BUCHAREST UNIVERSITY OF ECONOMIC STUDIES

FACULTY OF ECONOMIC CYBERNETICS, STATISTICS AND INFORMATICS

Bachelor Thesis
Unity Game: Cyber Odyssey

Supervisor

Prof. **Liviu-Adrian COTFAS**, PhD.

Student

Theodora ASLAN

Bucharest

2024

Declaration regarding the originality of the content and the assumption of responsibility

I hereby declare that the results presented in this thesis are entirely the result of my creation, unless references are made to the results of other authors. I confirm that any material used from other sources (magazines, books, articles and Internet sites) is clearly referenced in the work and is indicated in the list of bibliographic references.

Table of Contents

Introduction

I have always liked to play games, though it wasn’t until university that it dawned on me that I could also create them. I knew I wanted to create a game for my thesis, though at first I didn’t know what kind.

In the past years, developments in AI have led to the improvement of various domains, like finance and ecommerce. Since ChatGPT has been a helpful tool during my study years, I wanted to integrate it in my game.

Because I have never created one before, I decided that a text adventure is a suitable choice for a first game. Using the OpenAI API to integrate ChatGPT in my game means that the game is adapted to each user, using their prompts and choices to create a personalized and engaging experience.

In this thesis, I will detail the development process of the game, from the informatic system to the final application. The aim is to gain insight into the potential of AI in interactive storytelling and to test its ability to respond to user choices.

The game will be developed in Unity and in Visual Studio, using the OpenAI API for accessing the chat models.

1. Describing the Technologies

1.1 Unity Engine

Unity is one of the best-known game engines, providing advanced graphics and exports for free. First announced at Apple's Worldwide Developers Conference in 2005, the engine is more suitable for beginners due to its simple user interface, wide range of tutorials and examples, many available assets and few requirements in hardware. Unity uses object-oriented programming and C#, as opposed to other game engines that use script-based languages. [1]

The following components have been used to develop the game's UI:

- Canvas – shown as a rectangle area in the Scene View, contains other UI elements [2]

- Visual Components - used for creating and managing the appearances of game objects [3]:
    - Text – displays to the user a non-interactive piece of text, can be used to show captions, labels, or instructions [3]

- Interaction Components - objects that must be combined with other visual elements in order for them to work properly [4]:
    - Button – used together with an "onClick" event to define its action [4]
    - Input Filed –to make a text of a Text Element editable by the user [4]
    - Dropdown – used to store a list of options, usually a text string [4]

1.2 Open AI API

The OpenAI API provides a simple interface for developers to create an intelligence layer in their applications, powered by OpenAI's models. The Chat Completions endpoint powers ChatGPT and provides a simple way to take an input text and use a model like GPT-3.5 to generate an output. The models can be used for a wide array of tasks such as writing code, drafting documents, analyzing text, and tutoring in a variety of subjects. [5]

Text Generation

Having been trained to understand languages, code, and images, OpenAI's text generation models provide text outputs in response to inputs. These inputs, also referred to as "prompts", act as training for the model, providing instructions or examples in order for it to complete a task successfully. [6]

The chat models take a list of messages as input and return a model-generated message as output. Messages must be an array of message objects, where each object has a role (system, user, assistant) and content. The conversation is initially formatted with a system message, which sets the desired behaviour or the personality of the assistant, followed by requests from the user and responses from the assistant. The system message is completely optional, and the default system message is going to be something generic such as "You are a helpful assistant". [6]

Request Body:

- messages (required) – a list of messages which contains the conversation so far [7]
- model (required) – ID of the model to use [7]
- max_tokens (optional) – the maximum numbers of tokens that can be generated in the chat completion [7]
- temperature (optional) – determines the randomness of the chat output: lower values will make the output more focused, while higher values will make it more random; the value can between 0 and 2, defaults to 1 [7]

Tokens

The models process text in chunks called tokens. Short and common words like “the” are represented as one token, while longer words are split into multiple parts. For example, the word “tokenization” is split into “token” and “ization”. Unicode characters like emojis are split into many tokens that contain the underlying bytes. As a rule of thumb, 1 token is roughly 4 characters or 0.77 words for English text. [5]

Wrapper

For accessing the OpenAI API, the unofficial and simple OpenAI-API-dotnet C# .NET wrapper library developed by OkGoDoIt was used. [8]

1.3 C#

C# is an object-oriented programming language developed by Microsoft, used for building a variety of applications including web, mobile and games. [9]

In Unity, C# is the main language for developing the functionality of games. For example, by utilizing Unity’s MonoBehaviour methods, such as ‘Start’ or ‘Update’, we can synchronize the game behaviours with the actions of the player. [10]

In order to handle API requests to the OpenAI service without interrupting the game’s UI flow, async operations were used. Asynchronous programming allows tasks to run without interrupting the main thread, increasing the performance of the application.

1.4 Visual Studio

Visual Studio is an IDE developed by Microsoft, used to develop a variety of programs, including websites, web apps and mobile apps. The IDE currently supports a multitude of programming languages such as C, C++, C#, JavaScript, HTML, as well as providing support for other languages.

2. The Informatic System

2.1 Describing the System

The purpose of the application is to entertain the user, as well as discovering and challenging the capabilities of AI bots. The user chooses prompts and makes decisions, which are then sent to the bot. The bot analyses the prompts and creates a chapter and two options, sends it back to the user, who must select one of the options to continue the story.

One the application is started, the user can either start a new story or continue a previous story. Upon starting a new story, the user is asked to give it a suggestive title and write a list of prompts they want included in the story. The chat model receives a message containing instructions and the prompts written by the user, then writes the chapter and the two decisions and sends them back to the user, who will choose one option in order to continue the story. After each decision, the contents of the story are saved to a text file.

If the user wants to continue a previous story, they will select the title of the story they want to continue. The story will be loaded from the file with the same name, then it will be sent to the chat model, which will write the chapter and decisions and send them back to the user.

The diagrams below have been created in Visual Paradigm.

2.2 Use Case Diagram

By analysing the Use Case Diagram, we can observe how the user can interact with the game.

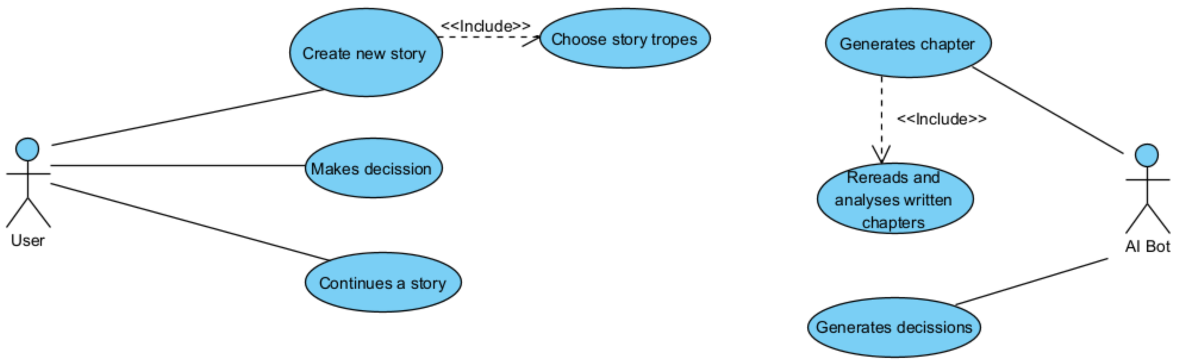


Figure 2.2.1 General Use Case Diagram

The diagram includes two actors: the user and the chat bot. The user can create new stories, with selecting the story tropes being a mandatory step, and make decisions to drive the plot forward. They can also continue a previously started story.

The bot receives the tropes from the user and creates a story based on them. The bot can also analyze previous chapters.

2.3 Activity Diagram

The activity diagram shown in Figure 2.3.1 contains all the actions the user can do to interact with the game.

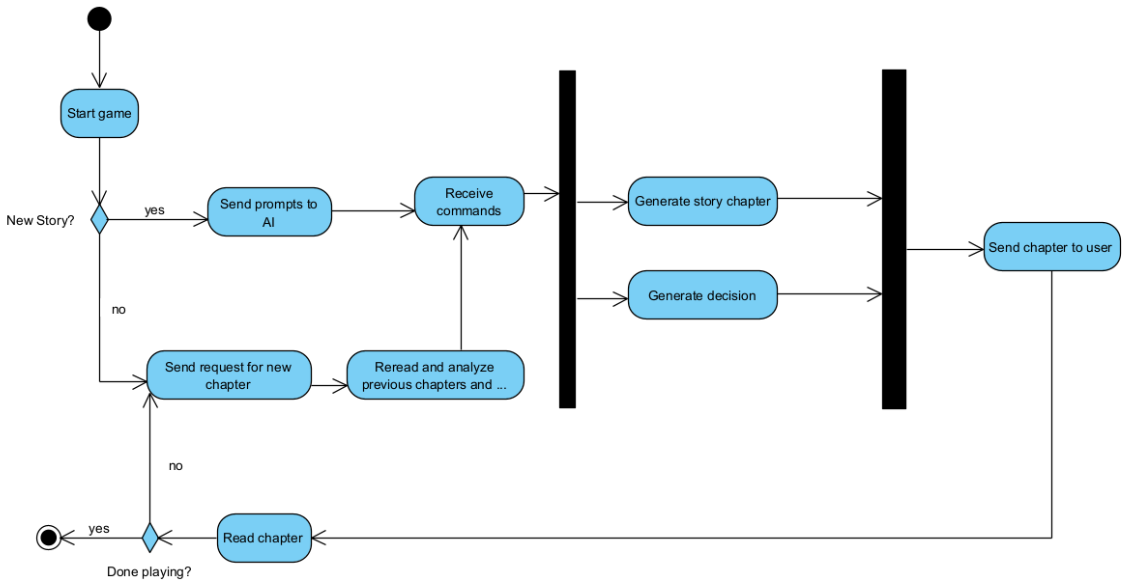


Figure 2.3.1 Activity Diagram

2.4 State Machine Diagram

The State Machine Diagram below shows the states and transitions the game goes through when the user starts playing.

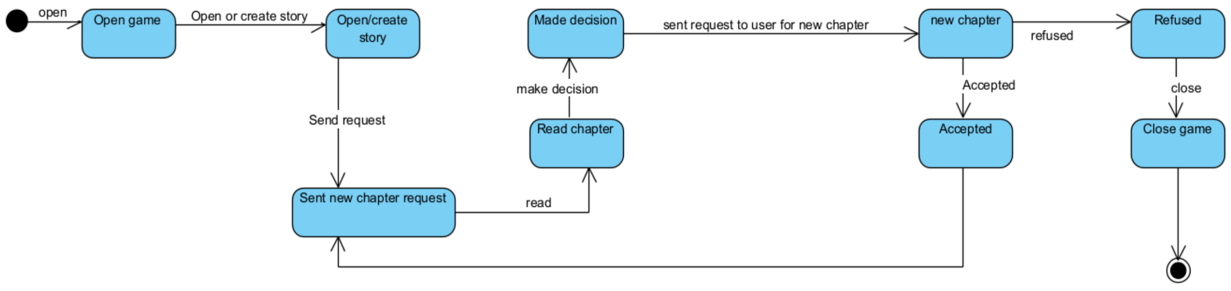


Figure 2.4.1 State Machine Diagram

The process begins when the game is opened. The user can start a new story or open an existing one, leading to the state where a new chapter request is sent. The chat model analyzes the prompts for the new story or rereads the old chapters, then generates a new story chapter and the decisions, which are then sent back to the user to be read.

The process continues with the user making the decision, and the request for a new chapter is sent again. The final state is when the game is closed, after the user decides to close the session.

2.5 Sequence Diagram

Sequence diagrams are used to show the interaction between objects, as well as the order in which they occur.

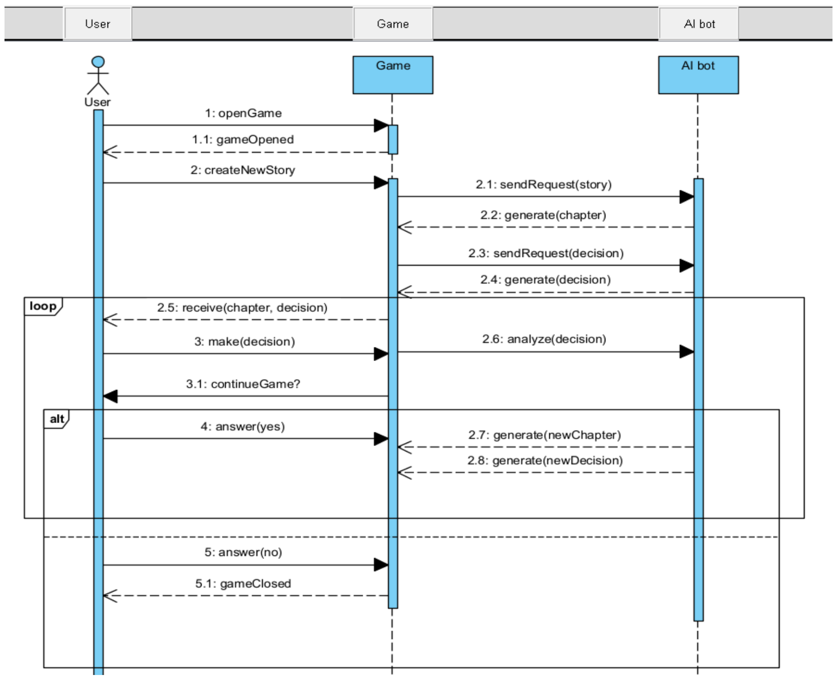


Figure 2.5.1 Sequence Diagram

The diagram above depicts the interactions of the user with the AI bot through the game interface when starting a new story.

The process begins with the user making the choice of starting a new story. A request is sent to the chat model through the game interface. The AI bot returns the story chapter and the decisions to the user, who then has to choose one of the two options to continue the story. This process is repeated until the user closes the game.

2.6 Business Process Diagram

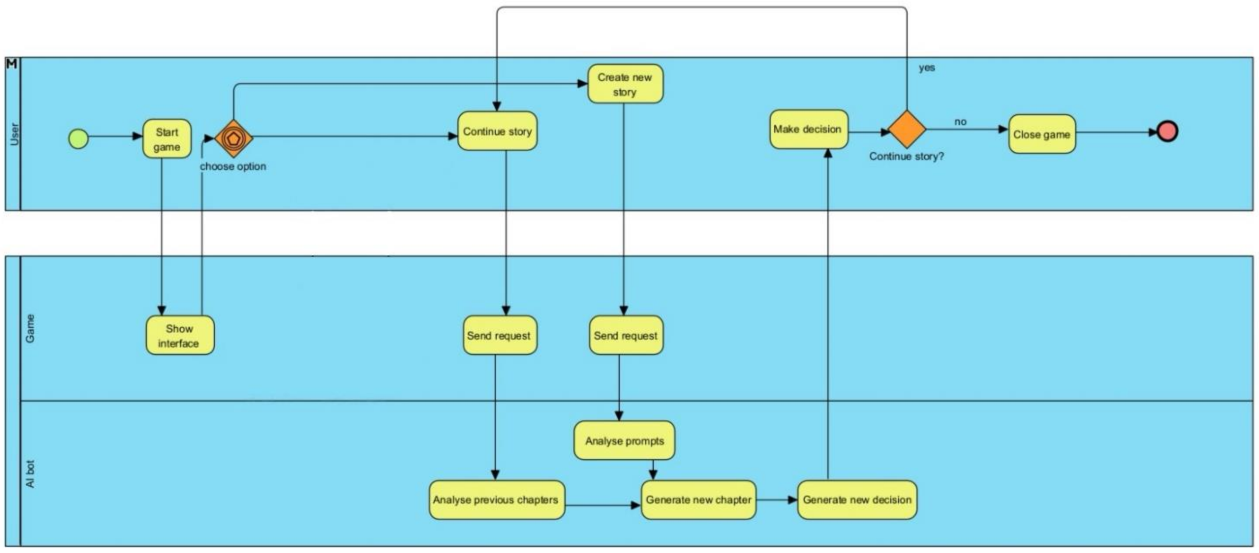


Figure 2.6.1 Business Process Diagram

The diagram in Figure 2.6.1 depicts the collaboration between the user, game interface and the chat model. After the user decides to start the game, a request is sent to the AI bot, which generates the text and sends it back to the user.

2.7 Designing the User Interfaces

Designing the interfaces is an important step that is going to help with developing the game. It provides an opportunity to consider what functionalities need to be implemented, as well as a visual representation of how the application looks like.

- Menu Interface:

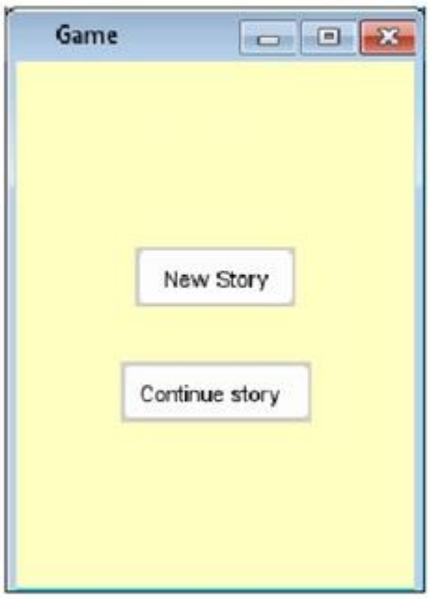


Figure 2.7.1 Menu Interface

- New Story Interface:

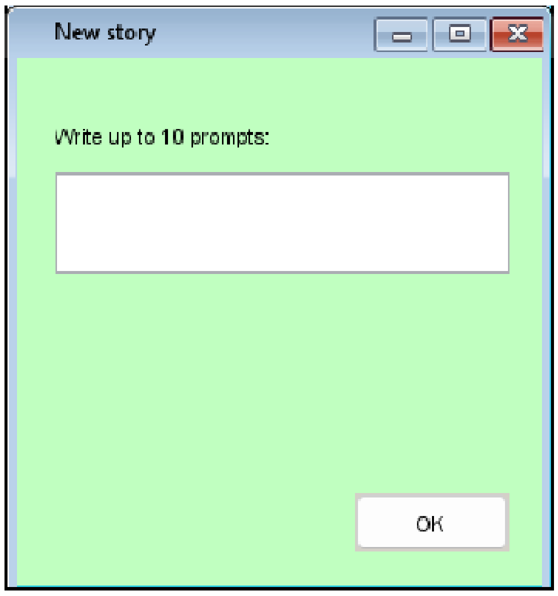


Figure 2.7.2 New Story Interface

- Continue Story Interface:

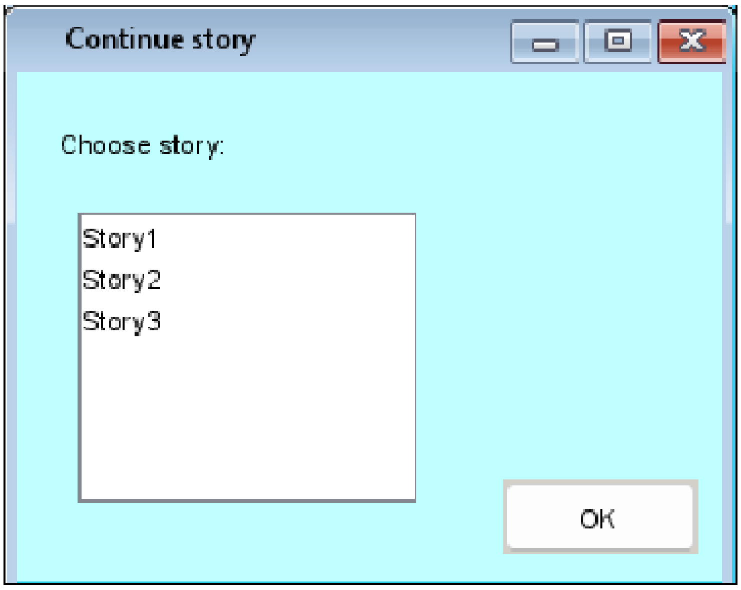


Figure 2.7.3 Continue Story Interface

- Gameplay Interface:

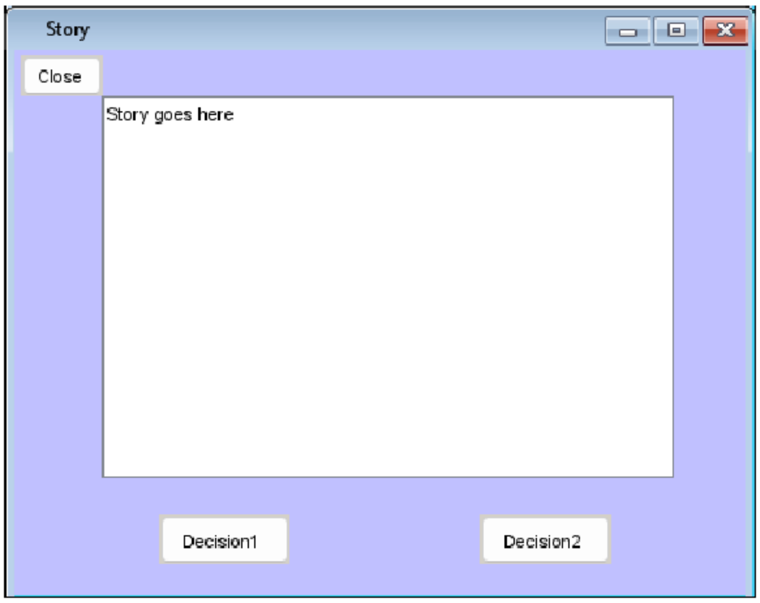


Figure 2.7.4 Gameplay Interface

3. Developing the game

3.1 Developing the game in Unity

While developing a text adventure, perhaps the most important part is designing the user interface, through which the user interacts with the game and subsequently, with the AI.

The user interfaces were created using Canvas elements. Each Canvas contains a combination of button, text, input text and dropdown elements using the integrated Unity UI system.

The first screen the user is welcomed with is the menu canvas, containing three buttons which prompt the user to choose between 3 actions: start a new story, continue an old story, or quit the game.

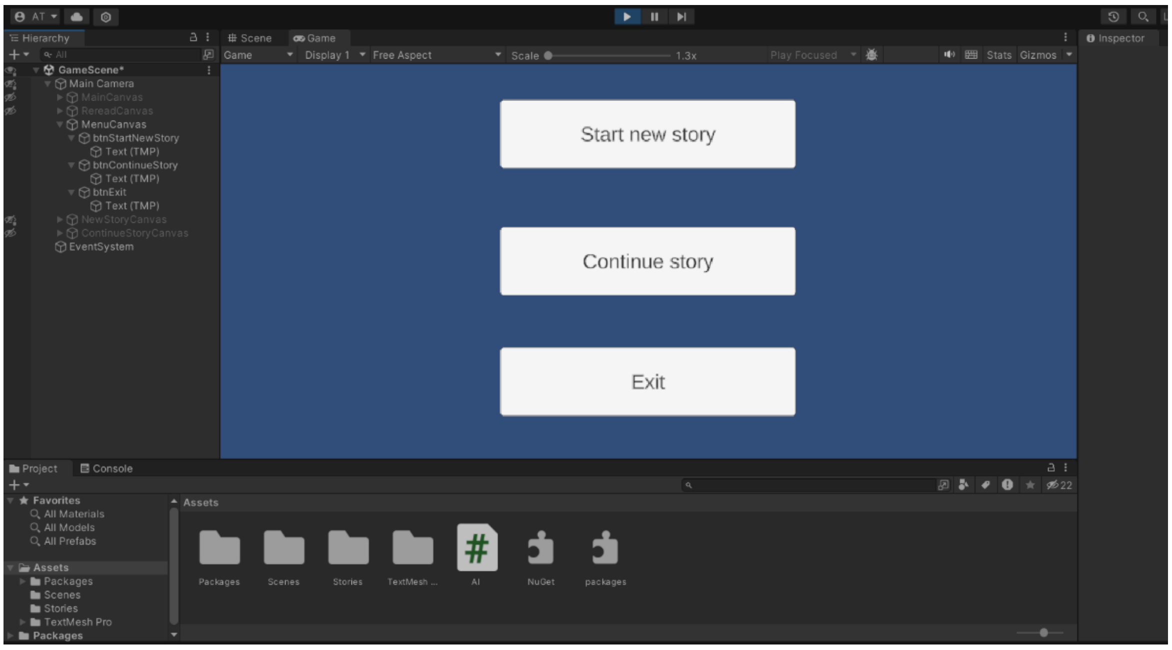


Figure 3.1.1 Start Menu Interface in Unity

Clicking the Start New Story button, the user will be presented with another canvas, containing, two non-editable text elements telling the user the kind of information they need to input, namely a title and some prompts they want to find in the story, two editable text fields in which the information should be inputted, a Cancel button, which leads the user back to the menu canvas, and an OK button that leads to the main story canvas.

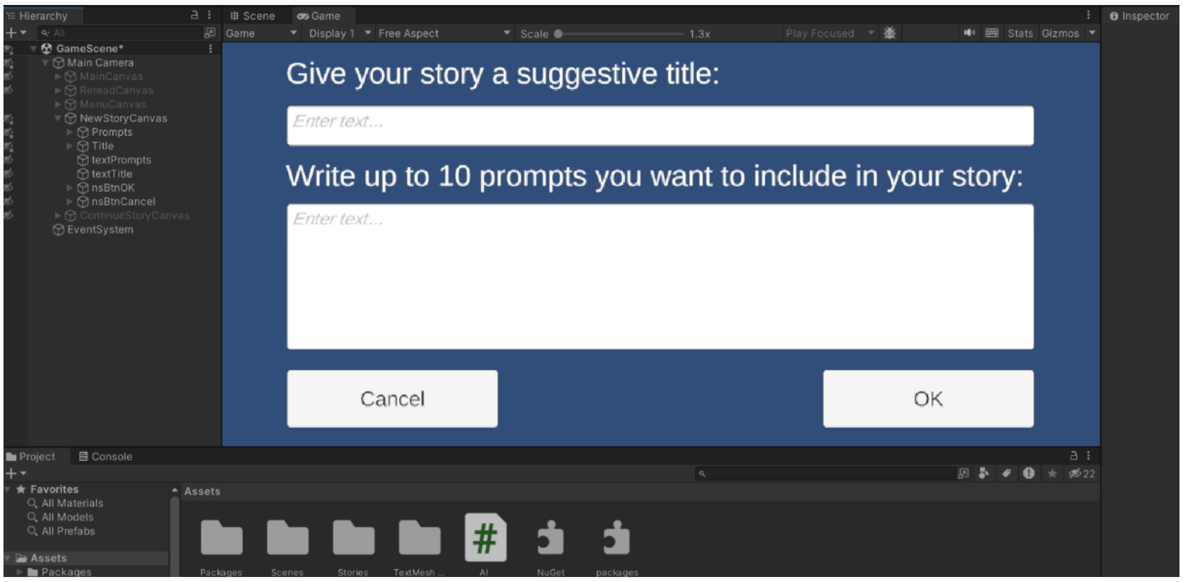


Figure 3.1.2 New Story Interface in Unity

After inputting the title and prompts and clicking the OK button, the user is presented with the main story canvas, that contains a text element with the name of the game itself, another text element in which the story chapters and the choices are written, two buttons for selecting the wanted choice and continuing the story, and an additional button for returning to the menu canvas .

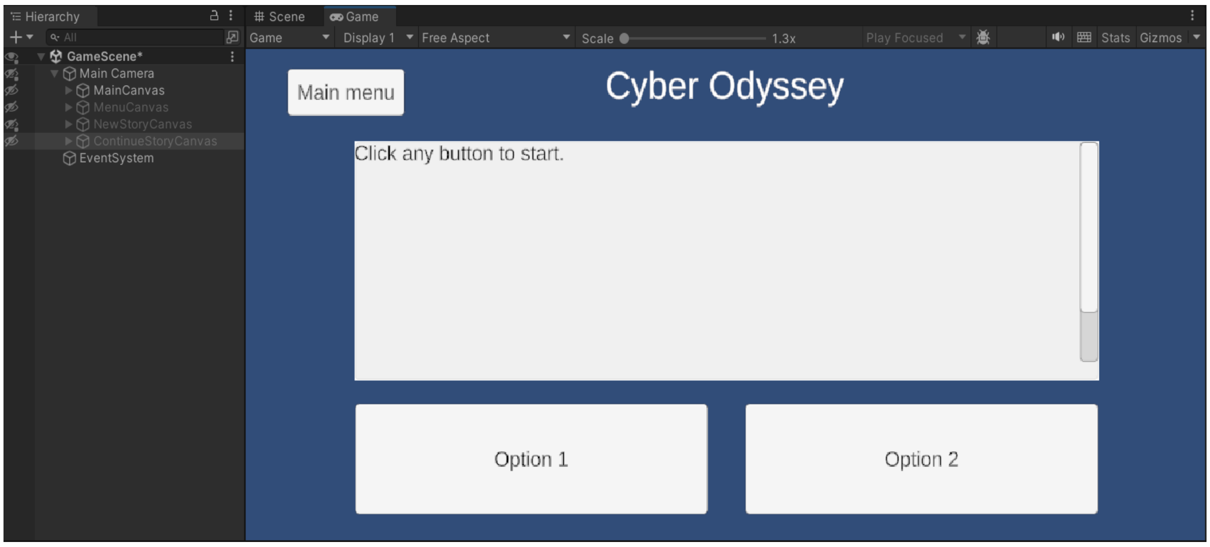


Figure 3.1.3 Gameplay Interface in Unity

Returning to the menu canvas, the user can also opt to continue a story started previously. To do that, the Continue Story button is clicked, after which the user is presented with another canvas containing a Dropdown element, a Cancel button and an OK button. The Dropdown element contains a list with all the titles of stories the user has played previously. Upon selecting a title and pressing OK, the user is brought back to the Main Story canvas, where they can continue the story from where they left off.

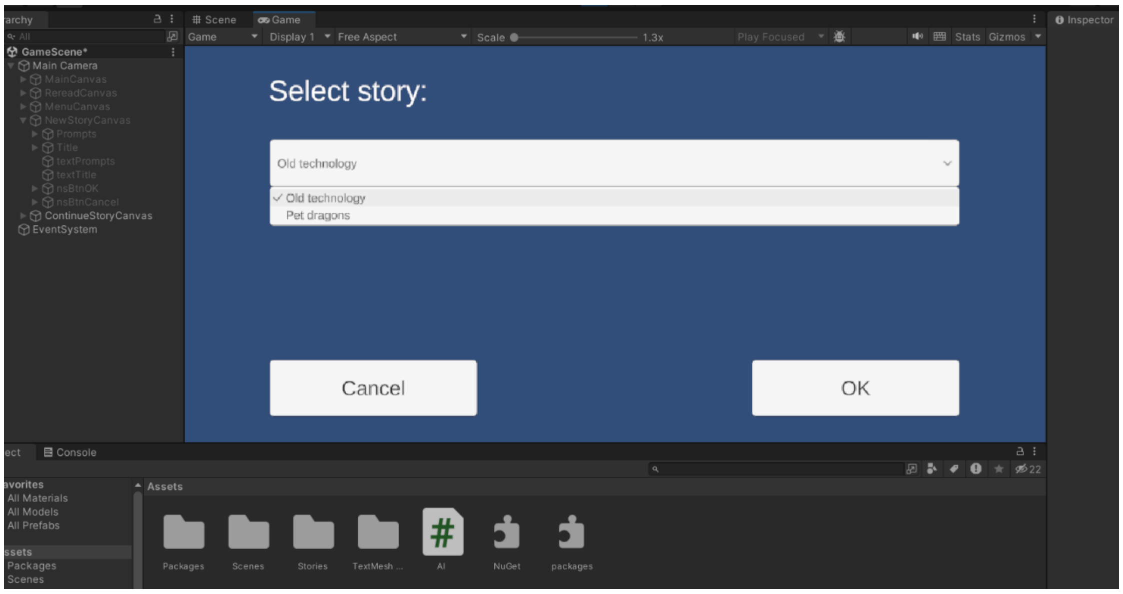
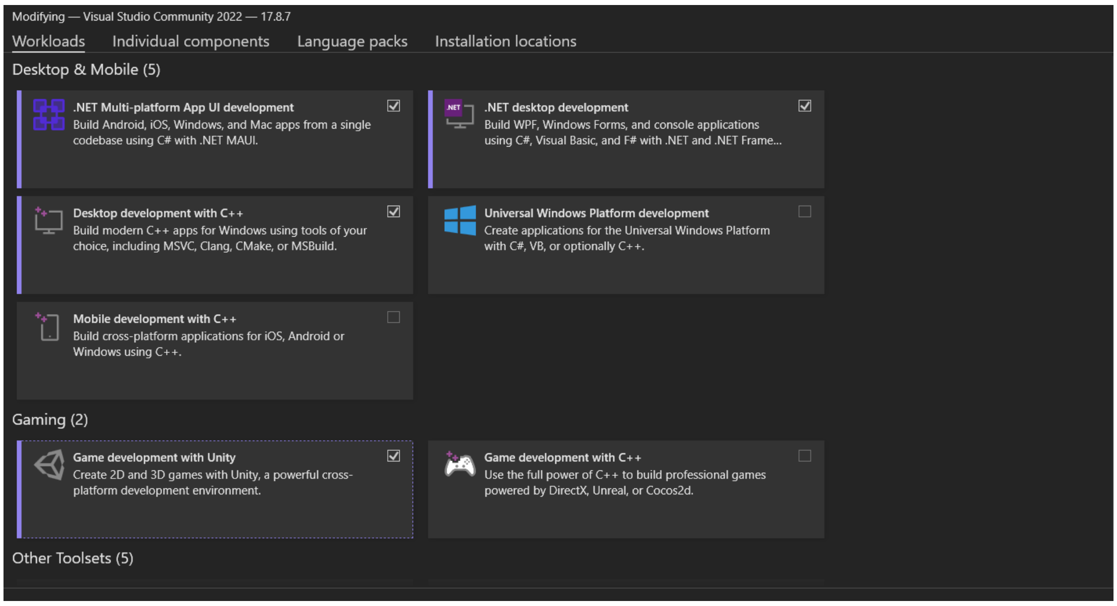


Figure 3.1.4 Continue Story Interface in Unity

3.2 Developing the game in Visual Studio

To be able to write the Unity script in Visual Studio, we need to install the necessary extension. After doing this, we are able to create a C# script and open it in Visual Studio.


Modifying — Visual Studio Community 2022 — 17.8.7

Workloads · Individual components · Language packs · Installation locations

Desktop & Mobile (5)

- ☑ .NET Multi-platform App UI development — Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.
- ☑ .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- ☑ Desktop development with C++ — Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- ☐ Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- ☐ Mobile development with C++ — Build cross-platform applications for iOS, Android or Windows using C++.

Gaming (2)

- ☑ Game development with Unity — Create 2D and 3D games with Unity, a powerful cross-platform development environment.
- ☐ Game development with C++ — Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Other Toolsets (5)

Figure 3.2.1 Installing the Visual Studio Extension

The next thing that needs to be done after creating the C# script, is to declare the elements used in the UI of the game: canvases, buttons and text fields.

```
public class GameScript : MonoBehaviour
{
    private OpenAIAPI api;
    private List<ChatMessage> messages;

    public GameObject menuCanvas;
    public Button btnStartNewStory;
    public Button btnContinueStory;
    public Button btnExit;

    public GameObject mainCanvas;
    public TMP_Text story;
    public Button option1;
    public Button option2;
    public Button btnMainMenu;

    public GameObject newStoryCanvas;
    public TMP_InputField inputPrompts;
    public TMP_InputField inputTitle;
    public Button nsBtnOK;
    public Button nsBtnCancel;
    public GameObject continueStoryCanvas;
    public TMP_Dropdown dropdownOldStories;
    public Button csBtnOK;
    public Button csBtnCancel;

    private const String storyPath = "Assets\\Stories";

}
```

We notice that the class contains “MonoBehaviour”. “MonoBehaviour” is a base class from which Unity scripts derive and contains various functions such as start() or update(). [10]

Using the class OpenAI API, which is the entry point to the API, we declare the variable that is going to store our API key.

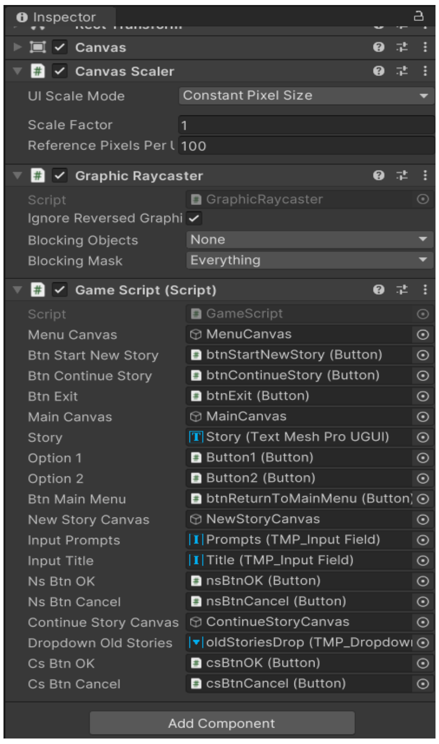
We then declare a list of ChatMessage elements to store the messages between the system, assistant and user. ChatMessage allows the sending and receiving messages from the API and contains the role and the content of a message.

GameObject is the base class for elements in Unity scenes. We use GameObject to declare our canvases. [11]

For the text, input field and dropdown elements we notice the TMP abbreviation. TMP stands for Text Mesh Pro, which is an improved version of Unity’s Text and Legacy Text Mesh. It provides more flexibility for text formatting. [12]

We also declare a path to the location we want to save our stories. The “Stories” folder will be automatically created if it does not exist already.

After declaring these elements in Visual Studio, we need to bind them with their Unity counterparts. For this, we drag and drop the script in any of the canvases’ Inspector window and select the corresponding elements.



Inspector

Canvas

Canvas Scaler

UI Scale Mode	Constant Pixel Size
Scale Factor	1
Reference Pixels Per U	100

Graphic Raycaster

Script	GraphicRaycaster
Ignore Reversed Graphi	✓
Blocking Objects	None
Blocking Mask	Everything

Game Script (Script)

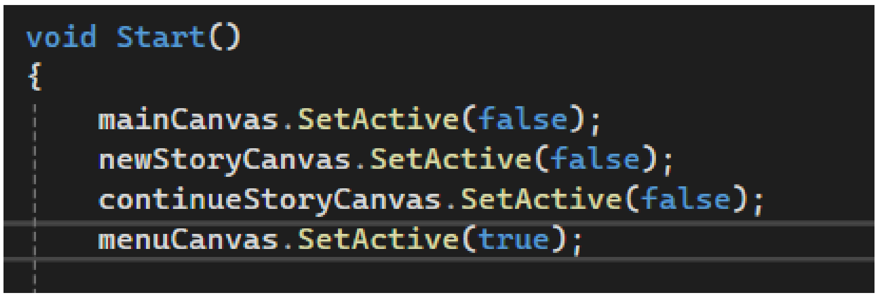
Script	GameScript
Menu Canvas	MenuCanvas
Btn Start New Story	btnStartNewStory (Button)
Btn Continue Story	btnContinueStory (Button)
Btn Exit	btnExit (Button)
Main Canvas	MainCanvas
Story	Story (Text Mesh Pro UGUI)
Option 1	Button1 (Button)
Option 2	Button2 (Button)
Btn Main Menu	btnReturnToMainMenu (Button)
New Story Canvas	NewStoryCanvas
Input Prompts	Prompts (TMP_Input Field)
Input Title	Title (TMP_Input Field)
Ns Btn OK	nsBtnOK (Button)
Ns Btn Cancel	nsBtnCancel (Button)
Continue Story Canvas	ContinueStoryCanvas
Dropdown Old Stories	oldStoriesDrop (TMP_Dropdown
Cs Btn OK	csBtnOK (Button)
Cs Btn Cancel	csBtnCancel (Button)

Add Component

Figure 3.2.2 Binding the Components

"MonoBehaviour.Start" is the first the first method called when starting the script. It is also called once in the lifetime of the script. [13]

"SetActive" is a method that modifies the visibility of a canvas. In the code below, we are setting all canvases, except one, as invisible, so when starting the game we are presented with the menu canvas. [14]



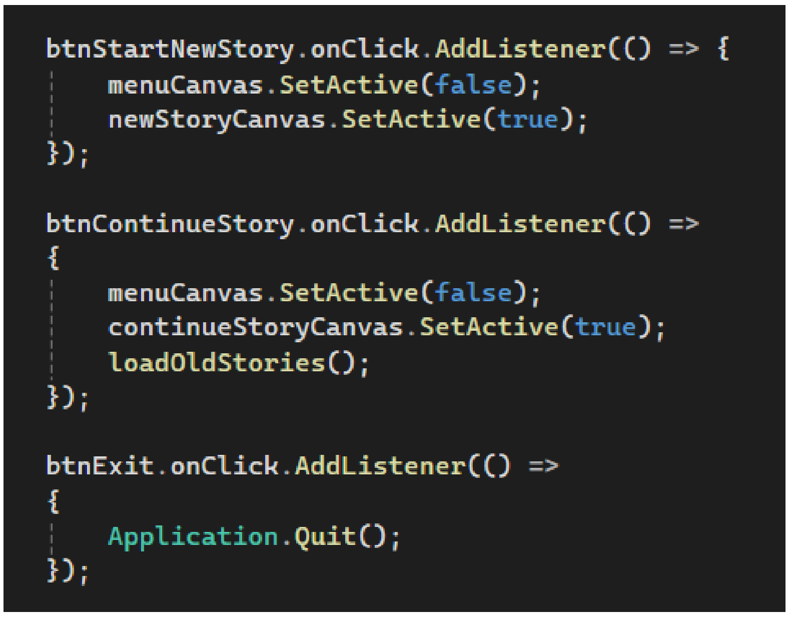
```
void Start()
{
    mainCanvas.SetActive(false);
    newStoryCanvas.SetActive(false);
    continueStoryCanvas.SetActive(false);
    menuCanvas.SetActive(true);
```

Figure 3.2.3 Modifying the Visibility for the Canvases

We then have to write the "onClick" functions for the buttons in the menu canvas. For this, we will use again the "setActive" function for the "StartNewStory" and "ContinueStory" buttons.

The “ContinueStory” button has an additional function called “loadOldStories”, which adds the titles of old stories saved in the dropdown element to be selected and restarted later.

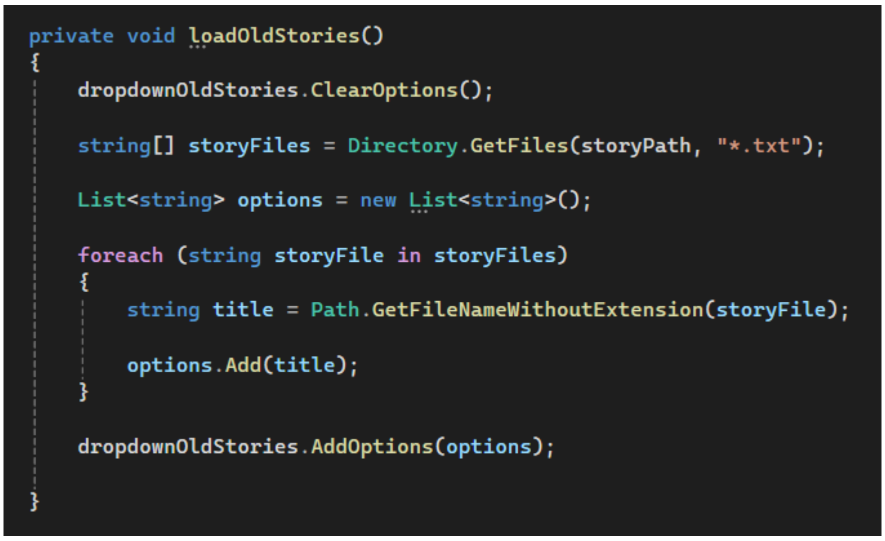
For the Exit button, we use the “Quit” function to exit the game.



```
btnStartNewStory.onClick.AddListener(() => {
    menuCanvas.SetActive(false);
    newStoryCanvas.SetActive(true);
});

btnContinueStory.onClick.AddListener(() =>
{
    menuCanvas.SetActive(false);
    continueStoryCanvas.SetActive(true);
    loadOldStories();
});

btnExit.onClick.AddListener(() =>
{
    Application.Quit();
});
```

Figure 3.2.4 Functions for Interacting with the Menu

In the “loadOldStories” function, we first clear the initial options of the Dropdown and save the file titles in the “storyFiles” array. Then, for each file in the “storyFiles” array, the name of the file is saved and added to the dropdown.



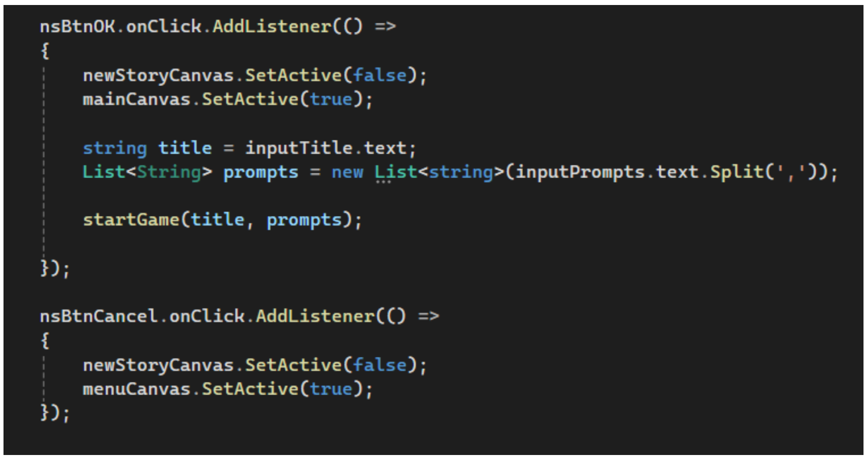
```
private void loadOldStories()
{
    dropdownOldStories.ClearOptions();

    string[] storyFiles = Directory.GetFiles(storyPath, "*.txt");

    List<string> options = new List<string>();

    foreach (string storyFile in storyFiles)
    {
        string title = Path.GetFileNameWithoutExtension(storyFile);

        options.Add(title);
    }

    dropdownOldStories.AddOptions(options);

}
```

Figure 3.2.5 Function for Adding the Old Stories to the Dropdown

Moving on to the "NewStory" Canvas, we have two "onClick" events.

After inputting the title and prompts that we want included in the story and clicking the OK button, the "startGame" function is called.

The Cancel button leads us back to the main menu.

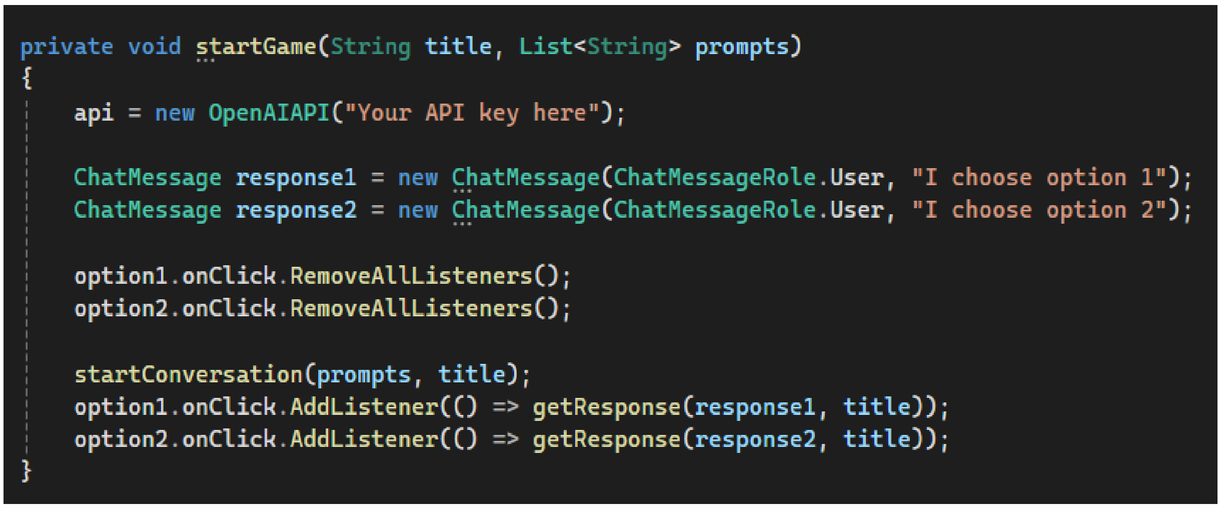


```
nsBtnOK.onClick.AddListener(() =>
{
    newStoryCanvas.SetActive(false);
    mainCanvas.SetActive(true);

    string title = inputTitle.text;
    List<String> prompts = new List<string>(inputPrompts.text.Split(','));

    startGame(title, prompts);

});

nsBtnCancel.onClick.AddListener(() =>
{
    newStoryCanvas.SetActive(false);
    menuCanvas.SetActive(true);
});
```

Figure 3.2.6 Functions for Starting a Story and returning to the Menu

In the "startGame" function, we initialize an API instance with a provided API key and two ChatMessage objects which represent the user responses. To ensure that there are no duplicate event handlers, we remove all existing listeners using the "RemoveAllListeners" method. We then call the "startConversation" function which sends the title, prompts and the initial system message to the chat model.

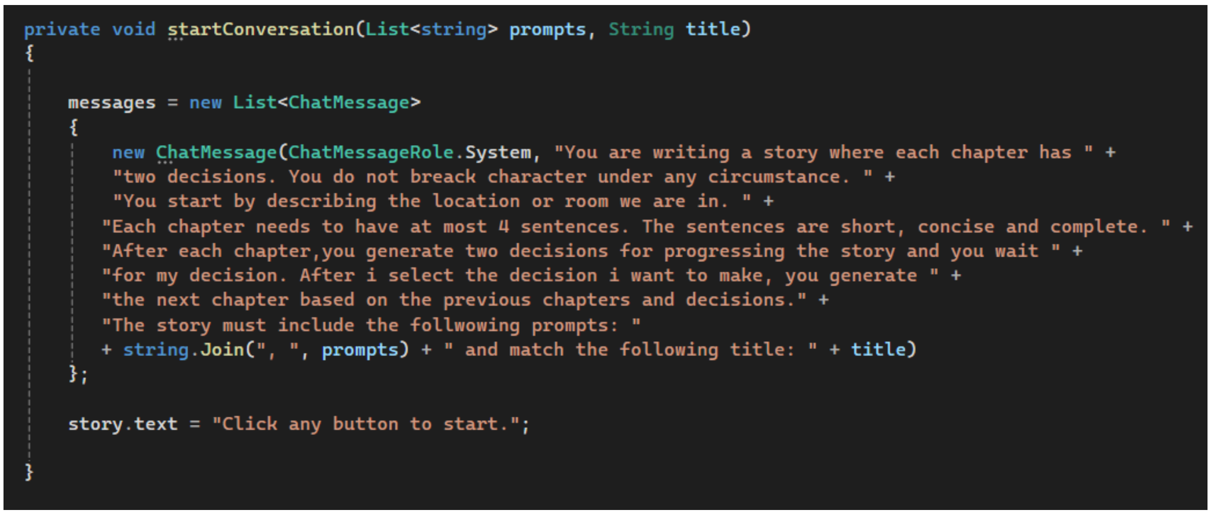
Finally, we attach event listeners to the buttons, so that when the button is clicked, the "getResponse" function is called, allowing the user to start and continue the story.



```
private void startGame(String title, List<String> prompts)
{
    api = new OpenAIAPI("Your API key here");

    ChatMessage response1 = new ChatMessage(ChatMessageRole.User, "I choose option 1");
    ChatMessage response2 = new ChatMessage(ChatMessageRole.User, "I choose option 2");

    option1.onClick.RemoveAllListeners();
    option2.onClick.RemoveAllListeners();

    startConversation(prompts, title);
    option1.onClick.AddListener(() => getResponse(response1, title));
    option2.onClick.AddListener(() => getResponse(response2, title));
}
```

Figure 3.2.7 Function for Interacting with the Chat Model

The "startConversation" function receives the title and the list of prompts inputted by the player. Using these, we are composing the initial message that will provide instructions for the chat model.



```
private void startConversation(List<string> prompts, String title)
{

    messages = new List<ChatMessage>
    {
        new ChatMessage(ChatMessageRole.System, "You are writing a story where each chapter has " +
        "two decisions. You do not breack character under any circumstance. " +
        "You start by describing the location or room we are in. " +
       "Each chapter needs to have at most 4 sentences. The sentences are short, concise and complete. " +
       "After each chapter,you generate two decisions for progressing the story and you wait " +
       "for my decision. After i select the decision i want to make, you generate " +
       "the next chapter based on the previous chapters and decisions." +
       "The story must include the follwowing prompts: "
       + string.Join(", ", prompts) + " and match the following title: " + title)
    };

    story.text = "Click any button to start.";

}
```

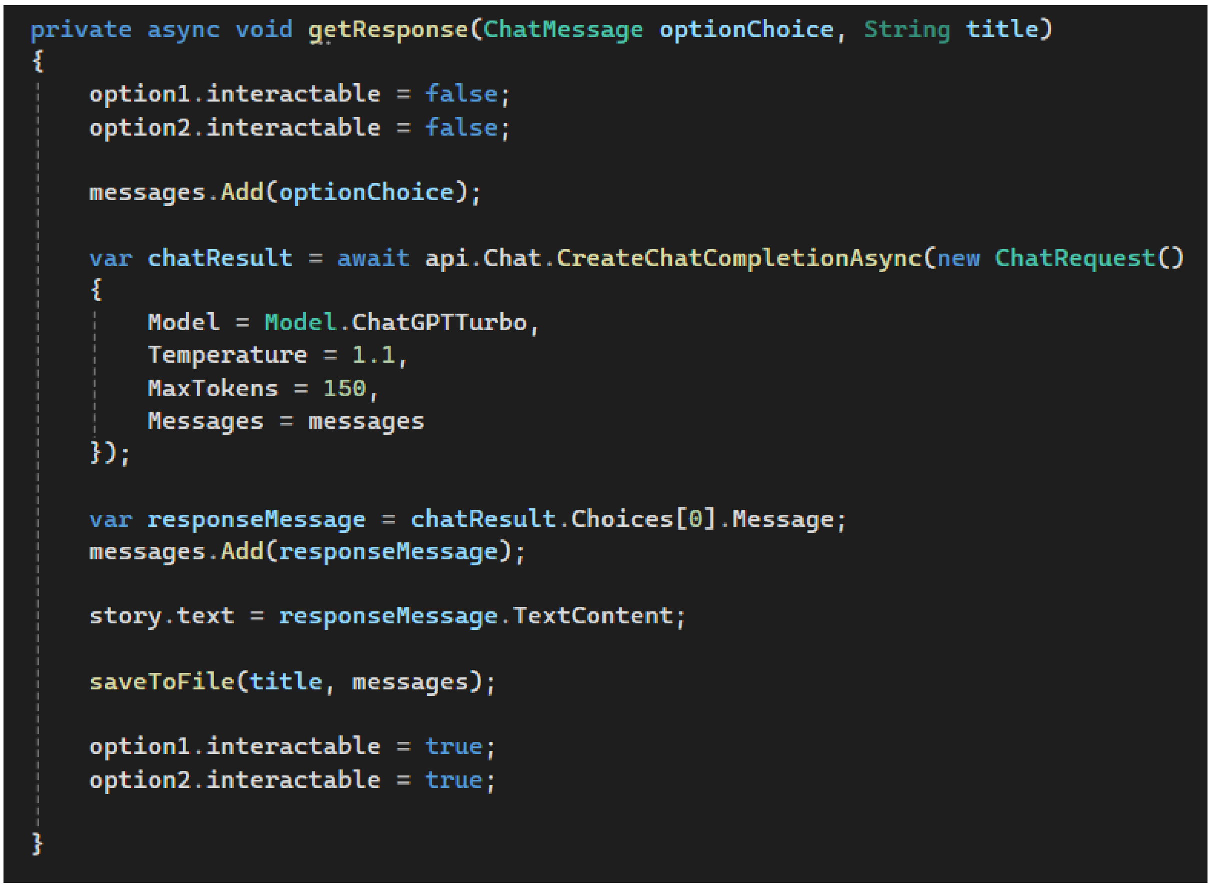
Figure 3.2.8 Function for Sending the Initial Message to the Chat Model

The asynchronous function "getResponse" starts by disabling the two decision buttons to avoid multiple inputs while processing the responses. The option chosen by the user is added to the messages list. Then, an API request is made to the selected chat model, in our case GPT Turbo, with the specified parameters: a high temperature to allow more creativity, a token limit of 150, and the messages list.

After receiving the response, it is extracted from the Choices array and stored in the variable "respondeMessage", then added to the messages list. The Choices array stores all possible generated responses. By default, only one message is received, but it can also return multiple. In the “getResponse” function, “chatResult.Choices[0].Message” is used to retrieve the first generated response from the Choices array, located at index 0.

The response is then displayed in the “story.text” field and the messages list is saved in a file using the “saveToFile” function.

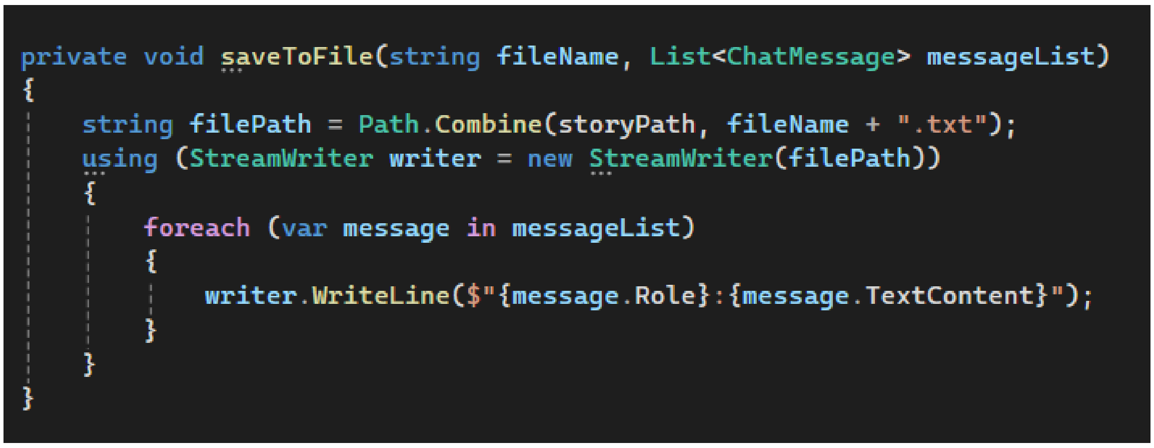
Finally, the two buttons are reenabled to allow the user to continue the story.



```
private async void getResponse(ChatMessage optionChoice, String title)
{
    option1.interactable = false;
    option2.interactable = false;

    messages.Add(optionChoice);

    var chatResult = await api.Chat.CreateChatCompletionAsync(new ChatRequest()
    {
        Model = Model.ChatGPTTurbo,
        Temperature = 1.1,
        MaxTokens = 150,
        Messages = messages
    });

    var responseMessage = chatResult.Choices[0].Message;
    messages.Add(responseMessage);

    story.text = responseMessage.TextContent;

    saveToFile(title, messages);

    option1.interactable = true;
    option2.interactable = true;

}
```

Figure 3.2.9 Function for Sending and Receiving Messages to and from the Chat Model

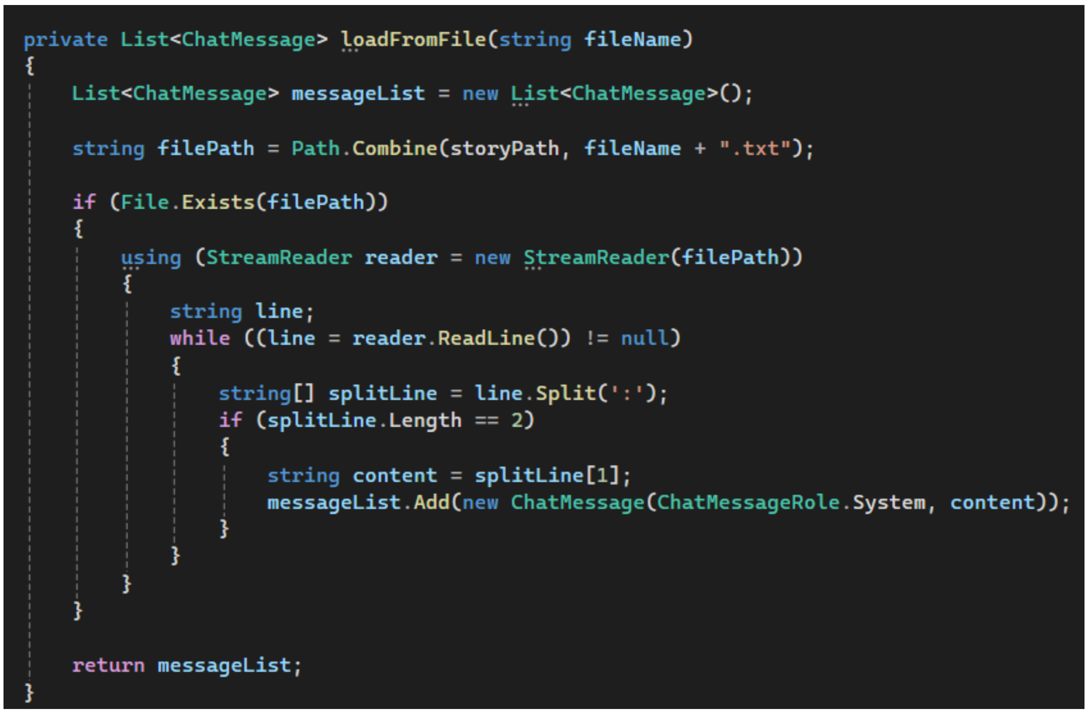
The “saveToFile” function is first constructing the file location by concatenating the path declared in the beginning with the file name passed in the function header. Then, for each message in the list, its role and content are written into the text file.



```
private void saveToFile(string fileName, List<ChatMessage> messageList)
{
    string filePath = Path.Combine(storyPath, fileName + ".txt");
    using (StreamWriter writer = new StreamWriter(filePath))
    {
        foreach (var message in messageList)
        {
            writer.WriteLine($"{message.Role}:{message.TextContent}");
        }
    }
}
```

Figure 3.2.10 Function for Saving Stories to Files

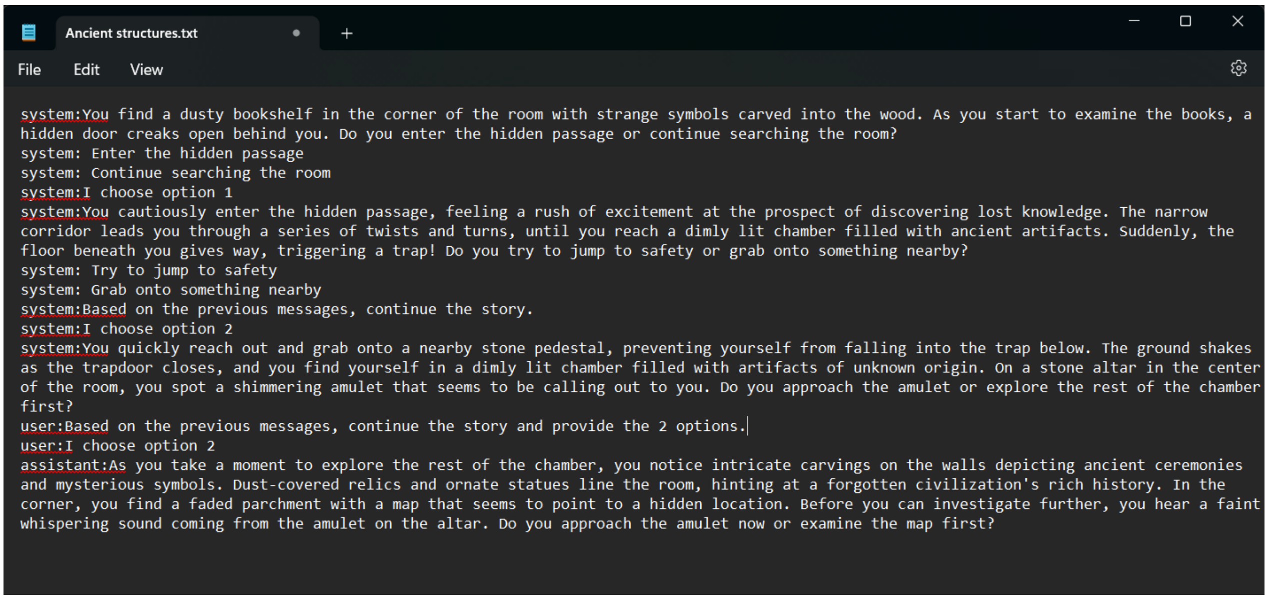
The “loadFromFile” function is used in the context of continuing a story. After selecting the title of the story we want to continue, it is stored in the filename variable. We then declare a new ChatMessage list and construct the file path. Each line read from the file is split into role and content, the latter being saved in the content variable. The content is then added to the message list with the system role.



```
private List<ChatMessage> loadFromFile(string fileName)
{
    List<ChatMessage> messageList = new List<ChatMessage>();

    string filePath = Path.Combine(storyPath, fileName + ".txt");

    if (File.Exists(filePath))
    {
        using (StreamReader reader = new StreamReader(filePath))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                string[] splitLine = line.Split(':');
                if (splitLine.Length == 2)
                {
                    string content = splitLine[1];
                    messageList.Add(new ChatMessage(ChatMessageRole.System, content));
                }
            }
        }
    }

    return messageList;
}
```

Figure 3.2.11 Function for Loading Stories from Files

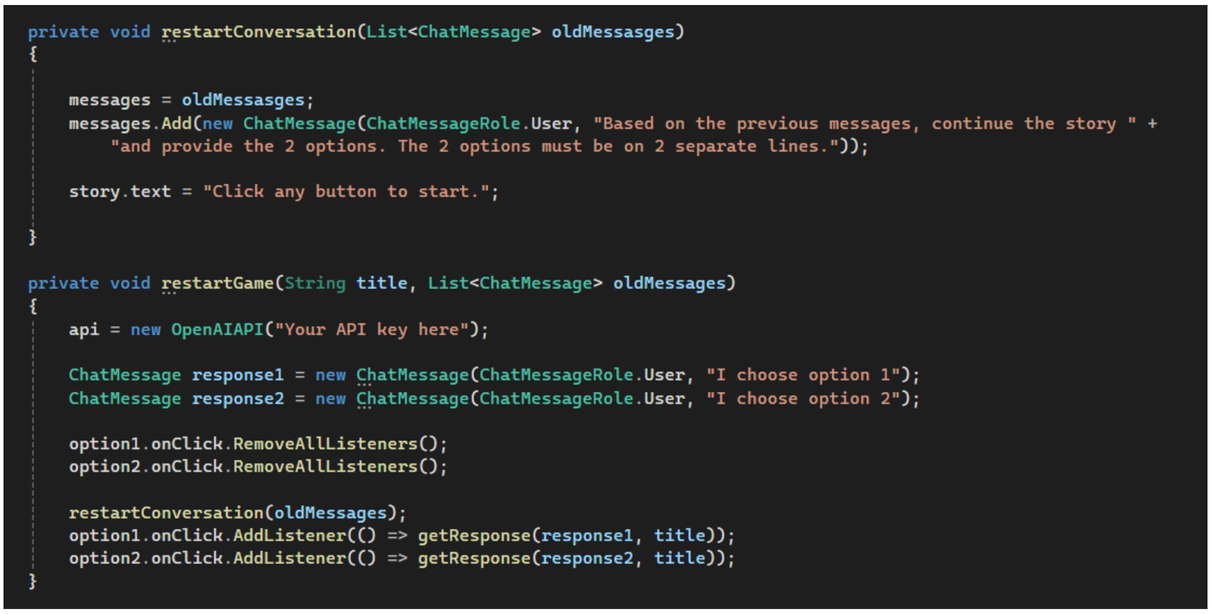
The example story in the file shown in Figure 3.2.12 clearly shows the interactions between the system, the assistant and the user.



```
system:You find a dusty bookshelf in the corner of the room with strange symbols carved into the wood. As you start to examine the books, a
hidden door creaks open behind you. Do you enter the hidden passage or continue searching the room?
system: Enter the hidden passage
system: Continue searching the room
system:I choose option 1
system:You cautiously enter the hidden passage, feeling a rush of excitement at the prospect of discovering lost knowledge. The narrow
corridor leads you through a series of twists and turns, until you reach a dimly lit chamber filled with ancient artifacts. Suddenly, the
floor beneath you gives way, triggering a trap! Do you try to jump to safety or grab onto something nearby?
system: Try to jump to safety
system: Grab onto something nearby
system:Based on the previous messages, continue the story.
system:I choose option 2
system:You quickly reach out and grab onto a nearby stone pedestal, preventing yourself from falling into the trap below. The ground shakes
as the trapdoor closes, and you find yourself in a dimly lit chamber filled with artifacts of unknown origin. On a stone altar in the center
of the room, you spot a shimmering amulet that seems to be calling out to you. Do you approach the amulet or explore the rest of the chamber
first?
user:Based on the previous messages, continue the story and provide the 2 options.
user:I choose option 2
assistant:As you take a moment to explore the rest of the chamber, you notice intricate carvings on the walls depicting ancient ceremonies
and mysterious symbols. Dust-covered relics and ornate statues line the room, hinting at a forgotten civilization's rich history. In the
corner, you find a faded parchment with a map that seems to point to a hidden location. Before you can investigate further, you hear a faint
whispering sound coming from the amulet on the altar. Do you approach the amulet now or examine the map first?
```

Figure 3.2.12 Example of a File

To continue a story, the “restartConversation” and “restartGame” functions are used, which are similar to the previous ones.



```
private void restartConversation(List<ChatMessage> oldMessasges)
{

    messages = oldMessasges;
    messages.Add(new ChatMessage(ChatMessageRole.User, "Based on the previous messages, continue the story " +
        "and provide the 2 options. The 2 options must be on 2 separate lines."));

    story.text = "Click any button to start.";

}

private void restartGame(String title, List<ChatMessage> oldMessages)
{
    api = new OpenAIAPI("Your API key here");

    ChatMessage response1 = new ChatMessage(ChatMessageRole.User, "I choose option 1");
    ChatMessage response2 = new ChatMessage(ChatMessageRole.User, "I choose option 2");

    option1.onClick.RemoveAllListeners();
    option2.onClick.RemoveAllListeners();

    restartConversation(oldMessages);
    option1.onClick.AddListener(() => getResponse(response1, title));
    option2.onClick.AddListener(() => getResponse(response2, title));
}
```

Figure 3.2.13 Functions for Restarting a Story

Conclusion

Creating this application gave me the opportunity to learn about both game development and about the capabilities of artificial intelligence.

The AI has proven a capable storyteller, offering the player a unique experience tailored to their choices. Using AI has enriched the gaming experience and showcased its potential in creative fields.

However, while AI can be efficient and adaptable, it is important to approach its use in creative domains with caution. Creativity is a human trait, something that AI cannot fully replicate. By relying on AI, we risk losing emotion, intuition, and culture, qualities that are irreplaceable.

In conclusion, while AI can be a useful tool for creative processes, it should be used to enhance, rather than replace, human creativity.

Table of Figures

Bibliography

[1] E. Christopoulou and S. Xinogalos, “Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices,” *International Journal of Serious Games,* 2017.

[2] Unity, “Canvas,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/UICanvas.html.

[3] Unity, “Visual Components,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/UIVisualComponents.html.

[4] Unity, “Interaction Components,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/UIInteractionComponents.html.

[5] OpenAI, “OpenAI Platform - Introduction,” [Online]. Available: https://platform.openai.com/docs/introduction.

[6] OpenAI, “OpenAI Platform - Text Generation,” [Online]. Available: https://platform.openai.com/docs/guides/text-generation?lang=python.

[7] OpenAI, “OpenAI Platform - api-reference,” [Online]. Available: https://platform.openai.com/docs/api-reference/chat.

[8] R. Pincombe, “OpenAI API for C# / .NET,” 2022. [Online]. Available: https://rogerpincombe.com/openai-dotnet-api.

[9] Microsoft, “A tour of the C# language,” [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview.

[10] Unity, “MonoBehaviour,” [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.

[11] Unity, “GameObject,” [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.html.

[12] Unity, “QuickStart to TextMesh Pro,” [Online]. Available: https://learn.unity.com/tutorial/working-with-textmesh-pro#.

[13] Unity, “MonoBehaviour.Start(),” [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html.

[14] Unity, “GameObject.SetActive,” [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html.