Jurnal Teknik Informatika (JUTIF)
Vol. 4, No. 6, December 2023, pp. 1437-1446

DOI: https://doi.org/10.52436/1.jutif.2023.4.6.546
p-ISSN: 2723-3863
e-ISSN: 2723-3871

DESIGNING A 3D ROGUELIKE GAME WITH PROCEDURAL CONTENT GENERATION USING THE GRAPH GRAMMARS METHOD

Arie Vatresia*[1], Ferzha Putra Utama[2], Adi Yulianto[3]

[1,3]Informatics, Faculty of Engineering, Universitas Bengkulu, Indonesia
[2]Information Systems, Faculty of Engineering, Universitas Bengkulu, Indonesia
Email: [1]arie.vatresia@unib.ac.id, [2]fputama.edu@unib.ac.id, [3]adiyulianto888@gmail.com

(Article received: August 11, 2022; Revision: September 16, 2022; published: December 23, 2023)

Abstract

Roguelike is a genre of role-playing video game in which the player explores dungeons through procedurally generated levels. If they lose, the player loses progress, and the character starts over again. Procedural Content Generation (PCG) is a computer program that can create game content automatically, randomly, and uniquely, either by itself or with human assistance. In this study, the 3D roguelike game was designed with players playing a character to explore dungeons. Players will enter at one point, explore the surrounding environment, defeat the enemies encountered, avoid traps, collect treasure, and finally exit at another point. Each time the player starts a new game session, the game will generate a dungeon with a mission structure that changes randomly to create a variety of gameplay. This mission generation is implemented using the Graph Grammar method. The game is built using the Unity game engine and is intended to run on Android devices. Based on the black box test results, all the game's features are running well according to their functions. The built games will be evaluated using the GUESS-18 to determine the level of player satisfaction. Based on the evaluation results, the game is included in the "GOOD" category, with an overall score of 49.07 out of 63 maximum scores. The game that has been built is superior in the aspect of personal gratification, while it is weak in the aspect of social connectivity.

Keywords: *Game, Graph Grammars, GUESS-18, Procedural Content Generation, Roguelike.*

1. INTRODUCTION

Roguelike is a genre of role-playing video games in which players explore dungeons through procedurally generated levels. If the player loses, he will lose progress, and the character starts again from the beginning. Players play a character by exploring dungeons. He will enter at one point, explore the surrounding environment, defeat enemies, avoid traps, find ways to enter the locked section, defeat boss enemies, collect treasures, and finally exit at another point. Players usually explore several dungeons with different views, themes, and difficulties in a mission [1], [2].

Permadeath in roguelike games requires players to repeat the level from the beginning if they lose. Roguelike games rely on PCG to keep this repetitive gameplay from giving players a bland impression. Procedural Content Generation (PCG) is a computer program that can create game content automatically, randomly, and uniquely, either by itself or with human assistance. The content here is anything contained in the game. It can be levels, maps, game rules, textures, stories, items, missions, music, weapons, vehicles, or characters [3]–[6]. According to Dormans, Graph Grammars can be used to generate game levels by dividing them into mission generation and area generation [7]. However, in his study, Dormans still applies it as a graph representation, and it has not yet become a game that can be played. The author is interested in building 3D games by applying the Graph Grammar method for dungeon generation in a playable game [7].

The main problem raised in this research is how to create a dynamic game mission structure so that the generated dungeon and game mission structure continuously varies every time players start a new game. The first problem will be given a solution by applying the Graph Grammar method during the dungeon generation process. The second problem discussed in this final project is how to design gameplay from a 3D roguelike game to a ready-to-play game. The design will be done by making Game Design, which includes game flow, game mechanics and character control, visual and audio display, and mission/objective structure. The last problem that will be discussed in this research is how to find out the satisfaction and playing experience of the players in the games. Play satisfaction will be measured by distributing questionnaires to the game community "Team Kito".

In this study, the author designs a 3D roguelike game with the camera configuration on top facing down (top-down). Players will go through many dungeons to reach the main enemy [2], [8]. The primary mission in the game will be defined first in the form of a mission graph. Then, the game will apply PCG with the Graph Grammars method to change the basic mission graph to be more varied by applying content generation rules. The rules that will be designed include adding enemies in the middle of the journey, placing items at several points in the dungeon to help players, and adding or randomizing the arrangement of dungeons. Other rules include locked dungeons, so players must find specific keys to access the next dungeon. After the new mission structure is formed, 3D game assets such as the environment, enemies, items, and characters will be generated. So that after content generation, a game arena will be formed, and each game will vary starting from the beginning.

Game design is done by the Prototyping method. The game is built using the Unity engine and can be played on Android devices. The built games will be tested with a black box to determine whether the available features follow the expected design. After the game design has been successfully implemented, the game will be evaluated to determine the level of satisfaction and playing experience of players using the GUESS-18 framework. The evaluation was carried out on the game community "Team Kito" using a questionnaire for data collection.

2. METHODS

2.1. Types of research

The type of research used is applied research. In this case, the research aims to apply Procedural Content Generation using the Graph Grammars method in a 3D roguelike game.

2.2. Research Objects and Subjects

The object of research is the state of an object that is the target in a study [9], [10]. The object of this research is the application of Procedural Content Generation using the Graph Grammars method into a 3D roguelike game, namely the game mission structure. Research subjects are individuals who are targeted in collecting research data. The research subject in this study is the "Team Kito" game community in Bengkulu City.

2.3. Data Collection Methods

Data collection methods for system development using secondary data [9]. Secondary data was obtained from a literature study by reviewing several works of literature, reference books, and scientific journals that discussed 3D roguelike games and Procedural Content Generation. For system evaluation, data collection was carried out using a questionnaire [11]. The questionnaire method was used to measure program indicators related to the satisfaction of the playing experience. The questionnaire is given after the game that has been built has been completed. The research subject will fill out the questionnaire.

2.4. Application Development Method

The system development method in this research is the Prototype method[1]. The prototype method is intended to get an initial representation of the game to be made. After potential users evaluate the prototype, the next stage is further development to the production scale.

2.5. *Graph Grammar* Method

Graph Grammars is a method that can be used to generate procedural content [7], [8]. Graphics are better suited than strings to represent missions and spaces in a game, especially when those missions and spaces must have a certain level of need. For example, a completely linear mission (which may be represented by a string) might be suitable for simple and linear games. However, for an exploratory adventure game such as a roguelike or RPG, the game developer would want the mission to contain puzzles and keys, bonus levels, and perhaps multiple paths to complete. They are leading to the final level. A graph can express this type of structure more easily. For example, Figure 1 contains missions that can be completed in two different ways.

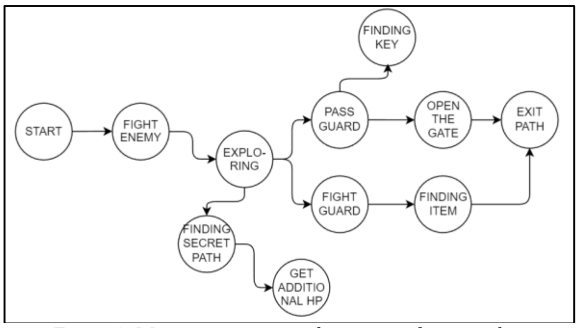


Figure 1. Mission structure with two completion paths

A graph consists of several nodes and edges as links [3]. Graphs are used to describe the missions contained in the game and the routes that players can take. Graph Grammars generate content by changing the basic node structure with another node structure defined as rules. These rules are separated by position into two, namely the left side, or can be called the left-hand side (LHS), and the right side or can be called the right-hand side (RHS). LHS is a node structure that will be searched in the whole graph, which will then be changed, while RHS is a node structure that will replace LHS [7].

The method with graph grammar is constructive, in which the generator will produce only one output, requires a short time, and there is no evaluation of the results. The advantage of the Graph Grammars approach is that the result is controllable. It can convey the narrative and maintain the gameplay. Since the primary mission can be defined in advance, Graph Grammars only increases the variety of the primary mission [7], [8].

2.7. Testing Method

The testing will be done on applications built using the black box. In this black box test, the author will observe the execution results focusing on the functional requirements of the game. This test comprises a series of input conditions to ensure program functionality and find errors that may occur during the development process. The percentage of each tested condition will be calculated by equation (1) as follows [12], [13]:

$$\frac{Number\ of\ successful\ activities}{Total\ activity} x100\% \tag{1}$$

2.8. Player Satisfaction Test Method

The player satisfaction test is carried out to assess the player's satisfaction with the game that has been made [14]. This evaluation uses the GUESS-18 framework. GUESS-18 is a validated game scale with 18 question items to assess 9 (nine) aspects of video games for player satisfaction in playing. The nine aspects are usability/playability, play engrossment, narratives, enjoyment, audio aesthetics, creative freedom, personal gratification, visual aesthetics, and social connectivity [11].

The GUESS rating subscale will be assessed on a 7-point Likert scale. The calculation of the GUESS score consists of calculating the average of the items in a subscale. The overall score is calculated by adding up the scores of all subscales. For the overall score, the minimum score is 9, and the maximum score is 63 [11]. The scale used in this study is:

1) Strongly disagree. The weight for this scale is 1.
2) Don't agree. The weight for this scale is 2.
3) Simply disagree. The weight for this scale is 3.
4) Neutral. The weight for this scale is 4.
5) Quite agree. The weight for this scale is 5.
6) Agree. The weight for this scale is 6.
7) Strongly agree. The weight for this scale is 7.

The results of the calculation process are presented in tabular form, so the value of the feasibility test for the system is obtained. The following equation (2) is for calculating the final score using a Likert scale. Symbol *i* is class interval, *m* is highest score, *n* is lowest score, and *k* is number of classes [15]:

$$i = \frac{m-n}{k} \tag{2}$$

3. RESULT

3.1. Generation Process with Graph Grammars

Graph grammars work very much like language grammars; Graph grammars rules also have a left section that shows a particular graph construction that can be replaced by any of the constructs on the right of the rule. However, to perform the transformation, it is essential to identify each node on the left individually and match it to each node on each right. Figure 2 shows the rules of graph grammars and the use of numbers to identify each vertex.

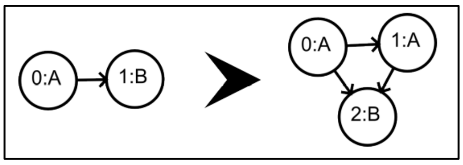


Figure 2. Node Editor for creating a mission graph

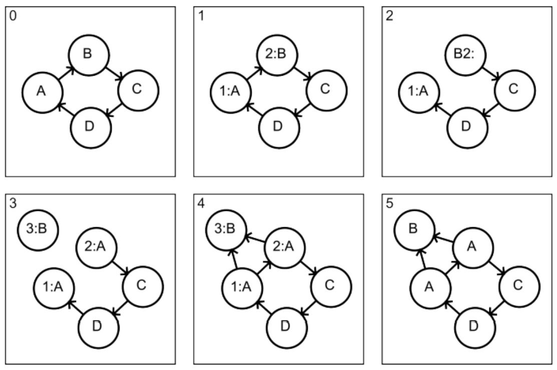


Figure 3. The process of generating graph grammars

In graph grammars, the following five steps are performed to apply rules (as illustrated in Figure 3):

1. Find the subgraph in the target graph that matches the left side of the rule (LHS) and mark the subgraph with a number to copy the node index.
2. Delete all edges between marked nodes.
3. Transform the graph by converting the marked nodes into corresponding nodes on the right-hand side (RHS), adding a node for each node on the right-hand side that does not match the target graph, and removing nodes that do not have a matching node on the right-hand side.
4. Copy the edge as defined by the right side of the rule (RHS).
5. Remove all marks.

3.2. Mission Representation

To apply the graph grammar method, we can start by building a mission representation in the form of letter symbols. The mission representation, which consists of nodes and edges, is as follows:

a. Start (node with symbol s): the starting node where the graph grammar starts generating missions and the player enters the game area.
b. Tasks (node with symbol T): nodes that can be customized according to the needs of the game challenge. This node can be replaced with other nodes such as enemy, key-and-lock, and trap nodes.
c. Enemy (node with symbol e): node with enemies challenge that can attack the player.
d. Items (node with symbol i): nodes with rewards for players to find or acquire.

e. Key (node with symbol k): node with a key to unlock the locked node.
f. Lock (node with symbol l): node with a locked door/obstacle that must be unlocked using the appropriate item key.
g. Trap (node with symbol t): node with the challenge of avoiding traps.
h. Crossroad (node with symbol c): node with four possible paths.
i. Shop (node with the symbol p): the node in which there is an option to purchase items by the player.
j. Boss (node with symbol b): the node where the player fights boss enemies.
k. Goal (node with symbol g): the final node where the player is deemed to have completed the mission.

3.3. Node Editor

Node Editor is a program created as a tool to define the nodes and edges of Graph Grammar in the Unity Editor, making it easier for designers to compose missions in the game. To open the node editor, inside Unity Editor, select Tools -> Node Editor menu. In making Graph Grammar, each node is given a unique ID. Node is divided into several types, namely:

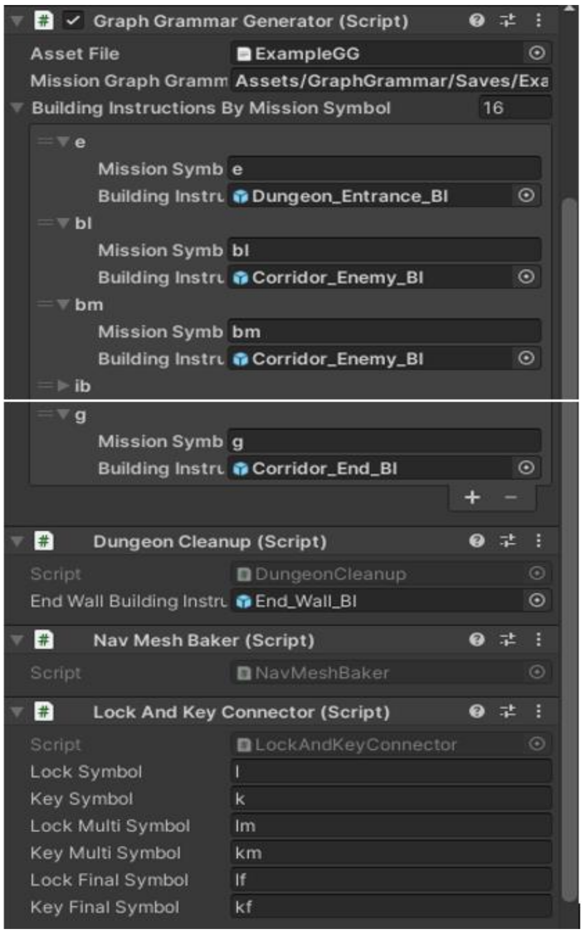
1. Start Graph Node: node with green color is the node as the main graph where the program will read the initial mission defined by the designer.
2. Graph Node: is a node to represent a graph. The implementation of the Graph Grammar method consists of many graphs: the initial mission graph, the reference graph or LHS, and the new mission graph or RHS. LHS and RHS graphs can be connected with red edges to form a rule. This collection of rules will be referred to as grammar.
3. Mission Node: a node represents the mission defined at the design stage. A mission node must have one parent graph and the same mission symbol defined in the design. It can be connected with the green edge to form a series of missions.

After the designer has finished creating a mission with the Node Editor, he can export it to an XML file by selecting the File -> Export -> XML Graph Grammar menu. The exported XML file must be saved in the “Asset/Resources/” folder to include it in the program code when the game is compiled into an APK. A designer can create as many mission files as needed.

3.4. Dungeon Generator

Dungeon Generator is a script program implementing the Graph Grammar method. Dungeon Generator is used in adventure scenes and requires input in the form of an XML file exported from the Node Editor in the previous discussion. Dungeon Generator implementation consists of several components, namely:

a. *Building Instructions* for settings the 3D dungeon type. A 3D dungeon is pre-designed to be prefab, so it is easy to reuse during mission generation. Building Instructions is responsible for storing information about the type of mission (marked with the mission symbol) and the corresponding 3D dungeon. It can also store various layouts of a dungeon to increase the variety of generation results.
b. *Dungeon Clean Up* is a script program in charge after generation with the Graph Grammar method complete to clean up unused connecting points in 3D dungeons. So that the series of dungeons is closed, and the player character does not leave the game arena.
c. *NavMesh Baker* is a script program in charge of making the floor (ground) of a 3D dungeon so that NPC-AI can move in the game arena.
d. *Lock and Key Connector* is a script program in charge of connecting locked mission dungeons with the appropriate key so that the key-and-lock function can function correctly.



```
Graph Grammar Generator (Script)
  Asset File                      ExampleGG
  Mission Graph Gramm             Assets/GraphGrammar/Saves/Exa
  Building Instructions By Mission Symbol      16
    e
      Mission Symb                e
      Building Instru             Dungeon_Entrance_BI
    bl
      Mission Symb                bl
      Building Instru             Corridor_Enemy_BI
    bm
      Mission Symb                bm
      Building Instru             Corridor_Enemy_BI
    ib
    g
      Mission Symb                g
      Building Instru             Corridor_End_BI
                                                 + -
Dungeon Cleanup (Script)
  Script                          DungeonCleanup
  End Wall Building Instru        End_Wall_BI
Nav Mesh Baker (Script)
  Script                          NavMeshBaker
Lock And Key Connector (Script)
  Script                          LockAndKeyConnector
  Lock Symbol                     l
  Key Symbol                      k
  Lock Multi Symbol               lm
  Key Multi Symbol                km
  Lock Final Symbol               lf
  Key Final Symbol                kf
```

Figure 5. Dungeon Generator configuration

Figure 5 is a display for setting the dungeon Generator in the Unity Editor. The components in this setting view were described in the previous section. Designers can set the dungeon generation configuration by filling in the input fields.

3.5. Gameplay Implementation

This section results from implementing the gameplay design into a ready-to-play application. The game is divided into several scenes. Table 1 below is a list of scenes made in the 3D roguelike game that has been built.

Table 1. List of in-game scenes

No.	Scene Name	Description
1	Scene 1 - Main Menu	Scene for the game's main menu
2	Scene 1 - Settings Menu	Scene for game settings menu
3	Scene 2 - Village Area	Scenes for gameplay in the village area
4	Scene 2 - Pause Menu	Scene for game pause menu
5	Scene 3 - Adventure	Scenes for gameplay in the adventure area

1) *Scene* 1 - Main Menu

Figure 6 is the Main menu interface. This main menu will appear when the application is first opened. In this scene, there are three options in the game process, namely the play option, the setting option, and the exit option.

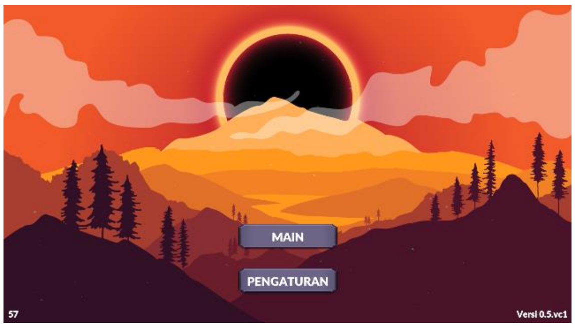
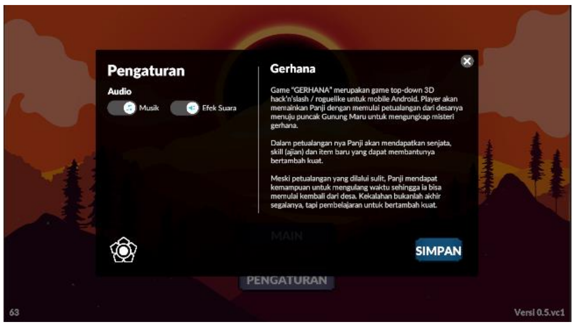


Figure 6. Main menu interface

2) *Scene* 1 – Settings Menu


Pengaturan

Audio

Musik

Efek Suara

Gerhana

Game "GERHANA" merupakan game top-down 3D hack'n'slash / roguelike untuk mobile Android. Player akan memainkan Panji dengan memulai petualangan dari desanya menuju puncak Gunung Maru untuk mengungkap misteri gerhana.

Dalam petualangan nya Panji akan mendapatkan senjata, skill (ajian) dan item baru yang dapat membantunya bertambah kuat.

Meski petualangan yang dilalui sulit, Panji mendapat kemampuan untuk mengulang waktu sehingga ia bisa memulai kembali dari desa. Kekalahan bukanlah akhir segalanya, tapi pembelajaran untuk bertambah kuat.

Figure 7. Settings menu interface

Figure 7 is a display of the settings menu interface. In this settings menu, the player can set several features such as audio consisting of music and sound effects, mode or difficulty level, and other settings.

3) Scene 2 – Village Area

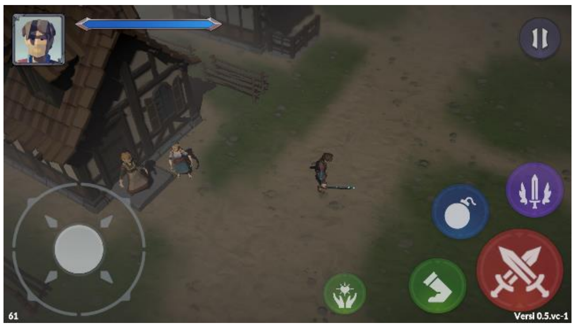


Figure 8. Village area interface

Figure 8 is the interface of the village area in the game. This village area page aims to prepare each character before going on an adventure, such as learning the character's control, learning new skills, interacting with existing NPCs, and buying or upgrading weapons.

4) Scene 2 – Pause Menu

Figure 9 is the interface of the pause menu, which serves to pause the ongoing game. In addition to this menu, there are options such as resuming the game, repeating the current scene, setting options, and exiting options to the main menu.

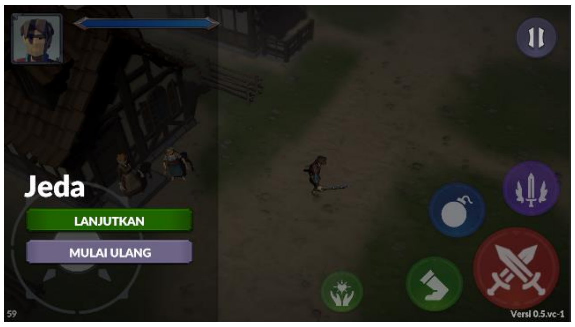


Figure 9. Pause menu interface

5) Scene 3 – Adventure Area

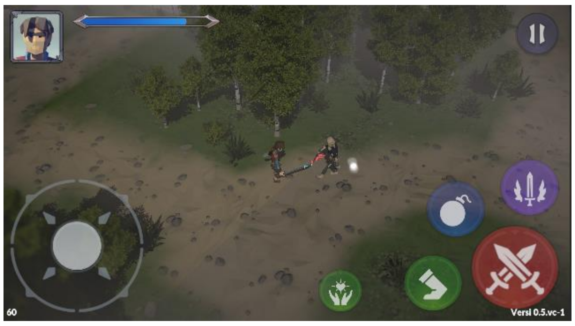


Figure 10. Adventure area interface

Figure 10 is the Adventure area interface where in this area, the player character explores dungeons, defeats enemies, completes an area's objectives, looks for equipment, and survives until he can open the next area.

3.6. Black box testing

Black box testing is carried out to test whether the system developed follows what is stated in the functional specifications of the system. The truth of the software being tested is only seen based on the output generated from the data or input conditions given for the existing function without seeing how the process is to get the output. The test was carried out using the Redmi Note 10S Android device, which has the Android 11 operating system, with an Octa-core Helio G95 CPU, Mali-G76 GPU, and 6GB RAM.

Overall black box testing activities are 48 activities, with all activities successfully implementing the expected design. Therefore, based on equation (1), the results of the black box test can be concluded that the activity test is successful with a percentage value of:

$$\frac{48}{48} x\, 100\% = 100\%$$

3.7. Testing the Application of the Graph Grammar Method

Testing the application of the Graph Grammar method was carried out to determine the success of applying the method to the dungeon generator program that had been built. In this section, a demonstration program will be explained with input in the form of an initial mission graph and compared with the results of the generation by the program.

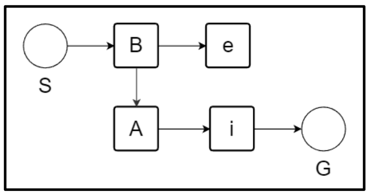


Figure 11. Initial mission graph with 6 nodes

The first test creates an initial mission graph with six nodes, as shown in Figure 11. The player will enter the game arena with the symbol node S in this mission. Then, the player will pass through the area (dungeon) with node B. After that, he can choose a path to the area with symbol e or the area with symbol A. For the player to reach the final destination, namely the node with the symbol G, the mission requires passing through areas A and i. This mission graph created by the game developer can then be generated by the dungeon generator program that applies the graph grammar method.

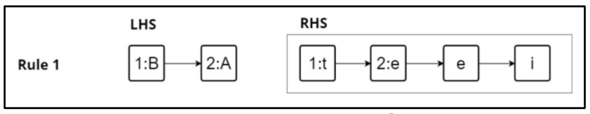


Figure 12. A generation rules

Game developers can create generation rules or rules, as shown in Figure 12. This generation rule consists of an LHS with two nodes and one RHS with four nodes in a straight or linear shape. Creating missions like this can increase the number of challenges or the variety of gameplay. The program will generate a graph using the Graph Grammar method by searching for nodes according to LHS and replacing them with nodes such as RHS. In this test, only one generation rule is used. The result of this generation is a new mission graph with a node structure, as shown in Figure 13. The blue line shows the route the player must take to complete the mission, which is to reach the node with the symbol G.

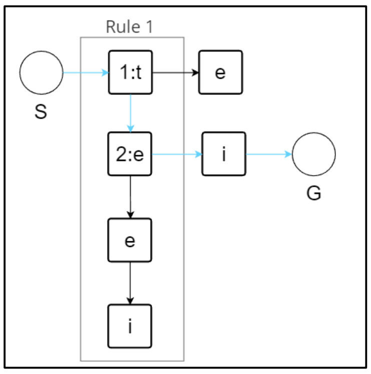


Figure 13. Graph of the new mission generated

The dungeon generator program that has been built will generate the game arena according to the mission structure generated by the graph grammar method above. The game arena includes a 3D dungeon environment, items, enemies, and main characters. The results of this application produce a game arena with variations in the mission structure, as shown in Figure 14, Figure 15, and Figure 16. The three images are screenshots of the scene window in the Unity game engine that the author uses to show the mission structure behind the final result or game the player plays. The image's blue line is used to indicate the path the player can take to complete the mission.

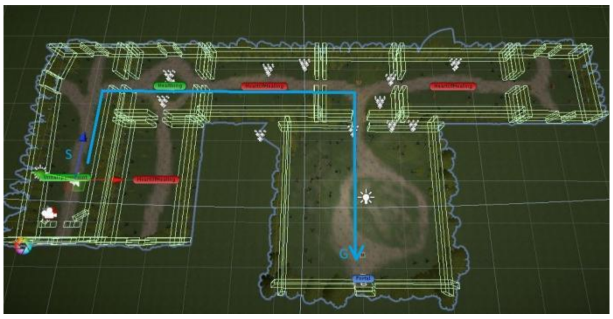


Figure 14. Variation of 1st generation results

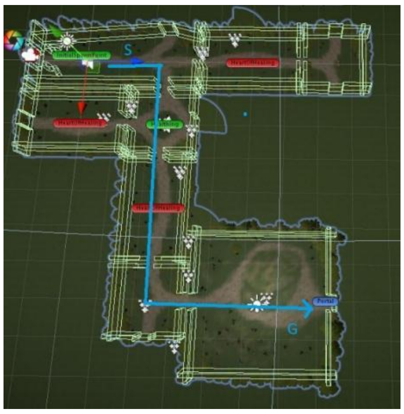


Figure 15. Variation of 2nd generation results

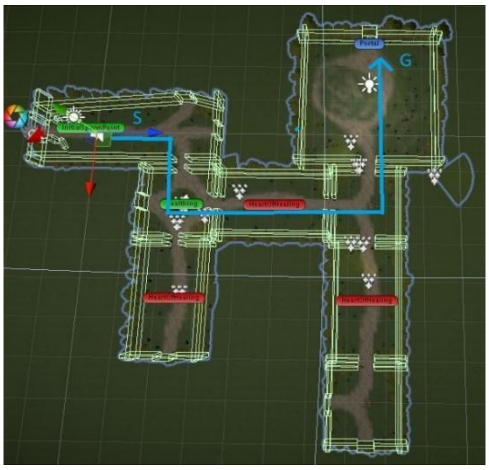


Figure 16. Variation of 3rd generation results

3.8. Play Satisfaction Evaluation

Evaluation of player satisfaction is carried out to get a direct assessment of player satisfaction with the game that has been built. This evaluation uses the GUESS-18 framework. GUESS-18 is a validated game scale with 18 question items to assess 9 (nine) subscale of video games for player satisfaction in playing. The list of question items from GUESS-18 is presented in Table 2 below [11].

Table 2. Question items for GUESS-18

Subscale	Symbol	Question
Usability / Playability	U	Is this game easy to play?
		Is the display of this game easy to understand?
Narratives	N	Is the story at the beginning of this game interesting?
		Did you enjoy the story of this game?
Play Engrossment	PE	Does this game make you feel like entering a fantasy world?
		Does playing this game make you forget the time in the real world?
Enjoyment	E	Is this game fun to play?
		Do you feel bored while playing this game? (Inverted Point)
Creative Freedom	CF	Does this game increase your imagination?
		Does this game make you more creative?
Audio Aesthetics	AA	Did you enjoy the sound effects of this game?
		Do the sound effects and music in this game enhance your gaming experience?
Personal Gratification	PG	Are you very focused on improving the quality of your character while playing this game?
		Do you want to do your best while playing this game?
Social Connectivity	SC	Does this game make it easier for you to communicate with other players in the game?
		Do you like playing this game with other players?
Visual Aesthetics	VA	Did you enjoy the graphics of this game?
		Is the graphic display of this game attractive?

The evaluation was given through a questionnaire distributed to 28 respondents from the "Team Kito" game community after they played the game on their Android devices. The summary of the questionnaire results in Table 3 shows each GUESS-18 subscale score and the overall GUESS score for each respondent. The GUESS subscale scores are calculated by averaging the scores in each subscale. Subscale scores can range from 1 to 7. One item on GUESS-18 has been coded as inverted points (i.e., "Do you feel bored while playing this game" in the Enjoyment subscale). For example, if the questionnaire gets 1 point, the score calculation will be changed to 7 points, 2 points to 6 points, 3 points to 5 points, 5 points to 3 points, 6 points to 2 points, and 7 points to 1 point.

Table 3. Questionnaire Evaluation Results

No	Subscale	Mean	Standard Deviation
1	Usability / Playability	5.660714286	0.707807928
2	Narratives	5.446428571	0.89586794
3	Play Engrossment	5.464285714	1.05346493
4	Enjoyment	5.125	0.898816094
5	Creative Freedom	5.285714286	0.843587702
6	Audio Aesthetics	5.928571429	0.846717909
7	Personal Gratification	6.142857143	0.606403222
8	Social Connectivity	4.339285714	1.563789151
9	Visual Aesthetics	5.678571429	0.807537245
	Total	**49.07142857**	3.245265171

The final evaluation in Table 3 is the result of all participants for each GUESS subscale score and the overall GUESS score. This score is calculated from the summation of the mean of the data entered for all participants in each GUESS subscale and the overall GUESS score. The overall evaluation score of the game that has been built is 49.07 out of 63 maximum scores. After calculating the final score, the assessment category interval will be searched using equation (3.2). It is known that the highest score (m) = 63; lowest score (n) = 9; and many classes from the eligibility category (k) = 4. So with equation (2), we can find the class interval (i), which is 13.5. Then the lowest scale determination is 9.0. Then the resulting assessment category can be seen in Table 4.

Table 4. Rating Category

Interval	Category
49,6 – 63,0	Very Good
36,1 – 49,5	Good
22,6 – 36,0	Fair
9,0 – 22,5	Poor

Based on the rating category, games built with a score of 49.07 are included in the “GOOD” category.

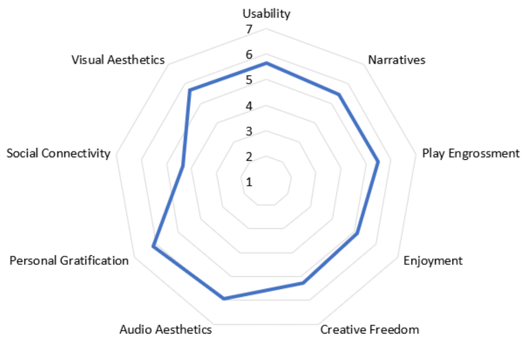


Figure 14. Average points of GUESS-18 evaluation results

Figure 14 is a graphical representation of the average points from the evaluation results using the GUESS-18 framework. The game has a high value of playing experience satisfaction in the subscale of personal satisfaction (personal gratification). The personal satisfaction subscale refers to the motivational aspects of the game (e.g., challenge) that promote the player's sense of accomplishment and desire to succeed and continue playing the game. Meanwhile, the evaluation results have the lowest value of playing experience satisfaction in the aspect of social connectivity. The social connectivity subscale refers to how games facilitate social relationships between players through in-game features and functions. The low score on this subscale is because the games that have been built do not facilitate social relationships like multiplayer features.

4. DISCUSSION

Game developers want variations in the gameplay so that player's paths are not straight or too linear [16]–[18]. Using graph grammars method, game developers make generation rules for mission variatons. In this example, generation rules consisting of 2 rules, as shown in Figure 15 and Figure 16.

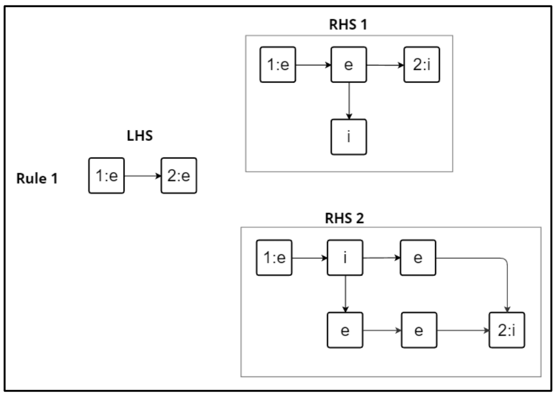


Figure 15. Rule 1

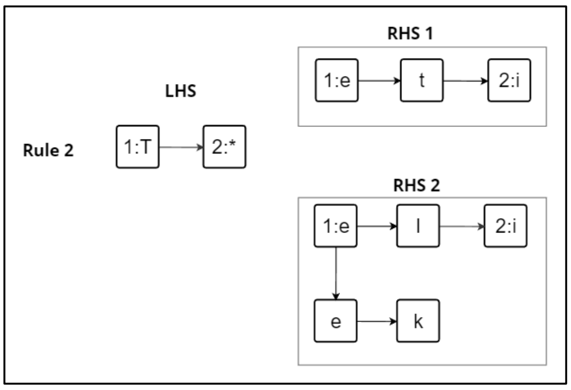


Figure 16. Rule 2

The RHS variation can be composed more according to the requirement of the game developer. In this case, variations are used to add variations to the flow of the game if two nodes of type *e* (enemy) are found close together, see Figure 15. The first RHS variation of the Rule 1 is used to reward the player with an item (node with the symbol *i*) after completing the enemy. The second RHS variation of the Rule 1 is used to increase the difficulty level by increasing the number of enemies and providing additional paths.

The second-generation rule, as shown in Figure 16, adds variety to the gameplay if a node with the symbol *T* is found (task node for missions that require action as described in the section on establishing mission representation).Quite steps in developing the rules in the game have been taken to generate new missions.

Further research is needed to find a better method in applying the method to create new missions that can improve the playing experience [19]–[21] .

5. CONCLUSION

Based on the results of research, implementation, and discussion, it can be concluded that the 3D roguelike game has been successfully built on Android devices with 100% black box testing percentage results from 48 successful activities. Procedural generation of missions using the Graph Grammar method has been successfully implemented, including creating mission graphs using the Node Editor tools in the Unity Editor and configuring generation rules in the dungeon Generator. Procedural generation of missions is used in Scene – Adventure Area, where the player goes through many dungeons with various challenges to complete specific missions. Based on the evaluation of playing satisfaction with GUESS-18, the 3D roguelike game that has been built is included in the "GOOD" category with an overall score of 49.07 out of 63 maximum scores. The game excels in the subscale of personal gratification but is weak in social connectivity.

REFERENCES

[1] T. Fullerton, *Game Design Workshop: A Playcentric Approach to Creating Innovative Games Fourth Edition*. Boca Raton: CRC Press, 2019.

[2] M. Aresa, “Rancang Bangun Game Dungeon Crawler Dengan Procedural Content Generation Menggunakan Algoritma Drunkard’s Walk,” Tangerang, 2021.

[3] J. Togelius, N. Shaker, and M. J. Nelson, *J. Togelius, N. Shaker and M. J. Nelson, Procedural Content Generation in Games: Springer, 2016*. Gewerbestrasse: Springer, 2016.

[4] M. H. Menori, “Procedural Content Generation Untuk Pembentukan Dungeon Pada Permainan,” Bandung, 2020.

[5] G. Smith, “Procedural Content Generation: An Overview,” in *Game AI Pro 2*, Natick, MA: A K Peters/CRC Press, 2015, pp. 501–518.

[6] R. van der Linden, R. Lopes, and R. Bidarra, “Procedural Generation of Dungeons,” *IEEE Trans Comput Intell AI Games*, vol. 6, no. 1, pp. 78–89, 2014.

[7] J. Dormans and S. Bakkes, “Generating Missions and Spaces for Adaptable Play Experiences,” *IEEE Trans Comput Intell AI Games*, vol. 3, no. 3, pp. 216–228, Sep. 2011, doi: 10.1109/TCIAIG.2011.2149523.

[8] B. Lavender, “The Zelda Dungeon Generator: Adopting Generative Grammars to Create Levels for Action-Adventure Games,” Derby, 2015.

[9] U. Silalahi, *Metode Penelitian Sosial*. Bandung: PT. Refika Aditama, 2012.

[10] S. Arikunto, *Prosedur Penelitian Suatu Pendekatan Praktik*. Jakarta: Rineka Cipta, 2006.

[11] J. R. Keebler, W. J. Shelstad, D. C. Smith, B. S. Chaparro, and M. H. Phan, “Validation of the GUESS-18: A Short Version of the Game User Experience Satisfaction Scale (GUESS),” *J Usability Stud*, vol. 16, no. 1, pp. 49–62, 2020.

[12] K. Yuliawan, G. Prayitno, S. Wijono, Y. J. Prasetyo, and S. Trihandaru, “Android-Based Educational Game: Recognition Of Papua Endemic Animals,” *Jurnal Teknik Informatika (JUTIF)*, pp. 889–896, 2022, doi: 10.20884/1.jutif.2022.3.4.319.

[13] K. R. E. Septiani and F. Y. al Irsyadi, “Game Edukasi Tari Tradisional Indonesia Untuk Siswa Tunarungu Kelas VI Sekolah Dasar,” *Jurnal Teknik Informatika (Jutif)*, vol. 1, no. 1, pp. 7–12, Jul. 2020, doi: 10.20884/1.jutif.2020.1.1.11.

[14] M. H. Phan, J. R. Keebler, and B. S. Chaparro, “The Development and Validation of the Game User Experience Satisfaction Scale (GUESS),” *Hum Factors*, vol. 58, no. 8, pp. 1217–1247, Dec. 2016, doi: 10.1177/0018720816669646.

[15] N. Apriyana, “Aplikasi Media Pembelajaran Membaca dan Menulis Aksara Rejang (Kaganga) Berbasis Android untuk Siswa Sekolah Dasar,” Bengkulu, 2019.

[16] A. P. Utomo, “Rancang Bangun Modul Procedural Content Generation Pada Permainan Friend And Foe ,” Surabaya, 2015.

[17] I. N. Kiting, “Implementasi Procedural Level Generation pada Aplikasi Game Pyramid Exploration,” 2020.

[18] M. A. Muslim, E. M. A. Jonemaro, and M. A. Akbar, “Penerapan Procedural Content Generation untuk Perancangan Level pada 2D Endless Runner Game menggunakan Genetic Algorithm,” *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 3, no. 5, pp. 4406–4414, 2019, [Online]. Available: http://j-ptiik.ub.ac.id

[19] C. S. Putri, E. M. A. Jonemaro, and M. A. Akbar, “Penerapan Procedural Content Generation pada Pembangkit Level Gim Maze Heksagonal,” *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 3, no. 9, pp. 8563–8571, 2019, [Online]. Available: http://j-ptiik.ub.ac.id

[20] A. Setyamurti, W. Sukmo Wardhono, and T. Afirianto, "Implementasi Procedural Generation untuk Membangun Level Tactical RPG dengan menggunakan Metode Occupancy Regulated Extension," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 2, no. 8, pp. 2416–2420, 2018, [Online]. Available: http://j-ptiik.ub.ac.id

[21] M. H. Naufal Azzmi, L. Husniah, and A. Sofyan Kholimi, "Island Generator pada Game Open World Menggunakan Algoritma Perlin noise," *REPOSITOR*, vol. 2, no. 7, pp. 965–976, 2020.