BAB 5
IMPLEMENTASI

Pada bab ini akan dijelaskan tentang bagaimana mengimplementasi *game* ke dalam bentuk digital di lingkungan perangkat keras dan lunak menggunakan *game engine* Unity. Pengimplementasian *game* akan disesuaikan dengan hasil perancangan pada bab sebelumnya.

5.1 Spesifikasi Sistem

Spesifikasi sistem perangkat keras dan perangkat lunak dalam *game* ini akan dideskripsikan di sini.

1. Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan dalam proses pengembangan aplikasi game ini akan dijelaskan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras

No	Perangkat Keras	
1	*Processor*	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz (8 CPUs), ~2.5GHz
2	*Memory(RAM)*	4096MB RAM
3	*Harddisk*	1 TB
4	*Graphic Card*	NVIDIA GeForce 840M

2. Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam proses pengembangan aplikasi game ini akan dijelaskan pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak

No	Perangkat Lunak	
1	*Operating System*	Windows 8.1 Pro 64-bit (6.3, Build 9600)
2	*DirectX version*	DirectX 11
3	*Programming Languange*	C#
4	*Software Development Kit*	Game Engine Unity ver.5.3.3

5.2 Implementasi *Game*

Pada tahap implementasi *game* ini akan dibagi menjadi tiga bagian yaitu implementasi kode *program*, implementasi *art* dan UI dan juga terakhir yaitu implementasi dan simulasi *gameplay*.

5.2.1 Implementasi Kode Program

Pada sub bab ini kan dijelaskan mengenai implementasi code dari game "*Slime Collector"*. Kode yang akan dicantumkan oleh penulis adalah berupa yang paling utama dan yang paling penting dalam pembuatan *game*.

1. Implementasi Kode Program Karakter Pemain

Implementasi Kode Program Karakter Pemain akan menjelaskan proses apa saja yang dapat dilakukan karakter *player* pada *gameplay* di *game* "*Slime Collector"* ini.

- **Implementasi *Source Code* pada Player Character**

Tabel 5.3 *Source Code* Player

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	
4	`public class Player : MonoBehaviour {`
5	
6	`    public float speed = 10f;`
7	`    public Vector2 maxVelocity = new Vector2(3, 5);`
8	`    public bool standing;`
9	`    public float flySpeed = 15f;`
10	`    public float airSpeedMultiplier = .3f;`
11	
12	`    private PlayerController controller;`
13	`    private Animator animator;`
14	
15	`    void Start() {`
16	`        controller = GetComponent<PlayerController>();`
17	`        animator = GetComponent<Animator>();`
18	`    }`
19	
20	`    // Update is called once per frame`
21	`    void Update () {`
22	`        var forceX = 0f;`
23	`        var forceY = 0f;`
24	
25	`        var absVelX =`
26	`Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x);`
27	

```
28          var absVelY =
29  Mathf.Abs(GetComponent<Rigidbody2D>().velocity.y);
30
31          if (absVelY < .2f){
32              standing = true;
33          } else {
34              standing = false;
35          }
36
37          if(controller.moving.x != 0) {
38              if (absVelX < maxVelocity.x) {
39                  forceX = standing ? speed *
40  controller.moving.x : (speed * controller.moving.x *
41  airSpeedMultiplier);
42                  transform.localScale = new
43  Vector3(forceX > 0 ? 1 : -1, 1, 1);
44                  animator.SetInteger("AnimationState",
45  1);
46              } else {
47                  animator.SetInteger("AnimationState",
48  0);
49              }
50          }
51
52          if (controller.moving.y > 0) {
53              if (absVelY < maxVelocity.y)
54                  forceY = flySpeed *
55  controller.moving.y;
56          }
57
58          GetComponent<Rigidbody2D>().AddForce(new
59  Vector2(forceX, forceY));
60      }
61  }
62
63
```

Penjelasan baris source code dari table di atas:

1. 15-18: Inisialisasi awal, mengambil data dari PlayerController dan Animator.
2. 22-23: menentukan nilai forceX/Y awal.
3. 25-29: menentukan nilai absVelX dan Y yang diambil dari *velocity* RigidBody2D.
4. 31-35: menentukan apakah *Player* berdiri atau tidak dari nilai absVelY.

5. 37-49: menentukan kecepatan bergerak horizontal Player saat berdiri dan saat terbang dan *Animation State*.
6. 52-60: menentukan kecepatan terbang vertikal Player.

- **Implementasi *Source Code* Player Controller**

Tabel 5.4 *Source Code* Player Controller

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	
4	`public class PlayerController : MonoBehaviour {`
5	
6	`    public Vector2 moving = new Vector2();`
7	`    public float threshold = 0f;`
8	
9	`     // Use this for initialization`
10	`     void Start () {`
11	
12	`     }`
13	
14	`    // Update is called once per frame`
15	`    void Update() {`
16	`        moving.x = moving.y = 0;`
17	
18	`        if (Input.GetAxis("Horizontal") > threshold) {`
19	`            moving.x = 1;`
20	`        } else if (Input.GetAxis("Horizontal") < -`
21	`threshold) {`
22	`            moving.x = -1;`
23	`        }`
24	
25	`        if (Input.GetButton("Fire1")) {`
26	`            moving.y = 1;`
27	`        } else if (Input.GetKey("down")) {`
28	`            moving.y = -1;`
29	`        }`
30	
31	`    }`
32	`}`

Penjelasan baris *source code* dari table di atas:

1. 16: menentukan nilai moving.x dan moving.y awal.
2. 18-23: menentukan nilai moving.x jika Input “Horizontal” tersebut positif atau negatif yang berarti menentukan *Player* begerak ke kiri atau kanan.
3. 25-29: menentukan nilai moving.y jika Input “Fire1” tersebut positif atau negatif yang berarti menentukan *Player* terbang atau tidak.

- **Implementasi *Source Code* Dead pada *Player***

Tabel 5.5 *Source Code* Dead

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	`using UnityEngine.SceneManagement;`
4	
5	`public class Dead : MonoBehaviour {`
6	
7	`    public ExplodingPart explodingPart;`
8	`    public int totalParts = 5;`
9	
10	`     // Use this for initialization`
11	`     void Start () {`
12	
13	`     }`
14	
15	`     // Update is called once per frame`
16	`     void Update () {`
17	
18	`     }`
19	
20	`    void OnTriggerEnter2D(Collider2D target) {`
21	`        if(target.gameObject.tag == "Deadly") {`
22	`            OnDead();`
23	`        }`
24	`    }`
25	
26	`    void OnCollisionEnter2D(Collision2D target) {`
27	`        if (target.gameObject.tag == "Deadly") {`
28	`            OnDead();`
29	`        }`
30	`    }`

```
31      void OnDead() {
32          Destroy(gameObject);
33
34          var t = transform;
35
36          for(int i=0; i<totalParts; i++) {
37              t.TransformPoint(0, -100, 0);
38              ExplodingPart clone =
39  Instantiate(explodingPart, t.position,
40  Quaternion.identity) as ExplodingPart;
41
42  clone.GetComponent<Rigidbody2D>().AddForce(Vector3.righ
43  t * Random.Range(-50, 50));
44
45  clone.GetComponent<Rigidbody2D>().AddForce(Vector3.up *
46  Random.Range(100, 400));
47          }
48
49          GameObject go = new GameObject("Click to
50  Continue");
51          ClickToContinue script =
52  go.AddComponent<ClickToContinue>();
53          script.scene =
54  SceneManager.GetActiveScene().name;
55          go.AddComponent<RestartText>();
56      }
    }
```

Penjelasan baris source code dari table di atas:

1. 19-29: menjalankan fungsi onDead() jika Player *trigger* atau *collision* dengan *deadly-tagged game object*
2. 32: *destroy gameObject Player*
3. 36-46: menampilkan explodingPart berkeping-keping dengan cloning dan menambahkan *force* kepada keeping tersebut jika *game object Player* destroyed.
4. 48-54: menampilkan tampilan *Click to Continue* dan mengulang *scene* tersebut ketika tombol apapun ditekan.

2. Implementasi Kode Program Karakter Enemy

Implementasi Kode Program Karakter *Enemy* akan menjelaskan proses apa saja yang dilakukan pada karakter *Enemy* pada *gameplay* di *game* "*Slime Collector*" ini.

- **Implementasi *Source Code* MoveForward pada Enemy**

Tabel 5.6 *Source Code* MoveForward

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	
4	`public class MoveForward : MonoBehaviour {`
5	
6	`    public float speed = .5f;`
7	
8	`      // Use this for initialization`
9	`      void Start () {`
10	
11	`      }`
12	
13	`      // Update is called once per frame`
14	`      void Update () {`
15	`         GetComponent<Rigidbody2D>().velocity = new`
16	`Vector2(transform.localScale.x, 0) * speed;`
17	`      }`
18	`}`
19	

Penjelasan baris source code dari table di atas:

1. 14-17: menggerakkan maju enemy dalam *velocity* yang didpat dari dari *transform* dikali *speed* yang terlah ditentukan.

- **Implementasi *Source Code* LookForward**

Tabel 5.7 *Source Code* LookForward

```
No
 1  using UnityEngine;
 2  using System.Collections;
 3
 4  public class LookForward : MonoBehaviour {
 5
 6      public Transform sightStart, sightEnd;
 7      public bool needsCollision = true;
 8
 9      private bool collision = false;
10
11        // Use this for initialization
12        void Start () {
13
14        }
15
16        // Update is called once per frame
17        void Update () {
18           collision =
19  Physics2D.Linecast(sightStart.position,
20  sightEnd.position, 1 <<
21  LayerMask.NameToLayer("Solid"));
22           Debug.DrawLine(sightStart.position,
23  sightEnd.position, Color.green);
24
25           if (collision == needsCollision)
26               this.transform.localScale = new
27  Vector3((transform.localScale.x == 1) ? -1 : 1, 1, 1);
28        }
29  }
```

Penjelasan baris source code dari table di atas:

1. 18-23: menggunakan *Linecast Physic2D* untuk melihat beberapa jarak ke depan musuh sampai dengan *collision* ke *layer Solid* dan juga menggambarkan *line sight*-nya
2. 25-27: membalikkan arah jalan musuh ketika *collide* dengan *layer Solid*

3. **Implementasi Kode Program *Crystal Collectable***

Implementasi Kode Program Karakter Enemy akan menjelaskan proses apa saja yang dilakukan pada karakter *enemy* pada *gameplay* di game "*Slime Collector*" ini.

- **Implementasi *Source Code* CrystalCollectable**

Tabel 5.8 *Source Code* CrystalCollectable

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	
4	`public class CrystalCollectable : MonoBehaviour {`
5	
6	`    private SpawnCrystal controller;`
7	`      // Use this for initialization`
8	`      void Start () {`
9	`         GameObject gameControllerObject =`
10	`GameObject.FindWithTag("Spawn");`
11	`         if(gameControllerObject != null)`
12	`         {`
13	`             controller =`
14	`gameControllerObject.GetComponent<SpawnCrystal>();`
15	`         }`
16	
17	`      }`
18	
19	`      // Update is called once per frame`
20	`      void Update () {`
21	
22	`      }`
23	
24	`    void OnTriggerEnter2D(Collider2D target) {`
25	`         if (target.gameObject.tag == "Player")`
26	`         {`
27	
28	`             controller.addScore(10);`
29	`             Destroy(gameObject);`
30	`         }`
31	`    }`
32	`}`

Penjelasan baris *source code* dari table di atas:

1. 8-15: inisialisasi gameControllerObject dari *object* yang ber-*tag* “Spawn”
2. 24-31: menambahkan skor 10 jika player *collide* dengan *crystal* dan menghancurkan *game object crystal* tersebut

- **Implementasi *Source Code* SpawnCrystal**

Tabel 5.9 *Source Code* SpawnCrystal

No	
1	`using UnityEngine;`
2	`using System.Collections;`
3	`using UnityEngine.UI;`
4	`using System.Collections.Generic;`
5	
6	`public class SpawnCrystal : MonoBehaviour {`
7	
8	`    public GameObject crystal;`
9	`    public float spawnTime = 10f;`
10	`    public Transform[] spawnPoint;`
11	`    public int score;`
12	`    public Text scoreText;`
13	
14	`    // Use this for initialization`
15	`    void Start () {`
16	`        score = 0;`
17	`        updateScore();`
18	`        InvokeRepeating("Spawn", spawnTime,`
19	`spawnTime);`
20	
21	`    }`
22	
23	`     // Update is called once per frame`
24	`     void Update () {`
25	
26	`     }`
27	
28	`    void Spawn()`
29	`    {`
30	`        for (int i = 0; i < spawnPoint.Length; i++)`
31	`        {`
32	`            if (spawnPoint[i].childCount == 0)`
33	`            {`
34	
35	

```
36                 GameObject stopcrystal =
37 (GameObject)Instantiate(crystal,
38 spawnPoint[i].position, spawnPoint[i].rotation);
39                 stopcrystal.transform.parent =
40 spawnPoint[i];
41             }
42         }
43     }
44
45     public void addScore(int newScore)
46     {
47         score += newScore;
48         updateScore();
49     }
50
51     void updateScore()
52     {
53         scoreText.text = score.ToString();
54     }
55 }
```

Penjelasan baris *source code* dari table di atas:

1. 15-21: inisialisasi *score* awal, pemanggilan fungsi updateScore dan InvokeRepeating fungsi Spawn
2. 28-43: memeriksa apakah ada *gameobject crystal* di spawnPoint dan dimunculkan jika tidak ada.
3. 45-49: mengupdate *score*
4. 51-54: mengupdate *text* nilai *score*

5.2.2 Implementasi *Art* dan UI

Pada sub-bab implementasi *Art* ini terdiri dari beberapa bagian yaitu implementasi *artwork* yang digunakan untuk *tilemap*, karakter *Player*, karakter *Enemy*, *collectable*, dan *display text*.

1. Implementasi *Art Tilemap*

Implementasi art *tilemap* ditunjukkan pada gambar 5.1.

Gambar 5.1 *Tilemap*

2. **Implementasi *Art* Karakter *Player***

Implementasi *art* karakter *player* ditunjukkan pada gambar 5.2

Gambar 5.2 Karakter *Player*

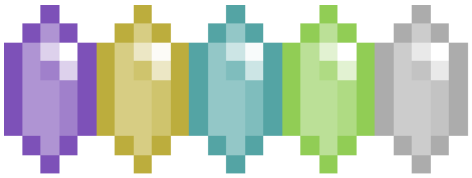
3. **Implementasi *Art* Karakter *Enemy***

Implementasi art karakter *enemy* ditunjukkan pada gambar 5.3

Gambar 5.3 Karakter *Enemy*

4. **Implementasi *Art Crystal Collectable***

Implementasi art *crystal collectable* ditunjukkan pada gambar 5.4.



Gambar 5.4 *Crystal Collectable*

5. **Implementasi *Art Display Text***

Implementasi art *display text* ditunjukkan pada gambar 5.5 dan 5.6.

Gambar 5.5 *Display Text* 1

Gambar 5.6 *Display Text* 2

6. **Implementasi *Game Screen***

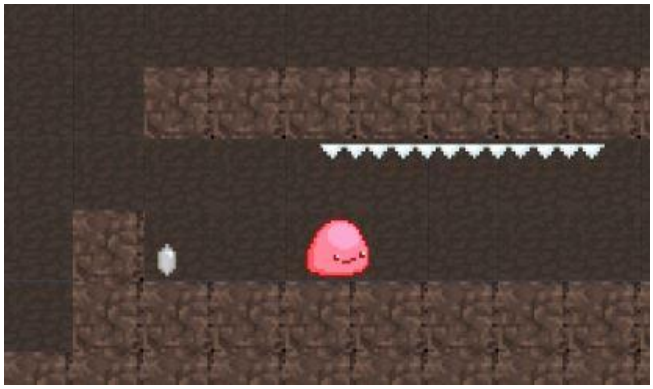
Implementasi art *game screen* ditunjukkan pada gambar 5.7

Gambar 5.7 *Screen Main Menu/Splash Screen*

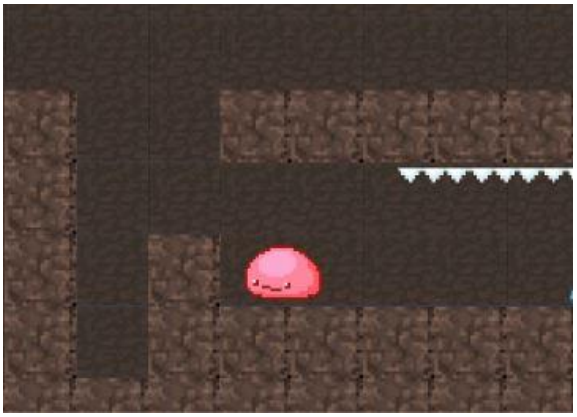
5.2.3 Simulasi *Gameplay*

Simulasi *gameplay* ini berisi hasil implementasi dari perancangan *gameplay* di bab sebelumnya. Pada tahap ini akan ditampilkan dari cara menggerakkan karakter, kondisi karakter menyentuh *hazard* ataupun *enemy*, dan kondisi *player* mendapatkan *crystal collectible*.

1. **Pergerakan Karakter**



Gambar 5.8 Implementasi Karakter Berjalan ke Kanan

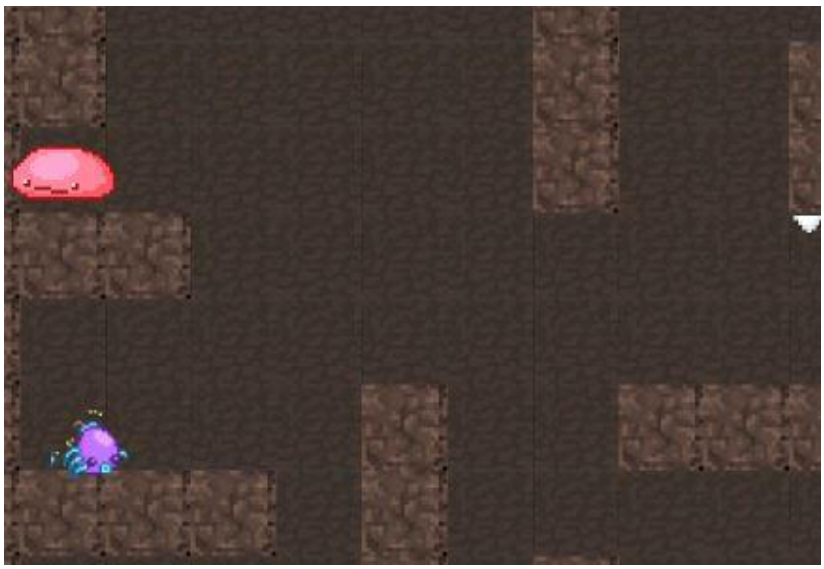


Gambar 5.9 Implementasi Karakter Berjalan ke Kiri

Gambar 5.10 Implementasi Karakter Terbang/Melompat

Pada gambar 5.8, gambar 5.9 dan gambar 5.10 disimulasikan bagaimana karakter *Player* berjalan ke kanan, berjalan ke kiri dan terbang/melompat.

2. Interaksi Dengan Objek di Lingkungan *Game*



Gambar 5.11 Implementasi Karakter Mengambil *Crystal Collectable*

Pada gambar 5.11 diatas disimulasikan kondisi jika karakter menyentuh dan mengambil *crystal collectable*. Pada kondisi tersebut akan tampak bahwa *collectable* tersebut hilang dan *score* bertambah 10.

Gambar 5.12 Implementasi Karakter Menyentuh *Enemy* atapun Menyentuh *Hazards*

Pada gambar 5.12 disimulasikan kondisi jika karakter menyentuh *enemy* ataupun *hazard* berupa *spike*. Pada kondisi tersebut *player* akan hancur berkeping-keping dan muncul *display text "Please Click Anywhere to Try Again*".