A Neuroevolution Approach to Imitating Human-Like Play in Ms. Pac-Man Video Game

Maximiliano Miranda, Antonio A. Sánchez-Ruiz, and Federico Peinado

Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
c/ Profesor José García Santesmases 9, 28040 Madrid (Spain)
m.miranda@ucm.es - antsanch@ucm.es - email@federicopeinado.com
http://www.narratech.com

Abstract. Simulating human behaviour when playing computer games has been recently proposed as a challenge for researchers on Artificial Intelligence. As part of our exploration of approaches that perform well both in terms of the instrumental similarity measure and the phenomenological evaluation by human spectators, we are developing virtual players using Neuroevolution. This is a form of Machine Learning that employs Evolutionary Algorithms to train Artificial Neural Networks by considering the weights of these networks as chromosomes for the application of genetic algorithms. Results strongly depend on the fitness function that is used, which tries to characterise the human-likeness of a machine-controlled player. Designing this function is complex and it must be implemented for specific games. In this work we use the classic game Ms. Pac-Man as an scenario for comparing two different methodologies. The first one uses raw data extracted directly from human traces, i.e. the set of movements executed by a real player and their corresponding time stamps. The second methodology adds more elaborated game-level parameters as the final score, the average distance to the closest ghost, and the number of changes in the player's route. We assess the importance of these features for imitating human-like play, aiming to obtain findings that would be useful for many other games.

Keywords: Virtual Video Game Player, Human Behaviour Imitation, Artificial Neural Networks, Evolutionary Computing, Genetic Algorithms, Machine Learning, Artificial Intelligence

1 Introduction

Researchers on Artificial Intelligence (AI) are always looking for problems that are challenging but feasible at the same time, in order to progress their mission of computationally recreating intelligence. Imitating video game players has been recently considered an interesting challenge for the AI research community. Examples of this trend are the competitions regarding the development of believable video game characters that have taken place during the last years [5, 12].

In the Digital Game Industry there is a widespread assumption that wherever there is a machine-controlled character, the game experience will benefit from this character to be controlled by the computer in a less robotic and more human-like way. For this reason, player modelling in video games has been a increasingly important field of study, not only for academics but for professional developers as well [13].

Human-like bots, as they are also called, can not only be used to confront the human player, but also to collaborate with him or to demonstrate how to succeed in a particular game level to help players who get stuck. It seems clear that these demonstrations will be more useful if the bot imitates a human style of playing. In fact, this last application has shown a great potential: there has been a very important growth in the consumption of gaming videos during the last years and 95% of people who plays video games watch also gaming videos online [3]. There is a great interest in watching how other people play, either to learn how a particular game works (how to improve some skills, how to solve a particular puzzle, etc.), just for mere entertainment, or as business and mass spectacle (e.g. the so-called electronic sports).

Another possible application of human-like bots can be focused on the testing phase in video game production process. These bots could be used to check that the game levels have the right difficulty, that none of them are "broken" by design, or even to find completely new ways for solving a puzzle. In case of procedural generation of levels, a human-like bot could help us to select which ones are the best to be played for real people.

The goal of this work is to develop a human-like virtual player that, as it will be explained later, has been developed using the Neuroevolution approach, i.e. evolving an artificial neural network with the purpose of knowing what moves to chose for automatically playing an action video game. This approach has proved successful for training bots of classic games as Super Mario Bros [8]. In our case, we will use another classic arcade game, Ms. Pac-Man, using a Java version specifically designed for the development of bots that we have tested in previous experiments [7].

The rest of the paper is organised as follows. Next section reviews other works related to the simulation of human players. In Section 3 the features of our testbed and problem domain, Ms. Pac-Man vs Ghosts, are explained in detail. Section 4 outlines our modelling process of the game state. Section 5 explains how do we obtain the training set and what features do we select for the machine learning process. Our neuroevolutive algorithm is presented in the Section 6 and the results of our experiments appear and are discussed in Section 7. Finally Section 8 summarises our conclusions about this first attempt of using the approach for creating player bots for Ms. Pac-Man.

2 Related Work

As a sub-field of Computer Science especially related to AI, Machine Learning aims to gives computers the ability to learn by themselves, not following the

instructions of a static program. It includes the study and construction of algorithms that can make predictions on data, building a computational model that relates inputs with predictions or decisions in the form of outputs.

Neuroevolution (NE) is a form of machine learning that employs an Evolutionary Algorithm (EA) to train an Artificial Neural Network (ANN) [9] by considering the weights of the ANN as the chromosomes of the EA.

Regarding the imitation of human behaviour in video games, there is plenty of work that could be categorised into direct and indirect imitation [11].

In direct imitation developers use supervised learning in order to associate the state of the controller with the actions the human player takes given the game state. This imitation approach seems to be the most popular and its main problem is the generalization of previously unseen situations which leads to controllers that perform worse than the player whose traces were used in the supervised learning. The main cause is this training aiming to repeat the same decisions of the player given the same (or similar) state of the game, and not playing "well" in terms of game goals.

Indirect imitation tries to measure certain properties of the player's behaviour, then using some form of reinforcement learning the algorithm optimise a fitness function that measures the human-likeness of a controller behaviour.

NE seems to perform well both in terms of the instrumental similarity measure and the phenomenological evaluation by human spectators when imitating human playing style in games like Super Mario Bros [8]. However, in order to use NE to imitate human behaviour, we need to design a fitness function to characterise the human likeness of the automatic game controller. These fitness functions are domain dependent and not easy to define. In this work we take the first steps towards choosing an appropriate set of features for this fitness function.

In games with a restrained number of player outputs and a discrete set of actions, controllers which implement indirect imitation like NE and dynamic scripting perform better than those that implements direct imitation, both in convincing human observers as well as in a raw score [8].

3 Ms. Pac-Man vs. Ghosts

Pac-Man is an arcade video game produced by Namco and created by Toru Iwatani and Shigeo Fukani in 1980. Since its launch it has been considered as an icon, not only for the video game industry, but for the twentieth century popular culture [4]. In this game, the player has direct control over Pac-Man (a small yellow character), pointing the direction it will follow in the next time-step (up, down, left or right). The level is a simple maze full of white pills that Pac-Man eats gaining points. There are four ghosts with different behaviours trying to capture Pac-Man, causing it to lose one of its lives. Pac-Man initially has three lives and the game ends when the player looses all of them. In the maze there are also four special pills, bigger than the normal ones, which make the ghosts

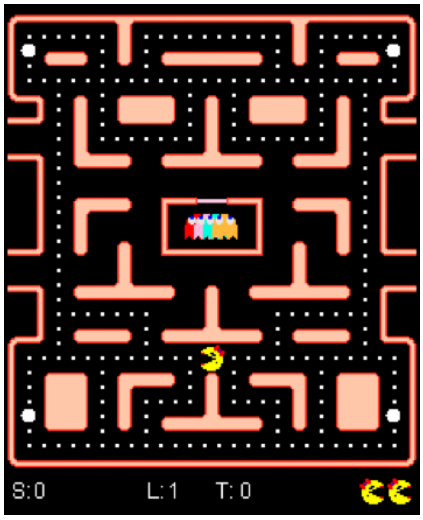

Fig. 1. An screenshot of Ms. Pac-Man vs. Ghosts

to be "edible" during a short period of time. Every time Pac-Man eats one of the ghosts during this period, the player is rewarded with several points.

Two years after the release of the original Pac-Man video game, Midway Manufacturing Corporation (distributors of the original version of Pac-Man in the USA) produced the next version of the game: Ms. Pac-Man. This version of the game introduced a female protagonist, Ms. Pac-Man, and some changes in the gameplay: the game is slightly faster than the original one and, in contrast with the prequel, the ghosts do not have a deterministic behaviour, being their path through the maze not predefined [6].

Ms. Pac-Man vs Ghosts (Figure 1) is a new implementation of the game that provides a Java API to easily developing bots to control both the protagonist and the antagonists of the game, allowing a full access to the game logic and internal structures. In fact, there have been several academic competitions during the recent years (for example in the CIG conference) to build bots with the obvious goal of maximising the score. These bots are able to obtain very high scores but their behaviour is usually not very *human*. For example, these bots can compute optimal routes taking into account the speed of the ghosts and pass very close to them while human players tend to keep more distance and avoid potential dangerous situations.

The decision of using Ms. Pac-Man as our testbed game is mainly due to three factors. Firstly the game presents a discrete state space and the number of actions required by the player is quite limited (continuous movement in only four possible directions). Secondly, we assume that this game is sufficiently different to already explored platform video games as Super Mario Bros, since the movement of the characters is not limited to one axis (plus jumping) but in a two-dimensional maze, the levels are designed with a labyrinthine structure with linear paths (in

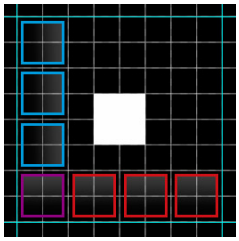

Fig. 2. Any visible tile of the maze internally contains up to 7 nodes.

which only "fits" one character on the way), and there are persistent enemies that chase the player with a non deterministic behaviour. Finally, as it was previously mentioned, we have already worked with this game before [7] and we are familiar with the implementation of this type of bots.

4 Game State Representation

The API of Ms. Pac-Man vs. Ghosts represents the state of the game as a graph in which each node corresponds to a passable region of the maze (a square of 2 x 2 pixels). Each node can be connected to up to other four nodes, one in each direction (north, east, south and west), and can contain a small pill, a big pill, one or more ghosts, Ms. Pac-Man herself or nothing at all (i.e. the node represents an empty portion of the maze). Note that the graph only contains passable regions, so the walls of the maze are never represented, i.e. there are no edges connecting passable nodes to "wall nodes". The full graph representing the state of the game contains 1293 nodes.

Although this graph representation is useful to compute routes for the characters using the A* algorithm, it also provides an excessive level of detail. In order to facilitate learning, we decided to simplify the game state using a tiled-based representation in which each tile corresponds to a screen region of 8 x 8 pixels. One tile can contain either 4 nodes on one side (upwards or side to side) or 7 nodes (see Figure 2): 3 in the base, 3 upwards and 1 shared in the corner. Pills are located always in the center of the tiles (corresponding to the left-bottom corner node, the purple one in the Figure 2) and separated by 4 nodes so that every game cycle Ms. Pac-Man moves one node and every four moves in the same direction Ms. Pac-Man advances one tile. This way the whole maze can be represented as a grid of 26 x 29 tiles (see Figure 3) where some of them represent walls and other are passable. Each tile, in turn, can contain a pill, a wall, one or more ghosts, Ms. Pac-Man or an empty space. We think that this simplified state representation not only reduces the number of different states to consider to a great extend, it is also closer to the human perception of the maze.

In order to reduce even more the number of possible states, in every game cycle we will only consider a small window of tiles around the player (see Figure

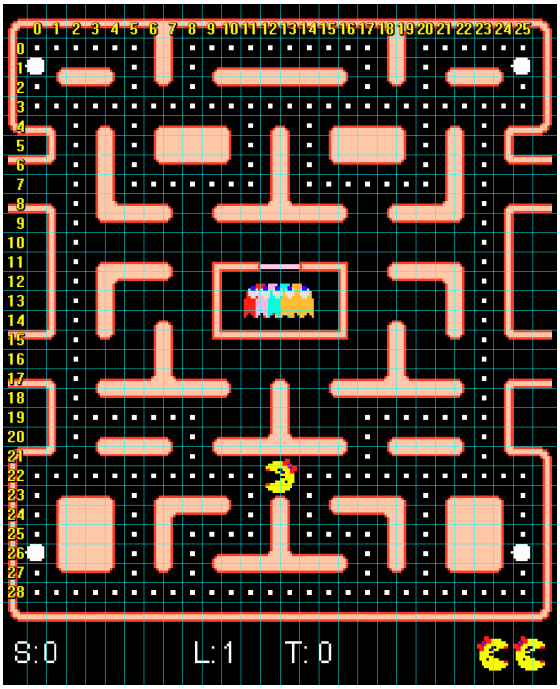

Fig. 3. The whole maze modelled as a grid of 26 x 29 tiles.

4). This approach has proven successful in previous works with artificial neural networks in Pac-Man [2] and it is based on the intuition that the content of nearby tiles is more important and therefore has more influence on the player's decisions. In our experiments we use a 7 x 7 tiles window centered on Ms. Pac-Man.

We keep record of three binary parameters for each tile in the window: (1) if the cell is accessible (not a wall), (2) if there is a pill on it (big or small does not matter) and (3) if there is a non-edible ghost in the cell, (edible ghosts are currently ignored because they are not seen as a danger). Thus, we need 7 x 7 x 3 binary parameters to represent the current game state. Note that even working with this simplified representation the number of potential different states is quite big.

5 Training Set and Feature Selection

In order to train our bot to play like a human player, we asked a particular player to play 100 different games. Each game trace contains an exhaustive representation of the game state every cycle so that the games can be reproduced anytime (a normal game takes around 1200 cycles and there are 254 different parameters to consider in each cycle).

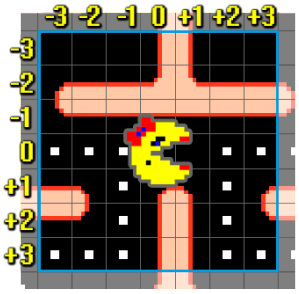

Fig. 4. The state representation is a 7 x 7 tiles window centered on Ms. Pac-Man.

Game states in consecutive cycles are very similar, so we decided to consider only some of them in our training set. In particular, we collect a new game state each time the player changes of tile or changes his direction. That way, from the 100 game traces we extract 34000 game states represented as a 7 x 7 tiles window and for each state we store the direction in which the player was moving.

This way, we can use any supervise machine learning algorithm in order to learn the "right" direction given a game state. This approach is known as a direct imitation strategy and, as we already explained has the problem of generalising the decision-making knowledge to deal with new unseen situations.

To complement the previous training set we also extracted some other parameters at the game level to characterise the style of the player with a higher level of abstraction. These parameters try to measure the behaviour of the player looking at whole games instead of looking at the particular moves:

- *Game duration*: Until the first level is cleared or the player loses three lives.
- *Average distance to the closest non-edible ghost*: It measures the risk the player is willing to take.
- *Average score per second*: It measures the number of pills collected and ghosts eaten.
- *Number of direction changes per second*: It defines different playing styles.
- *Number of ghosts eaten by Ms. Pac-Man in the game*: It measures the player aggressiveness.

Different players will be characterised by different values of these parameters. For example, if the second parameter is very low you could say that the player is playing with recklessness because during the game he does not keep a wide distance from the ghosts. Or if this parameter is low, and also the fifth parameter (number of ghosts eaten) is quite big, one could say that the player is quite aggressive.

These high level parameters will let us to implement a mix strategy in which part of the fitness function depends on direct imitation and other part depends on reproducing playing styles using an indirect approach.

6 Neuroevolutive Algorithm

Our bot is based on an ANN implemented using Neuroph [1], an open source Java neural network framework. A multi-layer perceptron with a single hidden layer and a sigmoid transfer function was used because it is a configuration that has been proved successfully previously [2].

The input of the network is based on the game-state window previously described, in this way the input layer is formed by 7 x 7 x 3 neurons (7 x 7 cells, and 3 parameters per cell), it has one hidden layer with 7 x 7 neurons and the output layer has 4 neurons, representing each of the directions Ms. Pac-Man can move. The direction of the bot is chosen based on the output neuron with the greatest value. The total number of weights in the network is 7452.

We used three different approaches to train the ANN: back-propagation, neuroevolution with a fitness function based on direct imitation and neuroevolution with a fitness function extended with game-level parameters. In the first two approaches we used the training set extracted from the 100 games played by the human player. The precision of the network was computed as the ratio between the number of times the ANN chose the same direction than the human player and the total number of game states in the training set.

The evolutionary algorithm used in this work for the last two approaches is a genetic algorithm (GA) developed in Java to operate directly with the Ms. Pac-Man vs ghosts API. This algorithm implements a codification based on floating points genes for the chromosome of the individuals, so every chromosome in the population represents a different ANN for its genes correspond to the weights of this ANN. At the beginning the GA initialise a population with 200 individuals, all of them created with random genes. In each generation, we use the usual genetic operators (selection, crossover, mutation and replacement) to produce new individuals, we evaluate them using a fitness function and discard the worst. This process is repeated 500 times.

The operators used in the experiments of this work were:

- *Selection by tournament*: 10 individuals of the population are selected randomly. Among these individuals the one with the better fitness value is selected.
- *Plain Crossover*: 2 descendants are generated. For every gene in the chromosome, the value of the gene of the descendant in the position i is chosen randomly in a range defined by the genes of the progenitors that are located in the same position.
- *Mutation by interchange*: Randomly the 1% of the genes are selected and mutates and their values are interchanges.
- *Substitution of the worse*: The descendants replace the individuals with worse fitness from all the population.

We use two different fitness functions to evaluate the human-likeness of the bot:

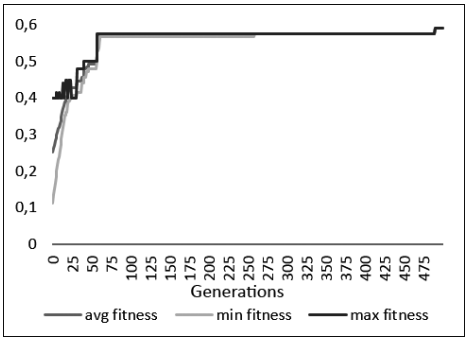

Fig. 5. Evolution of the NE algorithm using cross-validation with the ANN.

- The first one transforms each individual into an equivalent ANN and evaluates the number of times the ANN produces the same direction than the human in each game state from the training set (precision value P_1).
- The second fitness function, besides computing P_1, also plays 50 new games measuring the values of the high level parameters described in the previous section and computes a second precision value based on the mean squared error between the values obtained by the bot and the values obtained by the human player (precision value P_2). The final fitness value of the individual is computed as lineal combination of the 2 precision values. Initially in our experiments we assigned the same importance to both of them (50-50). Note that using different weights we can give more importance to the direct imitation approach or to the indirect one, and in the future we plan to study how that affects the behaviour of the bot.

7 Results and Discussion

The results of our experiments are summarized in Figures 5, 6, 7 and 8.

The precision of the ANN with back-propagation using cross-validation with the training set of the 100 human games in 100 iteration is approximately 50%. Using NE, the precision of the system after 500 generations is 60% (see Figure 5), representing a 20% improvement over back-propagation.

Using only the high level parameters obtained by the bot implemented for the fitness function produced very poor results: %32 precision (see Figure 6).

When we combine NE with high-level parameters (HL Params) and a virtual player (50% ANN precision + 50% quadratic error of the HL Params obtained by the bot) precision is close to 40%. The Figure 7 shows the evolution of the NE algorithm with the final values (after adding the two fitness values), and the Figure 8 shows the evolution with the different fitness values disaggregated (as can be seen, the ANNs cross-validation is considerably better than the fitness obtained by the bot and the HL Params).

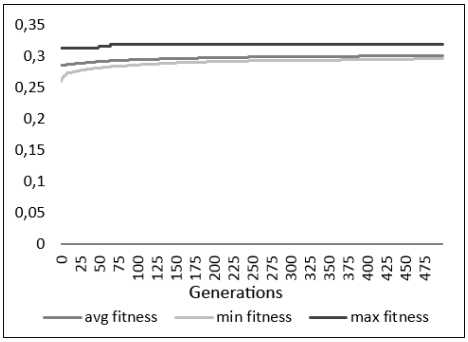

Fig. 6. Evolution of the NE algorithm using the HL Params obtained by the bot.

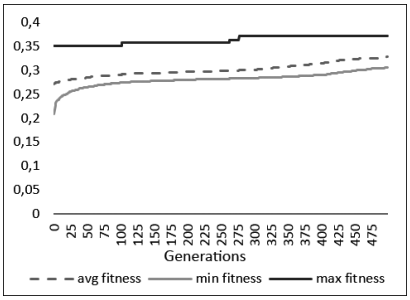

Fig. 7. Evolution (min., max. and average precision) of the NE algorithm using cross-validation with the ANN and the HL Params.

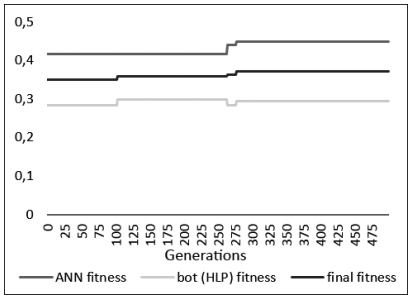

Fig. 8. Breakdown of the different fitness values in the evolution of the NE algorithm.

At this preliminary stage in the system development, ANN virtual player is an incompetent player. It easily enters in non-sense cycles, losing the game really soon. It chooses non-accessible tiles frequently, which shows there is plenty of room for improvement.

Although the NE improves the ANN back-propagation training results in a 20%, we think that before using the NE approach it is important to improve our ANN-only version. 50% of precision is a low value for just 4 directions available to the virtual player.

As part of our experiments we tried to complicate the ANNs topology by adding more neurons in the hidden layer but this did not produced better results, rather the opposite, precision was significantly worse. Despite this, We think the topology of the network should be reconsidered, because it is very simple right now.

Results achieved using only the HL Params obtained by the bot suggest that the controller is not able to play in a minimally acceptable way. As said, this is because it enters easily in cyclical states being unable to explore the level without getting stuck soon, so until this controller is not more competent there

is little point to measure these parameters. In addition, the accuracy obtained when using both fitness (40%) is due in part to the fitness obtained by the bot.

The train set used for these experiments is also not complex enough for learning (34.000 game states are traced in our 100 game sessions, but only 600 are different), so we have plans to enrich it.

8 Conclusions

As part of our research on simulating human play in video games, we are working on training virtual players using Neuroevolution. In our work we use classic arcade Ms. Pac-Man as testbed to compare two different methods. First one uses raw data extracted directly from human traces, while second one also considers more elaborated and abstract parameters as the final score, the average distance to the closest ghost or the number of times the player turns and changes its route.

Our findings suggest that it is necessary to establish a reasonable performance level for automatic controllers before comparing virtual and human players from an external point of view. A significant level of competence in the basics of the game and its strategy seems to be a prerequisite for starting any training for human-like imitation.

It also seems necessary to extend the information on the domain of the game state, as our 7 x 7 window representation does not take into account any type of information outside its tiles. Thus the network does not know if there is an important amount of pills more than 3 tiles away from Ms. Pac-Mans position. A solution for this would be to extend the size of the window (extending significantly the entries in the ANNs) or adding some extra information to the state (for example the number of pills and ghosts in each external quadrant to the window).

We assume that an improved game state representation will mean significant changes in the accuracy of the ANN but also a more complex configuration in the network will have a big impact. So rethink the topology of the ANN will be an important point to consider.

Once the ANN obtained better results, we would improve the bot extending its logic to play in a more competent way, and then measure the HL Params obtained in the fitness function of the NE algorithm.

Another field of study we want to address is using NEAT (NeuroEvolution of Augmenting Topologies) to make the NE not only evolve the weights of the ANN but also its structure [10]. Probably we are also going to explore other areas of machine learning, as Reinforcement Learning, which is a promising technique different to standard supervised learning.

Acknowledgements

Antonio A. Sánchez-Ruiz would like to thank the support of the Spanish Ministry of Economy and Competitiveness under grant TIN2014-55006-R.

References

1. Apache Software Foundation: Neuroph (java neural network framework), http://neuroph.sourceforge.net/
2. Gallagher, M., Ledwich, M.: Evolving pac-man players: Can we learn from raw input? In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, CIG 2007, Honolulu, Hawaii, USA, 1-5 April, 2007. pp. 282–287 (2007)
3. Getomer, M.J., Okimoto, B.J.: Gamers on youtube: Evolving video consumption (July 2013), https://www.thinkwithgoogle.com/articles/youtube-marketing-to-gamers.html
4. Goldberg, H.: All Your Base are Belong to Us: How 50 Years of Videogames Conquered Pop Culture
5. Hingston, P.: A new design for a turing test for bots. In: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Copenhagen, Denmark, 18-21 August, 2010. pp. 345–350 (2010)
6. Kent, S.: The Ultimate History of Video Games: From Pong to Pokémon and Beyond : the Story Behind the Craze that Touched Our Lives and Changed the World
7. Miranda, M., Peinado, F.: Improving the performance of a computer-controlled player in a maze chase game using evolutionary programming on a finite-state machine. In: Proceedings 2o Congreso de la Sociedad Española para las Ciencias del Videojuego, Barcelona, Spain, June 24, 2015. pp. 13–23 (2015), http://ceur-ws.org/Vol-1394/paper_2.pdf
8. Ortega, J., Shaker, N., Togelius, J., Yannakakis, G.N.: Imitating human playing styles in super mario bros. Entertainment Computing 4(2), 93 – 104 (2013)
9. Schaffer, J.D., Whitley, D., Eshelman, L.J.: Combinations of genetic algorithms and neural networks: A survey of the state of the art. In: Whitley, D., Schaffer, J. (eds.) Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92). pp. 1–37. IEEE Computer Society Press (1992)
10. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 7(1), 99,127 (6 2002)
11. Togelius, J., Nardi, R.D., Lucas, S.M.: Towards automatic personalised content creation for racing games. In: 2007 IEEE Symposium on Computational Intelligence and Games. pp. 252–259 (2007)
12. Togelius, J., Yannakakis, G., Shaker, N., Karakovskiy, S.: Believable Bots, chap. Assessing believability, pp. 219–233. P. Hingston (Ed.) (2012)
13. Yannakakis, G.N., Maragoudakis, M.: Player modeling impact on player's entertainment in computer games. In: User Modeling 2005, 10th International Conference, UM 2005, Edinburgh, Scotland, UK, July 24-29, 2005, Proceedings. pp. 74–78 (2005)