BAB VI
PENGUJIAN DAN ANALISIS

Bab ini membahas mengenai tahapan pengujian dan analisis perangkat lunak permainan Simulator Angklung pada yang telah dikembangkan. Proses pengujian dilakukan melalui empat tahapan yaitu pengujian unit, pengujian integrasi, pengujian validasi, dan pengujian peforma. Pada pengujian unit dan integrasi, akan digunakan teknik pengujian *White-Box* (*White-Box Testing*). Pada pengujian validasi akan digunakan teknik pengujian *Black-Box* (*Black-Box Testing*). Pada pengujian peforma akan digunakan analisis *transfer rate* untuk mengetahui peforma dari perangkat Simulator Angklung.

6.1. Pengujian

Proses pengujian yang dilakukan melalui empat tahapan yaitu pengujian unit, pengujian integrasi, pengujian validasi, dan pengujian peforma.

6.1.1. Pengujian Unit

Pada pengujian unit digunakan metode *White-Box Testing* dengan teknik *Basis-Path Testing*. Pada teknik *Basis-Path Testing* proses pengujian dilakukan dengan memodelkan kode pada sebuah *flow graph*, menentukan *cyclometic complexcity* dan melakukan uji kasus untuk setiap path yang ada.

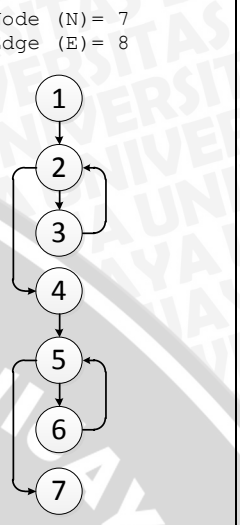
6.1.1.1. Pengujian unit untuk algoritma inisialisasi id angklung

Kode penggunaan touch pada angklung pada proses inisialiasi id angklung dilakukan pada method `onLoadScene()` pada class MultiToneActivity.java. Pemodelan kode penggunaan *touch* untuk inisialisasi id angklung dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.1

Tabel 6.1. Pemodelan kode inisialisasi id angklung dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***

```
ALGORITHM NAME: Inisialisasi id angklung
DECLARATION:
        TYPE sound IS Sound
        Tone IS List
        allsprite IS List
        texture IS Texture
        angklung IS TextureRegion
        angklungSprite IS Sprite
        position IS float
        positionY IS float
        a IS int
INPUT:
DESCRIPTION:
1   tone <- CREATE OBJECT List
    allSprite <- CREATE OBJECT List
    texture <- CREATE OBJECT Texture(128,128);
4   angklung <- CREATE OBJECT  TextureRegion ("texture",
5   "path_gambar.png")
6   FOR a=0 to 7 DO                                        <- 2
7       Sound <- CREATE OBJECT Sound("nada_" + a + ".mp3")  <- 3
8       CALL tone.add(sound)
4   ENDFOR
    positionX <- 50
    positionY <- 180                                       <- 5
12  FOR a=0 to 7 DO
13      angklungSprite <- CREATE OBJECT Sprite (positionX,  positionY,
14  6   angklung)
15      CALL angklungSprite.setAreaTouchId(a)
16      positionX <- positionX + CALL angklungSprite.getWidth() + 50
17      CALL allsprite.add(angklungSprite)
7   ENDFOR
    END
```



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi executeAction() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 8 – 7 + 2 = 3

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 2 - …

Jalur 2: 1 – 2 – 4 – 5 - 6 – 5 - …

Jalur 3: 1 – 2 – 4 – 5 – 7

Tabel 6.2. Kasus uji untuk kode inisalisasi id angklung

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Nilai a pada pendeklarasian suara kurang dari 7	Sistem membuat file audio sebanyak tujuh	Sistem membuat file audio sebanyak tujuh
2	Nilai a pada pendeklarasian	Sistem membuat objek	Sistem membuat

	sprite kurang dari 7	sprite sebanyak tujuh	objek sprite sebanyak tujuh
3	Nilai a pada pendeklarasian sprite lebih dari atau sama dengan 7	Menampilkan gambar angklung sebanyak tujuh	Menampilkan gambar angklung sebanyak tujuh

Sumber: Pengujian dan Analisis

6.1.1.2. Pengujian unit untuk kode pengolahan *input touch*

Kode penggunaan touch pada angklung pada proses pengolahan input touch dilakukan di method `onExecuteAction()` pada class MultiToneActivity.java. Pemodelan kode pengolahan input *touch* pada angklung dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.3.

Tabel 6.3. Pemodelan kode pengolahan *input touch* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***

ALGORITHM NAME: Pengolahan input *touch* pada angklung
DECLARATION:
TYPE a IS int
tone IS List
INPUT: touch IS TouchEvent
DESCRIPTION:
IF CALL touch.getPointerID() > -1 THEN
FOR b=0 to 7 DO
IF CALL touch.getPointerID() == b THEN
CALL tone.get(a).play()
ELSE
CALL tone.get(a).stop()
ENDIF
ENDFOR
ENDIF
END
	Node (N)= 7
	Edge (E)= 9

Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi executeAction() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 9 – 7 + 2 = 4

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 7

Jalur 2: 1 – 2 – 7

Jalur 3: 1 – 2 – 3 – 4 – 6 – 2 - …

Jalur 4: 1 – 2 – 3 – 5 – 6 – 2 - …

Tabel 6.4. Kasus uji untuk kode pengolahan *input touch* ada angklung

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Jika nilai `touch.getPointerID` kurang dari sama dengan -1	Proses pengolahan input touch selesai	Proses pengolahan input touch selesai
2	Nilai b pada perulangan bernilai lebih dari sama dengan 7	Proses pengolahan input touch selesai	Proses pengolahan input touch selesai
3	Menyentuh layar pada daerah gambar angklung. Jika nilai `touch.getPointerID` sama dengan b	mengeluarkan suara angklung	mengeluarkan suara angklung
4	Menyentuh layar pada daerah gambar angklung. Jika nilai `touch.getPointerID` tidak sama dengan b	Mematikan suara angklung	Mematikan suara angklung

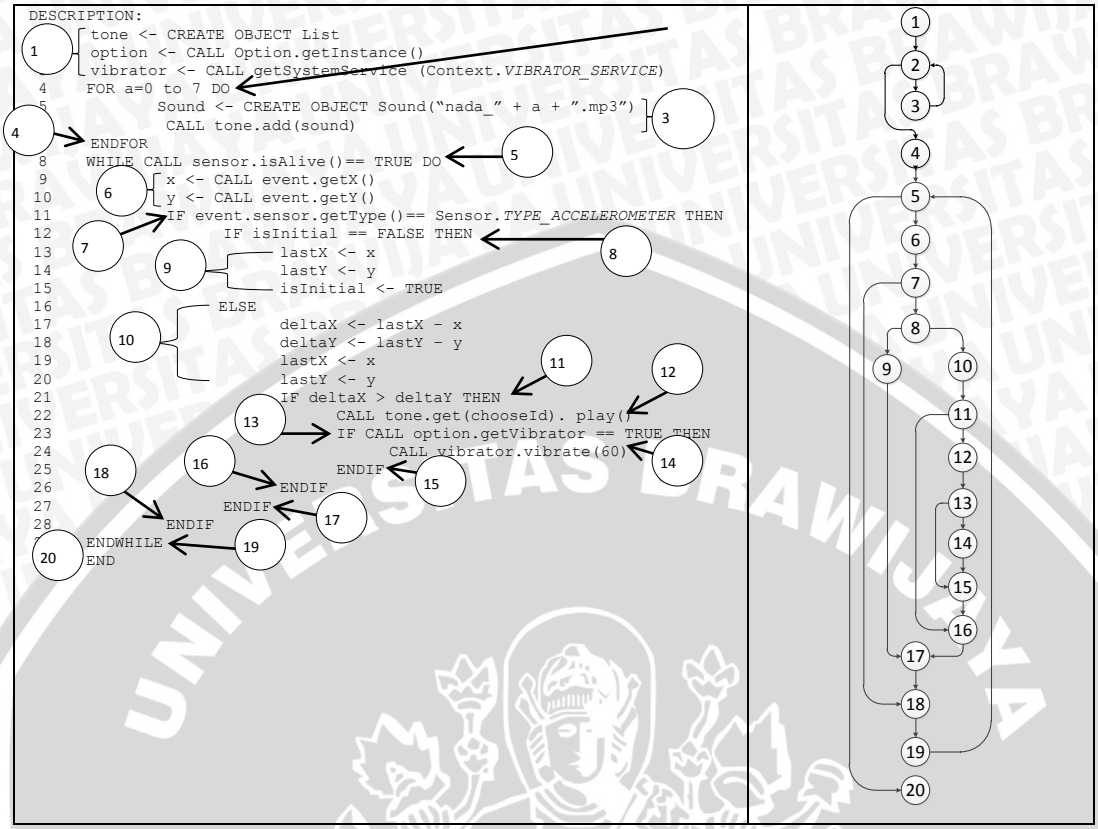
Sumber: Pengujian dan Analisis

6.1.1.3. Pengujian unit untuk kode penggunaan *accelerometer*

Kode penggunaan touch pada angklung dilakukan pada method `onSensorChanged(SensorEvent event)` pada class SingleToneActivity.java. Pemodelan kode penggunaan *accelerometer* pada angklung dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.5.

Tabel 6.5. Pemodelan kode penggunaan *accelerometer* dalam bentuk *flowgraph*

Pseudo Code	*Flowgraph*
ALGORITHM NAME: Penggunaan *accelerometer* pada angklung DECLARATION: TYPE sound IS Sound tone IS List a IS int option IS Option vibrator IS Vibrator x IS float y IS Float deltaX IS Float deltaY IS Float lastX IS Float lastY IS Float sensor IS Thread INPUT: event IS SensorEvent, chooseId IS int	Node (N)= 20 Edge (E)= 25 2



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi sesnsorChanged(SensorEvent event) menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 25 – 20 + 2 = 7

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 2 - …

Jalur 2: 1 – 2 – 4 – 5 – 20

Jalur 3: 1 – 2 – 4 – 5 – 6 – 7 – 8 – 10 – 11 – 12 – 13 –14–15–16–17 –18–19–5 - …

Jalur 4: 1 – 2 – 4 – 5 – 6 – 7 – 8 – 10 – 11 – 12 – 13– 15 – 16 – 17 – 18–19–5 - …

Jalur 5: 1 – 2 – 4 – 5 – 6 – 7 – 8 – 10 – 11 – 16 – 17 – 18 – 19 – 5 - …

Jalur 6: 1 – 2 – 4 – 5 – 6 – 7 – 8 – 9 – 17 – 18 – 19 – 5 - …

Jalur 7: 1 – 2 – 4 – 5 – 6 – 7 — 18 – 19 – 5 - …

Tabel 6.6. Kasus uji untuk kode penggunaan *accelerometer* ada angklung

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Nilai a pada pendeklarasian suara kurang dari 7	Membuat file audio sebanyak tujuh	Membuat file audio sebanyak tujuh
2	Nilai sensor.isAlive() bernilai *false*	Sistem tidak mengolah input dari sensor	Sistem tidak mengolah input dari sensor
3	Nilai isInitial sama dengan true. Smartphone digerakkan pada sumbu x dan pengaturan vibrator aktif	Mengeluarkan suara angklung dan *Smartphone* bergetar	Mengeluarkan suara angklung dan *Smartphone* bergetar
4	Nilai isInitial sama dengan true. Smartphone digerakkan pada sumbu x dan pengaturan vibrator tidak aktif	Mengeluarkan suara angklung dan *Smartphone* tidak bergetar	Mengeluarkan suara angklung dan *Smartphone* tidak bergetar
5	Nilai isInitial sama dengan true. Smartphone digerakkan selain pada sumbu x	Tidak mengeluarkan suara angklung	Tidak mengeluarkan suara angklung
6	Nilai isInitial sama dengan false.	Sistem memproses nilai posisi awal pada sumbu x	Sistem memproses nilai posisi awal pada sumbu x
7	Nilai input bukan dari sensor accelerometer	Sistem tidak memproses input	Sistem tidak memproses input

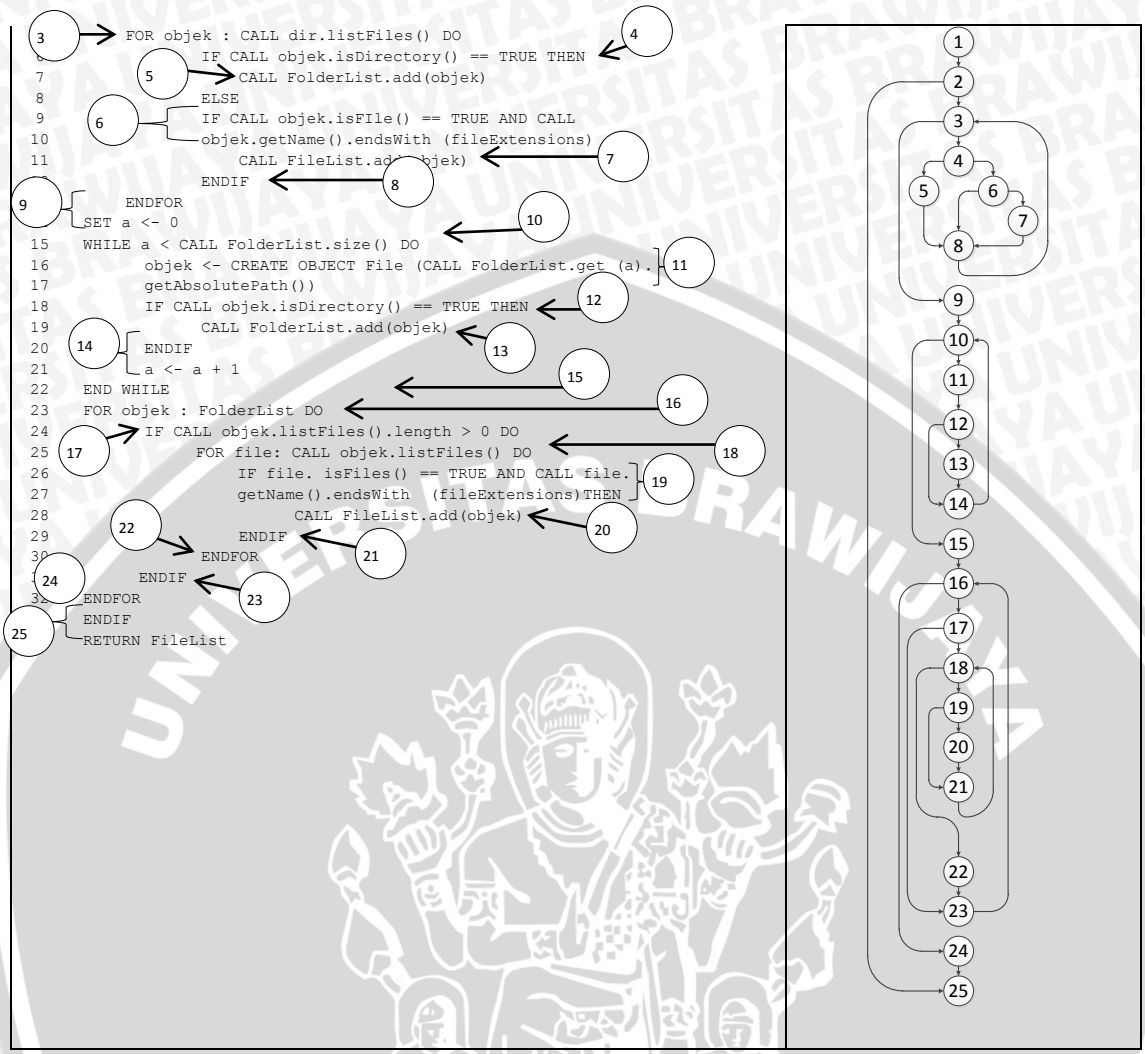
Sumber: Pengujian dan Analisis

6.1.1.4. Pengujian unit untuk pendeteksian *file beat*

Kode pendeteksian *file beat* dilakukan pada method `filter()` pada class FileGameFilter.java. Pemodelan kode pendeteksian *file beat* dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.7.

Tabel 6.7. Pemodelan kode pendektesian *file beat* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***
ALGORITHM NAME: Pendeteksian file beat DECLARATION: TYPE fileExtensions IS String <- ".ang" FolderList IS List FileList IS List dirpath IS STRING <- "sdcard/Sarinande"; dir IS File objek IS File file IS File a IS int b IS int DESCRIPTION: 1 FolderList <- CREATE OBJECT List 2 FileList <- CREATE OBJECT List 3 dir <- CREATE OBJECT File(dirpath) 4 IF CALL dir.listFiles().length > 0 THEN	Node(N) : 25 Edge (E) : 34



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi filter() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 34 – 25 + 2 = 11

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 8 – 3 - …

Jalur 2: 1 – 2 – 3 – 4 – 6 – 8 – 3 - …

Jalur 3: 1 – 2 – 3 – 4– 6 – 7 – 8 – 3 - …

Jalur 4: 1 – 2 – 25

Jalur 5: 1 – 2 – 3 – 9 – 10 – 11 – 12 – 13 – 14 – 10 –…

Jalur 6: 1 – 2 – 3 – 9 – 10 – 11 – 12 – 14 – 10 –…

Jalur 7: 1 – 2 – 3 – 9 – 10 – 15 – 16 – 24 – 25

Jalur 8: 1 – 2 – 3 – 9 – 10 – 15 – 16 – 17 – 18 – 19 – 20 – 21 – 18 – …

Jalur 9: 1 – 2 – 3 – 9 – 10 – 15 – 16 – 17 – 18 – 19 – 21 – 18 – …

Jalur 10: 1 – 2 – 3 – 9 – 10 – 15 – 16 – 17 – 18 – 22 – 23 – 16 – …

Jalur 11: 1 – 2 – 3 – 9 – 10 – 15 – 16 – 17 – 23 – 16 – …

Tabel 6.8. Kasus uji untuk kode pendeteksian *file beat*

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Jika objek merupakan folder pada sdcard/sarinande	Menambahkan objek pada folder list	Menambahkan objek pada folder list
2	Jika objek merupakan file yang berekstensi .ang	menambahkan objek pada file list	menambahkan objek pada file list
3	Jika objek bukan merupakan file yang berekstensi .ang dan bukan merupakan folder	Tidak menambahkan objek pada folder list dan file list	Tidak menambahkan objek pada folder list dan file list
4	Jika tidak terdapat folder dan file pada sdcard/sarinande	Sistem mengembalikan nilai kosong	Sistem mengembalikan nilai kosong
5	Jika objek merupakan sub folder pada semua folder di sdcard/sarinande	Menambahkan objek pada folder list	Menambahkan objek pada folder list
6	Jika objek bukan merupakan sub folder pada semua folder di sdcard/sarinande	Tidak menambahkan objek pada folder list	Tidak menambahkan objek pada folder list
7	Jika data pada folder list sama dengan 0	Mengembalikan semua file yang berekstensi .ANG pada sdcard/sarinande	Mengembalikan semua file yang berekstensi .ANG pada sdcard/sarinande
8	Jika objek pada semua folder di folder list merupakan file berekstensi .ang	Menambahkan objek pada file list	Menambahkan objek pada file list
9	Jika objek pada semua folder di folder list bukan merupakan file berekstensi .ang	Tidak menambahkan objek pada file list	Tidak menambahkan objek pada file list
10	Jika semua objek pada folder telah dideteksi	Sistem menghentikan proses seleksi kondisi pencarian file berekstensi .ANG pada folder tersebut	Sistem menghentikan proses seleksi kondisi pencarian file berekstensi .ANG pada folder tersebut

11	Jika folder yang dideteksi tidak memiliki file di dalamnya	Sistem menghentikan pencarian file berekstensi .ANG pada folder tersebut	Sistem menghentikan proses seleksi kondisi pencarian file berekstensi .ANG

Sumber: Pengujian dan Analisis

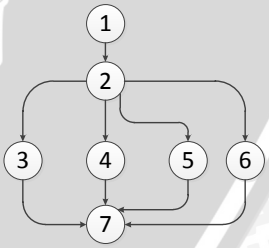
6.1.1.5. Pengujian unit untuk pembacaan file beat

Kode pendeteksian *file beat* dilakukan pada class MySAXParser.java. Pembacaan file beat dilakukan melalui lima tahap, yaitu init, pembacaan start element, pembacaan character, pembacaan end element, dan penyimpanan nilai pada *file beat*.

A. Init

Proses pembacaan file beat pada proses init dilakukan pada method parseDocument(). Pemodelan kode pembacaan *file beat* pada proses init dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.9.

Tabel 6.9. Pemodelan proses init dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***
ALGORITHM NAME: Pembacaan file beat: Init DECLARATION: TYPE spf IS SAXParserFactory time IS List node IS List sax IS SAXParser gf IS GameFile tempVal IS String DESCRIPTION: 1 time <- CREATE OBJECT List 2 node <- CREATE OBJECT List spf <- CALL SAXParserFactory BEGIN sax <- CALL spf.newSAXParser() gf <- CALL GameFile.getInstance() path <- CALL gf.getGameFile().getPath() CALL sp.parse(CREATE OBJECT InputSource(CREATE OBJECT FileInputStream("path"))) EXCEPTION WHEN ParserConfiguration e WRITE e WHEN SAXException e WRITE e WHEN IOExcepetion e WRITE e END	Node (N): 7 Edge(E): 9 

Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi parseDocument() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 9 – 7 + 2 = 4

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 7

Jalur 2: 1 – 2 – 4 – 7

Jalur 3: 1 – 2 – 5 – 7

Jalur 4: 1 – 2 – 6 – 7

Tabel 6.10. Kasus uji untuk kode pembacaan file beat pada proses init

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Kasus uji berhasil melakukan init	Proses berlanjut dan tidak ada pesan yang ditampilkan	Proses berlanjut dan tidak ada pesan yang ditampilkan
2	Kasus uji gagal melakukan init karena kesalahan konfigurasi parser	Sistem memberhentikan proses parsing	Sistem memberhentikan proses parsing
3	Kasus uji gagal melakukan init karena kesalahan pembuatan file beat dengan format xml	Sistem memberhentikan proses parsing	Sistem memberhentikan proses parsing
4	Kasus uji gagal melakukan init karena kesalahan pemasukan *file beat* yang di akan dibaca	Sistem memberhentikan proses parsing	Sistem memberhentikan proses parsing

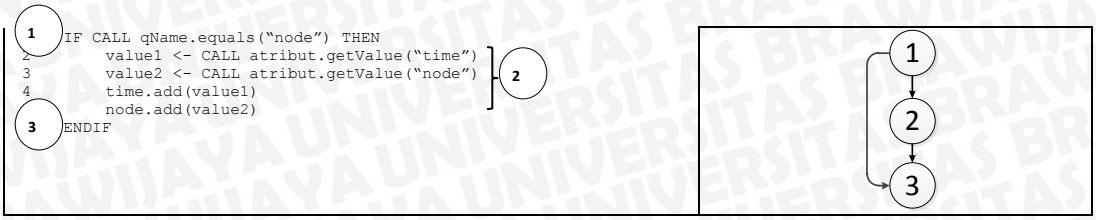
Sumber: Pengujian dan Analisis

B. Pembacaan *start element*

Proses pembacaan file beat pada pembacaan *start element* dilakukan pada method startElement(). Pemodelan kode pembacaan *file beat* pada pembacaan start element dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.11.

Tabel 6.11. Pemodelan kode proses pembacaan *start element* dalam bentuk *flowgraph*

Pseudo Code	*Flowgraph*
ALGORITHM NAME: Pembacaan file beat: startElement DECLARATION: TYPE value1 IS String value2 IS String INPUT: uri IS String, localName IS String, qName IS String, atribut IS Attributes DESCRIPTION:	Node(N): 3 Edge(E): 3



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi startElements() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 3 - 3 + 2 = 2

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3

Jalur 2: 1 – 3

Tabel 6.12. Kasus uji untuk kode pada pembacaan *start element*

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Membaca start elemen node	Menyimpan nilai atribut pada elemen node	menyimpan nilai atribut pada elemen node
2	Membaca start elemen selain elemen node	Tidak menyimpan nilai atribut	Tidak menyimpan nilai atribut

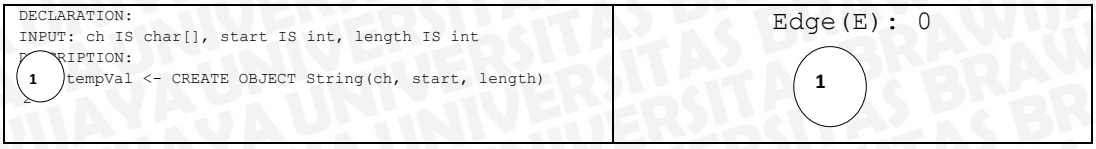
Sumber: Pengujian dan Analisis

C. Pembacaan *character*

Proses pembacaan file beat pada pembacaan *character* dilakukan pada method characters(). Pemodelan kode pembacaan *file beat* pada pembacaan *character* dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.13.

Tabel 6.13. Pemodelan kode proses pembacaan *character* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***
ALGORITHM NAME: Pembacaan file beat: characters	Node(N) : 1



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi character() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 0 - 1 + 2 = 1

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1

Tabel 6.14. Kasus uji untuk kode proses pembacaan *characters*

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Membaca karakter pada elemen	membaca karakter diantara elemen	Membaca karakter diantara elemen

Sumber: Pengujian dan Analisis

D. Pembacaan *end element*

Proses init pada pembacaan file beat pada pembacaan *character* dilakukan pada method endElements(). Pemodelan kode pembacaan *file beat* pada pembacaan end element dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.15.

Tabel 6.15. Pemodelan kode pembacaan *end element* dalam *flowgraph*

Pseudo Code	***Flowgraph***
ALGORITHM NAME: Pembacaan file beat: endElement DECLARATION: INPUT: uri IS String, localName is String, qName IS String DESCRIPTION: CASE qName OF title : title <- tempVal BREAK author : author <- tempVal BREAK song : song <- tempVal BREAK ENDCASE	Node(N): 7 Edge(E): 9

Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi endElements() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 9 - 7 + 2 = 4

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 5

Jalur 2: 1 – 3 – 5

Jalur 3: 1 – 4 – 5

Tabel 6.16. Kasus uji untuk kode pembacaan *end elements*

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Jika nilai qName sama dengan *title*	Membaca dan menyimpan karakter sebelum elemen *title*	Membaca dan menyimpan karakter sebelum elemen *title*
2	Jika nilai qName sama dengan *author*	Membaca dan menyimpan karakter sebelum akhir elemen *author*	Membaca dan menyimpan karakter akhir sebelum akhir elemen *author*
3	Jika nilai qName sama dengan *song*	Membaca dan menyimpan karakter sebelum akhir elemen *song*	Membaca dan menyimpan karakter sebelum akhir elemen *song*
4	Jika nilai qName tidak sama dengan *title, author,* dan *song*	Tidak membaca nilai apapun	Tidak membaca nilai apapun

Sumber: Pengujian dan Analisis

E. Penyimpanan nilai

Proses pembacaan file beat pada penyimpanan nilai dilakukan pada method print(). Pemodelan kode pembacaan *file beat* pada penyimpanan nilai dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.17.

Tabel 6.17. Pemodelan kode proses penyimpanan nilai dalam bentuk *flowgraph*

Pseudo Code	*Flowgraph*
ALGORITHM NAME: Pembacaan file beat: Penyimpanan hasil pembacaan DECLARATION: INPUT: uri IS String, localName is String, qName IS String DESCRIPTION: 1 CALL gf.setTitle(title) 2 CALL gf.setAuthor(author) 3 CALL gf.setMusicGame(song) 4 CALL gf.setTime(time) 5 CALL gf.setNode(node) 1	Node(N): 1 Edge(E): 0 1

Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi print() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 0 - 1 + 2 = 1

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1

Tabel 6.18. Kasus uji untuk kode penyimpanan nilai

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Menyimpan semua data pembacaan elemen disimpan pada GameFile.Java	Semua data pada elemen akan tersimpan pada objek GameFile	Semua data pada elemen tersimpan pada objek GameFile

Sumber: Pengujian dan Analisis

6.1.1.6. Pengujian unit pembuatan kotak beat

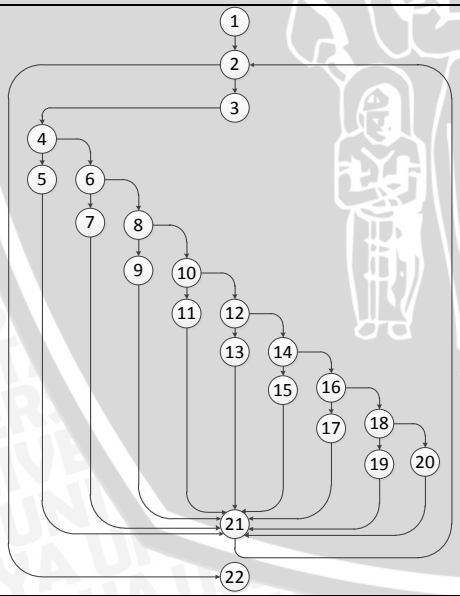
Kode pembuatan kotak *beat* dilakukan pada method `tapLoads()` pada class TapCollector.java. Pemodelan kode pembuatan kotak *beat* dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.19.

Tabel 6.19. Pemodelan kode pembuatan kotak *beat* dalam bentuk *flowgraph*

Pseudo Code
ALGORITHM NAME: Pembuatan kotak beat DECLARATION: TYPE tap IS LIST tr IS TextureRegion scene IS Scene nodeLis IS List timeList IS List game IS GameFile notasi IS String

```
        time IS Float
        beat IS AbstractTap
        a IS int
        switcher IS int
DESCRIPTION:
 1    tap <- CREATE OBJECT List
 2    scene <- CREATE OBJECT Scene
      tr <- CREATE OBJECT TextureRegion ("path_gambar.png")
 1    nodeList <- CREATE OBJECT List
      timeList <- CREATE OBJECT List
 6    game <- CALL GameFile.getInstance()
 7    nodeList <- CALL game.getNode()
 8    timeList <- CALL game.getTime()
 9    FOR a=0 to CALL nodeList.size() DO                          2
10          notasi <- CALL nodeList.get(a)                       3
11          time <- CALL Float.parseFloat(timeList.get())/1000
12    4     CASE notasi OF
13              do    :
14                      beat <- CREATE OBJECT DoTap              5
15                      (tr, timing, scene) BREAK
16        6     re    :
17                      beat <- CREATE OBJECT ReTap              7
18                      (tr, timing, scene) BREAK
19      8       mi    :
20                      beat <- CREATE OBJECT MiTap              9
21                      (tr, timing, scene)BREAK
22        10    fa    :
23                      beat <- CREATE OBJECT FaTap              11
24                      (tr, timing, scene) BREAK
25      12      sol   :
26                      beat <- CREATE OBJECT SolTap             13
27                      (tr, timing, scene) BREAK
28        14    la    :
29                      beat <- CREATE OBJECT LaTap              15
30                      (tr, timing, scene) BREAK
31      16      si    :
32                      beat <- CREATE OBJECT SiTap              17
33                      (tr, timing, scene) BREAK
34        18    do2   :
35                      beat <- CREATE OBJECT Do2Tap             19
36    20                (tr, timing, scene) BREAK
37              default: beat <- NULL BREAK
38          ENDCASE
39          CALL [illegible].add(beat)                           21
40    ENDFOR                                                     22
41    END
```

Flowgraph



Node(N): 22

Edge(E):30

Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi tapLoads() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 30 - 22 + 2 = 10

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 22

Jalur 2: 1 – 2 – 3 – 4 – 5 – 21 – 2 - …

Jalur 3: 1 – 2 – 3 – 4 – 6 – 7 – 21 – 2 – …

Jalur 4: 1 – 2 – 3 – 4 – 6 – 8 – 9 – 21 – 2 – …

Jalur 5: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 11 – 21 – 2 – …

Jalur 6: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 13 – 21 – 2 – …

Jalur 7: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 15 – 21 – 2 – …

Jalur 8: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 17 – 21 – 2 – …

Jalur 9: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 18 – 19 – 21 – 2 – …

Jalur 10: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 18 – 20 – 21 – 2 – …

Tabel 6.20. Kasus uji untuk kode pembuatan kotak *beat*

No	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Jika nilai a pada perulangan berjumlah sama dengan data pada nodeList	Sistem berhenti membuat kotak *beat*	Sistem berhenti membuat kotak *beat*
2	Membuat kotak beat jika nilai notasi sama dengan do	Membuat objek doTap	Membuat objek doTap
3	Membuat kotak beat jika nilai notasi sama dengan re	Membuat objek reTap	Membuat objek reTap
4	Membuat kotak beat jika nilai notasi sama dengan mi	Membuat objek miTap	Membuat objek miTap
5	Membuat kotak beat jika nilai notasi sama dengan fa	Membuat objek faTap	Membuat objek faTap
6	Membuat kotak beat jika nilai notasi sama dengan sol	Membuat objek solTap	Membuat objek solTap
7	Membuat kotak beat jika nilai notasi sama dengan la	Membuat objek laTap	Membuat objek laTap
8	Membuat kotak beat jika nilai notasi sama dengan si	Membuat objek siTap	Membuat objek siTap
9	Membuat kotak beat jika nilai notasi sama dengan do2	Membuat objek do2Tap	Membuat objek do2Tap
10	Membuat kotak beat jika nilai	Tidak membuat objek	Tidak membuat

	notasi tidak sama dengan do, re, mi, fa, sol, la, si, do2	AbstractTap	objek AbstractTap

Sumber: Pengujian dan Analisis

6.1.1.7. Pengujian unit untuk pengoneksian client ke server

Kode pengoneksian client ke server pada sisi *server* terletak pada class Server.java sedangkan pada sisi *client* terletak pada Client.java.

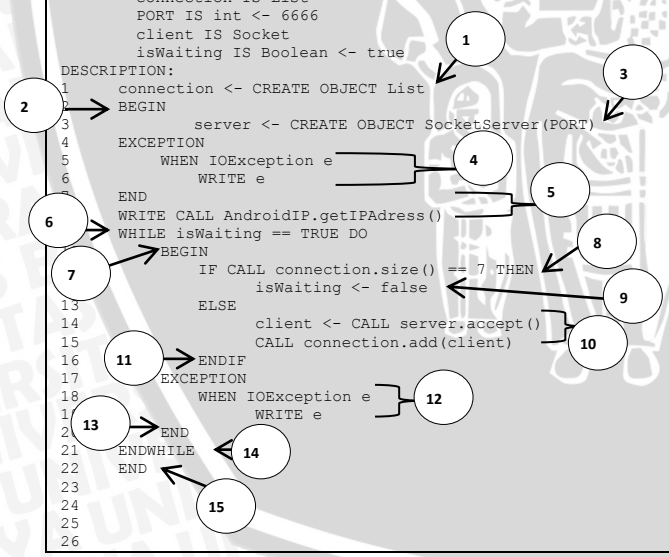
A. Sisi *Server*

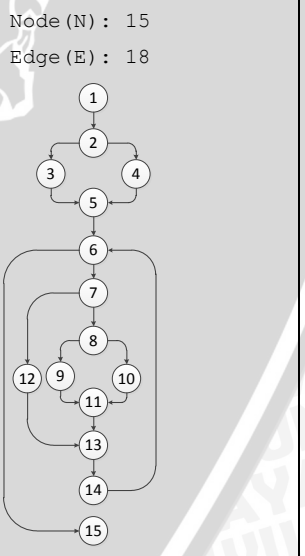
Pengoneksian client dan server pada sisi server dilakukan di method startServer() pada Server.java Pemodelan pengoneksian *client* ke *server* pada sisi server dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.21.

Tabel 6.21. Pemodelan kode pengoneksian *client* ke *server* pada sisi *server* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***

```
ALGORITHM NAME: TCP Server
DECLARATION:
        TYPE server IS SocketServer
        connection IS List
        PORT IS int <- 6666
        client IS Socket
        isWaiting IS Boolean <- true
DESCRIPTION:
1       connection <- CREATE OBJECT List
2       BEGIN
3               server <- CREATE OBJECT SocketServer(PORT)
4       EXCEPTION
5           WHEN IOException e
6               WRITE e
        END
        WRITE CALL AndroidIP.getIPAdress()
        WHILE isWaiting == TRUE DO
            BEGIN
                IF CALL connection.size() == 7 THEN
                    isWaiting <- false
13              ELSE
14                  client <- CALL server.accept()
15                  CALL connection.add(client)
16          ENDIF
17      EXCEPTION
18          WHEN IOException e
1               WRITE e
2       END
21      ENDWHILE
22      END
23
24
25
26
```





Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi startServer() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 18 - 15 + 2 = 5

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 5 – 6 – 15

Jalur 2: 1 – 2 – 4 – 5 – 6 – 15

Jalur 3: 1 – 2 – 3 – 5 – 6 – 7 – 8 – 9 – 11 – 13 – 14 – 6 - …

Jalur 4: 1 – 2 – 3 – 5 – 6 – 7 – 8 – 10 – 11 – 13 – 14 – 6 - …

Jalur 5: 1 – 2 – 3 – 5 – 6 – 7 – 12 – 13 – 14 – 6 - …

Tabel 6.22. Kasus uji kode pengoneksian *client ke server* pada sisi *server*

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Server berhasil dijalankan dan nilai isWaiting sama dengan false	Server berjalan dan proses menunggu client berhenti	Server berjalan dan proses menunggu client berhenti
2	Server tidak berhasil dijalankan dan nilai isWaiting sama dengan false	Server tidak berjalan dan tidak dilakukan proses menunggu client	Server tidak berjalan dan tidak dilakukan proses menunggu client
3	Jumlah client yang ditangkap berjumlah tujuh	proses menunggu client berhenti	proses menunggu client berhenti
4	Jumlah client yang ditangkap kurang dari tujuh	proses menunggu client berjalan	proses menunggu client berjalan
5	Terdapat kesalahan dalam penangkapan *client*	Server dimatikan dan mematikan proses menunggu *client*	Server dimatikan dan mematikan proses menunggu *client*

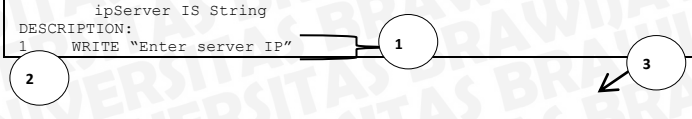
Sumber: Pengujian dan Analisis

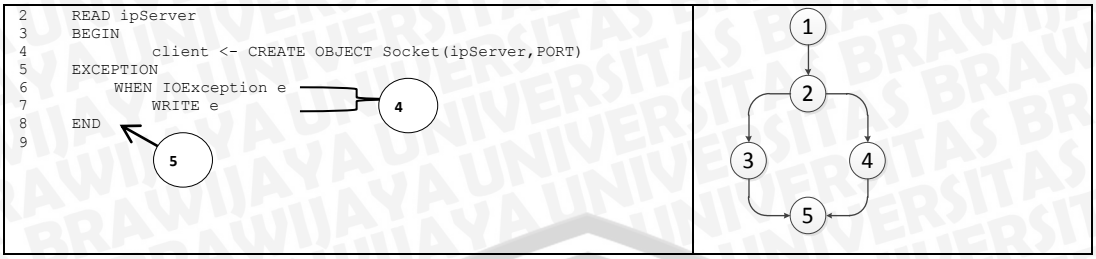
B. Sisi *Client*

Pengoneksian client dan server pada sisi client dilakukan di method initClient() pada Client.java Pemodelan pengoneksian client ke server pada sisi client dalam bentuk *flowgraph* ditunjukkan pada Tabel 6.23.

Tabel 6.23. Pemodelan kode pengoneksian *client* ke *server* pada sisi *client* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***
ALGORITHM NAME: TCP Client DECLARATION: TYPE PORT IS int <- 6666 client IS Socket ipServer IS String DESCRIPTION: 1 WRITE "Enter server IP"	Node(N): 5 Edge (E): 5





Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi initClient() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 5 - 5 + 2 = 2

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 5

Jalur 2: 1 – 2 – 4 – 5

Tabel 6.24. Kasus uji kode pengoneksian *client* ke *server* pada sisi *client*

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Mengoneksikan ke server pada ip tertentu pada port 6666	Client terkoneksi ke server	Client terkoneksi ke server
2	Gagal mengoneksikan ke server pada ip tertentu pada port 6666	Client gagal terkoneksi ke server	Client gagal terkoneksi ke server

Sumber: Pengujian dan Analisis

6.1.1.8. Pengujian unit pembuatan *file beat*

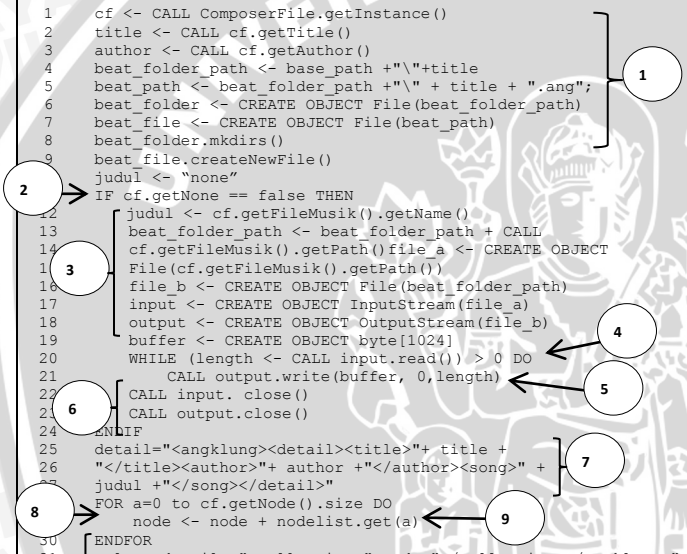
Kode pembuatan file beat dilakukan di method makeFile terletak pada class MakeComposerFile.java. Pemodelan kode pembuatan *file beat* ditunjukkan pada Tabel 6.25.

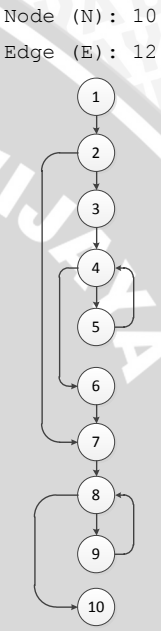
Tabel 6.25. Pemodelan kode pembuatan *file beat* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***

```
ALGORITHM NAME: Pembuatan file beat
DECLARATION:
          TYPE save_node IS String
          title IS String
          author IS String
          file_path IS String
          beat_path IS String
          beat_folder_path IS String
          detail IS String
          xml IS String
          base_path IS String <-"sdcard/Sarinande"
          beat_file IS File
          beat_folder IS File
          file_a IS FIle
          file_b IS File
          out IS FileOutputStream
          fout IS OutputStreamWriter
          input IS InputStream
          output IS OutputStream
          music IS Music
          input_byte IS byte
          cf IS ComposerFile
DESCRIPTION:
 1    cf <- CALL ComposerFile.getInstance()
 2    title <- CALL cf.getTitle()
 3    author <- CALL cf.getAuthor()
 4    beat_folder_path <- base_path +"\"+title
 5    beat_path <- beat_folder_path +"\" + title + ".ang";
 6    beat_folder <- CREATE OBJECT File(beat_folder_path)
 7    beat_file <- CREATE OBJECT File(beat_path)
 8    beat_folder.mkdirs()
 9    beat_file.createNewFile()
      judul <- "none"
      IF cf.getNone == false THEN
        judul <- cf.getFileMusik().getName()
        beat_folder_path <- beat_folder_path + CALL
        cf.getFileMusik().getPath()file_a <- CREATE OBJECT
        File(cf.getFileMusik().getPath())
        file_b <- CREATE OBJECT File(beat_folder_path)
        input <- CREATE OBJECT InputStream(file_a)
        output <- CREATE OBJECT OutputStream(file_b)
        buffer <- CREATE OBJECT byte[1024]
        WHILE (length <- CALL input.read()) > 0 DO
            CALL output.write(buffer, 0,length)
        CALL input. close()
        CALL output.close()
      ENDIF
      detail="<angklung><detail><title>"+ title +
      "</title><author>"+ author +"</author><song>" +
      judul +"</song></detail>"
      FOR a=0 to cf.getNode().size DO
          node <- node + nodelist.get(a)
      ENDFOR
      xml <- detail +"<collection>"+node+"</collection></angklung>"

      out <- CREATE OBJECT FileOutputStream(beat_file)
      fout <- CREATE OBJECT OutputStreamWriter(out)
      CALL fout.write(xml)
      CALL fout.close()
      CALL out.close()
      END
```





Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi makeFile() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 12 - 10 + 2

= 4

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 4 –…

Jalur 2: 1 – 2 – 4– 6 – 7 – 8 – 9 – 8 – …

Jalur 3: 1 – 2 – 7 – 8 – 9 – 8 – …

Jalur 4: 1 – 2 – 4– 6 – 7 – 8 – 10

Tabel 6.26. Kasus uji untuk kode pembuatan *file beat*

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Jika Nilai cf.getNone sama dengan false dan Nilai length lebih dari 0	Memberikan nilai pada judul dengan nama musik dan proses penduplikasian file musik masih dilakukan	Memberikan nilai pada judul dengan nama musik dan proses pemindahan file musik masih dilakukan
2	Jika nilai cf.getNone dama dengan false dan nilai a kurang dari jumlah nodelist	proses penduplikasian file musik telah dilakukan dan Proses penggabungan node masih dilakukan	proses penduplikasian file musik telah dilakukan dan Proses penggabungan node masih dilakukan
3	Jika nilai cf.getNone dama dengan true dan nilai a kurang dari jumlah nodelist	Tidak terjadi penduplikasian file musik dan proses penggabungan node masih dilakukan	Tidak terjadi penduplikasian file musik dan proses penggabungan node masih dilakukan
4	nilai a lebih dari atau sama dengan jumlah nodelist	File beat telah dibuat	File beat telah dibuat

Sumber: Pengujian dan Analisis

6.1.2. Pengujian Integrasi

Pengujian integrasi memastikan bahwa dua atau lebih unit atau integrasi lainya dapat bekerja bersama dengan baik dan agar lebih fokus pada antarmuka yang terspesifikasi

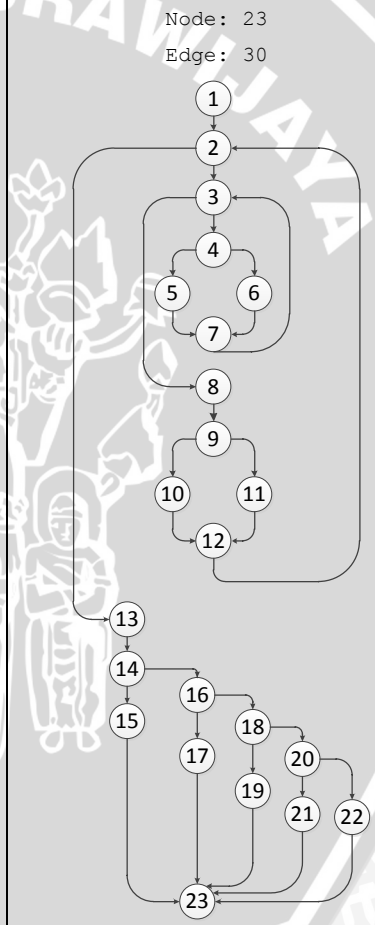
6.1.2.1. Pengujian integrasi pada penambahan *score*

Kode penambahan score dilakukan di method playGame() pada class PlayGameActivity.java. Pemodelan kode penambahan score ditunjukkan pada Tabel 6.27.

Tabel 6.27. Pemodelan algoritma penambahan *score* dalam bentuk *flowgraph*

Pseudo Code	***Flowgraph***

```
ALGORITHM NAME: Penambahan score
DECLARATION:
        TYPE score ARE int
        inGame IS Thread
        areaHit IS rectangle
        scene IS Scene
        factory IS TapCollector
        startTime IS long
        finishTime IS long
        elapseTime IS long
        beat IS AbstractTap
        beat2 IS AbstractTap
        listBeat IS List
        file IS GameFile
        time IS String
        maxScore IS Float
        finalScore IS Float
        rank IS String
INPUT: touch IS TouchEvent
DESCRIPTION:
  1     areaHit <- CREATE OBJECT Rectangle(0,0,720,20)
  2     file <- CALL GameFile.getInstance()
  3     time <- CALL file.getTime().get(CALL file.size() -1)
  4     finishTime <- CALL Long.parseToLong(time) + 1000
  5     listBeat <- CREATE OBJECT List
  6     factory <- CALL TapCollector.getInstance()
        CALL factory.tapLoads()
        listBeat <- CALL factory.getAllTaps()
        startTime <- CALL System.CurrentTimeMilis()
        score <- 0
        elapseTime <- CALL System.CurrentTi      is() - startTi
        WHILE elapseTime <= finishTime DO
              FOR beat : listBeat DO
                    IF CALL beat.collidesWith(areaHit)
                          beat.setInAreaHit(true)
                          beat2 <- beat
                    ELSE
                          beat.setInAreaHit(true)
                    ENDIF
              ENDFOR
              IF CALL touch.getPointerID() > -1 AND beat2 != NULL
              AND   beat2.getInAreaHit()   ==   TRUE   AND   IF   CALL
              touch.getPointerID() == beat.getIdNode() THEN
                    score <- score + 1
                    CALL good()
                    CALL super.executeAction(touch)
              ELSE
                    CALL miss()
                    CALL super.executeAction(touch)
              ENDIF
              elapseTime   <-   CALL   System.CurrentTimeMilis()   -
              startTime
        ENDWHILE
        maxScore <- CALL factory.getAllTaps()
        finalScore <- score/maxScore
        IF finalScore == 1 THEN
              rank <- "A"
        ELSE IF finalScore >= 0.75
        AND finalScore < 1 THEN
              rank <- "B"
        ELSE IF finalScore >= 0.5
        AND finalScore < 75 THEN
              rank <- "C"
        ELSE IF finalScore >= 0.25
        AND finalScore < 0.5 THEN
              rank <- "D"
        ELSE
              rank <- "E"
        ENDIF
```



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi playGame() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 30 - 23 + 2 = 9

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 7 – 3 - …

Jalur 2: 1 – 2 – 3 – 4 – 6 – 7 – 3 - …

Jalur 3: 1 – 2 – 3 – 8 – 9 – 10 – 12 – 2 - …

Jalur 4: 1 – 2 – 3 – 8 – 9 – 11 – 12 – 2 - …

Jalur 5: 1 – 2 – 13 – 14 – 15 – 23

Jalur 6: 1 – 2 – 13 – 14 – 16 – 17 – 23

Jalur 7: 1 – 2 – 13 – 14 – 16 – 18 – 19 – 23

Jalur 8: 1 – 2 – 13 – 14 – 16 – 18 – 20 – 21 – 23

Jalur 9: 1 – 2 – 13 – 14 – 16 – 18 – 20 – 22 – 23

Tabel 6.28. Kasus uji untuk kode penambahan *score*

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Kotak beat bersinggungan dengan area hit	Memberi nilai flag pada kotak beat sama dengan true	Memberi nilai flag pada kotak beat sama dengan true
2	Kotak beat tidak bersinggungan dengan area hit	Memberi nilai flag pada kotak beat sama dengan false	Memberi nilai flag pada kotak beat sama dengan false
3	Nilai id angklung yang disentuh sama dengan id kotak beat.	Menampilkan tulisan *good* , menambah *score,*	Menampilkan tulisan *good* , menambah *score,*
4	Nilai id angklung yang disentuh tidak sama dengan id kotak beat.	Menampilkan tulisan *miss*	Menampilkan tulisan *miss*
5	Jika nilai final score sama dengan 1	Nilai rank sama dengan A	Nilai rank sama dengan A
6	Jika nilai final score di antara 0.75 sampai dengan 1	Nilai rank sama dengan B	Nilai rank sama dengan B
7	Jika nilai final score di antara	Nilai rank sama	Nilai rank sama

	0.5 sampai dengan 0.75	dengan C	dengan C
8	Jika nilai final score di antara 0.25 sampai dengan 0.5	Nilai rank sama dengan D	Nilai rank sama dengan D
9	Jika nilai final score kurang dari 0.25	Nilai rank sama dengan E	Nilai rank sama dengan E

Sumber: Pengujian dan Analisis

6.1.2.2. Pengujian integrasi pada pengomposisian *file beat*

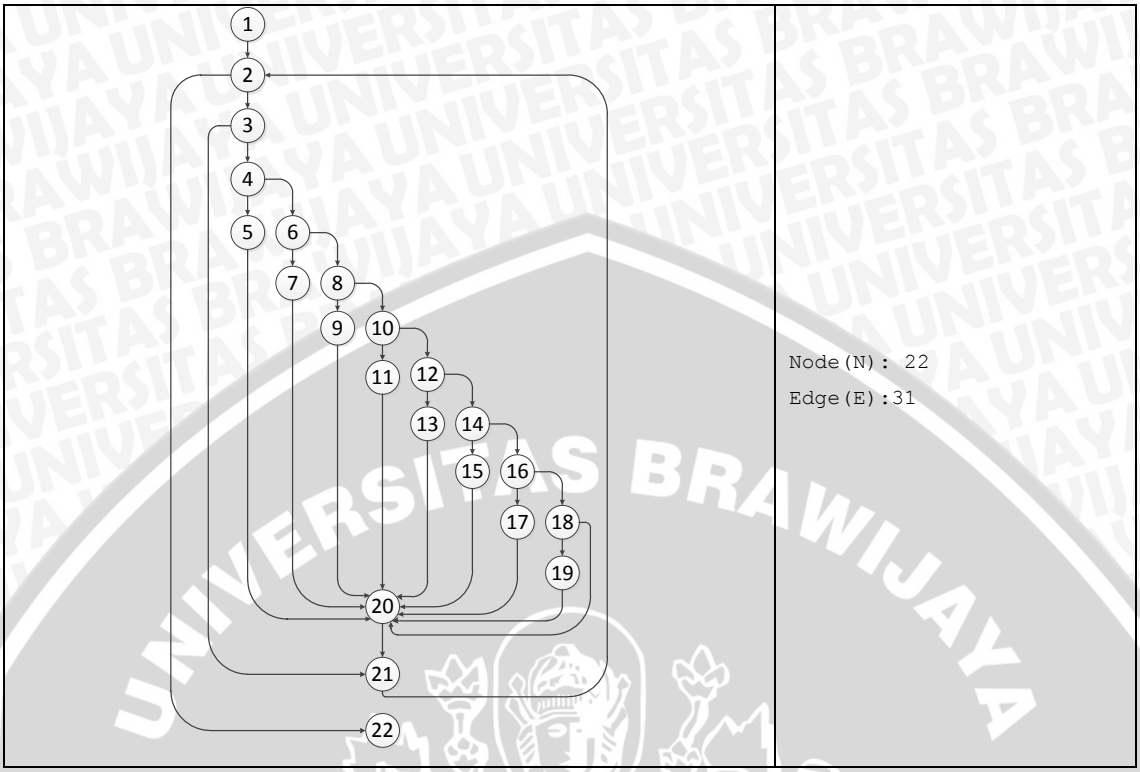
Pengomposisian *file beat* menerapkan kode penangkapan aktivitas angklung yang dilakukan pada method executeAction() pada class ComposerActivity.java. Pemodelan kode penangkapan aktivitas angklung ditunjukkan pada Tabel 6.29.

Tabel 6.29. Pemodelan algoritma penangkapan aktivitas angklung dalam bentuk *flowgraph*

Pseudo Code

```
ALGORITHM NAME: Penangkapan aktivitas angklung
DECLARATION:
        TYPE Nodelist IS List
          start_time IS Long
          elapse_time IS Long
          a IS Int
          isFinish IS boolean <- false
          node IS String
          mcf IS MakeComposerFile
          cf IS ComposerFile
INPUT: touch IS TouchEvent
DESCRIPTION:
        nodelist <- CREATE OBJECT List
        cf <- CALL ComposerFile.getInstance()
        file_a <- CALL cf.getFileMusik()
        music <- CREATE OBJECT Music(file_a)
        CALL music.play()
        startime <- CALL System.CurrentTimeMilis()
        WHILE isFinish == FALSE THEN
            IF CALL touch.getPointerID() > -1 THEN
                CASE CALL touch.getPointerID() OF
                    0 :
                        node <- "do" BREAK
                    1 :
                        node <- "re" BREAK
                    2 :
                        node <- "mi" BREAK
                    3 :
                        node <- "fa" BREAK
                    4 :
                        node <- "sol" BREAK
                    5 :
                        node <- "la" BREAK
                    6 :
                        node <- "si" BREAK
                    7 :
                        node <- "do2" BREAK
                ENDCASE
                elapse_time <- CALL System.CurrentTimeMilis() - start_time
                save_node <- "<node time=\""+elapse_time+"\"type=\""+type+"\" node=\""+node+"\" />"
                CALL nodelist.add(save_node)
                node <- ""
            ENDIF
            READ isFinish
        ENDWHILE
        CALL cf.setNode(nodelist)
        mcf <- CALL MakeComposerFile.getInstance()
        CALL mcf.MakeFile()
        END
```

Flowgraph



Sumber: Pengujian dan Analisis

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi executeAction() menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2 .

V(G) = E – N + 2

= 31 - 22 + 2 = 11

Dari nilai *cyclomatic complexity* yang telah dihasilkan dari perhitungan yaitu ditentukan empat buah basis set dari jalur independent yaitu:

Jalur 1: 1 – 2 – 22

Jalur 2: 1 – 2 – 3 – 20 – 2 – …

Jalur 3: 1 – 2 – 3 – 4 – 5 – 20 – 21 – 2 – …

Jalur 4: 1 – 2 – 3 – 4 – 6 – 7 – 20 – 21 – 2 – …

Jalur 5: 1 – 2 – 3 – 4 – 6 – 8 – 9 – 20 – 21 – 2 – …

Jalur 6: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 11 – 20 – 21 – 2 – …

Jalur 7: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 13 – 20 – 21 – 2 – …

Jalur 8: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 15 – 20 – 21 – 2 – …

Jalur 9: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 17 – 20 – 21 – 2 – …

Jalur 10: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 18 – 19 – 20 – 21 – 2 – …

Jalur 11: 1 – 2 – 3 – 4 – 6 – 8 – 10 – 12 – 14 – 16 – 18 – 20 – 21 – 2 – …

Tabel 6.30. Kasus uji untuk kode penangkapan aktivitas angklung

Jalur	Kasus Uji	Hasil yang diharapkan	Hasil yang didapatkan
1	Nilai isFinish sama dengan true	Sistem berhenti merekam input touch	Sistem berhenti merekam input touch
2	Nilai `touch.getPointerID()` bernilai sama dengan -1	Sistem tidak memroses input touch	Sistem tidak memroses input touch
3	Nilai `touch.getPointerID()` bernilai sama dengan 0	Menyimpan aktivitas penekanan angklung nada do	Menyimpan aktivitas penekanan angklung nada do
4	Nilai `touch.getPointerID()` bernilai sama dengan 1	Menyimpan aktivitas penekanan angklung nada re	Menyimpan aktivitas penekanan angklung nada re
5	Nilai `touch.getPointerID()` bernilai sama dengan 2	Menyimpan aktivitas penekanan angklung nada mi	Menyimpan aktivitas penekanan angklung nada mi
6	Nilai `touch.getPointerID()` bernilai sama dengan 3	Menyimpan aktivitas penekanan angklung nada fa	Menyimpan aktivitas penekanan angklung nada fa
7	Nilai `touch.getPointerID()` bernilai sama dengan 4	Menyimpan aktivitas penekanan angklung nada sol	Menyimpan aktivitas penekanan angklung nada sol
8	Nilai `touch.getPointerID()` bernilai sama dengan 5	Menyimpan aktivitas penekanan angklung nada la	Menyimpan aktivitas penekanan angklung nada la
9	Nilai `touch.getPointerID()` bernilai sama dengan 6	Menyimpan aktivitas penekanan angklung nada si	Menyimpan aktivitas penekanan angklung nada si
10	Nilai `touch.getPointerID()` bernilai sama dengan 7	Menyimpan aktivitas penekanan angklung nada do’	Menyimpan aktivitas penekanan angklung nada do’
11	Nilai `touch.getPointerID()` bernilai lebih dari 7	Tidak menyimpan nilai apapun	Tidak menyimpan nilai apapun

Sumber: Pengujian dan Analisis

6.1.3. Pengujian Validasi

Pengujian validasi digunakan untuk mengetahui apakah sistem yang dibangun sudah benar sesuai dengan yang dibutuhkan. Daftar kebutuhan dan hasil

analisis kebutuhan akan menjadi acuan untuk melakukan pengujian validasi. Pengujian validasi menggunakan metode pengujian *Black Box,* karena tidak diperlukan konsentrasi terhadap alur jalannya kode program dan lebih ditekankan untuk menemukan konformitas antara kinerja sistem dengan daftar kebutuhan. Pada skripsi ini dilakukan pengujian validasi terhadap perangkat lunak Permainan Sarinande Angklung.

6.1.3.1. Kasus uji memainkan *shake single player*

Pada Tabel 6.31. ditunjukkan kasus uji pemain memainkan shake single player pada permainan Sarinande angklung.

Tabel 6.31. Kasus uji untuk memainkan *shake single player*

Nama Kasus Uji	Kasus Uji memainkan *shake single player*
Objek Uji	Kebutuhan Fungsional (SRS_01)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memainkan *shake single player*
Prosedur Uji	1. Pemain menekan tombol *play* pada main menu 2. Pemain menekan tombol untuk bermain angklung dengan *accelerometer* 3. Pemain menekan tombol *single player* 4. Pemain menggerakkan *smartphone* searah sumbu x
Hasil yang Diharapkan	Aplikasi dapat mengeluarkan suara angklung dan *smartphone* bergetar ketika digerakkan searah sumbu x
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.2. Kasus uji mengganti nada pada *shake single player*

Pada Tabel 6.32. ditunjukkan kasus uji ketika pemain mengganti nada pada shake single player.

Tabel 6.32. Kasus uji untuk mengganti nada pada *shake single player*

Nama Kasus Uji	Kasus Uji mengganti nada pada *shake single player*

Objek Uji	Kebutuhan Fungsional (SRS_02)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memilih nada pada angklung satu nada.
Prosedur Uji	1. Pemain membuka *halaman shake single player* 2. Pemain menekan keypad menu 3. Pemain memilih nada angklung 4. Pemain menggerakkan *smartphone* searah sumbu x
Hasil yang Diharapkan	Aplikasi dapat mengeluarkan suara angklung yang dipilih oleh pemain ketika digerakkan searah sumbu x
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.3. Kasus uji memilih *file beat*

Pada Tabel 6.33. ditunjukkan kasus uji ketika pemain memilih *file beat* pada permainan Sarinande angklung.

Tabel 6.33. Kasus uji untuk memilih *file beat*

Nama Kasus Uji	Kasus Uji memilih *file beat*
Objek Uji	Kebutuhan Fungsional (SRS_03)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memilih *file beat*.
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *touch* 3. Pemain memilih *file beat* dari semua file *beat* yang berada SD card
Hasil yang Diharapkan	Aplikasi dapat menampilkan semua *file beat* di sdcard dan menyimpan file yang dipilih untuk dimainkan pada fitur yang dipilih
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.4. Kasus uji menjalankan *demo beat*

Pada Tabel 6.34. ditunjukkan kasus uji ketika pemain menjalankan demo *beat* pada permainan Sarinande angklung.

Tabel 6.34. Kasus uji untuk menjalankan *demo beat*

Nama Kasus Uji	Kasus Uji menjalankan *demo beat*
Objek Uji	Kebutuhan Fungsional (SRS_04)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk menjalankan *demo beat*
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *touch* 3. Pemain memilih *file beat* dari semua file *beat* yang berada SD card 4. Pemain menekan tombol *record* pada touch menu 5. Pemain menekan keypad menu 6. Pemain menekan tombol play
Hasil yang Diharapkan	Aplikasi dapat menampilkan demo *file beat* yang dipilih. File beat akan dijalankan dan dimainkan secara otomatis.
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.5. Kasus uji membuat *file audio* dari *file beat*

Pada Tabel 6.35. ditunjukkan kasus uji ketika membuat *file audio* dari *file beat* pada permainan Sarinande angklung.

Tabel 6.35. Kasus uji membuat *file audio* dari *file beat*

Nama Kasus Uji	Kasus Uji membuat *file audio* dari *file beat*
Objek Uji	Kebutuhan Fungsional (SRS_05)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk mengonversikan *file beat* menjadi *file audio*
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *touch* 3. Pemain memilih *file beat* dari semua file *beat* yang berada SD card 4. Pemain menekan tombol *record* pada touch menu 5. Pemain menekan keypad menu 6. Pemain menekan tombol *record and save in sd card*

Hasil yang Diharapkan	Aplikasi dapat merekam permainan *file beat* dan menyimpannya pada sdcard/sarinande
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.6. Kasus uji memainkan *touch game*

Pada Tabel 6.36. ditunjukkan kasus uji ketika pemain memainkan *touch game* pada permainan Sarinande angklung.

Tabel 6.36. Kasus uji untuk memainkan *touch game*

Nama Kasus Uji	Kasus uji memainkan *touch game*
Objek Uji	Kebutuhan Fungsional (SRS_06)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memainkan *touch game*
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *touch* 3. Pemain memilih *file beat* dari semua file *beat* yang berada SD card 4. Pemain menekan tombol *game* pada touch menu 5. Pemain menyentuh angklung sesuai dengan kotak beat yang muncul
Hasil yang Diharapkan	Aplikasi dapat menampilkan proses memainkan file beat menggunakan input touch
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.7. Kasus uji melihat *score touch game*

Pada Tabel 6.37. ditunjukkan kasus uji ketika pemain melihat *score touch game* pada permainan Sarinande angklung.

Tabel 6.37. Kasus uji untuk melihat *score touch game*

Nama Kasus Uji	Kasus uji melihat *score touch game*
Objek Uji	Kebutuhan Fungsional (SRS_07)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat *score touch game*

Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *touch* 3. Pemain memilih *file beat* dari semua file *beat* yang berada SD card 4. Pemain menekan tombol *game* pada touch menu 5. Pemain menyentuh angklung sesuai dengan kotak beat yang muncul 6. Ketika permainan selesai, pemain akan melihat hasil *score* yang diperoleh
Hasil yang Diharapkan	Aplikasi dapat menampilkan *score* yang diperoleh setelah memainkan *touch game*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.8. Kasus uji memilih peran *multiplayer*

Pada Tabel 6.38. ditunjukkan kasus uji ketika pemain memilih peran *multiplayer* pada permainan Sarinande angklung.

Tabel 6.38. Kasus uji untuk memilih peran *multiplayer*

Nama Kasus Uji	Kasus Uji memilih peran *multiplayer*
Objek Uji	Kebutuhan Fungsional (SRS_08)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memilih peran *multiplayer*
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *shake* 3. Pemain menekan tombol *multiplayer* 4. Pemain menekan tombol server untuk bermain sebagai *server* atau join *multiplayer* untuk bermain sebagai *client*
Hasil yang Diharapkan	Aplikasi dapat menampilkan *menu* untuk memilih peran multiplayer. Ketika berperan sebagai server, aplikasi akan mengarahkan ke halaman konfigurasi dan ketika berperan sebagai client, aplikasi akan memunculkan kotak dialog pengoneksian ke server
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.9. Kasus uji pengonfigurasian *multiplayer*

Pada Tabel 6.39. ditunjukkan kasus uji ketika pemain melakukan pengonfigurasian *multiplayer* pada permainan Sarinande angklung.

Tabel 6.39. Kasus uji untuk pengonfigurasian *multiplayer*

Nama Kasus Uji	Kasus uji pengonfigurasian *multiplayer*
Objek Uji	Kebutuhan Fungsional (SRS_09)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melakukan konfigurasi *multiplayer*
Prosedur Uji	1. Pemain menekan tombol *play* 2. Pemain menekan tombol bermain angklung dengan *shake* 3. Pemain menekan tombol *multiplayer* 4. Pemain menekan tombol server 5. Pemain server memilih *file beat* 6. Pemain server akan menunggu *client* yang akan terkoneksi 7. Pemain server memilih nada yang akan dimainkan oleh pemain *client* 8. Pemain server mengirimkan judul file beat dan nada yang akan dimainkan *client* 9. Pemain menekan *keypad menu* 10. Pemain menekan tombol *start*
Hasil yang Diharapkan	Aplikasi dapat mengirimkan judul file beat dan nada yang akan dimainkan oleh setiap *client*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.10. Kasus uji memainkan *shake multiplayer*

Pada Tabel 6.40. ditunjukkan *kasus uji* ketika pemain memainkan *shake multiplayer* pada permainan Sarinande angklung.

Tabel 6.40. Kasus uji memainkan *shake multiplayer*

Nama Kasus Uji	Kasus uji memainkan *shake multiplayer*
Objek Uji	Kebutuhan Fungsional (SRS_10)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memainkan *shake multiplayer*

Prosedur Uji	1. Pemain membuka halaman permainan *multiplayer* 2. Pemain *server* mengirimkan pesan secara broadcast ke semua pemain untuk mulai bermain 3. Pemain menggerakkan *smartphone* searah sumbu x ketika kotak beat berada pada area hit
Hasil yang Diharapkan	Aplikasi *client server* dapat bermain secara bersamaan ketika pesan dikirim. Serta aplikasi dapat menampilkan proses memainkan file beat menggunakan input *accelerometer*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.11. Kasus uji melihat hasil *score multiplayer*

Pada Tabel 6.41. ditunjukkan kasus uji ketika pemain melihat hasil *score multiplayer* pada permainan Sarinande angklung.

Tabel 6.41. Kasus uji untuk melihat hasil *score multiplayer*

Nama Kasus Uji	Kasus uji melihat hasil *score multiplayer*
Objek Uji	Kebutuhan Fungsional (SRS_11)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat hasil *score multiplayer*
Prosedur Uji	1. Pemain memainkan shake *multiplayer* 2. Pemain melihat hasil *score* yang didapatkan dari permainan
Hasil yang Diharapkan	Aplikasi dapat menampilkan *score* setelah memainkan *file beat* pada *shake multiplayer*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.12. Kasus uji memilih *background* musik

Pada Tabel 6.42. ditunjukkan kasus uji ketika pemain memilih *background* musik di *composer* pada permainan Sarinande angklung.

Tabel 6.42. Kasus uji untuk memilih *background* musik

Nama Kasus Uji	Kasus uji memilih *background* musik
Objek Uji	Kebutuhan Fungsional (SRS_12)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk memilih *background* music yang akan dipakai pada *composer*
Prosedur Uji	1. Pemain menekan tombol *composer* pada *main menu* 2. Pemain memilih musik pada daftar yang telah ditampilkan
Hasil yang Diharapkan	Aplikasi dapat menampilkan daftar musik yang terletak pada sdcard/sarinande/ composer dan menyimpan *file* musik yang dipilih
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.13. Kasus uji membuat *file beat*

Pada Tabel 6.43. ditunjukkan kasus uji ketika pemain membuat *file beat* pada permainan Sarinande angklung.

Tabel 6.43. Kasus uji untuk membuat *file beat*

Nama Kasus Uji	Kasus uji membuat *file beat*
Objek Uji	Kebutuhan Fungsional (SRS_13)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk membuat *file beat*
Prosedur Uji	1. Pemain menekan tombol *composer* pada *main menu* 2. Pemain memilih musik pada daftar yang telah ditampilkan 3. Pemain menekan *keypad menu* 4. Pemain menekan tombol yes untuk memulai 5. Pemain menyentuh angklung sesuai dengan nada yang dikomposisikan 6. Pemain menekan *keypad menu* 7. Pemain menekan tombol yes untuk menyimpan 8. Pemain memasukkan author dan title pada form 9. Pemain menekan tombol save untuk

	menyimpan *file beat*
Hasil yang Diharapkan	Aplikasi dapat membuat file beat yang kemudian akan disimpan pada folder di sdcard/sarinande dan *background music* yang digunakan akan di-copy ke folder tersebut
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.14. Kasus uji melihat *menu about*

Pada Tabel 6.44. ditunjukkan kasus uji ketika pemain melihat *menu about* pada permainan Sarinande angklung.

Tabel 6.44. Kasus uji untuk melihat *menu about*

Nama Kasus Uji	Kasus uji melihat *menu about*
Objek Uji	Kebutuhan Fungsional (SRS_14)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat *menu about*
Prosedur Uji	1. Pemain menekan tombol *about* pada *main menu* 2. Pemain menekan tab *credit* 3. Pemain menekan tab *about*
Hasil yang Diharapkan	Aplikasi menampilkan halaman about dan menampilkan informasi mengenai angklung dan kredit pembuat aplikasi
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.15. Kasus uji melihat *menu setting*

Pada Tabel 6.45. ditunjukkan kasus uji ketika pemain melihat *menu setting* pada permainan Sarinande angklung.

Tabel 6.45. Kasus uji untuk melihat *menu setting*

Nama Kasus Uji	Kasus uji melihat *menu setting*
Objek Uji	Kebutuhan Fungsional (SRS_15)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat *menu setting*

Prosedur Uji	1. Pemain menekan tombol berbentuk *gear* pada *main menu* 2. Pemain merubah volume suara 3. Pemain merubah volume musik 4. Pemain mengaktifkan vibrasi
Hasil yang Diharapkan	Aplikasi menampilkan halaman *setting* dan dapat mengatur konfigurasi *setting*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.16. Kasus uji melihat *menu highscore*

Pada Tabel 6.46. ditunjukkan kasus uji ketika pemain melihat *menu setting* pada permainan Sarinande angklung.

Tabel 6.46. Kasus uji untuk melihat menu *high score*

Nama Kasus Uji	Kasus Uji melihat *menu highscore*
Objek Uji	Kebutuhan Fungsional (SRS_16)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat *menu highscore*
Prosedur Uji	1. Pemain menekan tombol *highscore* pada *main menu*
Hasil yang Diharapkan	Aplikasi menampilkan halaman nilai *high score*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.3.17. Kasus uji melihat *menu how to play*

Pada Tabel 6.47. ditunjukkan kasus uji ketika pemain melihat *menu how to play* pada permainan Sarinande angklung.

Tabel 6.47. Kasus uji untuk melihat *menu how to play*

Nama Kasus Uji	Kasus Uji melihat *menu how to play*
Objek Uji	Kebutuhan Fungsional (SRS_17)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional untuk melihat *menu how to play*
Prosedur Uji	1. Pemain menekan tombol dengan tanda tanya pada *main menu* 2. Pemain menekan tombol angka pada

	layar
Hasil yang Diharapkan	Aplikasi menampilkan halaman *how to play*
Hasil yang Didapatkan	Valid

Sumber: Pengujian dan Analisis

6.1.4. Pengujian Peforma

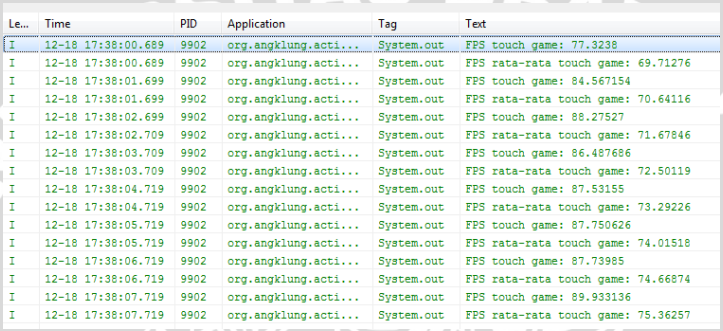
Pengujian peforma permainan Sarinande Angklung dilakukan dengan melihat FPS (*Frame Per Second*) yang dihasilkan oleh *game*. Pengujian FPS pada permainan Sarinande Angklung dilakukan dengan menggunakan class FPSCounter yang telah disediakan oleh *AndEngine*. pengujian peforma dilakukan dengan melihat fps yang dihasilkan pada setiap halaman. Adapun halaman yang digunakan dalam pengujian ini adalah main menu, halaman permainan *single player*, halaman permainan *multiplayer*, *composer*, *touch game*, dan *record*. Pengujian dilakukan dengan menjalankan aplikasi pada beberapa *smartphone* dengan spesifikasi yang berbeda. Hasil pengujian peforma *game* Sarinande Angklung ditunjukkan pada Tabel 6.33. Hasil penangkapan fps pada permainan Sarinande angklung ditunjukkan pada Gambar 6.14.

Tabel 6.48. Hasil pengujian peforma dengan melihat FPS

NO	SMARTPHONE	FITUR					
		COMPOSER	MAIN MENU	MULTIPLAYER	SINGLE PLAYER	RECORD	TOUCH GAME
1	GALAXY ACE GT-S5830 ANDROID 2.3.4 CPU: 800 MHz ARM 11	83.629	84.20	82.28	82.51	80.85	77.63
2	GALAXY TAB GT-P1000 ANDROID 2.3.3 CPU: 1 GHZ Cortex A8	56.44	58.94	57.98	58.65	59.11	57.57
3	GALAXY WONDER GT I8150 ANDROID 2.3.6 CPU: 1.4 GHz Scorpion	54.56	55.71	51.00	54.74	54.30	54.50
4	GALAXY S2 GT I9100	58.83	59.80	59.86	58.8	59.63	59.25

	ANDROID 4.0.3 CPU: Dual-core 1.2 GHz Cortex-A9						
5	GALAXY MINI GT-S5570 ANDROID 2.3.4 CPU 600 MHz ARMv6	91.1	92.9	89.05	92.4	73.4	89.7
MIN		54.56	55.71	51.00	54.74	54.30	54.30
MAX		91.1	92.9	89.05	92.4	80.85	89.7
RATA-RATA		83.629	70.31	68.31	69.42	65.45	67.73

Sumber: Pengujian dan Analisis



Le...	Time	PID	Application	Tag	Text
I	12-18 17:38:00.689	9902	org.angklung.acti...	System.out	FPS touch game: 77.3238
I	12-18 17:38:00.689	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 69.71276
I	12-18 17:38:01.699	9902	org.angklung.acti...	System.out	FPS touch game: 84.567154
I	12-18 17:38:01.699	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 70.64116
I	12-18 17:38:02.699	9902	org.angklung.acti...	System.out	FPS touch game: 88.27527
I	12-18 17:38:02.709	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 71.67846
I	12-18 17:38:03.709	9902	org.angklung.acti...	System.out	FPS touch game: 86.487686
I	12-18 17:38:03.709	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 72.50119
I	12-18 17:38:04.719	9902	org.angklung.acti...	System.out	FPS touch game: 87.53155
I	12-18 17:38:04.719	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 73.29226
I	12-18 17:38:05.719	9902	org.angklung.acti...	System.out	FPS touch game: 87.750626
I	12-18 17:38:05.719	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 74.01518
I	12-18 17:38:06.719	9902	org.angklung.acti...	System.out	FPS touch game: 87.73985
I	12-18 17:38:06.719	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 74.66874
I	12-18 17:38:07.719	9902	org.angklung.acti...	System.out	FPS touch game: 89.933136
I	12-18 17:38:07.719	9902	org.angklung.acti...	System.out	FPS rata-rata touch game: 75.36257

Gambar 6.1. Gambar hasil *capture* FPS pada halaman *touch game*

Sumber: Pengujian dan Analisis

6.2. Analisis

Analisis bertujuan untuk mendapatkan kesimpulan dari hasil pengujian permainan Sarinande Angklung yang telah dilakukan. Analisis dilakukan terhadap hasil pengujian di setiap tahap pengujian. Proses analisis yang dilakukan meliputi analisis hasil pengujian unit, analisis hasil pengujian integrasi, analisis hasil pengujian validasi, dan analisis hasil pengujian peforma.

6.2.1. Analisis Pengujian Unit

Hasil analisa yang didapatkan dari pengujian unit yaitu:

1. Berdasarkan kesesuaian antara hasil pengujian tiap unit dengan output unit yang diharapkan pada *game* Sarinande Angklung, maka dapat diambil kesimpulan bahwa unit modul dari program sudah sesuai dengan output yang diharapkan.

2. Berdasarkan hasil perhitungan *cyclomatic complexity* dari tiap *flow grap* kode unit, kode unit yang menghasilkan *independent path* terbanyak adalah pada kode pendeteksian *file beat* sejumlah sebelas jalur *independent*. Jumlah *path* kode pendeteksian *file beat* sebagian besar dipengaruhi dari logika perulangan dan seleksi kondisi, karena kode ini berisi proses untuk pencarian *file beat* dengan tipe data .ANG pada sdcard.

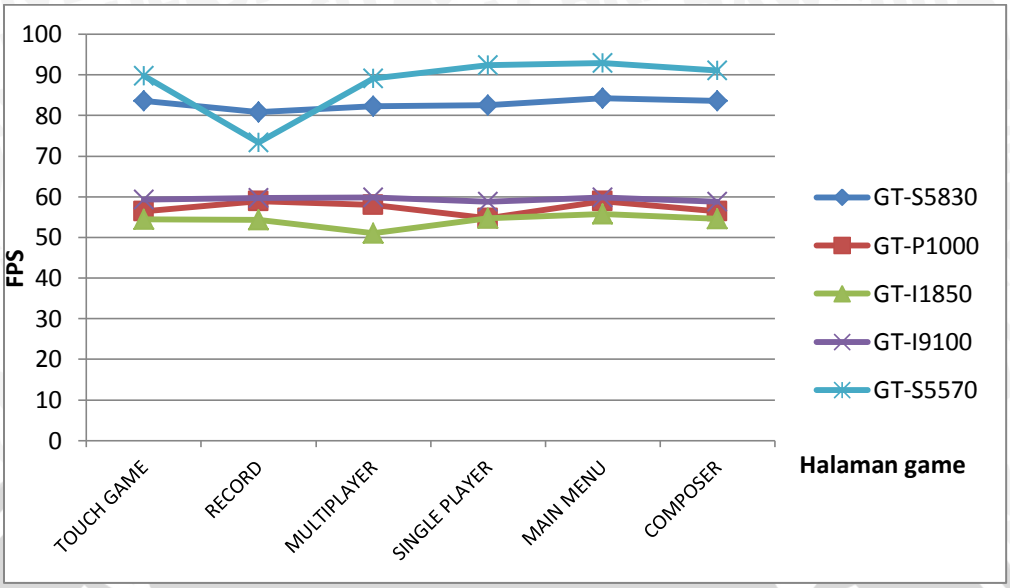
6.2.2. Analisis Pengujian Integrasi

Hasil analisa yang didapatkan dari pengujian integrasi yaitu:

1. Berdasarkan kesesuaian antara hasil pengujian tiap integrasi dengan output integrasi yang diharapkan pada *game* Sarinande Angklung, maka dapat diambil kesimpulan bahwa unit modul dari program sudah sesuai dengan output yang diharapkan.
2. Berdasarkan hasil perhitungan *cyclomatic complexity* dari tiap *flow grap* kode integrasi, kode integrasi yang menghasilkan *independent path* terbanyak adalah pada kode pengomposisian *file beat* sejumlah sebelas jalur *independent*. Jumlah *path* kode pendeteksian *file beat* sebagian besar dipengaruhi dari logika perulangan dan seleksi kondisi, karena kode ini berisi proses untuk menangkap semua aktivitas angklung yang dibunyikan ketika memebuat *file beat*.

6.2.3. Analisis Pengujian Validasi

Proses analisis terhadap hasil pengujian validasi dilakukan dengan melihat kesesuaian antara hasil uji terhadap implementasi dan fungsionalitas *game* Sarinande Angklung dengan hasil yang diharapkan dalam daftar kebutuhan *game* Sarinande Angklung. Dengan demikian dapat disimpulkan bahwa implementasi dan fungsionalitas permainan Sarinande Angklung telah memenuhi kebutuhan yang telah dijabarkan pada daftar kebutuhan.



Gambar 6.2. Gambar grafik hasil pengujian peforma

Sumber: Pengujian dan Analisis

6.2.4. Analisis Pengujian Peforma

Proses analisis terhadap hasil pengujian peforma dilakukan dengan melihat hubungan antara fps yang dihasilkan dengan peforma tampilan yang dihasilkan pada halaman *game*. Berdasarkan hasil pengujian peforma yang telah dilakukan pada lima *smartphone* yang berbeda, nilai fps yang didapatkan berkisar antara 50 fps sampai dengan 92.9 fps. Grafik hasil pengujian peforma ditunjukkan pada Gambar 6.2.

Berdasarkan grafik hasil pengujian peforma pada Gambar 6.2. didapatkan hasil analisa sebagai berikut.

1. Untuk pengujian peforma pada smartphone GT-P1000, GT-I850, GT-I9100 didapatkan fps dengan rentang 50 – 60 fps. Sedangkan pada smartphone dengan tipe GT-S5830 dan GT-S5570 didapatkan rentang fps antara 75 – 93 fps. Semakin besar nilai prosesor yang dimiliki oleh *smartphone* seharusnya dapat menghasilkan fps yang tinggi. Namun, pada hasil pengujian ini terdapat anomali atau keanehan. Dimana smartphone pada nomor 1 dan 5 yang memiliki spesifikasi lebih rendah daripada smartphone no 2, 3, dan 4 mendapatkan nilai fps yang lebih tinggi. Hal itu

disebabkan karena adanya pembatasan peforma GPU pada *smartphone android* seri terbaru (hanya pada vendor tertentu). Hal tersebut dilakukan untuk meminimalisir terjadinya *overheat* ketika menjalankan sebuah game pada fps di atas 60.

2. Berdasarkan hasil pengujian peforma didapatkan nilai fps terendah yaitu sebesar 51 fps pada *smartphone* nomor 3. Dimana nilai tersebut telah memenuhi standar minimal fps yaitu sebesar 30 fps.
3. Berdasarkan analisis pengujian peforma dari nomor 1, pengambilan nilai fps terendah pada game Simulator Angklung tidak dapat dijadikan patokan kelayakan pada *game* karena adanya pembatasan peforma GPU pada *smartphone android*. Oleh karena itu spesifikasi minimal untuk menjalankan *game* Sarinande Angklung diambil dari pengujian yang menghasilkan fps di atas 30 dengan spesifikasi rendah. Sehingga spesifikasi minimal *smartphone* yang disarankan untuk menjalankan *game* Sarinande Angklung adalah Samsung Galaxy Mini GT-S5770 dengan spesifikasi
    a. CPU: 600 MHz ARMv6
    b. GPU: Adreno 200