Asthma App

Manar Ajhar
manarajhar@hotmail.com

Dr. Suha Al-Naimi
sal-naimi@sharjah.ac.ae

October 23, 2024

Abstract

[Text Here]

1 Introduction

[Text Here]

2 Methodology and Procedures

[Text Here]

2.1 Programming and Mathematical Terms

There are several components and aspect in the fields of programming and mathematics that we need to discuss before heading to the methodology:

- Class: An asset that contains special properties and represents a specific object or entity in a system or program.
- Game Engine: A computer program that is used to develop video games.
- Inheritance: The process of a class to derive the properties from another class to have all its properties and other more. For example, if "Class B" inherits from "Class A", then "Class B" is a sub-class of "Class A", and "Class B" gets access to all (or most) of the properties of "Class A", while also has its own. Inheritance represents an "is a" relationship.
- Singleton: A class that can only have one instance of it.
- Library: A collection of prewritten code that serves a unique purpose to help programmers enhance their work tasks and projects [1].
- Script: A computer file that contains code written in a specific programming language. Some scripts contain only one class, while others can have as many classes as possible.
- Encryption: The process of converting a file in a readable format to an unreadable format to prevent users from reading and tampering with the information [2].
- Decryption: The process of converting a file in an unreadable format back to its original (readable) format [2].
- Axis: A long, straight, and infinite line that represents a specific direction. In 3D graphics, there are three axes (each in their negative and positive directions, respectively):
    1. X-Axis: Left and Right
    2. Y-Axis: Down and Up
    3. Z-Axis: Backwards and Forward
- Transform: The geometrical measurement of an object in a 3D environment. There are three components of a transform:
    - Position
    - Rotation

  - Size

- Renderer: The component of making an object visible under certain properties or rules.
- Physicsbody: The component to make an object have gravity, collisions, and other physics-based behaviours.

2.2 Unity

The program that the team agreed to build this endeavor with is Unity [3]. It is the main program used to develop the application. It is a game engine that has a friendly user interface and allows developers to flexibly program their games with as many utilizations as possible. The version that was used for this project was "2021.3.5f1". In Unity, the programming language that was used was C#.[1] C# is an object-oriented programming language, which means we can create classes and assign them to certain objects.

The target platforms that were desired for this game are smartphones and tablets. They allow augmented reality (AR) technologies, having virtual scenes and objects "blend in" with the physical world. AR is an interesting technology that is being more accessible to consumers. Smartphones also have touch as their input method, which can simulate everyday actions, such as picking-up objects, pressing buttons, or shaking objects. Unity provides packages to make programming AR for smartphones as accessible as possible. These packages will be discussed in the section "**AR Packages**".

However, other tools were used to complete the application. For example, Photoshop was used in this project to design badges, buttons, icons, and other user interface (UI) elements. It also helped change the color or hue of UI elements. The version of Photoshop that was used is "20.0.4". Another software that was used is Blender (Ver. 4.1). It was used to make simple modifications on 3D models and assets and make some 3D animations. A third tool is GitHub (Ver. 3.4.1), which was used to sync all of our work and updates to all the other clients involved in the project, as well as keep track of the project's history and updates. More tools will be discussed throughout this article.

2.3 AR Packages

The AR packages that are implemented for the project are Google's ARCore package and Apple's ARKit for Android and iOS devices, respectively. The reason that these packages are chosen for this project include:

1. Unity can let the user download the packages natively from it

[1] C# is pronounced "C-sharp".

2. They have the ability to detect physical surfaces, such as floors, walls, ceilings, and tables via the device's camera. Therefore, no specific pattern is required, unlike traditional AR methods
3. The base package that is needed to program AR in Unity is called "AR Foundation". It is also downloaded and installed natively. Once installed, the other two packages are used with it, making it less needed to program further for a specific device

Because we are using an AR environment, there needs to be special components to run the application in AR, such as AR cameras and sessions. These are assets that are within Unity. In the phone, when the camera detects a physical surface, an on-screen button will appear to tell the player to start the game. When the button is pressed the virtual scene will be placed on the very spot that the camera picked.

2.4 Loading and Saving

Loading and saving are very important when opening an application to avoid losing progress from any session. It is done using a ".json" file. It stores information in text-form. There is a C# script that reads from the file and writes to it for loading and saving, respectively. It also encrypts the file to avoid users from easily tempering with the data. Each class that needs its data to be saved has an interface (The third definition of the word in the programming glossary) which contains two functions: "`LoadData(GameData _input)`" and "`SaveData(ref GameData _input)`" ("`GameData`" will be explained in the section "**The "`GameData`" Class**"). The class also needs an ID so it can be referred to later on. The IDs are strings of 32 random characters that can be generated in C#. The information that we are saving are:

- The badges earned
- The answers on the action plan
- The settings
- The type of user

The "`GameData`" Class

There is another class that represents the collection of data, called "`GameData`". This class includes dictionaries, where their keys are strings, and their values are of the value type based on the information. The keys of the dictionaries are the 32-character ID strings that were discussed earlier. This class is responsible for keeping track of which data needs to be saved. It also has two more variables. All the variables of the class are presented in the following table:

No.	Variable Name	Variable Type	Variable Description
1.	`_userType`	"`UserTypeEnum`" Enum	To know what is the type of user. In the ".json" file, it is saved as an integer.
2.	`_userTypeSelected`	Boolean	To know if the user has selected their type and agreed to the Terms & Conditions. This is to avoid loading the "First Use" scene everytime.
3.	`_badgesCollected`	"String-Boolean" Dictionary	To load and save which badges have been earned.
4.	`_actionPlanAnswers`	"String-String" Dictionary	To load and save the user's answers of the action plan. When loading, the values will be converted from their text forms to their correct value types.
5.	`_settingsValue`	"String-String" Dictionary	To load and save the user's values in the settings menu. The process of loading the values is similar to that in "`_actionPlanAnswers`".

The mentioned enum type, "`UserTypeEnum`", has three values that represent the possible types of user. "`_userType`" is the variable of this enum type. There are three types of possible users: 1. doctors; 2. patients; and 3. random users. In the beginning of the application (for the very first time only), the user will be presented with a canvas that contains the question "What type of user are you?" and three choices, each with a tick box next to it. Also a "Select" button will be below the choices. After selecting an option and pressing the button, the value of "`_userType`" will change according to the selected value, while the user will be directed to the "Terms and Conditions" canvas, containing a "Next" button. When the button is pressed, the value of "`_userTypeSelected`" will change to '`true`', and both the values of the Boolean and the value of "`_userType`" will be saved. Once the Boolean is saved as '`true`', the user will never be asked the question again. Instead, they will go straight into the menu scene the next time the open the application.

The Processes of Loading and Saving

The process of loading is:

1. Take the entire text from the .json file
2. Decrypt the text
3. Convert the Json format text into information for the class "`GameData`".
4. Load the information of each savable data. In each iteration:
   (a) Try to get the text value based on the key
   (b) Convert the value from text form to the correct value type
   (c) Assign the converted value to the correct variable.

The process of saving is:

1. Save the information of each savable data. In each iteration:
   (a) If, and **only if**, the corresponding list contain's the current object's key, remove the key-value pair from the list based on the key.

(b) Convert the new value into a string

(c) Add (or add back) the key and the new value into the corresponding list.

2. Convert the information from the class "`GameData`" into Json format in text form.
3. Encrypt the text
4. Rewrite the .json file with the new encrypted text

2.5 Programming and Designing the Action Plan

The action plan is a survey that prompts the user to answer questions about their status with asthma. It also has some questions about their personal information such as birthday, name, and gender. The overall number of questions are twenty. However, each question has a different type of answer. The types are:

1. Integer Answers
2. Float Answers
3. String Answers
4. Enumerator (Enum) Answers
5. Answer with Two Strings
6. Date (DD/MM/YYYY Format, all integers)

Eventually, only if all questions are answered, the user will be able to make a PDF form of the plan (The PDF is explained in section "**The Action Plan's PDF Form**"). Also, they are saved (as explained in section "**Loading and Saving**"), and if the user accesses the action plan again, the input fields will be filled with the previously saved answers.

Enums

There were two types enum questions: 1. The color of the inhaler; and 2. The user's gender. These are to ensure that the user do not provide various responses to questions that only need a specific answer. Visually, the input method for answers of enum questions is a dropdown menu, rather than a text field. When the user taps on the menu, a dropdown box will appear below it, listing the possible choices based on the enum. The user can tap on a specific answer or accept the already selected answer.

For the gender enum, there is a third choice called "`Unknown`" in the C# script. However, in the canvas's dropdown menu, the option is displayed as "I do not wish to specify". Also, if this choice is selected, it will be printed as "Does not wish to specify gender." in the PDF form (Explained in section **The Action Plan's PDF Form**).

Two Strings

There was only one question that needed two strings, which is the doctor's contact methods: phone number and e-mail. The phone number must be a string because if it is an integer (or any numerical variable type), the leading zeroes in the sequence will be discarded since they would be treated as numbers. Besides, phone numbers do not provide any mathematical purpose. Visually, there are two input field, each prompting for a contact method. For the phone number, only numerical values can be added. So, in a smartphone, when the input field for the phone number is selected, the on-screen number-pad will appear, rather than the whole on-screen keyboard.

Date

There are three questions that ask for the date: 1. The birthdate of the user; 2. The expiry date of the medication; and 3. The next appointment with the user's doctor. There are three input fields: day, month, and year, and each one is an integer. The former two only accepts two digits, while the latter accepts four.

The Action Plan's PDF Form

All the answers of the Action Plan can be presented in a PDF. This was done using the iTextSharp library [4], which is compatible with C# and Unity. There is a teal button called "Print PDF" in the action plan's UI canvas, which is responsible for creating the PDF file. When the file is created, the very first attributes that are printed are 1. the date and time of printing; and 2. the time zone of the device's current location. Below them, a table is printed showing the personal information of the user, which are:

1. Name
2. Date of Birth
3. Age
4. Gender

The age is automatically calculated by the difference between the date of printing and the date of birth. The others are taken from the user's input from the app. After this, a list of all the questions and their answers (Other than the personal information) are printed across the document. As for the date and time formats:

- All the dates in the PDF are in the following format: "MMMM dd, yyyy (dddd)", for example "September 02, 2024 (Monday)"
- The time format used is in 12-hour format. Also, it measures the timezone by using "UTC" as a base. For example, if the timezone in UTC is 2:01 PM, then "6:01 PM (UTC+4:00)" is printed if the timezone is in the United Arab Emirates, or "4:00 PM (UTC+1:59)" if in Croatia.[2]

[2]Manar has tested printing the action plan in Croatia in late July of 2024. The time zone was

2.6 The Exhibition

The exhibition is a unique experience that users can try. It is similar to seeing exhibits in a museum, and if there is a specific object or display that they want to know more about, they press the corresponding number that is displayed next to the object on their small digital companion. The exhibits that we are showing in the exhibition are the types and components of inhalers, as well as a whole assembled inhaler.

Preparing the Exhibition

In order to make the exhibition visually compelling in a 3D space (Even in AR), some math skills are required. The exhibits are displayed by being spread across in a perfect circle. To make the circle, we need to get the direction of each exhibit and choose a magnitude.

Programmatically, the whole process is done in a "`for`" loop. The loop has an integer variable, called "`_i`", which is the index of each iteration in the loop. Another integer that is involved in the loop is called "`_n`", which is the total number of exhibits. Initially, "`_i`" is equal to zero, and with each next iteration, it increases by one until it equals to "`_n`", exclusively. The following two subsections will describe how the exhibition is created, specifically what is happening in each iteration.

Step 1: Getting the Direction The first thing that we need to do is to instantiate (create a copy and put it in the scene) the current object. The rest of the work in the iteration is done on that instantiated object, which will be the exhibit. Secondly, we need to get the angle for the current exhibit, which is done using the following equation: "$\theta_i = \left(\frac{i * 360}{n}\right)$".[3] In the equation, "$\theta_i$" is the angle of the current index (The angle is in degrees). "i" and "n" correspond to their respective integer values that were described in the heading section. Once we get the angle, we need to calculate its sine and cosine values to establish the direction in the z-axis and the x-axis, respectively, and store the values in a `Vector3` variable:

- $dir_{i,x} = cos\left(\theta_i * \frac{\pi}{180}\right)$
- $dir_{i,z} = sin\left(\theta_i * \frac{\pi}{180}\right)$

In Unity, the standard unit for angles is radians. Thus, we need to convert the angle from degrees to radians before getting the sine and cosine values, by multiplying it with the fraction $\left(\frac{\pi}{180}\right)$.[4] "$dir_{i,y}$" is equal to zero, since we do not want to spread the exhibits across the y-axis, regardless of the value of "i".

Step 2: Assigning the Position and Adding the Script After we get the direction, we multiply the values of "dir_i" with the radius to determine the position of the current exhibit (in meters, which is Unity's default unit for length or distance

accurate.

[3] "θ" is a Greek symbol, pronounced "Theta". It is a variable used in geometry to represent angles.

[4] "π" is a Greek symbol, pronounced "Pi". It is a constant value, which equals to 3.141592

measurement). Therefore, the final function is: "$pos_i = dir_i * r$", where "r" is a positive float, and it represents the magnitude (or the radius of the circle). It will tell how far the exhibits should be from the center. We set the radius to be 500. While exhibits' positions are different from each other, their distances from the center are the same.

Lastly, before the iteration is complete, we add a C# component to the exhibit, called "`ExhibitionObjectScript`". This script will be further described in the section "**Showcasing the Exhibition**".

Showcasing the Exhibition

In Unity, there is a technique called "Raycasting", that is used to make the exhibition work. A raycast is a straight, invisible line that starts from one point and continues infinitely (by default) until it detects an object in the way.[5] It is, conceptually, similar to a laser beam. At the exhibition, the raycasting line/beam starts from the center of the user's camera. There is a special UI for the exhibition, which has the following components:

- A red circle in the center of the screen, which is at the same position as the starting point of the raycasting line
- A back button
- A microphone button which appears only when an exhibit is hit by the line and disappears otherwise
- A text field to display the name of an exhibit

All exhibits have one script component in common, called "`ExhibitionObjectScript`". When the user places the center of his/her camera at an object, the raycasting line will detect if it contains the mentioned script. If so, the object will be highlighted in a specific color, the red circle will turn green, the name of the object will be displayed in the text field, and the microphone button will appear below. When the microphone button is pressed, Dr. Salem will talk about the highlighted object.

2.7 The Draggable Class

There are objects that need to be dragged by the user. To make this work, two classes were created: "`DraggableClass`" and "`DraggableManagerClass`". The former is a class that can be a component for any object in a scene. Basically, when the user taps on an object that has this script, the object will follow the finger's position and movements. This technique is also done using raycasting (Explained in section "Showcasing the Exhibition"), where the beam starts at the position of the

[5] In Unity, you can set a limit for the raycast detection by a number, making it finite.

user's finger on the screen. When the user lets go of the screen, the object will stop following. The class has an enum of four possible states:

1. "Canceled" (The object is not being dragged)
2. "Started" (The object has started being dragged)
3. "Following" (The object is being dragged)
4. "Ended" (The object has stopped being dragged)

The class also contains a Boolean variable, called "`_draggableOn`", to turn on or off the ability to have the object be draggable, and its default value is '`true`'. "`DraggableManagerClass`" is a singleton class used to keep track of all the objects that contain "`DraggableClass`". It also allows other scripts to know which object is being currently dragged by the user.

The Shakable Inhaler

There is a simple feature to allow children to shake a virtual inhaler in the application. They can shake the object by flicking their finger up and down to simulate shaking the inhaler in real time. The inhaler object contains the "`DraggableClass`" component. We can also measure how many shakes were done and how fast each shake was.

2.8 Programming the Games

There are five mini-games established for the application:

1. Matching the letters
2. Card matching
3. Multiple Choice Questions (MCQ)
4. Matching the inhalers
5. Assembling the inhaler components

From a programming's perspective, there is a base-class for all the games that has variables which are applicable to all or most of the games, such as the progress percentage, the assigned badge for the game, or the spawning location of the game's components. However, each game also has its own sub-class, which derives from the base-class mentioned. These sub-classes have variables that are only needed for its specific game. Also in the games, we always provide positive feedback since it is an educational application that is targeted to an audience between the ages of 6 and 13 years, even if they did something incorrectly. Positive feedback gives these demographics encouragement to keep trying to get the answers correctly and learn more about a specific subject.

One special property that all the games have is a meter. It tells how the user progressed in the game so far. The meter has two factors: a counter and a percentage.

The counter tells how many items are completed correctly out of the total of items. In the UI, the counter is displayed as "Completed / Total" fraction. The percentage is the mentioned fraction's ratio multiplied by 100. When something is done correctly in the game, the counter increments by one, and the percentage is updated with it automatically.

Block Matching Games

The matching games are games where the user is given a certain number of blocks and holes, and the user needs to drag the blocks into their correct holes. There are three of them: One for the letters, the other for the inhaler types and components, and the third is for assembling the inhaler. All the blocks have physicsbodies and the "`DraggableClassScript`" components (Which were described in section "**The Draggable Class**"). Because they have physicsbodies, they can fall due to gravity if they are not dragged.[6]

For the letters, there are six blocks and holes, and each block and hole are the shape of a specific letter. The letters are: 'A', 'S', 'T', 'H', 'M', and 'A', which spell "ASTHMA". When a letter block is placed in the correct spot (hole), the following will happen:

1. the hole's renderer will be invisible
2. the game's counter increments by one
3. Dr. Salem will talk about a word that starts with that letter
4. a particle effect of stars will appear once
5. the blocks will no longer have a physicsbody
6. the "`_draggableOn`" variable in "`DraggableClass`" will be '`false`'.

Also, the letter block will rotate indefinitely at 100 degrees per second in the y-axis, counter-clockwise. The user can drag any letter in no specific order. When the game is complete, Dr. Salem will say what asthma is, then reward the user with a badge. The badge is saved is saved in the ".json" file.

The other matching game, which involves the inhalers and their components instead, works the same way. However, there was a visual problem: There are three types of inhalers, which are the reliever, the preventer, and the hybrid. While they are in different color, they have the same physical shape and size. Meaning, while the blocks are different, the holes are identical in color and opacity. So, to solve this matter, there is a world-space canvas above each hole, which displays the name of its corresponding item (whether it is a type of inhaler, a component of an inhaler, or the whole assembled object).

The assembly game is about putting the components of the inhaler together to

[6] In the scene, there is an invisible floor to prevent objects from falling endlessly.

make the whole object. Programmatically, it works that same way as the previous two games, where each block should be dragged to the correct hole. However, the difference is that this is a procedural game, meaning that the blocks must be dragged to their holes in a specific order. Each step has a textual description of what the user should do. With each step done correctly, the meter's counter increases by one, and the next step will be described. The steps of the procedure are:

1. Place the neck block into its hole
2. Place the medicine block into its hole
3. Remove the cap from the neck
4. Place the spacer block into its hole

Once the game is done, the player will be rewarded with a badge.

Card Matching Game

The card matching game is about showing which two cards are identical. The game itself is very well known. In this game's case, though, it is specifically for the triggers of asthma, such as pollens, cigarettes, and pollution. In each session of the game there are 12 cards, aligned in a 4×3 structure. However, there is something additional in this game: Two of the cards are information cards, which only provide random information about asthma. Their front side is a picture of a question mark. Thus, we have ten trigger cards (each two are identical), and two information cards. If the player reveals an information card, the game's meter increments by one, and Dr. Salem will talk about an information about asthma. Otherwise, if the player reveals two identical cards about a specific trigger, the meter increments by two, and Dr. Salem will define the trigger. When the player completes the game, he/she is rewarded with a badge.

Multiple Choice Question Game

There is also a fifth game, which is multiple choice questions. We have made up to 24 questions, and each question has four choices, where one of them is correct. In a game session, only five of them are randomly selected. When a question is selected, their choices' orders are randomly shuffled before they are displayed. Unlike the other games where they take place in the AR world, this is the only game that happens on-screen. When the player answers incorrectly, Dr. Salem provides positive feedback to encourage users to continue. When correctly:

- Dr. Salem praises the player and explains the answer
- The answer buttons are disabled for interaction
- A green "Next" button will appear below When pressed:
    1. The next question will be displayed with its answers

2. the answer buttons will be interactable again
3. the "Next" button will disappear

When all five questions are answered correctly, the user will be rewarded with a badge.

The UI Canvas of the MCQ Game The game has its own UI canvas, since it needs to be on-screen. It has:

- A slider to show progression
- A text to display the current question
- Four buttons, where each button shows one choice
- A text below the buttons for responses

Designing the canvas was challenging for a device as small as a smartphone and in portrait mode, specifically for the buttons. Aligning the buttons on top of each other could prevent space for other contents, such as the questions' text bar, the slider, the response text bar, and Dr. Salem's image. The most ideal structure for the group of four buttons in a small display was to have two buttons above and two below to accommodate enough text for the choices, as well as the other UI elements in the canvas.

2.9 Managers

In programming, there needs to be a class that is responsible for monitoring certain things, maintaining values, and informing other classes of current updates. This is where managers come in. A manager is a singleton class that keep track of certain assets happening across the application and let other classes know about these assets and any other updates. There are several manager classes in this application, one for each set of asset:

- Badges,
- Settings,
- Draggable Objects (Discussed in section "**The Draggable Class**"),
- Action Plan,
- Persistence (Responsible for Loading and Saving Data).

These classes are always available and accessible throughout the game. Usually, they contain lists of objects, items, or components that they are responsible for keeping track of. For example, there is a list for the badges' manager. This list is for a class that contains all the properties of a badge, such as:

- Name (String)

- ID (String, which it's used as a key, that is described in section "**Loading and Saving**")
- Sprite (Sprite)
- Description (String)
- Description on How to Earn It (String)
- Collection Status (Boolean)

When a badge is earned, a setting attribute's value is changed, or an answer of a question in the action plan has changed, their respective managers will store their changes and inform the persistence manager to save the requested change. The persistence manager will then write the changes into the ".json" file to save them, exactly as described in "**Loading and Saving**". In loading, when opening the app, the persistence manager will read the data from the file, convert the values from their text forms, and give the converted values to the manager responsible for the matter.

2.10 The Indicator

One of the first aspects that had to be programmed for the application was the indicator. It is a feature to tell the user where the target scene is. The indicator has its own C# script and UI canvas. The script has two variables to make this feature work: A camera variable and a target's transform variable. The feature has two cases: on-screen and off-screen, which are both based on the script's two variables. If one of the two variable is not assigned, the whole feature will not be executed. In the on-screen's case, when the scene is far enough from the target, a UI indicator will mark on it. The icon of the indicator will be a squared border. Below the square is a number telling how far the user is from the scene. If the user gets close enough, the indicator will disappear.

However, if the target is not seen by the camera (which would be the off-screen's case), then more work needs to be done to make the indicator tell the user where to have his/her camera face [5]. Also, the icon of the indicator will be changed into an arrow. The process also involves some math. The procedure is:

1. Get the on-screen position of the target (Which is the position of the target in the screen).
    - `Vector3 _pos = _camera.WorldToScreenPoint(_targetObject.transform.position)`
2. Substract the on-screen position of the target by the center of the screen.
    - `_screenCenter = new Vector3(Screen.Width, Screen.Height, 0.0f) / 2.0f`
    - `_pos = _pos - _screenPosition`

3. Get the angle of the on-screen position using $Atan2$.
    - $\theta = Atan2(\frac{\_pos.y}{\_pos.x})$
4. Multiply the angle by $\frac{\pi}{2}$ radians (which is equal to 90 degrees).
5. Get the cosine and sine values of the angle. Multiply the sine value by -1.
6. Get the ratio "`_m`".
    - $\_m = \frac{cos(\theta)}{sin(\theta)}$.
7. Get the bounds:
    - `Vector3 _bounds = _screenCenter * _threshold`.
    - "`_threshold`" is a float variable that ranges between 0.0f and 1.0f, inclusively. It is to tell how close to the edge of the screen should the arrow be. 0.0 means it stays in the center of the screen, while 1.0 means it touches the edge of the screen. We set the float to 0.65. This float variable can be changed by the user in the settings UI (Which is explained in section "**The Settings**").
8. Assign the position of the arrow (indicator) with the following code:

```
if(_cos > 0.0f)
{
    _pos = new Vector3((_bounds.y / _m), _bounds.y, 0.0f);
}
else
{
    _pos = new Vector3((-_bounds.y / _m), -_bounds.y, 0.0f);
}


if (_pos.x > _bounds.x)
{
    _pos = new Vector3(_bounds.x, (_bounds.x * _m), 0.0f);
}
else if (_pos.x < -_bounds.x)
{
    _pos = new Vector3(-_bounds.x, -(_bounds.x * _m), 0.0f);
}
```

9. Assign the UI position of the arrow to the values of "`_pos`".
10. Assign the UI rotation of the arrow in the z-axis to the value of θ radians. The rotation in the x and y axes will stay 0.0.

In the settings UI, there is a tick box that the user can toggle with to switch on or

off the feature. The tick box and the Boolean variable that it is based on will be explained in section "**The Settings**". The name of the tick box is "Indicator On".

In the application, the default target is the main scene, where Fasty is with his rocket spaceship and the main scenery. However, when playing a mini-game (with the exception of the MCQ game), the target will be the scene of the game, for example, the holes an the matching games, or the cards' set in the cards game.

2.11 The Settings

Obviously, all users have different preferences of certain aspects. To have the user decide on these preferences, we need a settings UI. It controls and keeps track of utilities on how the user wants to experience the application. As of now, there are six settings that the user can control:

1. Languages ("`LanguagesEnum`" Enum)
2. Indicator On (Boolean)
3. Indicator Threshold (Ranged float between 0.0 and 1.0, inclusively)
    - This setting's UI element will be disabled if the "Indicator On" Boolean is set to '`false`'.
4. Voice Dialogues On (Boolean)
5. Text Dialogues On (Boolean)
6. Exhibition Raycast Distance (Ranged float between 10.0 and 100.0, inclusively)

The settings have their own UI canvas as well as their own manager class. Also each type of setting has their own sub-class. The settings' super class is a generic class, meaning the super-class have variables where their type is not specified. The type would need to be specified in each sub-class. Let us assume an example:

```
public class SettingClass
{
    ...
}

public class SettingGenericClass<T>:SettingClass
{
    protected T _value;

    public T GetValue()
    {
        return _value;
```

```
    }

    public void SetValue(T _input)
    {
        _value = _input;
    }
    ...
}

public class BooleanSettingClass : SettingClass<bool>
{
    ...
}
```

In the following piece of code, "`SettingGenericClass<T>`" is the super-class (and is a sub-class of "`SettingClass`"). The declared "`T`" represents the unspecified type of variable. All the properties that are of type "`T`" (such as "`_value`", "`GetValue()`", or "`_input`") are not of any specific type, but do correspond to one generic type. When declaring the class as a variable type or a super-class for another class, the type **MUST** be specified. In other words, the "`T`" needs to be replaced by a specific type. "`BooleanSettingClass`" is a sub-class that inherits from "`SettingGenericClass`". Notice that the "`T`" in the inheritance has been replaced with "`bool`", which means that all of the properties that were of type "`T`" in the super-class become Booleans in this specific sub-class.

There are three types of settings, and each of the types inherits from "`SettingClass`". The following table will summarize the types and UI input methods:

No.	Variable Type	UI Input Method
1.	"`LanguageEnum`" Enum	Flag Icon
2.	Boolean	Toggle Button
3.	Float	Slider and Input Field

In the setting's UI canvas, there is a panel for each setting. Only one type of setting is an enum, which is the language select. As of now, there are two languages: English and Arabic. The languages are presented in country flags: For English, the icon is the left half of USA's flag and the right half of UK's flag merged together, whereas in Arabic, the icon is the flag of the UAE. The default language is English. In the panel of the language select, the displayed icon will be the flag of the selected language. If the user taps on the icon, it will change, and the corresponding language

should change with it (Although, for the application to work in Arabic has not been implemented yet). The values of "`LanguageEnum`" are '`English`' and '`Arabic`'.

For the Boolean settings, their panels contain toggle buttons. Their defaults values are '`true`'. The toggle button will have a tick mark if the value of its corresponding Boolean is '`true`', and not if '`false`'. For floats, there are two UI input methods in their panels: a slider and an input text field. If the value of one of them is modified, the value of the other will change with it, automatically. For the text field, if the user inputs a number that exceeds or deceeds the allowed range of its float setting, the input field will retype its value to maximum or minimum. Lasty, once the user is done modifying the settings, they can either press the "Confirm" button to save the changes into the ".json" file, or press the "Cancel" button to load the previous settings' values from the file. The loading and saving processes are done by the settings' and the persistence managers.

Other Assets

The voice acting of Dr. Salem and Fasty was done by Said. The animations were done by two student in the University of Sharjah, as well as an external talent. The talents were Alaa Laghoub and Fatima Alqaydi, where they used Blender to animate Fasty's animations, such as:

- Jumping off the rocket
- Greeting the player
- Cheering
- Staying idle

. However, there is another essential animation that had to be implemented, which was have Fasty show how to use the inhaler. This particular animation was so challenging that we needed assisstance from outside the university's domain. Thus, we contacted an external talent via a platform called Upwork. The name of the talent is Vaishali Goyani. She did an outstanding job on how Fasty should use the inhaler. The procedure is:

1. Shake the inhaler for three times
2. Remove the cap from the spacer
3. Place the spacer's mouthpiece into Fasty's mouth firmly
4. Press the medicine
5. Count to 10 seconds with his fingers (A world-space canvas appears above of Fasty's head when he is counting, and the canvas also displays the countdown.).

All animations files were saved as ".fbx" files.

3 Evaluation

[Text Here]

4 Conclusion and Future Work

[Text Here]

5 Definitions

- Cosine: The ratio of the adjacent side of a corner in a right triangle to the hypotenuse of the triangle: $cos(\theta) = \frac{adjacent}{hypotenuse}$.
- Sine: The ratio of the opposite side of a corner in a right triangle to the hypotenuse of the triangle: $sin(\theta) = \frac{opposite}{hypotenuse}$.
- Raycasting: using a straight line that starts from one point and continues infinitely (by default) until it detects an object in the way.
- Canvas: A UI element that contains other elements, including (but not limited to) text, images, buttons, or other canvases.
  - On-Screen Canvas: A canvas that is always fixed in the display's screen.
  - World Space Canvas: A canvas that exist in a world or scene with other objects.

5.1 Types of Programming Variables

- Enumerator: A variable type in programming that has a certain number of possible values where one is selected, for example:
  - InhalerColor:
    1. Blue,
    2. Brown,
    3. Orange,
    4. Purple,
- Integer: A whole number, such as: 0, 1, 3, -8, 100, 2006.
- Float: A decimal number, such as: 0.0, 0.54, 1.0, 5.2, -10.33, 7604.099.
- String: A text or sequence of characters, such as: "Hello World, 12381 42!".
- `Vector2`: A collection/array of two float variables. The first one is named 'x', and the second one is named 'y'.

- `Vector3`: A collection/array of three float variables. In addition to the two variables in `Vector2`, the third one is named 'z'.
- Boolean: A variable type that only has two possible values: '`true`' or '`false`'.
- List: A dynamic collection/array of values or data of the same variable type. Examples of variable types include (but not limited to) integers, floats, strings, classes and their sub-classes, objects, or other lists.
- Dictionary: A special list that contains data as key-value pairs. The key:
    1. **MUST** be unique
    2. Is associated with a value
    3. **MUST** be a simple value type (typically an integer or a string)
    4. Is a quicker way to index values, even in unordered data [6].
- Interface (three meanings):
    1. A way information is shown in a computer or device to its user. Example of interfaces in this document's topic are the canvases and the elements they contain and the way they are structured and presented [7]. Other examples of interfaces include the green arrows and texts in the assembly game that tell they user what to do, and the unique canvas design in the MCQ game.
    2. The way that a user is able to interact with a program. For example, the buttons, sliders, and input fields that are shown in the canvases [7].
    3. A programming structure that enforces certain properties on an object or class. For example, there can be a phone class, a computer class, and a TV class. All of these classes **MUST HAVE** a "`TurnOn()`" function, but how the function is written in each specific class is up to the programmer [8].

6 Acronyms

- Enum: Enumerator
- AR: Augmented Reality
- UI: User Interface
- UTC: Coordinated Universal Timezone

References

[1] *What is a Programming Library? A Beginner's Guide*. Accessed on: October 4, 2024. Jan. 2023. URL: https://careerfoundry.com/en/blog/web-development/programming-library-guide/%5C#what-is-a-programming-library.

[2] *Encryption and Decryption*. Accessed on: October 4, 2024. 2010. URL: https://docs.oracle.com/cd/E19047-01/sunscreen151/806-5397/i996724/index.html%5C#:%5C~:text=Encryption%5C%20is%5C%20the%5C%20process%5C%20by,its%5C%20original%5C%20(readable)%5C%20format..

[3] *Unity*. URL: https://unity.com/.

[4] *iTextSharp*. URL: https://itextpdf.com/products/itextsharp.

[5] digijin. *How to make a Offscreen Target Indicator Arrow in Unity*. Accessed on: November 2023. Feb. 2013. URL: https://www.youtube.com/watch?v=gAQpR1GNOOs.

[6] *Dictionaries*. Accessed on: October 14, 2024. URL: https://adacomputerscience.org/concepts/struct_dictionary?examBoard=ada&stage=all.

[7] *Interface*. Accessed on: October 14, 2024. URL: https://www.britannica.com/dictionary/interface.

[8] *Interfaces*. Accessed on: October 14, 2024. URL: https://users.cs.utah.edu/%5C~germain/PPS/Topics/interfaces.html%5C#:%5C~:text=Interfaces%5C%20in%5C%20Object%5C%20Oriented%5C%20Programming,on%5C%20an%5C%20object%5C%20(class).