

## Useful Links

### 1. Play A Lucid Dream

The link to the desktop game files can be found [here](#). Please download, unzip the files and double click the game shortcut to launch the game.

# Table of Contents

<b>1. Introduction</b>	<b>4</b>
1.1 Team Name	4
1.2 Level of Achievement	4
1.3 Project Scope	4
1.4 Motivation	4
1.5 User Stories	5
1.5.1 Gameplay and features	5
1.5.2 UX	5
1.6 Player Profiling	5
1.7 Tech Stack	5
<b>2. Storyline</b>	<b>6</b>
2.1 Plot Overview	6
2.2 Plot Development	6
2.2.1 Dream Area 1	6
2.2.2 Dream Area 2	6
2.2.3 Dream Area 3	6
2.2.4 Dream Area 4	7
<b>3. Core Features</b>	<b>8</b>
3.1 Game Controls	8
3.2 Level Design	8
3.2.1 Level 1	8
3.2.2 Level 2	10
3.2.3 Level 3	12
3.2.4 Boss Level	13
3.2.5 Final Cutscene	14
3.3 Login System	15
3.3.1 Overview	15
3.3.2 Firebase Authentication	17
3.3.3 Login and Register Logic	19
3.3.4 Email Verification, Auto Login and Reset Password Logic	21
3.3.5 Save System	23
3.4 Pause Menu	26
3.5 Cutscenes	27
<b>4. Testing</b>	<b>29</b>
4.1 Self-Conducted Tests	29
4.2 Third-Party User Testing	31
<b>5. Management</b>	<b>34</b>

## README

5.1 GitHub	34
5.2 Code Management	34
<b>6. Development Plan</b>	<b>36</b>
<b>7. Acknowledgements</b>	<b>39</b>

# 1. Introduction

## 1.1 Team Name

PestControl

## 1.2 Level of Achievement

Gemini

## 1.3 Project Scope

A Lucid Dream will be a single-player dungeon-crawler roguelike game available on the PC that is set in a dream world featuring a 2D pixel art style and allows for unique gameplay experiences with each replay.

Similar to popular roguelike games in the past such as the Binding of Isaac and Hades, A Lucid Dream's gameplay will primarily consist of players defeating hordes of enemies in each level, with each level consisting of a randomly generated map with rooms. A Lucid Dream will also consist of various dream areas that the levels reside in, with each dream area introducing their own unique terrain mechanics that players will have to adapt to.

Players are able to gain experience upon defeating enemies, which will be used to choose from a plethora of abilities and their respective upgrades.

Further gameplay details will be elaborated upon in the relevant sections below.

## 1.4 Motivation

As avid gamers ourselves, we were inspired by renowned titles in the roguelike genre such as Cult of the Lamb, Hades, Binding of Isaac and Vampire Survivors and hoped to take on the challenge of creating our very own roguelike game. Roguelike inherently provides many avenues to express our creativity, such as the abilities players can choose from, and is part of why we chose to tackle this genre for our very first project. We also wanted players that played A Lucid Dream to relieve the built up stress accumulated in their daily lives through non-repetitive gameplay.

As such, we hoped to not only apply the basic programming concepts we have learnt in our programming modules in our very own project, but also to explore the process of game design and development.

While popular roguelike games delve deep into fantasy related themes, we had yet to see roguelike games which made dreams the focal point of the stories they wished to tell. Aside from

## README

having a unique storyline, we also wanted to make the storyline relatable to our players by incorporating real life experiences.

### 1.5 User Stories

#### 1.5.1 Gameplay and features

1. As an individual who enjoys playing games of the rogue-like genre, I want to be able to play a game that provides non-repetitive gameplay.
2. As a player, I want to be able to gain achievements in the game as I progress in each level.
3. As a player, I want to be able to gain rewards and unlock new items as I progress in the game.
4. As a player, I want to be able to immerse myself in the plot of the game.
5. As a player, I want the game to progressively increase in difficulty in order to put my skills to the test.

#### 1.5.2 UX

1. As a player, I want to be able to immerse myself in the visual aspects of the game.
2. As a player, I want the controls to not be too complicated and intuitive.

### 1.6 Player Profiling

The target audience of our game are young people between the ages of 10 to early 20s. Motivations to play the game include:

- wind down through a fun and engaging game after a long day at work or school
- play a game that piques young people's interest
- challenge themselves in a game of increasing difficulty
- to immerse themselves in a gripping storyline as they play the game

### 1.7 Tech Stack

1. Unity
2. GitHub
3. Firebase
4. Canva
5. LucidChart
6. PixilArt

## **2. Storyline**

### **2.1 Plot Overview**

A Lucid Dream features a little spirit that will act as the main protagonist of the game. The spirit wakes up in an unfamiliar dream-like world with no recollections of the past, from who he was to how he ended up in this dream world. Only a large carnation flower lays by his side. However, the spirit is soon greeted by the hostile denizens of the dream world and has to quickly adapt to his surroundings in order to survive. The spirit picks up the carnation flower to fend off the creatures of the dream and embarks on a long journey of rediscovery - of the dream world, of the secrets it holds and of himself.

### **2.2 Plot Development**

#### **2.2.1 Dream Area 1**

This area serves to introduce the player to the basic mechanics of the game, such as moving, attacking as well as choosing from various abilities. A notable character will also be introduced in this area, a hooded figure. The little spirit will also find a long term goal to work towards in this area, as the hooded figure informs the spirit that he will have to collect 'Shards of Reflection' scattered all across the dream world in order to unlock the Dream Gate that will lead the spirit back to his original world.

#### **2.2.2 Dream Area 2**

This area serves to introduce the player to additional mechanics of the game, such as map mechanics unique to each dream area. The spirit will also unlock an upgrade to its projectile. The spirit will be able to obtain additional 'Shards of Reflection' in this area.

#### **2.2.3 Dream Area 3**

As the little spirit progresses through this dream area, the spirit is disoriented as he remembers glimpses of his past. This dream area features an environment that mimics the familiar real life environment of the little spirit. As the spirit is slowly regaining his memories, the stability of the dream world is adversely affected and in addition to the hostile residents of the dream world, the little spirit will now have to fend off Nightmares as well.

It is in this dream area where the Attune mechanic is introduced, in which the spirit has to tune himself to the same dimension as the Nightmare in order to deal damage to it. The Nightmare will be able to deal damage to the little spirit regardless of dimensions. The ultimate skill upgrade will be unlocked in this area.

### **2.2.4 Dream Area 4**

The little spirit has finally collected all the 'Shards of Reflection' necessary to unlock the Dream Gate and head back to his original world. The little spirit prepares to unlock the Dream Gate with all of the 'Shards of Reflection' he has collected thus far.

However, the hooded figure which has been patiently guiding the little spirit every step of the way through the dream world suddenly steps between the little spirit and the Dream Gate. The hooded figure takes off his hood and it is revealed to be a previous version of the little spirit that was unable to overcome his Nightmares. There is a clash of ideologies as the jaded hooded figure wants the little spirit to remain in the dream world where the little spirit can be forever happy instead of returning to the real world, where only despair and anguish awaits. The true final battle begins. A final upgrade to the projectile is unlocked.

After the final battle, the two little spirits reconcile and the little spirit chooses to return to its original world.

## 3. Core Features

### 3.1 Game Controls

Players will play as the little spirit, the main character of the game. The following table illustrates the little spirit's abilities:

Ability	Control
Move Left	A
Move Right	D
Move Up	W
Move Down	S
Shoot Projectile	Right-Click
Attune	Q
Ultimate Skill	Spacebar

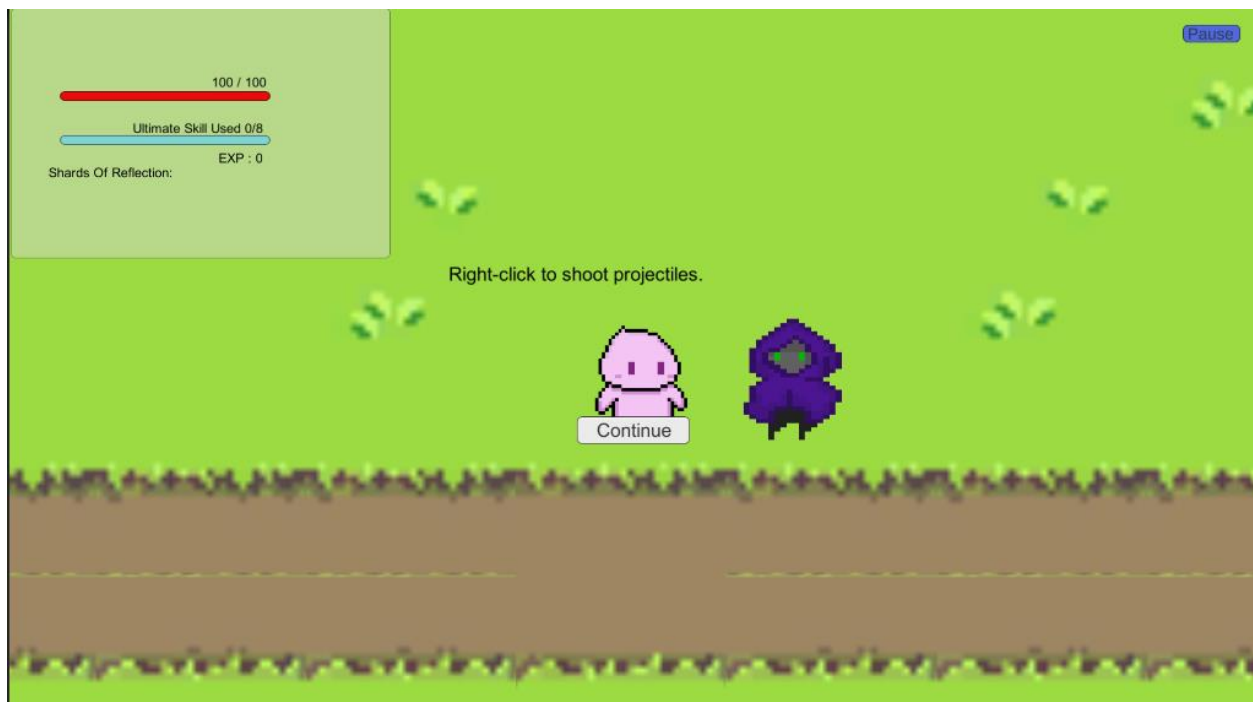
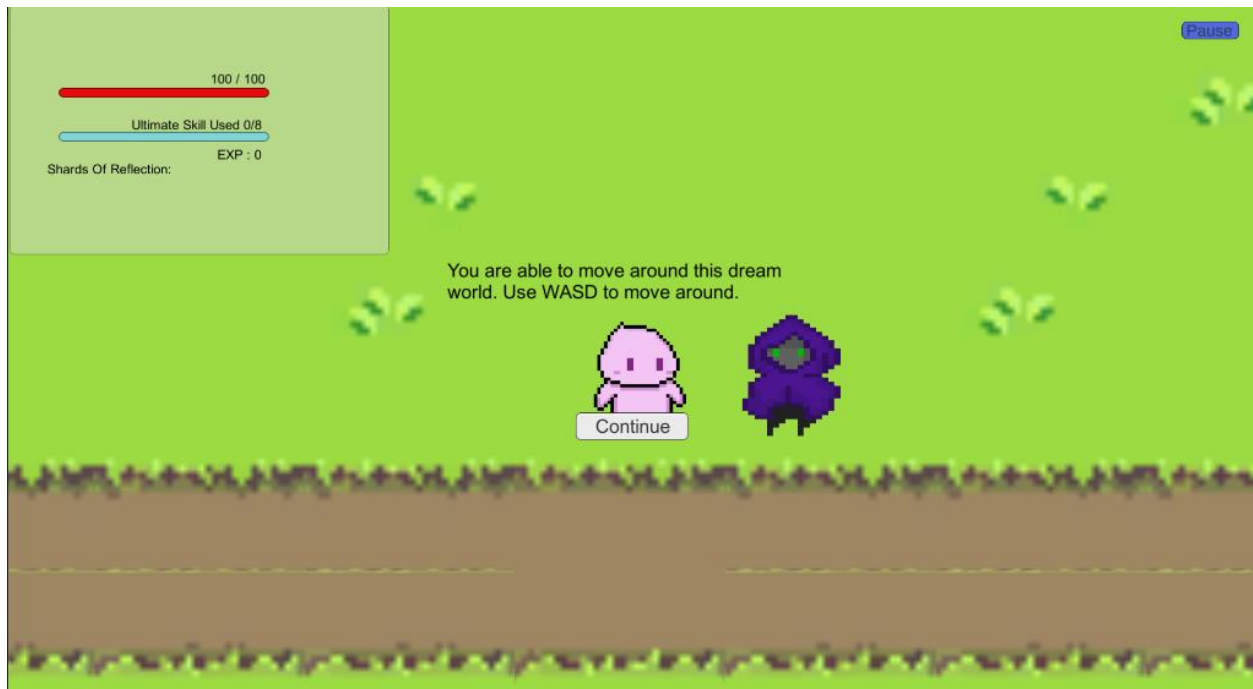
### 3.2 Level Design

#### 3.2.1 Level 1

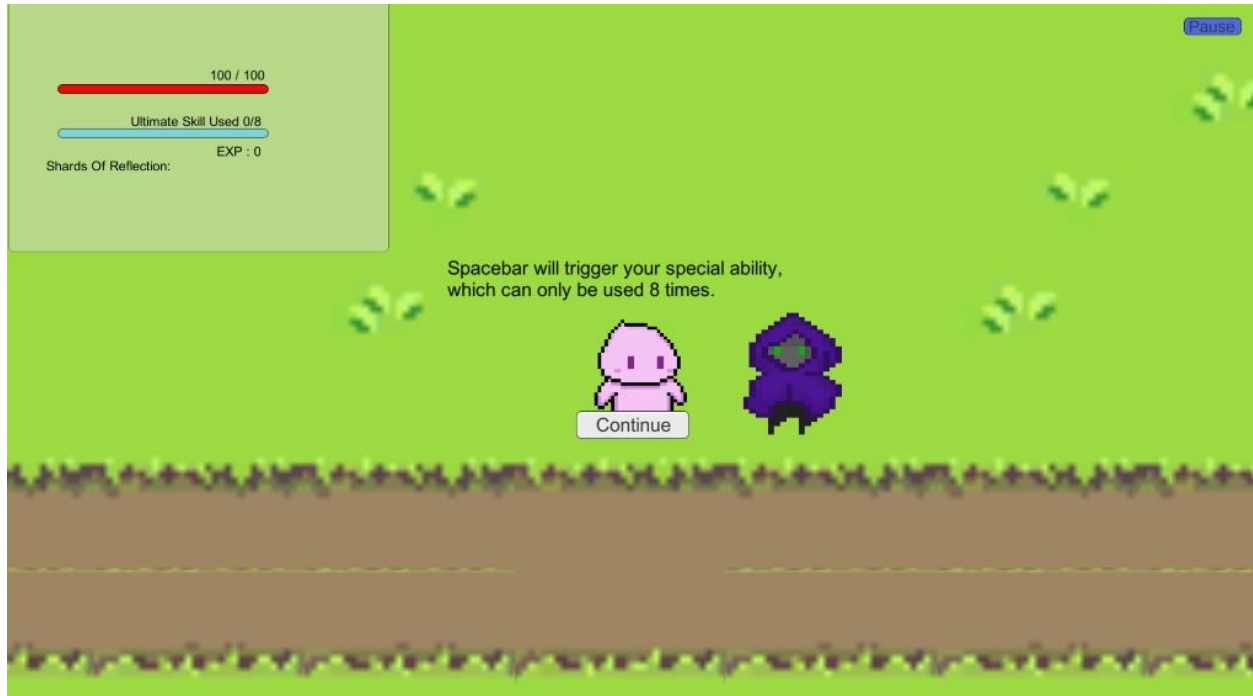
Level 1 starts off with a quick tutorial of the basic controls of the game. That is followed by the spawning of the mobs and the objective is to kill all of them. The player is able to fire projectiles at the mobs. Killing mobs fill up the Ultimate Skill bar and once fully filled, the player is able to activate it using the spacebar. This can only be used a total of 8 times. The ultimate skill, which is Lightning, acts as a safety net that players can fall back on whenever they feel overwhelmed by the number of monsters.



# README



# README



*Diagram 1: Snapshot of Tutorial Cutscene*

The mobs are implemented with a simple Enemy AI (Seek Behaviour), that will follow the player and stop at a certain distance and shoot out projectiles directed at the player. The mobs spawn at a certain rate and the level only ends once they are all dead. The player also gains experience points for every mob killed. The experience points are important as the player is able to upgrade its projectiles/ultimate skill in the subsequent levels.

In this level, only the Hamster mob is spawned, which shoots out a blue fireball. This level is set in the first Dream Area, a grassy patch surrounded by water. The level should not be too difficult for the player to get through.

## 3.2.2 Level 2

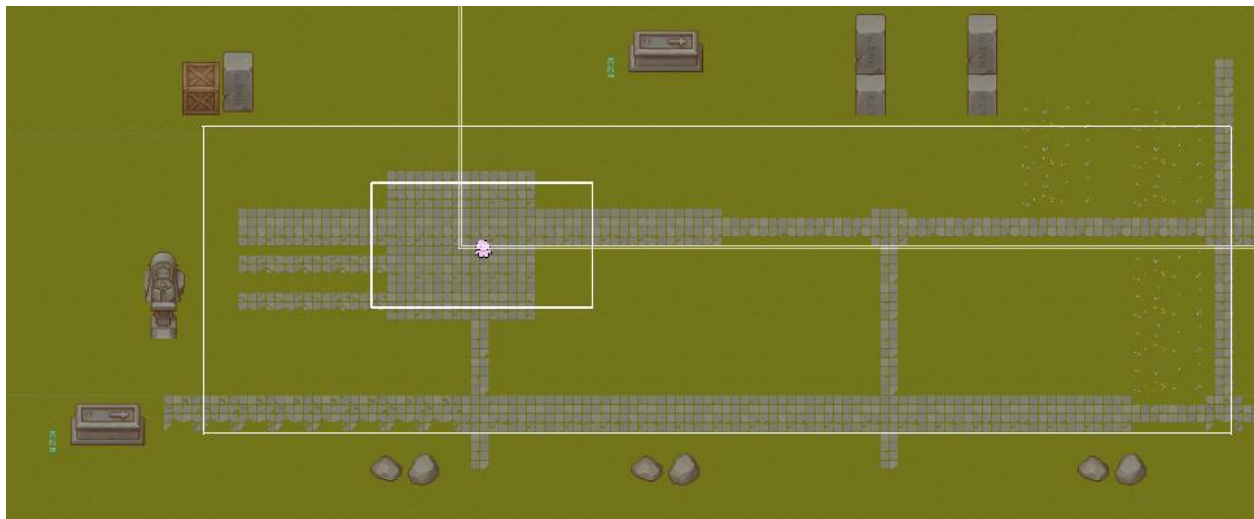
Level 2 starts off with a cutscene that elaborates on the Shards Of Reflection that was collected in Level 1. The Player also receives an upgrade to its projectile, allowing it to shoot arrows instead of carnations, which deals more damage.

## README



*Diagram 2: Purpose of Shards of Reflection*

The level saw the introduction of a new mob, the Chicken, which shoots out ice projectiles that deal quite a fair bit of damage. The Hamster mob is also buffed in this level. Additionally, more mobs are spawned in this level, making it more difficult for the player.



*Diagram 3: Graveyard Area*

This level is set at a graveyard area, which is the second Dream Area.

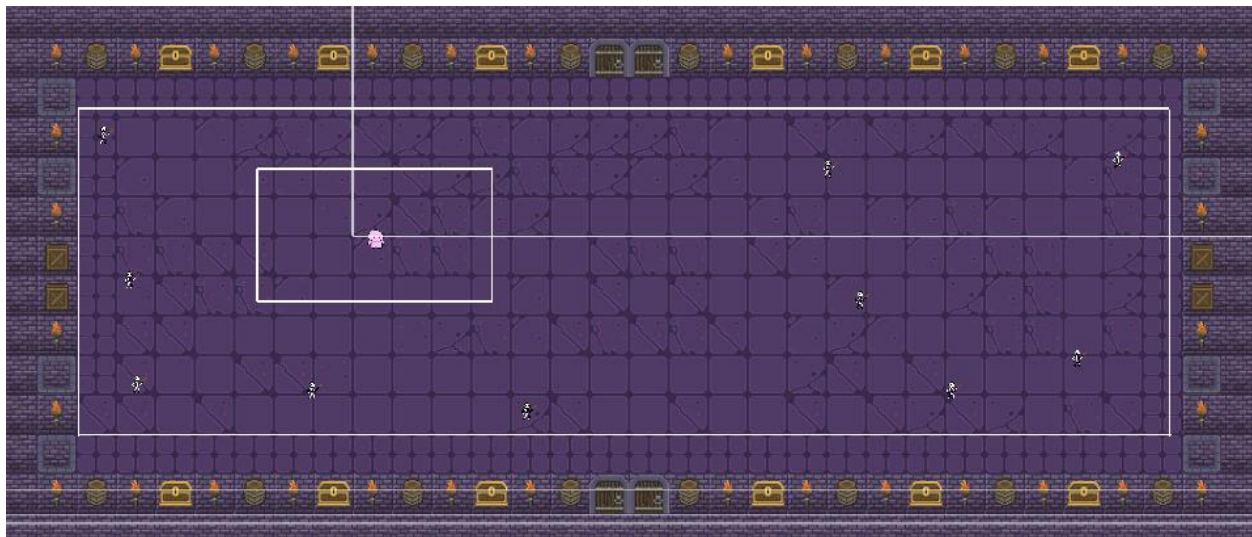
## README

### 3.2.3 Level 3

Level 3 starts with a cutscene that introduces the player to Nightmares. The Hooded Figure tells the player that it could only attack Nightmare mobs if they are attuned to the same dimension as the mob. This introduces the new mechanic in the game, Attune. The player needs to fend off these mobs in addition to the other mobs in the dream area.



*Diagram 4: Attune Mechanic Activated*



*Diagram 5: Dungeon Area*

The player also receives an upgrade to its ultimate skill, an upgraded Lightning. The number of mobs spawned is also much larger and this increases the difficulty for the player. The setting of

## README

this level is much darker than the previous level. This dream area is a dungeon, filled with Skeletons, which is the Nightmare mob.

### 3.2.4 Boss Level

The level starts with the player thinking that they could finally escape, having collected the Shards of Reflection and fending off the Nightmares. However, the Hooded Figure stops the player and reveals itself as the final boss of the game. The true appearance of the Hooded Figure is shown, a variation of the player.

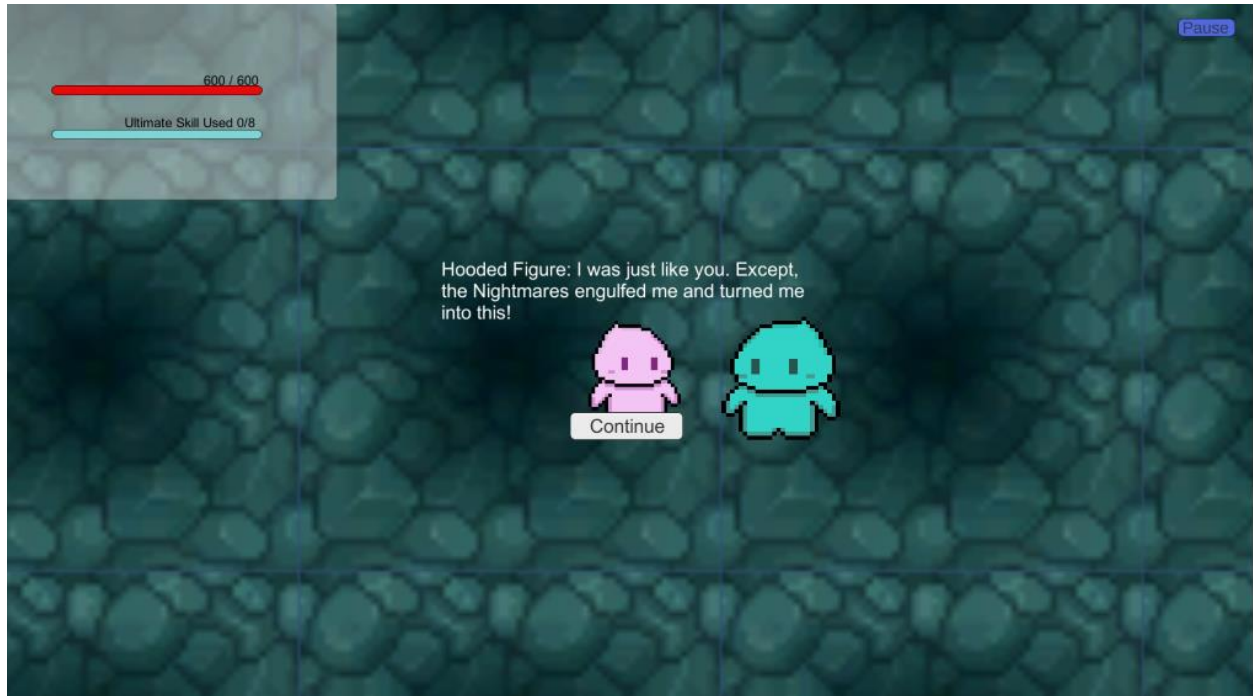
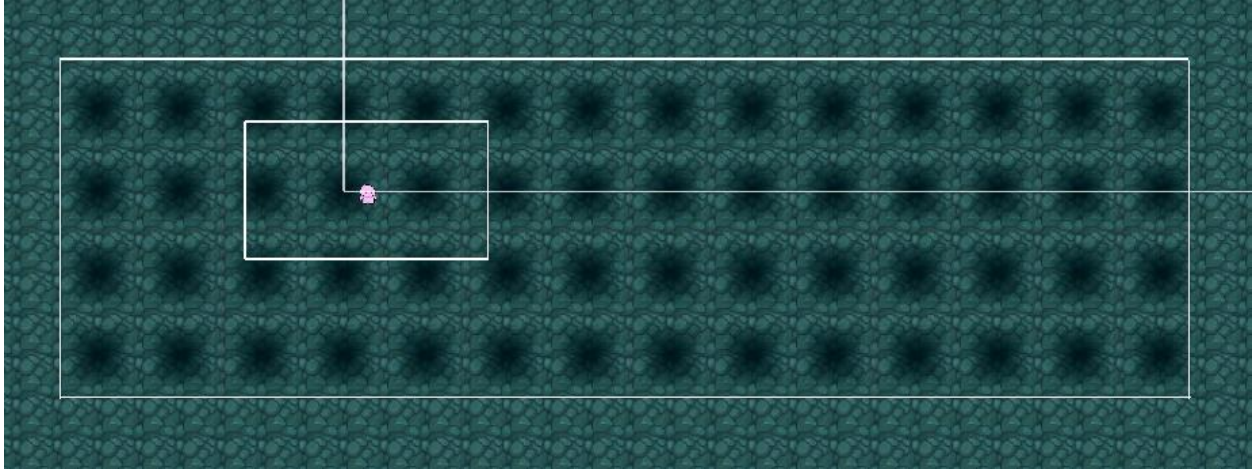


Diagram 6: Boss as a variant of the player

The Boss tells the player that he was once in the same position as the player, but was devoured by the Nightmares and had been stuck in the dream world ever since. The Boss splits its soul into several pieces and attacks the player from all over the place.



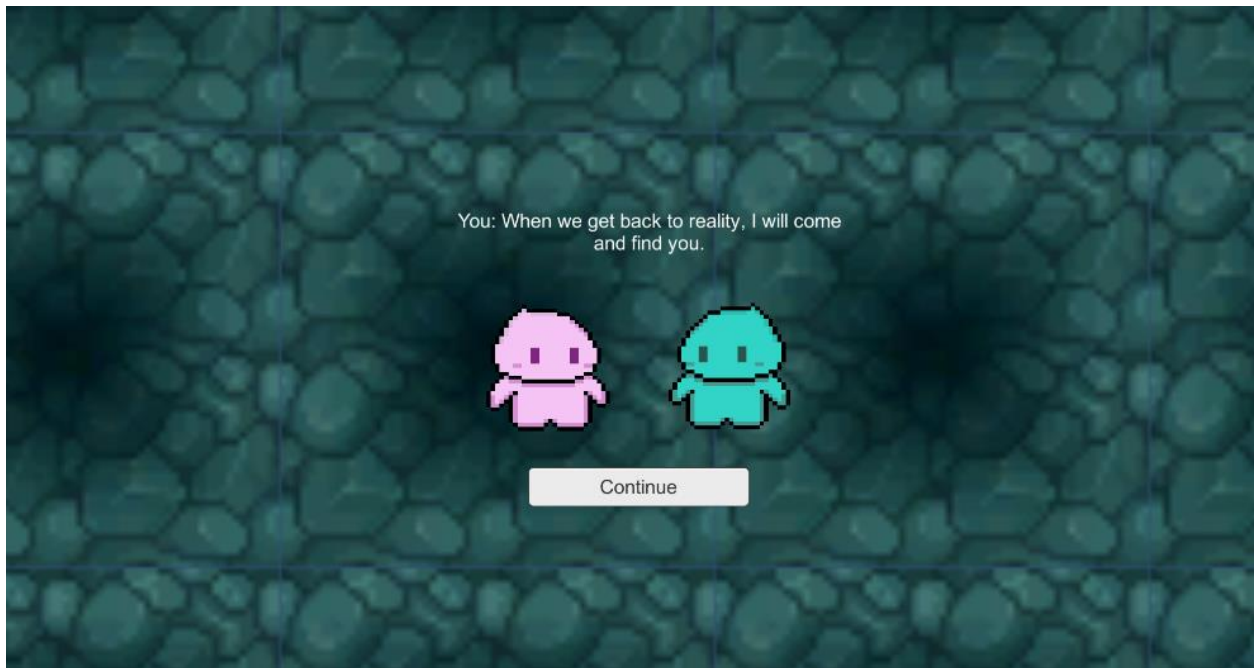
## README



*Diagram 7: Dark Cave Area*

The player, however, will not go down without a fight. The player is now equipped with the final upgrade, a fireball, to replace the arrows. A final showdown commences in this dark cave, which is the final dream area.

### 3.2.5 Final Cutscene



*Diagram 8: Ending Cutscene*

The player defeats the Boss and reconciles. The Boss reveals that both of them will finally be able to escape and the player promises to find the Boss when both of them escape back to reality.

## 3.3 Login System

### 3.3.1 Overview

We have a login system, developed using Firebase, to allow players to save the progress of their game. The login system has the following features:

Features	Details
Login	Players can login with their existing account. Their game progress will be saved to their account, and they can continue their current game progress even after exiting the game.
Register Account	New players can sign up for an account.
Email Verification	New players will have to verify the email address indicated before they are able to login.
Reset Password	Allows existing players to reset their password in case they forgot. An email will be sent to them to allow them to reset it.
Exit Game	Allows players to exit game application

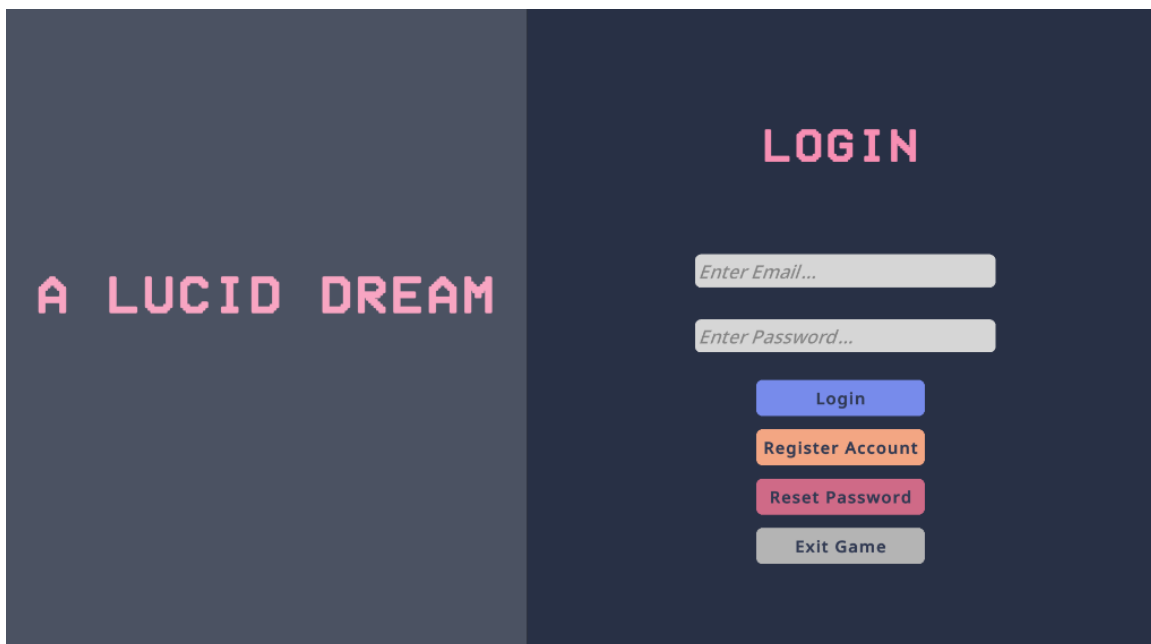
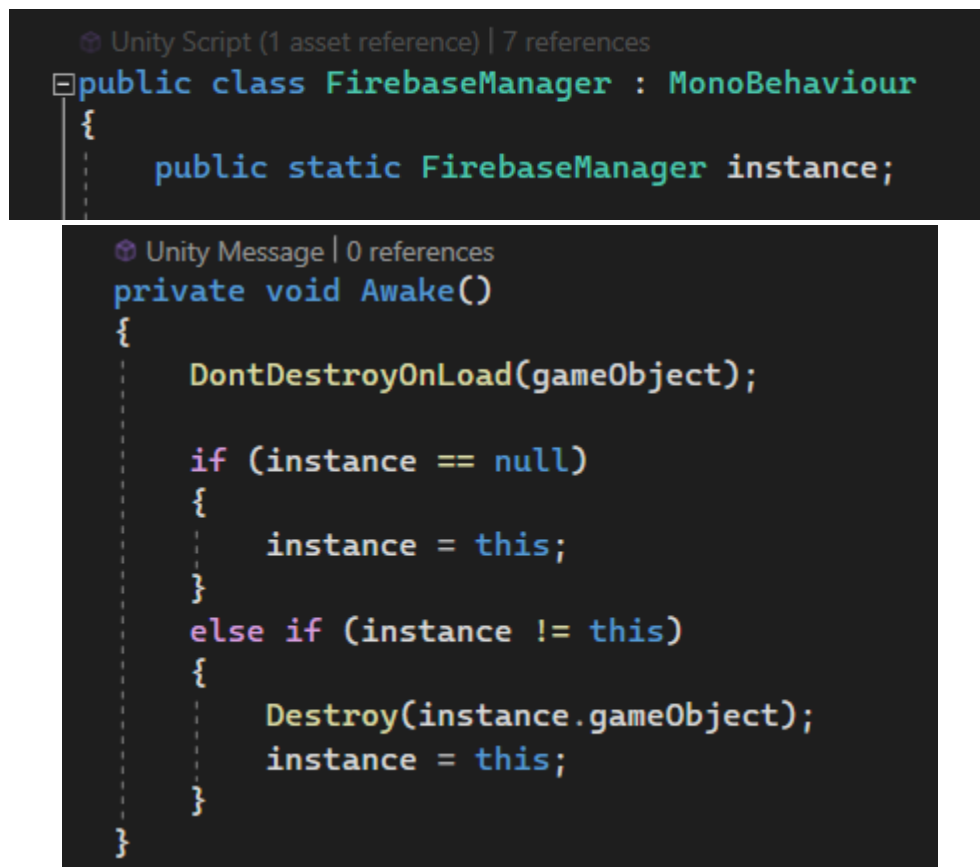


Diagram 9: Snapshot of Login Screen

## README

A Lucid Dream supports the storage and retrieval of user information by utilizing Firebase as our backend. It also supports the display of such information through various UIs and changing stored information based on user inputs. The Model View Controller (MVC) software design pattern decouples the data, presentation and control logic of an application by separating them into three different components: Model, View and Controller.

We adhered to the **MVC** software design pattern by separating the scripts responsible for handling the retrieval of Firebase user information and the scripts responsible for UI display. This reduces the high coupling that arises from the interlinked nature of these features.



```
Unity Script (1 asset reference) | 7 references
public class FirebaseManager : MonoBehaviour
{
    public static FirebaseManager instance;

    Unity Message | 0 references
    private void Awake()
    {
        DontDestroyOnLoad(gameObject);

        if (instance == null)
        {
            instance = this;
        }
        else if (instance != this)
        {
            Destroy(instance.gameObject);
            instance = this;
        }
    }
}
```

Diagram 10: Code Snippets of the FirebaseManager script

Moreover, we have also incorporated the Singleton design pattern for the various classes (FirebaseManager etc.) as these classes are responsible for managing the various UIs and retrieving information from the Firebase backend. As such, when writing the code for our game, other classes will only be able to access the public property instance rather than being able to instantiate the class at will. Moreover, to ensure that no more than one instance is instantiated, the game object will be destroyed if there is an existing game object containing the script.



### 3.3.2 Firebase Authentication

In order to create a login system that will authenticate users that interact with A Lucid Dream, we made use of Firebase Authentication, which provides a plethora of secure sign-in methods to choose from. Since we envisioned A Lucid Dream to be a game that can be played on the desktop, we opted to use the email and password sign-in method.

```
if (loginTask.Exception != null)
{
    FirebaseException firebaseException = (FirebaseException)loginTask.Exception.GetBaseException();
    AuthError error = (AuthError) firebaseException.ErrorCode;
    string output = "Unknown Error, Please Try Again";

    switch (error)
    {
        case AuthError.MissingEmail:
            output = "Please Enter Your Email";
            break;
        case AuthError.MissingPassword:
            output = "Please Enter Your Password";
            break;
        case AuthError.InvalidEmail:
            output = "Invalid Email";
            break;
        case AuthError.WrongPassword:
            output = "Incorrect Password";
            break;
        case AuthError.UserNotFound:
            output = "Account Does Not Exist";
            break;
    }
    loginOutputText.text = output;
}
```

Diagram : Code Snippet from FirebaseManager to handle Authentication Errors

LOGIN

invalidemailaddress

\*\*\*\*\*

Login

Register Account

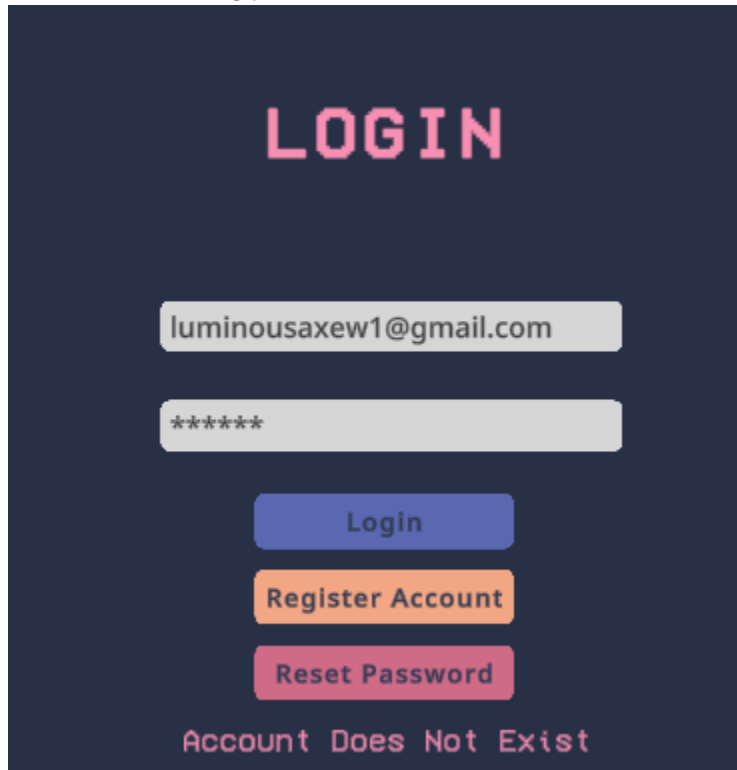
Reset Password

Invalid Email

## README

*Diagram : Login UI with error message prompt upon attempting to login with an invalid email*

The Firebase Authentication API has built-in checks to handle various test cases. For instance, if a user were to key in an invalid email address, an error would occur and this error can be accessed from AuthError Enumeration and the appropriate error message can then be displayed in the Login UI to prompt the user accordingly.



*Diagram : Login UI with error message prompt upon attempting to login with an unregistered email address*

To cite another example, if a user were to key in a valid email address that is not registered with Firebase Authentication, another error message would be displayed to prompt the user that the account does not exist.

### 3.3.3 Login and Register Logic

```
1 reference
private IEnumerator RegisterLogic(string _username, string _email, string _password, string _confirmPassword)
{
    if (_username == "")
    {
        registerOutputText.text = "Please Enter A Username";
    }
    else if (_password != _confirmPassword)
    {
        registerOutputText.text = "Passwords Do Not Match!";
    }
    else
    {
        var registerTask = auth.CreateUserWithEmailAndPasswordAsync(_email, _password);

        yield return new WaitUntil(predicate: () => registerTask.IsCompleted);
    }
}
```

## README

```
else
{
    // Create a user profile and set the username
    UserProfile profile = new UserProfile
    {
        DisplayName = _username,
    };

    var defaultUserTask = user.UpdateUserProfileAsync(profile);

    yield return new WaitUntil(predicate: () => defaultUserTask.IsCompleted);

    if (defaultUserTask.Exception != null)
    {
        user.DeleteAsync();
        FirebaseException firebaseException = (FirebaseException)defaultUserTask.Exception.GetBaseException();
        AuthError error = (AuthError) firebaseException.ErrorCode;
        string output = "Unknown Error, Please Try Again";

        switch (error)
        {
            case AuthError.Canceled:
                output = "Update User Cancelled";
                break;
            case AuthError.SessionExpired:
                output = "Session Expired";
                break;
        }
        registerOutputText.text = output;
    }
    else
    {
        Debug.Log($"Firebase User Created Successfully: {user.DisplayName} ({user.UserId})");

        StartCoroutine(SendVerificationEmail());
    }
}
}
```

*Diagram : Code Snippet of the RegisterLogic() function in FirebaseManager*

When a user registers an account with Firebase Authentication, and manages to get past the various checks implemented (EG: using a valid email address that has not been registered yet, input the same password twice in the “password” and “confirm password” input fields), an account will be created by calling Firebase’s **CreateUserWithEmailAndPasswordAsync()** function as shown in the above diagram.

Before the user will be allowed to login to their freshly created account, the user will have to verify the email address through a verification link sent to the email they registered with.

### 3.3.4 Email Verification, Auto Login and Reset Password Logic

```
private IEnumerator SendVerificationEmail()
{
    if (user != null)
    {
        var emailTask = user.SendEmailVerificationAsync();

        yield return new WaitUntil(predicate: () => emailTask.IsCompleted);

        if (emailTask.Exception != null)
        {
            FirebaseException firebaseException = (FirebaseException)emailTask.Exception.GetBaseException();
            AuthError error = (AuthError)firebaseException.ErrorCode;

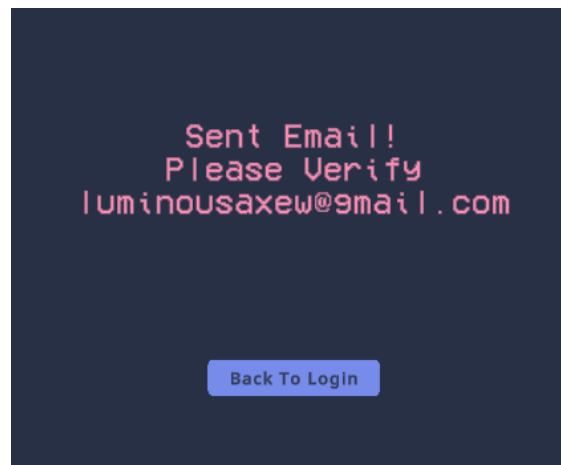
            string output = "Unknown Error, Try Again!";

            switch (error)
            {
                case AuthError.Canceled:
                    output = "Verification Task was Cancelled";
                    break;
                case AuthError.InvalidRecipientEmail:
                    output = "Invalid Email";
                    break;
                case AuthError.TooManyRequests:
                    output = "Too Many Requests";
                    break;
            }

            AuthUIManager.instance.AwaitVerification(false, user.Email, output);
        }
        else
        {
            AuthUIManager.instance.AwaitVerification(true, user.Email, null);
            Debug.Log("Email Sent Successfully");
        }
    }
}
```

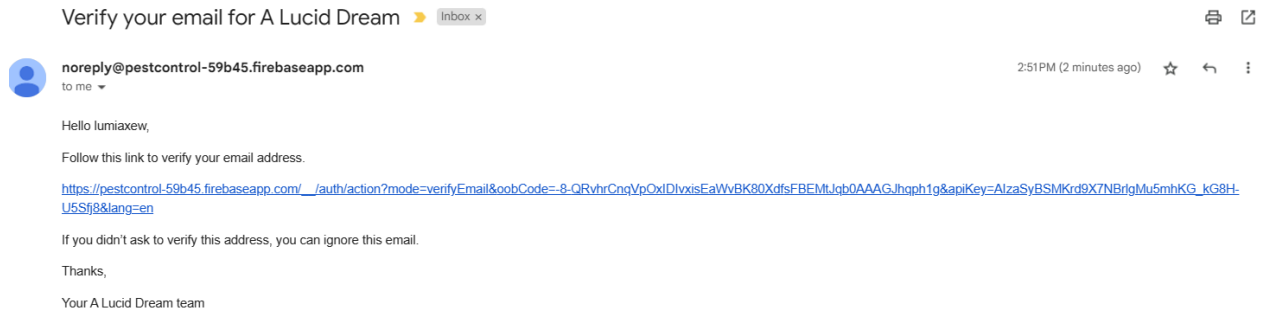
Diagram : Code Snippet of the SendVerificationEmail() function in the FirebaseManager class

While the verification email feature can be viewed as an additional roadblock to gameplay for some users, it is a necessary evil to ensure that the users' data are kept safe and secure. Each registered email address only needs to be verified once through the verification link sent before users are allowed to play A Lucid Dream.



# README

*Diagram : Verification Email Sent UI to prompt users to verify their email address*



*Diagram : Verification Email that users will receive upon successfully registering*

A Verification Email Sent UI will be displayed once users have successfully registered to prompt users to verify their email address.

Since A Lucid Dream will be a game that can be played on the desktop, some users might feel that it is a hassle to have to constantly login to their account each time the game is closed, even by accident. As such, we have implemented an Auto Login feature that automatically logs users in to the last account they used before the game was closed if they did not manually sign out.

```
// If no modifications were made to account details between the previous and current game session,
// user will be logged into the game,
// user will be directed to the Login Screen to login otherwise.
1 reference
private void AutoLogin()
{
    if (user != null)
    {
        if (user.IsEmailVerified)
        {
            GameManager.instance.ChangeScene(1);
        }
        else
        {
            StartCoroutine(SendVerificationEmail());
        }
    }
    else
    {
        AuthUIManager.instance.LoginScreen();
    }
}
```

*Diagram : Code Snippet containing the AutoLogin() function in FirebaseManager*

One caveat is that if the user has successfully registered but **has yet to verify** their email address, the user will not be auto logged in and will still have to verify their email address before they can log into the game.

## README

```
1 reference
private IEnumerator ResetPasswordLogic(string _email)
{
    if (_email == "")
    {
        loginOutputText.text = "Please Enter An Email";
    }
    else
    {
        var resetPasswordTask = auth.SendPasswordResetEmailAsync(_email);
    }
}
```

*Diagram : Code Snippet containing the ResetPasswordLogic() function in FirebaseManager*

If users were to have forgotten their passwords, they can simply key in their email address in the login UI and select the “Reset Password” button and an email to reset the password will be sent to the user’s email address through the SendPasswordResetEmailAsync() function.

### 3.3.5 Save System

To cater to users who wish to play A Lucid Dream across multiple play sessions and have the option to continue on from where they last left off each time, we have implemented a cloud save system that makes use of the Firebase Realtime Database. The Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON and synchronized in real time to every connected client. As a NoSQL database, rather than tables and records, the Firebase Realtime Database is structured in the form of a JSON tree in which added data becomes a node in the JSON structure with an associated key. This provides much freedom in the way we can format the data we wish to save to the database.

Compared to saving the play session data in a local file, saving the game data in a cloud database provides much more merits. For instance, the data is stored securely in the cloud and can be accessed across different desktops if the user wishes to play A Lucid Dream on a different computer. Moreover, the user will be unable to manually edit or tamper the game data as well.

## README

```
[System.Serializable]
22 references
public class GameData
{
    public int deathCount;

    public int shardsCollected;

    public int expEarned;

    public int currentScene;

    // Values defined in this constructor will be the default values
    // the game starts with when there is no data to load
    1 reference
    public GameData()
    {
        this.shardsCollected = 0;
        this.deathCount = 0;
        this.expEarned = 0;
        this.currentScene = 2;
    }
}
```

Diagram : Code Snippet displaying the GameData class

A Lucid Dream makes use of asynchronous functions provided by the Firebase Realtime Database API to load and save game data to the cloud. While creating the save system, we wanted to minimize the number of asynchronous calls necessary for every instance of loading and saving data to reduce the number of bugs that will arise. One such example is code, reliant on loaded data, running before the most updated game data is loaded from the database.

As such, the GameData class is created to be responsible for storing all the data we wish to preserve across play sessions. The class is set to be [System.Serializable] so that it can be converted to raw JSON by the built-in unity JSON formatter and sent to be stored in the Firebase Realtime Database in a single asynchronous call rather than multiple asynchronous calls to save and load various data we wish to store.

```
Unity Script (1 asset reference) | 16 references
public class DataPersistenceManager : MonoBehaviour
{
    private GameData gameData;
    private List<IDataPersistence> dataPersistenceObjects;

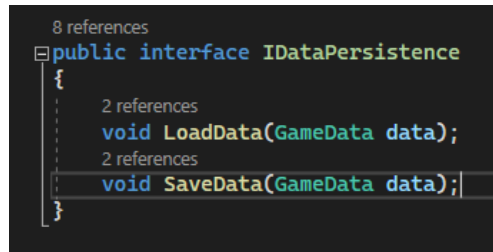
    private string selectedSaveId = "test";

    17 references
    public static DataPersistenceManager instance { get; private set; }

    5 references
    public bool IsLoadingGame { get; private set; }
}
```

Diagram : Code Snippet of the attributes and properties of the DataPersistenceManager class

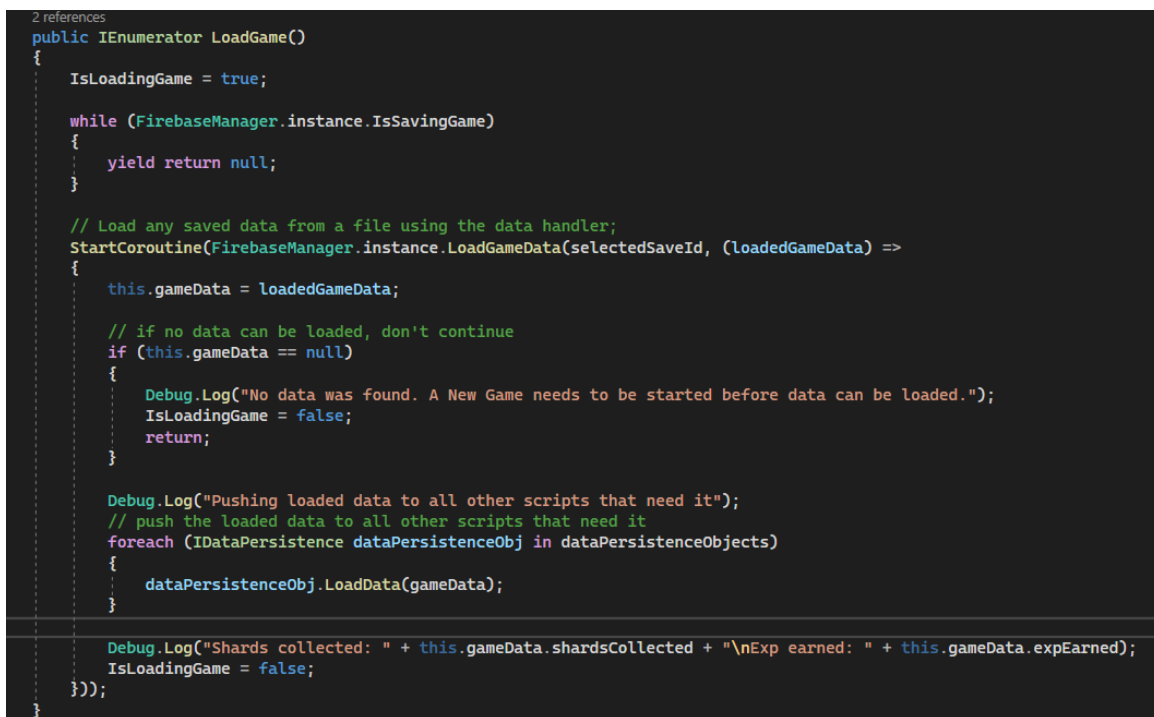
## README



```
8 references
public interface IDataPersistence
{
    2 references
    void LoadData(GameData data);
    2 references
    void SaveData(GameData data);
}
```

Diagram : Code Snippet of the IDataPersistence interface

The DataPersistenceManager class is responsible for managing the save and load state of the game. The dataPersistenceObjects list attribute contains a list of scripts that implement the IDataPersistence interface, which specifies a function for loading data and a function for saving data.



```
2 references
public IEnumerator LoadGame()
{
    IsLoadingGame = true;

    while (FirebaseManager.instance.IsSavingGame)
    {
        yield return null;
    }

    // Load any saved data from a file using the data handler;
    StartCoroutine(FirebaseManager.instance.LoadGameData(selectedSaveId, (loadedGameData) =>
    {
        this.gameData = loadedGameData;

        // if no data can be loaded, don't continue
        if (this.gameData == null)
        {
            Debug.Log("No data was found. A New Game needs to be started before data can be loaded.");
            IsLoadingGame = false;
            return;
        }

        Debug.Log("Pushing loaded data to all other scripts that need it");
        // push the loaded data to all other scripts that need it
        foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
        {
            dataPersistenceObj.LoadData(gameData);
        }

        Debug.Log("Shards collected: " + this.gameData.shardsCollected + "\nExp earned: " + this.gameData.expEarned);
        IsLoadingGame = false;
    }));
}
```

Diagram : Code Snippet of the LoadGame() function in the DataPersistenceManager class

A Lucid Dream will be saved each time right before a scene is loaded and when a scene is being loaded, the most updated game data will be loaded from the database and passed to every single script that implements the IDataPersistence interface to load the current gameplay scene.

Such an implementation reduces the direct dependencies that the DataPersistenceManager has and ensures that the save system can be easily expanded upon if more data needs to be saved in the future.



## README



Diagram : Data saved in a JSON tree in the Firebase Realtime Database

Currently, the save system is only able to save the current level that the user is on. This means that even if the user completes a level, the user will have to click the “Next Level” button first before exiting the game in order to start from the next level. A save slot feature has also been implemented in which users can save up to three game instances.

### 3.4 Pause Menu

We have a pause menu that allows players to stop the game. The player can either Resume the game whenever they are ready, Restart the level, or Exit the level completely.

Features	Details
Resume	Allows players to resume the game.
Restart	Allows players to restart the level.
Exit	Go back to the main menu.

## README

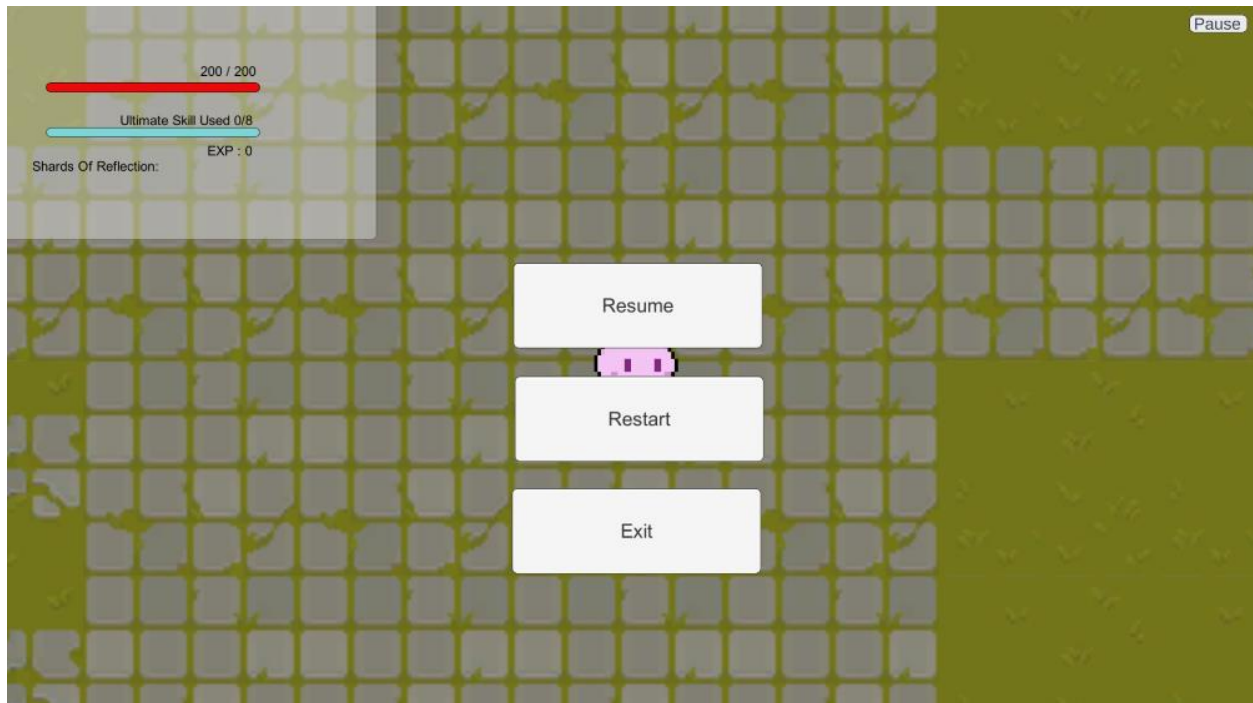


Diagram 12: Pause Menu

### 3.5 Cutscenes

We incorporated cutscenes at the start of each level in order to develop the storyline, as well as go through the mechanics of the game. The Diagram below shows the information of each cutscene.

Level	Synopsis
Level 1	Player wakes up inside the dream world and is met by the hooded figure, who tells the player how to fend off enemies inside the world; a basic tutorial of the game
Level 2	Player defeats the enemies in the first dream area and is met again by the Hooded Figure in the second dream area, who tells the player about the importance of the Shards of Reflection. Hooded Figure also informs the player of the projectile upgrade, allowing the player to shoot out arrows instead of carnations.
Level 3	Player is transported to the third dream area, which is far darker than the previous two. The

## README

	<p>Hooded Figure appears before the player again and tells the player about Nightmares and how to fend them off. Player is introduced to the Attune Mechanic, which allows the player to attack Nightmare mobs. The Hooded Figure also allows the player to upgrade their ultimate skill.</p>
Boss Level	<p>Player is now in the final dream area and questions the Hooded Figure as to why it cannot escape the world when all it did was complete the tasks given. The Hooded Figure reveals itself as the Boss, a fellow spirit who was devoured by the Nightmares. The Boss looks similar to the player and is the most powerful enemy in the game. The Boss tells the player that it is able to split its soul into several pieces and attack the player.</p>
Final Scene	<p>Player and the Boss reconcile and escape back to reality.</p>

## 4. Testing

### 4.1 Self-Conducted Tests

After the creation of every feature, we test the feature and identify any bugs that need to be fixed.

While developing our game, we made use of a combination of **unit**, **integration** and **system testing** to ensure that the various features of our game function as expected individually and work together. This was accomplished through rudimentary **developer testing** while developing the various features.

For instance, we made use of positive and negative test cases. **Positive** test cases are designed to produce **valid behaviors** while **negative** test cases are designed to produce **unexpected behaviors**.

Features	Test Cases
Login	Positive: Input an email address registered with Firebase Authentication along with the registered password  Result: Debug message produced to indicate that the user is successfully logged in.
	Negative: Input an email address that is not registered with Firebase Authentication  Result: Error message produced to indicate that user is not found.
	Negative: Input an email address registered with Firebase Authentication and an incorrect password  Result: Error message produced to indicate that the wrong password has been input.
Register	Positive: Input a valid email address with a strong password (more than 6 characters)  Result: Debug message produced to indicate that the user has been successfully registered.

## README

	<p>Negative: Input an invalid email address / input a valid email address with mismatching passwords in the “Password” and “Confirm Password” fields.</p> <p>Result: Respective debug messages produced to indicate that a valid email address / matching passwords has to be input.</p>
Attune Mechanic	<p>Positive: Activate Attune Mechanic using Q and attack Nightmare mobs.</p> <p>Result: Damage is dealt to the Nightmare mobs.</p>
	<p>Negative: Attack Nightmare mobs without activating Attune Mechanic.</p> <p>Result: No damage is dealt to Nightmare mobs.</p>

Features	Test Cases	Expected Outcome	Actual Outcome	Bug Fix
Login UI	Transitioning between the Login UI and the Register UI	Any existing inputs by the user in the various UI input fields should be cleared when transitioning from Login UI to Register UI and back to Login UI again and vice versa.	Previous inputs were still present in the input fields after transitioning between UIs	Initialized the input fields to a blank string each time a transition occurs.
Player Movement	Transitioning between levels, test player movement	Player able to move around	Movement was disabled	Input Player Movement as Active at the Start() function for Game Controller script
Attune Mechanic	When Attuned, let player take damage from	Damage dealt to player	Damage is only dealt by Nightmare	Since player's tag changes when attuned, a

## README

	mobs		mobs; player is immune to attacks from normal mobs	condition was added to enemy attack in order to identify player correctly and deal damage accordingly
Pause	Test player movement and projectile when game is paused	No movement or shooting of projectiles; no attacks from enemies either	Player movement and projectile is disabled; enemy attacks are also disabled but when the movement keys of A and D were spammed, the ultimate skill bar could be manipulated	Ultimate skill bar as well as the health bar was made non-interactable on Unity Editor

### 4.2 Third-Party User Testing

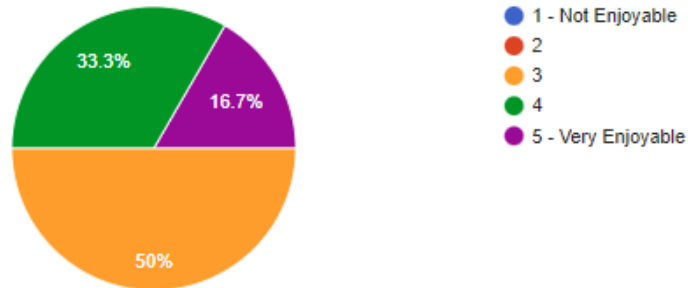
We allow our friends to test out the game and provide feedback through a Google Form in order to identify any bugs that they have encountered as well as feedback regarding user experience.

## README

On a scale of 1 to 5, how enjoyable is it to play the game?

 Copy

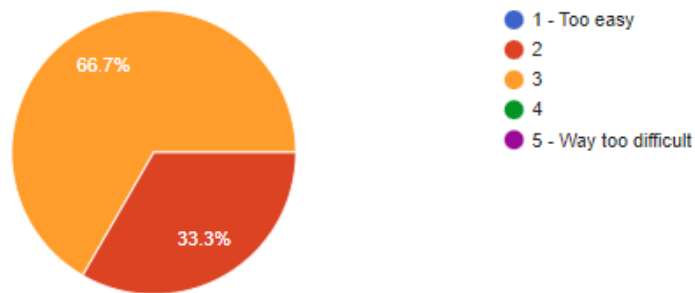
6 responses



On a scale of 1 to 5, how difficult is it to play this game?

 Copy

6 responses



On a scale of 1 to 5, how intuitive are the controls of the game?

 Copy

6 responses

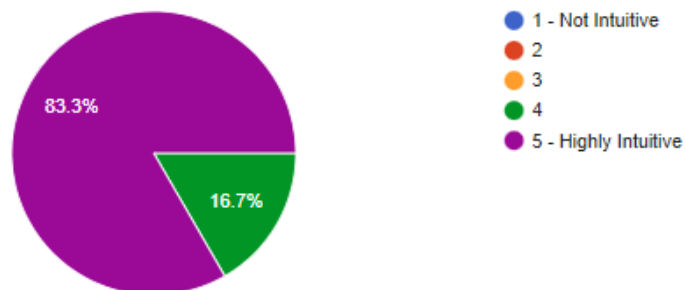


Diagram 13: User Experience Feedback

## README

Were there any bugs that you encountered? (Put NIL if none)

6 responses

NIL

when the game is paused, i could manipulate the skill bar by spamming my A and D keys

*Diagram : Bugs Encountered by Users*

Are there any improvements that you can suggest? (Put NIL if none)

6 responses

storyline can be more developed

more ability upgrades can be added to make it more fun

NIL

currently the game cannot save my progress so it would be nice if i could do so

include map design

theres no map in the game which makes it rather boring in terms of aesthetics

*Diagram 14: Suggested Improvements from Users*



## 5. Management

### 5.1 GitHub

Our team made use of GitHub for version control. In particular, we made use of the branching workflow as it is easier for us to keep track of the commits of the individual features we are each working on. For instance, we have created separate branches for Firebase user authentication related features (EG: Login, Register, Reset Password) and gameplay related features (EG: Player movement, attacking, enemy AI).

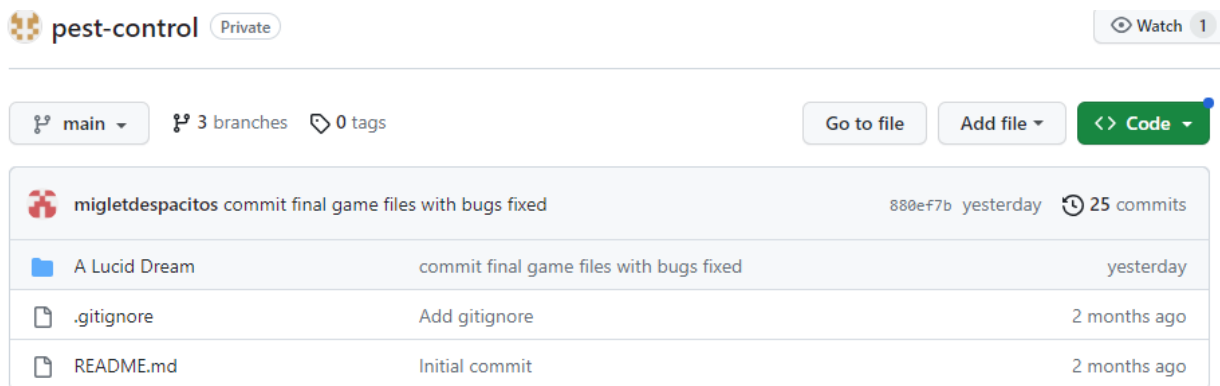


Diagram 15: GitHub Repository

Once a particular feature has been fully developed, we will then make pull requests to merge the feature branch into the main branch, ensuring that the other member has done a Code Review before merging the pull request.

### 5.2 Code Management

Code-level comments are included in scripts to make it readable and easier for anyone to interpret the code.

At the top of each script, we included a one-liner comment, highlighting the purpose of the class.

## README

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 //This class represents the projectile mechanic for the player
6
7 public class Fireball : MonoBehaviour
8 {
9     public GameObject projectile;
10    public GameObject ultimateProjectile; // Ultimate skill's projectile prefab
11    public float minDamage;
12    public float maxDamage;
13    public float projectileForce;
14    public float ultimateProjectileForce; // Ultimate skill's projectile force
15    public float angle;
16    public PlayerStats playerStats;
17
18    void Start()
19    {
20        playerStats = FindObjectOfType<PlayerStats>();
21    }
22 }
```

Diagram 16: Comments highlighting purpose of class

We included one-liner comments next to lines of code for clarity.

```
void Start()
{
    levelCharacter.SetActive(false);
    Time.timeScale = 0; //Pauses the game at the start
    UpdateStep();
}

public void NextStepButton()
{
    levelCharacter.SetActive(true); //character appears when cutscene starts
    currentStep++;
    if (currentStep >= levelSteps.Length)
    {
        levelCharacter.SetActive(false);
        levelPanel.SetActive(false); //Hides the panel
        Time.timeScale = 1; //Resumes the game
    }
    else
    {
        UpdateStep();
    }
}
```

Diagram 17: Example of code-level comments in the script for a cutscene

## 6. Development Plan

MS	Tasks	Details	Period	In-charge
1	Learning	<ul style="list-style-type: none"> <li>- Unity &amp; C#</li> <li>- Github &amp; Git</li> <li>- Login system (using Firebase)</li> <li>- Software Engineering Practices</li> </ul>	2nd and 3rd week of May	Both
	Liftoff	Poster and Video	2nd Week of May	Both
	Setting up	- Setting up Unity Project	3rd week of May	Miguel
		- Setting up Github repository		Wei Sheng
	Planning & Design	Develop storyline	2nd and 3rd week of May	Wei Sheng
		Design of characters and sourcing for relevant assets		Both
	Gameplay	Player mechanics: <ul style="list-style-type: none"> <li>- idle, move, attack, damage logic</li> </ul>	3rd week of May	Miguel
		Enemy mechanics: <ul style="list-style-type: none"> <li>- attack, damage logic</li> </ul>	4th week of May	
	Non-Gameplay	Login system	3rd and 4th week of May	Wei Sheng
		Lobby screen		
	Gameplay	Integrating of basic gameplay scene and login scene	4th week of May	Both
		Self-testing and debugging		
	Milestone 1	Poster	4th week of May	Wei Sheng

# README

	Submission	Video		Miguel
		Readme		Both
		Project Log		Wei Sheng
		Technical Proof of Concept		Miguel
Milestone 1: Ideation (29/5/2023)				
2	Learning	<ul style="list-style-type: none"><li>- Enemy AI</li><li>- Pathfinding Algorithms</li><li>- Pause menu tutorial</li></ul>	5th Week of May to 1st Week of June	Both
	Planning & Design	<ul style="list-style-type: none"><li>- Planning of levels 1 to 2</li><li>- Design of maps for each level</li></ul>	1st Week of June	Both
	Gameplay	Improve gameplay features using feedback from testers and peer review	1st Week of June	Both
		Include health bar on player interface	2nd Week of June	Miguel
	Planning & Design	Design new weapons and abilities for characters and enemies to be implemented by MS3	3rd Week of June	Both
	Gameplay	Build Enemy AI		Miguel
		Implement Ultimate Skill and Loot Drop System		
		Implement Enemy Spawner		
		Create Simple Tutorial		
	Testing	Test Level 1 and get feedback	Both	

# README

	Non-Gameplay	Integrate scenes from Login Page to Level 1		Both
	Milestone 2 Submission	Poster	4th Week of June	Wei Sheng
		Video		Miguel
		Readme		Both
		Project Log		Both
Milestone 2: Prototype (26/6/2023)				
3	Planning & Design	<ul style="list-style-type: none"><li>- Planning of level 3 and boss level</li><li>- Design of map of level 2 and 3</li><li>- Design of boss</li></ul>	5th Week of June	Both
	Gameplay	Complete level 2	5th Week of June to 1st Week of July	Both
	Testing	Test prototype up to level 2 and gather feedback		
	Gameplay	Improve gameplay based on feedback	2nd Week of July	Both
		Implement Attune Mechanic		Miguel
		Complete level 3 and boss level		Miguel
		Integrate all levels and scenes		Miguel
	Non-Gameplay	Implement Pause System		Miguel
		Implement Save System		Wei Sheng
	Testing	Test prototype up to boss level and gather feedback	3rd Week of July	Miguel
		Improve on the levels based on feedback		
	Milestone 3	Poster		

## README

	Submission	Video		Miguel
		Readme		Both
		Project Log		
Milestone 3: Extension (24/7/2023)				
4	Gameplay	Improve gameplay	4th Week of July to 1st Week of August	Both
		Debugging		
	Testing	Extensive self-testing of full prototype and gather feedback	2nd to 3rd Week of August	
	Refinement	Remove unnecessary assets and scripts and refine functionalities		
Splashdown: Game Launch (23/8/2023)				

## 7. Acknowledgements

- InfoGamer - Among Us In Unity - Coding Player Movement [Among Us in Unity - Coding Player Movement \(Lesson 2\)](#) (Reference for Player Movement Code)
- DapperDino - Creating Game Mechanics - Creating a Simple Spell and Enemy Health - 2D Roguelike [Creating Game Mechanics - Creating a Simple Spell and Enemy Health - 2D Roguelike](#) (Reference for Damage Logic and Projectile)
- DapperDino - Creating Game Mechanics - Player Stats and Enemy Health-bars - 2D Roguelike [https://www.youtube.com/watch?v=jrYsmiG4rFw&list=PLS6sInD7ThM3JnoOsur24\\_3h3dvPpfNA9&index=8](https://www.youtube.com/watch?v=jrYsmiG4rFw&list=PLS6sInD7ThM3JnoOsur24_3h3dvPpfNA9&index=8) (Reference for Player Stats and Enemy Health-Bars)
- DapperDino - Creating Game Mechanics - Player Health UI and Dashing Mechanic - 2D Roguelike [https://www.youtube.com/watch?v=5okCh\\_pbMul&list=PLS6sInD7ThM3JnoOsur24\\_3h3dvPpfNA9&index=8](https://www.youtube.com/watch?v=5okCh_pbMul&list=PLS6sInD7ThM3JnoOsur24_3h3dvPpfNA9&index=8) (Reference for Player Health UI)
- DapperDino - Creating Game Mechanics - Base Classes and Spawning Enemies - 2D Roguelike [https://www.youtube.com/watch?v=3CAmSkNt9iU&list=PLS6sInD7ThM3JnoOsur24\\_3h3dvPpfNA9&index=11](https://www.youtube.com/watch?v=3CAmSkNt9iU&list=PLS6sInD7ThM3JnoOsur24_3h3dvPpfNA9&index=11) (Reference for Enemy Spawner)

## README

- Unity Game Development Cookbook Essentials - Chapter 10: Behaviour and AI, Addison, Manning, Nugent (Reference for Enemy AI)
- inScopeStudios - <https://inscopestudios.com/portfolio-items/rpg/> (Sprites for Arrow, Lightning, Fire, Ice Projectiles)
- Referenced the following videos for the AuthUIManager, FirebaseManager, GameManager and LobbyManager classes:
  - [Firebase & Unity | Login & Register | Advanced Account Tutorial #1](#)
  - [Firebase & Unity | Auto Login & Email Verification | Advanced Account Tutorial #2](#)
  - [Firebase & Unity | Profile Pictures | Advanced Account Tutorial #3](#)
- Referenced the following videos for the FirebaseManager, LobbyManager, DataPersistenceManager, SaveSlot, SaveSlotsUI and GameData classes and the IDataPersistence interface:
  - [How to make a Save & Load System in Unity | 2022](#)
  - [How to make a Save & Load System work across Multiple Scenes in Unity | 2022 tutorial](#)
  - [How to Implement Save Slots to Manage Multiple Saved Games in Unity | 2022 tutorial](#)
  - [Save & Load System in Unity - Bug Fixes, Scriptable Objects, Deleting Data, Backup Files, and More](#)
  - [Unity User Data & Scoreboard Tutorial - Firebase Realtime Database](#)
- Font used: [VCR OSD Mono](#)