Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 8, No. 3, August, Pp. 633-640

Threat construction for dynamic enemy status in a platformer game using classical genetic algorithm

Ardiawan Bagus Harisa*[1], Setiawan Nugroho[1], Liya Umaroh[1], Yani Parti Astuti[1]
Department of Computer Science, Universitas Dian Nuswantoro, Indonesia[1]

Article Info
Keywords:
Adaptive Enemy, Platformer, Procedural Content Generation, Genetic Algorithm, Game Pacing

Article history:
Received: May 02, 2023
Accepted: June 12, 2023
Published: August 31, 2023

Cite:
A. B. Harisa, S. Nugroho, L. Umaroh, and Y. P. Astuti, "Threat Construction for Dynamic Enemy Status in a Platformer Game using Classical Genetic Algorithm", KINETIK, vol. 8, no. 3, Aug. 2023. https://doi.org/10.22219/kinetik.v8i3.1724

*Corresponding author.
Ardiawan Bagus Harisa
E-mail address:
ardiawanbagus@dsn.dinus.ac.id

Abstract
Digital game genre such as Action-Platformer is widely popular among buyers on a platform like Steam. The non-playable character enemies in the game are important in action games. Unfortunately, they usually have static attributes like health points, damage, and enemy movement. Using the combination of procedural content generation and dynamic difficulty adjustment with a classical genetic algorithm, we drive the threat value of a platform to construct the enemy status, resulting in more dynamic enemies. We use the threat value as an input parameter calculated from the enemies' stats in every platform, such as total damage that the enemy might produce, the player's health point, and the enemy's movement speed. We conclude that using a classical genetic algorithm may produce dynamic enemy status through the desired threat or danger set by the game designer as an input parameter. Moreover, the game designer may limit the generation with constraints.

1. Introduction

A digital game is a game that is played on a computer-based platform, the interaction between a human who seeks pleasure and a machine that provides the interaction [1]. In today's game industry, in the Action category, the platformer is one among popular sub-genres [2]. According to data published by the Entertainment Software Association (ESA), the Action-Adventure Platformer game genre from 2017 to 2022 was in America's top 5 biggest sales [3], [4]. Platformer is separated into several subs, including, but not limited to, 2D Platformers, Precision Platformers, Puzzle Platformers, and Runner. Some new categories are rising from the old game, say Metroidvania, now considered a separate, independent sub-category [5]. Since 2020, more than 51 games have contained "Platformer" in their title on Steam, and now 395 games are under the platformer genre.

One of the reasons why platformer games are popular is because the avatar on the game connects the player to the virtual environment through a digital (imaginative) identity [6]. In platformer games, providing challenge and danger to players is critical [7]. One of the popular approaches from the game designer is placing the Non-Playable Character (NPC) like an enemy on the platform where the player moves as progresses in the game. Unfortunately, suppose the player has played the game several times. In that case, they already build a rigid strategy since the enemy status is always the same every time they play, which reduces the game dynamic. We can use a familiar method such as dynamic difficulty adjustment (DDA) to simply change the NPC's attributes without bothering about their different appearances [8], or procedural content generation (PCG) to make the enemy more dynamic or even adaptive [9]–[11]. One way is to generate the status of the enemies differently every time player plays the game, such as the enemy's health point (HP), speed, and damage produced.

Various DDA and PCG techniques can be used to generate dynamic danger in platformer games [12]-[17]. For example, we can use advanced techniques like machine learning, an evolutionary algorithm like a genetic algorithm (GA), or even as simple as a finite state machine (FSM). GA is one of the popular techniques in game research since it is the heuristic algorithm to find the optimal solution to a problem [9], [11], [14].

This paper shows how to minimize the boring, predicted experience by generating enemy status dynamically in a platformer game by constructing the danger perceived by players using classical GA. Our research is inspired by the PCG implementation among the research to provide a dynamic experience through the contents, for instance, the enemy. Although our research is strongly related to DDA, we are not only focused on the difficulty adjustment. Similar to [18], we can also provide the generation of contents and the constraints. For instance, the types of enemies (and their appearances), their positions, and more. Because the game's difficulty is in the game mechanics, and the mechanics itself is one of the game contents, then we may say that procedurally generating a dynamic difficulty on the

NPC is also a topic in PCG. Section 1 explains the research motivation and relevant works. Section 2 provides the main methodology. Section 3 contains the experimental results and discussions. Finally, section 4 presents the conclusions.

1.1 Dynamic Enemy

As we mentioned earlier, by using PCG, we can provide the player with experience to increase engagement. Here we show the related work to our research, using PCG to generate the intended level by the game designer. Dynamic enemy is a term in the video game used to describe that the enemy in it can change and adapt dynamically towards the given configurations, resulting in the NPC enemies can adapt in different situations and increasing player's engagement. However, the goal is to ensure that the difficulty fits the player according to their performance and keeps them motivated to play the game [16].

One example of dynamic enemy implementation is an adaptive enemy which will adapt to player performance [16], [17]. For instance, if a player with a high-performance score, say, kills many enemies or completes the tasks in a short period, the higher chance the game will evolve the provided enemy to increase the difficulty, enhancing the damage, speed, base health points, and may result in more advance strategy needed to defeat. In contrast, when the player is facing a defeat, the system reduces its difficulty by lowering the movement speed and damage of enemy. It ensures the game is still challenging based on any player progressing in the game.

1.2 PCG in Platformer Games

A platformer game is a video game genre where the player can move on an environment (platform) to take the player to reach the goal. The platform in the platformer game can be a flat, linear path to uneven, moving parts [8]. In our study, the platforms used are shown in Figure 1 and Figure 7, where in on each platform, there are enemies and other properties related to the game environment. The platforms used have a box shape since our game is a 3D platformer with a little bridge between platforms.

Procedural Content Generation (PCG) is a content creation process in the game automatically using the mathematic procedure or algorithm, such as game levels, items, character cosmetics, enemies, and more [19]. The goal is to help the developer build a massive amount of various content alternatives, avoiding manual work. Studies on implementing the PCGs techniques have been published.

Moghadam and Rafsanjani in [20] uses GA to generate and evaluate rhythms from creating 2D runner platformer levels. First, they generate rhythms using GA, then use the resulting rhythm patterns to generate the level's geometry, and finally decorate the level with props. Sarkar et al. developed a system to automatically place collectible coins in platformer games based on the desired path [21]. They use the player's trajectories to detect the desired path. Their study focused on placing the coins on the path so the player can naturally move while collecting them in a platformer game.

The use of LSTM (long-short term memory recurrent neural network) has been shown to generate Super Mario's level alternatives with the help of the snaking-depth-path approach [22]. Snaking is an alternative approach where instead of handling the platformer data as a sequence of objects bottom-to-top individually, we consider it continuous columns. Therefore, it can be bottom-to-up on the predecessor column and top-to-bottom on the next one. Similar to the previous study, their study uses player path trajectory as path information for level generation. Finally, the depth of the approach means they consider 12 past columns as information for LSTM to remember. To make it more interesting, they increase the emergence of special objects after every four columns.

While the studies focus on rhythm patterns and player trajectory to generate a platformer level, [18] uses GA to drive the level generation with pacing aspects as parameters. Pacing aspects such as threat, movement impetus, and tempo are intuitively designed and calculated to predict the game flow. The authors provide a simple and intuitive curve-based user interface to represent the pacing curves on a game level. Our research borrows the threat aspect to be a driver to generate the enemies' status. The original concept was applied to the dungeon, rogue-like game. However, as stated in their paper, we may still implement the concept since the threat aspect is calculated differently according to the game design, including the game's genre.

Dynamic enemy is a term in the video game used to describe that the enemy in it can change and adapt dynamically towards the given configurations, resulting in the NPC enemies can adapt in different situations and increasing player's engagement. However, the goal is to ensure that the difficulty fits the player according to their performance and keeps them motivated to play the game [16].

2. Research Method

Our study focuses on providing a player with dynamic enemies on a platformer game using a classical genetic algorithm, especially to evolve the enemies' status as designed by the game designer using one of the pacing aspects, threat. The threat aspect mostly discusses the danger on a platform chunk that emerge from the enemies. In summary, the input of our system is the original status of enemies, then modified with GA by calculating the threat value on each platform.

In this study, we borrow the concept of pacing aspects [18] and focus only on the threat. Since the threat implementation in every game genre varies, we modified the threat calculation as shown by Equation 1, Equation 2, and Equation 3. We normalize the threat value so that the designer can easily configure it and the constraints. We collect data through a literature study, questionnaire, and experiments. Through literature study, we search for what minimum criteria for making a platformer level. The platformer contains player objects, chunks of the platform, collectible items, obstacles and enemies, and finally, a finish point [8], [18], [22], [23].

In later section, we conducted user studies by asking seven students in Intelligent System Laboratory, Universitas Dian Nuswantoro, to fill out the questionnaire after they played generated game level. Furthermore, we will compare the questionnaire data and the recorded gameplay data to find a conclusion.

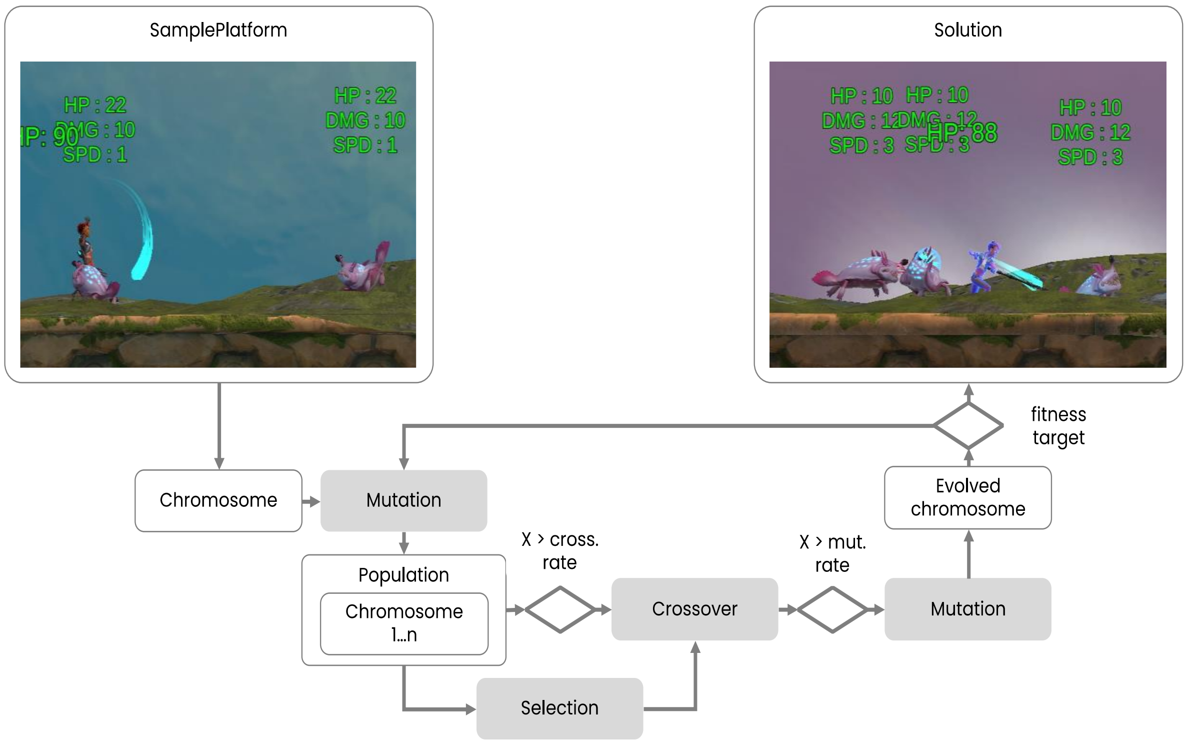

Figure 1. Our system's overview to generate dynamic enemy status

2.1 Input Parameters

Procedural Content Generation (PCG) is a content creation process in the game automatically using the mathematic procedure or algorithm. We create a dynamic enemy with a generator based on the threat value as a fitness target for genetic algorithm. Since we are focusing only on the threat produced by the enemy in the platforms, the level of space or geometry itself can be produced manually or automatically using our previous study [18]. For this study, we will provide a player with five main platforms and one boss platform. Please note that the equations below are seen from the enemy's threat perspective. The output of our system is also a chunk of platform where the enemies' status and number might be modified by the system. We are highly focus on generating the enemy status based on the designer's target threat automatically. Therefore, the designer does not need to manually craft the enemies in every platform. The designer just needs to set the target threat, and then the enemies' status are changed accordingly.

2.1.1 Damage

First, we modified the metrics and used the minimum parameters possible to implement the concept for various game genres easily. Total damage produced by the enemy is a significant parameter as input for our system, indicating the threat for the player. According to Equation 1, the effectiveness of total damage f_{DmgE} is the result of damage of every enemy in a platform $DmgE$ multiplied by the attack frequencies AfE then divided by player HP HPp.

$$f_{DmgE} = \sum_{i=1}^{nE} \frac{DmgE_i \times AfE_i}{HP_P} \quad (1)$$

2.1.2 Health Point

The base level of the player's health point is also a crucial factor contributing to the threat perceived by the player. Therefore, the player's hp ratio to the enemy's hp is also essential. The effectiveness of enemy's HP is shown with f_{HpE} which calculated from total enemies HPs HpE divided by player's HP HP_P.

$$f_{HpE} = \sum_{i=1}^{nE} \frac{HpE_i}{HP_P} \tag{2}$$

2.1.3 Movement Speed

Finally, the effectiveness of the enemy's movement speed compared to the player's is also considered as it also contributed to the threat for the player. f_{MoveE} is calculated from the ratio player's speed Sp_P compared to enemies's speed SE_i.

$$f_{MoveE} = \sum_{i=1}^{nE} \frac{SE_i}{Sp_P} \tag{3}$$

2.2 Genetic Algorithm

We run our GA for a single platform on the game scene. The number of the run is dependent on the number of target solutions for each platform. The flow of our GA is shown in Algorithm 1. We use the classical genetic algorithm to find the fittest solution represented as chromosomes. Here is how it works:

Input: The input to the algorithm is a sample platform $\mathrm{samplePlatform}$ where the enemies lie (it includes the enemies). The sample platform is then converted as a chromosome c, a solution individual.

Step 1: Initialize the while loop. The algorithm continues to iterate until the maximum fitness from the best chromosome in the population C (denoted as $\mathrm{MaxFit(C)}$) reaches a specified fitness target $\mathrm{fitTarget}$.

Step 2: For each iteration of the while loop, generate a new population C by cloning and mutating the existing individuals in the population as many as determined by nc.

Step 3: Update the population C with the newly generated individuals (clones) by adding them to the existing population.

Step 4: Calculate the maximum fitness among the individuals in the population C and select it as the current fittest individual sc.

Step 5: Perform crossover (denoted as $\mathrm{Cross(\,)}$) between the highest-fitness individual sc and each individual in the population C with a probability of r (random number) $< \mathrm{cRate}$, that is 0.05.

Step 6: Perform mutation (denoted as $\mathrm{Mut(\,)}$) on each individual in the population C with a probability of $\mathrm{r} < \mathrm{cRate}$, also 0.05, using the highest-fitness individual sc as a reference.

Step 7: Repeat steps 2-6 until the maximum fitness of the population $\mathrm{MaxFit(C)}$ reaches the fitness target.

Step 8: Once the fitness target is reached, return the maximum fitness of the final population C as the algorithm's output.

In summary, this algorithm uses genetic operations like cloning, crossover, and mutation to evolve a population of individuals towards a threat calculated using Equation 1, Equation 2, and Equation 3 as a fitness target. Then, the algorithm iteratively generates new individuals, updates the population, performs genetic operations, and repeats the process until the fitness target is achieved.

Algorithm 1. *The Flow of GA in our system.*

```
$s \leftarrow \mathrm{samplePlatform}$
$c \leftarrow s$
while $\mathrm{MaxFit(C)} < \mathrm{fitTarget}$ do:
  for $i = 0$ to $nc$ do:
    $c \leftarrow \mathrm{Mut(clone(c))}$
    $C \cup \{c\}$
  $sc \leftarrow \mathrm{MaxFit(C)}$
  $\forall c \in C, c \leftarrow \mathrm{Cross}(sc, c), r < \mathrm{cRate}$
  $\forall c \in C, c \leftarrow \mathrm{Mut}(sc, c), r < \mathrm{mRate}$
return $\mathrm{MaxFit(C)}$
```

3. Results and Discussion

3.1 Experiments

Our initial experiments use 30, 50, and 100 generations as genetic parameters with 25, 50, and 100 chromosomes in the populations. Figure 2 and Figure 3 shows the result averaged fitness of five runs. However, according to the result of experiments, using 30 generations produces the worst fitness, and the best fitness showed when using 100 generations and 100 chromosomes. Therefore, we use 100 generations and 100 chromosomes each to ensure the fitness converges in the expected iteration numbers and to avoid low-quality solutions resulting from the low number of generations or individuals.

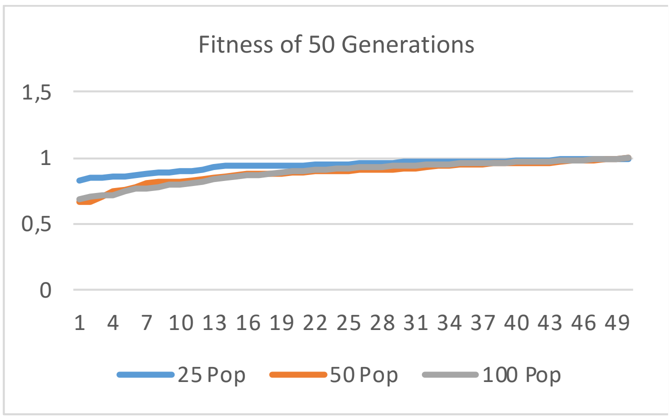

Figure 2. The ftness of using 25, 50, and 100 population size with 50 generations each

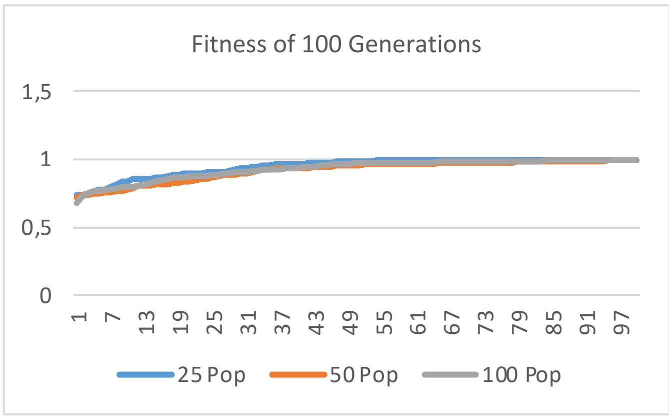

Figure 3. The fitness of using 25, 50, and 100 population size with 100 generations each

In our further experiments, we construct a platformer level using eight big platform chunks in the level experiment. Using the genetic parameter obtain from initial experiment, the GA will run on each platform individually. The designer can set the target threat as a fitness target and constraints such as the max value of any status attribute from enemies. Table 1 shows that the generated enemies' errors are low. The highest error rate was observed at platform 8. It is because we configure the constraints for all platforms uniformly, sometimes making the solution hard to converge. However, as mentioned in [18], adjusting the constraints can easily address such cases. Moreover, the space layout and properties, as mentioned before, will also have an impact on the error. Fortunately, we did not consider such attributes to be calculated. From Table 1, we set the lowest target threat to 0.1 on platform 1 and 0.9 on the boss platform. We intuitively categorize the threat as easy (0-0.3), medium (0.31-0.6), and hard (0.61-1) difficulty.

Table 1. Fitness differences from platforms

Platform	Target Threat	Avg. Fitness	Avg. Error Rate
1	0.1	0.999	0.13%
2	0.2	0.998	0.24%
3	0.3	0.998	0.18%
4	0.4	0.997	0.28%
5	0.5	0.996	0.37%

6	0.6	0.989	1.12%
7	0.7	0.984	1.58%
8	0.8	0.960	**3.98%**
Boss	0.9	0.997	0.31%
	Avg.	0.991	0.91%

3.2 User Study

In this user study, we record the seven players' gameplay data and ask them to complete the questionnaire. Figure 4 shows the average duration (elapsed time) played by players. In Figure, 4, Figure 5, and Figure 6, the green bar depicts the easy difficulty, the yellow means medium, and the red is the hard difficulty. According to the threat configuration in Table 1, higher threats result in a longer time played by the players on a platform. In an easy setting, the players spent 27 seconds on average. While on the medium platforms, they spent 72 seconds and 144 seconds on hard platforms. So the ratio of elapsed time between easy:medium:hard is 1:3:6. The average elapsed time in the boss platform is significantly higher because players tend to patiently attack with certain strategies to avoid critical damage.

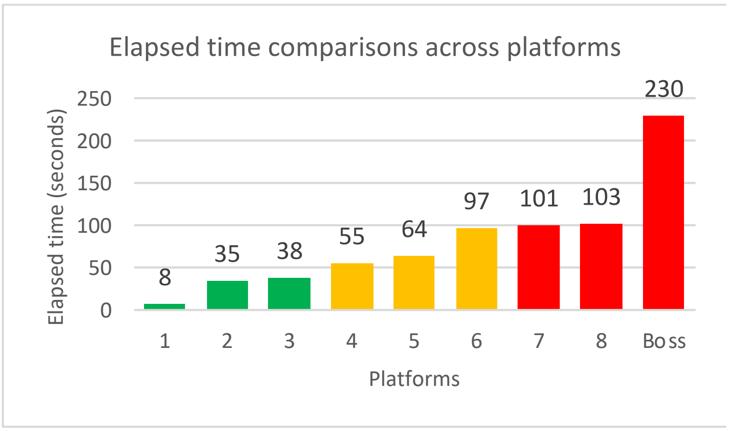

Figure 4. The comparisons of elapsed time by player across platforms

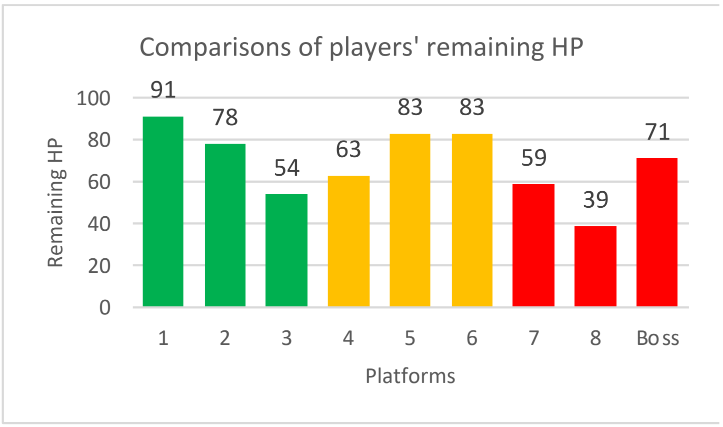

Figure 5. The comparisons of player's remaining HPs across platforms

Figure 6 shows the players' responses to the difficulty of the platforms. All players agree that the early platform 1 and 2 is easy to play, platform 3 to 5 are medium, and 6 to 8 are hard to play. However, again, there is an anomaly on the boss platform because some players have been defeated several times, and they need to replay the platform, and some other players wait very patiently to attack the boss enemy. That is why in Figure 4, the players spent significantly longer time, resulting in players sharing their difficulty perceptions differently to respond, as shown in Figure 6. The actual players' perception of threat perfectly fits our expectations on platforms 1, 2, 7, and 8. Platforms 4 and 5, respectively, show 71.4% and 57.1% towards our expectations. In contrast, platforms 3 and boss show low scores, below 50%. Still, the average player experience of our designed threat is 71%. Figure 7 shows the capture of our platformer game, which we have developed using Unity's asset.

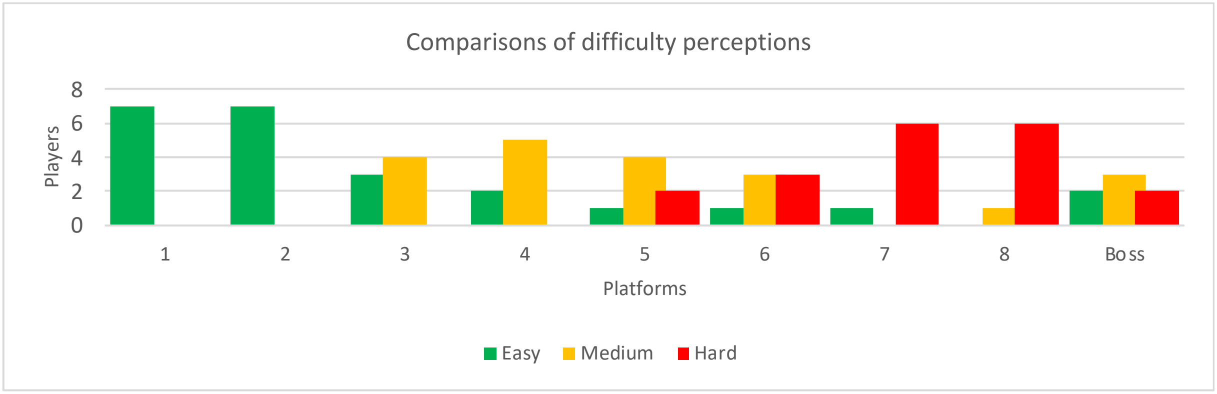

Figure 6. The questionnaire results from players across platforms

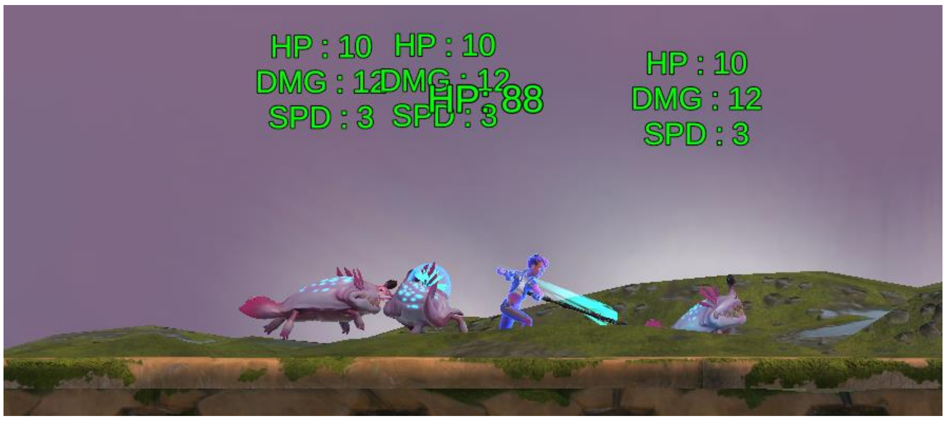

Figure 7. The screenshot of platform 4 in our platformer game. The figure shows the enemies' attributes, such as health point (HP), Damage (DMG), and speed (SPD). The free assets are downloaded from Unity Asset Store.

4. Conclusion

In this study, we construct the player experience in a platformer game, focusing on the concept of threat by modifying the enemy status in each platform with a genetic algorithm. We use enemies' damage, HP, and movement speed as input parameters to drive our system. To calculate the threat value as one of the pacing aspects, we modify the original one in [18] for a dungeon, rogue-like level, to a platformer level. The flow of our GA is straightforward. We create a population from a sample platform, then select the best individual and perform crossover and mutation with a certain probability rate within iterations. After the fitness threshold is reached, the GA produces the best solution.

We performed experiments to seek validation of our concept. First, we configured the threat between platforms in a level to be 0-0.3 (easy), 0.31-0.6 (medium), and 0.61-1 (hard). The average fitness of using those settings is 0.99, and the error rate is 0.91%. Next, we performed the user study by asking them to play a level containing nine platforms, including the boss platform. The recorded gameplay data and responses from the players are then analyzed, which are: 1) higher threat results in longer elapsed time, 2) higher threat results in higher damage produced by enemies, and 3) more than 71% of players' responses agree with the designed threat provided in the testing session. To conclude, we provide early work in a platformer game to provide the expected player experience using one of the pacing aspects, a threat, as a basis.

There are some notable limitations of our work. First, we focus only on one aspect of pacing, threat, and implement it on a platformer game which differs from the original concept (dungeon level). In the other more complex genres, more parameters must be considered as input to drive the system. This research is part of our roadmap to build a generative system for various game genres using pacing patterns aspects to drive the generation process. Our suggestion for further research is: 1) more experiments to validate and measure the generated level needed to perform, 2) the threat patterns can be presented with various representations, such as graph grammar, 3) the platforms can be built to be more dynamic and less linear (platform options in different vertical height).

Acknowledgement
We would like to express our gratitude to the Universitas Dian Nuswantoro to provide the funding for this research.

References

[1] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. The MIT Press, 2003.

[2] N. Peever, D. Johnson, and J. Gardner, "Personality & Video Game Genre Preferences," in *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, in IE '12. New York, NY, USA: Association for Computing Machinery, 2012. https://doi.org/10.1145/2336727.2336747

[3] L. Husniah, F. Fannani, A. S. Kholimi, and A. E. Kristanto, "Game Development to Introduce Indonesian Traditional Weapons using MDA Framework," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, pp. 27–36, Nov. 2018. https://doi.org/10.22219/kinetik.v4i1.713

[4] Entertainment Software Association (ESA), "2022 essential facts About the video game industry," Entertainment Software Association. 2022.

[5] Valve, "Steam Store," Apr. 09, 2023.

[6] T. H. Apperley, "Genre and game studies: Toward a critical approach to video game genres," *Simul Gaming*, vol. 37, no. 1, pp. 6–23, Mar. 2006. https://doi.org/10.1177/1046878105282278

[7] E. F. Melcer and M. A. M. Cuerdo, "Death and rebirth in platformer games," *Game User Experience And Player-Centered Design*, pp. 265–293, 2020. https://doi.org/10.1007/978-3-030-37643-7_12

[8] A. B. Harisa, H. Haryanto, and H. A. Santoso, "Model Tingkat Kesulitan Dinamis berbasis Logika Fuzzy pada Game Wayang Ramayana," in SEMNASTEKNOMEDIA ONLINE, 2016, pp. 2–6.

[9] A. Gellel and P. Sweetser, "A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, in FDG '20. New York, NY, USA: Association for Computing Machinery, 2020. https://doi.org/10.1145/3402942.3402945

[10] G. N. Yannakakis and J. Togelius, "Experience-Driven Procedural Content Generation," *IEEE Trans Affect Comput*, vol. 2, no. 3, pp. 147–161, Jul. 2011. https://doi.org/10.1109/T-AFFC.2011.6

[11] M. Kaidan, C. Y. Chu, T. Harada, and R. Thawonmas, "Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm," in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, Oct. 2015, pp. 535–536. https://doi.org/10.1109/GCCE.2015.7398674

[12] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," *arXiv preprint arXiv:2002.11869*, 2020. https://doi.org/10.48550/arXiv.2002.11869

[13] S. M. Lucas and V. Volz, "Tile Pattern KL-Divergence for Analysing and Evolving Game Levels," in *Proceedings of the Genetic and Evolutionary Computation Conference*, in GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 170–178. https://doi.org/10.1145/3321707.3321781

[14] R. A. Pambudi, W. Lubis, F. R. Saputra, H. P. Maulidina, and V. N. Wijayaningrum, "Genetic Algorithm for Teaching Distribution based on Lecturers' Expertise," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 4, no. 4, pp. 297–304, Oct. 2019. https://doi.org/10.22219/kinetik.v4i4.859

[15] Nery Bandeira, I. et al. (2022). Dynamic Difficulty Adjustment in Digital Games: Comparative Study Between Two Algorithms Using Electrodermal Activity Data. In: Fang, X. (eds) *HCI in Games*. HCII 2022. Lecture Notes in Computer Science, vol 13334. Springer, Cham. https://doi.org/10.1007/978-3-031-05637-6_5

[16] J. Pfau, J. D. Smeddinck, and R. Malaka, "Enemy Within: Long-Term Motivation Effects of Deep Player Behavior Models for Dynamic Difficulty Adjustment," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, in CHI '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–10. https://doi.org/10.1145/3313831.3376423

[17] M. Zohaib, "Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review," *Advances in Human-Computer Interaction*, vol. 2018, p. 5681652, Nov. 2018. https://doi.org/10.1155/2018/5681652

[18] A. B. Harisa and W. K. Tai, "Pacing-based Procedural Dungeon Level Generation: Alternating Level Creation to Meet Designer's Expectations," *International Journal of Computing and Digital Systems*, vol. 12, no. 1, pp. 401–416, 2022. https://doi.org/10.12785/ijcds/120132

[19] B. M. F. Viana and S. R. dos Santos, "A Survey of Procedural Dungeon Generation," in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019, pp. 29–38. https://doi.org/10.1109/SBGames.2019.00015

[20] A. B. Moghadam and M. K. Rafsanjani, "A genetic approach in procedural content generation for platformer games level creation," in *2017 2nd Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, 2017, pp. 141–146. https://doi.org/10.1109/CSIEC.2017.7940160

[21] A. Sarkar, R. Padte, J. Cao, and S. Cooper, "Desire Path-Inspired Procedural Placement of Coins in a Platformer Game," in Joint Proceedings of the AIIDE 2018 Workshops co-located with 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2018), Edmonton, Canada, November 13-14, 2018, J. Zhu, Ed., in CEUR Workshop Proceedings, vol. 2282. CEUR-WS.org, 2018.

[22] A. Summerville and M. Mateas, "Super Mario as a String: Platformer Level Generation Via LSTMs," *CoRR*, vol. abs/1603.00930, 2016. https://doi.org/10.48550/arXiv.1603.00930

[23] B. Piller, C. Johanson, C. Phillips, C. Gutwin, and R. L. Mandryk, "Is a Change as Good as a Rest? Comparing BreakTypes for Spaced Practice in a Platformer Game," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, New York, NY, USA: ACM, Nov. 2020, pp. 294–305. https://doi.org/10.1145/3410404.3414225