VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FALCUTY OF COMPUTER NETWORKS AND COMMUNICATIONS

FINAL REPORT

CLIENT-SERVER GAME ARCHITECTURE

BALL ISLAND

Instructor: M.S.Lê Minh Khánh Hội

Course: NT106.N21.MMCL
Group members: Huỳnh Đình Khải Minh - 21521123 - 21521123@gm.uit.edu.vn
Trần Thành Lợi - 21522296 - 21522296@gm.uit.edu.vn
Nguyễn Nguyễn Duy An - 21520546 - 21520546@gm.uit.edu.vn

Lời mở đầu

Trong thời đại công nghệ thông tin như hiện nay, sản phẩm công nghệ ngày càng chịu sự đánh giá khắt khe hơn từ phía những người dùng, đặc biệt là về sản phẩm Game được nhận rất nhiều sự đánh giá từ phía các Game thủ, hay chỉ là những người chơi bình thường. Trải qua một quãng thời gian cách ly xã hội thì việc kết nối giữa người với người và trải nghiệm người dùng luôn là ưu tiên hàng đầu của các tựa game. Đó cũng là xu hướng mà các tựa game lớn hiện nay đang hướng tới, bằng việc xây dựng một máy chủ cho phép nhiều người dùng cùng kết nối, giao tiếp và trải nghiệm trò chơi cùng nhau qua internet.

Đây sẽ là một cách tiếp cận mới hơn về trải nghiệm của người dùng đối với trò chơi điện tử, nơi mà sự kết nối sẽ tạo nên sự tương tác và sự cạnh tranh sẽ đưa người chơi vào một trải nghiệm thú vị hơn phục vụ nhu cầu giải trí và kết nối đơn thuần.

Từ xu hướng phát triển trên, đồ án này sẽ triển khai một tựa game Multiplayer bằng cách sử dụng một Máy chủ chuyên dụng. Dưới sự hỗ trợ của Unity, một trong những game engine phổ biến nhất hiện nay trong việc phát triển game, đồng thời là thư viện mạng của C# cho phép ta tạo các kết nối giữa Client và Server. Với các công cụ đó chúng em sẽ triển khai tựa game Ball Island. Nhằm chuẩn bị kiến thức và kỹ năng cho sau này, đồng thời phục vụ yêu cầu môn học. Các chương đầu trong bài báo cáo trình bày lần lượt về Unity cũng như các công cụ hỗ trợ trong quá trình phát triển game.

Các chương tiếp theo sẽ giới thiệu về quá trình triển khai Server cũng như về các thành phần trong trò chơi được tối ưu để kết nối qua mạng. Các thành phần cơ sở dữ liệu phục vụ việc đăng ký và đăng nhập xác thực người chơi là không thể thiếu, cũng như các bảo mật dữ liệu khi gửi lên cơ sở dữ liệu. Cuối cùng sẽ là những khó khăn cũng như thành quả đạt được trong suốt quá trình phát triển tựa game.

Table of contents

1 Unity và các phần mềm hỗ trợ

1.1 Unity

1.1.1 Unity là gì

Unity là một phần mềm phát triển game hay còn được gọi là game engine đa nền tảng được phát triển bởi công ty Unity Technologies, chủ yếu được dùng để phát triển video game cho 27 nền tảng bao gồm máy tính, game consoles (như PlayStation) và điện thoại,...

Hơn 50% số lượng game trên thị trường được sản xuất bởi Unity. Một vài tựa game vô cùng nổi tiếng được tạo ra bởi Unity có thể được kể đến như Pokémon Go, Hearthstone, Ori And The Blind Forest, Monument Valley, Axie Infinity,... Độ "phủ sóng" của Unity rất rộng, có thể được áp dụng phổ biến trong nhiều dòng game khác nhau từ game "hạng nặng" Triple A (AAA) cho đến game giáo dục đơn giản cho con nít.

1.1.2 Ưu điểm của Unity

Là một cross-platform engine xét về sức mạnh tổng quan thì Unity có nhiều ưu điểm như:

- Ngôn ngữ hỗ trợ: Unity hỗ trợ người dùng sử dụng C#. Với thư viện .NET của C# có thể cho phép phục vụ mục đích của đồ án này.
- Editor: Với Editor, nhà phát triển không cần thiết phải viết Code để sắp đặt các đối tượng trong Game như những Engine khác mà Developer có thể kéo thả, thay đổi vị trí của từng đối tượng trong Game trực tiếp trên Editor.
- Đa nền tảng: là lợi ích thứ 3 rất quan trọng với nhiều công ty cũng như developer. Vì với việc bạn tạo ra Game mà Game đó có thể chạy được trên hầu hết những hệ điều hành quan trọng như Desktop (Mac, Window và Linux) hay Mobile (iOS, Android) hoặc Web (WebGL) thì cũng đã tiết kiệm công sức cũng như chi phí rất nhiều cho doanh nghiệp đó.
- Miễn phí: Và yếu tố cuối cùng chính là chi phí. Với Unity, miễn phí là một điểm thu hút rất nhiều Developer chọn làm việc với game engine này. Tuy nhiên, với các game được tạo ra miễn phí thì bắt buộc phải có Logo Unity trong Game.

1.2 MongoDB

1.2.1 MongoDB là gì

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL mã nguồn mở đa nền tảng viết bằng C++. Bản ghi trong MongoDB được lưu trữ dạng một dữ liệu văn bản (Document), là một cấu trúc dữ liệu bao gồm các cặp giá trị và trường tương tự như các đối tượng JSON.[5] MongoDB được phát triển bởi MongoDB Inc. dưới dạng giấy phép Server Side Public License (SSPL).

1.2.2 Các phiên bản hỗ trợ

- MongoDB Atlas: MongoDB cung cấp phiên bản chạy trên nền tảng điện toán đám mây (cloud) gọi là MongoDB Atlas, đây là gói sản phẩm dịch vụ tích hợp cơ sở dữ liệu đám mây và các dịch vụ dữ liệu.[7] MongoDB Atlas hỗ trợ các nền tảng AWS, Microsoft Azure, và Google Cloud Platform. Đây cũng là phiên bản được sử dụng trong đồ án.

- MongoDB Community Server: Phiên bản cài đặt máy chủ địa phương (on-premises) bao gồm 2 phiên bản là MongoDB Enterprise Advanced[8] và MongoDB Community Server.[9] Trong đó, phiên bản Enterprise Advanced là phiên bản trả phí còn phiên bản Community Server là phiên bản Cộng đồng của cơ sở dữ liệu. Phiên bản MongoDB Community miễn phí trên các hệ điều hành Windows, Linux, và macOS.
- MongoDB Enterprise Server: MongoDB Enterprise Server là phiên bản thương mại của MongoDB, tính phí theo chương trình thuê bao MongoDB Enterprise Advanced.

1.2.3 Ưu điểm khi sử dụng

- Là một hệ quản trị CSDL NoSQL: NoSQL là 1 dạng CSDL mã nguồn mở không sử dụng Transact-SQL để truy vấn thông tin. NoSQL viết tắt bởi: None-Relational SQL, hay có nơi thường gọi là Not-Only SQL. CSDL này được phát triển trên Javascript Framework với kiểu dữ liệu JSON. (Cú pháp của JSON là "key:value") NoSQL ra đời như là 1 mảnh vá cho những khuyết điểm và thiếu xót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS về tốc độ, tính năng, khả năng mở rộng, memory cache,...
- Cấu trúc của một đối tượng rõ ràng: MongoDB lưu trữ các bản ghi dữ liệu dưới dạng dữ liệu văn bản BSON. BSON là một đại diện dạng nhị phân của tài liệu JSON, tuy nhiên nhờ xây dựng dưới dạng nhị phân, nó được thiết kế để chứa nhiều kiểu dữ liệu hơn JSON.


```
_id: ObjectId('646f1bcd3dbacf2aacd75d4e')
username: "kaimink"
password: "57484F150FF9D26EA0549E31CF2F5C1767E351BDACA9B554BB530089A1031808"
salt: "5/25/2023 3:26:53 PM"
isOnline: "false"
Point: 2
Death: 6
```


Figure 1: Cấu trúc của một đối tượng.

- Cloud database: MongoDB cho phép ta tạo và sử dụng một cơ sở dữ liệu đám mây. Giúp dễ dàng cho việc quản lý thông tin và lưu trữ.

2 Tổng quan về trò chơi

2.1 Ý tưởng

Ball Island được dựa trên 2 tựa game nổi tiếng trên dòng máy Nokia cũ là Snake và Bounce Tales. Nơi mà người chơi là một quả bóng sẽ phải lăn để tìm điểm và đồng thời phải cạnh tranh với những người khác bằng cách tận dụng những cơ chế vật lý trong game.

Figure 2: Snake và Bounce trên dòng máy nokia.

2.2 Các thành phần trong game

2.2.1 Giao diện

Phần giao diện hỗ trợ nhập xuất thông tin khi đăng nhập và đăng ký, đồng thời cũng cho phép người dùng kết nối đến Server và tham gia trò chơi. Được thiết kế gồm 2 trang cho phép người chơi đăng ký và đăng nhập ứng với từng trang.

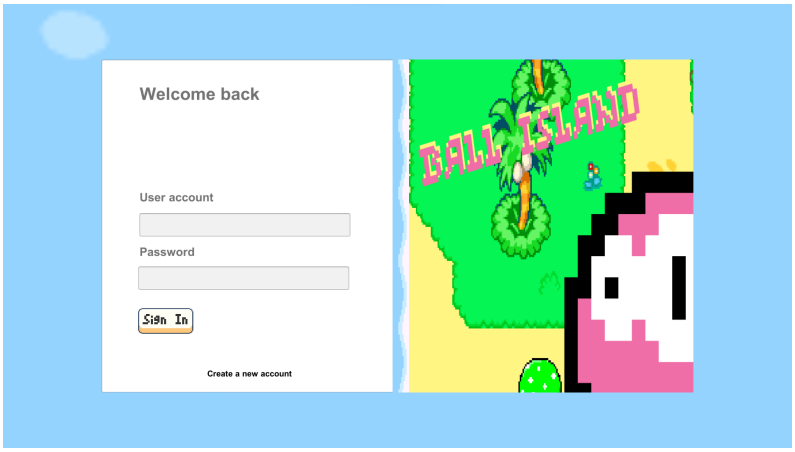

Figure 3: Giao diện đăng nhập và đăng ký.

2.2.2 Các vật thể trong game

Nhằm tạo sự đa dạng cho trò chơi, một số vật thể được tạo ra trong trò chơi bao gồm:

- Player/Enemy: Là một quả bóng được người chơi và đối thủ trong tựa game điều khiển
- Bounce mushroom: Một cây nấm to sẽ đẩy người chơi ra xa khi va chạm.
- Bụi cỏ: Giúp người chơi có thể ẩn nấp tránh sự chú ý từ các người chơi khác.
- Sông và biển: Khi người chơi rơi xuống sẽ bị đuối nước, đồng thời phải tốn thời gian để hồi sinh lại.
- Crabby: Một loại quái vật tự động di chuyển khắp bản đồ và tấn công người chơi. Người chơi tiêu diệt nó sẽ có thêm điểm.

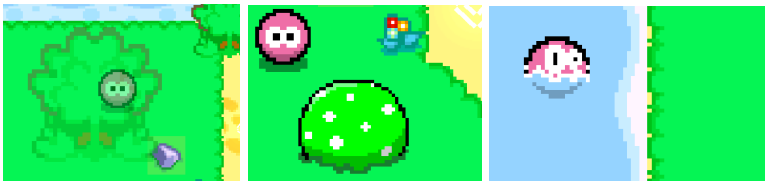

Figure 4: Các vật thể tương tác trong game.

Figure 5: Các vật thể đặc biệt trong game.

2.2.3 Bản đồ trong game

Được thiết kế là một hòn đảo rộng lớn, tạo sự thoải mái khi di chuyển. Cấu trúc đảo chia làm 3 phần:

- Bờ biển: Khu vực khởi đầu của người chơi, là một nơi rộng rãi và thoáng ít các chướng ngại vật.
- Đảo trung tâm: Khu vực to với nhiều vật thể chặn đường, cấu trúc phức tạp gây khó khăn khi di chuyển.
- Đảo nhỏ: Một khu vực ở trên cùng, được thiết kế cấu trúc hẹp khó khăn khi di chuyển.

Figure 6: Bản đồ của tựa game.

Xung quanh hòn đảo được bao bọc bởi biển cũng như là phần giới hạn người chơi có thể di chuyển trong trò chơi.

2.3 Cơ chế game

2.3.1 Gameplay

Mỗi người chơi sẽ điều khiển một quả bóng di chuyển trong một khu vực đảo để nhặt vật phẩm tăng điểm cho bản thân, đồng thời phải tránh các chướng ngại vật cũng như những người chơi khác.

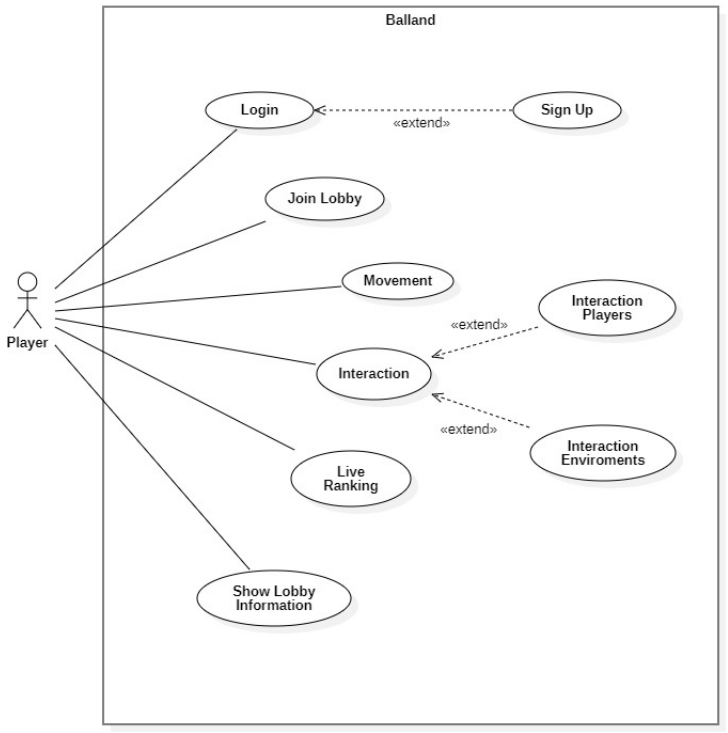

Figure 7: Sơ đồ chức năng gameplay dành cho player.

2.3.2 Game mechanic

Một số cơ chế chính có mặt trong game để hỗ trợ cho gameplay như:

- Điều khiển: Người chơi sử dụng các phím WASD để di chuyển và kiểm soát quả bóng.
- Cơ chế tạo vật phẩm: Các vật phẩm tăng điểm cho người chơi sẽ được tạo ra ở những nơi đã sắp đặt trên bản đồ, sau khi nhặt thì phải đợi một khoảng thời gian vật phẩm sẽ xuất hiện lại.
- Đuối nước: Sau khi người chơi rơi xuống nước, sẽ mất một khoảng thời gian để người chơi có thể hồi sinh và di chuyển. Trong quãng thời gian này người chơi sẽ không thể tác động vào quả bóng.
- Tính điểm và xếp hạng: Bảng xếp hạng của người chơi sẽ dựa trên giá trị Point/Death. Bảng xếp hạng này sẽ hiển thị sau khi người chơi ấn nút Tab.

3 Server-Client Architecture

3.1 Cơ sở lý thuyết

3.1.1 Unity và thư viện mạng .NET

Việc sử dụng Unity sử dụng C# giúp ta dễ dàng tiếp cận được tài nguyên từ thư viện .NET của C# đặc biệt là class Socket cho phép dễ dàng thực hiện việc kết nối và gửi dữ liệu qua mạng. Nó cho phép ta thực hiện tạo các kết nối, gửi dữ liệu UDP và TCP đến server và ngược lại. Tùy thuộc vào nhu cầu mà ta sẽ xác định phương thức gửi riêng cho từng tác vụ.

3.1.2 Game server và Client

Server và Client hay còn gọi là máy chủ và máy khách. Mặc dù cùng một tựa game, nhưng cả hai đều mang những nhiệm vụ khác biệt, trong vấn đề xử lý thông tin:

- Game server: Là một hệ thống máy chủ được thiết kế để quản lý các thông tin và dữ liệu của game. Đây là nơi lưu trữ tất cả những gì bạn thấy trong game, bao gồm cả thông tin về nhân vật, vật phẩm, bản đồ, cốt truyện, đồng thời cũng là nơi thực hiện việc tính toán các tương tác vật lý trong trò chơi. Tất cả những gì bạn làm trong game đều được lưu lại trên máy chủ.
- Client: Nơi người chơi tương tác vào trò chơi như di chuyển, tấn công kẻ địch. Client không trực tiếp tính toán và xử lý mà sẽ gửi thông tin những hành động này tới server để xử lý, server sau khi xử lý và tính toán sẽ trả lại kết quả và hiển thị cho người chơi.

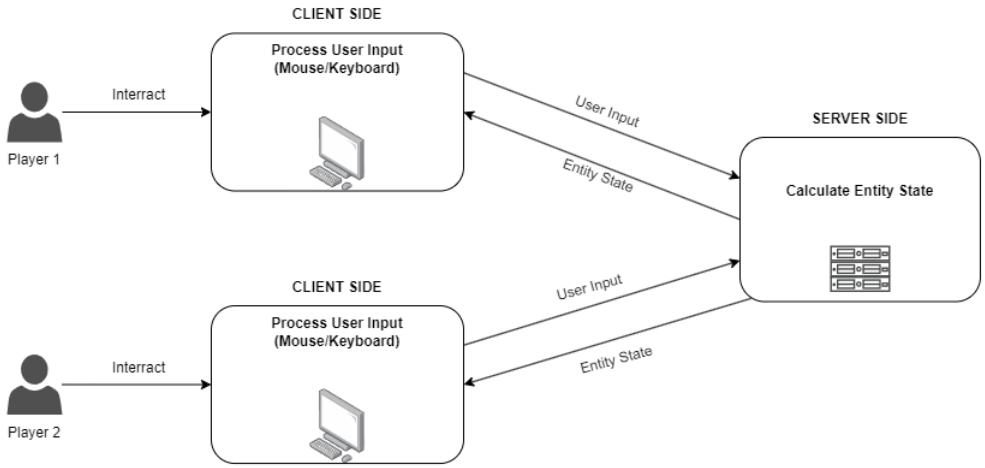

Figure 8: Xử lý giao tiếp Client - Server.

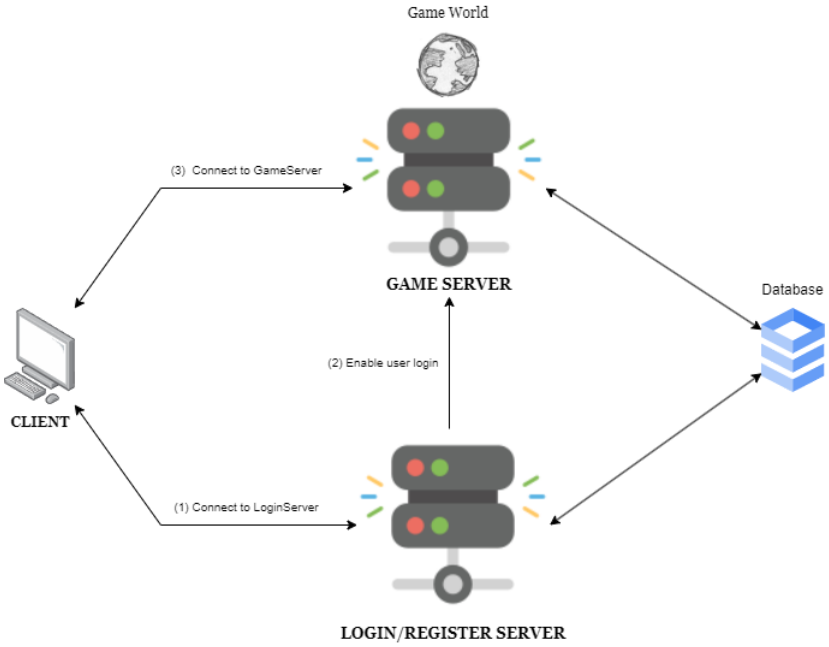

Figure 9: Client - Server Architechture.

3.2 Triển khai vào tựa game

Để triển khai vào tựa game, đầu tiên ta sẽ thiết lập một server để nhận kết nối và dữ liệu từ phía client. Đồng thời cũng ta cũng thực hiện việc xử lý tính toán di chuyển cho game ở phía server.

- Server side:

Listing 1: Server.cs

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Net;
5  using System.Net.Sockets;
6  using UnityEngine;
7  using UnityEngine.Networking;
8
9  public class Server
10 {
11     public static int MaxPlayers { get; private set; }
12     public static int Port { get; private set; }
13     public static Dictionary<int, Client> clients = new Dictionary<int, Client
           >();
14     public delegate void PacketHandler(int _fromClient, Packet _packet);
15     public static Dictionary<int, PacketHandler> packetHandlers;
16     private static TcpListener tcpListener;
17     private static UdpClient udpListener;
18
19     /// <summary>Starts the server.</summary>
```

```
20      /// <param name="_maxPlayers">The maximum players that can be connected
            simultaneously.</param>
21      /// <param name="_port">The port to start the server on.</param>
22      public static void Start(int _maxPlayers, int _port)
23      {
24          MaxPlayers = _maxPlayers;
25          Port = _port;
26
27          Debug.Log("Starting server...");
28          InitializeServerData();
29
30          tcpListener = new TcpListener(IPAddress.Any, Port);
31          tcpListener.Start();
32          tcpListener.BeginAcceptTcpClient(TCPConnectCallback, null);
33
34
35          udpListener = new UdpClient(Port);
36          udpListener.BeginReceive(UDPReceiveCallback, null);
37
38          Debug.Log($"Server started on port {Port}.");
39      }
40
41      /// <summary>Handles new TCP connections.</summary>
42      private static void TCPConnectCallback(IAsyncResult _result)
43      {
44          TcpClient _client = tcpListener.EndAcceptTcpClient(_result);
45          tcpListener.BeginAcceptTcpClient(TCPConnectCallback, null);
46          Debug.Log($"Incoming connection from {_client.Client.RemoteEndPoint
                }...");
47
48          for (int i = 1; i <= MaxPlayers; i++)
49          {
50              if (clients[i].tcp.socket == null)
51              {
52                  clients[i].tcp.Connect(_client);
53                  return;
54              }
55          }
56
57          Debug.Log($"{_client.Client.RemoteEndPoint} failed to connect: Server
                full!");
58      }
59
60      /// <summary>Receives incoming UDP data.</summary>
61      private static void UDPReceiveCallback(IAsyncResult _result)
62      {
63          try
64          {
65              IPEndPoint _clientEndPoint = new IPEndPoint(IPAddress.Any, 0);
66              byte[] _data = udpListener.EndReceive(_result, ref _clientEndPoint
                    );
67              udpListener.BeginReceive(UDPReceiveCallback, null);
68
69              if (_data.Length < 4)
70              {
71                  return;
72              }
73
74              using (Packet _packet = new Packet(_data))
75              {
76                  int _clientId = _packet.ReadInt();
77
```

```
78                  if (_clientId == 0)
79                  {
80                      return;
81                  }
82
83                  if (clients[_clientId].udp.endPoint == null)
84                  {
85                      // If this is a new connection
86                      clients[_clientId].udp.Connect(_clientEndPoint);
87                      return;
88                  }
89
90                  if (clients[_clientId].udp.endPoint.ToString() ==
                        _clientEndPoint.ToString())
91                  {
92                      // Ensures that the client is not being impersonated by
                            another by sending a false clientID
93                      clients[_clientId].udp.HandleData(_packet);
94                  }
95              }
96          }
97          catch (Exception _ex)
98          {
99              Debug.Log($"Error receiving UDP data: {_ex}");
100         }
101     }
102
103     /// <summary>Sends a packet to the specified endpoint via UDP.</summary>
104     /// <param name="_clientEndPoint">The endpoint to send the packet to.</
            param>
105     /// <param name="_packet">The packet to send.</param>
106     public static void SendUDPData(IPEndPoint _clientEndPoint, Packet _packet)
107     {
108         try
109         {
110             if (_clientEndPoint != null)
111             {
112                 udpListener.BeginSend(_packet.ToArray(), _packet.Length(),
                        _clientEndPoint, null, null);
113             }
114         }
115         catch (Exception _ex)
116         {
117             Debug.Log($"Error sending data to {_clientEndPoint} via UDP: {_ex}
                    ");
118         }
119     }
120
121     /// <summary>Initializes all necessary server data.</summary>
122     private static void InitializeServerData()
123     {
124         for (int i = 1; i <= MaxPlayers; i++)
125         {
126             clients.Add(i, new Client(i));
127         }
128
129         packetHandlers = new Dictionary<int, PacketHandler>()
130             {
131                 { (int)ClientPackets.welcomeReceived, ServerHandle.
                        WelcomeReceived },
132                 { (int)ClientPackets.playerMovement, ServerHandle.
                        PlayerMovement },
```

```
133             };
134         Debug.Log("Initialized packets.");
135     }
136     public static void Stop()
137     {
138         tcpListener.Stop();
139         udpListener.Close();
140     }
141 }
```

Listing 2: Player.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player : MonoBehaviour
6  {
7
8      public int id;
9      public string username;
10
11     public bool isPressMoveKey, isCanConotrol;
12     public ParticleSystem obtainEff, dushEffect;
13     private float moveSpeed = 700f / Constants.TICKS_PER_SEC;
14     public float bounceForce;
15     private bool[] inputs;
16     public bool isDrown;
17     public Rigidbody2D rb;
18
19     void Start()
20     {
21         rb = gameObject.GetComponent<Rigidbody2D>();
22         isCanConotrol = true;
23         DatabaseManager.instance.SetStatus("true", username);
24
25     }
26
27
28     public void Initialize(int _id, string _username)
29     {
30         id = _id;
31         username = _username;
32
33
34         inputs = new bool[5];
35     }
36
37     /// <summary>Processes player input and moves the player.</summary>
38     public void FixedUpdate()
39     {
40         Vector2 _inputDirection = Vector2.zero;
41         if (inputs[0])
42         {
43             _inputDirection.y += 1;
44         }
45         if (inputs[1])
46         {
47             _inputDirection.y -= 1;
48         }
49         if (inputs[2])
```

```
50          {
51              _inputDirection.x -= 1;
52          }
53          if (inputs[3])
54          {
55              _inputDirection.x += 1;
56          }
57
58          Move(_inputDirection);
59      }
60       Vector3 enterWaterPos;
61
62
63      /// <summary>Calculates the player's desired movement direction and moves
            him.</summary>
64      /// <param name="_inputDirection"></param>
65      private void Move(Vector2 _inputDirection)
66      {
67
68          if (isCanConotrol && isDrown == false)
69          {
70          rb.AddForce(new Vector2(_inputDirection.x * moveSpeed, _inputDirection
                .y * moveSpeed));
71          }
72
73          ServerSend.PlayerPosition(this);
74
75      }
76
77
78       private void OnTriggerEnter2D(Collider2D collision)
79      {
80
81
82          if (collision.gameObject.tag == "Water")
83          {
84              enterWaterPos = transform.position - new Vector3(Mathf.Clamp(rb.
                    velocity.x, 0, 1), Mathf.Clamp(rb.velocity.y, 0, 1), 0);
85          }
86          if(collision.gameObject.tag == "Player")
87          {
88              collision.gameObject.GetComponent<Rigidbody2D>().velocity = (
                    collision.transform.position-transform.position).normalized *
                    bounceForce;
89          }
90
91      }
92      Vector3 spawnpos = new Vector3 (0, 0, 0);
93      private void OnCollisionEnter2D(Collision2D collision)
94      {
95          if (collision.gameObject.tag == "Water")
96          {
97              print("hit water");
98              Vector3 landPos = transform.position + new Vector3(collision.
                    contacts[0].normal.x, collision.contacts[0].normal.y, 0);
99              if (collision.contacts[0].normal.x > 0.9 || collision.contacts[0].
                    normal.x < -0.9)
100                 transform.Translate(new Vector2(-collision.contacts[0].normal.
                        x, 0));
101             if (collision.contacts[0].normal.y > 0.9)
102                 transform.Translate(new Vector2(0, -collision.contacts[0].
                        normal.y));
```

```
103             if (collision.contacts[0].normal.y < -0.9)
104                 transform.Translate(new Vector2(0, -collision.contacts[0].
                        normal.y * 0.3f));
105             //transform.Translate(new Vector2(-collision.contacts[0].normal.x
                    *1.2f, -collision.contacts[0].normal.y*1.2f));
106 
107             rb.velocity = Vector2.zero;
108             DatabaseManager.instance.DeadCounter(username);
109             StartCoroutine(Respawn(10, spawnpos));
110         }
111 
112     }
113     private void OnCollisionStay2D(Collision2D collision)
114     {
115         if (collision.gameObject.tag == "Water")
116         {
117             Vector3 landPos = enterWaterPos;
118 
119             rb.velocity = Vector2.zero;
120             isDrown = true;
121 
122             StartCoroutine(Respawn(10, spawnpos));
123         }
124     }
125     IEnumerator Respawn(float sec, Vector3 landPos)
126     {
127         yield return new WaitForSeconds(sec);
128         transform.position = spawnpos;
129         isDrown = false;
130 
131     }
132     public void OnDestroy()
133     {
134         DatabaseManager.instance.SetStatus("false",username);
135     }
136     public void OnApplicationQuit()
137     {
138         DatabaseManager.instance.SetStatus("false",username);
139     }
140 
141     /// <summary>Updates the player input with newly received input.</summary>
142     /// <param name="_inputs">The new key inputs.</param>
143     /// <param name="_rotation">The new rotation.</param>
144 
145     public void SetInput(bool[] _inputs, Quaternion _rotation)
146     {
147         inputs = _inputs;
148         transform.rotation = _rotation;
149     }
150 }
```

Với class Server ta sẽ thiết lập thông số cho Game Server bao gồm người chơi và port, đồng thời tiến hành lắng nghe từ phía client bằng hai giao thức UDP và TCP để có thể nhận dữ liệu và gửi về phía client. Với các dữ liệu như di chuyển ta sẽ thực hiện gửi bằng giao thức UDP, nhằm tạo sự phản hồi nhanh và tốt nhất đến người chơi khi di chuyển. Các vấn đề cần sự chính xác về đồng bộ như số lượng vật phẩm đang được khởi tạo, số lượng quái vật, người chơi nào đã thoát,... sẽ được tiến hành gửi đến Client bằng giao thức TCP nhằm tạo sự chính xác và đồng nhất về thông tin. Để không bị xung đột giữa các người chơi server sẽ lưu và thêm người chơi đã kết nối vào danh sách với mỗi người chơi mang một id riêng.

Ngoài ra để có thể di chuyển và thực hiện các tính toán vật lý cho game, Với class Player mỗi player kết nối đến server sẽ được khởi tạo thông tin trong trò chơi bao gồm username và id cho mỗi player. Sau khi đã được khởi tạo thành công server sẽ nhận dữ liệu input từ phía client, dữ liệu input đó sẽ được server nhận và xử lý di chuyển, đồng thời các tương tác với nước và hồi sinh cũng được diễn ra tại server. Sau khi thực hiện tính toán server sẽ gửi dữ liệu về vị trí hiện tại đến client tương ứng. Vậy nên việc di chuyển của client chỉ là phân tích vị trí tọa độ có sẵn từ phía server gửi về, client không thực sự xử lý trong quá trình này.

Để giải quyết nhiều thông tin được gửi đi, phía Server sẽ đóng gói các thông tin dưới dạng môt Packet, các Packet được Server xử lý như: PlayerPossition, ItemSpawner, EnemySpawner,... Bằng cách chia thành các Packet phía client có thể dễ dàng phân biệt và xử lý dữ liệu.

- Client Side:

Listing 3: Client.cs

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

public class Client : MonoBehaviour
{
    public static Client instance;
    public static int dataBufferSize = 4096;

    public string ip ="127.0.0.1";
    public int port;
    public int myId = 0;
    public TCP tcp;
    public UDP udp;
    private DatabaseManager databaseaccess;

    private bool isConnected = false;
    private delegate void PacketHandler(Packet _packet);
    private static Dictionary<int, PacketHandler> packetHandlers;

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            databaseaccess = GameObject.FindGameObjectWithTag("Database").
                GetComponent<DatabaseManager>();
        }
        else if (instance != this)
        {
            Debug.Log("Instance already exists, destroying object!");
            Destroy(this);
        }
    }

    private void Start()
    {

    }

```

```
44      private void OnApplicationQuit()
45      {
46          Disconnect(); // Disconnect when the game is closed
47      }
48
49      /// <summary>Attempts to connect to the server.</summary>
50      public void ConnectToServer()
51      {
52          tcp = new TCP();
53          udp = new UDP();
54          InitializeClientData();
55
56          isConnected = true;
57          tcp.Connect(); // Connect tcp, udp gets connected once tcp is done
58      }
59
60      public class TCP
61      {
62          public TcpClient socket;
63
64          private NetworkStream stream;
65          private Packet receivedData;
66          private byte[] receiveBuffer;
67
68          /// <summary>Attempts to connect to the server via TCP.</summary>
69          public void Connect()
70          {
71              socket = new TcpClient
72              {
73                  ReceiveBufferSize = dataBufferSize,
74                  SendBufferSize = dataBufferSize
75              };
76
77              receiveBuffer = new byte[dataBufferSize];
78              socket.BeginConnect(instance.ip, instance.port, ConnectCallback,
                    socket);
79          }
80
81          private void ConnectCallback(IAsyncResult _result)
82          {
83              socket.EndConnect(_result);
84
85              if (!socket.Connected)
86              {
87                  return;
88              }
89
90              stream = socket.GetStream();
91
92              receivedData = new Packet();
93
94              stream.BeginRead(receiveBuffer, 0, dataBufferSize, ReceiveCallback
                    , null);
95          }
96
97          public void SendData(Packet _packet)
98          {
99              try
100             {
101                 if (socket != null)
102                 {
103                     stream.BeginWrite(_packet.ToArray(), 0, _packet.Length(),
```

```
                        null, null); // Send data to server
104                 }
105             }
106             catch (Exception _ex)
107             {
108                 Debug.Log($"Error sending data to server via TCP: {_ex}");
109             }
110         }
111
112         /// <summary>Reads incoming data from the stream.</summary>
113         private void ReceiveCallback(IAsyncResult _result)
114         {
115             try
116             {
117                 int _byteLength = stream.EndRead(_result);
118                 if (_byteLength <= 0)
119                 {
120                     instance.Disconnect();
121                     return;
122                 }
123
124                 byte[] _data = new byte[_byteLength];
125                 Array.Copy(receiveBuffer, _data, _byteLength);
126
127                 receivedData.Reset(HandleData(_data)); // Reset receivedData
                        if all data was handled
128                 stream.BeginRead(receiveBuffer, 0, dataBufferSize,
                        ReceiveCallback, null);
129             }
130             catch
131             {
132                 Disconnect();
133             }
134         }
135
136         /// <summary>Prepares received data to be used by the appropriate
                packet handler methods.</summary>
137         /// <param name="_data">The recieved data.</param>
138         private bool HandleData(byte[] _data)
139         {
140             int _packetLength = 0;
141
142             receivedData.SetBytes(_data);
143
144             if (receivedData.UnreadLength() >= 4)
145             {
146                 // If client's received data contains a packet
147                 _packetLength = receivedData.ReadInt();
148                 if (_packetLength <= 0)
149                 {
150                     // If packet contains no data
151                     return true; // Reset receivedData instance to allow it to
                            be reused
152                 }
153             }
154
155             while (_packetLength > 0 && _packetLength <= receivedData.
                    UnreadLength())
156             {
157                 // While packet contains data AND packet data length doesn't
                        exceed the length of the packet we're reading
158                 byte[] _packetBytes = receivedData.ReadBytes(_packetLength);
```

```
159                 ThreadManager.ExecuteOnMainThread(() =>
160                 {
161                     using (Packet _packet = new Packet(_packetBytes))
162                     {
163                         int _packetId = _packet.ReadInt();
164                         packetHandlers[_packetId](_packet); // Call
                               appropriate method to handle the packet
165                     }
166                 });
167 
168                 _packetLength = 0; // Reset packet length
169                 if (receivedData.UnreadLength() >= 4)
170                 {
171                     // If client's received data contains another packet
172                     _packetLength = receivedData.ReadInt();
173                     if (_packetLength <= 0)
174                     {
175                         // If packet contains no data
176                         return true; // Reset receivedData instance to allow
                               it to be reused
177                     }
178                 }
179             }
180 
181             if (_packetLength <= 1)
182             {
183                 return true; // Reset receivedData instance to allow it to be
                       reused
184             }
185 
186             return false;
187         }
188 
189         /// <summary>Disconnects from the server and cleans up the TCP
                connection.</summary>
190         private void Disconnect()
191         {
192             instance.Disconnect();
193 
194             stream = null;
195             receivedData = null;
196             receiveBuffer = null;
197             socket = null;
198         }
199     }
200 
201     public class UDP
202     {
203         public UdpClient socket;
204         public IPEndPoint endPoint;
205 
206         public UDP()
207         {
208             endPoint = new IPEndPoint(IPAddress.Parse(instance.ip), instance.
                    port);
209         }
210 
211         /// <summary>Attempts to connect to the server via UDP.</summary>
212         /// <param name="_localPort">The port number to bind the UDP socket to
                .</param>
213         public void Connect(int _localPort)
214         {
```

```
215             socket = new UdpClient(_localPort);
216
217             socket.Connect(endPoint);
218             socket.BeginReceive(ReceiveCallback, null);
219
220             using (Packet _packet = new Packet())
221             {
222                 SendData(_packet);
223             }
224         }
225
226         /// <summary>Sends data to the client via UDP.</summary>
227         /// <param name="_packet">The packet to send.</param>
228         public void SendData(Packet _packet)
229         {
230             try
231             {
232                 _packet.InsertInt(instance.myId); // Insert the client's ID at
                        the start of the packet
233                 if (socket != null)
234                 {
235                     socket.BeginSend(_packet.ToArray(), _packet.Length(), null
                            , null);
236                 }
237             }
238             catch (Exception _ex)
239             {
240                 Debug.Log($"Error sending data to server via UDP: {_ex}");
241             }
242         }
243
244         /// <summary>Receives incoming UDP data.</summary>
245         private void ReceiveCallback(IAsyncResult _result)
246         {
247             try
248             {
249                 byte[] _data = socket.EndReceive(_result, ref endPoint);
250                 socket.BeginReceive(ReceiveCallback, null);
251
252                 if (_data.Length < 4)
253                 {
254                     instance.Disconnect();
255                     return;
256                 }
257
258                 HandleData(_data);
259             }
260             catch
261             {
262                 Disconnect();
263             }
264         }
265
266         /// <summary>Prepares received data to be used by the appropriate
                packet handler methods.</summary>
267         /// <param name="_data">The recieved data.</param>
268         private void HandleData(byte[] _data)
269         {
270             using (Packet _packet = new Packet(_data))
271             {
272                 int _packetLength = _packet.ReadInt();
273                 _data = _packet.ReadBytes(_packetLength);
```

```
274             }
275
276             ThreadManager.ExecuteOnMainThread(() =>
277             {
278                 using (Packet _packet = new Packet(_data))
279                 {
280                     int _packetId = _packet.ReadInt();
281                     packetHandlers[_packetId](_packet); // Call appropriate
                        method to handle the packet
282                 }
283             });
284         }
285
286         /// <summary>Disconnects from the server and cleans up the UDP
                connection.</summary>
287         private void Disconnect()
288         {
289             instance.Disconnect();
290
291             endPoint = null;
292             socket = null;
293         }
294     }
295
296     /// <summary>Initializes all necessary client data.</summary>
297     private void InitializeClientData()
298     {
299         packetHandlers = new Dictionary<int, PacketHandler>()
300         {
301             { (int)ServerPackets.welcome, ClientHandle.Welcome },
302             { (int)ServerPackets.spawnPlayer, ClientHandle.SpawnPlayer },
303             { (int)ServerPackets.playerPosition, ClientHandle.PlayerPosition
                    },
304             { (int)ServerPackets.playerDisconnected, ClientHandle.
                    PlayerDisconnected },
305             { (int)ServerPackets.createItemSpawner, ClientHandle.
                    CreateItemSpawner },
306             { (int)ServerPackets.itemSpawned, ClientHandle.ItemSpawned },
307             { (int)ServerPackets.ItemPickedUp, ClientHandle.ItemPickedUp },
308             { (int)ServerPackets.spawnEnemy, ClientHandle.SpawnEnemy },
309             { (int)ServerPackets.enemyPosition, ClientHandle.EnemyPosition },
310             { (int)ServerPackets.enemyHealth, ClientHandle.EnemyHealth },
311         };
312         Debug.Log("Initialized packets.");
313     }
314
315     /// <summary>Disconnects from the server and stops all network traffic.</
            summary>
316     public void Disconnect()
317     {
318         if (isConnected)
319         {
320             //databaseaccess.SetStatus("false");
321             isConnected = false;
322             tcp.socket.Close();
323             udp.socket.Close();
324
325             Debug.Log("Disconnected from server.");
326         }
327     }
328 }
```

Listing 4: PlayerControler.cs

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

public class PlayerController : MonoBehaviour
{
    [SerializeField] GameObject ballSpriteObj, afterImageObj;
    public GameObject cameraController;
    public bool isPressMoveKey, isDashing, isCanConotrol, isJumping,
        isCanBeHurted, isConfuse, isDie, isGoal, isDrown;
    Animator animSprite, anim;
    Rigidbody2D rb;
    float xdir, ydir;
    float dashRecoil;
    [SerializeField] float timeDashing;
    public ParticleSystem obtainEff, dushEffect;

    void Awake()
    {
        animSprite = ballSpriteObj.GetComponent<Animator>();
        anim = gameObject.GetComponent<Animator>();
        rb = gameObject.GetComponent<Rigidbody2D>();
        isCanConotrol = true;
        cameraController = GameObject.FindGameObjectWithTag("MainCamera");
        cameraController.GetComponent<CameraControler>().canTarget = true;

    }
    private void Update()
    {
        if (isDrown == false && isCanConotrol)
        {


        xdir = Input.GetAxis("Horizontal");
        ydir = Input.GetAxis("Vertical");

        animSprite.speed = 1;
        animSprite.SetFloat("xSpeed", xdir);
        animSprite.SetFloat("ySpeed", ydir);
        }
        else
        {
            animSprite.SetFloat("xSpeed", 0);
            animSprite.SetFloat("ySpeed", 0);
        }
        SendInputToServer();


    }
   // void Walking()
  //  {
   //  if (Time.timeScale != 0)
      //  {
      //  anim.speed = 1;
       // anim.SetFloat("xSpeed", xdir);
       // anim.SetFloat("ySpeed", ydir);
      //  }
   // }


```

```
60      Vector3 enterWaterPos;
61
62      private void OnCollisionEnter2D(Collision2D collision)
63      {
64          if (collision.gameObject.tag == "Water")
65          {
66
67              //transform.Translate(new Vector2(-collision.contacts[0].normal.x
                    *1.2f, -collision.contacts[0].normal.y*1.2f));
68              animSprite.SetBool("isDrown", true);
69              anim.Play("Drown");
70              animSprite.speed = 1;
71              animSprite.Play("Drown");
72              StartCoroutine(LenBo(5));
73
74          }
75
76
77      }
78
79      private void OnCollisionStay2D(Collision2D collision)
80      {
81          if (collision.gameObject.tag == "Water")
82          {
83
84              animSprite.SetBool("isDrown", true);
85              anim.Play("Drown");
86              animSprite.speed = 1;
87              animSprite.Play("Drown");
88              StartCoroutine(LenBo(5));
89          }
90      }
91      IEnumerator LenBo(float sec)
92      {
93          yield return new WaitForSeconds(sec);
94          animSprite.SetBool("isDrown", false);
95          anim.SetBool("isDrown", false);
96
97
98      }
99
100
101
102     /// <summary>Sends player input to the server.</summary>
103     private void SendInputToServer()
104     {
105
106         bool[] _inputs = new bool[]
107         {
108             Input.GetKey(KeyCode.W),
109             Input.GetKey(KeyCode.S),
110             Input.GetKey(KeyCode.A),
111             Input.GetKey(KeyCode.D),
112         };
113         ClientSend.PlayerMovement(_inputs);
114     }
115 }
```

Khi server đã được khởi tạo, client sẽ có thể kết nối đến server qua địa chỉ IP và Port cung cấp từ phía server. Ngoài ra để có thể đồng bộ dữ liệu từ các client mỗi client sẽ nhận được thông tin từ server như vị trí của các player khác, những player đã thoát và vật phẩm đang hiện có

trên map để có thể tiến hành đồng bộ di chuyển cũng như hiển thị chính xác những thông tin từ phía server trong khi các player khác đang thực hiện tác vụ.

Về class Player ở phía client ta sẽ nhận các dữ liệu đầu vào khi người dùng nhập vào bàn phím những phím di chuyển như WASD, những dữ liệu này sẽ được đưa vào một mảng bool và gửi đến server ngay lập tức và trả về dữ liệu vị trí đã được tính toán từ phía server. Phía client chỉ xử lý các thông tin liên quan đến hiển thị như hiệu ứng và animation và nhận dữ liệu được gửi về sau đó hiển thị đến người chơi.

4 Cơ sở dữ liệu và lưu trữ

4.1 Quản lý tài khoản

Dữ liệu tài khoản của người chơi sẽ được lưu trữ trên một cloud database, cloud database này sẽ đóng vai trò như một server lưu trữ cũng như việc truy xuất thông tin đăng nhập và đăng ký tài khoản. Cũng như sẽ đảm bảo rằng các tài khoản được tạo sẽ dùng chung cho nhiều server khác nhau. Cấu trúc của một tài khoản trong cơ sở dữ liệu bao gồm các thông tin:

- Username: Tên tài khoản dùng để đăng nhập của người chơi.
- Password: Là phần password + salt đã được hash.
- Point: Cơ chế tính điểm của trò chơi.
- Death: Số lần người chơi rơi xuống nước.
- Status: Trạng thái online hoặc offline của người chơi.


```
_id: ObjectId('646f1bcd3dbacf2aacd75d4e')
username: "kaimink"
password: "57484F150FF9D26EA0549E31CF2F5C1767E351BDACA9B554BB530089A1031808"
salt: "5/25/2023 3:26:53 PM"
isOnline: "false"
Point: 2
Death: 6
```


Figure 10: Dữ liệu tài khoản được lưu trên database.

4.2 Quản lý server

Khi đã đăng nhập người dùng sẽ được hiển thị một danh sách bao gồm các server đang hoạt động trong hệ thống và có thể đăng nhập vào bất cứ server nào nằm trong danh sách này. Danh sách server này sẽ được lưu và thay đổi trạng thái trong cơ sở dữ liệu mỗi khi bật hoặc tắt.

```
_id: ObjectId('646c74c6cca217246c849516')
IP: "26.191.34.211"
port: "7777"
status: "offline"
```

Figure 11: Quản lý server status.

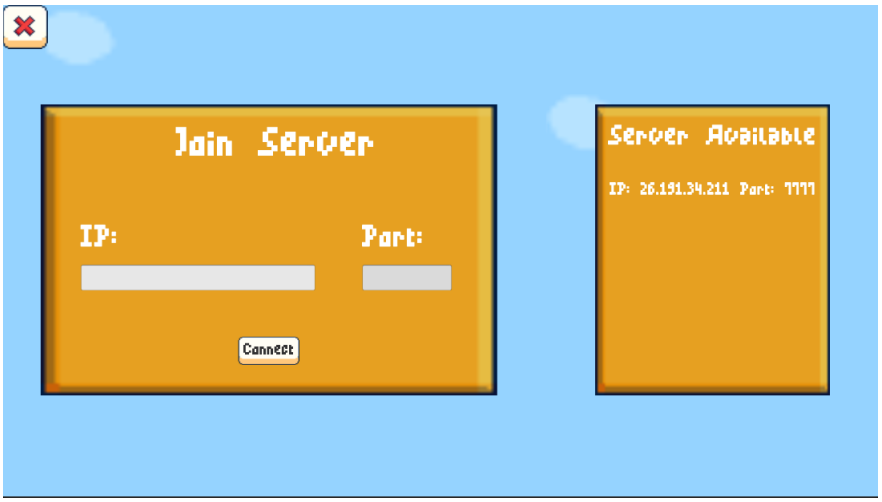

Figure 12: Hiển thị danh sách server online.

4.3 Bảo mật trong việc lưu trữ

4.3.1 Hashing

Hàm băm là hàm một chiều cho phép ta thực hiện biến đổi mật khẩu ban đầu thành một chuỗi thể hiện cho nó và không thể giải mã ngược. Mỗi giá trị băm sẽ chỉ tương ứng với một mật khẩu. Nó cho phép ta thực hiện việc gửi dữ liệu mật khẩu và một số thông tin quan trọng lên server mà không bị lộ nội dung gốc, nhưng vẫn được sử dụng để xác thực.

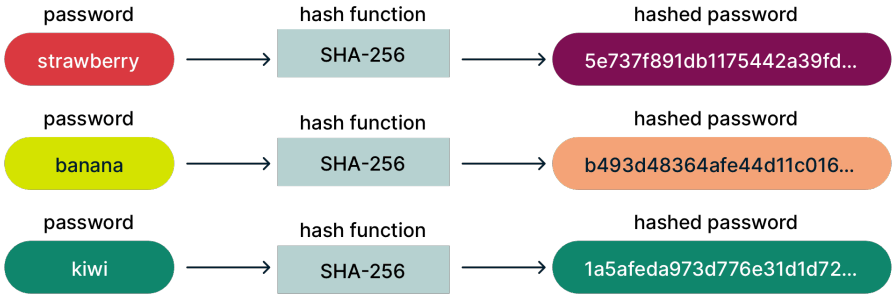

Figure 13: Quá trình hash password.

4.3.2 Salting

Như đã trình bày ở trên, việc tra ngược dữ liệu từ giá trị băm là điều không thể. Tuy vậy vẫn có một số cách để có thể gián tiếp suy ra được mật khẩu. Một số phương pháp phổ biến như Brute Force và Dictionary Attack hay có thể hiểu đơn giản hơn là thử và đoán. Tuy nhiên điều này cũng đặt ra thêm 2 vấn đề bao gồm:

- Hai password trùng nhau: Việc người dùng đặt mật khẩu trùng nhau thì giá trị băm sẽ giống nhau.

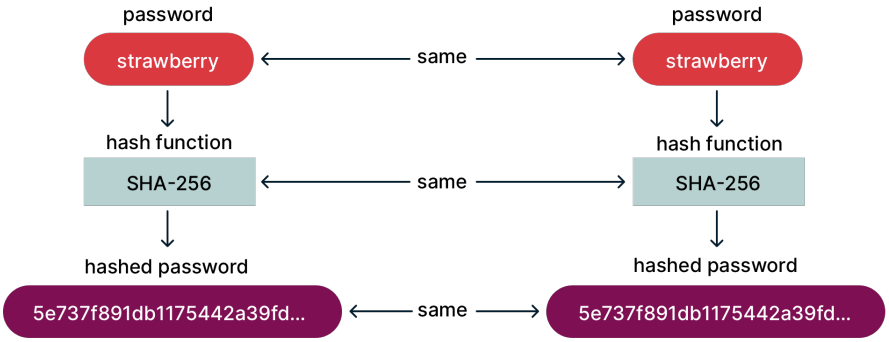

Figure 14: Hai mật khẩu trùng nhau.

- Mật khẩu dễ đoán: Người dùng đặt mật khẩu là một chuỗi số hoặc ký tự đơn giản (ví dụ như 12345678), điều này sẽ khiến cho mật khẩu dễ dàng bị tra ngược khi kẻ tấn công sử dụng một số kiểu mật khẩu đơn giản để suy ra giá trị băm.

Salting hoạt động bằng cách thêm một ký tự ngẫu nhiên vào mật khẩu cùng một ký tự ngẫu nhiên mỗi lần, nhưng người dùng và người tạo ra đều không biết nó là gì. Sau đó salt sẽ được thêm vào phần mật khẩu đã tạo và tạo giá trị băm. Vậy nên quá trình này sẽ giúp ta giải quyết được 2 vấn đề trên đồng thời cũng tăng chi phí tính toán của kẻ tấn công nếu như dữ liệu trên database bị xâm phạm.

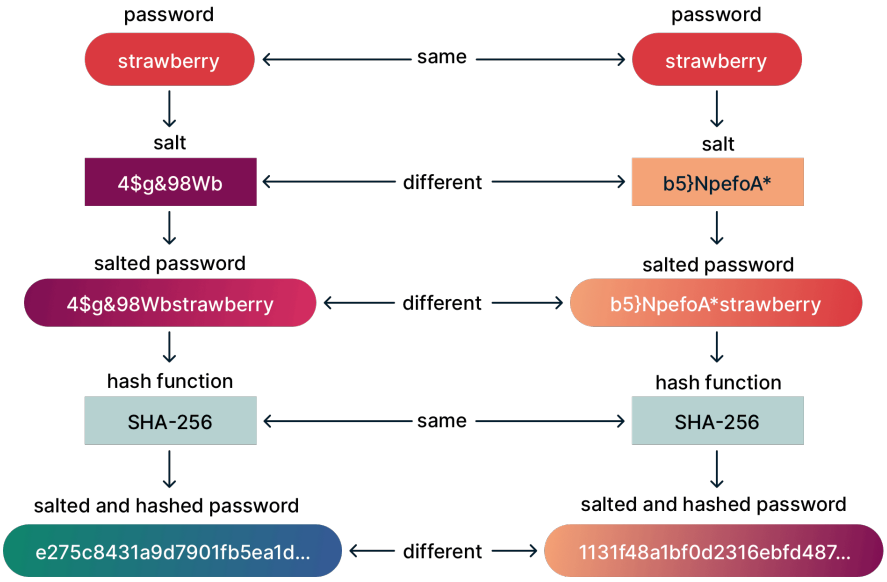

Figure 15: Hash & salt password.

4.3.3 Triển khai hash & salt

Listing 5: PlayerControler.cs

```
using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Core.Authentication;
using MongoDB.Driver.Linq;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.UI;

public class DatabaseManager : MonoBehaviour
{
    // Start is called before the first frame update
    MongoClient client = new MongoClient("mongodb+srv://DuuuKieee:899767147
        @loginserver.hqnkiia.mongodb.net/?retryWrites=true&w=majority");
    IMongoDatabase database;
    IMongoCollection<BsonDocument> collection, servercollection;
    private UIManager UI;
    [SerializeField] public Text leaderBoard, serverCount;

    void Start()
    {
        database = client.GetDatabase("UserDB");
        collection = database.GetCollection<BsonDocument>("UserCollection");
        servercollection = database.GetCollection<BsonDocument>("
            ServerCollection");
```

```
28          UI = GameObject.FindGameObjectWithTag("UI").GetComponent<UIManager>();
29      }
30      public async void serverList()
31      {
32          var filter = Builders<BsonDocument>.Filter.Eq("status", "online");
33          var serveronline = await servercollection.Find(filter).ToListAsync();
34          var sb = new StringBuilder();
35          foreach (var server in serveronline)
36          {
37          var ip = server["IP"].AsString;
38          var port = server["port"].AsString;
39          sb.AppendLine($"IP: {ip} Port: {port}");
40          }
41          serverCount.text = sb.ToString();
42      }
43
44      // Update is called once per frame
45      public async void HomepageManager(string Username, string Password)
46      {
47          string user_ = Username;
48          var filter = Builders<BsonDocument>.Filter.Eq("username", user_);
49          var user = await collection.Find(filter).FirstOrDefaultAsync();
50          if(user == null)
51          {
52              if (UI.isLoginPage == true)
53              {
54                  UI.DisplayNoti("Invalid Username or Password.", false);
55              }
56              else
57              {
58                  if (Password.Length >= 8 && Password.Length <= 15)
59                  {
60                      var salt_ = DateTime.Now.ToString();
61                      string pass_ = HashPassword(Password + salt_);
62                      var document = BsonDocument.Parse($"{{ username: \"{user_
                            }\", password: \"{pass_}\", salt: \"{salt_}\",Point:
                            {0},Death: {0} }}");
63                      await collection.InsertOneAsync(document);
64                      UI.DisplayNoti("Congratulations, your account has been
                            successfully created.", true);
65                  }
66                  else
67                  {
68                      UI.DisplayNoti("Your password must be at least 8
                            characters.", false);
69                  }
70
71              }
72          }
73          else
74          {
75              if(UI.isLoginPage == true)
76              {
77                  var salt_ = user["salt"].AsString;
78                  string pass_ = HashPassword(Password + salt_);
79                  var savedPassword = user["password"].AsString;
80                  if (savedPassword.Equals(pass_, StringComparison.
                        OrdinalIgnoreCase))
81                  {
82                      UI.ConnectToServerMenu();
83                      Debug.Log("Da chay");
84                  }
```

```
85                  else
86                  {
87                      UI.DisplayNoti("Invalid Username or Password.", false);
88                  }
89              }
90              else
91              {
92                  UI.DisplayNoti("This username is already being used.", false);
93              }
94          }
95      }
96
97      public void GenerateLeaderBoardGlobal()
98      {
99          var filter = Builders<BsonDocument>.Filter.Empty;
100         var sort = Builders<BsonDocument>.Sort.Descending("Point");
101         var limit = 5;
102         var topPlayers = collection.Find(filter).Sort(sort).Limit(limit).
                ToList();
103         var sb = new StringBuilder();
104         foreach (var player in topPlayers)
105         {
106          var name = player.GetValue("username").AsString;
107          var point = player.GetValue("Point").AsInt32;
108          var death = player.GetValue("Death").AsInt32;
109          double PD = 0;
110          if(death == 0)
111          {
112             PD = point;
113          }
114          else if (point == 0)
115          {
116             PD = 0;
117          }
118          else if (point > 0 && death > 0)
119          {
120             PD = point/death;
121             }
122          sb.AppendLine($"Username: {name}");
123          sb.AppendLine($"Point: {point} Death: {death} P/D: {Math.Round(PD,3).
                ToString()}");
124         }
125         leaderBoard.text = sb.ToString();
126
127     }
128
129     string HashPassword(string password)
130     {
131         SHA256 hash = SHA256.Create();
132
133         var passwordBytes = Encoding.UTF8.GetBytes(password);
134
135         var hashedpassword = hash.ComputeHash(passwordBytes);
136
137         return BitConverter.ToString(hashedpassword).Replace("-", "");
138     }
139 }
```

Việc Hashing và Salting sẽ được thực hiện khi ta tiến hành đăng ký tài khoản hay khi đăng nhập. Khi đăng ký tài khoản server sẽ kiểm tra tài khoản đã tồn tại trên database hay chưa, nếu chưa thì sẽ tiến hành tạo Salt bằng cách chọn Salt là ngày giờ mà tài khoản được đăng ký,

sau đó tiến hành Hash dữ liệu là một chuỗi bao gồm Password + Salt và các thông tin khác lên cơ sở dữ liệu. Trong phạm vi đồ án hàm băm SHA-256 sẽ được sử dụng.

5 Triển khai ứng dụng

5.1 Triển khai qua LAN

Để có thể triển khai ứng dụng qua LAN ta cần một số điều kiện từ 2 bên Server và Client là:

- Kết nối chung một mạng cục bộ: Cả 2 sẽ phải kết nối chung một mạng cục bộ như wifi.
- Địa chỉ IP và Port: Client cần phải biết được địa chỉ IP và Port của server được tạo để có thể kết nối.

Ngoài ra ta có thể sử dụng loop back ip là 127.0.0.1 để có thể tiến hành kết nối nếu server và client cùng chạy trên một máy.

5.2 Triển khai qua internet

Để triển khai server qua internet thì việc sử dụng Local IP để kết nối là điều không thể. Vậy nên ta cần sử dụng một số dịch vụ cung cấp từ bên thứ 3, dưới đây sẽ là một số dịch vụ có thể được sử dụng.

5.2.1 Unity Gaming Service

Unity Gaming Service là một bộ công cụ để hỗ trợ nhà phát triển cung cấp các giải pháp cho multiplayer game, vận hành trò chơi, quản lý người dùng và kiếm tiền.

Trong đó dịch vụ Game Server Hosting cho phép nhà phát triển chạy và quản lý server trên Cloud. Nó cho phép ta thực hiện host một server được build trên hệ điều hành Linux và các client có thể kết nối với chúng qua internet. Điểm đặc biệt của dịch vụ này là chúng hoàn toàn miễn phí với các dự án nhỏ, vậy nên ta sẽ dùng dịch vụ này để thực hiện host server trong quy mô đồ án môn học.

5.2.2 Các dịch vụ Server Dedicated khác

Ta cũng có thể thực hiện triển khai một Game Server qua một số dịch vụ như Cloud, VPS, Server Dedicated thông qua các bên trung gian khác. Tuy nhiên các dịch vụ này thường sẽ tốn chi phí để có thể đặt và thuê máy chủ.

6 Kết quả

6.1 Đánh giá

Phần ứng dụng được hoàn thiện khá tốt ở các phần, có thể chạy được trên Local, LAN và internet qua dịch vụ cloud. Tuy nhiên, vẫn còn một số chức năng dự định thêm vào gặp khó khăn trong quá trình triển khai. Dưới đây sẽ là một số điểm mà nhóm đã làm và chưa làm được.

Figure 16: Người chơi kết nối qua mạng.

6.1.1 Những điểm làm được

Một số phần được nhóm thực hiện hoàn chỉnh và hoạt động tốt trong quá trình chạy ứng dụng như:

- Phần game được hoàn thiện khá tốt các tính năng để chơi.
- Làm việc với các luồng nhập xuất
- Kết nối và làm việc với database.
- Sử dụng đa luồng để tối ưu các tác vụ.
- Hoàn thiện được cơ chế đăng ký, đăng nhập và lưu trữ trạng thái.
- Nhiều client có thể cùng kết nối tới một hoặc nhiều server khác trong hệ thống.
- Phân chia các công việc được thực hiện ở phía server hợp lý.
- Nhóm đã có thể thực hiện demo qua LAN và Internet.

6.1.2 Những điểm chưa làm được

Tuy vậy vẫn có một số chức năng chưa chỉn chu và hoàn thiện:

- Phần hiển thị server đang online chưa làm được do server cloud của Unity Game Server Hosting không thể hiện IP qua code.
- Việc tạo phòng chơi phải phụ thuộc vào số lượng server.
- Chức năng quên mật khẩu và các biện pháp khôi phục mật khẩu người dùng.

6.2 Phân công công việc

Trong quá trình triển khai, các công việc phân công được các thành viên hoàn thành khá tốt. Tiến độ triển khai hoàn thành sớm hơn dự tính, các vấn đề cần sự giao tiếp giữa các thành viên được giải quyết một cách hoàn chỉnh. Từng thành viên trong nhóm đều hoàn thành tốt công việc của mình trong quá trình thực hiện đồ án.

NO	Tasks	Description	Contributor
1	Game Developer	Thực hiện triển khai tựa game, bao gồm các phần gameplay, game mechanic,... có trong tựa game	Trần Thành Lợi
2	Client Server Architechture	Triển khai mô hình kết nối mạng Client-Server vào tựa game. Giải quyết các vấn đề gửi nhận dữ liệu, đồng bộ thông tin.	Trần Thành Lợi, Huỳnh Đình Khải Minh, Nguyễn Nguyễn Duy An
3	Database Manager	Quản lý thông tin tài khoản, server qua một hệ thống cơ sở dữ liệu. Được thể hiện qua quá trình đăng nhập, đăng ký tài khoản.	Huỳnh Đình Khải Minh
4	Account Security	Các vấn đề liên quan tới bảo mật thông tin người dùng như tài khoản và mật khẩu.	Nguyễn Nguyễn Duy An
5	Testing and Optimization	Chạy thử và tối ưu hóa tựa game. Áp dụng một số Design Pattern như Singleton Pattern và sử dụng các tài nguyên một cách hợp lý.	Trần Thành Lợi, Nguyễn Nguyễn Duy An, Huỳnh Đình Khải Minh

Figure 17: Bảng phân công công việc.

7 Tài liệu tham khảo

Arias, D. (2018). Adding salt to hashing: A better way to store passwords. *Auth0*. https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

BillWagner. (n.d.). .net documentation. *learn.microsoft.com*. https://learn.microsoft.com/en-us/dotnet/

Justice, B. (2020). Client-server architecture. https://www.brutalhack.com/blog/client-server-architecture/

MongoDB, I. (n.d.). Mongodb documentation. *mongodb.com/docs*. https://www.mongodb.com/docs/

Technologies, U. (n.d.). Unity documentation. *docs.unity.com*. https://docs.unity.com/

Yahyavi, A., & Kemme, B. (2013). Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Computing Surveys (CSUR), 46*. https://doi.org/10.1145/2522968.2522977