Shriek: A Role Playing Game Using Unreal Engine 4 and Behaviour Trees

Savio Rodrigues
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, India
savio1246@gmail.com

Harmanjyot Kaur Rayat
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, India
harmanrayat@gmail.com

Ritson Mathews Kurichithanam
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, India
ritsonmathews@gmail.com

Smita Rukhande
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Vashi, India
smita.rukhande@fcrit.ac.in

2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE) | 978-1-7281-9061-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICNTE51185.2021.9487723

***Abstract*—Shriek is a Role Playing Game which offers an immersive experience to players in a medieval fantasy environment. Here, the player can see stunning magical landscapes and levels created using variety of lighting, meshes and materials. It has a choice based story line, which adds a depth to the choices made and affects the story line further into the game. Player can play with any of the four characters who have their own independent but intertwining stories. It has an attractive GUI which helps the player to navigate through the menus and other options. Shriek combines additionally gives dynamic difficulty and adaptive AI, implemented using Unreal Engine 4's Behaviour Trees, which assesses how well the player is playing, and based on that dynamically adjusts the difficulty of the game and the AI.**

***Index Terms*—RPG, Game, AI, Unreal Engine 4, Behaviour Trees, NPC**

I. Introduction

Role playing games (RPG) are a video game genre where the player has to play and explore the world as a character. This world is developed in a 3D-Space. The lands are filled with lush greens and some riddled with war and have not seen light of day due to everlasting smoke over the battlefield. The residents of the world, live in cities or villages. The player can take on several quests to make their lives better. They can help them make peace or make their home another battlefield.

Exploring the world will help their character to grow by earning points or experience which lets them increase or upgrade their abilities and mature in order to achieve the game's ending. RPGs have been a primary source of gaming entertainment and have been gaining popularity ever since they have been a reality. They allow players to fully engage and immerse themselves in fictional settings of a variety of environments and explore what it has to offer. From collecting herbs to fighting enemies, RPGs in general have a range of missions which the player can complete.

Shriek offers quests in a similar fashion, being set in medieval times, where there is magic and different beings of other origins are present among humans to make you experience a whole new reality. The game focuses on the four main characters, a princess, a thief, a mercenary and a mage. This project is made possible with Unreal Engine 4.

II. Literature Survey

A. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees

Michele Colledanchise, Petter Ögren proposed [1] "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees".The Non-Playable-Characters (NPCs) are characters that are present in the game world but cannot be controlled by the player. They simply exist to fulfill the needs of the story or to help the player learn and practice new moves and skills. These NPCs are controlled by AI .Video games make use of a tool that helps in controlling the structure of in-game components called Behaviour Trees (BTs). BTs make use of a two-way control transfer where the switching between behaviours is done by calculating the parent node's behaviour. A BT is a three tuple:

$$T_i = \{f_i, r_i, \Delta t\} \tag{1}$$

Where i is the index of the tree, f_i is the right-hand side of a difference equation, Δt is a time step and r_i is the return status that can be Running(R), Success (S), or Failure (F). The leaf nodes are Action or Condition nodes and the internal nodes are Fallbacks, Sequences or Parallels.

B. Dynamic Difficulty Adjustment on MOBA Games

M.P. Silva, V.d. Nascimento Silva, L. Chaimowicz proposed [4] "Dynamic Difficulty Adjustment on MOBA Games". This paper talks about how the difficulty setting of a game can impact playability of the game itself while posing a a risk to

978-1-7281-9061-7/21/$31.00 ©2021 IEEE

the players experience. Hence, the need to have a dynamic system that will increase or decrease the level of difficulty based on the player's skill is required. The player's performance $P(x_t)$ at a particular time (t) in game will be determined on the basis of certain parameters in the MOBA game; player's level(H_l), player's death count (H_d), and number of towers destroyed(T_d). It is mathematically given as:

$$P(x_t) = H_l - H_d + T_d \tag{2}$$

Next is to find the difference between performance of time t and (t-1):

$$P'(x) = P(x_t) - P(x_{t-1}) \tag{3}$$

The equation (2) and (3), are used to find out the AI's performance which is denoted as P'(y). Using P'(x) and P'(y) we get the difference between their performances.

$$\alpha = P'(x) - P'(y) \tag{4}$$

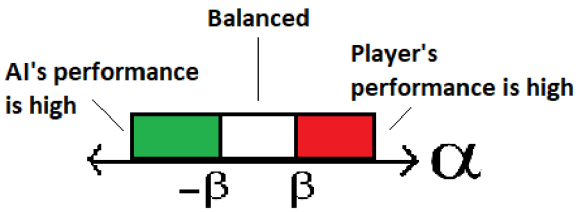

Fig. 1. Performance difference between the AI and the player

Based on the α value, the AI will behave differently. Given Fig 1. shows the expected result when the alpha value changes. If α is high then the AI will be less active, if it is less then the AI will be more aggressive, else there will be no change in their behaviour and that their performance is balanced.

C. Player Choices, Game Endings and the Design of Moral Dilemmas in Games

Nicoletta Tancred, Nicole Vickery, Peta Wyeth, Selen Turkay proposed [5] "Player Choices, Game Endings and the Design of Moral Dilemmas in Games". Storytelling has became an integral part of a game now but sometimes the games do not have enough replay value. To counter this, choices have been introduced in video games which allows the player to have more control over the actions of the character which also affect NPC's. This system allows the player to change the flow of the story and how the game will end.
According to the report, there are two types of narrative endings:

- Linear Narratives: These games provide a single ending only.
- Narratives: These games have several endings which differ according to the player's choices throughout the game.

The choices in a game may be used for different reasons. They are of two types:

- Systematic Choices: These choices are hidden to players. It does not depend upon a choice that is provided to the player, but it depends on what the player does throughout the game. Example: If the player steals too much coin, he will be put in jail.
- Scripted Choices: These choices are provided to the player. The player will be given options to choose from and the outcome of each choice will be scripted.

III. Proposed System

The proposed system shows implementation of different proposed modules and their working. It accounts for the tools used in creation of proposed system. It also explains the architecture of the system and methodology used to implement said system.

A. Tools Used

Developed by Epic Games, Unreal Engine 4 [8] is a game engine which was originally meant for creating first person shooter, RPGs, Platformer, etc. but has also been adopted by other non-gaming projects like movies, architecture and visual design. It provides a high range of portability, by supporting a wide range of platforms like Windows, Linux, Android, iOS and HTML5. It is written in C++ and provides a robust platform for all its users, who can develop using C++ or their proprietary Blueprint based visual scripting.

Unreal Engine 4 is the latest and the most powerful video game creation tool as well as world's most advanced real time 3D creation tool which can be used for creating architecture, cinematography, simulations, in the market today.

B. Methodology

The GUI provides interfaces through which the player interacts with the system. Options such as, ability to load or save the game state and to switch characters at will inside the game are provided. Character movement and abilities allow the player to use certain actions like walking, idle, attacking, etc. to determine its location in the environment and also its interactions with its elements. Inside the 3D playable environment, various levels which can be explored by the player are situated. These levels hold static objects such as trees and houses as well as dynamic objects, like other characters, enemies, the position of the sun and moon over the course of time, etc. The environment is integrated with a responsive system which provides features and utilities depending on the player and their interaction with the environment. If the player is wounded after a battle, a health regenerating consumable spawns near the player to replenish the character's health.

Once the environment is defined, characters can be spawned in it. These characters are not controllable by the player, hence called Non-Playable Characters. They are defined based on their alliance with the player's character and may be friendly, enemy or neutral. Their parameters are modified based on input received by the dynamic difficulty module,

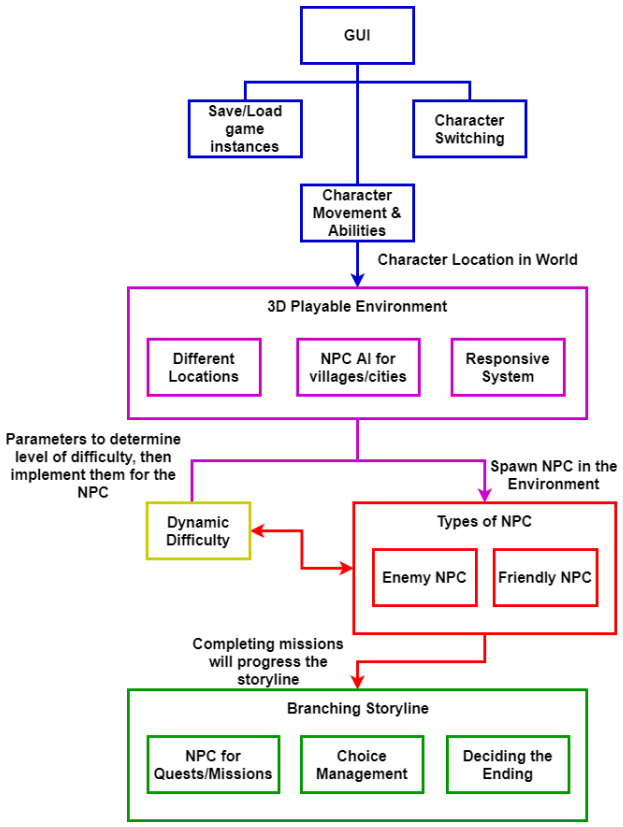

Fig. 2. Architecture diagram for the proposed system

making them easier or tougher for the player to clear through based on their level of skill. After the individual elements of the game are defined with respect to each other, they are integrated with a story created to fit the theme of medieval and magic based fantasy realm. The game offers choices to the player throughout the session, making it different every time the player replays the game, thereby adding a weight of consequence to the choices they make in the game, making it more interesting, session after session.

Fig 2. describes the architecture diagram for Shriek

C. Implementation

1) Creation of Environments and Level Design: Shriek has several levels/environments that the player can explore. Each level is different; ranging from mountain tops to dark and foggy battlefields. They all have different experiences and breath-taking sites to offer. These levels, in Unreal Engine 4 (UE4), are developed using a 3D virtual-space referred to as the viewport. Each level has a variety of lighting to illuminate the environment. The level's sky and atmosphere is created and expressed using SkySphere and SkyAtmosphere. meshes like trees, rocks, grass, etc. are to be populated on a large scale in the level. So Foliage is used to mass populate any level. Objects such as cubes or spheres are dragged into the viewport and given characteristics such as scale, rotation and location. Often, materials are applied to these objects to alter their look and feel. However, we can not only place cubes or spheres, but anything that has been created using a 3D modeling software can also be placed inside the viewport. These 3D models are called Meshes.

Meshes are of two types:

- Static Meshes - They cannot change their scale, rotation and location in the game.
- Dynamic Meshes - They can change their scale, rotation and location in the game.

a) Creation of Lights: Every level needs some amount of lighting. If there is no light, even if there are meshes in the level they will not be visible to the player.

UE4 has five light types:

1) Directional Lights - They are the primary source of the outdoor light. They act as the sun.
2) Sky Lights - They capture lights in the background and apply it to the level's meshes.
3) Spot Light - They emit light from a source and travel in a cone.
4) Point Light - They act like a classic 'light bulb'.
5) Rect Light - They emit light from a rectangular plane. e.g: light from a television screen.

b) Creation of Materials: Materials are assets that are applied to a mesh to alter its visual look. They are made up of several textures and images. These textures can also be given several characteristics. Every mesh in the game has a Material applied to it. A Material can be transparent, opaque, metallic, solid, rough, etc.

c) Implementation of Sky Sphere and Sky Atmosphere: A Sky Sphere or Sky Box is a huge spherical mesh, that is used to portray the sky in a manner similar to the real world. It contains a sky-Material that has clouds, stars and even the moon. The sky-Material which changes based on the in-game time. It will be an orange hue when the sun is near the horizon, bright blue when it is afternoon and dark when the sun sets. Fig 3. shows how the SkySphere is applied to the level.

A Sky Atmosphere is a component in UE4 that allows the light in the 3D-space to scatter off of surfaces and give a more realistic feel to the environment. It works in tandem with the Directional Light to simulate absorption with Mie and Rayleigh scattering.

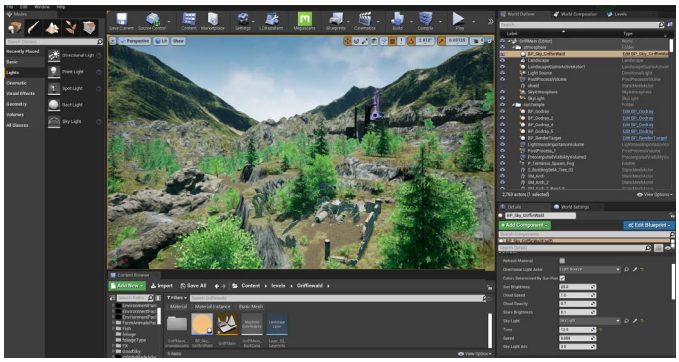

Fig. 3. SkySphere parameters and representation in 3D environment

d) Creation of Foliage: Foliage is a set of Static Meshes placed in the level. They are generally used for grass, trees and stones. These meshes cannot be placed separately on the level, hence Foliage Tool is used to place them in the level. Fig 4. shows how foliage is used to populate a level.

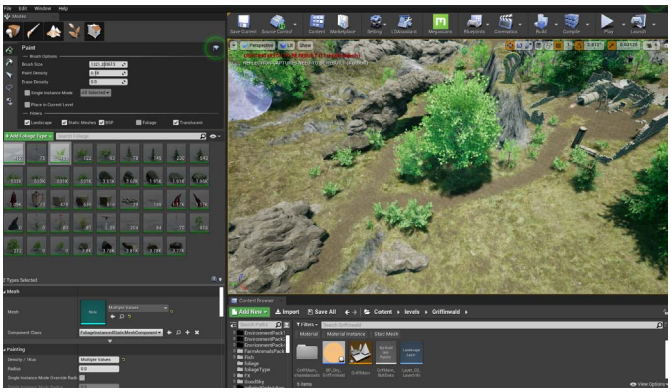

Fig. 4. Environmental foliage

2) Development of Characters and Locomotion: Shriek has four playable or player controllable characters. Each character has a set of actions they can perform. They are basic environmental interaction, primary and secondary attack and defense respectively. In Unreal Engine 4, each character has a player blueprint, which is a file which holds code and logic for anything the character might perform in the game. From walking to dying, every possible action is coded for in this file. To supplement this, the player blueprint file has an animation blueprint file, which holds code and logic for animations and their transitions between one set of movement to another. This files work in sync and have to be linked together.

a) Player Character Blueprint: This is the main blueprint file, which contains the 3D character model, its size and scale specifications and other logic needed to represent the character in a 3D environment. The file also tells the character on what and how to perform a certain action, based on events. Events can be like *onGameStart*, which triggers when the game starts, or *onEventTick*, which triggers for every frame, etc. Custom events can also be created to indicate character states like if the character is in the air, or if the character is dead, etc. The blueprint allows to add extensions the the 3D model using any socket predefined in the 3D character's skeleton. Extensions like swords, staff, armour and so on are also added here itself.

b) Character Animation Blueprint: This blueprint file contains logic necessary to transition the character from one set of animation to another. At any given character state, the character is said to performing an action. By default, it is the Idle state. If the character is moving, it is said to be in the Moving state. With the help of character animation blueprint, we can transition the character from one state's animation to another. When this blueprint is linked to the main player blueprint, it has access to its variables and states as well. Hence, it can detect change of states and perform animation transitions, making it smoother and more natural to observe a 3D character model change states. Given Figure 5 below shows how different states are connected together.

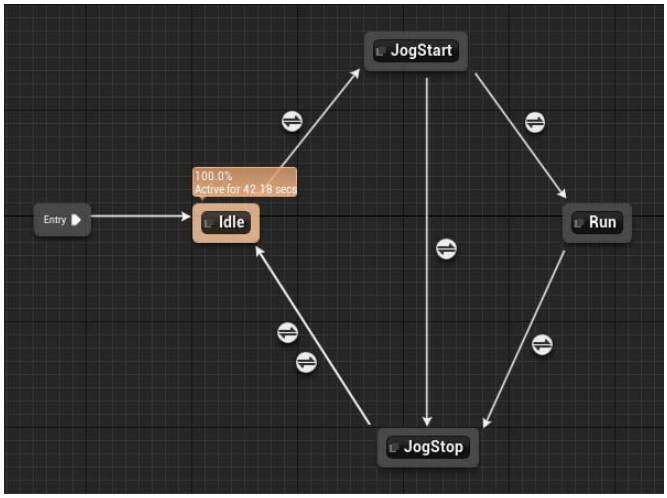

Fig. 5. Player character animation states present in an animation Blueprint

c) Animation File: This file contains a frame by frame information of the character's position in 3D environment. Animation files like these are linked together in the character animation Blueprint to give better transitions.

d) Blend Space File: A Blend Space file is a special file which allows to interpolate between a set of individual animations based on certain character blueprint values. When it is used inside of a character animation Blueprint, it has access to the player's global state values. Using these values, the blend space file can interpolate between animations, making them appear more fluid in movement and realistic.

3) Creation of Quest System and UI Tools: A role playing game's quest system is an important tool to store and display data relevant to missions, quests and additional information such as lore and puzzle information. The player interacts with a widget which contains different tabs for current, completed and failed quests. Inside each tab, the status and additional description related to a quest is available. Upon accepting a new quest, it is added to the quest system widget, which will then display this new quest as 'Active'. If it is not accepted then it will not show in the widget.

a) Quest Actor Blueprint: A quest Blueprint simply holds the information regarding a quest which the player will undertake in the game. Data like name of the quest, type, can be rejected or not, reward upon completion, type of quest, is completed or not, etc. is stored for each quest. This data would then be used to create quests and managed in the quest manager.

b) Quest Manager Actor Blueprint: The quest manager Blueprint is the main file and contains logic on how to process a quest when it is added, when it is currently active, completed or when the quest has failed. This Blueprint is linked to each player character's Blueprint and interacts with the quest widget to display these quests based on their state of completion.

c) Quest Widget Blueprint: The quest widget is a visual UI display of quests which the player can read and click on. It takes in data from the quest manager Blueprint and displays them accordingly. the widget can be displayed or hidden by a set keyboard key binding.

4) Non Playable Characters: Non playable characters (NPCs) are 3D modelled characters which cannot be controlled by the player. They can interact with the player characters and can give quests, missions, general conversations and more upon interaction. NPCs are spread throughout the map and are mainly used to fill up an environment with characters other than the players. They make the game more engaging and the level of realism given to them directly translates to the level of quality of the game.

NPCs are of the following types:

1) Interactable NPC: These can interact or be interacted with to provide some sort of quest or information in the game which adds to the game's main content.
2) Non Interactable NPC: These cannot be interact with and are present as just another part of the environment.
3) Object based NPC: These NPCs are not living beings, but provide a similar function to that of the interactable NPC.
4) Objective oriented NPC: These NPCs are of type 1 or 3, but disappear after their assigned quest or mission or reward is completed/collected.

5) Artificial Intelligence: Artificial Intelligence is used to provide a sense of realism to NPC. It gives depth to character's presence in the environment. The richer and more complex a game is, the more skills and abilities it requires. Thus, creating a truly smart and fully autonomous agent for a complex video game can be as challenging as replicating a large part of the complete human intelligence. This gives a real sense of being, feeling and experiencing a totally different world.

AIs are of the following types:

1) Friendly NPC: These AI aid and provide support to the player character in game.
2) Neutral NPC: NPCs come under this category
3) Enemy NPC: These AI try to deal damage to the player character. The player has to fight against this type of AI throughout the game, to complete it.

An AI is created by integrating behaviour trees file and state variables holder called Blackboard file. The behaviour tree file is then linked to an AI Controller Blueprint. Finally, the AI Controller Blueprint is linked to a character Blueprint.

a) Behaviour Tree Blueprint: This file consists of a tree, which can be used to manipulate and define the flow of actions an AI must make based on certain conditions. The conditions are checked with correspondence to the variables provided by the Blackboard file. Given Figure 6 below shows a behaviour tree representation in UE4.

b) Blackboard Blueprint: This file holds variables which can be changed or updated or checked against in the behaviour tree. While the behaviour tree executes logic, blackboard is used to store information.

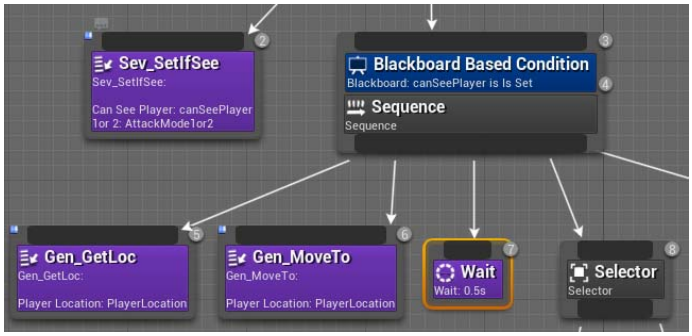

Fig. 6. An AI behaviour tree representation

c) AI Controller Blueprint: AI Controller is a special type of Blueprint, which executes specific tasks based on events. It is used primarily to execute the behaviour tree file when an event takes place. On top of that, it can also set blackboard variables based on events. This Blueprint file is then linked to the NPC character's Blueprint so that the character can perform tasks based on the behaviour tree state.

6) Implementation of Dynamic Difficulty: To provide a challenging experience, enemies are crafted with high skill-set, making it difficult to clear levels, but if the enemy is too easy, it makes the experience meagre and uneventful. To tackle this, games add a difficulty setting at the start of the game. This allows players to set their desired difficulty, but it cannot be changed later on. This makes the game more difficult if the player cannot cope with the selected difficulty or too easy if a lower difficulty is selected.

Dynamic difficulty assesses how well the player is playing and accordingly adjusts parameters pertaining to enemy AI settings. This is updated upon each interaction between the player and enemy NPC. If the player clears through levels quicker and better, then the difficulty is increased for the next level, and vice versa.

Parameters such as time taken to clear level, health left, combo count, previous encounters, etc. are taken into account.

IV. Experimental Results

Character based abilities such as movement and locomotion work based on the input received from the player is shown in Fig 7. Switching between character also works smoothly without any reduction in system performance, as shown in Fig 8.

Fig. 7. A Player Character in 3D Environment

The Quest System widget and GUI as shown in Fig 9 and 10 are easily interactable through set of keys and mouse clicks. It displays quests based on their state of completion and also any related description.

Fig. 8. Implementation of Character Switching

Fig. 9. Quest System Widget

Fig. 10. Main Menu GUI for Shriek

V. Conclusion

Shriek ticks off many of the boxes required by role playing games and even adds more extensive functionalities like dynamic difficulty and multiple playable characters. Shriek gives player extensions like swords, magic projectiles to interact and combat enemies. It also showcases a quest system which allows them to view and manage quests.

On top of it, the game offers a branching story-line which is affected by the choices the player makes throughout the game, there by providing a variety of gameplay elements and ending prospects to experience. Features like these enable the player to play the same game more than once, which is a clear indicator of its quality. This in turn adds to the game's replayability factor and entertainment value.

References

[1] M. Colledanchise and P. Ögren, "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees," in IEEE Transactions on Robotics, vol. 33, no. 2, pp. 372-389, April 2017.

[2] Tom Wijman, *The World's 2.7 Billion Gamers Will Spend $159.3 Billion on Games in 2020; The Market Will Surpass $200 Billion by 2023*, Newzoo, May 8, 2020. [Online]. Available: https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/

[3] R. A. Boyd and S. E. Barbosa, "Reinforcement Learning for All: An Implementation Using Unreal Engine Blueprint," 2017 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, pp. 787-792, 2017.

[4] Silva Mirna, Silva Victor and Chaimowicz Luiz, "Dynamic Difficulty Adjustment on MOBA Games", Entertainment Computing. 18. 10.1016/j.entcom.2016.10.002., 2016.

[5] Nicoletta Tancred, N. Mcmahon, P. Wyeth and S. Turkay, "Player Choices, Game Endings and the Design of Moral Dilemmas in Games," in Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts, pp. 627–636, Oct. 8, 2018.

[6] Xue Su, Wu Meng,Kolen John, Aghdaie Navid and Zaman Kazi, "Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games," in WWW '17 Companion: Proceedings of the 26th International Conference on World Wide Web Companion 2017, pp. 465–471, 2017.

[7] Frankie Wallace, *Mind Games: How RPGs Can Benefit The Mind*, Headstuff, Jan. 4, 2019. [Online]. Available: https://www.headstuff.org/entertainment/gaming/how-rpgs-can-benefit-the-mind/

[8] Tom Shannon, "Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings." ISBN: 9780134680767, 2017

[9] Patrick Allan, *The Surprising Benefits of Role-Playing Games (and How to Get Started)*, Lifehacker, Sep. 2, 2015. [Online]. Available: https://lifehacker.com/the-surprising-benefits-of-role-playing-games-and-how-1684582789