Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

\_\_\_\_\_\_\_\_ Максим ГОЛОВЧЕНКО

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 р.

**ГРА ДЛЯ МОБІЛЬНОГО ПРИСТРОЮ «МОНОПОЛІЯ»**

**Текст програми**

КПІ.ІП-1402.045490.03.12

“ПОГОДЖЕНО”

Керівник роботи:

\_\_\_\_\_\_\_\_\_\_\_\_ Максим ГОЛОВЧЕНКО

| Консультант: | Виконавець: |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_ Максим ГОЛОВЧЕНКО | \_\_\_\_\_\_\_\_\_\_ Денис БАБІЧ |

Київ – 2024

**Файл GameCoordinator.cs**

using System;

using UnityEngine;

using Unity.Netcode;

using Unity.Services.Core;

using Unity.Services.Relay;

using System.Threading.Tasks;

using Unity.Services.Lobbies;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using Unity.Services.Relay.Models;

using Unity.Netcode.Transports.UTP;

using Unity.Services.Authentication;

using Unity.Services.Lobbies.Models;

using Unity.Networking.Transport.Relay;

#if UNITY\_EDITOR

using ParrelSync;

#endif

internal sealed class GameCoordinator : MonoBehaviour

{

public enum MonopolyScene : byte

{

Bootstrap,

MainMenu,

GameLobby,

MonopolyGame

}

private const string CONNECTION\_TYPE = "dtls";

private Scene activeScene;

private int initializationCount;

private LinkedList<Type> objectsToLoad;

private LinkedList<Type> initializedObjects;

public static GameCoordinator Instance { get; private set; }

public event Action OnAuthenticationFailed;

public event Action<RelayServiceException> OnEstablishingConnectionRelayFailed;

public event Action<LobbyServiceException> OnEstablishingConnectionLobbyFailed;

public Player LocalPlayer { get; private set; }

public MonopolyScene ActiveScene { get; private set; }

private void Awake()

{

if (Instance != null)

throw new System.InvalidOperationException($"Singleton {this.GetType().FullName} has already been initialized.");

Instance = this;

UnityEngine.Object.DontDestroyOnLoad(this.gameObject);

}

private void OnEnable()

{

SceneManager.activeSceneChanged += this.HandleActiveSceneChanged;

}

private void OnDisable()

{

SceneManager.activeSceneChanged -= this.HandleActiveSceneChanged;

}

private async void Start()

{

this.objectsToLoad = new LinkedList<Type>();

this.initializedObjects = new LinkedList<Type>();

try

{

#if UNITY\_EDITOR

InitializationOptions options = new InitializationOptions();

options.SetProfile(ClonesManager.IsClone() ? ClonesManager.GetArgument() : "Primary");

await UnityServices.InitializeAsync(options);

#else

await UnityServices.InitializeAsync();

#endif

await AuthenticationService.Instance.SignInAnonymouslyAsync();

this.InitializeLocalPlayer(PlayerPrefs.GetString(LobbyManager.KEY\_PLAYER\_NICKNAME));

}

catch

{

this.OnAuthenticationFailed?.Invoke();

return;

}

await this.LoadSceneAsync(GameCoordinator.MonopolyScene.MainMenu);

}

#region Updating Player

public void UpdateLocalPlayer(string newNickname)

{

newNickname = newNickname.Trim();

this.LocalPlayer.Data[LobbyManager.KEY\_PLAYER\_NICKNAME].Value = newNickname;

PlayerPrefs.SetString(LobbyManager.KEY\_PLAYER\_NICKNAME, newNickname);

PlayerPrefs.Save();

}

public void InitializeLocalPlayer(string nickname)

{

nickname = nickname.Trim();

Player player = new Player(AuthenticationService.Instance.PlayerId)

{

Data = new Dictionary<string, PlayerDataObject>

{

{ LobbyManager.KEY\_PLAYER\_NICKNAME, new PlayerDataObject(PlayerDataObject.VisibilityOptions.Member, nickname) },

{ LobbyManager.KEY\_PLAYER\_SCENE, new PlayerDataObject(PlayerDataObject.VisibilityOptions.Member, GameCoordinator.Instance.ActiveScene.ToString()) }

}

};

PlayerPrefs.SetString(LobbyManager.KEY\_PLAYER\_NICKNAME, nickname);

PlayerPrefs.Save();

this.LocalPlayer = player;

}

#endregion

#region Scenes Management

public void LoadSceneNetwork(MonopolyScene scene)

{

NetworkManager.Singleton.SceneManager.LoadScene(scene.ToString(), LoadSceneMode.Single);

}

public async Task LoadSceneAsync(MonopolyScene scene)

{

await SceneManager.LoadSceneAsync(scene.ToString(), LoadSceneMode.Single);

}

public void UpdateInitializedObjects(Type gameObject)

{

if (this.objectsToLoad == null)

{

throw new System.InvalidOperationException($"You have to call {nameof(this.SetupInitializedObjects)} at first.");

}

if (!this.objectsToLoad.Contains(gameObject))

{

throw new System.ArgumentException($"{nameof(gameObject)} is not in {nameof(this.SetupInitializedObjects)}.");

}

if (this.initializedObjects.Contains(gameObject))

{

throw new System.ArgumentException($"{nameof(gameObject)} has already been initialized.");

}

this.initializedObjects.AddLast(gameObject);

if (this.initializedObjects.Count == this.objectsToLoad.Count)

{

LobbyManager.Instance?.UpdateLocalPlayerData();

}

}

public void SetupInitializedObjects(params Type[] gameObjectsToLoad)

{

foreach (Type gameObject in gameObjectsToLoad)

{

this.objectsToLoad.AddLast(gameObject);

}

}

private void HandleActiveSceneChanged(Scene previousActiveScene, Scene newActiveScene)

{

this.objectsToLoad?.Clear();

this.initializedObjects?.Clear();

this.activeScene = SceneManager.GetActiveScene();

switch (newActiveScene.name)

{

case nameof(GameCoordinator.MonopolyScene.MainMenu):

this.ActiveScene = GameCoordinator.MonopolyScene.MainMenu;

break;

case nameof(GameCoordinator.MonopolyScene.GameLobby):

{

this.ActiveScene = GameCoordinator.MonopolyScene.GameLobby;

this.SetupInitializedObjects(typeof(UIManagerGameLobby), typeof(ObjectPoolPanelPlayerLobby));

LobbyManager.Instance?.OnGameLobbyLoaded?.Invoke();

}

break;

case nameof(GameCoordinator.MonopolyScene.MonopolyGame):

{

this.ActiveScene = GameCoordinator.MonopolyScene.MonopolyGame;

this.SetupInitializedObjects(typeof(GameManager), typeof(MonopolyBoard), typeof(UIManagerMonopolyGame));

LobbyManager.Instance?.OnMonopolyGameLoaded?.Invoke();

}

break;

}

}

#endregion

#region Establishing Connection

public async Task HostLobbyAsync()

{

if (this.LocalPlayer == null)

{

throw new System.InvalidOperationException($"{nameof(this.LocalPlayer)} is null.");

}

try

{

Allocation hostAllocation = await RelayService.Instance.CreateAllocationAsync(LobbyManager.MAX\_PLAYERS);

RelayServerData relayServerData = new RelayServerData(hostAllocation, GameCoordinator.CONNECTION\_TYPE);

NetworkManager.Singleton?.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

string relayCode = await RelayService.Instance.GetJoinCodeAsync(hostAllocation.AllocationId);

await LobbyManager.Instance?.HostLobbyAsync(relayCode);

}

catch (RelayServiceException relayServiceException)

{

this.OnEstablishingConnectionRelayFailed?.Invoke(relayServiceException);

}

catch (LobbyServiceException lobbyServiceException)

{

this.OnEstablishingConnectionLobbyFailed?.Invoke(lobbyServiceException);

}

}

public async Task ConnectLobbyAsync(string joinCode)

{

if (this.LocalPlayer == null)

{

throw new System.InvalidOperationException($"{nameof(this.LocalPlayer)} is null.");

}

try

{

JoinAllocation clientAllocation = await RelayService.Instance.JoinAllocationAsync(joinCode);

RelayServerData relayServerData = new RelayServerData(clientAllocation, GameCoordinator.CONNECTION\_TYPE);

NetworkManager.Singleton?.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

await LobbyManager.Instance?.ConnectLobbyAsync(joinCode);

}

catch (RelayServiceException relayServiceException)

{

this.OnEstablishingConnectionRelayFailed?.Invoke(relayServiceException);

}

catch (LobbyServiceException lobbyServiceException)

{

this.OnEstablishingConnectionLobbyFailed?.Invoke(lobbyServiceException);

}

}

#endregion

}

**Файл LobbyManager.cs**

using System;

using System.Linq;

using UnityEngine;

using Unity.Netcode;

using System.Collections;

using System.Threading.Tasks;

using Unity.Services.Lobbies;

using System.Collections.Generic;

using Unity.Services.Lobbies.Models;

internal sealed class LobbyManager : MonoBehaviour

{

private const float LOBBY\_UPTIME = 25.0f;

public const int MIN\_PLAYERS = 2;

public const int MAX\_PLAYERS = 5;

public const float LOBBY\_LOADING\_TIMEOUT = 15.0f;

public const string KEY\_PLAYER\_SCENE = "Scene";

public const string KEY\_PLAYER\_NICKNAME = "Nickname";

public const string KEY\_LOBBY\_STATE = "State";

public const string LOBBY\_STATE\_GAME = "Game";

public const string LOBBY\_STATE\_LOBBY = "Lobby";

public const string LOBBY\_STATE\_LOADING = "Loading";

public const string LOBBY\_STATE\_PENDING = "Waiting";

public const string LOBBY\_STATE\_RETURNING = "Returning";

private string lobbyName

{

get => $"LOBBY\_{this.JoinCode}";

}

private ILobbyEvents localLobbyEvents;

private QueryLobbiesOptions queryCurrentLobby

{

get

{

return new QueryLobbiesOptions()

{

Filters = new List<QueryFilter>()

{

new QueryFilter(QueryFilter.FieldOptions.Name, this.JoinCode, QueryFilter.OpOptions.CONTAINS)

}

};

}

}

public static LobbyManager Instance { get; private set; }

public Action OnGameLobbyLoaded;

public Action OnMonopolyGameLoaded;

public Action OnGameLobbyFailedToLoad;

public Action OnMonopolyGameFailedToLoad;

public bool IsHost { get; private set; }

public string JoinCode { get; private set; }

public bool HasHostLeft { get; private set; }

public Lobby LocalLobby { get; private set; }

public bool HavePlayersLoaded

{

get

{

return this.LocalLobby != null ? this.LocalLobby.Players.All(player => player.Data[LobbyManager.KEY\_PLAYER\_SCENE].Value.Equals(GameCoordinator.Instance.ActiveScene.ToString(), StringComparison.Ordinal)) : false;

}

}

public bool HasLocalPlayerLeft { get; private set; }

public LobbyEventCallbacks LocalLobbyEventCallbacks { get; private set; }

private void Awake()

{

if (Instance != null)

throw new System.InvalidOperationException($"Singleton {this.GetType().FullName} has already been initialized.");

Instance = this;

UnityEngine.Object.DontDestroyOnLoad(this.gameObject);

}

private void OnEnable()

{

this.LocalLobbyEventCallbacks = new LobbyEventCallbacks();

this.OnGameLobbyLoaded += this.HandleGameLobbyLoaded;

this.OnMonopolyGameLoaded += this.HandleMonopolyGameLoaded;

this.OnGameLobbyFailedToLoad += this.HandleGameLobbyFailedToLoadAsync;

this.OnMonopolyGameFailedToLoad += this.HandleMonopolyGameFailedToLoad;

this.LocalLobbyEventCallbacks.PlayerLeft += this.HandlePlayerLeft;

this.LocalLobbyEventCallbacks.DataChanged += this.HandleDataChanged;

this.LocalLobbyEventCallbacks.LobbyDeleted += this.HandleLobbyDeleted;

this.LocalLobbyEventCallbacks.PlayerJoined += this.HandlePlayerJoined;

this.LocalLobbyEventCallbacks.PlayerDataChanged += this.HandlePlayerDataChanged;

this.LocalLobbyEventCallbacks.KickedFromLobby += this.HandleKickedFromLobbyAsync;

NetworkManager.Singleton.OnTransportFailure += this.HandleTransportFailureAsync;

}

private void OnDisable()

{

this.LocalLobbyEventCallbacks = new LobbyEventCallbacks();

this.OnGameLobbyLoaded -= this.HandleGameLobbyLoaded;

this.OnMonopolyGameLoaded -= this.HandleMonopolyGameLoaded;

this.OnGameLobbyFailedToLoad -= this.HandleGameLobbyFailedToLoadAsync;

this.OnMonopolyGameFailedToLoad -= this.HandleMonopolyGameFailedToLoad;

this.LocalLobbyEventCallbacks.PlayerLeft -= this.HandlePlayerLeft;

this.LocalLobbyEventCallbacks.DataChanged -= this.HandleDataChanged;

this.LocalLobbyEventCallbacks.LobbyDeleted -= this.HandleLobbyDeleted;

this.LocalLobbyEventCallbacks.PlayerJoined -= this.HandlePlayerJoined;

this.LocalLobbyEventCallbacks.PlayerDataChanged -= this.HandlePlayerDataChanged;

this.LocalLobbyEventCallbacks.KickedFromLobby -= this.HandleKickedFromLobbyAsync;

if (NetworkManager.Singleton != null)

{

NetworkManager.Singleton.OnTransportFailure -= this.HandleTransportFailureAsync;

}

}

private async void OnDestroy()

{

if (this.IsHost)

{

this.StopCoroutine(this.PingLobbyCoroutine());

}

if (this.LocalLobby != null)

{

await this.DisconnectFromLobbyAsync();

}

}

#region Start & End Game

public void StartGameAsync()

{

if (!this.HavePlayersLoaded)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerGameLobby.Instance.MessageNotAllPlayersLoaded, PanelMessageBoxUI.Icon.Warning);

return;

}

if (this.LocalLobby.Players.Count < LobbyManager.MIN\_PLAYERS)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerGameLobby.Instance.MessageTooFewPlayers, PanelMessageBoxUI.Icon.Warning);

return;

}

this.UpdateLocalLobbyData(LobbyManager.LOBBY\_STATE\_LOADING, true);

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.None, UIManagerGameLobby.Instance.MessagePendingGame, PanelMessageBoxUI.Icon.Loading);

GameCoordinator.Instance.LoadSceneNetwork(GameCoordinator.MonopolyScene.MonopolyGame);

}

#endregion

#region Lobby API

private async Task LeaveLobbyAsync()

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.None, UIManagerGameLobby.Instance?.MessageDisconnecting ?? UIManagerMainMenu.Instance?.MessageDisconnecting, PanelMessageBoxUI.Icon.Loading);

NetworkManager.Singleton?.Shutdown();

if (this != null)

{

await this.localLobbyEvents?.UnsubscribeAsync();

}

await GameCoordinator.Instance?.LoadSceneAsync(GameCoordinator.MonopolyScene.MainMenu);

if (!this.IsHost)

{

if (this.HasHostLeft)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerGameLobby.Instance?.MessageHostDisconnected ?? UIManagerMainMenu.Instance?.MessageHostDisconnected, PanelMessageBoxUI.Icon.Error);

}

else if (!this.HasLocalPlayerLeft)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerGameLobby.Instance?.MessageKicked ?? UIManagerMainMenu.Instance?.MessageKicked, PanelMessageBoxUI.Icon.Error);

}

}

this.IsHost = false;

this.LocalLobby = null;

this.HasHostLeft = false;

this.HasLocalPlayerLeft = false;

}

public async Task DisconnectFromLobbyAsync()

{

this.HasLocalPlayerLeft = true;

if (this.IsHost)

{

this.StopCoroutine(this.PingLobbyCoroutine());

await LobbyService.Instance.DeleteLobbyAsync(this.LocalLobby.Id);

}

else

{

await LobbyService.Instance.RemovePlayerAsync(this.LocalLobby.Id, GameCoordinator.Instance.LocalPlayer.Id);

}

}

public async Task HostLobbyAsync(string relayCode)

{

this.IsHost = true;

this.HasHostLeft = false;

this.JoinCode = relayCode;

this.HasLocalPlayerLeft = false;

CreateLobbyOptions lobbyOptions = new CreateLobbyOptions()

{

Player = GameCoordinator.Instance.LocalPlayer,

Data = new Dictionary<string, DataObject>()

{

{ LobbyManager.KEY\_LOBBY\_STATE, new DataObject(DataObject.VisibilityOptions.Member, LobbyManager.LOBBY\_STATE\_LOBBY) }

}

};

try

{

this.LocalLobby = await LobbyService.Instance.CreateLobbyAsync(this.lobbyName, LobbyManager.MAX\_PLAYERS, lobbyOptions);

this.localLobbyEvents = await LobbyService.Instance.SubscribeToLobbyEventsAsync(this.LocalLobby.Id, this.LocalLobbyEventCallbacks);

NetworkManager.Singleton?.StartHost();

}

catch (LobbyServiceException lobbyServiceException)

{

throw lobbyServiceException;

}

if (this != null)

{

this.StartCoroutine(this.PingLobbyCoroutine());

await GameCoordinator.Instance.LoadSceneAsync(GameCoordinator.MonopolyScene.GameLobby);

}

}

public async Task ConnectLobbyAsync(string joinCode)

{

this.IsHost = false;

this.HasHostLeft = false;

this.JoinCode = joinCode;

this.HasLocalPlayerLeft = false;

JoinLobbyByIdOptions joinOptions = new JoinLobbyByIdOptions()

{

Player = GameCoordinator.Instance.LocalPlayer

};

try

{

QueryResponse queryResponse = await Lobbies.Instance.QueryLobbiesAsync(this.queryCurrentLobby);

this.LocalLobby = await LobbyService.Instance.JoinLobbyByIdAsync(queryResponse.Results.FirstOrDefault().Id, joinOptions);

this.localLobbyEvents = await LobbyService.Instance.SubscribeToLobbyEventsAsync(this.LocalLobby.Id, this.LocalLobbyEventCallbacks);

NetworkManager.Singleton?.StartClient();

}

catch (LobbyServiceException lobbyServiceException)

{

throw lobbyServiceException;

}

catch (NullReferenceException nullReferenceException)

{

throw new LobbyServiceException(LobbyExceptionReason.InvalidJoinCode, "Invalid Join Code.", nullReferenceException);

}

}

public async Task KickFromLobbyAsync(string playerId)

{

await LobbyService.Instance.RemovePlayerAsync(this.LocalLobby.Id, playerId);

}

#endregion

#region Lobby Ping

private IEnumerator PingLobbyCoroutine()

{

WaitForSeconds waitForSeconds = new WaitForSeconds(LobbyManager.LOBBY\_UPTIME);

while (this.LocalLobby != null)

{

Lobbies.Instance.SendHeartbeatPingAsync(this.LocalLobby?.Id);

yield return waitForSeconds;

}

}

#endregion

#region Lobby Update

public async void UpdateLocalPlayerData()

{

GameCoordinator.Instance.LocalPlayer.Data[LobbyManager.KEY\_PLAYER\_SCENE] = new PlayerDataObject(PlayerDataObject.VisibilityOptions.Member, GameCoordinator.Instance.ActiveScene.ToString());

UpdatePlayerOptions updatePlayerOptions = new UpdatePlayerOptions()

{

Data = GameCoordinator.Instance.LocalPlayer.Data

};

this.LocalLobby = await LobbyService.Instance.UpdatePlayerAsync(this.LocalLobby.Id, GameCoordinator.Instance.LocalPlayer.Id, updatePlayerOptions);

}

public async void UpdateLocalLobbyData(string lobbyState, bool isPrivate = true)

{

this.LocalLobby.Data[LobbyManager.KEY\_LOBBY\_STATE] = new DataObject(DataObject.VisibilityOptions.Member, lobbyState);

UpdateLobbyOptions updateLobbyOptions = new UpdateLobbyOptions()

{

IsPrivate = isPrivate,

Data = this.LocalLobby.Data

};

await Lobbies.Instance.UpdateLobbyAsync(this.LocalLobby.Id, updateLobbyOptions);

}

#endregion

#region Lobby Callbacks

private void HandleLobbyDeleted()

{

this.HasHostLeft = true;

}

private async void HandleKickedFromLobbyAsync()

{

await this.LeaveLobbyAsync();

}

private async void HandleTransportFailureAsync()

{

await LobbyManager.Instance.DisconnectFromLobbyAsync();

}

private void HandlePlayerLeft(List<int> leftPlayers)

{

foreach (int playerIndex in leftPlayers)

{

this.LocalLobby.Players.RemoveAt(playerIndex);

}

}

private void HandlePlayerJoined(List<LobbyPlayerJoined> joinedPlayers)

{

foreach (LobbyPlayerJoined newPlayer in joinedPlayers)

{

this.LocalLobby.Players.Add(newPlayer.Player);

}

}

private void HandleDataChanged(Dictionary<string, ChangedOrRemovedLobbyValue<DataObject>> changedLobbyData)

{

foreach (string key in changedLobbyData.Keys)

{

this.LocalLobby.Data[key] = changedLobbyData[key].Value;

}

switch (this.LocalLobby.Data[LobbyManager.KEY\_LOBBY\_STATE].Value)

{

case LobbyManager.LOBBY\_STATE\_LOADING:

case LobbyManager.LOBBY\_STATE\_PENDING:

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.None, UIManagerGameLobby.Instance?.MessagePendingGame ?? UIManagerMonopolyGame.Instance?.MessageWaitingOtherPlayers, PanelMessageBoxUI.Icon.Loading, stateCallback: () => this.HavePlayersLoaded);

break;

case LobbyManager.LOBBY\_STATE\_RETURNING:

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.None, UIManagerGameLobby.Instance?.MessageFailedToConnect ?? UIManagerMonopolyGame.Instance?.MessagePlayersFailedToLoad, PanelMessageBoxUI.Icon.Loading, stateCallback: () => this.HavePlayersLoaded);

break;

}

}

private void HandlePlayerDataChanged(Dictionary<int, Dictionary<string, ChangedOrRemovedLobbyValue<PlayerDataObject>>> changedPlayerData)

{

foreach (int playerIndex in changedPlayerData.Keys)

{

foreach (string key in changedPlayerData[playerIndex].Keys)

{

this.LocalLobby.Players[playerIndex].Data[key] = changedPlayerData[playerIndex][key].Value;

}

}

}

#endregion

#region Loading Callbacks

private void HandleGameLobbyLoaded()

{

if (this.IsHost)

{

this.UpdateLocalLobbyData(LobbyManager.LOBBY\_STATE\_LOBBY, false);

}

}

private void HandleMonopolyGameLoaded()

{

if (this.IsHost)

{

this.UpdateLocalLobbyData(LobbyManager.LOBBY\_STATE\_LOBBY, true);

}

}

private void HandleMonopolyGameFailedToLoad()

{

if (this.IsHost)

{

this.UpdateLocalLobbyData(LobbyManager.LOBBY\_STATE\_RETURNING, true);

GameCoordinator.Instance.LoadSceneNetwork(GameCoordinator.MonopolyScene.GameLobby);

}

}

private async void HandleGameLobbyFailedToLoadAsync()

{

await this.DisconnectFromLobbyAsync();

}

#endregion

}

**Файл TradeCredentialsSerializer.cs**

using Unity.Netcode;

public struct TradeCredentials : INetworkSerializable

{

public ulong SenderId;

public ulong ReceiverId;

public int SenderOffer;

public int ReceiverOffer;

public int SenderNodeIndex;

public int ReceiverNodeIndex;

public void NetworkSerialize<T>(BufferSerializer<T> serializer) where T : IReaderWriter

{

serializer.SerializeValue(ref this.SenderId);

serializer.SerializeValue(ref this.ReceiverId);

serializer.SerializeValue(ref this.SenderOffer);

serializer.SerializeValue(ref this.ReceiverOffer);

serializer.SerializeValue(ref this.SenderNodeIndex);

serializer.SerializeValue(ref this.ReceiverNodeIndex);

}

}

**Файл GameManager.cs**

using System;

using System.Linq;

using UnityEngine;

using Unity.Netcode;

using System.Collections;

using System.Collections.Generic;

using System.Collections.ObjectModel;

internal sealed class GameManager : NetworkBehaviour

{

#region Setup

#region Values

[Header("Values")]

[Space]

[SerializeField] [Range(0, 100\_000)] private int startingBalance = 15\_000;

[Space]

[SerializeField] [Range(0, 10)] private int maxTurnsInJail = 3;

[Space]

[SerializeField] [Range(0, 10)] private int maxDoublesInRow = 2;

[Space]

[SerializeField] [Range(0, 100\_000)] private int circleBonus = 2\_000;

[Space]

[SerializeField] [Range(0, 100\_000)] private int exactCircleBonus = 3\_000;

[Space]

[SerializeField] [Range(0.0f, 100.0f)] private float playerMovementSpeed = 35.0f;

#endregion

#region Visuals

[Space]

[Header("Visuals")]

#region Player

[Space]

[Header("Player")]

[Space]

[SerializeField] private GameObject player;

[Space]

[SerializeField] private GameObject playerPanel;

#endregion

#region Players Tokens

[Space]

[Header("Players Visuals")]

[Space]

[SerializeField] private MonopolyPlayerVisuals[] monopolyPlayersVisuals = new MonopolyPlayerVisuals[5];

#endregion

#endregion

#endregion

private int rolledDoubles;

private List<MonopolyPlayer> players;

private ulong[] targetCurrentClient;

private ulong[] targetClientOtherClients;

private List<ulong[]> targetHostOtherClients;

public static GameManager Instance { get; private set; }

public int CircleBonus

{

get => this.circleBonus;

}

public int MaxTurnsInJail

{

get => this.maxTurnsInJail;

}

public int TotalRollResult

{

get => this.FirstDieValue + this.SecondDieValue;

}

public int MaxDoublesInRow

{

get => this.maxDoublesInRow;

}

public int StartingBalance

{

get => this.startingBalance;

}

public bool HasRolledDouble

{

get => this.FirstDieValue == this.SecondDieValue;

}

public int ExactCircleBonus

{

get => this.exactCircleBonus;

}

public float PlayerMovementSpeed

{

get => this.playerMovementSpeed;

}

public MonopolyPlayer CurrentPlayer

{

get

{

if (this.CurrentPlayerIndex >= 0 && this.CurrentPlayerIndex < this.players.Count)

{

return this.players[this.CurrentPlayerIndex];

}

else

{

return null;

}

}

}

public int FirstDieValue { get; private set; }

public int SecondDieValue { get; private set; }

public int CurrentPlayerIndex { get; private set; }

public ServerRpcParams ServerParamsCurrentClient

{

get

{

return new ServerRpcParams

{

Receive = new ServerRpcReceiveParams { SenderClientId = NetworkManager.Singleton.LocalClientId }

};

}

}

public ClientRpcParams ClientParamsCurrentClient

{

get

{

this.targetCurrentClient[0] = NetworkManager.Singleton.ConnectedClientsIds[this.CurrentPlayerIndex];

return new ClientRpcParams

{

Send = new ClientRpcSendParams { TargetClientIds = this.targetCurrentClient }

};

}

}

public ClientRpcParams ClientParamsHostOtherClients

{

get

{

return new ClientRpcParams

{

Send = new ClientRpcSendParams { TargetClientIds = this.targetHostOtherClients[this.CurrentPlayerIndex] }

};

}

}

public ClientRpcParams ClientParamsClientOtherClients

{

get

{

return new ClientRpcParams

{

Send = new ClientRpcSendParams { TargetClientIds = this.targetClientOtherClients }

};

}

}

public ReadOnlyCollection<MonopolyPlayerVisuals> MonopolyPlayersVisuals { get; private set; }

private void Awake()

{

if (Instance != null)

{

throw new System.InvalidOperationException($"Singleton {this.GetType().FullName} has already been initialized.");

}

Instance = this;

}

private void Start()

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.None, UIManagerMonopolyGame.Instance.MessageWaitingOtherPlayers, PanelMessageBoxUI.Icon.Loading, stateCallback: () => LobbyManager.Instance.HavePlayersLoaded);

this.players = new List<MonopolyPlayer>();

this.MonopolyPlayersVisuals = new ReadOnlyCollection<MonopolyPlayerVisuals>(this.monopolyPlayersVisuals);

if (LobbyManager.Instance.IsHost)

{

this.StartCoroutine(this.WaitOtherPlayersCoroutine());

}

GameCoordinator.Instance?.UpdateInitializedObjects(this.GetType());

}

private void OnEnable()

{

if (NetworkManager.Singleton != null && NetworkManager.Singleton.IsHost)

{

NetworkManager.Singleton.OnClientDisconnectCallback += this.HandleClientDisconnectCallback;

}

}

private void OnDisable()

{

if (NetworkManager.Singleton != null && NetworkManager.Singleton.IsHost)

{

NetworkManager.Singleton.OnClientDisconnectCallback -= this.HandleClientDisconnectCallback;

}

}

#region Callbacks

public MonopolyPlayer GetPlayerById(ulong clientId)

{

return this.players.Where(player => player.OwnerClientId == clientId).FirstOrDefault();

}

public ClientRpcParams GetRedirectionRpc(ulong clientId)

{

this.targetCurrentClient[0] = clientId;

return new ClientRpcParams

{

Send = new ClientRpcSendParams { TargetClientIds = this.targetCurrentClient }

};

}

private void HandleClientDisconnectCallback(ulong surrenderedClientId)

{

if (this.players.Any(player => player.OwnerClientId == surrenderedClientId))

{

this.RemovePlayerServerRpc(surrenderedClientId, this.ServerParamsCurrentClient);

}

}

[ServerRpc]

public void RemovePlayerServerRpc(ulong surrenderedClientId, ServerRpcParams serverRpcParams)

{

bool hasCurrentLeft = false;

if (this.players.Any(player => player.OwnerClientId == surrenderedClientId))

{

int surrenderedPlayerIndex = this.players.IndexOf(this.players.Where(player => player.OwnerClientId == surrenderedClientId).First());

if (this.CurrentPlayer == this.players[surrenderedPlayerIndex])

{

hasCurrentLeft = true;

}

this.players.RemoveAt(surrenderedPlayerIndex);

this.targetHostOtherClients.RemoveAt(surrenderedPlayerIndex);

this.targetClientOtherClients = this.targetClientOtherClients?.Where(clientId => clientId != surrenderedClientId).ToArray();

this.targetHostOtherClients = this.targetHostOtherClients.Select(array => array.Where(id => id != surrenderedClientId).ToArray()).ToList();

if (this.players.Count == 1 && this.players.First().OwnerClientId == NetworkManager.Singleton.LocalClientId && NetworkManager.Singleton.IsConnectedClient)

{

UIManagerMonopolyGame.Instance.HidePaymentProperty();

UIManagerMonopolyGame.Instance.HideButtonRollDice();

UIManagerMonopolyGame.Instance.HidePaymentChance();

UIManagerMonopolyGame.Instance.HideMonopolyNode();

UIManagerMonopolyGame.Instance.HideReceiveTrade();

UIManagerMonopolyGame.Instance.HideTradeOffer();

UIManagerMonopolyGame.Instance.HideOffer();

UIManagerMonopolyGame.Instance.ShowButtonDisconnect();

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageWon, PanelMessageBoxUI.Icon.Trophy);

}

else

{

this.RemovePlayerClientRpc(surrenderedClientId, this.ClientParamsClientOtherClients);

if (hasCurrentLeft)

{

this.SwitchPlayerForcefullyServerRpc(this.ServerParamsCurrentClient);

}

}

}

}

[ClientRpc]

private void RemovePlayerClientRpc(ulong surrenderedClientId, ClientRpcParams clientRpcParams)

{

this.players.Remove(this.players.Where(player => player.OwnerClientId == surrenderedClientId).First());

this.targetClientOtherClients = this.targetClientOtherClients?.Where(clientId => clientId != surrenderedClientId).ToArray();

if (this.players.Count == 1 && this.players.First().OwnerClientId == NetworkManager.Singleton.LocalClientId)

{

UIManagerMonopolyGame.Instance.HidePaymentProperty();

UIManagerMonopolyGame.Instance.HideButtonRollDice();

UIManagerMonopolyGame.Instance.HidePaymentChance();

UIManagerMonopolyGame.Instance.HideMonopolyNode();

UIManagerMonopolyGame.Instance.HideReceiveTrade();

UIManagerMonopolyGame.Instance.HideTradeOffer();

UIManagerMonopolyGame.Instance.HideOffer();

UIManagerMonopolyGame.Instance.ShowButtonDisconnect();

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageWon, PanelMessageBoxUI.Icon.Trophy);

}

}

#endregion

#region Initialization

private IEnumerator WaitOtherPlayersCoroutine()

{

float elapsedTime = 0f;

while (!LobbyManager.Instance.HavePlayersLoaded && elapsedTime < LobbyManager.LOBBY\_LOADING\_TIMEOUT)

{

elapsedTime += Time.deltaTime;

yield return null;

}

if (!LobbyManager.Instance.HavePlayersLoaded)

{

LobbyManager.Instance?.OnMonopolyGameFailedToLoad?.Invoke();

}

else

{

this.InitializeGameServerRpc(this.ServerParamsCurrentClient);

}

}

public void AddPlayer(MonopolyPlayer monopolyPlayer)

{

this.players.Add(monopolyPlayer);

}

[ServerRpc]

private void InitializeGameServerRpc(ServerRpcParams serverRpcParams)

{

this.targetCurrentClient = new ulong[1];

this.targetHostOtherClients = new List<ulong[]>();

this.targetClientOtherClients = NetworkManager.Singleton.ConnectedClientsIds.Where((value) => value != NetworkManager.Singleton.ConnectedClientsIds[0]).ToArray();

for (int i = 0; i < NetworkManager.Singleton?.ConnectedClientsIds.Count; ++i)

{

this.CurrentPlayerIndex = i;

this.targetHostOtherClients.Add(NetworkManager.Singleton.ConnectedClientsIds.Where((value) => value != NetworkManager.Singleton.ConnectedClientsIds[i]).ToArray());

this.SwitchPlayerClientRpc(this.CurrentPlayerIndex, this.ClientParamsClientOtherClients);

this.player = GameObject.Instantiate(this.player);

this.playerPanel = GameObject.Instantiate(this.playerPanel);

this.player.GetComponent<NetworkObject>().SpawnAsPlayerObject(NetworkManager.Singleton.ConnectedClientsIds[i], true);

this.playerPanel.GetComponent<NetworkObject>().SpawnWithOwnership(NetworkManager.Singleton.ConnectedClientsIds[i], true);

}

this.CurrentPlayerIndex = 0;

this.SwitchPlayerClientRpc(this.CurrentPlayerIndex, this.ClientParamsClientOtherClients);

this.CurrentPlayer.PerformTurnClientRpc(this.ClientParamsCurrentClient);

}

#endregion

#region Turn-based Game Loop

[ServerRpc(RequireOwnership = false)]

public void SwitchPlayerServerRpc(ServerRpcParams serverRpcParams)

{

if (this.HasRolledDouble)

{

++this.rolledDoubles;

if (this.rolledDoubles >= this.MaxDoublesInRow)

{

this.rolledDoubles = 0;

this.CurrentPlayer.GoToJailClientRpc(this.ClientParamsCurrentClient);

}

}

else

{

this.rolledDoubles = 0;

this.CurrentPlayerIndex = ++this.CurrentPlayerIndex % this.players.Count;

}

this.SwitchPlayerClientRpc(this.CurrentPlayerIndex, this.ClientParamsClientOtherClients);

UIManagerMonopolyGame.Instance.HideButtonRollDiceClientRpc(this.ClientParamsClientOtherClients);

this.CurrentPlayer.PerformTurnClientRpc(this.ClientParamsCurrentClient);

}

[ServerRpc(RequireOwnership = false)]

public void SwitchPlayerForcefullyServerRpc(ServerRpcParams serverRpcParams)

{

this.rolledDoubles = 0;

this.CurrentPlayerIndex = ++this.CurrentPlayerIndex % this.players.Count;

this.SwitchPlayerClientRpc(this.CurrentPlayerIndex, this.ClientParamsClientOtherClients);

UIManagerMonopolyGame.Instance.HideButtonRollDiceClientRpc(this.ClientParamsClientOtherClients);

this.CurrentPlayer.PerformTurnClientRpc(this.ClientParamsCurrentClient);

}

[ClientRpc]

private void SwitchPlayerClientRpc(int currentPlayerIndex, ClientRpcParams clientRpcParams)

{

this.CurrentPlayerIndex = currentPlayerIndex;

}

#endregion

#region Rolling Dice & Syncing

public void RollDice()

{

const int MIN\_DIE\_VALUE = 1;

const int MAX\_DIE\_VALUE = 6;

this.FirstDieValue = UnityEngine.Random.Range(MIN\_DIE\_VALUE, MAX\_DIE\_VALUE + 1);

this.SecondDieValue = UnityEngine.Random.Range(MIN\_DIE\_VALUE, MAX\_DIE\_VALUE + 1);

this.RollDiceServerRpc(this.FirstDieValue, this.SecondDieValue, this.ServerParamsCurrentClient);

}

[ServerRpc(RequireOwnership = false)]

private void RollDiceServerRpc(int firstDieValue, int secondDieValue, ServerRpcParams serverRpcParams)

{

this.RollDiceClientRpc(firstDieValue, secondDieValue, this.ClientParamsHostOtherClients);

}

[ClientRpc]

private void RollDiceClientRpc(int firstDieValue, int secondDieValue, ClientRpcParams clientRpcParams)

{

this.FirstDieValue = firstDieValue;

this.SecondDieValue = secondDieValue;

}

#endregion

}

**Файл MonopolyBoard.cs**

using UnityEngine;

using System.Collections.Generic;

public sealed class MonopolyBoard : MonoBehaviour

{

#region Setup

#region Special nodes

[Header("Special nodes")]

[Space]

[SerializeField] private MonopolyNode jail;

[Space]

[SerializeField] private MonopolyNode start;

[Space]

[SerializeField] private MonopolyNode sendJail;

[Space]

[SerializeField] private MonopolyNode freeParking;

#endregion

#region Monopolies

[Space]

[Header("Monopolies")]

[Space]

[SerializeField] private List<MonopolySet> monopolies = new List<MonopolySet>();

#endregion

#region Chance & Tax nodes

[Space]

[Header("Chance & Tax nodes")]

[Space]

[SerializeField] private List<ChanceNodeSO> taxNodes = new List<ChanceNodeSO>();

[Space]

[SerializeField] private List<ChanceNodeSO> chanceNodes = new List<ChanceNodeSO>();

#endregion

#endregion

private List<MonopolyNode> nodes;

public static MonopolyBoard Instance { get; private set; }

public List<MonopolySet> Monopolies { get => this.monopolies; }

public int NumberOfNodes { get => this.nodes.Count; }

public MonopolyNode NodeJail { get => this.jail; }

public MonopolyNode NodeStart { get => this.start; }

public MonopolyNode NodeSendToJail { get => this.sendJail; }

public MonopolyNode NodeFreeParking { get => this.freeParking; }

private void Awake()

{

if (Instance != null)

{

throw new System.InvalidOperationException($"Singleton {this.GetType().FullName} has already been initialized.");

}

Instance = this;

}

private void Start()

{

this.nodes = new List<MonopolyNode>();

foreach (Transform child in this.transform)

{

if (child.TryGetComponent(out MonopolyNode monopolyNode))

{

this.nodes.Add(monopolyNode);

}

}

GameCoordinator.Instance?.UpdateInitializedObjects(this.GetType());

}

public MonopolyNode this[int index]

{

get

{

if (index < 0 || index >= this.nodes.Count)

{

throw new System.IndexOutOfRangeException($"{nameof(index)} is out of range.");

}

return this.nodes[index];

}

}

public int this[MonopolyNode monopolyNode]

{

get

{

if (monopolyNode == null)

{

throw new System.NullReferenceException($"{nameof(monopolyNode)} is null.");

}

return this.nodes.IndexOf(monopolyNode);

}

}

public ChanceNodeSO GetTaxNode()

{

return this.taxNodes[UnityEngine.Random.Range(0, this.taxNodes.Count)];

}

public ChanceNodeSO GetChanceNode()

{

return this.chanceNodes[UnityEngine.Random.Range(0, this.chanceNodes.Count)];

}

public MonopolySet GetMonopolySet(MonopolyNode monopolyNode)

{

if (monopolyNode == null)

{

throw new System.ArgumentNullException($"{nameof(monopolyNode)} is null.");

}

foreach (MonopolySet monopolySet in this.monopolies)

{

if (monopolySet.Contains(monopolyNode))

{

return monopolySet;

}

}

return null;

}

public int GetDistance(int fromNodeIndex, int toNodeIndex)

{

int clockwiseDistance = (toNodeIndex - fromNodeIndex + this.NumberOfNodes) % this.NumberOfNodes;

int counterclockwiseDistance = (fromNodeIndex - toNodeIndex + this.NumberOfNodes) % this.NumberOfNodes;

return Mathf.Min(clockwiseDistance, counterclockwiseDistance) == counterclockwiseDistance ? -counterclockwiseDistance : clockwiseDistance;

}

public int GetDistance(MonopolyNode fromNode, MonopolyNode toNode)

{

int clockwiseDistance = (this[toNode] - this[fromNode] + this.NumberOfNodes) % this.NumberOfNodes;

int counterclockwiseDistance = (this[fromNode] - this[toNode] + this.NumberOfNodes) % this.NumberOfNodes;

return Mathf.Min(clockwiseDistance, counterclockwiseDistance) == counterclockwiseDistance ? -counterclockwiseDistance : clockwiseDistance;

}

}

**Файл MonopolyNode.cs**

using System.Linq;

using UnityEngine;

using Unity.Netcode;

using UnityEngine.UI;

using System.Collections.Generic;

public sealed class MonopolyNode : NetworkBehaviour

{

#region Setup (Editor)

[SerializeField] private Type type;

[SerializeField] private Image imageLogo;

[SerializeField] private Sprite spriteLogo;

[SerializeField] private Image imageOwner;

[SerializeField] private Image imageMonopolyType;

[SerializeField] private Image imageMortgageStatus;

[SerializeField] private Image imageLevel1;

[SerializeField] private Image imageLevel2;

[SerializeField] private Image imageLevel3;

[SerializeField] private Image imageLevel4;

[SerializeField] private Image imageLevel5;

[SerializeField] private int pricePurchase;

[SerializeField] private int priceUpgrade;

[SerializeField] private List<int> pricesRent = new List<int>();

#endregion

public enum Type : byte

{

Tax,

Jail,

Start,

Chance,

SendJail,

Property,

Gambling,

Transport,

FreeParking

}

private const int LEVEL\_MORTGAGE = 0;

private const int LEVEL\_OWNERSHIP = 1;

public const int PROPERTY\_MIN\_LEVEL = 0;

public const int PROPERTY\_MAX\_LEVEL = 6;

public NetworkVariable<int> Level { get; private set; }

public Type NodeType

{

get => this.type;

}

public int PriceRent

{

get

{

return this.pricesRent[this.LocalLevel] \* (this.NodeType == MonopolyNode.Type.Gambling ? GameManager.Instance.TotalRollResult : 1);

}

}

public bool IsMortgaged

{

get => this.LocalLevel == 0;

}

public int PriceUpgrade

{

get

{

if (this.NodeType == MonopolyNode.Type.Property)

{

return this.LocalLevel == 0 ? this.pricePurchase : this.priceUpgrade;

}

else

{

return this.pricePurchase;

}

}

}

public int PricePurchase

{

get => this.pricePurchase;

}

public int PriceDowngrade

{

get

{

if (this.NodeType == MonopolyNode.Type.Property)

{

return this.LocalLevel == 1 ? this.pricePurchase : this.priceUpgrade;

}

else

{

return this.pricePurchase;

}

}

}

public Sprite NodeSprite

{

get => this.spriteLogo;

}

public bool IsTradable

{

get

{

if (this.NodeType == MonopolyNode.Type.Property)

{

return this.LocalLevel == MonopolyNode.LEVEL\_OWNERSHIP ? true : false;

}

else if (this.NodeType == MonopolyNode.Type.Transport || this.NodeType == MonopolyNode.Type.Gambling)

{

return this.LocalLevel > MonopolyNode.LEVEL\_MORTGAGE ? true : false;

}

else

{

return false;

}

}

}

public bool IsUpgradable

{

get

{

if (this.NodeType == MonopolyNode.Type.Property)

{

bool isEquallySpread = this.AffiliatedMonopoly.NodesInSet.All(node => node.LocalLevel >= this.LocalLevel);

return (isEquallySpread && this.LocalLevel < MonopolyNode.PROPERTY\_MAX\_LEVEL) || this.IsMortgaged;

}

else if (this.NodeType == MonopolyNode.Type.Transport || this.NodeType == MonopolyNode.Type.Gambling)

{

if (this.LocalLevel == MonopolyNode.LEVEL\_MORTGAGE)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

}

public bool IsDowngradable

{

get

{

if (this.NodeType == MonopolyNode.Type.Property)

{

bool isEquallySpread = this.AffiliatedMonopoly.NodesInSet.All(node => node.LocalLevel <= this.LocalLevel);

return isEquallySpread && this.LocalLevel > MonopolyNode.PROPERTY\_MIN\_LEVEL;

}

else if (this.NodeType == MonopolyNode.Type.Gambling || this.NodeType == MonopolyNode.Type.Transport)

{

return this.LocalLevel > MonopolyNode.PROPERTY\_MIN\_LEVEL ? true : false;

}

else

{

return false;

}

}

}

public int LocalLevel { get; private set; }

public MonopolyPlayer Owner { get; private set; }

public MonopolySet AffiliatedMonopoly { get; private set; }

private void Awake()

{

this.imageLogo.sprite = this.spriteLogo;

this.Level = new NetworkVariable<int>(1, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Server);

switch (this.NodeType)

{

case MonopolyNode.Type.Property:

case MonopolyNode.Type.Gambling:

case MonopolyNode.Type.Transport:

{

this.LocalLevel = 1;

this.AffiliatedMonopoly = MonopolyBoard.Instance.GetMonopolySet(this);

this.imageMonopolyType.color = this.AffiliatedMonopoly.ColorOfSet;

}

break;

}

}

private void OnEnable()

{

this.Level.OnValueChanged += this.HandleLevelChanged;

}

private void OnDisable()

{

this.Level.OnValueChanged -= this.HandleLevelChanged;

}

#region Visuals

private void UpdateVisualsSpecial()

{

if (this.Owner == null)

{

this.imageOwner.gameObject.SetActive(false);

this.imageMortgageStatus.gameObject.SetActive(false);

}

else

{

if (this.LocalLevel == MonopolyNode.LEVEL\_MORTGAGE)

{

this.imageMortgageStatus.gameObject.SetActive(true);

}

else

{

this.imageMortgageStatus.gameObject.SetActive(false);

this.imageOwner.gameObject.SetActive(true);

this.imageOwner.color = this.Owner.PlayerColor;

}

}

}

private void UpdateVisualsProperty()

{

if (this.Owner == null)

{

this.imageOwner.gameObject.SetActive(false);

this.imageLevel1.gameObject.SetActive(false);

this.imageLevel2.gameObject.SetActive(false);

this.imageLevel3.gameObject.SetActive(false);

this.imageLevel4.gameObject.SetActive(false);

this.imageLevel5.gameObject.SetActive(false);

this.imageMortgageStatus.gameObject.SetActive(false);

}

else

{

switch (this.LocalLevel)

{

case MonopolyNode.LEVEL\_MORTGAGE:

{

this.imageMortgageStatus.gameObject.SetActive(true);

}

break;

case MonopolyNode.LEVEL\_OWNERSHIP:

{

this.imageOwner.gameObject.SetActive(true);

this.imageOwner.color = this.Owner.PlayerColor;

this.imageLevel1.gameObject.SetActive(false);

this.imageMortgageStatus.gameObject.SetActive(false);

}

break;

case 2:

{

this.imageLevel1.gameObject.SetActive(true);

this.imageLevel2.gameObject.SetActive(false);

}

break;

case 3:

{

this.imageLevel2.gameObject.SetActive(true);

this.imageLevel3.gameObject.SetActive(false);

}

break;

case 4:

{

this.imageLevel3.gameObject.SetActive(true);

this.imageLevel4.gameObject.SetActive(false);

}

break;

case 5:

{

this.imageLevel1.gameObject.SetActive(true);

this.imageLevel2.gameObject.SetActive(true);

this.imageLevel3.gameObject.SetActive(true);

this.imageLevel4.gameObject.SetActive(true);

this.imageLevel5.gameObject.SetActive(false);

}

break;

case 6:

{

this.imageLevel5.gameObject.SetActive(true);

this.imageLevel1.gameObject.SetActive(false);

this.imageLevel2.gameObject.SetActive(false);

this.imageLevel3.gameObject.SetActive(false);

this.imageLevel4.gameObject.SetActive(false);

}

break;

}

}

}

#endregion

#region Ownership

public void ResetOwnership()

{

this.Owner = null;

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

this.ResetOwnershipServerRpc(GameManager.Instance.ServerParamsCurrentClient);

}

public void UpdateOwnership(ulong ownerId)

{

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

this.Owner = GameManager.Instance.GetPlayerById(ownerId);

if (this.Owner == null)

{

return;

}

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

this.UpdateOwnershipServerRpc(ownerId, GameManager.Instance.ServerParamsCurrentClient);

if (this.NodeType != MonopolyNode.Type.Transport && this.NodeType != MonopolyNode.Type.Gambling)

{

return;

}

if (!this.Owner.HasPartialMonopoly(this, out \_))

{

return;

}

foreach (MonopolyNode node in this.AffiliatedMonopoly.OwnedByPlayerNodes)

{

if (!node.IsMortgaged)

{

while (node.LocalLevel < this.AffiliatedMonopoly.OwnedByPlayerCount)

{

node.Upgrade();

}

}

}

}

[ServerRpc(RequireOwnership = false)]

public void ResetOwnershipServerRpc(ServerRpcParams serverRpcParams)

{

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

this.Level.Value = MonopolyNode.LEVEL\_OWNERSHIP;

if (this.Owner != null)

{

this.Owner = null;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

}

this.ResetOwnershipClientRpc(GameManager.Instance.ClientParamsClientOtherClients);

}

[ClientRpc]

private void ResetOwnershipClientRpc(ClientRpcParams clientRpcParams)

{

if (this.Owner != null)

{

this.Owner = null;

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

}

}

[ServerRpc(RequireOwnership = false)]

private void UpdateOwnershipServerRpc(ulong ownerId, ServerRpcParams serverRpcParams)

{

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

this.Level.Value = MonopolyNode.LEVEL\_OWNERSHIP;

MonopolyPlayer playerOwner = GameManager.Instance.GetPlayerById(ownerId);

if (this.Owner != playerOwner)

{

this.Owner = playerOwner;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

}

this.UpdateOwnershipClientRpc(ownerId, GameManager.Instance.ClientParamsClientOtherClients);

}

[ClientRpc]

private void UpdateOwnershipClientRpc(ulong ownerId, ClientRpcParams clientRpcParams)

{

MonopolyPlayer playerOwner = GameManager.Instance.GetPlayerById(ownerId);

if (this.Owner != playerOwner)

{

this.Owner = playerOwner;

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

}

}

#endregion

#region Upgrade/Downgrade

public void Upgrade()

{

if (this.NodeType == MonopolyNode.Type.Property)

{

++this.LocalLevel;

this.UpdateVisualsProperty();

this.ChangeLevelServerRpc(this.LocalLevel, GameManager.Instance.ServerParamsCurrentClient);

}

else

{

this.LocalLevel = MonopolyNode.LEVEL\_OWNERSHIP;

this.UpdateVisualsSpecial();

this.ChangeLevelServerRpc(this.LocalLevel, GameManager.Instance.ServerParamsCurrentClient);

if (this.Owner.HasPartialMonopoly(this, out \_))

{

this.LocalLevel = this.AffiliatedMonopoly.OwnedByPlayerCount;

this.ChangeLevelServerRpc(this.LocalLevel, GameManager.Instance.ServerParamsCurrentClient);

}

}

}

public void Downgrade()

{

if (this.NodeType == MonopolyNode.Type.Property)

{

--this.LocalLevel;

this.UpdateVisualsProperty();

}

else

{

this.LocalLevel = MonopolyNode.LEVEL\_MORTGAGE;

this.UpdateVisualsSpecial();

}

this.ChangeLevelServerRpc(this.LocalLevel, GameManager.Instance.ServerParamsCurrentClient);

}

[ServerRpc(RequireOwnership = false)]

public void ChangeLevelServerRpc(int level, ServerRpcParams serverRpcParams)

{

this.Level.Value = level;

}

#endregion

private void HandleLevelChanged(int previousValue, int newValue)

{

this.LocalLevel = newValue;

if (this.NodeType == MonopolyNode.Type.Property)

{

this.UpdateVisualsProperty();

}

else

{

this.UpdateVisualsSpecial();

}

}

}

**Файл MonopolyPlayer.cs**

using System;

using System.Linq;

using UnityEngine;

using Unity.Netcode;

using UnityEngine.UI;

using System.Collections;

using System.Collections.Generic;

public sealed class MonopolyPlayer : NetworkBehaviour

{

#region Setup

#region Visuals

[Header("Visuals")]

[Space]

[SerializeField] private Image playerImageToken;

#endregion

#endregion

private bool isInJail;

private int turnsInJail;

private bool isSkipTurn;

public Action OnBalanceUpdated;

public bool IsTrading { get; set; }

public bool HasBuilt { get; private set; }

public bool HasRolled { get; private set; }

public string Nickname { get; private set; }

public bool IsAbleToBuild { get; private set; }

public bool HasCompletedTurn { get; private set; }

public Color PlayerColor { get; private set; }

public MonopolyNode SelectedNode { get; set; }

public MonopolyNode CurrentNode { get; private set; }

public MonopolyPlayer PlayerTradingWith { get; set; }

public NetworkVariable<int> Balance { get; private set; }

public List<MonopolyNode> OwnedNodes { get; private set; }

public ChanceNodeSO CurrentChanceNode { get; private set; }

private void Awake()

{

this.Balance = new NetworkVariable<int>(GameManager.Instance.StartingBalance, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Owner);

}

public override void OnNetworkSpawn()

{

this.OwnedNodes = new List<MonopolyNode>();

this.Balance.Value = GameManager.Instance.StartingBalance;

this.CurrentNode = MonopolyBoard.Instance.NodeStart;

this.transform.position = MonopolyBoard.Instance.NodeStart.transform.position;

this.PlayerColor = GameManager.Instance.MonopolyPlayersVisuals[GameManager.Instance.CurrentPlayerIndex].ColorPlayerToken;

this.playerImageToken.sprite = GameManager.Instance.MonopolyPlayersVisuals[GameManager.Instance.CurrentPlayerIndex].SpritePlayerToken;

this.Nickname = LobbyManager.Instance.LocalLobby.Players[GameManager.Instance.CurrentPlayerIndex].Data[LobbyManager.KEY\_PLAYER\_NICKNAME].Value;

GameManager.Instance.AddPlayer(this);

this.Balance.OnValueChanged += this.HandleBalanceChanged;

if (this.OwnerClientId == NetworkManager.Singleton?.LocalClientId)

{

UIManagerMonopolyGame.Instance.ButtonRollDiceClicked += this.HandleButtonRollDiceClicked;

}

}

public override void OnNetworkDespawn()

{

this.Balance.OnValueChanged -= this.HandleBalanceChanged;

if (this.OwnerClientId == NetworkManager.Singleton?.LocalClientId)

{

this.Surrender();

UIManagerMonopolyGame.Instance.ButtonRollDiceClicked -= this.HandleButtonRollDiceClicked;

}

}

#region Monopoly

public bool HasFullMonopoly(MonopolyNode monopolyNode, out MonopolySet monopolySet)

{

monopolySet = MonopolyBoard.Instance.GetMonopolySet(monopolyNode);

return monopolySet?.NodesInSet.Intersect(this.OwnedNodes).Count() == monopolySet.NodesInSet.Count;

}

public bool HasPartialMonopoly(MonopolyNode monopolyNode, out MonopolySet monopolySet)

{

monopolySet = MonopolyBoard.Instance.GetMonopolySet(monopolyNode);

return monopolySet?.NodesInSet.Intersect(this.OwnedNodes).Count() > 1;

}

#endregion

#region Movement

private void Move(int steps)

{

this.IsAbleToBuild = false;

const float POSITION\_THRESHOLD = 0.01f;

Vector3 targetPosition;

bool movedOverStart = false;

int currentNodeIndex = MonopolyBoard.Instance[this.CurrentNode];

this.StartCoroutine(MoveCoroutine());

IEnumerator MoveCoroutine()

{

while (steps != 0)

{

if (steps < 0)

{

++steps;

currentNodeIndex = Mathf.Abs(--currentNodeIndex + MonopolyBoard.Instance.NumberOfNodes);

currentNodeIndex = currentNodeIndex % MonopolyBoard.Instance.NumberOfNodes;

}

else

{

--steps;

currentNodeIndex = ++currentNodeIndex % MonopolyBoard.Instance.NumberOfNodes;

}

targetPosition = MonopolyBoard.Instance[currentNodeIndex].transform.position;

if (MonopolyBoard.Instance.NodeStart == MonopolyBoard.Instance[currentNodeIndex])

{

movedOverStart = true;

}

yield return StartCoroutine(MoveStepCoroutine(targetPosition));

}

this.CurrentNode = MonopolyBoard.Instance[currentNodeIndex];

if (movedOverStart && this.CurrentNode != MonopolyBoard.Instance.NodeStart)

{

this.Balance.Value += GameManager.Instance.CircleBonus;

}

this.HandleLanding();

}

IEnumerator MoveStepCoroutine(Vector3 targetPosition)

{

while (Vector3.Distance(this.transform.position, targetPosition) > POSITION\_THRESHOLD)

{

this.transform.position = Vector3.MoveTowards(this.transform.position, targetPosition, GameManager.Instance.PlayerMovementSpeed \* Time.deltaTime);

yield return null;

}

this.transform.position = targetPosition;

}

}

private void HandleLanding()

{

switch (this.CurrentNode.NodeType)

{

case MonopolyNode.Type.Tax:

this.HandleChanceLanding();

break;

case MonopolyNode.Type.Jail:

this.HandleJailLanding();

break;

case MonopolyNode.Type.Start:

this.HandleStartLanding();

break;

case MonopolyNode.Type.Chance:

this.HandleChanceLanding();

break;

case MonopolyNode.Type.SendJail:

this.HandleSendJailLanding();

break;

case MonopolyNode.Type.Property:

this.HandlePropertyLanding();

break;

case MonopolyNode.Type.Gambling:

this.HandlePropertyLanding();

break;

case MonopolyNode.Type.Transport:

this.HandlePropertyLanding();

break;

case MonopolyNode.Type.FreeParking:

this.HandleFreeParkingLanding();

break;

}

}

private IEnumerator PerformTurnCoroutine()

{

yield return new WaitUntil(() => this.HasCompletedTurn);

if (this.isInJail)

{

GameManager.Instance.SwitchPlayerForcefullyServerRpc(GameManager.Instance.ServerParamsCurrentClient);

}

else

{

GameManager.Instance.SwitchPlayerServerRpc(GameManager.Instance.ServerParamsCurrentClient);

}

}

[ClientRpc]

public void PerformTurnClientRpc(ClientRpcParams clientRpcParams)

{

this.HasBuilt = false;

this.IsTrading = false;

this.HasRolled = false;

this.IsAbleToBuild = true;

this.HasCompletedTurn = false;

this.CurrentChanceNode = null;

this.PlayerTradingWith = null;

this.StartCoroutine(this.PerformTurnCoroutine());

if (this.isSkipTurn)

{

this.isSkipTurn = false;

this.HasCompletedTurn = true;

return;

}

UIManagerMonopolyGame.Instance.ShowButtonRollDice();

}

#endregion

#region Utility

public void GoToJail()

{

this.isInJail = true;

this.turnsInJail = 0;

this.Move(MonopolyBoard.Instance.GetDistance(this.CurrentNode, MonopolyBoard.Instance.NodeJail));

if (this.CurrentChanceNode != null)

{

this.CurrentChanceNode = null;

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageSentJail, PanelMessageBoxUI.Icon.Warning);

}

}

public void Surrender()

{

this.DeclineTradeServerRpc(GameManager.Instance.ServerParamsCurrentClient);

UIManagerMonopolyGame.Instance.HidePaymentProperty();

UIManagerMonopolyGame.Instance.HideButtonRollDice();

UIManagerMonopolyGame.Instance.HidePaymentChance();

UIManagerMonopolyGame.Instance.HideMonopolyNode();

UIManagerMonopolyGame.Instance.HideReceiveTrade();

UIManagerMonopolyGame.Instance.HideTradeOffer();

UIManagerMonopolyGame.Instance.HideOffer();

foreach (MonopolyNode node in this.OwnedNodes)

{

node.ResetOwnership();

}

this.SurrenderServerRpc(GameManager.Instance.ServerParamsCurrentClient);

}

private void ReleaseFromJail()

{

this.turnsInJail = 0;

this.isInJail = false;

}

public void HandleJailLanding()

{

this.HasCompletedTurn = true;

}

public void HandleStartLanding()

{

this.Balance.Value += GameManager.Instance.ExactCircleBonus;

this.HasCompletedTurn = true;

}

public void HandleChanceLanding()

{

this.CurrentChanceNode = MonopolyBoard.Instance.GetChanceNode();

if (this.CurrentChanceNode.ChanceType != ChanceNodeSO.Type.Penalty)

{

UIManagerMonopolyGame.Instance.ShowInfo(this.CurrentChanceNode.Description, this.CallbackChance);

}

else

{

UIManagerMonopolyGame.Instance.ShowPaymentChance(this.CurrentChanceNode.Description, this.CallbackPayment);

}

UIManagerMonopolyGame.Instance.ShowInfoServerRpc(this.CurrentChanceNode.Description, GameManager.Instance.ServerParamsCurrentClient);

}

public void HandleSendJailLanding()

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageSentJail, PanelMessageBoxUI.Icon.Warning);

this.GoToJail();

}

public void HandleFreeParkingLanding()

{

this.HasCompletedTurn = true;

}

[ClientRpc]

public void GoToJailClientRpc(ClientRpcParams clientRpcParams)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageSentJail, PanelMessageBoxUI.Icon.Warning);

this.GoToJail();

}

[ServerRpc(RequireOwnership = false)]

private void SurrenderServerRpc(ServerRpcParams serverRpcParams)

{

GameManager.Instance.RemovePlayerServerRpc(serverRpcParams.Receive.SenderClientId, GameManager.Instance.ServerParamsCurrentClient);

if (NetworkManager.Singleton.ConnectedClients.ContainsKey(serverRpcParams.Receive.SenderClientId))

{

NetworkClient client = NetworkManager.Singleton.ConnectedClients[serverRpcParams.Receive.SenderClientId];

foreach (NetworkObject ownedObject in client.OwnedObjects)

{

if (!(bool)ownedObject.IsSceneObject && ownedObject.IsSpawned)

{

ownedObject.Despawn();

}

}

}

}

#endregion

#region Property

private void UpgradeProperty()

{

if (this.SelectedNode.NodeType == MonopolyNode.Type.Transport || this.SelectedNode.NodeType == MonopolyNode.Type.Gambling)

{

if (!this.SelectedNode.IsMortgaged)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageCannotUpgradeMaxLevel, PanelMessageBoxUI.Icon.Warning);

}

else

{

if (this.Balance.Value >= this.SelectedNode.PriceUpgrade)

{

UIManagerMonopolyGame.Instance.HideMonopolyNode();

this.Balance.Value -= this.SelectedNode.PriceUpgrade;

this.HasBuilt = true;

this.SelectedNode.Upgrade();

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageInsufficientFunds, PanelMessageBoxUI.Icon.Warning);

}

}

}

else if (this.SelectedNode.NodeType == MonopolyNode.Type.Property)

{

if (!this.HasFullMonopoly(this.SelectedNode, out \_) && !this.SelectedNode.IsMortgaged)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageCompleteMonopolyRequired, PanelMessageBoxUI.Icon.Warning);

}

else if (this.HasBuilt)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageAlreadyBuilt, PanelMessageBoxUI.Icon.Warning);

}

else if (!this.SelectedNode.IsUpgradable)

{

if (this.SelectedNode.LocalLevel == MonopolyNode.PROPERTY\_MAX\_LEVEL)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageCannotUpgradeMaxLevel, PanelMessageBoxUI.Icon.Warning);

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageOnlyEvenBuildingAllowed, PanelMessageBoxUI.Icon.Warning);

}

}

else

{

if (this.Balance.Value >= this.SelectedNode.PriceUpgrade)

{

UIManagerMonopolyGame.Instance.HideMonopolyNode();

this.Balance.Value -= this.SelectedNode.PriceUpgrade;

this.HasBuilt = true;

this.SelectedNode.Upgrade();

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageInsufficientFunds, PanelMessageBoxUI.Icon.Warning);

}

}

}

}

private void DowngradeProperty()

{

if (!this.SelectedNode.IsDowngradable)

{

if (this.SelectedNode.LocalLevel == MonopolyNode.PROPERTY\_MIN\_LEVEL)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageCannotDowngradeMinLevel, PanelMessageBoxUI.Icon.Warning);

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageOnlyEvenBuildingAllowed, PanelMessageBoxUI.Icon.Warning);

}

}

else

{

UIManagerMonopolyGame.Instance.HideMonopolyNode();

this.Balance.Value += this.SelectedNode.PriceDowngrade;

this.SelectedNode.Downgrade();

}

}

public void CallbackMonopolyNode()

{

if (UIManagerMonopolyGame.Instance.PanelMonopolyNode.MonopolyNodeDialogResult == PanelMonopolyNodeUI.DialogResult.Upgrade)

{

this.UpgradeProperty();

}

else

{

this.DowngradeProperty();

}

}

private void HandlePropertyLanding()

{

if (this.CurrentNode.Owner == null)

{

UIManagerMonopolyGame.Instance.ShowOffer(this.CurrentNode.NodeSprite, this.CurrentNode.AffiliatedMonopoly.ColorOfSet, this.CurrentNode.PricePurchase, this.CallbackPropertyOffer);

}

else if (this.CurrentNode.Owner == this || this.CurrentNode.IsMortgaged)

{

this.HasCompletedTurn = true;

}

else

{

UIManagerMonopolyGame.Instance.ShowPaymentProperty(this.CurrentNode.NodeSprite, this.CurrentNode.AffiliatedMonopoly.ColorOfSet, this.CurrentNode.PriceRent, this.CallbackPayment);

}

}

private void CallbackPropertyOffer()

{

if (UIManagerMonopolyGame.Instance.PanelOffer.OfferDialogResult == PanelOfferUI.DialogResult.Accepted)

{

if (this.Balance.Value >= this.CurrentNode.PricePurchase)

{

UIManagerMonopolyGame.Instance.HideOffer();

this.OwnedNodes.Add(this.CurrentNode);

this.CurrentNode.UpdateOwnership(NetworkManager.Singleton.LocalClientId);

this.Balance.Value -= this.CurrentNode.PricePurchase;

this.HasCompletedTurn = true;

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageInsufficientFunds, PanelMessageBoxUI.Icon.Warning);

}

}

else

{

UIManagerMonopolyGame.Instance.HideOffer();

this.HasCompletedTurn = true;

}

}

[ClientRpc]

private void AddNodeClientRpc(int monopolyNodeIndex, ClientRpcParams clientRpcParams)

{

this.OwnedNodes.Add(MonopolyBoard.Instance[monopolyNodeIndex]);

}

[ClientRpc]

private void RemoveNodeClientRpc(int monopolyNodeIndex, ClientRpcParams clientRpcParams)

{

this.OwnedNodes.Remove(MonopolyBoard.Instance[monopolyNodeIndex]);

}

[ServerRpc(RequireOwnership = false)]

private void AddNodeServerRpc(int monopolyNodeIndex, ulong ownerdId, ServerRpcParams serverRpcParams)

{

this.AddNodeClientRpc(monopolyNodeIndex, GameManager.Instance.GetRedirectionRpc(ownerdId));

}

[ServerRpc(RequireOwnership = false)]

private void RemoveNodeServerRpc(int monopolyNodeIndex, ulong ownerdId, ServerRpcParams serverRpcParams)

{

this.RemoveNodeClientRpc(monopolyNodeIndex, GameManager.Instance.GetRedirectionRpc(ownerdId));

}

#endregion

#region GUI Callbacks

private void CallbackChance()

{

if (UIManagerMonopolyGame.Instance.PanelInfo.InfoDialogResult == PanelInfoUI.DialogResult.Confirmed)

{

switch (this.CurrentChanceNode.ChanceType)

{

case ChanceNodeSO.Type.Reward:

{

this.Balance.Value += this.CurrentChanceNode.Reward;

this.HasCompletedTurn = true;

}

break;

case ChanceNodeSO.Type.SkipTurn:

{

this.isSkipTurn = true;

this.HasCompletedTurn = true;

}

break;

case ChanceNodeSO.Type.SendJail:

this.GoToJail();

break;

case ChanceNodeSO.Type.MoveForward:

{

this.CurrentChanceNode = null;

GameManager.Instance.RollDice();

UIManagerMonopolyGame.Instance.ShowDiceAnimation();

this.Move(GameManager.Instance.TotalRollResult);

}

break;

case ChanceNodeSO.Type.MoveBackwards:

{

this.CurrentChanceNode = null;

GameManager.Instance.RollDice();

UIManagerMonopolyGame.Instance.ShowDiceAnimation();

this.Move(-GameManager.Instance.TotalRollResult);

}

break;

}

}

}

private void CallbackPayment()

{

if (this.CurrentNode.NodeType == MonopolyNode.Type.Chance || this.CurrentNode.NodeType == MonopolyNode.Type.Tax)

{

if (this.Balance.Value >= this.CurrentChanceNode.Penalty)

{

UIManagerMonopolyGame.Instance.HidePaymentChance();

this.Balance.Value -= this.CurrentChanceNode.Penalty;

this.HasCompletedTurn = true;

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageInsufficientFunds, PanelMessageBoxUI.Icon.Warning);

}

}

else

{

if (this.Balance.Value >= this.CurrentNode.PriceRent)

{

UIManagerMonopolyGame.Instance.HidePaymentProperty();

this.Balance.Value -= this.CurrentNode.PriceRent;

this.SendBalanceServerRpc(this.CurrentNode.PriceRent, this.CurrentNode.Owner.OwnerClientId, GameManager.Instance.ServerParamsCurrentClient);

this.HasCompletedTurn = true;

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageInsufficientFunds, PanelMessageBoxUI.Icon.Warning);

}

}

}

public void CallbackTradeOffer()

{

if (UIManagerMonopolyGame.Instance.PanelTradeOffer.TradeOfferDialogResult == PanelTradeOfferUI.DialogResult.Offer)

{

UIManagerMonopolyGame.Instance.SendTradeOffer();

}

else

{

this.IsTrading = false;

if (!this.HasRolled)

{

UIManagerMonopolyGame.Instance.ShowButtonRollDice();

}

}

}

public void CallbackReceiveTrade()

{

UIManagerMonopolyGame.Instance.HideReceiveTrade();

if (UIManagerMonopolyGame.Instance.PanelReceiveTrade.ReceiveTradeDialogResult == PanelReceiveTradeUI.DialogResult.Accept)

{

this.AcceptTradeServerRpc(UIManagerMonopolyGame.Instance.PanelReceiveTrade.Credentials, GameManager.Instance.ServerParamsCurrentClient);

}

else

{

this.DeclineTradeServerRpc(GameManager.Instance.ServerParamsCurrentClient);

}

}

private void HandleButtonRollDiceClicked()

{

this.HasRolled = true;

UIManagerMonopolyGame.Instance.HidePaymentProperty();

UIManagerMonopolyGame.Instance.HidePaymentChance();

UIManagerMonopolyGame.Instance.HideMonopolyNode();

UIManagerMonopolyGame.Instance.HideReceiveTrade();

UIManagerMonopolyGame.Instance.HideTradeOffer();

UIManagerMonopolyGame.Instance.HideOffer();

GameManager.Instance.RollDice();

UIManagerMonopolyGame.Instance.ShowDiceAnimation();

if (this.isInJail)

{

if (GameManager.Instance.HasRolledDouble || ++this.turnsInJail > GameManager.Instance.MaxTurnsInJail)

{

this.ReleaseFromJail();

this.Move(GameManager.Instance.TotalRollResult);

}

else

{

this.HasCompletedTurn = true;

}

}

else

{

this.Move(GameManager.Instance.TotalRollResult);

}

}

[ServerRpc(RequireOwnership = false)]

public void DeclineTradeServerRpc(ServerRpcParams serverRpcParams)

{

this.CallbackTradeResponseClientRpc(false, GameManager.Instance.ClientParamsCurrentClient);

}

[ClientRpc]

private void CallbackTradeResponseClientRpc(bool result, ClientRpcParams clientRpcParams)

{

if (GameManager.Instance.CurrentPlayer == null || !GameManager.Instance.CurrentPlayer.IsTrading)

{

return;

}

GameManager.Instance.CurrentPlayer.IsTrading = false;

if (result)

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageTradeAccepted, PanelMessageBoxUI.Icon.Warning);

}

else

{

UIManagerGlobal.Instance.ShowMessageBox(PanelMessageBoxUI.Type.OK, UIManagerMonopolyGame.Instance.MessageTradeDeclined, PanelMessageBoxUI.Icon.Warning);

}

if (!GameManager.Instance.CurrentPlayer.HasRolled)

{

UIManagerMonopolyGame.Instance.ShowButtonRollDice();

}

}

[ServerRpc]

public void AcceptTradeServerRpc(TradeCredentials tradeCredentials, ServerRpcParams serverRpcParams)

{

UIManagerMonopolyGame.Instance.HideMonopolyNode();

MonopolyPlayer sender = GameManager.Instance.GetPlayerById(tradeCredentials.SenderId);

MonopolyPlayer receiver = GameManager.Instance.GetPlayerById(tradeCredentials.ReceiverId);

if (sender == null || receiver == null)

{

return;

}

if (tradeCredentials.SenderNodeIndex != -1)

{

MonopolyBoard.Instance[tradeCredentials.SenderNodeIndex].UpdateOwnership(tradeCredentials.ReceiverId);

receiver.AddNodeServerRpc(tradeCredentials.SenderNodeIndex, tradeCredentials.ReceiverId, GameManager.Instance.ServerParamsCurrentClient);

sender.RemoveNodeServerRpc(tradeCredentials.SenderNodeIndex, tradeCredentials.SenderId, GameManager.Instance.ServerParamsCurrentClient);

}

if (tradeCredentials.ReceiverNodeIndex != -1)

{

MonopolyBoard.Instance[tradeCredentials.ReceiverNodeIndex].UpdateOwnership(tradeCredentials.SenderId);

sender.AddNodeServerRpc(tradeCredentials.ReceiverNodeIndex, tradeCredentials.SenderId, GameManager.Instance.ServerParamsCurrentClient);

receiver.RemoveNodeServerRpc(tradeCredentials.ReceiverNodeIndex, tradeCredentials.ReceiverId, GameManager.Instance.ServerParamsCurrentClient);

}

if (tradeCredentials.SenderOffer != 0)

{

sender.SendBalanceServerRpc(tradeCredentials.SenderOffer, receiver.OwnerClientId, GameManager.Instance.ServerParamsCurrentClient);

GameManager.Instance.GetPlayerById(tradeCredentials.SenderId).PayChargeClientRpc(tradeCredentials.SenderOffer, GameManager.Instance.GetRedirectionRpc(tradeCredentials.SenderId));

}

if (tradeCredentials.ReceiverOffer != 0)

{

sender.SendBalanceServerRpc(tradeCredentials.ReceiverOffer, sender.OwnerClientId, GameManager.Instance.ServerParamsCurrentClient);

GameManager.Instance.GetPlayerById(tradeCredentials.ReceiverId).PayChargeClientRpc(tradeCredentials.ReceiverOffer, GameManager.Instance.GetRedirectionRpc(tradeCredentials.ReceiverId));

}

this.CallbackTradeResponseClientRpc(true, GameManager.Instance.ClientParamsCurrentClient);

}

#endregion

#region Updating Balance

private void HandleBalanceChanged(int previousValue, int newValue)

{

this.OnBalanceUpdated?.Invoke();

}

[ClientRpc]

private void PayChargeClientRpc(int amount, ClientRpcParams clientRpcParams)

{

this.Balance.Value -= amount;

}

[ServerRpc(RequireOwnership = false)]

private void SendBalanceServerRpc(int amount, ulong receiverClientId, ServerRpcParams serverRpcParams)

{

this.ReceiveBalanceClientRpc(amount, receiverClientId, GameManager.Instance.GetRedirectionRpc(receiverClientId));

}

[ClientRpc]

private void ReceiveBalanceClientRpc(int amount, ulong receiverClientId, ClientRpcParams clientRpcParams)

{

GameManager.Instance.GetPlayerById(receiverClientId).Balance.Value += amount;

}

#endregion

}