Imitating human playing styles in Super Mario Bros

Juan Ortega, Noor Shaker, Julian Togelius and Georgios N. Yannakakis

IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen, Denmark
{*juor, nosh, juto, yannakakis*}*@itu.dk*

Abstract

We describe and compare several methods for generating game character controllers that mimic the playing style of a particular human player, or of a population of human players, across video game levels. Similarity in playing style is measured through an evaluation framework, that compares the play trace of one or several human players with the punctuated play trace of an AI player. The methods that are compared are either hand-coded, direct (based on supervised learning) or indirect (based on maximising a similarity measure). We find that a method based on neuroevolution performs best both in terms of the instrumental similarity measure and in phenomenological evaluation by human spectators. A version of the classic platform game "Super Mario Bros" is used as the testbed game in this study but the methods are applicable to other games that are based on character movement in space.

Keywords: Game AI, Neuroevolution, Dynamic scripting, Imitation learning, Behaviour cloning, Behaviour imitation

1. Introduction

There are several reasons one might want to develop AI that can play a particular game in a human-like manner, or even in the manner of a particular human. To begin with, there is a commonly held assumption that having *believable* non-player characters (NPCs) in a game, for example as opponents or team mates, makes the game more fun and/or engaging. An NPC can be said to be believable when it would be plausible, given the in-game behaviour of the character, that it was controlled by a human. While there is no conclusive empirical evidence that believable NPCs improve games, this topic

has attracted some attention from the research community in recent years and there have even been a few competitions about believable agents organised [1, 2].

Another reason for having AI that can play in a human-like manner is that it can help demonstrate how to play the game — either in general or a particular level. For example, Nintendo's recent *New Super Mario Bros Wii* has a feature called *Super Guide* that can show a player which gets stuck on a level how to play the rest of that level. Being able to demonstrate how to solve the level in the style of that particular player might mean that the advice is more useful or more easily accepted.

A third reason is that it might be useful to understand how a particular player would have played some game content, for example a level, without having the player taking the time to play through that game content. In particular, this is very useful in search-based procedural content generation, where a simulation-based evaluation function uses an AI to play through the candidate game content, assigning a numerical fitness value (partly) depending on how playable the content is. For example, when evolving a platform game level, the fitness of the level might depend on whether an AI can play through the level or not and how fast. As the evolutionary process might need to evaluate tens of thousands of candidate levels when evolving a single level, it is typically not feasible to use human play testers for this. Using an AI that can play the content in a human-like manner can substantially improve the content evaluation, as compared to using an AI that plays the game in a non-humanlike manner. This argument holds even when not generating new content, but merely selecting among already generated content for content that suits a particular player.

It is worth pointing out that not all AI plays games in a human-like manner; quite the opposite. Both controllers that are hand-coded to play a particular game, and controllers that are trained to play a game using some sort of machine learning mechanism, frequently display behaviour that strikes observers as "unnatural" or "mechanical". When Garry Kasparov lost to the IBM software/hardware Deep Blue, he famously complained that the computer played in an implausibly human-like manner, given that all other chess computers played in a distinctly machine-like fashion [3]. A similar tendency is true for the game that is used as a testbed in this article, *Infinite Mario Bros*: the AI that won the first annual Mario AI Competition played some levels in the game extremely well, but it's playing style looked so unnatural that a video of the winning controller playing a level became

an Internet phenomenon [4]. Even when researchers set out specifically to create human-like controllers rather than well-playing ones, success is far from guaranteed. In the 2k BotPrize, which has been organised annually since 2008, competitors submit controllers for the first-person shooter game *Unreal Tournament 2004* with the goal of fooling human judges (who also play the game) that the bots are controlled by humans. So far, the humans have had higher humanness ratings than the bots by a comfortable margin [1, 5].

It is also worth pointing out that human-like behaviour is not the same as artificial general intelligence. The original Turing test has been interpreted as many as a test of "real AI", even though Turing's own view on this was remarkably complex [6]. One common idea of "general intelligence" is a capacity for performing well over a distribution of different problems, or in a distribution of different environments [7]. It has been suggested that games, especially games which can be expressed in a generative language, could serve as good testbeds for artificial general intelligence [8]. However, there is no guarantee that an agent that performs well over a range of games does this in a human-like fashion at all. Creating an agent that performs in a human-like fashion in a particular game might or might not be a step towards creating an agent that performs well over a range of games.

In sum, we believe that creating human-like game character controllers is an important unsolved problem in game AI research. We also believe that a useful way of attacking this problem is to develop methods to imitate humans from gameplay traces.

1.1. Imitating human playing behaviour

A number of attempts to create controllers that imitate human playing behaviour, with varying degrees and types of success, can be found in both the academic literature and among published games. These attempts can broadly be divided into *direct* and *indirect* behaviour imitation [9]. In direct imitation, which seems to be the most common approach, some form of supervised learning is used to train a controller to output the same actions as the human took when faced with the same situation. Traces of human gameplay are used as training sets, with input features being based on descriptions of the state the game was in (typically the environment of the player character) and the target being the action issued by the player. Indirect imitation, on the other hand, uses some form of optimisation of reinforcement learning algorithm to optimise a fitness/reward function that measures the human-likeness of an agent's gameplay.

In racing games, whose constrained action and state spaces make them useful testbeds for research into behaviour imitation, a number of attempts have been made to train car racing controllers that drive like humans using direct imitation. Togelius et al. trained neural network controllers to replicate human driving styles through associating simulated sensor readings with steering and thrust controls in a simple 2D car racing game [9]. A similar approach was taken by Chaperot and Fyfe in a 3D motocross game [10]; both studies used variants of the backpropagation algorithm for training. A few years earlier, the commercial racing game *Colin McRae Rally 2.0* used backpropagation for learning part of the driving behaviour of NPC cars from human examples [11, 12]. In that game, learning was used as a way of assisting the construction of well-playing AI rather than to learn the driving behaviour. Microsoft's *Forza Motorsport*, on the other hand, uses a form of direct imitation to learn the driving style of particular human players so that it can estimate the performance of those players on particular tracks, and also let a player's "drivatar" compete against other players online even in the absence of the original player. Instead of a standard machine learning method, an ad hoc approach was used where racing lines were recorded on prototypical track segments. This proved effective, but limited the shape of tracks that could be used [13].

One problem with direct imitation methods is generalisation to unseen situations. When a controller is trained with supervised learning, it is not rewarded for playing well, only for taking the same actions as the human in all recorded situations. When the trained controller is faced with a situation which does not closely resemble the training data, it is likely to take an action which is very unlike what the player would have taken in the same situation; if the player had played the game at least moderately well, the agent's action is likely to be considerably worse (in terms of e.g. score) than the action the player would have taken. (For example, if a car racing controller has been trained on data from a well-driving human the training data might not include situations where the human drives off the track. When faces with an unknown track segment, it might still drive off the track, but will not know how to get back on the track as this situation is not part of the training data.) Consequently, direct imitation can easily lead to controllers that perform worse than the behaviour they were trained on.

Indirect imitation was proposed as an attempt to overcome the generalisation problem. Examples within racing games include Togelius et al. who evolve neural networks to drive similarly to human drivers according to ex-

tracted features of driving style [9], and van Hoorn et al. [14] who use multi-objective evolution to create driving behaviour which is both human-like and well-performing.

Outside of racing games, attempts at imitating human player behaviour are more scarce. One example is Thurau et al. [15, 16] who train controllers to play a first-person shooter game using Bayesian methods (a form of direct imitation). The gameplay in the commercial game *Black and White* relies on a form of mixed imitation learning and reinforcement learning, where the player teaches a companion creature to act in the game world using demonstrations, rewards and punishment; however, technical details about the algorithms used here are hard to come by.

As far as we know, there has not been any research published on imitation learning in platform games or closely related genres. Neither has there been any study published comparing different behaviour imitation techniques in any genre. This motivates the work presented in this paper in which we will address these two points in detail.

1.2. This paper

In this paper, we compare three different methods for imitating human playing behaviour, and also compare the controllers they produce with three other controllers of varying quality that are not developed specifically with the objective of playing in a human-like manner. The main questions we address are (1) can we create controllers that appear to be as human-like as actual humans to external observers? (2) are controllers trained to imitate humans perceived as more human-like than controllers simply trained to play the game well? and (3) when imitating human behaviour, which controller architecture and training method (direct or indirect) gives the best results?

Section 2 describes the Mario AI benchmark, a version of *Super Mario Bros* that we use as the testbed game; section 3 describes the different controllers that are tested. In section 3.1 we describe the method for automatically evaluating the human-likeness of these controllers, which is also used as an evaluation/fitness function for indirect imitation. Section 4 describes how we gathered data from human players, and section 5 describes the training of the controllers and results of automatic testing. Section 6 reports the results of using human spectators the judge human-likeness of our controllers. We conclude by discussing what this research tells us about prospects for imitating human behaviour in platform games and other games.

2. The testbed game

The testbed game used for the study presented in this paper is a modified version of Markus Persson's *Infinite Mario Bros* which is a public domain clone of Nintendo's classic 2D platform game *Super Mario Bros.* The original Infinite Mario Bros and its source code is available on the web[1]. The game has the advantage of being well known among the general public,

This game was made into a benchmark ("The Mario AI Benchmark") for the Mario AI Championship [2], a series of competitions that have been running in association with several international academic conferences on games and AI since 2009. The Mario AI Championship has four tracks: the Gameplay track, where competitors submit controllers that are judged on their capability to play the game as well as possible [4, 17]; the Learning track, where submitted controllers are allowed to play each level 10000 times before being evaluated, in order to test the capability of the controllers to learn to play particular levels [17]; the Level Generation track, where competitors submit level generators that are judged on their capacity to generate engaging levels for human players [18]; and the Turing Test track, a recent addition where submitted controllers compete for being the most human-like, as judged by human spectators [2].

Different versions of the Mario AI Benchmark have been in a number of research projects including but not limited to: player experience modelling [19, 18], procedural content generation [20, 21, 22] and game adaptation [18].

The gameplay in Infinite Mario Bros takes place on two-dimensional levels in which the player avatar (*Mario*) has to move from left to right avoiding obstacles and interacting with game objects. Mario can move left, right or duck using left, right, and down arrow keys. An additional two keys can be used to allow Mario to run, jump, or fire (depending on the state he is in). Mario can be in one of three states: *Small*, *Big*, and *Fire*.

The levels' difficulty increases as the player advances in the game by presenting complicated matters such as gaps and moving enemies. Gaps can be of different width, and wide gaps requires a combination of different keys to be pressed together for a number of game cycles in order to reach the other side of the hole. Enemies can be of different types necessitating

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

[2] http://www.marioai.org/

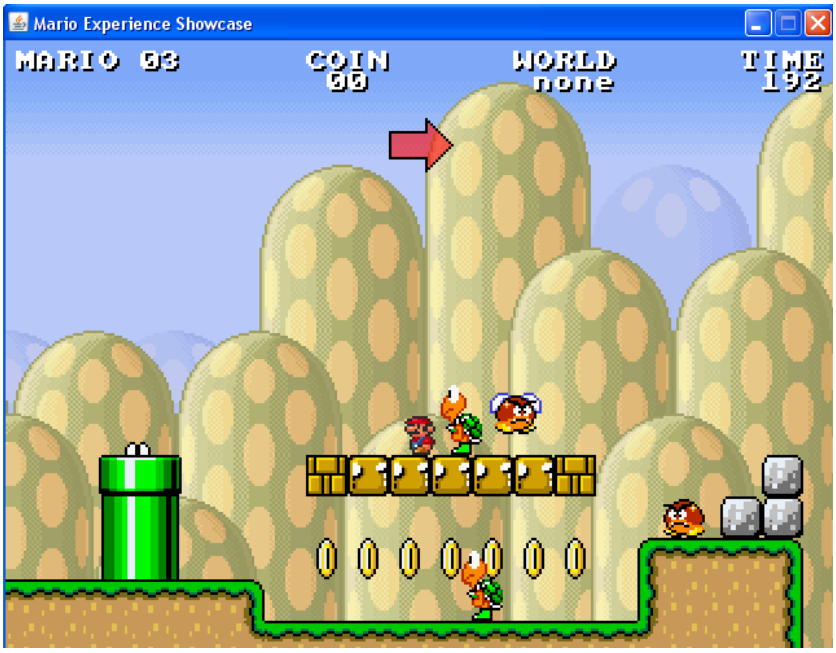


Figure 1: Snapshot from Infinite Mario Bros, showing Mario standing on horizontally placed blocks surrounded by different types of enemies.

different types of behaviour and affecting the level of difficulty. Mario can avoid enemies by shooting fireballs or turtle shells, stomping or jumping over them.

If Mario falls down a gap, he loses a life. If he touches an enemy, he gets hurt; this means losing a life if he is currently in the Small state. Otherwise, his state degrades from Fire to Big or from Big to Small.

Mario can interact with a certain number of items scattered around the level such as coins placed out in the open or hidden inside blocks. Two different types of blocks presented and Mario's ability to interact with each type depends on his state. Blocks hide coins, mushrooms which make Mario grow Big, or flowers which make Mario turn into the Fire state if he is already Big. These items appear only when Mario jumps at these blocks from below so that he smashes his head into them.

The main goal of each level is to reach the end of the level as fast as possible. Auxiliary goals include collecting as many as possible of the coins that are scattered around the level and killing as many as possible of the enemies.

The original Super Mario Bros game does not introduce any new game mechanics after the first level, and only a few new level elements (enemies and other obstacles). The player's interest is kept through rearranging the same well known elements throughout several levels.

For this paper, 40 different levels were generated using the standard level generator supplied with the Mario AI Benchmark. The same levels are used for gathering data from human participants, for evaluating the performance of controllers and for creating the videos which were used in phenomenological evaluation. These levels vary somewhat in difficulty, and are comparable in difficulty and general layout to those levels that were used in the 2009 edition of the Mario AI Competition.

2.1. Environment representation

In order to create a human-like controller using the Mario Bros testbed previously explained, a representation of the level environment was established. This representation was used as to model players behaviour in the game and to apply that representation to the artificial neural network (supervised learning and neuroevolution) implemented. The Mario Bros testbed provides various information about the current level that is being played so it can be accessed and recorded. This information is divided into level scene information and enemies information. The level scene information matrix contains all the different elements that appear in the screen (the screen is divided into cells which are mapped in the level scene matrix) in a concrete moment during the game, representing the different elements with numbers. The enemy information matrix works in the same way, representing the different enemies as numbers. The representation used consisted on two grids, one with the information of the enemies and one with the information of the level. The size of the two grids is 4x7, as it can be seen in figure 2, where both grids are marked.

Apart from the two grids previously described and in order to be more precise with the knowledge representation, additional information was added to it:

- Gap in front of Mario: Calculates the distance to the gap and returns 0 if it is further than three cells in front of Mario, 0.33 in case it is 3 cells in front of Mario, 0.66 in case of 2 cells and 1.0 in case the gap is just one cell in front of Mario.

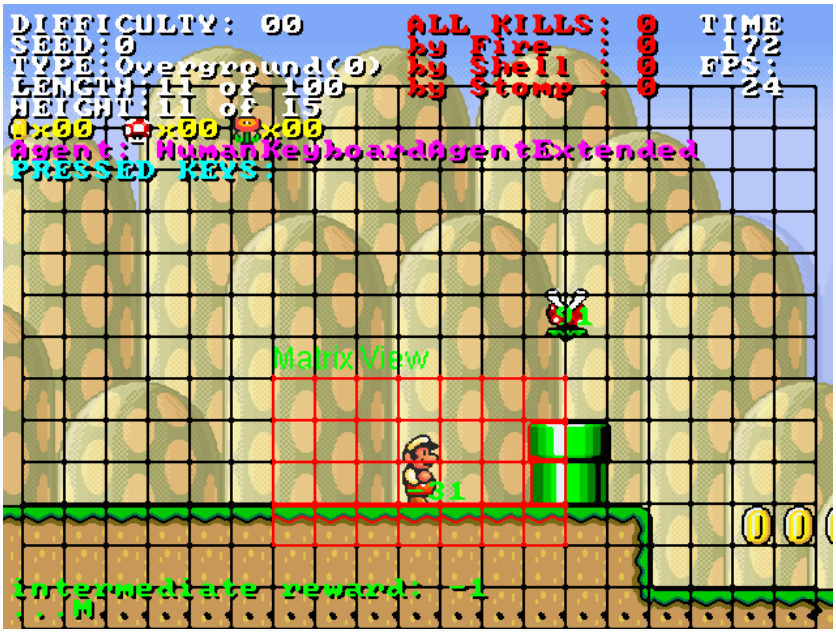


Figure 2: Mario grid representation. Both enemies and level grids contain the information from the same area around Mario which appears marked.

- Mario is able to shoot: Returns 1 if Mario is in fire mode and able to shoot or 0 otherwise.
- Mario is able to jump: Returns 1 if Mario is able to jump or 0 otherwise.
- Mario is on the ground: Returns 1 if Mario is on the ground or 0 otherwise.
- Mario facing direction: Returns 1 if Mario is facing to the right or 0 if Mario is facing to the left.
- Mario is carrying: This checks if Mario is carrying a shell or not
- Mario mode: It returns a number representing the mode of Mario: small (0), big (0.5) or fire (1).
- Distance to obstacle: Works in a similar way as for checking the distance to a gap, giving a number according to the distance to an obstacle.

- Distance to power-up: It is calculated in the same way as the distance to an obstacle and to a gap.

Thus, the environment representation is compound by a total of 65 elements, used as inputs for the Artificial Neural Network.

3. Controllers

Once the environmental representation was set up, six different methods were used and compared in order to simulate human behaviour. The methods are based either on hand-coded rules, direct (as in supervised learning) or indirect representation using a fitness value. Three of the methods used were implemented specifically for this study. They are based on backpropagation, neuroevolution and dynamic scripting. Those methods are used both for instrumental similarity measure and the phenomenological evaluation by human spectators. In the first subsection the evaluation fitness used for neuroevolution and dynamic scripting is introduced.

Two other methods, which were used for the phenomenological evaluation, are based on previous controllers created by participants of the *Mario AI Championship* (REALM and Grammatically Evolved Behaviour Trees). The last method (Forward Jumping agent) consists in a hand-coded rule-based controller, included in the testbed game, which was used as baseline for the specific human imitation experiments.

3.1. Evaluation fitness

This subsection presents the framework designed to evaluate the human-likeness of a given agent. We generally let the agent play on several levels, compare the traces of the agent's trajectory through space with those left by one or several human players and assign the agent an *imitation fitness* score. This score can be used either to assess the human-likeness of an already finished agent or as a fitness function when evolving human-like agents (such as in the case of neuroevolution and dynamic scripting). A key innovation is that the framework does not only create a single trace per agent and level, but resets the agent's position whenever it deviates too much from the human trace in the same level, in order to achieve a more accurate and less biased score. Agents can be evaluated against a specific player or against a set of players.

Different approaches were considered in order to compare the performance of a controller against the performance of a human player. The first approach considered the positions the agent and the human player passed through a level. Once these traces were calculated for both the human and the AI player it was possible to visualize and compare them via an error value that is based on the distance between the traces' points. The error can be measured at different frame rates providing dissimilar error value approximations with respect to time granularity. The problem with this approach are the disproportionally high error values obtained when the Mario AI controller gets stuck in a level because of a dead end, cannot overcome an obstacle or dies early. This is a problem as the error continues increasing until the end of the trace, skewing the error measure considerable towards errors early in the trace.

Thus, a new approach was considered resetting the AI agent's position to the one that the human player had in the same frame, only if the distance between the human player and the AI agent exceeds 200 pixels in that frame. Therefore the fitness function (f) used for imitating specific human behaviour is based on the number of times the controller had to reset its position to the human player (R) plus the distance error (D) used as tuning value across all levels (L) played (see eq. (1)).

The number of repositions is normalized between 0 and 10 and multiplied by 100 while the average distance error is normalized between 0 and 50. By giving a higher weight to the number of reposition it was intended to use the distance error as tuning value. Having a number of reposition within 0 and 1000 plus a distance error between 0 and 50, resulted in a better fitness differentiation than if the error distance were normalized within a higher range as it would overlap the values obtained from the number of repositions not being possible to use the distance error as tuning value.

$$f = \sum_{i=1}^{L} \frac{(R * 100 + D)}{L} \tag{1}$$

3.2. Heuristic

The first controller architecture tested, is based on hand-coded rules that feature no learning and hardly even takes the game environment into account: the policy is to make Mario constantly run right and jump whenever possible. This approach is very simple compared to the other methods, and included

for comparison purposes as it is one of the example controllers distributed with the Mario AI benchmark, called *Forward Jumping Agent* (FJA).

3.3. Artificial neural networks

An artificial neural network (ANN) is a machine learning technique loosely inspired by the principles and structure of biological neural networks. In order to simulate human behaviour, two different methods have been implemented using ANNs: Supervised learning and Neuroevolution. In both approaches, the environment information (from the grid representation) from each frame is used as input to the neural network and the keys outputs are interpreted as keys pressed.

3.3.1. Supervised learning

The first ANN training method makes use of a direct representation by using the game environment information obtained from human gameplay as training set. Thus, an approach based on supervised learning [23] was implemented as it can be used as a behaviour cloning method. This approach uses backpropagation in an attempt to minimize the error between human player actions and ANN outputs.

Gameplay data from a set of different human players (10 players playing through 40 different levels), has been collected and used as training samples for the ANN (more details are provided in section 4). The training set contains the environment state for each frame using the representation presented (inputs) and the actions the player carried out for that state based on the keys pressed (outputs).

Once the information from the players was gathered, data preprocessing was carried out by biasing the different state-action pairs so every action appears the same number of times. This is done by sorting all the state-action pairs according to the number of times the same action was performed. After this, the number of times the most repeated action appears is obtained. Thus, the rest of the actions are copying until they get the same number as the most repeated action. By doing so, we make sure that all actions had the same opportunity of being learned by the ANN.

The ANN topology used incorporates 65 inputs, 5 perceptrons in one hidden layer and 5 outputs as it presented faster convergence compared to other topologies used (refer to table 1). In order to train and validate the ANN, 10-fold cross-validation was used. The ANN was trained during 2000 epochs (iterations) or until the error decreased to a value lower than 0.001

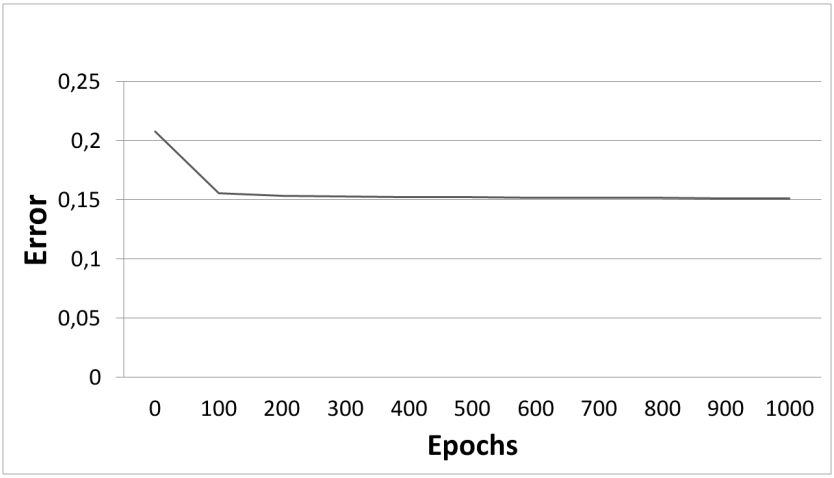


Figure 3: Error (grey line) decreases after ANN is being trained

per fold. The error was calculated as the mean squared error between the actual and the desired output.

Topology	**Time (ms.)**
65-5-5	735000
65-10-5	1094359
65-15-5	1537828
65-5-5-5	849719
65-10-5-5	1308234

Table 1: Time performance measurement with different ANN topologies. Time is measured using each of the 40 training set corresponding to the 40 levels, during 2000 epochs. These results were obtained in an Intel Core 2 Duo 1.83Ghz, 2Gb RAM

The ANN weights obtained after the training phase, were used as initial weights of the Neuroevolution approach. Therefore the same topology was used for that method, having a chromosome size smaller than if a bigger topology were used (table 1). The maximum number of epochs was set to 2000 as the error presented a small variation after 1000 epochs (figure 3). The learning rate was set to 0.3 as it gave the lowest error compared to other values (figure 4).

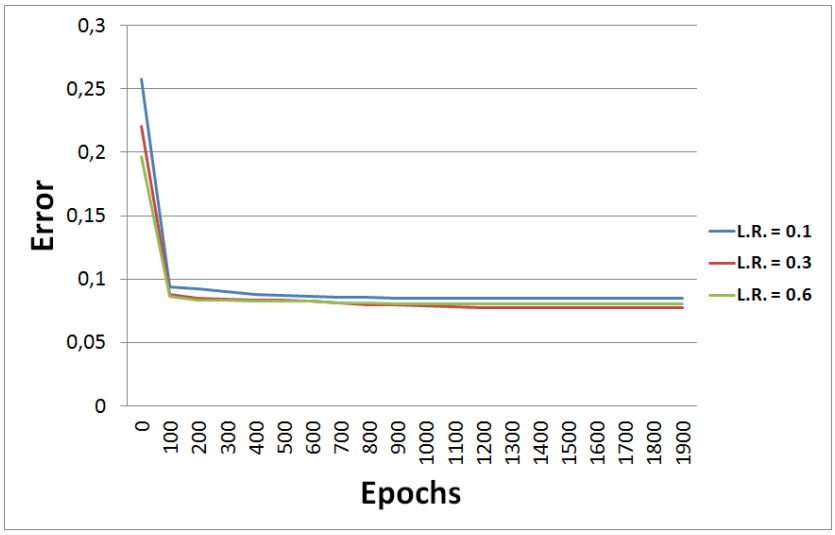


Figure 4: Learning rate experiments showed that 0.3 gave the lowest error

3.3.2. Neuroevolution

The second approach for ANN training is based on an indirect representation where the weights of the ANN are adjusted by means of Evolutionary Algorithms (EA). This neuro-evolutionary (NE) [24] approach attempts to minimise a fitness value corresponding to the mean squared error distance from the desired outputs (human actions). The fitness function used was the one explained in detail in section 3.1. Both the inputs and the topology of the NE controller are the same as the corresponding ANN used for supervised learning.

Each chromosome (ANN) is evaluated on 40 levels. These levels present an increasing difficulty based on the number of enemies, obstacles and gaps width. The levels used do not present a long distance from the beginning to the level goal: it is possible for a player to complete one level within 50 to 90 seconds of in-game time. Hills are easy to overcome and there are no mazes to solve in order to get to a further point. The fitness is calculated as the average of the fitness attained on those levels. The genetic algorithm used for evolving the ANN weights contains a population of 10 chromosomes, uses roulette-wheel selection, two-point crossover and Gaussian mutation with standard deviation equal to 5.0 and probability of 0.3. The network is evolved for 2000 generations.

The mutation probability was set to 0.1 after experimenting with different

probabilities (refer to figure 5) and the population was set to 10 chromosomes as using a bigger one (15 chromosomes) resulted in more than 25 hours of training for NE and DS, while with 10 chromosomes got down to 23 hours.

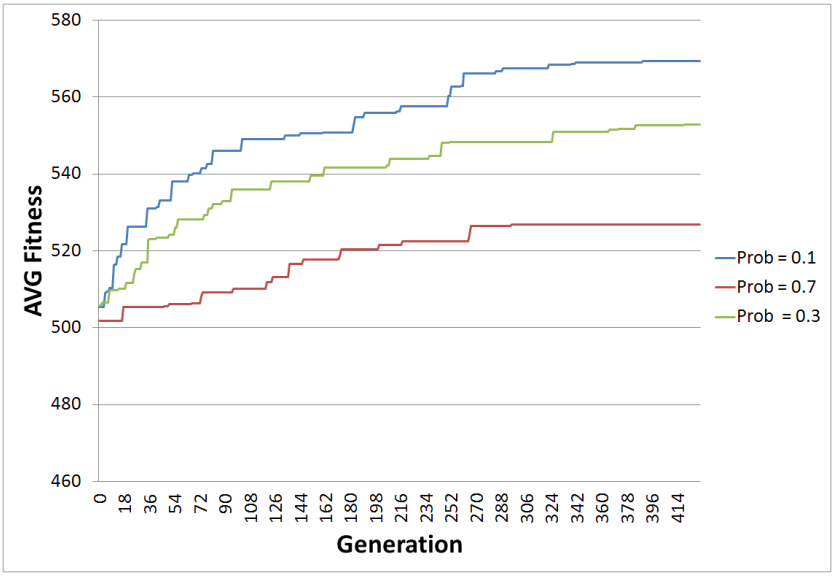


Figure 5: Mutation probability shows that 0.1 gets the highest average fitness

3.4. Dynamic scripting

Dynamic Scripting (DS) is a reinforcement learning technique which resembles learning classifier systems. It is described by Spronck et al. [25] as "an online competitive machine-learning technique for game AI, that can be characterised as stochastic optimisation". DS contains a rule base (RB) with the possible rules that can be applied into a game. Each rule has a weight which reflects how well that rule made the agent perform in prior games. In every game, a script is generated using roulette-wheel in order to select a small subset of the rules in the RB. The agent will play according to the rules contained in the script and those will have their weights updated via a standard Widrow-Hoff delta rule which is based on the immediate reward received by the environment.

The dynamic scripting method provides an interesting comparison to the methods based on neural networks as it represents a very different policy representation; unlike the neural networks, this representation is symbolic,

which makes it more prone to discrete transitions and makes it more human-readable.

DS will update the weights of the rules not after one action is performed but in the end of the game, adding a reward or a penalty proportional to how well the agent performed in a level. This goodness value is provided by the fitness function (F) in order to calculate the weight adjustment after a game. A reward (R) and a penalty (P) constants are used (see eq. 2) [25]. Whether penalty or reward is calculated in order to adjust the weights, is given by the break-even point (b). Thus if (F) is greater than (b) a proportional reward is given otherwise the delta weight value is calculated using the penalty value.

$$\Delta W = \begin{cases} -\lfloor P_{max} \dfrac{b - F}{b} \rfloor & \text{if F} < \text{b} \\ \\ \lfloor R_{max} \dfrac{F - b}{1 - b} \rfloor & \text{if F} \geq \text{b} \end{cases} \tag{2}$$

3.4.1. Modification to the DS algorithm

Originally Spronck et al. [25] developed DS to allow game AI to adapt its difficulty level based on the player's performance. In our case, the implementation created differs in this principle as we want to obtain the most human-like Mario AI controller, meaning the one that best imitates a specific human player. Once the algorithm is executed and after a maximum of 2000 iterations (forty levels are played in each iteration), the agent will store the best script generated, i.e. the one that obtained the best reward value during those iterations. This algorithm modification implies that, in our case, there is no adaptability in the final script as the rules that are contained in it are the ones that are always going to be used by a player. Another aspect that differs from the standard DS implementation is the way the RB is created.

In DS, the RB contains the rules introduced by hand by the game designer. In our approach the rules are generated by creating all the possible combinations of conditions and actions. Therefore one rule can contain a combination of one to three conditions and one action. The amount of conditions that can be selected are six and the number of actions are five. By assigning a number to the conditions from one to six and a number to the actions from one to five, it is possible to generate all the possible permutations (up to three conditions and one action) without symmetry from those numbers, adding the resultant rules to the RB.

3.4.2. Rules

Rules in the dynamic scripting implementation are structured as sets of conditions and actions with a restricted maximum number of conditions and actions per rule. The maximum number of conditions and actions per rule were set to three and one, respectively. The six possible conditions are defined using the environment variables provided by the Mario AI Benchmark:

- Mario able to jump
- Mario on the ground
- Obstacle in front
- Gap in front
- Mario able to shoot
- Enemies in front

Rule actions are based on *high-level actions*, meaning any action that is compound by the combination of different keys repeated during a predefined number of frames. These high-level actions are:

- **Walk**: The right key is pressed. This action is carried out during one frame.
- **Run**: The right key and the speed key are pressed. This action is carried out during one frame.
- **Right Small Jump**: Jump and right keys are pressed during two frames.
- **Right High Jump**: Jump and right keys are pressed during ten frames
- **Avoid enemy**: According to where the enemy is, compared to Mario's facing direction, the action will return the opposite enemies direction key (Left or Right) and jump. This is done during three frames.

Examples of rules can be seen in Table 2.

Conditions			Action
Gap in front?	Mario able to jump?		Right High Jump
Enemies in front?	Mario on the ground?	Mario able to jump?	Run

Table 2: Two examples of rules in the DS rule base.

3.4.3. Dynamic scripting parameters

Regarding the parameters used in the implementation of the algorithm, the maximum and minimum weight threshold are set to 2000 and 0, respectively. The script size was set to 10 rules as those proved to be sufficient to cover a large diversity of agent behaviours without slowing down the performance during the agent's training phase. For the break-even point the value selected was 0.3 as it provided a fast distinction between those rules that returned a high reward and those that gave a low reward value. The reward and penalty values are 700 and 250 respectively, because as the RB contains many different rules (315 after getting rid of the symmetric ones) giving a much a higher value to the reward, speeds up the process of differentiating between the rules that better adapt to a specific player and those that do not. Related to the break-even point and the reward and penalty values, the formula used for adjusting the rules weights is the same as the one proposed by Spronck et al. [25] (eq. 2).

3.5. REALM

REALM is the name of the agent architecture created by Slawomir Bojarski and Clare Bates Congdon for the *Mario AI Championship* in 2010. This algorithm, which is a "rule-based evolutionary computation agent", won the final phase of the Mario AI Championship (gameplay track) in 2010 [26]. It also won the Turing Test track, a human-likeness competition among Mario agents [2].

REALM follows the principles of learning classifier systems, by which rules are evolved according to a fitness value [27]. Each rule contains conditions based on different information obtained from game including Mario size, enemies' positions, presence of power-ups and presence of blocks in the level. The head of a REALM classifier includes high-level plans of actions instead of simple reactive combinations of keys presses. The possible actions are as follows: Progress, Attack, Evade and Powerup.

Compared to DS, REALM differs in several respects: 1) the rule base in REALM is evolved via playing 12 levels and using a fitness value based

on the distance travelled, number of wins and deaths, Mario mode and time left [26]; 2) each chromosome contains a single rule set of 20 rules and the size of the population is fixed to 50; 3) each rule contains seven conditions which may take the values true (1), false (0) or irrelevant (#); and 4) Whereas DS selects low-level control outputs corresponding to key presses, REALM uses high-level actions combined with the A* pathfinding algorithm which allow the controller to perform well in difficult levels containing several obstacles, gaps and enemies.

3.6. Grammatically evolved behaviour trees

The last Mario controller used is based on Grammatical Evolution of Behaviour Trees (GEBT). This controller, which participated in the gameplay track of the 2010 Mario AI Championship, was created by Diego Perez and Miguel Nicolau [28]. Behaviour trees, which have recently become popular in commercial game AI, provide a top down organisation from the root of the tree down to the leaves [29]. The control nodes are those that decide which branches of the tree will be executed next, while the leaf nodes contain the actions that are going to be carried out. The different elements of the tree are specified in a grammar which is evolved by applying genetic operators to the sub-trees created. While the evolutionary mechanism is similar to that used in neuroevolution, the behaviour tree representation differs significantly from both neural networks and dynamic scripting.

4. Data gathering

The machine learning approaches examined in this paper (neuroevolution, backpropagation and dynamic scripting) require recorded data from play sessions in order to be able to imitate human gameplay. Human gameplay data is also required for testing the humanness of each controller by comparing it against human play. For these purposes, data was gathered from different human players with a wide range of experience playing games in general and Super Mario Bros in particular. Note that the differing expertise levels and playing styles of players make it possible to play the same level in different ways; therefore, the performance of imitation learning is expected to differ from one player to another. Data was collected from ten players who played through 40 different levels. The players played the game locally in a java applet, while the gameplay data was logged to a file. The data gathered

contains the game state and the corresponding action taken for that state as well as the Mario traces in each level.

It could be argued that ten players is a very small sample. However, each player played 40 games, so that each imitation-based controller was trained on 400 games. (Playing 40 games takes time, so recording data from ten players required nontrivial effort.) Additionally, the number of players used to collect data is less interesting as this simply represents what the controller is trained on; it would in our opinion have been methodologically correct to train on data from a single player if we wanted to imitate that player's behaviour, as long as an appropriate number of different people judged the player-likeness of the controllers derived from that player's behaviour. The more interesting number is that 67 spectators judged the human-likeness of the agents – see section 6 for more on this.

Prior to the play session the players were asked to rate — on a scale between 1 (low) and 5 (high) — their experience with playing games in general and their experience with playing Super Mario Bros. The average self-rated experience with games in general was 3.3 (min=1, max=5) and with Super Mario Bros 2.7 (min=1, max =4).

5. Experiments

In this section, we describe the results of the training the controllers and evaluating their performance in playing the game, and their human-likeness as judged by similarity behaviour similarity to the human gameplay traces used to train them.

5.1. Performance score

The first experiment presented compares the three controllers implemented (Backpropagation, NE and DS) in terms of the average score obtained through the 40 levels (L) (see eq. 3). The score (S) is given by the Mario AI benchmark [4, 17] which is used in the Mario AI competition, as a weighted sum taking into account different factors: the distance travelled in grid blocks (DT), number of flowers devoured (FD), if Mario has won (W), Mario mode (M), mushrooms devoured (MD), green mushrooms devoured (GMD), coins collected (CC), hidden blocks found (HB), number of kills (K), number of enemies killed by stomp (KS), number of enemies killed with a shell (KWS), number of enemies killed with fire balls (KWF) and the time left (TL).

$$S = \sum_{i=1}^{L} \frac{(DT * 1 + FD * 64 + W * 1024 + M * 32 + MD * 58 + GMD * 58}{L}$$
$$\frac{+CC * 16 + HB * 24 + K * 42 + KS * 12 + KWS * 17 + KWF * 4 + TL * 8)}{L} \tag{3}$$

The controllers are trained independently on the data of each of the 10 human players; Table 3 shows the results obtained for each controller and human player.

Player No.	**Player's Score**	**BP**	**NE**	**DS**
1	2526.5	217	267.0	261.9
2	2933.3	451	247.2	1996.3
3	3471.9	136	1505.0	1328.7
4	2853.1	711	1475.0	1713.1
5	3518.9	147	235.0	1253.7
6	2674.0	214	1084.0	736.9
7	3752.2	1069	1266.0	1710.7
8	3375.9	180	235.0	253.1
9	2048.6	222	1136.9	1145.4
10	1974.1	173	842.2	733.3
Average	2912.8	352	829.3	1113.3

Table 3: Average score across all 40 levels played obtained for the three different approaches (BP, NE and DS) trained to imitate a specific player. For comparison purposes the table includes each human player's average score across the 40 levels played. T-Test results show that human player's score is significantly higher, while NE and DS scores are higher than BP. T-test calculation details are explained below.

The results obtained in 3, showed that NE and DS have higher score compared to BP, but they are still way lower than Human. NE gets a score higher than 1000 for players 3, 4, 6 , 7 and 9 while DS gets a score over 1000 for players 2, 3, 4, 5, 7 and 9. Thus, both algorithms present high scores for players 3, 4, 7 and 9. According to the players game experience classification (table **??**), player 3 and 7 have more experience than 4 and 9, while 4 has more experience than 9. This shows that there is not a clear relationship between the score obtained by the controllers and how good the imitation of

a player is. BP gets the lower score as it gets stuck and dies more often than NE or DS.

To test the significance of the score obtained between the different approaches and the human players we apply a t-test (significance is 0.05 in this paper). The results of the t-Test, show that human players have a significantly higher score compared to the other methods (BP: T = 11.72 , p-value < 0.0001; NE: T = 8.05, p-value < 0.0001; DS: T = 6.57, p-value < 0.0001) while NE and DS scores, are significantly higher compared to BP (T = -2.44, p-value = 0.02526 and T = -3.54, p-value = 0.0035, respectively). NE compared to DS, on the other hand, does not show significantly different results (T = -1.11, p-value = 0.281608).

5.2. Imitation fitness

The experiment presented in this subsection focuses on comparing the average fitness obtained through the 40 levels for the controllers implemented. The three baseline controllers that make use of general fixed rules (REALM, GEBT, FJA) are compared against the three controllers implemented (BP, NE and DS). Table 4 presents the results obtained for each controller which demonstrate that the designed fitness function was successful for generating agents that imitate human game players.

Player No.	**BP**	**NE**	**DS**	**FJA**	**REALM**	**GEBT**
1	450	641	638	161	305	391
2	421	624	682	121	266	318
3	352	587	641	355	529	540
4	643	774	750	427	650	688
5	346	576	578	301	453	416
6	598	678	688	204	351	434
7	531	608	614	548	633	605
8	406	547	589	140	294	342
9	741	864	905	450	655	762
10	698	886	867	411	578	722
Average	518.6	678.5	**695.2**	311.2	471.4	521.8

Table 4: Average imitation fitness obtained by every agent imitating each player, through the forty levels. The results obtained from the t-Test show that NE and DS fitnesses are significantly higher than the others while BP fitness is significantly higher than FJA. GEBT and REALM have a significantly higher fitness than FJA. T-Test calculation details are explained below.

A key observation from the results obtained is that differences in fitness values across players does not correlate perfectly with the quality of imitation. Instead, those differences could be due to the fact that some players (such as Player No. 9 and 10) died early in the level which results to a low number of repositions and therefore to a high fitness. It is, however, possible to compare the fitness value obtained for each controller imitating the same player.

Figure 6 shows that the NE and DS mechanisms yield controllers of the highest fitness values compared to the other four mechanisms demonstrating that the fitness function chosen is successful in generating Mario agents that imitate human players well. Moreover, the BP mechanism — which is the controller that yields the worst fitness among the three machine learning mechanisms — on average achieves higher fitness than REALM and FJA while performing similarly to GEBT.

In order to test the significance of the fitness obtained between the different approaches used we apply a t-test (significance is 0.05 in this paper). The results of the t-Test show, that compared to the other methods, NE (BP: T = -2.68, p-value =0.015; DS: T = -0.32, p-value = 0.753; FJA: T = 6.03, p-value $<$ 0.0001; REALM: T = 3.3, p-value = 0.0004; GEBT: T = 2.43, p-value = 0.026) and DS (BP: T = -3.05, p-value = 0.0073; FJA: T = 6.48, p-value $<$ 0.0001; REALM: T = 3.66, p-value = 0.002; GEBT: T = 2.75, p-value = 0.014) obtained a significantly higher fitness compared to the other methods but not compared to each other. BP (FJA: T = 3.15, p-value = 0.005; REALM: T = 0.7, p-value = 0.49; GEBT: T = -0.05, p-value = 0.96) presented significantly higher fitness compared to FJA but is not significantly different from REALM and GEBT. REALM (FJA: T = -2.33, p-value = 0.031; GEBT: T = -0.7, p-value = 0.49) and GEBT (FJA: T = -2.99, p-value = 0.007) got significantly higher fitness values compared to FJA but they are not significantly different compared to each other.

5.3. Traces comparison

Another way of testing the efficiency of the controllers obtained and assessing their capability of imitating different human players focuses on the comparison of the different traces left by both the controller and the player on the same level. For space consideration only a small subject of example traces are illustrated in Figure 7; in that figure we illustrate the traces of two players on a particular level as well as the traces of the NE and DS mechanisms that attempt to imitate them.

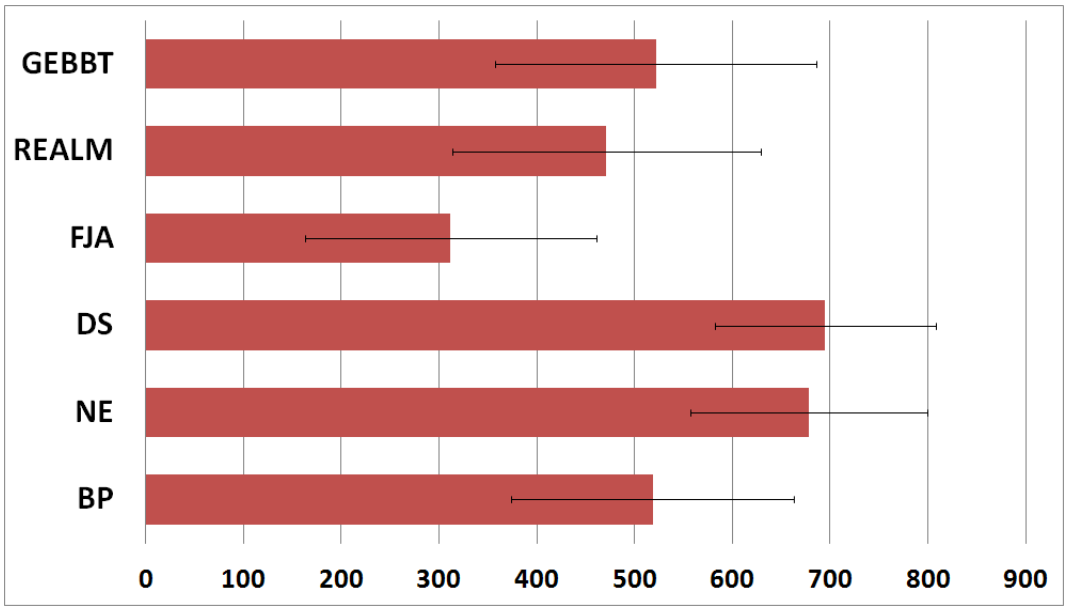
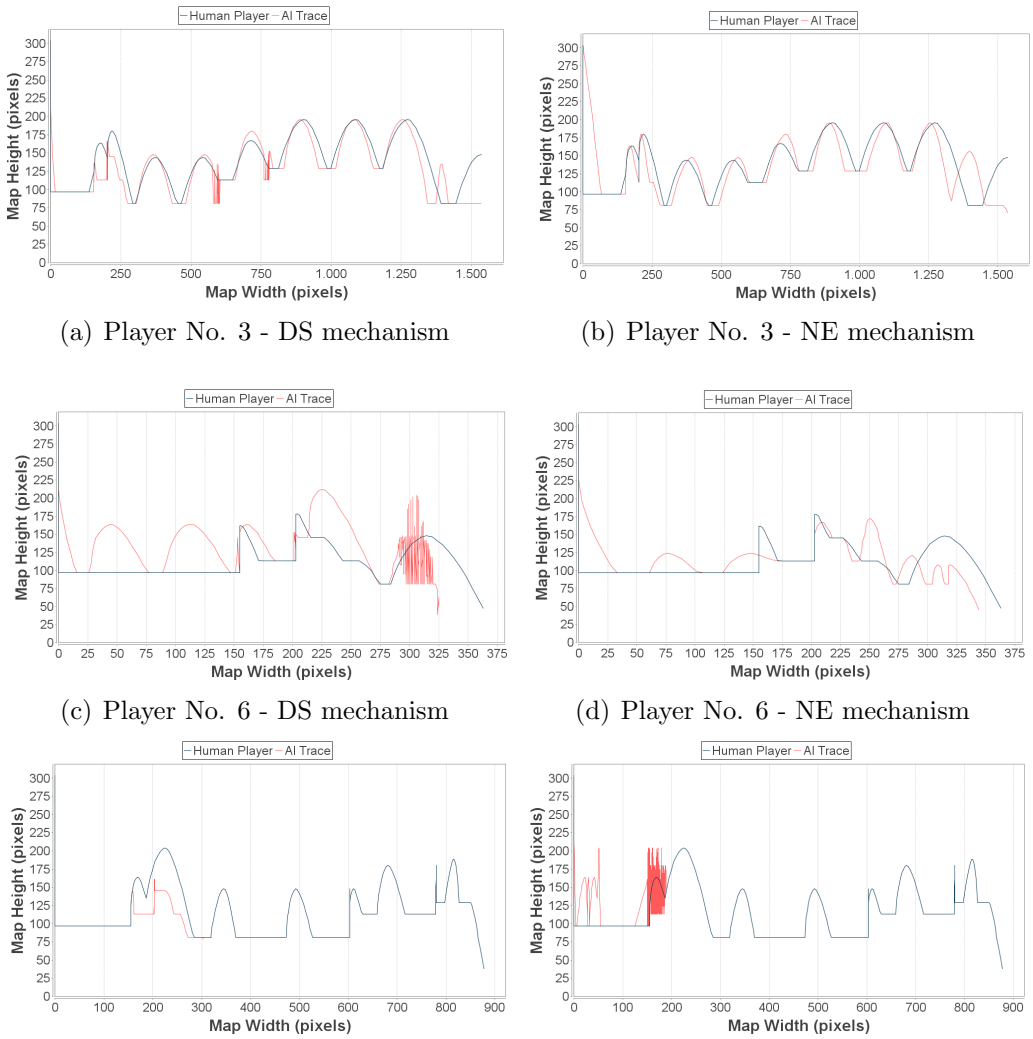


Figure 6: Fitness comparison between the three machine learning mechanisms examined (BP, NE and DS) and three baseline controllers (REALM, FJA and GEBT). The bars appearing in the graph are the average fitness values across all 10 players.

It can be appreciated that NE and DS adapt to both players rather than using the same actions in both cases. Player 6 dies early in the level and so does the controllers imitating him (figures 7(c) and 7(d)), while Player 3 gets to complete the level and so does both NE and DS controllers showing pretty similar traces (figures 7(a) and 7(b)). Player 3 DS and NE traces are quite similar, although DS presents more jumps in some concrete areas. For Player 6 using DS, jumps are higher compared to the traces obtained for NE and in the end of the trace we can see that the agent is jumping around the same area (figure 7(c)). By replaying DS in that level, it was observed that when gap was in front of Mario, DS jumped and went back instead of trying to overcome it. This makes sense as Player 6 hesitates and dies often when a gap is in front, compared to Player 3. It appeared that results imitating specific humans are worst when those present a lot of hesitation. This factor is related to the experience the player has in playing games (table **??**). Player 10 is a clear example of player who hesitates a lot, hence dying in early stages in the level. This produces as outcome a non human-like behaviour as it is difficult to adjust the controller to the traces left by that player (figures 7(f)

and fig:P10NE:e).



(a) Player No. 3 - DS mechanism

(b) Player No. 3 - NE mechanism

(c) Player No. 6 - DS mechanism

(d) Player No. 6 - NE mechanism

(e) Player No. 10 - DS mechanism

(f) Player No. 10 - NE mechanism

Figure 7: Gameplay traces from three human players (Player No. 3, 6 and 10) and traces generated by the NE and DS mechanisms in their attempt to imitate those players.

6. Phenomenological evaluation of controllers

To assess the believability of the different controllers implemented we used an online survey, where participants annotated the believability of various

Mario AI (and human) agents, thereby implementing a 3rd person assessment, crowd-sourced version of the Turing test [6].

The Turing Test aims for the evaluation of human-like behaviour in computers by trying to deceive a human interrogator. If we extrapolate this idea into the believability assessment of the obtained controllers, it is possible to design a survey where the user (interrogator) can guess if the player visualized is indeed an AI agent or a human. Therefore it is possible to rank the believability of the different controllers and also obtain the number of times one controller was chosen when compared against the others. Six pairs of videos — showing different Mario agents playing the game — are displayed to the user; after watching each video pair the user is asked to choose which of the two agents was the human player using a 2-alternative forced choice questionnaire. The sequence of questions and the agents compared are generated randomly every time a user enters the survey; moreover, the same controller is not compared against itself and each controller appears twice in each of the six pairs presented. The results of this crowd-sourcing experiment[3] involving 67 completed surveys (total number of pairs watched is 67*6 = 402 pairs) are shown in Table 5.

The table shows the total number of points (Total column) that each of the controllers obtained and how many times each controller got selected compared to the other controllers (Human, BP, NE, DS, REALM and GEBT columns) whenever they appeared together in the same question. It can be seen that the human player was the one that was selected most often, followed in descending order by NE, DS, REALM, GEBT and BP. BP gets the worst result compared to the other controllers as it resembles the least human of all.

The *Mann-Whitney U Test* (p equals 0.05) is applied to the crowd-sourcing results of Table 5, in order to get the significance between each pair of controllers. In order to apply this test, the ranks are calculated as zeros and ones, i.e when comparing NE with BP, NE was selected 21 times and BP one, then NE gets 21 ones and one zero value, and the same applies for the rest of controllers compared to NE. The results of applying this test are presented in Table 6.

As expected, the human player is significantly different from all AI controllers. In the comparison among AI agents BP and NE are, respectively,

[3]Believability Survey website: www.juop.es.

Agents	Human	BP	NE	DS	REALM	GEBT	**Total**
Human	–	21	20	24	26	19	110
BP	1	–	1	6	7	6	21
NE	9	20	–	14	19	27	89
DS	8	24	6	–	12	24	74
REALM	4	24	12	9	–	14	63
GEBT	2	25	6	6	6	–	45

Table 5: Total number of "humanness" assessments obtained from the crowd-sourcing experiment. Results obtained from the Mann-Whitney U Test show that the human player is significantly different from the rest while BP and NE are the least and most human-like agents respectively. Mann-Whitney U Test calculation details are discussed in section 6.

Agents	Human	BP	NE	DS	REALM	GEBT
Human	–	**-9.39**	**-2.21**	**4.75**	**3.8**	**6.44**
BP	–	–	**-6.97**	**-5.39**	**-4.43**	**-2.74**
NE	–	–	–	1.58	2.53	**4.22**
DS	–	–	–	–	0.95	**2.64**
REALM	–	–	–	–	–	1.69

Table 6: The statistical values obtained from applying the *Mann-Whitney U Test* on each pair of controllers. The statistically significant differences ($p-value < 0.01$) are presented in bold.

the least and most human-like AI agents compared to the other AI controllers; however, there is not a significant difference between NE and DS. The humanness of DS is ranked significantly higher than the humanness of BP and GEBT while REALM and GEBT are only more human-like than BP; however, compared to each other they do not present a significant difference. From these results we can conclude that the NE agent is the most human-like Mario controller, getting higher humanness scores when compared to the REALM agent which has been ranked as the most human-like agent in another 3rd person believability experiment existent in the literature [2] that involved 100 participants at the Turing Test track of the 2010 Mario AI Championship.

In addition to the forced choice question users had the possibility of sending their additional comments about their judgement in a free text box at the end of the experiment. The main conclusions derived from those comments are as follows:

- It is difficult to decide who is the human without having a human player reference; for instance, a controller that moves towards a wall could be a 3 year old child playing Super Mario Bros.
- The human player was classified correctly mainly due to: 1) pausing the gameplay after risky manoeuvres; 2) trying to go to the right when the player run over a cliff; 3) hesitating at dangerous places; and 4) making little errors due to miss-pressed buttons (e.g. ducking for half second).

7. Discussion

There were a number of encouraging outcomes of the present study. In particular, all of the agents that were trained to imitate human players, achieved higher imitation scores than those that were simply created in order to gain a high score in the game. Those two agents that implemented indirect imitation (neuroevolution and dynamic scripting) also managed to convince human observers of their human-likeness significantly more often than the other agents. (The backpropagation-trained agent, which was the only one implementing direct imitation, attained dismal human-likeness scores in the phenomenological evaluation.) These results largely confirm the trends observed in [9] and [14], though in a different domain (and with more methodological rigour).

The two indirect imitation methods also outperformed the direct imitation method in terms of raw score, with neuroevolution almost always outperforming backpropagation, and dynamic scripting outperforming backpropagation by a large margin. However, all the trained controllers performed much worse than the players whose data they had been trained on. It seems that our use of indirect imitation has not completely solved the problem with performance drop between original human and imitation that is so apparent when using direct imitation. As the two more sophisticated non-imitating controllers (REALM and GEBT) are capable of clearing almost all levels of the difficulty used in these experiments, we can truthfully say that with current techniques, there is a tradeoff between performance and human-likeness. This is a serious problem for most of the envisioned uses of behaviour imitation listed in the introduction, and we would argue that this problem deserves further study. One possible angle of attack could be to combine the multiobjective approach from [14] with the more sophisticated evaluation framework used in this study.

We have not investigated to what extent a model trained on player X replicates the playing style of player Y, or indeed the average playing style of a population of players. These would be interesting investigations to which the current methodology would be well suited, but which we will have to defer until a future paper.

A key question is to what extent the methods and finding from this study apply to other domains than platform games, in particular to other genres of video games. We have discussed the general problem of behaviour imitation in games elsewhere, where we have also tried to delimit this problem against those aspects of believability which are more the domain of computer graphics [2]. In the following discussion, we will confine ourselves to the difficulty of imitating player behaviour in Super Mario Bros as compared to other games, for example racing games, where behaviour imitation has been studied previously (see the introduction).

The number of separate action outputs (5 buttons, yielding 32 possible actions each time step) is larger than in most racing games, and it can be argued that the navigation problem is more complex for platform games than in racing games as they occasionally require the player stop and wait (e.g. for a flower to recede into its pipe) or even move backwards (e.g. to collect a coin). There is also a larger number of qualitatively different actions that need to be taken by the player in a platform game (run, shoot, jump etc.) as compared to a racing game. On the other hand, the action space is discrete, which might be argued makes it easier to imitate actions. Previous attempts to evolve well-performing controllers for Super Mario Bros indicate that it is a slightly harder task than evolving controllers for racing games, suggesting that something similar goes for imitating human playing behaviour [30, 31].

Compared to some other games, Super Mario Bros could be said to be significantly simpler to imitate human playing behaviour in. *Unreal Tournament 2004*, which is the game used for the annual 2k BotPrize (a competition for human-like bots) is a first-person shooter game which features continues state and action space, movement in three dimensions, and multiple bots moving on the same map simultaneously. The complexity of movement in three dimensions together with the continuous action space implies that any function approximator would need to differentiate among a large number of meaningfully different actions, posing challenges from learning behaviour from a limited-size training set. In order to successfully use the methods presented here for such a game, one would probably need to define relevant and quite complex macro actions in order to limit the action space. Such ac-

tions could themselves perhaps be extracted from human playing data using clustering or frequent sequence mining.

There is a significant number of instances in the training data with the same observation but different actions, especially in situations where the player was standing still while hesitating. In other words, we have a significant incidence of *perceptual aliasing*. This is in principle a problem for all reactive controllers (all of the trainable controllers in this study were reactive), and might necessitate a move to stateful controllers such as recurrent neural neural nets, or evolutionary behaviour trees, in order to achieve higher imitation performance.

The crowd-sourcing method used for gathering the ground truth about the human-likeness of our controllers worked well for the current experiment, but for future experiments comparing a larger number of controllers and players of different skill levels, we will need to find new methods of gathering the opinions of a larger and perhaps more representative set of spectators.

8. Conclusion

We have presented a method for assessing the behavioural similarity of different agents playing a platform game to humans (or to other agents). Using this method, we have evaluated the similarity of a number of agents, some of which were trained on human gameplay data and some which were simple or sophisticated attempts at developing well-performing agents, to recorded data of ten human players. We found significant differences among the agents, with those agents that were trained on human data coming out as most human-like. None of the agents were judged to be as human-like as the human. Among the trained agents, those trained with indirect imitation performed much better than the agent trained with direct imitation, but did not perform as well on the game itself than those agents optimised only for playing well. These results might or might not generalise to other domains and other controller architectures. Returning to the research questions posed in section 1.2, we can answer them as follows:

1. *Can we create controllers that appear to be as human-like as actual humans to external observers?* Not yet. The methods proposed here appear promising, but there are apparently still aspects of the displayed behaviour that give the bots away as non-human, even for this relatively simple game.

2. *Are controllers trained to imitate humans perceived as more human-like than controllers simply trained to play the game well?* Yes, if the controller performs well enough. The controller trained with neuroevolution was judged the most humanlike of all controllers, whereas the controller trained with backpropagation was judged the *least* human-like, even though it was trained to imitate the same humans. The likely explanation is that the insufficient performance gave the controller away by making errors that a human would be very unlikely make, such as running in place against a wall.
3. *When imitating human behaviour, which controller architecture and training method (direct or indirect) gives the best results?* Indirect methods. The decisive victory of the neuroevolution-based and dynamic scripting-based controllers over all other controllers show that for the problem and method range considered, indirect methods solve the problem best.

8.1. Future work

We believe these results are promising and might contribute to the development of more human-like agents, for the purposes of more immersive gameplay, personalised game demonstration, and automatic content evaluation. Some recent work in automatically generating levels for Super Mario Bros already use artificial agents to automatically test the playability of generated levels [32, 33] and having more human-like controllers at their disposal could improve the quality of these content generator.

The most important line of future work would be to create more reliable and higher performing indirect imitation methods, preferably methods that use less data, so that models of the playing style of a particular player can be quickly learned. One way of improving indirect imitation would to be complement the trace-based human-likeness evaluation method used in this paper with other measures of human-likeness. For example, other metrics could be identified by identifying the main parameters of variation between numerous examples of human- and non-human gameplay using unsupervised learning methods such as deep networks of frequent subsequence mining; these parameters could then form the basis of new features used to model the particular playing style that should be imitated. Another way of achieving better indirect imitation could be to provide a library of common primitive or "molecular" in-game behaviours (stomping an enemy, collecting all coins

from a question mark block) and use these to compose "ensemble controllers" that match the observed playing style.

Ultimately, these techniques need to be verified on games that have higher dimensions of control and/or a larger number of common in-game atomic game mechanics. Where Super Mario Bros five binary dimensions of control yielding 32 possible atomic actions, a first-person shooter played with the XBox controller has six to eight binary dimensions of control (buttons), and four continuous dimensions (two control sticks). This yields a literally infinite number of possible atomic actions, but there are reasons to believe the number of frequently occurring actions is much lower. Unsupervised learning techniques could be very useful in analysing which actions to build into a repertoire of output classes for a controller based on indirect imitation.

References

[1] P. Hingston, A new design for a turing test for bots, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2010.

[2] J. Togelius, G. N. Yannakakis, N. Shaker, S. Karakovskiy, Assessing believability, in: P. Hingston (Ed.), Believable Bots, 2012.

[3] M. Newborn, Kasparov Vs. Deep Blue: Computer Chess Comes of Age, Springer, 1997.

[4] J. Togelius, S. Karakovskiy, R. Baumgarten, The 2009 mario ai competition, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2010.

[5] P. Hingston, A turing test for computer game bots, IEEE Trans. Comput. Intellig. and AI in Games 1 (3) (2009) 169–186.

[6] A. Turing, Computing machinery and intelligence, Mind 59 (1950) 433–460.

[7] S. Legg, M. Hutter, Universal Intelligence : A Definition of Machine Intelligence, Minds and Machines 17 (4) (2007) 391–444.

[8] T. Schaul, J. Togelius, J. Schmidhuber, Measuring intelligence through games (2011).

[9] J. Togelius, R. De Nardi, S. M. Lucas, Towards automatic personalised content creation in racing games, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG), 2007.

[10] B. Chaperot, C. Fyfe, Improving artificial intelligence in a motocross game, in: IEEE Symposium on Computational Intelligence and Games, 2006.

[11] J. Matthews, Interview with jeff hannan, http://www.generation5.org/content/2001/hannan.asp (2001).

[12] M. Buckland, Interview with jeff hannan (Publication date unknown). URL `http://www.ai-junkie.com/misc/hannan/hannan.html`

[13] R. Herbrich, (personal communication) (2006).

[14] N. van Hoorn, J. Togelius, D. Wierstra, J. Schmidhuber, Robust player imitation using multiobjective imitation, in: Proceedings of the Congress on Evolutionary Computation, 2009.

[15] C. Thurau, C. Bauckhage, G. Sagerer, Learning human-like Movement Behavior for Computer Games, in: S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, J.-A. Meyer (Eds.), From Animals to Animats 8: Proceedings of the 8^{th} International Conference on Simulation of Adaptive Behavior (SAB-04), The MIT Press, Santa Monica, LA, CA, 2004, pp. 315–323.

[16] B. Gorman, C. Thurau, C. Bauckhage, M. Humphrys, Believability testing and bayesian imitation in interactive computer games, in: Proceedings of the Conference on Simulation of Adaptive Behavior (SAB), 2006.

[17] S. Karakovskiy, J. Togelius, The mario ai benchmark and competitions, in: IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4, 2012, pp. 55 – 67.

[18] N. Shaker, G. N. Yannakakis, J. Togelius, Towards Automatic Personalized Content Generation for Platform Games, in: Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10), AAAI Press, Palo Alto, CA, 2010, pp. 63–68.

[19] C. Pedersen, J. Togelius, G. N. Yannakakis, Modeling Player Experience for Content Creation, IEEE Transactions on Computational Intelligence and AI in Games 2 (1) (2010) 54–67.

[20] P. A. Mawhorter, M. Mateas, Procedural level generation using occupancy-regulated extension, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2010, pp. 351–358.

[21] N. Sorenson, P. Pasquier, Towards a generic framework for automated video game level creation, in: Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications), Vol. 6024, Springer LNCS, 2010, pp. 130–139.

[22] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, R. Baumgarten, The 2010 Mario AI championship: Level generation track, IEEE Transactions on Computational Intelligence and Games.

[23] A. Champandard, The dark art of neural networks, AI Game Programming Wisdom (2002) 640–651.

[24] D. Floreano, P. Drr, C. Mattiuss, Neuroevolution: from architectures to learning, Evolutionary Intelligence In Evolutionary Intelligence 1 (2008) 47–62.

[25] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, E. Postma, Adaptive game ai with dynamic scripting, Machine Learning 63 (2006) 217–248.

[26] S. Bojarski, C. Congdon, Realm: A rule-based evolutionary computation agent that learns to play mario, in: Computational Intelligence and Games (CIG), 2010 IEEE Symposium, 2010, pp. 83–90.

[27] L. Bull, Learning classifier systems: A brief introduction, in: In Bull, L (Ed.): Applications of Learning Classifier Systems. Berlin u.a, Springer, 2004, p. 14.

[28] D. Perez, M. Nicolau, M. O'Neill, A. Brabazon, Evolving behavior trees for the mario ai competition using grammatical evolution, in: Applications of Evolutionary Computing, Springer, 2011, pp. 121–130.

[29] A. Champandard, Understanding behavior trees, AiGameDev.com.

[30] J. Togelius, S. Karakovskiy, J. Koutník, J. Schmidhuber, Super mario evolution, in: Proceedings of the 5th international conference on Computational Intelligence and Games, CIG'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 156–161.
URL http://dl.acm.org/citation.cfm?id=1719293.1719326

[31] J. Togelius, S. M. Lucas, Evolving controllers for simulated car racing, in: Proceedings of the Congress on Evolutionary Computation, 2005.

[32] N. a. Shaker, in: Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2012.

[33] M. Kerssemakers, J. Tuxen, J. Togelius, G. Yannakakis, A procedural procedural level generator generator, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2012.