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

Name: _______Lim Sean Jet___________

Group No: _________2_______________

Project Title: ____Kopimon____________

## **Contributions to the Project** (1 page)

- Added **Start Up** page which includes Sign-Up & Login subpages
- Integration of **Firebase** with Unity as backend for **Player Authentication**
- Improved **Battle System Architecture** by programming **Status Condition** changes (Burn, Freeze, Poison, Sleep, Paralyze, Confusion)
    - Used scripts to add probability of a Pokémon move's ability to inflict status condition and damage to an enemy unit
    - Added secondary effects to status condition, for example a unit with 'Burn' status condition will receive additional damage for 3 turns
- Improved **Stats Boosting Algorithm** (speed, attack etc.)
    - Reset stat boost at the end of a battle
    - Using speed stat to determine which Pokémon attacks first
- Added Move's **Accuracy** and **Evasion**
    - Used to determine if a move will land on the enemy
- Added **Menu** to store Pokémon Party, Inventory, Save & Load functions
- Added **Inventory** to show player's items
- Integration of Artwork into all Unity Scenes (**UI design**)
    - Importing artworks and apply it to the grids of every scene as per what the art team has in mind
- Added **Additive Scene Loading** to the game
    - Used for loading connected scenes and unloading not connected scenes
    - Added Essential Objects Loader to load all Essential Objects such as Player prefab, Menu, Battle System etc.
- Added **Game Saving Mechanism** for player to load and save game data
    - Player can save and load data that are savable entities such as trainers and quest objects
- Added **Portals** for players to transit between indoor and outdoor scenes
    - Added location portal scripts to spawn player into different locations
    - Added a fader for incoming scenes to fade in and outgoing scenes to fade out (smooth transition)
- Ideation of **Quest Flow** for single player and multiplayer interactions
- Created single player **Quest** Interaction with NPC in the Hawker Indoor scene
- Imported **Photon Networking tools** for multiplayer functionality
- Added **Multiplayer Chat** function in the multiplayer game scene
- **Testing** and **Debugging** is done multiple times with the programming team to ensure different functionality of the game integrates well together

## 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: Project Management and Finance

When embarking on a new game project, designing the story flow must first be completed before planning what features to program. A basic guideline for a good story flow is that it should be targeted at a specific group of audience. What seems to be interesting in your eyes might not be the same to other's. Hence, consulting external parties for advice can be considered to improve the story. Gradual improvements can be done to the story, however the overarching one must be fixed.

Once it is fixed, the team can then decide what features should be built to enhance the game experience. Functions that the team plans to include is usually restricted by the member's expertise. It is important to note that the game doesn't have to include complex functions which are not in line with the story flow because what really matters is for the game to be fun. To top it off, when adding a new function, one must always keep in mind whether it can integrate smoothly with the other functions. Weighing the pros and cons of implementing the new function is of utmost importance.

Speaking of programming functions, a project management tool should be deployed to keep track of all tasks at hand. Conflicts in the game program tend to arise when multiple programmers work on a common script or scene. To keep such issues to the minimal, a project manager should be elected to assign tasks efficiently. Any new requirements and old tasks completed should be consistently updated in the project management software. The creative team and technical team can also use the project management software as a means for communication. Requests for artwork can be submitted by the technical team. Once the artwork is completed by the creative team, the technical team can then move on to integrate it into the game system.

In conclusion, when going through the development process, one must follow a certain methodology or project management system. In doing so, it allows the project to be completed in a more efficient manner. Also, consistent communication is key to keeping conflicts minimal and resolving them in a timely manner. Not to forget the importance of a project manager in ensuring tasks are assigned efficient based on expertise and availability.

## Point 2: Problem Analysis

Problems can be caused by many reasons such as poor communication, bad management, knowledge gap and unforeseen circumstances.

Problems that arise from poor communication and bad management are usually problems that can be avoided. When working in groups, it is common to encounter information not being transmitted clearly. This breakdown in communication can potentially be fatal to the progression of a project, which is why project management tools must be deployed to close the communication gap. If every group member consistently utilizes the project management software to update tasks completed, tasks in progress and submit requests, conflicts can be kept to the minimal.

To top it off, project management tools can assist the project manager in identifying any lapse in the project flow. It allows the project manager to identify who/what/when/where/why/how the problem was caused. Identifying problems that arises in a timely manner allows for resolutions before it evolves into a bigger issue.

When analyzing any problems, always start from the root cause of it. If it was caused by a gap in knowledge such as not being able to fix a bug due to lack of expertise in coding. Then the best remedy might be to seek an expert's help or simply bridge the gap in knowledge by researching and learning how to resolve it. Learning is a never-ending process as gaps in knowledge is inevitable whenever new technology emerges. The technical team must constantly upgrade, test, and implement bug fixes to ensure the program's robustness.

So, what happens when a program is not running due to unforeseen circumstances? For example, Unity Engine introduces a new update to their editor which caused the current program to produce a bug. Usually in this unforeseen situation, you might not be the only one encountering the issue. Best remedy is to either approach an online community for help or wait for the official remedy documentations provided by Unity Engine.

Finally, never forget to create a feedback loop and document it. So, when similar situations reappear, one can quickly resolve it.

## Point 3: Design and Development of Solutions

Main objective: To design a game that is fun and attractive to Singaporeans
Our group decided to work on a game which most of us can relate to. Hence, Pokémon was chosen since many of us have played it before.

The game project can be broken down into 2 phases.
Phase 1: Remake Pokémon original game
Phase 2: Add local elements, new quests, and multiplayer functionality

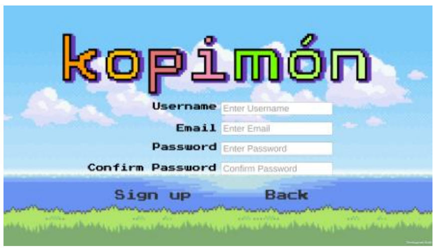
**Here is a list of features I had worked on.** Steps were documented down for future references.

### Phase 1: Firebase (Authentication of Users)

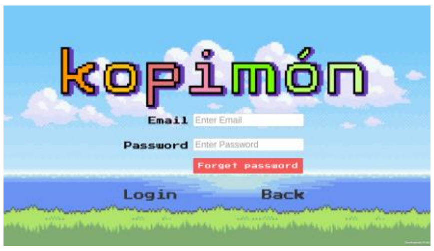
Firebase was something new that I had learned how to configure. It is used to store users email & passwords and to also authenticate users when they login. This feature was jointly done with Fuguo.

Sequence of events to set up authentication:

1. Complete Start Up scene which includes login and signup subpages
2. Create a Firebase account, configure it and import the SDK & JSON files into the project
3. Create onclick functions for login and signup buttons which pushes the relevant input texts into the firebase authentication

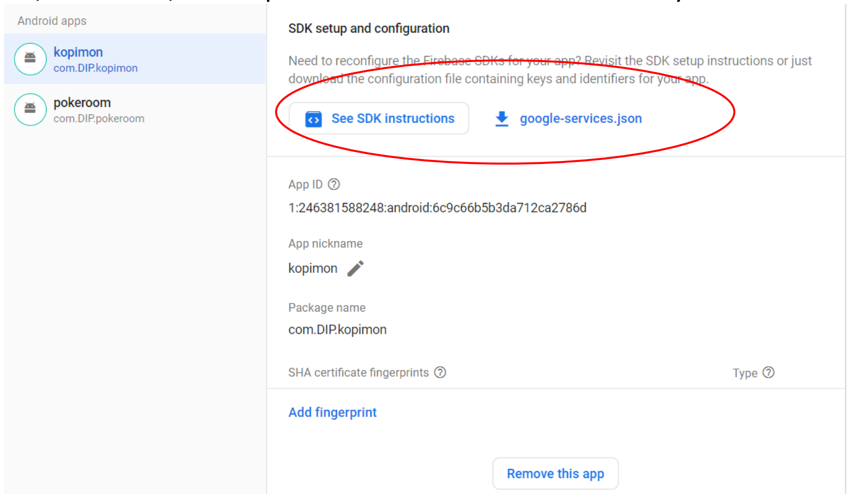


Signup Screen



Login Screen

- Configure, download, and import the SDK & JSON files into unity



SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

App ID
1:246381588248:android:6c9c66b5b3da712ca2786d

App nickname
kopimon

Package name
com.DIP.kopimon

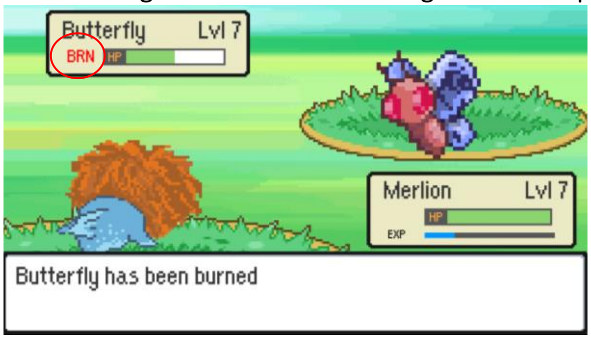
SHA certificate fingerprints | Type

Phase 1: Battle System Architecture

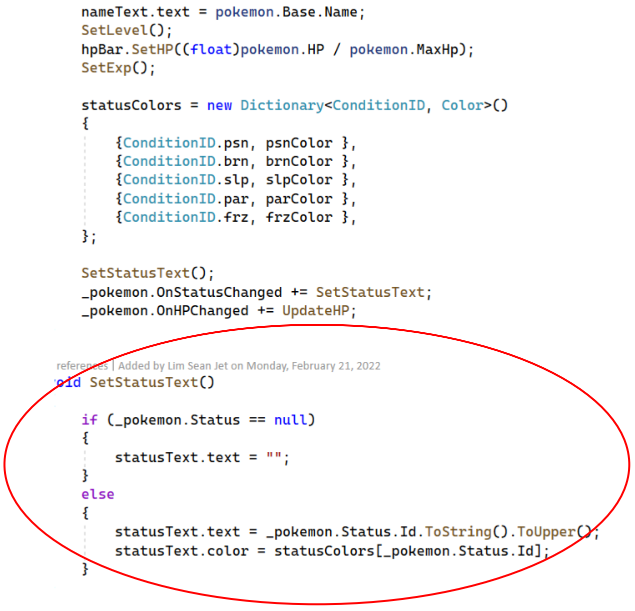
Battle System Architecture refers to the UI elements and Algorithm behind every *Pokémon* battle. This feature is tedious to program, so we had to allocate the workload to different individuals.

Here's a list of features I individually programmed for the battle system architecture:

1. Showing Status Condition changes onto the player's and enemy unit's HUD



Butterfly has been burned

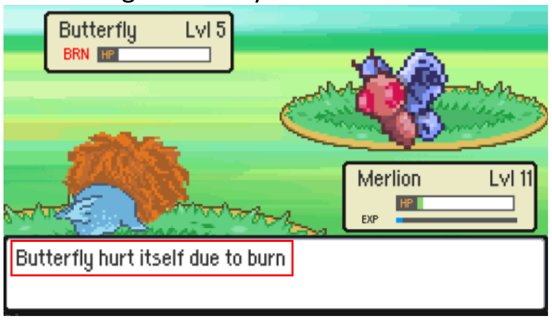


```
nameText.text = pokemon.Base.Name;
SetLevel();
hpBar.SetHP((float)pokemon.HP / pokemon.MaxHp);
SetExp();

statusColors = new Dictionary<ConditionID, Color>()
{
    {ConditionID.psn, psnColor },
    {ConditionID.brn, brnColor },
    {ConditionID.slp, slpColor },
    {ConditionID.par, parColor },
    {ConditionID.frz, frzColor },
};

SetStatusText();
_pokemon.OnStatusChanged += SetStatusText;
_pokemon.OnHPChanged += UpdateHP;
```

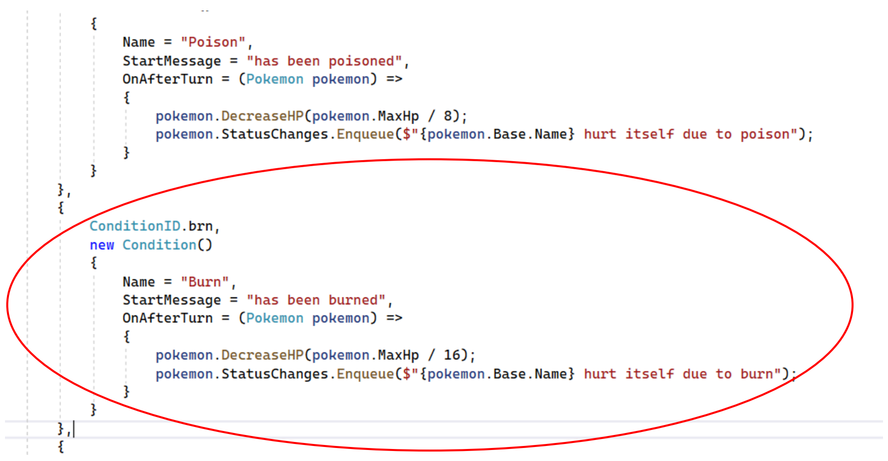
references | Added by Lim Sean Jet on Monday, February 21, 2022

```
oid SetStatusText()

if (_pokemon.Status == null)
{
    statusText.text = "";
}
else
{
    statusText.text = _pokemon.Status.Id.ToString().ToUpper();
    statusText.color = statusColors[_pokemon.Status.Id];
}
```

2. Adding secondary effects to Status Conditions

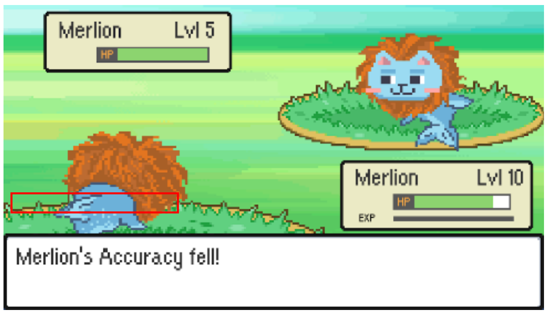


Butterfly hurt itself due to burn

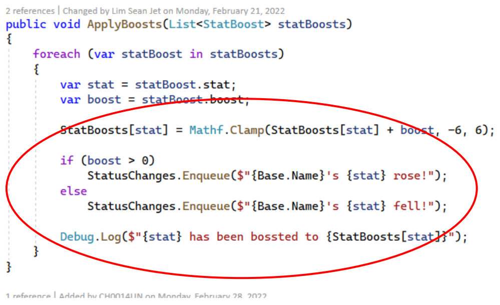


```
        {
            Name = "Poison",
            StartMessage = "has been poisoned",
            OnAfterTurn = (Pokemon pokemon) =>
            {
                pokemon.DecreaseHP(pokemon.MaxHp / 8);
                pokemon.StatusChanges.Enqueue($"{pokemon.Base.Name} hurt itself due to poison");
            }
        }
    },
    {
        ConditionID.brn,
        new Condition()
        {
            Name = "Burn",
            StartMessage = "has been burned",
            OnAfterTurn = (Pokemon pokemon) =>
            {
                pokemon.DecreaseHP(pokemon.MaxHp / 16);
                pokemon.StatusChanges.Enqueue($"{pokemon.Base.Name} hurt itself due to burn");
            }
        }
    },
    {
```

3. Adding probability of inflicting secondary effects that comes with Status Conditions
4. Showing Stats changes in a dialog box

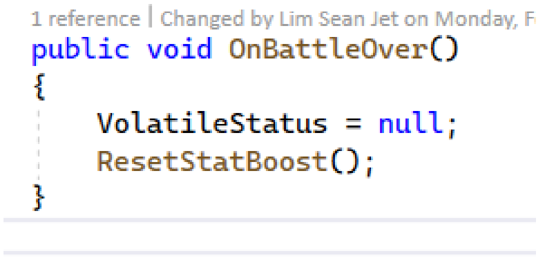


Merlion's Accuracy fell!

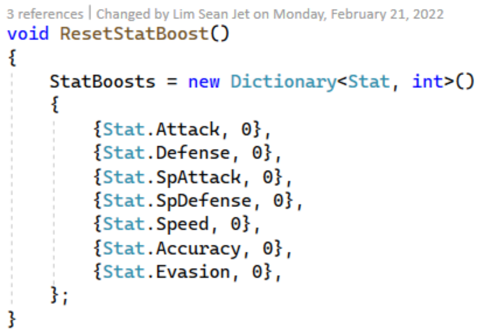


```
2 references | Changed by Lim Sean Jet on Monday, February 21, 2022
public void ApplyBoosts(List<StatBoost> statBoosts)
{
    foreach (var statBoost in statBoosts)
    {
        var stat = statBoost.stat;
        var boost = statBoost.boost;

        StatBoosts[stat] = Mathf.Clamp(StatBoosts[stat] + boost, -6, 6);

        if (boost > 0)
            StatusChanges.Enqueue($"{Base.Name}'s {stat} rose!");
        else
            StatusChanges.Enqueue($"{Base.Name}'s {stat} fell!");

        Debug.Log($"{stat} has been bossted to {StatBoosts[stat]}");
    }
}
1 reference | Added by CH0014UN on Monday, February 28, 2022
```

5. Using Speed Stat to determine which *Pokémon* gets to make a move first
6. Added Move's Accuracy and Evasion to determine the probability of landing a hit
7. Reset Stat changes at the end of a battle (Note that Status Condition does not reset)



```
1 reference | Changed by Lim Sean Jet on Monday, F
public void OnBattleOver()
{
    VolatileStatus = null;
    ResetStatBoost();
}
```



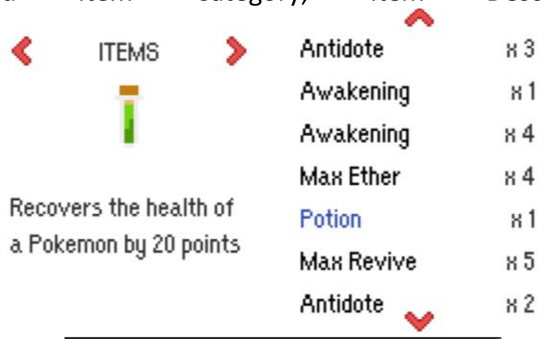
```
3 references | Changed by Lim Sean Jet on Monday, February 21, 2022
void ResetStatBoost()
{
    StatBoosts = new Dictionary<Stat, int>()
    {
        {Stat.Attack, 0},
        {Stat.Defense, 0},
        {Stat.SpAttack, 0},
        {Stat.SpDefense, 0},
        {Stat.Speed, 0},
        {Stat.Accuracy, 0},
        {Stat.Evasion, 0},
    };
}
```

## Phase 1: Inventory UI

Inventory UI refers to the display of all items held by the player. This UI will then later be added to the Essential Objects to open from Menu and Battle System.

Sequence of events to set up Inventory UI:

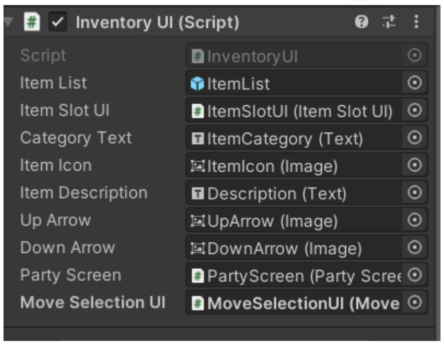
1. Add a Vertical Layout Group with Scroll View
2. Add Item Category, Item Description, and Item Slots etc.



ITEMS

Recovers the health of a Pokemon by 20 points

| | |
|---|---|
| Antidote | x 3 |
| Awakening | x 1 |
| Awakening | x 4 |
| Max Ether | x 4 |
| Potion | x 1 |
| Max Revive | x 5 |
| Antidote | x 2 |



Inventory UI (Script)

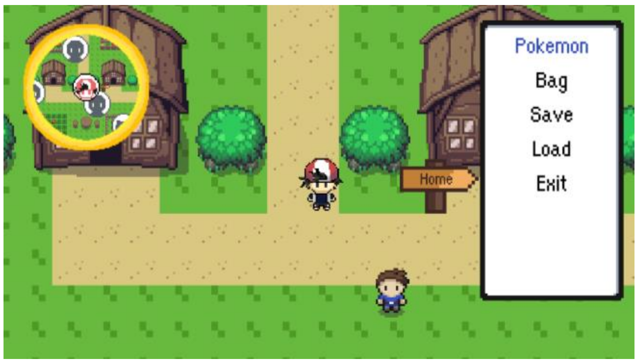
| | |
|---|---|
| Script | InventoryUI |
| Item List | ItemList |
| Item Slot UI | ItemSlotUI (Item Slot UI) |
| Category Text | ItemCategory (Text) |
| Item Icon | ItemIcon (Image) |
| Item Description | Description (Text) |
| Up Arrow | UpArrow (Image) |
| Down Arrow | DownArrow (Image) |
| Party Screen | PartyScreen (Party Scree |
| Move Selection UI | MoveSelectionUI (Move |

## Phase 1: Menu

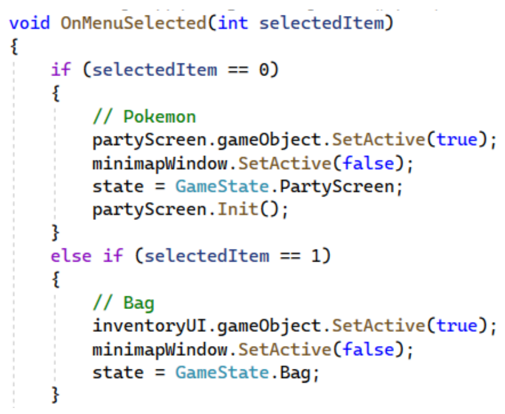
Menu is basically used to store basic functions that a user can access anytime, excluding multiplayer scene. Reason why menu is removed inside the multiplayer scene is because save and load functions should not be included.

Sequence of events to set up Menu:

1. Add a Vertical Layout group
2. Add Menu Items



3. Add Menu to Essential Object
4. Game Controller script can set active different UI Canvas child objects



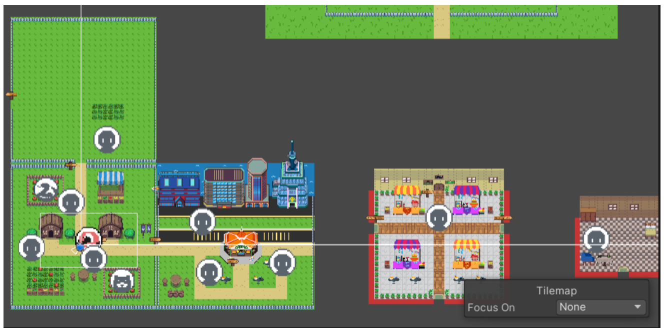
```
void OnMenuSelected(int selectedItem)
{
    if (selectedItem == 0)
    {
        // Pokemon
        partyScreen.gameObject.SetActive(true);
        minimapWindow.SetActive(false);
        state = GameState.PartyScreen;
        partyScreen.Init();
    }
    else if (selectedItem == 1)
    {
        // Bag
        inventoryUI.gameObject.SetActive(true);
        minimapWindow.SetActive(false);
        state = GameState.Bag;
    }
```

<u>Phase 1: Additive Scene Loading</u>

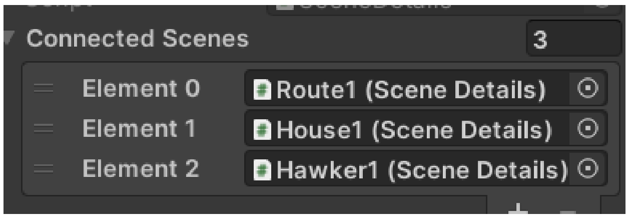
Additive Scene Loading is used to create smoother transition between scenes and easy to implement each time a new scene is created.

Sequence of events to set up Additive Scene Loading:

1. Create a new Gameplay scene
2. Rearrange the layout of every subscenes within the Gameplay



3. Add connected scenes component to each child object in Gameplay scene

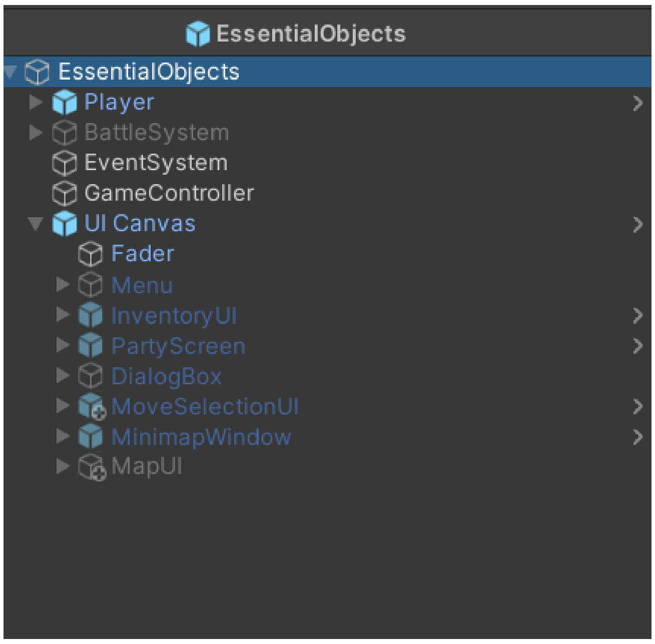


| Connected Scenes | 3 |
|---|---|
| Element 0 | Route1 (Scene Details) |
| Element 1 | House1 (Scene Details) |
| Element 2 | Hawker1 (Scene Details) |

4. Create scripts to load subscenes which are connected and unload scenes which are not connected

```
// Load all connected scenes
foreach (var scene in connectedScenes)
{
    scene.LoadScene();
}

// Unload the scenes that are no longer connected
var prevScene = GameController.Instance.PrevScene;
if (prevScene != null)
{
    var previouslyLoadedScenes = prevScene.connectedScenes;
    foreach (var scene in previouslyLoadedScenes)
    {
        if (!connectedScenes.Contains(scene) && scene != this)
            scene.UnloadScene();
    }

    if (!connectedScenes.Contains(prevScene))
        prevScene.UnloadScene();
}
```

5. Create the Essential Object Loader
6. Add Essential objects such as Player prefab, Menu, Inventory UI etc. to the Essential Object Loader

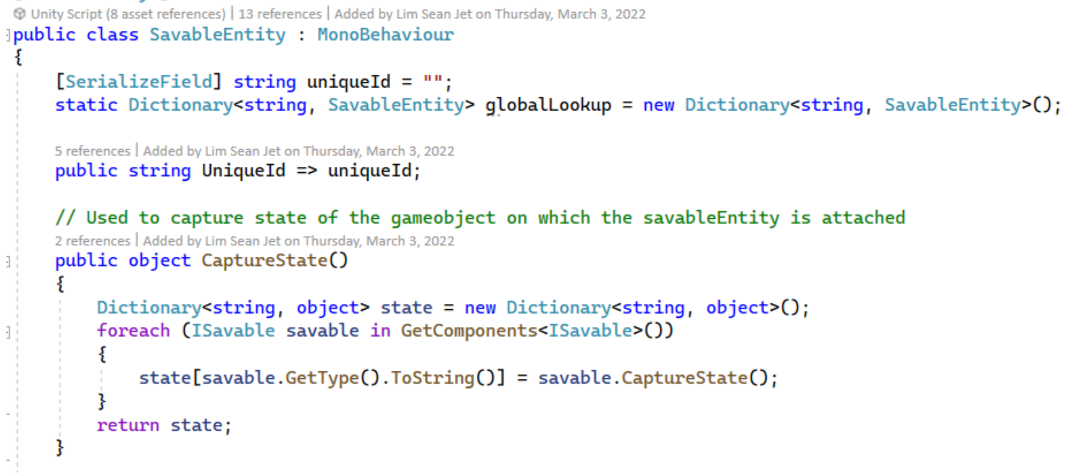


## Phase 1: Saving and Loading Mechanism

This feature allows the user to save its player location and other savable entities such as completed trainer battles and NPC quests.

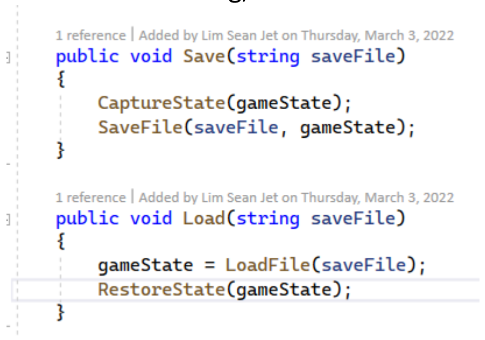
Sequence of events to set up Saving and Loading mechanism:

1. Create scripts to identify savable entities



```
Unity Script (8 asset references) | 13 references | Added by Lim Sean Jet on Thursday, March 3, 2022
public class SavableEntity : MonoBehaviour
{
    [SerializeField] string uniqueId = "";
    static Dictionary<string, SavableEntity> globalLookup = new Dictionary<string, SavableEntity>();

    5 references | Added by Lim Sean Jet on Thursday, March 3, 2022
    public string UniqueId => uniqueId;

    // Used to capture state of the gameobject on which the savableEntity is attached
    2 references | Added by Lim Sean Jet on Thursday, March 3, 2022
    public object CaptureState()
    {
        Dictionary<string, object> state = new Dictionary<string, object>();
        foreach (ISavable savable in GetComponents<ISavable>())
        {
            state[savable.GetType().ToString()] = savable.CaptureState();
        }
        return state;
    }
```

2. Assign scripts to game objects such as Trainers, NPCs, and Kopimon Level etc.
3. Create a script to capture state, serialize it and store the file for Saving
4. For Loading, we need to load back the saved file, deserialize it and restore the game state



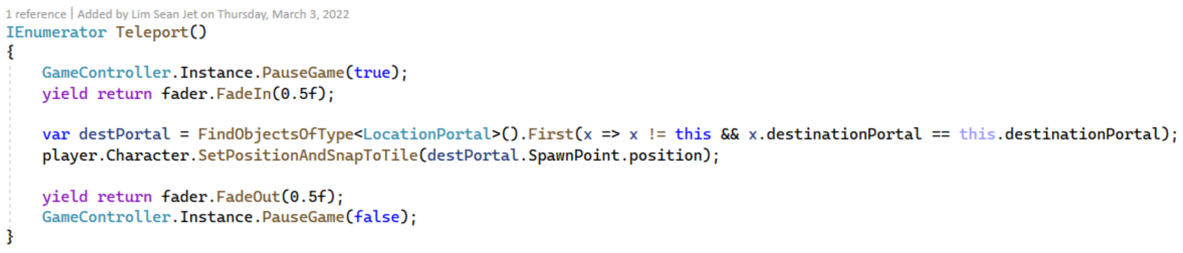
```
1 reference | Added by Lim Sean Jet on Thursday, March 3, 2022
public void Save(string saveFile)
{
    CaptureState(gameState);
    SaveFile(saveFile, gameState);
}

1 reference | Added by Lim Sean Jet on Thursday, March 3, 2022
public void Load(string saveFile)
{
    gameState = LoadFile(saveFile);
    RestoreState(gameState);
}
```

## Phase 1: Location Portals

Location portals are used from transiting between indoor and outdoor scenes. Since indoor and outdoor scenes are not lying beside one another, I needed to teleport the location of the player prefab to another scene.

Sequence of events to set up location portals:

1. Create scripts to shift the player to destination portal
2. Add a fader for incoming scene to fade in and outgoing scenes to fade out (smooth transition)

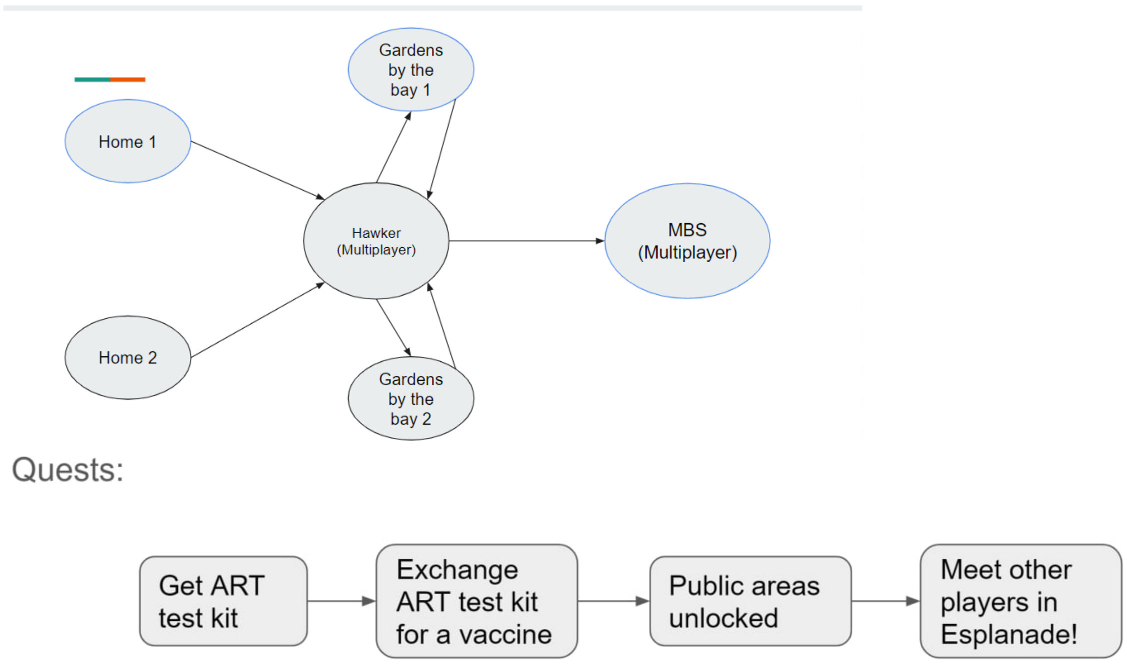


```
1 reference | Added by Lim Sean Jet on Thursday, March 3, 2022
IEnumerator Teleport()
{
    GameController.Instance.PauseGame(true);
    yield return fader.FadeIn(0.5f);

    var destPortal = FindObjectsOfType<LocationPortal>().First(x => x != this && x.destinationPortal == this.destinationPortal);
    player.Character.SetPositionAndSnapToTile(destPortal.SpawnPoint.position);

    yield return fader.FadeOut(0.5f);
    GameController.Instance.PauseGame(false);
}
```

Phase 2: Ideation of Quest Flow

This was the start of Phase 2 and the main objective of introducing quests is to make the game more interesting by introducing a goal for the user to achieve. Since everyone is still pondering when will covid end, our group decided to work on a Covid Quest where the user must battle Kopimons, retrieve *ART test kits* to exchange for a *Moderna Vaccine.* After getting vaccinated, the player is safe to walk around neighborhoods and visit other players in malls/centers such as Esplanade.

To aid the programming of quest's interactions, a quest chart was curated.



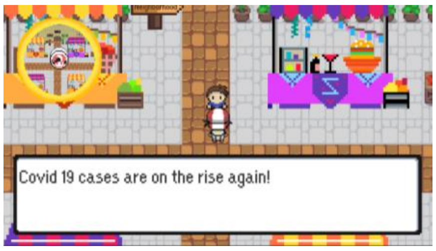
## Phase 2: Quest Programming

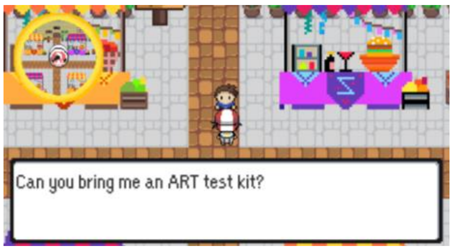
Since I was the one to introduce the Additive Scene Loading, I oversaw creating the basic Gardens by The Bay, Hawker Indoor, Neighborhood, Esplanade and Esplanade Indoor scene. Beautifying the scenes were later done by me as per the instructions given by creative team.

As for the quest programming I was only assigned to program the interaction with NPC in the 'Hawker Indoor' scene where the player was not allowed to proceed into the 'Neighborhood' scene unless he obtains the *Moderna Vaccine*. To obtain the *Moderna Vaccine*, the player must first enter the 'Gardens by The Bay' scene, obtain the *ART Test Kit* from the Otter then return to the 'Hawker Indoor' scene to exchange the *ART Test Kit* for the *Moderna Vaccine*.

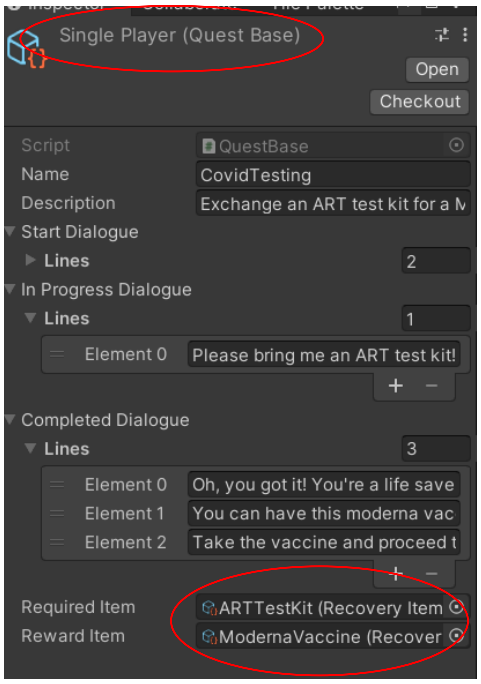
Sequence of events to set up Quest in Hawker Indoor scene:

1. Create a Quest object with a story dialog and road blocker
2. Add the Quest object to the NPC for interaction with player
3. Add dialogs to the NPC to prompt the player

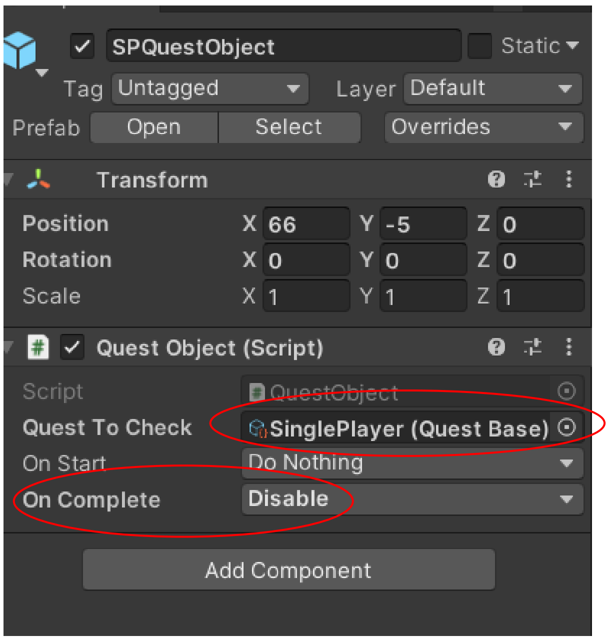




4. Add Reward Item for the NPC to give the Moderna Vaccine once quest is completed
5. Disable road blocker once quest is completed



Single Player (Quest Base)
Open
Checkout
Script QuestBase
Name CovidTesting
Description Exchange an ART test kit for a M
Start Dialogue
Lines 2
In Progress Dialogue
Lines 1
Element 0 Please bring me an ART test kit!
Completed Dialogue
Lines 3
Element 0 Oh, you got it! You're a life save
Element 1 You can have this moderna vac
Element 2 Take the vaccine and proceed t
Required Item ARTTestKit (Recovery Item
Reward Item ModernaVaccine (Recover



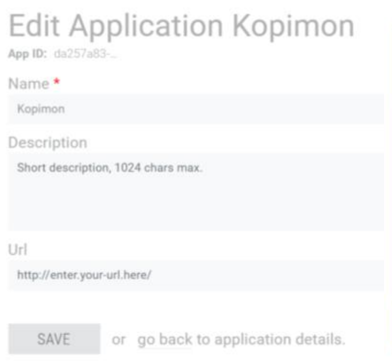
SPQuestObject Static
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X 66 Y -5 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Quest Object (Script)
Script QuestObject
Quest To Check SinglePlayer (Quest Base)
On Start Do Nothing
On Complete Disable
Add Component

## Phase 2: Multiplayer (Photon Pun 2)

To introduce multiplayer functionality, Photon Networking tools were introduced for our game to connect to an online server. Implementing multiplayer function into our game was not easy as our game was based off a single player game. We had to weigh the pros and cons of introducing multiplayer functionality because it affects the existing game features. What our group decided to do was to add the multiplayer functionality into the Esplanade Indoor scene which was also our last scene.

Here's what I did to set up the Photon Network:

1. Register a new Photon account
2. Import Photon Pun 2 into unity game project and Configure Photon



Edit Application Kopimon

App ID: da257a83-...

Name *

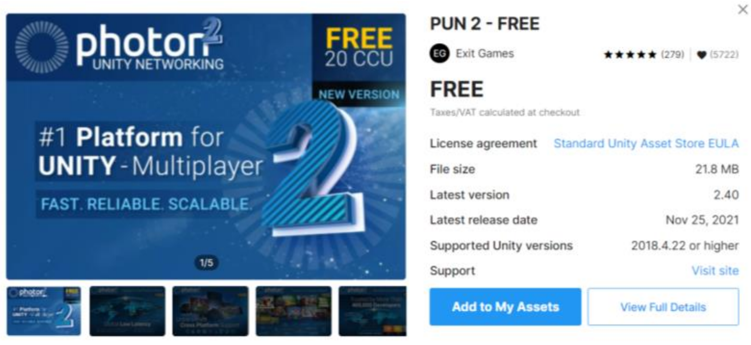
Kopimon

Description

Short description, 1024 chars max.

Url

http://enter.your-url.here/



PUN 2 - FREE

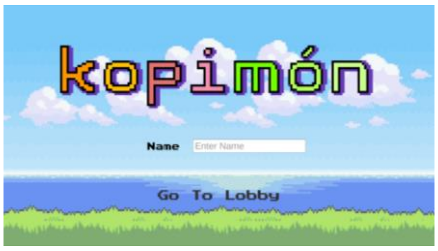
Exit Games ★★★★★ (279) | ♥ (5722)

FREE

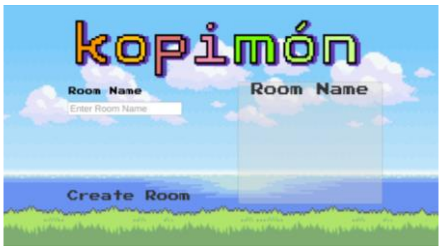
Taxes/VAT calculated at checkout

| | |
|---|---|
| License agreement | Standard Unity Asset Store EULA |
| File size | 21.8 MB |
| Latest version | 2.40 |
| Latest release date | Nov 25, 2021 |
| Supported Unity versions | 2018.4.22 or higher |
| Support | Visit site |

3. Create Registration and Lobby scenes for players connecting to server

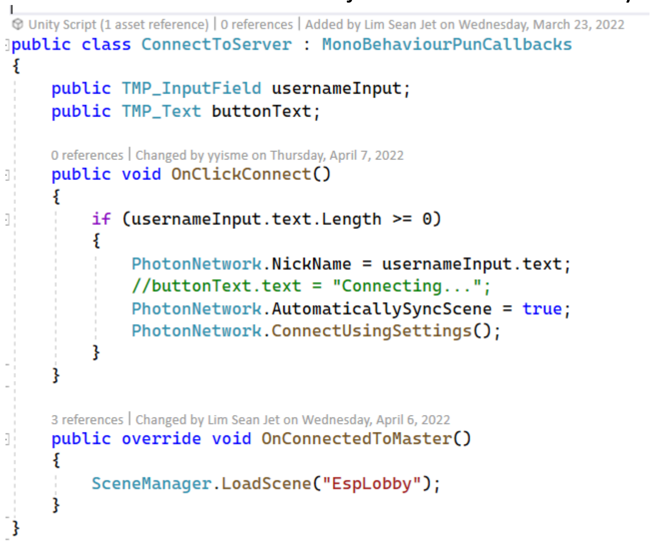


Enter Nickname

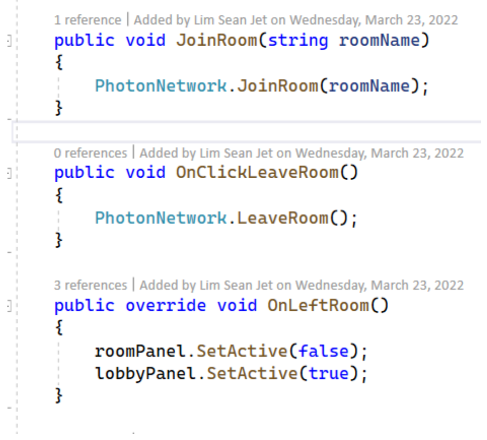


Create Room

4. Add onclick functions to join server -> Join Lobby -> Create Room



```
Unity Script (1 asset reference) | 0 references | Added by Lim Sean Jet on Wednesday, March 23, 2022
public class ConnectToServer : MonoBehaviourPunCallbacks
{
    public TMP_InputField usernameInput;
    public TMP_Text buttonText;

    0 references | Changed by yyisme on Thursday, April 7, 2022
    public void OnClickConnect()
    {
        if (usernameInput.text.Length >= 0)
        {
            PhotonNetwork.NickName = usernameInput.text;
            //buttonText.text = "Connecting...";
            PhotonNetwork.AutomaticallySyncScene = true;
            PhotonNetwork.ConnectUsingSettings();
        }
    }

    3 references | Changed by Lim Sean Jet on Wednesday, April 6, 2022
    public override void OnConnectedToMaster()
    {
        SceneManager.LoadScene("EspLobby");
    }
}
```



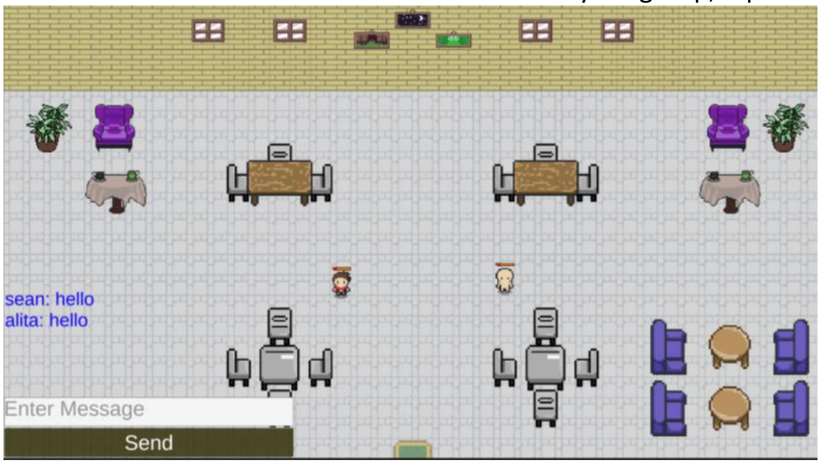
```
1 reference | Added by Lim Sean Jet on Wednesday, March 23, 2022
public void JoinRoom(string roomName)
{
    PhotonNetwork.JoinRoom(roomName);
}

0 references | Added by Lim Sean Jet on Wednesday, March 23, 2022
public void OnClickLeaveRoom()
{
    PhotonNetwork.LeaveRoom();
}

3 references | Added by Lim Sean Jet on Wednesday, March 23, 2022
public override void OnLeftRoom()
{
    roomPanel.SetActive(false);
    lobbyPanel.SetActive(true);
}
```

Phase 2: Multiplayer Chat

In the Esplanade Indoor scene, players who joined the same room can interact with one another. I was tasked to complete the Multiplayer Chat functionality. As this was a basic feature, I was able to complete it myself.

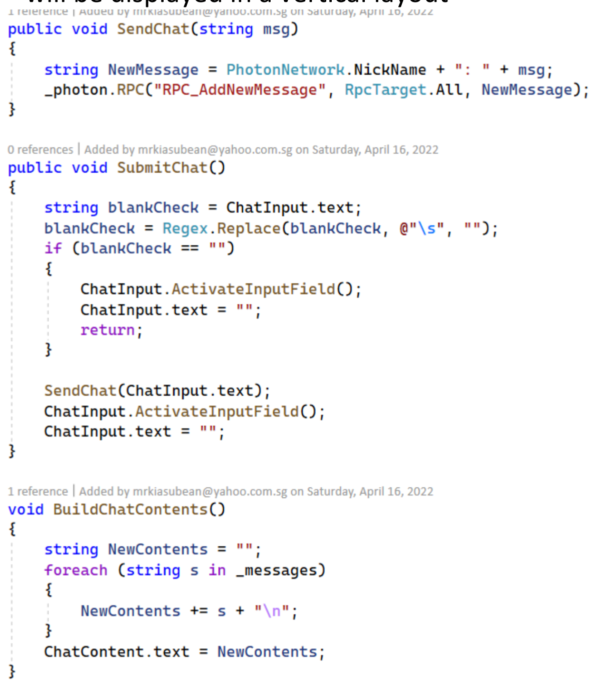
Sequence of events to set up Multiplayer Chat:

1. Create a canvas that consists of a Vertical Layout group, Input field and Button



sean: hello
alita: hello

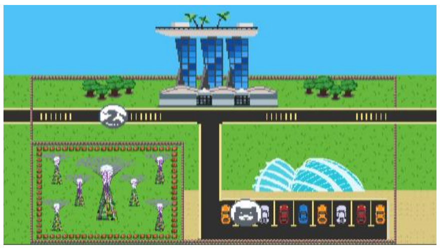
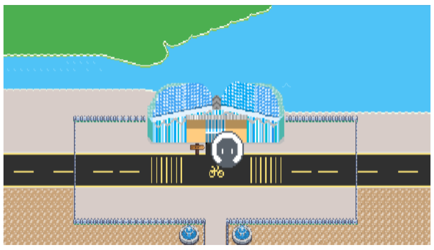
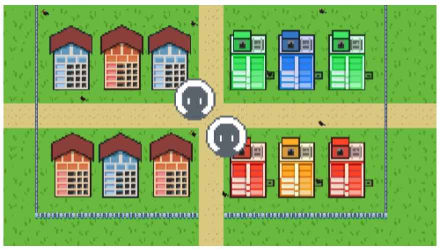
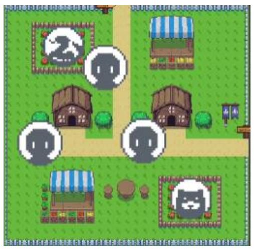
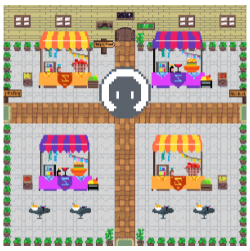
2. Add onclick functions to the button such that messages sent by players in the same room will be displayed in a vertical layout



```
1 reference | Added by mrkiasubean@yahoo.com.sg on Saturday, April 16, 2022
public void SendChat(string msg)
{
    string NewMessage = PhotonNetwork.NickName + ": " + msg;
    _photon.RPC("RPC_AddNewMessage", RpcTarget.All, NewMessage);
}

0 references | Added by mrkiasubean@yahoo.com.sg on Saturday, April 16, 2022
public void SubmitChat()
{
    string blankCheck = ChatInput.text;
    blankCheck = Regex.Replace(blankCheck, @"\s", "");
    if (blankCheck == "")
    {
        ChatInput.ActivateInputField();
        ChatInput.text = "";
        return;
    }

    SendChat(ChatInput.text);
    ChatInput.ActivateInputField();
    ChatInput.text = "";
}

1 reference | Added by mrkiasubean@yahoo.com.sg on Saturday, April 16, 2022
void BuildChatContents()
{
    string NewContents = "";
    foreach (string s in _messages)
    {
        NewContents += s + "\n";
    }
    ChatContent.text = NewContents;
}
```

Phase 1 and Phase 2: Importing Artwork and Beautifying Scenes

As a member of the technical team, I had to consistently alias with the creative team to ensure the UI elements of our game is up to standard. Sprites given to me had to be imported, configured, and added to the tile sheets. As per the instructions of the creative team, I had beautified all the scenes with our group's artwork and tried my best to materialize what the creative team had in mind. Here are some scenes I had beautified.

Phase 1 and Phase 2: Testing and Debugging

Throughout the whole development process, rigorous testing was done. Bugs were common as our group integrates more features. I was consistently testing the whole game and looked out for minute details to ensure the robustness of our program. Countless hours were spent on the rigorous testing and not to mention the bug fixes.

Major Bug Fixes include:

1. Fixing of Firebase integration
2. Fixing of Photon integration
3. Fixing of Saving mechanism
4. Fixing of Scene transitions
5. Fixing of Bugs caused by renaming prefabs
6. Fixing of Sprites renderer
7. Fixing of Build Settings