IM3080 Design and Innovation Project (AY2021/22 Semester 2)
Individual Report

Name: Xue Fuguo

Group No: 2

Project Title: Kopimon Game Project Development

Contributions to the Project (1 page)

Backend:

- Completed sign up & login page UI design
- Connected Firebase to Unity as backend for player authentication

Frontend:

- Coded NPC dialog system triggered by keyboard pressing
- Coded custom animation system for more convenient control over multiple characters
- UI element organization and UI visibility control
- Coded minimap showing live location of characters (player, NPCs, gaming opponents, and other players)
- Completed scene transition issue debugging

Art assets creation and application:

- Created pixelate frame assets for minimap
- Designed Game title shown in start, sign up and login scenes
- Created road signs
- Converted art assets from PNG format to Unity sprites ready to use

Ideation:

- Ideated game mechanics and game reward system
- Ideated for quests and storyline
- Ideated video content
- Mapped storyline nodes and created of flowchart to envision it clearly to ease scene and game creation in Unity
- Suggested to add user-friendly game functions (e.g., exit game button, road signs, etc.)

Teamwork management:

- Introduced Unity Collaborate and Plastic SCM to teammates as a convenient tool for game development
- Transported art assets from Google Drive to Unity
- Demo video recording for final presentation
- Double checked game map
- Gave suggestions on project document consistency (game, poster, and slides should all have pixelated theme)

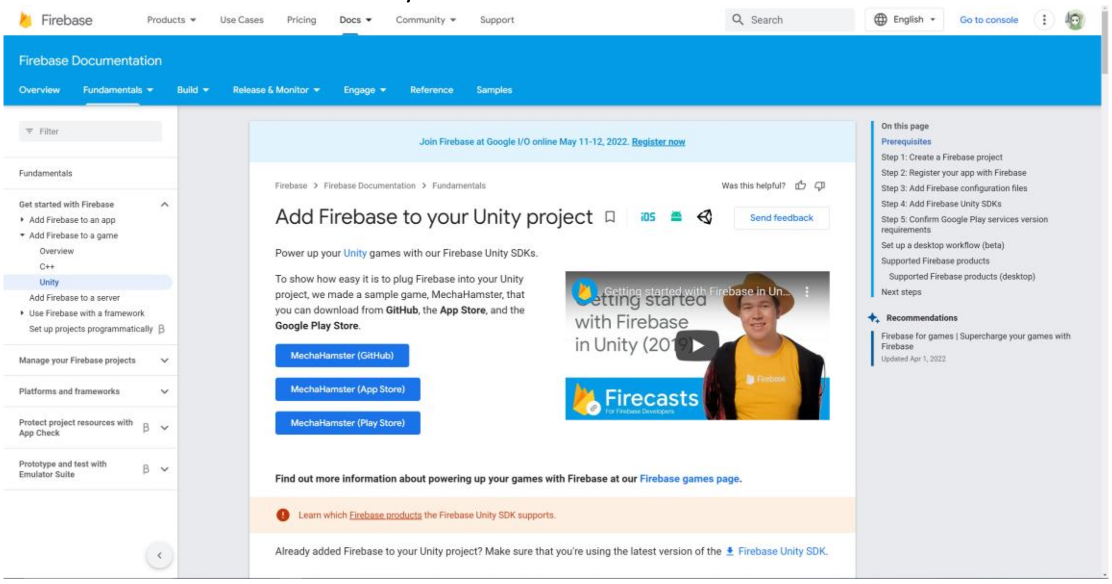
Reflection on Learning Outcome Attainment

Reflect on your experience during your project and the achievements you have relating to <u>at least two</u> of the points below:

(a) Engineering knowledge
(b) Problem Analysis
(c) Investigation
(d) Design/development of Solutions
(e) Modern Tool Usage
(f) The Engineer and Society
(g) Environment and Sustainability
(h) Ethics
(i) Individual and Team Work
(j) Communication
(k) Project Management and Finance
(l) Lifelong Learning

Point 1: Engineering knowledge

1. Firebase authentication with Unity



Join Firebase at Google I/O online May 11-12, 2022. Register now

Firebase > Firebase Documentation > Fundamentals

Was this helpful?

Add Firebase to your Unity project

Send feedback

Power up your Unity games with our Firebase Unity SDKs.

To show how easy it is to plug Firebase into your Unity project, we made a sample game, MechaHamster, that you can download from **GitHub**, the **App Store**, and the **Google Play Store**.

MechaHamster (GitHub)

MechaHamster (App Store)

MechaHamster (Play Store)

Find out more information about powering up your games with Firebase at our Firebase games page.

Learn which Firebase products the Firebase Unity SDK supports.

Already added Firebase to your Unity project? Make sure that you're using the latest version of the Firebase Unity SDK.

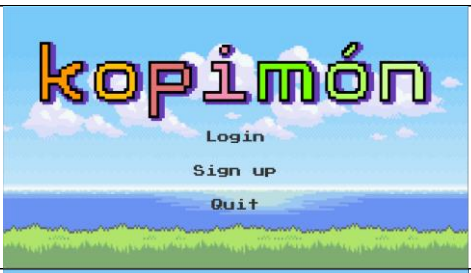
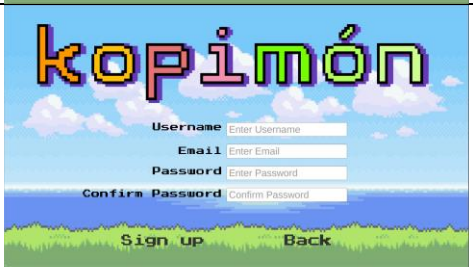
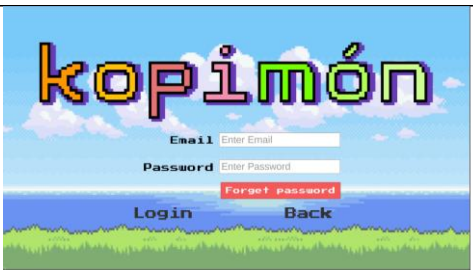
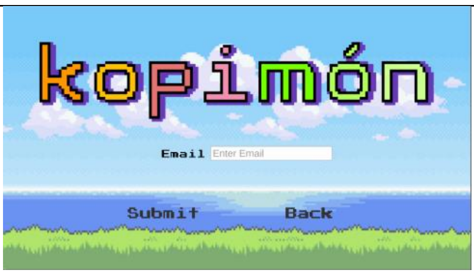
On this page
Prerequisites
Step 1: Create a Firebase project
Step 2: Register your app with Firebase
Step 3: Add Firebase configuration files
Step 4: Add Firebase Unity SDKs
Step 5: Confirm Google Play services version requirements
Set up a desktop workflow (beta)
Supported Firebase products
Supported Firebase products (desktop)
Next steps

Recommendations
Firebase for games | Supercharge your games with Firebase
Updated Apr 1, 2022

Firebase is a commonly used online tool provided by Google for database management. It has various functions like real-time database, user sign up, and user login, which can be integrated into a Unity project. Moreover, many functions of Firebase are free, which is good for budget control.
In our game *Kopimon*, Firebase authentication database is applied for user sign up and login. Unity project's UI receives input from the player, and directly sends player's input (email and password) to Firebase for user sign up or authentication.

a. Unity UI
Unity project's UI receives input from the user. We designed our own UI interface to fit it into our game's pixelated theme.

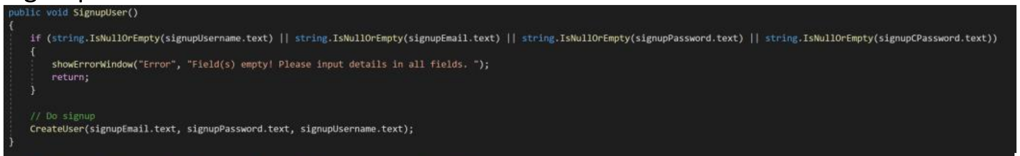
Name	Image	Explanation

Start UI		This is the start UI of Kopimon. Player sees this page every time when opening Kopimon.
Sign-up UI	 kopimón Username Enter Username Email Enter Email Password Enter Password Confirm Password Confirm Password Sign up Back	Upon clicking 'Sign up' in start scene, player sees this sign-up scene that requires username, email, and password. This information will be sent to Firebase to create a user account.
Login UI	 kopimón Email Enter Email Password Enter Password Forget password Login Back	Player with an account can click on 'Login' in start scene and then sees this login scene. Upon entering email and password, Firebase will verify the input and let player enter the game if the authentication process is successful.
Reset password scene		If player clicks 'Forget password' in login scene, he/she will see this scene which requires player's email for resetting password. After entering email, player will be directed back to start scene. Player will also soon receive an email for password resetting.

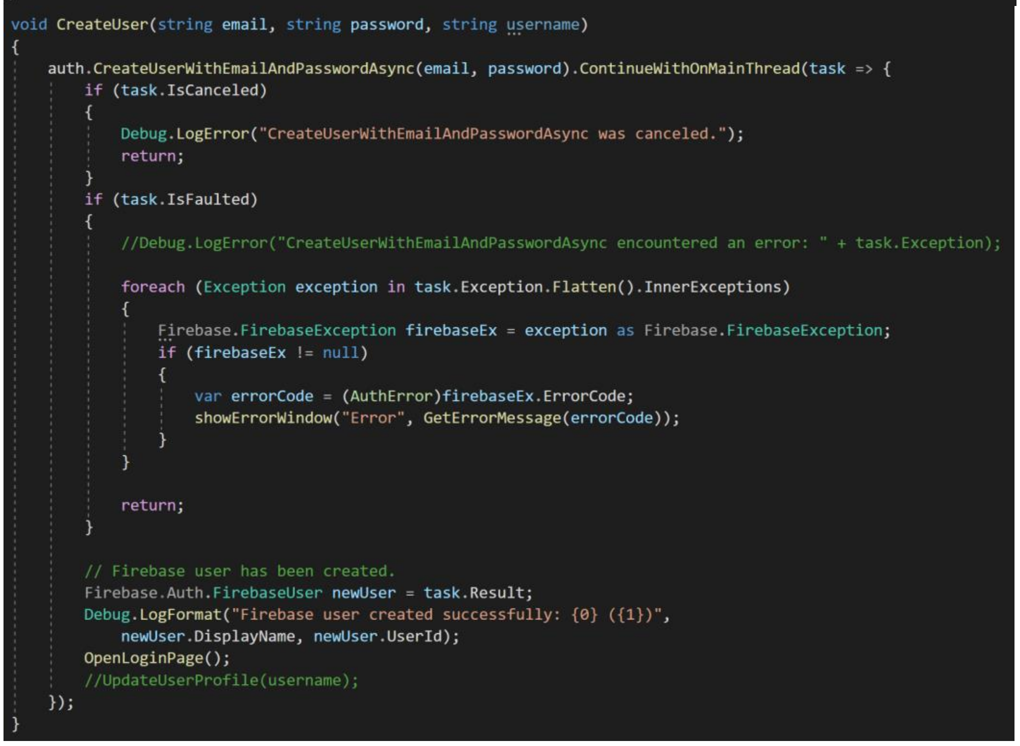
b. Firebase integration
To integrate Firebase into Unity, I firstly follow the official video for initial setup: https://www.youtube.com/watch?v=A6du3DUTIPI.

Based on our needs, I then modified codes from Firebase website for realizing functions below.

Sign-up:

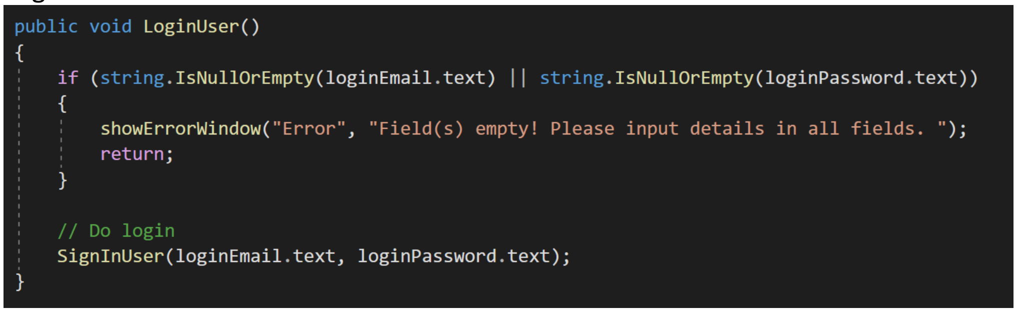


```
public void SignupUser()
{
    if (string.IsNullOrEmpty(signupUsername.text) || string.IsNullOrEmpty(signupEmail.text) || string.IsNullOrEmpty(signupPassword.text) || string.IsNullOrEmpty(signupCPassword.text))
    {
        showErrorWindow("Error", "Field(s) empty! Please input details in all fields. ");
        return;
    }

    // Do signup
    CreateUser(signupEmail.text, signupPassword.text, signupUsername.text);
}
```

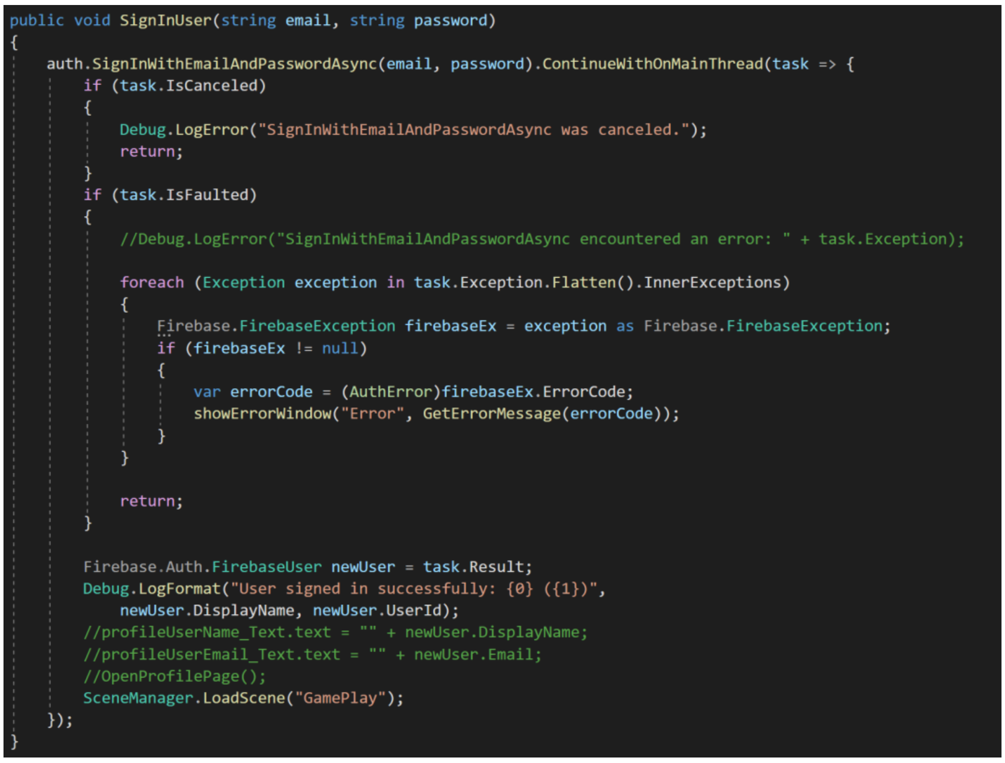


```
void CreateUser(string email, string password, string username)
{
    auth.CreateUserWithEmailAndPasswordAsync(email, password).ContinueWithOnMainThread(task => {
        if (task.IsCanceled)
        {
            Debug.LogError("CreateUserWithEmailAndPasswordAsync was canceled.");
            return;
        }
        if (task.IsFaulted)
        {
            //Debug.LogError("CreateUserWithEmailAndPasswordAsync encountered an error: " + task.Exception);

            foreach (Exception exception in task.Exception.Flatten().InnerExceptions)
            {
                Firebase.FirebaseException firebaseEx = exception as Firebase.FirebaseException;
                if (firebaseEx != null)
                {
                    var errorCode = (AuthError)firebaseEx.ErrorCode;
                    showErrorWindow("Error", GetErrorMessage(errorCode));
                }
            }

            return;
        }

        // Firebase user has been created.
        Firebase.Auth.FirebaseUser newUser = task.Result;
        Debug.LogFormat("Firebase user created successfully: {0} ({1})",
            newUser.DisplayName, newUser.UserId);
        OpenLoginPage();
        //UpdateUserProfile(username);
    });
}
```

Login:

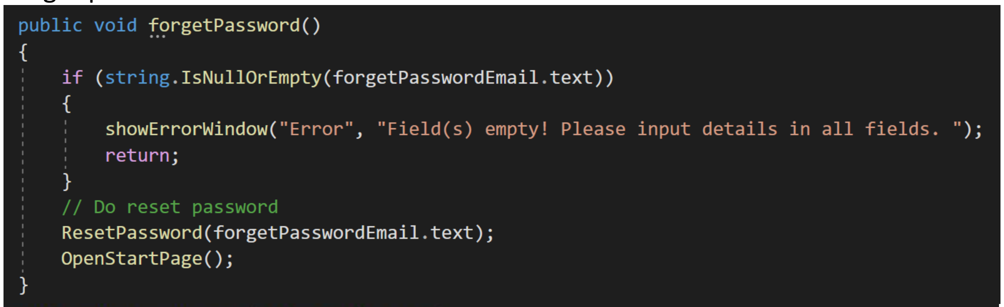


```
public void LoginUser()
{
    if (string.IsNullOrEmpty(loginEmail.text) || string.IsNullOrEmpty(loginPassword.text))
    {
        showErrorWindow("Error", "Field(s) empty! Please input details in all fields. ");
        return;
    }

    // Do login
    SignInUser(loginEmail.text, loginPassword.text);
}
```

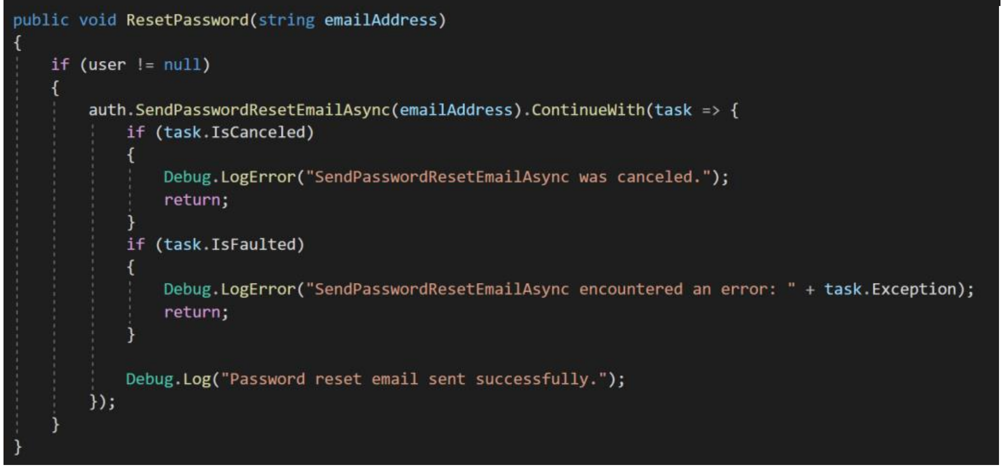


```
public void SignInUser(string email, string password)
{
    auth.SignInWithEmailAndPasswordAsync(email, password).ContinueWithOnMainThread(task => {
        if (task.IsCanceled)
        {
            Debug.LogError("SignInWithEmailAndPasswordAsync was canceled.");
            return;
        }
        if (task.IsFaulted)
        {
            //Debug.LogError("SignInWithEmailAndPasswordAsync encountered an error: " + task.Exception);

            foreach (Exception exception in task.Exception.Flatten().InnerExceptions)
            {
                Firebase.FirebaseException firebaseEx = exception as Firebase.FirebaseException;
                if (firebaseEx != null)
                {
                    var errorCode = (AuthError)firebaseEx.ErrorCode;
                    showErrorWindow("Error", GetErrorMessage(errorCode));
                }
            }

            return;
        }

        Firebase.Auth.FirebaseUser newUser = task.Result;
        Debug.LogFormat("User signed in successfully: {0} ({1})",
            newUser.DisplayName, newUser.UserId);
        //profileUserName_Text.text = "" + newUser.DisplayName;
        //profileUserEmail_Text.text = "" + newUser.Email;
        //OpenProfilePage();
        SceneManager.LoadScene("GamePlay");
    });
}
```

Forget password:



```
public void forgetPassword()
{
    if (string.IsNullOrEmpty(forgetPasswordEmail.text))
    {
        showErrorWindow("Error", "Field(s) empty! Please input details in all fields. ");
        return;
    }
    // Do reset password
    ResetPassword(forgetPasswordEmail.text);
    OpenStartPage();
}
```

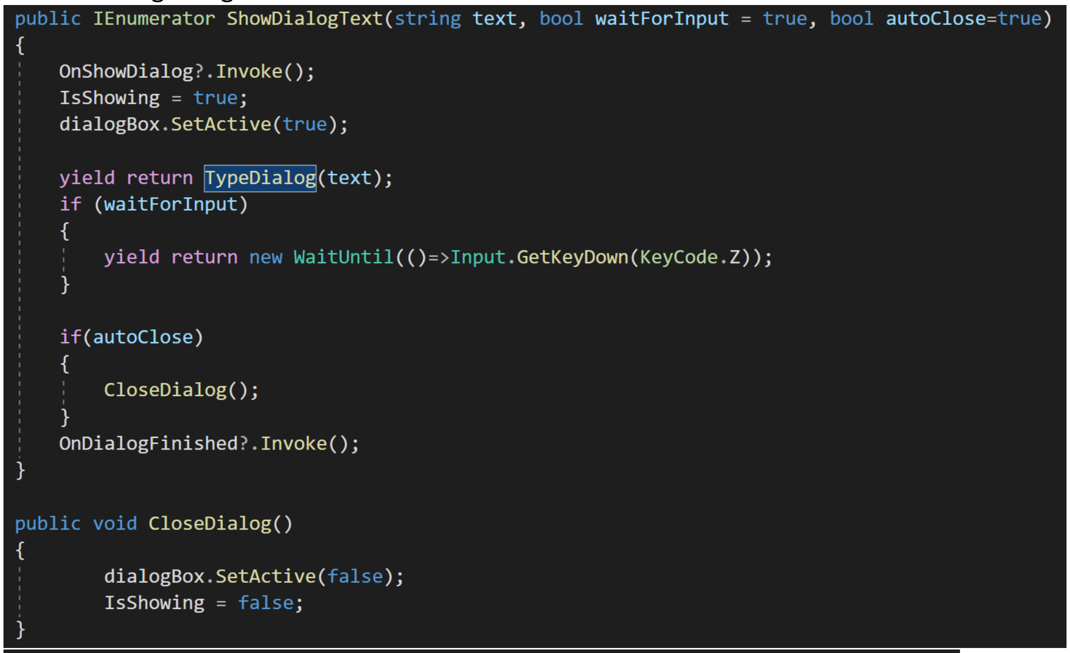


```
public void ResetPassword(string emailAddress)
{
    if (user != null)
    {
        auth.SendPasswordResetEmailAsync(emailAddress).ContinueWith(task => {
            if (task.IsCanceled)
            {
                Debug.LogError("SendPasswordResetEmailAsync was canceled.");
                return;
            }
            if (task.IsFaulted)
            {
                Debug.LogError("SendPasswordResetEmailAsync encountered an error: " + task.Exception);
                return;
            }

            Debug.Log("Password reset email sent successfully.");
        });
    }
}
```

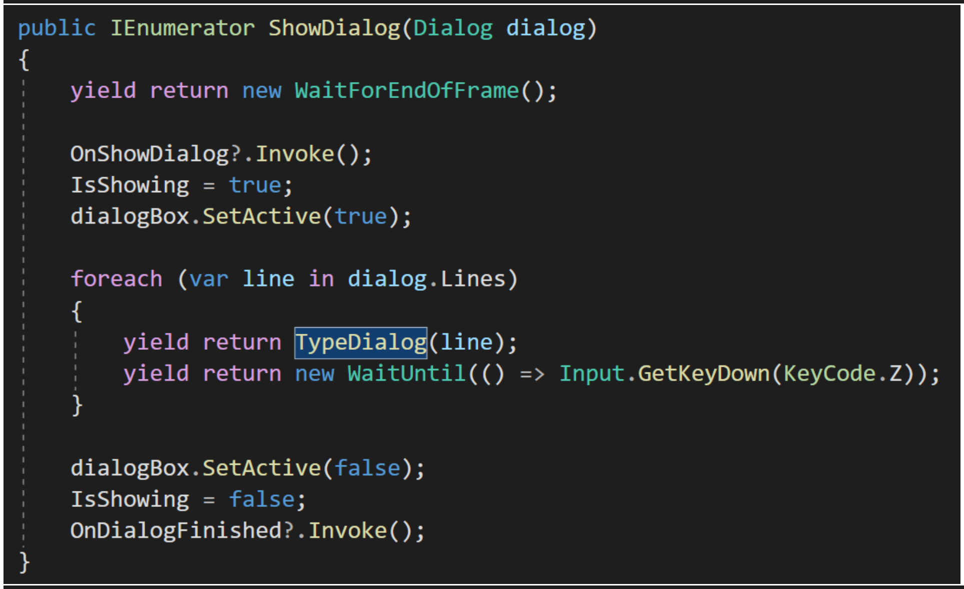
Overall, although I've faced some difficulties when connecting Firebase to Unity, the function implementation afterwards went well.

2. Unity game development
   Unity is a powerful tool for game development. When developing Kopimon, I learned various common but useful coding technics for developing a 2D game.

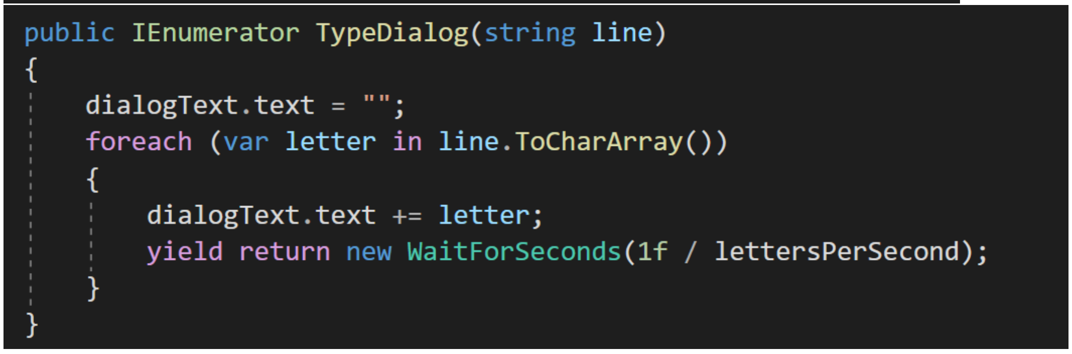
a. Coded NPC dialog system triggered by keyboard pressing
NPCs (Non-Player Characters) are an important element in a quest game because they give player information about the quests to complete. Therefore, NPCs are usually interactable: when player approaches and presses a preset key (the key is ‘z’ in Kopimon), an NPC speaks to player via a dialog box.
Below are screenshots of the codes I wrote.
Inside `DialogManager.cs`:



```
public IEnumerator ShowDialogText(string text, bool waitForInput = true, bool autoClose=true)
{
    OnShowDialog?.Invoke();
    IsShowing = true;
    dialogBox.SetActive(true);

    yield return TypeDialog(text);
    if (waitForInput)
    {
        yield return new WaitUntil(()=>Input.GetKeyDown(KeyCode.Z));
    }

    if(autoClose)
    {
        CloseDialog();
    }
    OnDialogFinished?.Invoke();
}

public void CloseDialog()
{
        dialogBox.SetActive(false);
        IsShowing = false;
}
```

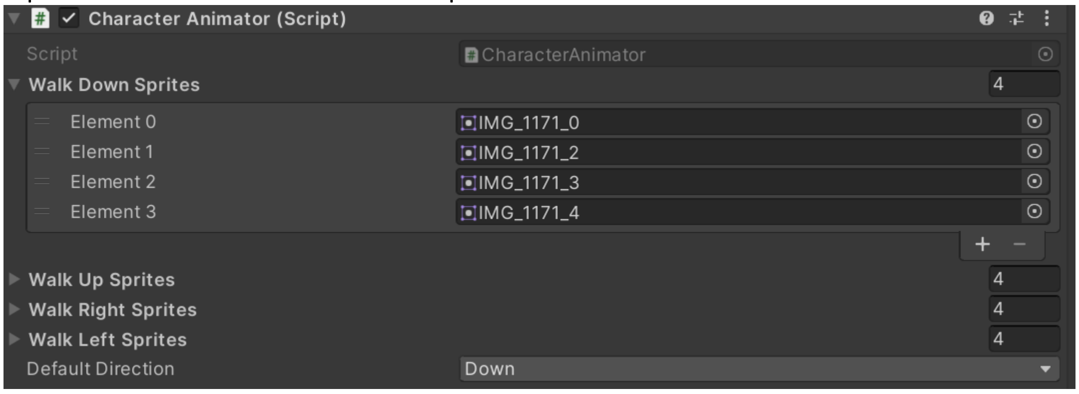


```
public IEnumerator ShowDialog(Dialog dialog)
{
    yield return new WaitForEndOfFrame();

    OnShowDialog?.Invoke();
    IsShowing = true;
    dialogBox.SetActive(true);

    foreach (var line in dialog.Lines)
    {
        yield return TypeDialog(line);
        yield return new WaitUntil(() => Input.GetKeyDown(KeyCode.Z));
    }

    dialogBox.SetActive(false);
    IsShowing = false;
    OnDialogFinished?.Invoke();
}
```



```
public IEnumerator TypeDialog(string line)
{
    dialogText.text = "";
    foreach (var letter in line.ToCharArray())
    {
        dialogText.text += letter;
        yield return new WaitForSeconds(1f / lettersPerSecond);
    }
}
```

b. Coded custom animation system for more convenient control over multiple characters
Animations of 2D characters are usually done by creating animation sequences of 4 directions and connect them using a blend tree. However, this method is time consuming if there are many characters, each needs its own set of animations and blend tree.
Therefore, NPCs' animation in our game is controlled by script `CharacterAnimator.cs` and `SpriteAnimator.cs`.
a) `CharacterAnimator.cs`
Input to `CharacterAnimator.cs` are sprites for animations and default direction:



Character Animator (Script)

Script	CharacterAnimator
Walk Down Sprites	4
Element 0	IMG_1171_0
Element 1	IMG_1171_2
Element 2	IMG_1171_3
Element 3	IMG_1171_4
Walk Up Sprites	4
Walk Right Sprites	4
Walk Left Sprites	4
Default Direction	Down

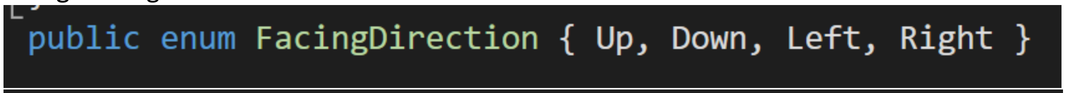
Initialization:

```
private void Start()
{
    spriteRenderer = GetComponent<SpriteRenderer>();
    walkDownAnim = new SpriteAnimator(walkDownSprites, spriteRenderer);
    walkUpAnim = new SpriteAnimator(walkUpSprites, spriteRenderer);
    walkRightAnim = new SpriteAnimator(walkRightSprites, spriteRenderer);
    walkLeftAnim = new SpriteAnimator(walkLeftSprites, spriteRenderer);
    SetFacingDirection(defaultDirection);

    currentAnim = walkDownAnim;
}
```

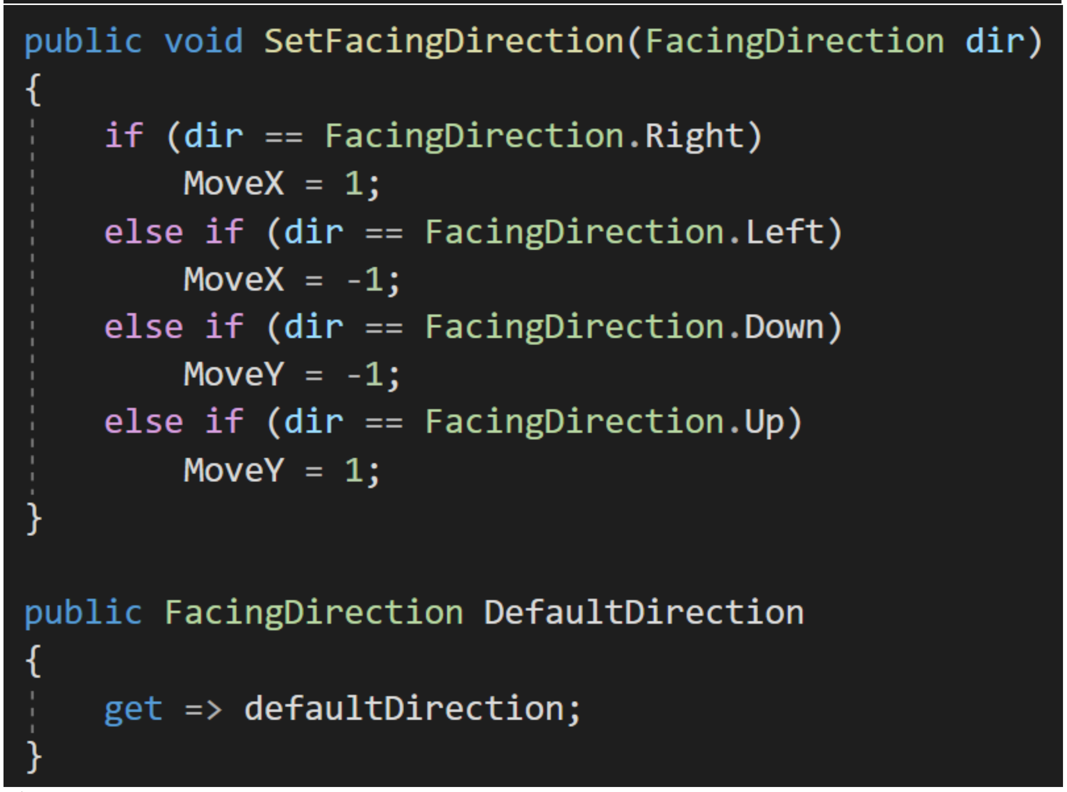
Handling updates of animation:

```
private void Update()
{
    var prevAnim = currentAnim;

    if (MoveX == 1)
        currentAnim = walkRightAnim;
    else if (MoveX == -1)
        currentAnim = walkLeftAnim;
    else if (MoveY == 1)
        currentAnim = walkUpAnim;
    else if (MoveY == -1)
        currentAnim = walkDownAnim;

    if (currentAnim != prevAnim || IsMoving != wasPreviouslyMoving)
        currentAnim.Start();

    if (IsMoving)
        currentAnim.HandleUpdate();
    else
        spriteRenderer.sprite = currentAnim.Frames[0];
    wasPreviouslyMoving = IsMoving;
}
```

Using `FacingDirection` to control `MoveX` and `MoveY`:



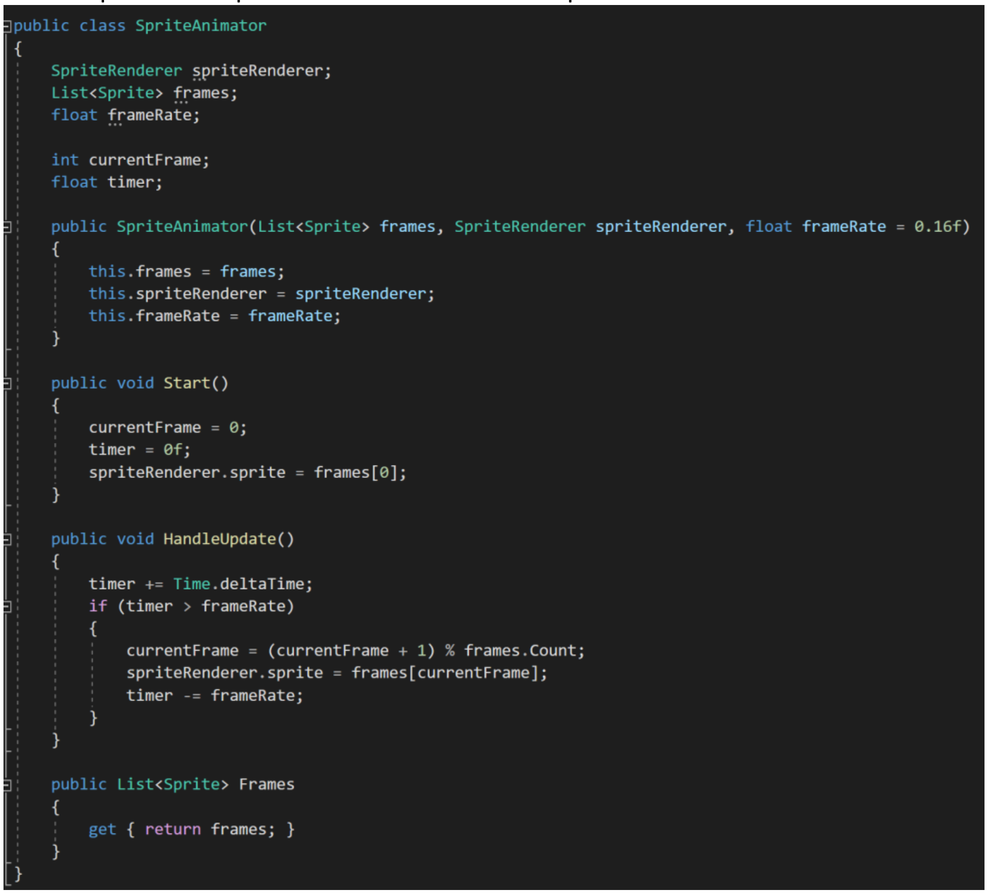
```
public enum FacingDirection { Up, Down, Left, Right }
```



```
public void SetFacingDirection(FacingDirection dir)
{
    if (dir == FacingDirection.Right)
        MoveX = 1;
    else if (dir == FacingDirection.Left)
        MoveX = -1;
    else if (dir == FacingDirection.Down)
        MoveY = -1;
    else if (dir == FacingDirection.Up)
        MoveY = 1;
}

public FacingDirection DefaultDirection
{
    get => defaultDirection;
}
```

b) `SpriteAnimator.cs`

This script controls parameters like animation speed of a character.

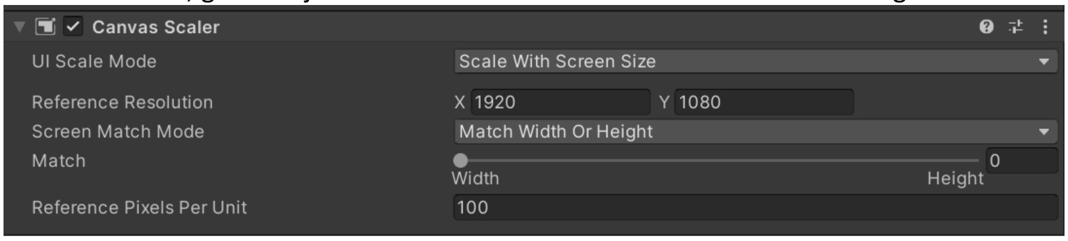


```
public class SpriteAnimator
{
    SpriteRenderer spriteRenderer;
    List<Sprite> frames;
    float frameRate;

    int currentFrame;
    float timer;

    public SpriteAnimator(List<Sprite> frames, SpriteRenderer spriteRenderer, float frameRate = 0.16f)
    {
        this.frames = frames;
        this.spriteRenderer = spriteRenderer;
        this.frameRate = frameRate;
    }

    public void Start()
    {
        currentFrame = 0;
        timer = 0f;
        spriteRenderer.sprite = frames[0];
    }

    public void HandleUpdate()
    {
        timer += Time.deltaTime;
        if (timer > frameRate)
        {
            currentFrame = (currentFrame + 1) % frames.Count;
            spriteRenderer.sprite = frames[currentFrame];
            timer -= frameRate;
        }
    }

    public List<Sprite> Frames
    {
        get { return frames; }
    }
}
```

c. UI element organization and UI visibility control

When playing a game on different screens, it is expected that UI elements are in their correct positions. For example, start scene should always have the game title at the top center of any screen.

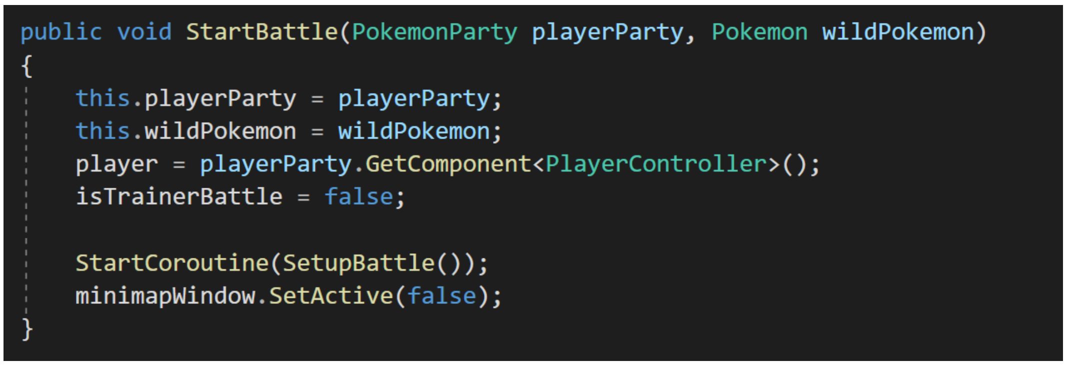
To achieve that, game object 'Canvas' in each scene should have the settings as below:



Canvas Scaler

UI Scale Mode	Scale With Screen Size
Reference Resolution	X 1920 Y 1080
Screen Match Mode	Match Width Or Height
Match	0
Reference Pixels Per Unit	100

UI visibility can be controlled by scripts. For instance, I controlled visibility of minimap when a battle starts:



```
public void StartBattle(PokemonParty playerParty, Pokemon wildPokemon)
{
    this.playerParty = playerParty;
    this.wildPokemon = wildPokemon;
    player = playerParty.GetComponent<PlayerController>();
    isTrainerBattle = false;

    StartCoroutine(SetupBattle());
    minimapWindow.SetActive(false);
}
```

Inside script `BattleSystem.cs`, minimap is set invisible via the line "`minimapWindow.SetActive(false);`".

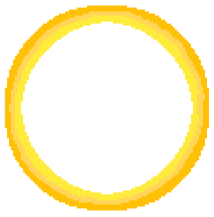
d. Coded minimap showing live location of characters (player, NPCs, gaming opponents, and other players)

Minimap is a favorable function in many quest games. It provides player a cleaner, wider view of his/her surroundings.

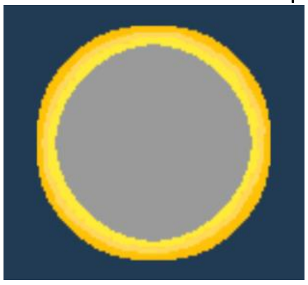
Following steps are followed when coding a minimap:

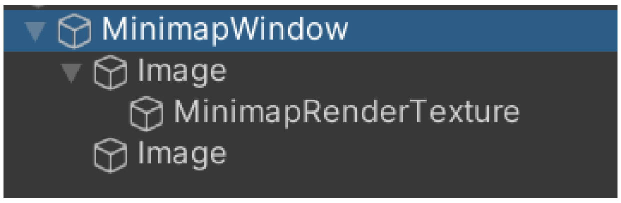
a) Set up minimap UI

To create a minimap UI, I firstly created a pixelated ring sprite as its frame.



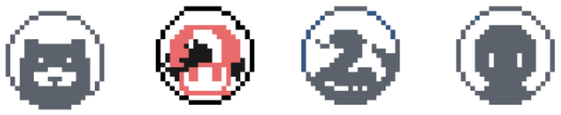
Then I formed a minimap game object that requires input of texture as its view.



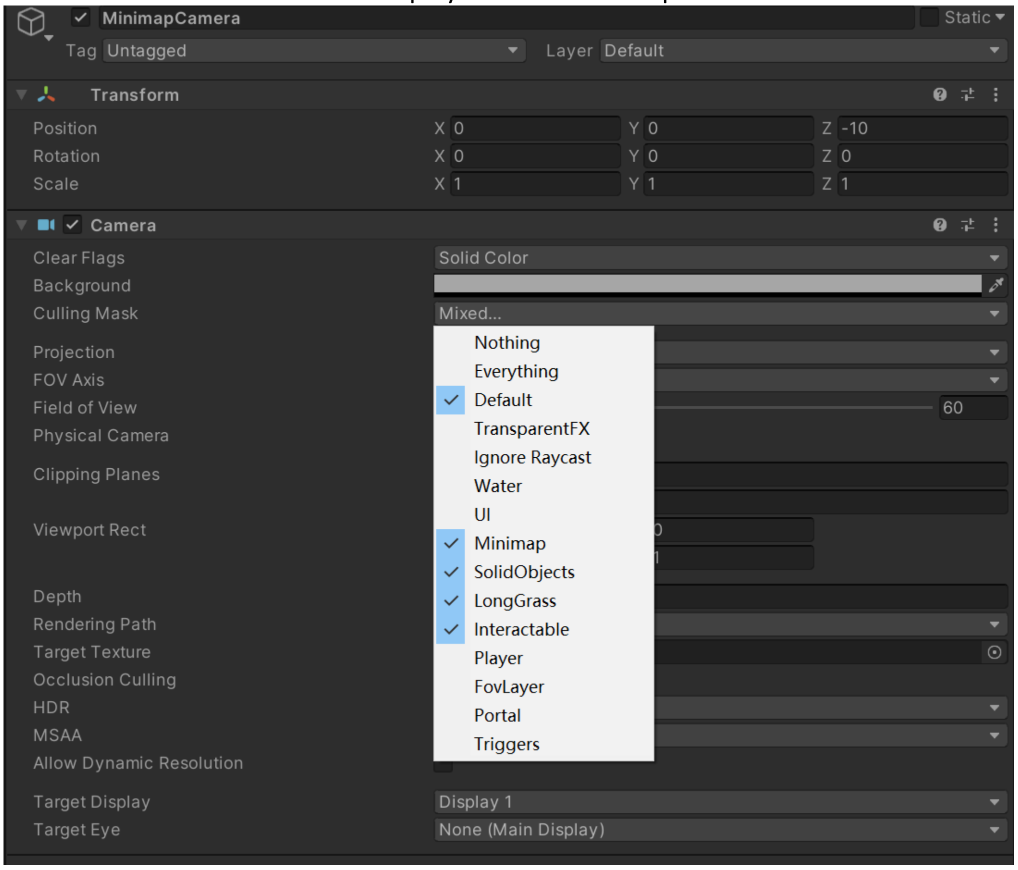


b) Get minimap view from a minimap camera
To produce the texture needed, another camera is attached to the player as a child of the main camera so that minimap view and main game view both follow the player.

c) Replace NPCs and player's character with icons
To make minimap view cleaner, NPCs and player are replaced by their respective minimap icons (created by Charm).

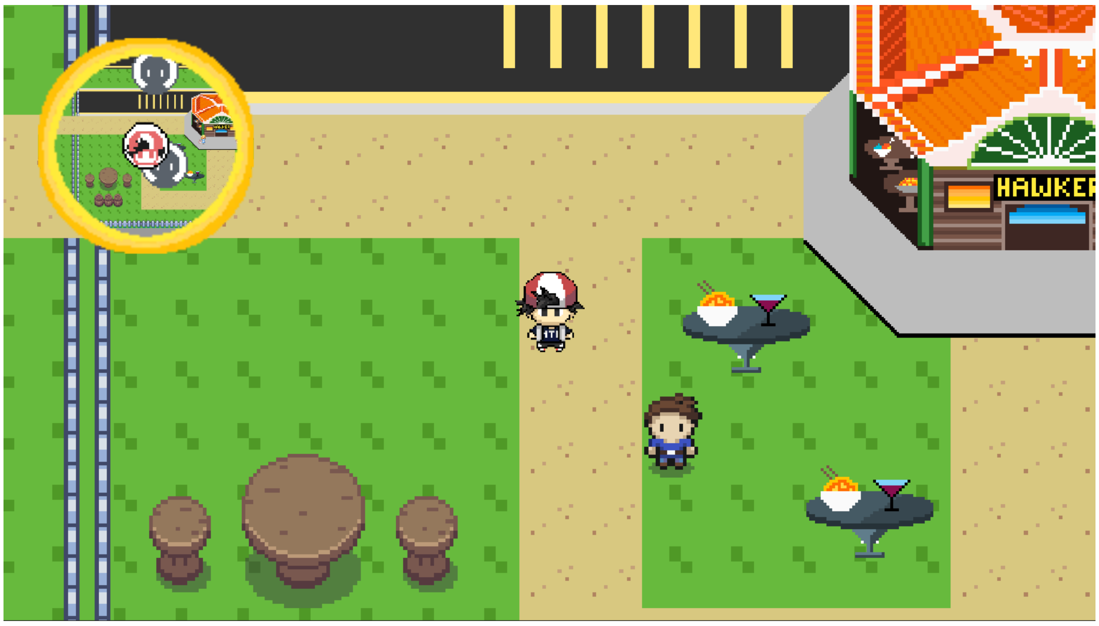


d) Change render settings of the minimap camera
Minimap camera should render all the background and minimap icons but not the player layer, where real character is in. This is to make sure only minimap icons rather than real characters are displayed inside minimap.



```
✓ MinimapCamera                                              Static
Tag Untagged                         Layer Default

Transform
Position          X 0        Y 0        Z -10
Rotation          X 0        Y 0        Z 0
Scale             X 1        Y 1        Z 1

✓ Camera
Clear Flags       Solid Color
Background
Culling Mask      Mixed...
Projection          Nothing
FOV Axis            Everything
Field of View     ✓ Default                                  60
Physical Camera     TransparentFX
Clipping Planes     Ignore Raycast
                    Water
Viewport Rect       UI
                  ✓ Minimap
Depth             ✓ SolidObjects
Rendering Path    ✓ LongGrass
Target Texture    ✓ Interactable
Occlusion Culling   Player
HDR                 FovLayer
MSAA                Portal
Allow Dynamic Resolution  Triggers
Target Display    Display 1
Target Eye        None (Main Display)
```

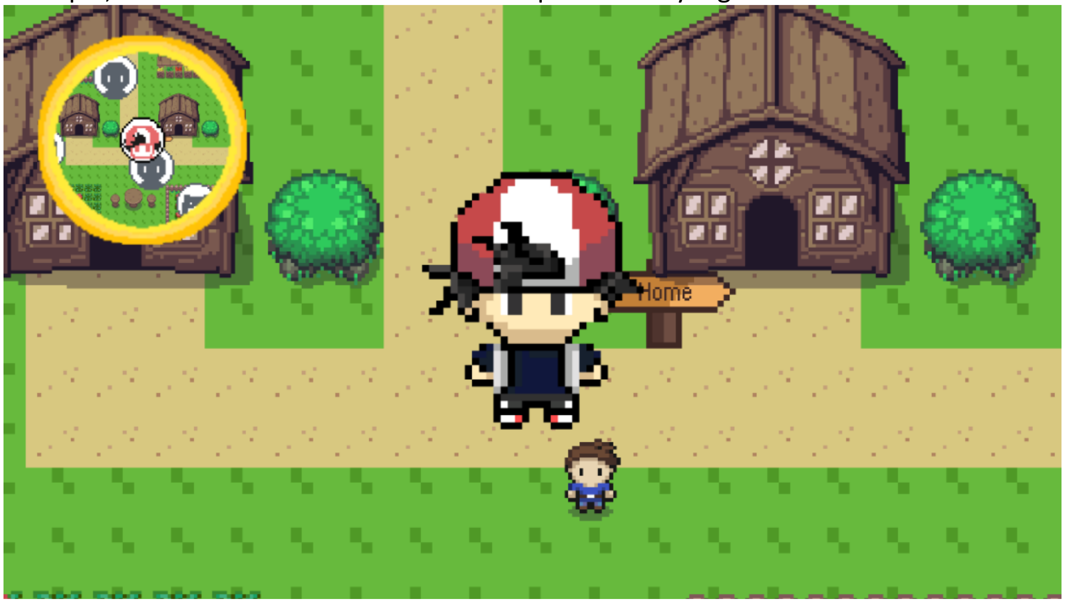
Final minimap looks like below inside the game:



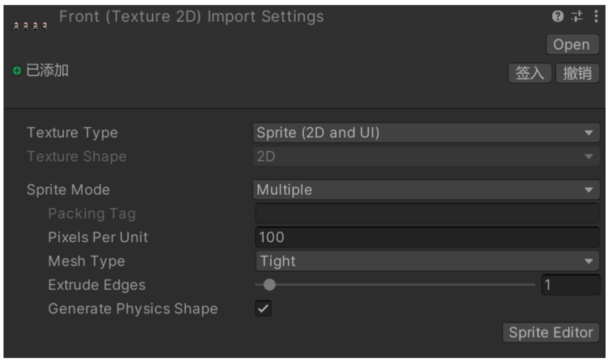
e. Completed scene transition issue debugging
The whole game consists of multiple scenes. While scene additive is used for Gameplay scene, transitions like from Start scene to Gameplay scene still requires scene transition. This is done by a simple line of code inside `StartController.cs`, the last line in the function `SignInUser(string email, string password)`:

```
SceneManager.LoadScene("GamePlay");
```

f. Handling sprites size
When converting PNG files to sprites, there might be issue with the sprite size. For example, the default size of main character sprites is very big:



However, there's no need to redraw sprites again in Pixlrart. Unity has the function of changing sprite size:


Front (Texture 2D) Import Settings

Open

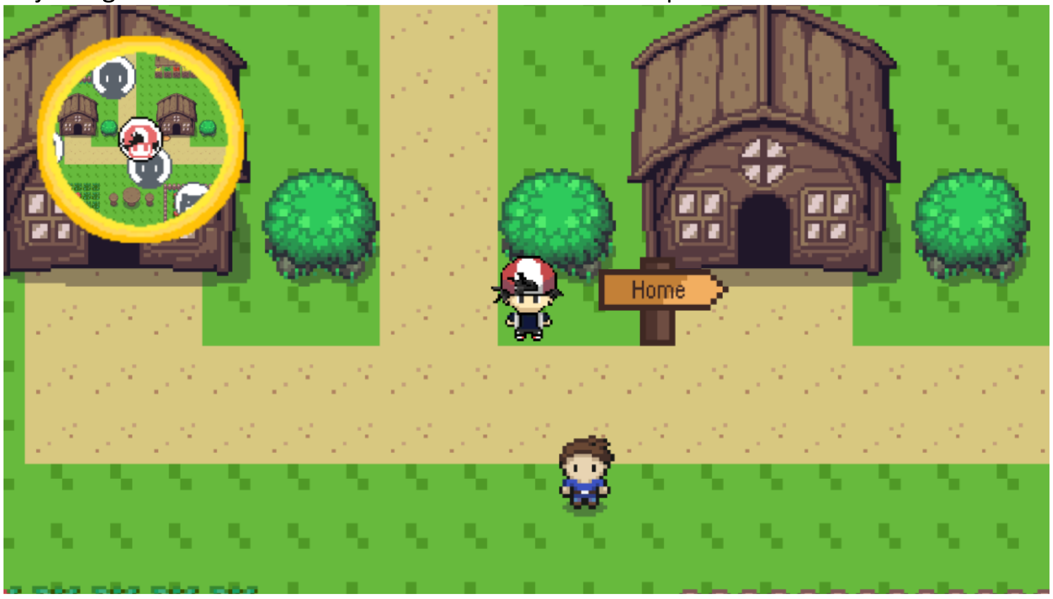
已添加

签入 撤销

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Multiple
Packing Tag:
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Generate Physics Shape: ✓

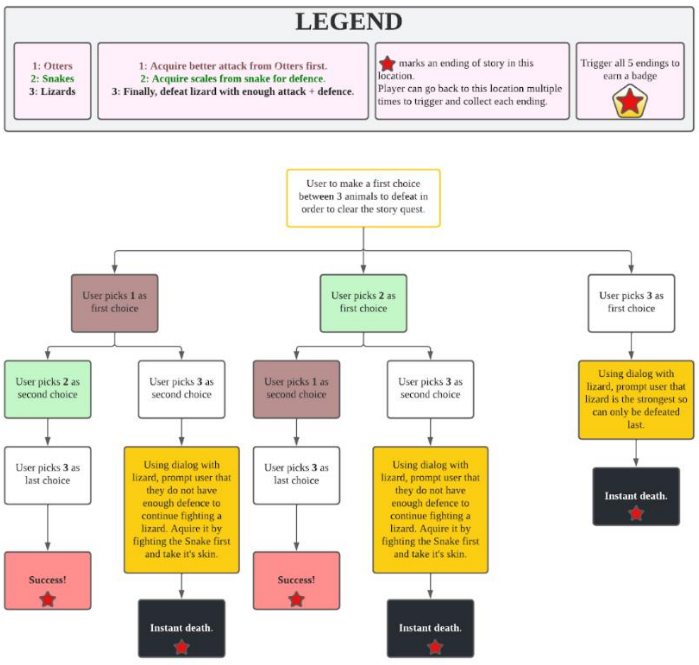
Sprite Editor

Adjusting Pixels Per Unit from 100 to 300 will solve this problem.



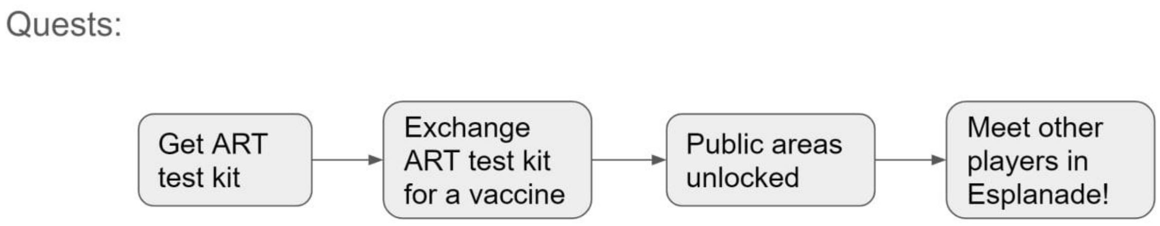
Point 2: Design/development of Solutions

1. Game mechanism, game reward system, quest, and storyline ideation
   One of the most important things about a game is being interesting to play. Upon finishing version 1 of our game, which follows the form of a Pokémon game, our group decided to localize the game and make it more interesting for players.
   After discussions with Swapnil, we worked out a flowchart of our game story as below using Lucidchart.


LEGEND
1: Otters
2: Snakes
3: Lizards
1: Acquire better attack from Otters first.
2: Acquire scales from snake for defence.
3: Finally, defeat lizard with enough attack + defence.
marks an ending of story in this location.
Player can go back to this location multiple times to trigger and collect each ending.
Trigger all 5 endings to earn a badge
User to make a first choice between 3 animals to defeat in order to clear the story quest.
User picks 1 as first choice
User picks 2 as first choice
User picks 3 as first choice
User picks 2 as second choice
User picks 3 as second choice
User picks 1 as second choice
User picks 3 as second choice
Using dialog with lizard, prompt user that lizard is the strongest so can only be defeated last.
User picks 3 as last choice
Using dialog with lizard, prompt user that they do not have enough defence to continue fighting a lizard. Aquire it by fighting the Snake first and take it's skin.
User picks 3 as last choice
Using dialog with lizard, prompt user that they do not have enough defence to continue fighting a lizard. Aquire it by fighting the Snake first and take it's skin.
Instant death.
Success!
Instant death.
Success!
Instant death.

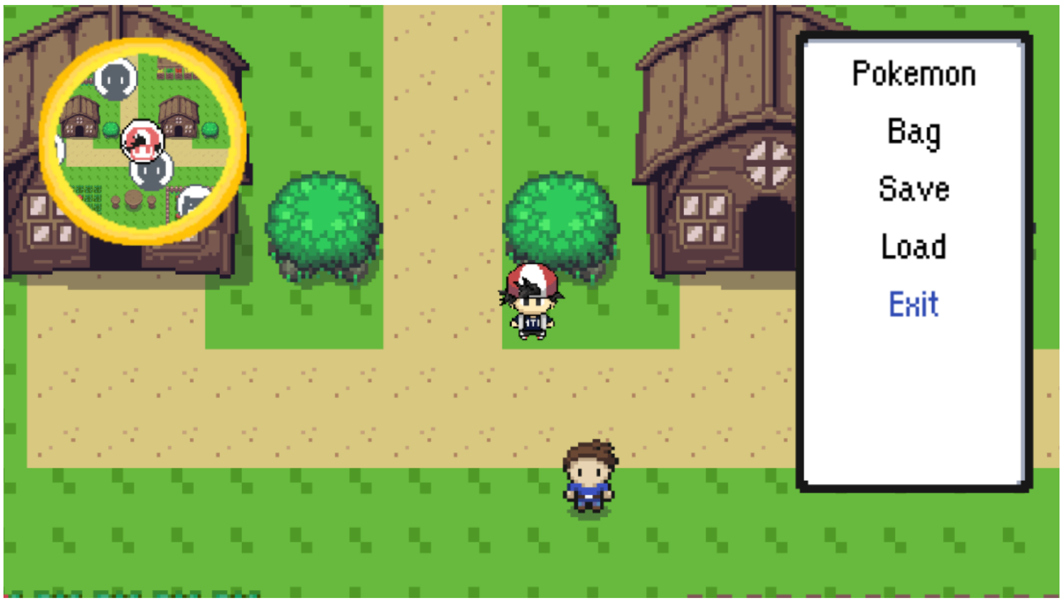
a. Localization
   Firstly, the game story has Singapore as its background, so scenes like Marina Bay Sands and hawker center are added to the game. Second, since there are many wild animals like otters, snakes, and lizards in Singapore, they are planned to be added in as special NPCs that player must battle with.
b. Interactive storytelling
   Aside from localization, we also wanted to implement a new technic called 'interactive storytelling'. This way of storytelling gives player certain level of freedom to make choices while playing, and the outcomes differ based on different choices made.
c. Reward system
   There is also a game reward system. In our game, the choice is about which animal to fight first. While player can only fight lizard successfully after defeating otters and snakes, player is encouraged to explore various set of choices to get different battle outcomes. The badge will only be obtained upon collection of all the 5 outcomes. This reward system aims at encouraging player to replay a story to find more possibilities in our game, which makes our game more interesting.

Although this flowchart is not implemented exactly as it is due to existing structure of our version 1 game, it is still an inspiring exploration of game storytelling. Upon discussion with Sean and Ruokun, the quests are implemented as below:


Quests:
Get ART test kit
Exchange ART test kit for a vaccine
Public areas unlocked
Meet other players in Esplanade!

2. User-friendly game functions
   As a product, a game should also be user-friendly. I've implemented the following functions to achieve the goal.
   a. 'Exit' button

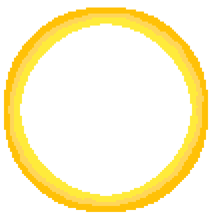
Exit button is a must for a game. Although one can exit via pressing the key 'esc', it is not the convention in a game context. Therefore, I modified the menu and added in the 'Exit' button.



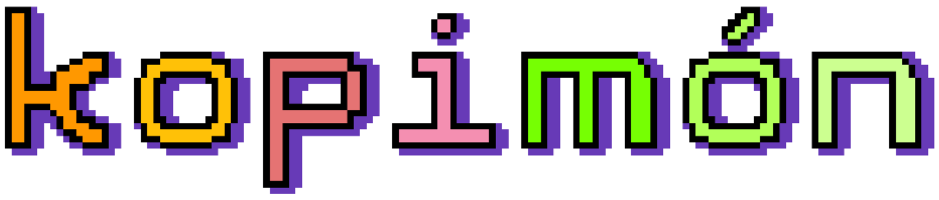
b. Road signs
   Since there are many scenes in our game, road signs help players find their way around. Please see Point 3 below for more details.

Point 3: Modern Tool Usage

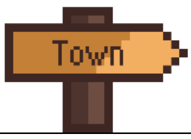
1. Unity
   - main tool for the game development
   Our game is developed using Unity version 2020.3.26f1.
2. Unity Hub
   - project management
   Instead of finding Unity files directly from file explorer, Unity Hub provides a more convenient solution by finding and listing all the local Unity projects. It also allows users to open Unity remote projects, i.e., downloading projects in cloud to local folders.
3. Unity Collaborate and Plastic SCM
   - project version control
   Unity Collaborate is a tool that keeps Unity projects in the Unity cloud. For free version, at most 3 members in the same organization can simultaneously have a seat. Members with a seat can push their changes to the cloud and pull other members' published changes. During April 2022, Unity is updating all unity projects from Unity Collaborate to Plastic SCM, a more convenient tool for version control. Aside from pushing and pulling changes, users can search a version by comment or date published, and investigating changes made in this version.
4. Pixlrart
   - game assets creation
   Pixlrart is a commonly used free online tool for creating pixelated game assets, which is useful for our pixelated style game development. I created assets listed below:
   a. Minimap frame



b. Game title



5. Photoshop
   To help players find their way around, I suggested to put road signs in game. After Swapnil created the plain road sign sprites, I tried several ways and find Photoshop the most convenient way to put text on road sign sprites.

	Hawker Center		Neighborhood
Garden by the Bay	Garden by the Bay		

   The reason for separating assets creation and text adding is that we want the text font to be ‘Orange Kid’, the same font as NPC dialogs. If add text by drawing in Pixlrart, it would be time-consuming and hard to recreate the font exactly.
6. Firebase
   - user account management
   Firebase is a well-developed tool for user account management. Please read further in Point 1 of this report about our Firebase implementation.
7. Lucidchart
   - game story, mechanics, and reward system ideation
   Lucidchart is a free online mind mapping tool. It is used for creating the flowchart shown in Point 2.

Point 4: The Engineer and Society

1. Gaming industry insight
   Gaming industry is one of the few industries that has experienced growth despite the COVID-19 pandemic (Ángeles et al.,2020). Although it is young among various industries, it grows fast along with technology advancement.
   Video games used to be considered simply as entertainment. However, more and more researchers start to focus on their use in other sectors like healthcare. For example, video games can act as a handy tool for chronic disease management and curing brain trauma

(Ceranoglu, 2010), and those without competitiveness and violence help parents interact with children(MacNeil & Hembree-Kigin, 2011). Therefore, gaming industry is with great potential. With my strong interest in video game development, I want to accumulate more experience in this area and enter the gaming industry in the future.

2. Target group
   The target group of Kopimon is the Singapore gaming community with an interest in quest games. Kopimon has many local elements like local wild animals and famous landmarks in Singapore. Moreover, both single-player and multiplayer modes are available for players to explore. Especially, the multiplayer function is expected to be popular among players during the COVID-19 pandemic.

Point 5: Individual and Team Work

1. Individual work
   During DIP development, I've worked independently on various tasks:
   a) Art assets creation and application:
   - Minimap frame creation
   - Game title design
   - Converted art assets from PNG format to Unity sprites ready to use
   - Road signs creation

   b) Frontend:
   - Coded NPC dialog system triggered by keyboard pressing
   - Coded custom animation system for more convenient control over multiple characters
   - UI element organization
   - Minimap creation
   - Game map double check

2. Team work
   I've also got the chance to cooperate with several group members.
   a) Firebase implementation and Start scene UI design (collaborator: Sean)
   b) Art assets compatibility check (collaborators: Charm, Swapnil, and Shin Hui)
   c) Game mechanism, game reward system, quest, and storyline ideation (collaborators: Swapnil, Sean, and Ruokun)
   d) Minimap visibility control (collaborator: Yu Yue)
   e)

Point 6: Communication

There are mainly 4 platforms for our group's communication:

1. Telegram
   Telegram is the main communication channel for project issue discussions.
2. Discord
   Discord is a widely used app for gaming community. Our group uses Discord for conducting online meetings.
3. Google cloud
   Google cloud is used by us for storing weekly presentation slides, video links, and art assets.
4. Unity Collaborate and Plastic SCM
   These two platforms are used for convenient update of game projects. They both have commenting function, which makes it easy for us to trace changes and manage project versions.

Conclusion

Upon finishing Kopimon, I've learned a lot about developing 2D games using Unity. Aside from skills like art asset creation, backend engineering, and frontend engineering, I also gained experience in cooperating with people working on different kinds of tasks.

Overall, DIP is a fruitful learning experience, which gave me a deeper insights into the game industry and game production.

References

Ceranoglu, T. A. (2010). Star wars in psychotherapy: Video games in the office. *Academic Psychiatry*, *34*(3), 233–236. https://doi.org/10.1176/appi.ap.34.3.233

López-Cabarcos, M. Á., Ribeiro-Soriano, D., & Piñeiro-Chousa, J. (2020). All that glitters is not gold. the rise of gaming in the covid-19 pandemic. *Journal of Innovation & Knowledge*, *5*(4), 289–296. https://doi.org/10.1016/j.jik.2020.10.004

MacNeil, C. B., & Hembree-Kigin, T. L. (2011). *Parent-child interaction therapy*. Springer.