Enemy Evaluation AI for 2D Action-Platform Game

Pichit Promsutipong and **Vishnu Kotrajaras**
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University, Bangkok, Thailand
pichit.prp@gmail.com and vishnu@cp.eng.chula.ac.th

Abstract— **Enemy plays an important role in providing challenge in action-platform games. Our intention is to generate enemies automatically. To ensure that our enemies can be used in actual games, we need to evaluate their qualities. Using human players for the evaluation is not feasible if we generate a lot of enemies or if we are going to use the algorithm in actual games. Our approach is to create a player AI that can play action-platform games with the same performance as human players. That AI's battle performance will then be used to evaluate the generated enemies. This paper presents the definition of battle performance and the AI implementation, combining FSM, search and rule-based heuristics. The AI is shown to perform similar to human players.**

Keywords-artificial intelligence; AI; evaluator; action game;

I. INTRODUCTION

Designing games and creating game contents require time, effort and resources. These issues are particularly important for small game studios with limited budget and human resources.

Together with players' increase in demand for game contents, these problems have led to an increase in interest for procedural content generation (PCG). PCG is a technique for automating content design and creation. Various PCG techniques have been proposed to alleviate designers' burden [1]. PCG is even used in a number of commercial titles, such as Dwarf fortress [2], Spore [3] and Minecraft [4]. There are still ongoing academic researches in PCG [5].

In 2D avatar-based action games, mainly platformers, researchers usually focus on level generation. However, enemies, which are components of game levels, are also important. Each enemy provides players with a challenge, which they have to overcome in a timely manner.

To help designers explore new kinds of enemies, the ultimate goal of our research is to create an algorithm that is able to generate enemy characteristics. Generated enemies will need to be verified whether they are acceptable or not. This normally requires human playtesters. However, if we are going to generate a large batch of enemies or incorporate the generation process into an actual game, playtesting using human players is rather slow and impractical. Therefore, we have to create an AI for playtesting and evaluating the enemies.

In this paper, we present an AI controller that is able to perform similar to human players, in terms of battle performance. The reason that we use battle performance comes from how we expect to evaluate the generated enemies.

978-1-5090-4834-2/17/$31.00 ©2017 IEEE

II. GENRE OVERVIEW AND RELATED WORK

A. Genre overview

An action game is a game that requires hand-eyes coordination and fast response from players. In 2D avatar-based action-platform game, a player has to navigate an avatar through 2D environment while fighting off approaching enemies and avoiding obstacles or enemies' attacks. The goal of these games can be different from game to game. Usually, the goal is to reach the finish line of each level.

Enemies in this genre of games can be intelligent agents or fixed pattern agents. Our focus is on the latter. Players handle fixed pattern enemies by learning its behavior pattern and deciding which action to respond: fighting or avoiding. The avatar is given a number of moves (for attacking and changing the avatar position). Also, both the avatar and enemies have limited number of hits that they can take before they are considered defeated. We call this value "HP". Moreover, games of this genre usually employ invulnerable mechanic. When an avatar or an enemy is in an invulnerable state, received hits are not counted towards its HP reduction.

B. Previous work

Our previous work [6] introduced Agent Description Language (ADL), a language for describing enemies in 2D avatar-based action games. ADL was a script-like language with different types of blocks. A block described a behavior sequence. More than one blocks could be executed concurrently by a set of interpreters.

The previous work also went into detail about an experimental game that could read and execute ADL. Mechanics employed in the game were those of platformer genre. Each game level was a single empty room surrounded by walls and floor. An avatar could move horizontally, jump and shoot bullets. When an agent was attacked, if it was not in invulnerable state, that agent lost its HP.

In this paper, we used ADL to create a dataset of enemies from existing action-platform games. These enemies were then used to evaluate our AI through battle performance comparison. The game from [6] was extended to use as our testbed.

C. Other related works

Works on AI controller used to evaluate game contents were explored but we found no direct similarity with our work.

Researches involving evaluation AI included AI controller for evaluating generated simple games [7], AI controller for evaluating some aspects of strategy games described by Strategy Game Description Language (SGDL) [8] and AI controller for analyzing board game [9]. Nelson and Mateas [7] generated simple games using randomized rulesets and parameters selected from pre-defined ones. Their AI controller was evolved to evaluate learnability qualities of each generated game. Mahlmann, Togelius, and Yannakakis [8] used SGDL to create strategy games. Three aspects (balance, tension, and lead changes) of the games were evaluated using two AIs. The first AI used Monte-Carlo Tree Search technique while the other one used heuristics. Silva et al. [9] used 2 rule-based AIs, inspired by different strategies used by human players, to gather gameplay information for board game Ticket to Ride. While these works used AI controller to play games, the game genres and playtesting objectives were different from our work.

There were works on AI controller for playing Super Mario, which was a game in the same genre as ours. Baumgarten, Karakovskiy and, Togelius [10] summarized multiple AI controllers submitted for the 2009 Mario AI competition. The goal of submitted AIs was to beat a randomly generated level where an avatar had to go from left to the right-most area while avoiding taking damage from enemies. Our goal was to evaluate enemy AIs so we need our AI to control an avatar to fight against the enemies until one side was beaten or left the screen. The research goals were very different. Even though the game was in the same genre, AIs in [10] were not usable in our research.

The simulation and planning technique in [10] simulated exact future position of enemies and planned an avatar's path using that information. This made the AI too precise compared to humans. Thus, the technique could not be used in our work.

III. WORK OVERVIEW

Our AI controllers were 2-state FSM. It searched for best buttons to press at anytime based on predicted enemy path as a cost function. FSM were designed based on observed human player behavior. We expected our AIs to perform similar to human thus AI evaluation was based on battle performance comparison between AIs' and human players'. We first gave definition of battle performance and how to use it to evaluate our AIs in Section IV. Section V discussed the implementation of our AIs. Section VI presented evaluation result of our AIs. Section VII concluded our work.

IV. BATTLE PERFORMANCE

Battle performance was a set of values stating how well a player performed when fighting against an enemy. A player could either be a human or an AI. More detailed definition was described in subsection A.

Originally, we planned to use human testers to playtest the generated enemies. Acceptable enemies should force our testers to have similar battle performance score to the scores when fighting with similar enemies in well-known games. With problems stated in Section I, however, we resorted to use AI as our tester instead. For us to be able to judge the enemies in the same way as using human testers, the AI tester must yield similar battle performance as human players when fighting against the same type of enemy.

Battle performance for each battle against an enemy could be calculated from battle profile that was generated after fighting that enemy. We gave detail on what data to collect for battle profile and the collection process in subsection B. Subsection C explained how to calculate battle performance from battle profile.

A. Defining battle performance

During an encounter with an enemy in avatar-based action games, a player decided whether to fight or avoid the enemy to progress further. Fighting an enemy helped reducing threats and made level progression smoother. Avoiding an enemy and its attacks prevented damage being dealt to the avatar. To sum it up, enemy in this genre was designed so that it could be beaten or avoided. Our battle performance must therefore be able to describe how much the player could avoid attacks. It must also be able to tell us whether the player was able to beat the enemy or not.

To describe these 2 separate aspects, we defined 2 values for the battle performance: miss rate and remaining enemy HP. Miss rate was a number of attacks that hit the player compared to all attacks launched by the enemy. Remaining enemy HP was how much the enemy's HP was left compared to its initial HP at any given point in time. Detailed calculation of these 2 values is explained in subsection C.

B. Battle profile collection

To collect data for battle profile of any battle against an enemy, a player must control an avatar to fight the enemy in the testbed game until the battle end. We used the same game as in [6], with added rules. The added rules were as follows:

- When an avatar controlled by AI was attacked, it became invulnerable for 60 frames.
- When an enemy classified as Miniboss or Boss was attacked, it became invulnerable for 30 frames.

We continued to use a simple room (empty room with walls on all sides) to collect data from battle as the expected generation process did not take any level layout into consideration.

Data collector was added to the game so that data needed to calculate battle performance for each battle session could be collected. The collection process for each battle was:

1) An avatar was spawned into the level at position (50,500). The bottom-left of the screen was at (0,0).
2) At the same time, an enemy was spawned. Its initial position depended on its behavior described by ADL.
3) When the avatar reached the floor, data collection started.
4) There was a grace period when the avatar could not attack for 120 frames. This was done to prevent the enemy from getting beaten before being able to perform its action.

5) If the enemy's HP was reduced to 0, it was removed from the game. The battle then continued for 100 more frames.

6) If the avatar's HP was reduced to 0, the battle ended immediately.

7) If the enemy was marked as unbeatable, there would be a battle time limit. If time ran out, the battle ended.

8) If there was no time limit and the player thought the enemy could not be beaten, the player could skip the battle. This ended the battle immediately.

9) When the battle ended, data collection also stopped. Current session data was then stored.

The data we collected for each battle were information of each agent (the enemy and its attacks) that appeared in the game world. Collected information included: initial HP, current HP (just before the battle ended), spawning frame, lifetime, distance set, collision set and hit set. Distance set was an ordered set where each element was a distance between the agent and the avatar for each frame. Collision set consisted of frame number that the agent collided with the avatar. Hit set consisted of frame number that the agent took damage.

There was also an extra mode added to the game to collect relevant range. We defined relevant range as an average distance between an avatar and an agent that human players started to consider avoiding the agent. The relevant range collecting process was per human basis. During each relevant range collection session, a human player must control the avatar to jump and avoid an incoming attack that was launched only once. The avatar could jump only once during each session. If the avatar dodged successfully, the distance between the attack and the avatar was saved. If not, the session was repeated. There were 4 different speeds of an attack (2, 6, 10, and 14 pixels per frame), 6 sessions for each attack. Each player went through all 24 sessions in a random sequence.

C. Battle performance as AI metric

In this subsection, we discuss how to calculate battle performance values based on collected data. Battle performance was calculated per battle session. Evaluation detail is explained in Section V.

Miss rate was defined as a number of hits compared to a number of launched enemy attacks. However, some attacks might last long throughout the battle. For example, the enemy‘s body could damage the avatar, hence the enemy itself was considered an attack. Each attack usually had different lifetime. We therefore reformulated miss rate as a number of hits compared to a number of possible hits during battle. Miss rate, given relevant range *Rel*, that must be assigned beforehand, was calculated using (1):

$$MissRate = \begin{cases} \dfrac{\sum_{re \in RE} \dfrac{Hit(re)}{MaxHit(re)}}{|RE|} & , RE \neq \phi \\ 0 & , RE = \phi \end{cases} \qquad (1)$$

where *RE* was a set of all relevant agents in the battle. Only agents with distance to the avatar less than *Rel* during its lifetime were considered relevant. *MaxHit(a)* calculated a maximum number of damaging collisions that could happen between the avatar and agent *a*. When each damaging collision happened, the avatar changed to invulnerable state for 60 frames. If the avatar always took damage from *a* as soon as it came out of invulnerable state, the maximum collisions count would be *ceil(life(a)/60)*, where *life(a)* was the lifetime of agent *a*. *Hit(a)* was a number of potential damaging collisions that occurred during the battle. We only counted damaging collisions that happened when the avatar was vulnerable. The avatar was also considered being invulnerable for 60 frames if collision happened. The calculation was shown in Figure 1.

```
Initialize HitCount = 0, LastHitFrame = –1
For each frame number c in collision set of a
        If(c – LastHitFrame >= 60) Then
                HitCount++
                LastHitFrame = c
        End
End
Return HitCount as result
```

Figure 1. Pseudocode for calculation of *Hit(a)*.

Remaining enemy HP was the enemy HP at a specified time compared to its initial HP. The remaining HP, given time *t*, was calculated using (2):

$$\mathrm{Re}\,mainingHP = 1 - \frac{|\{h \mid h \in MD, h < t\}|}{MaxHP} \qquad (2)$$

where *MaxHP* and *MD* were the initial HP and hit set of the enemy, respectively.

There was also a need to calculate the duration that the enemy got beaten. It was calculated by finding the frame that the last agent was removed from the game. If the avatar lost before it could beat the enemy, the duration was extrapolated using the enemy's current HP and its initial HP.

V. AI Implementation

Our player AI was based on observed human players' behavior. The human players avoided attacks by predicting attacks path and fought back if able. We implemented the AI using 2 states FSM. Search was used to decide the best button combination to press by verifying possible outcomes when pressing certain buttons. Rule-based heuristics helped deciding the button to press if search could not determine the best one.

A. The AI and FSM

Before implementing the AI, we observed how human players battled against a number of prepared enemies. 6 players with varied experience in action-platform games were observed. We took note of how they reacted to each enemy categorized by enemy characteristics: regular enemy, unbeatable enemy (enemy that could not be beaten but could be avoided), and hidden enemy (enemy that did not appear on

the screen at first but showed up when an avatar got close enough).

The observation yielded that there were 2 separate states of behaviors, a behavior when an enemy could be damaged and a behavior when an enemy was invulnerable.

When an enemy could be damaged, the players tried to attack while avoiding incoming attacks. To avoid attacks, players usually moved far away from the enemy if able so that they had enough time to read and predict attacks' trajectory.

When an enemy became invulnerable, players were usually puzzled and started to attack and moved randomly until they were able to damage the enemy again.

The strategy led us to create our AI with these characteristics:

- It had 2 states: wandering (when could not deal damage) and fighting.
- It used prediction to determine the path of the enemy movement and its attacks.
- It re-decided button press once every 4 frames. We imposed this constraint to imitate human reaction limitation.

In the fighting state, the AI searched for the best button combination to press by checking possible outcomes from pressing each button combination. We explain the selection detail in subsection C. If there were more than one best combinations, the AI used heuristics explained in subsection D to decide the best combination depending on its previous decision and current state of the game. This process only handled the avatar's movement. For attacking, the AI tried to attack whenever the avatar face towards an enemy and the enemy was in attacking range. If the AI could not damage the enemy for 240 frames, it checked if the enemy was invulnerable. If so, the AI changed its state to wandering.

In the wandering state, the AI moved the avatar forward until it reached the wall then moved it backward. This state continued until any attack approached or the enemy became vulnerable again and was in attacking range.

B. Path prediction

The search process required future information to check for possible outcomes from pressing certain buttons. The information came from regression analysis, predicting an enemy position and its attacks movement path. We did not use simulation to get the information because prediction was how actual human players played the game and we wanted to imitate how human played games as close as possible.

Our AI stored 50 latest positions (including current position) of each active agent. These positions were used as sample data to create a regression model to predict future position of each agent. However, it was possible for an agent to change its path and caused the generated model to be inaccurate. Humans were able to detect such change by simply observing the change in speed or direction. To imitate this detection, our AI was implemented such that if any agent suddenly changed its movement speed or direction, the AI discarded all position data before the change happened. The threshold for speed change was set at 15 pixels per frame. The threshold for direction change was set at 20 degrees.

After positions were preprocessed, 2 regression models were constructed to predict the agent's x-position and y-position separately. We used linear regression analysis for the x-position, with time as independent variable and the x-position as dependent variable. The oldest position was treated as though it happened at time = 0 (0th frame). In contrast, we used degree-2 polynomial regression analysis for the y-position because our testbed had gravity, which additionally affected the change in the y-position. Time was used as our independent variable and the y-position was used as dependent variable. Figure 2 showed predicted path of enemy movement and its attacks trajectory calculated using the models.

We could not generate both regression models if there were less than 3 positions. Therefore, if there was 1 or fewer position, we could not predict anything and had to assume the agent did not move. If there were 2 positions, we assumed that the agent moved at constant speed and direction.

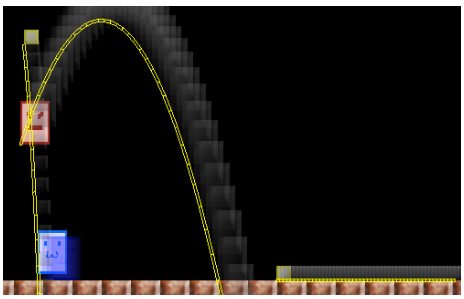

Figure 2. Predicted path indicated by yellow lines.

C. Button combination selection

Our AI searched for the best button combination and moved the avatar to the best position that was least likely to get attacked and gave the best opportunity to attack an enemy. During the searching process, the cost of selecting each button combination was calculated and the combination with the least cost was selected. Since this searching process only decided movement control, only 6 combinations were explored. These 6 combinations consisted of left press, right press, jump press, left-jump press, right-jump press, and no button press. The AI did not search any unreasonable combinations such as pressing left and right at the same time.

The cost of each combination determined how well the avatar did if the chosen buttons were pressed for 50 frames. The lower cost meant the better performance. We used prediction to get future positions of the enemy and its attacks. For our avatar's position and state, we used simulation to calculate the information. The reason for using simulation for the avatar was because the avatar was directly controlled by the controller, unlike enemy of which controlling scheme was unknown to players.

The cost was then calculated using these positions and the avatar's state in future frames. If the avatar was attacked or collided with a wall too soon in the future, the cost went up. If there was a chance for the avatar to attack the enemy, the cost went down. The calculation, given future data, was as follows:

$$Cost = EC + WC - S$$

EC calculated the collision cost that went up if the avatar collided with the enemy or its attacks. If the first collision took place before the 30th frame, this function returned 50 – (collision frame number). The cost went high if the collision happened early. However, if the first collision happened from the 30th frame onward, this function returned 0. We ignored any collision that happened in the far future as the AI would be able to change buttons beforehand.

WC calculated wall collision cost. This function returned 100 if the avatar collided with a wall before the 10th frame. Otherwise, it returned 0. We used this function to prevent the AI to run into walls and get stuck.

S calculated reward that helped to reduce cost. Reward of 1 point was given for every future frame that the avatar was considered being able to attack the enemy. To be able to attack, the avatar must face toward the enemy, the enemy must stay in the avatar's attack range, and the avatar did not attack during the previous frame.

D. Rule-based heuristics

When there were more than 1 button combinations with the lowest cost, a number of rules were applied to pick a more effective combination. We had 3 strategies: Stand, Escape, and Fallback. Firstly, the AI checked for 3 conditions: 1) there was an active enemy, 2) the avatar was on the floor, 3) the AI was not currently using Escape strategy. If all conditions were true, the AI decided to use either Stand or Escape strategy. If any of the conditions was false, the AI used Fallback strategy. The AI always picked Stand strategy if it was safe to stand (the enemy was >240 pixels away). Otherwise, it picked Escape strategy.

Before going into the details of each strategy, we introduced pick operation *Pick(A)*. When *Pick(A)* was executed, the AI randomly picked a combination from all possible combinations that satisfied a directional button condition, *A*. *Pick(A)* could be enhanced with *Priority(B,C,…)*, where *Priority(B,C,…)* was an additional preference, telling the AI to pick a combination that satisfied button condition *B* as well. If *B* could not be satisfied, the AI tried *C*, and so on. Moreover, each strategy had rules that the AI used to pick combination in order. If the AI failed to pick combination with first rule, the AI tried the next one. If all rules failed for non-Fallback strategy, the AI switched to Fallback strategy.

In Stand strategy, the 3 rules were:

1) If the agent did not face toward the enemy, *Pick*(Direction towards the enemy), *Priority*(No jump, Jump).
2) If the enemy was above of the avatar, *Pick*(Jump), *Priority*(No horizontal movement, Direction towards the enemy).
3) *Pick*(No button pressed).

In Escape strategy, there were also 3 rules. All rules used the same priority indicator. If the enemy was above the avatar, *Priority*(Jump, No jump) was used. Otherwise, *Priority*(No jump, Jump) was used. The rules were:

1) If the enemy's width or height was larger than 128 pixels, *Pick*(Direction towards area with more space).
2) If the enemy was 120 or fewer pixels away from the avatar, *Pick*(Direction away from the enemy).
3) *Pick*(Direction towards the enemy). The AI would then be forced into Escape strategy for 30 frames.

In Fallback strategy, the rules were:

1) If the avatar was falling from its jump, *Pick*(Same direction as previous decision).
2) *Pick*(Same direction as previous decision), *Priority*(Button that produced an exact combination as the previous decision, Any).
3) *Pick*(Any).

E. The second AI

Another AI controller was created, its difference from our main controller was its enemy collision cost calculation (*EC*). The first 10 frames from each calculation were removed to prevent biased cost that happened when the avatar already collided with any attacks. Without the modification, all possible collision costs would be the same if the avatar already collided with other objects.

VI. EVALUATION AND RESULT

We first re-created enemy dataset from multiple action games using ADL. The titles included Metroid, Megaman, Megaman 4, Super C, and Shovel Knight. Enemies were classified into 4 tiers: Enemy, Elite, Miniboss and Boss, depending on their HP, appearances and roles in their original games. Any enemy that was considered unbeatable by design was marked as unbeatable. Then, we collected baseline battle information values by collecting data from 14 human players.

Each player fought 5 Enemies, 4 Elites, 3 Minibosses and 3 Bosses. The enemies were selected randomly from the dataset. Any skipped battle did not count towards the required numbers. Players also participated in relevant range data collection.

Each of our AIs then battled all enemies in the dataset, 3 battles for each enemy. The evaluation process for each AI was as follows:

1) The average relevant range was calculated, by averaging all ranges collected from 14 human players.
2) Battle data where an enemy battled against AI (in all rounds) and against at least 1 player, the enemy was not unbeatable, and the battle was not skipped, were stored as battle data *A*.
3) Battle data where an enemy battled against AI (in all rounds) and against at least 1 player, and there existed an agent that had distance to the avatar less than or equal to the average relevant range during its lifetime, were stored as battle data *B*.
4) Average battle durations for humans (for each enemy tier) were calculated from available human data in battle data *A*.

5) Average miss rates for humans (for each enemy tier) were calculated from available human data in battle data *B*.

6) Average miss rates for AI (for each enemy tier) were calculated from available AI data in battle data *B*.

7) Average remaining enemy HPs when fighting AI (for each enemy tier) were calculated from available AI data in battle data *A*. The average human battle durations from step 4 were used as capped durations.

8) AI's miss rates from step 6 were compared with humans' miss rates from step 5. If the difference between humans' and an AI's miss rates was 0.1 or lower for all enemy tiers, we considered that the AI had similar miss rate compared to humans.

9) For each enemy tier, its remaining enemy HP from step 7 was compared with zero. If the difference was 0.1 or lower for all tiers, we considered that the AI was able to fight its enemies as good as human players did. The value from step 7 was essentially the remaining enemy HP that the AI could cause within the average duration that humans took to beat an enemy of that tier.

10) If both AI's miss rates and remaining enemy HP values were similar to humans', the AI was considered similar to human in terms of battle performance.

Our evaluation result was shown in Table I. All performance differences (miss rates and remaining enemy HPs) of both AIs were lower than 0.1. Therefore, we concluded that both AIs were able to perform similar to human according to our metrics. Table I also showed average performance difference of both AIs. The second AI had slightly lower difference values, which meant it performed better.

TABLE I. AI EVALUATION RESULT

Performance	**Enemy tier**			
	Enemy	***Elite***	***Miniboss***	***Boss***
Human				
Miss rate	0.09356	0.11046	0.15996	0.20183
First AI				
Miss rate	0.07878	0.07866	0.11228	0.24947
Miss rate difference	**0.01478**	**0.03180**	**0.04768**	**0.04764**
Remaining enemy HP	**0.05469**	**0**	**0**	**0.00452**
	Average performance difference = 0.02514			
Second AI				
Miss rate	0.07245	0.06670	0.12458	0.18738
Miss rate difference	**0.02111**	**0.04376**	**0.03538**	**0.01445**
Remaining enemy HP	**0.05789**	**0**	**0**	**0.00132**
	Average performance difference = 0.02174			

From the table, we can see that the major difference was in miss rate when AIs battled against Boss tier enemies. The only difference in calculation between the two AIs was how we calculated *EC* in the cost function, where the second AI ignored its first 10 future frames. Since Boss tier enemies usually caused more collisions, we could say that the second AI was more similar to human when many collisions took place. Indeed, when his avatar took damage, a human player tended to ignore the colliding agent and focused on the next incoming attack instead.

VII. CONCLUSION AND FUTURE WORK

We created 2 AIs for evaluating enemies in 2D avatar-based action-platform games. The AIs were required to be similar to human players in terms of battle performance.

The AIs used 2-state FSM, fighting and wandering. In fighting state, we employed searching to select button combination with the lowest cost to navigate an avatar. Simulation was used to find the avatar's state in the future. In contrast, we used regression analysis to predict future position of an enemy to replicate how humans made decision to control the avatar. If the AIs could not decide which combination was the best, we used a number of rules to help the AIs select a combination. Our two AIs differed in their cost function where the second AI ignored some frames in the future.

We collected battle performance for both AIs and compared them against those of 14 human players to test for similarity. Both AIs performed similar to human. The second AI showed slightly better overall result, especially in a situation where its avatar fought against boss enemies.

For our future work, we plan to shape up our enemy behavior generation algorithm. After the algorithm is finished, the second AI will be used to evaluate the algorithm's generated enemies.

REFERENCES

[1] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, "Procedural content generation for games: A survey," ACM Trans. Multimedia Comput. Commun. Appl., vol. 9, pp. 1-22, 2013.

[2] Dwarf Fortress, Bay 12 Games, 2006.

[3] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. v. Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," ACM Trans. Graph., vol. 27, pp. 1-11, 2008.

[4] M. A. Persson (2011). Terrain generation, Part1. Available: http://notch.tumblr.com/post/3746989361/terrain-generation-part-1

[5] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," Computational Intelligence and AI in Games, IEEE Transactions on, vol. 3, pp. 172-186, 2011.

[6] P. Promsutipong, P. Kaewmark, and V. Kotrajaras, "Towards automatic enemy generation for 2D avatar-based action games: agent model and description language," presented at the International Conference on Computer Games, Multimedia & Allied Technology (CGAT). Proceedings, 2015.

[7] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On, 2008, pp. 111-118.

[8] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Modelling and evaluation of complex scenarios with the strategy game description language," presented at the Computational Intelligence and Games (CIG), 2011 IEEE Conference on, 2011.

[9] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "AI as Evaluator: Search Driven Playtesting of Modern Board Games," 2017.

[10] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," presented at the Evolutionary Computation (CEC), 2010 IEEE Congress on, 2010.