Learning Levels of Mario AI using Genetic Algorithms

Alejandro Baldominos*, Yago Saez, Gustavo Recio, and Javier Calle

Universidad Carlos III de Madrid
Avenida de la Universidad, 30. 28911 Leganes. Spain
{abaldomi,ysaez,grecio,fcalle}@inf.uc3m.es

Abstract. This paper introduces an approach based on Genetic Algorithms to learn levels from the Mario AI simulator, based on the Infinite Mario Bros. game (which is, at the same time, based on the Super Mario World game from Nintendo). In this approach, an autonomous agent playing Mario is able to learn a sequence of actions in order to maximize the score, not looking at the current state of the game at each time.
Different parameters for the Genetic Algorithm are explored, and two different stages are executed: in the first, domain independent genetic operators are used; while in the second knowledge about the domain is incorporated to these operators in order to improve the results.
Results are encouraging, as Mario is able to complete very difficult levels full of enemies, resembling the behavior of an expert human player.

Keywords: Mario AI, Games, Genetic Algorithms, Learning

1 Introduction

Super Mario Bros. is a sidescroller platform videogame designed by Shigeru Miyamoto and released for the Nintendo Entertainment System three decades ago, in 1985. This game has become a great success, achieving over 40 million sales and making the fifth position in the list of best-selling videogames, the other four being released after 2005. Today, the Mario franchise has reaped significant success and Mario videogames and merchandising generate millions of dollars.

In 1990, another Mario game was released: Super Mario World. This game implied a technical improvement in graphics, audio and gameplay over the original sidescroller, and introduced new characters like Yoshi. In 2009, the Mario AI Championship was introduced [10], aiming at developing intelligent agents able to complete levels of increasing difficulty of a game based on Infinite Mario Bros., a game based at the same time on Super Mario World (but with pseudo-randomly generated levels). In 2010, the Mario AI Championship introduced a new track: the Learning track, where an agent was intended to learn the best strategy to obtain the maximum score in a fixed level of the game, being able to play a maximum of 10,000 games of that same level before the competition in order to learn it.

* Corresponding author.

While the competition is no longer organized (it was discontinued in 2013 in favour of the Platformer AI Competition), this paper aims at building an intelligent agent able to compete following the rules of the Mario AI Learning track. Genetic Algorithms will be used in order to learn the best strategy (i.e., sequence of actions performed by Mario) to maximize the score. This research work is an extension of a B.Sc. thesis published by Hector Valero [12].

This paper is structured as follows: section 2 describes related work. Later, section 3 describes the proposal, providing further details about how individuals are encoded and evaluated and how genetic operators are used. Experiments are conducted to validate and evaluate the proposal, and their setup and results are discussed in section 4. Finally, section 5 provides some conclusive remarks and proposes future lines of work.

2 State of the Art

Some work related to this paper can be found in the papers published by the organizers of the Mario AI Championship. For instance, the paper published by Togelius et al. summarizing the main results from the 2009 edition in the GamePlay track [10] describes the winner solution involving the use of the A* graph search algorithm, and briefly introduces other solutions using rule-based controllers, reactive controllers or finite state machines. Even when this paper referred to the Gameplay track, some solutions used learning algorithms such as genetic programming, stack-based virtual machines, and imitation or reinforcement learning; in some cases controllers are evolved using genetic algorithms. However, these approaches are not discussed in the paper, but rather mentioned.

Another work by Togelius et al. [11] discusses approaches using neural networks for learning controllers for the Super Mario game, involving multilayer perceptrons, simple recurrent networks and HyperGP for evolving the weights. Finally, a work by Karakovskiy and Togelius was published in 2012 [6] discussing the conclusions regarding the competition organization and summarizing the AI techniques used by contestants in the different tracks. Another approach using Q-learning imposing biological constraints for imitating the behavior of human players in the Infinite Mario Bros. is proposed by Fujii et al. [4, 5].

Besides Super Mario, other authors have used AI techniques in order to learn controllers for videogame characters, imitating the behavior of a human player. It is specially outstanding a work published by Google DeepMind in Nature [8] describing the development of an agent to play several games for Atari 2600, using so-called deep Q-networks (neuron-based networks for reinforcement learning), where the inputs are the pixels in the screen. A framework for evaluating other agents in this same domain is provided by Bellemare et al. [2].

3 Proposal

This paper proposes the development of an agent able to maximize the score obtained in one specific Mario AI level. This agent is designed so that it could

compete in the Learning Track of the Mario AI Championship, even when the last edition of this competition happened in 2012. The Learning track allows the agent to learn the level over N games, evaluating the agent in the game $N + 1$. The score is computed considering different aspects of the game, including the number of collected coins, killed enemies, remaining time after completing the level, etc. Details about how these aspects are weighted to compute the score are provided later, when the fitness function is described.

There are two constraints which must be considered when training the agent: *a*) the agent is limited to 10,000 games ($N = 10000$) in order to be able to learn the level; and *b*) the response time to decide the next action to be performed by the agent must not exceed 42 ms.

In order to generate the agent who will optimize the obtained score during a Mario AI game, genetic algorithms will be used. In this approach, the learning algorithm will not consider the current state of the game (i.e., how Mario is placed in the environment in a certain point in time), but rather will compute a predefined sequence of actions (the same actions that could be performed by a human player) and evaluate it over the game level. This sequence will be evolved in order to maximize the final score obtained when the level is finished, either because it is successfully completed or because the character dies.

3.1 Encoding

As described above, an agent is defined as a sequence of actions to be performed in a specific level of Mario AI. The chromosome must be able to represent a sequence of all possible actions performed by the agent in the game. In order to control the character, the player can use a D-Pad with four positions (up, left, down, right) and two additional buttons, namely A (jump) and B (run and shoot). The system allows several buttons to be pressed at once, resulting in a space of $2^6 = 64$ possible actions. However, this set of actions can be significantly reduced by introducing some domain knowledge: *a*) the button *up* performs no action in the game; and *b*) some combinations are not feasible, such as pressing *left* and *right* at the same time. With these considerations in mind, the number of actions can be reduced to 22, as pointed out in table 1. For the genetic algorithm, we have encoded each gene as an integer in the range 0 to 21.

Once the definition of genes are formally described, the chromosome length must be determined. In this domain, there is not a fixed length for the sequence of actions. However, we can estimate a maximum length knowing that *a*) the maximum time for completing a level are 200 seconds and *b*) each second can be discretized in 15 ticks. As a result, we define sequences of actions of length 3,000, which implies chromosomes of 3,000 genes, even if not all the actions can be performed (i.e., if Mario completes the game or dies before performing 3,000 actions). This implies that there will be 22^{3000} combinations, so the search space is noticeably big. For this reason, in the first stage a reduced set of actions will be used where we assume that Mario is running everytime, i.e., we only consider actions where the button B is pressed, reducing the search space from 22 actions to 11 at the expense of imposing limits on the representation.

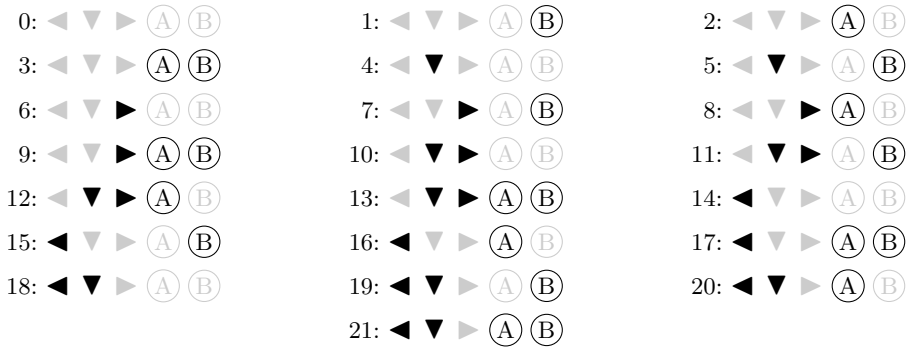

Table 1: List of actions along with the pressed buttons for each one.

3.2 Fitness

The fitness function is defined to be the score function, and the genetic algorithm will look towards maximizing this value. The score function used is defined by the Mario AI Championship rules, and follows equation 1, where D is the physical distance traveled by Mario from the start to his final position; d_f, d_m and d_{gm} are the number of devoured flowers, mushrooms and green mushrooms respectively; k is the number of killed enemies; k_{st}, k_{sh} and k_f are the number of enemies killed by stomp (jumping), by throwing shells or by throwing fireballs respectively; s is the final status of the game, either won (1) or lost (0); m is the final status of Mario, either small (0), big (1) or fire (2); b_h is the total number of hidden blocks found; c is the total number of coins collected and t' is the time left.

$$\begin{aligned} S = D + 64d_f + 58d_m + 58d_{gm} + 42k + 12k_{st} + 17k_{sh} \\ + 4k_f + 1024s + 32m + 24b_h + 16c + 8t' \end{aligned} \tag{1}$$

3.3 Genetic Operators

The Genetic Algorithm used performs tournament selection, crossover and mutation. When generating the new population, the offspring will replace their parents. Two different versions for each of these operators have been implemented, the first one being domain-independent and the second one introducing domain knowledge to optimize the behavior of the operator.

Initialization In the first version, a naive initialization is used, where each action in the chromosome is randomly chosen. However, it is interesting to try a guided initialization, as the most frequent actions for completing the level are running to the right (*right* + B) or running to the right while jumping (*right* + A + B). For this reason, in the second version we introduce a hybrid initialization approach, where either one of the previous initialization methods (random or guided) is randomly for each action.

Selection Tournament selection is performed, where T_s individuals are randomly selected from the population and face each other, and the one with highest fitness will be one of the parents used for generating the next population. The second version incorporates elitism.

Crossover The first version uses single-point crossover, where a random point n is chosen, so that n is a number strictly smaller than the length of the chromosome($n < l = 3000$). If the first parent is $p^1 = \langle b_1^1, b_2^1, \ldots, b_l^1 \rangle$ and the second is $p^2 = \langle b_1^2, b_2^2, \ldots, b_l^2 \rangle$, then the following two childs are begotten: $c^1 = \langle b_1^1, \ldots, b_n^1, b_{n+1}^2 \ldots, b_l^2 \rangle$ and $c^2 = \langle b_1^2, \ldots, b_n^2, b_{n+1}^1 \ldots, b_l^1 \rangle$.

The second version incorporates domain knowledge into the crossover operator. In particular, the crossover is guaranteed to be performed in a point where the absolute position of Mario in the game is similar, i.e. a value of n is pursued so that the position of Mario in both games is close (the euclidean distance between the pairs $[x_1, y_1]$ and $[x_2, y_2]$ falls below a threshold Δ). This ensures continuity in the game, for instance, if one parent had a good start but fails to keep playing well after a certain point $n^+ > n$ and the other parent starts playing bad but improves after a point $n^- < n$, then finding point n will generate offspring in which one child would be better than both parents.

Mutation In the first version, mutation is performed randomly in M genes in the chromosome. In the second version, mutation is performed over the last w_t^i actions performed by the individual i in the last evaluation before the game ended. w_t^i is a value intrinsic to the individual and which may vary from one generation to another, and is computed as follows:

$$w_t^i = \begin{cases} 2 \times w_{t-1}^i & \text{if } S_t^i \leq S_{t-W}^i \\ w_0 & \text{if } S_t^i > S_{t-1}^i \end{cases}$$

where S_t^i is the fitness for individual i at iteration t and W is defined as a mutation window, which is variable over time.

The fact that the mutated actions are those before the game ended will mostly change the behavior of the character before he dies, at least in the first generations where it is likely to encounter bad individuals which will rarely complete the game. The size of the mutation window (the number of mutated actions) will double every W generations until the individual improves its fitness (S^i), and at this time its size will be reset to the default value w_0.

4 Evaluation

This section describes the parametrization and results for the two stages, the first using domain-independent genetic operators and a reduced set of 11 actions; and the second incorporating specific domain knowledge and all 22 actions.

4.1 1st Stage Experimental Setup

In order to execute the Genetic Algorithm, JGAP library [7] has been used. Regarding the parametrization of the experiments, the tested values are described below. In some cases, different values have been assigned to the same parameter:

Selection The tournament size (T_s) is defined as a fraction over the population size (P). In the experiments, it is defined as $T_s = 15\%$ and forced to be $T_s \geq 3$.

Crossover Different values are tried for the crossover rate: $C = 0.1\%$, $C = 0.2\%$, $C = 0.3\%$, $C = 0.4\%$ and $C = 0.5\%$.

Mutation Different values are tried for the mutation rate: $M = 5\%$, $M = 3.3\%$, $M = 2.5\%$, $M = 2\%$, $M = 1.3\%$ and $M = 1\%$.

Population Two values have been tested for the population size: $P = 20$ and $P = 50$.

Generations The maximum number of evaluations is fixed by the Mario AI Championship rules to be $E_M = N = 10000$. The number of generations (G) must be defined as $G = {}^{E_M}/_{P}$.

Granularity This parameter tries to resemble human behavior, as it is often the case that the time that happens since players press a button until they release it exceeds one tick. Granularity indicates how many ticks involve each action. Three different values are tested: $g = 1$, $g = 2$ and $g = 5$.

Besides the previous parameters, Mario AI accepts additional arguments in order to generate a level. The next arguments have been used:

- Visualization of the game is disabled (`-vis off`) as otherwise fitness evaluation time would increase significantly.
- Hidden blocks are enabled (`-lhb on`), so they can appear in the level.
- Enemies are enabled (`-lt on`), so they can appear in the level.
- Ladders are disabled (`-lla off`), so they cannot appear in the level.
- Dead ends are enabled (`-lde on`), so they can appear in the level.
- The level type is set to overground (`-lt 0`), other options being underground (1), castle (2) or random (3).
- The level difficulty is set to 1 (`-ld 1`) in a scale from 1 to 12.
- The level length is defined as 300 (`-ll 300`), in a scale from 50 to $2^{31} - 1$. While the maximum value is quite high, we have selected an average length based on the maps of the real game.
- The level PRNG seed is set to 20002.

In the first stage, where only domain-independent definitions of the generic operators are used, a total of 180 experiments have been executed, this number resulting from all the possible combinations for the previous parametrization.

Average Fitness		Maximum Fitness		Completed Games	
P = 20	**P = 50**	**P = 20**	**P = 50**	**P = 20**	**P = 50**
5,513.36	5,622.30	8,773.07	8,551.13	61,921	89,842

Table 2: Fitness (average and max.) and completed games for each value of P

	Average Fitness		Completed Games	
	P = 20	**P = 50**	**P = 20**	**P = 50**
g = 1	4,917.90	4,770.22	4,796	0
g = 2	5,391.58	5,438.07	20,101	23,678
g = 5	6,230.61	6,658.62	38,024	66,164

Table 3: Average fitness and completed games for each value of P vs. g

4.2 Sensitivity Analysis of the Parameters

The high combination of parameters makes it impossible to describe all the results, for that reason, this section provides the main conclusions on how each parameter affects the score. Results are computed as the average of 10 different executions.

Population size (P) Table 2 shows the average fitness, the maximum fitness and the number of completed games for each value of the population size. It can be seen that there are no significant differences in the score (neither average nor maximum), but still a higher population size leads to a higher number of completed games.

Granularity (g) The impact of the granularity in the average fitness and completed games is shown in table 3. It can be clearly seen that the value $g = 5$ provides better results for both metrics.

Mutation rate (M) As it can be seen in table 4a, both the average fitness and the completed games increases when the mutation rate is decreased. This may

	Avg. F.		Compl. G.	
	P = 20	**P = 50**	**P = 20**	**P = 50**
5%	4,468.19	4,577.51	0	0
3.3%	4,906.72	5,024.67	73	1,277
2.5%	4,866.14	5,519.18	0	3,887
2%	5,596.04	5,567.10	6,631	4,350
1.3%	6,213.64	6,120.62	11,081	30,569
1%	7,029.44	6,924.71	44,136	49,758

(a) P vs. M

	Avg. F.		Compl. G.	
	P = 20	**P = 50**	**P = 20**	**P = 50**
0.1	5,718.19	5,571.73	17,059	13,341
0.2	5,511.82	5,669.82	13,594	18,903
0.3	5,565.07	5,684.42	17,184	19,991
0.4	5,317.70	5,586.35	8,791	20,090
0.5	5,454.03	5,599.19	5,293	17,517

(b) P vs. C

Table 4: Average fitness and completed games for each value of P vs. M and C

be due to the fact that high mutation rates are promoting exploration, but not the exploitation of good solutions.

Crossover rate (C) As shown in table 4b, it is difficult to extract conclusions regarding the impact of the crossover rate: there are not significant differences in the average fitness, and while there are differences in the number of completed games, these changes do not adhere to any clear pattern.

4.3 2nd Stage Experimental Setup

After obtaining the results described in the previous section, the second phase starts. This phase incorporates domain-specific knowledge into the genetic operators and uses the full set of actions; and also the total number of experiments is reduced by removing some parameters assignations which have not performed well. In particular, the next parameters are affected:

Mutation The only value left for the mutation rate is $M = 1\%$. The initial mutation window is set to $w_0 = 2$, and the values tested for W are $W = 2$, $W = 3$ and $W = 5$.

Granularity The value $g = 1$ is removed because it was outperformed by the others, thus leading to values $g = 2$ and $g = 5$.

Moreover, new Mario AI combinations have been tested, all of them having visualization disabled (`-vis off`), hidden blocks enabled (`-lhb on`), dead ends enabled (`-lde on`), and level length of 300 (`-ll 300`):

- **Scenario 1**: level difficulty 4 (`-ld 4`), ladders disabled (`-lda off`), enemies enabled (`-le on`) and seed 01121987 (`-ls 01121987`).
- **Scenario 2**: level difficulty 4 (`-ld 4`), ladders enabled (`-lda on`), enemies enabled (`-le on`) and seed 201183 (`-ls 201183`).
- **Scenario 3**: level difficulty 4 (`-ld 4`), ladders enabled (`-lda on`), enemies disabled (`-le off`) and seed 334 (`-ls 334`).
- **Scenario 4**: level difficulty 4 (`-ld 4`), ladders enabled (`-lda on`), enemies enabled (`-le on`) and seed 333 (`-ls 333`).
- **Scenario 5**: level difficulty 4 (`-ld 4`), ladders disabled (`-lda off`), enemies enabled (`-le on`) and seed 11062011 (`-ls 11062011`).
- **Scenario 6**: level difficulty 4 (`-ld 4`), ladders enabled (`-lda on`), enemies enabled (`-le on`) and seed 444 (`-ls 444`).

4.4 2nd Stage Results

Again, the number of combinations is too big to describe the results thoroughly. Still, this section shows the evolution of the average fitness along each generation, which is displayed in figure 1. It can be noticed that the best configuration always involves the highest granularity ($g = 5$) and the lowest mutation rate ($M = 0.1\%$) with $W = 2$. However, the best crossover rate varies across scenarios. Results are computed as the average of 10 executions.

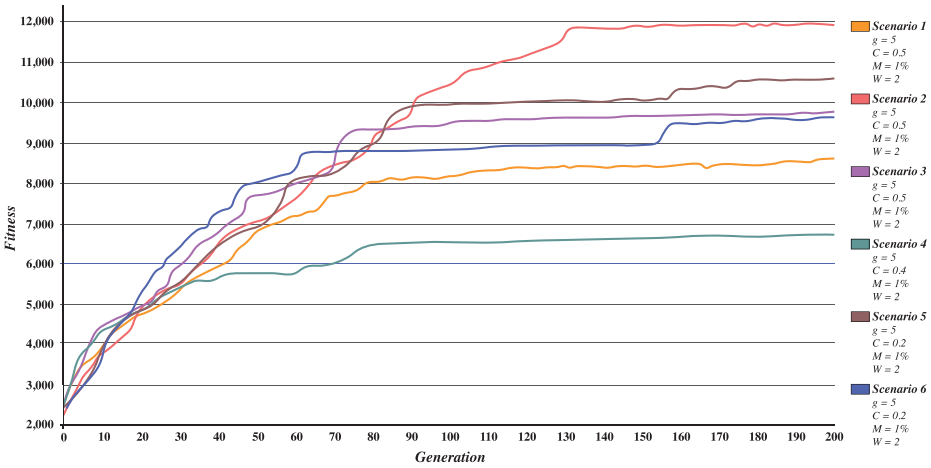

Fig. 1: Fitness evolution for the best configuration of each scenario

4.5 Discussion

Results clearly show how agents evolve by learning the best strategy to complete the game and maximize the score they obtain. In the 2010 Mario AI Championship celebrated during CIG in Copenhagen [1] the winner obtained 45,017 points for five different games, i.e., an average score of 9,003.4 points per level. If we average the fitness of our best agent for each scenario, we obtain an average score of 12,059, outperforming the winner of that year.

5 Conclusions and Future Work

This paper has proposed the development of an agent able to compete in the Learning track of the Mario AI Championship. This agent learns a sequence of actions by using a genetic algorithm with integer encoding, in order to maximize the attained score after ending the level, not considering the state of the game at a certain point in time.

The approach has been evaluated using the Mario AI framework in two different stages: in the first one, the set of actions has been simplified and domain-independent genetic operators have been used; while in the second the full set of actions has been used and operators have been enriched by incorporating domain-specific information.

Most agents are able to learn how to complete the game, obtaining an average of 12,059 points. A video showing the best agent in action can be seen in YouTube [3] and it has shown significant impact, as it has been cited in Wired [9].

However, there is still room for improvements. For instance, we have noticed that while the button *down* is pressed, *right* and *left* perform no action; so the space of actions can be reduced even more, resulting in 16 combinations which

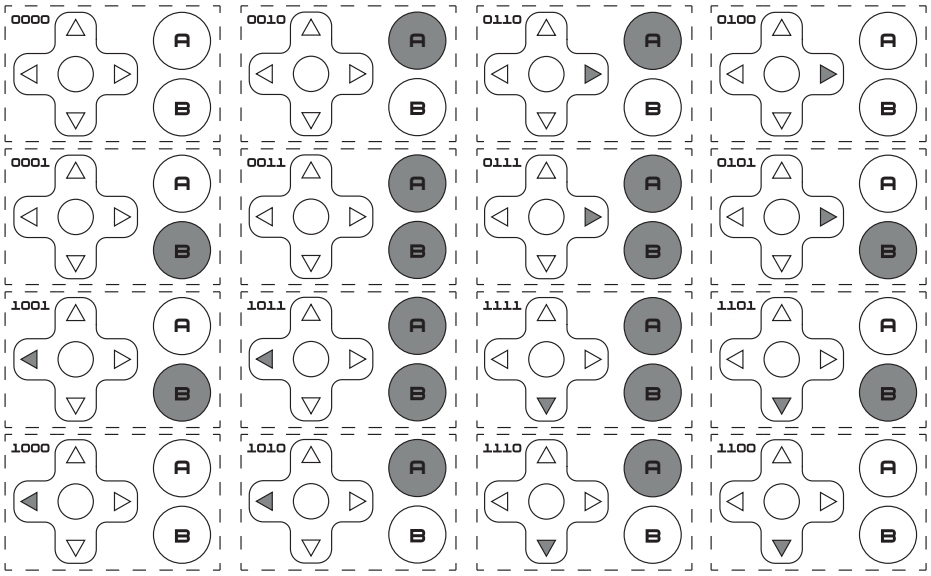

Fig. 2: Proposed new set of feasible actions, along with their encoding (4 bits).

are shown in figure 2 along with their encoding using 4 bits. The chosen encoding resembles Gray code as small changes in the genotype are translated into small changes in the phenotype, and it consists of the sequence of bits $\langle b_{ld}, b_{dr}, b_a, b_b \rangle$; where b_a and b_b respectively determine whether the A and B buttons are pressed, while b_{ld} and b_{dr} indicate whether *left*, *right* or *down* buttons are pressed using the following convention: if only one of b_{ld} or b_{dr} is 1, then the pressed button is *left* or *right* respectively, while if both b_{ld} and b_{dr} are one, then the pressed button is *down*. This encoding has been implemented and evaluating its quality is left for future work.

Finally, additional experimental setups can be tried in order to further improve Mario's performance. However, experiments with bigger populations or higher granularities are expensive to be evaluated, and the results obtained in this paper are satisfactory, so this task is left for future work.

Acknowledgements

This research work is co-funded by the Spanish Ministry of Industry, Tourism and Commerce under grant agreement no. TSI-090302-2011-11. Special acknowledgements are addressed at Hector Valero due to his contributions to the work.

References

1. Mario AI Championship 2010: Results. `https://sites.google.com/a/marioai.com/www/results` (2010), [Online; accessed May 24, 2015]

2. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The Arcade Learning Environment: An Evaluation Platform for General Agents. JAIR 47, 253–279 (2013)
3. Emgallar: Intelligent NPC for Mario AI Championship. https://www.youtube.com/watch?v=u_OpgFQ8HcM (2011), [Online; accessed May 22, 2015]
4. Fujii, N., Sato, Y., Wakama, H., Katayose, H.: Autonomously Acquiring a Video Game Agents Behavior: Letting Players Feel Like Playing with a Human Player. LNCS 7624, 490–493 (2012)
5. Fujii, N., Sato, Y., Wakama, H., Kazai, K., Katayose, H.: Evaluating Human-like Behaviors of Video-Game Agents Autonomously Acquired with Biological Constraints. LNCS 8253, 61–76 (2013)
6. Karakovskiy, S., Togelius, J.: The Mario AI Benchmark and Competitions. IEEE Trans Comput Intell AI Games 4(1), 55–67 (2012)
7. Meffert, K., Rotstan, N.: JGAP: Java Genetic Algorithms Package. http://jgap.sourceforge.com (2015), [Online; accessed July 6, 2015]
8. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-Level Control through Deep Reinforcement Learning. Nature 518, 529–533 (2015)
9. Steadman, I.: This AI 'Solves' Super Mario Bros. and Other Classic NES Games. http://www.wired.co.uk/news/archive/2013-04/12/super-mario-solved (2013), [Online; accessed May 22, 2015]
10. Togelius, J., Karakovskiy, S., Baumgarten, R.: The 2009 Mario AI Competition. In: 2010 IEEE Congr on Evol Comput. pp. 1–8 (2010)
11. Togelius, J., Karakovskiy, S., Koutnik, J., Schmidhuber, J.: Super Mario Evolution. In: 2009 IEEE Symp Comput Intell and Games. pp. 156–161 (2009)
12. Valero, H., Saez, Y., Recio, G.: Computacin Evolutiva Aplicada al Desarrollo de Videojuegos: Mario AI. http://e-archivo.uc3m.es/handle/10016/13308 (2011)