BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Tahap implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari kegiatan perancangan aplikasi, tahap ini merupakan tahap dimana aplikasi siap untuk dioperasikan. Aplikasi ini diimplementasikan dengan nama lutung kasarung *the jumper* dan *file extension*-nya *.apk* (*application package file*).

4.1.1 Implementasi Perangkat Keras

Perangkat keras yang dapat digunakan untuk menjalankan aplikasi *game* ini tidak harus memiliki spesifikasi *device* yang tinggi, seperti *processor* 1.4Ghz dan *memory* 1GB yang dimiliki oleh *Samsung galaxy III*. Sebagai ilustrasi sistem minimum agar dapat menjalankan *game* lutung kasarung *the jumper* adalah sebagai berikut :

1. *Processor* dengan kecepatan ~ *800Mhz*.
2. *RAM* dengan kapasitas 128 *Mb*.
3. Memiliki sensor *accelerometer*

4.1.2 Implementasi Perangkat Lunak

Adapun spesifikasi perangkat lunak yang dibutuhkan untuk menjalankan aplikasi lutung kasarung *the jumper* adalah Sistem *Operasi Android* versi 2.2 (*froyo*) atau diatasnya.

4.1.3 Implementasi Instalasi Aplikasi *Game*

Sebelum dapat memainkan *game* lutung kasarung *the jumper*, aplikasi harus terinstal di *device android*. Petunjuk dalam instalasi aplikasi *game* ini adalah sebagai berikut :

1. *Copy file* lutung kasarung *the jumper.apk* pada *memory card device*.
2. Buka *android market* dan cari untuk aplikasi *App Installer*, *z-App Installer* atau *Fast Installer*.
3. Buka dan klik pada tombol *Install*.
4. Setelah terinstal, buka program tersebut. Program tersebut akan menunjukan semua program *APK* yang ada pada *memory card.*
5. Kemudian cari *file* lutung kasarung *the jumper.apk*, buka dan klik pada tombol *install*.

4.1.4 Implementasi Antarmuka

Antarmuka dapat membuat pengguna merasa nyaman dan bertahan lama berada dalam aplikasi atau sebaliknya. Berikut ini merupakan antarmuka yang ada pada aplikasi *game* lutung kasarung *the jumper* :

1. Antarmuka Menu Utama

Saat pengguna membuka aplikasi pertama kali, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka menu utama. Berikut ini merupakan tampilan dari antarmuka menu utama :

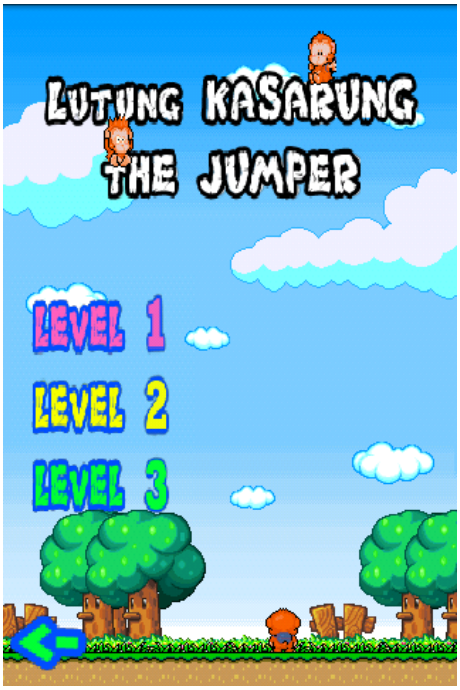


Gambar 4.1 Antarmuka Menu Utama

Gambar di atas merupakan menu utama pada *game* lutung kasarung *the jumper*. Terdapat 3 (tiga) sub menu di dalamnya, yaitu menu *play* yang berfungsi untuk memulai permainan, menu *highscores* yang berfungsi untuk melihat pencapaian 5 (lima) skor tertinggi, menu *help* yang berfungsi untuk melihat petunjuk bermain, dan *volume on/off* yang berada di sudut kiri bawah.

2. Antarmuka Menu *Play*

Saat pengguna memilih menu *play*, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka yang didalamnya terdapat menu *level*. Berikut ini merupakan tampilan dari antarmuka menu *play* :



Gambar 4.2 Antarmuka Menu Play

Gambar di atas merupakan menu *play* yang terdapat pada *game* lutung kasarung *the jumper*. Terdapat 3 (tiga) sub menu di dalamnya, yaitu *level* 1, *level* 2, dan *level* 3.

3. Antarmuka Menu *Highscores*

Saat pengguna memilih menu *highscores*, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka *highscores* dengan informasi pencapaian 5 (lima) skor tertinggi. Berikut ini merupakan tampilan dari antarmuka menu *highscores* :



Gambar 4.3 Antarmuka Menu Highscores

4. Antarmuka Menu *Help*

Saat pengguna memilih menu *help*, maka antarmuka yang ditampilkan oleh sistem adalah informasi mengenai petunjuk-petunjuk cara bermain dalam *game* lutung kasarung *the jumper*. Berikut ini merupakan tampilan dari antarmuka menu *help* :



Gambar 4.4 Antarmuka Menu Help

4.2 Pengujian

Pada umunya ada 2 (dua) macam teknik pengujian yaitu *white box testing* dan *black box testing*. Pengujian yang menggunakan teknik *white box testing* lebih menitikberatkan pada struktur internal (*source code*) program, di mana setiap baris kode yang akan di *compile* dicek satu persatu untuk mengetahui apakah masih terjadi *error*. Pengujian pada *black box testing* tidak terlalu memperhatikan struktur internal (*source code*) dari program, tetapi lebih kepada menemukan kesalahan program. Cara pengujian dilakukan melalui dua tahap, yaitu pengujian *alpha* (fungsional) dan pengujian *beta*.

4.2.1 Pengujian *Alpha*

Pengujian dilakukan terhadap aplikasi untuk memastikan bahwa aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Pengujian *alpha* berfokus pada persyaratan fungsional perangkat lunak.

4.2.1.1 Skenario Pengujian Aplikasi

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada aplikasi *game* lutung kasarung *the jumper*. Skenario pengujian yang akan dilakukan pada aplikasi ini selengkapnya dapat dilihat pada table 4.1 :

Tabel 4.1 Skenario Pengujian Aplikasi

No	Komponen yang di uji	Skenario Pengujian	Jenis Pengujian
1.	Menu Utama	Sentuh menu *play*	*Black box*
		Sentuh menu *highscores*	*Black box*
		Sentuh menu *help*	*Black box*
		Sentuh *Sound*	*Black box*
2.	Menu *Play*	Sentuh *level* 1	*Black box*
		Permainan *level* 1	*White box*
		Sentuh *level* 2	*Black box*
		Permainan *level* 2	*White box*
		Sentuh *level* 3	*Black box*
		Permainan *level* 3	*White box*
		Sentuh tombol kembali	*Black box*
3.	Menu *Highscores*	Sentuh Menu *Highscores*	*Black box*
		Sentuh tombol kembali	*Black box*
4.	Menu *Help*	Sentuh Menu *Help*	*Black box*
		Sentuh tombol kembali	*Black box*
5.	*Sound*	Sentuh sound (gambar berupa terompet menyala)	*Black box*
		Sentuh sound (gambar berupa terompet redup)	*Black box*
6.	*Accelerometer*	*Device* diam	*Black box*
		Device cenderung miring ke kiri	*Black box*

		Device cenderung miring ke kanan	*Black box*

4.2.1.2 Kasus dan Hasil Pengujian (*Black Box Testing*)

Berikut ini merupakan kasus untuk menguji perangkat lunak yang sudah dibangun dengan menggunakan metode *black box testing* berdasarkan skenario pengujian aplikasi yang terdapat pada tabel 4.1 yang sudah dijabarkan. Hasil pengujian yang akan dilakukan pada aplikasi ini selengkapnya dapat dilihat pada tabel 4.2 :

Tabel 4.2 Pengujian Black Box

No	Kasus yang akan di uji	Skenario dan Hasil Uji		
		Skenario Uji	**Hasil yang Diharapkan**	**Kesimpulan**
1.	Menu Utama	Sentuh Menu *Play*	Sistem menampilkan halaman menu *Level*	[√] Berhasil [ ] Tidak Berhasil
		Sentuh Menu *Highscores*	Sistem menampilkan halaman menu *highscores*	[√] Berhasil [ ] Tidak Berhasil
		Sentuh Menu *Help*	Sistem menampilkan halaman menu *help*	[√] Berhasil [ ] Tidak Berhasil
		Sentuh *Sound*	Sistem melakukan *event* menghidupkan atau mematikan	[√] Berhasil [ ] Tidak Berhasil

			suara	
2.	Menu *Play*	Sentuh Menu *Play*	Sistem menampilkan halaman menu *Level*	[√] Berhasil [ ] Tidak Berhasil
		Sentuh *Level* 1	Sistem menampilkan dunia *level* 1	[√] Berhasil [ ] Tidak Berhasil
		Permainan *Level* 1	Dapat dilihat pada sub bab 4.2.1.3 Kasus dan Hasil Pengujian (*White box Testing*)	
		Sentuh *Level* 2	Sistem menampilkan dunia *level* 2	[√] Berhasil [ ] Tidak Berhasil
		Permainan *Level* 2	Dapat dilihat pada sub bab 4.2.1.3 Kasus dan Hasil Pengujian (*White box Testing*)	
		Sentuh *Level* 3	Sistem menampilkan dunia *level* 3	[√] Berhasil [ ] Tidak Berhasil
		Permainan *Level* 3	Dapat dilihat pada sub bab 4.2.1.3 Kasus dan Hasil Pengujian (*White box Testing*)	
		Sentuh tombol kembali	Sistem menampilkan menu utama	[√] Berhasil [ ] Tidak Berhasil
3.	Menu *Highscores*	Sentuh Menu *Highscores*	Sistem menampilkan 5 (lima) skor tertinggi	[√] Berhasil [ ] Tidak Berhasil
		Sentuh back	Sistem	[√] Berhasil

			menampilkan menu utama	[ ] Tidak Berhasil
4.	Menu *Help*	Sentuh Menu *Help*	Sistem menampilkan informasi cara bermain	[√] Berhasil [ ] Tidak Berhasil
		Sentuh back	Sistem menampilkan menu utama	[√] Berhasil [ ] Tidak Berhasil
5.	*Sound*	Sentuh *Sound* (gambar berupa terompet yang menyala)	Sistem mematikan suara dan gambar terompet menjadi redup	[√] Berhasil [ ] Tidak Berhasil
		Sentuh *Sound* (gambar berupa terompet yang redup)	Sistem mnghasilkan suara dan gambar terompet menjadi terang	[√] Berhasil [ ] Tidak Berhasil
6.	*Accelerometer*	*Device* diam	System melakukan *event* karakter menjadi tidak bergerak ke kiri ataupun ke kanan	[√] Berhasil [ ] Tidak Berhasil
		Device cenderung miring ke kiri	System melakukan *event* karakter bergerak ke kiri	[√] Berhasil [ ] Tidak Berhasil
		Device cenderung	System melakukan *event*	[√] Berhasil [ ] Tidak Berhasil

		miring ke kanan	karakter bergerak ke kanan	

4.2.1.3 Kasus dan Hasil Pengujian (*White box Testing*)

Berikut ini merupakan kasus menu *play* untuk menguji permainan *level* 1, permainan *level* 2, dan permainan *level* 3 di dalam perangkat lunak yang sudah dibangun dengan menggunakan metode *white box testing* berdasarkan skenario pengujian aplikasi yang terdapat pada tabel 4.1 yang sudah dijabarkan.

1. Pengujian Permainan *level* 1 (*white box*)

   Menguji apakah metode *procedural content generation* yang diterapkan untuk *level* 1 sudah berjalan sesuai dengan yang diharapkan. Berikut ini merupakan pengujian metode *procedural content generation* yang di terapkan pada permainan *level* 1 dengan menggunakan metode *white box*.

   a. Algoritma Yang Akan Di Uji

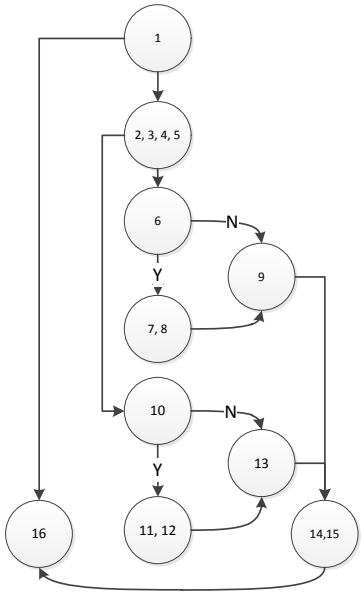
   Berikut ini merupakan algoritma dari metode *procedural content generation* untuk *level* 1 yang penerapannya menggunakan bahasa pemrograman *java* :

```
1.  While ( y < WORLD_HIGHT – WORLD_WIDTH / 2) {
2.  Type ← rand.nextFloat() > 0,5 ?
    Platform(PLATFORM_TYPE_MOVING) : Platform
    (PLATFORM_TYPE_STATIC)
3.  X ← rand.nextFloat() * (WORLD_WIDTH –
    Platform(PLATFORM_WIDTH)) + Platform(PLATFORM_WIDTH) /
    2
4.  Platform(platform) ← new Platform(type, x, y)
5.  Platform.add(platform)
6.  if (rand.nextFloat() > 0,7 && type !=
    Platform(PLATFORM_TYPE_MOVING)) {
7.  Spring(spring) ← new Spring(platform.position.x,
    platform.position.y + Platform(PLATFORM_HEIGHT) / 3 +
    Spring(SPRING_HEIGHT) / 2)
8.  springs.add(spring)
9.  }
```

```
10.  if (rand.nextFloat() > 0,6) {
11.  Poin(poin) ← new Poin(platform.position.x +
     rand.nextFloat(), platform.position.y +
     Poin(POIN_HEIGHT) + rand.nextFloat() * 3)
12.  poins.add(poin)
13.  }
14.  y+ ← (maxJumpHeight – 0,5f)
15.  y- ← rand.nextFloat() * (maxJumpHeight / 3)
16.  } pintu level1 ← new Pintu Level1(WORLD_WIDTH / 2, y)
```

b. *Flow Graph* Permainan *Level* 1

Berikut ini merupakan *flow graph* permainan *level* 1 yang dapat dilihat pada gambar 4.5 :



Gambar 4.5 *Flow Graph* Permainan *Level* 1

Keterangan :

= Menggambarkan Kondisi

= Menggambarkan Aksi

c. *Cyclomatic Complexity* V(G)

E = 13, N = 10

V(G) = E – N + 2

V(G) = 13 – 10 + 2

V(G) = 3 + 2 = 5

Keterangan :

E = Jumlah aksi

N = Jumlah Kondisi

d. *Independent Path*

Path 1 = (1) – (2,3,4,5) – (6) – (7,8) – (9) – (14,15)

Path 2 = (1) – (2,3,4,5) – (6) – (9) – (14,15)

Path 3 = (1) – (2,3,4,5) – (10) – (11,12) – (13) – (14,15)

Path 4 = (1) – (2,3,4,5) – (10) – (13) – (14,15)

Path 5 = (1) – (16)

e. *Graph Matriks* Permainan *Level* 1

Tabel 4.3 *Graph Matriks* Permainan *Level* 1

	(1)	(2,3,4,5)	(6)	(7,8)	(9)	(10)	(11,12)	(13)	(14,15)	(16)
(1)		1								1
(2,3,4,5)			1			1				
(6)				1	1					
(7,8)					1					
(9)									1	
(10)							1	1		
(11,12)								1		
(13)									1	
(14,15)										
(16)										

Keterangan :

1. Baris dan kolom merepresentasikan simpul
2. Nilai 1 merepresentasikan adanya keterhubungan antar simpul

f. Hasil Uji Permainan *Level* 1

Tabel 4.4 Hasil Uji Permainan *Level* 1

Kasus dan Hasil Uji				
No. *Path*	**Butir uji permainan *level* 1**	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
1	Memunculkan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 6,(*true*) 7,8,(*void*) 9,(*void*) 14,15,(*void*)	Menampilkan *spring* diantara *platform* dengan kemunculan acak	*Path* dikerjakan
2	Menyembunyikan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 6,(*false*) 9,(*void*) 14,15,(*void*)	*Spring* tidak ditampilkan	*Path* dikerjakan
3	Memunculkan poin	1,(*void*) 2,3,4,5,(*void*) 10,(*true*) 11,12,(*void*) 13,(*void*) 14,15,(*void*)	Menampilkan poin diantara *platform* dengan kemunculan acak	*Path* dikerjakan
4	Menyembunyikan poin	1,(*void*) 2,3,4,5,(*void*) 10,(*false*) 13,(*void*) 14,15,(*void*)	Poin tidak ditampilkan	*Path* dikerjakan
5	Menampilkan tinggi maksimum dunia *game*	1,(*void*) 16,(*void*)	Menampilkan pintu *level* 2	*Path* dikerjakan

g. Kesimpulan

Dari hasil pengujian permainan *level* 1 yang dilakukan, maka didapat kesimpulan bahwa seluruh *node* yang ada pada setiap *independent path* telah dikerjakan, serta tidak terjadi pengulangan tak berhingga.

2. Pengujian Permainan *level* 2 (*white box*)

   Menguji apakah *procedural content generation* yang diterapkan untuk *level* 2 sudah berjalan sesuai dengan yang diharapkan. Berikut ini merupakan pengujian *procedural content generation* yang di terapkan pada *level* 2 dengan menggunakan metode *white box testing*.

   a. Algoritma Yang Akan Di Uji

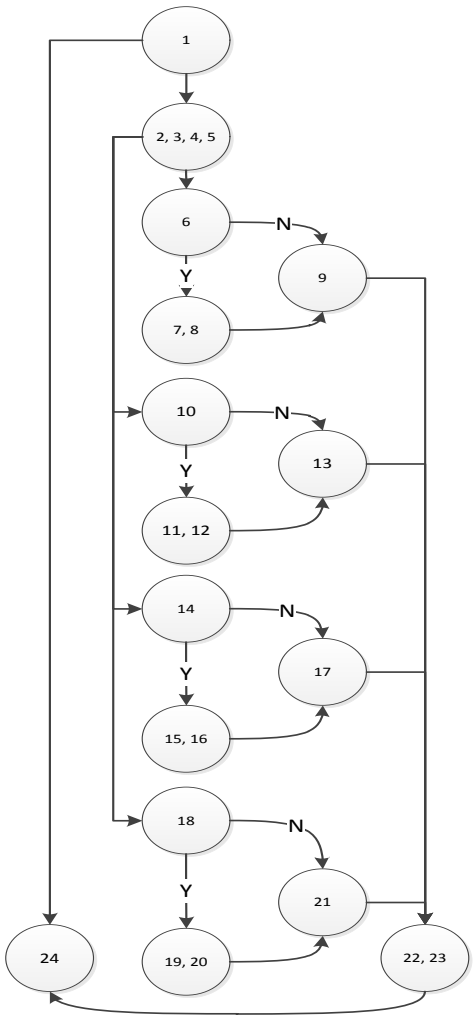
   Berikut ini merupakan algoritma dari metode *procedural content generation* untuk *level* 2 yang penerapannya menggunakan bahasa pemrograman *java* :

```
1. While ( y < WORLD_HIGHT – WORLD_WIDTH / 2) {
2. Type ← rand.nextFloat() > 0,5 ?
   Platform(PLATFORM_TYPE_MOVING) : Platform
   (PLATFORM_TYPE_STATIC)
3. X ← rand.nextFloat() * (WORLD_WIDTH –
   Platform(PLATFORM_WIDTH)) +
   Platform(PLATFORM_WIDTH) / 2
4. Platform(platform) ← new Platform(type, x, y)
5. Platform.add(platform)
6. If ( y > WORLD_HEIGHT / 5 && rand.nextFloat() > 0,8
   && type != Platform(PLATFORM_TYPE_MOVING)) {
7. Bungaracun(bungaracun) ← new Bungaracun
   (Platform.position.x, Platform.position.y +
   Platform(PLATFORM_HEIGHT) / 2 +
   Bungaracun(BUNGARACUN_HEIGHT) / 2)
8. bungaracun.add(bungaracun)
9. }
10. if (rand.nextFloat() > 0,7 && type !=
   Platform(PLATFORM_TYPE_MOVING)) {
11. Spring(spring) ← new Spring(platform.position.x,
   platform.position.y + Platform(PLATFORM_HEIGHT) / 3
   + Spring(SPRING_HEIGHT) / 2)
12. springs.add(spring)
13. }
14. if (y > WORLD_HEIGHT / 2 && rand.nextFloat() > 0,9)
   {
15. Enemy(enemyfly) ← new Enemy(platform.position.x +
   rand.nextFloat(), platform.position.y +
   Enemy(ENEMY_HEIGHT) + rand.nextFloat() * 2)
16. enemys_fly.add(enemyfly);
17. }
```

```
18. if (rand.nextFloat() > 0,6) {
19. Poin(poin) ← new Poin(platform.position.x +
    rand.nextFloat(), platform.position.y +
    Poin(POIN_HEIGHT) + rand.nextFloat() * 3)
20. poins.add(poin)
21. }
22. y+ ← (maxJumpHeight – 0,5f)
23. y- ← rand.nextFloat() * (maxJumpHeight / 3)
24. } pintu level2 = new Pintu Level2(WORLD_WIDTH / 2, y)
```

b. *Flow Graph* Permainan *Level* 2

Berikut ini merupakan *flow graph* permainan *level* 2 yang dapat dilihat pada gambar 4.6 :



Gambar 4.6 *Flow Graph* Permainan *Level* 2

Keterangan :

 = Menggambarkan Kondisi

 = Menggambarkan Aksi

c. *Cyclomatic Complexity* V(G)

E = 23, N = 16

V(G) = E – N + 2

V(G) = 23 – 16 + 2

V(G) = 7 + 2 = 9

Keterangan :

E = Jumlah aksi

N = Jumlah Kondisi

d. *Independent Path*

Path 1 = (1) – (2,3,4,5) – (6) – (7,8) – (9) – (22,23)

Path 2 = (1) – (2,3,4,5) – (6) – (9) – (22,23)

Path 3 = (1) – (2,3,4,5) – (10) – (11,12) – (13) – (22,23)

Path 4 = (1) – (2,3,4,5) – (10) – (13) – (22,23)

Path 5 = (1) – (2,3,4,5) – (14) – (15,16) – (17) – (22,23)

Path 6 = (1) – (2,3,4,5) – (14) – (17) – (22,23)

Path 7 = (1) – (2,3,4,5) – (18) – (19,20) – (21) – (22,23)

Path 8 = (1) – (2,3,4,5) – (18) – (21) – (22,23)

Path 9 = (1) – (24)

e. *Graph Matriks* Permainan *Level* 2

Tabel 4.5 *Graph Matriks* Permainan *Level* 2

	(1)	(2,3,4,5)	(6)	(7,8)	(9)	(10)	(11,12)	(13)	(14)	(15,16)	(17)	(18)	(19,20)	(21)	(22,23)	(24)
(1)		1														1
(2,3,4,5)			1			1			1			1			1	
(6)				1	1											
(7,8)					1											
(9)															1	
(10)							1	1								
(11,12)								1								
(13)															1	
(14)										1	1					
(15,16)											1					
(17)															1	
(18)													1	1		
(19,20)														1		
(21)															1	
(22,23)																
(24)																

Keterangan :

1. Baris dan kolom merepresentasikan simpul
2. Nilai 1 merepresentasikan adanya keterhubungan antar simpul

f. Hasil Uji Permainan *Level* 2

Tabel 4.6 Hasil Uji Permainan *Level* 2

Kasus dan Hasil Uji				
No. *Path*	**Butir uji permainan *level* 2**	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
1	Memunculkan bunga beracun	1,(*void*) 2,3,4,5,(*void*) 6,(*true*) 7,8,(*void*) 9,(*void*) 22,23,(*void*)	Menampilkan bunga beracun diantara *platform* dengan kemunculan acak	*Path* dikerjakan
2	Menyembunyikan bunga beracun	1,(*void*) 2,3,4,5,(*void*) 6,(*false*) 9,(*void*) 22,23,(*void*)	Bunga beracun tidak ditampilkan	*Path* dikerjakan
3	Memunculkan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 10,(*true*) 11,12,(*void*)	Menampilkan *spring* diantara *platform* dengan kemunculan	*Path* dikerjakan

		13,(*void*) 22,23,(*void*)	acak	
4	Menyembunyikan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 10,(*false*) 13,(*void*) 22,23,(*void*)	*Spring* tidak ditampilkan	*Path* dikerjakan
5	Memunculkan *enemy fly*	1,(*void*) 2,3,4,5,(*void*) 14,(*true*) 15,16,(*void*) 17,(*void*) 22,23,(*void*)	Menampilkan *enemy fly* diantara *platform* dengan kemunculan acak	*Path* dikerjakan
6	Menyembunyikan *enemy fly*	1,(*void*) 2,3,4,5,(*void*) 14,(*false*) 17,(*void*) 26,27,(*void*)	*Enemy fly* tidak ditampilkan	*Path* dikerjakan
7	Memunculkan poin	1,(*void*) 2,3,4,5,(*void*) 18,(*true*) 19,20,(*void*) 21,(*void*) 22,23,(*void*)	Menampilkan poin diantara *platform* dengan kemunculan acak	*Path* dikerjakan
8	Menyembunyikan poin	1,(*void*) 2,3,4,5,(*void*) 18,(*false*) 21,(*void*) 22,23,(*void*)	Poin tidak ditampilkan	*Path* dikerjakan
9	Menampilkan tinggi maksimum dunia *game*	1,(*void*) 24,(*void*)	Menampilkan pintu *level* 3	*Path* dikerjakan

g. Kesimpulan

Dari hasil pengujian pada permainan *level* 2 yang dilakukan, maka didapat kesimpulan bahwa seluruh *node* yang ada pada setiap *independent path* telah dikerjakan, serta tidak terjadi pengulangan tak berhingga.

3. Pengujian Permainan *level* 3 (*white box*)

Menguji apakah *procedural content generation* yang diterapkan untuk *level* 3 sudah berjalan sesuai dengan yang diharapkan. Berikut ini merupakan pengujian *procedural content generation* yang di terapkan pada *level* 3 dengan menggunakan metode *white box testing*.

a. Algoritma Yang Akan Di Uji

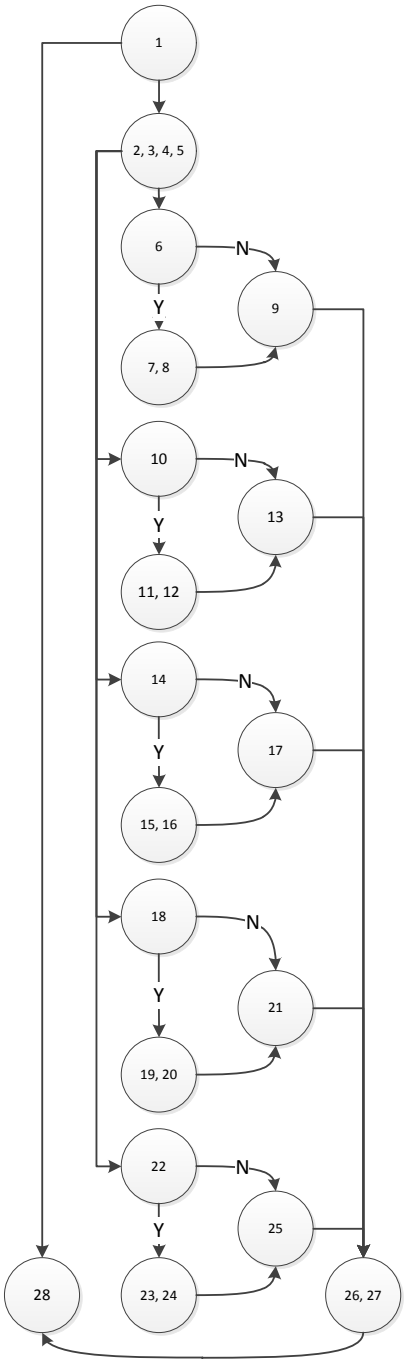
Berikut ini merupakan algoritma dari metode *procedural content generation* untuk *level* 3 yang penerapannya menggunakan bahasa pemrograman *java* :

```
1.  While ( y < WORLD_HIGHT – WORLD_WIDTH / 2) {
2.  Type ← rand.nextFloat() > 0,5 ?
    Platform(PLATFORM_TYPE_MOVING) : Platform
    (PLATFORM_TYPE_STATIC)
3.  X ← rand.nextFloat() * (WORLD_WIDTH –
    Platform(PLATFORM_WIDTH)) +
    Platform(PLATFORM_WIDTH) / 2
4.  Platform(platform) ← new Platform(type, x, y)
5.  Platform.add(platform)
6.  If ( y > WORLD_HEIGHT / 5 && rand.nextFloat() >
    0,8 && type != Platform(PLATFORM_TYPE_MOVING)) {
7.  Bungaracun(bungaracun) ← new Bungaracun
    (Platform.position.x, Platform.position.y +
    Platform(PLATFORM_HEIGHT) / 2 +
    Bungaracun(BUNGARACUN_HEIGHT) / 2)
8.  bungaracun.add(bungaracun)
9.  }
10. if (rand.nextFloat() > 0.7 && type !=
    Platform(PLATFORM_TYPE_MOVING)) {
11. Spring(spring) ← new Spring(platform.position.x,
    platform.position.y + Platform(PLATFORM_HEIGHT) /
    3 + Spring(SPRING_HEIGHT) / 2)
12. springs.add(spring)
13. }
14. if (y > WORLD_HEIGHT / 4 && rand.nextFloat() >
    0.8) {
15. EnemyFall(enemyfall) ← new
    EnemyFall(platform.position.x + rand.nextFloat(),
    platform.position.y + EnemyFall(ENEMY_HEIGHT) +
    rand.nextFloat() * 2)
16. enemys_fall.add(enemyfall)
```

```
17. }
18. if (y > WORLD_HEIGHT / 2 && rand.nextFloat() >
    0.9) {
19. Enemy(enemyfly) ← new Enemy(platform.position.x +
    rand.nextFloat(), platform.position.y +
    Enemy(ENEMY_HEIGHT) + rand.nextFloat() * 2)
20. enemys_fly.add(enemyfly);
21. }
22. if (rand.nextFloat() > 0.6) {
23. Poin(poin) ← new Poin(platform.position.x +
    rand.nextFloat(), platform.position.y +
    Poin(POIN_HEIGHT) + rand.nextFloat() * 3)
24. poins.add(poin)
25. }
26. y+ ← (maxJumpHeight - 0.5f)
27. y- ← rand.nextFloat() * (maxJumpHeight / 3)
28. } purbasari = new Purbasari(WORLD_WIDTH / 2, y)
```

b. *Flow Graph* Permainan *Level* 3

Berikut ini merupakan *flow graph* permainan *level* 3 yang dapat dilihat pada gambar 4.7 :



Gambar 4.7 *Flow graph* Permainan *Level* 3

Keterangan :

= Menggambarkan Kondisi

= Menggambarkan Aksi

c. *Cyclomatic Complexity* V(G)

E = 28, N = 19

V(G) = E – N + 2

V(G) = 28 – 19 + 2

V(G) = 9 + 2 = 11

Keterangan :

E = Jumlah aksi

N = Jumlah Kondisi

d. *Independent Path*

Path 1 = (1) – (2,3,4,5) – (6) – (7,8) – (9) – (26,27)

Path 2 = (1) – (2,3,4,5) – (6) – (9) – (26,27)

Path 3 = (1) – (2,3,4,5) – (10) – (11,12) – (13) – (26,27)

Path 4 = (1) – (2,3,4,5) – (10) – (13) – (26,27)

Path 5 = (1) – (2,3,4,5) – (14) – (15,16) – (17) – (26,27)

Path 6 = (1) – (2,3,4,5) – (14) – (17) – (26,27)

Path 7 = (1) – (2,3,4,5) – (18) – (19,20) – (21) – (26,27)

Path 8 = (1) – (2,3,4,5) – (18) – (21) – (26,27)

Path 9 = (1) – (2,3,4,5) – (22) – (23,24) – (25) – (26,27)

Path 10 = (1) – (2,3,4,5) – (22) – (25) – (26,27)

Path 11 = (1) – (28)

e. *Graph Matriks* Permainan *Level* 3

Tabel 4.7 *Graph Matriks* Permainan *Level* 3

	(1)	(2,3,4,5)	(6)	(7,8)	(9)	(10)	(11,12)	(13)	(14)	(15,16)	(17)	(18)	(19,20)	(21)	(22)	(23,24)	(25)	(26,27)	(28)
(1)		1																	1
(2,3,4,5)			1			1			1			1			1				
(6)				1	1														
(7,8)					1														
(9)																		1	
(10)							1	1											
(11,12)								1											
(13)																		1	
(14)										1	1								
(15,16)											1								
(17)																		1	
(18)													1	1					
(19,20)														1					
(21)																		1	
(22)																1	1		
(23,24)																	1		
(25)																		1	
(26,27)																			
(28)																			

Keterangan :

1. Baris dan kolom merepresentasikan simpul
2. Nilai 1 merepresentasikan adanya keterhubungan antar simpul

f. Hasil Uji Permainan *Level* 3

Tabel 4.8 Hasil Uji Permainan *Level* 3

Kasus dan Hasil Uji				
No. *Path*	**Butir uji permainan *level* 3**	***Node* (n), input kondisi**	**Pengamatan**	**Kesimpulan**
1	Memunculkan bunga beracun	1,(*void*) 2,3,4,5,(*void*) 6,(*true*) 7,8,(*void*) 9,(*void*) 26,27,(*void*)	Menampilkan bunga beracun diantara *platform* dengan kemunculan acak	*Path* dikerjakan
2	Menyembunyikan bunga beracun	1,(*void*) 2,3,4,5,(*void*) 6,(*false*) 9,(*void*) 26,27,(*void*)	Bunga beracun tidak ditampilkan	*Path* dikerjakan
3	Memunculkan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 10,(*true*)	Menampilkan *spring* diantara *platform* dengan	*Path* dikerjakan

		11,12,(*void*) 13,(*void*) 26,27,(*void*)	kemunculan acak	
4	Menyembunyikan *spring* (benda pegas)	1,(*void*) 2,3,4,5,(*void*) 10,(*false*) 13,(*void*) 26,27,(*void*)	*Spring* tidak ditampilkan	*Path* dikerjakan
5	Memunculkan *enemy fall*	1,(*void*) 2,3,4,5,(*void*) 14,(*true*) 15,16,(*void*) 17,(*void*) 26,27,(*void*)	Menampilkan *enemy fall* diantara *platform* dengan kemunculan acak	*Path* dikerjakan
6	Menyembunyikan *enemy fall*	1,(*void*) 2,3,4,5,(*void*) 14,(*false*) 17,(*void*) 26,27,(*void*)	*Enemy fall* tidak ditampilkan	*Path* dikerjakan
7	Memunculkan *enemy fly*	1,(*void*) 2,3,4,5,(*void*) 18,(*true*) 19,20,(*void*) 21,(*void*) 26,27,(*void*)	Menampilkan *enemy fly* diantara *platform* dengan kemunculan acak	*Path* dikerjakan
8	Menyembunyikan *enemy fly*	1,(*void*) 2,3,4,5,(*void*) 18,(*false*) 21,(*void*) 26,27,(*void*)	*Enemy fly* tidak ditampilkan	*Path* dikerjakan
9	Memunculkan poin	1,(*void*) 2,3,4,5,(*void*) 22,(*true*) 23,24,(*void*) 25,(*void*) 26,27,(*void*)	Menampilkan poin diantara *platform* dengan kemunculan acak	*Path* dikerjakan
10	Menyembunyikan poin	1,(*void*) 2,3,4,5,(*void*) 22,(*false*) 25,(*void*) 26,27,(*void*)	Poin tidak ditampilkan	*Path* dikerjakan
11	Menampilkan tinggi maksimum dunia *game*	1,(*void*) 28,(*void*)	Menampilkan purbasari	*Path* dikerjakan

g. Kesimpulan

Dari hasil pengujian pada permainan *level* 3 yang dilakukan, maka didapat kesimpulan bahwa seluruh *node* yang ada pada setiap *independent path* telah dikerjakan, serta tidak terjadi pengulangan tak berhingga.

4.2.1.4 Kesimpulan Pengujian *Alpha*

Dari hasil pengujian yang dilakukan, dapat disimpulkan bahwa sistem sudah dapat berjalan sesuai dengan yang diharapkan beserta proses penanganan kesalahannya. Jadi, secara fungsional sistem sudah dapat menghasilkan keluaran yang diharapkan.

4.2.2 Pengujian *Beta*

Pengujian *beta* merupakan pengujian yang dilakukan secara objektif, dimana dilakukan pengujian secara langsung terhadap pengguna dengan menggunakan kuesioner mengenai kepuasan pengguna atas aplikasi yang telah dibangun. Adapun metode penilaian pengujian yang digunakan adalah metode kuantitatif berdasarkan data dari pengguna.

4.2.2.1 Kuesioner

Kuesioner disebarkan menggunakan teknik *Simple Random Sampling* dimana anggota *sample* dari populasi dilakukan secara acak tanpa memperhatikan strata yang ada dalam populasi. Kuesioner disebarkan kepada 30 orang dari umur 10 tahun hingga 15 tahun, dari hasil kuesioner tersebut akan dilakukan

perhitungan agar dapat diambil kesimpulan terhadap penilaian penerapan sistem yang telah dibangun. Kuesioner terdiri dari 6 pernyataan dengan menggunakan skala 1 sampai 5 (skala likert).

Untuk mencari nilai persentase dari masing-masing jawaban kuesioner digunakan rumus skala likert sebagai berikut :

$$P = \frac{S}{skor\ ideal} \times 100$$

Keterangan :

P = Nilai persentase yang dicari

S = Jumlah frekuensi dikalikan dengan nilai yang di tetapkan jawaban

skorideal = Nilai tertinggi dikalikan dengan jumlah sampel

Berikut ini adalah hasil persentase masing-masing jawaban yang sudah dihitung nilainya dengan menggunakan rumus diatas.

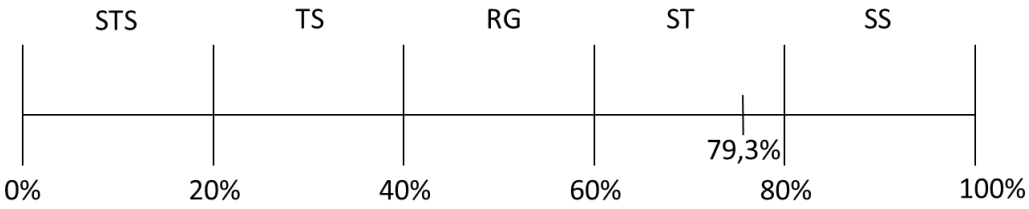
1. Interaksi kontrol karakter pada *game* ini mudah.

Tabel 4.9 Perhitungan persentase pertanyaan ke-1

Pernyataan	Keterangan	Skor	Frekuensi	S
1	Sangat Setuju	5	6	30
	Setuju	4	17	68
	Ragu-ragu	3	7	21
	Tidak Setuju	2	-	-
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**119**

$$P = \frac{119}{150} \times 100\% = 79.3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap kemudahan dalam control karakter yang menggunakan sensor *accelerometer* pada permainan lutung kasarung *the jumper* adalah 79,3% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.8 *Interpretasi* skor pertanyaan ke-1

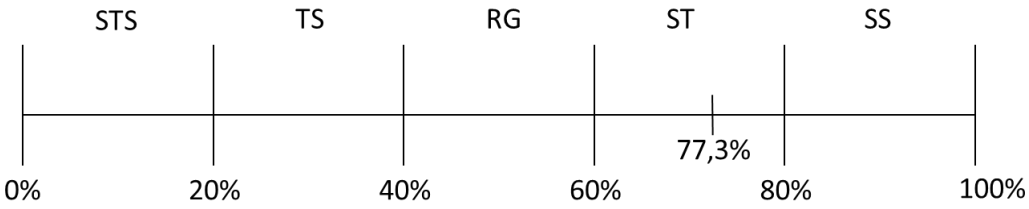
2. Sulit untuk menyelesaikan *game* ini.

Tabel 4.10 Perhitungan persentase pertanyaan ke-2

Pernyataan	Keterangan	Skor	Frekuensi	S
2	Sangat Setuju	5	5	25
	Setuju	4	18	72
	Ragu-ragu	3	5	15
	Tidak Setuju	2	2	4
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**116**

$$P = \frac{116}{150} \times 100\% = 77.3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap kesulitan untuk menyelesaikan permainan pada permainan lutung kasarung *the jumper* adalah 77,3% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.9 *Interpretasi* skor pertanyaan ke-2

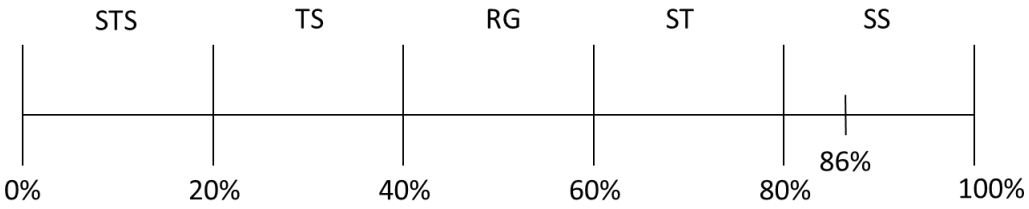
3. *Game* ini dapat membantu anda untuk mengingat kembali tokoh cerita rakyat Jawa Barat yaitu cerita Lutung Kasarung.

Tabel 4.11 Perhitungan persentase pertanyaan ke-3

Pernyataan	Keterangan	Skor	Frekuensi	S
3	Sangat Setuju	5	9	45
	Setuju	4	21	84
	Ragu-ragu	3	-	-
	Tidak Setuju	2	-	-
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**129**

$$P = \frac{129}{150} \times 100\% = 86\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian untuk membantu pengguna dalam mengingat tokoh lutung kasarung adalah 86% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.10 *Interpretasi* skor pertanyaan ke-3

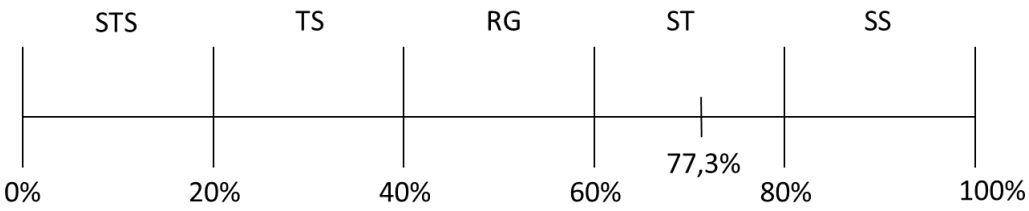
4. *Interface* (antamuka) yang di buat sudah cocok untuk di gunakan pada *handphone*.

Tabel 4.12 Perhitungan persentase pertanyaan ke-4

Pernyataan	Keterangan	Skor	Frekuensi	S
4	Sangat Setuju	5	4	20
	Setuju	4	18	72
	Ragu-ragu	3	8	24
	Tidak Setuju	2	-	-
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**116**

$$P = \frac{116}{150} \times 100\% = 77{,}3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap kecocokan *interface* (antarmuka) yang dibuat untuk di gunakan pada *handphone* adalah 77,3% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.11 *Interpretasi* skor pertanyaan ke-4

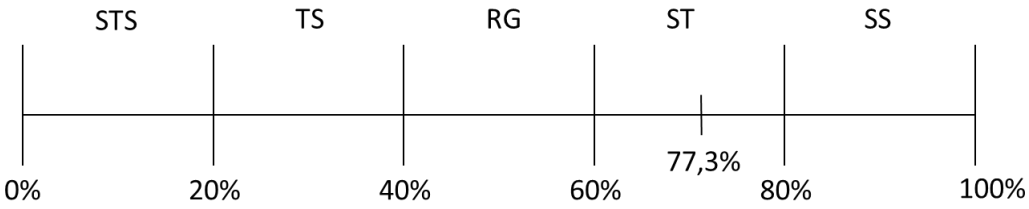
5. Tampilan grafik 2D pada *game* ini sudah bagus.

Tabel 4.13 Perhitungan persentase pertanyaan ke-5

Pernyataan	Keterangan	Skor	Frekuensi	S
5	Sangat Setuju	5	6	30
	Setuju	4	14	56
	Ragu-ragu	3	10	30
	Tidak Setuju	2	-	-
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**116**

$$P = \frac{116}{150} \times 100\% = 77{,}3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap grafik 2D yang dipakai adalah 77,3% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.12 *Interpretasi* skor pertanyaan ke-5

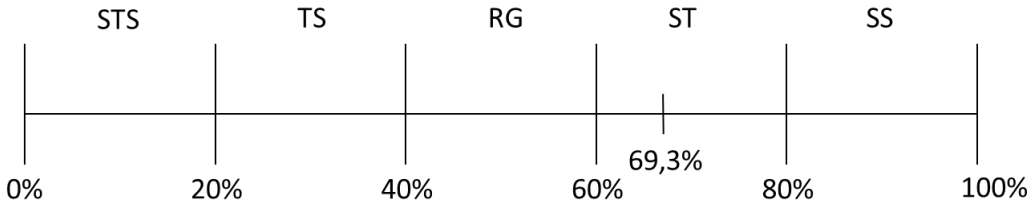
6. *Game* ini menarik untuk dimainkan secara berulang-ulang.

Tabel 4.14 Perhitungan persentase pertanyaan ke-6

Pernyataan	Keterangan	Skor	Frekuensi	S
6	Sangat Setuju	5	2	10
	Setuju	4	10	40
	Ragu-ragu	3	18	54
	Tidak Setuju	2	-	-
	Sangat Tidak Setuju	1	-	-
Jumlah			**30**	**104**

$$P = \frac{104}{150} \times 100\% = 69{,}3\%$$

Berdasarkan hasil presentase nilai di atas, maka dapat disimpulkan bahwa penilaian terhadap ketertarikan pengguna terhadap permainan lutung kasarung *the jumper* adalah 69,3% dari yang diharapkan 100%. Berikut kriteria interpretasi skor :



Gambar 4.13 *Interpretasi* skor pertanyaan ke-6