# BAB 4

# IMPLEMENTASI DAN PEMBAHASAN

## 4. 1 Implementasi Sistem

*Photon Unity Networking* (PUN) telah diimplementasikan pada sebuah game ber-*genre survival* berbentuk labirin. Implementasi tersebut dilakukan dengan menggunakan aplikasi *Unity 3D* dengan bahasa *C#.* Berdasarkan analisis dan perancangan pada BAB 3, maka pada implementasi dan pembahasan merupakan tahap pengoprasian dan uji coba dalam keadaan sebenarnya. Berikut merupakan implementasi dari program yang ada:

### *4. 1. 1 Input Player*

Pada bagian *input player* merupakan langkah persiapan *player* sebelum memasuki permainan.


**Gambar 4.1 Antarmuka pengaturan *game play***

Pada gambar 4.1 *input* yang diminta antara lain; nama *player*, tipe labirin, jumlah NPC dan pilihan untuk menyalakan mode *debug* atau tidak. Input tersebut

digunakan ketika dalam metode pencocokan room. *Value* dari *input* disimpan dalam *PlayerPrefs*.

```
void Start()
{
    string defaultName = string.Empty;
    InputField _inputField = this.GetComponent<InputField>();
    if (_inputField != null)
    {
        if (PlayerPrefs.HasKey(playerNamePrefKey))
        {
            defaultName = PlayerPrefs.GetString(playerNamePrefKey);
            _inputField.text = defaultName;
        }
    }
    PhotonNetwork.NickName = defaultName;
}
#endregion
#region Public Methods
// Menetapkan nama pemain, dan menyimpannya di PlayerPrefs untuk ses
// <param name = "value"> Nama Pemain </param>
0 references
public void SetPlayerName(string value)
{
    // #Important
    if (string.IsNullOrEmpty(value)){return;}
    PhotonNetwork.NickName = value;
    PlayerPrefs.SetString(playerNamePrefKey, value);
}
```

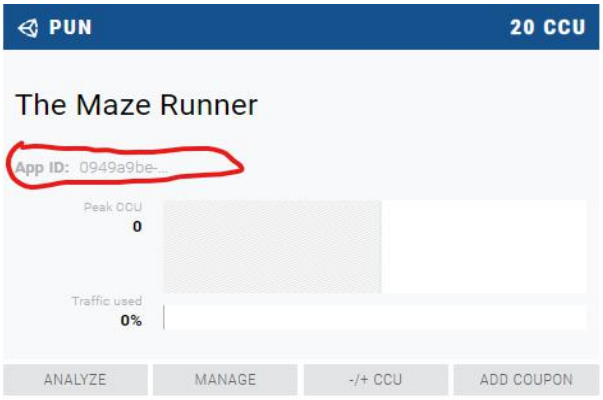
**Gambar 4.2 *Script* potongan dari *PlayerNameinputField.cs***

Fungsi dari potongan *script* yang terdapat pada gambar 4.2 adalah pada bagian metode *Start* untuk mengambil nilai *value* nama player dari *PlayerPrefs*, Sedangkan bagian metode *SetPlayerName* untuk menyimpan nilai *value* nama *player* dari *InputField* ke dalam *PlayerPrefs*. *Value* yang disimpan dalam *PlayerPrefs* menggunakan kunci sebagai berikut:

1. Nama *player* menggunakan kunci "*PlayerName*".
2. Tipe labirin menggunakan kunci "*TypeLabirin*" dan "*TypeLabirinDiff*".
3. Jumlah NPC menggunakan kunci "*NPCOnMap*".
4. Mode *debug* menggunakan kunci "*DebugerMode*".

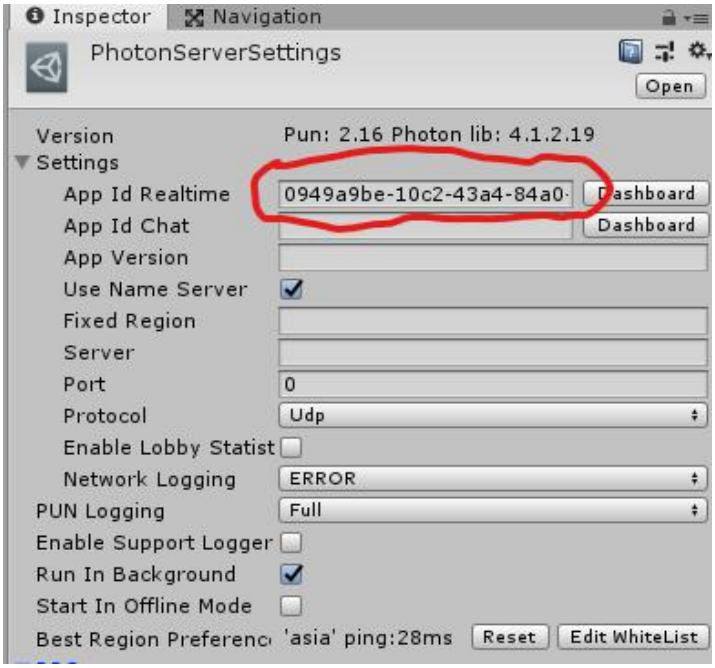
### 4. 1. 2 Memulai sambungan ke *Server*

*Server* yang digunakan dalam penelitian ini adalah *server* yang sudah disediakan oleh PUN.




**Gambar 4.3 *Photon Cloud Applications***

Dari gambar 4.3 menunjukkan *server* akan memberikan App ID yang perlu dimasukkan ke *Project Unity*. App ID dimasukkan ke dalam *Photon Server Setting* seperti pada gambar 4.4.




**Gambar 4.4 *Photon Server Setting***

Saat memulai permainan, tombol main akan memanggil metode *connect* pada *Launcher.cs*.

```
public void Connect()
{
    if (PlayerPrefs.GetString("PlayerName").Trim().Equals(""))
    {
        menuRule.GetComponent<MainMenu>().Pengaturan();
    }
    else
    {
        isConnecting = PhotonNetwork.ConnectUsingSettings();
        progressLabel.SetActive(true);
        controlPanel.SetActive(false);
        Debug.Log("PUN Basics Tutorial/Launcher: Run connect method");
        // kami memeriksa apakah kami terhubung atau tidak, kami bergabung jika kami terhubung, jik
        if (PhotonNetwork.IsConnected)
        {
            Debug.Log("PUN Basics Tutorial/Launcher: Photon ready was connected");
            JoinKeRoom();
        }
        else
        {
            Debug.Log("PUN Basics Tutorial/Launcher: PhotonNetwork.IsConnected is not connected");
            // #Critical, pertama-tama kita harus terhubung ke Photon Online Server.
            PhotonNetwork.ConnectUsingSettings();
            PhotonNetwork.GameVersion = gameVersion;
        }
    }
}
```

**Gambar 4.5 *Script* metode *connect* untuk menghubungkan ke Server PUN**

Dari gambar 4.5 menunjukkan program akan memeriksa koneksi internet, jika tersedia koneksi, program akan mencoba mencari *room* yang tersedia.

Dalam hal ini kemungkinan kode PUN mengalami masalah oleh karena itu diperlukan metode *OnConnectedToMaster* untuk menangkap status koneksi ke *server* PUN dan memulai pencarian kembali.

```
public override void OnConnectedToMaster()
{
    Debug.Log("PUN Basics Tutorial/Launcher: OnConnectedToMaster() was called by PUN");
    // #Critical: Yang pertama kami coba lakukan adalah bergabung dengan ruang potensia
    if (isConnecting)
    {
        JoinKeRoom();
        isConnecting = false;
    }
}
```

**Gambar 4.6 *Script OnconnectToMaster***

### *4. 1. 3* Melakukan Pencocokan *Room*

Saat koneksi telah tersambung ke *server* PUN, selanjutnya adalah memulai pencarian *room* yang sesuai dengan spesifikasi, menggunakan parameter tipe labirin (*TypeLabirin* dan *TypeLabirinDiff*) dan jumlah NPC (*NPCOnMap*).

```
void JoinKeRoom() {
    Hashtable prop=null;
    if (PlayerPrefs.GetInt("TypeLabirin") == 0)
    {
        prop = new Hashtable()
        {
            { "tm", 0},
            { "nc", PlayerPrefs.GetInt("NPCOnMap") }
        };
    }
    else if (PlayerPrefs.GetInt("TypeLabirin") == 1)
    {
        prop = new Hashtable()
        {
            { "tm", PlayerPrefs.GetInt("TypeLabirinDiff") + 1 },
            { "nc", PlayerPrefs.GetInt("NPCOnMap") }
        };
    }
    // #Critical: Yang pertama kami coba lakukan adalah bergabung der
    PhotonNetwork.JoinRandomRoom(prop, maxPlayersPerRoom);
}
```

**Gambar 4.7 *Script* pencarian room**

### 4. 1. 4 Pencocokan Berhasil dan Bergabung ke Room

Jika terdapat ruang dengan spesifikasi yang sesuai dan dalam memungkinkan untuk bergabung, maka akan berpindah ke adegan selanjutnya, ini tergantung tipe labirin yang sebelumnya dipilih, area statis untuk labirin dan area dinamis untuk labirin dinamis.

```
public override void OnJoinedRoom()
{
    Debug.Log("PUN Basics Tutorial/Launcher: OnJoinedRc
    // #Critical: Kami hanya memuat jika kami adalah pe
    if (PhotonNetwork.CurrentRoom.PlayerCount == 1)
    {
        if (PlayerPrefs.GetInt("TypeLabirin") == 1)
            PhotonNetwork.LoadLevel("Arena Dinamis");
        else
            PhotonNetwork.LoadLevel("Arena Statis");
    }
}
```

**Gambar 4.8 Script untuk me-load adegan**

### 4. 1. 5 Pencocokan Gagal dan Membuat *Room* Baru

Namun jika pencocokan gagal atau tidak menemukan *room* dengan spesifikasi yang dimaksud, maka akan dibuat *room* baru menggunakan spesifikasi tersebut.

```
public override void OnJoinRandomFailed(short returnCode, string message)
{
    Debug.Log("PUN Basics Tutorial/Launcher:OnJoinRandomFailed() was called
    RoomOptions roomOptions = new RoomOptions();
    roomOptions.MaxPlayers = maxPlayersPerRoom;
    roomOptions.IsOpen = true;
    roomOptions.IsVisible = true;
    roomOptions.CustomRoomPropertiesForLobby = new string[] { "tm","nc" };
    if (PlayerPrefs.GetInt("TypeLabirin") == 0)
    {
        roomOptions.CustomRoomProperties = new Hashtable()
        {
            { "tm", 0},
            { "nc", PlayerPrefs.GetInt("NPCOnMap") }
        };
    }
    else if (PlayerPrefs.GetInt("TypeLabirin") == 1)
    {
        roomOptions.CustomRoomProperties = new Hashtable()
        {
            { "tm", PlayerPrefs.GetInt("TypeLabirinDiff")+1 },
            {"nc", PlayerPrefs.GetInt("NPCOnMap") }
        };
    }
    // #Critical: kami gagal bergabung dengan ruang acak, mungkin tidak ada
    PhotonNetwork.CreateRoom(null, roomOptions, TypedLobby.Default);
}
```

**Gambar 4.9 *Script* untuk membuat room baru**

Dari gambar 4.9 *room* yang dibuat merupakan *room* dengan spesifikasi yang dicari sebelumnya. *Room* memuat adegan yang sudah ada di dalam *game*, sehingga ketika membuat *room* sebenarnya kita me-*load* data adegan yang sudah di definisikan, beberapa *object* sudah tersedia dan tidak perlu dibuat ulang, namun beberapa *object* tambahan perlu dibuat dan disinkronisasikan.

Semua *object* yang akan dibuat dan disinkronisasikan harus memenuhi beberapa syarat sebagai berikut :

1. *Object* harus dibuat menjadi *prefab*.

2. *Object* memuat komponen *Photon View*.
3. Disimpan pada *directory* "*Resources*".

### 4. 1. 6 Membuat Karakter Utama Player

*Player* yang gagal terhubung secara otomatis teridentifikasi sebagai *Master Client* dan mempunyai hak untuk membuat lingkungan dan *object* di dalam *room*. Untuk *object* yang pertama kali dibuat setelah membuat *room* adalah karakter utama *player*. Untuk perintah pembuatan karakter utama terdapat dalam *GameManager.cs*.

```
void Start()
{
    if (PhotonNetwork.IsMasterClient) {Debug.Log("Master");}
    Debug.Log("Registered Time Delta Start " + PhotonNetwork.ServerTimestamp);
    mulai = true;
    if (playerPrefab == null)
    {
        Debug.LogError("<Color=Red><a>Missing</a></Color> playerPrefab Reference. Please set it u
    }
    else
    {
        if (PlayerManager.LocalPlayerInstance == null)
        {
            Debug.LogFormat("We are Instantiating LocalPlayer from {0}", SceneManagerHelper.Activ
            // kita ada di kamar. menelurkan karakter untuk pemain lokal. itu akan disinkronkan d
            respownPoint = PhotonNetwork.CurrentRoom.PlayerCount;
            player =PhotonNetwork.Instantiate(
                this.playerPrefab.name,
                GameObject.Find("Respown"+ respownPoint).GetComponent<Transform>().position,
                Quaternion.identity,
                0
                );
        }
        else
        {
            Debug.LogFormat("Ignoring scene load for {0}", SceneManagerHelper.ActiveSceneName);
        }
    }
}
```

**Gambar 4.10 *Script* membuat karakter *player***

Pada saat *player* diinstansiasi, saat itu juga *player* membuat berbagai *object* lain bersamanya seperti UI *controller* dan status. Hal tersebut dilakukan

oleh *PlayerManager.cs* berikut adalah potongan program yang diinstansiasi berbagai macam *object* terkait :

```
void Start()
{
    logSave = GameObject.Find("Log Saver").GetComponent<LogSaverAndSender>();
    sudahSave = false;
    debugMode = PlayerPrefs.GetInt("DebugerMode");
    recoverHP = true;
    hit_v = false;
    total_delay = 0;
    count_delay = 0;
    CameraForwad _cameraWork = this.gameObject.GetComponent<CameraForwad>();
    characterController = GetComponent<CharacterController>();
    PhotonNetwork.NickName = PlayerPrefs.GetString("PlayerName");
    if (photonView.IsMine && actionButton1 != null && joy != null && angle != null
        && playerHp != null && _cameraWork != null) {
        //membuat HP bar curret Player
        _uiGo = Instantiate(playerHp);
        //membuat controller
        _uiAng = Instantiate(angle);
        _uiJoy = Instantiate(joy);
        _uiAcB = Instantiate(actionButton1);
        _cameraWork.OnStartFollowing();
        _uiGo.SendMessage("SetTarget", this, SendMessageOptions.RequireReceiver);
        joy_v = _uiJoy.GetComponent<NavigationVirtualJoystick>();
        angle_v = _uiAng.GetComponent<AngleVirtualJoystick>();
        actionButton1_v = _uiAcB.GetComponent<TapButton>();
    }
    else if (OtherHp != null)
    {
        //membuat tampilan HP bar di Player lain
        _uiGoOther = Instantiate(OtherHp);
        _uiGoOther.SendMessage("SetTarget", this, SendMessageOptions.RequireReceiver);
    }

    animator = GetComponent<Animator>();
    if (!animator)
    {
        Debug.LogError("PlayerAnimatorManager is Missing Animator Component", this);
    }
    if (photonView.IsMine) {
        gamePlay = GameObject.Find("Environment").GetComponent<GamePlay>();
        hpCritical = GameObject.Find("BloodDmg");
        death = GameObject.Find("Game Over");
        panelWinner = GameObject.Find("WinnerPanel");
        winnerCountDown = GameObject.Find("WinnerCountDown").GetComponent<Text>();
        deathCountDown = GameObject.Find("Button Return to Menu").GetComponent<Text>();
        hpCritical.SetActive(false);
        death.SetActive(false);
        panelWinner.SetActive(false);

        //bagian radar NPC
        remNPC1 = GameObject.Find("RemNPC1").GetComponent<Text>();
        remNPC2 = GameObject.Find("RemNPC2").GetComponent<Text>();
        remNPC3 = GameObject.Find("RemNPC3").GetComponent<Text>();
    }
    status_ping = GameObject.Find("Ping").GetComponent<Text>();
    status_syns = GameObject.Find("Time Latency" + photonView.Owner.ActorNumber).GetComponent<Text>();
    string_avg = "0";
}
```

**Gambar 4.11 *Script* UI *controller* dan status**

Dari kode program pada gambar 4.11 beberapa *object* yang dibuat adalah :

1. HP *Player* utama.
2. HP *Player* musuh.
3. *Control Virtual Joystick*.
4. *Control Camera*.
5. *Control Attack*.

### 4. 1. 7 Membuat Lingkungan Labirin

Untuk pembuatan lingkungan labirin di ambil alih oleh bagian yang membahas topik kecerdasan buatan untuk membuat labirin. Namun pada dasarnya untuk membuat suatu *object* yang disinkronisasikan antar *player* menggunakan perintah *Photon.InstantiateSceneObject*. Potongan kode program *MazeGenerator.cs* untuk membuat labirin adalah sebagaimana terlihat pada gambar 4.12.

```
tempTembok = PhotonNetwork.InstantiateSceneObject(
    this.tembok.name,
    myPos,
    Quaternion.Euler(0, 90, 0)
    ) as GameObject;
tempTembok = PhotonNetwork.InstantiateSceneObject(
    this.tembok.name,
    myPos,
    Quaternion.identity
    ) as GameObject;
```

**Gambar 4.12 *Script* untuk membuat dinding**

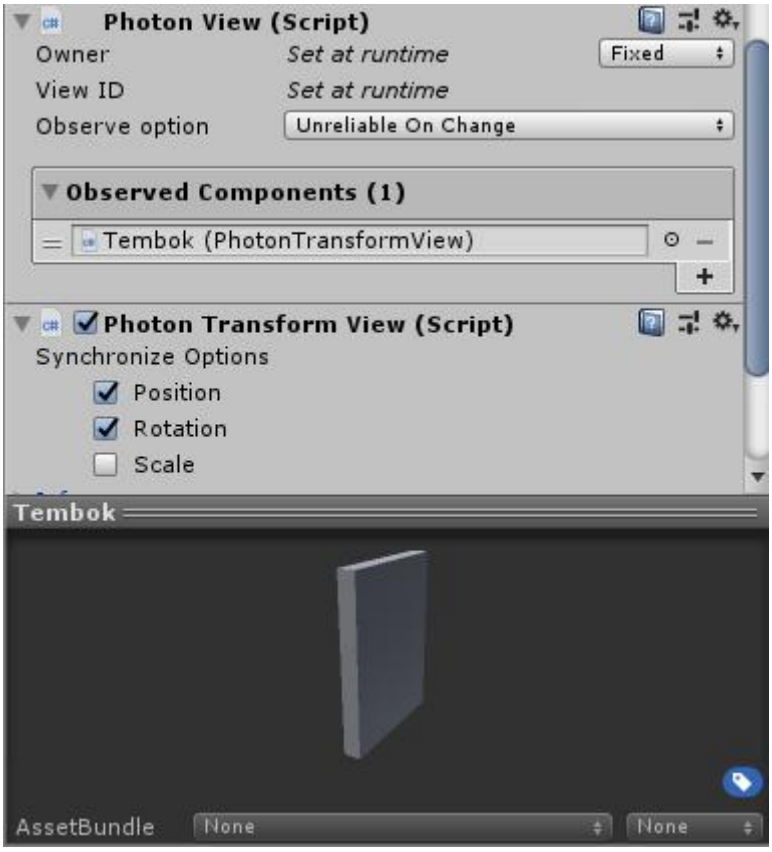
```
switch (tetangga)
{
    case 1:
        PhotonNetwork.Destroy(sel[selSkrg].atas);
        break;
    case 2:
        PhotonNetwork.Destroy(sel[selSkrg].kanan);
        break;
    case 3:
        PhotonNetwork.Destroy(sel[selSkrg].kiri);
        break;
    case 4:
        PhotonNetwork.Destroy(sel[selSkrg].bawah);
        break;
    default:
        break;
}
```

**Gambar 4.13 *Script* untuk menghapus dinding**

Dikarenakan *object* yang dibuat bersifat akan ada selama *room* tersebut tersedia maka yang membuat *object* tersebut haruslah *Master Client*. *Client* biasa dapat membuat *object* namun sifatnya hanya sementara sampai client meninggalkan *room* tersebut, perintah yang digunakan menggunakan *Photon.Instantiate.*

### 4. 1. 8 Sinkronisasi Bentuk Labirin

Bentuk labirin yang disinkronisasikan merupakan sekumpulan *game object* tembok. Dalam *object* tersebut menggunakan komponen *Photon View* untuk memonitoring setiap perubahan yang terjadi dengan komponen observasi komponen *Photon Transform*. Sehingga setiap perubahan nilai pada *transform* akan dipantau oleh *Photon View* dan disinkronisasikan, data yang sinikronisasikan meliputi data posisi dan rotasi, terkecuali untuk skala karena tidak ada manipulasi skala labirin selama game berlangsung.




**Gambar 4.14 *Inspector object* Tembok**

### 4. 1. 9 Membuat NPC

Pada bagian ini NPC yang dibuat akan menggunakan kecerdasan buatan untuk mencari setiap *player*. Algoritma untuk kecerdasan buatan tersebut diambil alih oleh bagian yang membahas topik untuk algoritma A*.

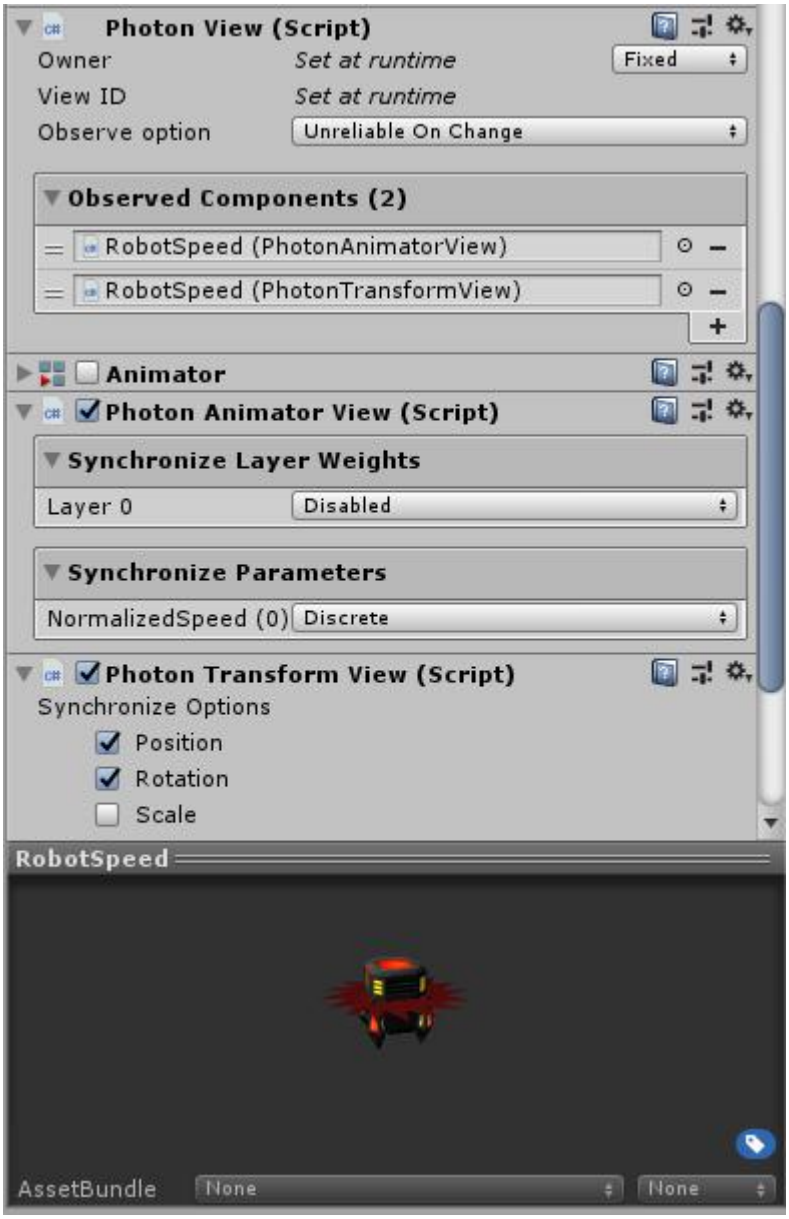
Untuk membuat membuat NPC sama halnya seperti membuat labirin yaitu menggunakan *Photon.InstantiateSceneObject.* Potongan kode program *GamePlay.cs* untuk membuat object NPC adalah seperti yang terlihat pada gambar 4.14.

```
public void CreateNPC(int i){
    string tempatNPC =
        (PlayerPrefs.GetInt("TypeLabirin") == 1) ?
        PlayerPrefs.GetInt("TypeLabirinDiff") + 1 + "" : "";
    PhotonNetwork.InstantiateSceneObject(aStart.name, Vector3.zero, Quaternion.identity);
    if (i == 1){
        PhotonNetwork.InstantiateSceneObject(
            npc1.name,
            GameObject.Find("RespownNPC"+ tempatNPC+"1").GetComponent<Transform>().position,
            Quaternion.identity);
    }else if (i == 2){
        PhotonNetwork.InstantiateSceneObject(
            npc1.name,
            GameObject.Find("RespownNPC" + tempatNPC + "1").GetComponent<Transform>().position,
            Quaternion.identity);
        PhotonNetwork.InstantiateSceneObject(
            npc2.name,
            GameObject.Find("RespownNPC" + tempatNPC + "2").GetComponent<Transform>().position,
            Quaternion.identity);
    }else if (i == 3){
        PhotonNetwork.InstantiateSceneObject(
            npc1.name,
            GameObject.Find("RespownNPC" + tempatNPC + "1").GetComponent<Transform>().position,
            Quaternion.identity);
        PhotonNetwork.InstantiateSceneObject(
            npc2.name,
            GameObject.Find("RespownNPC" + tempatNPC + "2").GetComponent<Transform>().position,
            Quaternion.identity);
        PhotonNetwork.InstantiateSceneObject(
            npc3.name,
            GameObject.Find("RespownNPC" + tempatNPC + "3").GetComponent<Transform>().position,
            Quaternion.identity);
    }
}
```

**Gambar 4.15 *Script* untuk membuat NPC**

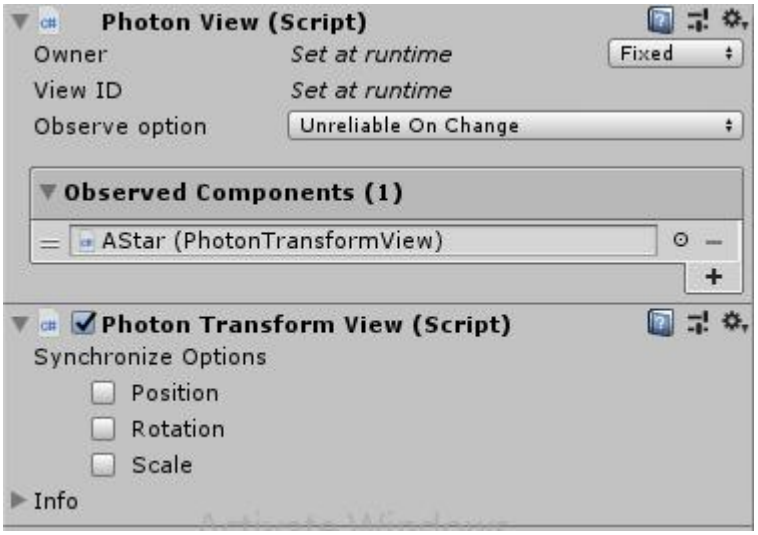
### 4. 1. 10 Sinkronisasi NPC

Untuk NPC hampir sama seperti dinding hanya saja untuk NPC mempunyai tambahan komponen observasi berupa animasi.




**Gambar 4.16 *Inspector object Robot* (NPC)**

Selain itu untuk NPC memerlukan *game object* tambahan yaitu AStar, *object* tersebut diperlukan untuk fungsionalitas dari NPC itu sendiri.



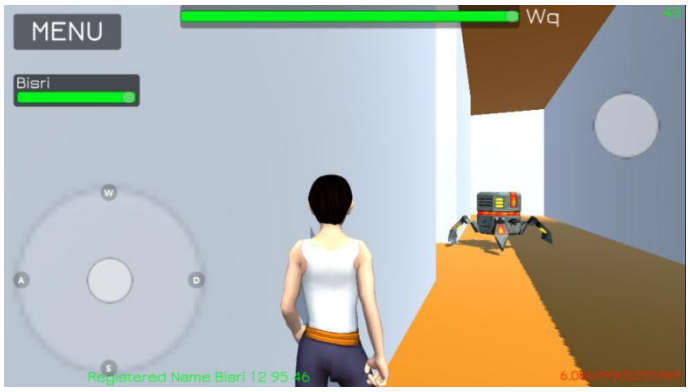

**Gambar 4.17 Inspector object *AStar***

Dari gambar 4.16 terlihat *object* AStar tidak memerlukan observasi komponen apapun, jadi selama *object* tersebut ada di dalam *room* itu sudah cukup.

### 4. 1. 11 Memulai Permainan

Sesaat sebelum permainan sebenarnya berlangsung, para *player* akan ditempatkan dalam suatu ruangan sambil menunggu *player* lain, jumlah maksimal *player* adalah 3. Jika jumlah *player* yang ada dalam *room* hanya satu maka permainan tidak dapat dimulai. Ketika *player* baru bergabung sehingga *player* berjumlah 2 orang akan ada batas waktu 100 detik sampai permainan sebenarnya dimulai. Apabila sampai 100 detik tidak ada *player* lain yang bergabung, maka permainan akan dipaksa untuk mulai. Jika sebelum waktu habis 100 detik dan ada *player* yang bergabung maka waktu mundur akan dipercepat menjadi 10 detik.


**Gambar 4.18 *Player* sedang berada di Ruang Tunggu**




**Gambar 4.19 *Player* sedang berada di dalam Labirin**

Ketika permainan sebenarnya telah dimulai maka room berhenti untuk menerima koneksi baru dari *player* yang akan bergabung. Hal itu karena pengaturan *room* dibuat menjadi tertutup dan tidak terlihat. Perintah tersebut terdapat dalam *GamePlaye.cs*.

```
void GoPlay() {
    countDown.SetActive(false);
    readykuy.SetActive(false);
    //Menutup koneksi player baru
    PhotonNetwork.CurrentRoom.IsOpen = false;
    //membuat room menjadi tidak terlihat oleh player baru
    PhotonNetwork.CurrentRoom.IsVisible = false;
    mulaiPlay = true;
    CreateNPC(PlayerPrefs.GetInt("NPCOnMap"));
    eksekusiSudah = true;
}
```

**Gambar 4.20 *Script* untuk memulai permainan**

### 4. 1. 12 Sinkronisasi Data antar *Player*

Karena player mempunyai data yang harus disinkronisasikan selama game beralngsung maka value tersebut dimasukan ke dalam metode *OnPhotonSerializeView* pada *PlayerManager.cs*.

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info){
    if (stream.IsWriting){
        // We own this player: send the others our data
        //mengirim data attack
        stream.SendNext(hit_v);
        //mengirim data HP
        stream.SendNext(Health);
        //mengirim data tag
        stream.SendNext(gameObject.tag);
    }else{
        //menerima data attack
        hit_v = (bool)stream.ReceiveNext();
        //menrima data HP
        this.Health = (float)stream.ReceiveNext();
        //menghitung waktu yang dibutuhkan untuk mengirim data
        float lag = Mathf.Abs((float)(PhotonNetwork.Time - info.timestamp));
        count_delay++;
        string fill_log = "Registered Name "
            + this.photonView.Owner.NickName + " "
            + count_delay + " " + ((int)(lag * 1000f)) + " "
            + ping;
        Debug.Log(fill_log);
        status_syns.text = fill_log;
        //menerima data tag
        this.gameObject.tag = (string) stream.ReceiveNext();
    }
}
```

**Gambar 4.21 *Script OnPhotonSerializeView***

### 4. 1. 13 Bagian Penerima Serangan

Gambar 4.21 adalah bagian kode program untuk menerima serangan, sehingga terjadi pengurangan *Health Point* (HP). HP dapat berkurang jika *player* memiliki tag “Player”, bersinggungan dengan object yang memiliki tak “*Hit*” atau “*Robot*” dan bukan *player* yang memenangkan permainan.

```
void OnTriggerEnter(Collider other){
    if (other.tag.Trim().Equals("Finish")) {
        //script untuk finish
        gameObject.tag = "Winner";
        speed = 0;
    }
    if (!photonView.IsMine){return;}
    else if (photonView.IsMine && other.name.Trim().Equals("Hit")){
        if (gameObject.tag.Equals("Player")){
            Health -= 0.2f * Time.deltaTime;
        }
    }
}
Unity Message | 0 references
void OnTriggerStay(Collider other){
    if (!photonView.IsMine){return;}
    else if (photonView.IsMine &&  other.tag.Trim().Equals("Robot")){
        if (gameObject.tag.Equals("Player")){
            Health -= 0.1f * Time.deltaTime;
        }
    }
}
```

**Gambar 4.22 *Script* untuk menerima *damage***

Selama *player* bermain status dibedakan berdasarkan *tag*. Ada 3 buat *tag* untuk membedakan status pemain, yaitu:

1. *Player*, *tag* ini bersifat netral dan menandakan bahwa player akan dikejar NPC sebagai *object* sasaran.
2. *Player Death*, ini menandakan bahwa *player* sudah mati atau *game over* sehingga tidak dapat melanjutkan permainan. *Player* dengan *tag* ini sudah mati sehingga tidak akan menjadi target sasaran pengejaran NPC dan akan segera dikeluarkan dari *room* dalam 10 detik. *Tag* ini juga didapatkan ketika salah satu *player* telah mendapat tag *Winner*.
3. *Winner*, *tag* ini merupakan tag kemenangan ketika ada *player* mendapatkan *tag* ini itu berarti menandakan berakhirnya permainan dan

semua *player* yang tersisa (mempunyai *tag player*) akan mendapatkan *tag Player Death*. *Player* dengan *tag* ini tidak akan dikejar NPC.

### 4. 1. 14 Meninggalkan *Room*

Meninggalkan *room* dapat dilakukan dengan sengaja atau pun karena kondisi tertentu seperti menang atau kalah. Namun pada dasarnya ini menggunakan fungsi yang sama dari *GameManager.cs* yaitu *LeaveRoom*.

```
//Metode override dari MonoBehaviourPunCallbacks
//meload scane 0 ketika meninggalkan room
16 references
public override void OnLeftRoom(){
    SceneManager.LoadScene(0);
}
#endregion
#region Public Methods
//Metode yang dapat dipanggil secara bebas
0 references
public void LeaveRoom(){
    PhotonNetwork.LeaveRoom();
}
```

**Gambar 4.23 *Script* untuk meninggalkan *room***

Ketika meninggalkan *room OnLeftRoom* akan mengarahkan *player* ke dalam adegan *Lobby* dan secara otomatis *Photon* akan menghancurkan semua *object* dari adegan *room* yang ditinggalkan. Setelah itu player akan dapat bersiap untuk permainan selanjutnya.

## 4. 2 Pembahasan Sistem

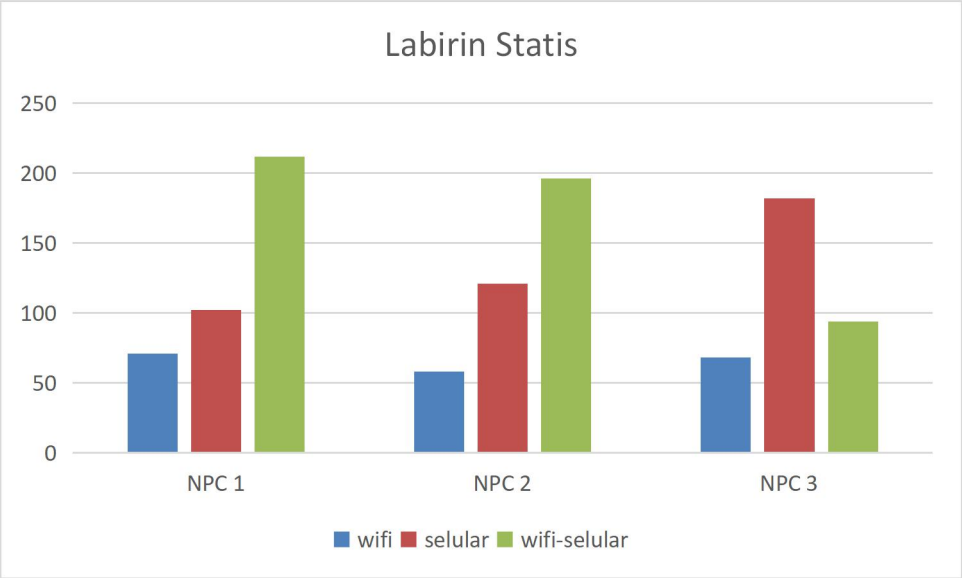
Program aplikasi *game* ini menggunakan teknologi PUN sebagai sarana untuk menjembatani *player* agar dapat bermain secara *multiplayer*. Nilai yang dihasilkan dari implementasi ini berupa *latency*. Data *latency* diperoleh dengan cara absolut pengurangan nilai waktu saat ini oleh waktu saat data dikirimkan.

*float lag = Mathf.Abs((float) (PhotonNetwork.Time - info.timestamp));*

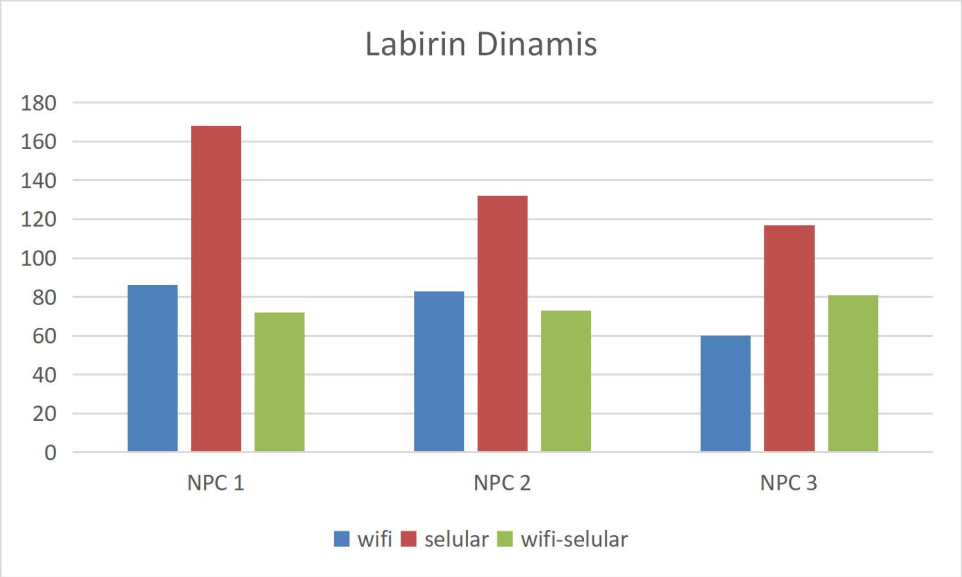
Berikut ini adalah beberapa contoh kasus yang diuji coba:

**Tabel 4.4 Data pengujian**

| No. | Kondisi yang diujikan | | | Rata-rata latency (ms) |
|---|---|---|---|---|
| | Jaringan | Jumlah NPC | Jenis Labirin | |
| 1. | *Wi-fi* | 1 | Statis | 71 |
| 2. | Selular | 1 | Statis | 102 |
| 3. | *Wi-fi* dan selular | 1 | Statis | 212 |
| 4. | *Wi-fi* | 2 | Statis | 58 |
| 5. | Selular | 2 | Statis | 121 |
| 6. | *Wi-fi* dan selular | 2 | Statis | 196 |
| 7. | *Wi-fi* | 3 | Statis | 68 |
| 8. | Selular | 3 | Statis | 182 |
| 9. | *Wi-fi* dan selular | 3 | Statis | 94 |
| 10. | *Wi-fi* | 1 | Dinamis | 86 |
| 11. | Selular | 1 | Dinamis | 168 |
| 12. | *Wi-fi* dan selular | 1 | Dinamis | 72 |
| 13. | *Wi-fi* | 2 | Dinamis | 83 |
| 14. | Selular | 2 | Dinamis | 132 |
| 15. | *Wi-fi* dan selular | 2 | Dinamis | 73 |
| 16. | *Wi-fi* | 3 | Dinamis | 60 |
| 17. | Selular | 3 | Dinamis | 117 |
| 18. | *Wi-fi* dan selular | 3 | Dinamis | 81 |




**Gambar 4.24 Diagram labirin statis**




**Gambar 4.25 Diagram labirin dinamis**

Pada tabel 4.4 adalah hasil pengujian dari beberapa kasus berdasarkan parameter jaringan, jumlah NPC dan jenis labirin. Beberapa hal yang dapat diambil dari hasil pengujian tersebut antara lain :

*Latency* cenderung lebih kecil pada kasus yang menggunakan jaringan wi-fi daripada selular atau gabungan selular dan wi-fi.

*Latency* dipengaruhi oleh jenis labirin, labirin dengan tipe statis cenderung memiliki *latency* lebih kecil, pada saat menggunakan jaringan dan jumlah NPC yang sama, pada jaringan selular dan wi-fi kecuali gabungan.

Karena hasil pengujian pada kasus jaringan gabungan nampak tidak beraturan, untuk mengetahui alasan tersebut dilakukan analisa kepada beberapa data sample dari kasus pengujian jaringan gabungan dan diperoleh hasil sebagai mana terlihat pada tabel 4.5.

**Tabel 4.5 Data *ping* pada jaringan gabungan**

| No. | NPC | Jenis Labirin | Rata-Rata Ping Player | | Latency /ms |
|---|---|---|---|---|---|
| | | | Pengirim | Penerima | |
| 1. | 1 | Statis | 207 | 47 | 212 |
| 2. | 2 | Statis | 303 | 53 | 196 |
| 3. | 3 | Statis | 105 | 53 | 94 |
| 4. | 1 | Dinamis | 78 | 46 | 72 |
| 5. | 2 | Dinamis | 85 | 47 | 73 |
| 6. | 3 | Dinamis | 83 | 47 | 81 |

Dari tabel 4.5 diketahui penyebab besarnya *latency* pengiriman data adalah koneksi internet yang cenderung lambat, koneksi lambat dapat terjadi pada seluruh atau sebagian perangkat yang terhubung. Untuk menjalankan *game online* rekomendasi untuk ping adalah di bawah 100 ms.

Jumlah NPC tidak begitu mempengaruhi besar kecilnya *latency*, pada saat menggunakan jenis labirin dan jaringan yang sama.

*Latency* yang tinggi juga dapat terjadi ketika perangkat melakukan *load balancing*, yaitu ketika sebuah perangkat menggunakan dua jalur internet sekaligus (wifi dan selular) yang aktif secara bersamaan. Ketika kedua jaringan tersebut aktif maka :

1. *Android* akan *test ping* jika aplikasi perlu koneksi dengan prioritas tinggi, atau aplikasi yang berjalan pada *foreground*.
2. *Test ping* melalui jaringan selular dan wifi, kemudian keduanya akan diperoleh hasil dan *ping* ter cepat akan digunakan.
3. Jika kedua jaringan memiliki *ping* yang hampir sama atau ada yang tidak stabil maka *android* akan sering melakukan *test ping* untuk menentukan yang

terbaik. Pada *point* ini jumlah ping menjadi lebih banyak, mengakibatkan peningkatan *latency*.