Interactive Design Exploration of Game Stages Using Adjustable Synthetic Testers

Hirotaka Suetake
The University of Tokyo
suecharo@g.ecc.u-tokyo.ac.jp

Tsukasa Fukusato
The University of Tokyo
tsukasafukusato@is.s.u-tokyo.ac.jp

Christian Arzate Cruz
The University of Tokyo
arzate.christian@ui.is.s.u-tokyo.ac.jp

Andy Nealen
The USC School of Cinematic Arts
anealen@cinema.usc.edu

Takeo Igarashi
The University of Tokyo
takeo@acm.org

ABSTRACT

Game designers take into account the wide range of play-styles and skill levels of players to create enjoyable experiences. One important step in the game design process involves playtests with professional testers; this process is time-consuming and expensive. Hence, there exist several methods to create synthetic testers to test a game automatically. However, one shortcoming is the lack of realistic-playing with different play-styles and skill levels. In this paper, we propose a game level authoring tool that incorporates synthetic testers, which enable the control of play-styles and skill levels. Furthermore, we utilize visualization techniques to help assess the difficulty level of each part of the stage. Our user studies confirmed that our tool was effective for designing game stages appropriate for a particular type of player.

CCS CONCEPTS

• **Applied computing** → **Computer games**; • **Computing methodologies** → **Game tree search**; *Artificial intelligence*; • **Human-centered computing** → *Graphical user interfaces.*

KEYWORDS

Interactive stage design and editing, Bot, Playtest

ACM Reference Format:
Hirotaka Suetake, Tsukasa Fukusato, Christian Arzate Cruz, Andy Nealen, and Takeo Igarashi. 2020. Interactive Design Exploration of Game Stages Using Adjustable Synthetic Testers. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3402942.3402982

1 INTRODUCTION

When designing game stages and tuning game parameters, game designers must consider the player archetypes of the expected audience of the game. In particular, understanding the common play-styles and skill levels required to complete the game is crucial for designing an enjoyable game for a wide base of player archetypes.

FDG '20, September 15–18, 2020, Bugibba, Malta

ACM ISBN 978-1-4503-8807-8/20/09…$15.00
https://doi.org/10.1145/3402942.3402982

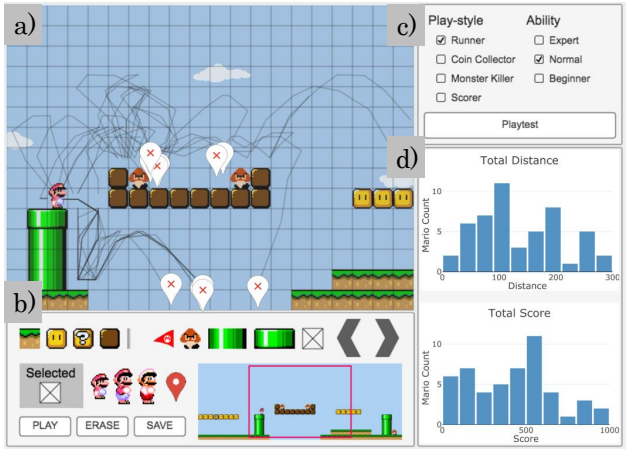

Figure 1: Screenshot of the proposed interface that identifies (a) the game view, (b) the stage design panel, (c) the bot setting panel, and (d) the visualization panel.

Many games include multiple levels of difficulty. For example, for expert players, a "hard" mode might involve a wider variety of hazards and stronger enemies. In contrast, for novice players, an "easy" mode should implement simpler challenges. Tuning the game features that influence the perceived level of difficulty is a time-consuming iterative task. In this paper, we address this problem by proposing a game level authoring tool, called *Mario Builder* (see Figure 1). This tool incorporates synthetic testers that help the game designer better understand which section of the game level contributes the most to the difficulty of *Super Mario Brothers.*

The use of bots as synthetic testers has been proposed in the past [Gaina et al. 2017; Gudmundsson et al. 2018; Holmgård et al. 2018; Isaksen et al. 2017; Khalifa et al. 2016; Nelson 2011]. They provide the advantages of reducing the bias from a human player's psychomotor skill improvement and facilitating the exploration of the impact of a narrower range of game parameters (e.g., character speed, width, and height). Nevertheless, synthetic testers can not be a substitute for the actual players' playtest, but the current synthetic testers could be used earlier in the development cycle. One key element that is needed to create synthetic testers that can be of more use in later stages of the development – closer to human playtests – is the simulation of multiple play-styles and skill levels.

We incorporate play-styles and skill-levels into standard synthetic testers based on the Monte Carlo tree search (MCTS) algorithm [Holmgård et al. 2018; Jacobsen et al. 2014; Khalifa et al. 2016].

We found the best parameters to achieve the desired play-styles automatically using genetic algorithms (GA). We also adjust the depth of the search in the MCTS and added randomness to reproduce human-like operational and decision errors.

Furthermore, to enhance the user's understanding of what makes a game level difficult, *Mario Builder* includes visualization techniques that provide insight into which elements of the game contribute the most to the difficulty. In particular, we use histograms of the gameplay scores, similar to Isaksen et al. [2017], and heat maps to show the locations where the game character dies [Thompson 2019].

Finally, we evaluated the effectiveness of *Mario Builder* with a user study. Our results were positive, as most of the users preferred to use our synthetic testers. The user feedback was an important part of the process. It led to our conclusions for improving the interaction and visualization approaches for game designers, which may broaden the use of synthetic testers in the game design process.

2 RELATED WORK

2.1 AI-Assisted Game Design

Artificial intelligence (AI) techniques can support parts of the game design process [Gaina et al. 2017]. Isaksen et al. [2017] introduced a method to explore the high-dimensional game space (i.e., the relationship between game parameters and player experience) in minimal score-based action games like *Flappy Bird*, by using an overall playing score. First, they made a simpler player model with human motor skills that had several parameters, such as reaction speed and actions-per-second. Second, they predicted game difficulty using an exponential survival analysis of score histograms. While these systems are suitable for a simple game space, they require a fixed strategy (i.e., perfect motor skills not to lose) in advance. There exist various game play-styles for extracting some knowledge from games using playtest results [Nelson 2011]. So, it is essential to customize the bot properties in real-time for complex games, such as *Super Mario Brothers*.

Therefore, we adopted automatic playtesting but created a new function to customize the play-styles and skill levels.

2.2 Human-like AI Behavior in Games

Imitating human playing behavior has been thoroughly investigated. For example, Gudmundsson et al. [2018] propose a deep learning method to learn the actual player's play-styles from game-playing data. A problem with the deep-learning approach is that it requires large-scale training datasets because of the large number of parameters that need to be tuned. Then, researchers have compared several methods for generating character controllers that mimic the human-like playing, and they report that the MCTS algorithm is suitable for human-like behavior [Holmgård et al. 2018; Jacobsen et al. 2014; Khalifa et al. 2016]. Hence, we utilized the MCTS algorithm to customize bot properties.

2.3 Data Visualization in Games

Several techniques have been proposed for teaching users about the results of playtests. One approach is to utilize gameplay scores from multi-agents' playtests [Isaksen et al. 2017]. However, a player's score focuses only on predicting a game's difficulty, and it doesn't provide information to understand where to refine the game stages. The other approach is to use a heatmap, which is a color-coded representation based on event frequencies, such as locations where game characters often die [Thompson 2019]. These systems enable users to understand the results of the bot playtests visually. However, this representation is insufficient to support designing game stages because the designers must consider both the overall game difficulty and event frequencies. Therefore, we employed both playing scores and a heatmap for visualizing the quality of the games in development.

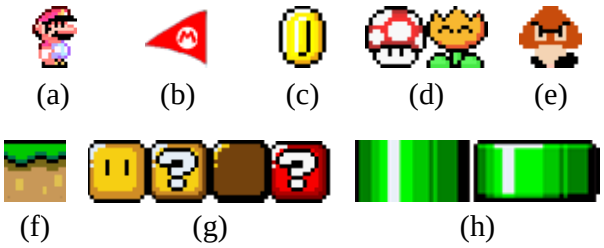

Figure 2: Stage elements; (a) bots start point, (b) goal point (flag pole), (c) coin, (d) power-up items (mushroom and fire flower), (e) enemy, (f) ground, (g) blocks, and (h) pipe.

3 MARIO BUILDER

The user interaction proceeds as follows. First, a game stage without any elements appears in the game view, see Figure 1(a). Then, the user starts by manually placing the stage elements, see Figure 1(b). Note that the users can play a stage in development by themselves. Our system allows the users to set the AI-bot properties (i.e., play-style and skill level) with the setting bot function, as shown in Figure 1(c). Next, click the *Playtest* button simulates an AI-based playtest, and the users can see the simulation results as an animation of Mario behavior in the game view and difficulties on the visualization panel, see Figure 1(d). Based on the visualization, users can efficiently design the game stages.

3.1 Stage Design Function

The user first sets a start point and an end goal (i.e., the flag pole at the end of each stage) on the screen, as shown in Figures 2(a) and 2(b). Next, the user adds ground and block elements (e.g., coins, blocks, pipes, and enemies) on the 2D stage design panel via trial and error, as shown Figures 2(c)-(h). When the user puts a new element over an existing element, the system removes the old element and inserts the new one. Note that our system assumes that Mario moves from the left side of the screen to the right side.

3.2 Bot Setting Function

By clicking checkboxes, see Figure 1(c), the user can freely change the play-styles and skill levels (from beginner to expert). In this paper, the play-styles are categorized into four simple types: (a) "Runner" to clear the games quickly, (b) "Coin Collector" to gather coin preferentially, (c) "Monster Killer" to defeat many enemies, and (d) "Scorer" to earn a total score preferentially (i.e., gathering coin and power-up items).

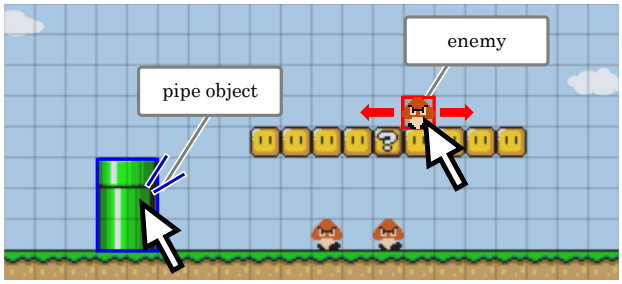

Figure 3: Stage Design Function. The game stage is covered by a grid, and the user places stage elements on the grid.

Table 1: Game metrics according to the bot play-style.

Play-style	Game Metric
Runner	(a) the total distance (d) the total score
Coin Collector	(a) the total distance (b) the number of collected coins
Monster Killer	(a) the total distance (c) the number of defeated enemies
Scorer	(a) the total distance (d) the total score

3.3 Playtest Function

When the users click on the *Playtest* button, our system simultaneously runs M synthetic testers with the user-set properties on the designing stage (in this paper, we set $M = 30$ by referring to the number of actual testers in the game development site). Note that each one of the testers acts individually and follows a different movement route. This playtest will continue until the characters of all the testers reach the end goal (e.g., the flag pole) or die. In our experience, it took about 10 seconds to complete the playtest.

3.4 Visualization Function

After the AI-based playtest, our system automatically presents (i) the travel trajectories and (ii) the locations where the synthetic testers die with "×" marks on the stage design panel.

Our system also produces a visual of the result statistics as a one-dimensional histogram, similar to Isaksen at al. [Isaksen et al. 2017]. However, there exist various play-styles in *Super Mario Brothers*, so it is necessary to change the displayed result statistics according to the bot play-styles. We empirically chose four metrics: (a) the total distance traveled on the stages, (b) the number of collected coins, (c) the number of defeated enemies, and (d) the total score, and visualize two of them according to the plan shown in Table 1.

4 SYNTHETIC TESTER MODEL

We use the MCTS algorithm, which is a heuristic search algorithm based on a randomized exploration of the search space instead of a static evaluation function, to make a synthetic tester model. This algorithm builds up a game tree in memory, and this accurately estimates the values of the most promising moves. In order to strike a balance between exploitation (better opportunities to win) and exploration (moves that lack simulation data), we adopted the upper confidence tree (UCT) as our tree policy.

Table 2: Stage characteristics from the current game state.

#	Characteristic	Description
1	canJump	whether Mario can jump or not
2	canShoot	whether Mario can shoot a fireball or not
3	nearCoinBox	whether coin boxes are near Mario or not
4	nearEnemy	whether enemies are near Mario or not
5	nearItemBox	whether item boxes are near Mario or not
6	nearStep	whether steps (i.e., differences in stage heights) are near Mario or not
7	allFalse	whether all the other values (i.e., 1 through 6) are false or not

4.1 Game State Vector

Actual players often change their playing rules in the current game state, so we designed the default policy and the reward function to reflect this in the current game state. First, we empirically chose seven characteristics which are expressed by binary values ($1 = true$, $0 = false$) from the current game states, and rewrote them in a vector form, treating the values as columns, called state vector $\vec{S} \in \mathbb{R}^7$ (see Table 2).

4.2 Default Policy

When playing *Super Mario Brothers*, the actual players use five buttons: right (R), left (L), dash (D), jump (J), and fireball (F), for controlling Mario's behaviors (11 actions). In the simulation from the MCTS algorithm, if these behaviors are selected at random per f frame (in this paper, we set $f = 4$), it may not be suitable for human-like playing. That is, it is necessary to add a default policy which prefers certain behaviors in the simulation process.

We first defined an action weight vector $\vec{aw}_i \in \mathbb{R}^7$, which is related to the above stage state from each button $i \in \{R, L, D, J, F\}$, and computed a ratio b_i where the bots select each button as follows:

$$b_i = \vec{aw}_i \cdot \vec{S} \tag{1}$$

Furthermore, we consider the condition of multi-button selection. We calculated the ratio of each action P_j that can be expressed by the product of each pressed button's weight b_i as follows:

$$P_j = \prod_i b_i \tag{2}$$

Based on the ratio of each action, the simulation can be performed in the MCTS algorithm. Note that we initially set $P_j = 1.0$.

4.3 Reward Function

We built a reward function based on the following six elements: (1) the coin score d_{coin} (i.e., the number of earned coins), (2) the kill score d_{kill} (i.e., the number of defeated enemies), (3) the power-up item score d_{power} (i.e., the number of earned power-up items), (4) the distance along the x-axis d_x, (5) the distance along the y-axis d_y, and (6) the total distance traveled on the stages d_{dist} based on the Manhattan distance. These were used to create the vector $\vec{d} = (d_{coin}, d_{kill}, d_{power}, d_x, d_y, d_{dist})$.

Next, to reflect the game state in the reward, we defined a reward weight vector $\vec{rw} \in \mathbb{R}^7$ that was related to the above game stage state from each element i and computed a ratio r_i that adds each

Table 3: Fitness functions F according to each play-style.

Play-style	Fitness Function F
Runner	1.0 * the total distance
Coin Collector	0.1 * the total distance + the coin score
Monster Killer	0.1 * the total distance + the kill score
Scorer	0.1 * the total distance + the total score

Table 4: Synthetic tester skill level statistics.

Skill Level	N	d	r
Expert	300	12	0.01
Normal	250	9	0.05
Beginner	200	6	0.10

reward element to the total reward as follows:

$$r_i = \vec{rw}_i \cdot \vec{S} \tag{3}$$

We rewrote them in vector form treating the values as columns, creating the vector $\vec{r} = (r_{coin}, r_{kill}, r_{power}, r_x, r_y, r_{dist})$. Finally, we computed a reward value R where

$$R = \vec{r} \cdot \vec{d} \tag{4}$$

After the simulation, the information in each node is updated by adding the reward value R to the total reward and increasing the number of visits from the simulated node to the root node.

4.4 Play-style Parameters

For controlling the bot's play-styles, we must set all values of the action weight vectors $\vec{aw}$ and the reward weight vectors $\vec{rw}$ appropriately. Then, we define fitting functions F for each play-style (see Table 3) and maximize them by using the GA.

4.5 Skill Level Parameters

We produced skill levels based on three parameters (1) the total selection number of the MCTS N, (2) the depth of the simulation of the MCTS d, and (3) the randomness to reproduce human-like operational and decision errors r, as shown in Table 4.

5 USER STUDY

We performed a user study to compare a fully-manual design (without an AI-based playtest) with our method to investigate the effectiveness of our interface in assisting users in understanding the characteristics of user-designed game stages qualitatively. We invited 12 participants, aged 23 to 35, who had an amateur level of knowledge of game designing/playing.

We asked them to keep working on their stages until they were satisfied with them in the following order: first without, then with the AI-based playtest. In the end, each participant was asked to answer two questions: (Q1) Which did you find more comfortable to use for design? (w/o or w/ the AI-based playtest) and (Q2) Score the overall usability of each method using a five-point Likert scale (1 = "*extremely dissatisfied*," and 5 = "*extremely satisfied*").

Table 5 shows the post-experiment questionnaire results with the mean values and standard deviations (SD). We calculated p-values by running a Wilcoxon signed-rank test. The result was $p = 3.38e-03$, which is significant at $p < 0.01$ for a two-tailed hypothesis. The participants' comments are summarized below:

Table 5: Comparison results (Manual vs AI Method).

#		Manual	AI Method
Q1	Vote [%]	16.67 (=2/12)	83.33 (=10/12)
Q2	Mean	3.08	4.00
	SD	0.79	0.74

- *I thought that the proposed system allowed us to understand whether the current stage was reachable or not. In addition. it will also be useful for finding game bugs such as boundary conditions.*
- *Bots can be helpful to repeatedly design partial stages because the users do not worry about their skill.*

In summary, by visualizing the AI-based playtests results, the participants were able to understand where to improve in the current stage while balancing the game difficulty and the player's properties. The user study results suggest that our system could significantly reduce the time and effort of game stage design required in practice.

6 LIMITATIONS AND FUTURE WORK

The present paper focuses mainly on the visualization of playtests with bot play-styles and skill levels. However, it may be interesting to explore the possibility of extending it to include auto-tuning tools for optimizing stage components (e.g., moving or removing blocks) and game parameters (e.g., enemy speed) in the future.

7 CONCLUSION

We have proposed a user interface with an AI-based playtest for interactively designing games stages of *Super Mario Brothers*. Our system enables the users to control the detail of the bot properties (play-style and skill level) and visualize the AI-based playing results of the game in development. This system is particularly useful for finding "good" game stages. We believe that our system is a step toward accelerating game difficulty balancing for production use.

ACKNOWLEDGMENTS

This work was supported by JST CREST under grant JPMJCR17A1.

REFERENCES

Raluca D. Gaina, Rokas Volkovas, Carlos Gonzalez Diaz, and Rory Davidson. 2017. Automatic game tuning for strategic diversity. In *Proc. 9th Computer Science and Electronic Engineering Conference (CEEC)*. IEEE, Colchester, UK, 195–200.

Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen, Bartlomiej Kozakowski, Richard Meurling, and Lele Cao. 2018. Human-like playtesting with deep learning. In *Proc. IEEE Conference on Computatonal Intelligence and Games (CIG)*. IEEE, Maastricht, Netherlands, 1–8.

Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. 2018. Automated playtesting With procedural personas through MCTS with evolved heuristics. *IEEE Trans. on Games* 11, 4 (2018), 352–362.

Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. 2017. Exploring game space of minimal action games via parameter tuning and survival analysis. *IEEE Trans. on Games* 10, 2 (2017), 182–194.

Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. 2014. Monte mario: Platforming with MCTS. In *Proc. the 2014 Genetic and Evolutionary Computation Conference (GECCO'14)*. ACM, New York, NY, USA, 293–300.

Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. 2016. Modifying MCTS for human-like general video game playing. In *Proc. the 25th International Joint Conference on Artificial Intelligence (IJCAI'14)*. AAAI Press, 2514–2520.

Mark J. Nelson. 2011. Game metrics without players: Strategies for understanding game artifacts. In *Proc. the 19th AIIDE Conference on Artificial Intelligence in the Game Design Process (AIIDE'11)*. AAAI Press, 14–18.

Clive Thompson. Online; accessed 9-december-2019. Halo 3: How Microsoft Labs invented a new science of play. https://www.wired.com/2007/08/ff-halo-2/.