produce offspring, have mutation in offspring, use natural selection to choose the best children, rinse and repeat. Eventually, an AI agent will be evolved out and it is able to seek out the most appropriate strategies in order to achieve the task according to the environment after getting the experience of the previous cases. The performance of the AI agent after applies the EA and the function representation is anticipated.

Since the emergence of the Mario series game, there are many competitions that let the programmers to participate to develop an agent that gets as far as possible or proceeds as many level as possible. In the game, the AI Mario acts based on the environment and to proceed out output in order to respond to the environment. The research starts to implement to the main protagonist, Mario by applying EA techniques in hopes that it can creates some forms of intelligence in the AI Mario. A number of researches have been done and some of the researchers implemented different technique to the Mario, like evolving it to be a better and smart bot, or even just cutting short of the process in evolved the Mario by using EA techniques. There is a Mario AI competition @CIG from the internet that needs the participators to create an AI for the Mario game. Due to literatures review [7][8][9][10][11], there is still no research has been conducted using finite state machine hybridized with Genetic Algorithm to evolve the required Mario AI. Hence, this motives us to create an AI agent that uses FSM and GA.

The rest of the paper is structured as follow: Section II provides some discussion on the Infinite Mario Bros platform,

while Section III covers the methodology approaches used. Section IV provides Experimental Setup. Then, Section V discusses on Results and Discussion. Lastly Conclusion is included in Section VI.

II. INFINITE MARIO BROS

The game engine used in this research is named as Infinite Mario Bros [12]. It is slightly differs compared to Nintendo's classic platform game Super Mario Bros. Infinite Mario Bros was made by Markus Persson for a Super Mario themed java programming contest [12]. The game engine has been modified to random generate level which means each time the game is started, the level is randomly generated by traversing a fixed width and adding features (such as blocks, gaps and enemies) according to certain heuristics. The main goal of each level is to get to the end of the level by traversing from the left most to the right most (of course, cannot hit by the enemies during traversing). The auxiliary goal include collect as many as possible coins that are scattered around the level (collected 100 coins Mario will be rewarded one extra life), clearing the level as fast as possible (there is time limit for each level). The gaps and moving enemies are the main challenges of Mario. If Mario falls down into the gap, he loses a life immediately. If he touches an enemy, he gets hurts. Hurts mean the Mario will shift to the lower state, if he is in "fire state", he will shift to "big state" and if he gets hurts again he will reduce to "small state" and if he gets hurts during "small state" then he will loses a life. Mario can jump, however, if he jumps and landed on the enemies above, the outcome is dependent on the enemy: (1) most enemies (e.g. goombas) die from this treatment, (2) others (e.g. piranha plants) are not vulnerable to this and proceed to hurt Mario, (3) finally, turtles withdraw into their shells if jumped on, and these shells can then be picked up by Mario and thrown at other enemies to kill them. The cannon shell will be dropped to the land if Mario landed above it, however Mario also can choose dodge by press down to avoided by hit by the cannon shell. The items such as coins, mushroom and flower are hidden inside the block and only appearing when Mario jumps at these blocks from below.

The main different between Super Mario Bros and Infinite Mario Bros is the difficulty of the level. As discussed previous, each level is randomized before the game start. Hence, there is impossible to create simple AI that can pass all levels. In our work, we prefixed in early and used only one of the maps to test our proposed AI. Therefore, there is no comparison against a wide variety of Mario playing algorithms will be discussed. We aimed only to investigate the feasibility of the proposed combination of FSM and GA approach to the problem.

III. ALGORITHM APPROACHES

As previously discussed, a combination of FSM and conventional GA approaches are used. Hence, the following section provides brief discussion of these two approaches.

A. *Finite State Machine (FSM)*

In a state machine, there are states that are associated with some kind of actions or behaviors, and agent will occupied this state and perform the same action or behavior. Each state is connected together by transition. The transitions are some trigger or conditions that need to be met in order for the agent to change to different states. In game AI, it is called the finite state diagram that constituent a number of state and directed transition between them. Typically, there are two types of FSM; (1) deterministic FSM and (2) non-deterministic FSM. The original simple finite state diagram is the deterministic FSM. The state transition can be predicted if input and current state is given. In non-deterministic FSM, the transition cannot be predicted. Means that the transition from current state to another state cannot be known until inputs are received.

The implementation of FSM in game design is very common in gaming industry. In the game, the Character AI can be modeled as a sequence of mental states. The World events can force a change in state. For example, the games Thief 3 applying stack FSM in its game design. Stack allows AI to move back and forth between states and Leads to more realistic behavior without increasing FSM complexity. For more advanced FSM like the Fuzzy State Machines have the features of degrees of truth allow multiple FSM's to contribute to character actions. This type of FSM is use in create a game AI that collectively cooperate together to achieved single goal such as sports game like FIFA Football game or any real-time strategy war game like Red Alert and General are using this FSM during the design of the game AI [13]. In other research, Chang et al [14] have successfully proofed that FSM work well in Unreal Tournament (U) First Person Shooting (FPS) game. The researchers combined FSM with Evolutionary Programming to generate AI bot that can fight in a customized map. Non-deterministic finite state machine is chosen and used in this project. Non-deterministic finite state machine has more than one trace for each input string hence the state transition cannot be predicted. When at the state that has more than one trace, a transition will be chosen randomly by the machine. Figure 1 shows the proposed FSM for this research used.

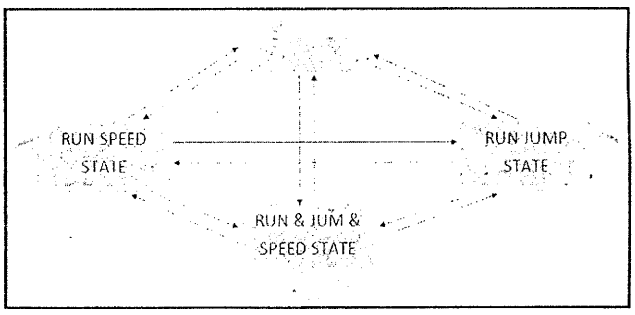

Figure 1. Finite state machine designed for Infinite Mario Bros AI

Fig. 1 shows the designed structure for the four states of FSM. At first all the transitions and states are set and resemble to a tradition FSM. The four states are as follow:

- Run State - AI agent will move forward or go to the right hand side of the game.
- Run Jump State - AI agent will move forward and jump over in any necessary condition.
- Run Speed State - AI agent perform move forward and go faster.
- Run Jump Speed State - AI Mario will perform move forward, jump and run faster actions in once.

To change the current state to a different state in the FSM, a trigger has to be made, and that is when the transition takes place. There are a total 16 lines of transitions in the FSM that connected the states. The inputs of the transition are as follow.

- Seen an enemy
- Seen an obstacle
- Seen nothing
- Seen enemy & seen hole
- Seen enemy & seen obstacle
- Seen hole & seen obstacle

The input mention above will be place into the respective transition line, when an input triggers the state transition will happens. The states are represents with numbers from 0 to 3 as a representation in the codes.

The generated Mario AI agent needs to think as fast as possible within 40 milliseconds before next action takes place. Since Mario can perform one of the four states given, and do many these things on combination, the AI agent has more than a handful of possible moves that it can choose to do at any one time. It wills call the function reside in the representation and that particular associated action that presents in the function will be trigger. For the AI agent to switch between states, triggering a transition is required. The pseudo code given below shows a simple trigger action.

```
Switch (State):
case 0: //RunState():
        if (seen nothing)
            state = 0;
        else if (seen enemy)
            state = 1;
        else if (seen enemy || seen obstacle):
            state = 2;
        else if (seen enemy || seen hole);
            state = 3;
      break;
```

Pseudo code given above shows how to change state from RUN State to others state. Initially, current state is in the RUN state. If the agent receives input 0, that means an enemy found, the system will trigger the transition to change state to "1" which means RUN JUMP state. If there is no enemy seen, the agent will continue on with the current state. All 16 transitions were not previously defined early. The transition represents the controller chromosome. Hence, it will be evolved by the Genetic Algorithm.

B. Genetic Algorithm (GA)

Genetic Algorithms were developed by Prof. John Holland and his students at the University of Michigan during the 1960s and 1970s [15]. It is a model of machine learning which derives its behavior from a metaphor of the processes of evolution in nature. The crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by these two operators. Processes loosely based on natural selection, crossover, and mutation are repeatedly applied to a population of binary strings which represent potential solutions. Over time, the number of above-average individual's increases, and highly-fit building blocks are combined from several fit individuals to find good solutions to the problem at hand. Outline of the GA used is described below.

- [Start] Generate random population of n chromosomes (suitable solutions for the problem)
- [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
- [New population] Create a new population by repeating following steps until the new population is complete
- [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
- [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
- [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
- [Accepting] Place new offspring in a new population
- [Replace] Use new generated population for a further run of algorithm
- [Test] If the end condition is satisfied or at least one individual has the desired fitness and enough generation have passed, terminate, and return the best solution in current population
- [Loop] Go to step 2

The termination conditions are:

- Gaming time reach 0.
- Mario loses all lives.

- Mario reached end of the level.
- Generation reached 500.

Typically, the transition of the FSM has been previously prefixed in a common. Nevertheless, for this project, the transition of the FSM has been chosen to be evolved instead of hard code the transition criteria. Hence, GA acts as an adjustment system to figure out which transition could be best suit the state. The individual chromosome is represented by 16 integers which are depended to the following rules.

- Seen an enemy == 0
- Seen an obstacle == 1
- Seen nothing == 2
- Seen an enemy & hole == 3
- Seen an enemy & obstacle == 4
- Seen a hole & obstacle == 5

Each of the genotypes is associated with a transition line that will react based on the received inputs. Typically, Uniform Crossover and Uniform Mutation have been utilized during the optimization processes.

IV. Experimental Setup and Fitness Function

The crossover rates used are in between 60%, 70%, 80% and 90%. The mutation rate used is 10%. There are a total of 10 individuals involved. Each run is limited for 500 generation only. Each of the generated optimal solution is tested 10 times for an average score.

A simple fitness function is used to evaluate each of the individual. The fitness function is represented as below.

Fitness = distancePassedPhys sov.distance*

This means the longer the AI agent move forward, the higher the fitness value it obtained. Hence, this simple fitness function represents simple Mario AI behavior. It is not necessary for Mario AI to either kill any enemy or collect any coin as mark/point is not accumulated during game plays. However, time is one of the invisible important feature even it is not included in the fitness function. Mario AI still has to move quietly to reach the goal otherwise, that particular individual will be considered as failed child if the time reach zero before it reached the goal.

V. Results and Discussions

The experimentation results obtained can be simplified as tabulated in Table I below.

TABLE I. Experimentation Results Obtained After 10 Runs

Mutation Rates	Average Success Rate	Min Generation	Max Generation	Average Scores
0.6	70%	47	353	2785
0.7	60%	55	410	2215
0.8	80%	27	477	3619
0.9	60%	17	89	3214

Table I clear shows, the controllers evolved for crossover 80% generated highly promising result. It reached average score of 3619 whilst the actual maximum score is 4096. Minimum generation used is 27 and maximum generation reached 477. Overall, highest successful rate is 80%, the controllers evolved with 80% crossover rate. Minimum generation involved is 17, the controllers evolved with 90% crossover rate. Minimum average score is 2215, the controllers evolved with 70% crossover rate. Fig. 2, 3, 4, and 5 below show four graphs generated from random selected individuals, respectively from four different rates of crossover experimentation.

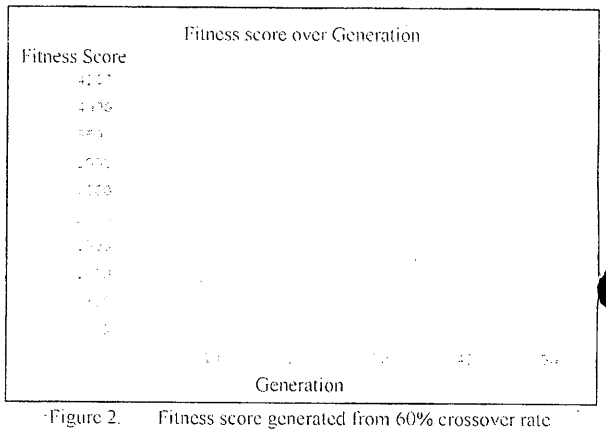

Figure 2. Fitness score generated from 60% crossover rate

Fig. 2 shows fitness over generation for individual generated from 60% crossover rate. Fig. 2 shows, the Mario AI had been successfully evolved for the required behavior during generation 47. The fitness score was sudden increased during generation 40. Then, it increased again during generation 46 and reached optimal fitness score during generation 47. This shows the Mario AI learned how to pass a level.

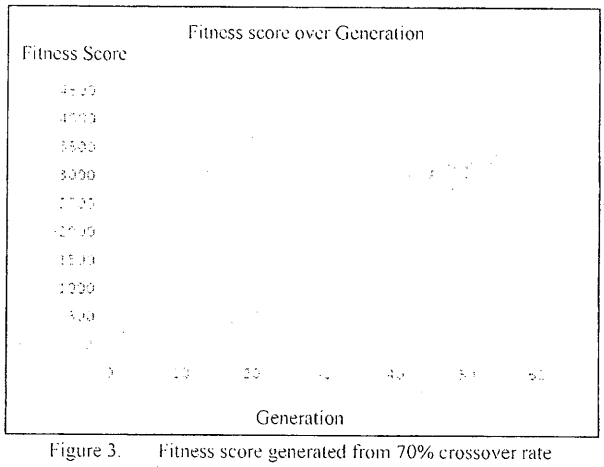

Figure 3. Fitness score generated from 70% crossover rate

Fig. 3 shows the fitness scored slightly increased during generation 41. Then, it increased again during generations 44 and 46. However, there were sudden drops on generations 45 and 47. Nevertheless, the solutions maintained after generation 48 and it increased again and reached maximum fitness during

generation 55. The sudden drop problem has been overcome with elitism inclusion during the optimization processes.

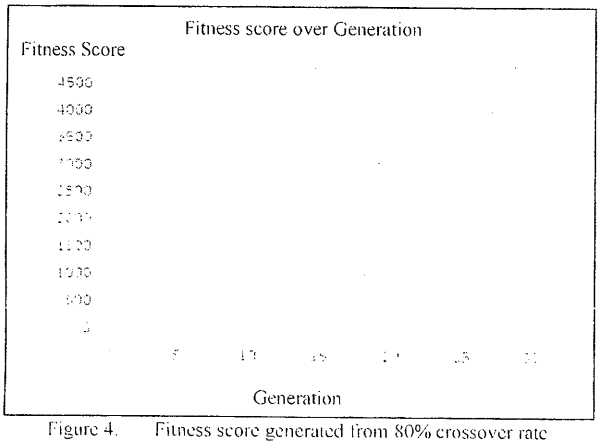

Figure 4. Fitness score generated from 80% crossover rate

Fig. 4 shows there is a sudden big jump during generation 16 to 18. However, the fitness score decreased again to 1700 after generation 19. Then, there is a sudden jump again during generation 27 and optimal solution found during that particular generation.

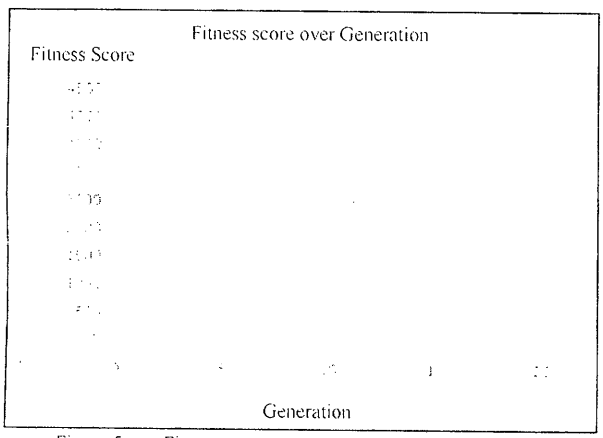

Figure 5. Fitness score generated from 90% crossover rate

Fig. 5 shows the solutions were maintained no change until generation 12. There is a sudden increase of fitness during generation 13. Then, the fitness score drop again during generation 14. But, it increase again during generation 17 to reach optimal solution score.

Typically, there were two simple experiments conducted for the generated optimal solutions. The first test had been conducted purposely to verify the effectiveness and efficiency of the generated controllers in different level of the Mario game. The experimentation results showed that the generated Mario AI was unable to perform well in different level. It happened due to the changes of the environment. There is a big gap for the difficulty of the level. Then, the experiment had been conducted to test the feasibility of the algorithm and method used in this study. The experimentation results found that the algorithms can be hybridized and used to evolve Mario AI in different level. However, only independent Mario AI could be generated to pass every different level.

VI. Conclusion

We have successfully developed simple AI controller which capable to play the Infinite Mario Bros. The generated controllers performed well and optimal solutions could be generated with 80% crossover rate used. Hence, this proofs that the proposed finite state machines can be combined with Genetic Algorithm to generate promising controller. Nevertheless, there are still rooms to be improved as the controller has limited its capability to work and perform well in certain environment/level.

References

[1] Fox, Matt (2006). The Video Games Guide. Boxtree Ltd. pp. 261–262.

[2] Ellis, David (2004). "A Brief History of Video Games". Official Price Guide to Classic Video Games. Random House. p. 9.

[3] Julian Togelius, Sergey Karakovskiy, Jan Koutník and Jürgen Schmidhuber. Super Mario Evolution. 2009 IEEE Symposium on Computational Intelligence and Games. 156-161.

[4] T. Schaul and J. Schmidhuber, "Scalable neural networks for board games," in Proceedings of the International Conference on Artificial Neural Networks (ICANN), 2008.

[5] C. Bateman and R. Boon, 21st Century Game Design. Charles River Media, 2005.

[6] G. N. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," Applied Artificial Intelligence, vol. 21, pp. 933–971, 2007.

[7] K. Compton and M. Mateas, "Procedural level design for platform games," in Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE), 2006.

[8] C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland University of Technology, Brisbane, Australia, 2008.

[9] J. Marks and V. Hom, "Automatic design of balanced board games," in Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE), 2007, pp. 25–30.

[10] T. W. Malone, "What makes computer games fun?" Byte, vol. 6, pp. 258–277, 1981.

[11] G. N. Yannakakis and J. Hallam, "Game and Player Feature Selection for Entertainment Capture," in Proceedings of the IEEE Symposium on Computational Intelligence and Games. Hawaii, USA: IEEE, April 2007, pp. 244–251.

[12] Markus Persson. Infinite Mario Bros. http://www.mojang.com/notch/mario/. 2011.

[13] Jarret Raim. Finite State Machine In Game, from: www.cse.lehigh.edu/~munoz/CSE497/classes/FSM_In_Games.ppt. 2011.

[14] K.-T. Chang, K.-O. Chin, and J. Teo, "The Evolution of Gamebots for 3D First Person Shooter (FPS)," in press. The Sixth International Conference on Bio-Inspired Computing: Theories and Applications, 2011.

[15] A.E. Eiben, and J.E. Smith, Introduction to Evolutionary Computing. Springer, 2003.