ออกแบบ server โดยสร้างรูปร่าง เขียนโค้ด ออกแบบระบบเกมต่างๆ เพื่อให้ client รองรับระบบที่เสถียรและแสดงผลออกมา
เริ่มจากสร้างไฟล์ในยูนิตี้สร้าง client กับ server และไปที่ไฟล์ server
สร้างไฟล์ Controller
เขียนโค้ด Server Controller

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

Unity Script (1 asset reference) | 7 references
public class ServerController : MonoBehaviour
{
    [SerializeField]
    private Server server;

    [SerializeField]
    private PlayerController playerControllerPrefab;

    [SerializeField]
    private CoinController coinController;

    [SerializeField]
    private BombController bombController;

    [SerializeField]
    private SpawnArea spawnArea;

    private Dictionary<int, PlayerController> playerControllers = new Dictionary<int, PlayerController>();
    private List<int> playerRemoveIds = new List<int>();
    private List<int> playerDeathIds = new List<int>();

    1 reference
    public Vector3 RandomSpawnPoint() => spawnArea.RandomSpawnPoint();
```

```
    1 reference
    public void CreatePlayer(PeerConnection peerConnection)
    {
        var playerController = Instantiate(playerControllerPrefab);
        var id = peerConnection.Id;
        playerController.Setup(id, server);
        playerControllers.Add(id, playerController);
        peerConnection.AddPlayer(playerController);
    }

    1 reference
    public void Remove(PlayerController playerController)
    {
        var playerId = playerController.Id;
        playerRemoveIds.Add(playerId);
        playerControllers.Remove(playerId);
    }
```

```
1 reference
public void UpdateData()
{
    var model = new UpdateModel();

    foreach (var clientConnection in server.PeerConnections.Values)
    {
        var player = clientConnection.Player;
        if (player != null)
        {
            if (player.isUpdateScore)
            {
                var playerModel = new UpdatePlayerModel();
                playerModel.Score = player.Score;
                clientConnection.Send(playerModel);
                player.isUpdateScore = false;
            }

            if (player.isUpdatePosition)
            {
                var playerPositionModel = new PlayerPositionModel { PlayerId = player.Id, Position = player.Position };
                model.PlayerPositionModels.Add(playerPositionModel);
                player.isUpdatePosition = false;
            }
        }
    }
```

Player Controller

Assembly-CSharp — PlayerController

```
using System;
using UnityEngine;


public class PlayerController : MonoBehaviour
{
    private const float MIN_MOVE_DISTANCE = 0.0625f;
    private const float GRAVITY = 9.8f;
    private const float LAY_BOMB_DELAY = 3f;

    [SerializeField] private CharacterController characterController;
    private BombController bombController;
    private int id;
    private int score = 0;
    private ServerController serverController;
    private Server server;
    private Vector3 target;

    private float speed = 1;
    private float gravitySpeed = 0;
    private bool isDeath = false;

    public int Id => id;

    public int Score => score;
    1 reference
    public bool IsDeath => isDeath;
```

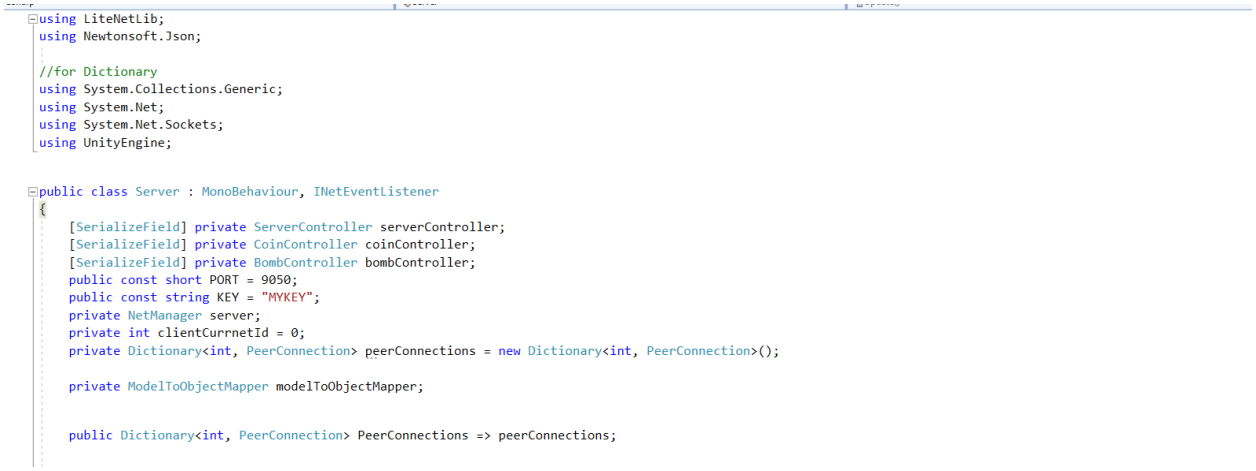
```
ssembly-CSharp                                                    PlayerController
24      public Vector3 Position => transform.position;
25
26      public bool isUpdatePosition;
27      public bool isUpdateScore;
28      public float lastLayBombTime;
29
30      private void Awake()
31      {
32          serverController = FindObjectOfType<ServerController>();
33          bombController = FindObjectOfType<BombController>();
34      }
35
36      public void GetCoin()
37      {
38          score++;
39          isUpdateScore = true;
40      }
41
42      // Start is called before the first frame update
43      private void Start()
44      {
45          server.SendCreatePlayer(this);
46      }
47
48      // Update is called once per frame
```

```
embly-CSharp                                                      PlayerController
            Unity Message | 0 references
49      private void Update()
50      {
51          if (!isDeath)
52          {
53              var v2Position = new Vector2(transform.position.x, transform.position.z);
54              var v2Target = new Vector2(target.x, target.z);
55              if (Vector2.Distance(v2Target, v2Position) > MIN_MOVE_DISTANCE)
56              {
57                  var dir = target - transform.position;
58                  characterController.Move(dir.normalized * speed * Time.deltaTime);
59                  isUpdatePosition = true;
60              }
61
62              if (!characterController.isGrounded)
63              {
64                  gravitySpeed += GRAVITY * Time.deltaTime;
65                  characterController.Move(Vector3.down * gravitySpeed * Time.deltaTime);
66                  isUpdatePosition = true;
67              }
68              else
69              {
70                  gravitySpeed = 0;
71              }
72          }
73      }
74
```

```
1 reference
public void Death()
{
    isDeath = true;
    serverController.Death(this);
    Destroy(gameObject);
}

1 reference
public void LayBomb()
{
    if (!isDeath)
    {
        if (Time.time > lastLayBombTime + LAY_BOMB_DELAY)
        {
            var bomb = bombController.Create();
            var position = transform.position;
            bomb.transform.position = new Vector3(position.x, 0, position.z);
            lastLayBombTime = Time.time;
        }
    }
}
```

```
    }

    1 reference
    internal void Setup(int id, Server server)
    {
        this.id = id;
        this.server = server;
    }

    1 reference
    internal void Move(Vector3 target)
    {
        this.target = target;
    }

    Unity Message | 0 references
    private void OnCollisionEnter(Collision collision)
    {
        Debug.Log("OnCollisionEnter");
        Debug.Log(collision.gameObject.name);
    }
}
```

, และสร้างโฟเดอร์Scripts เขียนโค้ด Server



```
using LiteNetLib;
using Newtonsoft.Json;

//for Dictionary
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using UnityEngine;


public class Server : MonoBehaviour, INetEventListener
{
    [SerializeField] private ServerController serverController;
    [SerializeField] private CoinController coinController;
    [SerializeField] private BombController bombController;
    public const short PORT = 9050;
    public const string KEY = "MYKEY";
    private NetManager server;
    private int clientCurrnetId = 0;
    private Dictionary<int, PeerConnection> peerConnections = new Dictionary<int, PeerConnection>();

    private ModelToObjectMapper modelToObjectMapper;


    public Dictionary<int, PeerConnection> PeerConnections => peerConnections;
```

```
    {
        Debug.Log("Awake");
        server = new NetManager(this);
        server.Start(PORT);
    }

    Unity Message | 0 references
    private void Start()
    {
        modelToObjectMapper = new ModelToObjectMapper(serverController);
    }

    Unity Message | 0 references
    private void Update()
    {
        server.PollEvents();
        serverController.UpdateData();
    }

    1 reference
    public void SendCreatePlayer(PlayerController playerController)
    {
        var model = new CreatePlayerModel { Id = playerController.Id, Position = playerController.Position };
        SendAll(model);
    }
```

```
2 references
public void SendAll(BaseModel model)
{
    var json = JsonConvert.SerializeObject(model);
    foreach (var clientConnection in peerConnections.Values)
    {
        clientConnection.Send(json);
    }
}

2 references
public void OnConnectionRequest(ConnectionRequest request)
{
    Debug.Log("OnConnectionRequest");
    CreatePeerConnection(request.AcceptIfKey(KEY));
}

2 references
public void OnNetworkError(IPEndPoint endPoint, SocketError socketError) => Debug.Log("OnNetworkError");

2 references
public void OnNetworkLatencyUpdate(NetPeer peer, int latency)
{
}
```

```
    2 references
    public void OnNetworkReceive(NetPeer peer, NetPacketReader reader, DeliveryMethod deliveryMethod)
    {
        //Debug.Log("OnNetworkReceive");
        modelToObjectMapper.DeserializeToFunction(peer, reader.GetString());
    }

    3 references
    public void OnNetworkReceiveUnconnected(IPEndPoint remoteEndPoint, NetPacketReader reader, UnconnectedMessageType messageType) => Debug.Log("OnNetworkReceiveUnconnected");

    1 reference
    private void CreatePeerConnection(NetPeer peer)
    {
        var id = clientCurrnetId++;
        var peerConnection = new PeerConnection(this, peer, id);
        peer.Tag = peerConnection;

        var model = new InitDataModel();
        foreach (var clientConnection in peerConnections.Values)
        {
            var player = clientConnection.Player;
            if (player != null)
            {
                //Debug.Log($"clientConnection {clientConnection.Id} player.Id {player.Id}");
                var createPlayerModel = new CreatePlayerModel { Id = player.Id, Position = player.Position };
                model.CreatePlayerModels.Add(createPlayerModel);
            }
        }
```

```
        }

        foreach (var pair in coinController.Coins)
        {
            var coinId = pair.Key;
            var position = pair.Value.transform.position;
            model.CreateCoins.Add(coinId, new Vector3Model(position));
        }
        peerConnections.Add(id, peerConnection);
        serverController.CreatePlayer(peerConnection);

        foreach (var bomb in bombController.Bombs.Values)
        {
            var bombModel = new BombModel { CurrnetTime = bomb.CurrentTime, Position = bomb.transform.position };
            model.Bombs.Add(bombModel);
        }

        var playerId = peerConnection.PlayerId;
        model.PlayerId = playerId;
        peerConnection.Send(model);
    }

    2 references
    public void OnPeerConnected(NetPeer peer)
    {
        Debug.Log("OnPeerConnected");
    }
```

2 references

```
public void OnPeerDisconnected(NetPeer peer, DisconnectInfo disconnectInfo)
{
    Debug.Log("OnPeerDisconnected");
    if (peer != null)
    {
        var peerConnection = peer.Tag as PeerConnection;
        peerConnection?.Disconnected();
    }
}
```

1 reference

```
public void Remove(PeerConnection peerConnection)
{
    peerConnections.Remove(peerConnection.Id);
}
```

, Model To Object Mapper

```
using LiteNetLib;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using UnityEngine;

3 references
public class ModelToObjectMapper
{
    private ServerController serverController;
    private Dictionary<string, Action<PeerConnection, JObject>> Deserializes;

    1 reference
    private Dictionary<string, Action<PeerConnection, JObject>> CreateDeserializes() => new Dictionary<string, Action<PeerConnection, JObject>>
    {
        //TODO [Model.CLASS_NAME] = callFunction,
        [MovePlayerModel.CLASS_NAME] = OnMovePlayer,
        [LayBombModel.CLASS_NAME] = OnLayBomb,
    };

    1 reference
    private void OnMovePlayer(PeerConnection peerConnection, JObject jObject)
    {
        var model = jObject.ToObject<MovePlayerModel>();
        //Debug.Log($"{model.ClassName} : ( x : {model.Target.X}, y : {model.Target.Z} )");
        peerConnection.Player.Move(model.Target.ToUnityVector3());
    }
```

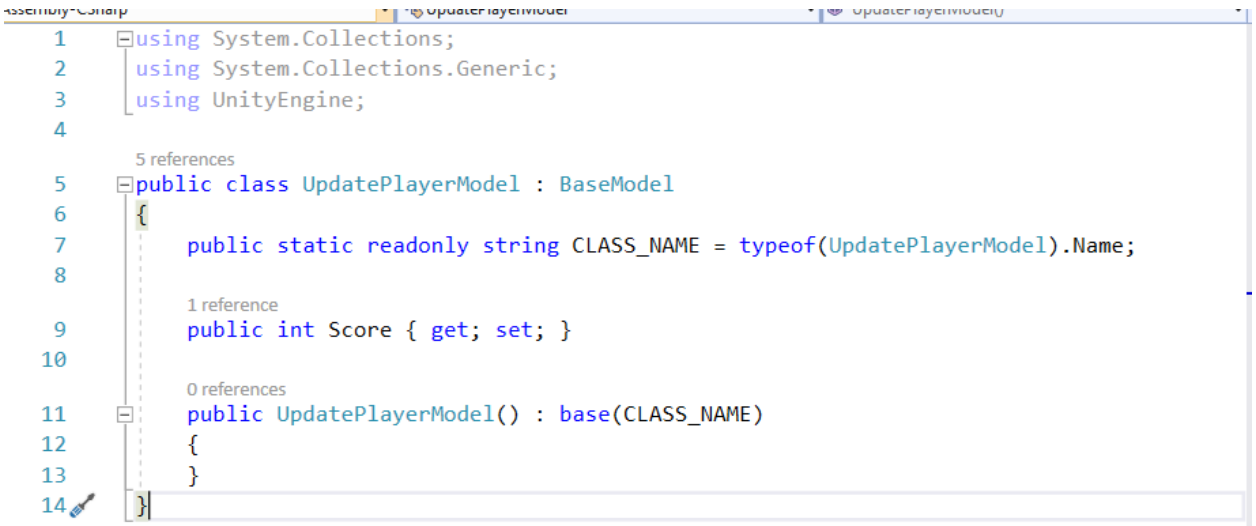
```
    1 reference
    private void OnLayBomb(PeerConnection peerConnection, JObject jObject)
    {
        var model = jObject.ToObject<LayBombModel>();
        if (model != null)
        {
            peerConnection.Player.LayBomb();
        }
    }

    1 reference
    public ModelToObjectMapper(ServerController serverController)
    {
        Deserializes = CreateDeserializes();
        this.serverController = serverController;
    }

    1 reference
    public void DeserializeToFunction(NetPeer peer, string json)
    {
        var jObject = JObject.Parse(json);
        if (jObject.TryGetValue("ClassName", out var value))
        {
            var className = value.ToString();
            var peerConnection = peer.Tag as PeerConnection;
            Deserializes[className](peerConnection, jObject);
        }
    }
}
```

ของ Client

UpdateModel

```
embly-CSharp                    UpdateModel                    CreateCoins
1   using System.Collections.Generic;
2
    6 references
3   public class UpdateModel : BaseModel
4   {
5       public static readonly string CLASS_NAME = typeof(UpdateModel).Name;
6
        1 reference
7       public List<PlayerPositionModel> PlayerPositionModels { get; set; } = new List<PlayerPositionModel>();
        1 reference
8       public List<int> PlayerRemoveIds { get; set; } = new List<int>();
        1 reference
9       public List<BombModel> NewBombs { get; set; } = new List<BombModel>();
        1 reference
10      public List<int> PlayerDeathIds { get; set; } = new List<int>();
11
12      public Dictionary<int, Vector3Model> CreateCoins = new Dictionary<int, Vector3Model>();
13      public List<int> DeletedCoins = new List<int>();
14
        0 references
15      public UpdateModel() : base(CLASS_NAME)
16      {
17      }
18  }
```

UpdatePlayeerModel



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

5 references
public class UpdatePlayerModel : BaseModel
{
    public static readonly string CLASS_NAME = typeof(UpdatePlayerModel).Name;

    1 reference
    public int Score { get; set; }

    0 references
    public UpdatePlayerModel() : base(CLASS_NAME)
    {
    }
}
```

Player Controller

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

public class PlayerController : MonoBehaviour
{
    [SerializeField] private Animator animator;
    [SerializeField] private CharacterController characterController;
    [SerializeField] private LayerMask layerMask;
    private int id;
    private Client client;
    private bool isCurrentPlayer;
    private bool isDeath;
    private bool isMyPlayer;

    private List<float> distances = new List<float>();
```

```
    private Vector3 lastPosition;
    public int Id => id;

    internal void Setup(CreatePlayerModel model, Client client)
    {
        id = model.Id;
        transform.position = model.Position;
        this.client = client;
    }

    // Start is called before the first frame update
    private void Start()
    {
    }

    // Update is called once per frame
    private void Update()
    {
        if (isCurrentPlayer && Input.GetMouseButton(0))
        {
            Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out var hit, int.MaxValue,
layerMask))
            {
                //Debug.Log(hit.point);
                var target = new VectorXZModel { X = hit.point.x, Z =
hit.point.z };
                client.Send(new MovePlayerModel { Target = target });
            }
```

```
        }

        if (Input.GetKeyUp(KeyCode.B))
        {
            client.Send(new LayBombModel());
        }

        var distance = Vector3.Distance(lastPosition,
transform.position);

        distances.Add(distance);
        if (distances.Count > 30)
        {
            distances.RemoveAt(0);
        }

        var sum = 0f;
        foreach (var d in distances)
        {
            sum += d;
        }
        if (sum / distances.Count > 0.125f * Time.deltaTime)
        {
            animator.SetBool("IsRun", true);
        }
        else
        {
            animator.SetBool("IsRun", false);
        }
        lastPosition = transform.position;
```

```
        if (isDeath)
        {
            animator.SetBool("IsDeath", true);
        }
    }

    public void SetCurrentPlayer()
    {
        isCurrentPlayer = true;
    }

    public void Move(Vector3 position)
    {
        var target = new Vector3(position.x, transform.position.y,
position.z);
        if (Vector3.Distance(transform.position, target) >
Time.deltaTime * 0.125f)
        {
            transform.LookAt(target, Vector3.up);
        }
        transform.position = position;
    }

    public void Remove()
    {
        if (isDeath && isMyPlayer)
        {
            SceneManager.LoadScene("GameOverScene");
        }
```

```
        Destroy(gameObject);
    }

    public void Death(bool isMyPlayer)
    {
        if (!isDeath)
        {
            this.isMyPlayer = isMyPlayer;
            isDeath = true;
            Invoke("Remove", 3f);
        }
    }
}
```

Client Controller

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

public class ClientController : MonoBehaviour
{
    [SerializeField] private PlayerController playerControllerPrefab;
    [SerializeField] private CoinController coinController;
    [SerializeField] private BombController bombController;
    [SerializeField] private Client client;
    [SerializeField] private Text scoreText;
    private int playerId = -1;
```

```
    private Dictionary<int, PlayerController> playerControllers =
new Dictionary<int, PlayerController>();

    // Start is called before the first frame update
    private void Start()
    {
    }

    // Update is called once per frame
    private void Update()
    {
    }

    public void OnCreatePlayer(CreatePlayerModel model)
    {
        var playerController = Instantiate(playerControllerPrefab);
        var id = model.Id;
        playerController.Setup(model, client);
        playerControllers.Add(id, playerController);
        CheckIsCurrentPlayer(playerController);
    }

    public void OnInitData(InitDataModel model)
    {
        playerId = model.PlayerId;
        foreach (var playerModel in model.CreatePlayerModels)
        {
            OnCreatePlayer(playerModel);
        }
```

```
        OnCreateCoins(model.CreateCoins);

        foreach (var bomb in model.Bombs)
        {
            bombController.Create(bomb);
        }
    }

    private void OnCreateCoins(Dictionary<int, Vector3Model>
createCoins)
    {
        coinController.OnCreateCoins(createCoins);
    }

    private void CheckIsCurrentPlayer(PlayerController
playerController)
    {
        if (playerController.Id == playerId)
        {
            playerController.SetCurrentPlayer();
        }
    }

    public void UpdatePlayerModel(UpdateModel model)
    {
        foreach (var playerPositionModel in
model.PlayerPositionModels)
        {
            var playerId = playerPositionModel.PlayerId;
            if (playerControllers.TryGetValue(playerId, out var player))
```

```
            {
                player.Move(playerPositionModel.Position);
            }
        }

        foreach (var removeId in model.PlayerRemoveIds)
        {
            if (playerControllers.TryGetValue(removeId, out var
player))
            {
                player.Remove();
                playerControllers.Remove(removeId);
            }
        }

        foreach (var deathId in model.PlayerDeathIds)
        {
            if (playerControllers.TryGetValue(deathId, out var player))
            {
                player.Death(deathId == playerId);
                playerControllers.Remove(deathId);
            }
        }

        foreach (var bomb in model.NewBombs)
        {
            bombController.Create(bomb);
        }

        coinController.OnUpdate(model);
```

```
    }

    public void OnUpdatePlayerModel(UpdatePlayerModel model)
    {
        scoreText.text = $"Score : {model.Score}";
    }
}
```

Client

```
using LiteNetLib;
using LiteNetLib.Utils;
using Newtonsoft.Json;
using System.Net;
using System.Net.Sockets;
using UnityEngine;

public class Client : MonoBehaviour, INetEventListener
{
    [SerializeField] private ClientController clientController;

    private NetManager client;
    private NetPeer peer;
    private string host = "localhost";
    private short port = 9050;
    private ModelToObjectMapper modelToObjectMapper;

    private void Awake()
    {
        Debug.Log("Awake");
```

```
        client = new NetManager(this);
        client.Start();
        client.Connect(host, port, "MYKEY");
    }

    private void Start()
    {
        modelToObjectMapper = new
ModelToObjectMapper(clientController);
    }

    private void Update()
    {
        client.PollEvents();
    }

    public void Send(BaseModel baseModel)
    {
        var json = JsonConvert.SerializeObject(baseModel);
        var netDataWriter = NetDataWriter.FromString(json);
        peer?.Send(netDataWriter,
DeliveryMethod.ReliableOrdered);
    }

    public void OnConnectionRequest(ConnectionRequest
request) => Debug.Log("OnConnectionRequest");

    public void OnNetworkError(IPEndPoint endPoint, SocketError
socketError) => Debug.Log("OnNetworkError");
```

```
    public void OnNetworkLatencyUpdate(NetPeer peer, int
latency)
    {
    }

    public void OnNetworkReceive(NetPeer peer,
NetPacketReader reader, DeliveryMethod deliveryMethod)
    {

modelToObjectMapper.DeserializeToFunction(reader.GetString());
    }

    public void OnNetworkReceiveUnconnected(IPEndPoint
remoteEndPoint, NetPacketReader reader,
UnconnectedMessageType messageType) =>
Debug.Log("OnNetworkReceiveUnconnected");

    public void OnPeerConnected(NetPeer peer)
    {
        Debug.Log("OnPeerConnected");
        this.peer = peer;
    }

    public void OnPeerDisconnected(NetPeer peer, DisconnectInfo
disconnectInfo)
    {
        Debug.Log("OnPeerDisconnected");
    }

    private void OnApplicationQuit()
```

```
    {
        client.DisconnectAll();
    }
}
```

และสร้างไฟล์Model ไว้สำหรับพวกโมเดลตัวละครหรือไอเทม ต่างๆเมื่อง เขียนโค้ดเสร็จคัดลอก Controller, Model To Object Mapper,Player Controller ใส่โฟเดอร์ client และเขียนโค้ดController,Client และใส่โค้ด Player Controller ใส่โมเดล Playerเมื่อแสดงผลให้เล่น Server ก่อน แล้วกดเล่น Client