BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam Minilemon Studio, terdapat divisi-divisi yang diawasi oleh *project manager*. *Project manager* melakukan tugasnya dengan memberikan arahan untuk proyek-proyek yang dikerjakan. Divisi yang diawasi oleh *project manager* adalah divisi *web developer*, divisi *illustrator*, dan divisi *game developer*.

Divisi *game developer* memiliki campuran anggota dari divisi *illustrator*, dan *game developer*. Divisi ini berisi mahasiswa-mahasiswa yang melaksanakan program magang di dalam minilemon studio yang akan melaksanakan tugas yang diberikan oleh *project manager* dan akan melaporkan *progress* sekali dalam 2 minggu.

3.2 Tugas yang Dilakukan

Selama magang di Minilemon Studio, pekerjaan yang diberikan adalah untuk membuat sebuah *multiplayer video game* yang memiliki fitur komunikasi melalui internet. pekerjaan-pekerjaan yang dilakukan diuraikan dalam Tabel 3.1.

Proses pembangunan gim dibagi menjadi 3 tahap yaitu *pre-production, production,* dan *post-production* [6].

Tahap *pre-production* berupa perencanaan dan pembelajaran. Perencanaan dilakukan dengan menentukan beberapa aspek penting yang akan ada dalam gim contohnya:

- Dimensi karakter dan dunia gim
- Mekanisme pergerakan
- Tujuan akhir
- *Gameloop*
- Jumlah pemain dalam satu sesi permainan

Pembelajaran lebih tertuju kepada *developer* gim tersebut, tahap ini meliputi penelaahan *engine* untuk menemukan kelebihan dan kekurangan *engine* tersebut dan penentuan apakah *engine* layak untuk dipakai saat proses pembangunan. Tahap

production termasuk pembangunan sistem, pembuatan aset serta implementasinya. Pembangunan sistem mencakup semua mekanisme gim mulai dari pergerakan karakter, mekanisme tubrukan, serta mekanisme khusus lainnya yang diperlukan. Pembuatan aset dilakukan oleh *illustrator* dan *audio engineer* untuk membuat elemen grafis dan elemen audio/suara yang akan digunakan dalam gim. Tahap *post-production* berisi *polishing*, *QA testing*, dan *bugfixing*. Tahap yang dilakukan setelah gim terbuat ini berfungsi untuk membuat gim menjadi suatu produk yang baik, tanpa/sedikit *error*, dan layak digunakan bagi publik/pemain.

Tabel 3.1. Tabel tugas yang dilakukan

Minggu ke-	**Kegiatan yang dilakukan**
1	*Briefing* alur kerja di perusahaan perkenalan dalam *server discord* dan pembagian
2	Mempelajari cara kerja *engine playcanvas* dengan membaca dokumentasi dan membuat mekanisme pergerakan karakter
3	Membuat mekanisme pergerakan karakter dalam dua dimensi
4	Mempelajari mekanisme tubrukan dengan melakukan eksperimen pada suatu *scene* baru
5	Membuat mekanisme tubrukan dan interaksi karakter dengan objek lain
6	Membuat mekanisme *spawning* objek selain karakter pemain
7	Membuat *script* mekanisme penyimpanan lokasi objek
8	Mengimplementasikan mekanisme *object permanence* yang memungkinkan objek untuk terus muncul di *runtime* yang berbeda
9	Membuat mekanisme kustomisasi karakter dengan membuat *script* untuk menampilkan *sprite* rambut untuk pemain
10	Melanjutkan pembuatan mekanisme kustomisasi karakter dengan membuat *template/prefab* rambut karakter
11	Melanjutkan pembuatan mekanisme kustomisasi karakter dengan menambahkan *template/prefab* rambut ke dalam *accessorymanager*
12	Membuat *script* untuk menciptakan dan menghapus aksesoris karakter
13	Melanjutkan pembuatan *script* penciptaan dan penghapusan aksesoris karakter
14	Membuat *template/prefab* kustomisasi karakter dalam bentuk aksesoris kepala, badan, dan tangan

Lanjut pada halaman berikutnya

Tabel 3.1 Tabel tugas yang dilakukan(lanjutan)

Minggu ke-	Tugas yang dilakukan
15	Melanjutkan pembuatan *template/prefab* kustomisasi karakter dengan membuat sebuah
16	Menetapkan *template/prefab* yang telah dibuat ke dalam *accessorymanager*
17	Memulai implementasi mekanisme *multiplayer* menggunakan *photon engine* dengan melakukan *setup server* melalui platform photon
18	Melanjutkan pengerjaan mekanisme *multiplayer* dengan mengembangkan sistem sinkronisasi antar pemain
19	Membangun demo dan melakukan pembenahan kode-kode dalam *script* agar lebih mudah untuk dilihat dan dimodifikasi

3.3 Uraian Pelaksanaan Magang

Kegiatan yang paling penting dalam proses pembuatan gim adalah penguasaan alat-alat yang digunakan seperti fitur-fitur *engine* serta *plugin-plugin* yang diperlukan, dan alat komunikasi dan koordinasi antar *programmer*. *Engine* yang telah diputuskan untuk dipakai adalah *engine Playcanvas* yang merupakan sebuah *engine* berbasis *web* dengan bahasa pemrograman *javascript*.

3.3.1 Pergerakan Karakter Pemain

Mekanisme pergerakan karakter yang dibuat adalah pergerakan karakter dalam dua dimensi dengan posisi kamera berada di atas dan menyorot ke arah bawah untuk menciptakan *top-down view*. Pergerakan karakter pemain akan dioperasi kan menggunakan *keyboard* dan akan menggerakkan karakter secara langsung tanpa menggunakan simulasi gaya ataupun inersia. Karakter pemain juga memiliki sebuah komponen yaitu *rigidbody* yang memungkinkan objek-objek untuk saling bertubrukan. Agar objek yang memiliki komponen *rigidbody* dapat bergerak, perlu digunakan gaya dorong sementara mekanisme pergerakan karakter saat ini tidak menggunakan metode gaya dorong untuk menggerakkan karakter. Oleh karena itu, *script* pergerakan karakter saat ini perlu diganti dengan *script* pergerakan karakter yang menggunakan gaya dorong.

```
1 if (canmove == true){
```

```
        if (this.spriteobject != null){
            if (this.app.keyboard.isPressed(pc.KEY_LEFT) || this.app.keyboard.isPressed(pc.KEY_A))
            {
                horidir = -1;
                this.movedir = "L"
                // this.entity.rigidbody.mask = 1
            }
            if (this.app.keyboard.isPressed(pc.KEY_RIGHT) || this.app.keyboard.isPressed(pc.KEY_D))
            {
                horidir = 1;
                this.movedir = "R"
            }
            if (this.app.keyboard.isPressed(pc.KEY_UP) || this.app.keyboard.isPressed(pc.KEY_W)){
                vertdir = 1;
                this.movedir = "U"
            }
            if (this.app.keyboard.isPressed(pc.KEY_DOWN) || this.app.keyboard.isPressed(pc.KEY_S)){
                vertdir = -1;
                this.movedir = "D"

                // this.entity.rigidbody.mask = 0
            }
            if (horidir == 0 && vertdir == 0){

                // this.spriteobject.sprite.play('Idle')
                this.movedir = null
            }
        }
        else{
            this.spriteobject = this.playersprite
        }
```

```
32          facescript.bodyanimation(this.movedir)
33      }
34      forcex = horidir * this.movespeed;
35      forcez = vertdir * this.movespeed;
36      this.force.set(forcex, forcey, forcez);
37      this.entity.rigidbody.linearVelocity = this.force;
38      // this.bodyanimation(this.movedir)
39       const { x, y, z } = this.entity.getPosition();
40          // console.log(x)
41      var dir = this.movedir
42      this.app.fire("loadbalancing:sendPlayerPosition", {
    x, y, z,}, dir);
```

Kode 3.1: Potongan Kode dari Script Movement

Kode 3.1 menampilkan sebuah potongan kode yang digunakan untuk menerima input dari pemain dengan mengubah variabel horidir dan vertdir. Variabel horidir dan vertdir kemudian digunakan untuk mengalkulasi kecepatan dari karakter pemain. Rumus kecepatan pemain menggunakan variabel horidir dan vertdir yang dikalikan dengan variabel movespeed untuk menentukan pergerakan horizontal dan vertikal dari karakter pemain. Karakter pemain akan digerakkan dalam sumbu x dan sumbu z dikarenakan *engine playcanvas* merupakan sebuah *game engine 3D* [7] dan kamera ditetapkan di atas menyorot ke bawah yang menyebabkan perhitungan vertikal menjadi terbalik.

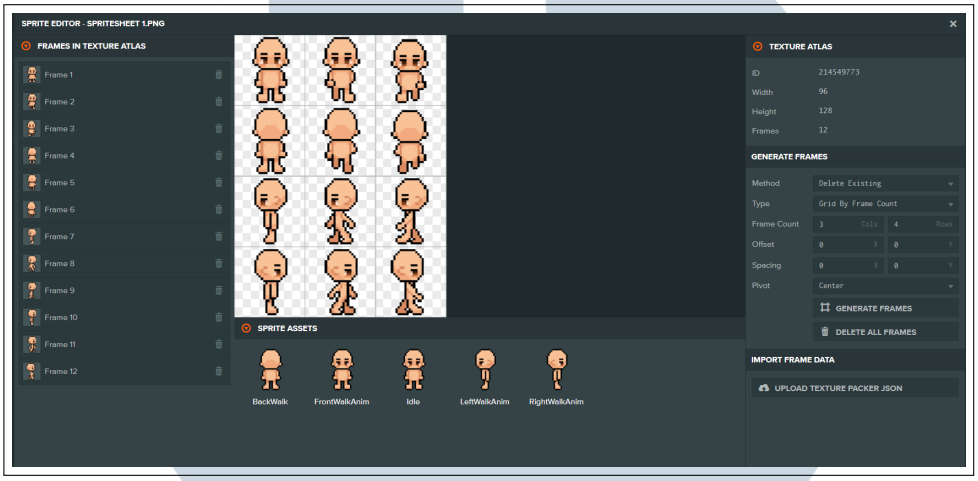
3.3.2 Animasi Karakter Pemain

Karakter yang akan dikendalikan pemain berupa sebuah gambar dua dimensi yang diberi nama *sprite* [8]. *Sprite-sprite* karakter akan disatukan menjadi sebuah *spritesheet* yang akan digunakan untuk membuat animasi pergerakan karakter.

Gambar 3.1. Sprite dari karakter pemain default

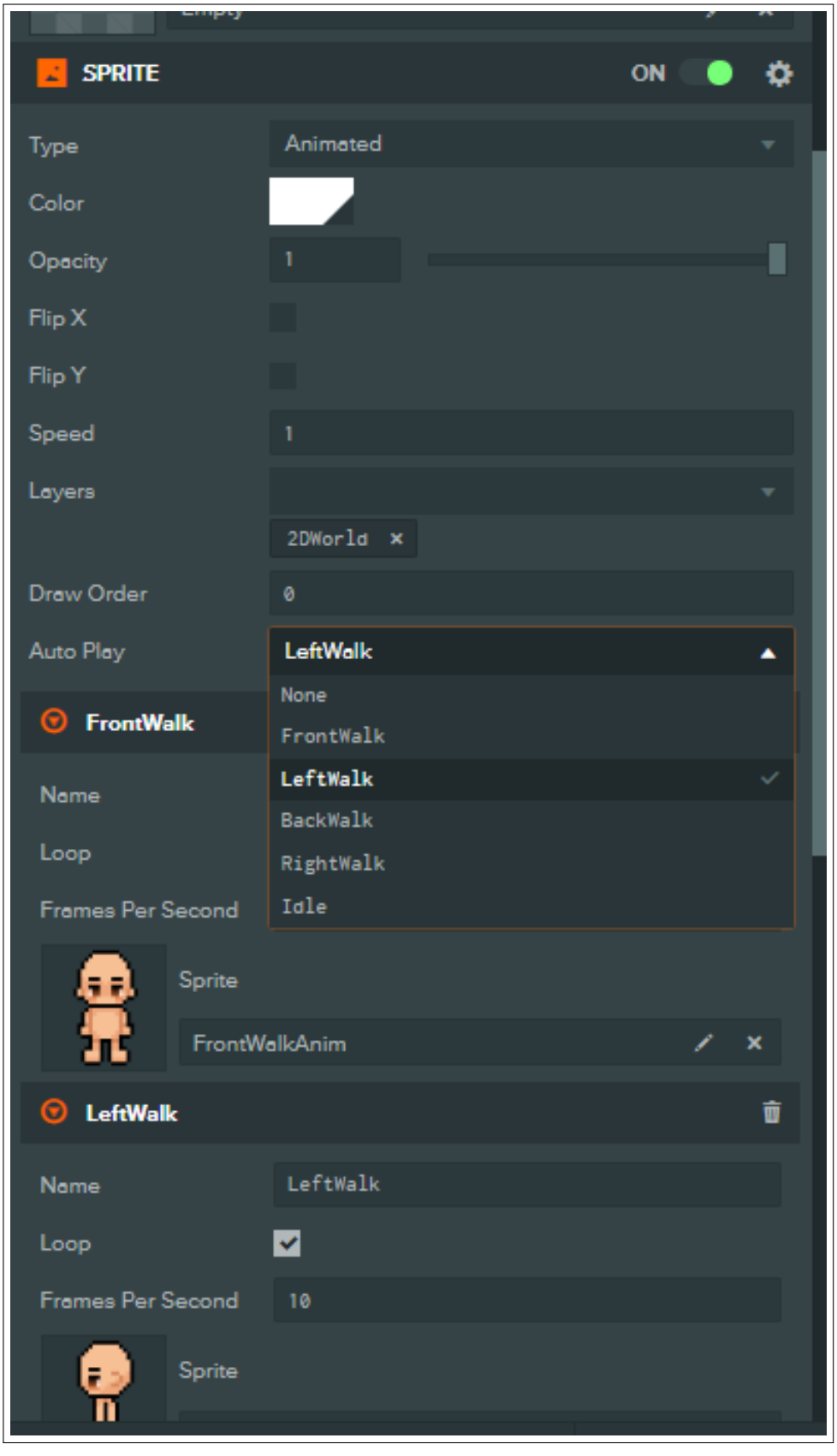
Gambar 3.1 merupakan contoh salah satu *sprite* yang digunakan untuk membuat sebuah karakter yang akan dikendalikan oleh pemain. Seperti yang dapat dilihat dari Gambar 3.1, *sprite* terdiri dari 12 gambar karakter yang sama tetapi dengan sedikit perbedaan setiap baris. Setiap baris *sprite* menunjukkan arah hadapan karakter, sementara setiap kolom *sprite* menunjukkan *frame-frame* animasi dari karakter. Agar *engine* dapat membagi *sprite* menjadi kolom dan baris, *sprite*

perlu diubah menjadi sebuah *spritesheet*. *Engine Playcanvas* menggunakan fitur *texture atlas* yang akan membuat sebuah gambar yang berisi *sprite-sprite* untuk dijadikan animasi karakter. Contoh dari *texture atlas* dapat dilihat pada Gambar 3.2.



Gambar 3.2. Tampilan sprite editor playcanvas

Dalam tampilan yang terlihat pada Gambar 3.2, *Spritesheet* sudah terbagi menjadi sembilan buah *sprite* yang dapat dikelompokkan lalu ditampilkan secara berurutan sesuai dengan urutan pemilihan. *Sprite* akan dikelompokkan per baris yang menggambarkan karakter menghadap ke arah depan/bawah, belakang/atas, kanan, dan kiri. Selanjutnya untuk mengendalikan animasi yang perlu ditunjukkan, akan digunakan sistem *clips* dalam fitur *animated sprite*.



Gambar 3.3. Tampilan *clip* yang sudah dibagi-bagi di playcanvas

Gambar 3.3 menunjukkan animasi-animasi yang sudah dimasukkan kedalam *clip* dan akan lebih mudah dikendalikan menggunakan *script* pergerakan. Pengendalian animasi menggunakan fungsi di dalam *script movement* yang dapat dilihat pada Kode 3.2

```
Movement.prototype.bodyanimation = function(animdir)
{
    switch(animdir)
    {
        case "U":
        this.spriteobject.sprite.resume()
        this.spriteobject.sprite.play("BackWalk")
        break;
        case "D":
        this.spriteobject.sprite.resume()
        this.spriteobject.sprite.play("FrontWalk")
        break;
        case "L":
        this.spriteobject.sprite.resume()
        this.spriteobject.sprite.play("LeftWalk")
        break;
        case "R":
        this.spriteobject.sprite.resume()
        this.spriteobject.sprite.play("RightWalk")
        break;
        default:
        this.spriteobject.sprite.stop()
    }
}
```

Kode 3.2: Fungsi yang digunakan untuk mengendalikan animasi karakter

Cara kerja fungsi *bodyanimation* adalah dengan membaca variabel animdir dari argumen fungsi yang telah diisi dengan *value* dari movedir. Variabel movedir berisi sebuah data bertipe *string* yang berganti-ganti saat pemain melakukan input. Pada Kode 3.1 terlihat nilai dari variabel movedir berisi huruf U, D, R, atau L yang merupakan arah pergerakan karakter saat pemain menekan tombol W, S, A, atau D.

3.3.3 Objek Dalam Gim

Selain karakter pemain, di dalam gim juga terdapat sebuah objek yang tidak dapat dikendalikan oleh pemain. Objek tersebut memiliki *sprite* berupa kursi, meja, televisi, serta furnitur-furnitur lainnya. Objek-objek memiliki dua cara untuk dimunculkan, cara pertama adalah melalui editor *playcanvas* dengan meletakkan secara langsung *sprite* yang diinginkan. cara kedua adalah melalui *scripting* yang akan membuat fungsi untuk mengambil data *sprite* dari editor lalu akan dimunculkan ketika fungsi dipanggil. Jika hanya *sprite* yang dimunculkan, *sprite* tersebut tidak akan memiliki fungsi lain selain menampilkan gambar oleh karena itu, perlu dibuat sebuah *prefab/template* agar objek dapat berfungsi dengan baik.

Prefab/template adalah sebuah objek yang telah dimodifikasi melalui *engine* [9]. Modifikasi tersebut berupa penambahan elemen seperti *script*, perubahan ukuran atau warna, serta penambahan anak objek. Saat *prefab/template* diciptakan, modifikasi-modifikasi yang dilakukan di editor akan tersimpan dan akan memudahkan proses pembuatan mekanisme penciptaan objek. *Prefab* untuk objek furnitur berisi:

- elemen *sprite* furnitur,
- elemen *rigidbody*,
- elemen *script*,
- objek anak yang memiliki elemen *collider*

Potongan *script* yang terdapat dalam *prefab* objek furnitur adalah Kode 3.3

```
1 ObjectScript.prototype.onMouseMove = function(event){
2     if (this.objectstatescript.currentstate ==
    objstatus['Mfollow']){
3     var depth = 0;
4     var cameraEntity = this.app.root.findByName('Camera
    ');
5     cameraEntity.camera.screenToWorld(event.x, event.y,
     depth, this.pos);
6
7    if (this.entity.rigidbody.enabled == false || this.
    entity.rigidbody == null){
```

```
8       console.log(this.pos)
9       this.entity.setPosition(new pc.Vec3(this.pos.x, 0,
        this.pos.z));
10      console.log("shldmove")
11      }
12      else
13      {
14      this.entity.rigidbody.enabled = false
15      }
16
17      }
18  }
19  ================================================
20  var objstatus = {
21      'static': 'static',
22      'Mfollow':'mousefollow',
23      'Moving':'canmove'
24  }
25  var currentstate
26  // initialize code called once per entity
27  ObjState.prototype.initialize = function() {
28      this.app.on('objstate:change', this.swapobjstate,
        this)
29      this.currentstate = objstatus['static']
30  };
31
32  // update code called every frame
33  ObjState.prototype.update = function(dt) {
34
35  };
36
37  ObjState.prototype.swapobjstate = function(state){
38      this.currentstate = objstatus[state]
39  }
40  %
```

Kode 3.3: Potongan kode yang terdapat pada prefab objek

Kode 3.3 menampilkan potongan *script* yang terdapat pada *prefab* objek furnitur yang berfungsi untuk menempatkan posisi objek ke posisi *cursor* saat pemain menggerakkan mouse. Mekanisme ini bekerja bersama dengan fungsi pada Kode 3.5 untuk membuat mekanisme dekorasi ruangan. Agar sebuah objek furnitur dapat ditempatkan ke suatu posisi sekaligus dapat bergerak mengikuti *cursor* pemain, perlu diberikan suatu syarat yang menentukan fungsi yang ingin digunakan. Syarat yang digunakan berupa sebuah *finite state machine* [10]. *State* dari objek akan diubah saat objek pertama diciptakan pada *runtime* dan saat pergerakan objek furnitur dihentikan untuk menetapkan posisi dari objek tersebut. Kode **??** berisi *state* yang akan dikendalikan melalui *Objectscript*.

3.3.4 Penciptaan Objek dalam Gim

Objek dalam gim akan diciptakan menggunakan kedua cara yaitu melalui editor dan saat *runtime*. Saat *runtime*/gim berjalan, penciptaan objek akan menggunakan sebuah *script* yang dapat dilihat pada Kode 3.4

```
1  Spawntest.prototype.spawningentity = function(prefab) {
2      var instantiateitem = prefab.resource.instantiate()
3
4      this.entity.addChild(instantiateitem)
5
6      if (instantiateitem.script.objState){
7          instantiateitem.script.objState.currentstate =
       objstatus['Mfollow']
8      }
9      console.log(instantiateitem.script.objectScript)
10 }
11
12 %%
```

Kode 3.4: Potongan kode yang menunjukkan fungsi untuk menciptakan objek

Script spawntest menggunakan fitur *events* untuk mendapat sinyal untuk memanggil fungsi untuk menciptakan objek. *Event* tersebut dapat dilihat pada fungsi *initialize* yang akan dijalankan pada awal gim dimulai. *this.app.on* merupakan sebuah fungsi dari *playcanvas* yang memungkinkan sebuah *script* menjadi suatu *listener* yang dapat menangkap suatu sinyal lalu memanggil sebuah fungsi yang telah ditetapkan. Argumen pertama dari fungsi tersebut berisi nama dari sinyal yang

akan ditangkap, untuk fungsi ini sinyal tersebut dinamakan *buttonpress:spawn*. Argumen kedua berisi nama fungsi yang akan dipanggil saat *script* menangkap sinyal *buttonpress:spawn*, fungsi yang dipanggil adalah fungsi *spawningentity*. Argumen ketiga berisi *scope* dari pemanggilan fungsi, *scope* ini berfungsi untuk menentukan isi dari variabel *this*, *scope* yang diinginkan untuk *event* ini adalah *script* spawntest itu sendiri oleh karena itu, argumen tersebut berisi variabel *this*.

Fungsi spawningentity memiliki argumen ”prefab” yang akan berisi prefab/template dari objek yang akan diciptakan saat sinyal dibuat. Isi dari argumen ini akan dipanggil saat sinyal dibuat melalui *script* buttonpress 3.5 yang dihubungkan dengan sebuah tombol.

```
1  ButtonScript.prototype.postInitialize = function(){
2      var me =  this // me is a var to point to current
       script (make sure to put it on top of everything
       else on a func)
3      // in unity, var me == private Currentscript
       currentscript
4      this.entity.button.on('click', function() {
5          if (me.buttonfunc == 1)
6          {
7              me.app.fire('buttonpress:spawn', me.
       itemspawned)
8
9          }
10         else if (me.buttonfunc == 2)
11         {
12             var thisplayer = me.app.root.findByTag('
       player')[0]
13             var thisplayerscript = thisplayer.script.
       playerAccSetup
14             console.log(thisplayerscript)
15             if (thisplayerscript != null)
16             {
17                 thisplayerscript.swaphair(me.whichhair)
18             }
19
20         }
```

```
21          else if (me.buttonfunc == 3)
22          {
23              var thisplayer = me.app.root.findByTag('
    player')[0]
24
25              var thisplayerscript = thisplayer.script.
    playerAccSetup
26              console.log(thisplayer.name)
27              if (thisplayerscript != null)
28              {
29                  // thisplayerscript.swapacctext('head',
     'head2')
30                  var atttostring = String(me.whichacc)
31                  var stringactual = "lmao"
32                  console.log(atttostring)
33                  thisplayerscript.spawnaccessories(
    atttostring)
34              }
35          }
36          else if (me.buttonfunc == 4)
37          {
38              //make spawnbody script in playerAccSetup
39          }
40          // self.spawn(spawneditem)
41          // console.log('buttclick')
42      })
43      // console.log(this.itemspawned)
44  }
45  % %
```

Kode 3.5: Potongan kode yang terdapat pada elemen tombol

Saat spawningentity dipanggil, variabel instantiateitem akan dibuat dengan isi data dari argumen prefab yang telah diciptakan ke dalam gim, lalu variabel instantiateitem dimasukkan ke dalam objek bernama "spawning" yang memiliki *script* spawntest. Agar pemain yang menciptakan objek tersebut dapat menggerakkan serta meletakkan objek ke lokasi yang diinginkan, variabel

currentstate yang terdapat pada *script objstate* di dalam instantiateitem akan diisi dengan nilai *state 'Mfollow'* yang memenuhi syarat fungsi onMouseMove yang mengendalikan pergerakan objek ke posisi *cursor*. Saat objek berada dalam *state 'Mfollow'*, pemain dapat melakukan input tombol mouse3 atau klik *scrollwheel* untuk mengubah *state* dari objek furnitur menjadi *state 'static'* yang tidak memenuhi syarat fungsi onMouseMove dan mengakibatkan posisi objek furnitur tidak lagi mengikuti *cursor*.

3.3.5 Penyimpanan Objek yang Diciptakan saat Runtime

Objek yang diciptakan saat *runtime* sebuah gim hanya akan bertahan sampai *runtime* selesai atau gim ditutup. Agar objek furnitur yang diciptakan tetap ada di tempat yang ditentukan, akan digunakan sebuah *script* yang dapat menyimpan lokasi dan penampilan objek yang telah ditentukan saat *runtime*. *Script* tersebut dapat dilihat pada Kode 3.6.

```
1  ObjectScript.prototype.saveItem = function(itemID,
       template)
2  {
3      var self = this
4      if (this.saved == false) {
5      var stuff = this.entity.name
6      this.mypos = this.pos
7      console.log(this.mypos)
8      objarray.length = 0
9      objarray.push({templatename: stuff,posx: this.mypos
       .x, posy: 0, posz: this.mypos.z})
10     this.stringifysave("myobj", objarray)
11     console.log(objarray)
12     this.saved = true
13     }
14 }
15 %%
```

Kode 3.6: Potongan kode fungsi untuk menyimpan data objek furnitur

Kode 3.6 menampilkan potongan *script* yang berfungsi untuk menyimpan lokasi dari objek saat objek tersebut dipasang di dalam *scene* gim. Fungsi saveitem ini

menggunakan fitur localstorage yang dimiliki oleh browser untuk menyimpan data-data dari objek yang sudah ditentukan. Untuk dapat menyimpan data yang dapat dibaca oleh *script* untuk menciptakan objek, data tersebut perlu diubah menjadi tipe data *JSON string*, hal ini akan dilakukan menggunakan fungsi *script* yang terlihat pada Kode 3.9.

```
1  ObjectScript.prototype.stringifysave = function(
       savedname, savedval)
2  {
3
4      var self = this
5      var saveddata = this.getsave(savedname)
6      var currentsave = self.getsave(savedname)
7      console.log(saveddata)
8      if (saveddata == null){
9          localStorage.setItem(savedname, JSON.stringify(
       savedval))
10     }
11     else
12     {
13         saveddata.push(savedval[0])
14         console.log(saveddata)
15         localStorage.setItem(savedname, JSON.stringify(
       saveddata))
16     }
17
18 }
19
20 ObjectScript.prototype.getsave = function(savedname) {
21     return JSON.parse(localStorage.getItem(savedname))
22 }
23
24 ObjectScript.prototype.spawntemplate = function() {
25     var self = this
26      // run a foreach at the start or "load"
27         objarray.forEach(function spawnitem(arrayobj) {
28             console.log(arrayobj)
```

```
29                  var assetneeded = typecheck(arrayobj.
    templatename)
30                  var assetpos = new pc.Vec3(arrayobj.
    posx,arrayobj.posy,arrayobj.posz)
31                  //spawn item here
32                  console.log(assetneeded,assetpos)
33                  var newspawn = assetneeded.resource.
    instantiate()
34                  self.app.root.addChild(newspawn)
35                  newspawn.setPosition(assetpos.x,
    assetpos.y,assetpos.z)
36              })
37 }
38
39 function globalgetsave(savedname) {
40     return JSON.parse(localStorage.getItem(savedname))
41 }
42 %%
```

Kode 3.7: Potongan kode fungsi untuk mengubah data menjadi JSON string

Kode 3.9 berisi potongan *script* yang digunakan untuk mengubah data-data dari objek furnitur yang akan disimpan dalam bentuk *JSON string*. Proses ini dilakukan dengan menggunakan fungsi *JSON stringify* yang akan mengambil sebuah nilai *javascript* sebagai argumen.

3.3.6 Interaksi Karakter Pemain dengan Objek furnitur

Objek-objek furnitur yang diciptakan tidak hanya berfungsi untuk menambah elemen visual, tetapi juga dapat menjadi elemen interaktif bagi para pemain. Interaksi antara karakter pemain dan objek dapat dilakukan menggunakan elemen *collision* dan *collider* yang diletakkan pada karakter pemain. *Collider* tersebut memungkinkan karakter pemain untuk mengetahui objek yang akan dilakukan interaksi. Setelah objek diketahui, pemain dapat melakukan input untuk memanggil fungsi yang akan melakukan interaksi tersebut seperti yang dapat dilihat pada Kode 3.8

```
1 if (this.app.keyboard.wasPressed(pc.KEY_E)){
```

```
            if(canInteract == true){
                console.log('presse')
                if(playerscript.statescript.state != myenum['canWalk'])
                {
                this.app.fire('player:swapstate', 'canWalk')
                // console.log(this.entity.position)
                this.locktoitem(null)
                this.queuetonorm = true
                }
                else{
                this.app.fire('player:swapstate', 'sitDown')
                // console.log(this.entity.position)
                this.locktoitem(touchingobject)
                }
            }

}
================================================
Movement.prototype.locktoitem = function(object) {
    var self = this
    if (object != null){


    self.entity.rigidbody.teleport(object.position);
    var entspot = self.entity.getLocalPosition()
    self.entity.rigidbody.teleport(entspot.x, 0, entspot.z)
    // console.log(object.position)
    self.entity.rigidbody.linearVelocity = pc.Vec3.ZERO;
    self.entity.rigidbody.angularVelocity = pc.Vec3.ZERO;
    this.playersprite.drawOrder = 1;
```

```
    }
    else{
        var entspot = self.entity.getLocalPosition()
        self.entity.rigidbody.teleport(entspot.x, 0,
    entspot.z)
        console.log(self.entity.position)
        this.playersprite.drawOrder = -1;
    }
}
%%
```

Kode 3.8: Potongan kode yang berisi fungsi-fungsi untuk mekanisme interaksi

Kode 3.8 berisi potongan *script* yang digunakan untuk menerima input yang memanggil fungsi *locktoitem*. Fungsi *locktoitem* akan mendeteksi karakter pemain terutama elemen *rigidbody*nya lalu memindahkan posisi karakter pemain ke posisi objek furnitur yang bertubrukan dengan *collider* pemain. *Script* yang melakukan deteksi dapat dilihat di Kode 3.10.

```
%%
```

Kode 3.9: Potongan kode fungsi untuk mengubah data menjadi JSON string

```

    this.app.on('Objek:touching', this.onTouchObject,
    this)
    this.app.on('Objek:leaving', this.onLeavingObject,
    this)

Movement.prototype.onTouchObject = function(obj) {
    //obj should be entity
    if (obj.script.objCollide){
        canInteract = true
        touchingobject = obj.parent
        console.log(touchingobject)
    }
}

Movement.prototype.onLeavingObject = function(obj) {
    var self = this
```

```
    if (obj.script.objCollide){
        canInteract = false
        touchingobject = null
        if (self.queuetonorm == true)
        {
            var mypos = self.entity.getLocalPosition()
            self.entity.rigidbody.teleport(mypos.x, 0,
    mypos.z)
            self.queuetonorm = false
        }
    }
}
%%
```

Kode 3.10: Potongan kode fungsi untuk mengubah data menjadi JSON string

Kode 3.10 adalah potongan *script* pada karakter pemain yang akan melakukan pemenuhan syarat-syarat untuk menjalankan mekanisme interaksi. Fungsi *onTouchObject* dan *onLeavingObject* akan dipanggil sesuai dengan saat sinyal Objek:touching dan Objek:leaving dikirimkan. Sumber dari sinyal tersebut adalah dari objek yang mendeteksi karakter pemain yang berada di area deteksi objek.

```
ObjCollide.prototype.onTriggerEnter = function(entidy)
    {
    if (entidy.script.movement){
        console.log(String(entidy.name) + ' entering');
        this.app.fire('Objek:touching', this.entity)
    }
};

ObjCollide.prototype.onTriggerLeave = function(entidy){

    if (entidy.script.movement){
        console.log(String(entidy.name) + ' leaving');
        this.app.fire('Objek:leaving', this.entity)
    }
};
```

```
15 % %
```

Kode 3.11: Potongan kode yang digunakan untuk melakukan deteksi pemain yang bertubrukan

Kode 3.11 adalah sebuah fungsi yang berisi pemicu pemenuhan syarat mekanisme interaksi antara karakter pemain dan objek furnitur agar mekanisme dapat dilakukan dengan baik. Syarat yang dilengkapi sebuah variabel boolean dan objek yang sedang bertubrukan dengan karakter. Agar *script* tersebut dapat mengetahui objek yang ingin di-interaksi kan, objek tersebut akan mengirimkan sebuah sinyal dengan argumen yang berisi objek itu sendiri saat *collider* objek mendeteksi sebuah *collider* yang memiliki *script* bernama *Movement*.

3.3.7 Implementasi Komponen Multiplayer

Komponen *multiplayer* dalam gim memungkinkan lebih dari satu pemain untuk berada dalam *scene* yang sama serta dapat melihat karakter pemain lain. Komponen *multiplayer* yang digunakan adalah *photon engine*. *Photon engine* adalah sebuah *framework multiplayer* yang memungkinkan pengembang gim untuk mengintegrasikan multiplayer ke dalam gim yang sudah dibuat.

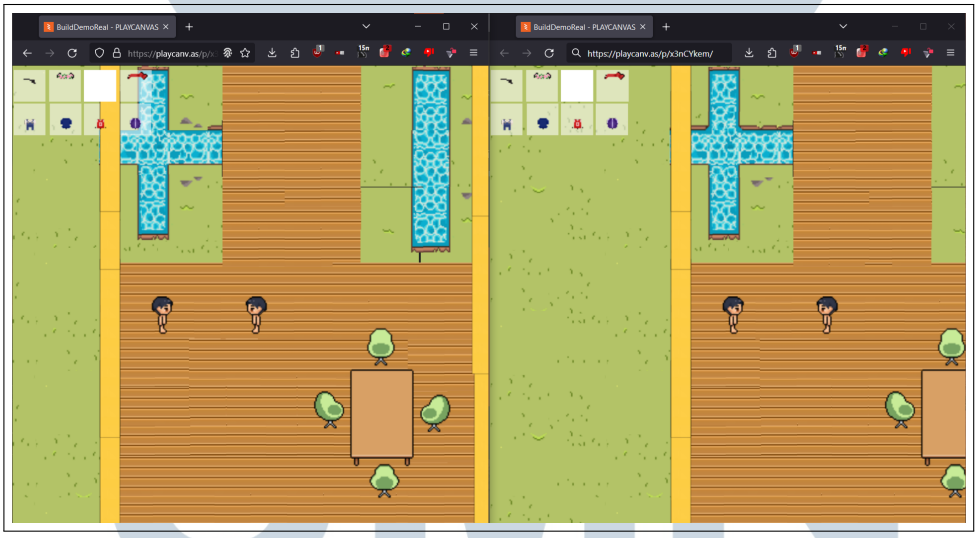
```
1  this.app.on("createOtherPlayerEntity", this.
     createOtherPlayerEntity, this);
2      this.app.on("loadbalancing:sendPlayerPosition",
     this.sendPlayerPosition, this);
3  PhotonLoadBalancingPlayCanvas.prototype.
     createOtherPlayerEntity = function (actor) {
4      const { actorNr } = actor;
5      const playerfab = this.app.assets.get(233323161)
6      const entity = playerfab.resource.instantiate()
7      // call char creation here
8      // entity.addComponent("render", { type: "capsule"
     });
9      // entity.addComponent("script")
10     entity.setLocalPosition(0, 0, 0);
11     entity.name = actorNr;
12     this.app.root.children[0].addChild(entity);
13 };
14
```

```
PhotonLoadBalancingPlayCanvas.prototype.sendPlayerPosition = function (position,direction) {
    this.loadBalancingClient.raiseEvent(1, { position });
    this.loadBalancingClient.raiseEvent(2, direction)
};

PhotonLoadBalancingPlayCanvas.prototype.onEvent = function (code, content, actorNr) {
    switch (code) {
        case 1: {
            const otherPlayer = this.app.root.findByName(actorNr);
            if (otherPlayer) {
                const { x, y, z } = content.position;
                otherPlayer.setLocalPosition(x, y, z);
            }
            break;
        }
        case 2:
        {
            const otherPlayer = this.app.root.findByName(actorNr);
            if(otherPlayer.script.playerFacing)
            {
                console.log(content)
                otherPlayer.script.playerFacing.bodyanimation(content)
            }
        }
        default:
    }


    ==========================================
    Movement.prototype.update = function(dt) {
```

```
45      this.app.fire("loadbalancing:sendPlayerPosition", {
        x, y, z,}, dir);
46      }
47  };
```

Kode 3.12: Potongan kode komponen multiplayer menggunakan photon

Kode 3.12 adalah potongan kode dari *script multiplayer photon* yang mengendalikan sinkronisasi posisi karakter pemain. Fungsi onEvent akan dipanggil jika sinyal loadBalancing:sendplayerposition diterima, sinyal tersebut akan dikirimkan dari *script movement* yang dimiliki oleh karakter pemain. Interval dari sinyal yang diberikan bergantung kepada kekuatan dari komputer atau browser dari pemain karena fungsi *update* di playcanvas bergantung kepada *framerequest* dari browser. Jika gim menghasilkan enam puluh *frame per second*, sinyal akan dikirimkan setiap 0.16 detik.



Gambar 3.4. Tampilan dari demo project yang sudah memiliki komponen multiplayer

Gambar 3.4 menampilkan demo dari proyek playcanvas yang sudah dapat menampilkan pemain lain. Agar pemain lain tidak dapat mengontrol pemain, sudah disiapkan dua *prefab/template* yang berbeda untuk karakter pemain dan karakter selain pemain. *Prefab/template* karakter pemain dan karakter selain pemain hanya memiliki satu perbedaan, yaitu *script* yang tercantum. *Prefab/template* Karakter milik pemain utama memiliki *script movement* sementara karakter selain pemain tidak memiliki *script* tersebut.

Sinkronisasi Animasi dilakukan bersamaan dengan sinkronisasi pergerakan. Sinyal loadBalancing:sendplayerposition yang mengirimkan argumen posisi

karakter memiliki argumen kedua yaitu variabel dir yang digunakan oleh Kode 3.2 untuk menentukan animasi yang akan dimainkan.

3.4 Kendala dan Solusi yang Ditemukan

Selama pelaksanaan magang, Kendala-kendala yang ditemukan dalam pelaksanaan pekerjaan adalah:

- Kurangnya pencatatan *progress* yang dilakukan
- Kurangnya koordinasi antara tim developer yang memperlambat *progress*

Dari kendala tersebut, solusi-solusi yang ditemukan adalah:

- Menggunakan *queue* sistem agar koordinasi dapat menjadi lebih mudah
- Memperbanyak komunikasi antar developer dan *project manager* agar rencana yang dibuat dapat dimengerti oleh semua anggota