Breeding a Diversity of Super Mario Behaviors Through Interactive Evolution

Patrikk D. Sørensen, Jeppeh M. Olsen and Sebastian Risi
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
{pdyr, jmao, sebr}@itu.dk

Abstract—**Creating controllers for NPCs in video games is traditionally a challenging and time consuming task. While automated learning methods such as *neuroevolution* (i.e. evolving artificial neural networks) have shown promise in this context, they often still require carefully designed fitness functions. In this paper, we show how casual users can create controllers for *Super Mario Bros.* through an *interactive evolutionary computation* (IEC) approach, without prior domain or programming knowledge. By iteratively selecting Super Mario behaviors from a set of candidates, users are able to guide evolution towards behaviors they prefer. The result of a user test show that the participants are able to evolve controllers with very diverse behaviors, which would be difficult through automated approaches. Additionally, the user-evolved controllers perform as well as controllers evolved with a traditional fitness-based approach in terms of distance traveled. The results suggest that IEC is a viable alternative in designing diverse controllers for video games that could be extended to other games in the future.**

I. Introduction

In recent years it has become more and more popular for video games to enable users to create and share game content. The examples are many and include Sony's puzzle game Little Big Planet[1], Nadeos racing game TrackMania that allows users to build race tracks[2] and the recent Super Mario Maker, in which users can build their own Mario levels[3]. The user-created content most often comes in the form of levels; very few games let the user create or modify the behavior or underlying structure of the Non-Player-Characters (NPCs). Usually, the NPC behaviors are constructed by programmers and function in predetermined and static ways.

In this paper we show that casual users are able to create sophisticated behaviors for the *Super Mario Bros.* video game by using a simple interface to evaluate several candidate behaviors that is reiterated upon. The approach is based on *interactive evolutionary computation* (IEC) [19] and requires no prior knowledge of neither AI methods nor programming. The developed IEC framework presents users with a selection of GIFs that show short level playthroughs, from which they can then choose the parents of the next generation. Human-in-the-loop approaches, such as IEC, have shown promise in a variety of different domains [9, 19, 21], because they allow human intuition to guide the search process.

[1]http://littlebigplanet.playstation.com
[2]http://en.wikipedia.org/wiki/TrackMania
[3]http://en.wikipedia.org/wiki/Super_Mario_Maker

The IEC results in this paper are compared to an automated fitness-based search. Interestingly, users were able to not only interactively evolve a variety of interesting and unique behaviors but also behaviors that perform competitively in comparison to the automated search. Additionally, and potentially more important, users reported that they (1) had fun while evolving Mario behaviors, and (2) felt that they had an impact on evolution. These results indicate that IEC could be a viable and entertaining approach to empowering players to create their own NPC behaviors.

This paper is organized as follows. Section II reviews the evolutionary computation methods this project builds upon, followed by a brief description of the Super Mario domain and game mechanics in Section III. In Section IV the domain representation, methods and core algorithms are described in detail. In Section V we give a description of our experiments, followed by the results in Section VI. Section VII highlights several pitfalls of the current approach and discusses future work that could improve it.

II. Background

This section gives a brief introduction to neuroevolution (NE) in general and the *NeuroEvolution of Augmenting Topologies algorithm* (NEAT), which evolves the Mario controllers in this paper. Lastly, we review several examples in which NE has been combined with IEC.

A. The Mario Framework

The Mario framework has been used extensively for various AI related research projects and gameplay competitions[4]. Projects range from imitating player behavior [10], evolving behavior trees through grammatical evolution [12], or creating content based on player experience [11], to Mario controllers optimized through reinforcement learning [20].

B. Neuroevolution

Neuroevolution (NE), the artificial evolution of artificial neural networks (ANNs), takes inspiration from the process that created our biological nervous system and has shown

[4]For more info see: http://www.marioai.org/gameplay-track

promise in a variety of different domains [3], and video games [13]. Typically, a population of neural networks controlling an agent is tested in a given environment, in which those individuals with higher scores – based on some defined fitness criteria – will have a higher chance of being recombined and/or mutated to produce the next generation of solution candidates.

In this paper, the neural networks controlling Mario are evolved with the NEAT algorithm [17]. NEAT begins with a population of simple neural networks and then *adds complexity* over generations by adding new nodes and connections through mutations. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. For a complete overview of NEAT see Stanley and Miikkulainen [17]. Most importantly, such complexification, which resembles how genes are added over the course of natural evolution, allows NEAT to establish high-level features early in evolution and then later elaborate on them.

C. *Interactive Evolutionary Computation*

In *interactive evolutionary computation* (IEC), users make aesthetic decisions by rating individual candidates, thereby deciding which individuals breed and which ones die instead of relying on fitness functions designed by developers [19]. Several examples of combining NE and IEC exist, such as users evolving 2D pictures in Picbreeder [16], 3D models through EndlessForms [2] or sound timbres [6].

The combination of IEC and NE has also allowed the creation of new types of games [13]. In the Galactic Arms Race video game [4], weapons are evolving based on how frequently they are shot by the players, and in the social game Petalz [14, 15], players can breed their own unique flowers. Other examples include the NERO game, in which players breed an army of robots in order to fight a team evolved by other players [18]. Each robot in NERO is controlled by a neural network evolved by the NEAT algorithm. By designing different training exercises players can train their robots to behave in specific ways when encountering various challenges on the battlefield.

Recently, a video from Seth Bling showing the evolution of a neural network for Super Mario World, has gathered over two million YouTube hits[5]. The popularity of this work shows the potential of using video games to reach a broader audience. However, so far, none of the aforementioned approaches allow players to design NPCs in a totally user-driven process, which the approach in this paper will try to attempt.

III. Super Mario Game mechanics

This section describes the game environment and the possible actions the Mario controlling ANN can perform. Super Mario is a world-famous game franchise, containing several games both in 2D and 3D. The game used in this paper is a modified version of the Infinite Super Mario Bros., originally created by Markus Persson and modified by Julian Togelius and Sergey Karakovskiy for the Mario AI competition[6]. The Mario framework features many of the same enemies and types of levels as the original games, though the specific terrains are randomly generated.

A. *Level Terrain*

In each level the avatar Mario has to overcome different challenges with the goal to ultimately reach his princess at the end of the game. To successfully navigate the levels, Mario needs to be able to notice changes in the terrain such as holes that he could fall into or obstacles. Several types of blocks, which can be interacted with, are scattered throughout the level, some of which will yield either coins or power-ups. Likewise, coins are dispersed and can be collected.

B. *Enemy Types*

Mario is also facing a multitude of enemies. Most of them are either walking on the ground, jumping around or flying in the air. The majority can be killed by jumping on them, but some have spikes on their backs that prevents this kind of attack. All of them can be shot by fireballs as well. More uncommon are missiles shot from canons and flowers that continuously appear from and disappear into green tubes. The former can be destroyed by jumping on them, while the latter can only be shot by a fireball.

C. *Mario States*

The character of Mario can exist in three different states: small, big and big with fire-shooting ability. Every time Mario is hurt by an enemy he regresses to a lower state, and dies if he is hurt in the small state. The previously mentioned power-ups can advance his state to a higher level.

D. *Actions*

In the original game, players can control Mario through the Nintendo controller, with buttons right, left, down, up, A and B. Mario is able to move left or right, at a normal or fast speed, crouch, jump in the air and, if he is in the right state, shoot fireballs. In the Infinite Super Mario Bros. version the AI controlled Mario's have access to the same controls.

IV. Methods and representation

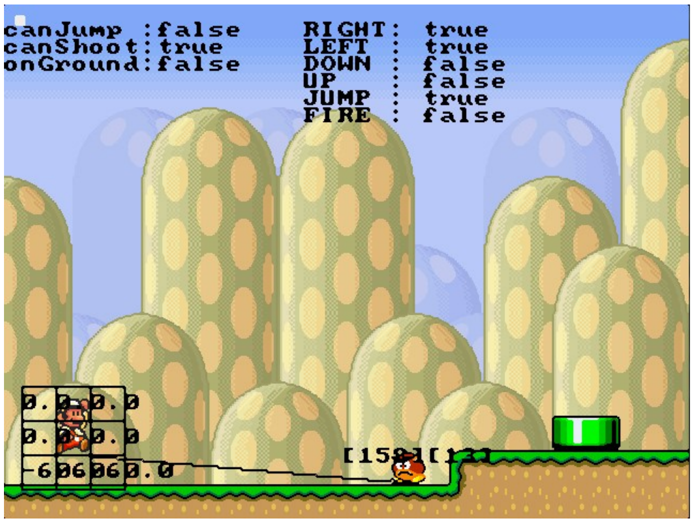
This section details the implemented algorithms, neural network setup and IEC interface.

A. *Neural Network Setup*

The ANN controlling Mario in this paper, receives a 3×3 grid of cells centered around Mario as input, in which different cell values represent different terrain types (Figure 1). In detail, the specific values are as follows: 1.0 = ground, 0.2 = coin, 0.0 = unreachable ground or air, -0.2 = question mark box, -0.5 = breakable standard box, -0.75 = green tube, and finally -1.0 for a hill piece.

[5]https://www.youtube.com/watch?v=qv6UVOQ0F44

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



```
canJump  :false      RIGHT:  true
canShoot:true        LEFT :  true
onGround:false       DOWN :  false
                     UP   :  false
                     JUMP :  true
                     FIRE :  false
```

Figure 1. **Mario Representation.** The controlling ANN receives a 3×3 grid as input together with information about the distance and angles to enemies, and conditional domain variables $canJump$ and $onGround$. The ANN outputs (shown at the top-right corner) determine the action that Mario performs each tick.

Additionally, the ANN receives the distance and angle to the two nearest enemies relative to Mario; the enemy type is currently not provided to the network. The values for both angle and distance are normalized in the range $[-1, 1]$. The domain conditional inputs *canJump* and *onGround* are represented as either -1.0 (false) or 1.0 (true).

The ANN has six outputs: right, left, up, down, jump, and fire. If the output value for an action is higher than a threshold of 0.5, the particular button is pressed. Mario can perform multiple actions at the same time (e.g. shooting and jumping), except to run left and right. In case both left and right are chosen by the ANN, the one with the highest activation is performed; if both have the same value Mario will not move.

As the Mario framework implementation is written in JAVA, the framework in this paper is build on the NEAT and Java Genetic Algorithms Package (JGAP)[7] based framework *Another NEAT Java Implementation* (ANJI)[8].

B. The IEC Interface

The motivation for the IEC approach in this paper is the potential to enable casual users to create Mario controller without any technical skills. Figure 2 shows the developed IEC interface, which aims to accomplish this goal. While the user is watching, a total of nine controllers are playing – one after the other – through a small part of a Mario level. The number of presented controllers tries to strike a balance between giving the user enough variety to choose from while still allowing a reasonable quick evaluation of all behaviors shown.

During the playthroughs, GIFs are recorded for each of the different controllers that show Mario in action. Once the whole population has been played and recorded, a window with all nine recorded GIFs is shown to the user (Figure 2). The user is then able to evaluate and compare each individual in the population and select one preferred controller by simply clicking on the particular GIF. Based on the user's selection, the next generation of controllers is created through mutating the selected individual and the process starts again. That way users can guide the evolutionary search towards Mario behaviors they find interesting.

[7]http://jgap.sourceforge.net

[8]http://anji.sourceforge.net/

Table I
EVOLUTIONARY APPROACHES.

Approach	Evaluation
Fitness-based Controller	Fitness function awarding cells passed
IEC Free Play Controller	User without any specific goal
IEC Competition Controller	User with the goal to pass as many cells as possible

V. EXPERIMENTS

To test whether users can evolve Mario behaviors, a user study was performed on site at the IT University of Copenhagen with a total of 20 participants. Each participant was asked to evolve controllers for 20 generations through the interface presented in Section IV. Additionally, we divided the participants in groups of ten and gave them different instructions in order to determine under what conditions certain behaviors evolve. The first ten participants were given no constraints as what to create, and were encouraged to evolve whichever behaviors they preferred. The other ten were told to evolve controllers that could travel as far as possible in the level. The second user study was set-up as a contest with a small prize for the best performing controller, to give some additional incentive for the players to do their best.

After the users evolved Mario controller for 20 generations, they were asked to answer two questions on a scale from 1–6 (where 6 is best). The questions read as follows:

1) *How much impact do you feel that your choices had on the evolutionary process?*
2) *How much fun was it to develop your AI this way?*

Controllers evolved through IEC were compared to controllers evolved with a traditional automated fitness-based approach. The fitness for the automated approach was the number of cells passed at the end of the simulation. The simulation was terminated if Mario reached the end of the level, he died or the time limit was reached. Additionally, to create more robust controllers, the starting position of the avatar was moved every four generations to a different location in the same level, for both the automated and IEC approach (Figure 3). Table I shows an overview of the three approaches.

A. Experimental Setup and Parameters

The duration of GIFs shown to the users is initially set to 2,1 seconds, but increases gradually each generation, with a maximum of 4 seconds in the final generation. As it is often

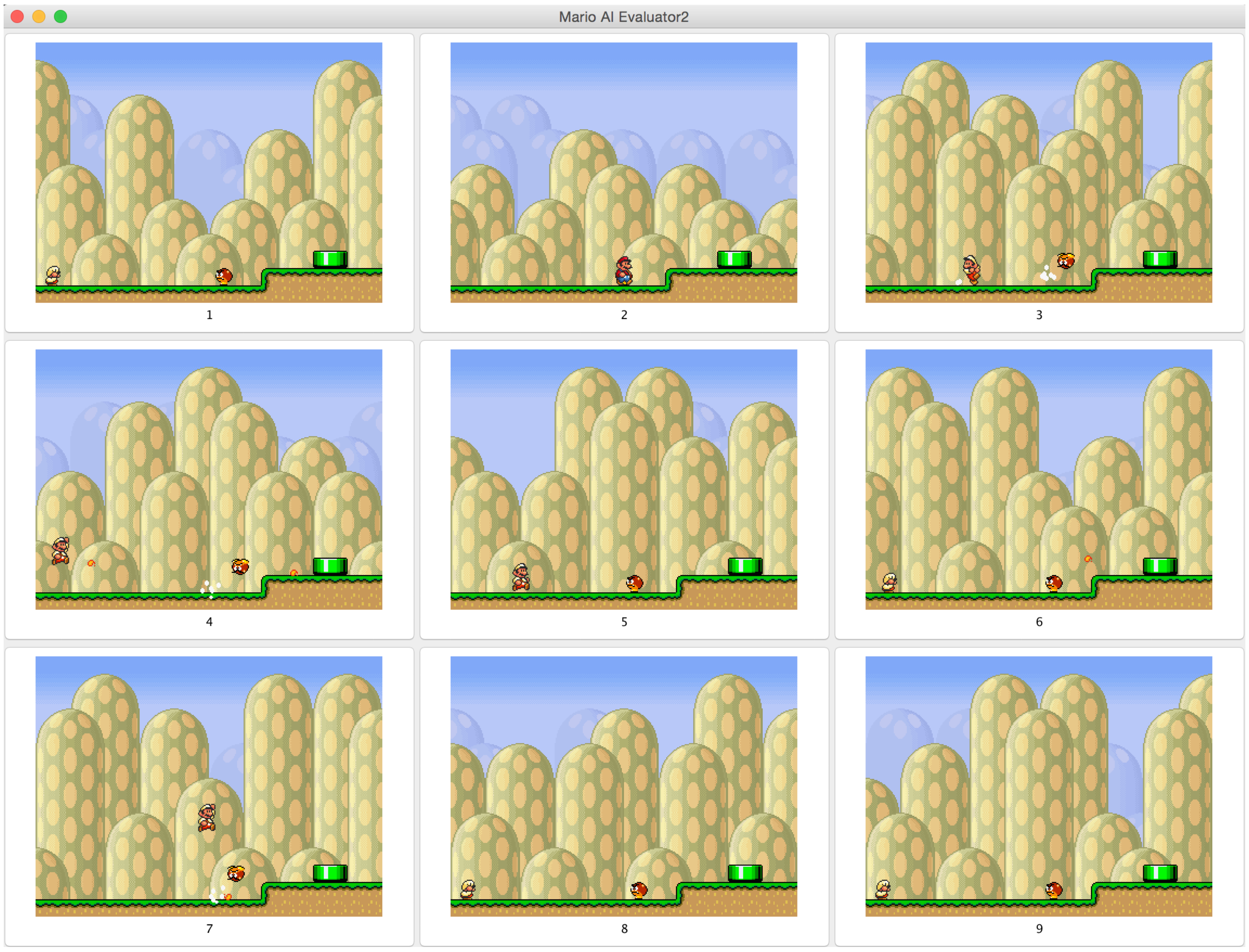


Figure 2. **The IEC User Interface.** The user is presented with nine small playthroughs recorded as GIFs, from which the parent for the next generation can be chosen. The IEC approach offers casual users the ability to breed Mario controllers without requiring technical skills.

easier to eliminate inferior behaviors at earlier generations, the duration was limited in the beginning to reduce user fatigue.

The population size was set to nine for both the automated and IEC approach. The number of generations was set to 20. Offspring had a weight mutation chance of 0.55, 0.01 chance of node addition, and 0.01 chance of link addition.

VI. Results

The IEC results are based on the ten participants of each experiment and the fitness-based results are collected from ten independent evolutionary runs. Figure 4 depicts a general trend for all approaches to improve over generations. In particular, there is a significant increase in performance for all methods comparing first and last generations ($p < 0.05$; Student's t-test). The pair-wise differences between the approaches are on the other hand not significantly different, which indicates that both automated and IEC approaches (free play and competitive) are able to evolve similar performing Mario controllers.

Not surprisingly, for all three approaches there are drops in performance when Mario's starting position is moved in generations 5, 9, 13 and 17. Not only does the level layout change at a new starting position, which might break less general controllers, later parts of the level often contain new object types (e.g. a question mark box) that the ANN controller first has to learn to respond to.

Additionally, the results show that the standard deviation for cells passed for the user-evolved controllers is often higher than for the automated approach. This difference indicates that a fitness-based approach tends to create more uniform controllers, while there is more variety in the type of behaviors evolved through IEC, which we examine next.

A. *IEC Evolved Behavior Examples*

Indeed, the participants in the user tests were able to evolve controllers displaying a variety of different behaviors. While most users would first focus on creating controllers capable of jumping and moving to the right (a strategy often also

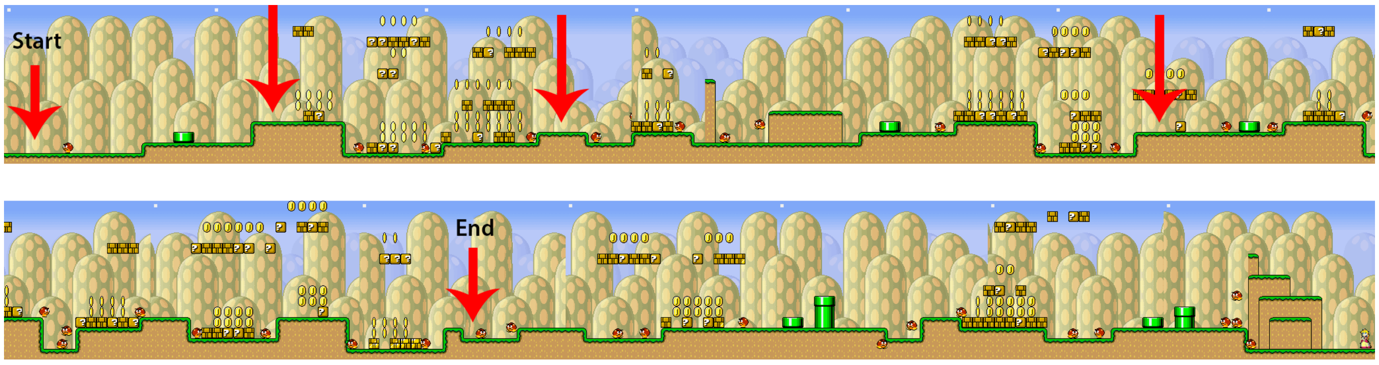


Figure 3. **Training Starting Positions.** To encourage the evolution of general behaviors, the starting position of Mario is moved every four generations to the next pre-defined position in the same level.

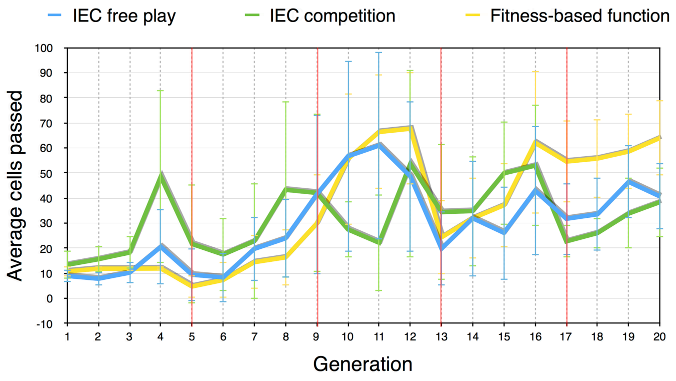


Figure 4. **Training Performance.** Shown are average cells passed during training over generations. The vertical red lines indicate the generations when Mario is set to a new position in the level.

discovered by the automated fitness-based search), some tried to evolve more specific and unique behavior.

Examples include behaviors that would try to collect as many coins as possible or rush through the level without paying attention to enemies or coins. Other controllers would act more cautiously, standing still and ducking when encountering enemies, and would only move once the enemies were behind Mario to his left.

Figure 5 shows a unique IEC created Mario behavior called *living-on-the-edge*. Mario would first try to run into enemies when in his normal state, and then try to keep a specific distance to the enemy in the small Mario state (see Section III-C for details on the different Mario states).

More aggressive Mario controllers such as *im-gonna-kill-everything-that-moves-mario*, which tries to jump on every encountered enemy while shooting fireballs in all directions, was also evolved in the IEC free play session. Other participants would try to evolve slightly more pacifistic Mario behaviors that used enemies as pads to jump higher, progressing more easily through difficult obstacles in the level (e.g. a high cliff). The reader is encouraged to take a look at the video accompanying this paper (available at https://goo.gl/Ell82m), to get a better sense of the types of controllers that were evolved and the IEC system in action.

Table II
QUESTIONNAIRE RESULTS.

Questions	Average Rating	Standard Deviation
Q1: Impact on evolution	4.29	0.8
Q2: Amount of fun when evolving	4.88	1.0

B. Questionnaire Results

The results from the questionnaire are shown in Table II. In general, the participants felt that they had significant impact on evolution with an average rating of 4.29 out of 6. However, there were no participants who felt that they had complete control, which could be due to the fact that (1) Mario's starting position was moved every fourth generation leading to drops in performance and unexpected behaviors, or that (2) IEC is simply a stochastic process.

Maybe slightly surprising, over 25% gave the maximum score when asked about the level of fun they had breeding Mario controllers, with a score of 4.88 on average. The user responses indicate that IEC-based interfaces have potential for other video games as well, which can benefit from casual users evaluating and evolving NPC behaviors. Similarly to games like NERO [18] and EvoCommander [5], the results in this paper suggest that the process of evolving NPC behaviors can be a novel and entertaining game mechanic.

C. Generalization Test

NPC controllers employed in video games should be able to deal with variations in their environment and ideally generalize to situations they might not have encountered during training. To test the generality of the evolved behaviors in this paper, the selected IEC and fitness-based controllers from each generation are tested in ten variations of the level they were trained in. In each variation the level layout, amount and location of enemies, enemy types, tubes, coins and breakable boxes is changed randomly.

Figure 6 shows the generalization performance of the three approaches. IEC free play reaches the highest generalization performance, however, the pair-wise differences between the

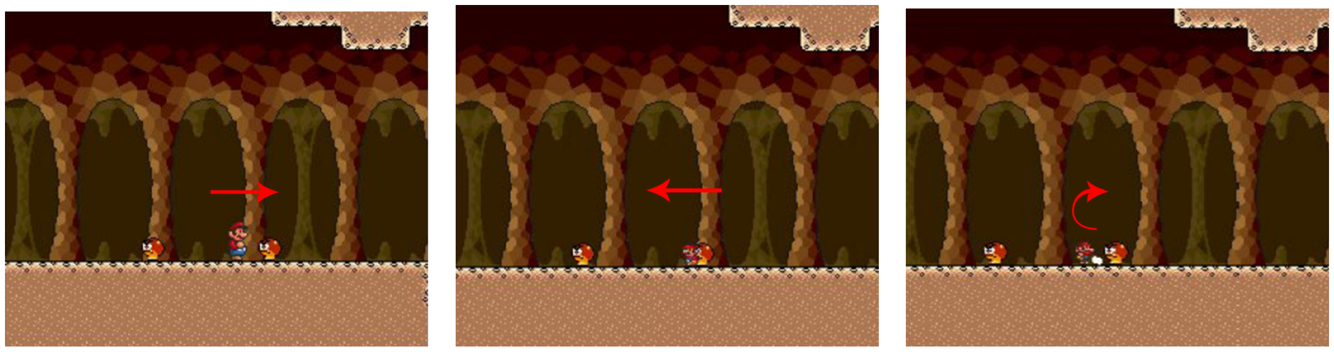

(a) (b) (c)

Figure 5. **Storyboard of the “living-on-the-edge” Mario controller.** The controller runs into an enemy (a) and then after converting to the smallest Mario state (b), walks as close to the enemies as possible without touching it (c).

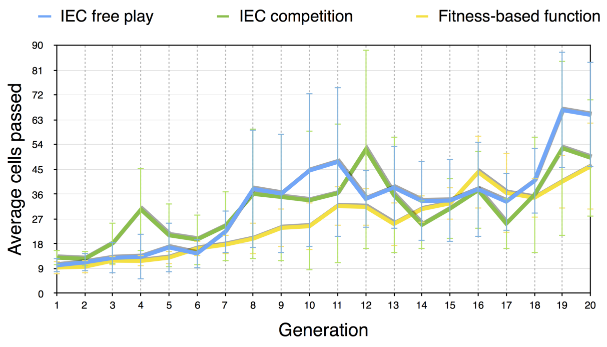


Figure 6. **Generalization Performance.** The controllers selected during training in each generation are tested on their ability to perform in levels they have not seen during training. Each controller is evaluated on ten different level layouts.

approaches are not significant. Similarly to the training performance, all approaches have a statistically higher generalization performance when comparing the first to the last generation ($p < 0.05$). The performance of the controllers on the generalization test is mostly comparable to the performance in training, indicating that the evolved behaviors are able to deal with some variation in the layout of levels.

Overall the results suggest that casual users are able to evolve a variety of unique behaviors that often also perform on par to controllers evolved with a fitness-based approach. It is important to note that the main goal of this comparison is not to determine which method produces the highest performing controllers, but rather to establish a base-line that the IEC-based approaches can be compared to. Inevitably, some will argue that by giving the fitness-based approach more generations it would eventually outperform the IEC approach. While this is likely true, the main advantage of the IEC approach is that even in a game like Mario (i.e. a game whose main goal it is to just advance to the right), players can guide evolution towards unique and interesting behaviors that would not have been rewarded by a naive fitness-based approach (see Section VI-A for examples).

VII. Discussion and Future Work

This section discusses the strengths and weaknesses of the presented approach to create Mario controllers. We also try to articulate suggestions for improvements and draw perspectives to other projects and methods.

A. *Representation*

There exist a variety of different ways to represent the Mario world to an ANN-based controller, which differ in terms of the number of ANN inputs and level of sensory abstraction. The representation chosen in this paper tries to strike a balance between selecting the inputs that the controller needs in order to perceive its world, and keeping their number as small as possible; each additional input can lead to an increase in the evolutionary search space and greater demands on the information processing capabilities of the network.

The number of inputs for each object type (e.g. terrain grid size, number enemies) can also have a significant influence on the behavior of the agent. Representing too many enemies and having too small a grid, means the network will initially be highly sensitive to nearby enemies and less sensitive to the terrain. While the ANN can be trained to compensate for this or its inputs can be scaled, it will require additional training time or domain-dependent manual tweaking.

B. *User Fatigue*

As IEC can be a time consuming process, user fatigue can be an issue. To produce a desired and usable result the process of iteratively selecting behaviors can take many hours. In the current system it takes approximately 60 seconds per generation, both to record the GIFs and to choose among them. The current implementation is not optimal as users have to wait for the recording of each GIF, and even though they can watch the controllers perform during this process, it increases the evaluation time significantly.

Initially, we experimented with combining IEC with an automated fitness-based search in between each IEC step to speed up the evolutionary process. The automated search would run for a few generations, rewarding Mario for moving towards the right. However, this addition sometimes introduced an unwanted bias, leading evolution away from the direction sought by the user (e.g. a Mario controller that moves left).

User fatigue can also be negatively influenced if the controllers presented to the user are all very similar; unvaried controllers might not be very meaningful for the user to choose between. As proposed by Woolley and Stanley [21], a potential solution to this problem could be *novelty-assisted interactive evolution* (NA-IEC). By combining IEC with *novelty search* [7], a divergent evolutionary search method, the users would only choose from candidate solutions of *novel* behaviors, thereby accelerating the evolutionary search. Additionally, to gain insights into the quality of user-evolved versus automation-evolved *behaviors*, the IEC approach should be compared to a novelty search based approach. Another future extension to accelerate the IEC approach could be a rank-based IEC approach that was introduced by Liapis et al. [8] for content generation. The rank-based approach has shown advantages over the standard IEC approach with respect to speed of convergence.

C. User Evaluation

A potential pitfall to the human evaluation step is the relative short duration of the GIF; the user only gets a small glimpse of Mario's behavior. Currently, the duration of the GIF is a compromise between how long the user should have to wait for the recording and how much gameplay the user needs to see in order to evaluate the controller properly. A solution could be to automatically learn a model of the user [1] and only show a few longer playthroughs to the user that the learned user model is uncertain about.

In the initial generations the ANNs often perform seemingly random behaviors. Instead of starting from random controllers, users could instead build upon the work of others, similar to how users collaborate in Picbreeder [16].

VIII. Conclusion

The presented approach allows users, for the first time, to interactively evolve behaviors for Super Mario. The results show that controllers evolved with IEC perform similarly well compared to a fitness-based search in terms of distance traveled, but importantly display more varied strategies and behaviors. Moreover, the IEC users reported that they had fun while evaluating and evolving controllers. In the future, this system could be extended to other video games and to allow many users to evolve behaviors collaboratively online.

References

[1] J. C. Bongard and G. S. Hornby. Combining fitness-based search and user modeling in evolutionary robotics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 159–166. ACM, 2013.

[2] J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pages 144–148, 2011.

[3] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[4] E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 241–248. IEEE, 2009.

[5] D. Jallov, S. Risi, and J. Togelius. EvoCommander: A novel game based on evolving and switching between artificial brains. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2016.

[6] B. Jónsson, A. K. Hoover, and S. Risi. Interactively evolving compositional sound synthesis networks. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 321–328. ACM, 2015.

[7] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[8] A. Liapis, H. P. Martinez, J. Togelius, and G. N. Yannakakis. Adaptive game level creation through rank-based interactive evolution. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[9] M. Löwe and S. Risi. Accelerating the evolution of cognitive behaviors through human-computer collaboration. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2016.

[10] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[11] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):54–67, 2010.

[12] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon. Evolving behaviour trees for the Mario AI competition using grammatical evolution. In *Applications of evolutionary computation*, pages 123–132. Springer, 2011.

[13] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[14] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley. Combining search-based procedural content generation and social gaming in the Petalz video game. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012.

[15] S. Risi, J. Lehman, D. D'Ambrosio, R. Hall, and K. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2015.

[16] J. Secretan, N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM, 2008.

[17] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10 (2):99–127, 2002.

[18] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189, 2005.

[19] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275 – 1296, 2001.

[20] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, IEEE Symposium on*, pages 156–161. IEEE, 2009.

[21] B. G. Woolley and K. O. Stanley. A novel human-computer collaboration: combining novelty search with interactive evolution. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 233–240. ACM, 2014.