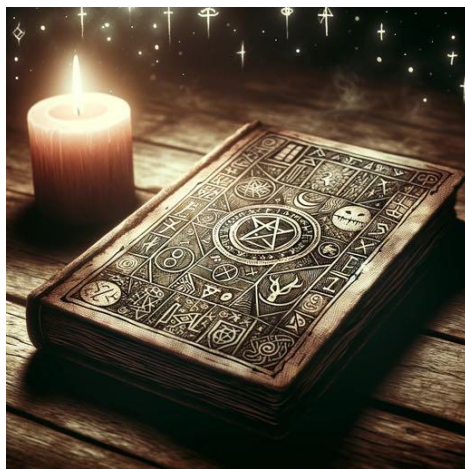


Software Engineering Department

Braude College

Capstone Project Phase B – 61999

TaleWeaver



24-1-D-40

Advisor: PhD. Sulamy Moshe

MosheSu@braude.ac.il

Students:

Nitsan Maman – Nitsan.Maman@e.braude.ac.il

Ron Shahar – Ron.Shahar@e.braude.ac.il

<https://github.com/Seth7171/TaleWeaver>

Table of Contents

Abstract	3
1. Project Description	4
2. Architecture Overview	5
3. Activity Diagram	6
4. Development Process	8
5. Challenges Faced	10
6. Tools Used During Development	12
7. Results and Conclusions	13
8. Lessons Learned	14
9. Did We Meet the Project Criteria?	15
10. Client Interaction During Development	16
11. Testing Process	18
12. User Guide	20
13. Maintenance Guide	31
14. References	38

Abstract

In the realm of digital gaming, engaging the younger audience has become increasingly challenging due to a decline in interest towards traditional storytelling methods. "TaleWeaver" emerges as an innovative solution, leveraging the Unity engine to craft a dynamic, AI-driven narrative experience. This project integrates advanced AI storytelling and text recognition to adapt the game environment in real-time, offering a unique, interactive story with each playthrough. By prompting players to input their desired adventure, the game generates personalized story pages, each presenting a distinct dilemma or event, enriched with AI-generated images and text. This paper delves into the development process of "TaleWeaver," highlighting its use of the DALL·E and ChatGPT 4 APIs for image and text generation, respectively. It also explores the game's approach to thematic consistency through keyword analysis and background adaptation, providing a seamless, immersive experience. "TaleWeaver" not only represents a significant advancement in personalized gaming but also sets a new standard for incorporating AI in interactive storytelling.

1. Project Description

TaleWeaver is an AI-powered, interactive storytelling game that is taking place in books. built in Unity, aimed at offering players unique, personalized adventures. It is designed to address the declining interest in traditional book reading and the static nature of many game narratives. TaleWeaver gives players the ability to shape their own stories while maintaining immersive gameplay.

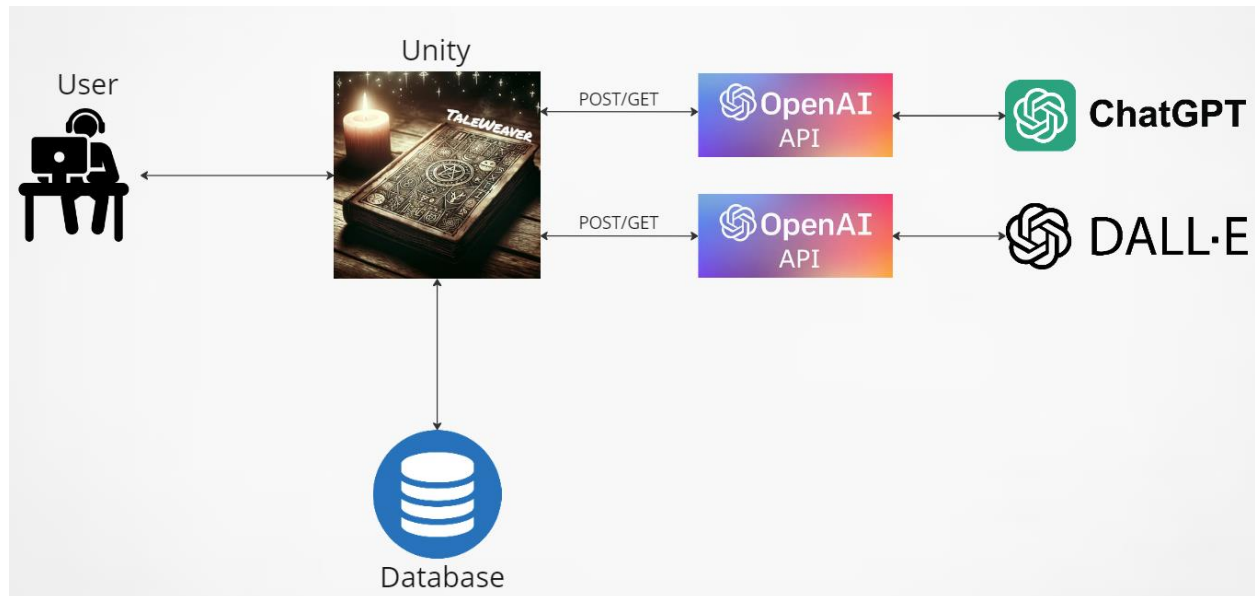
At the core, the game does not rely on the AI alone for story creation. Instead, the AI works in collaboration with game scripts and mechanics, receiving specific commands about the next gameplay mechanic and difficulty level. For instance, when a combat encounter or skill check is needed, the AI receives instructions on what elements to include, ensuring consistency with game rules and player progression. The AI-generated story becomes the foundation, while the game mechanics layer on top of it, integrating seamless interactions like choices, skill checks, and luck-based events.

This game caters to fans of interactive fiction and gamers seeking replayability. Its blend of narrative depth and dynamic world interaction appeals to both casual and dedicated players. Players navigate a 10-page adventure, where each page offers unique encounters. These encounters include dice rolls, combat, riddles, skill checks, and choices, with an overarching mechanic of player-driven decisions. The final encounter is a boss battle, designed to test the player cumulative choices and skills from the adventure.

The RAKE algorithm extracts keywords from the AI-generated narrative, shaping the dynamic game world. This ensures that environments adapt based on story developments, providing immersive and varied experiences.

The AI doesn't function independently. The game provides instructions for each encounter, directing the AI on mechanics like combat difficulty, skill checks, and life loss scenarios. The parser in the game processes this information, ensuring that the gameplay is tightly woven with the generated story. This creates an experience where AI and game mechanics operate in harmony, with room for further expansion and enhancement.

2. Architecture Overview



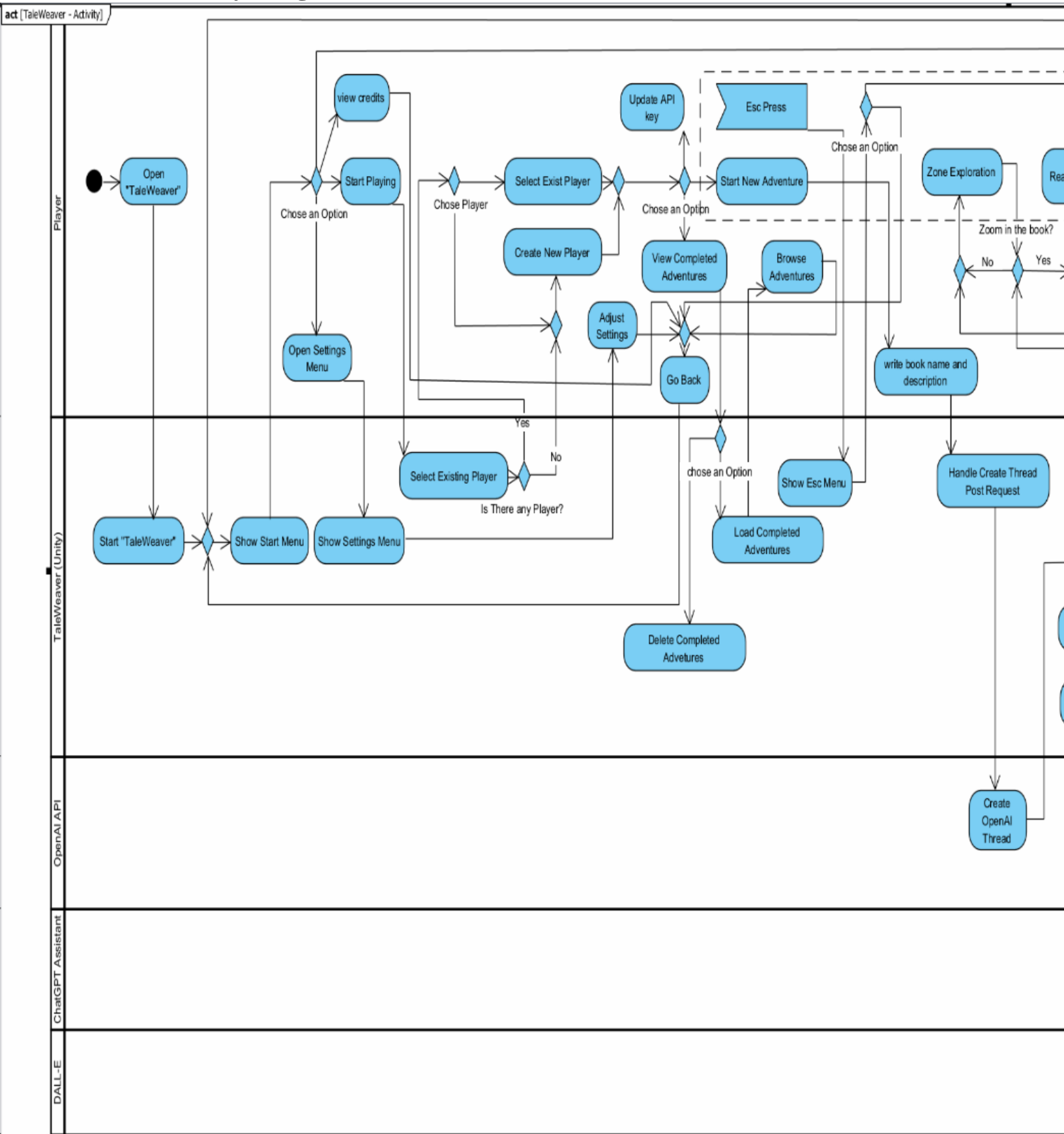
User Interaction: Players interact with the game through a Unity-based interface, where they engage with dynamically generated stories.

Unity Application: The core game logic resides in Unity, which handles the player's actions, story progression, and interactions with external APIs.

OpenAI Integration: The game utilizes OpenAI's APIs to generate story content using ChatGPT and create visual elements using DALL·E. Unity sends POST/GET requests to these APIs to fetch narrative and graphical content.

Database: All player data, including previous adventures and game state, are stored in a database that the Unity application accesses.

3. Activity Diagram





4. Development Process

We began our project by familiarizing ourselves with the Unity engine, creating a basic 2D space shooter game and a vampire bullet-hell game to understand the fundamentals. Following that, we developed the main menu, player selection screen, and set up a basic database.

The core development process included:

1. **Interface and Interaction:** We built the main menu, pause menu, player selection, and book creation screens, ensuring a smooth user experience. Music integration and screen adjustments were also addressed.
2. **Game World and Mechanics:** We implemented the game world, including player movement, book and book animations, and the health system. As we progressed, we integrated the OpenAI API, allowing dynamic story generation using ChatGPT.
3. **AI and Story Integration:** We connected the OpenAI API to the game, enabling real-time story generation. We also added features like viewing previous adventures and selecting options within the game world.
4. **Mechanics and Game Logic:** We integrated various game mechanics, including dice rolls, combat, skill checks, riddles, player choices and luck. The RAKE algorithm was implemented to dynamically adjust the story's location and content.
5. **Bug Fixes and Enhancements:** Throughout the development process, we addressed numerous bugs and added features like epilogues, death screens, and a warning system for returning to the main menu. We also optimized singleton instances and ensured all mechanics functioned seamlessly.
6. **Final Integration:** We completed the full integration of all game mechanics, ensuring they worked correctly with the loading screen and UI buttons. The final stages involved refining combat difficulty, adding item bookmarks, Reroll mechanics, revamping of the view previous adventure section and implementing health and skill modifiers.
7. **Adding the Environments:** We added Seventeen game worlds for the player to explore while in the game world. The RAKE algorithm will extract the location keyword from the story and then the player will be teleported to the correct environment.

8. **Adding the Tests:** To ensure smooth gameplay, we implemented several unit and integration tests throughout the project. These tests covered mechanics such as player choices, combat outcomes, and skill checks to verify the functionality of core game features. We also tested the RAKE algorithm to ensure it extracted accurate keywords, and that the correct environments were loaded in response to story elements. Additionally, we verified the API communication by testing both successful and failed responses, ensuring the AI-generated narrative was consistent and reliable.

5. Challenges Faced

1. **Data Management from OpenAI API:** The challenge was to save and read data from the OpenAI API in-game. We solved this by creating an advanced parsing script that handles the saved JSON files, ensuring seamless data retrieval.
2. **Book Animations and Page Flipping:** Creating a visually appealing book with animations and dynamic page content was challenging. We addressed this by using Unity cameras, each assigned to a page, and copying mechanics across pages. Each page contains numerous game objects, and the camera setup allowed smooth transitions.
3. **UI Button Integration in the Game World:** Integrating UI buttons into the book in the game world was difficult. We solved this by implementing a fade-in UI element that appears only when zoomed in and fades out when zoomed out, ensuring a seamless user experience.
4. **Expanding Gameplay Mechanics:** The game initially lacked diverse gameplay mechanics. To enhance gameplay, we designed mechanics like roll, combat, skill checks, luck, player choices, and riddles. Additionally, we expanded beyond health stats by introducing buffs, reroll stats, and items.
5. **API Costs:** The high costs of API calls during testing were a concern. We mitigated this by saving six different mechanics pages and using them for testing. When the AI generates a specific mechanic, it mimics the saved page, reducing API usage.
6. **Asset Costs:** We needed affordable assets for the game world. We solved this by purchasing a budget-friendly Humble Bundle of Unity assets, enabling us to build the game world within budget.
7. **Reliability of AI Responses:** Ensuring that the AI adhered to specific templates was crucial, as deviations could break the game. We developed intelligent parsing techniques to extract the required information, regardless of the API's output, ensuring consistency.
8. **Interaction with the Game World:** Designing engaging interactions within the game world was a challenge, and due to time constraints, we couldn't fully solve this problem.
9. **Learning Unity Testing:** Understanding how to effectively test within the Unity engine was another challenge, requiring us to familiarize ourselves with Unity's testing environment and tools.
10. **Unfinished Item System:** Due to time constraints, the item section of the game remains unfinished and unimplemented.

11. Game Crashes due to Camera Rendering: One of the major challenges we faced after we implemented the 17 game worlds was persistent game crashes after completing three pages of the in-game adventure book. The crashes were traced to the camera system—each camera attached to a page in the book was rendering the entire game world, leading to extremely high GPU usage. This overload caused the game to crash. We resolved the issue by optimizing the camera system. Each page's camera was configured to render only its corresponding page content, significantly reducing GPU strain and preventing further crashes.

6. Tools Used During Development

1. **Unity:** Unity was the primary development platform for the game, providing an integrated environment to design, build, and test our game. It allowed us to implement game mechanics, manage assets, and create the overall gameplay experience.
2. **Photoshop:** We used Photoshop to design the art and UI elements of the game, ensuring a visually appealing and consistent look across all screens and elements.
3. **Postman:** Postman was essential for testing and validating the API interactions. It helped us understand the expected inputs and outputs, which were crucial for integrating the OpenAI API into the game.
4. **OpenAI Assistance API Webpage:** This tool was used for developing and testing the OpenAI scripts outside of Unity. It allowed us to fine-tune the API interactions and ensure the responses met the game's requirements.
5. **Git / Git LFS :** We used git to share and work on the game and upload updates safely to GitHub. We also used Git LFS to upload large files to GitHub.
6. **GitHub:** GitHub served as the version control system, facilitating collaboration among the team. It safeguarded our code, tracked progress, documented bugs, and enabled effective team communication.
7. **Visual Studio:** Visual Studio was the IDE used for coding the C# scripts. It provided us with debugging tools and an efficient coding environment to implement game logic and mechanics.
8. **Unity Asset Store:** The Unity Asset Store was used to download free assets, such as dice models, which were integrated into the game to enrich the gameplay experience.
9. **Inno Setup:** We created the Installer of the game using Inno Setup.
10. **NUGET NUnit testing:** We used NUnit for the testing of the game.

7. Results and Conclusions

Overall, we successfully achieved the primary goals of our project, though the Item System remained incomplete, as it was a late addition and not part of the original plan. We exceeded our initial scope by implementing additional gameplay mechanics such as dice rolls, a luck system, and conclusion pages for each adventure.

Decision-Making Process: Our approach was guided by a "fun-first" philosophy. We consistently asked ourselves if the game was enjoyable, exciting, and engaging for testers. Decisions prioritizing gameplay features such as roll, combat, and push-your-luck mechanics were influenced by user feedback, which indicated a need for more gameplay variety.

In addition to prioritizing fun, we aimed for simplicity over complexity, especially in the UI design. We wanted to keep the interface contained within the book, ensuring that players felt fully immersed in the game's world and story. Simplification was also applied to gameplay elements where possible, focusing on what would enhance the experience without overwhelming the player.

9. Lessons Learned

Our development process was effective, but there are several areas where improvements could be made. One major issue was with the BookLoader script, which became overly complex as it managed all components for every page. In hindsight, we should have split the BookLoader into individual loaders for each mechanic, with a parent loader to manage them. This would have simplified our codebase and made future updates easier.

Similarly, the GameMechanicsManager script housed all the game mechanics, which led to a cumbersome design. Separating each mechanic into its own script would have made the code more modular and maintainable.

Another mistake was creating ten separate pages for the book, each with its own components. This design choice was inefficient, leading to repetitive work and difficulty in implementing new features. A better approach would have been to create a single page template that could be reused and modified as needed throughout the game.

On the positive side, we're satisfied with the game's data structure, particularly how we organized and stored user data and books. This structure is well-organized and allows for easy expansion in the future.

In the OpenAIInterface Script, we created two separate routines for handling API requests from OpenAI and DALL-E. These routines are used in different contexts: one is triggered in the new book scene, and the other is in the game world (for existing books). Unfortunately, this led to duplicated code for these operations. In hindsight, it would have been more efficient to create a unified routine that handles both scenarios, reducing redundancy and simplifying future maintenance.

9. Did We Meet the Project Criteria?

Based on the criteria we set for "TaleWeaver," we met most of our objectives, successfully implementing key features like AI integration, gameplay mechanics, and an engaging visual design. However, we did fall short in a few areas:

1. **FR7:** We were unable to dynamically alter the sound based on extracted keywords due to time constraints.
2. **NFR4:** The game interface was not always intuitive. Test subjects sometimes struggled to understand the objective, indicating a steeper learning curve than intended.
3. **Motivating Reward System:** We did not implement a reward system, which was initially part of our success criteria, again due to time limitations.

Despite these shortcomings, all other areas, including game scalability, stability, and innovation in gameplay mechanics, were successfully met.

10. Client Interaction During Development

During the development of TaleWeaver, client interaction played a crucial role in shaping the game's final form. Initially, the gameplay mechanics were limited and didn't offer the engaging experience we hoped for. To gather feedback, we shared the game with 20 peers and colleagues, allowing them to test the initial build, which had only the core mechanics—referred to as the "options" mechanic. The feedback we received emphasized that the gameplay was lacking in depth, prompting us to spend an additional month developing five new mechanics: combat, roll, luck, riddles, and skill checks.

We then created seven separate builds, each focusing on a single new mechanic. These included the original build, as well as separate builds for combat, roll, and other mechanics, allowing us to assess each mechanic's individual impact. Afterward, we sent these builds to different testers and gathered their feedback. The most popular builds were those that included the combat, luck, and options mechanics, but the final build, which combined all six mechanics, proved to be the most fun and engaging overall.

User feedback also influenced the game's design choices, such as adjusting the color of the "new player" screen from purple to orange for better visual appeal. Additionally, based on player input, we added gender-specific voices for when characters took damage, further enhancing player immersion.

In addition to the feedback we received on gameplay mechanics, we engaged with testers throughout the development process to gather insights on other aspects of the game. For instance, we noticed that some users had difficulty understanding the interface and objectives of the game, which led us to implement clearer instructions and tooltips.

Our client also suggested that the game's balance could be improved, particularly in terms of the difficulty level of encounters. We tweaked the combat mechanics and random events to ensure that players felt challenged without being overwhelmed. Additionally, we fine-tuned the interactions with AI-generated content, making sure that narrative transitions felt seamless and responsive. Another area where feedback was instrumental was in refining the audio experience. While we had initially planned for minimal sound effects, testers emphasized how immersive sounds—like page-turning, background ambiance, and unique audio cues—would enhance the overall experience. Consequently, we expanded the game's sound design to include these elements.

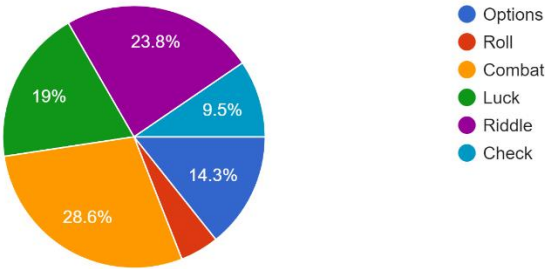
10.1 User Feedback and survey

While writing this book, we play tested the game and asked 20 players to provide feedback on various aspects of the game. Here are the results of the survey:

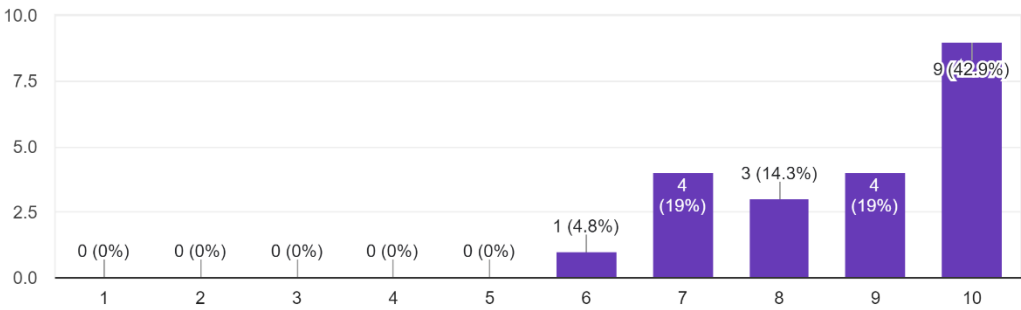
https://docs.google.com/forms/d/10wkUwin59e_q8PAXzCPYpQNTTnKrEw22Uilw0l2peNQ/viewanalytics

Here are some pictures of the survey :

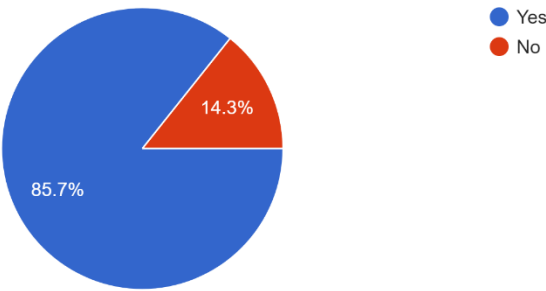
Choose the best mechanic in the game
21 responses



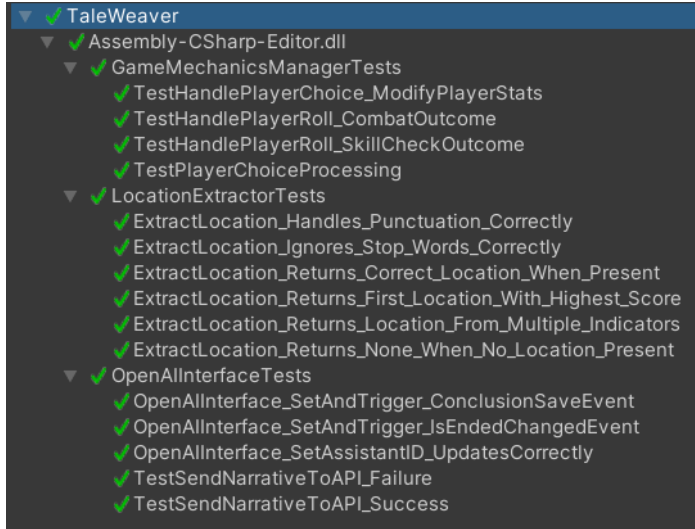
How fun is the game?
21 responses



Did you find the AI-generated story to be coherent?
21 responses



11. Testing Process



Test Number	Test Name	Description	Status
1	TestPlayerChoiceProcessing	Check the whole mechanics process, from player choice to assignment and functionality of each mechanic (6 total)	PASS
2	RAKE – Handle_Punctuation	Check if RAKE handle Punctuation chars correctly	PASS
3	RAKE – Ignores_Stop_Words_Correctly	Check if RAKE Ignore the stop words from the stop words list.	PASS
4	RAKE – Returns_Correct_Location	Check if RAKE recognize the correct environment given a text.	PASS
5	RAKE – Returns_First_Location_With_Highest_Score	Check if RAKE recognize priorities. Some locations are not the correct location of the	PASS

		story. We only want the correct location to be returned	
6	RAKE – Returns_Location_From_Multiplie_Indicators	Check if RAKE can return the correct location form only correct locations to the story.	PASS
7	RAKE – Return_None_When_No_Location_Present	Return None if there is no location from the text.	PASS
8	TestHandlePlayerChoice_Modify player stat	Check the modifiers stats of the player (life,luck)	PASS
9	TestHandlePlayerChoice_CombatOutcome	Check if the combat mechanic is working	PASS
10	TestHandlePlayerChoice_SkillCheckOutcome	Check if the skill Check mechanic is working	PASS
11	OpenAllInterface_SetAndTrigger_Conclusion_SaveEvent	Check if the Conclusion page is saved (Victory or Defeat pages)	PASS
12	OpenAllInterface_SetAndTrigger_IsEndedChangedEvent	Check if the current page has been fully loaded	PASS
13	OpenAllInterface_SetAssistantID_UpdatesCorrectly	Check if the change between pages are correct	PASS
14	TestSendNarrativeToAPI_Failure	Check if we got a response from the API	PASS
15	TestSendNarrativeToAPI_Success	Check if did not got a response from the API	PASS

12. User Guide

Welcome to TaleWeaver!

We are thrilled to have you on this journey through TaleWeaver, where every decision you make will not only shape your own story but also influence the world around you in unique and unexpected ways. Step into a realm where imagination and artificial intelligence combine, creating ever-evolving adventures that are personal, dynamic, and entirely tailored to your choices. Whether you prefer a casual playstyle or thrive on complex decision-making, TaleWeaver promises an engaging experience where you can create your own literary masterpieces.

System requirements:

- Memory: 16 GB RAM
- Graphics Card: NVidia GTX 1050ti or newer
- CPU: Intel i5
- File Size: 15.5 GB
- OS: Windows 7 or newer

Getting Started

Installation:

To get started with TaleWeaver, follow these steps for a seamless installation:

- 1) Visit the official GitHub page of TaleWeaver at: [TaleWeaver GitHub](#).
- 2) Download the game files from the repository readme or from [here](#).
- 3) Extract the downloaded files and install the game via the Installer.
- 4) Run the executable to launch the game and begin your adventure.

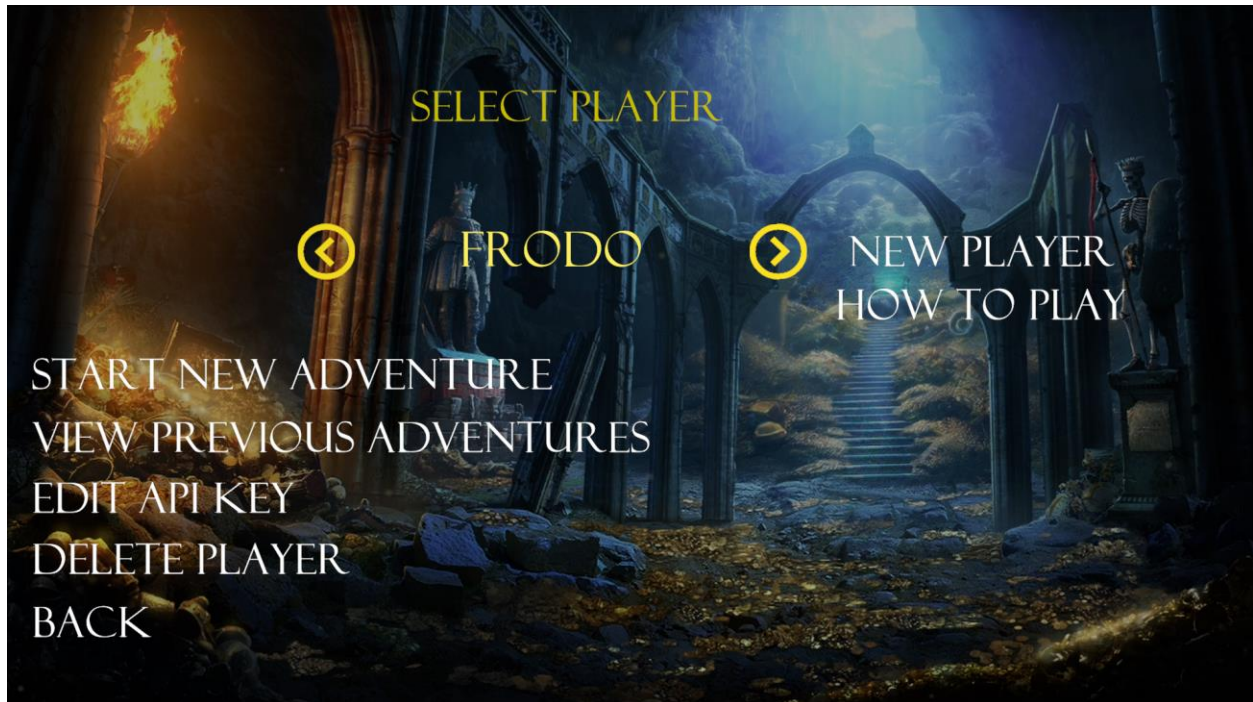
Navigating the Main Menu



Once you have successfully created or selected a player, you will be directed to the main menu. Here, you will find several important options that allow you to control your gaming experience:

- **Start Game:** Begin a new journey filled with unexpected twists and turns.
- **Settings:** Adjust the volume of the in-game music to your preference and the DALL-E model. Note that DALL-E 3 is more expansive and a bit slower loading times, but with far better pictures.
- **Credits:** View the names of developers and contributors behind TaleWeaver.
- **Quit:** Exit the game.

Creating a Player



Creating a player is simple and intuitive:

- 1) Once the game is launched, click on the "**New Player**" option to create a new player profile.

The image shows a player creation form with a dark background. At the top, there's a decorative gold border. Below it, the text 'NAME :' is followed by a red rectangular input field. Underneath, 'API KEY :' is followed by a larger red rectangular input field. In the bottom right corner of the API key field, there is a small logo for OpenAI and the text 'Get Your Own OpenAI API Key'. At the bottom of the form, there are two buttons: 'CREATE' and 'BACK'.

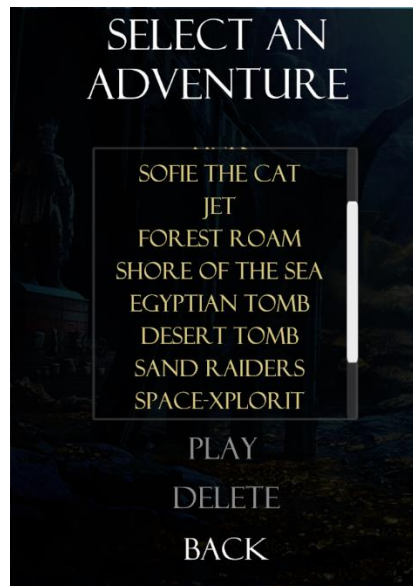
- 2) Enter your chosen player name, which will identify your character throughout the game.

- 3) **You will need an OpenAI API key** for interaction with the AI-driven story mechanics. If you don't have one, no worries! A link to purchase or obtain an API key is conveniently located in the bottom right corner of the screen during the player creation process. **NOTE TO**

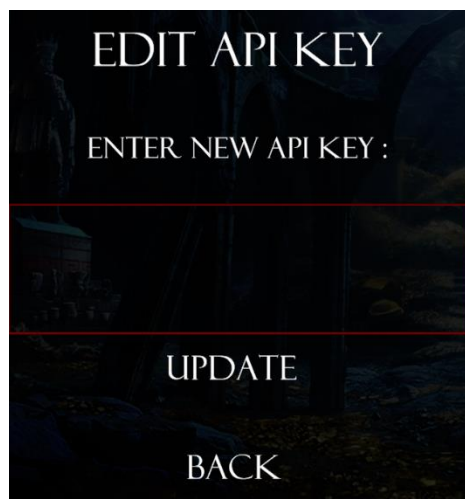
TESTER : key for you : **sk-taleweaver-dJIVNExk2buHWK54hyCfT3BIbkFJPluuUdA8v2AbvIZqfcsz**

- 4) After entering your API key, click "**Create**" to finalize your player profile. You are now ready to step into your first adventure!

If you have multiple players saved, you can choose an existing player profile at the start. Alternatively, you can create a new player by providing a name and API key, as previously described. This player selection screen also allows you to:



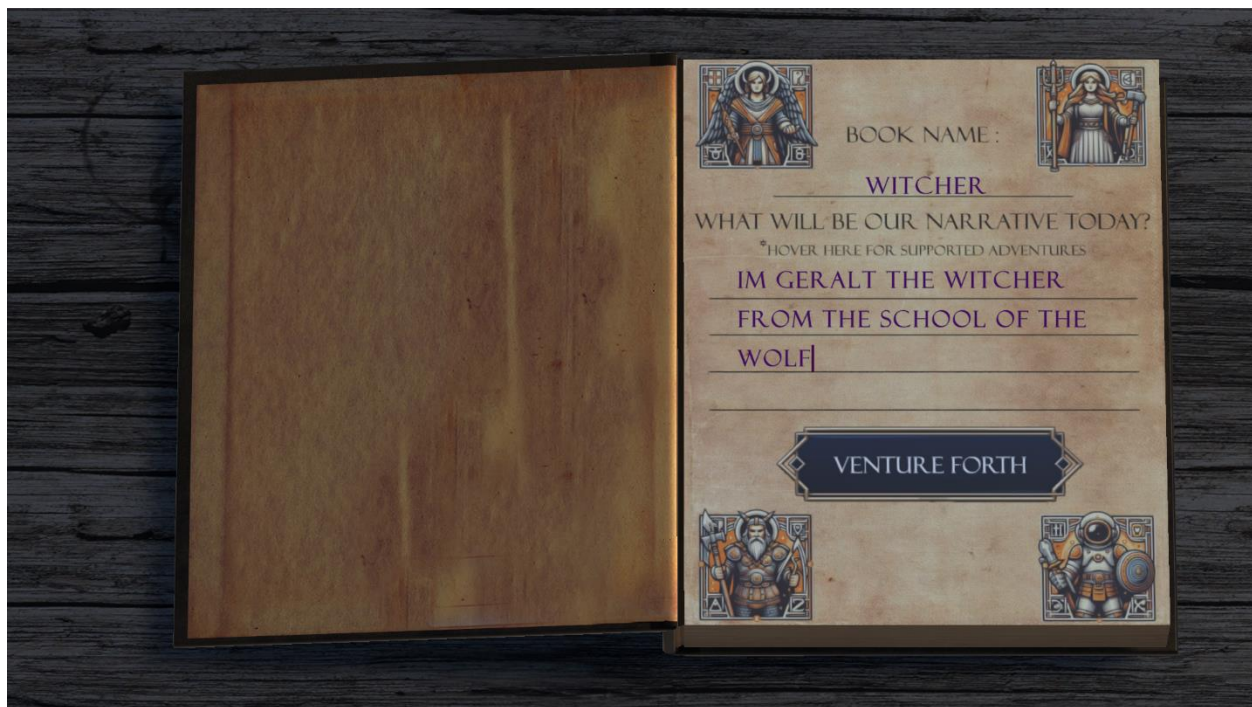
- **View Past Adventures:** Browse through your previously completed adventures, reminiscing over the stories you've created and the challenges you've overcome.
- **Edit API Key:** Change your OpenAI API key if needed for a more personalized experience.



- **Delete Player:** Permanently remove a player profile from the system. Be cautious as this action cannot be undone.



Starting a New Adventure



Once you're ready to embark on a new quest, follow these steps:

- 1) Open the book by clicking on the brown book cover "Tale Weaver"
- 2) Choose a name for your new adventure book. This will be the title under which the story is saved and can be referenced later.
- 3) Write an adventure description that provides some context or inspiration for the story generation process. This description will help guide the AI in crafting a tailored narrative based on your input.

- 4) Review the list of supported adventure locations. While many locations are pre-programmed into the game, don't worry if your preferred setting isn't on the list! You can still play your adventure in the library or similar alternative settings.
- 5) When you're satisfied with your setup, click "Venture Forth" to enter the game world.

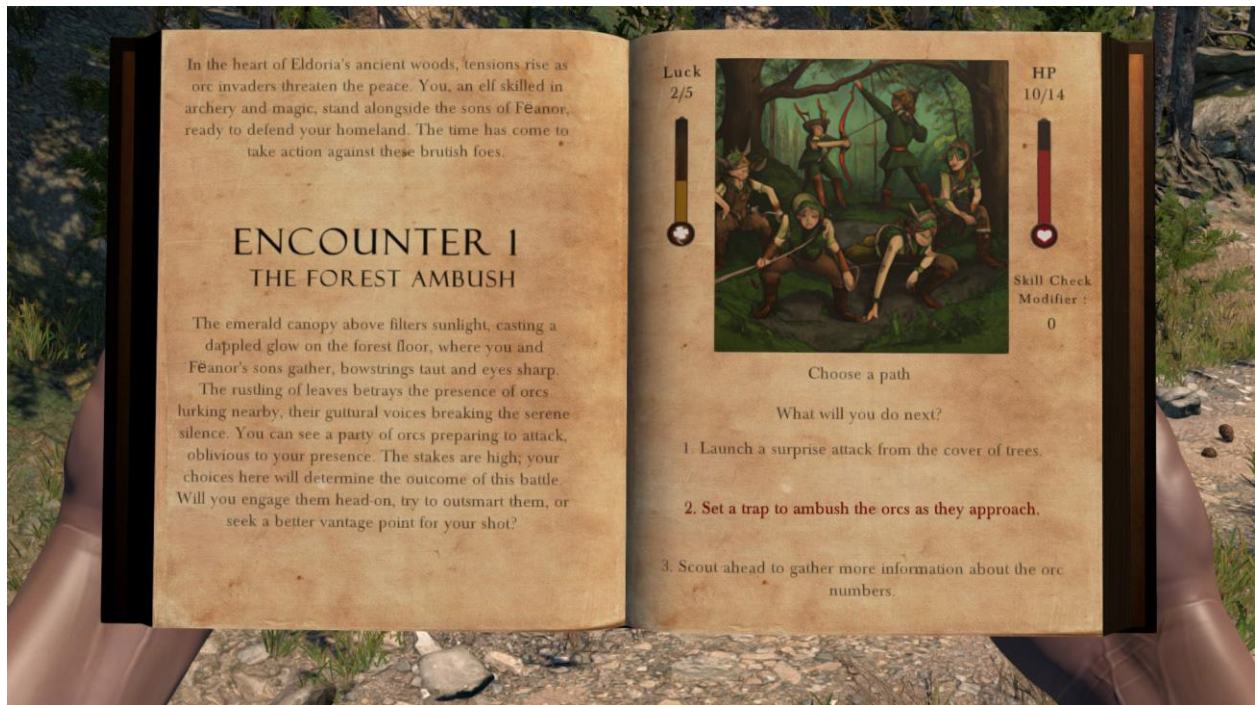
Gameplay Overview



Once inside the game, you'll find yourself in a vibrant and immersive open world, armed with nothing but your trusty adventure book. You can explore the landscape by walking (WSDA), jumping (Space Bar), and even sprinting (Holding shift). The primary game mechanic revolves around interacting with your book, which contains the story pages that will guide your journey.

To interact with the book:

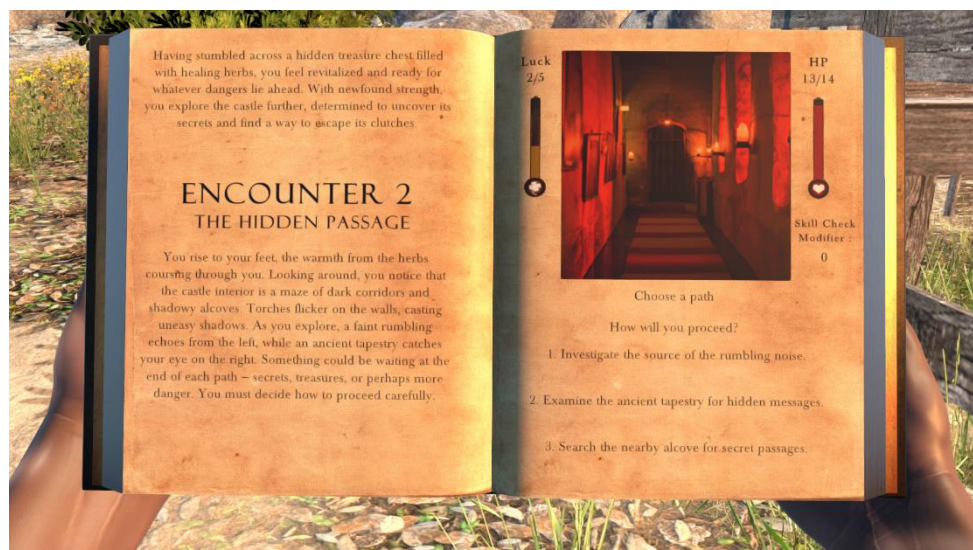
- Use the mouse wheel to **zoom in or out on the book**.
- Each page will present you with a unique piece of the adventure and prompt you to engage with one of the game's six mechanics.



Mechanic Types:

Here is a breakdown of the six types of mechanics you will encounter throughout your adventures:

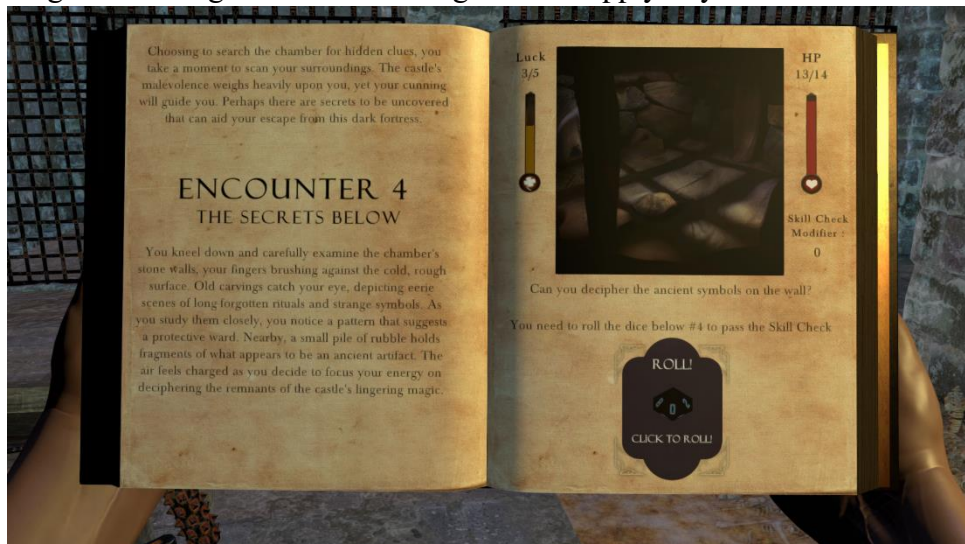
- 1) **Options Mechanic** : The Options Mechanic allows you to choose between multiple paths or actions. These choices will directly impact the subsequent pages of your story, setting off a chain reaction of events that shape the rest of the adventure. Consider your choices carefully, as every decision carries weight and could change the outcome of your journey.



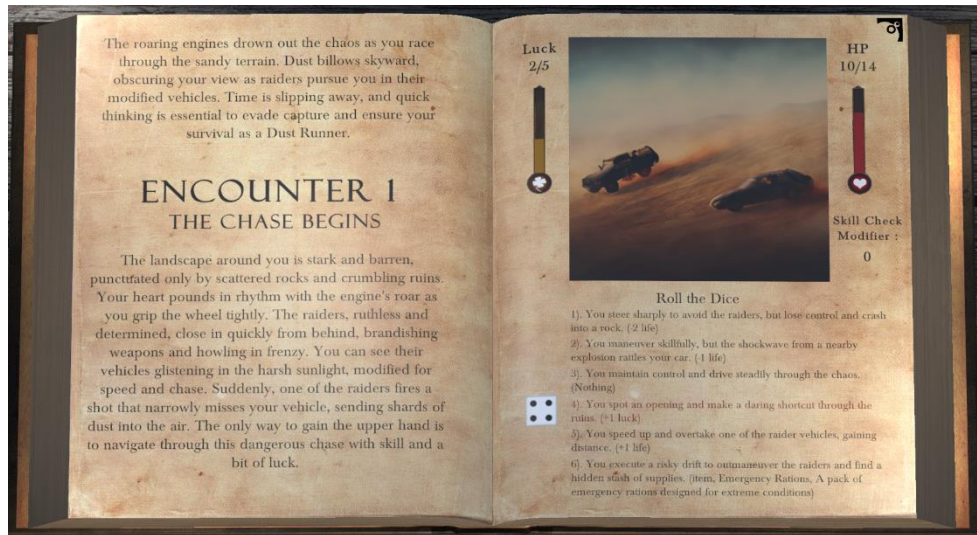
- 2) **Combat Mechanic** : In the Combat Mechanic, you'll engage in battles against adversaries, which could range from simple foes to fearsome bosses. The game will provide details on the enemy, including their name and difficulty level. You must roll a D20 (20-sided dice) and hope to roll a number equal to or greater than the enemy's difficulty score. If successful, you'll emerge victorious. If not, you'll lose a life but can try again. Luck points can be used to reroll the dice and improve your chances.



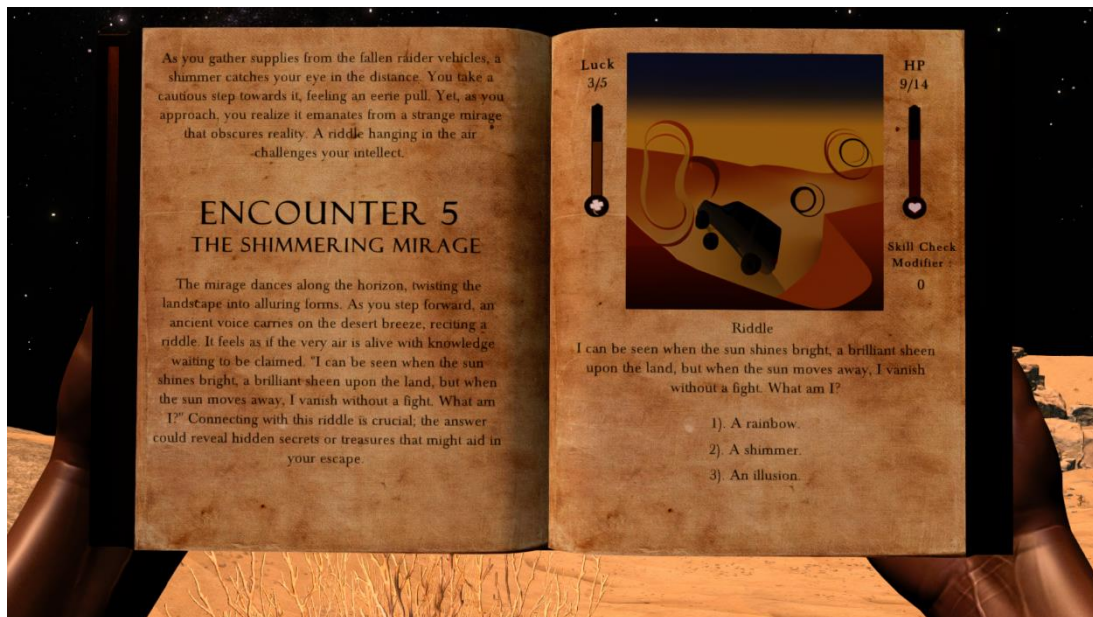
- 3) **Check Mechanic** : The Check Mechanic presents a challenge where you must roll a D10 (10-sided dice) and score below or equal to a difficulty number to succeed. If you fail, you lose a life and move on. You can use luck points to reroll. Any modifiers you've gained during the course of the game will apply to your final result.



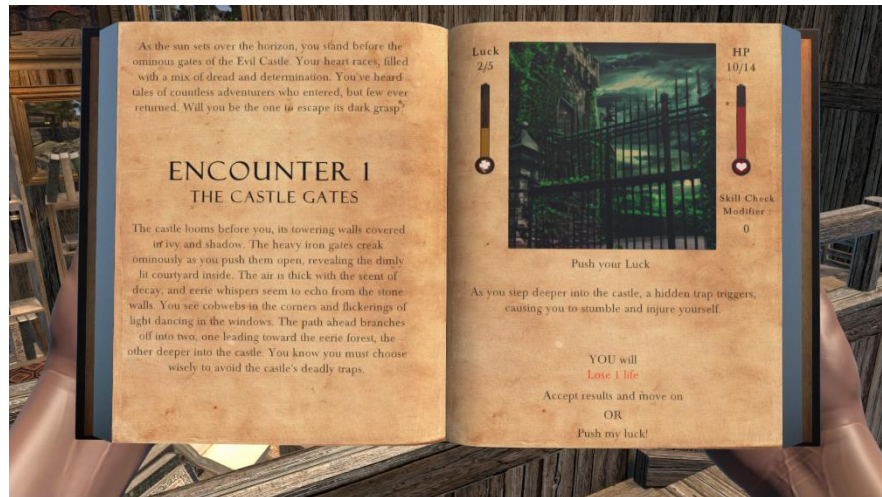
- 4) **Roll Mechanic** : In this mechanic, the outcome is entirely based on the dice roll. You roll a D6 (6-sided dice), and the result determines what happens next. You may experience a beneficial event or face a negative consequence, adding an element of unpredictability to your adventure. Luck points are again available for rerolling.



- 5) **Riddle Mechanic** : The Riddle Mechanic challenges you to solve a riddle. Answer correctly to advance, but failure to solve the riddle will result in the loss of a life. The stakes are high, and each riddle serves as a mental test to keep you on your toes.



- 6) **Luck Mechanic** : In the Luck Mechanic, you will be presented with a situation and a corresponding outcome, which may be either good or bad. You have the option to accept the result as it is, or you can choose to "Push my Luck!" This action gives you the opportunity to ignore the initial outcome for a chance at a significantly better outcome, but beware—there's also the possibility of a much worse result. The choice is yours to make, but as the saying goes, fortune favors the bold!



Adventure Objective

Your main objective is to successfully complete all 10 pages of the adventure book. Each page brings you closer to the final boss encounter, which occurs on the 10th page. Defeat the boss in combat to triumph and finish your journey, at which point the game will generate a conclusion that wraps up your story.

However, if you perish before reaching the end, don't despair! TaleWeaver will still generate a defeat story, allowing you to reflect on your valiant efforts and relive the adventure from a different perspective.

Game World

The Game World will give you an immersive and relaxing place to read the book.

Some key Features about the Game World:

- Movement: Walk, jump, and sprint across open landscapes.
- The book is your constant companion and key to progressing through the game. Make sure to consult it regularly as each page will present new opportunities and challenges.

- Transitions between worlds are indicated by black loading screens. These brief moments signify a change in location or scenario of the world.

Game Controls and Interface

To pause the game, press the "Esc" key at any time. This will bring up the pause menu, from which you can:

- Return to the main menu.
- Unpause the game.
- Change settings such as your character's voice, weather conditions, or time of day.



Previous Adventures

At any point, you can return to the "Select Player" menu to access your previous adventures. By clicking on a saved adventure, you'll have the option to:

- **Read** the adventure and revisit the story you crafted.
- **Delete** the adventure, permanently removing it from the game. Note: **this action cannot be undone**, so be mindful before deleting your adventures!



13. Maintenance Guide

Please note that we developed the game with Unity version 2022.3.16f and we used TextMeshPRO.

Also note that because the game dev kit is very large and because of copyright coding material you need to download and replace the Assets folder with the following folder to develop the game:

<https://drive.google.com/file/d/1JI6CcHlnrJdDXwCe0dU61oRv9CrOW5he/view?usp=sharing>

13.1 How to add more locations –

First we need to go to Rake.cs (In the scripts/Enviroments/Rake) and add the new location in the following line :

```
1 reference | 0 changes | 0 authors, 0 changes
public Rake(HashSet<string> stopWords)
{
    _stopWords = stopWords ?? new HashSet<string>();
    // Initialize with typical location indicators, customize as necessary for your domain
    _locationIndicators = new HashSet<string> { "library", "office", "desert", "dungeon", "pirates",
        "village", "swamp", "asia", "tavern", "cave", "mountain", "space", "forest",
        "hell", "meadow", "city", "graveyard" };
}
```

After that we need to go to KnownLocations.cs and add keywords that associate with your new location. For example here is the Library section :

```
2 references | Seth7171, 52 days ago | 1 author, 1 change
public static class KnownLocations
{
    public static Dictionary<string, string> Locations = new Dictionary<string, string>
    {
        {"library", "library"},
        {"archives", "library"},
        {"bookshelf", "library"},
        {"reading room", "library"},
        {"study hall", "library"},
        {"public library", "library"},
        {"reference section", "library"},
        {"digital library", "library"},
        {"manuscript room", "library"},
        {"library stacks", "library"},
        {"bookstore", "library"},
        {"lending library", "library"},
        {"research library", "library"},
        {"reading nook", "library"},
    }
}
```

Then you need to prepare the scene. Make a sperate scene in URP with your new location, and after you finish the scene we need to import the player capsule and the player camera to place the player spawn location after teleporting in the new game world. We recommend the player to stay at 0,0,0 and adjust the map according to the player location.

Go to TeleportManger.cs and inside TeleportTo function add a new case with your location name and a fade image for the location loading screen and a teleport location vector, rotation vector and the name of the scene, see the following picture :

```
public void TeleportTo(string mapName)
{
    //SetAllMapsOff(); // Deactivate all maps at the beginning
    if (BookLoader.Instance.sceneName.Contains(mapName, StringComparison.OrdinalIgnoreCase))
        return;
    if (Enviro.Instance != null)
        Enviro.Instance.gameObject.SetActive(true);

    switch (mapName.ToLower()) // Ensure the map name is case-insensitive
    {
        case "library":
            fadeImage = imageList[0];
            TeleportToMap(new Vector3(0, 0, 0), Quaternion.Euler(0, 0, 0), "LibraryMap");
            break;
    }
}
```

You should also go to the script Enviro.cs located in Scripts/GameWorld then you need to add a new case in OnSceneLoaded method :

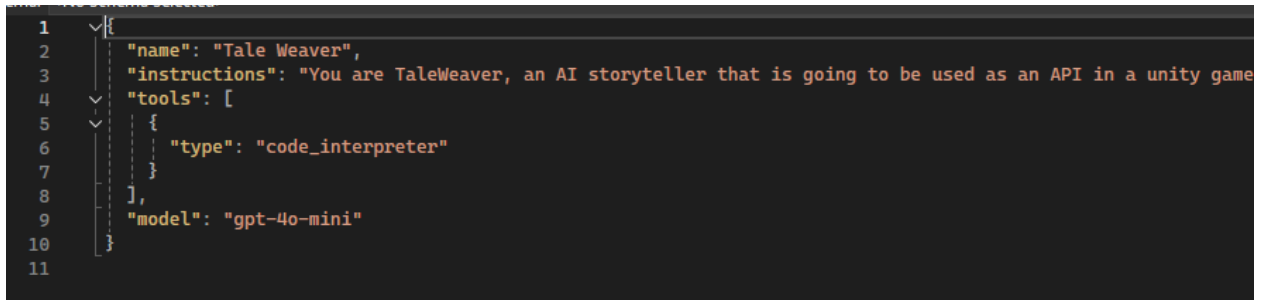
```
2 references | 0 changes | 0 authors, 0 changes
void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    // Enable the entire EnviroSkyMgr GameObject
    EnviroSkyMgr.instance.gameObject.SetActive(true);
    EnviroSkyMgr.instance.SetTimeProgress(EnviroTime.TimeProgressMode.None); // Stop the time progression
    EnviroSkyMgr.instance.useFlatClouds = false; // Enable flat clouds
    switch (scene.name) // Ensure the map name is case-insensitive
    {
        case "LibraryMap":
            EnviroSkyMgr.instance.SetTimeOfDay(24f); // 24-hour format
            EnviroSkyMgr.instance.ChangeWeather(0); // weather preset for "Clear" with ID 0
            break;
    }
}
```

13.2 How to edit the model Instructions –

If you want to edit the model instructions, first your should find the “message_to_create_model.txt” text file and edit your instructions there. After you done, we need to convert the text file to a json file. The current json is located in Scripts/OpenAI API the json is called assistant_create.json note that when you create a new build you **must** also add the new json file in

BUILDNAME\TaleWeaver_Data\TaleWeaver_Data\Scripts\OpenAI API else the new build will not have the new instructions. Note that you need to convert your instructions to a single line like in the picture below. Also note that we use /n for a new line.

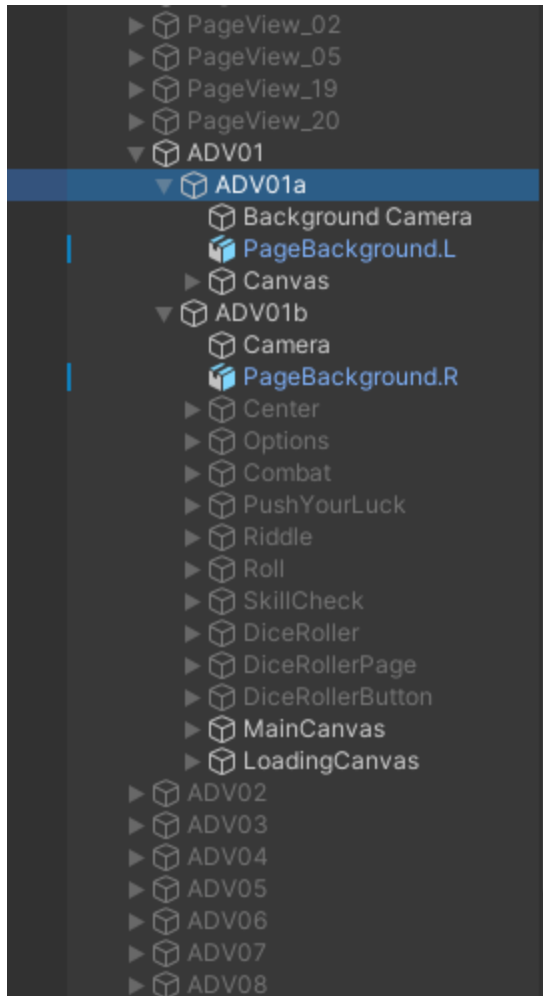
I would highly recommend letting AI do your conversion to one line because this is a very tedious task. Also remember that you need to code the parser.cs located in /Assets/Scripts/BookInfo to handle the new response that the Assistant API will provide. Other than that you should also modify BookLoader.cs (located in /Assets/Scripts/BookInfo) and GameMechanicsManager.cs located in (/Assets/Scripts/Gameplay) to use the new response in gameplay.



```
1  {
2    "name": "Tale Weaver",
3    "instructions": "You are TaleWeaver, an AI storyteller that is going to be used as an API in a unity game",
4    "tools": [
5      {
6        "type": "code_interpreter"
7      }
8    ],
9    "model": "gpt-4o-mini"
10 }
11
```

13.3 How to add new mechanics –

The main logic for mechanics is located in GameMechanicsManager.cs located in (/Assets/Scripts/Gameplay) but more scripts like BookLoader.cs, Parser.cs (located in /Assets/Scripts/BookInfo) and OpenAIInterface.cs located in /Assets/Scripts /OpenAI API/ also you could add more stuff into ClickableText.cs and CreateButtonsInBook.cs located in /Assets/Scripts/GameWorld . after you add the logic, you should know that inside the Unity editor the BookLoader script plays a large role of connecting the gameplay to the actual book in the game, currently each two pages are connected to this script. See the picture below and we will explain how it works after the picture.

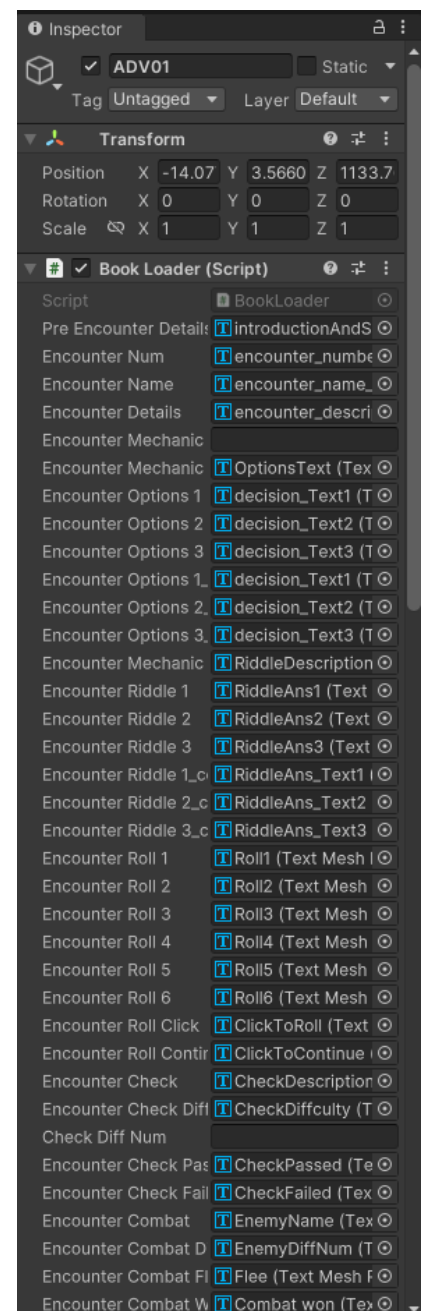


As you can see, each 2 pages share ADV## (## |1,2,...,10) and inside each side of the open book have ADV##a (left side) and ADV##b (right side) and ADV## is connected to the book loader script as you see in the picture on the right.

Note that each component like info or text we parsed will be loaded here and is listed, and if you create a mechanic, this mechanic in the book loader should be connected to each of the pages (1,2,...,10)

Another aspect is the player in game stats, lets say you want to add +DMG in combat as a new feature, then you need to incorporate that in the previous game mechanic scripts and the model, but also another script : PlayerInGame.cs that is located in Assets/Scripts/GameWorld/

There you can add the logic for the new stat, but you should also add a new script in /Assets/Scripts/PlayerStats/ see the example there on how we added the modifier for the check dice.



13.4 How the data structure of book and player works –

We have 2 data structures : Players and Books. Each Play can hold books.

The Data is saved inside .json files and we also save the pictures for each page.

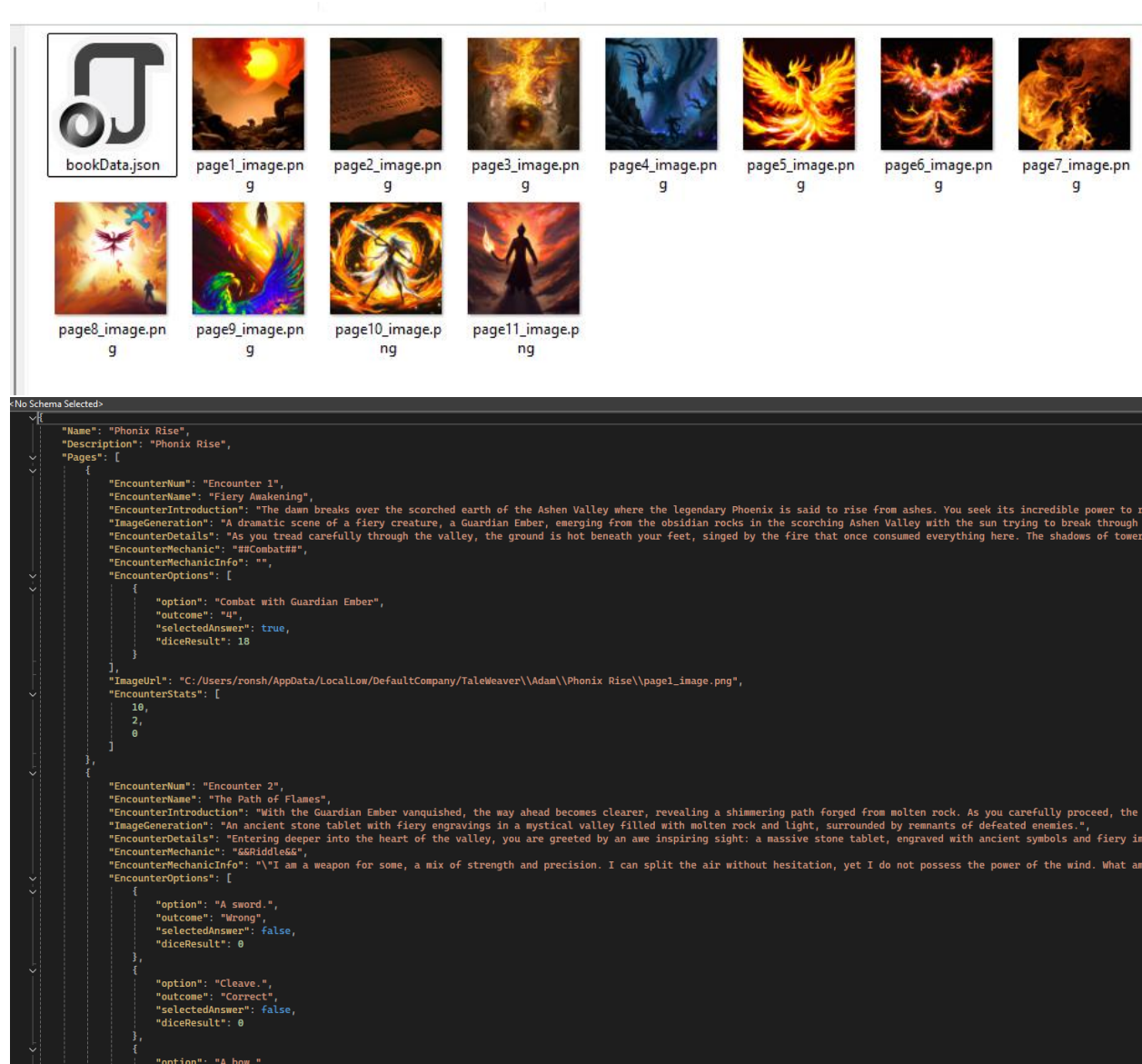
The save files are located in :

AppData\LocalLow\DefaultCompany\TaleWeaver

There you will have a “playerManager.json” that hold all the player names and you will have a folder for each player.

Inside the folder of each player you have the saved books (that the players can view later). Each book will be recorded first inside the playerData.json and each book will have a folder. The playerData.json will also save the API key for each player and the assistantID.

Each book will have images for each page and most importantly the bookData.json



As you can see in the picture, each mechanic is saved with the following :

```
{
  "EncounterNum": "Encounter 3",
  "EncounterName": "The Ashen Shrine",
  "EncounterIntroduction": "Your answer, \"A bow,\" echoes in the air, but the anc:",
  "ImageGeneration": "A mystical Ashen Shrine surrounded by swirling ashes and fie",
  "EncounterDetails": "As you contemplate your next move, the ground trembles, and",
  "EncounterMechanic": "$Roll$",
  "EncounterMechanicInfo": "",
  "EncounterOptions": [
    {
      "option": "The ashes engulf you, and you lose your footing. (-2 life)",
      "outcome": "-2 life",
      "selectedAnswer": false,
      "diceResult": 0
    }
  ],
}
```

This is important because we use this data later to load the book and see the player selection and mechanic, story elements.

Each page will end with :

```
},
  "ImageUrl": "C:/Users/ronsh/AppData/LocalLow/DefaultCompany/TaleWeaver\\Adam\\Phonix Rise\\page3_image.png",
  "EncounterStats": [
    9,
    1,
    0
  ]
}
```

The first number is the Health of the player in this current stage of the game, the second number is the luck, and the third is the modifier to the dice.

All of this is managed in the following scripts :

All the scripts from /assets/Scripts/PlayerInfoData/

And Book.cs from /assets/Scripts/BookInfo/

The saving of data is done with a chain of several key scripts like GameMechanicsManager.cs

And bookloader.cs

You can see in the picture below in line 300 , that the GameMechanicsManager.cs handle the player click to flee battle, we need to save that data so we call Bookloader.cs as you can see in the second picture below. Please note that currentbookData in line 216 is the Book.cs script.

1)GameMechanicsManager.cs

```
277 public void HandlePlayerChoice(string bookName, string choice, Option mechnismOption, bool callEndCall = true)
278 {
279     if (choice.Contains("next Is Combat"))
280     {
281         _isNextIsCombat = true;
282     }
283
284     if (choice.Contains("Push my luck!"))
285     {
286         //need to reveal some how the scenario 2
287         BookLoader.Instance.ReveallLuckSenario2();
288         _isSenario2 = true;
289         //return to not triger senario 1 outcome.
290         return;
291     }
292
293     if (_isSenario2)
294     {
295         mechnismOption = BookLoader.Instance.currentpage.EncounterOptions[1];
296     }
297
298     if (choice.Contains("Flee!"))
299     {
300         BookLoader.Instance.SaveChangedData(0, -1);
301         PlayerInGame.Instance.Loselife(3);
302         //ADD LOADING!!!
303         GetNextMechanicBasedOnChoice(bookName, "Fled Combat");
304         //return to not triger next if cases.
305         return;
306     }
307 }
```

2)Bookloader.cs :

```
212 public void SaveChangedData(int optionIndx, int diceResult = 0)
213 {
214
215     // Find the specific option and change isCorrectAnswer to true
216     var encounterOptions = currentbookData.Pages[pageNumBasedon_objectName].EncounterOptions[optionIndx];
217     if (encounterOptions != null)
218     {
219         encounterOptions.selectedAnswer = true;
220         encounterOptions.diceResult = diceResult;
221     }
222
223     // Serialize the updated JSON data
224     string updatedJson = JsonUtility.ToJson(currentbookData, true);
225     File.WriteAllText(bookFilePath, updatedJson);
226     Console.WriteLine("Updated JSON data with played choice");
227
228 }
```

14. References

1. Christina C, "Children and young people's reading in 2019" National Literacy Trust research report
2. Jerry Huang, "ChatGPT in Gaming Industry".
3. Harshitha H, "Choose Your Own Adventure: The Evolution of Digital Interactive Fiction and Its Use in Language Pedagogy."
4. Chen-Chung Liu, "An analysis of children's interaction with an AI chatbot and its impact on their interest in reading."
5. Sarah Thorne, "Hey Siri, tell me a story: Digital storytelling and AI authorship."
6. Nelius, J. (2020). This AI-powered choose-your-own-adventure text game is super fun and makes no sense. Gizmodo Australia. Retrieved from <https://www.gizmodo.com.au/2020/08/this-ai-powered-choose-your-own-adventure-text-game-is-super-fun-and-makes-no-sense/>
7. J.S. Baruni, "Keyphrase Extraction from Document Using RAKE and TextRank Algorithms"
8. OpenAI developer platform - <https://platform.openai.com/docs/introduction>
9. OpenAI DALL-E image generator - <https://openai.com/index/dall-e-3>
10. Unity Asset Store - [Unity Asset Store - The Best Assets for Game Making](#)