Panda Mahjong
Final Report

40847011S 高子翔, 40847019S 施育衡, 40847024S 連庭萱. 40947025S 黃冠棠, 40947047S 洪盛益

I. Summary

A. Project Description

麻將遊戲，每間遊戲房最多可供四名玩家加入，玩家將獨自進行遊戲，可於單場遊戲後回到房間查看所有玩家的遊戲情況。規則主要採台麻規則，但不支援部分牌型。

B. Work Breakdown

分工主要分為伺服器連線與遊戲本體製作。

1) 遊戲本體製作*:*

高子翔：算台，開始、房間介面程式
連庭萱：麻將遊戲流程
黃冠棠：遊戲互動介面及音效

2) 網路連線等伺服器實作*:*

施育衡：同步顯示遊玩房間、玩家狀態同步化
洪盛益：房間內斷線重連，Host 斷線偵測，重構專案

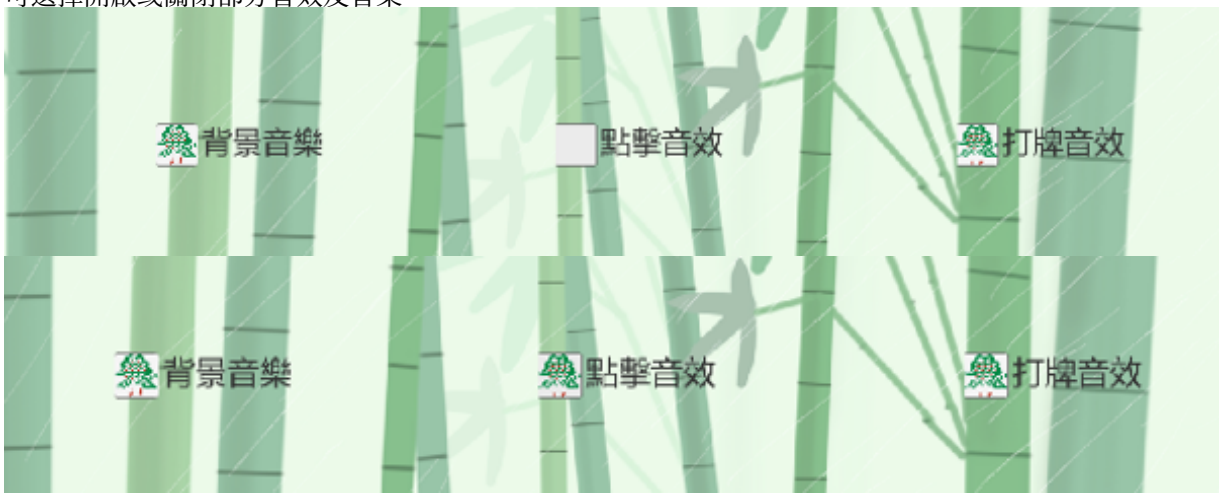
II. Use cases

A. 玩家資訊

玩家可設置自己的遊戲名稱，並隨機產生 guid

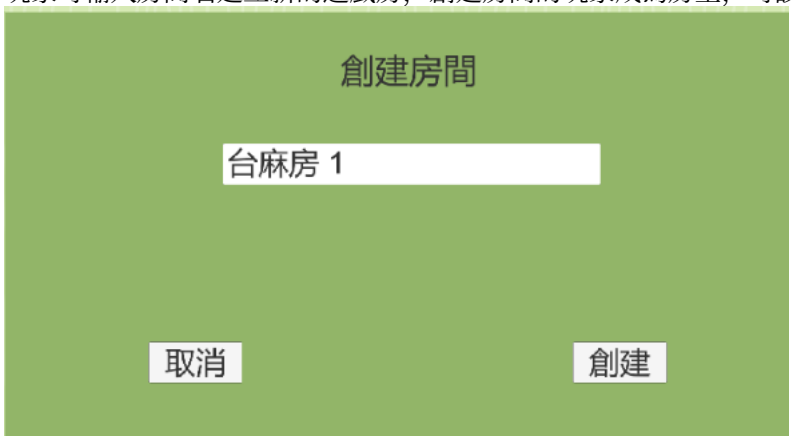
B. 音訊設置

可選擇開啟或關閉部分音效及音樂



C. 建立遊戲房

玩家可輸入房間名建立新的遊戲房，創建房間的玩家成為房主，可設定底、台數與開始遊戲。



D. 加入遊戲房

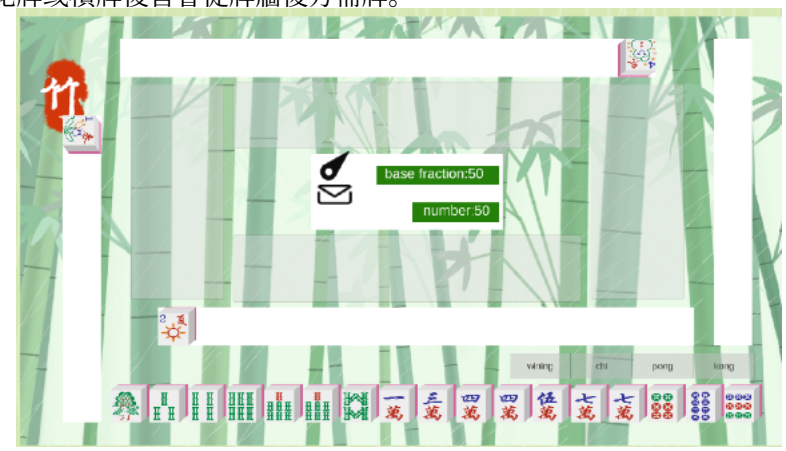
玩家可從房間列表中選擇房間加入，進入後按準備等待房主開始遊戲。

E. 進入牌桌與牌堆初始化

當所有玩家皆按下準備後，房主即可點擊開始遊戲，所有玩家進入牌桌畫面，並由系統分配位置與莊家（摸牌順序）。每次遊戲的牌堆皆會隨機洗牌。

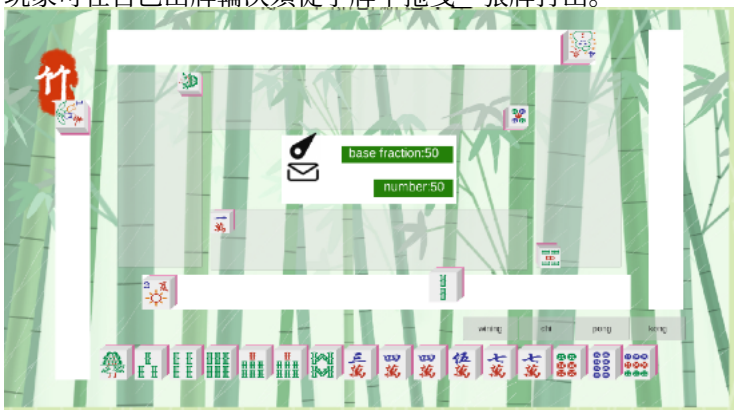
F. 發牌與規則補花與摸牌

依照麻將規則進行每輪的發牌與補牌。正常輪替下每位玩家將於自己的輪次從牌牆最前端獲得一張牌。當玩家摸到花牌或槓牌後皆會從牌牆後方補牌。



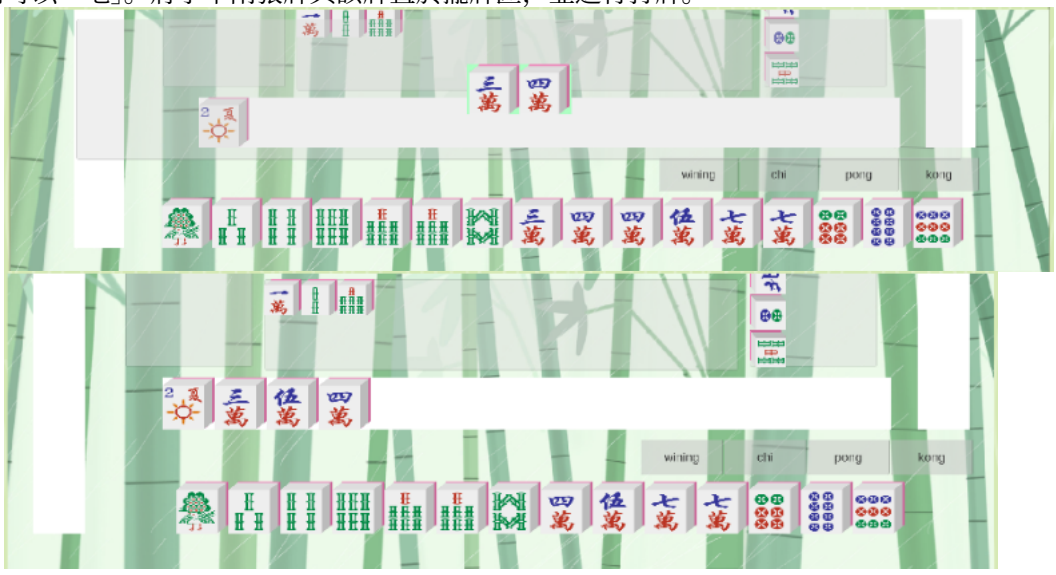
G. 打牌

玩家可在自己出牌輪次須從手牌中拖曳一張牌打出。

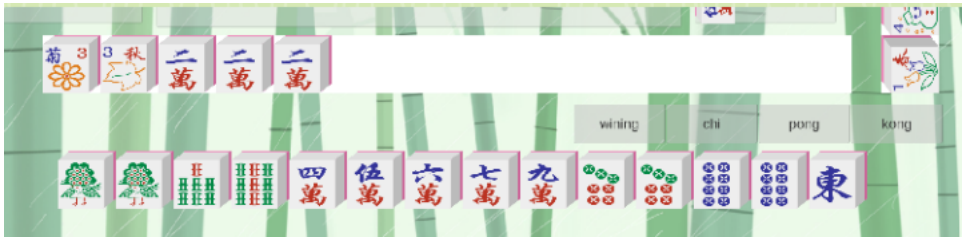


H. 吃碰槓胡

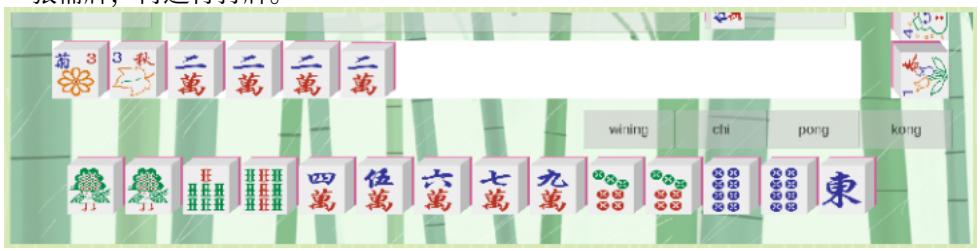
1) 吃: 當上一位玩家（麻將中稱上家）打出的牌可與自身手牌中的兩張牌組合成「順子」（同花色的三個接續數字），則可以「吃」。將手中兩張牌與該牌置於擺牌區，並進行打牌。



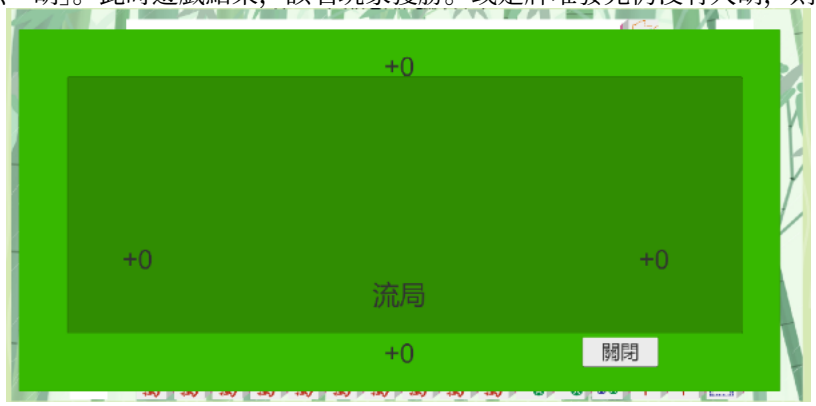
2) 碰: 當其他玩家打出的牌可與手牌中的兩張牌組合成「刻子」（三張相同數字花色牌），則可以「碰」。將手中兩張牌與該牌置於擺牌區，並進行打牌。



3) 槓*:* 當手中已有「刻子」，此時其他玩家再打出同樣的牌，則可以「槓」。將手中的刻子與該牌置於擺牌區，接著取得一張補牌，再進行打牌。

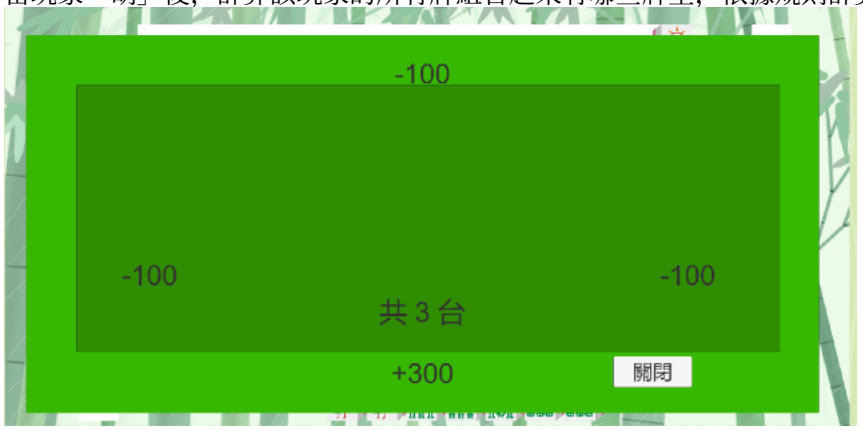


4) 胡*/*流局*:* 當玩家手牌與擺牌加起來有五組對子（「刻子」及「順子」）加上一對「眼睛（兩張相同的牌）」時，則可以「胡」。此時遊戲結束，該名玩家獲勝。或是牌堆發完仍沒有人胡，則流局。



I. 計算台數

當玩家「胡」後，計算該玩家的所有牌組合起來有哪些牌型，依據規則計算所有牌型共可以獲得多少分數。



J. 同步化

房間內的玩家資訊同步，在 waitingRoom 內部點擊 ready 按鈕就可以同步顯示到其他玩家。

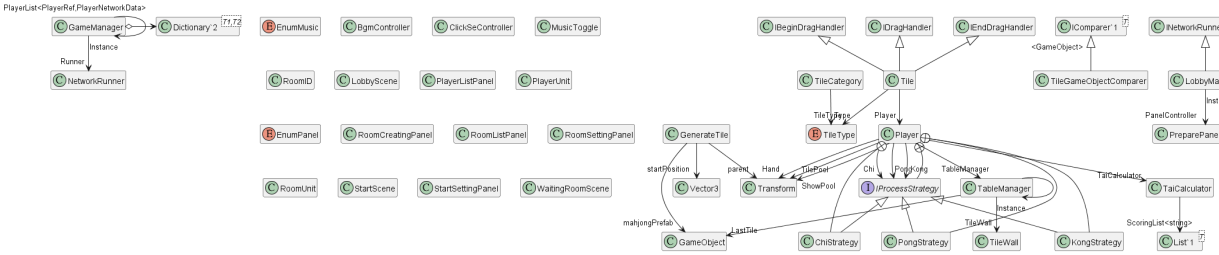
III. Analysis and design techniques

A. Requirements

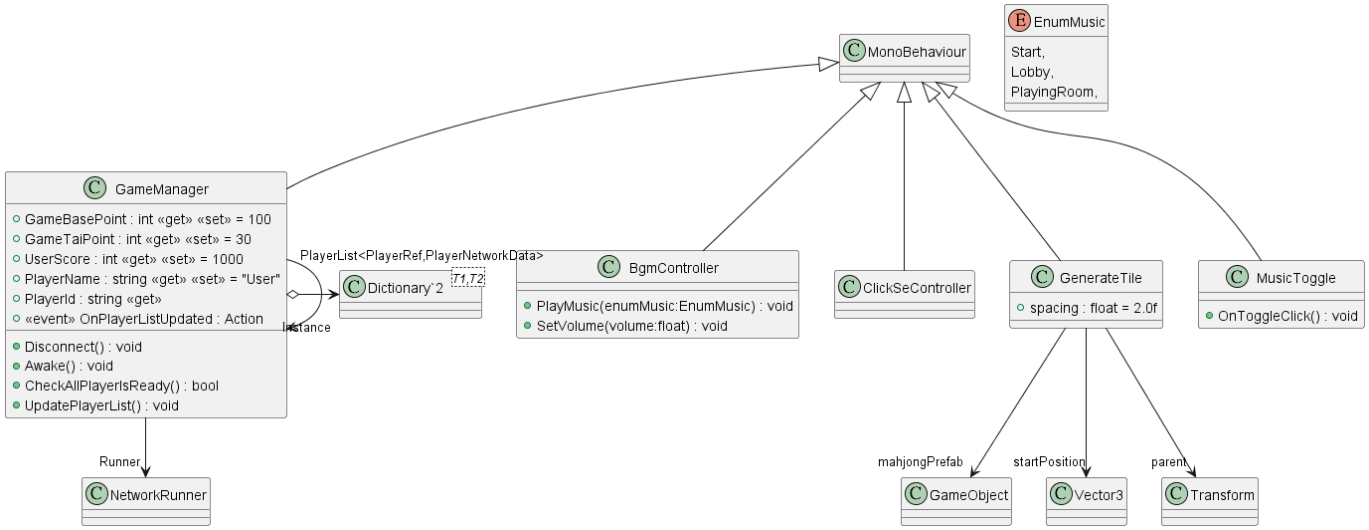
1) 連線*:* 使用者可以看到其他玩家的遊玩結果。
2) 遊戲*:* 使用者能完整遊玩所有麻將行為，包含出牌摸牌補牌、吃碰槓和胡。
3) 計分*:* 使用者可設定基準分數，並於獲勝後計算該獲勝手牌的牌型得分。
4) 個人化設定*:* 使用者可設定使用者名稱等資訊，並調整遊戲內音效。

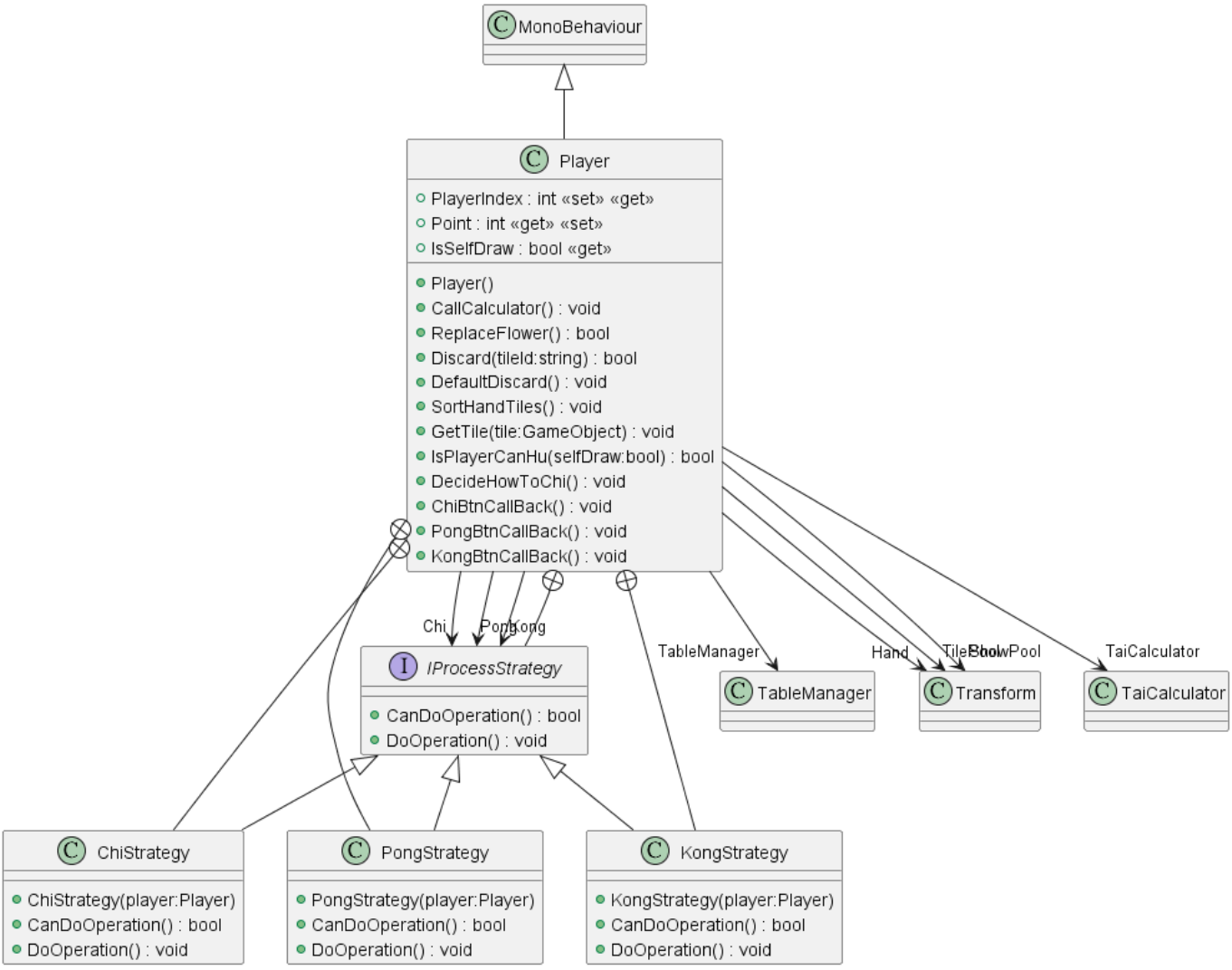
B. Class diagrams

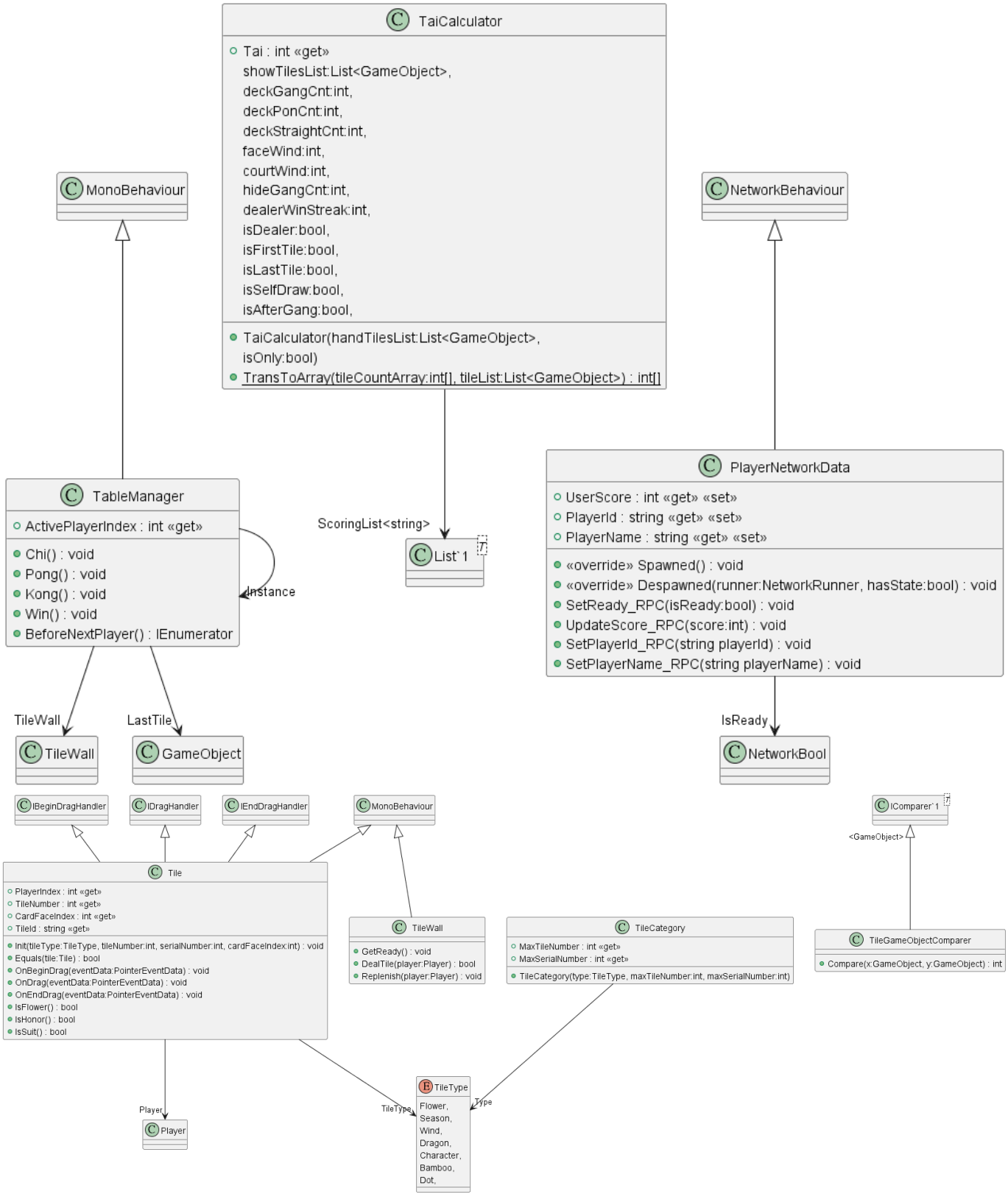
1) All in one(simple version): 除去因為使用 Unity 開發遊戲而大量繼承 MonoBehaviour 以外的 class diagram。

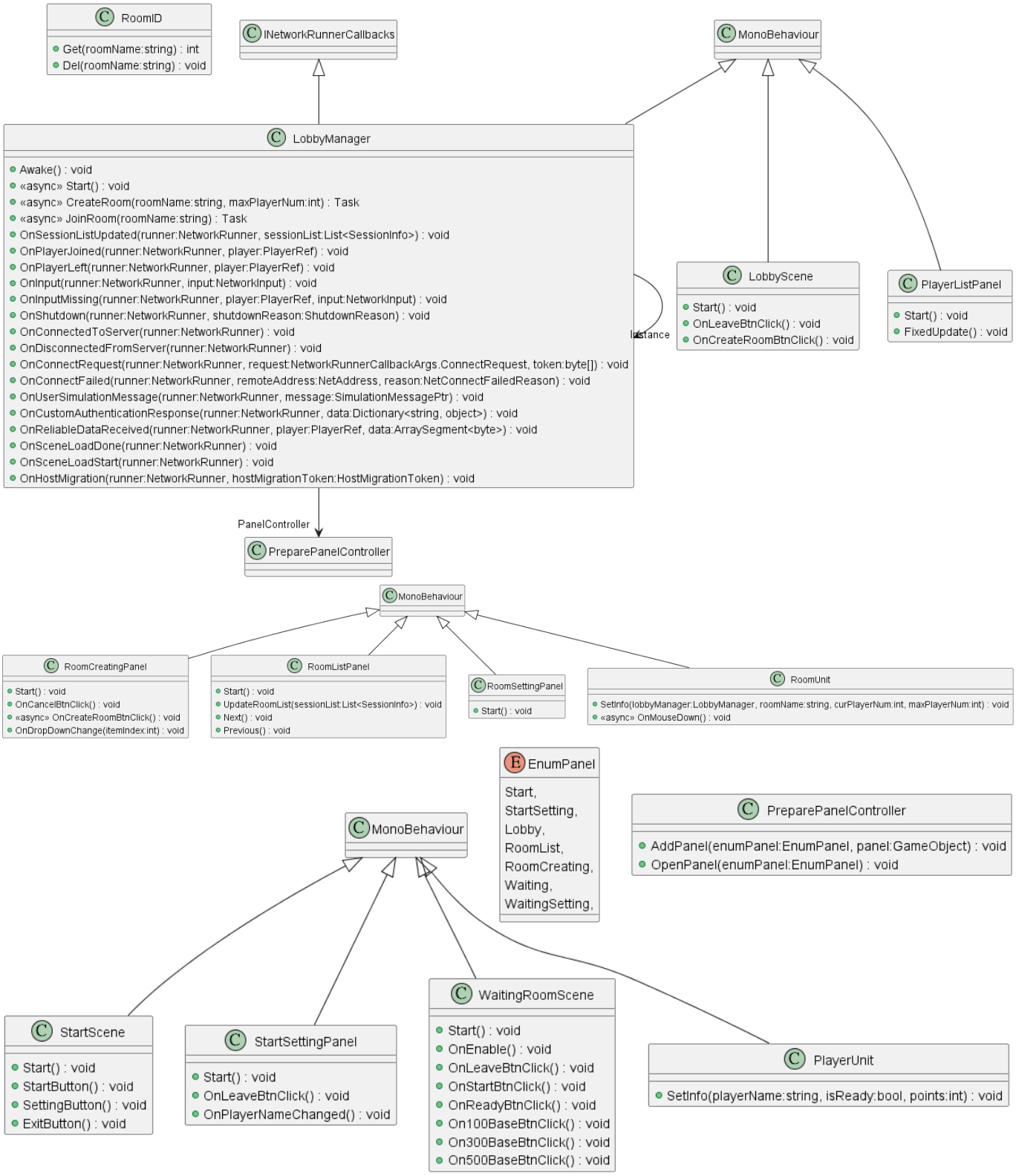


2) Detail:

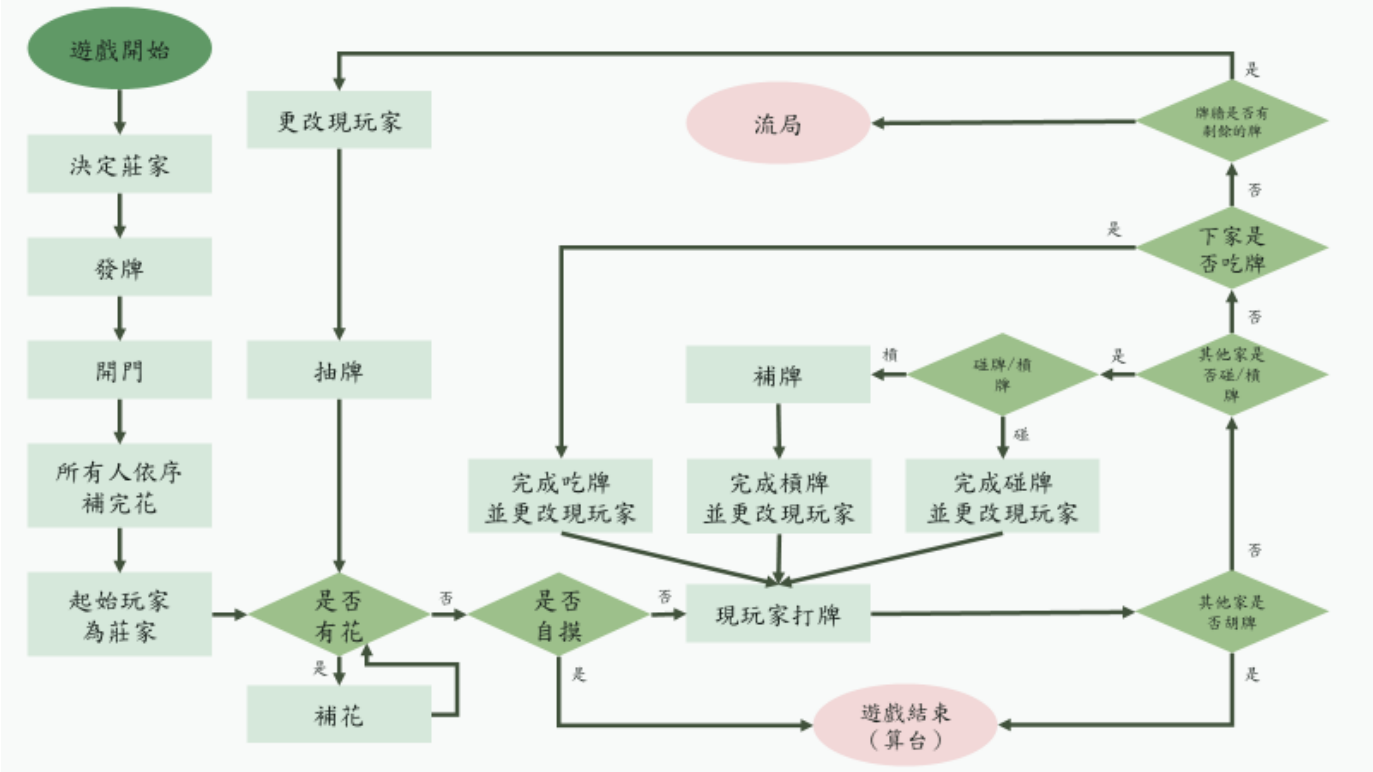



MonoBehaviour
Player
PlayerIndex : int «set» «get»
Point : int «get» «set»
IsSelfDraw : bool «get»
Player()
CallCalculator() : void
ReplaceFlower() : bool
Discard(tileId:string) : bool
DefaultDiscard() : void
SortHandTiles() : void
GetTile(tile:GameObject) : void
IsPlayerCanHu(selfDraw:bool) : bool
DecideHowToChi() : void
ChiBtnCallBack() : void
PongBtnCallBack() : void
KongBtnCallBack() : void
Chi
Pong
Kong
IProcessStrategy
CanDoOperation() : bool
DoOperation() : void
TableManager
Hand
TileShowPool
TaiCalculator
TableManager
Transform
TaiCalculator
ChiStrategy
ChiStrategy(player:Player)
CanDoOperation() : bool
DoOperation() : void
PongStrategy
PongStrategy(player:Player)
CanDoOperation() : bool
DoOperation() : void
KongStrategy
KongStrategy(player:Player)
CanDoOperation() : bool
DoOperation() : void


TaiCalculator
Tai : int «get»
showTilesList:List<GameObject>,
deckGangCnt:int,
deckPonCnt:int,
deckStraightCnt:int,
faceWind:int,
courtWind:int,
hideGangCnt:int,
dealerWinStreak:int,
isDealer:bool,
isFirstTile:bool,
isLastTile:bool,
isSelfDraw:bool,
isAfterGang:bool,
TaiCalculator(handTilesList:List<GameObject>, isOnly:bool)
TransToArray(tileCountArray:int[], tileList:List<GameObject>) : int[]
MonoBehaviour
NetworkBehaviour
TableManager
ActivePlayerIndex : int «get»
Chi() : void
Pong() : void
Kong() : void
Win() : void
BeforeNextPlayer() : IEnumerator
Instance
ScoringList<string>
List`1
PlayerNetworkData
UserScore : int «get» «set»
PlayerId : string «get» «set»
PlayerName : string «get» «set»
«override» Spawned() : void
«override» Despawned(runner:NetworkRunner, hasState:bool) : void
SetReady_RPC(isReady:bool) : void
UpdateScore_RPC(score:int) : void
SetPlayerId_RPC(string playerId) : void
SetPlayerName_RPC(string playerName) : void
TileWall
LastTile
IsReady
TileWall
GameObject
NetworkBool
IBeginDragHandler
IDragHandler
IEndDragHandler
MonoBehaviour
IComparer`1
<GameObject>
Tile
PlayerIndex : int «get»
TileNumber : int «get»
CardFaceIndex : int «get»
TileId : string «get»
Init(tileType:TileType, tileNumber:int, serialNumber:int, cardFaceIndex:int) : void
Equals(tile:Tile) : bool
OnBeginDrag(eventData:PointerEventData) : void
OnDrag(eventData:PointerEventData) : void
OnEndDrag(eventData:PointerEventData) : void
IsFlower() : bool
IsHonor() : bool
IsSuit() : bool
TileWall
GetReady() : void
DealTile(player:Player) : bool
Replenish(player:Player) : void
TileCategory
MaxTileNumber : int «get»
MaxSerialNumber : int «get»
TileCategory(type:TileType, maxTileNumber:int, maxSerialNumber:int)
TileGameObjectComparer
Compare(x:GameObject, y:GameObject) : int
Player
Player
TileType
TileType
Flower,
Season,
Wind,
Dragon,
Character,
Bamboo,
Dot,
Type


RoomID
Get(roomName:string) : int
Del(roomName:string) : void
INetworkRunnerCallbacks
MonoBehaviour
LobbyManager
Awake() : void
«async» Start() : void
«async» CreateRoom(roomName:string, maxPlayerNum:int) : Task
«async» JoinRoom(roomName:string) : Task
OnSessionListUpdated(runner:NetworkRunner, sessionList:List<SessionInfo>) : void
OnPlayerJoined(runner:NetworkRunner, player:PlayerRef) : void
OnPlayerLeft(runner:NetworkRunner, player:PlayerRef) : void
OnInput(runner:NetworkRunner, input:NetworkInput) : void
OnInputMissing(runner:NetworkRunner, player:PlayerRef, input:NetworkInput) : void
OnShutdown(runner:NetworkRunner, shutdownReason:ShutdownReason) : void
OnConnectedToServer(runner:NetworkRunner) : void
OnDisconnectedFromServer(runner:NetworkRunner) : void
OnConnectRequest(runner:NetworkRunner, request:NetworkRunnerCallbackArgs.ConnectRequest, token:byte[]) : void
OnConnectFailed(runner:NetworkRunner, remoteAddress:NetAddress, reason:NetConnectFailedReason) : void
OnUserSimulationMessage(runner:NetworkRunner, message:SimulationMessagePtr) : void
OnCustomAuthenticationResponse(runner:NetworkRunner, data:Dictionary<string, object>) : void
OnReliableDataReceived(runner:NetworkRunner, player:PlayerRef, data:ArraySegment<byte>) : void
OnSceneLoadDone(runner:NetworkRunner) : void
OnSceneLoadStart(runner:NetworkRunner) : void
OnHostMigration(runner:NetworkRunner, hostMigrationToken:HostMigrationToken) : void
Instance
LobbyScene
Start() : void
OnLeaveBtnClick() : void
OnCreateRoomBtnClick() : void
PlayerListPanel
Start() : void
FixedUpdate() : void
PanelController
PreparePanelController
MonoBehaviour
RoomCreatingPanel
Start() : void
OnCancelBtnClick() : void
«async» OnCreateRoomBtnClick() : void
OnDropDownChange(itemIndex:int) : void
RoomListPanel
Start() : void
UpdateRoomList(sessionList:List<SessionInfo>) : void
Next() : void
Previous() : void
RoomSettingPanel
Start() : void
RoomUnit
SetInfo(lobbyManager:LobbyManager, roomName:string, curPlayerNum:int, maxPlayerNum:int) : void
«async» OnMouseDown() : void
EnumPanel
Start,
StartSetting,
Lobby,
RoomList,
RoomCreating,
Waiting,
WaitingSetting,
MonoBehaviour
PreparePanelController
AddPanel(enumPanel:EnumPanel, panel:GameObject) : void
OpenPanel(enumPanel:EnumPanel) : void
StartScene
Start() : void
StartButton() : void
SettingButton() : void
ExitButton() : void
StartSettingPanel
Start() : void
OnLeaveBtnClick() : void
OnPlayerNameChanged() : void
WaitingRoomScene
Start() : void
OnEnable() : void
OnLeaveBtnClick() : void
OnStartBtnClick() : void
OnReadyBtnClick() : void
On100BaseBtnClick() : void
On300BaseBtnClick() : void
On500BaseBtnClick() : void
PlayerUnit
SetInfo(playerName:string, isReady:bool, points:int) : void

C. Activity Diagrams



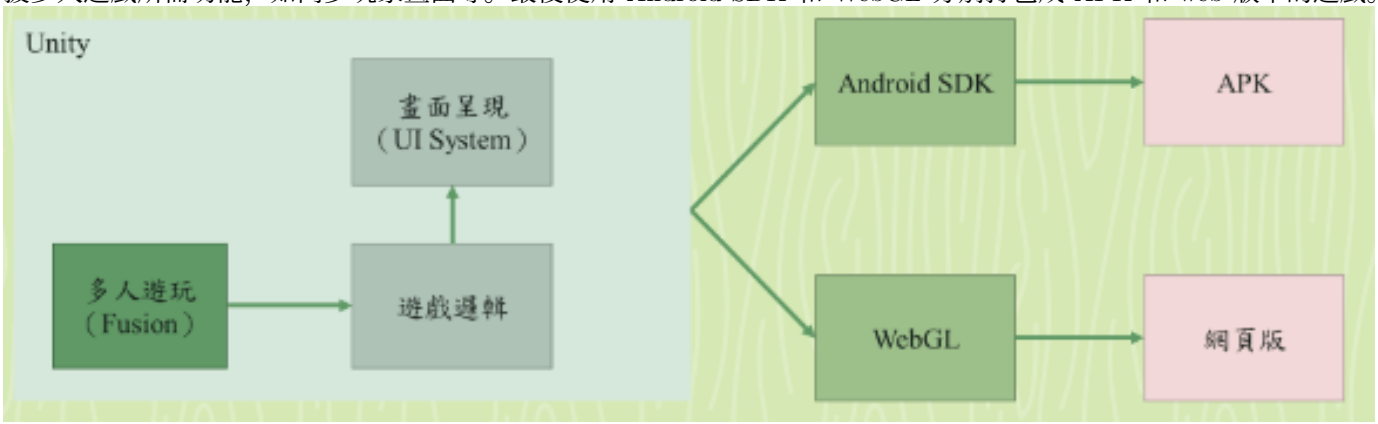
IV. Design Patterns and Object-Oriented Principles

A. Design Patterns

1) 單例模式*:* 在遊戲過程中只有一個 Game Manager 及 Loby Manager，並給其他 class 使用
2) 發布/訂閱模式*:* 我們同步玩家出牌狀況是使用 Photon Fusion 同步，底層是使用發布/訂閱模式
3) 策略模式*:* 使用策略模式處理吃碰槓三種不同情形的行動方式。

V. System architecture

專題使用 Unity 開發，遊戲邏輯自己使用 C# 實作，如 Game Manager、Table Manager 等。畫面以 Unity 的 UI 系統呈現，如 Image、Button、Layer 等功能。多人遊玩的連線使用 Fusion 實現，Fusion 為 Unity 的 Library，可支援多人遊戲所需功能，如同步玩家畫面等。最後使用 Android SDK 和 WebGL 分別打包成 APK 和 web 版本的遊戲。



VI. The degree of completion

完成度大概 80%。缺少完整的麻將行動（暗槓、加槓）和特殊牌型計算。尚無法與其他玩家進行對局。

VII. Comments

A. 遊戲流程

吃碰槓會破壞原本的打牌順序，要如何控制切換玩家的同時不出錯有點困難。特別是槓的時候還需要補牌，還有多種不同的槓的情況。一開始因為沒有想到還有加槓和暗槓的可能，在其他功能都寫完的情況下又加入這兩個可能後發現會需要打兩次牌才會切換到下一個玩家。最後因為時間因素沒辦法再去修改其他部分，我們決定拿掉這兩種可能，因為這兩個主要只會對獲勝後的計分有影響，而不會影響玩家是否胡牌（除去補牌可能剛好會進牌）。

B. 算台

由於流程上的控管困難，我們補花基本上都是立即補花，所以沒有辦法達成特殊的花胡，也就是七搶一及八仙過海，同時槓上開花也不太好完成。由於算台要考慮到所有的可能性，因此使用遞迴來達成，原本會擔心時間複雜度會不會造成遊戲卡頓，不過好險沒有。因為有太多的牌型需要判斷，算台原先的專門 class 有超過 1000 行的 code，在檢查和撰寫時都花了不少時間。

C. 連線

一開始以為連線版只需要基於單機的麻將再擴充，但後來發現有太多東西的設計都不一樣。時間因素導致無法重新設計，因此最後放棄連線對局。但因連線已經做好一部份，所以我們決定讓玩家還是可以建立房間，只是遊玩過程還是單機，當個別使用者玩完後會回到房間，並顯示該場遊戲得到多少分數。

- 網路同步化的處理是使用第三方工具去處理連線的部分，雖然開發上避免了一些連線錯誤處理，但是要熟悉這麼大的工具也花了一定的時間
- fusion 的做法是把網路處理的部分跟基本遊戲邏輯綁在一起，所以在功能實現上需要去考量當前執行的功能是誰執行，像是要同步每個遊戲中的玩家順序，host 玩家來決定順序，其他 client 玩家需要接受來自 host 玩家的同步資訊來修改本地的玩家順序，為之後的遊戲流程順利進行

Appendix A
完整版的 class diagram

https://drive.google.com/file/d/1x1uTPJS11FbPLA-CWRz6ks3FRGH6rsMK/view?usp=sharing

Appendix B
code

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

namespace Game.Music
{
    public enum EnumMusic
    {
        Start,
        Lobby,
        PlayingRoom
    }

    public class BgmController : MonoBehaviour
    {
        private Dictionary<EnumMusic, AudioClip> musicDict = new Dictionary<EnumMusic, AudioClip>();

        [SerializeField] private AudioClip startMusic;
        [SerializeField] private AudioClip lobbyMusic;
        [SerializeField] private AudioClip playingRoomMusic;

        void Start()
        {
            AddMusic(EnumMusic.Start,startMusic);
            AddMusic(EnumMusic.Lobby,lobbyMusic);
            AddMusic(EnumMusic.PlayingRoom,playingRoomMusic);
        }

        private void AddMusic(EnumMusic enumMusic, AudioClip music)
        {
            Debug.Log($"add music: {enumMusic}");
            musicDict.TryAdd(enumMusic, music);
        }

        public void PlayMusic(EnumMusic enumMusic)
        {
            AudioSource audioSource = gameObject.GetComponent<AudioSource>();
            audioSource.Stop();
            switch (enumMusic)
```

```
            {

                case EnumMusic.Start:
                    try
                    {
                        audioSource.clip = musicDict[EnumMusic.Start];
                        audioSource.Play();
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumMusic.Lobby:
                    try
                    {
                        audioSource.clip = musicDict[EnumMusic.Lobby];
                        audioSource.Play();
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumMusic.PlayingRoom:
                    try
                    {
                        audioSource.clip = musicDict[EnumMusic.PlayingRoom];
                        audioSource.Play();
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
            }
        }

        public void SetVolume(float volume)
        {
            AudioSource audioSource = gameObject.GetComponent<AudioSource>();
            audioSource.volume = volume;
            audioSource.Play();
        }
    }

}
```

```
using UnityEngine;

public class ClickSeController : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Mouse0) && PlayerPrefs.GetFloat("Click") > 0.0f)
        {
            gameObject.GetComponent<AudioSource>().Play();
        }
    }
}
```

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

namespace Game.Core
{
    public class GameManager : MonoBehaviour
    {
        public static GameManager Instance
        {
            get;
            private set;
        }

        [SerializeField] private NetworkRunner runner;

        public NetworkRunner Runner
        {
            get
            {
                if (runner == null)
```

```
                {
                    runner = gameObject.AddComponent<NetworkRunner>();
                    runner.ProvideInput = true;
                }
                return runner;
            }
        }

        public int GameBasePoint { get; set; } = 100;

        public int GameTaiPoint { get; set; } = 30;

        public int UserScore { get; set; } = 1000;

        public string PlayerName { get; set; } = "User";

        public string PlayerId { get; } = Guid.NewGuid().ToString("D");


        public event Action OnPlayerListUpdated;

        [FormerlySerializedAs("playerList")] public Dictionary<PlayerRef, PlayerNetworkData> PlayerList =
            new Dictionary<PlayerRef, PlayerNetworkData>();

        public void Disconnect()
        {
            Runner.Shutdown(false);
        }

        public void Awake()
        {
            Runner.ProvideInput = true;

            if (Instance == null)
            {
                Instance = this;
            }
            else
            {
                Destroy(gameObject);
            }

            DontDestroyOnLoad(gameObject);
        }

        public bool CheckAllPlayerIsReady()
        {
            if (!Runner.IsServer) return false;

            foreach (var playerData in PlayerList.Values)
            {
                if (!playerData.IsReady) return false;
            }

            foreach (var playerData in PlayerList.Values)
            {
                playerData.IsReady = false;
            }

            return true;
        }

        public void UpdatePlayerList()
        {
            OnPlayerListUpdated?.Invoke();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;

public class GenerateTile : MonoBehaviour
{
    public GameObject mahjongPrefab; // 麻將Prefab
    public Vector3 startPosition; // 生成起始位置
    public float spacing = 2.0f; // 麻將間距
    public Transform parent;

    void Start()
    {
        GenerateMahjong();
    }

    void GenerateMahjong()
```

```
    {
        int cardIndex = 1;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 1; j <= 9; j++)
            {
                GameObject mahjong = Instantiate(mahjongPrefab, startPosition + new Vector3(j * spacing, 0,
                     i * spacing), Quaternion.identity, parent);
                GameObject face = mahjong.transform.Find("Face").gameObject;
                Image faceImage = face.GetComponent<Image>();

                Sprite img = Resources.Load<Sprite>("Image/Mahjong/" + cardIndex.ToString());
                if (img)
                {
                    faceImage.sprite = img;
                    Debug.Log("成功設定圖像" + "Image/Mahjong/" + cardIndex.ToString());
                }
                else
                {
                    Debug.Log("無法設定圖像" + "Image/Mahjong/" + cardIndex.ToString());
                }
                mahjong.name = $"Mahjong_{i + 1}_{j}";
                ++cardIndex;
            }
        }
    }
}
```

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using Fusion;
using Fusion.Sockets;
using Game.Core;
using Utils;

namespace Game.Lobby
{
    public class LobbyManager : MonoBehaviour, INetworkRunnerCallbacks
    {
        private const int MaxRoomNum = 100;
        private readonly RoomID roomID = new();
        private GameManager gameManager;
        private SortedSet<string> roomNameSet = new();
        private PreparePanelController preparePanelController = new();

        [SerializeField] private RoomListPanel roomListPanel;
        [SerializeField] private PlayerNetworkData playerNetworkDataPrefab;

        public PreparePanelController PanelController => preparePanelController;

        public static LobbyManager Instance
        {
            get;
            private set;
        }

        public void Awake()
        {
            if (Instance != null && Instance != this)
            {
                Destroy(this);
            }
            else
            {
                Instance = this;
                DontDestroyOnLoad(this);
            }
        }

        public async void Start()
        {
            gameManager = GameManager.Instance;

            // AddCallbacks to trigger callback function
            gameManager.Runner.AddCallbacks(this);

            // TODO: user can not operate until join session completely
            await JoinLobby(gameManager.Runner);
        }

        #region - RoomRelated
```

```
        private async Task JoinLobby(NetworkRunner runner)
        {
            var result = await runner.JoinSessionLobby(SessionLobby.ClientServer);

            if (!result.Ok)
                Debug.LogError($"Fail to start: {result.ShutdownReason}");
        }

        public async Task CreateRoom(string roomName, int maxPlayerNum)
        {
            if (roomNameSet.Count <= MaxRoomNum && !roomNameSet.Contains(roomName))
            {
                // TODO: session add unique id
                var customProps = new Dictionary<string, int>(){
                    {"roomID", roomID.Get(roomName)}
                };

                // TODO: add fusion object pool
                var result = await gameManager.Runner.StartGame(new StartGameArgs()
                {
                    GameMode = GameMode.Host,
                    SessionName = roomName,
                    PlayerCount = maxPlayerNum,
                    Scene = SceneManager.GetActiveScene().buildIndex,
                    SceneManager = gameManager.gameObject.AddComponent<NetworkSceneManagerDefault>()
                });


                if (result.Ok)
                    preparePanelController.OpenPanel(EnumPanel.Waiting);
                else
                    Debug.LogError($"Failed to Start: {result.ShutdownReason}");
            }
            // TODO: display error message
        }

        public async Task JoinRoom(string roomName)
        {
            var result = await gameManager.Runner.StartGame(new StartGameArgs()
            {
                GameMode = GameMode.Client,
                SessionName = roomName,
                Scene = SceneManager.GetActiveScene().buildIndex,
                SceneManager = gameManager.gameObject.AddComponent<NetworkSceneManagerDefault>()
            });


            if (result.Ok)
                preparePanelController.OpenPanel(EnumPanel.Waiting);
            else
                Debug.LogError($"Failed to Start: {result.ShutdownReason}");
        }

        #endregion

        private void Disconnect()
        {
        }


        #region - PhontonCallBack

        public void OnSessionListUpdated(NetworkRunner runner, List<SessionInfo> sessionList)
        {
            roomNameSet.Clear();
            foreach (var session in sessionList)
            {
                roomNameSet.Add(session.Name);
            }
            roomListPanel.UpdateRoomList(sessionList);
        }

        public void OnPlayerJoined(NetworkRunner runner, PlayerRef player)
        {
            runner.Spawn(playerNetworkDataPrefab, Vector3.zero, Quaternion.identity, player);
        }

        public void OnPlayerLeft(NetworkRunner runner, PlayerRef player)
        {
            if (gameManager.PlayerList.TryGetValue(player, out PlayerNetworkData playerNetworkData))
            {
                runner.Despawn(playerNetworkData.Object);
            }
        }

        #endregion
```

```
        #region - unused callbacks
        public void OnInput(NetworkRunner runner, NetworkInput input) { }
        public void OnInputMissing(NetworkRunner runner, PlayerRef player, NetworkInput input) { }

        public void OnShutdown(NetworkRunner runner, ShutdownReason shutdownReason)
        {
            if (gameManager.PlayerList.TryGetValue(runner.LocalPlayer, out PlayerNetworkData
                playerNetworkData))
            {
                runner.Despawn(playerNetworkData.Object);
            }
            Disconnect();
        }
        public void OnConnectedToServer(NetworkRunner runner) { }

        public void OnDisconnectedFromServer(NetworkRunner runner)
        {
            runner.Shutdown(false);
        }
        public void OnConnectRequest(NetworkRunner runner, NetworkRunnerCallbackArgs.ConnectRequest request
            , byte[] token) { }
        public void OnConnectFailed(NetworkRunner runner, NetAddress remoteAddress, NetConnectFailedReason
            reason) { }
        public void OnUserSimulationMessage(NetworkRunner runner, SimulationMessagePtr message) { }
        public void OnCustomAuthenticationResponse(NetworkRunner runner, Dictionary<string, object> data) {
             }
        public void OnReliableDataReceived(NetworkRunner runner, PlayerRef player, ArraySegment<byte> data)
             { }
        public void OnSceneLoadDone(NetworkRunner runner) { }
        public void OnSceneLoadStart(NetworkRunner runner) { }
        public void OnHostMigration(NetworkRunner runner, HostMigrationToken hostMigrationToken) { }
        #endregion
    }
}
```

```
using UnityEngine;

namespace Game.Lobby
{
    public class LobbyScene : MonoBehaviour
    {
        private LobbyManager lobbyManager;

        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.Lobby, gameObject);
        }

        #region - BtnCallBack

        public void OnLeaveBtnClick()
        {
            lobbyManager.PanelController.OpenPanel(EnumPanel.Start);
        }

        public void OnCreateRoomBtnClick()
        {
            lobbyManager.PanelController.OpenPanel(EnumPanel.RoomCreating);
        }

        #endregion
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;
namespace Game.Music
{
    public class MusicToggle : MonoBehaviour
    {
        [SerializeField] private string saveType = "";
        [SerializeField] private BgmController musicController;

        void Start()
        {
            if(PlayerPrefs.HasKey(saveType) == false)
            {
                PlayerPrefs.SetFloat(saveType, 1.0f);
            }
            SetVolume();
        }

        public void OnToggleClick()
```

```
        {
            PlayerPrefs.SetFloat(saveType, gameObject.GetComponent<Toggle>().isOn ? 1.0f : 0.0f);
            SetVolume();
        }

        private void SetVolume()
        {
            if(musicController != null)
            {
                musicController.SetVolume(PlayerPrefs.GetFloat(saveType));
            }
            gameObject.GetComponent<Toggle>().isOn = (PlayerPrefs.GetFloat(saveType) > 0.0f);
        }
    }
}
```

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

namespace Game.Play {
    public class Player : MonoBehaviour
    {
        private readonly List<GameObject> handTiles = new();
        private readonly List<GameObject> showTiles = new();

        [SerializeField] private Transform hand;
        [SerializeField] private Transform tilePool;
        [SerializeField] private Transform showPool;

        [SerializeField] private GameObject chiTileSets;
        [SerializeField] private Transform chiTileSetsTransform;

        [SerializeField] private GameObject setPrefab;

        private TaiCalculator taiCalculator = new(null, null);
        private readonly List<List<GameObject>> canChiTileSet = new();

        private int kongCnt;
        private int ponCnt;
        private int straightCnt;

        private bool isSelfDraw;
        private readonly bool isDealer = true;
        private bool isOnly;

        public readonly IProcessStrategy Chi;
        public readonly IProcessStrategy Pong;
        public readonly IProcessStrategy Kong;

        public Player()
        {
            Chi = new ChiStrategy(this);
            Pong = new PongStrategy(this);
            Kong = new KongStrategy(this);
        }

        public TableManager TableManager { get; set; }

        public int PlayerIndex { set; get; } = -1;

        public Transform Hand => hand;

        public Transform TilePool => tilePool;

        public Transform ShowPool => showPool;

        public TaiCalculator TaiCalculator => taiCalculator;

        public int Point { get; set; }

        public bool IsSelfDraw => isSelfDraw;

        private int HandTilesCnt => handTiles.Count;


        #region - TileOperations

        public void CallCalculator()
        {
            if (!isSelfDraw)
            {
                showTiles.Add(TableManager.LastTile);
            }
```

```
            taiCalculator = new TaiCalculator(handTiles, showTiles, kongCnt, ponCnt, straightCnt, 1, 1, 0,
                0,
                isDealer, false, false, isSelfDraw, false, isOnly);
        }

        public bool ReplaceFlower()
        {
            int cnt = 0;
            for(int i = handTiles.Count-1; i >= 0; --i)
            {
                if (handTiles[i].GetComponent<Tile>().IsFlower())
                {
                    ShowTile(i);
                    ++cnt;
                }
            }

            for(int i = 0; i < cnt; ++i)
            {
                TableManager.TileWall.Replenish(this);
            }

            return cnt != 0;
        }

        public bool Discard(string tileId)
        {
            if (TableManager.ActivePlayerIndex != PlayerIndex)
                return false;

            int idx = FindTileByTileId(handTiles, tileId);

            if (idx != -1)
            {
                TableManager.LastTile = handTiles[idx];
                handTiles[idx].transform.SetParent(tilePool, false);
                handTiles[idx].SetActive(true);
                handTiles.RemoveAt(idx);
            }
            else
            {
                Debug.LogError("Cannot find tile from Tile ID");
                return false;
            }

            SortHandTiles();
            StartCoroutine(TableManager.BeforeNextPlayer());
            return true;
        }

        public void DefaultDiscard()
        {
            Discard(handTiles[Random.Range(0, HandTilesCnt)].GetComponent<Tile>().TileId);
            // Discard(handTiles[handTiles.Count-1].GetComponent<Tile>().TileId);
        }

        public void SortHandTiles()
        {
            handTiles.Sort(new TileGameObjectComparer());
            foreach(GameObject t in handTiles)
            {
                t.transform.SetAsLastSibling();
            }
        }
        public void GetTile(GameObject tile)
        {
            handTiles.Add(tile);
            tile.GetComponent<Tile>().Player = this;
            tile.transform.SetParent(hand, false);
            tile.SetActive(true);
        }

        void ShowTile(int indexOfHandTiles)
        {
            handTiles[indexOfHandTiles].transform.SetParent(showPool, false);
            showTiles.Add(handTiles[indexOfHandTiles]);
            handTiles.RemoveAt(indexOfHandTiles);
        }

        void TakeTileFromOther()
        {
            TableManager.LastTile.transform.SetParent(showPool, false);
            showTiles.Add(TableManager.LastTile);
        }

        #endregion
```

```
#region - ChinPongGangHu

public interface IProcessStrategy
{
    public bool CanDoOperation();
    public void DoOperation();
}

private class ChiStrategy: IProcessStrategy
{
    private readonly Player player;

    public ChiStrategy(Player player)
    {
        this.player = player;
    }
    public bool CanDoOperation()
    {
        if (player.TableManager.ActivePlayerIndex == player.PlayerIndex)
            return false;
        if ((player.TableManager.ActivePlayerIndex + 1) % 4 != player.PlayerIndex)
            return false;
        player.canChiTileSet.Clear();
        Tile newTile = player.TableManager.LastTile.GetComponent<Tile>();
        // Debug.Log(newTile.TileType.ToString() + newTile.TileNumber.ToString());
        bool isCanChi = false;
        if (!newTile.IsSuit())
            return false;

        int i = newTile.TileNumber;
        // i-2, i-1, i
        isCanChi |= player.Find(newTile.TileType, i-2, i-1);
        // i-1, i, i+1
        isCanChi |= player.Find(newTile.TileType, i-1, i+1);
        // i, i+1, i+2
        isCanChi |= player.Find(newTile.TileType, i+1, i+2);

        return isCanChi;
    }

    public void DoOperation()
    {
        GameObject set = UnityEngine.EventSystems.EventSystem.current.currentSelectedGameObject;
        Transform[] children = set.GetComponentsInChildren<Transform>();

        bool first = true;
        foreach (Transform child in children)
        {
            if (child == set.transform) continue;
            string tileId = child.name;

            int idx = player.FindTileByTileId(player.handTiles, tileId, "REVERSE");

            if (idx != -1)
            {
                player.ShowTile(idx);
                if (first)
                {
                    player.TakeTileFromOther();
                    first = false;
                }
            }
        }

        CloseChiSelection();
        player.straightCnt++;
    }

    private void CloseChiSelection()
    {
        player.chiTileSets.SetActive(false);
        Transform[] children = player.chiTileSets.GetComponentsInChildren<Transform>();
        foreach (Transform child in children)
        {
            if (child == player.chiTileSets.transform) continue;
            Destroy(child.gameObject);
        }
    }
}

private class PongStrategy : IProcessStrategy
{
    private readonly Player player;

    public PongStrategy(Player player)
```

```
        {
            this.player = player;
        }
        public bool CanDoOperation()
        {
            if (player.TableManager.ActivePlayerIndex == player.PlayerIndex)
                return false;
            bool isCanPong = false;
            int numSame = 0;
            foreach(GameObject tile in player.handTiles)
            {
                if (player.TableManager.LastTile && tile.GetComponent<Tile>().Equals(player.
                    TableManager.LastTile.GetComponent<Tile>()))
                    ++numSame;
            }
            if (numSame >= 2)
            {
                isCanPong = true;

            }
            return isCanPong;
        }

        public void DoOperation()
        {
            player.TakeTileFromOther();
            int cnt = 0;
            for(int i = player.handTiles.Count-1; i >= 0; --i)
            {
                if (player.handTiles[i].GetComponent<Tile>().Equals(player.TableManager.LastTile.
                    GetComponent<Tile>()))
                {
                    ++cnt;
                    player.ShowTile(i);
                }
                if (cnt == 2)
                    break;
            }
            player.ponCnt++;
        }
    }

    private class KongStrategy : IProcessStrategy
    {
        private readonly Player player;

        public KongStrategy(Player player)
        {
            this.player = player;
        }
        public bool CanDoOperation()
        {
            if (((player.TableManager.ActivePlayerIndex + 1) % 4 == player.PlayerIndex) || player.
                TableManager.LastTile.GetComponent<Tile>().PlayerIndex == player.PlayerIndex)
            {
                return false;
            }
            bool isCanGang = false;
            int numSame = 0;
            foreach(GameObject tile in player.handTiles)
            {
                if (player.TableManager.LastTile && tile.GetComponent<Tile>().Equals(player.
                    TableManager.LastTile.GetComponent<Tile>()))
                    ++numSame;
            }
            if (numSame == 3)
            {
                isCanGang = true;

            }
            return isCanGang;
        }

        public void DoOperation()
        {
            player.TakeTileFromOther();
            int cnt = 0;
            for(int i = player.handTiles.Count-1; i >= 0; --i)
            {
                if (player.handTiles[i].GetComponent<Tile>().Equals(player.TableManager.LastTile.
                    GetComponent<Tile>()))
                {
                    ++cnt;
                    player.ShowTile(i);
                }
                if (cnt == 3)
```

```
                break;
            }
            player.kongCnt++;
        }
    }

    public bool IsPlayerCanHu(bool selfDraw)
    {
        int[] tileCountArray = new int[50];
        TransToArray(tileCountArray, handTiles);

        isSelfDraw = selfDraw;

        if (!isSelfDraw)
        {
            this.isOnly = CheckIsOnly((int[])tileCountArray.Clone());
            GameObject[] lastTileArray = {TableManager.LastTile};
            List<GameObject> lastTile = new List<GameObject>(lastTileArray);
            TransToArray(tileCountArray, lastTile);
        }

        return CheckHu((int[])tileCountArray.Clone(), false);
    }

    #endregion

    #region - HelperFunction

    private int FindTileByTileId(List<GameObject> tiles, string tileId, string mode = "NORMAL")
    {
        if (mode == "NORMAL")
        {
            for (int i = 0; i < tiles.Count; ++i)
            {
                if (tiles[i].GetComponent<Tile>().TileId == tileId)
                {
                    return i;
                }
            }
        }
        else if(mode == "REVERSE")
        {
            for (int i = tiles.Count - 1; i >= 0; --i)
            {
                if (tiles[i].GetComponent<Tile>().TileId == tileId)
                {
                    return i;
                }
            }
        }

        return -1;
    }

    private bool CheckIsOnly(int[] nowHandArray)
    {
        int cnt = 0;
        for (int i = 1; i <= 37; i++)
        {
            if (i == 10 || i == 20 || i == 30)
                continue;

            nowHandArray[i]++;
            if (CheckHu((int[])nowHandArray.Clone() ,false))
            {
                cnt ++;
            }
            nowHandArray[i]--;
        }

        return cnt == 1;
    }

    // recursion
    private static bool CheckHu(int[] nowTileArray, bool havePair)
    {
        if(nowTileArray.Sum() == 0)
        {
            return true;
        }

        for(int i = 1; i <= 37; i++)
        {
            if(nowTileArray[i] == 0)
                continue;
```

```
            // have pair
            if (havePair)
            {
                // check straight
                if(nowTileArray[i] > 0 && nowTileArray[i - 1] > 0 && nowTileArray[i + 1] > 0)
                {
                    nowTileArray[i - 1] -= 1;
                    nowTileArray[i] -= 1;
                    nowTileArray[i + 1] -= 1;
                    if (CheckHu((int[])nowTileArray.Clone(), true))
                    {
                        return true;
                    }
                    nowTileArray[i - 1] += 1;
                    nowTileArray[i] += 1;
                    nowTileArray[i + 1] += 1;
                }
                // check pon
                if(nowTileArray[i] >= 3)
                {
                    nowTileArray[i] -= 3;
                    if (CheckHu((int[])nowTileArray.Clone(), true))
                    {
                        return true;
                    }
                    nowTileArray[i] += 3;
                }
            }
            // no pair yet
            else
            {
                if(nowTileArray[i] >= 2)
                {
                    nowTileArray[i] -= 2;
                    if (CheckHu((int[])nowTileArray.Clone(), true))
                    {
                        return true;
                    }
                    nowTileArray[i] += 2;
                }
            }
        }
        return false;
    }

    private void TransToArray(int[] tileCountArray, List<GameObject> tileList)
    {
        TaiCalculator.TransToArray(tileCountArray, tileList);
    }


    private bool Find(TileType tileType, int lostA, int lostB)
    {
        if (lostA < 1 || lostB > 9)
            return false;
        GameObject a = handTiles.Find(t => t.GetComponent<Tile>().TileType == tileType &&
                                           t.GetComponent<Tile>().TileNumber == lostA);
        GameObject b = handTiles.Find(t => t.GetComponent<Tile>().TileType == tileType &&
                                           t.GetComponent<Tile>().TileNumber == lostB);


        if (a != null && b != null)
        {
            List<GameObject> set = new List<GameObject>
            {
                a,
                b
            };
            canChiTileSet.Add(set);
            return true;
        }
        return false;
    }

    #endregion

    #region - BtnCallBack

    public void DecideHowToChi()
    {
        chiTileSets.SetActive(true);
        foreach (var t in canChiTileSet)
        {
            GameObject set = Instantiate(setPrefab, new Vector3(0, 0, 0), Quaternion.identity,
                chiTileSetsTransform);
            set.SetActive(true);
```

```
                GameObject a = Instantiate(t[0], set.transform, false);
                a.name = t[0].GetComponent<Tile>().TileId;
                GameObject b = Instantiate(t[1], set.transform, false);
                b.name = t[1].GetComponent<Tile>().TileId;
            }
        }

        public void ChiBtnCallBack()
        {
            chiTileSets.SetActive(false);
            Chi.DoOperation();
        }

        public void PongBtnCallBack()
        {
            Pong.DoOperation();
        }

        public void KongBtnCallBack()
        {
            Kong.DoOperation();
        }

        #endregion
    }

}
```

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

namespace Game.Lobby
{
    public class PlayerListPanel : MonoBehaviour
    {
        private GameManager gameManager;
        private List<PlayerUnit> playerUnits = new();

        [SerializeField] private Transform contentTrans;
        [SerializeField] private PlayerUnit playerUnitPrefab;

        public void Start()
        {
            gameManager = GameManager.Instance;
        }

        public void FixedUpdate()
        {
            // not require high frequency to update
            UpdatePlayerList();
        }

        private void UpdatePlayerList()
        {
            foreach (var unit in playerUnits)
            {
                Destroy(unit.gameObject);
            }

            playerUnits.Clear();

            foreach (var player in gameManager.PlayerList)
            {
                var unit = Instantiate(playerUnitPrefab, contentTrans);
                var playerData = player.Value;

                unit.SetInfo(playerData.PlayerName, playerData.IsReady, playerData.UserScore);
                playerUnits.Add(unit);
            }
        }

    }
}
```

```
using Fusion;

namespace Game.Core
{
    public class PlayerNetworkData : NetworkBehaviour
    {
        private GameManager gameManager = null;

        [Networked] public int UserScore { get; set; } = 0;
        [Networked] public string PlayerId { get; set; }
```

```
        [Networked] public string PlayerName { get; set; }

        [Networked(OnChanged = nameof(OnIsReadyChanged))] public NetworkBool IsReady { get; set; }

        public override void Spawned()
        {
            gameManager = GameManager.Instance;

            transform.SetParent(GameManager.Instance.transform);

            gameManager.PlayerList.Add(Object.InputAuthority, this);
            gameManager.UpdatePlayerList();

            if (Object.HasInputAuthority)
            {
                SetPlayerId_RPC(gameManager.PlayerId);
                UpdateScore_RPC(gameManager.UserScore);
                SetPlayerName_RPC(gameManager.PlayerName);
            }
        }

        public override void Despawned(NetworkRunner runner, bool hasState)
        {
            gameManager.PlayerList.Remove(Object.InputAuthority);
            gameManager.UpdatePlayerList();
        }

        [Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.StateAuthority)]
        public void SetPlayerId_RPC(string playerId)
        {
            PlayerId = playerId;
        }

        [Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.StateAuthority)]
        public void SetPlayerName_RPC(string playerName)
        {
            PlayerName = playerName;
        }

        [Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.StateAuthority)]
        public void UpdateScore_RPC(int score)
        {
            UserScore += score;
        }

        [Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.StateAuthority)]
        public void SetReady_RPC(bool isReady)
        {
            IsReady = isReady;
        }

        private static void OnIsReadyChanged(Changed<PlayerNetworkData> changed)
        {
            GameManager.Instance.UpdatePlayerList();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;

namespace Game.Lobby
{
    public class PlayerUnit : MonoBehaviour
    {
        [SerializeField] private Text playerNameTxt;
        [SerializeField] private Text isReadyTxt;
        [SerializeField] private Text pointsTxt;


        public void SetInfo(string playerName, bool isReady, int points)
        {
            playerNameTxt.text = playerName;
            isReadyTxt.text = isReady ? "Ready" : "Not Ready";
            pointsTxt.text = points.ToString();
        }
    }
}
```

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

namespace Game.Lobby
```

```
{
    public enum EnumPanel
    {
        Start,
        StartSetting,
        Lobby,
        RoomList,
        RoomCreating,
        Waiting,
        WaitingSetting
    }

    public class PreparePanelController
    {
        private Dictionary<EnumPanel, GameObject> panelDict = new Dictionary<EnumPanel, GameObject>();

        public void AddPanel(EnumPanel enumPanel, GameObject panel)
        {
            Debug.Log($"add panel: {enumPanel}");
            panelDict.TryAdd(enumPanel, panel);
        }

        public void ClearPanel()
        {
            panelDict.Clear();
        }

        public void OpenPanel(EnumPanel enumPanel)
        {
            foreach (var content in panelDict)
                content.Value.SetActive(false);

            switch (enumPanel)
            {
                case EnumPanel.Start:
                    try
                    {
                        panelDict[EnumPanel.Start].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumPanel.StartSetting:
                    try
                    {
                        // panelDict[EnumPanel.Start].SetActive(true);
                        panelDict[EnumPanel.StartSetting].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumPanel.Lobby:
                    try
                    {
                        panelDict[EnumPanel.Lobby].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumPanel.RoomList:
                    try
                    {
                        panelDict[EnumPanel.Lobby].SetActive(true);
                        panelDict[EnumPanel.RoomList].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
                case EnumPanel.RoomCreating:
                    try
                    {
                        panelDict[EnumPanel.Lobby].SetActive(true);
                        panelDict[EnumPanel.RoomCreating].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
```

```
                    break;
                case EnumPanel.Waiting:
                    try
                    {
                        panelDict[EnumPanel.Waiting].SetActive(true);
                        // if current user is host, it will show game setting panel
                        if (GameManager.Instance.Runner.IsServer)
                            panelDict[EnumPanel.WaitingSetting].SetActive(true);
                    }
                    catch (ArgumentException error)
                    {
                        Debug.Log(error);
                    }
                    break;
            }
        }
    }
}
```

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

namespace Game.Lobby
{
    public class RoomCreatingPanel : MonoBehaviour
    {
        private LobbyManager lobbyManager;
        [SerializeField] private Dropdown dropDown;
        [SerializeField] private InputField roomNameInputField;
        [SerializeField] private Music.BgmController musicController;

        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.RoomCreating, gameObject);

            List<int> valueList = new List<int>();
            for (int i = 1; i <= 4; ++i)
                valueList.Add(i);
            SetDropdownOptions(valueList);
        }

        #region - BtnCallBack

        public void OnCancelBtnClick()
        {
            musicController.PlayMusic(Music.EnumMusic.Start);
            lobbyManager.PanelController.OpenPanel(EnumPanel.RoomList);
        }

        public async void OnCreateRoomBtnClick()
        {
            Debug.Log(roomNameInputField.text);
            await lobbyManager.CreateRoom(roomNameInputField.text, 4);
            musicController.PlayMusic(Music.EnumMusic.Lobby);
        }

        #endregion

        #region - Helper Function

        private void SetDropdownOptions(List<int> valueList)
        {
            if (valueList.Count > 0)
            {
                dropDown.ClearOptions();
                foreach (int value in valueList)
                {
                    Dropdown.OptionData data = new Dropdown.OptionData
                    {
                        text = value.ToString()
                    };
                    dropDown.options.Add(data);
                }
            }
        }

        public void OnDropDownChange(int itemIndex)
        {
            if (itemIndex >= dropDown.options.Count)
                itemIndex = dropDown.options.Count - 1;
            if (itemIndex < 0)
                itemIndex = 0;
            dropDown.value = itemIndex;
        }
```

```
        #endregion
    }
}
```

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

namespace Game.Lobby
{
    public class RoomListPanel : MonoBehaviour
    {
        [SerializeField] private Text panelText;
        [SerializeField] private Transform hideTrans;
        [SerializeField] private RoomUnit roomUnitPrefab;
        [SerializeField] private List<Transform> parentPositions = new List<Transform>();
        private int pageIndex = 1;
        private int pageCount;
        private int roomCount;
        private LobbyManager lobbyManager;
        private readonly List<RoomUnit> roomList = new List<RoomUnit>();

        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.RoomList, gameObject);
        }

        public void UpdateRoomList(List<SessionInfo> sessionList)
        {
            foreach (Transform parentPos in parentPositions)
            {
                if (parentPos.childCount > 0)
                    Destroy(parentPos.GetChild(0).gameObject);
            }

            foreach (Transform child in hideTrans)
            {
                Destroy(child.gameObject);
            }

            roomList.Clear();
            foreach (var session in sessionList)
            {
                RoomUnit roomUnit = Instantiate(roomUnitPrefab, hideTrans);
                roomUnit.GetComponent<RoomUnit>().SetInfo(lobbyManager, session.Name, session.PlayerCount,
                    session.MaxPlayers);
                roomList.Add(roomUnit);
            }

            roomCount = roomList.Count;
            pageCount = (roomCount % 8) == 0 ? roomCount / 8 : (roomCount / 8) + 1;
            BindPage(pageIndex);
            panelText.text = $"{pageIndex.ToString()}/{pageCount.ToString()}";

            Debug.Log($"room count: {roomCount}");
        }

        public void Next()
        {
            pageIndex += 1;
            UpdatePage();
        }

        public void Previous()
        {
            pageIndex -= 1;
            UpdatePage();
        }

        private void UpdatePage()
        {
            pageIndex = Mathf.Max(1, pageIndex);
            pageIndex = Mathf.Min(pageIndex, pageCount);

            BindPage(pageIndex);
            panelText.text = $"{pageIndex.ToString()}/{pageCount.ToString()}";
        }

        private void BindPage(int index)
        {
            if (roomList == null || roomCount <= 0) return;
            if (index < 0 || index > roomCount) return;
```

```
            for (int i = 0; i < roomCount; i++)
            {
                roomList[i].transform.SetParent(hideTrans);
            }


            for (int i = 0, j = 8 * (index - 1); j < Mathf.Min(8 * index, roomCount); i++, j++)
            {
                roomList[j].transform.SetParent(parentPositions[i]);
                roomList[j].transform.localPosition = new Vector3(0, 0, 0);
            }
        }
    }
}
```

```
using UnityEngine;

namespace Game.Lobby
{
    public class RoomSettingPanel : MonoBehaviour
    {
        public void Start()
        {
            LobbyManager.Instance.PanelController.AddPanel(EnumPanel.WaitingSetting, gameObject);
        }
    }
}
```

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;

namespace Game.Lobby
{
    public class RoomUnit : MonoBehaviour
    {
        private LobbyManager lobbyManager;

        [SerializeField] private Text roomName;
        [SerializeField] private TextMeshProUGUI curPlayerNum;
        [SerializeField] private TextMeshProUGUI maxPlayerNum;

        public void SetInfo(LobbyManager lobbyManager, string roomName, int curPlayerNum, int maxPlayerNum)
        {
            this.lobbyManager = lobbyManager;
            this.roomName.text = roomName;
            this.curPlayerNum.text = curPlayerNum.ToString();
            this.maxPlayerNum.text = maxPlayerNum.ToString();
        }

        public async void OnMouseDown()
        {
            await lobbyManager.JoinRoom(roomName.text);
        }
    }
}
```

```
using UnityEngine;

namespace Game.Lobby
{
    public class StartScene : MonoBehaviour
    {
        private LobbyManager lobbyManager;
        [SerializeField] private Music.BgmController musicController;

        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.Start, gameObject);
            lobbyManager.PanelController.OpenPanel(EnumPanel.Start);
            musicController.PlayMusic(Music.EnumMusic.Start);
        }

        #region - BtnCallBack
        public void StartButton()
        {
            lobbyManager.PanelController.OpenPanel(EnumPanel.RoomList);
        }

        public void SettingButton()
        {
            lobbyManager.PanelController.OpenPanel(EnumPanel.StartSetting);
        }
```

```
        public void ExitButton()
        {
            Application.Quit();
        }
        #endregion
    }
}
```

```
using Game.Core;
using UnityEngine;
using UnityEngine.UI;

namespace Game.Lobby
{
    public class StartSettingPanel : MonoBehaviour
    {
        [SerializeField] private InputField playerName;
        private LobbyManager lobbyManager;
        // Start is called before the first frame update
        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.StartSetting, gameObject);
            // Debug.Log("StartSetting");
        }

        #region - BtnCallBack

        public void OnLeaveBtnClick()
        {
            lobbyManager.PanelController.OpenPanel(EnumPanel.Start);
        }

        public void OnPlayerNameChanged()
        {
            GameManager.Instance.PlayerName = playerName.text;
        }

        #endregion
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Game.Core;
using Game.Lobby;
using UnityEngine.SceneManagement;

namespace Game.Play
{

    public class TableManager : MonoBehaviour
    {
        public static TableManager Instance => null;

        [SerializeField] private List<Player> players = new List<Player>();
        [SerializeField] private TileWall tileWall;

        private int activePlayerIndex;

        private GameObject lastTile = null;

        private GameManager gameManager;

        [SerializeField] private GameObject winningBtn, chiBtn, pongBtn, kongBtn;
        [SerializeField] private GameObject roundPoints;
        [SerializeField] private Text points;
        [SerializeField] private List<Text> pointsChangeText;
        [SerializeField] private List<Text> winningType;

        private List<GameObject> buttons;

        public TileWall TileWall => tileWall;

        public GameObject LastTile { set; get; }

        public int ActivePlayerIndex => activePlayerIndex;


        private bool chiActive;
        private bool pongActive;
        private bool kongActive;
```

```
        private bool huActive;


        void Start()
        {
            gameManager = GameManager.Instance;
            tileWall.GetReady();
            // todo: player will set player id, now is 0-3
            for(int i = 0; i < 4; i++)
            {
                players[i].PlayerIndex = i;
                players[i].TableManager = this;
            }

            buttons = new List<GameObject>() { winningBtn, chiBtn, kongBtn, pongBtn };

            // PickSeatsAndDecideDealer();

            DealTiles();
            OpenDoor();
        }

        #region - GameLogic

        void OpenDoor()
        {
            tileWall.DealTile(players[activePlayerIndex]);
            int cnt = 0;

            while(cnt < 4)
            {
                foreach (Player player in players)
                {
                    cnt += player.ReplaceFlower() ? 0 : 1;
                }
            }

            foreach(Player player in players)
            {
                player.SortHandTiles();
            }
        }

        void DealTiles()
        {
            for (int round = 0; round < 4; ++round)
            {
                for (int playerIndex = 0; playerIndex < 4; ++playerIndex)
                {
                    for (int i = 0; i < 4; ++i)
                    {
                        tileWall.DealTile(players[playerIndex]);
                    }
                }
            }
        }

        void PickSeatsAndDecideDealer()
        {
            for(int i = 0; i < players.Count; ++i)
            {
                int j = Random.Range(0, players.Count);

                (players[i], players[j]) = (players[j], players[i]);
            }
            activePlayerIndex = 0;
        }

        private void Draw()
        {
            if(!tileWall.DealTile(players[activePlayerIndex]))
            {
                EndGame(-1);
            }

            while (players[activePlayerIndex].ReplaceFlower()) { }

            if (activePlayerIndex == 0 && players[0].Kong.CanDoOperation())
            {
                SetButton(kongBtn, true);
            }

            if (activePlayerIndex == 0 && players[0].IsPlayerCanHu(true))
            {
                SetButton(winningBtn, true);
            }
```

```
        }

        private void BuKong()
        {
            tileWall.Replenish(players[activePlayerIndex]);
            while(players[activePlayerIndex].ReplaceFlower()) { }
        }

        #endregion

        #region - BtnCallBack

        public void Chi()
        {
            chiActive = true;
        }
        public void Pong()
        {
            pongActive = true;
        }
        public void Kong()
        {
            kongActive = true;
        }
        public void Win()
        {
            if (gameManager.PlayerList.TryGetValue(gameManager.Runner.LocalPlayer, out var
                playerNetworkData))
            {
                playerNetworkData.SetReady_RPC(false);
            }

            huActive = true;
            int winningPlayerIndex = 0;
            TurnToPlayer(winningPlayerIndex);
            EndGame(winningPlayerIndex);
        }

        public void OnRoundPointClose()
        {
            int curSceneIndex = SceneManager.GetActiveScene().buildIndex;
            LobbyManager.Instance.PanelController.ClearPanel();
            gameManager.Runner.SetActiveScene(curSceneIndex - 1);
            LobbyManager.Instance.PanelController.OpenPanel(EnumPanel.Waiting);
        }

        #endregion

        #region - HelperFunction

        IEnumerator Countdown(int second)
        {
            yield return new WaitForSeconds(second);
        }

        private void NextPlayer()
        {
            activePlayerIndex = (activePlayerIndex + 1) % 4;
        }

        private void TurnToPlayer(int playerIndex)
        {
            activePlayerIndex = playerIndex;
        }
        //  only from single player


        private void AutoPlay()
        {
            if(activePlayerIndex != 0)
            {
                players[activePlayerIndex].DefaultDiscard();
            }
        }

        private void SetButton(GameObject btn, bool isInteractable)
        {
            btn.GetComponent<Button>().interactable = isInteractable;
        }
        private void EndGame(int winningPlayerIndex)
        {
            roundPoints.SetActive(true);
            InitScoringList();
            if(winningPlayerIndex == -1)
            {
                points.text = "流局";
```

```
            foreach(Text obj in pointsChangeText)
            {
                obj.text = "+0";
            }
            huActive = true;
        }
        else
        {
            players[winningPlayerIndex].CallCalculator();

            points.text = "共 " + players[winningPlayerIndex].TaiCalculator.Tai + " 台";
            int pointsChange = gameManager.GameBasePoint + gameManager.GameTaiPoint * players[
                winningPlayerIndex].TaiCalculator.Tai;

            if (players[winningPlayerIndex].IsSelfDraw)
            {
                pointsChangeText[winningPlayerIndex].text = "+" + (pointsChange * 3).ToString();
                pointsChangeText[(winningPlayerIndex + 1) % 4].text = "-" + pointsChange.ToString();
                pointsChangeText[(winningPlayerIndex + 2) % 4].text = "-" + pointsChange.ToString();
                pointsChangeText[(winningPlayerIndex + 3) % 4].text = "-" + pointsChange.ToString();

                if (gameManager.PlayerList.TryGetValue(gameManager.Runner.LocalPlayer, out var
                    playerNetworkData))
                {
                    playerNetworkData.UpdateScore_RPC(pointsChange * 3);
                }
            }
            else
            {
                for (int i = 0; i < 4; i++)
                {
                    if (LastTile.GetComponent<Tile>().PlayerIndex == i)
                    {
                        pointsChangeText[i].text = "-" + pointsChange.ToString();
                    }
                    else
                    {
                        pointsChangeText[i].text = "+0";
                    }
                }
                pointsChangeText[winningPlayerIndex].text = "+" + pointsChange.ToString();

                if (gameManager.PlayerList.TryGetValue(gameManager.Runner.LocalPlayer, out var
                    playerNetworkData))
                {
                    playerNetworkData.UpdateScore_RPC(pointsChange);
                }
            }

            List<string> scoringList = players[winningPlayerIndex].TaiCalculator.ScoringList;

            for (int i = 0; i < scoringList.Count; i++)
            {
                winningType[i].text = scoringList[i];
            }

        }
    }

    private void InitScoringList()
    {
        foreach(Text obj in winningType)
        {
            obj.text = "";
        }
    }

    public IEnumerator BeforeNextPlayer()
    {
        // only control local player's button
        if (players[0].Chi.CanDoOperation())
        {
            SetButton(chiBtn, true);
        }
        if (players[0].Pong.CanDoOperation())
        {
            SetButton(pongBtn, true);
        }
        if (players[0].Kong.CanDoOperation())
        {
            SetButton(kongBtn, true);
        }
        if (players[0].IsPlayerCanHu(false))
        {
            SetButton(winningBtn, true);
        }
```

```
            yield return new WaitForSeconds(2f);

            buttons.ForEach((button) => {SetButton(button, false);});

            // todo: need to deal multiplayer move
            if (huActive)
            {

            }
            else if (kongActive)
            {
                TurnToPlayer(0);
                kongActive = false;
                BuKong();
            }
            else if (pongActive)
            {
                TurnToPlayer(0);
                pongActive = false;
            }
            else if (chiActive)
            {
                TurnToPlayer(0);
                chiActive = false;
            }
            else
            {
                NextPlayer();
                Draw();
                AutoPlay();
            }

        }
        #endregion
    }
}
```

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

namespace Game.Play
{
    public class TaiCalculator
    {
        private readonly List<GameObject> handTilesList;
        private readonly List<GameObject> showTilesList;
        private List<string> scoringList = new List<string>();
        private readonly int deckGangCnt;
        private readonly int deckPonCnt;
        private readonly int deckStraightCnt;
        private readonly int faceWind; //門風 1東2南3西4北
        private readonly int courtWind; //場風 1東2南3西4北
        private readonly int hideGangCnt;
        private readonly int dealerWinStreak;
        private readonly bool isDealer;
        private readonly bool isFirstTile;
        private readonly bool isLastTile;
        private readonly bool isSelfDraw;
        private readonly bool isAfterGang;
        private readonly bool isOnly;
        private int tai;

        public List<string> ScoringList => scoringList;

        public int Tai => tai;

        public TaiCalculator(
            List<GameObject> handTilesList,
            List<GameObject> showTilesList,
            int deckGangCnt = 0,
            int deckPonCnt = 0,
            int deckStraightCnt = 0,
            int faceWind = 0,
            int courtWind = 0,
            int hideGangCnt = 0,
            int dealerWinStreak = 0,
            bool isDealer = false,
            bool isFirstTile = false,
            bool isLastTile = false,
            bool isSelfDraw = false,
            bool isAfterGang = false,
            bool isOnly = false)
        {
```

```
            int[] tileCountArray = new int[50]; // 1~9: 萬、11~19: 筒、21~29: 條、31~37: 東南西北中發白、41
                ~48: 春夏秋冬梅蘭竹菊

            this.handTilesList = handTilesList;
            this.showTilesList = showTilesList;
            this.deckGangCnt = deckGangCnt;
            this.deckPonCnt = deckPonCnt;
            this.deckStraightCnt = deckStraightCnt;
            this.faceWind = faceWind;
            this.courtWind = courtWind;
            this.hideGangCnt = hideGangCnt;
            this.dealerWinStreak = dealerWinStreak;
            this.isDealer = isDealer;
            this.isFirstTile = isFirstTile;
            this.isLastTile = isLastTile;
            this.isSelfDraw = isSelfDraw;
            this.isAfterGang = isAfterGang;
            this.isOnly = isOnly;

            if (handTilesList != null)
            {
                tileCountArray = TransToArray((int[])tileCountArray.Clone(), handTilesList);
                //recursion
                FindHighestTai((int[])tileCountArray.Clone(), false, 0);
            }
        }

        public static int[] TransToArray(int[] tileCountArray, List<GameObject> tileList)
        {
            foreach (var tile in tileList)
            {
                int offset = 0;
                switch (tile.GetComponent<Tile>().TileType)
                {
                    case TileType.Character:
                        offset = 0; // 1~9 萬
                        break;
                    case TileType.Dot:
                        offset = 10; // 11~19 筒
                        break;
                    case TileType.Bamboo:
                        offset = 20; // 21~29 條
                        break;
                    case TileType.Wind:
                        offset = 30; // 31~34 東南西北
                        break;
                    case TileType.Dragon:
                        offset = 34; // 35~37 中發白
                        break;
                    case TileType.Season:
                        offset = 40; // 41~44 春夏秋冬
                        break;
                    case TileType.Flower:
                        offset = 44; // 45~48 梅蘭竹菊
                        break;
                    default:
                        Debug.Log("Error tile type: " + tile.GetComponent<Tile>().TileType);
                        break;
                }

                tileCountArray[offset + tile.GetComponent<Tile>().TileNumber] += 1;
            }

            return tileCountArray;
        }

        #region Helper Function

        private void FindHighestTai(int[] nowTileArray, bool havePair, int ponCnt)
        {
            if((isSelfDraw && nowTileArray.Sum() == 0) || ((!isSelfDraw) && nowTileArray.Sum() == 2))
            {
                int tmpTai = CalculateTai(ponCnt);
                if (tmpTai > this.tai)
                {
                    this.tai = tmpTai;
                    this.scoringList = CalculateScoring(ponCnt);
                }
                return;
            }

            for(int i = 1; i <= 37; i++)
            {
                if(nowTileArray[i] == 0)
                    continue;
```

```
            // have pair
            if (havePair)
            {
                // check straight
                if(nowTileArray[i] > 0 && nowTileArray[i - 1] > 0 && nowTileArray[i + 1] > 0)
                {
                    nowTileArray[i - 1] -= 1;
                    nowTileArray[i] -= 1;
                    nowTileArray[i + 1] -= 1;
                    FindHighestTai((int[])nowTileArray.Clone(), true, ponCnt);
                    nowTileArray[i - 1] += 1;
                    nowTileArray[i] += 1;
                    nowTileArray[i + 1] += 1;
                }
                // check pon
                if(nowTileArray[i] >= 3)
                {
                    nowTileArray[i] -= 3;
                    FindHighestTai((int[])nowTileArray.Clone(), true, ponCnt + 1);
                    nowTileArray[i] += 3;
                }
            }
            // no pair yet
            else
            {
                if(nowTileArray[i] >= 2)
                {
                    nowTileArray[i] -= 2;
                    FindHighestTai((int[])nowTileArray.Clone(), true, ponCnt);
                    nowTileArray[i] += 2;
                }
            }
        }
    }

    private int CalculateTai(int handPonCnt = 0)
    {
        int calTai = 0;
        int[] tileCountArray = new int[50];

        //將所有手牌和擺牌轉為 array
        tileCountArray = TransToArray((int[])tileCountArray.Clone(), handTilesList);
        tileCountArray = TransToArray((int[])tileCountArray.Clone(), showTilesList);

        Debug.Log(tileCountArray);

        //地胡不計門清、自摸、不求人
        //天胡不計門清、自摸、不求人、獨聽、槓上開花
        //七搶一、八仙不計正花、花槓
        //花槓不計正花
        //小、大三元不再計三元刻
        //小、大四喜不再計圈風台、門風台
        if (Dealer())
        {
            calTai += 1;
            if (dealerWinStreak > 0)
            {
                calTai += dealerWinStreak * 2;
            }

            if (TianHu())
            {
                calTai += 24;
            }
        }
        if (DiHu())
        {
            calTai += 16;
        }

        //風
        if (DaSiXi(tileCountArray))
        {
            calTai += 16;
        }
        else if (XiaoSiXi(tileCountArray))
        {
            calTai += 8;
        }
        else
        {
            if (faceWind == 1 && Dong(tileCountArray))
            {
                calTai += 1;
```

```
        }
        else if (faceWind == 2 && Nan(tileCountArray))
        {
            calTai += 1;
        }
        else if (faceWind == 3 && Xi(tileCountArray))
        {
            calTai += 1;
        }
        else if (faceWind == 4 && Bei(tileCountArray))
        {
            calTai += 1;
        }

        else if (courtWind == 1 && Dong(tileCountArray))
        {
            calTai += 1;
        }
        else if (courtWind == 2 && Nan(tileCountArray))
        {
            calTai += 1;
        }
        else if (courtWind == 3 && Xi(tileCountArray))
        {
            calTai += 1;
        }
        else if (courtWind == 4 && Bei(tileCountArray))
        {
            calTai += 1;
        }
    }

    //三元
    if (DaSanYuan(tileCountArray))
    {
        calTai += 8;
    }
    else if (XiaoSanYuan(tileCountArray))
    {
        calTai += 4;
    }
    else
    {
        if (Zhong(tileCountArray))
        {
            calTai += 1;
        }
        if (Fa(tileCountArray))
        {
            calTai += 1;
        }
        if (Bai(tileCountArray))
        {
            calTai += 1;
        }
    }

    //花
    if (HuaGang(tileCountArray, true)) //春夏秋冬
    {
        calTai += 2;
    }
    else if(HuaTai(tileCountArray, 40 + faceWind))
    {
        calTai += 1;
    }

    if (HuaGang(tileCountArray, false)) //梅蘭竹菊
    {
        calTai += 2;
    }
    else if(HuaTai(tileCountArray, 44 + faceWind))
    {
        calTai += 1;
    }

    //門清、自摸、不求人
    if (!(TianHu() || DiHu()))
    {
        if (MenQing() && SelfDraw())
        {
            calTai += 3;
        }
        else if(MenQing())
        {
```

```
                calTai += 1;
            }
            else if(SelfDraw())
            {
                calTai += 1;
            }
        }

        //獨聽、槓上開花
        if (!TianHu())
        {
            if (DuTing())
            {
                calTai += 1;
            }
            if (GangShangKaiHua())
            {
                calTai += 1;
            }
        }

        if (HaiDiLaoYue())
        {
            calTai += 1;
        }

        if (HeDiLaoYu())
        {
            calTai += 1;
        }

        if (QuanQiuRen())
        {
            calTai += 2;
        }

        if (PingHu(tileCountArray, handPonCnt))
        {
            calTai += 2;
        }

        if (PengPengHu(handPonCnt))
        {
            calTai += 4;
        }

        if (SanAnKe(handPonCnt))
        {
            calTai += 2;
        }
        else if (SiAnKe(handPonCnt))
        {
            calTai += 5;
        }
        else if (WuAnKe(handPonCnt))
        {
            calTai += 8;
        }

        if (ZiYiSe(tileCountArray))
        {
            calTai += 16;
        }
        else if (QingYiSe(tileCountArray))
        {
            calTai += 8;
        }
        else if (HunYiSe(tileCountArray))
        {
            calTai += 4;
        }

        return calTai;
    }

    private List<string> CalculateScoring(int handPonCnt = 0)
    {
        List<string> calScoring = new List<string>();
        int[] tileCountArray = new int[50];

        //將所有手牌和擺牌轉為 array
        tileCountArray = TransToArray((int[])tileCountArray.Clone(), handTilesList);
        tileCountArray = TransToArray((int[])tileCountArray.Clone(), showTilesList);

        //地胡不計門清、自摸、不求人
```

```
            //天胡不計門清、自摸、不求人、獨聽、槓上開花
            //七搶一、八仙不計正花、花槓
            //花槓不計正花
            //小、大三元不再計三元刻
            //小、大四喜不再計圈風台、門風台
            if (Dealer())
            {
                calScoring.Add("莊家");
                if (dealerWinStreak > 0)
                {
                    calScoring.Add("連"+dealerWinStreak.ToString()+"拉"+dealerWinStreak.ToString());
                }

                if (TianHu())
                {
                    calScoring.Add("天胡");
                }
            }
            if (DiHu())
            {
                calScoring.Add("地胡");
            }

            //風
            if (DaSiXi(tileCountArray))
            {
                calScoring.Add("大四喜");
            }
            else if (XiaoSiXi(tileCountArray))
            {
                calScoring.Add("小四喜");
            }
            else
            {
                if (faceWind == 1 && Dong(tileCountArray))
                {
                    calScoring.Add("門風東");
                }
                else if (faceWind == 2 && Nan(tileCountArray))
                {
                    calScoring.Add("門風南");
                }
                else if (faceWind == 3 && Xi(tileCountArray))
                {
                    calScoring.Add("門風西");
                }
                else if (faceWind == 4 && Bei(tileCountArray))
                {
                    calScoring.Add("門風北");
                }

                if (courtWind == 1 && Dong(tileCountArray))
                {
                    calScoring.Add("場風東");
                }
                else if (courtWind == 2 && Nan(tileCountArray))
                {
                    calScoring.Add("場風南");
                }
                else if (courtWind == 3 && Xi(tileCountArray))
                {
                    calScoring.Add("場風西");
                }
                else if (courtWind == 4 && Bei(tileCountArray))
                {
                    calScoring.Add("場風北");
                }
            }

            //三元
            if (DaSanYuan(tileCountArray))
            {
                calScoring.Add("大三元");
            }
            else if (XiaoSanYuan(tileCountArray))
            {
                calScoring.Add("小三元");
            }
            else
            {
                if (Zhong(tileCountArray))
                {
                    calScoring.Add("中");
```

```
        }
        if (Fa(tileCountArray))
        {
            calScoring.Add("發");
        }
        if (Bai(tileCountArray))
        {
            calScoring.Add("白");
        }
    }

    //花
    if (HuaGang(tileCountArray, true)) //春夏秋冬
    {
        calScoring.Add("四季");
    }
    else if(HuaTai(tileCountArray, 40 + faceWind))
    {
        calScoring.Add(faceWind.ToString()+"花");
    }

    if (HuaGang(tileCountArray, false)) //梅蘭竹菊
    {
        calScoring.Add("四君子");
    }
    else if(HuaTai(tileCountArray, 44 + faceWind))
    {
        calScoring.Add(faceWind.ToString()+"花");
    }

    //門清、自摸、不求人
    if (!(TianHu() || DiHu()))
    {
        if (MenQing() && SelfDraw())
        {
            calScoring.Add("門清一摸三");
        }
        else if(MenQing())
        {
            calScoring.Add("門清");
        }
        else if(SelfDraw())
        {
            calScoring.Add("自摸");
        }
    }

    //獨聽、槓上開花
    if (!TianHu())
    {
        if (DuTing())
        {
            calScoring.Add("獨聽");
        }
        if (GangShangKaiHua())
        {
            calScoring.Add("槓上開花");
        }
    }

    if (HaiDiLaoYue())
    {
        calScoring.Add("海底撈月");
    }

    if (HeDiLaoYu())
    {
        calScoring.Add("河底撈魚");
    }

    if (QuanQiuRen())
    {
        calScoring.Add("全求人");
    }

    if (PingHu(tileCountArray, handPonCnt))
    {
        calScoring.Add("平胡");
    }

    if (PengPengHu(handPonCnt))
    {
        calScoring.Add("碰碰胡");
```

```
            }

            if (SanAnKe(handPonCnt))
            {
                calScoring.Add("三暗刻");
            }
            else if (SiAnKe(handPonCnt))
            {
                calScoring.Add("四暗刻");
            }
            else if (WuAnKe(handPonCnt))
            {
                calScoring.Add("五暗刻");
            }

            if (ZiYiSe(tileCountArray))
            {
                calScoring.Add("字一色");
            }
            else if (QingYiSe(tileCountArray))
            {
                calScoring.Add("清一色");
            }
            else if (HunYiSe(tileCountArray))
            {
                calScoring.Add("混一色");
            }

            return calScoring;
        }

        #endregion

        #region Judge

        private bool Dealer() //莊家
        {
            return isDealer;
        }

        private bool SelfDraw() //自摸
        {
            return isSelfDraw;
        }

        private bool MenQing() //門清
        {
            return deckGangCnt + deckPonCnt + deckStraightCnt == 0;
        }

        private bool Zhong(int[] tileCountArray) //中
        {
            return tileCountArray[45] >= 3;
        }

        private bool Fa(int[] tileCountArray) //發
        {
            return tileCountArray[46] >= 3;
        }

        private bool Bai(int[] tileCountArray) //白
        {
            return tileCountArray[47] >= 3;
        }

        private bool Dong(int[] tileCountArray) //東
        {
            return tileCountArray[31] >= 3;
        }

        private bool Nan(int[] tileCountArray)//南
        {
            return tileCountArray[32] >= 3;
        }

        private bool Xi(int[] tileCountArray)//西
        {
            return tileCountArray[33] >= 3;
        }

        private bool Bei(int[] tileCountArray)//北
        {
            return tileCountArray[34] >= 3;
        }
```

```
        private bool HuaGang(int[] tileCountArray, bool isSeason)//花槓
        {
            if (isSeason && tileCountArray[41] + tileCountArray[42] + tileCountArray[43] + tileCountArray
                [44] == 4)
            {
                return true;
            }
            else if ((!isSeason) && tileCountArray[45] + tileCountArray[46] + tileCountArray[47] +
                tileCountArray[48] == 4)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        private bool HuaTai(int[] tileCountArray, int pos)//花台
        {
            return tileCountArray[pos] == 1;
        }

        private bool DuTing()//獨聽
        {
            return isOnly;
        }

        private bool GangShangKaiHua()//槓上開花
        {
            return isAfterGang;
        }

        private bool HaiDiLaoYue()//海底撈月
        {
            return isLastTile && isSelfDraw;
        }

        private bool HeDiLaoYu()//河底撈魚
        {
            return isLastTile && (!isSelfDraw);
        }

        private bool QuanQiuRen()//全求人
        {
            return (!isSelfDraw) && deckGangCnt + deckPonCnt + deckStraightCnt == 5;
        }

        private bool PingHu(int[] tileCountArray, int handPonCnt)//平胡
        {
            //檢查無字無花
            int check = 0;
            for (int i = 31; i <= 48; i++)
            {
                check += tileCountArray[i];
            }

            return (!isSelfDraw) && deckGangCnt + deckPonCnt + hideGangCnt + handPonCnt + check == 0;
        }

        private bool SanAnKe(int handPonCnt)//三暗刻
        {
            return handPonCnt + hideGangCnt == 3;
        }

        private bool PengPengHu(int handPonCnt)//碰碰胡
        {
            return handPonCnt + hideGangCnt + deckPonCnt + deckGangCnt == 5;
        }

        private bool HunYiSe(int[] tileCountArray)//混一色
        {
            int characterCnt = 0;
            int dotCnt = 0;
            int bambooCnt = 0;
            int letterCnt = 0;

            for (int i = 1; i <= 9; i++)
            {
                characterCnt += tileCountArray[i];
            }
            for (int i = 11; i <= 19; i++)
            {
```

```
            dotCnt += tileCountArray[i];
        }
        for (int i = 21; i <= 29; i++)
        {
            bambooCnt += tileCountArray[i];
        }
        for (int i = 31; i <= 37; i++)
        {
            letterCnt += tileCountArray[i];
        }

        return (letterCnt > 0) && ((characterCnt + dotCnt == 0) || (characterCnt + bambooCnt == 0) || (
            dotCnt + bambooCnt == 0));
    }

    private bool XiaoSanYuan(int[] tileCountArray)//小三元
    {
        return tileCountArray[35] >= 2 && tileCountArray[36] >= 2 && tileCountArray[37] >= 2;
    }

    private bool SiAnKe(int handPonCnt)//四暗刻
    {
        return handPonCnt + hideGangCnt == 4;
    }

    private bool WuAnKe(int handPonCnt)//五暗刻
    {
        return handPonCnt + hideGangCnt == 5;
    }

    private bool QingYiSe(int[] tileCountArray)//清一色
    {
        int characterCnt = 0;
        int dotCnt = 0;
        int bambooCnt = 0;
        int letterCnt = 0;

        for (int i = 1; i <= 9; i++)
        {
            characterCnt += tileCountArray[i];
        }
        for (int i = 11; i <= 19; i++)
        {
            dotCnt += tileCountArray[i];
        }
        for (int i = 21; i <= 29; i++)
        {
            bambooCnt += tileCountArray[i];
        }
        for (int i = 31; i <= 37; i++)
        {
            letterCnt += tileCountArray[i];
        }

        return (letterCnt == 0) && ((characterCnt + dotCnt == 0) || (characterCnt + bambooCnt == 0) ||
            (dotCnt + bambooCnt == 0));
    }

    private bool XiaoSiXi(int[] tileCountArray)//小四喜
    {
        return tileCountArray[31] >= 2 && tileCountArray[32] >= 2 && tileCountArray[33] >= 2 &&
            tileCountArray[34] >= 2;
    }

    private bool DaSanYuan(int[] tileCountArray)//大三元
    {
        return tileCountArray[35] >= 3 && tileCountArray[36] >= 3 && tileCountArray[37] >= 3;
    }

    private bool ZiYiSe(int[] tileCountArray)//字一色
    {
        //檢查無萬筒條
        int cnt = 0;
        for (int i = 1; i <= 29; i++)
        {
            cnt += tileCountArray[i];
        }

        return cnt == 0;
    }

    private bool DaSiXi(int[] tileCountArray)//大四喜
    {
        return tileCountArray[31] >= 3 && tileCountArray[32] >= 3 && tileCountArray[33] >= 3 &&
            tileCountArray[34] >= 3;
```

```
        }

        private bool DiHu()//地胡
        {
            return isFirstTile && isSelfDraw && (!isDealer);
        }

        private bool TianHu()//天胡
        {
            return isFirstTile && isSelfDraw && isDealer;
        }

        #endregion

    }
}
```

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

namespace Game.Play
{
    public enum TileType
    {
        Flower,
        Season,
        Wind,
        Dragon,
        Character,
        Bamboo,
        Dot
    }

    public class TileGameObjectComparer : IComparer<GameObject>
    {
        public int Compare(GameObject x, GameObject y)
        {
            return String.CompareOrdinal(x?.GetComponent<Tile>().TileId, y?.GetComponent<Tile>().TileId);
        }
    }

    public class Tile : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
    {
        // {tile_type}_{tile_number}_{1..4}
        private string id;
        private Transform self;
        private Vector3 oriPosition;

        public int PlayerIndex => Player.PlayerIndex;
        public TileType TileType { get; private set; }
        public int TileNumber { get; private set; }
        public int CardFaceIndex { get; private set; }
        public Player Player { get; set; }
        public string TileId => id;

        void Start()
        {
            self = GetComponent<Transform>();
        }

        public void Init(TileType tileType, int tileNumber, int serialNumber, int cardFaceIndex)
        {
            TileType = tileType;
            TileNumber = tileNumber;
            CardFaceIndex = cardFaceIndex;
            id = TileType + "_" + TileNumber + "_" + serialNumber;
        }

        public bool Equals(Tile tile)
        {
            return tile.CardFaceIndex == CardFaceIndex;
        }

        #region - Drag Process

        public void OnBeginDrag(PointerEventData eventData)
        {
            if (IsValidDrag(self.transform.parent))
            {
                oriPosition = self.transform.position;
            }
        }
```

```
        public void OnDrag(PointerEventData eventData)
        {
            if (IsValidDrag(self.transform.parent))
            {
                self.transform.position = eventData.position;
            }
        }

        public void OnEndDrag(PointerEventData eventData)
        {
            if (IsValidDrag(self.transform.parent))
            {
                if (transform.localPosition.y > 0)
                {
                    if(!Player.Discard(id))
                    {
                        self.transform.position = oriPosition;
                    }
                }
                else
                {
                    self.transform.position = oriPosition;
                }

                Debug.Log("Tile" + PlayerPrefs.GetFloat("Tile"));
                if(PlayerPrefs.GetFloat("Tile") > 0.0f)
                {
                    gameObject.GetComponent<AudioSource>().Play();
                }
            }
        }

        private bool IsValidDrag(Transform parent)
        {
            return parent == Player.Hand;
        }

        #endregion

        #region - Judge

        // Flower Season
        public bool IsFlower()
        {
            return TileType is TileType.Flower or TileType.Season;
        }

        // Wind Dragon
        public bool IsHonor()
        {
            return TileType is TileType.Wind or TileType.Dragon;
        }
        // Dot Bamboo Character
        public bool IsSuit()
        {
            return TileType is TileType.Dot or TileType.Bamboo or TileType.Character;
        }

        #endregion

    }
}
```

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

namespace Game.Play {
    internal class TileCategory
    {
        public TileType Type { get; private set; }
        public int MaxTileNumber { get; private set; }
        public int MaxSerialNumber { get; private set; }

        public TileCategory(TileType type, int maxTileNumber, int maxSerialNumber)
        {
            Type = type;
            MaxTileNumber = maxTileNumber;
            MaxSerialNumber = maxSerialNumber;
        }
    }

    public class TileWall : MonoBehaviour
    {
        [SerializeField] private Transform transformTileWall;
```

```
        [SerializeField] private GameObject mahjongPrefab; // 麻將Prefab
        [SerializeField] private Text tileRemain;
        private Vector3 startPosition;
        private readonly List<GameObject> tileList = new();

        private static readonly List<TileCategory> TileCategoriesList = new List<TileCategory>()
        {
            // Season       1-4
            new(TileType.Season, 4, 1),
            // Flower       5-8
            new(TileType.Flower, 4, 1),
            // Wind         9-12
            new(TileType.Wind, 4, 4),
            // Dragon       13-15
            new(TileType.Dragon, 3, 4),
            // Character    16-24
            new(TileType.Character, 9, 4),
            // Bamboo       25-33
            new(TileType.Bamboo, 9, 4),
            // Dot          34-42
            new(TileType.Dot, 9, 4)
        };

        public void GetReady()
        {
            GenerateAllTile();
            Shuffle();
        }

        public bool DealTile(Player player)
        {
            if(tileList.Count <= 16)
            {
                return false;
            }

            RemoveTile(player, 0);
            return true;
        }

        public void Replenish(Player player)
        {
            RemoveTile(player, tileList.Count - 1);
        }

        #region - HelperFunctions

        private void RemoveTile(Player player, int idx)
        {
            GameObject tile = tileList[idx];
            player.GetTile(tile);
            tileList.RemoveAt(idx);
            tileRemain.text = tileList.Count.ToString();
        }

        private void SetTileFace(GameObject mahjong)
        {
            int cardFaceIndex = mahjong.GetComponent<Tile>().CardFaceIndex;
            var faceImage = mahjong.transform.Find("Face").gameObject.GetComponent<Image>();

            var img = Resources.Load<Sprite>("Image/Mahjong/" + cardFaceIndex);
            if (img)
            {
                faceImage.sprite = img;
            }
            else
            {
                Debug.Log("無法設定圖像" + "Image/Mahjong/" + cardFaceIndex);
            }
        }
        private void GenerateTile(TileType tileType, int tileNumber, int cardFaceIndex, int serialNumber)
        {
            var tileObj = Instantiate(mahjongPrefab, startPosition + new Vector3(0, 0, 0), Quaternion.
                identity, transformTileWall);
            var tile =  tileObj.GetComponent<Tile>();
            if(tile!=null)
            {
                tile.Init(tileType, tileNumber, serialNumber, cardFaceIndex);
                tileObj.name = tile.TileId;
            }
            else
            {
                Debug.LogError("沒有 tile_script");
            }

            tileList.Add(tileObj);
```

```
            SetTileFace(tileObj);
        }

        private void GenerateAllTile()
        {
            int cardFaceIndex = 1;

            foreach (var tileCategory in TileCategoriesList)
            {
                for (int tileNumber = 0; tileNumber < tileCategory.MaxTileNumber; tileNumber++)
                {
                    for (int serialNumber = 0; serialNumber < tileCategory.MaxSerialNumber; serialNumber++)
                    {
                        GenerateTile(tileCategory.Type, tileNumber, cardFaceIndex, serialNumber);
                    }
                    cardFaceIndex++;
                }
            }
        }
        private void Shuffle()
        {
            for(int i = 0; i < tileList.Count; ++i)
            {
                int j = Random.Range(0, tileList.Count);
                (tileList[i], tileList[j]) = (tileList[j], tileList[i]);
            }
        }

        #endregion
    }
}
```

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

namespace Utils
{
    public class RoomID
    {
        private const int MinID = 100000;
        private const int MaxID = 999999;
        private readonly Random rand = new Random();
        private readonly Dictionary<string, int> roomIDDic = new Dictionary<string, int>();

        public int Get(string roomName)
        {
            if (roomIDDic.TryGetValue(roomName, out var value))
                return value;

            return roomIDDic[roomName] = rand.Next(MinID, MaxID);
        }

        public void Del(string roomName)
        {
            if (roomIDDic.ContainsKey(roomName))
                roomIDDic.Remove(roomName);
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.SceneManagement;
using Game.Core;

namespace Game.Lobby
{
    public class WaitingRoomScene : MonoBehaviour
    {
        private LobbyManager lobbyManager;
        private GameManager gameManager;

        [SerializeField] private GameObject startBtn;
        [SerializeField] private Music.BgmController musicController;

        public void Start()
        {
            lobbyManager = LobbyManager.Instance;
            lobbyManager.PanelController.AddPanel(EnumPanel.Waiting, gameObject);
            gameManager = GameManager.Instance;
            On100BaseBtnClick();
        }

        public void OnEnable()
        {
            startBtn.SetActive(GameManager.Instance.Runner.GameMode == Fusion.GameMode.Host);
```

```
        }

        #region - BtnCallBack

        public void OnLeaveBtnClick()
        {
            GameManager.Instance.Disconnect();
            musicController.PlayMusic(Music.EnumMusic.Start);
            lobbyManager.PanelController.OpenPanel(EnumPanel.RoomList);
        }

        public void OnStartBtnClick()
        {
            if (gameManager.CheckAllPlayerIsReady())
            {
                int curSceneIndex = SceneManager.GetActiveScene().buildIndex;
                gameManager.Runner.SetActiveScene(curSceneIndex + 1);
            }
        }

        public void OnReadyBtnClick()
        {
            var runner = gameManager.Runner;
            if (gameManager.PlayerList.TryGetValue(runner.LocalPlayer, out PlayerNetworkData
                playerNetworkData))
            {
                playerNetworkData.SetReady_RPC(true);
            }
        }

        public void On100BaseBtnClick()
        {
            gameManager.GameBasePoint = 100;
            gameManager.GameTaiPoint = 30;
        }

        public void On300BaseBtnClick()
        {
            gameManager.GameBasePoint = 300;
            gameManager.GameTaiPoint = 100;
        }

        public void On500BaseBtnClick()
        {
            gameManager.GameBasePoint = 500;
            gameManager.GameTaiPoint = 200;
        }

        #endregion
    }
}
```

References

[1] Unity Doucumentation: https://docs.unity3d.com/ScriptReference/
[2] 背景音樂 1: https://www.youtube.com/watch?v=MYRM5HIJpN0
[3] 背景音樂 2: https://www.youtube.com/watch?v=CwRRsOtEwWQ
[4] 背景音樂 3: https://www.youtube.com/watch?v=2g3mkjQh9pQ
[5] 麻將素材: https://demching.itch.io/mahjong
[6] Icon 熊貓: https://www.flaticon.com/free-icon/laugh_7326389?
[7] Icon 竹子: https://www.flaticon.com/free-icon/bamboo-canes_68143