Evolutionary FSM-Based Agents for Playing Super Mario Game

Conference Paper · January 2013

DOI: 10.1007/978-3-642-44973-4_39

Rosa M Hidalgo-Bermúdez
University of Granada
2 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Maria Sandra Rodríguez-Domingo
University of Granada
2 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Antonio Mora
University of Granada
239 PUBLICATIONS **1,967** CITATIONS
SEE PROFILE

Pablo García-Sánchez
University of Granada
150 PUBLICATIONS **958** CITATIONS
SEE PROFILE

Evolutionary FSM-Based Agents for Playing Super Mario Game

R.M. Hidalgo-Bermúdez, M.S. Rodríguez-Domingo, A.M. Mora,
P. García-Sánchez, J.J. Merelo, Antonio J. Fernández-Leiva

Depto. Arquitectura y Tecnología de Computadores -
Depto. Lenguajes y Ciencias de la Computación
University of Granada - University of Málaga, Spain
rosa.hb84@gmail.com, zandra@correo.ugr.es
{amorag,pgarcia,jmerelo}@geneura.ugr.es, afdez@lcc.uma.es

Abstract. Most of game development along the years has been focused on the technical part (graphics and sound), leaving the artificial intelligence aside. However computational intelligence is becoming more significant, leading to much research on how to provide non-playing characters with adapted and unpredictable behaviour so as to afford users a better gaming experience. This work applies strategies based on Genetic Algorithms mixed with behavioural models, to obtain an agent (or bot) capable of completing autonomously different scenarios on a simulator of Super Mario Bros. game. Specifically, the agent follows the rules of the *Gameplay* track of Mario AI Championship. Different approaches have been analysed, combining Genetic Algorithms with Finite State Machines, yielding agents which can complete levels of different difficulties playing much better than an expert human player.

1 Introduction

Mario Bros. games series were created by Shigeru Miyamoto[1], and appeared in early 80s. The most famous so far is the platform game Super Mario Bros. and its sequels (for instance the blockbuster Super Mario World).

All of them follow a well-known plot: the plumber Mario must rescue the princess of Mushroom Kingdom, Peach, who has been kidnapped by the king of the koopas, Bowser. The main goal is to go across lateral platforming levels, trying to avoid different types of enemies and obstacles and using some useful (but limited) items, such as mushrooms or fire flowers.

Due to their success, amusement and attractiveness, Mario series have become a successful researching environment in the field of Computational Intelligence (CI) [5,6,1]. The most used framework is Mario AI, a modified version of the game known as Infinite Mario Bros.[2], an open-code application where the researchers can implement, for instance interactive and autonomous behavioural

[1] Designer and producer of Nintendo Ltd., and winner of the 2012 Príncipe de Asturias Prize in Humanities and Communication

[2] http://www.mojang.com/notch/mario/

routines, using the set of functions and variables it offers. Moreover, in order to motivate the scientific community to perform these studies, a competition is proposed three times a year, inside several famous conferences, it is called the *Mario AI Championship*[3], and is composed by some tracks: Learning, Level generation, Turing test, and Gameplay. The latter is devoted to create the best autonomous agent (also known as bot) as possible for automatically playing and pass sequential levels with a growing difficulty.

This work presents different approaches of autonomous agents aimed to this track, which consider a behavioural model created by means of a finite state machine (FSM) [2]. This model, based on expert knowledge, has been latter improved by applying offline (not during game) optimisation using Genetic Algorithms (GAs) [3], in two different schemes.

These approaches have been widely tested and analysed, getting an optimal set of parameters for the EA and thus, very competent agents in a number of difficulty levels.

2 Mario AI: Competition and Environment

The proposed agents follow the rules of the Mario AI Championship, considering the GamePlay track (complete as many levels as possible). The game consists in moving the character, Mario, through bi-dimensional levels. He can move left and right, down (crouch), run (letting the button pushed), jump and shoot fireballs (when in "fire" mode).

The main goal is complete the level, whereas secondary goals could be killing enemies and collecting coins or other items. These items may be hidden and may cause Mario to change his state (for instance a fire flower placed 'inside' a block). The difficulty of the game lies in the presence of cliffs/gaps and enemies. Mario loses power (i.e., its status goes down one level) when touched by an enemy and dies if he falls off a cliff.

The Mario AI simulator provides information about Mario's surrounding areas. According to the rules of the competition, two matrices give this information, both of them are 19x19 cells size, centred in Mario. One contains the positions of surrounding enemies, and the other provides information about the objects in the area (scenery objects and items).

Every tick (40 ms), Mario's next *action* must be indicated. This action consist in a combination of the five possible movements that Mario can do (left, right, down, fire/speed, jump). This information is encoded into a boolean array, containing a true value when a movement must be done.

The action to perform depends, of course, in the scenery characteristics around Mario, but it is also important to know where the enemies are and their type. Thus, the agent could know if it is best to jump, shoot or avoid them. We have defined four main enemies groups according to what the agent needs to do to neutralize them: one for enemies who die by a fireball/jump/Koopa shell, other for those who only die by a fireball, others which only die jumping on them, and finally others which just die by a Koopa shell.

[3] `http://www.marioai.org/`

3 Evolutionary FSM-based Agent

The proposed agent, *evoFSM-Mario*, is based in a FSM which models a logical behaviour, and which has been designed following an expert player knowledge. We decided to combine this technique with EAs, since they have proved being an excellent adapting and optimisation method, very useful for improving predefined behavioural rules, as the FSMs model.

We have defined a table of possible states for the agent, including all the possible (valid) actions the agent can perform in a specific instant, i.e. feasible combinations of moving left, moving right, crouch (going down), jump and fire/run. Table 1 shows the codification of the states in boolean values.

	St 0	*St 1*	*St 2*	*St 3*	*St 4*	*St 5*	*St 6*	*St 7*	*St 8*	*St 9*	*St 10*	*St 11*	*St 12*	*St 13*
Right	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Left	0	0	0	0	1	1	1	1	0	0	0	0	0	0
Fire/Run	0	0	1	1	0	0	1	1	0	0	0	0	1	1
Jump	0	1	0	1	0	1	0	1	0	0	1	1	0	1
Down	0	0	0	0	0	0	0	0	0	1	0	1	0	0

Table 1. Codification of the feasible states of the FSM which will model the Mario agent's AI. 1 is $true/active$, 0 is $false/non-active$.

Depending on the input string, the state changes (or remains), so the transition is decided. Inputs are the possible situations of the agent in the environment: for example, find an enemy or being near a cliff/gap. Each input is represented as a boolean string, having a *true (1)* value in a position if a specific situation or event has happened.

These possible states along with the possible inputs and transitions will be evolved by means of the GA, considering that the output state for a new entry is randomly set, but according to a probability which models the preference of the states, in order to improve the convergence of the algorithm. Due to the huge search space a parameter indicating the percentage of new individuals that will be added in each generation is included, in order to control the diversity rate.

Every individual in the population is represented by a set of tables, one per state, and every table contains an output state for every possible input. The *fitness function* is calculated for each individual by setting the FSM represented in a chromosome as the AI of one agent. It is then placed in a level, and then, it plays for obtaining a fitness value. Two different schemes have been implemented: *mono-seed*, where all the individuals are tested in the same level (with the same difficulty), which grows in length with the generations; and a *multi-seed* approach, where every individual is tested in 30 levels (in the same difficulty) generated randomly (using different seeds). In both cases every agent plays until it pass the level, dies or gets stacked.

The aim of the latter scheme is: first, avoid the usual noise [4] present in this type of problems (videogames), i.e. it tries to get a fair valuation for an individual, since the same configuration could represent an agent which is very good sometimes and quite bad some others, due to the stochasticity present in every play (just in the agent's behaviour); and second, get individuals prepared to a wide set of situations in every level and difficulty, since 30 levels should present a high amount of different scenarios and configurations.

Thus, there is a *generic fitness* which has as restriction completely finish the level to be set to positive. On the contrary, individuals that have not finished the level start from the lowest fitness possible and their negativity is reduced according the behaviour during the level run. This generic fitness is a weighted aggregation based in the values: *marioWinner* (1 if the agent finish the level), *marioSize* (0 small, 1 big, and 2 fire), *numKilledEnemies*, *numTotalEnemies*, *numCellsPassed*, *remainingTime*, *timeSpent*, *coinsGathered*, *totalCoins*, *numCollisions* (number of times the agent has bumped with an enemy), *numGatheredPowerUps*, *causeOfDeath* (value representing how the agent has died).

This fitness is considered as the result of the evaluation for the individuals in the mono-seed approach, meanwhile multi-seed considers a *hierarchical fitness*, where the population is ordered according the next criteria: First, taking into account the percentage of levels where the individuals have been stacked or fallen from a cliff. Then, they are ordered considering the average percentage of levels completed. Finally, the individuals are ordered by the average generic fitness.

The *selection mechanism* considers the best individual and a percentage of the best ones, selected by tournament according to their fitness. The percentage of individuals to consider as parents follows to schemes: in mono-seed it is low at the beginning and will be increased when the number of generations grows; in multi-seed it is constant.

Uniform *Crossover* is performed considering the best individual of the present generation as one of the parents, and one of the individuals with positive fitness as the other parent. They generate a number of descendents which depends on the percentage of population to complete with the crossover.

The *Mutation operator* selects a percentage of individuals to be mutated, and a random set of genes to be changed in every individual, then the output state is randomly changed for an input in the table.

There is a $1-$ *elitism replacement* to form the new population (the best individual survives). The rest of the population is composed by the offspring generated in the previous generation (a percentage of the global population) and a set of random individuals, in order to increase the diversity.

4 Experiments and Results

In order to test the approaches, several experiments have been conducted. The aim was to find good enough agents for completing any level in any difficulty. Previously it was performed a hard fine-tuning stage (through systematic experimentation), where several different values for the parameters were tested, searching for the best configuration. The best values are presented in Table 2.

The values in the table (mono-seed) show that a small number of generations is required to get a competent agent in difficulty level 0, along with a population size quite high since every individual is just test once in this approach, so it is needed to ensure the evolution, even considering that none of the individuals may

	mono-seed	*multi-seed*
Population size	1000 (difficulty 0)	2000 (difficulty 4)
Number of generations	30 (difficulty 0)	500 (difficulty 4)
Crossover percentage	95%	95%
Mutation percentage	2% (individuals)	2% (individuals)
Mutation rate	1% (genes)	1% (genes)
Percentage of random individuals	5% (decreased with the generations)	5% (constant)
Fitness function	generic (aggregation)	hierarchical

Table 2. Optimal parameter values for mono- and multi-seed approaches.

not complete a level. In that cases, some other generations are run to improve them and give another step in the evolution process.

It is important to remark that a single play of an agent could spend around 40-50 seconds on average (depending on the level length and difficulty), because it must be played in real-time, not simulated. So a single evaluation for one individual in the multi-seed approach could take around 25 minutes.

Moreover, when the experiments were conducted, it could be noticed than the most difficult levels strongly limited the algorithmic behaviour of the approaches, making it hard to converge and even ending the execution abruptly. In addition to the high computational cost (one run may take several days), there was a problem with the structure that stores the FSM of the individuals, since it is huge (in memory terms) and grows exponentially during the run (new inputs and outputs are added to the tables in crossovers), so in levels higher than 4, the program frequently crashes. Thus, this structure implementation was redesigned to an optimal one, letting to evolve agents in all the possible levels, in the mono-seed approach, but with a shorter number of generations than recommended.

In multi-seed case the memory problem still remained, so the number of states where reduced to 12, by deleting those considered as non-useful (in Table 1): State 8 (no action is done) and State 11 (Mario jumps and crouch, since these actions are not possible simultaneously in this implementation of Infinite Mario). With this change, it was possible optimising competent agents from difficulty levels 0 to 4, which are, in turn, enough for the GamePlay competition.

The fitness evolution was studied, showing a grow (improvement) with the generations, as expected in a GA. Moreover there are always enough individuals with positive fitness to assure the offspring generation (at least in the easiest levels). Some examples of the obtained evolved FSM-based agents can be seen in action (in a video) from the next urls:

– *Difficulty level 1 (completed)*: `http://www.youtube.com/watch?v=6Pj6dZCE070`
– *Difficulty level 2 (completed)*: `http://www.youtube.com/watch?v=gtfuY-L0WDA`
– *Difficulty level 3 (completed)*: `http://www.youtube.com/watch?v=qQVQ43sWwYY`
– *Difficulty level 12 (stacked)*: `http://www.youtube.com/watch?v=zNGfBApX7sk`

The last one was evolved for some generations (not all the desired) in that level of difficulty, due to the commented problems, so it cannot complete this hard level in the simulator. However, as it can be seen, it is quite good in the first part of the play. Thus, if we could finish the complete evolution process in this difficulty level we think the agent could complete any possible level.

5 Conclusions

In this work, two different approaches for evolving, by means of Genetic Algorithms (GAs), agents which play Super Mario Bros. game have been proposed and analysed. They have been implemented using Finite State Machine (FSM) models, and considering different schemes: mono-seed and a multi-seed evaluation approaches, along with two different fitness functions. Both algorithms have been tested inside a simulator named Mario AI, implemented for the Mario AI Competition, focusing on the GamePlay Track.

Several experiments have been conducted to test the algorithms and a deep analysis has been performed in each case, in order to set the best configuration parameters for the GA. Some problems have arisen such as the high memory requirements, which have done it hard to complete the optimisation process in several cases. However, very competent agents have been obtained for the difficulty levels 0 to 4 in both approaches, which are, in turn, enough for the Game Play competition requirements.

In the comparison between the approaches, mono-seed can yield excellent agents for the level where they were 'trained' (evolved), having a quite bad behaviour in a different level. Multi-seed takes much more computational time and has higher resource requirements, but the agents it yields are very good playing in any level of the considered difficulty (in the evolution). All these agents play much better than an expert human player and can complete the levels in a time impossible to get for the human.

Acknowledgements

This work has been supported in part by the P08-TIC-03903 and P10-TIC-6083 projects awarded by the Andalusian Regional Government, the FPU Grant 2009-2942 and the TIN2011-28627-C04-01 and TIN2011-28627-C04-02 projects, awarded by the Spanish Ministry of Science and Innovation.

References

1. Bojarski, S., Bates-Congdon, C., *REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario.* In: Proceedings of the IEEE CIG 2011, IEEE Press, pp. 83–90, 2011.
2. Booth, T.L., *Sequential Machines and Automata Theory*, John Wiley and Sons, Inc., New York, 1st edition, 1967.
3. Goldberg D.E., Korb B., Deb K., *Messy genetic algorithms: motivation, analysis, and first results*, Complex Systems, 3(5), pp. 493–530, 1989.
4. Mora, A.M., Fernández-Ares, A., Merelo, J.J., García-Sánchez, P., *Dealing with noisy fitness in the design of a RTS game bot.* In Proc. Applications of Evolutionary Computing: EvoApplications 2012, LNCS, vol. 7248, Springer, pp. 234–244, 2012.
5. Pedersen, C., Togelius, J., Yannakakis, G., *Modeling Player Experience in Super Mario Bros.* In Proc. 2009 IEEE Symposium on Computational Intelligence and Games (CIG'09), IEEE Press, pp 132–139, 2009.
6. Togelius, J., Karakovskiy, S., Koutnik, J., Schmidhuber, J., *Super Mario evolution.* In Proc. 2009 IEEE Symposium on Computational Intelligence and Games (CIG'09), IEEE Press, pp 156–161, 2009.