Ministry of Higher Education
MISR Higher Institute for Commerce & Computers
Computer Science Department

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

Education system based on Augmented Reality and game programming (EduAR)

Under the supervision of

ASSOC.PROF/Lobna Mohamed Abou El-Magd
Eng/ Esraa Asem

Teamwork

Omar Mohammed Sobhy
Abdulrhman Hagag Elfallah
Ahmed Elhossiny Shindy
Ali Mohammed Saafan
Zakaria Rafeeq Mohamed

Index

Chapter 1: “Introduction”

Introduction:

Education will remain the cornerstone upon which nations are built. While accessing knowledge has become easier than ever, traditional methods of learning have become boring and useless. Technology has had a significant impact on people's lives, becoming one of the most distracting factors. Scientists have estimated that modern humans have a shorter attention span than their ancestors. The use of electronic games has become one of the most distracting and influential areas. Entertainment has become an integral part of the children’s life in this era. With our effort to benefit society and fulfill our national and scientific duties, we have come up with the idea of this project, in which we aim to use augmented reality technology.

EduAR: This system is a game supported by augmented reality that enhances the learning process, which we will explain in detail later. The aim of this system is to contribute to the creation of an interactive and miniature learning environment by applying many scientific and educational theories during its construction. For example, we rely on the system of moral reward, continuous encouragement of the user in case of any accomplishment while using the system, and the system of audio and visual interaction, and sometimes touch interaction, which enhances positive feelings and makes the learning process more enjoyable and easier.

1.1 Game Development life cycle:

Before we begin discussing the topics of this chapter, we must first familiarize ourselves with the game development life cycle, which is visually illustrated in this image

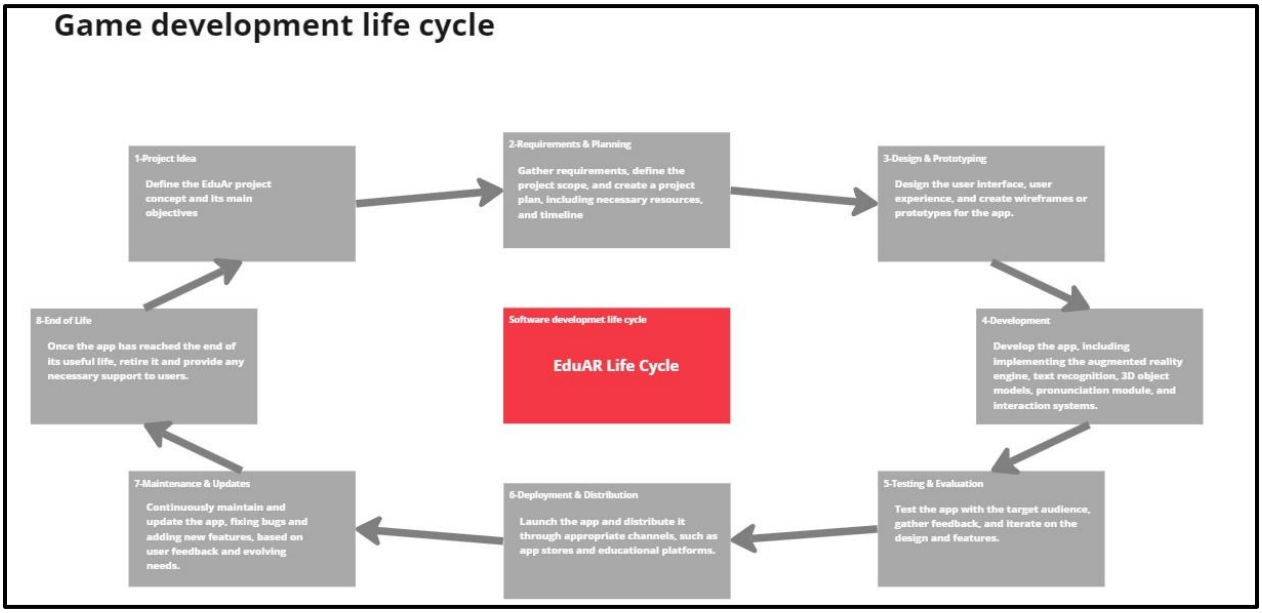


Figure 1 – EduAr Game development life cycle

1.2 IDEA :

- EduAr is a system that is a video game supported by augmented reality.
- It uses text recognition systems to recognize spoken text and convert it into sounds and words.
- It uses 3D object models known to the target audience and uses animations to add enjoyment to the learning process.
- Its aim is to teach children how to pronounce letters and reinforce correct pronunciation.
- It enhances the learning process by applying scientific and educational theories, such as moral reward and continuous encouragement of the user.

- It creates an interactive and enjoyable learning environment through audio, visual, and touch interactions.

1.3 Problems:

- Keeping up with modern learning methods and avoiding traditional methods without compromising the scientific value
- encouraging effective interaction between the target audience and the system in virtual environment.
- addressing the problem of lack of attention and boredom in the learning process.
- Taking into account individual differences among the target audience and dealing with children who cannot distinguish the meanings of silent letters and words through the visual and auditory aspects.
- Dealing with the difficulty of controlling the motor aspect for children in this age group, which is characterized by excessive activity, by taking the motor aspect of the system into account.

1.4 Goals:

- Creating an interactive and miniature educational system that utilizes advanced technologies like augmented reality to teach children how to pronounce letters and reinforce correct pronunciation.

- Enhancing the learning process by applying scientific and educational theories, such as moral reward and continuous encouragement of the user.
- Promoting effective interaction between the target audience and the system.
- addressing the problem of lack of attention and boredom in the learning process.
- Taking into account individual differences among the target audience and dealing with children who cannot distinguish the meanings of silent letters and words through the visual and auditory aspects.
- Dealing with the difficulty of controlling the motor aspect for children in this age group, which is characterized by excessive activity, by taking the motor aspect of the system into account.
- Providing an enjoyable and easy learning experience for children while maintaining the scientific value of the learning material.
- contributing to the development of modern learning methods that move away from traditional methods.

1.5 Tools:

- **Augmented Reality** (create interactive game)
- **Unity** (developing augmented reality,2D and 3D games)
- **C#** (OOP language)
- **Visual Studio 2022** (IDE that is used in game developing)

- **Unity ARFoundation package** (create augmented reality plane and surface detection)
- **Wit.ai** (convert text to speech with a human voice)
- **Speechly** (detect the user speech accurately, speech to text)
- **Blender** (creating, editing and animating 3d models)
- **Text Mesh Pro** (text rendering solution)
- **Photoshop** (designing user interfaces and visual assets)

Chapter2: “Background”

In this chapter, we will talk about the tools used in building our system, the features and how each tool works, starting with the Augmented Reality technology and passing through every tool used in building the system.

2.1 Augmented Reality:

Augmented reality (AR) is a technology that overlays digital information, such as images, graphics, or sound, onto the real world in real-time. AR is a type of mixed reality that blends the physical and virtual worlds, enhancing the user's perception of reality, AR technology typically works through a mobile device or wearable device, such as a smartphone, tablet, or smart glasses. The device uses its sensors, such as the camera and GPS, to track the user's location and surroundings, then digital content is superimposed onto the real-world view of the device, creating the illusion that the virtual objects exist in the physical environment, AR technology has a wide range of applications, including entertainment, education, marketing, and training.

It can be used to create interactive games, enhance educational experiences, provide virtual tours of real-world locations, and create immersive marketing campaigns, among other uses.

Overall, augmented reality technology has the potential to transform the way we interact with the world around us, creating new opportunities for entertainment, education, and communications.

2.1.1 The way of AR works:

- **Capture the real-world environment:** The AR device, such as a smartphone or tablet, uses its camera and other sensors to capture the user's physical environment. This data is then sent to the AR software for processing. Here is a Figure1 and Figure that illustrates this step.

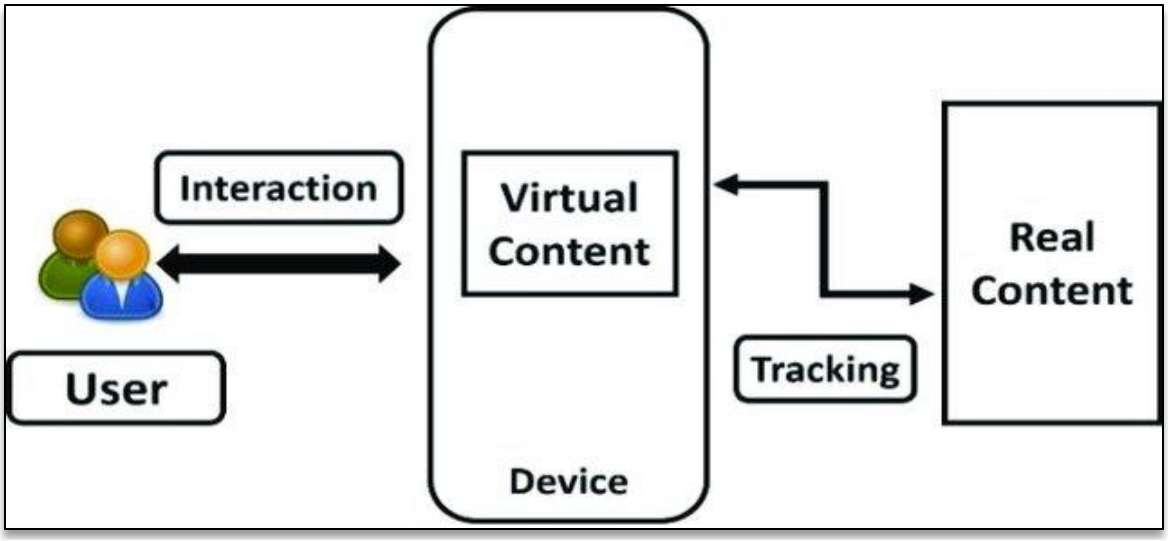


Figure 2

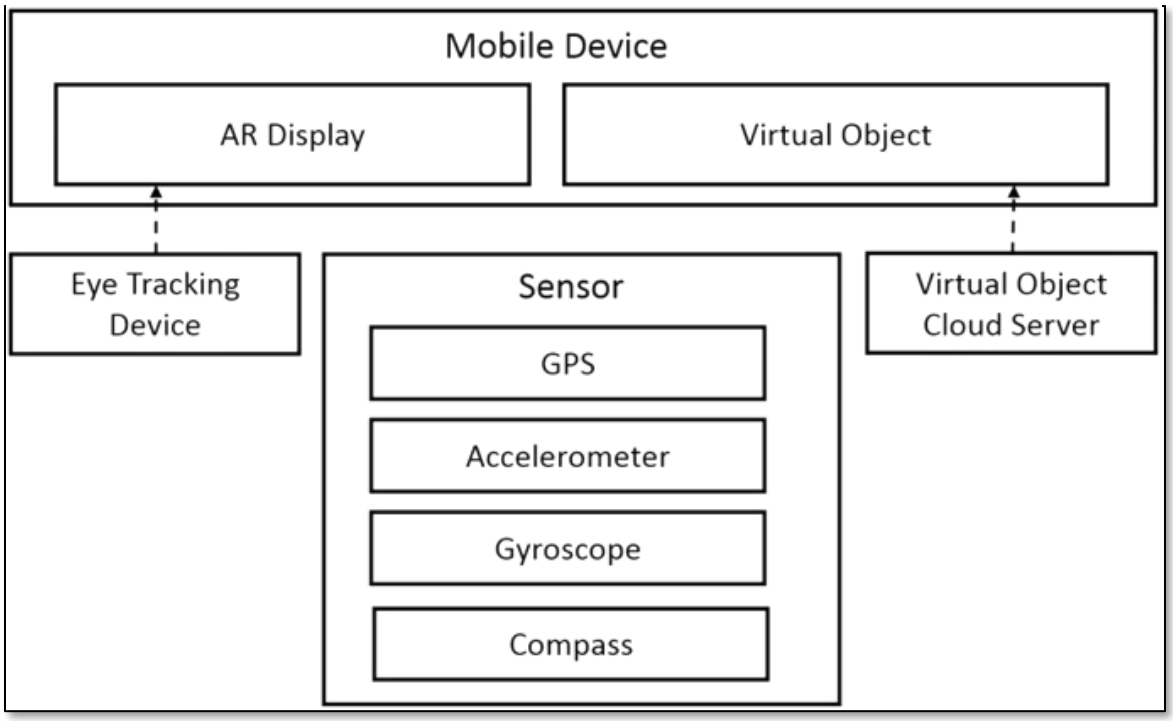


Figure 3

- **Analyze the environment:** The AR software analyzes the data captured by the device's sensors to determine the user's location, orientation, and other environmental factors. This information is used to generate a 3D model of the user's environment .
- **Generate digital content:** The AR software generates digital content, such as images or 3D models, based on the user's

environment and the specific AR experience. This content is then superimposed onto the user's view of the real world.

- **Overlay digital content onto the real world:** The AR software superimposes the digital content onto the user's view of the real world, using the device's screen or a dedicated AR display. This creates the illusion that the digital objects are part of the real world.

- **Track the user's movement:** The AR software continues to analyze the user's environment and track their movement, adjusting the digital content as needed to maintain the illusion of integration with the real world , The following image (Figure 3) illustrates how Augmented Reality technology captures images and interacts with different directions. It uses a real image and integrates it with a virtual object to produce Augmented Reality.

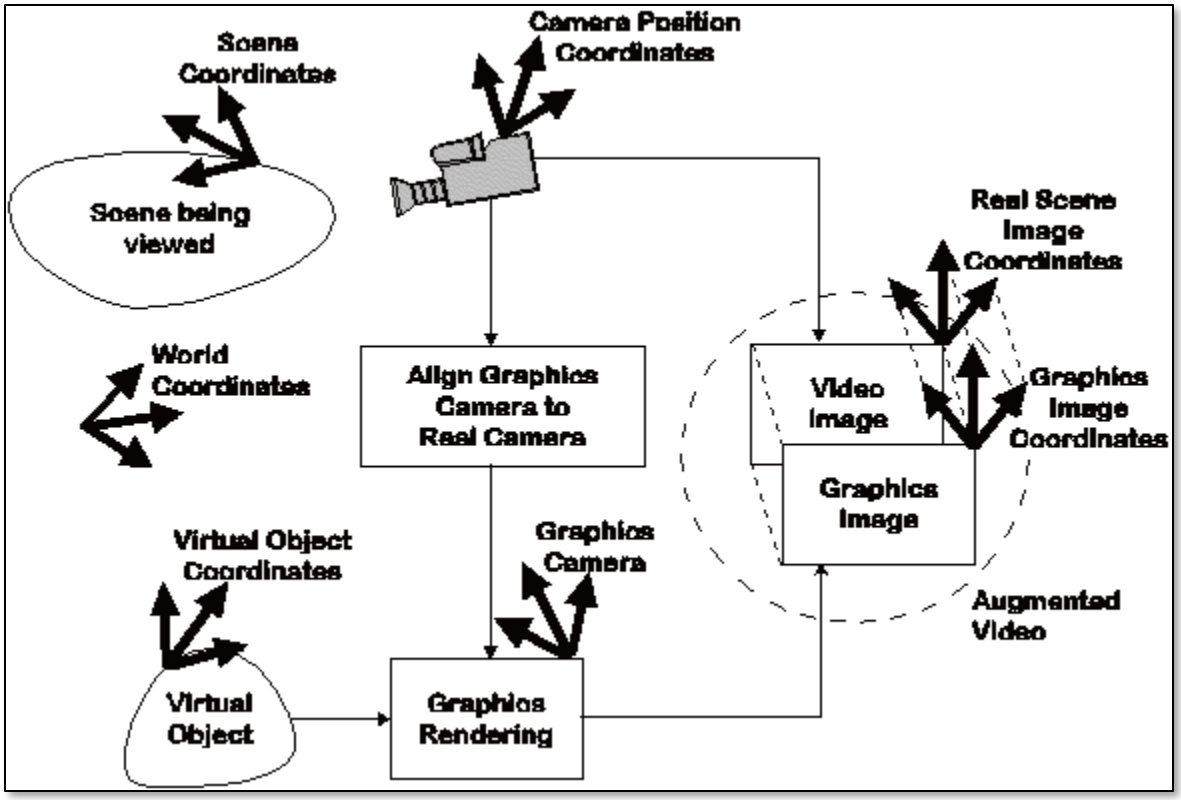


Figure 4 Tracking the interaction between objects and user's movement

2.1.2 The use AR in our project:

Augmented Reality allows for the creation of an interactive and virtual environment that enables the user to interact with letters and words in a fun and easy way. It can also be used to enhance understanding and memory of different educational concepts and ideas. Ultimately, augmented reality technology is a vital part of this project and helps improve the learning experience for the user.

2.1.3 Virtual reality:

(VR) is a technology that uses a headset or a display to create a fully immersive, simulated environment that users can interact with. VR typically involves wearing a headset that covers the eyes and ears, and often includes hand-held controllers that allow users to interact with virtual objects.

2.1.4 Mixed Reality:

Mixed reality (MR) is a technology that blends elements of both virtual reality (VR) and augmented reality (AR) to create a new type of immersive experience. In mixed reality, virtual objects are overlaid onto the real world, but they are able to interact with physical objects and the environment in a more natural way than in AR.

Mixed reality achieves this by using sensors and cameras to map the real world and track the user's position and movements in real time, and then overlaying virtual objects in a way that appears to be anchored to the physical world. This enables users to interact with virtual objects in a more natural way, such as picking up a virtual object and moving it around in the real world.

Figure 5-Microsoft's HoloLens

One example of mixed reality technology is Microsoft's HoloLens, which is a headset that allows users to see and interact with virtual objects overlaid onto the real world. Another example is Magic Leap, a company that has developed a mixed reality headset that uses a combination of sensors and projectors to create realistic, interactive virtual objects that appear to be part of the real world.

2.1.5 Metaverse:

The metaverse is a term used to describe a hypothetical future iteration of the internet that is fully immersive and interactive, where users can engage with a virtual world that is more like real life than the internet we know today. The concept of the metaverse

is still evolving, but it generally refers to a shared, persistent virtual space that is created by the convergence of physical and virtual reality.

2.1.6 The difference between AR and VR:

Augmented reality (AR) and virtual reality (VR) are technologies that are used to create immersive digital experiences. AR involves overlaying digital content on top of the real world, so that users can see and interact with virtual objects in their physical environment. VR, on the other hand, involves creating a completely artificial environment that users can explore and interact with in a

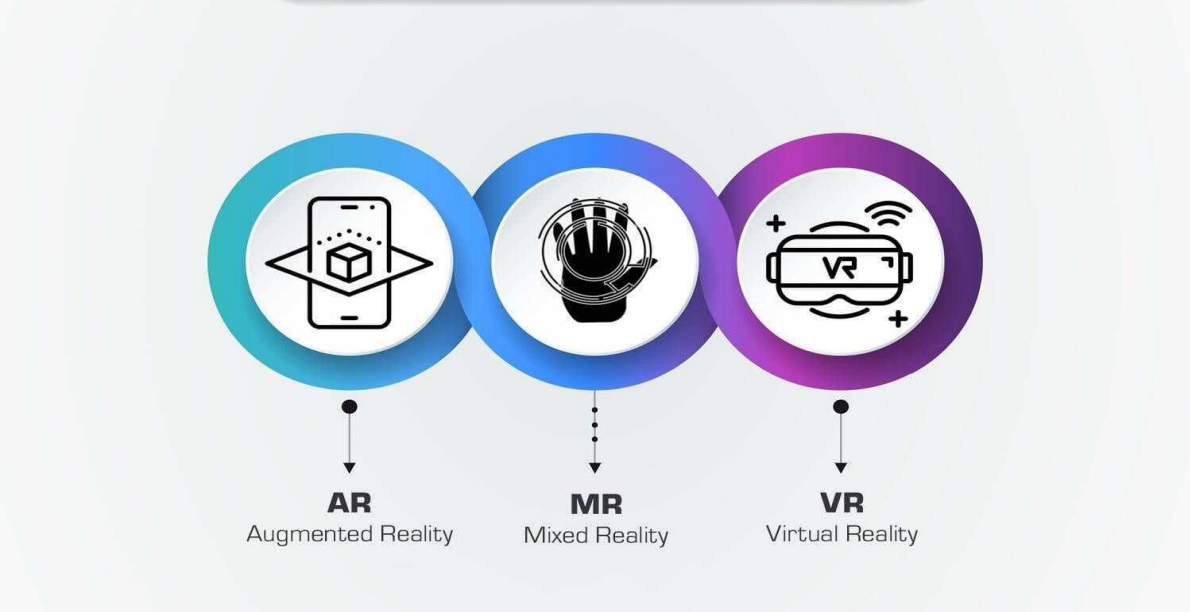


Figure 6

simulated way.

2.1.7 Metaverse vs AR and VR:

The main difference between the metaverse and AR/VR is that the metaverse is a broader concept that encompasses both AR and VR. While AR and VR are technologies used to create immersive digital experiences, the metaverse is a vision of a fully immersive and

interactive virtual world that is shared by many users. The metaverse is often described as a combination of AR and VR, as well as other technologies such as artificial intelligence, blockchain, and the internet of things, that work together to create a fully immersive digital world.

2.2 Unity:

Unity is a cross-platform game engine and development tool that is widely used for creating 2D and 3D games, as well as simulations, virtual reality and augmented reality applications, and other interactive content. Unity works by providing a variety of tools and features to help developers create their projects.

2.2.1 Core Features of Unity:

At its core, Unity is a game engine that provides a game development environment for creating and building games. It includes a variety of tools and features that help developers create game objects, add components to those objects, and create game logic and behaviors. The engine also includes tools for creating user interfaces, scripting, physics, and audio.

2.2.2 Scripting in Unity:

Unity supports a variety of programming languages, including C#, JavaScript, and Boo. Developers can use scripting languages to write code that controls the behavior of game objects and implements game logic and mechanics. Unity also provides access to a variety of third-party tools and plugins that can be used to enhance the development process.

Unity scripts are similar to regular programming in terms of syntax and structure, but they are tailored specifically for game development within

the Unity engine. Some key differences between Unity scripting and regular programming include:

Unity scripts are event-driven: Unity scripts are executed in response to specific events, such as when a game object is created, destroyed, or updated. This is different from traditional programming, where code is executed in a linear fashion.

Unity scripts use a component-based architecture: In Unity, game objects are composed of multiple components, and each component can have its own script. This allows developers to create modular, reusable code that can be attached to different game objects.

Unity scripts have access to Unity-specific APIs: Unity provides a wide range of APIs that allow developers to interact with the game engine and create game objects, animations, physics simulations, and more. These APIs are specific to Unity and are not available in traditional programming languages.

2.2.3 Cross-Platform Support:

One of the key features of Unity is its cross-platform support. Developers can create games and applications that can run on a variety of platforms, including desktop computers, mobile devices, gaming consoles, and virtual and augmented reality devices. Unity also provides tools for optimizing games and applications for specific platforms, such as mobile devices or gaming consoles.

2.2.4 Optimizing for Specific Platforms:

In summary, Unity is a game engine and development tool that provides a variety of tools and features for creating 2D and 3D games, simulations,

and other interactive content. It works by providing a game development environment that includes tools for creating game objects, adding components, scripting, physics, audio, and more. Unity also supports multiple programming languages and cross-platform development, making it suitable for a wide range of projects.

2.2.5 Unity components:

- **Hierarchy Window:**

Figure 7 – unity hierarchy window

- This is the hierarchy window. This is the hierarchical text representation of every object in the scene. It is where all the objects in your recently open scene are listed, along with their parent-child hierarchy.
- Each item in the scene has an entry in the hierarchy, so the two windows are linked. The hierarchy defines the structure of how objects are attached to one another.

- By default, the Hierarchy window lists GameObjects by order of creation, with the most recently created GameObjects at the bottom. We can reorder the GameObjects by dragging them up or down, or by making the parent or child GameObjects.

- **Scene View:**

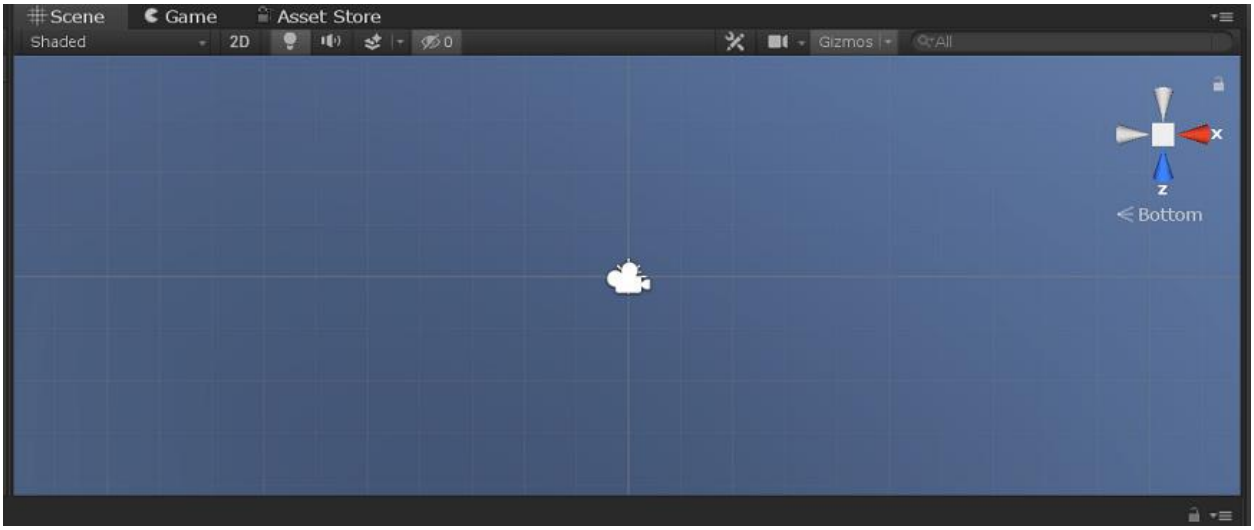


Figure 8 – unity Scene View

- This window is where we will create our scenes. This view allows you to navigate and edit your scene visually.
- The scene view can show a 2D or 3D perspective, depending on the type of project you are working on.
- We are using the scene view to select and position scenery, cameras, characters, lights, and all other types of GameObject.

- **Inspector Window:**

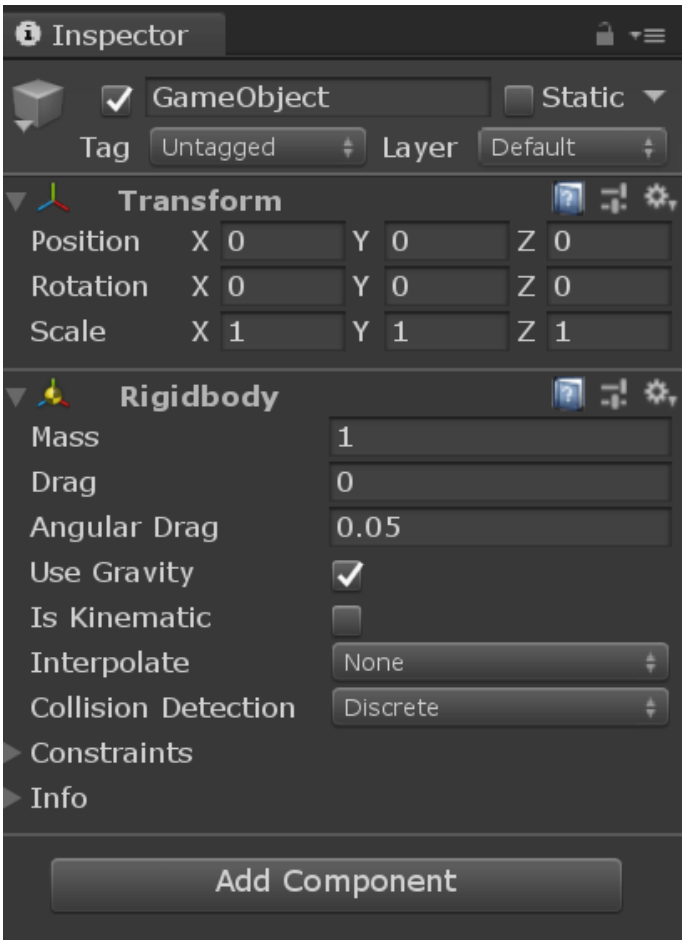


Figure 9 – unity inspector window

- The Inspector window allows you to view and edit all the properties of the currently selected object.
- Since different types of objects have different sets of properties, the layout and contents of the inspector window will vary.
- In this window, you can customize aspects of each element that is in the scene.
- You can select an object in the Hierarchy window or double click on an object in the scene window to show its attributes in the inspector panel.
- The inspector window displays detailed information about the currently selected GameObject, including all attached components

and their properties, and allows you to modify the functionality of GameObjects in your scene.

- **Game Window:**

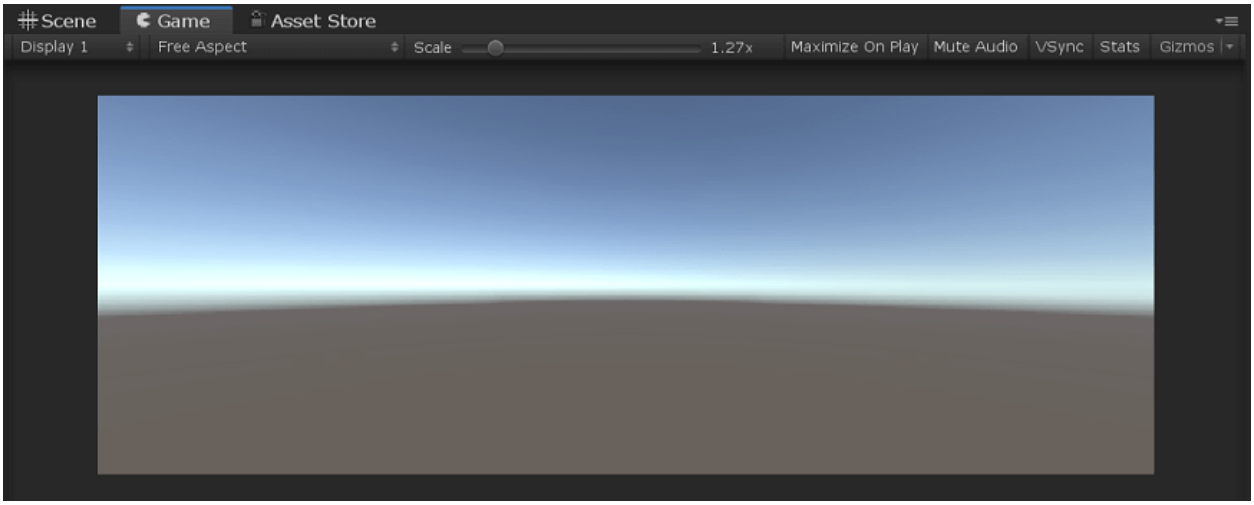


Figure 10–Unity Game window

- This window shows the view that the main camera sees when the game is playing. Means here, you can see a preview window of how the game looks like to the player.
- It is representative of your final game. You will have to use one or more cameras to control what the player actually sees when they are playing your game.

2.3 C#:

C# is a programming language that is widely used for developing a variety of applications, including games, desktop applications, web applications, and mobile applications. It was developed by Microsoft in the early 2000s as part of the .NET framework and has since become a popular language for software development.

2.3.1 Object-Oriented Programming Concepts in C#:

C# is an object-oriented language that is designed to be simple, modern, and efficient. It supports a wide range of programming concepts,

including classes, inheritance, interfaces, and polymorphism. C# is a statically typed language, which means that data types are determined at compile time rather than at runtime.

2.3.2 Integration with .NET Framework:

One of the key features of C# is its integration with the .NET framework, which provides a wide range of libraries and tools for developing applications. These libraries include support for common programming tasks, such as file input/output, networking, and database access. C# also supports the development of Windows Presentation Foundation (WPF) applications, which use XAML to create user interfaces.

2.3.3 Use in Game Development:

C# is widely used for game development, particularly with the Unity game engine. Unity provides a scripting API that allows developers to write C# scripts to control the behavior of game objects. C# scripts can be attached to game objects in Unity, allowing them to respond to user input, interact with other game objects, and implement game mechanics. For example, you could use C# to create a script that defines the movement and behavior of a character in your game, or to create a script that controls the behavior of enemies or other non-player characters, C# is also used for developing web applications with ASP.NET, mobile applications with Xamarin, and desktop applications with Windows Forms and WPF.

2.3.4 Game Development concept:

Game development is the process of creating interactive electronic games, which can be played on various platforms such as personal computers, gaming consoles, mobile devices, and virtual reality devices. Game development involves a range of activities, including designing game mechanics, creating game assets, programming game logic, and testing the game to ensure that it is fun and engaging to play.

The history of game development dates back to the early days of computing, when programmers created simple games as a form of entertainment. One of the earliest known computer games is "Tennis for Two," which was created by physicist William Higginbotham in 1958 using an oscilloscope and analog computer. In the 1960s and 1970s, game development became more widespread, with the creation of games like "Spacewar!" and "Pong.".

The video game industry really took off in the 1980s with the release of the first home video game consoles, such as the Atari 2600 and the Nintendo Entertainment System (NES). Since then, game development has grown into a multi-billion dollar industry, with thousands of developers creating games for a wide range of platforms and audiences

And we will be using game development technology to implement the concept of reward, where upon completing a certain number of stages, the player will be rewarded by being shown a small game within the system, in order to help them relax and take a break during the learning process.

2.4 Visual Studio 2022:

Visual Studio 2022 is the latest version of Microsoft's integrated development environment (IDE) for software development. It was released in November 2021 and comes with several new features and improvements over previous versions.

2.4.1 Here are some key features of Visual Studio 2022:

- **Enhanced IntelliSense:** Visual Studio 2022 comes with improved IntelliSense, which provides better code suggestions and completion based on context and your coding style.

- **Faster debugging:** The new Just My Code debugging feature in Visual Studio 2022 allows you to debug your code faster by skipping over third-party libraries and focusing on your own code.

- **Better collaboration:** Visual Studio 2022 provides better support for collaboration and code sharing, with features like Live Share and GitHub integration.

- **Improved code analysis:** Visual Studio 2022 includes improved code analysis tools, such as CodeLens and Code Analysis, which provide better insights into your code and help you identify potential issues.

- **Enhanced UI:** The user interface of Visual Studio 2022 has been updated with a new modern design, making it easier to use and more intuitive.

- **Cross-platform support:** Visual Studio 2022 provides cross-platform support for developing applications for Windows, macOS, and Linux.

- **Improved performance:** Visual Studio 2022 is faster and more responsive than previous versions, with improved startup times and reduced memory usage.

2.5 Unity ARFoundation package:

AR Foundation is a Unity package that allows developers to create augmented reality (AR) applications for mobile devices, it provides a set of tools and APIs for creating AR experiences that can be deployed on both Android and iOS devices.

2.5.1 ARFoundation Features:

- ARFoundation includes features for detecting and tracking real-world objects, such as planes and images.

- It also provides tools for placing virtual objects in the real world, using techniques such as raycasting and anchor points.

- ARFoundation can be used with a wide range of AR devices, including ARKit and ARCore.

2.5.2 How ARFoundation Works:

- ARFoundation uses a combination of computer vision and sensor data to detect and track real-world objects.

- It uses the device's camera to capture video and images, and analyzes them in real time to detect features such as planes or images.

- ARFoundation then uses this information to place virtual objects in the real world, using techniques such as raycasting to determine the position and orientation of the virtual object.

2.5.3 ARFoundation Components:

- ARFoundation includes several key components that are used to create AR experiences, including the AR Session, AR Session Origin, and AR Raycast Manager.

- The AR Session is responsible for managing the AR experience, while the AR Session Origin provides a reference point for virtual objects in the real world.

- The AR Raycast Manager is used to detect and track real-world objects, and provides information about the position and orientation of those objects.

2.5.4 ARFoundation Workflow:

- To create an AR experience using ARFoundation, you typically start by creating an AR Session and AR Session Origin object in your scene.

- You then use the AR Raycast Manager to detect real-world objects and place virtual objects in the scene.

- You can also use other ARFoundation components to add features such as image detection or plane tracking to your AR experience.

2.6 Wit.ai:

Wit.ai is a natural language processing (NLP) platform that allows developers to create chatbots and voice assistants It provides tools and APIs for processing user input and generating responses, including text-to-speech (TTS) functionality.

2.6.1 Text To Speech in Wit.ai:

- Wit.ai provides a TTS API that allows developers to convert text input into spoken output.

- The TTS API can be used to generate audio responses for chatbots or voice assistants, making them more interactive and engaging for users.

2.6.2 How TTS Works in Wit.ai:

- To use the TTS API in Wit.ai, you first need to create an account and set up a project.
- You can then use the API to send a text input to the Wit.ai server, which will convert it to speech using a speech synthesis engine.
- The API returns an audio file that can be played back to the user as a spoken response.

2.6.3 TTS Configuration in Wit.ai:

- Wit.ai allows developers to configure various settings for the TTS API, such as the voice and language used for speech synthesis.

- Developers can also customize the pronunciation of specific words or phrases, or add custom voice recordings for specific responses.

2.6.4 Benefits of TTS in Wit.ai:

- TTS can make chatbots and voice assistants more engaging and interactive for users, by providing a more natural and human-like response.

- It can also be used to make the user experience more accessible for people with visual impairments, who may have difficulty reading text on a screen.

2.7 Speechly:

Speechly is a speech recognition platform that allows developers to create natural language voice interfaces for their applications, it provides a set of tools and APIs for processing user input, including a Unity package for creating voice-enabled applications in Unity.

2.7.1 Speechly Features:

- Speechly includes features for real-time speech recognition and natural language understanding, allowing developers to create voice interfaces that can understand complex commands and queries.

- It also provides tools for managing and processing user data, including built-in support for user authentication and data encryption.

2.7.2 Speechly Unity Package:

- The Speechly Unity package provides a set of tools and APIs for creating voice-enabled applications in Unity.

- It includes support for speech recognition and natural language understanding, as well as integration with other Unity features such as UI elements and animations.

2.7.3 How Speechly Works:

- To use Speechly in Unity, you first need to create a Speechly project and obtain an API key.

- You can then use the Speechly Unity package to add voice-enabled functionality to your application, by configuring the Speechly client and setting up voice commands and queries.

- When the user speaks a command or query, the Speechly client sends the audio input to the Speechly server for processing, and returns a response that can be used to trigger events or update the application's state.

2.7.4 Benefits of Speechly:

- Speechly can make applications more intuitive and easier to use, by allowing users to interact with them using natural language.

- It can also be used to create more accessible user experiences, by providing an alternative input method for users with disabilities or limited mobility.

2.8 Blender:

Blender is a free and open-source 3D creation software that allows developers to create models, animations, and visual effects, it provides a wide range of tools and features for 3D modeling, including support for sculpting, rigging, and texturing.

2.8.1 Blender Features:

- Blender includes features for 3D modeling, animation, and visual effects, making it a powerful tool for creating 3D content.

- It also provides tools for rendering and compositing, allowing developers to create high-quality images and animations.

2.8.2 Blender and Unity:

- Blender can be used with Unity to create 3D models and assets that can be imported into Unity projects.

- By creating 3D models in Blender and importing them into Unity, developers can create more complex and detailed environments and characters for their games and applications.

2.8.3 Exporting from Blender to Unity:

- To export a 3D model from Blender to Unity, developers must first save the model in a supported file format, such as .fbx or .obj.

- They can then import the model into their Unity project, and use Unity's tools and features to set up animations, physics, and other properties.

2.8.4 Benefits of Using Blender with Unity:

- Using Blender with Unity allows developers to create more detailed and complex 3D models and environments for their projects.

- It also provides more flexibility and control over the modeling and animation process, allowing developers to create custom rigs and animations that are tailored to their specific use case.

2.9 Text Mesh Pro:

Text Mesh Pro is a Unity plugin that allows developers to create high-quality, customizable text in their projects, it provides a wide range of tools and features for text rendering, including support for advanced typography, dynamic text sizing, and text effects.

2.9.1 Text Mesh Pro Features:

- Text Mesh Pro includes features for creating high-quality, customizable text in Unity projects.

- It provides tools for adjusting text spacing, kerning, and line spacing, as well as support for dynamic text sizing and text effects such as shadows and outlines.

2.9.2 Using Text Mesh Pro in Unity:

- To use Text Mesh Pro in Unity, developers must first install the plugin from the Unity Asset Store.

- They can then create and customize text objects using the Text Mesh Pro component, which provides a wide range of options and settings for text rendering.

2.9.3 Benefits of Using Text Mesh Pro:

- Text Mesh Pro provides a powerful set of tools and features for creating high-quality, customizable text in Unity projects.

- It can be used to create dynamic, responsive text that adapts to changes in screen size and aspect ratio, making it ideal for use in mobile and web applications.

- Text Mesh Pro can also be used to create stylized text effects and typography, allowing developers to create unique and visually appealing user interfaces.

2.9.4 Text Mesh Pro and Unity UI:

- Text Mesh Pro can be used alongside Unity's built-in UI system to create custom text elements and user interfaces.

- By using Text Mesh Pro alongside Unity UI, developers can create high-quality, responsive text that integrates seamlessly with other UI elements and graphics.

2.10 Photoshop:

Photoshop is a popular image editing software developed by Adobe Systems, it provides a wide range of tools and features for editing and manipulating images, including support for layers, filters, and effects.

2.10.1 Photoshop Features:

- Photoshop includes features for image editing, manipulation, and creation, making it a powerful tool for creating graphics and artwork.

- It provides tools for adjusting colors, contrast, and brightness, as well as support for layers and masks for more complex image manipulation.

2.10.2 Using Photoshop with Unity:

- Photoshop can be used with Unity to create high-quality graphics and assets for Unity projects.

- Developers can create and manipulate images in Photoshop, and then import them into Unity for use in textures, sprites, and other graphics.

2.10.3 Benefits of Using Photoshop with Unity:

- Using Photoshop with Unity allows developers to create high-quality graphics and assets for their projects.

- It provides more flexibility and control over the image creation and manipulation process, allowing developers to create custom textures and graphics that are tailored to their specific use case.

- Photoshop can also be used to create stylized graphics and visual effects, adding a unique and visually appealing element to Unity projects.

2.10.3 Exporting from Photoshop to Unity:

- To export an image or graphic from Photoshop to Unity, developers must first save the image in a compatible file format, such as *.png or *.jpg .

- They can then import the image into their Unity project, and use Unity's tools and features to apply the image as a texture, sprite, or other graphic element.

Chapter3: "System Analysis"

In this chapter we provide a detailed knowledge about our system.

3.1 Determining Requirements:

The first step in system analysis is determining the requirements. In this section, we will discuss the functional and non-functional requirements for our system.

3.1.1 Functional Requirements:

Functional requirements are the features and functions that the system must perform to meet user needs. Here are the functional requirements for our system:

- The system should provide an augmented reality environment for the user.
- The system should have different levels for the user to progress through.
- The system should contain a new level, which appears after the user successfully completes a certain number of levels. This level as a reward and provides motivation to the user.
- The system should provide audio and visual interaction with the user.
- The system should provide touch interaction with the user when applicable.
- The system should have a database to store user information and progress.
- The system must work on android phones to make the use of system easy to user.
- The 3d models that displayed by AR must be colorful, funny and has an animation to make user have enjoyable time while using the system.

- The system should provide a track of music that will be run with a low volume at the scene to limit the silence and assist user in to concentrate at the level.
- The system should detect user voice to check from user spelling.
- The system should have text to speech feature to make the interaction between the system and the user mush easier.

3.1.2 Non-Functional Requirements:

Non-functional requirements are the system's qualities, such as performance, usability, and security. Here are the non-functional requirements for our system:

- The system should be easy to use and navigate for the user.
- The system should have fast response times to user input.
- The system should be compatible with multiple android devices.
- The system should have a high level of accuracy in tracking the user's spelling.
- The system should have a low rate of errors and bugs.
- The system should respect the user experience, and provide an environment that is not difficult to understand.

3.2 System Architecture

Once the system requirements have been identified, we will proceed to illustrate the key components of the system, their interdependencies and relationships, as well as their respective interactions, The system architecture of our augmented reality spelling game consists of the following components:

3.2.1 system architecture components:

- User Interface: The user interface will be the primary means of interaction between the user and the system. It will provide

graphical, audio, and touch-based interfaces for the user to interact with the system.

- Augmented Reality Engine: The augmented reality engine will enable the system to overlay 3D models and animations on the real world. It will use the camera of the user's device to detect the real-world environment, and then overlay virtual objects on top of it.
- Spelling Game Engine: The spelling game engine will be responsible for generating the spelling challenges for the user to complete. It will also track the user's progress, and adjust the difficulty of the challenges based on the user's performance.
- Database: The database will store all user information, including progress, scores, and user preferences. It will also store the 3D models and animations that will be displayed in the augmented reality environment.
- Text-to-Speech Engine: The text-to-speech engine will be responsible for converting text to speech. It will be used to provide audio feedback to the user during the game.
- Speech Recognition Engine: The speech recognition engine will be responsible for detecting the user's speech and converting it to text. It will be used to check the user's spelling during the game.
- Audio Track: The audio track will provide background music during the game. It will be designed to help the user concentrate while playing the game.

3.2.2 The components of the system will interact with each other as follows:

• The user interface will interact with the augmented reality engine to provide an immersive experience for the user.

• The spelling game engine will interact with the database to retrieve user information and generate spelling challenges.

• The text-to-speech engine and speech recognition engine will work together to provide audio feedback and check the user's spelling.

• The audio track will run in the background to provide an engaging and enjoyable experience for the user.

3.4 Development Methodology

After we knew the basic structure of the system. We are going to view all of its functions, the relation between them, and the sequence of their executions in the following subsections.

3.4.1 Context Diagram:

The context diagram is a high-level diagram that shows the system under consideration and its interactions with external entities. In the context of our augmented reality spelling game system, the context diagram would look like this:

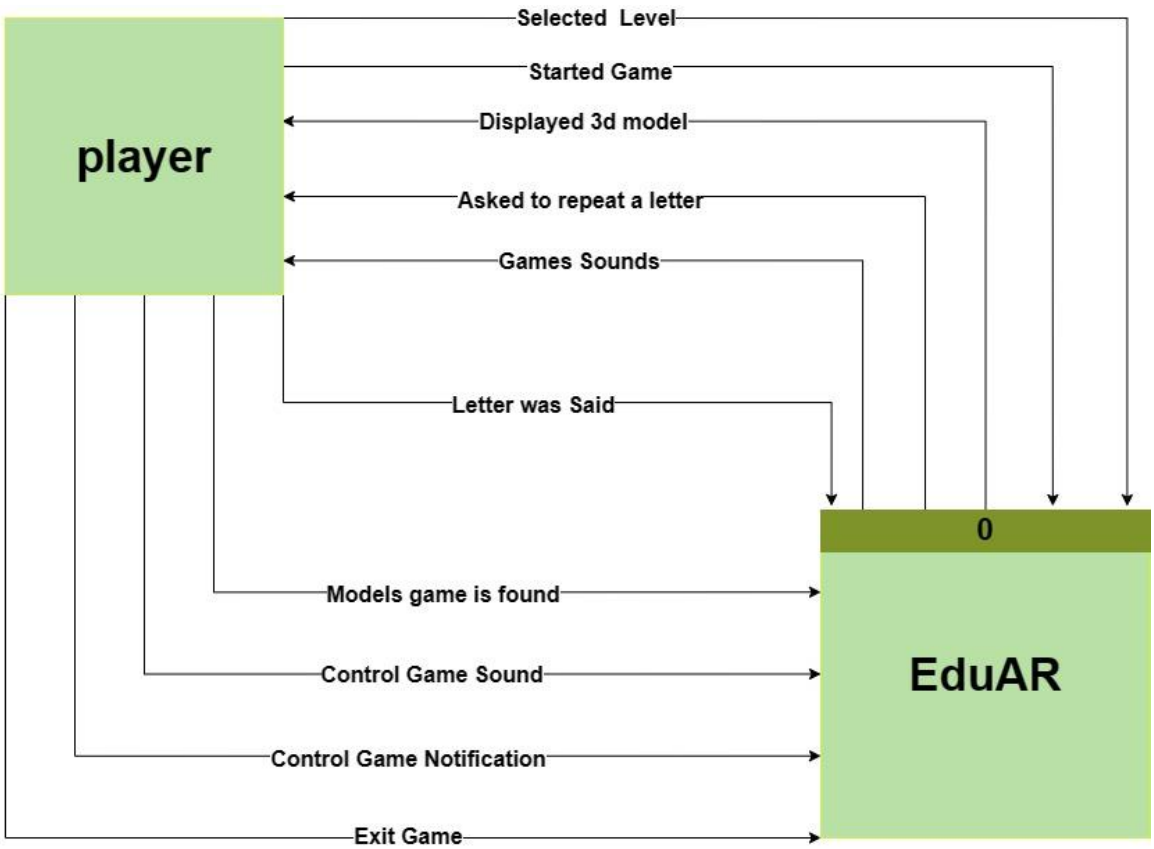


Figure 11 –EduAR Context Diagram

This diagram illustrates the main and essential components of the system in a concise manner. There are two main parties that make up the system, and the interaction occurs between them. The first is the system itself, and the other is the player or user who will use the system.

Each arrow in the diagram represents an event or interaction that occurs between the system and the user, going back and forth.

And here is an extended context diagram in Figure 7 that illustrate the system interaction and system components.

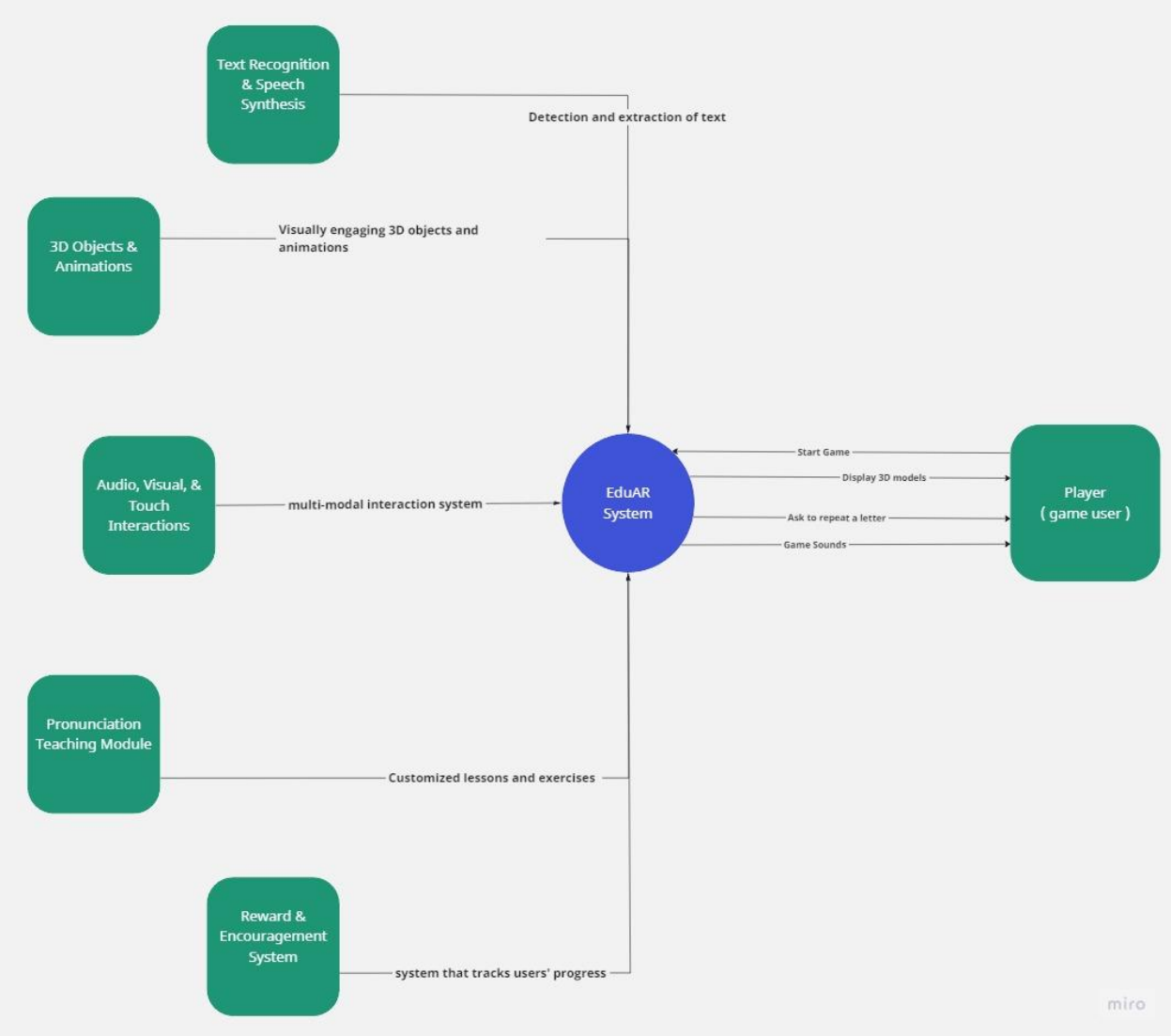


Figure 12 – Extended Context Diagram

System main components:

- **Augmented Reality Engine:** This component creates the augmented reality (AR) environment, where virtual objects and information are overlaid onto the real world, as seen through the device's camera. The output of this engine includes the real-time rendering of AR scenes and the tracking and positioning of virtual objects in the user's environment.

- **Text Recognition & Speech Synthesis:** The text recognition module scans and identifies text in the real world (e.g., written text on a page or a sign) and converts it into a digital format. The speech synthesis module then takes this text and generates an audio output of the correct pronunciation. The combined output of these modules includes the recognized text and the synthesized speech representing the correct pronunciation.

- **3D Objects & Animations**: This component provides the three-dimensional models and animations used within the app to create an engaging and immersive learning experience. The output includes the creation and rendering of 3D objects and animations that visually represent letters, words, and relevant educational content to facilitate learning and user interaction.

- **Pronunciation Teaching Module:** This module is responsible for delivering personalized lessons and exercises that target the user's pronunciation skills. The output of this component includes tailored learning content, such as phonetic breakdowns, auditory examples, and practice exercises, all designed to help users improve their pronunciation.

- **Reward & Encouragement System:** This component aims to motivate and engage users in the learning process by providing positive reinforcement and rewards for their achievements. The output of this system includes a variety of incentives, such as virtual badges, progress tracking, and personalized feedback, that help maintain user interest and commitment to learning.

- **Audio, Visual, & Touch Interactions:** This component manages the various input methods and interactions users have with the app, ensuring a seamless and intuitive experience. The output includes

the processing and interpretation of touch gestures, voice commands, and visual cues, allowing users to effectively engage with the app and its content. By working together, these components create an integrated, immersive, and engaging learning environment that supports users in developing their pronunciation skills and enjoying the learning process.

3.4.2 Use Case Diagram:

In this stage, we will display our system use case so what is the use case? A use case is a description of how a user interacts with a system to achieve a specific goal. It outlines the steps a user takes, the system's responses, and the result. Use cases are valuable tools in software development because they help identify the requirements of the system and ensure that it meets the needs of its users, we will specify the expected behavior (what) of the system, not the exact method of making it happen (how). This helps us to design the system from end user's perspective.

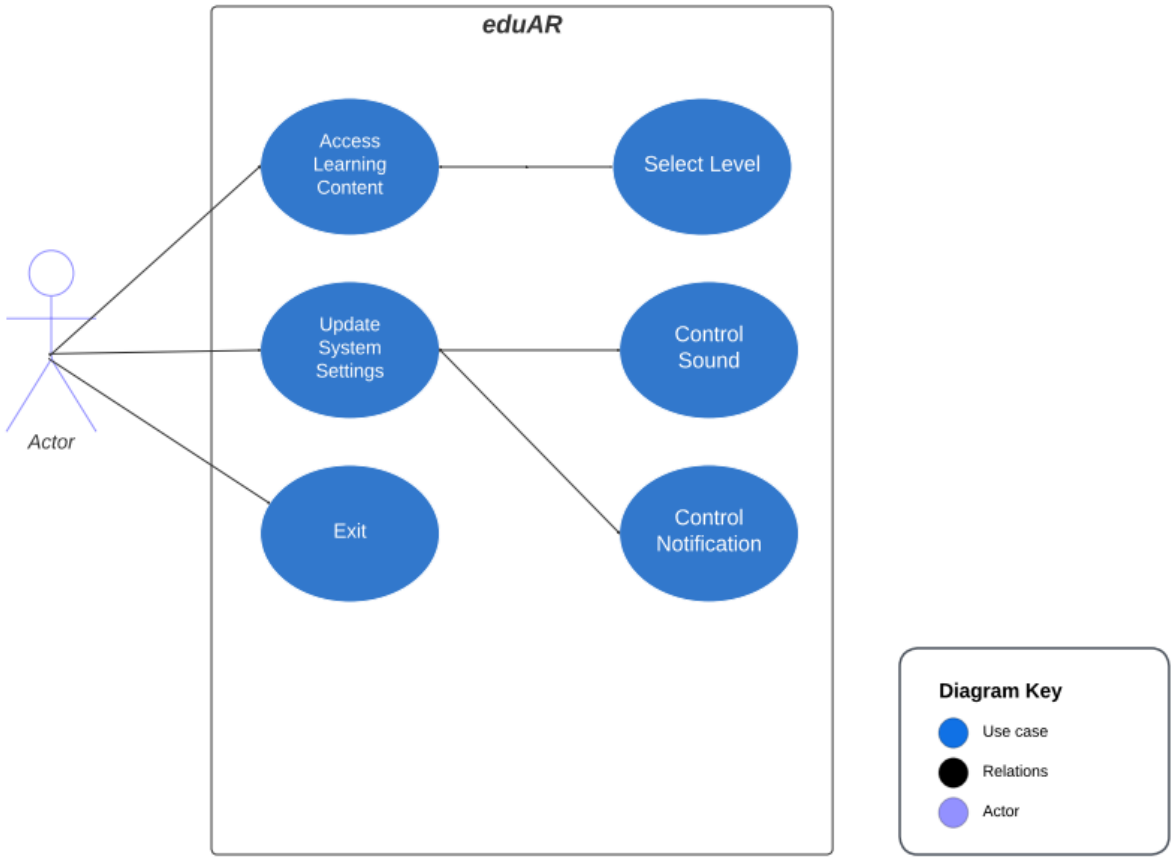


Figure 13 –EduAR Use case

The primary actor in this use case is a user who is accessing the learning content and interacting with the system. The use case can be broken down into the following components:

- **Accessing learning content:** The user logs into the EduAR system and accesses the learning content available in the system.
- **Selecting a level:** Once the user has accessed the learning content, they select a level to begin their learning journey. This level could be based on a variety of factors such as the user's previous progress that determine which English letter the user will be learn and what the 3d models that will be displayed to the user.
- **Updating system settings:** the user can update the system settings to customize their learning experience.

- **Controlling sound:** The user has the ability to control the music sound within the EduAR system. This could involve muting or unmuting the sound all together.
- **Controlling notifications:** The user can control the notifications they receive from the EduAR system. This could involve turning notifications on or off.
- **Exiting the system:** Finally, once the user has completed their learning session, they can exit the EduAR system.

3.4.3 Sequence Diagram:

Using the previous use case, we can implement a level sequence diagram to further illustrate the flow of events in the system. A sequence diagram is a type of interaction diagram that shows the interactions between objects in a system over time. It depicts the messages exchanged between objects, including the order in which they occur and the parameters involved.

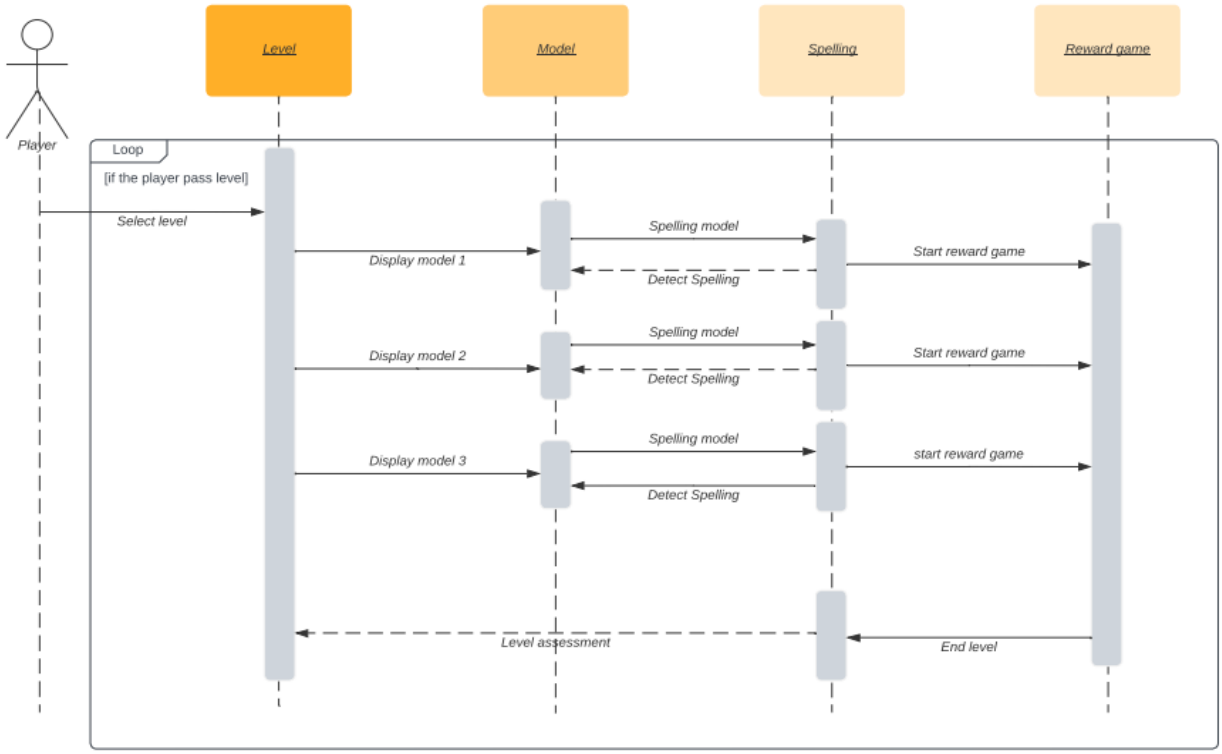


Figure 14 – EduAR Level sequence diagram

Chapter4: “System design & Implementation”

In this chapter, we will learn about the design and construction of the system. We will present the steps and stages that were taken during the application's development, and we will provide an overview of the implementation stages.

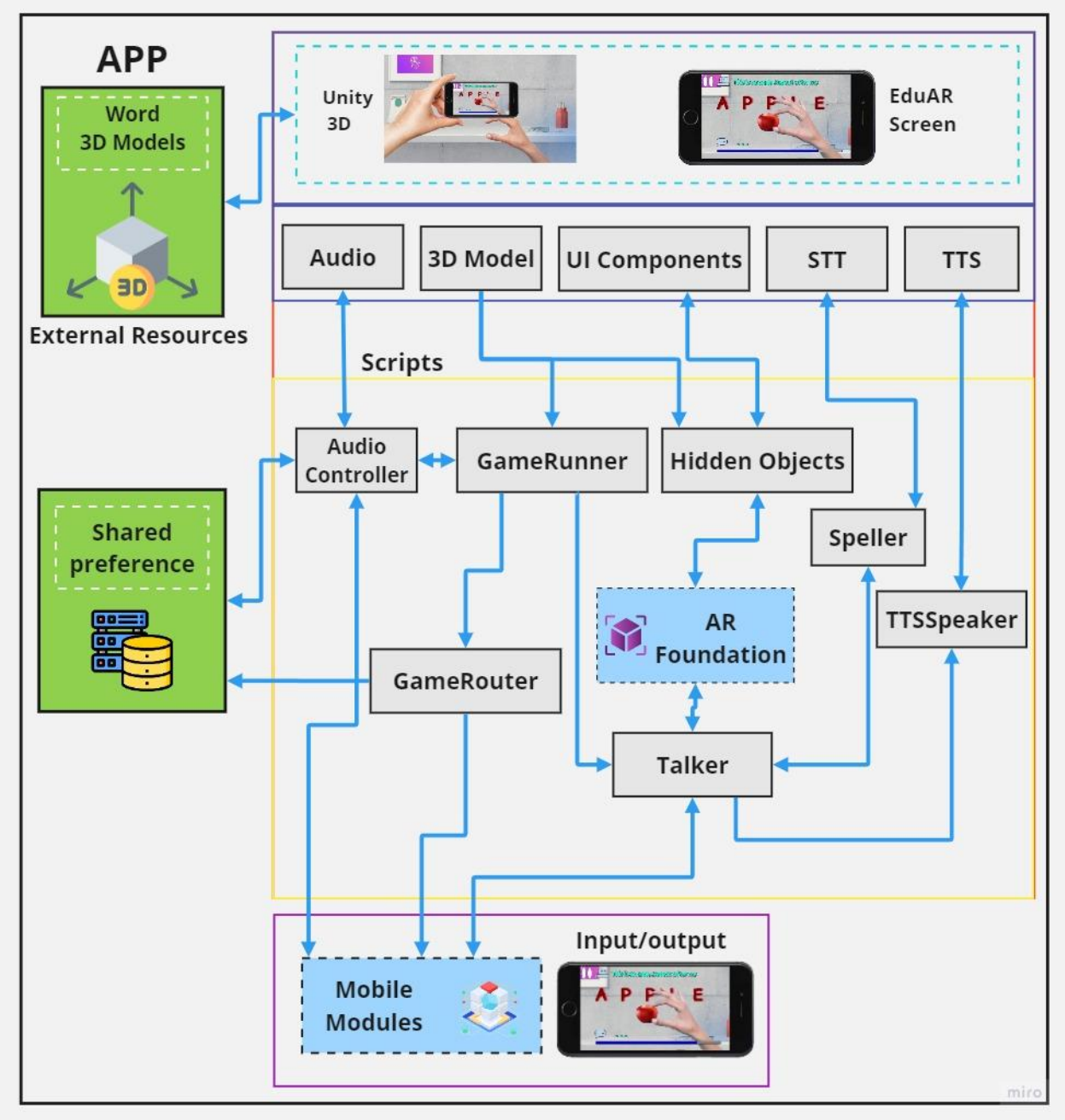


Figure 16 –illustration of the system component

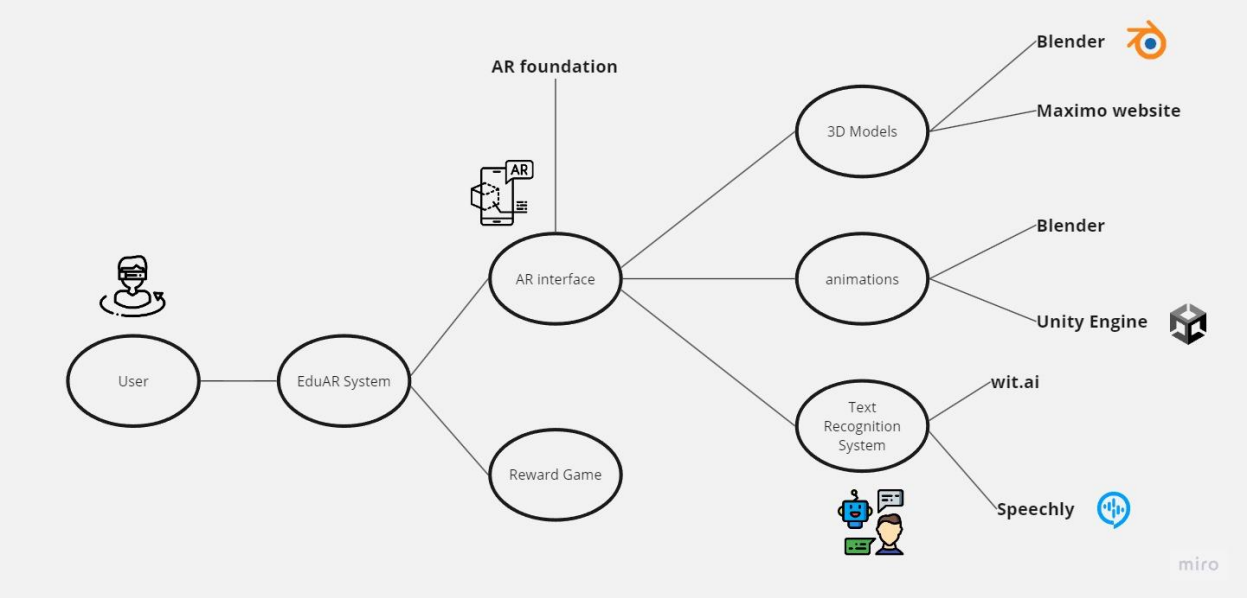


Figure 17 –illustration of the system component

In the previous diagram, we presented in detail all the main components of the system visually, and we will explain them further when we discuss the process of implementing the program. For now, we will learn about the system design and its different stages in detail, and we will clarify the method and the way in which the project was built based on it.

4.1 System Design:

In this section, we will discuss the design of the system, including the software architecture, user interface design, and database design.

4.1.1 Software Architecture:

The software architecture of the EduAR system is based on a layered architecture model. The layered architecture model is a design pattern that separates functionality into distinct layers. Each layer has a specific responsibility and interacts only with the layers above and below it.

The layers of the EduAR system are as follows:

- **Presentation Layer:** This layer is responsible for managing the user interface of the system. It interacts with the augmented reality engine and the spelling game engine to display the 3D models and animations and generate spelling challenges.
- **Application Layer:** This layer is responsible for managing the business logic of the system. It interacts with the database to retrieve user information and progress, and it interfaces with the pronunciation-teaching module to provide personalized lessons and exercises.
- **Data Access Layer:** This layer is responsible for managing the storage and retrieval of data from the database. It interacts with the application layer to provide access to user information, progress, and preferences.
- **Infrastructure Layer:** This layer is responsible for managing the hardware and software infrastructure that supports the system. It interacts with the text-to-speech engine and speech recognition engine to provide audio feedback and check user spelling, and it manages the audio track to provide background music during the game.

4.1.2 Explanation of how the user interacts with the system:

Before explaining the methodology and design followed to implement this system, we will first present a diagram that illustrates how the user interacts with the system interface and how the system interacts with the user in a brief manner. We will then explain this diagram in detail below.

In this diagram, the user enters the main system interface and clicks on the start button to move to the page where they can choose the level they want to play. When the user selects a level, the system displays a 3D model of a specific word for the selected letter. Using text-to-speech

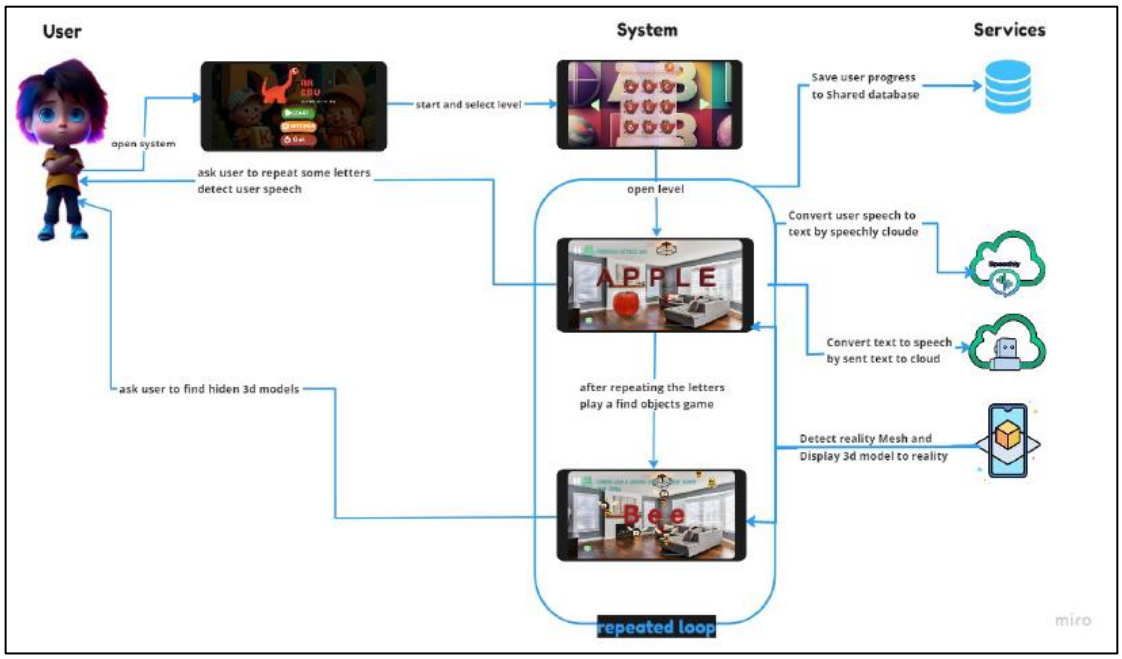


Figure 18-diagram to explain interaction between system components

technology, the system speaks to the user and instructs them to spell some of the letters that make up that word. This is where the user interacts with the system by spelling the letters, which the system recognizes using speech-to-text technology. After correctly spelling the word, the system displays another game for the user, which involves hiding a certain number of 3D objects related to that word and asking the user to use their mobile camera to search for those objects. When the user finds all the objects and interacts with them by clicking, the system calculates their score and moves them to complete the stage if there are more words. If the user completes all the words in the stage, the system records that the user has finished the stage and takes them back to the level selection page.

Each level corresponds to a different set of 3D models and animations, as well as a different set of spelling challenges, is designed to be visually appealing and easy to navigate. The user is presented with a grid of level icons, each representing a different level. Each level icon displays a preview of the 3D models and animations associated with that level, when the user taps on a level icon, they are taken to the level screen where they can begin playing the game. If the user has already completed a level, they can revisit that level to improve their spelling or move on to the next level

4.1.3 User Interface Design:

The user interface of the EduAR system is designed to be intuitive and easy to use. The interface is based on a minimalist design philosophy, with a focus on the essential elements needed to support learning.

The user interface consists of the following components:

- **Home screen:** The home screen provides access to the learning content available in the system, this is a wireframe that illustrates how the home screen design will be like.

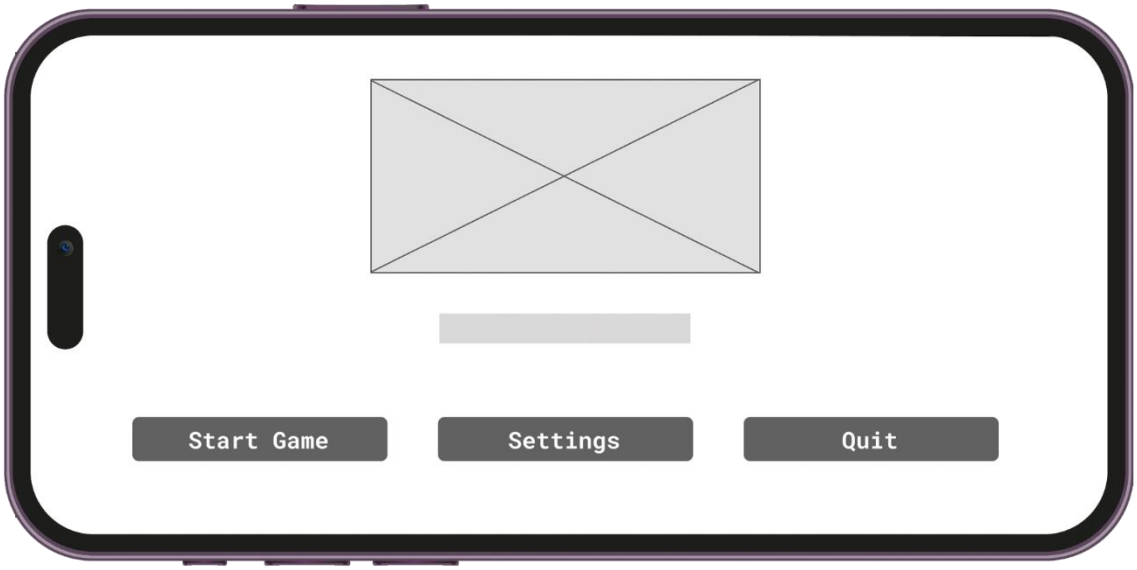


Figure 19 – home screen wireframe

- **Settings screen:** The settings screen allows the user to customize their learning experience by adjusting system settings, such as the sound and notifications.

Figure 20 – settings screen wire frame

Select level screen: is the interface where users can choose the level they want to play.

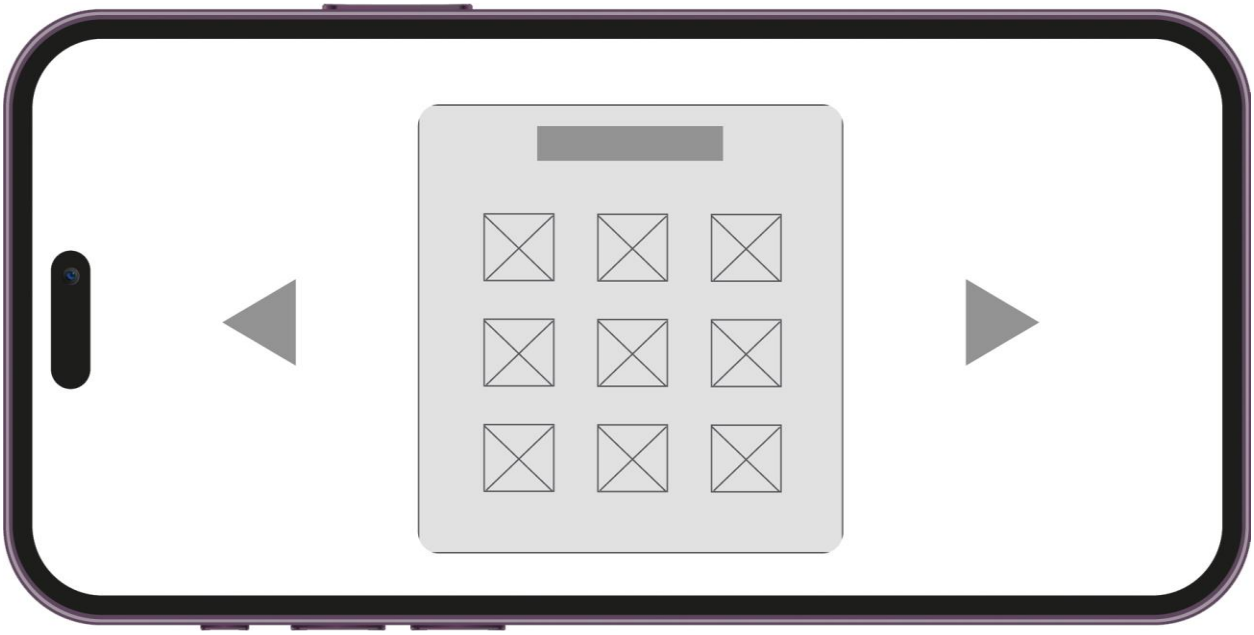


Figure 21 – select level screen wire frame

Level screen (Game Screen): The level screen displays the 3D models and animations associated with the level, as well as the spelling challenges

the user must complete, the gameplay level screen is where the user plays the spelling game associated with the selected level.

Figure 22 – level screen wireframe

Paused Screen: in level screen when we press the paused button the paused screen appears in which we can restart the level, go to main menu or close.

Figure 23 –level scree paused menu wireframe

Check Connection Screen: screen will appear stating that the internet connection has been lost if there is an error in the internet connection during the game. A message will be displayed to the player asking them to check their connection and try playing again.

Figure 24 check connection wire frame

4.1.3 Database Design:

The database design of the EduAR system is based on a non-relational database model. The database is designed to store user progress and settings, so there is no need for using the relational database in our case the best choose is to use non-relational because of best performance and quick response.

Shared preference is an Android-specific feature that provides a way to store small amounts of data in key-value pairs. It is often used for storing

user preferences, such as settings for sound and notification, as well as other small amounts of data that need to be persisted across sessions.

Shared preferences data is stored in an XML file in the app's private storage, which means that it is not accessible to other apps or users on the device. This provides a level of security and privacy for the user's data.

Shared preferences can be accessed and modified using the Shared Preferences class in Android. This class provides methods for reading and writing key-value pairs, as well as methods for checking if a specific key exists and clearing the entire shared preferences file.

4.3 Implementation:

In this section, we will discuss the implementation of the EduAR system. We will provide an overview of the development process and the technologies used to create the system; in this section of the chapter, we will provide the.

As we mentioned in chapter 2 at point 2.2.2 of the chapter, we use the scripting system in game development so that you can find one or more scripts that used in several screens, so we will display the script code as an image and if we use it again in another screen, we will only mention the code figure number.

In this section of the chapter, we will review with you the format of the pages after the design phase is completed. We will also go over the code used to implement each page, and display the hierarchical tree from the

Unity program, which provides us with the ability to clearly identify the components of the scene.

4.3.1 Implementation of Main screen:

4.3.1.1 Screen implemented Design:

Main screen, as we mentioned before, is the page that welcomes the user inside the system and consists of 4 main components: the Start button, which directs the user to the level selection page, the Settings button, which directs the user to the system settings control page, the Close button, which simply closes the system entirely, and finally, the system logo displayed in the center of the homepage.

Figure 25- Main Screen Implemented Design

4.3.1.2 Implementation and code:

1 - Game Router Script:

This script is one of the most important scripts in the system, as it has been used in many scenes and provides us with many useful functions such as.

GoToCustomLevel(): that function make us navigate the system to a custom level by passing a level name to it

RestartCurrentLevel(): this function redirecting the user to the current scene

GotoSettingsPage(): this function direct the user to settings page

GotoMainPage(): this function direct the user to home page

GotoSelectLevel() : this function used in select level page to go the custom selected level

SaveCurrentLevelAndBackToLevelPage():this function saves the current level progress with level number and go to select level page

All this functions is our script components and we use some of them in the main screen scene, and here is our code.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameRouter : MonoBehaviour
{
    /// Starts the application. This is called by Unity when the application is started but not on the main
    private void Start()
    {
        Application.runInBackground = true;

    }

    /// Restarts the current level. This is useful when you want to change the level to a different one
    public void RestartCurrentLevel()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    /// Loads the settings page and sets time scale to 1f. This is a hack to work around the bug in Game
    public void GoSettingsPage()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(1);
    }
```

Figure 26 –Game Router Script

```
    /// Saves the current level and back to the level page. This is used to prevent accidental changes in level
settings
    ///
    /// @param levelNumber - The current level number
    [SerializeField]
    public void SaveCurrentLevelAndBackToLevelPage(int levelNumber)
    {
        int oldLevelNumber = PlayerPrefs.GetInt("c_level");
        /// Set the level number to the new level number.
        if (oldLevelNumber == levelNumber)
        {
            PlayerPrefs.SetInt("c_level", levelNumber + 1);
            PlayerPrefs.Save();
        }
        Time.timeScale = 1f;
        SceneManager.LoadScene(2);
    }

    /// Go to a custom level. This is useful for loading scenes that don't have a level name
    ///
    /// @param levelName - The name of the
    public void GoToCustomLevel(string levelName)
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(levelName);
    }
    /// Loads the main page and sets time scale to 1. This is called when the user clicks the main
    public void GoToMainPage()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }
    /// Loads the level and sets timeScale to 1. Used for testing purposes to see if we need to go to a level
    public void GoToSelectLevel()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(2);
    }
}
```

Figure 27 – Game Router Script

2-GameQuitScript: this is a small script used to quit the game after user click quit button

```
using UnityEngine;

public class GameQuit : MonoBehaviour
{
    public void CloseTheGame()
    {
        Debug.Log("Game is closing");
        Application.Quit();
    }
}
```

Figure 28 – Game Quit script

3 – Button_click_sound : is a small script used to release a sound after user clicked on the button that has attached to this script.

```
using UnityEngine;

public class button_click_sound : MonoBehaviour
{
    public AudioSource audioS;
    /// Play sound when mouse click on button Sound is played in main window and not in window's
    public void clickSound()
    {
        try
        {
            audioS.Play();
        }
        catch (System.Exception e)
        {
            Debug.Log(e);
        }
    }
}
```

Figure 29 – Button click sound

4 – AndroidNotificationsScript: this script is used in several scenes , it used to send a notification to the mobile after some amount of time that make a user don't forget to visit the game every day.

And here is the script components.

notification Interval: is an integer variable to set the interval time between the notifications.

Start() : is the initial function that run by unity in default , in which we initialized the android channel to send a notification through it to the user.

ScheduleNotification(): that function sets a schedule to send a notifications to the user after amount of time.

CancelNotifications(): that function cancel notifications and prevent the channel from sending it.
ToggleNotifications():that function is used in settings scene to open or close sending notifications to the user.

```
using System.Collections.Generic;
using Unity.Notifications.Android;
using UnityEngine;

public class AndroidNotifications : MonoBehaviour
{
    public int notificationInterval = 3;

    /// Starts the notification service. Registers the notification channel and schedules the notification to be
sent to the notification
    void Start()
    {
        var channel = new AndroidNotificationChannel()
        {
            Id = "default_channel",
            Name = "Default Channel",
            Importance = Importance.Default,
            Description = "Default notification channel"
        };
        AndroidNotificationCenter.RegisterNotificationChannel(channel);

        ScheduleNotification();
    }

    /// Schedules a notification to be sent every notificationInterval seconds. This is used to prevent flickering
when there is a game
    void ScheduleNotification()
    {
        /// This method is called when the user clicks on the notification button.
        if (PlayerPrefs.HasKey("notification") == false)
        {
            PlayerPrefs.SetInt("notification", 1);
            PlayerPrefs.Save();
        }
        bool playerNotificationSettings = PlayerPrefs.GetInt("notification", 1) == 1;

        /// Send a notification to the AndroidNotificationCenter.
        if (playerNotificationSettings)
        {
            var notification = new AndroidNotification();
            notification.Title = "We miss you!😟";
            notification.Text = "Come to complete your learning journey!❤️";
            notification.FireTime = System.DateTime.Now.AddSeconds(notificationInterval);

            AndroidNotificationCenter.SendNotification(notification, "default_channel");
        }
    }
```

Figure 30 – Android Notifications Script

```
    /// Cancels all notifications. This is called when the user clicks the Cancel button in the notification center
to cancel all
    void CancelNotifications()
    {
        AndroidNotificationCenter.CancelAllNotifications();
    }

    /// Toggles notifications on and off. Notifications will be sent to the player when they are added or removed
    public void ToggleNotifications()
    {
        bool playerNotificationSettings = PlayerPrefs.GetInt("notification", 1) == 1;


        /// Schedule or cancel notifications if player notification settings are set.
        if (playerNotificationSettings)
        {
            ScheduleNotification();
        }
        else
        {
            CancelNotifications();
        }
    }
}
```

Figure 31 –Android Notifications

4.3.2 Implementation of Settings screen:

4.3.2.1 Screen implemented Design:

The settings page is a simple page dedicated to controlling the game settings. It consists of three main components: a button to toggle notifications on and off, a button to toggle in-game music on and off, and a button to return to the home page, and here is the scene in the developing phase.

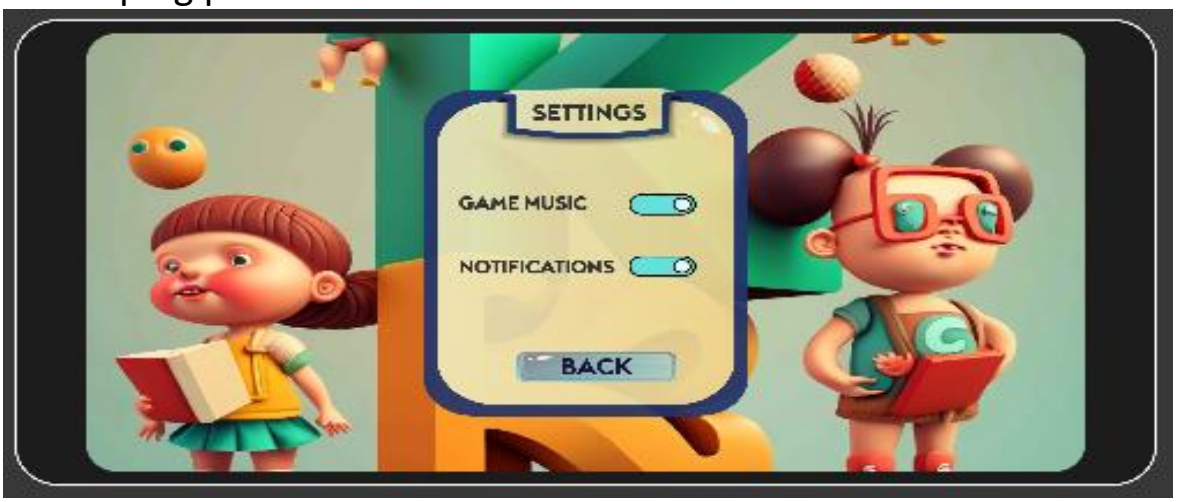


Figure 32 – settings page ui

4.3.2.2 Implementation and code:

1- GameRouterScript: this script we mentioned it before in the previous scene.

2- AndroidNotificatinsScript: this script we mentioned it before and here we use it to turn notifications on and off.

3- Toggle_ButtonScript: we use this script to create a custom beautiful toggle button and customize it, here is the code.

```
using UnityEngine;
public class toggel_button : MonoBehaviour
{
    [SerializeField]
    public GameObject onButton;
    public GameObject offButton;
    public AudioSource audioSource;
    private bool isClicked = true;
    /// Called when the button is clicked. Activates or deactivates the on / off buttons depending on whether the
button is clicked
    void Start()
    {
        /// Set the active state of the on and off buttons.
        if (isClicked)
        {
            onButton.SetActive(true);
            offButton.SetActive(false);
        }
        else
        {
            onButton.SetActive(false);
            offButton.SetActive(true);
        }
    }
    /// Clicks the on / off buttons with the value passed. This is useful for toggling the state of the mouse
    ///
    /// @param val - True to click on false to
    public void ClickWithValue(bool val)
    {
        /// Set the active state of the button
        if (val)
        {
            onButton.SetActive(false);
            offButton.SetActive(true);
            isClicked = val;
            Debug.Log("value from ctrl");
        }
        else
        {
            onButton.SetActive(true);
            offButton.SetActive(false);
            isClicked = val;
            Debug.Log("value from ctrl 2");
        }
    }
```

Figure 33 –toggle button script

```
    /// Toggles the on / off buttons and plays the audio source. This is called by the click handler
    public void ClickToggle()
    {
        Debug.Log(" Clicked 2");
        audioSource.Play();
        /// Set the active state of the on and off buttons.
        if (isClicked)
        {
            onButton.SetActive(false);
            offButton.SetActive(true);
            isClicked = false;
            Debug.Log(" Clicked 1");
        }
        else
        {
            onButton.SetActive(true);
            offButton.SetActive(false);
            isClicked = true;
            Debug.Log(" Clicked 2");
        }
    }
}
```

Figure 34 – toggle button script

4-LoadSettingsScript: this script is used to load a settings state from shared preference and set it values in UI, here we will talk about script component.

checkAndSetNotification(): this function check the notification value in shared preference and send it to be display in UI .

checkAndSetSound(): like the previous function we check the sound value in shared and change the toggling in UI.

Update(): update the UI if there is a new values.

Start() : Initialize the components on start of the scene.

```
using System;
using UnityEngine;

public class LoadSettings : MonoBehaviour
{

    public toggel_button toggel;
    public toggel_button toggle_notification;
    private bool isSoundOn;
    private bool isNotificationOn;

    /// Checks and sets sound and notification on / off. This is called when the game starts and it's safe to call
```

Figure 35 –Load Settings Script

```
    void Start()
    {
        try
        {
            checkAndSetSound();
            checkAndSetNotification();
            var tof = toggel.gameObject.GetComponent<toggel_button>();
            var tof_notification = toggle_notification.gameObject.GetComponent<toggel_button>();
            tof.ClickWithValue(isSoundOn);
            tof_notification.ClickWithValue(isNotificationOn);
        }
        catch (Exception e)
        {
            Debug.LogWarning(e.ToString());
        }
    }
    /// Updates the sounds and notifications. This is called every frame to ensure that we don't have stuck in an
infinite loop
    void Update()
    {
        checkAndSetSound();
        checkAndSetNotification();
    }
    /// Checks if notification is on and sets it if not. This is called every time PlayerPrefs. Save
    private void checkAndSetNotification()
    {
        /// Set the notification flag to 1.
        if (PlayerPrefs.HasKey("notification"))
        {
            isNotificationOn = PlayerPrefs.GetInt("notification") == 1 ? true : false;
        }
        else
        {
            PlayerPrefs.SetInt("notification", 1);
            PlayerPrefs.Save();
            isNotificationOn = true;
        }
    }

    /// Checks and sets sound on / off. This is called every time the game is loaded or unloaded
    private void checkAndSetSound()
    {
        var aud = this.GetComponent<AudioSource>();
        /// Set the sound on or off
        if (aud != null)
        {

            /// Set the sound on or off
            if (PlayerPrefs.HasKey("game_sound"))
            {
                isSoundOn = PlayerPrefs.GetInt("game_sound") == 1 ? true : false;

                /// Play or pause the audio if playing.
                if (isSoundOn && !aud.isPlaying)
                {
                    aud.Play();
                }
                /// Pause the audio if playing.
                else if (!isSoundOn && aud.isPlaying)
                {
                    aud.Pause();
                }
            }
            else
            {
                PlayerPrefs.SetInt("game_sound", 1);
                PlayerPrefs.Save();
                Debug.Log("set_sound");
            }
        }
    }
}
```

Figure 36 – load settings scripts

5-EditSettingsScript: this function edits the user settings such as notifications and game sounds.

```
using UnityEngine;

public class EditSettings : MonoBehaviour
{
    public AndroidNotifications androidNotifications;
    /// Turns off notifications and saves preferences to prevent it from disappearing. Notifications are disabled
when the player is in the middle of a
    public void TurnOffNotification()
    {
        Debug.Log("isva:" + 0);
        PlayerPrefs.SetInt("notification", 0);
        PlayerPrefs.Save();
        androidNotifications.ToggleNotifications();
    }
    /// Turns on notifications and saves preferences to prevent notifications from disappearing in future. This is
useful for debugging
    public void TurnOnNotification()
    {
        Debug.Log("isva:" + 1);
        PlayerPrefs.SetInt("notification", 1);
        PlayerPrefs.Save();
        androidNotifications.ToggleNotifications();
    }
    /// Turns off sounds for the game. This is called when the game is started but not when we want to play the
    public void TurnOffGameSounds()
    {
        Debug.Log("isva:" + 0);
        PlayerPrefs.SetInt("game_sound", 0);
        PlayerPrefs.Save();
    }
    /// Turns on sounds for the game. This is called when the player presses the button to turn on
    public void TurnOnGameSounds()
    {
        Debug.Log("isva:" + 1);
        PlayerPrefs.SetInt("game_sound", 1);
        PlayerPrefs.Save();
    }
}
```

Figure 37 –edit settings script

4.3.3 Implementation of Select Level screen:

4.3.3.1 Screen implemented Design:

This is a select level screen, in which we can select a level to play, this screen is contains 3 pages every page is contains a level button to navigate to the level, and here is a select level after development phase.

Figure 38 – select level screen

4.3.3.2 Implementation and code:

1- GameRouterScript(): this script we mentioned it before in the previous scene ,we use it here to go to the selected level.

2- button click sound script(): this script we mentioned it before in main screen .

3-LoadSettingsScript: we mentioned it before and we use it in this page to check the game music and turn it on and off.

4-ChangeLevelPage(): this script used to change the level button visibility and change level page.

```
using UnityEngine;

public class ChangeLevelPage : MonoBehaviour
{
    [SerializeField]
    public GameObject[] panels = new GameObject[3];
    public GameObject[] levels = new GameObject[28];
    public GameObject[] rewards = new GameObject[3];
    private int currentPage = 0;
    /// Called when player starts. This is where you change the values of level and rewards in PlayerPre
```

Figure 39 changeLevelPage script,variables

```
    void Start()
    {
        /// This method is called by the Player s main level.
        if (PlayerPrefs.HasKey("c_level"))
        {
            Debug.Log("c_isFound");
            int cl = PlayerPrefs.GetInt("c_level");
            Debug.Log(cl);
            /// This function is called by the LevelButtonScript. changeLevelValues method.
            for (int i = 0; i <= cl; i++)
            {
                var xi = levels[i].GetComponent<LevelButtonScript>();
                xi.changeLevelValues(true);
                /// Change the level values of the rewards
                if (i == 6)
                {
                    var re = rewards[0].GetComponent<LevelButtonScript>();
                    re.changeLevelValues(true);
                }
                Debug.Log(xi.name);
            }
        }
        else
        {
            Debug.Log("not_isFound");
            PlayerPrefs.SetInt("c_level", 0);
            PlayerPrefs.Save();
            var xi = levels[0].GetComponent<LevelButtonScript>();
            xi.changeLevelValues(true);
        }
    }
    /// Changes the page to the left. This is equivalent to pressing the left button on the page and then releasing
the
    public void leftChangePage()
    {
        changePage(-1);
    }
    /// Rieght einer Konfiguration ausgefuehrt wird. Diese Methode gesetzt
    public void rieghtChangePage()
    {
        changePage(1);
    }

    /// Changes the current page. This is called when the user clicks the page button. It will change the current
page and the panels that are shown on the page
    ///
    /// @param dir - The direction to change
    private void changePage(int dir)
    {
        try
        {
            currentPage += dir;
            currentPage = currentPage > 2 ? 0 : currentPage;
            currentPage = currentPage < 0 ? 2 : currentPage;
            /// Set the active state of all panels.
            for (int i = 0; i < panels.Length; i++)
            {
                /// Set the active state of the panel.
                if (i == currentPage)
                {
                    panels[i].SetActive(true);
                }
                else
                {
                    panels[i].SetActive(false);
                }
            }
        }
        finally { }
    }
}
```

Figure 40 change level page script

4.3.4 Implementation of Level screen (Game Screen):

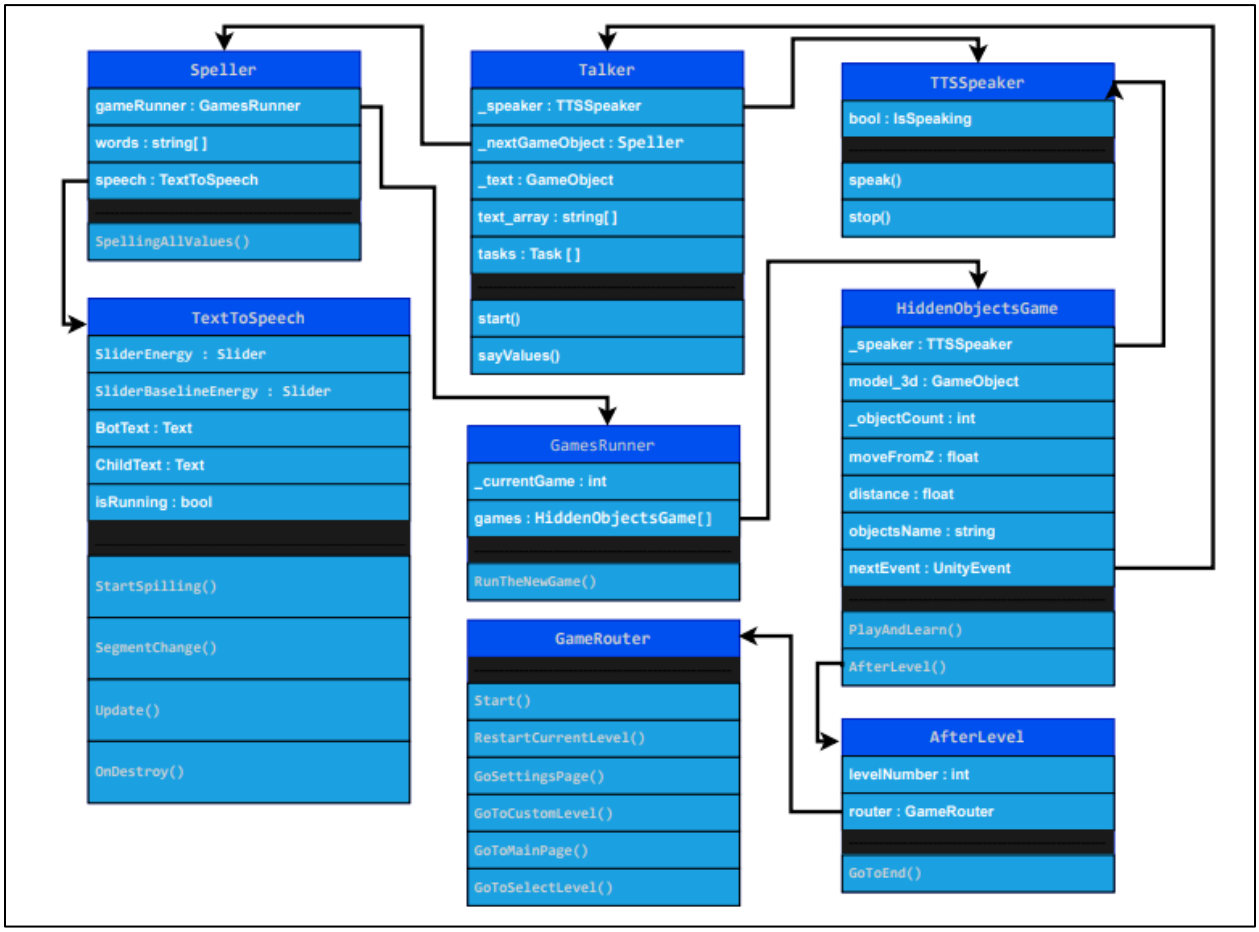


Figure 41 class diagram

4.3.4.1 Screen implemented Design:

This scene is the main scene in the system and includes the system's overall goal. The scene uses an augmented reality engine to display 3D models to the user and uses TextMeshPro to display the name of the 3D model that the user will spell out. The scene uses wit.ai to convert the written text to clear, human-like speech that the user can hear. Additionally, the scene uses speechly to listen to the words spoken by the user, compares them to the available words, and ensures that the user has pronounced them correctly. Once the user correctly spells out

the word, a game appears that randomly throws 3D models around the user in space and asks them to find them.

Figure 42- game level screen

4.3.4.2 Implementation and code:

Because the important of this scene there is a lot of scripts.

1-TalkerScript:is the first script works in the level and it welcomes the user with a welcome message and this message change every time the 3d model changed, and there is a some of the script components.

this script takes an array of texts and uses TTSSpeaker to continuously speak them to the user.

```
using System.Collections;
using System.Threading.Tasks;
using Meta.WitAi.TTS.Utilities;
using UnityEngine;
using UnityEngine.UI;
public class TalkerScript : MonoBehaviour
{
    [SerializeField] public TTSSpeaker _speaker;
    public GameObject _nextGameObject;
    public GameObject _text;
    public string[] text_array;
    private Task[] tasks = new Task[3];
    Coroutine xcoroutine;
    /// Start welcoming the game. This is called by Unity's StartCoroutine (...
    void Start()
    {
        xcoroutine = StartCoroutine(welcoming());
    }
    /// Coroutine to speak and read the text. This coroutine is used as a background task for the welcoming game.
    ///
    ///
    /// @return The coroutine to wait for the text to be spoken and read from the queue or null if the text is the
last
    IEnumerator welcoming()
    {
        /// Yields a coroutine to read the text from the queue.
        for (int i = 0; i < tasks.Length; i++)
        {
            // speak the text while text is in the queue
            /// Speak the text of the task.
            if (i < tasks.Length)
            {
                _text.GetComponent<Text>().text = text_array[i];
                _speaker.Speak(text_array[i]);
            }
            // wait until the text is spoken
            /// Yields the next sample from the speaker.
            while (_speaker.IsSpeaking || _speaker.IsLoading)
            {
                yield return null;
            }
            // wait for a bit to give the user time to read the text
            yield return new WaitForSeconds(.5f);
            // end the coroutine if the text is the last one
            /// This method is called when the next task is finished.
            if (i >= tasks.Length - 1)
            {
                Debug.Log("end welcoming");
                _nextGameObject.GetComponent<SpellingValues>().SpellingAllValues();
                StopCoroutine(xcoroutine);
                break;
            }
        }
    }
}
```

Figure 43- Talker script

2– SpillingValuesScript: it is a simple script that take an array of the current word letters and pass it to using text to speech script to check the user spilling.

This script work as a middle were to make the code mush safe to be edited later.

```
using UnityEngine;

public class SpellingValues : MonoBehaviour
{
    [SerializeField]
    public GamesRunner gamesRunner;
    public string[] words;
    // public UseTextToSpeech speech;
    public UsingTextSpeech speech;
    /// Spells all values in the text. This is called when the user presses the spell button
    public void SpellingAllValues()
    {
        speech.StartSpilling(words, gamesRunner);
    }
}
```

Figure 44 - SpellingValues

3 – UsingTextSpeechScript: this script takes a game runner and a list of letters and it ask the user to repeat the letters after it, and he moves from one letter to another letter after user repeat the letter correctly, after the list of letters ended the script using game runner to run the simi find game object game.

Here is the script.

```
using System.Collections;
using Meta.WitAi.TTS.Utilities;
using Speechly.SLUClient;
using UnityEngine;
using UnityEngine.UI;

public class UsingTextSpeech : MonoBehaviour
{
    public Slider SliderEnergy;
    /// Creates and initializes a base sliders wrapper. This is used to set the baseline
    public Slider SliderBaselineEnergy;
    /// Gets the bot text. This is used to display the bot's text
    public Text BotText;
    /// Gets the child text. This is used to display the child's text
    public Text ChildText;
    /// Called when the speaker is added to the game. This is where you can set the speakers
    public TTSSpeaker _speaker;
    /// Creates and returns a verygod clip. This is used to play the verygod clip
    public AudioClip verygodClip;
    private SpeechlyClient speechlyClient;
    private AudioSource audioSource;
    bool isRunning = false;
    public void StartSpilling(string[] words, GamesRunner gamesRunner) => StartCoroutine(_StartSpilling(words,
gamesRunner));
    public IEnumerator _StartSpilling(string[] words, GamesRunner gamesRunner)
    {
        audioSource = this.GetComponentInChildren<AudioSource>();
        var speechlyGameObject = MicToSpeechly.Instance;
        speechlyClient = MicToSpeechly.Instance.SpeechlyClient;
        speechlyGameObject.SetActive(false);
        for (int i = 0; i < words.Length; i++)
        {
            string currentWord = words[i].ToLower();
            BotText.text = currentWord;
            _speaker.Speak(currentWord);
            //wait for the speaker to finish speaking
            Debug.Log("Waiting for speaker to finish speaking");
            while (_speaker.IsSpeaking || _speaker.IsLoading)
            { yield return new WaitForSeconds(0.1f); }
            yield return new WaitForSeconds(.3f);
            //chenge [isRunning] to true to start listening for the segment
            //initialize the filtered text
            string filteredText = "";
            // check for the segment coming from the user
            #region SegmentChange
            //initialize the speechly client
            isRunning = true;
            speechlyGameObject.SetActive(true);
            //if(!speechlyClient.IsReady)yield return speechlyClient.Start();
            //start speechly segment
            // speechlyClient.OnTranscriptChange += (transcript) => TranscriptChangeSe(transcript,ref
filteredText);
```

Figure 45 -using text speech script

```
            // and the bot will say "very good"
            Debug.Log("Waiting for user to spill the word");
            while (!(filteredText == currentWord || filteredText.Contains(currentWord)))
            {
                yield return waitForSomeTime(0.1f);
            }
            speechlyClient.OnSegmentChange -= segmentionChange;
            speechlyClient.Update();
            Debug.Log("out of " + speechlyClient.IsActive);
            yield return waitForSomeTime(1f);
            //Debug.Log("out of while loop = " + x);
            isRunning = false;
            audioSource.PlayOneShot(verygodClip);
            Debug.LogWarning("Playing audio");
            //to make application wait for the audio to finish playing
            while (audioSource.isPlaying)
            {
                yield return waitForSomeTime(1f);
            }
            ChildText.text = "- - -";
            speechlyGameObject.SetActive(false);
        }
        Debug.Log("out of for loop ");
        ChildText.text = "- - -";
        //run hidin object script and show the counter
        isRunning = false;
        gamesRunner.RunTheNewGame();
        speechlyClient = null;
        yield return null;
    }
    IEnumerator waitForSomeTime(float timeout)
    {
        yield return new WaitForSeconds(timeout);
    }
    private void Update()
    {
        if (isRunning)
        {
            try
            {
                SliderBaselineEnergy.value = speechlyClient.Output.NoiseLevelDb;
                SliderEnergy.value = speechlyClient.Output.NoiseLevelDb + speechlyClient.Output.SignalDb;
            }
            finally { }
        }
    }
```

Figure 46 using text speech scirpt

```
    private void SegmentChangeSe(Segment segment, ref string filteredText, ref string currentWord, ref int i, ref
string[] words)
    {
        try
        {
            if (words.Length > 0 && i < words.Length && segment.isFinal)
            {
                filteredText = segment.ToString(
                (v) => ""
                , (a, c) => ""
                , "").Trim(' '
                ).ToLower();
                //check if the filtered text is equal to the current word
                // if it is not equal, then display the text as listening on the bot's text
                // to let user know that the bot is listening
                bool isEquals = filteredText == currentWord;
                bool isContains = filteredText.Contains(currentWord);
                if ((isEquals) || (isContains))
                {
                    // display the filtered text on the child's text
                    ChildText.text = currentWord;
                    // stop listening for the segment
                }
                else
                {
                    // if it is not equal, then display the text as listening on the bot's text
                    ChildText.text = "Lestening...";
                }
            }
            // this line is used to display the current word on the console
            // to check if the current word is equal to the filtered text
            Debug.Log("Current word: " + filteredText);
        }
        finally { }
    }
    private void OnDestroy()
    {
        if (speechlyClient != null)
        {
            speechlyClient.Stop();
            speechlyClient.StopStream();
            speechlyClient = null;
        }
        StopAllCoroutines();
    }
}
```

Figure 47 using text speech script

4- OnTouchGameObjectScript: this script is used to detect if the game object 3d was touched or not.

```
using UnityEngine;
using UnityEngine.UI;
public class OnTouchGameObject : MonoBehaviour
{
    private bool isTouched = false;
    public bool isAllowToBeTouched => isTouched;
    public Text counter;
    /// Allow touch events to be sent to the game object. This is useful for checking if you are touching a
GameObject
    public void TouchAllowing()
    {
        Debug.Log(this.gameObject.name + " touch allowed");
        isTouched = true;
    }
    /// Checks if the game object has touched and if so removes it from the game object list. This is called every
frame
    void Update()
    {
        /// This method will be called when the user clicks on the screen.
        if (isAllowToBeTouched)
        {
            /// This method will be called when the user presses a touch.
            if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began)
            {
                // Get the touch position
                Vector2 touchPosition = Input.GetTouch(0).position;
                Ray ray = Camera.main.ScreenPointToRay(touchPosition);
                RaycastHit hit;
                /// This method will be called when a Physics Raycast is called.
                if (Physics.Raycast(ray, out hit))
                {
                    Debug.Log("Touched " + hit.transform.gameObject.name);

                    // Destroy the game object that was touched
                    Destroy(hit.transform.gameObject);
                }
            }
        }
    }
}
```

Figure 48 – on touchgame object scripts

5 – EndleveScript: this script is the last script called in the game after all scripts run and he store the user progress and use Game Router to back to the select level screen.

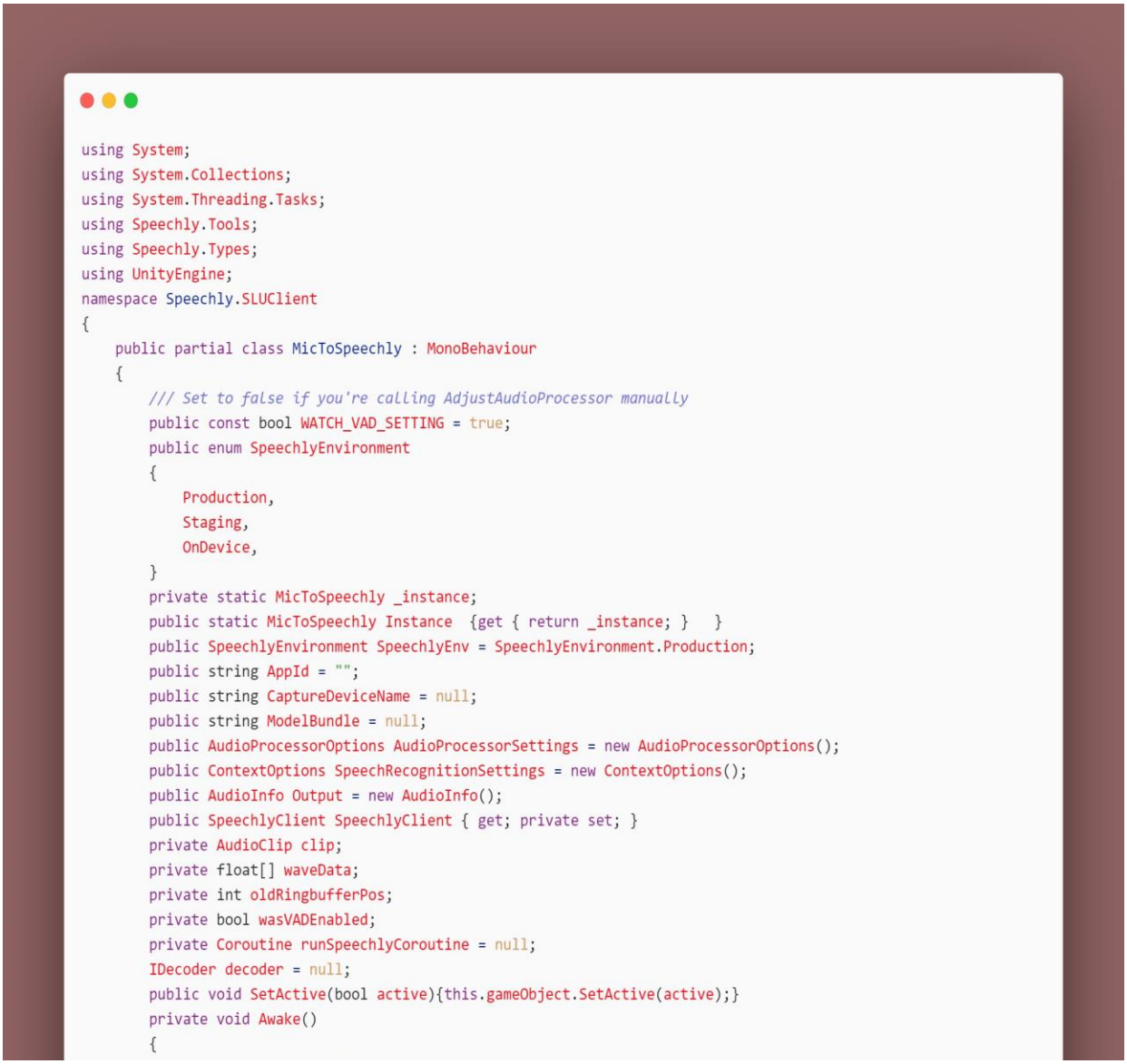
It takes the int level number, and TTSSpeaker to tell the user that he ended the level successfully and encourage him to learn more.

```
using System.Collections;
using System.Collections.Generic;
using Meta.WitAi.TTS.Utilities;
using UnityEngine;
using UnityEngine.UI;

public class EndLevelScript : MonoBehaviour
{
    public TTSSpeaker _speaker;
    public int levelNumber;
    /// Play the clip and wait for it to end. This is useful for things like sounding a game
    public void GoToEnd()
    {
        audioSource.PlayOneShot(clip);
        var xRoutine = StartCoroutine(WaitAndSpeak());
    }
    public GameRouter gameRouter;
    public AudioClip clip;
    public AudioSource audioSource;
    public Text text;
    /// Waits and spawns the speaker.
    /// This is a coroutine so we don't have to worry about looping in the coroutine.
    /// @return The coroutine that will wait and spawns the speaker.
    ///  This coroutine will return when the speak is done
    private IEnumerator WaitAndSpeak()
    {
        text.text = "Congratulations! You have completed the level. new level unlocked";
        _speaker.Speak("Congratulations! You have completed the level. new level unlocked");
        /// Yields the next sample from the speaker.
        while (_speaker.IsSpeaking || _speaker.IsLoading)
        {
            yield return null;
        }
        yield return new WaitForSeconds(2f);
        gameRouter.SaveCurrentLevelAndBackToLevelPage(levelNumber);
    }
}
```

Figure 49 End level script

5 – MicToSpeechlyScript: this script open the device microphone and listen to the user, it connect to the speechly cloud and use our API key to access to our speechly subscription, it sends the user voice as a sequence bytes stream and back with the voice detection from the cloud.



```
using System;
using System.Collections;
using System.Threading.Tasks;
using Speechly.Tools;
using Speechly.Types;
using UnityEngine;
namespace Speechly.SLUClient
{
    public partial class MicToSpeechly : MonoBehaviour
    {
        /// Set to false if you're calling AdjustAudioProcessor manually
        public const bool WATCH_VAD_SETTING = true;
        public enum SpeechlyEnvironment
        {
            Production,
            Staging,
            OnDevice,
        }
        private static MicToSpeechly _instance;
        public static MicToSpeechly Instance  {get { return _instance; }   }
        public SpeechlyEnvironment SpeechlyEnv = SpeechlyEnvironment.Production;
        public string AppId = "";
        public string CaptureDeviceName = null;
        public string ModelBundle = null;
        public AudioProcessorOptions AudioProcessorSettings = new AudioProcessorOptions();
        public ContextOptions SpeechRecognitionSettings = new ContextOptions();
        public AudioInfo Output = new AudioInfo();
        public SpeechlyClient SpeechlyClient { get; private set; }
        private AudioClip clip;
        private float[] waveData;
        private int oldRingbufferPos;
        private bool wasVADEnabled;
        private Coroutine runSpeechlyCoroutine = null;
        IDecoder decoder = null;
        public void SetActive(bool active){this.gameObject.SetActive(active);}
        private void Awake()
        {
```

Figure 50 - MicToSpeechlyScript

```
        private void Awake()
        {
            if (_instance != null && _instance != this)
            {
                Destroy(this.gameObject);
                return;
            }
            SpeechlyClient = new SpeechlyClient(
              manualUpdate: true,
              output: this.Output,
            );
            _instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        private IEnumerator RunSpeechly()
        {
            if (SpeechlyEnv == SpeechlyEnvironment.OnDevice)
            {
                decoder = new OnDeviceDecoder(
                  async () => await Platform.Fetch($"
{Application.streamingAssetsPath}/SpeechlyOnDevice/Models/{ModelBundle}"),
                  deviceId: Platform.GetDeviceId(SystemInfo.deviceUniqueIdentifier),
                );
            }
            if (SpeechlyEnv == SpeechlyEnvironment.Production || SpeechlyEnv == SpeechlyEnvironment.Staging)
            {
                decoder = new CloudDecoder(
                  apiUrl: SpeechlyEnv == SpeechlyEnvironment.Production ? null : "https://staging.speechly.com",
                  appId: String.IsNullOrWhiteSpace(this.AppId) ? null : this.AppId,
                  deviceId: Platform.GetDeviceId(SystemInfo.deviceUniqueIdentifier),
                );
            }
            // Wait for connect
            Task task;
            task = SpeechlyClient.Initialize(decoder, AudioProcessorSettings, SpeechRecognitionSettings,
preferLibSpeechlyAudioProcessor: SpeechlyEnv == SpeechlyEnvironment.OnDevice);
            yield return new WaitUntil(() => task.IsCompleted);
            while (true)
            {
                // Fire handlers in main Unity thread
                SpeechlyClient.Update();
                int captureRingbufferPos = Microphone.GetPosition(CaptureDeviceName);
                int samples;
                if (captureRingbufferPos < oldRingbufferPos){samples = (waveData.Length - oldRingbufferPos) +
captureRingbufferPos;}
                else{samples = captureRingbufferPos - oldRingbufferPos;}
                if (samples > 0)
                {
                    // Always captures full buffer length (MicSampleRate * MicBufferLengthMillis / 1000 samples),
starting from offset
                    clip.GetData(waveData, oldRingbufferPos);
                    oldRingbufferPos = captureRingbufferPos;
                    SpeechlyClient.ProcessAudio(waveData, 0, samples);
                }
                if (WATCH_VAD_SETTING)
                {
                    if (this.AudioProcessorSettings.VADControlsListening != wasVADEnabled)
                    {
                        SpeechlyClient.AdjustAudioProcessor(vadControlsListening:
this.AudioProcessorSettings.VADControlsListening);
                        wasVADEnabled = this.AudioProcessorSettings.VADControlsListening;
                    }
                }
                yield return null;}}}}
```

Figure 51 -MicToSpeechlyScript

6 –HideShowMenuScript: this script is used for hide and show settings menu and make the game stop until the user click resume to back again to the game.

```
using UnityEngine;
public class HideShowMenu : MonoBehaviour
{
    public GameObject menu;
    bool isPuased = false;
    /// Hides or shows the menu depending on whether the pupuased is true or false.
    ///  This is called by Unity
    public void HideOrShowMenu()
    {
        /// Set the menu to be paused
        if (isPuased)
        {
            menu.SetActive(false);
            Time.timeScale = 1f;
            isPuased = false;
        }
        else
        {
            menu.SetActive(true);
            Time.timeScale = 0f;
            isPuased = true;
        }
    }
}
```

Figure 52 –hide show menu script

6 – NetworkConnectionChecker:

this script used to check the user internet connection and if the user connection has been lost, he stops the game and tell the user to check his connection, if the connection is back the script immediate close himself and run the game again, because our text to speech system and

speech to text depend on the internet connection this script is important for our system.

```
using UnityEngine;
using System.Collections;

public class NetworkConnectionChecker : MonoBehaviour
{
    public float checkInterval = 1.5f; // Interval for checking network status in
seconprivate bool isConnectionLost = false; // Flag to track connection status
    public GameObject menu;
    public GameObject screenComponents;
    private void Start()
    {
        StartCoroutine(CheckNetworkStatus()); // Start checking network status
    }
    IEnumerator CheckNetworkStatus()
    {
        while (true)
        {
            // Check for network connection
            if (Application.internetReachability == NetworkReachability.NotReachable)
            {
                // No network connections
                if (!isConnectionLost)
                {
                    // Perform event when connection is lost (only once)
                    PerformEventOnConnectionLost();
                    Time.timeScale=0f;
                    isConnectionLost = true;
                    menu.SetActive(true);
                    screenComponents.SetActive(false);
                }
            }
            else
            {
                // Network connection available
                if (isConnectionLost)
                {
                    // Perform event when connection is restored (only once)
                    PerformEventOnConnectionRestored();
                    isConnectionLost = false;
                    Time.timeScale=1f;
                    menu.SetActive(false);
                    screenComponents.SetActive(true);
                }
            }
            yield return new WaitForSeconds(checkInterval);
        }
    }
    void PerformEventOnConnectionLost()
    {
        // Perform event when connection is lost
        Debug.Log("Network connection lost!");
    }
    void PerformEventOnConnectionRestored()
    {
        // Perform event when connection is restored
        Debug.Log("Network connection restored!");
    }
}
```

Figure 53 NetworkconnectionCheckerScript

7-GameRunnerScript:

This script is used to run the level several stages.

```
using UnityEngine;

public class GamesRunner : MonoBehaviour
{
    private int _currentGame = -1;
    public FindHiddenObjects[] games;
    /// Runs the new game. If there are no games left in the game list this will do nothing.
    ///  Otherwise it will loop through the games and learn
    public void RunTheNewGame()
    {
        _currentGame++;
        /// Play and learn the game.
        if (_currentGame < games.Length)
        {
            StartCoroutine(games[_currentGame].PlayAndLearn());
        }
    }
}
```

Figure 54- gamerunnerscript

4.3.5 Implementation of a reward game:

4.3.5.1 Screen implemented Design:

This game is designed as a reward for the user who uses the system and pass some levels, we design it to make the user take a rest and get out the mode of learning process, in this game the user play with a young girl that collects some stones and some other 3d models related to our system.

Figure 55 reward game design

4.3.5.2 Implementation and code:

1 – CameraControllerScript: this script controllers the camera and make it go with the player any place that player go to while he navigate the game map.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraControl : MonoBehaviour
{
    public GameObject player;
    private Vector3 cameraLocation;
    // Start is called before the first frame update
    void Start()
    {
        cameraLocation = this.transform.position;
    }

    // Update is called once per frame
    private void LateUpdate()
    {
        this.transform.position = player.transform.position+cameraLocation;
    }
}
```

Figure 56 CamerControllerScript

2 - GameMasterScript:

Controls the game paused and restart and checks if the user get game over or not.

```
using UnityEngine.SceneManagement;
using UnityEngine;

public class GameMaster : MonoBehaviour
{
    bool gameHasEnded = false;
    public float restartDelay = 1f;
   public void EndGame()
    {
        if (gameHasEnded == false) {
            gameHasEnded = true;
            Debug.Log("Game Over");
        }
        Invoke("Restart", restartDelay);
    }
    void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Figure 57 gamemasterscript

3- PlayerControllerScript:

This script used to control the player movement, collecting, move the player to the next game stage, make a game over if player falls, and all these features can be found in this script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class PlayerController : MonoBehaviour
{
    private int scoreCounter;
    public Text scoreText;
    public GameObject level2;
    public GameObject level3;
    public CharacterController controller;
    public Animator animator;
    [Header("Movement")]
    public float speed = 1f;
    public float gravity = -9.10f;
    public float jumpHeight = 1f;
    [Header("Rotation")]
    public float rotationSpeed = 1f;
    [Header("Ground Check")]
    public Transform ground_check;
    public float ground_distance = 0.4f;
    public LayerMask ground_mask;
    Vector3 velocity;
    bool isGrounded;
    public Camera myCamera;
    public AudioClip collectionSound;
    // Start is called before the first frame update
    void Start()
    {
        controller = GetComponent<CharacterController>();
        animator = GetComponent<Animator>();

    }
    // Update is called once per frame
```

Figure 58 - playercontrollerscript

```
    // Update is called once per frame
    void Update()
    {
        isGrounded = Physics.CheckSphere(ground_check.position, ground_distance, ground_mask);
        if (isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        animator.SetFloat("forward", z);
        animator.SetFloat("strafe", x);
        Vector3 move = transform.right * x + transform.forward * z;
        controller.Move(move * speed * Time.deltaTime);
        if(this.transform.position.y < -1)
        {
            FindObjectOfType<GameMaster>().EndGame();
            scoreText.text = "You lose";
        }
        // Rotation
        if (x > 0f)
        {
            transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.Euler(0f, 90f, 0f), rotationSpeed *
Time.deltaTime);
        }
        else if (x < 0f)
        {
            transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.Euler(0f, -90f, 0f), rotationSpeed
* Time.deltaTime);
        }
        // Jumping
        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
            animator.SetBool("jump", true);
        }
        else
        {
            animator.SetBool("jump", false);
        }
   // Gravity
```

Figure 59 playercontollerScript

```
            animator.SetBool("jump", false);
        }
    // Gravity
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "pickup")
        {
            other.gameObject.SetActive(false);
            scoreCounter++;
            scoreText.text = "Score" + " " + scoreCounter.ToString();
            AudioSource lol = myCamera.GetComponent<AudioSource>();
            lol.PlayOneShot(collectionSound);
            if (scoreCounter == 18)
            {
                level2.gameObject.SetActive(true);
            }if(scoreCounter == 4)
            {
                level3.gameObject.SetActive(true);
            }
        }
    }
}
```

Figure 60- playecontrollerscript

Chapter 5: “Conclusions and Future Work”

In our project, we have explored the process of developing an educational augmented reality system using AR, Unity and C#. We have covered the basics of AR development, including the game development in our system, designing the user interface, and programming the logic. We have also discussed important topics such as text recognition, text to speech and augmented Reality.

Throughout the book, we have used practical examples and step-by-step instructions to help you understand the concepts and techniques involved in AR development and Game Development, and We have reviewed in the book the history of video game development, as well as the differences between virtual reality, augmented reality, and mixed reality.

However, AR development and Game development is a complex and constantly evolving field, and there is always more to learn.

5.1 Conclusion:

As we reach the conclusion of our project, it is essential to reflect on the journey we have embarked upon and the transformative potential of EduAR in reshaping the educational landscape. In an era dominated by technology and ever-shortening attention spans, it has become increasingly challenging to engage learners and foster a genuine passion for knowledge. However, by harnessing the power of augmented reality, EduAR offers a beacon of hope, illuminating a path towards an enriched and interactive learning experience.

Throughout this project, we have delved into the intricate workings of EduAR, exploring the scientific and educational theories that underpin its design. By integrating moral reward systems, continuous

encouragement, and multi-sensory interactions, we have demonstrated that learning can be both enjoyable and effective. Our vision for EduAR is not to replace traditional education, but rather to supplement and enhance it, ushering in a new era of interactive and engaging learning experiences.

In conclusion, EduAR stands as a symbol of hope and progress in the face of the challenges that modern education presents. It is our belief that by embracing the potential of augmented reality and continually refining our approach to learning, we can foster a brighter future for all. The journey does not end here; rather, we have merely taken the first step towards realizing a world where education is a stimulating and rewarding experience for everyone. Let us forge ahead, united in our commitment to harnessing the transformative power of technology, and build a world where education is truly the cornerstone upon which nations are built.

5.2 Future work:

While EduAR has made significant strides in revolutionizing the educational experience, our journey is far from over. As we continue to explore the vast potential of augmented reality, we remain committed to developing innovative ideas and expanding our reach within the realm of interactive education. In this chapter, we will delve into some of the exciting future works that we have planned, all aimed at achieving the highest possible benefits for learners.

Augmented Reality Educational Visual Books

One of the promising ideas we intend to pursue involves creating augmented reality-enhanced Visual books. By integrating AR technology with traditional print media, we can bridge the gap between static text and dynamic, immersive learning experiences. Imagine a visual book

filled with illustrations that, when viewed through a smartphone or tablet camera, come to life as three-dimensional objects, complete with detailed educational information and interactive elements.

For example, a science textbook could feature an illustration of a cell. When viewed through an AR device, the cell would transform into a 3D model, allowing students to explore its various components, learn about their functions, and even interact with the model to see how different cellular processes work. Similarly, a history visual book could offer 3D reconstructions of historical events or sites, enabling learners to explore them from different angles and gain a deeper understanding of the past.

We also plan to enhance the interactivity of our augmented reality applications by incorporating more advanced features and technologies. This could include integrating haptic feedback, allowing users to feel simulated textures and forces when interacting with virtual objects. Additionally, we aim to incorporate artificial intelligence and machine learning to create more adaptive and personalized learning experiences, enabling EduAR to cater to each user's unique needs and learning styles.

In conclusion, the future of EduAR is bright and full of potential. As we continue to push the boundaries of augmented reality and explore new avenues for interactive learning, we remain steadfast in our commitment to transform education for the better. With each innovative idea and every new development, we take another step towards making education more engaging, enjoyable, and effective for all.

References

https://www.javatpoint.com/unity-interface

https://www.gamedeveloper.com/business/what-is-the-game-development-life-cycle

https://docs.speechly.com/

https://github.com/wit-ai/wit-unity

https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

https://en.wikipedia.org/wiki/Amusing_Ourselves_to_Death

https://www.wyzowl.com/human-attention-span/

https://www.xmreality.com/blog/augmented-reality-vs-virtual-reality#:~:text=AR%20uses%20a%20real%2Dworld,only%20enhances%20a%20fictional%20reality

https://www.sap.com/mena/products/scm/industry-4-0/what-is-augmented-reality.html

https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html

https://docs.blender.org/

https://helpx.adobe.com/mena_en/photoshop/user-guide.html

https://www.resco.net/blog/what-is-augmented-reality-and-how-does-ar-work/#:~:text=AR%20relies%20on%20sensors%20to,top%20of%20the%20real%20world.

https://www.microsoft.com/en-us/hololens

https://timesofindia.indiatimes.com/blogs/voices/how-is-the-technology-of-metaverse-transforming-the-future-of-banks/#:~:text=In%20essence%2C%20the%20metaverse%20is,and%20function%20effectively%20in%20both.

https://en.wikipedia.org/wiki/History_of_video_games#:~:text=The%20history%20of%20video%20games,games%20on%20a%20video%20display.

https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/#:~:text=Functional%20requirements%20define%20what%20a,also%20known%20as%20quality%20attributes.

https://www.educative.io/blog/complete-guide-to-system-design#:~:text=for%20Engineers%20%26%20Managers.-,What%20is%20System%20Design%3F,through%20coherent%20and%20efficient%20systems.

https://www.educative.io/blog/complete-guide-to-system-design#:~:text=for%20Engineers%20%26%20Managers.-,What%20is%20System%20Design%3F,through%20coherent%20and%20efficient%20systems.

https://en.wikipedia.org/wiki/Software_architecture#:~:text=Software%20architecture%20is%20the%20set,of%20both%20elements%20and%20relations.