

SKRIPSI

**ALGORITMA DETEKSI GERAKAN KEPALA DENGAN
GOOGLE CARDBOARD**



EGA PRIANTO

NPM: 2013730047

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2017**

UNDERGRADUATE THESIS

**HEAD MOTION DETECTOR ALGORITHM USING GOOGLE
CARDBOARD**



EGA PRIANTO

NPM: 2013730047

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY**

2017

LEMBAR PENGESAHAN

ALGORITMA DETEKSI GERAKAN KEPALA DENGAN GOOGLE CARDBOARD

EGA PRIANTO

NPM: 2013730047

Bandung, 30 Mei 2017

Menyetujui,

Pembimbing Tunggal

Dr. Veronica Sri Moertini

Ketua Tim Penguji

Anggota Tim Penguji

Aditya Bagoes Saputra, M.T.

Joanna Helga, M.Sc.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

ALGORITMA DETEKSI GERAKAN KEPALA DENGAN GOOGLE CARDBOARD

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 30 Mei 2017

Meterai

Ega Prianto
NPM: 2013730047

ABSTRAK

Virtual Reality adalah salah satu interaksi antara manusia dengan komputer yang membuat penggunanya merasakan berada pada suatu lingkungan buatan yang berada pada komputer. Banyak cara untuk dapat merasakan pengalaman *Virtual Reality* salah satunya adalah dengan menggunakan Google Cardboard. Google Cardboard merupakan gawai murah yang terbuat dari kardus untuk mengalami pengalaman *Virtual Reality* pada perangkat *smartphone* Android atau IOS. Google Cardboard dapat memberikan pengalaman *Virtual Reality* karena menggunakan sensor-sensor yang terdapat pada perangkat *smartphone*.

Masukkan pada Google Cardboard sangatlah terbatas. Masukkan tersebut adalah dengan menarik atau menekan tombol yang ada pada kacamata Google Cardboard dan orientasi dari perangkat *smartphone* saja. Tombol pada Google Cardboard ini pun terkadang tidak berfungsi dengan baik. Oleh karena itu dibuatlah aplikasi untuk mendeteksi gerakan kepala, khususnya mengangguk dengan menggeleng.

Pengujian dilakukan dengan mencoba menjalankan aplikasi yang telah dibuat pada beberapa perangkat dengan spesifikasi yang berbeda-beda. Selain itu pengujian juga dilakukan dengan mengajak beberapa orang untuk menggunakan aplikasi ini. Berdasarkan hasil pengujian, aplikasi ini dapat berjalan dengan baik pada perangkat *smartphone* dengan spesifikasi tertentu. Aplikasi ini juga sudah dapat membaca gerakan kepala yang sesuai yang dilakukan oleh pengguna, tetapi terkadang simpangan yang di tetapkan pada aplikasi ini tidak sesuai dengan kriteria masing-masing pengguna. Jadi pengguna harus menyesuaikan gerakan kepalamya agar sesuai dengan batasan simpangan yang telah ditetapkan pada aplikasi ini.

Kata-kata kunci: Virtual Reality, Google Cardboard, Sensor Gyroscope, Google VR, Android, Unity

ABSTRACT

Virtual Reality is one of the interactions between humans and computers that make its users feel in the artificial environment on a computer. There are a lot of ways to experience the Virtual Reality, one of them is using Google Cardboard. Google Cardboard is a low-cost system to encourage interest and development in Virtual Reality applications on a Android or IOS smartphone. Google Cardboard use sensors on smartphone devices to perform Virtual Reality.

Input on Google Cardboard is very limited. The only input on the Google Cardboard is to pull or press the button on the Google Cardboard. Sometimes the button doesn't work very well. Therefore this application is made to detect the head motion, especially nodding and shaking.

Testing done by trying to run this application into several devices with different specifications. In addition, testing also done by inviting some people to use this application. Based on the results of testing, this application runs well on smartphone devices with certain specifications. This application also able to read the appropriate head movement performed by the user, But sometimes the deviation set in this app does not match the criteria of each user. So the user must adjust his head movement to fit the deviation limit set in this application.

Keywords: Virtual Reality, Google Cardboard, Sensor Gyroscope, Google VR, Android, Unity

Teknik Informatika UNPAR dan diri sendiri

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas berkat yang diberikan kepada penulis sehingga dapat menyelesaikan tugas akhir dengan judul **Algoritma Deteksi Gerakan Kepala dengan Google Cardboard** dengan baik dan tepat waktu. Penulis juga berterima kasih kepada pihak-pihak yang telah memberikan dukungan dan bantuan kepada penulis dalam menyelesaikan tugas akhir ini, yaitu:

1. Keluarga yang selalu memberikan dukungan dan semangat kepada penulis.
2. Bapak Pascal Alfadian sebagai dosen pembimbing yang telah membimbing penulis hingga dapat menyelesaikan tugas akhir ini.
3. Bapak Aditya Bagoes Saputra dan Ibu Joanna Helga sebagai dosen penguji yang telah membantu dalam menguji dan memperbaiki tugas akhir ini.
4. Seluruh penguji aplikasi yang telah membantu untuk kelancaran dalam mengerjakan skripsi ini.
5. Teman-teman *admin* lab yang selalu menyediakan lab skripsi agar dapat mengerjakan skripsi dengan nyaman.
6. Bapak Kristopher David harjono karena telah membantu mengarahkan dalam mengajukan topik skripsi dan memberikan pengetahuan dasar tentang topik skripsi yang penulis ajukan.
7. Ricky Setiawan, Kevin Antonius, Hasudungan Dimas Nathanael Silalahi, Fahrizal, Fadhil Ahsan, Jacinta Delora, Antonius Kurnia, Devi Handevi, Devina Emily dan Jonathan Surya sebagai teman seperjuangan dan bertukar pikiran sekaligus telah memberikan semangat kepada penulis.
8. Teman-teman Teknik Informatika UNPAR angkatan 2013 yang telah berbagi ilmu kepada penulis.
9. Pihak-pihak lain yang belum disebutkan, yang berperan dalam penyelesaian tugas akhir ini.

Akhir kata, penulis berharap agar tugas akhir ini dapat bermanfaat bagi pembaca yang hendak melakukan penelitian dan pengembangan yang terkait dengan tugas akhir ini.

Bandung, Mei 2017

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metode Penelitian	2
1.6 Sistematika Penulisan	3
2 DASAR TEORI	5
2.1 Android SDK	5
2.1.1 Struktur File Android Studio Project	5
2.1.2 Membuat User Interface	5
2.1.3 Activity	8
2.1.4 Android Sensor Framework	10
2.2 Google VR SDK	18
2.2.1 API Audio	20
2.2.2 API Base	20
2.3 Teori Kuaternion	22
2.3.1 Struktur Ajabar	22
2.3.2 <i>Ajabar Kuaternion</i> dan Operasi-operasi pada Kuaternion	25
2.4 Unity Game Engine	27
2.4.1 Struktur Hierarki GameObject	28
2.4.2 Prefabs	28
2.4.3 Scene	28
2.4.4 GameObject	29
2.4.5 Scripting	29
2.4.6 Transformasi Rotasi dan Orientasi	32
2.5 Google VR SDK for Unity	33
3 ANALISIS	35
3.1 Analisis Aplikasi Sejenis	35
3.2 Perekaman Data Sensor	37
3.2.1 Perekaman Grafik Sensor <i>Accelerometer</i>	39
3.2.2 Perekaman Grafik Sensor <i>Gyroscope</i>	41
3.2.3 Perekaman Grafik Sensor <i>Rotation Vector</i>	42

3.3	Analisis Data Sensor untuk Mendeteksi Gerakan Kepala	45
3.4	Analisis Metode Pendekripsi Gerakan Kepala	46
3.4.1	Algoritma Mendekripsi Gerakan Mengangguk	46
3.4.2	Algoritma Mendekripsi Gerakan Menggeleng	49
3.5	Analisis Aplikasi Permainan dengan Algoritma Deteksi Gerakan Kepala	51
4	PERANCANGAN	55
4.1	Perancangan Kelas Algoritma Pendekripsi Gerakan Kepala	55
4.2	Perancangan Hierarki GameObject pada Unity	60
4.3	Perancangan <i>Gameplay</i> dan Perancangan Antarmuka	62
5	IMPLEMENTASI DAN PENGUJIAN	65
5.1	Implementasi	65
5.1.1	Lingkungan Implementasi	65
5.1.2	Hasil Implementasi	65
5.2	Pengujian	68
5.2.1	Pengujian Fungsional	68
5.2.2	Pengujian Eksperimental	70
5.3	Masalah yang Dihadapi pada Saat Implementasi	76
6	KESIMPULAN DAN SARAN	77
6.1	Kesimpulan	77
6.2	Saran	77
DAFTAR REFERENSI		79
A KODE PROGRAM APLIKASI UNTUK ANALISIS		81
B KODE PROGRAM APLIKASI <i>Game</i>		87

DAFTAR GAMBAR

2.1	Tampilan struktur folder pada <i>project</i> Android Studio	6
2.2	Ilustrasi bagaimana percabangan objek ViewGroup pada <i>layout</i> dan mengandung objek View lainnya.	7
2.3	<i>State diagram</i> siklus Activity	9
2.4	Sistem koordinat (relatif dengan perangkatnya) yang digunakan oleh Sensor API	12
2.5	Sistem koordinat sensor rotasi vektor terhadap Bumi	19
2.6	Contoh perputaran dengan bilangan kompleks	24
2.7	Contoh perputaran tiga puluh derajat dengan bilangan kompleks	25
2.8	Right-hand rule dalam <i>cross product</i> vektor	26
2.9	Contoh Hierarchy Parenting.	28
2.10	Contoh <i>scene</i> kosong.	29
2.11	Contoh penggunaan komponen pada GameObject.	30
2.12	Contoh komponen pada GameObject kubus.	30
2.13	Contoh Component <i>script</i>	31
2.14	Flowchart urutan pemanggilan fungsi-fungsi.	32
2.15	Rotasi dari Game Object ditampilkan dan di-edit pada <i>Inspector</i> dengan menggunakan metode sudut Euler, tetapi disimpan dalam memori dengan menggunakan metode Quaternion.	34
3.1	<i>Screenshot</i> task yang diberikan aplikasi InMind VR.	36
3.2	<i>Screenshot</i> aplikasi InMind VR ketika permainan berlangsung.	36
3.3	<i>Screenshot</i> aplikasi InMind VR ketika pengguna untuk mengangguk jika telah siap.	38
3.4	Gambar untuk mendeskripsikan posisi dari muka sebelum melakukan gerakan menggeleng atau mengangguk. Gambar (a) adalah kondisi muka ketika sedang menghadap ke depan. Gambar (b) adalah kondisi muka ketika sedang menghadap ke atas. Gambar(c) adalah kondisi muka ketika sedang menghadap ke serong kiri atas.	39
3.5	Grafik nilai kuaternion dari sensor <i>accelerometer</i> ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.	40
3.6	Grafik nilai kuaternion dari sensor <i>accelerometer</i> ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.	40
3.7	Gambar grafik nilai sensor <i>gyroscope</i> ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.	41
3.8	Gambar grafik nilai sensor <i>gyroscope</i> ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.	42
3.9	Grafik nilai kuaternion dari sensor <i>rotation vector</i> ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.	43

3.10	Grafik nilai kuaternion dari sensor <i>rotation vector</i> ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas	44
3.11	Dua buah rotasi yang identik dengan nilai kuaternion yang berbeda.	44
3.12	Grafik pada saat pengguna menggeleng.	46
3.13	Grafik pada saat pengguna mengangguk.	46
3.14	Diagram <i>use case</i> aplikasi permainan deteksi gerakan kepala	51
3.15	<i>Screenshot</i> task yang diberikan aplikasi InMind VR.	54
4.1	Diagram Kelas Algoritma Pendekripsi Gerakan Kepala. (Gambar diputar sembilan puluh derajat searah jarum jam)	56
4.2	Perancangan hierarki Game Object	61
4.3	Meteran pada permainan.	62
4.4	Desain ruangan untuk permainan yang dibuat.	63
4.5	Desain User Interface	64
5.1	Tampilan ketika aplikasi baru dimulai.	66
5.2	Tampilan ketika perangkat tidak memiliki sensor Gyroscope.	66
5.3	Tampilan ketika notifikasi muncul.	67
5.4	Tampilan ketika <i>pop-up</i> pertanyaan muncul.	67
5.5	Tampilan setelah pengguna mengangguk.	68
5.6	Tampilan setelah pengguna menggeleng.	69
5.7	Tampilan setelah permainan usai.	69
5.8	Pengujian oleh Ricky Setiawan.	71
5.9	Pengujian oleh Harseto Pandityo.	71
5.10	Pengujian oleh Herfan Heryandi.	72
5.11	Pengujian oleh Kevin Antonius.	72
5.12	Pengujian oleh Antonius Kurnia.	73
5.13	Grafik hasil <i>survey</i> pengujian pernyataan pertama.	73
5.14	Grafik hasil <i>survey</i> pengujian pernyataan kedua.	74
5.15	Grafik hasil <i>survey</i> pengujian pernyataan ketiga.	74
5.16	Grafik hasil <i>survey</i> pengujian pernyataan keempat.	74
5.17	Grafik hasil <i>survey</i> pengujian pernyataan kelima.	75
5.18	Grafik rata-rata nilai dari hasil <i>survey</i> yang dilakukan.	75

DAFTAR TABEL

2.1	Tipe-tipe Sensor pada Android	11
3.1	Tabel skenario memulai permainan.	52
3.2	Tabel skenario membuka <i>pop-up</i> pesan.	52
3.3	Tabel skenario memberikan respons <i>pop-up</i> dengan menggeleng atau mengangguk. .	52
3.4	Tabel skenario melihat kondisi permainan.	53
3.5	Tabel skenario mengulang permainan.	53
4.1	Tabel Aset-aset yang digunakan.	63
5.1	Tabel Perangkat yang digunakan	70

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Virtual Reality(VR) adalah teknologi yang mampu membuat penggunanya dapat berinteraksi dengan lingkungan buatan oleh komputer, suatu lingkungan yang sebenarnya ditiru atau hanya ada di dalam imajinasi.^[1] VR membuat pengalaman sensorik, di antaranya penglihatan, pendengaran, perabaan, dan penciuman secara buatan.^[2] Gawai VR terbaru menggunakan metode *head-mounted display*, Google Cardboard salah satunya. *Head-mounted display* adalah menempatkan layar di kepala, sehingga pengguna hanya dapat melihat tampilan yang ditampilkan oleh layar.^[3]

Google Cardboard^[4] adalah gawai murah yang terbuat dari kardus untuk dapat merasakan pengalaman VR dengan *smartphone* Android atau iOS. Kita dapat membuat Google Cardboard kita sendiri secara gratis dengan mengunduh *template*-nya di situs Google Cardboard. *Template* tersebut membantu dalam merakit kardus dengan dibentuk, dilipat dan dipotong sedemikian rupa sehingga berbentuk kacamata. Bahan-bahan untuk merakit Google Cardboard hanyalah kardus, lem, dan lensa dengan spesifikasi tertentu. Ada pula beberapa perusahaan yang membuat gawai ini dengan kualitas yang lebih baik dari pada kardus seperti plastik. Perusahaan-perusahaan tersebut diantaranya BoboVR, ShineconVR, VRBox, dan lain-lain.

Pada gawai Google Cardboard cara pengguna memberikan masukan kepada program sangatlah terbatas. Cara tersebut hanyalah dengan gerakan kepala dan tombol pada Google Cardboard. Tombol ini pun terkadang tidak berfungsi dengan baik. Cara lainnya agar dapat memberikan masukan kepada program adalah dengan menghubungkan *smartphone* yang digunakan dengan *bluetooth controller*.

Pada Skripsi ini dibuat dua buah aplikasi untuk menambahkan cara baru memberikan masukan pada Google Cardboard. Perangkat lunak pertama digunakan untuk menganalisis data yang didapat dari sensor-sensor pada Android. Data yang telah diperoleh menggunakan perangkat lunak pertama diubah menjadi grafik dengan bantuan aplikasi Microsoft Excel. Perangkat lunak kedua dapat mendeteksi gerakan kepala penggunanya ketika sedang menggeleng atau mengangguk. Perangkat lunak kedua merupakan permainan berbasis VR. Jenis masukan baru yang diberikan hanya ya(mengangguk) atau tidak(menggeleng).

Agar VR yang menggunakan Google Cardboard dapat berjalan dengan sempurna, dibutuhkan *smartphone* yang memiliki 3 jenis sensor. Ketiga sensor itu adalah *Magnetometer*, *Accelerometer*, dan *Gyroscope*.^[5] Hampir seluruh *smartphone* sekarang telah memiliki sensor *Accelerometer*, dan *Gyroscope*. Khusus untuk sensor *Gyroscope*, tidak seluruh *smartphone* milikinya, terutama pada

low-end smartphone. *Magnetometer* digunakan untuk mengetahui arah pandang pengguna. *Accelerometer* digunakan untuk mengetahui arah gaya gravitasi.^[6] *Gyroscope* digunakan untuk mengetahui percepatan perputaran sudut kepala pengguna. Ketiga sensor ini juga harus menggunakan sensor 3 sumbu. Ketiga sensor tersebut tidak hanya berfungsi agar dapat menjalankan VR dengan Google Cardboard dan *smartphone*, tetapi juga dapat berfungsi sebagai pendekripsi gerakan kepala.

1.2 Rumusan Masalah

- Bagaimana cara memperoleh data yang diambil dari sensor-sensor pada *smartphone*?
- Bagaimana cara menganalisis hasil data sensor-sensor dan merancang algoritma deteksi gerakan kepala dari data tersebut.
- Bagaimana membuat aplikasi permainan dengan fitur pendekripsi gerakan kepala.

1.3 Tujuan

- Mempelajari dan mengimplementasikan cara untuk menampilkan grafik data dari sensor-sensor pada *smartphone*.
- Mempelajari dan mengimplementasikan cara mendekripsi gerakan kepala dari data yang didapat dari sensor-sensor pada *smartphone*.

1.4 Batasan Masalah

Penelitian ini dibuat berdasarkan batasan-batasan sebagai berikut:

1. Program pertama yang dibuat pada skripsi ini hanya digunakan untuk membantu dalam menganalisis sensor.
2. Program kedua yang dibuat hanya dapat melakukan pendekripsi gerakan kepala khusus untuk mengangguk dan menggeleng saja.

1.5 Metode Penelitian

Berikut adalah metode penelitian yang digunakan dalam penelitian ini:

1. Melakukan studi literatur tentang Android SDK, Google VR SDK, Quaternion, *Sensor Fusion*, dan algoritma *Head Motion Detection*.
2. Merancang dan membuat aplikasi untuk menampilkan grafik sensor-sensor pada *smartphone* Android.
3. Menganalisis aplikasi-aplikasi sejenis.
4. Merekam dan menganalisis grafik dari sensor-sensor pada *smartphone* ketika mengangguk dan menggeleng.

5. Menganalisis metode pendekripsi gerakan kepala.
6. Merancang aplikasi permainan yang mengimplementasikan pendekripsi gerakan kepala
7. Mengimplementasikan algoritma pendekripsi gerakan kepala ke aplikasi permainan VR.

1.6 Sistematika Penulisan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan, yaitu membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.
2. Bab 2: Dasar Teori, yaitu membahas teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang Android SDK, Google VR SDK, dan Quaternion.
3. Bab 3: Analisis, yaitu membahas mengenai analisa masalah. Berisi tentang analisis aplikasi-aplikasi sejenis, perekaman data sensor, analisis grafik dari sensor-sensor pada *smartphone* ketika mengangguk dan menggeleng, analisis metode pendekripsi gerakan kepala, dan analisis aplikasi permainan yang mendekripsi gerakan kepala.
4. Bab 4: Perancangan, yaitu membahas mengenai perancangan aplikasi permainan VR yang dapat mendekripsi gerakan menggeleng dan mengangguk.
5. Bab 5: Implementasi dan Pengujian, yaitu membahas mengenai implementasi dan pengujian aplikasi yang telah dilakukan. Berisi tentang implementasi dan hasil pengujian aplikasi.
6. Bab 6: Kesimpulan dan Saran, yaitu membahas mengenai kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat diberikan untuk penelitian selanjutnya.

BAB 2

DASAR TEORI

Pada bab ini dijelaskan dasar-dasar teori mengenai Android SDK, Google VR SDK, Quaternion, *Sensor Fusion*, dan algoritma *Head Motion Detection*.

2.1 Android SDK

Android SDK(*software development kit*) adalah kumpulan *source code*, *development tools*, *emulator*, dan semua *library* untuk membuat suatu aplikasi untuk *platform* Android[7]. IDE (*integrated development environment*) yang resmi untuk Android SDK adalah Android Studio. Android Studio dapat diunduh di halaman situs Google Developer,¹ sekaligus dengan Android SDK-nya.

2.1.1 Struktur File Android Studio Project

Pada saat membuat *project* baru, Android Studio membuat *folder-folder* dasar secara otomatis. *Folder-folder* tersebut memiliki tujuannya masing-masing. Struktur *folder-folder* tersebut ditampilkan pada Gambar 2.1.

2.1.2 Membuat User Interface

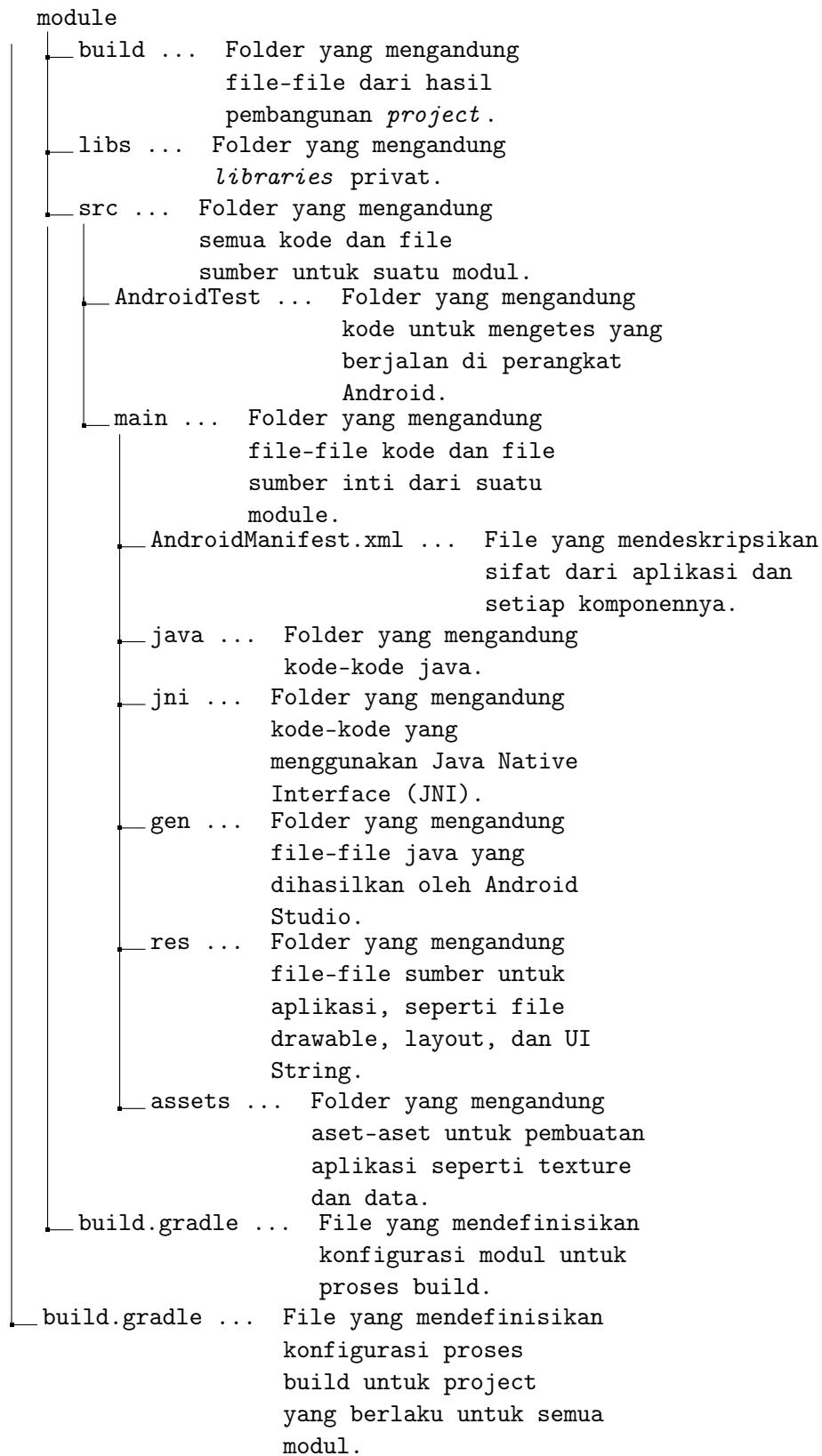
Pada subbab ini dijelaskan bagaimana membuat *layout* di XML termasuk *text field* dan *button*

Hierarki *Graphical User Interface (GUI)* untuk Aplikasi Android

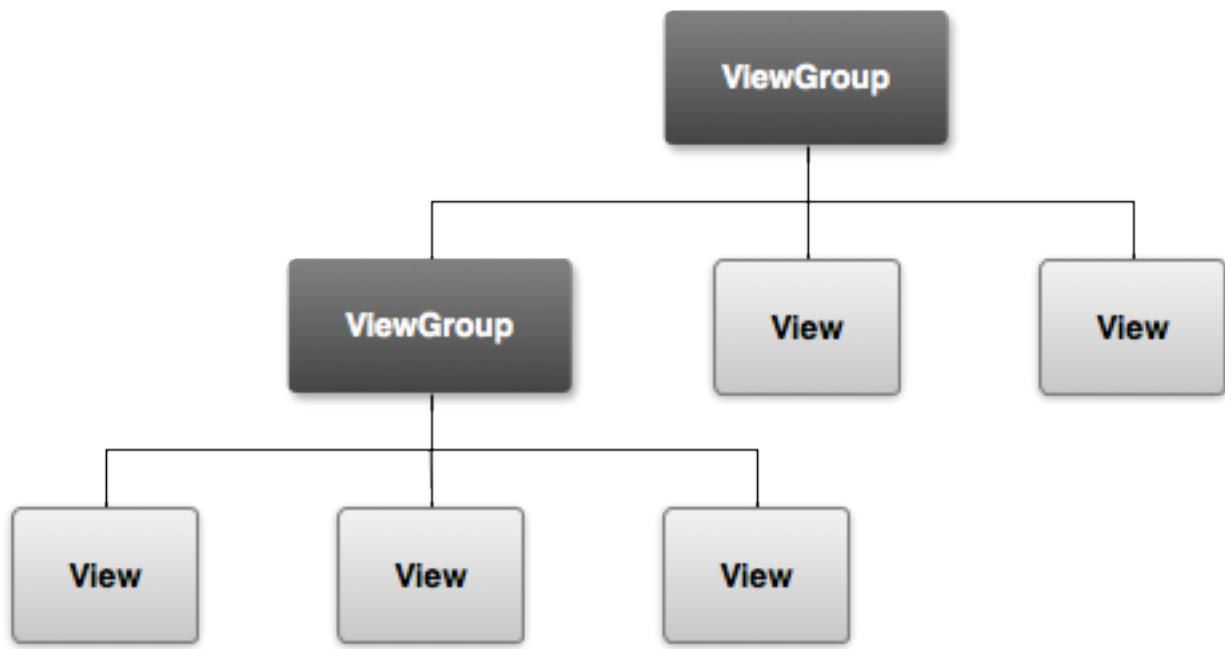
GUI untuk aplikasi Android dibuat dengan hierarki dari objek View dan ViewGroup (Gambar 2.2). Objek-objek dari View biasanya adalah *UI (User Interface) Widgets* seperti Button atau TextField. Objek-objek dari ViewGroup digunakan untuk penataan letak dan sebagai suatu tempat kumpulan tampilan. ViewGroup merupakan suatu View khusus yang dapat mengandung View lainnya. View yang berada di dalam suatu ViewGroup dapat disebut juga dengan *child view*. Seluruh *child view* tersebut dapat ditata seperti penataan *grid* atau *vertical list*

Android menggunakan *file* XML yang berkorespondensi kepada *subclasses* dari View dan ViewGroup, sehingga UI dapat dibuat dalam XML menggunakan hierarki dari elemen UI.

¹<https://developer.Android.com/studio/archive.html>



Gambar 2.1: Tampilan struktur folder pada *project* Android Studio



Gambar 2.2: Ilustrasi bagaimana percabangan objek ViewGroup pada *layout* dan mengandung objek View lainnya.

Atribut-atribut Objek View

Pada subbab ini dijelaskan atribut-atribut object View yang digunakan dalam membuat GUI pada file activity_main.xml

- **Android:id** Atribut ini merupakan identifikasi dari suatu view. Atribut ini dapat digunakan untuk menjadi referensi *object* dari kode aplikasi seperti membaca dan manipulasi objek tersebut (Akan dijelaskan lebih lanjut pada subbab 2.1.3). Tanda '@' dibutuhkan ketika melakukan referensi object dari suatu XML. Tanda '@' tersebut diikuti dengan tipe (id pada kasus ini), *slash*, dan nama (*edit_message* pada Listing 2.2). Tanda tambah (+) sebelum tipe hanya dibutuhkan jika ingin mendefinisikan *resource ID* untuk pertama kalinya.
- **Android:layout_width** dan **Android:layout_height** Atribut ini digunakan untuk mendefinisikan panjang dan lebar dari suatu objek View. Daripada menggunakan besar yang spesifik untuk panjang dan lebarnya, lebih baik menggunakan "wrap_content". Nilai "wrap_content" ini memberikan spesifikasi viewnya hanya sebesar yang dibutuhkan untuk memuat seluruh isi dari View. Jika menggunakan "match_parent" pada kasus Listing 2.2 besar View menjadi sebesar layar, karena besarnya mengikuti besar dari *parent*-nya LinearLayout.
- **Android:hint** Atribut ini merupakan *default string* untuk ditampilkan ketika objek View kosong. Daripada menggunakan *hard-coded string* sebagai *nilai* untuk ditampilkan, nilai "@string/edit_message" mengacu ke sumber string pada *file* yang berbeda. Karena mengacu ke sumber konkret, maka tidak dibutuhkan tanda tambah (+). Nilai string ini disimpan pada file Strings.xml yang ditunjukkan pada Listing 2.1.

Listing 2.1: Contoh kode pada string.xml

1| <resources>

```

2 |     <string name="app_name">MyFirstAndroidApp</string>
3 |     <string name="edit_message">Ini adalah hint</string>
4 |     <string name="button_send">Send</string>
5 |   </resources>

```

- **Android:onClick** Atribut ini memberitahu sistem untuk memanggil *method* yang sesuai namanya (contoh pada Listing 2.2 adalah `sendMessage()`) di Activity ketika pengguna melakukan klik pada *button* tersebut. Agar *system* dapat memanggil *method* yang tepat, *method* tersebut harus memenuhi kriteria berikut.
 - *Access Modifier* haruslah *public*.
 - Harus *void return value*nya.
 - Mempunyai *View* sebagai parameter satu-satunya. *View* ini diisi dengan *View* yang diklik.

Listing 2.2: Contoh kode file XML pada folder layout

```

1 | <LinearLayout
2 |   xmlns:Android="http://schemas.Android.com/apk/res/Android"
3 |   xmlns:tools="http://schemas.Android.com/tools"
4 |   Android:layout_width="match_parent"
5 |   Android:layout_height="match_parent"
6 |   Android:orientation="horizontal">
7 |     <EditText Android:id="@+id/edit_message"
8 |       Android:layout_weight="1"
9 |       Android:layout_width="0dp"
10 |       Android:layout_height="wrap_content"
11 |       Android:hint="@string/edit_message" />
12 |     <Button
13 |       Android:layout_width="wrap_content"
14 |       Android:layout_height="wrap_content"
15 |       Android:text="@string/button_send"
16 |       Android:onClick="sendMessage" />
17 |   </LinearLayout>

```

2.1.3 Activity

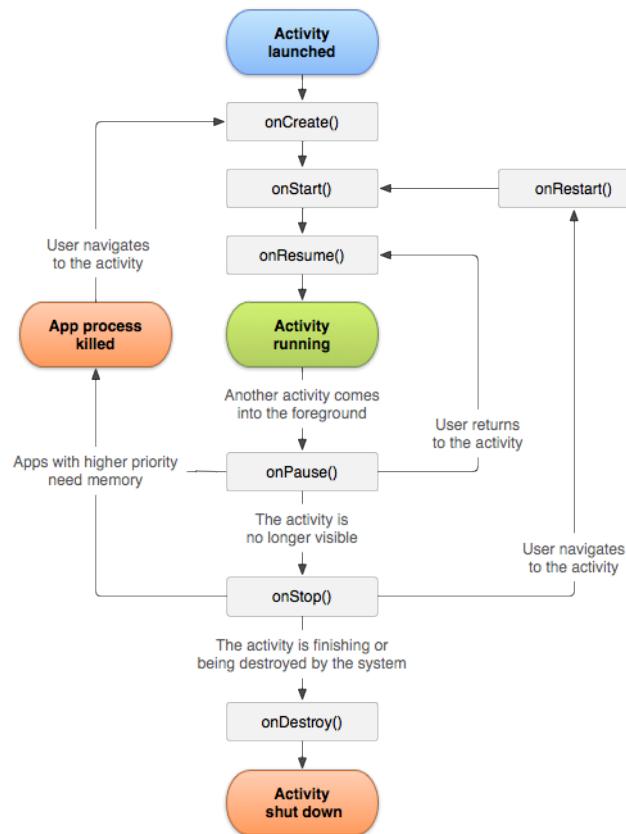
Activity adalah suatu hal yang fokus pada apa yang bisa pengguna lakukan. Hampir semua Activity berinteraksi dengan pengguna, jadi kelas Activity membuat suatu halaman baru yang bisa ditambahkan dengan *View*. Selain dapat ditampilkan kepada pengguna dengan halaman *full-screen*, Activity juga dapat ditampilkan dengan cara lain: seperti halaman *floating* atau tertanam di Activity lain.

Activity dapat diibaratkan sebagai halaman-halaman pada suatu aplikasi Android. Setiap perpindahan halaman pada Android merupakan pergantian Activity atau halaman yang lama menjadi Activity atau halaman baru yang akan ditampilkan kepada pengguna. Activity yang digantikan akan diberhentikan sementara. Activity yang sudah lama tidak digunakan lagi akan secara otomatis dihapus oleh sistem Android.

Activity Lifecycle

Aktivitas dalam sistem Android diatur sebagai *activity stack*. Ketika ada Activity baru yang dimulai, Activity tersebut ditempatkan di paling atas pada *stack* dan menjadi Activity aktif. Activity sebelumnya tetap berada di bawah *stack*, dan tidak muncul lagi sampai Activity yang baru berakhir.

Activity didasari dari empat kondisi:



Gambar 2.3: *State diagram* siklus Activity

- Jika Activity berada di latar depan pada layar, Activity tersebut sedang aktif.
- Jika Activity sudah tidak menjadi fokus tetapi masih dapat terlihat, Activity tersebut sedang berhenti sementara. Pada kondisi ini Activity tersebut masih berjalan, tapi bisa diberhentikan ketika sistem berada dalam situasi kekurangan memori.
- Jika suatu Activity benar-benar dihalangi oleh Activity lain, Activity tersebut telah berhenti. Activity tersebut tetap mengingat seluruh keadaan dan informasi anggota tetapi, Activity tersebut tidak lagi terlihat oleh pengguna jadi tampilan halamannya tersembunyi dan seringkali diberhentikan Activity-nya ketika sistem membutuhkan memori.
- Jika suatu Activity sedang berhenti sementara atau berhenti total, sistem dapat membuang Activity dari memori dengan cara menanyakan kepada pengguna untuk menghentikannya atau langsung diberhentikan oleh sistem. Jika Activity tersebut ditampilkan lagi kepada pengguna, Activity tersebut harus memulai dari awal dan kembali ke keadaan sebelumnya.

Gambar 2.3 menunjukkan pentingnya alur keadaan dari suatu Activity. Gambar segi empat menggambarkan *callback methods* yang dapat diberikan suatu implementasi untuk melakukan operasi ketika Activity berubah kondisi. Gambar oval berwarna merupakan kondisi-kondisi utama dari suatu Activity. Ada 3 kunci pengulangan untuk memantau suatu Activity:

- *Entire lifetime* terjadi diantara pemanggilan pertama pada `onCreate(Bundle)` sampai ke suatu pemanggilan terakhir `onDestroy()`. Suatu Activity melakukan semua persiapan pada kondisi

umum pada *method* `onCreate()`, dan melepaskan seluruh sisa pemrosesan pada *method* `onDestroy()`.

- *Visible lifetime* terjadi antara pemanggilan `onStart()` sampai pemanggilan yang sesuai pada `onStop()`. Pada tahap ini pengguna dapat melihat Activity pada layar meski pun tidak berada pada *foreground* dan berinteraksi dengan pengguna.
- *Foreground lifetime* terjadi antara pemanggilan *method* `onResume()` sampai ke satu pemanggilan akhir `onDestroy()`. Pada tahap ini Activity berada di depan semua Activity lainnya dan sedang berinteraksi dengan pengguna.

2.1.4 Android Sensor Framework

Sebagian besar dari perangkat Android sudah memiliki sensor yang mengukur gerakan, orientasi, dan berbagai keadaan lingkungan. Sensor-sensor ini dapat memberikan data mentah dengan tingkat akurasi yang tinggi. Sensor ini juga berguna untuk memantau pergerakan tiga dimensi atau posisi perangkat. Sensor ini juga dapat memantau perubahan keadaan lingkungan yang dekat dengan perangkat. Sensor-sensor yang didukung pada perangkat Android terbagi atas tiga kategori sebagai berikut:

- **Sensor gerak** Sensor-sensor ini mengukur akselerasi dan rotasi pada tiga sumbu. Kategori sensor ini meliputi *accelerometer*, sensor gravitasi, *gyroscope*, dan *rotation vector*.
- **Sensor keadaan lingkungan** Sensor-sensor ini mengukur berbagai keadaan lingkungan seperti suhu udara, tekanan, pencahayaan, dan kelembapan. Kategori sensor ini termasuk barometer, fotometer dan termometer.
- **Sensor posisi** Sensor-sensor ini mengukur posisi perangkat. Kategori sensor ini meliputi sensor orientasi dan magnetometer.

Android Sensor Framework membantu *developers* untuk mengakses berbagai jenis sensor pada Android. Sensor-sensor berbasis perangkat keras dan sensor-sensor berbasis perangkat lunak. Sensor berbasis perangkat keras mendapatkan data-nya dengan langsung mengukur suatu sifat lingkungan tertentu, seperti percepatan, kekuatan medan geomagnetik, atau perubahan sudut menggunakan perangkat keras yang dimiliki Android. Sensor berbasis perangkat lunak mendapatkan data dari satu atau lebih sensor berbasis perangkat keras. Sensor berbasis perangkat lunak ini juga terkadang disebut sensor virtual atau sensor sintetis. Pada tabel 2.1 dirincikan tipe-tipe setiap sensor posisi dan gerak, deskripsi, dan penggunaan umumnya.

Sistem Koordinat Sensor

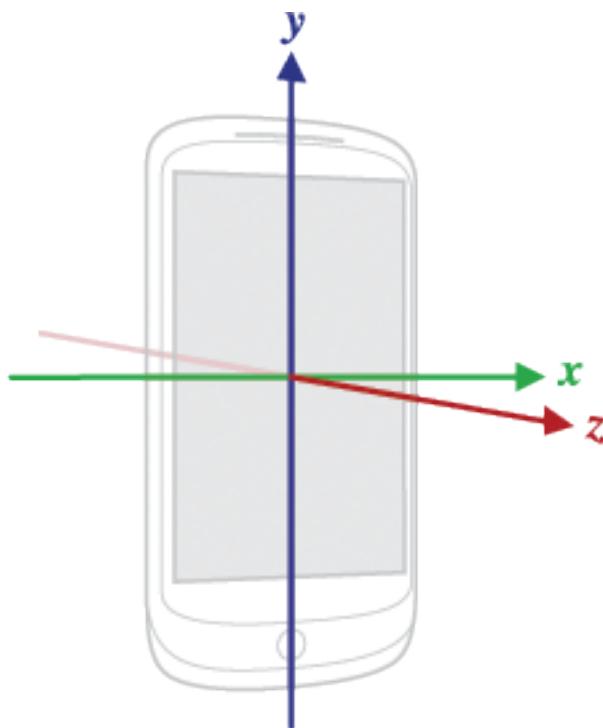
Pada umumnya, sensor *framework* menggunakan sistem tiga sumbu koordinat standar untuk mewakili nilai data. Sebagian besar sensor sistem koordinat relatif terhadap layar perangkat bila perangkat dibuat dalam orientasi standar (lihat Gambar 2.4).

Daftar sensor-sensor yang menggunakan sistem tiga sumbu seperti Gambar 2.4 adalah sebagai berikut :

Tabel 2.1: Tipe-tipe Sensor pada Android

Sensor	Tipe	Deskripsi	Penggunaan umum
TYPE_ACCELEROMETER	Perangkat Keras	Mengukur percepatan dalam m/s^2 yang terjadi pada perangkat di semua tiga sumbu fisik (x,y,z), termasuk percepatan gravitasi.	Deteksi gerak (goncangan, keseimbangan, dan lain-lain)
TYPE_GRAVITY	Perangkat Lunak atau Perangkat Keras	Mengukur percepatan gravitasi dalam m/s^2 yang terjadi pada perangkat di tiga sumbu fisik (x,y, dan z)	Deteksi gerak(goncangan, keseimbangan, dan lain-lain)
TYPE_GYROSCOPE	Perangkat Keras	Mengukur rata-rata rotasi sudut dalam rad/s di tiga sumbu fisik (x,y, dan z).	Deteksi rotasi (putaran, belokan, dan lain-lain).
TYPE_LINEAR_ACCELERATION	Perangkat Lunak atau Perangkat Keras	Mengukur percepatan dalam m/s^2 yang terjadi pada perangkat di semua tiga sumbu fisik (x,y,z), tidak termasuk percepatan gravitasi.	Memantau percepatan pada suatu sumbu.
TYPE_MAGNETIC_FIELD	Perangkat Keras	Mengukur medan magnet sekitar untuk semua tiga sumbu fisik (x,y, dan z) di satuan μT .	Membuat Kompas.
TYPE_ORIENTATION	Perangkat Lunak	Mengukur derajat rotasi yang terjadi pada perangkat pada semua tiga sumbu fisik (x,y, dan z).	Menentukan posisi perangkat
TYPE_ROTATION_VECTOR	Perangkat Lunak dan Perangkat Keras	Mengukur orientasi dari suatu perangkat dengan menyediakan tiga elemen dari vektor rotasi perangkat.	Deteksi gerak dan deteksi rotasi.

- Accelerometer
- Sensor Gravitasi
- Gyroscope
- Sensor Percepatan Linear
- Sensor Medan Geomagnetik



Gambar 2.4: Sistem koordinat (relatif dengan perangkatnya) yang digunakan oleh Sensor API

Koordinat sistem yang sumbu-nya tidak tertukar ketika orientasi perangkat berubah. Sistem koordinat sensor tidak pernah berubah seiring perangkatnya bergerak. Dalam aplikasi Android tidak dapat diberikan asumsi bahwa standar orientasi perangkat Android adalah *portrait*. Kebanyakan perangkat *Tablet* standar orientasi-nya adalah *landscape*. Sistem koordinat sensor selalu di dasarkan pada orientasi dasar dari suatu perangkat Android.

Mengidentifikasi Sensor dan Kemampuan Sensor

Android Sensor Framework menyediakan beberapa *method* yang dapat membuat developer mudah untuk menentukan sensor mana yang digunakan. API-nya juga dapat menyediakan *method* yang memungkinkan penggunanya menentukan kemampuan masing-masing sensor, seperti jangkauan maksimum, resolusi, dan kebutuhan dayanya. Untuk mengidentifikasi sensor-sensor yang ada pada perangkat, hal pertama yang perlu dilakukan adalah mendapatkan referensi sensor tersebut. Untuk mendapatkan referensi tersebut, dapat dilakukan dengan membuat instansi dari kelas SensorManager dan memanggil *method* getSystemService() dan memasukkan isi parameter-nya dengan SENSOR_SERVICE. Contohnya pada Listing 2.3.

Listing 2.3: Contoh inisialisasi kelas SensorManager

```

1| private SensorManager mSensorManager;
2| ...
3| mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

```

Kemudian untuk mendapatkan daftar dari setiap sensor pada suatu perangkat dapat dilakukan dengan cara memanggil *method* `getSensorList()` dan menggunakan konstanta `TYPE_ALL` pada kelas Sensor. Contohnya pada Listing 2.4

Listing 2.4: Contoh untuk mendapatkan daftar dari setiap sensor yang ada

```

1| List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);

```

Namun jika ingin mendapatkan sensor-sensor yang sesuai dengan tipe sensor yang diberikan, dapat menggunakan `TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION`, `TYPE_GRAVITY`, atau `TYPE_GRAVITY`.

Untuk menentukan jenis tertentu dari sensor yang ada pada perangkat dapat didapatkan dengan *method* `getDefaultSensor()` dengan dimasukkan dengan konstanta yang berada pada kelas Sensor. Jika perangkatnya memiliki lebih dari satu sensor dari tipe sensor yang diberikan, salah satu dari sensor tersebut dianggap sebagai sensor dasar. Jika sensor dasarnya tidak ada untuk sensor tersebut, perangkat tersebut berarti tidak memiliki sensor dengan jenis yang diberikan. Listing 2.5 adalah contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan.

Listing 2.5: Contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan

```

1| private SensorManager mSensorManager;
2| ...
3| mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
4| if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
5|     // Sukses! Perangkat ini memiliki sensor magnetometer.
6| }
7| else {
8|     // Gagal! Perangkat ini tidak memiliki sensor magnetometer.
9| }

```

Jika ingin membatasi versi atau vendor dari sensor yang digunakan, dapat menggunakan *method* `getVendor()` dan `getVersion()`. Seperti pada Listing 2.6 yang mengharuskan sensor gravitasi ber-*vendor* "Google Inc." dan memiliki versi 3. Jika sensor tersebut tidak tersedia pada perangkat, sensor accelerometer yang digunakan.

Listing 2.6: Contoh untuk mengecek apakah perangkat yang digunakan memiliki sensor dengan jenis yang diberikan

```

1| private SensorManager mSensorManager;
2| private Sensor mSensor;
3|
4| ...
5|
6| mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
7| mSensor = null;
8|
9| if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
10|     List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
11|     for(int i=0; i<gravSensors.size(); i++) {
12|         if ((gravSensors.get(i).getVendor().contains("Google Inc.)) &&
13|             (gravSensors.get(i).getVersion() == 3)){
14|                 // menggunakan sensor gravitasi versi 3.
15|                 mSensor = gravSensors.get(i);
16|             }
17|     }
18| }
19| if (mSensor == null){

```

```

20 // Use the accelerometer.
21 if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
22     mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
23 }
24 else{
25     // Tidak ada sensor gravitasi dan sensor accelerometer!
26 }
27 }
```

Salah satu *method* yang sangat berguna lagi adalah, *getMinDelay()*. Method ini digunakan untuk mengetahui minimum interval waktu suatu sensor dapat menerima data dalam mikro detik. Jika *method* *getMinDelay()* mengembalikan nilai nol, hal ini berarti sensor ini mengembalikan data setiap kali ada perubahan nilai pada sensor tersebut.

Memonitor Nilai Sensor

Untuk memonitor data mentah dari sensor, dibutuhkan untuk melakukan implementasi dua buah *method callback* yang berada pada interface *SensorEventListener*. Kedua *method* tersebut adalah *onAccuracyChanged(Sensor sensor, int accuracy)* dan *onSensorChanged(SensorEvent event)*. Sistem Android memanggil kedua *method* ini ketika terjadi salah satu kondisi ini:

- **Perubahan akurasi sensor.**

Dalam kasus ini sistem memanggil *method* *onAccuracyChanged(Sensor sensor, int accuracy)*. Parameter *sensor* diberi objek *Sensor* yang telah berubah akurasi-nya, dan parameter *accuracy* adalah nilai akurasi sensor yang baru.

- **Sensor memberitahu adanya nilai baru.**

Dalam kasus ini sistem memanggil *method* *onSensorChanged(SensorEvent event)*, dengan parameter *event* akan diisi dengan objek *SensorEvent* untuk mendapatkan nilai barunya. Objek *SensorEvent* Mengandung semua informasi tentang data sensor yang baru, termasuk: akurasi dari data, sensor yang mendapatkan data, dan catatan waktu data tersebut didapatkan, dan data yang baru yang telah didapatkan.

Pada Listing 2.7 akan ditunjukkan bagaimana menggunakan *method* *onSensorChanged(SensorEvent event)* untuk memonitor data dari sensor cahaya. Pada Listing 2.7 akan menampilkan data mentah dari sensor ke *TextView* yang telah dibuat pada file *main.xml* sebagai *sensor_data*.

Listing 2.7: Contoh memonitor data mentah pada sensor cahaya

```

1 public class SensorActivity extends Activity implements SensorEventListener {
2     private SensorManager mSensorManager;
3     private Sensor mLight;
4     private TextView textView;
5
6     @Override
7     public final void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.main);
10
11         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
12         mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
13     }
14
15     @Override
16     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
17         // Hal yang perlu dilakukan aplikasi ketika akurasinya berubah.
18     }
19
20     @Override
21     public final void onSensorChanged(SensorEvent event) {
```

```

22     // Sensor cahaya mengembalikan 1 nilai saja.
23     // Banyak sensor lain yang mengembalikan lebih dari 1 nilai.
24     float lux = event.values[0];
25     // Hal yang perlu dilakukan ketika ada perubahan data.
26 }
27
28 @Override
29 protected void onResume() {
30     super.onResume();
31     mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
32 }
33
34 @Override
35 protected void onPause() {
36     super.onPause();
37     mSensorManager.unregisterListener(this);
38 }
39 }
```

Pada *method* `onSensorChanged(SensorEvent event)`, struktur nilai-nilai yang dikembalikan dijelaskan pada subbab 2.1.4. Pada *method* `onResume()`, ada pemanggilan *method* `registerListener()`. *Method* `registerListener()` ini berguna untuk menetapkan waktu *delay* pada pemanggilan `onSensorChanged()`. Untuk menetapkan *delay*-nya dapat menggunakan konstanta yang ada pada kelas `SensorManager`. Konstanta-konstanta tersebut adalah `SENSOR_DELAY_NORMAL` (200.000 mikro detik),`SENSOR_DELAY_GAME` (20.000 mikro detik),`SENSOR_DELAY_UI` (60.000 mikro detik), atau `SENSOR_DELAY_FASTEST` (0 mikro detik). Pengaturan dasar untuk waktu *delay* ini menggunakan konstanta `SENSOR_DELAY_NORMAL`. Perlu diperhatikan juga pemesanan sensor ketika activity sedang dihentikan sementara maupun dilanjutkan kembali. Sistem tidak boleh tetap merekam sensor ketika activity tidak aktif. Hal ini diperlukan karena pada saat perangkat Android menggunakan sensor, perangkat menggunakan tenaga yang banyak. Akan lebih baik jika penggunaan sensor diberhentikan ketika activity-nya sudah tidak lagi digunakan.

Struktur Nilai yang Dikembalikan oleh Sensor

Setiap sensor Android mengembalikan nilai-nilainya dengan struktur-struktur tertentu. Pada bab ini dijelaskan secara detail struktur nilai sistem Android dalam mengembalikan nilai-nilai yang diperoleh dari sensor. Nilai ini didapatkan dengan tipe data *array of float*. Besar dan isi dari array tergantung pada sensor yang sedang dipantau. Berikut ini adalah struktur-struktur nilai dari setiap sensor pada sistem Android :

- **TYPE_ACCELEROMETER**

Semua nilai didefinisikan sebagai satuan meter per detik kuadrat m/s^2

- `values[0]`: Percepatan yang terjadi pada sumbu x dikali -1
- `values[1]`: Percepatan yang terjadi pada sumbu y dikali -1
- `values[2]`: Percepatan yang terjadi pada sumbu z dikali -1

Sensor ini mengukur percepatan(*Ad*) yang diterapkan pada perangkat. Sensor tersebut dapat mengukur percepatan dengan mengukur gaya(*Fs*) yang terjadi pada sensor menggunakan relasi berikut:

$$Ad = -\Sigma F_s/mass$$

Secara khusus, gravitasi selalu memberikan pengaruh pada percepatan yang diukur :

$$Ad = -g - \Sigma F/mass$$

Karena inilah ketika perangkat Android sedang diam, accelerometer membaca percepatan gravitasi sebesar $g = 9.81m/s^2$. Demikian pula ketika perangkat Android sedang dalam keadaan jatuh bebas. Perangkat mempercepat menuju ke tanah pada percepatan $9.81m/s^2$, sehingga accelerometer membaca percepatan total sebesar $0m/s^2$. Untuk mengukur percepatan asli yang terjadi pada perangkat, kontribusi gravitasi harus dieliminasi. Hal ini bisa dilakukan dengan menerapkan *high-pass* filter. Sebaliknya, *low-pass* filter dapat digunakan untuk mendapatkan nilai gravitasi saja.

Listing 2.8: Implementasi *low-pass* filter

```

1  public void onSensorChanged(SensorEvent event)
2  {
3      // alpha dikalkulasikan sebagai t / (t + dT)
4      // dengan t adalah low-pass filter's time-constant
5      // dan dT, rata-rata event tersampaikan
6
7      final float alpha = 0.8;
8
9      gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
10     gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
11     gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];
12
13     linear_acceleration[0] = event.values[0] - gravity[0];
14     linear_acceleration[1] = event.values[1] - gravity[1];
15     linear_acceleration[2] = event.values[2] - gravity[2];
16 }
```

Implementasi *low-pass* filter ditunjukkan pada Listing 2.8

• TYPE_MAGNETIC_FIELD

Sensor ini mengukur medan magnet sekitar perangkat pada sumbu X,Y, dan Z dalam satuan micro-tesla.

• TYPE_GYROSCOPE

Sensor ini mengukur rata-rata perputaran pada perangkat yang berputar di sumbu X,Y, dan Z dalam satuan radian per detik ($\frac{rad}{s}$). Sistem koordinat yang digunakan sama dengan sistem koordinat pada sensor percepatan(Accelerometer). Jika perangkat berputar berlawanan arah jarum jam pada sumbu tertentu, maka rotasi yang terjadi bernilai positif. Perhatikan bahwa standar perputaran ini adalah definisi matematika standar pada rotasi positif.

- values[0]: Percepatan angular pada sumbu X.
- values[1]: Percepatan angular pada sumbu Y.
- values[2]: Percepatan angular pada sumbu Z.

Biasanya keluaran dari gyroscope muncul dari waktu ke waktu untuk menghitung rotasi yang menggambarkan perubahan sudut atas langkah waktu, misalnya pada Listing 2.9

Listing 2.9: contoh implementasi gyroscope

```
1| private static final float NS2S = 1.0f / 1000000000.0f;
```

```

2     private final float[] deltaRotationVector = new float[4]();
3     private float timestamp;
4
5     public void onSensorChanged(SensorEvent event) {
6         // Pada tahapan ini delta rotasi dikalikan dengan rotasi saat ini
7         // setelah mengcomputasinya dari data gyro.
8         if (timestamp != 0) {
9             final float dT = (event.timestamp - timestamp) * NS2S;
10            // Sumbu dari rotasi, masih belum dinormalisasi.
11            float axisX = event.values[0];
12            float axisY = event.values[1];
13            float axisZ = event.values[2];
14
15            // Menghitung percepatan angular
16            float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY + axisZ*axisZ);
17
18            // Normalisasi rotasi vektor jika cukup besar untuk mendapatkan sumbunya.
19            if (omegaMagnitude > EPSILON) {
20                axisX /= omegaMagnitude;
21                axisY /= omegaMagnitude;
22                axisZ /= omegaMagnitude;
23            }
24
25            // Integrasi di setiap sumbu dengan setiap kecepatan sudut di setiap waktunya
26            // untuk mendapatkan delta rotasi dari waktu sebelumnya hingga waktu sekarang.
27            // Hasil representasi delta rotasi dari sudut ini diubah
28            // menjadi kuaternion sebelum merubahnya kembali menjadi rotasi matriks.
29            float thetaOverTwo = omegaMagnitude * dT / 2.0f;
30            float sinThetaOverTwo = sin(thetaOverTwo);
31            float cosThetaOverTwo = cos(thetaOverTwo);
32            deltaRotationVector[0] = sinThetaOverTwo * axisX;
33            deltaRotationVector[1] = sinThetaOverTwo * axisY;
34            deltaRotationVector[2] = sinThetaOverTwo * axisZ;
35            deltaRotationVector[3] = cosThetaOverTwo;
36        }
37        timestamp = event.timestamp;
38        float[] deltaRotationMatrix = new float[9];
39        SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
40        // User code should concatenate the delta rotation we computed with the current rotation
41        // in order to get the updated rotation.
42        // rotationCurrent = rotationCurrent * deltaRotationMatrix;
43    }
44}

```

Dalam prakteknya, gyroscope *noise* dan *offset* akan menyebabkan beberapa kesalahan yang harus diberikan kompensasi. Cara untuk memberikan kompensasi biasanya dilakukan dengan menggunakan informasi dari sensor lain.

- **TYPE_GRAVITY**

Sensor ini menunjukkan arah dan besarnya vektor gaya gravitasi. Sensor ini mengembalikan nilai dengan satuan m/s^2 . Sistem koordinat sama seperti sistem koordinat yang umum digunakan sensor percepatan.

Catatan: Bila perangkat sedang diam, maka keluaran dari sensor gravitasi harus identik dengan accelerometer.

- **TYPE_LINEAR_ACCELERATION**

Sensor yang menunjukkan percepatan pada setiap sumbu perangkat, tidak termasuk percepatan yang terjadi karena gravitasi. Nilai diberikan dalam satuan m/s^2 . Sistem koordinat yang digunakan sama seperti sistem koordinat yang digunakan sensor percepatan. Keluaran dari sensor accelerometer, gravitasi dan percepatan linear harus mengikuti aturan berikut:

$$\text{percepatan} = \text{gravitasi} + \text{percepatan linear}$$

- **TYPE_ORIENTATION**

Semua nilai adalah sudut dalam derajat.

- values[0]: Azimuth, sudut diantara arah magnetik utara dengan sumbu y, sekitar sumbu z (0 sampai 359). 0 = Utara, 90 = Timur, 180 = Selatan, 270 = Barat
- values[1]: Pitch, rotasi sekitar sumbu x (-180 sampai 180), dengan nilai positif ketika sumbu x bergerak menuju sumbu y.
- values[2]: Roll, perputaran sekitar sumbu y (-90 sampai 90) pada kondisi *portrait*, sensor bernilai 0. Pada kondisi *landscape* ke kanan sensor bernilai 90 dan sebaliknya yaitu kondisi *landscape* ke kiri sensor bernilai -90.

Definisi ini berbeda dengan definisi *Azimuth*, *pitch*, dan *roll* yang digunakan pada aviasi yang sumbu X adalah sepanjang sisi bidang. Sensor ini sudah tidak digunakan lagi(*deprecated*), yang digunakan sekarang adalah sensor rotasi vector.

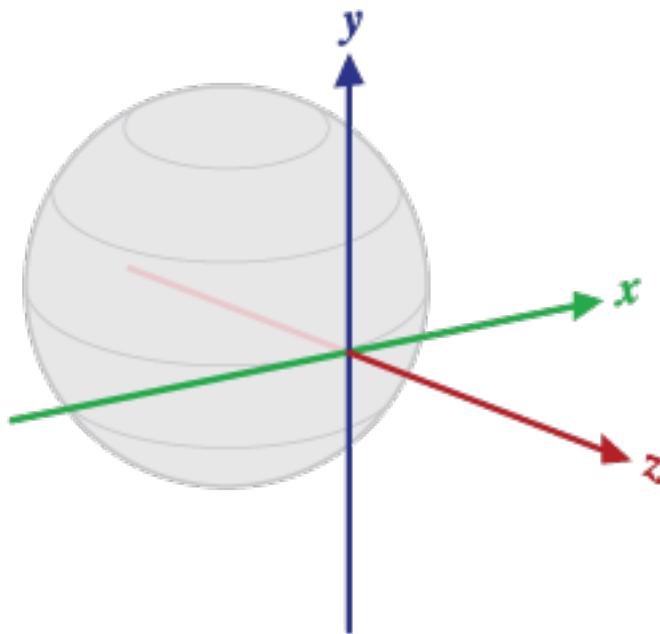
- **TYPE_ROTATION_VECTOR**

Sensor ini menunjukkan orientasi perangkat dengan kombinasi dari sumbu dan sudut. Perangkat diputar sebesar sudut θ mengelilingi sumbu x, y, z . Tiga elemen dari vektor rotasi adalah $(x \sin(\frac{\theta}{2}), y \sin(\frac{\theta}{2}), z \sin(\frac{\theta}{2}))$, sehingga besarnya vektor rotasi sama dengan $\sin(\frac{\theta}{2})$, dan arah vektor rotasi sama dengan sumbu rotasi. Tiga elemen dari vektor rotasi sama dengan tiga komponen terakhir pada unit kuaternion $(\cos(\frac{\theta}{2}), x \sin(\frac{\theta}{2}), y \sin(\frac{\theta}{2}), z \sin(\frac{\theta}{2}))$ yang dijelaskan pada subbab 2.3. Elemen dari vektor rotasi tak memiliki satuan. Sistem koordinat yang digunakan sama dengan sistem koordinat yang digunakan pada sensor percepatan. Referensi koordinat diartikan sebagai basis orthonormal, yaitu:

- X diartikan sebagai perkalian dot product **Y.Z**
- Y merupakan tangensial ke tanah pada lokasi perangkat saat ini dan menunjuk ke arah utara.
- Z menghadap ke langit dan tegak lurus dengan tanah. Untuk lebih jelasnya dapat dilihat pada Gambar 2.5
- values[0]: $x \sin(\frac{\theta}{2})$
- values[1]: $y \sin(\frac{\theta}{2})$
- values[2]: $z \sin(\frac{\theta}{2})$
- values[3]: $\cos(\frac{\theta}{2})$
- values[4]: Perkiraan akurasi (dalam radian) (-1 jika tidak tersedia)

2.2 Google VR SDK

Google VR SDK digunakan untuk membantu dalam pembuatan aplikasi VR pada *smartphone*[8]. Google VR SDK memberikan beberapa fitur sebagai berikut :



Gambar 2.5: Sistem koordinat sensor rotasi vektor terhadap Bumi

- **Binocular rendering:** Fitur untuk tampilan layar terpisah untuk masing-masing dalam pandangan VR.
- **Spatial audio:** Fitur untuk mengeluarkan suara yang datang dari daerah-daerah tertentu dari dunia VR.
- **Head movement tracking:** Fitur untuk mendapatkan memperbaharui penglihatan dunia VR yang sesuai dengan gerakan kepala pengguna.
- **Trigger input:** Fitur untuk memberikan masukan pada dunia VR dengan menekan tombol.

Ada beberapa persyaratan untuk menggunakan Google VR SDK, persyaratan tersebut adalah:

- Android Studio versi 1.0 atau lebih.
- Android SDK versi 23
- Gradle versi 23.0.1 atau lebih. Android Studio membantu meningkatkan versinya jika versinya terlalu rendah.
- Perangkat Android fisik yang menjalankan Android versi 4.4 (KitKat) atau lebih.

Dalam membuat aplikasi Google Cardboard VR membutuhkan beberapa API(Application Program Interface) dari Google VR SDK. API-API umum yang digunakan adalah sebagai berikut.

- API audio: API untuk membuat implementasi dari ***Spatial Audio*** (Metode untuk memberikan sumber suara yang sesuai dengan tempatnya dalam ruang tiga dimensi).
- API base: API untuk fondasi dari suatu aplikasi Google VR.

2.2.1 API Audio

API ini membantu developer untuk memberikan sumber suara secara spasial dalam tiga dimensi, termasuk jarak dengan tinggi isyarat sumber suara. Pada API ini hanya terdapat satu class utama yaitu **GvrAudioEngine**. **GvrAudioEngine** mampu memutarkan suara secara spasial dalam dua cara yang berbeda :

- Metode pertama dikenal sebagai *Sound Object rendering*. Metode ini memungkinkan pengguna membuat sumber suara virtual dalam ruang tiga dimensi.
- Metode kedua memungkinkan pengguna untuk memutar kembali rekaman *Ambisonic soundfield*. Rekaman *Ambisonic soundfield* adalah file audio spasial *multi-channel*.

API ini juga dapat memutarkan suara secara *stereo*. Kelas **GvrAudioEngine** memiliki tiga buah *nested class* yaitu:

- **GvrAudioEngine.DistanceRolloffModel**: Kelas ini mendefinisikan konstanta-konstanta yang mewakili perbedaan jarak dari efek model-model *rolloff*.
- **GvrAudioEngine.MaterialName**: Kelas ini mendefinisikan konstanta-konstanta yang mewakili bahan permukaan ruangan untuk disesuaikan dengan efek suara pada suatu ruangan.
- **GvrAudioEngine.RenderingMode**: Kelas ini mendefinisikan konstanta-konstanta untuk menyesuaikan dengan mode *rendering*. Semakin baik kualitas *rendering* akan semakin besar penggunaan CPU(Central Processing Unit).

2.2.2 API Base

API ini digunakan sebagai fondasi dari suatu aplikasi Google VR. Fitur-fitur *Binocular rendering*, *Head movement tracking*, dan *Trigger input* diberikan pada API ini. Kelas-kelas penting yang ada di API ini adalah **AndroidCompat**, **Eye**, **GvrActivity**, **GvrView**, **HeadTransform**, **Viewport**.

- **AndroidCompat**

Kelas ini merupakan kelas utilitas untuk menggunakan fitur VR. Fitur-fitur ini mungkin tidak tersedia pada semua versi Android. Kelas ini memiliki *method-method* sebagai berikut:

- `setSustainedPerformanceMode(Activity activity, boolean enabled)`:
Method ini digunakan untuk mengubah window Android ke mode performa secara berkelanjutan.
- `public static void setVrModeEnabled (Activity activity, boolean enabled)`:
Mengatur pengaturan yang tepat untuk "mode VR" pada suatu Activity. Method ini tidak digunakan karena hanya dapat digunakan pada Android N+.
- `public static boolean trySetVrModeEnabled (Activity activity, boolean enabled)`: Method ini kegunaanya sama dengan *method setVrModeEnabled (Activity activity, boolean enabled)*, namun mengembalikan boolean true jika berhasil dan sebaliknya.

- Eye

Kelas ini mendefinisikan detail pe-*render*-an stereoskopik mata. Stereoskopik adalah sebuah teknik untuk membuat atau menampilkan ilusi mendalam pada sebuah gambar untuk penglihatan binokular. Method penting yang dimiliki kelas ini adalah **public float[] getEyeView ()**. Method ini mengembalikan matriks yang mengubah kamera virtual ke mata. Transformasi yang diberikan termasuk melacak rotasi kepala, perubahan posisi dan perubahan IPD(interpupillary distance).

- GvrActivity

Kelas ini merupakan Activity dasar yang menyediakan integrasi yang mudah dengan headset Google VR. Kelas ini mengekspos kejadian untuk berinteraksi dengan headset Google VR dan menangani detail-detail yang biasa diperlukan saat membuat suatu Activity untuk pe-*render*-an VR. Activity ini membuat layar tetap menyala selama perangkat Android bergerak. Jika tidak ada pergerakan dari perangkat Android maka layar reguler (*wakeclock*) akan ditampilkan. Pada kelas ini terdapat method **onCardboardTrigger ()** untuk mendeteksi ketika Cardboard Trigger sedang ditarik dan dilepaskan (Magnet yang berada pada sisi Google Cardboard).

- GvrView

Kelas ini merupakan kelas View yang menyediakan pe-*render*-an VR. Kelas ini didesain untuk berkerja pada mode layar penuh dengan orientasi *landscape* atau *reverse landscape*. Kelas View ini dapat digunakan dengan mengimplementasikan salah satu Interface pe-*render*-an. Interface-interface tersebut adalah:

- GvrView.StereoRenderer: Interface untuk pe-*render*-an detail stereoskopik secara abstrak oleh pe-*render*-an.
- GvrView.Renderer: Interface untuk mesin yang kompleks yang membutuhkan untuk menangai semua detail pe-*render*-an.

Kelas GvrView.Renderer jarang digunakan dan sebaiknya tidak digunakan jika tidak sangat dibutuhkan. Ketika suatu kelas mengimplementasikan Kelas GvrView.StereoRenderer, kelas tersebut harus melakukan implementasi *method-method* berikut ini:

- **public void onNewFrame(HeadTransform headTransform)**

method ini terpanggil ketika Framebaru digambar. Method ini memungkinkan untuk membedakan antara pe-*render*-an pandangan mata dan frame-frame yang berbeda. Setiap operasi per-frame harus tidak spesifik pada satu tampilan saja.

- **public abstract void onDrawEye (Eye eye)**

Method ini meminta untuk menggambar suatu konten dari sudut pandang mata.

- **public abstract void onFinishFrame (Viewport viewport)**

Method ini dipanggil ketika suatu frame telah selesai. Dengan pemanggilan ini, konten frame telah digambar dan jika koreksi distorsi diaktifkan, koreksi distorsi diterapkan. Setiap pe-*render*-an pada tahap ini relatif terhadap seluruh permukaan, tidak terhadap satu pandangan mata tunggal.

– **public abstract void onRendererShutdown ()**

Method ini dipanggil ketika thread *pe-render-an* sedang menutup. Melepaskan sumber GL(Graphics Library) dan sedang melakukan penutupan operasi pada thread *pe-render-an*. Dipanggil hanya jika sebelumnya ada pemanggilan *method* onSurfaceCreated.

– **public abstract void onSurfaceChanged (int width, int height)**

Dipanggil ketika ada perubahan dimensi permukaan. Semua nilai adalah relatif ke ukuran yang dibutuhkan untuk *me-render* sebuah mata.

– **public abstract void onSurfaceCreated (EGLConfig config)**

Method ini dipanggil ketika suatu permukaan dibangun atau dibangun ulang.

- HeadTransform

Method ini mendeskripsikan transformasi kepala secara *independent* dari setiap parameter mata. Kelas ini digunakan di kelas GvrView.StereoRenderer sebagai parameter pada *method* onNewFrame. Method-method yang perlu diperhatikan pada kelas ini adalah:

– **public void getHeadView (float[] headView, int offset)**

Method ini digunakan untuk mendapatkan matriks transformasi dari kamera virtual ke kepala. Kepala di sini diartikan sebagai titik tengah diantara kedua mata. Matriks yang didapatkan disimpan pada parameter **headView**.

– **public void getQuaternion (float[] quaternion, int offset)**

Method ini digunakan untuk mendapatkan kuaternion yang menjadi representasi dari rotasi kepala.

- Viewport

Kelas ini diartikan sebagai *viewport*(area pandang) berbentuk persegi.

2.3 Teori Kuaternion

Pada Android SDK Sensor framework tipe sensor **Sensor.TYPE_ROTATION_VECTOR** merupakan tipe sensor yang mendekripsi vektor perputaran pada *smartphone*[9]. Tipe sensor ini mengembalikan nilai-nilai dari komponen kuaternion. Kuaternion adalah objek penggabungan dari suatu skalar dengan suatu vektor, sesuatu yang tidak dapat diartikan dalam aljabar linear biasa. Kuaternion ditemukan oleh William Rowan Hamilton dengan memperpanjang notasi dari bilangan kompleks menjadi Kuaternion.

2.3.1 Struktur Ajabar

Karena Kuaternion merupakan bilangan kompleks yang diperpanjang notasinya, struktur aljabar Kuaternion hampir mirip dengan bilangan kompleks. Untuk mengerti struktur-struktur aljabar kuaternion, diperlukan untuk mengerti bilangan kompleks terlebih dahulu. Berikut adalah penjelasan singkat tentang bilangan kompleks.

Bilangan Kompleks

Bilangan kompleks adalah bilangan yang merupakan gabungan dari bilangan imajiner dengan bilangan riil. Notasi umum dari bilangan kompleks adalah :

$$a + bi \quad (2.1)$$

Pada notasi 2.1 bilangan a dengan b merupakan bilangan riil, dan i merupakan bilangan imajiner tertentu yang memiliki sifat $i^2 = -1$. Bilangan kompleks juga dapat beroperasi dengan bilangan kompleks lainnya seperti penjumlahan, perkalian, pengurangan, dan pembagian. Berikut adalah contoh-contoh operasi pada bilangan kompleks 2.1 dengan bilangan kompleks $c + di$:

- Penjumlahan

$$(a + bi) + (c + di) = (a + c) + i(b + d)$$

- Perkalian

$$(a + bi)(c + di) = (acbd) + (bc + ad)i$$

- Pengurangan

$$(a + bi) - (c + di) = (a - c) + i(b - d)$$

- Pembagian

$$\frac{(a + bi)}{(c + di)} = \frac{(ac + bd)}{c^2 + d^2} + i \frac{bc - ad}{c^2 + d^2}$$

Pada operasi jumlah dan kali untuk kedua bilangan kompleks tersebut memiliki hukum assosiatif dan komutatif. Notasi 2.2 menunjukkan bagaimana bagaimana kedua hukum tersebut berlaku pada operasi jumlah.

$$\begin{aligned} (a + ib) + (c + id) &= (a + c) + i(b + d) = \\ (c + id) + (a + ib) &= (c + a) + i(d + b) \end{aligned} \quad (2.2)$$

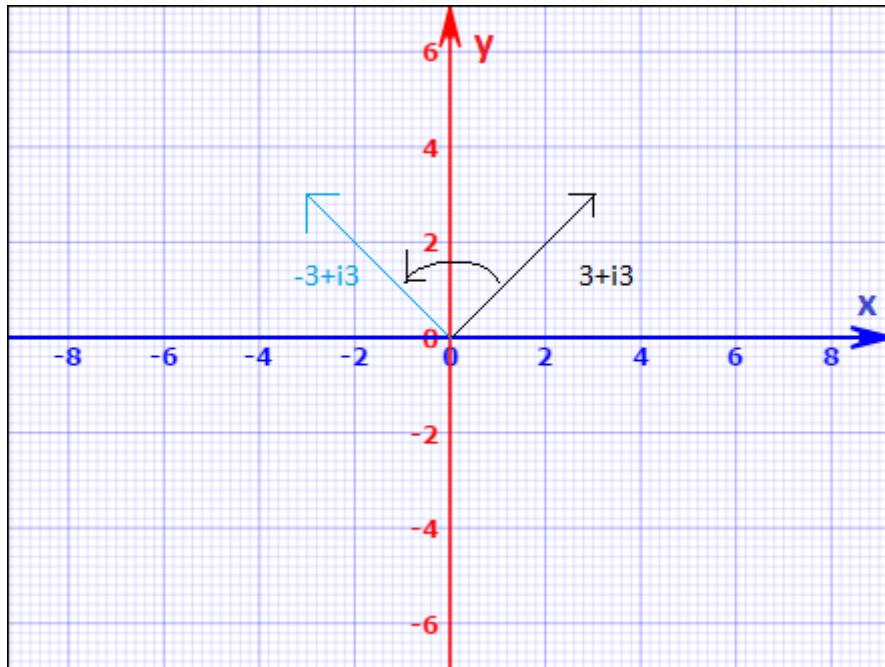
Suatu bilangan dapat dikatakan konjugasi kompleks dari suatu bilangan kompleks jika nilai bilangan riilnya sama, tetapi nilai bilangan imajinernya berlawanan dengan nilai pada bilangan kompleks tersebut. Maka konjugasi kompleks dari bilangan kompleks 2.1 adalah $a - bi$.

Bilangan kompleks ini dapat digunakan untuk rotasi vektor pada bidang dua dimensi. Rotasi ini dapat dilakukan dengan mengalikan suatu vektor dengan bilangan imajiner i . Mengalikan suatu vektor dengan bilangan imajiner i akan memutar vektor sebesar 90° berlawanan arah jarum jam. Mengalikan suatu vektor dengan bilangan imajiner i^2 akan memutar vektor sebesar 180° berlawanan arah jarum jam. Untuk memperjelas perputaran dengan bilangan kompleks diberikan contoh berikut: Sebuah vektor $v = 3 + i3$ akan diputar 90° berlawanan arah jarum jam dengan mengalikan

vektor tersebut dengan bilangan imajiner i . Maka vektor hasil perputarannya(v') adalah :

$$\begin{aligned}
 v' &= i(3 + i3) \\
 &= i3 + i^23 \\
 &= i3 + (-1)3 \\
 &= -3 + i3
 \end{aligned} \tag{2.3}$$

Dengan bilangan pada bilangan riil diasumsikan sebagai nilai pada sumbu x dan bilangan yang



Gambar 2.6: Contoh perputaran dengan bilangan kompleks

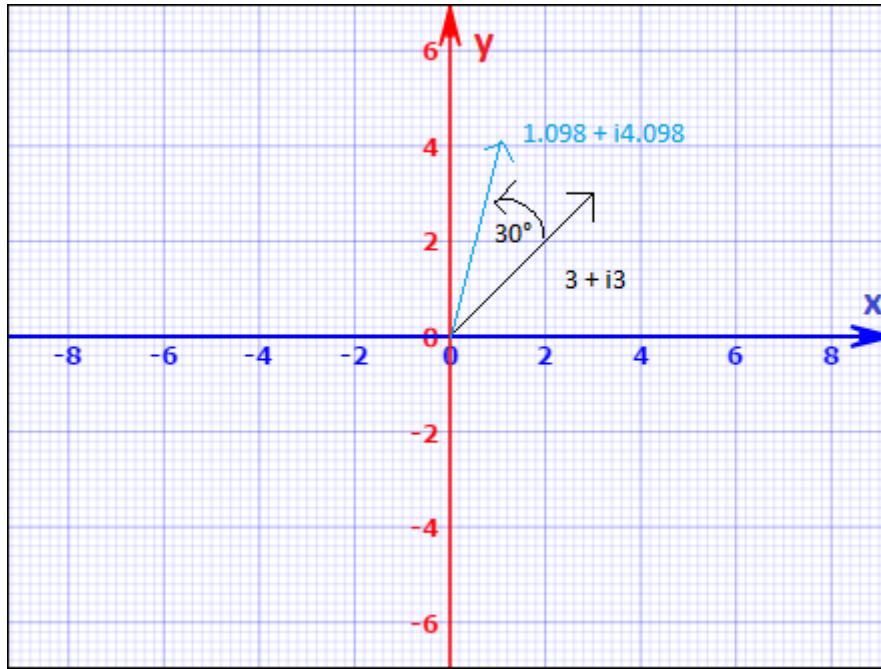
dikalikan dengan bilangan i diasumsikan pada sumbu y($x + iy$) Seperti yang ditunjukkan pada Gambar 2.6. Oleh karena itu rumus perputaran menggunakan bilangan kompleks dapat di rumuskan sebagai berikut:

$$v' = v \times (\cos \theta + i \sin \theta)$$

dengan θ adalah besar sudut perputaran. Jika vektor $v = 3 + i3$ diputar 30° berlawanan arah jarum jam menggunakan konsep diatas, akan menghasilkan vektor berikut:

$$\begin{aligned}
 v' &= (3 + i3)(\cos 30^\circ + i \sin 30^\circ) \\
 &= (3 + i3)\left(\frac{\sqrt{3}}{2} + i\frac{1}{2}\right) \\
 &= \frac{3\sqrt{3} - 3}{2} + i\frac{3\sqrt{3} + 3}{2} \\
 &= 1.098 + i4.098
 \end{aligned} \tag{2.4}$$

Dari persamaan tersebut dapat digambarkan pada Gambar 2.7.



Gambar 2.7: Contoh perputaran tiga puluh derajat dengan bilangan kompleks

2.3.2 Aljabar Kuaternion dan Operasi-operasi pada Kuaternion

Kuaternion ditemukan oleh ahli matematika dan astronomi Inggris, William Rowan Hamilton, dengan memperpanjang aritmetika dari bilangan kompleks. Dari penemuan tersebut William Rowan Hamilton menemukan bahwa dia tidak hanya membutuhkan bilangan imajiner i saja untuk melakukan rotasi pada ruang tiga dimensi. Dia menemukan bahwa dia juga membutuhkan tiga komponen imajiner lainnya yaitu i, j dan k . Persamaan umum Kuaternion memiliki empat bilangan riil atau skalar. Persamaan tersebut adalah

$$q = q_0 + iq_1 + jq_2 + kq_3 \quad (2.5)$$

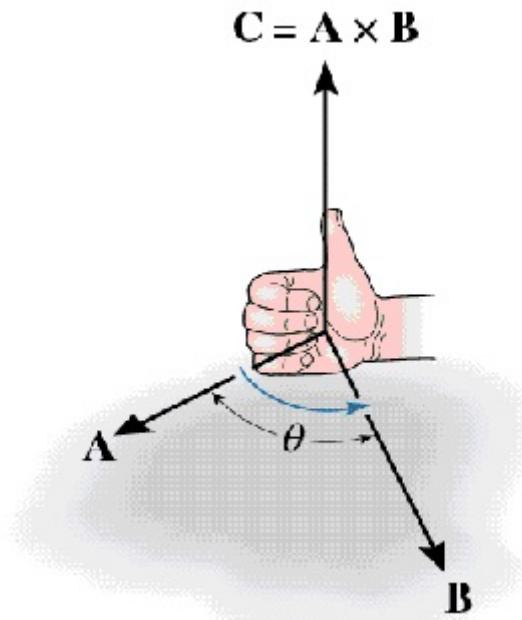
Ketiga komponen tersebut memiliki relasi sebagai berikut:

$$i^2 = j^2 = k^2 = ijk = -1$$

Hasil dari perkalian dua *kuaternion* memiliki aturan yang lebih rumit, sehingga memiliki aturan-aturan khusus. Berikut aturan-aturan khususnya :

$$\begin{aligned} ij &= k = -ji \\ jk &= i = -kj \\ ki &= j = -ik \end{aligned} \quad (2.6)$$

Ketiga persamaan di atas mirip dengan aturan tangan kanan (*right-hand rule*) pada perkalian cross product dari suatu vektor. Pada Gambar 2.8, A berperan sebagai i , B berperan sebagai j , dan C berperan sebagai k .



Gambar 2.8: Right-hand rule dalam *cross product* vektor

Seperti pada bilangan kompleks, kuaternion juga memiliki konjugasinya. Konjugasi kuaternion ini digunakan untuk melakukan operasi rotasi tiga dimensi (Akan dijelaskan pada subbab Operasi pada Kuaternion). Sama dengan konjugasi pada bilangan kompleks, kuaternion yang bilangan riilnya sama, dan bilangan imajinernya berlawanan dengan kuaternionnya disebut dengan konjugasi kuaternion. Oleh karena itu konjugasi kuaternion dari notasi kuaternion 2.5 adalah:

$$q = q_0 - iq_1 - jq_2 - kq_3 \quad (2.7)$$

Operasi pada Kuaternion

Dua buah kuaternion dapat dikatakan identik jika dan hanya jika kedua kuaternion memiliki komponen yang identik.

$$p = p_0 + ip_1 + jp_2 + kp_3$$

dan,

$$q = q_0 + iq_1 + jq_2 + kq_3$$

maka $p = q$ jika dan hanya jika

$$\begin{aligned} p_0 &= q_0 \\ p_1 &= q_1 \\ p_2 &= q_2 \\ p_3 &= q_3 \end{aligned} \quad (2.8)$$

Penjumlahan dari kedua kuaternion di atas dapat diartikan sebagai komponen penjumlahan yaitu:

$$(p + q) = (p_0 + q_0) + i(p_1 + q_1) + j(p_2 + q_2) + k(p_3 + q_3)$$

Peralihan dari kedua kuaternion di atas dapat diartikan sebagai komponen perkalian yaitu:

$$pq = (p_0 + ip_1 + jp_2 + kp_3)(q_0 + iq_1 + jq_2 + kq_3)$$

Begitu pula untuk komponen pengurangan dengan pembagian. Dari keempat operasi kuaternion tersebut, operasi kuaternion yang digunakan untuk rotasi bidang tiga dimensi adalah operasi perkalian. Fungsi rotasi vektor dapat menggunakan operasi kuaternion seperti pada bilangan kompleks, dengan rumus:

$$v' = qvq^*$$

dengan,

- $v = 1 + x_v i + y_v j + z_v k$
- $q = q_0 + iq_1 + jq_2 + kq_3$
- $q^* = q_0 - iq_1 - jq_2 - kq_3$
- $v' = 1 + x_{v'} i + y_{v'} j + z_{v'} k$

Seperti pada bilangan kompleks, bilangan q_0, iq_1, jq_2, kq_3 akan memiliki nilai sebagai berikut jika suatu kuaternion ingin digunakan untuk rotasi tiga dimensi:

$$\begin{aligned} q_0 &= \cos\left(\frac{\theta}{2}\right) \\ q_1 &= \sin\left(\frac{\theta}{2}\right)x_f \\ q_2 &= \sin\left(\frac{\theta}{2}\right)y_f \\ q_3 &= \sin\left(\frac{\theta}{2}\right)z_f \end{aligned} \tag{2.9}$$

dengan vektor f (x_f, y_f, z_f) merupakan sumbu perputaran dan θ merupakan besar sudut putar berlawanan arah dengan jarum jam.

2.4 Unity Game Engine

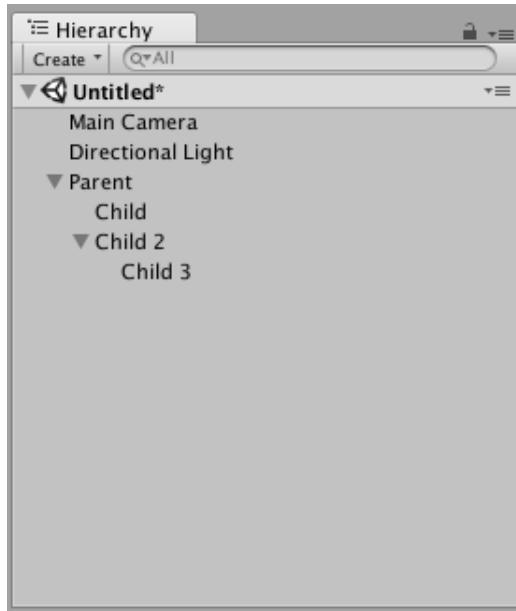
Unity merupakan salah satu *Game Engine* dalam pembuatan permainan dua dimensi atau tiga dimensi[10]. Unity merupakan *Game Engine multi-platform* sehingga permainan yang dibuat menggunakan Unity dapat berjalan di berbagai macam perangkat seperti *smartphone*, komputer, *console*, dan lain-lain. *Game Engine* ini menggunakan bahasa C# atau Javascript.

Game Engine ini dapat diperoleh secara gratis, yaitu dengan menggunakan Unity Personal Edition. Game Engine edisi Personal ini tidak berlaku jika penggunaan unity mendapatkan pendapatan kotor tahunan sebesar USD 100.000. Jika pendapatan kotor tahunan sudah melebihi USD 100.000, pengguna unity personal harus segera mengganti edisi Unity menjadi Unity Plus, Unity Pro, atau Unity Enterprise. Jika menggunakan Unity Personal, *splashscreen* dan logo *icon* pada permainan yang dibuat tidak dapat diganti maupun dihapus.

2.4.1 Struktur Hierarki GameObject

Hierarki pada unity merupakan kumpulan dari GameObject. GameObject dijelaskan lebih lanjut pada subbab 2.4.4. Beberapa dari GameObject pada hierarki ini merupakan instansi dari file *assets*. Setiap *scene* akan memiliki hierarki masing-masing. Setiap objek yang ditambahkan pada suatu *scene*(dijelaskan pada subbab 2.4.3) akan muncul pada hierarki juga.

Pada hierarki ini juga ada konsep *parent-child objects*. Konsep ini digunakan untuk setiap kumpulan objek. Setiap objek yang berada paling luar dinamakan "*parent object*", dan objek-objek yang berada didalamnya dinamakan "*child object*". Suatu *parent object* juga dapat memiliki parent objek(biasa disebut juga "*nested parent object*"). Contoh dari konsep *parent-child* ada pada Gambar 2.9.



Gambar 2.9: Contoh Hierarchy Parenting.

2.4.2 Prefabs

Unity memiliki suatu jenis assets yang dinamakan Prefab. Prefab ini adalah suatu GameObject yang disimpan menjadi assets. Prefab berperan sebagai contoh atau *blueprint* dari suatu GameObject. Prefab ini juga dapat dibentuk dari GameObject yang telah dibuat dan menjadi assets, sehingga dapat digunakan pada proyek lain. Prefab juga data diubah dan di-modifikasi sehingga menjadi sesuai dengan yang diinginkan.

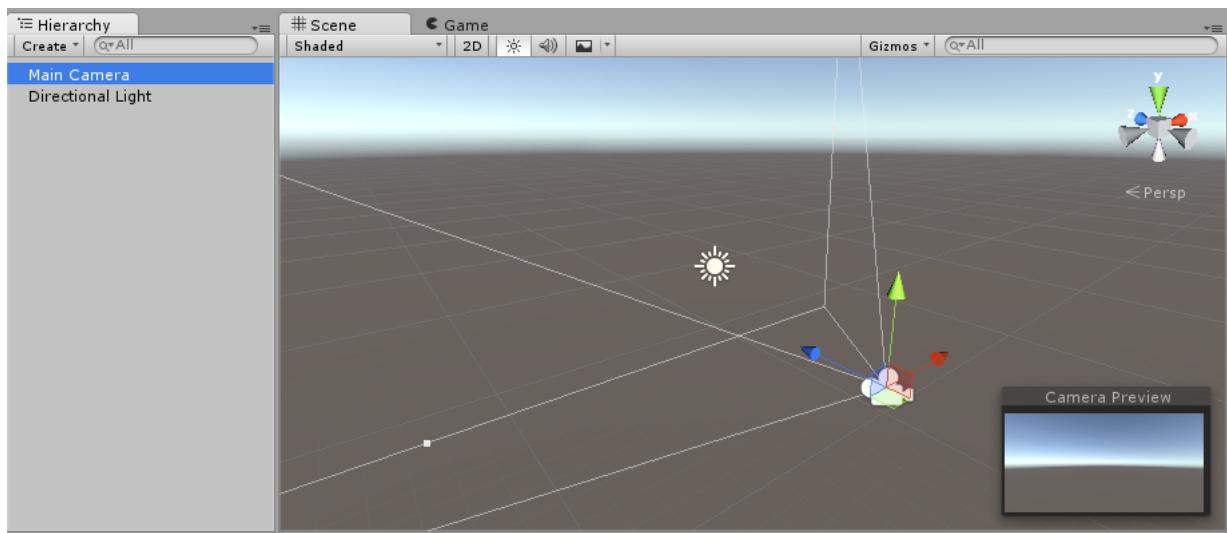
Dalam membuat Prefab dapat dengan mudah me-*drag and drop* suatu GameObject pada tab Scene ke tab Assets. Begitupula sebaliknya, untuk menggunakan suatu Prefab, dapat dengan mudah me-*drag and drop* dari tab Assets ke tab Scene.

2.4.3 Scene

Scene menyimpan seluruh objek pada permainan yang dibuat. *Scene* dapat digunakan untuk membuat *main menu*, *permainan level*, dan lain sebagainya. Untuk setiap *scene*, akan ditempatkan

barang, bangunan, dekorasi, dan lain sebagainya untuk merancang dan membangun suatu permainan bagian per bagian.

Pada saat pembuatan proyek pertama, unity membuatkan suatu *scene* baru. *Scene* tersebut merupakan *scene default*. *Scene* tersebut hanya diberikan objek-objek *default* saja seperti kamera dan cahaya. Kamera yang diberikan antara dua buah jenis yaitu *orthographic camera* atau *perspective camera*, proyek 2D diberikan *orthographic camera* dan proyek 3D diberikan *perspective camera*. Contoh dari *scene* default pada proyek 3D ditunjukkan pada Gambar 2.10.



Gambar 2.10: Contoh *scene* kosong.

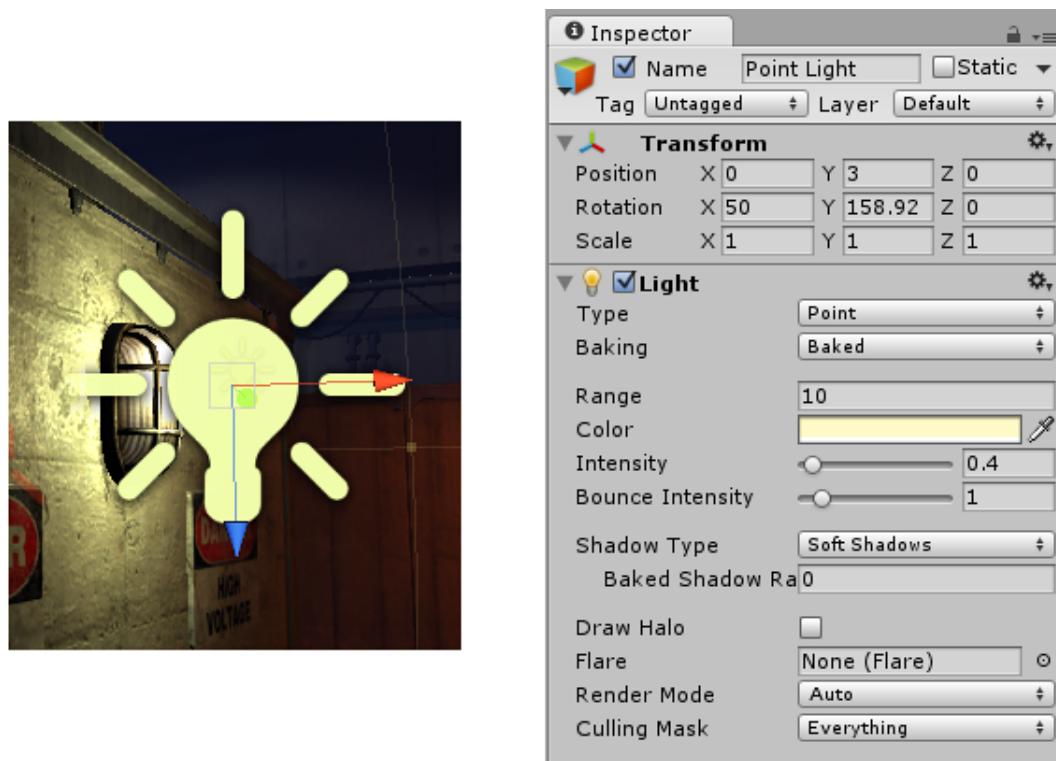
2.4.4 GameObject

GameObject pada unity yang merupakan representasi karakter, barang, dan pemandangan. *GameObject* tidak dapat melakukan apa pun dengan sendirinya, tetapi *GameObject* melakukan sesuatu sesuai dengan komponen yang ada pada *GameObject*. Komponen menjalankan fungsi-fungsi nyata pada *GameObject*. Contohnya objek cahaya dapat terbuat dengan menambahkan komponen Light kepada suatu *GameObject* (Gambar 2.11).

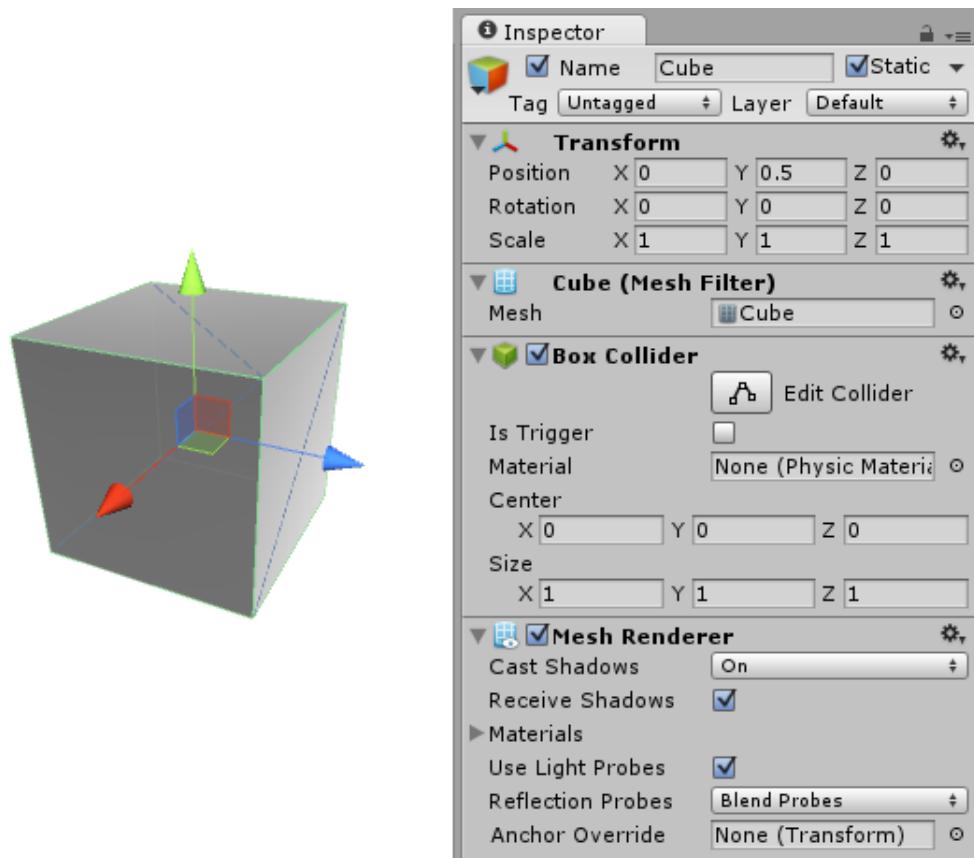
Selain itu ada pula contoh *GameObject* Kubus. *GameObject* ini memiliki komponen Cube(Mesh Filter) dan Mesh Renderer. Kedua komponen tersebut berguna untuk menggambar permukaan dari kubus tersebut. Selain itu ada juga komponen Box Collider yang digunakan untuk menggambarkan *GameObject* tersebut dalam hal-hal fisika seperti gravitasi, gaya, dan lain sebagainya(Gambar 2.12).

2.4.5 Scripting

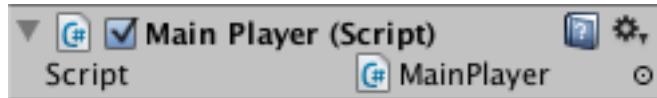
Pembuatan permainan pada Game Engine Unity membutuhkan *script* untuk memberikan logika permainan (*Game Logic*). Pembuatan *script* dapat dilakukan dengan membuat suatu kelas yang merupakan turunan kelas MonoBehavior agar dapat berjalan sesuai dengan sistem internal Unity. *Script* pada Unity bersifat seperti blueprint, sehingga *script* tidak dijalankan hingga *script* tersebut dihubungkan dengan suatu *GameObject*. Menghubungkan *script* dengan *GameObject* dapat dilakukan dengan menambahkan sebuah Component *Scripts* seperti yang ditunjukkan pada Gambar 2.13.



Gambar 2.11: Contoh penggunaan komponen pada GameObject.



Gambar 2.12: Contoh komponen pada GameObject kubus.



Gambar 2.13: Contoh Component *script*.

Dalam *scripting* pada Unity, ada sejumlah *method* yang dieksekusi dalam urutan yang telah ditentukan. Pada skripsi ini hanya dijelaskan fungsi-fungsi yang penting saja, untuk penjelasan lebih detail dijelaskan pada alamat web "Unity - Manual"². Perintah-perintah eksekusi ini dijelaskan di bawah ini:

- **Ketika Pertama Kali Scene Dimuat**

- Awake:

Fungsi ini selalu dipanggil sebelum fungsi Start dan juga tepat setelah Prefab diinstansiasikan. (Jika GameObject tidak aktif saat aplikasi baru dimulai, fungsi Awake tidak dipanggil, sampai GameObject diaktifkan).

- OnEnable:

Fungsi ini dipanggil sesaat setelah objek diaktifkan. Hal ini terjadi ketika contoh Mono-Behaviour dibuat, seperti saat level/scene dimuat atau GameObject dengan komponen *script* diinstansiasikan.

- **Sebelum Frame Pertama Dimuat**

- Start:

Fungsi ini dipanggil sebelum frame pertama dimuat. Untuk setiap objek yang ditambahkan ke *scene*, fungsi Start dipanggil pada semua *script* sebelum fungsi Update dipanggil.

- **Urutan Pemanggilan Update**

- FixedUpdate:

FixedUpdate lebih sering dipanggil dibandingkan dengan fungsi Update. Fungsi ini dapat terpanggil beberapa kali pada suatu *frame* jika *frame rate* rendah. Pada saat *frame rate* tinggi fungsi ini selalu terpanggil minimal satu kali setiap frame. Fungsi ini biasanya digunakan untuk melakukan implementasi perhitungan-perhitungan fisika pada permainan.

- Update:

Fungsi Update dipanggil di setiap *frame*. Fungsi ini merupakan fungsi inti untuk pergantian pada frame.

- LateUpdate:

LateUpdate dipanggil di setiap *frame*, setelah pemanggilan fungsi Update. Semua perhitungan pada Update dipastikan telah dieksekusi pada saat fungsi ini dipanggil. Fungsi ini biasanya digunakan ketika mengimplementasi *third-person camera*, karena *third-person*

²<https://docs.unity3d.com/Manual/class-ScriptExecution.html>

camera perlu memastikan kalkulasi pergerakan pemain sudah dieksekusi sebelum menggerakkan kamera tersebut.

- **Ketika Aplikasi Berhenti**

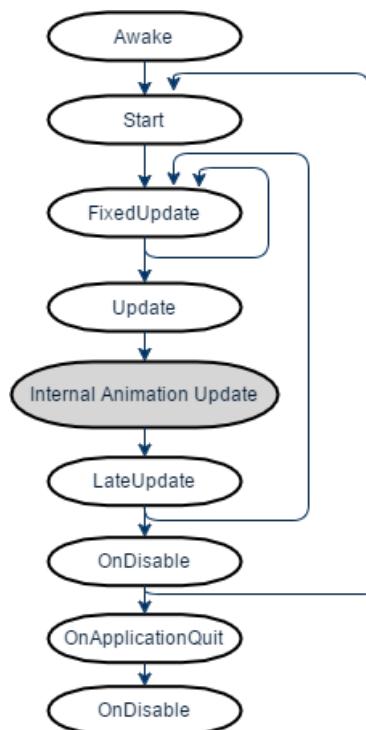
- OnApplicationQuit:

Fungsi ini dipanggil ketika Aplikasi berhenti.

- OnDisable:

Fungsi ini dipanggil ketika suatu tingkah laku (*script*) dinon-aktifkan.

- Flowchart Siklus Script Gambar 2.14 merupakan flowchart urutan pemanggilan fungsi-fungsi script.



Gambar 2.14: Flowchart urutan pemanggilan fungsi-fungsi.

2.4.6 Transformasi Rotasi dan Orientasi

Setiap GameObject selalu mempunyai komponen Transform untuk memberikan representasi posisi, orientasi, dan skala GameObject tersebut. Komponen ini tidak dapat dihapus. Komponen Transform ini nilainya relatif terhadap nilai komponen Transform pada *parent*-nya. Jika Transform ini tidak memiliki *parent*, nilainya relatif pada dunianya (*world space*).

Komponen Transform memiliki beberapa atribut. Atribut-atribut yang penting pada komponen ini adalah position, lossyScale, dan rotation. Atribut position menyimpan posisi dari suatu GameObject. Atribut ini disimpan dengan menggunakan kelas Vector3. Atribut lossyScale menyimpan besar penskalaan suatu GameObject. Atribut lossyScale ini pun disimpan dengan menggunakan kelas Vector3. Atribut rotation digunakan untuk menyimpan orientasi dari suatu GameObject. Atribut rotation berbeda dengan atribut position dan lossyScale. Atribut rotation disimpan dengan menggunakan kelas Quaternion.

Rotasi pada aplikasi tiga dimensi biasanya menggunakan antara metode Kuaternion atau sudut Euler. Masing-masing memiliki kelebihan dan kekurangan. Unity menggunakan Kuaternion dalam memberikan representasi rotasi dan orientasi. Pada kelas Kuaternion terdapat atribut yang menyimpan nilai-nilai orientasi pada sudut Euler, sehingga juga unity dapat memberikan representasi rotasi dan orientasi dengan menggunakan sudut Euler.

Kelebihan menggunakan representasi sudut Euler adalah:

- Sudut Euler lebih mudah untuk dipahami dalam format tiga sudut pada sumbu-sumbu tiga dimensi.
- Sudut Euler dapat memberikan representasi rotasi dari satu orientasi ke orientasi dengan sumbu yang lebih besar dari 180 derajat.

Kekurangan menggunakan representasi sudut Euler adalah:

- Sudut Euler dapat mengalami *Gimbal Lock*. *Gimbal Lock* adalah kejadian ketika dua sumbu dari sudut perputaran pada sudut Euler berputar pada poros yang sama.

Kelebihan menggunakan representasi Kuaternion adalah:

- Kuaternion tidak mengalami *Gimbal Lock*.

Kekurangan menggunakan representasi Kuaternion adalah:

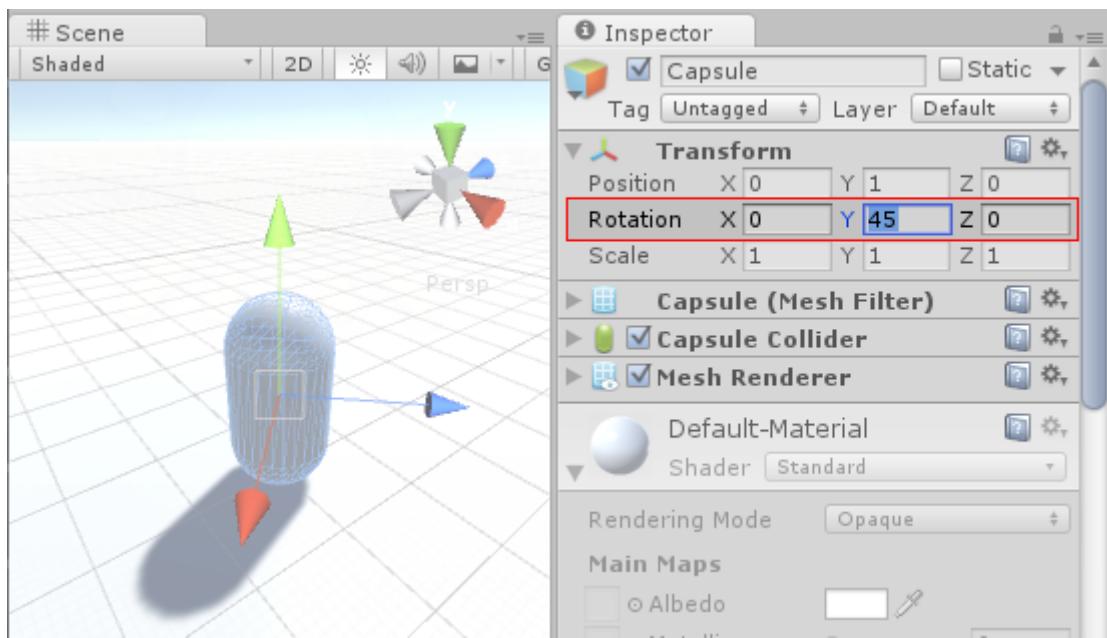
- suatu kuaternion tidak dapat memberikan representasi rotasi yang melebihi 180 derajat pada arah mana pun.
- Representasi angka-angka pada Kuaternion lebih susah untuk dimengerti.

Pada Unity, meski pun semua orientasi disimpan dengan menggunakan Kuaternion, tetapi pada *Transform Inspector* (Gambar 2.15) Unity menampilkan nilai-nilai orientasi pada sudut Euler. Hal ini bertujuan untuk mempermudah mengubah orientasi bagi pengguna, karena nilai-nilainya lebih mudah untuk dimengerti. Representasi pada *Transform Inspector* tersebut menyebabkan masukan yang diberikan oleh pengguna terkadang berbeda dengan data yang disimpan. Contohnya ketika berada pada sudut 365 derajat, maka data yang disimpan menunjukkan perputaran sebesar 5 derajat saja.

2.5 Google VR SDK for Unity

Google juga menyediakan SDK untuk *Game Engine* Unity. Berbeda dengan Google Cardboard API, Google VR SDK for Unity tidak hanya berfungsi untuk membuat aplikasi untuk Google Cardboard. Google VR SDK juga berfungsi untuk membuat aplikasi untuk Google Daydream, yaitu Aplikasi VR dari Google sebagai pembaruan Google Cardboard. Google daydream memiliki kelebihan memiliki *remote controller* yang dapat merekam gerakan tangan pengguna. Jadi jika menggunakan Google VR SDK for Unity, pembuat permainan juga dapat membuat pengaturan untuk *remote controller* pada Google Daydream.

Untuk mendapatkan Google VR SDK for Unity, SDK tersebut dapat diunduh pada halaman situs web Google VR Developers. Pada situs tersebut diberikan assets package dengan ekstensi file



Gambar 2.15: Rotasi dari Game Object ditampilkan dan di-edit pada *Inspector* dengan menggunakan metode sudut Euler, tetapi disimpan dalam memori dengan menggunakan metode Kuaternion.

".unitypackage". Assets package tersebut memasukkan seluruh file-file pendukung seperti gambar, tekstur, dan lain-lain untuk membuat Google VR pada proyek permainan yang sedang dikerjakan.

Untuk dapat me-*render* permainan dengan tampilan *stereo rendering* (tampilan layar yang dibagi menjadi dua bagian sesuai dengan mata manusia), hierarki GameObject pada scene yang sedang dikerjakan harus memiliki GvrViewerMain dengan komponen script GvrViewer. GameObject ini dapat diperoleh dengan menggunakan *Prefabs* yang telah di-*import*. Selain untuk me-*render*, permainan object tersebut juga melakukan implementasi orientasi pada *smartphone* secara langsung.

Seperti yang sudah dijelaskan Google Cardboard memberikan masukan dengan menarik/menelek pelatuk pada Google Cardboard tersebut. Untuk mendapatkan masukan tersebut, dapat dilakukan dengan cara menggunakan *Prefab* GvrEventSystem. Kemudian menambahkan komponen Event Trigger pada GameObject yang merespons masukan tersebut. Respons dapat dilakukan dengan menambahkan Script baru pada GameObject tersebut.

BAB 3

ANALISIS

Pada bab ini dijelaskan mengenai analisis grafik dari data sensor-sensor pada *smartphone* ketika sedang mengangguk dan menggeleng, analisis aplikasi-aplikasi sejenis, dan analisis metode pendeteksian gerakan kepala.

3.1 Analisis Aplikasi Sejenis

Pada saat skripsi ini dibuat, aplikasi sejenis pada perangkat VR Google Cardboard hanya ada satu aplikasi saja. Aplikasi tersebut adalah aplikasi InMind VR. Aplikasi sejenis pada perangkat VR Oculust Rift adalah Trial of the Rift Drifter dengan Asunder yang dikembangkan oleh AldinDynamics [11]. Aplikasi sejenis pada VR Oculust Rift tidak dapat dianalisis karena penulis tidak memiliki perangkat Oculust Rift.

Analisis InMind VR

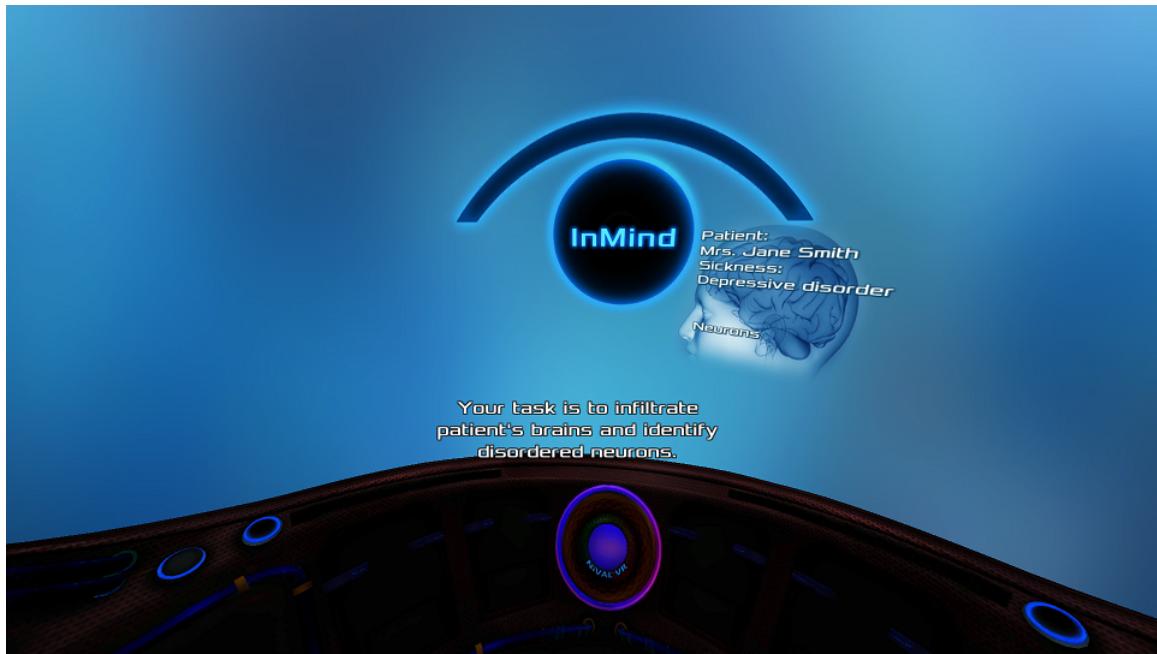
InMind VR merupakan permainan VR yang seolah-olah pengguna berada dalam sel-sel otak seorang manusia. Permainan ini dimainkan dengan mengarahkan arah pandang kepala ke arah sel-sel neuron yang dapat menyebabkan gangguan mental (Gambar 3.1). Ada 50 sel neuron yang dapat menyebabkan gangguan mental. Sel-sel yang dapat menyebabkan gangguan mental adalah sel-sel yang berwarna merah seperti pada Gambar 3.2.

Pada permulaan permainan pengguna diberi pertanyaan apakah pengguna sudah siap untuk melakukan permainan tersebut seperti yang ditunjukkan pada Gambar 3.3.

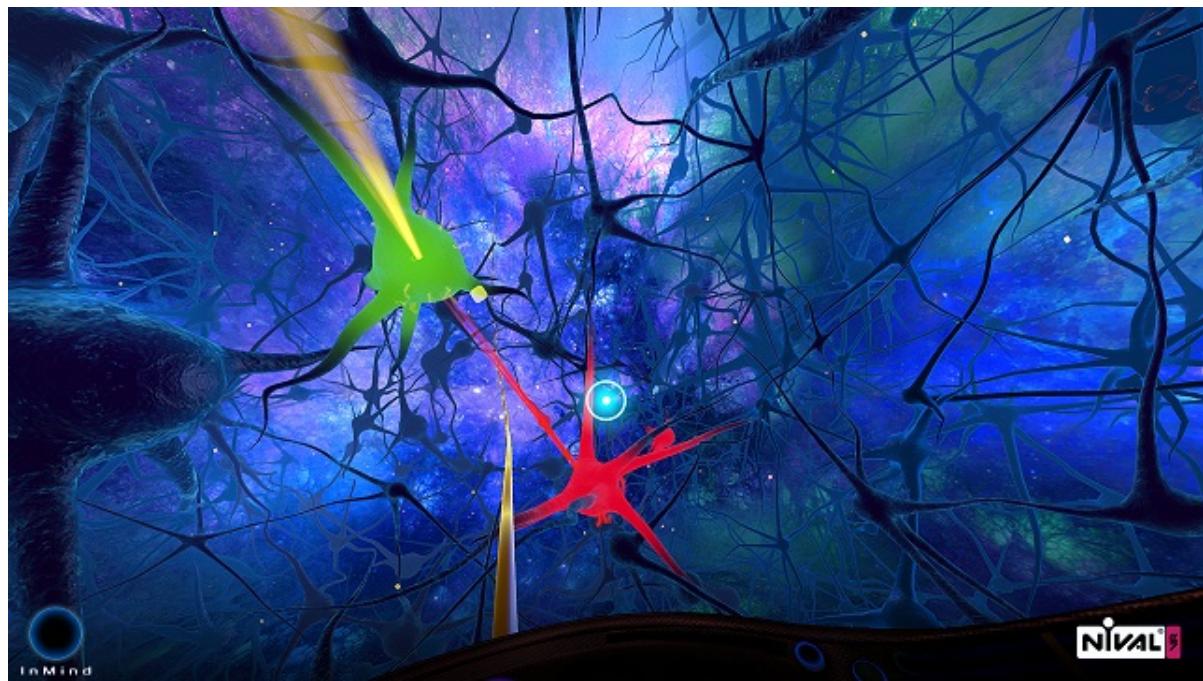
Analisis dilakukan dengan cara mengetes aplikasi ini dengan mengangguk dan menggeleng yang spesifik. Mengangguk dilakukan dengan cara-cara berikut:

1. Mengangguk secara pelan.
2. Mengangguk yang diawali dengan gerakan ke atas.
3. Melihat kebawah saja tanpa membalikkan kepala ke posisi semula.
4. Tidak menggerakkan kepala sama sekali.

Keempat percobaan tersebut memiliki tujuan masing-masing. Percobaan pertama dilakukan untuk apakah gerakan secara perlahan diartikan sebagai gerakan mengangguk atau tidak. Percobaan kedua dilakukan untuk mengetahui apakah gerakan anggukan dengan gerakan keatas terlebih



Gambar 3.1: *Screenshot* task yang diberikan aplikasi InMind VR.



Gambar 3.2: *Screenshot* aplikasi InMind VR ketika permainan berlangsung.

dahulu dianggap mengangguk atau tidak. Percobaan ketiga dilakukan untuk mengetahui apakah gerakan melihat kebawah dianggap mengangguk atau tidak. Percobaan keempat dilakukan untuk mengetahui respon dari aplikasi jika pengguna tidak melakukan gerakan anggukkan atau tidak.

Dari keempat percobaan mengangguk yang dilakukan, hanya percobaan pertama dengan percobaan ketiga yang terdeteksi mengangguk. Dapat disimpulkan dengan hanya menggerakkan kepala ke bawah saja sudah dapat dianggap mengangguk oleh aplikasi ini. Pada percobaan kedua dan keempat terjadi keganjilan dari pendektsian anggukan. Pada percobaan kedua dan keempat aplikasi tetap melanjutkan permainan setelah beberapa detik berlalu. Sel-sel yang dapat menyebabkan gangguan mental adalah sel-sel yang berwarna merah seperti pada Gambar 3.2.

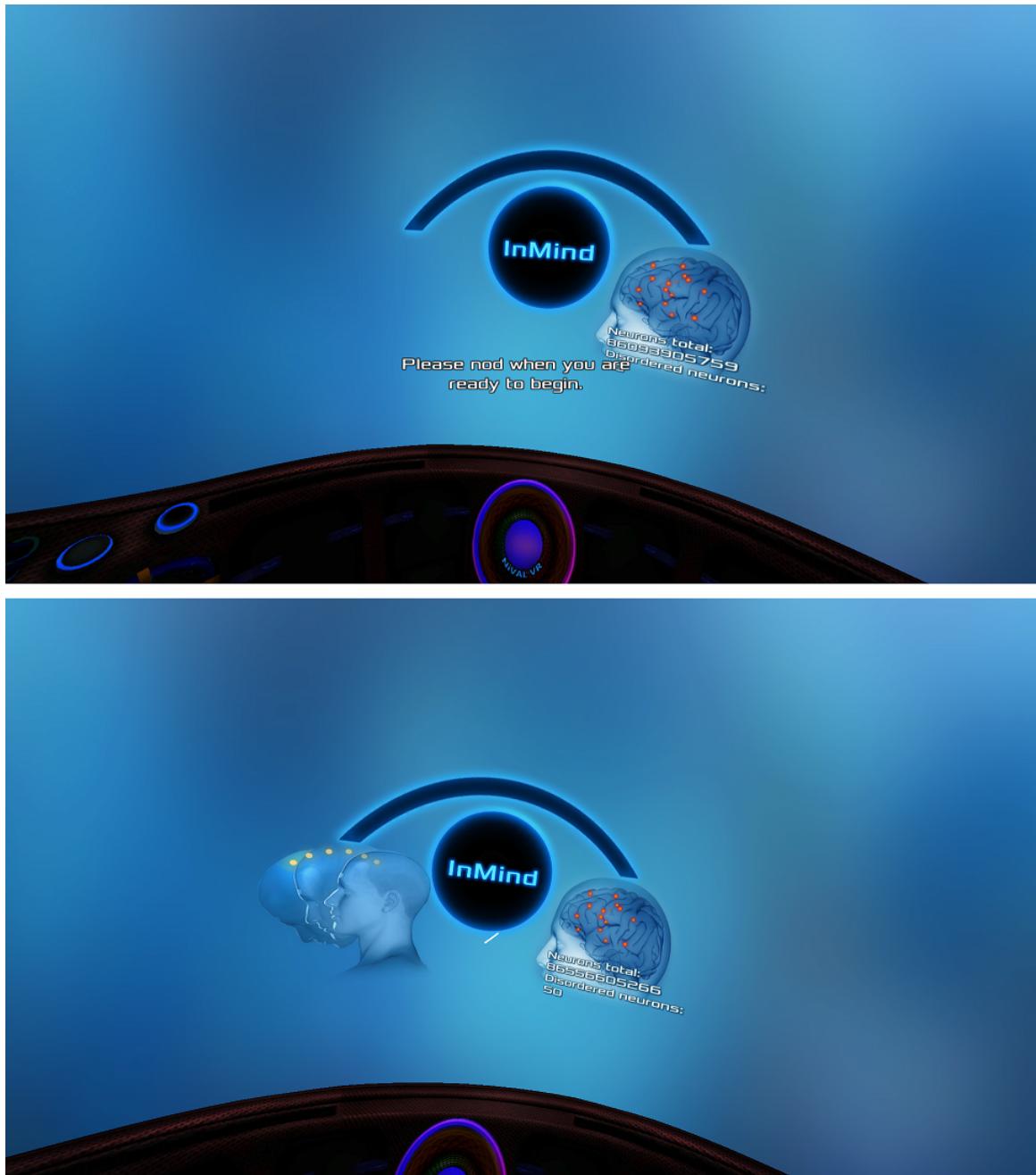
Ketika dilakukan percobaan menggeleng, tidak ada respons apa pun dari aplikasi ini. Aplikasi ini tetap melanjutkan ke permainan setelah beberapa detik telah berlalu. Hal tersebut menyimpulkan bahwa aplikasi ini tidak dapat mendekksi gerakan menggeleng. Oleh karena itu hal yang dilakukan oleh aplikasi ini hanyalah melihat apakah pengguna sudah melihat ke bawah atau belum saja.

3.2 Perekaman Data Sensor

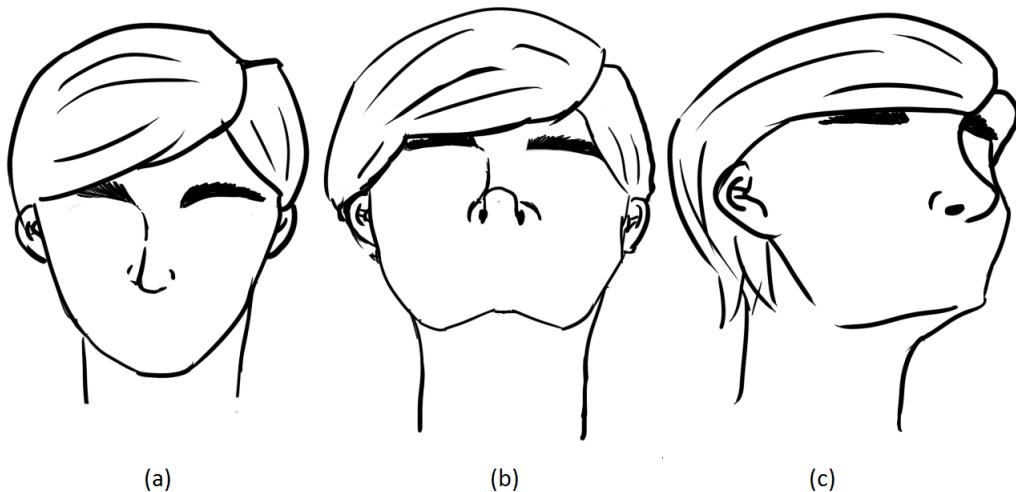
Pada analisis ini dilakukan perekaman mengangguk dan menggeleng dengan sensor-sensor pada Android. Perekaman-perekaman ini dilakukan pada tiga kondisi muka pengguna. Kondisi muka yang pertama adalah kondisi muka pengguna ketika menghadap ke depan, digambarkan dengan Gambar 3.4 bagian (a). Kondisi muka yang kedua adalah kondisi muka pengguna ketika menghadap ke atas sekitar 45° dari pandangan muka menghadap ke atas digambarkan dengan Gambar 3.4 bagian (b). Kondisi muka yang ketiga adalah kondisi muka ketika menghadap serong ke kiri atas digambarkan dengan Gambar 3.4 bagian (c). Anggukan yang dilakukan oleh pengguna hanya sebanyak satu kali mengangguk ke bawah saja. Sedangkan dalam menggeleng bergerak ke kiri terlebih dahulu dan ke kanan setelahnya dan diakhiri pada posisi muka kembali ke posisi awal. Perekaman grafik hanya akan dilakukan satu kali saja untuk setiap posisi. Setiap posisi akan dilakukan gerakan mengangguk dan menggeleng masing-masing satu kali.

Grafik-grafik yang ditunjukkan pada bab ini memiliki beberapa karakteristik. Sumbu y pada grafik menunjukkan representasi besar nilainya, sedangkan sumbu x menunjukkan representasi waktunya. Grafik yang ditunjukkan memiliki beberapa nilai, tergantung dari jumlah nilai yang dikembalikan untuk setiap sensornya. Contohnya pada sensor *accelerometer* yang memiliki tiga jenis nilai, pada grafik terbentuk tiga buah garis nilai. Aplikasi merekam beberapa sensor secara langsung ketika pengguna mengangguk atau menggeleng, sehingga nilai waktu sama untuk kondisi muka yang sama walaupun sensornya berbeda.

Analisis grafik data sensor-sensor pada Android dilakukan dengan membuat suatu aplikasi perrekam sensor-sensor yang ada pada *smartphone* Android terlebih dahulu. Aplikasi ini merekam nilai-nilai yang dihasilkan dari sebagian sensor-sensor pada android setiap ada perubahan. Pada skripsi ini, nilai sensor-sensor yang dibutuhkan adalah sensor-sensor yang merupakan sensor pendekksi gerak. Seperti yang sudah dijelaskan subbab 2.1.4, sensor gerak meliputi *accelerometer*, sensor gravitasi, *gyroscope*, dan *rotation vector*. Karena sensor gravitasi merupakan sensor *accelerometer* yang dikembangkan sehingga tidak digunakan pada analisis grafik sensor ini. Sehingga sensor-sensor yang digunakan untuk dianalisis adalah sensor *accelerometer*, *gyroscope*, dan *rotation vector*. Aplikasi menyimpan nilai sensor-sensor menggunakan format CSV (*Comma Separated Values*). Dari



Gambar 3.3: *Screenshot* aplikasi InMind VR ketika meminta pengguna untuk mengangguk jika telah siap.



Gambar 3.4: Gambar untuk mendeskripsikan posisi dari muka sebelum melakukan gerakan menggeleng atau mengangguk. Gambar (a) adalah kondisi muka ketika sedang menghadap ke depan. Gambar (b) adalah kondisi muka ketika sedang menghadap ke atas. Gambar(c) adalah kondisi muka ketika sedang menghadap ke serong kiri atas.

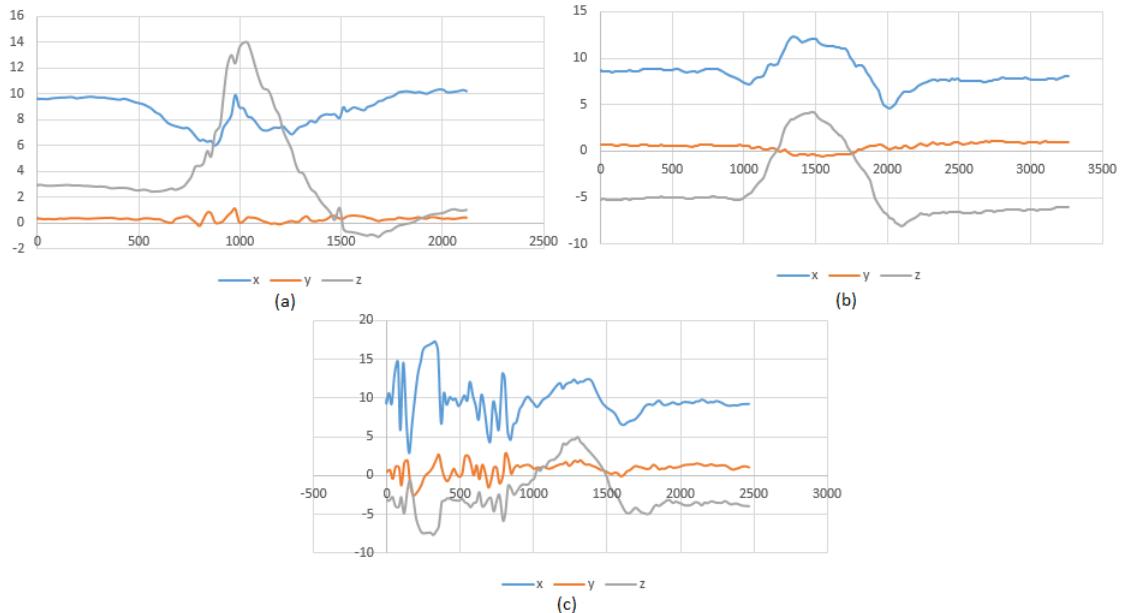
data yang diperoleh oleh aplikasi tersebut dibuat grafik-nya menggunakan aplikasi Microsoft Excel. Penjelasan dari setiap grafik dijelaskan pada subbab 3.2.1 hingga subbab 3.2.3.

3.2.1 Perekaman Grafik Sensor *Accelerometer*

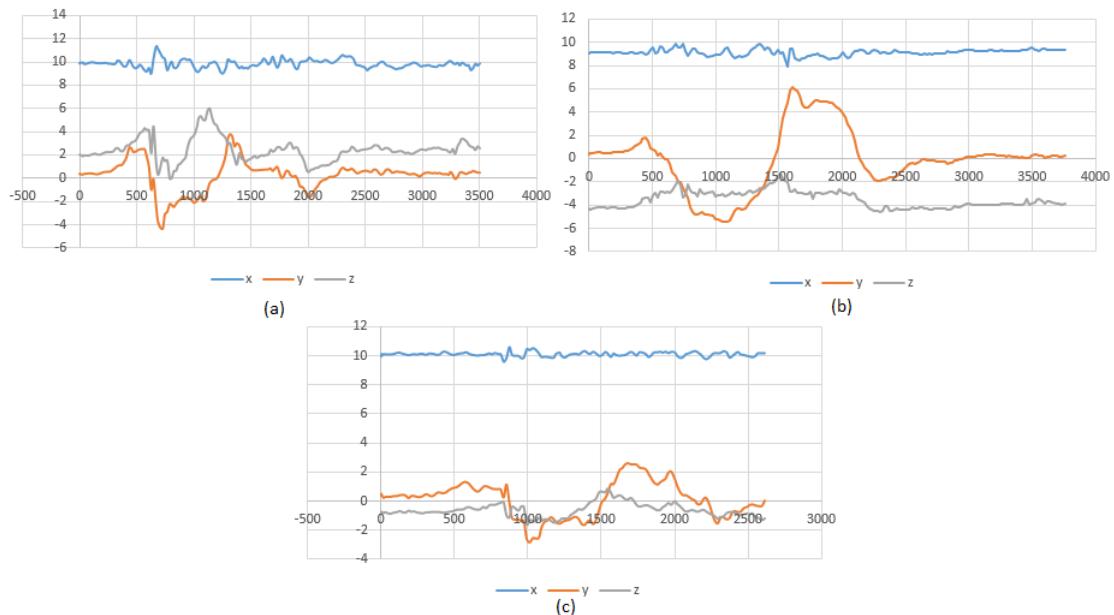
Seperti yang sudah dijelaskan pada bab sebelumnya, sensor *accelerometer* mendeteksi seluruh percepatan yang terjadi pada perangkat Android. Pada perekaman ini, perangkat Android diletakkan di depan muka pengguna, sehingga percepatan yang memengaruhi perangkat Android hanya percepatan gravitasi dengan percepatan yang dilakukan oleh gerakan kepala pengguna.

Gambar 3.5 merupakan grafik-grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna sedang mengangguk. Pada Gambar 3.5 grafik (a) terlihat nilai z menaik dan nilai x menurun ketika sedang mengangguk. Tetapi nilai x kembali menaik ketika nilai z sudah hampir mencapai nilai tertinggi. Sedangkan nilai y terlihat cukup konstan di sekitar angka 0. Pada grafik (b) terlihat nilai x dengan y memiliki pola yang serupa ketika mengangguk. Kedua nilai menaik ketika pengguna sedang mengangguk. Nilai x berada pada nilai sekitar sebesar 9 sedangkan nilai z bernilai sekitar sebesar -5. Sama seperti pada grafik (a) nilai y terlihat konstan di sekitar angka 0. Selanjutnya grafik (c) pada Gambar 3.5 merupakan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna mengangguk dan sedang menghadap ke kiri atas. Grafik pada bagian ini sangat tidak beraturan. Pada Grafik ini sulit untuk membedakan kondisi kapan pengguna sedang mengangguk. Goncangan yang terjadi terhadap *smartphone*, seperti munculnya notifikasi mungkin dapat menyebabkan hal ini. Namun pada waktu mencapai 1000 milidetik terlihat cukup stabil. Pada grafik (c) di Gambar 3.5 juga menunjukkan bahwa nilai x dengan z memiliki pola yang sama hingga akhir, dan nilai y konstan di sekitar angka 0. Hasil nilai tersebut serupa dengan kasus ketika menghadap ke atas.

Gambar 3.6 merupakan grafik yang terbentuk dari nilai yang diterima oleh sensor *accelerometer* ketika pengguna menggeleng. Pada grafik (a), nilai x konstan di sekitar angka 10. Nilai y dengan z



Gambar 3.5: Grafik nilai kuaternion dari sensor *accelerometer* ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.



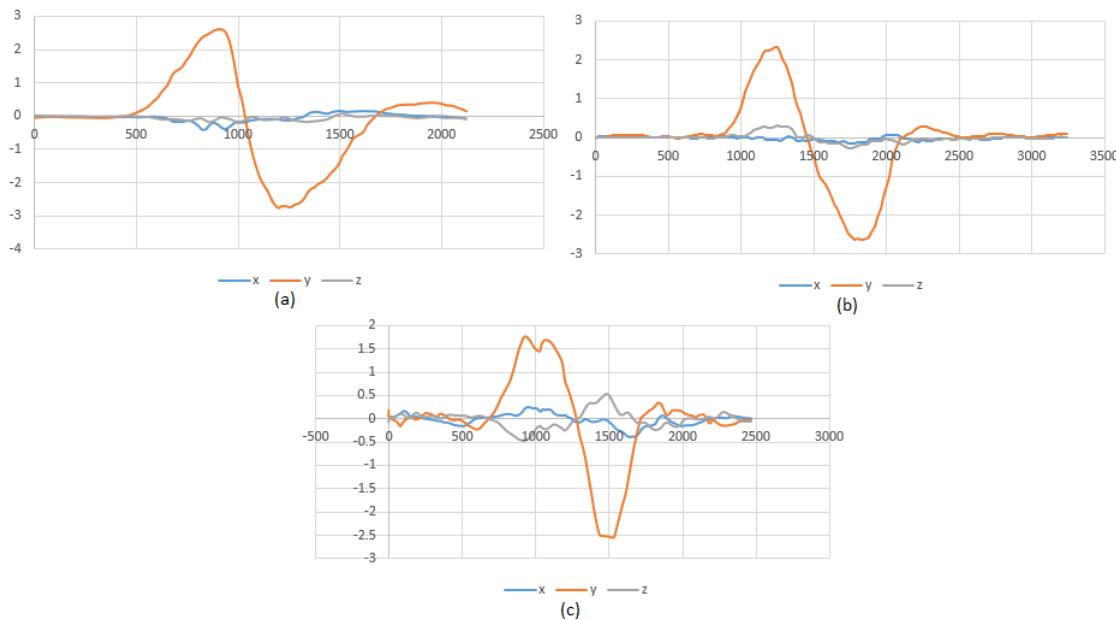
Gambar 3.6: Grafik nilai kuaternion dari sensor *accelerometer* ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

pada grafik (a) menaik dan menurun ketika pengguna menggelengkan kepala. Pada grafik (b) nilai x terlihat konstan di sekitar nilai 10 dan nilai z sedikit tidak beraturan di sekitar nilai -4 hingga -2. Nilai y mengalami kenaikan dan penurunan saat menggeleng. Pada grafik (c) di Gambar 3.6 nilai x konstan di sekitar nilai 10. Nilai y menaik dan menurun, tetapi tidak beraturan. Nilai pada z juga mengalami sedikit kenaikan dan penurunan. Pada grafik (b) dengan (c) nilai z mengalami perubahan yang tidak beraturan dan puncak tertinggi dengan puncak terendahnya tidak terlalu berbeda jauh dengan nilai awalnya.

Dari keenam grafik tersebut terlihat bahwa nilai yang terpengaruh ketika pengguna sedang mengangguk adalah nilai x dengan z. Nilai y tidak berpengaruh karena nilai y cenderung konstan. Nilai yang terpengaruh ketika pengguna sedang menggeleng adalah nilai y. Nilai x terlihat konstan pada setiap grafik, tetapi nilai z mengalami sedikit pergerakan ketika pengguna sedang mengangguk sehingga tidak dapat dipastikan bahwa nilai z terpengaruh gerakan menggeleng.

3.2.2 Perekaman Grafik Sensor *Gyroscope*

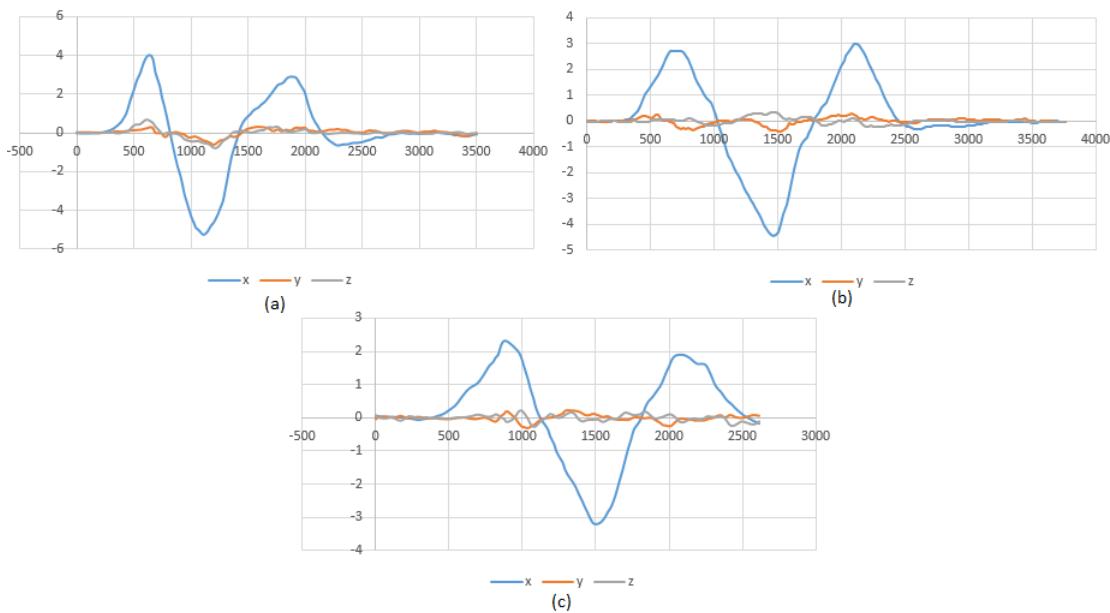
Perekaman menggunakan sensor *gyroscope* menghasilkan percepatan angular yang terjadi setiap waktunya. Berbeda dengan sensor *accelerometer* yang dapat terpengaruh oleh lingkungan sekitar seperti gravitasi dan percepatan lainnya, sensor ini hanya merekam perputaran yang terjadi pada perangkat saja. Hal ini sangat menguntungkan dalam mendeteksi suatu gerakan karena tidak harus memedulikan kasus dari pengaruh luar.



Gambar 3.7: Gambar grafik nilai sensor *gyroscope* ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

Gambar 3.7 menunjukkan nilai-nilai sensor *gyroscope* ketika pengguna sedang mengangguk. Grafik (a) membentuk sebuah *hill* dan *valley* pada nilai y. *Hill* yang terjadi menunjukkan ketika pengguna menggerakkan kepala ke bawah, dan ketika kepala pengguna kembali ke posisi semula percepatan angularnya berbalik arah sehingga menimbulkan *valley*. Nilai x dengan z cenderung bernilai 0. Grafik (b) mirip seperti grafik pada Gambar 3.7. Begitu pula pada grafik (c) yang

memiliki pola yang serupa dengan yang grafik-grafik sebelumnya.



Gambar 3.8: Gambar grafik nilai sensor *gyroscope* ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

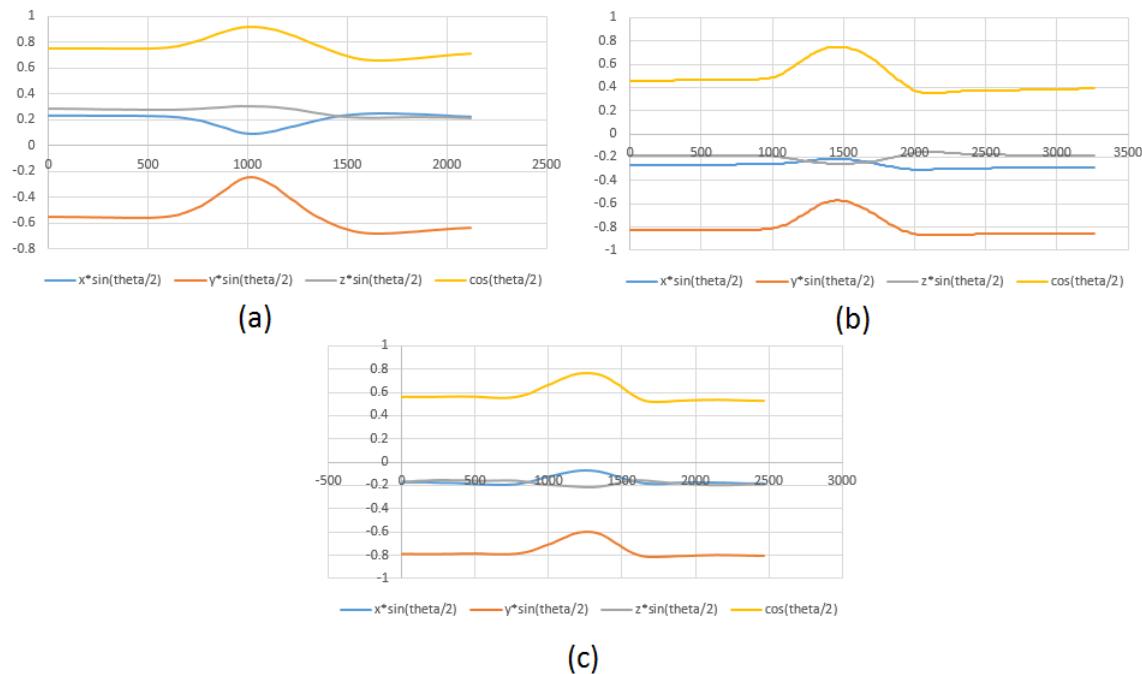
Gambar 3.8 menunjukkan nilai-nilai sensor *gyroscope* ketika pengguna sedang menggeleng. Pada grafik (a) nilai yang mengalami kenaikan dan penurunan adalah nilai x dan nilai-nilai lainnya cenderung berada pada nilai 0. Nilai x membentuk 2 buah *hill* dan 1 buah *valley*. *Hill* pertama terjadi ketika pengguna menggerakkan kepalamanya ke kiri. *Valley* pertama terjadi ketika pengguna menggerakkan kepalamanya ke kanan. *Hill* kedua terjadi ketika pengguna menggerakkan kepalamanya kembali ke posisi semula. Pada grafik (b) dengan grafik (c) menunjukkan pola grafik yang serupa dengan grafik pada Gambar (a).

Dari hasil-hasil tersebut dapat disimpulkan bahwa arah pandang pengguna tidak memengaruhi sensor *gyroscope* dalam mendekripsi gerakan kepala. Nilai-nilai yang dikembalikan oleh sensor *gyroscope* memiliki pola grafik yang jauh lebih rapi dibandingkan grafik-grafik yang dihasilkan oleh sensor *accelerometer*. Selain itu sensor *gyroscope* hanya menggunakan 1 jenis nilai yang dipengaruhi oleh pergerakan kepala, sedangkan *accelerometer* ada 2 jenis nilai yang dipengaruhi gerakan kepala pada saat mengangguk. Oleh karena itu sensor *gyroscope* lebih baik dalam mendekripsi gerakan yang terjadi pada perangkat android.

3.2.3 Perekaman Grafik Sensor *Rotation Vector*

Perekaman menggunakan sensor *rotation vector* menghasilkan sebuah kuaternion yang merupakan representasi perputaran yang terjadi pada perangkat Android. Perputaran ini diartikan dengan suatu vektor sebagai sumbu putarnya dan sudut perputarannya. Berbeda dengan sensor *gyroscope* yang merekam kecepatan perputaran yang terjadi pada suatu waktu, sensor *rotation vector* mengembalikan nilai kuaternion untuk mendefinisikan suatu kondisi putaran pada saat itu.

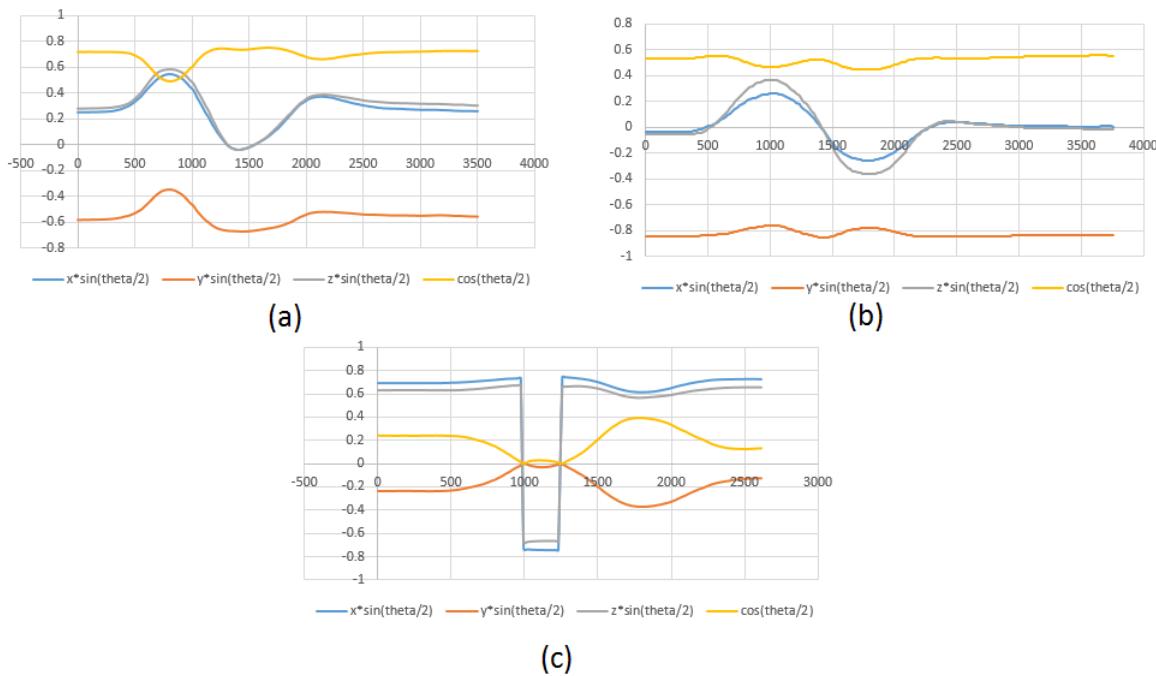
Gambar 3.9 menunjukkan nilai-nilai sensor *rotation vector* ketika pengguna sedang mengangguk. Pada grafik (a) terbentuk sebuah *hill* pada garis berwarna kuning, dan *valley*



Gambar 3.9: Grafik nilai kuaternion dari sensor *rotation vector* ketika pengguna mengangguk dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

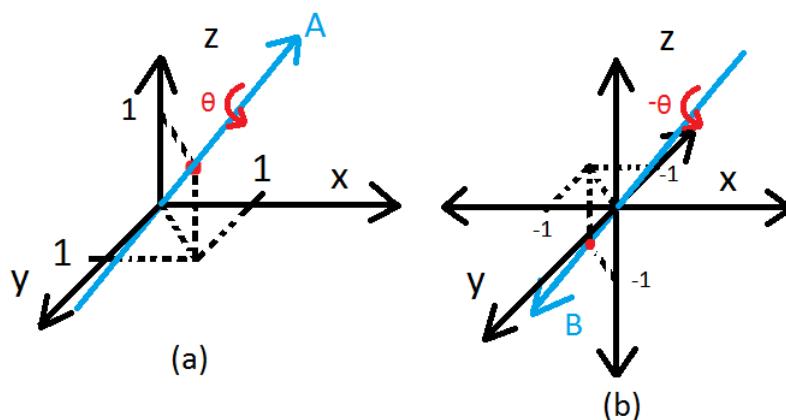
pada garis berwarna biru. Garis yang berwarna abu cenderung stabil di angka 0.3. Grafik (b) menunjukkan pola yang mirip pada bagian (a) namun berbeda nilainya saja. Pada grafik (a) garis berwarna kuning dimulai pada angka sekitar 0.7, sedangkan pada grafik (b) garis berwarna jingga dimulai pada angka sekitar 0.4. Garis biru pada grafik ini tidak membentuk sebuah *valley* seperti pada grafik pada grafik (a). Garis berwarna abu cenderung konstan pada nilai -0.2, sedangkan pada grafik (a) cenderung konstan di sekitar 0.3. Garis berwarna jingga memiliki pola yang sama dengan garis berwarna kuning, hanya berbeda pada nilainya saja. Seperti pada grafik-grafik sebelumnya, grafik (c) ini memiliki pola yang sama dengan grafik lainnya. Grafik ini juga hanya nilainya saja yang berbeda dengan grafik lainnya. Kemiripan pola ini memungkinkan mempermudah pendekripsi gerakan kepala.

Gambar 3.10 menunjukkan nilai-nilai sensor *rotation vector* ketika pengguna sedang menggeleng. Pada grafik (a) terbentuk sebuah *hill* yang diikuti dengan *valley* pada garis berwarna biru dengan abu-abu. Garis berwarna kuning membentuk suatu *valley* yang setelahnya cenderung konstan. Berbeda pada garis kuning yang membentuk *hill* kemudian setelahnya cenderung konstan. Grafik (b) memiliki kemiripan dengan grafik (a), garis biru dengan garis abu-abu memiliki pola yang sama yaitu membentuk *hill* dengan *valley* ketika pengguna menggelengkan kepala. Garis kuning dengan jingga cenderung konstan, berbeda dengan grafik (a) yang membentuk *hill* atau *valley*. Pada grafik (c) garis-garis membentuk suatu pola yang tidak normal. Garis kuning membentuk sebuah *valley*, tetapi membentuk suatu *hill* kecil ketika nilainya mencapai nilai 0 pada mili detik ke 1000. Garis jingga memiliki pola yang berlawanan dengan garis kuning. Pada saat garis kuning dengan garis jingga mencapai angka 0 perubahan drastis pun terjadi pada garis biru dengan abu-abu. Pada mili detik ke 1000 garis biru dengan abu mengalami perubahan nilai yang sangat drastis. Kedua nilai tersebut berubah dari nilai yang berkisar diantara 0.6 sampai 0.7 menjadi berkisar diantara -0.7



Gambar 3.10: Grafik nilai kuaternion dari sensor *rotation vector* ketika pengguna menggeleng dengan posisi muka awal (a) menghadap ke depan, (b) menghadap ke atas, (c) menghadap ke kiri atas.

sampai -0.8. Kasus ini tidak berarti sensor sedang mengalami kegagalan akurasi (*accuracy fail*), tetapi memang seperti itulah karakteristik dari perputaran menggunakan kuaternion pada android. Perputaran yang terjadi pada batas mendekati sebelum terjadinya dengan setelah perubahan nilai yang drastis memiliki hasil perputaran yang sama. Hal ini disebabkan karena melakukan perputaran dengan vektor sebagai sumbu dapat memiliki 2 buah nilai yang sejenis. Dua buah nilai tersebut menghasilkan perputaran yang sama ketika arah vektor dengan arah putarnya dibalikkan seperti yang ditunjukkan pada Gambar 3.11. Sepertinya pada sistem android nilai $\cos(\theta/2)$ didesain agar tidak bernilai negatif, sehingga nilai-nilai yang lainnya mengalami perubahan nilai yang drastis ketika nilai $\cos(\theta/2)$ mencapai angka 0.



Gambar 3.11: Dua buah rotasi yang identik dengan nilai kuaternion yang berbeda.

3.3 Analisis Data Sensor untuk Mendeteksi Gerakan Kepala

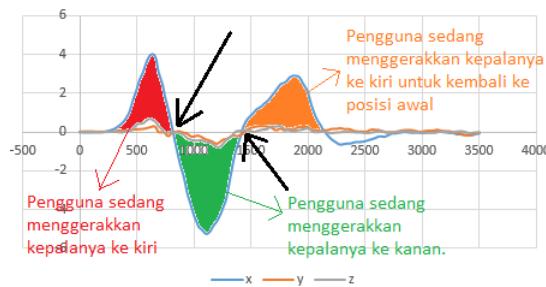
Dari ketiga hasil percobaan pada sensor-sensor pada Bab 3.2 dapat disimpulkan bahwa sensor *gyroscope* adalah sensor yang terbaik untuk mendeteksi gerakan kepala. Data dari sensor *accelerometer* sulit untuk digunakan dalam mendeteksi gerakan kepala karena terganggu dengan aktivitas-aktivitas di luar gerakan pengguna yang juga ikut terekam oleh sensor *accelerometer*. Data dari sensor *rotation vector* juga lebih rumit dibandingkan sensor *gyroscope*. Hal ini karena perputaran yang direkam oleh sensor *rotation vector* merekam kondisi putar pada suatu saat. Hasil rekaman ini mempersulit pada saat pendekripsi gerakan kepala karena harus melakukan proses untuk menghitung kecepatan kepala bergerak, agar dapat membedakan gerakan mengangguk atau menggeleng atau hanya menoleh biasa. Dalam mendeteksi gerakan mengangguk dengan menggeleng banyak batas-batas yang perlu diperhatikan. Batas-batas tersebut untuk mengetahui apakah pengguna benar-benar mengangguk atau menggeleng atau hanya menoleh biasa. Batas-batas yang perlu diperhatikan adalah:

- Kecepatan pengguna menoleh.
- Selang waktu antara gerakan kepala pengguna yang berlawanan.
- Simpangan terbesar kepala saat mengangguk atau menggeleng.

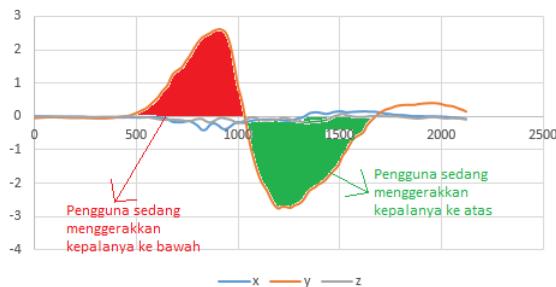
Kecepatan pengguna menjadi batas karena gerakan kepala yang kecepatannya cenderung pelan biasanya bukan merupakan gerakan mengangguk atau pun menggeleng. Kecepatan ini dapat langsung diperoleh menggunakan sensor *gyroscope*. Kecepatan yang dibutuhkan adalah kecepatan perputaran maksimum yang dilakukan oleh pengguna. Kecepatan maksimum dapat diperoleh dengan mengambil nilai puncak tertinggi dengan terendah. Jika nilai puncaknya mencapai kecepatan tertentu, gerakan tersebut dapat diperkirakan merupakan gerakan mengangguk atau pun menggeleng.

Gerakan menggeleng atau mengangguk biasanya memiliki selang waktu yang sangat sempit, karena pengguna biasanya langsung melawan arah gerakan kepala secara langsung ketika sedang mengangguk atau pun menggeleng. Nilai ini dapat diperoleh dengan menghitung jarak antara *hill* dengan *valley* yang terbentuk pada grafik seperti yang dijelaskan pada Gambar 3.12 dan Gambar 3.13. Idealnya gerakan menggeleng tidak memiliki rentang waktu ini, namun mungkin sebagian orang masih menghasilkan rentang waktu yang cukup sedikit. Oleh karena itu mungkin batas waktu yang ditentukan di sini adalah sekitar 100 milidetik. Jika rentang waktunya melebihi batas waktu tersebut, maka gerakan tersebut mungkin bukan merupakan gerakan mengangguk atau pun gerakan menggeleng.

Simpangan terbesar ini juga penting untuk dijadikan batasan-batasan dalam mendeteksi gerakan mengangguk atau pun menggeleng. Simpangan kepala yang sangat kecil dapat diragukan untuk dianggap sebagai gerakan mengangguk atau menggeleng. Simpangan ini dapat diperoleh dengan menghitung luas yang dibentuk dari *hill* atau *valley* yang terbentuk pada grafik. Contoh pada Gambar 3.12 simpangan terbesar yang terjadi ketika pengguna menggerakkan kepalanya pertama kali ke kiri adalah luas pada bidang yang diarsir merah. Simpangan terbesar yang terjadi ketika pengguna menggerakkan kepalanya ke kanan adalah luas bidang yang di-arsir berwarna hijau. Con-



Gambar 3.12: Grafik pada saat pengguna menggeleng.



Gambar 3.13: Grafik pada saat pengguna mengangguk.

toh pada Gambar 3.13 simpangan terbesar yang terjadi ketika pengguna menggerakkan kepalanya ke atas adalah luas pada bidang yang diarsir merah.

3.4 Analisis Metode Pendeksi Gerakan Kepala

Seperti yang sudah dijelaskan pada bab 2.1.4, data didapatkan setiap ada perubahan nilai pada sensor dengan rentang waktu tertentu, bergantung dengan konfigurasinya. Pendeksi ini membutuhkan data yang *real-time*, sehingga lebih baik jika menggunakan konfigurasi kecepatan pengambilan data setiap 20 mikro detik. Penggunaan konfigurasi dengan kecepatan 0 mikro detik tidak terlalu baik, karena menggunakan kemampuan processor yang sangat tinggi.

3.4.1 Algoritma Mendeksi Gerakan Mengangguk

Algoritma dipanggil setiap kali ada perubahan nilai dari sensor. Algoritma ini menyimpan nilai-nilai batas-batas yang telah ditentukan, luas yang terbentuk, nilai tertinggi, waktu mulai, dan waktu akhir dari suatu *hill* atau *valley*. Algoritma ini dipanggil secara berulang-ulang hingga data yang dikumpulkan dapat menghasilkan suatu kesimpulan mengangguk. Data disimpan pada atribut, agar setiap pemanggilan dapat melanjutkan dari pemanggilan sebelumnya.

Berikut adalah keterangan dari data-data yang disimpan pada atribut:

- **currPassLimitUp**, atribut ini menunjukkan bahwa pada saat ini gerakan pengguna sedang bergerak ke atas dan melebihi batas kelebihan sudutnya.
- **currPassLimitDown**, atribut ini menunjukkan bahwa pada saat ini gerakan pengguna sedang bergerak ke bawah dan melebihi batas kelebihan sudutnya.

- `angSpeedLimit`, atribut ini menyimpan batas kecepatan sudut.
- `lastYAngSpeed`, atribut ini memiliki kecepatan sudut Y pada pemanggilan method sebelumnya.
- `lastT`, atribut ini memiliki waktu dipanggilnya method sebelumnya.
- `calcArea`, atribut ini menyimpan besar luas *hill* atau, *valley* yang terjadi. Luas *hill* bernilai positif, dan nilai *valley* bernilai negatif.
- `startTValley`, atribut ini menyimpan waktu mulai *valley* yang memenuhi syarat kecepatan sudut minimal.
- `endTValley`, atribut ini menyimpan waktu akhir *valley* yang memenuhi syarat kecepatan sudut minimal.
- `startTCurrMotion`, atribut ini menyimpan waktu mulai suatu gerakan(*hill* atau *valley*) yang sedang berlangsung sekarang.
- `endTCurrMotion`, atribut ini menyimpan waktu akhir suatu gerakan(*hill* atau *valley*) yang sedang berlangsung sekarang.
- `deviationLimit` adalah batas simpangan untuk melakukan gerakan mengangguk.
- `deviationMotionDown`, atribut ini menyimpan simpangan yang terjadi ketika pengguna menggerakkan kepalanya ke bawah.
- `deviationMotioUp`, atribut ini menyimpan simpangan yang terjadi ketika pengguna menggerakkan kepalanya ke atas.

Algoritma 1 mengembalikan nilai true jika pengguna sedang mengangguk. Baris ke-2 hingga baris ke-8 adalah untuk mengecek kecepatan putar sekarang, apakah sudah melampaui batas yang ditentukan atau belum. Untuk mengetahui titik potong secara akurat dapat menggunakan rumus persamaan garis dari dua buah titik yang dinotasikan dengan Notasi 3.1.

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \quad (3.1)$$

dengan,

$$\begin{aligned} y_1 &= lastYAngSpeed \\ y_2 &= yAngSpeed \\ x_1 &= lastT \\ x_2 &= currT \end{aligned}$$

Notasi 3.1 dijabarkan sesuai dengan penjabaran 3.2 sehingga menjadi rumus yang berada pada algoritma 1 baris ke-10. Nilai titik potong x ini juga menjadi indikator waktu untuk rentang waktu dari suatu *hill* atau *valley*, yang digunakan untuk mendapatkan jarak waktu pada saat pengguna melawan arah gerakan. Baris ke-14 hingga ke-25 pada algoritma 1 adalah untuk mengecek apakah

Algorithm 1 Nod Detection Algoritm

```

1: function DETECTNOD( $yAngSpeed$ )
2:    $currT \leftarrow$  current Time
3:   if  $yAngSpeed > angSpeedLimit$  then
4:      $currPassLimitUp \leftarrow$  true
5:   else if  $yAngSpeed < angSpeedLimit * -1$  then
6:      $currPassLimitDown \leftarrow$  true
7:   if X values intersect with x(time) axis then
8:      $xIntersect \leftarrow ((-lastYAngSpeed/(yAngSpeed - lastYAngSpeed)) * (currT - lastT)) +$ 
       $lastT$ 
9:      $endTCurrMotion \leftarrow xIntersect$ 
10:     $areaBeforeIntersect \leftarrow ((xIntersect - lastT)/1000) * lastYAngSpeed/2$ 
11:     $calcArea \leftarrow calcArea + areaBeforeIntersect$ 
12:    if  $currPassLimitDown$  then
13:       $startTValley \leftarrow startTCurrMotion$ 
14:       $endTValley \leftarrow endTCurrMotion$ 
15:       $deviationMotionDown \leftarrow calcArea$ 
16:      if  $deviationMotionDown < deviationLimit * -1$  then
17:         $passLimitDownDeviation \leftarrow$  true
18:    else if  $currPassLimitUp$  then
19:       $startTHill \leftarrow startTCurrMotion$ 
20:       $endTHill \leftarrow endTCurrMotion$ 
21:       $deviationMotionUp \leftarrow calcArea$ 
22:      if  $deviationMotionUp > deviationLimit$  then
23:         $passLimitUpDeviation \leftarrow$  true
24:       $calcArea \leftarrow 0$ 
25:       $areaAfterIntersect \leftarrow ((currT - xIntersect)/1000) * yAngSpeed/2$ 
26:       $calcArea \leftarrow calcArea + areaAfterIntersect$ 
27:       $startTCurrMotion \leftarrow xIntersect$ 
28:       $currPassLimitDown \leftarrow$  false
29:       $currPassLimitUp \leftarrow$  false
30:    else
31:       $calcArea \leftarrow calcArea + ((lastYAngSpeed + yAngSpeed)/1000) * (currT - lastT)/2$ 
32:       $lastYAngSpeed \leftarrow yAngSpeed$ 
33:       $lastT \leftarrow currT$ 
34:      if hill and valley time values has been set AND all condition have been met then return
          $true$ 
35:    else return  $false$ 

```

simpangan yang terjadinya sudah melewati batas yang ditetapkan atau belum dan mencatatnya ke dalam atribut `passLimitDownDeviation` atau `passLimitUpDeviation`. Pada baris ke-26 hingga ke-28 untuk memulai menghitung luas yang baru. Baris ke-32 dan baris ke-33 untuk menghitung luas ketika tidak berpotongan dengan sumbu x. Baris ke-36 untuk pengecekan jarak antara *hill* dengan *valley* dapat diselesaikan dengan mengurangi waktu mulai *hill* dengan waktu akhir *valley*. Sehingga hasil untuk mengecek apakah semua batas sudah terpenuhi dapat diselesaikan dengan operasi (AND) biasa.

$$\begin{aligned}
 y &= 0 \\
 \frac{0 - y_1}{y_2 - y_1} &= \frac{x - x_1}{x_2 - x_1} \\
 \frac{-y_1}{y_2 - y_1}(x_2 - x_1) &= x - x_1 \\
 x &= \left(\frac{-y_1}{y_2 - y_1}(x_2 - x_1) \right) + x_1
 \end{aligned} \tag{3.2}$$

3.4.2 Algoritma Mendeteksi Gerakan Menggeleng

Sama seperti pada pendekstian gerakan mengangguk, algoritma ini dipanggil setiap kali ada perubahan nilai sensor. Algoritma ini juga menyimpan batas-batas, waktu mulai dan berakhirnya suatu *hill* atau *valley*. Algoritma ini dipanggil setiap ada perubahan nilai sensor dan data-data yang diperoleh disimpan pada atribut. Algoritma ini juga dipanggil secara berulang-ulang hingga data yang dikumpulkan dapat menghasilkan suatu kesimpulan menggeleng.

Sebagian atribut memiliki kegunaan yang sama dengan algoritma pendekstian gerakan mengangguk. Berikut adalah keterangan dari data-data yang disimpan pada atribut yang belum dijelaskan pada pendekstian gerakan mengangguk:

- `currPassLimitLeft`, atribut ini menunjukkan bahwa pada saat ini gerakan pengguna sedang bergerak ke kiri dan melebihi batas kecepatan sudutnya.
- `currPassLimitRight`, atribut ini menunjukkan bahwa pada saat ini gerakan pengguna sedang bergerak ke kanan dan melebihi batas kecepatan sudutnya.
- `lastXAngSpeed`, atribut ini memiliki kecepatan sudut X pada pemanggilan method sebelumnya.
- `deviationMotionLeft`, atribut ini menyimpan simpangan yang terjadi ketika pengguna menggeraka kepalanya ke kiri.
- `deviationMotioRight`, atribut ini menyimpan simpangan yang terjadi ketika pengguna menggeraka kepalanya ke kanan.

Sebagian besar algoritma untuk mendekksi gerakan menggeleng memiliki kemiripan dengan algoritma untuk mendekksi gerakan mengangguk. Pada algoritma menggeleng data waktu mulai dan berakhirnya suatu *hill* atau *valley* disimpan pada atribut dengan tipe data *List*. Atribut-atribut yang menggunakan tipe data *List* ini adalah `valleyTimes` dan `hillTimes`. Data-data waktu terjadinya bukti atau *valley* yang dimasukkan ke dalam *List* ini diartikan sudah memenuhi syarat

Algorithm 2 Shook Detection Algorit

```

1: function DETECTSHOOK( $xAngSpeed$ )
2:    $currT \leftarrow$  current Time
3:   if  $xAngSpeed > angSpeedLimit$  then
4:      $currPassLimitLeft \leftarrow$  true
5:   else if  $xAngSpeed < angSpeedLimit * -1$  then
6:      $currPassLimitRight \leftarrow$  true
7:   if X values intersect with x(time) axis then
8:      $xIntersect \leftarrow ((-lastXAngSpeed/(xAngSpeed - lastXAngSpeed)) * (currT - lastT)) +$ 
       $lastT$ 
9:      $endTCurrMotion \leftarrow xIntersect$ 
10:     $areaBeforeIntersect \leftarrow ((xIntersect - lastT)/1000) * lastXAngSpeed/2$ 
11:     $calcArea \leftarrow calcArea + areaBeforeIntersect$ 
12:    if  $currPassLimitRight$  then
13:       $deviationMotionRight \leftarrow calcArea$ 
14:      if  $deviationMotionRight < deviationLimit$  then
15:         $valleyTimes.add(\text{start and end time valley})$ 
16:    else if  $currPassLimitLeft$  then
17:       $deviationMotionUp \leftarrow calcArea$ 
18:      if  $deviationMotionUp > deviationLimit * -1$  then
19:         $hillTimes.add(\text{start and end time hill})$ 
20:       $calcArea \leftarrow 0$ 
21:       $areaAfterIntersect \leftarrow ((currT - xIntersect)/1000) * xAngSpeed/2$ 
22:       $calcArea \leftarrow calcArea + areaAfterIntersect$ 
23:       $startTCurrMotion \leftarrow xIntersect$ 
24:       $currPassLimitRight \leftarrow false$ 
25:       $currPassLimitLeft \leftarrow false$ 
26:    else
27:       $calcArea \leftarrow calcArea + ((lastXAngSpeed + xAngSpeed)/1000) * (currT - lastT)/2$ 
28:       $lastXAngSpeed \leftarrow xAngSpeed$ 
29:       $lastT \leftarrow currT$ 
30:      if head moves right first AND time range between hill and valley no exceed the limit. then
31:        return true
32:      else if head moves left first AND time range between hill and valley no exceed the limit.
        then return true
33:      else return false

```

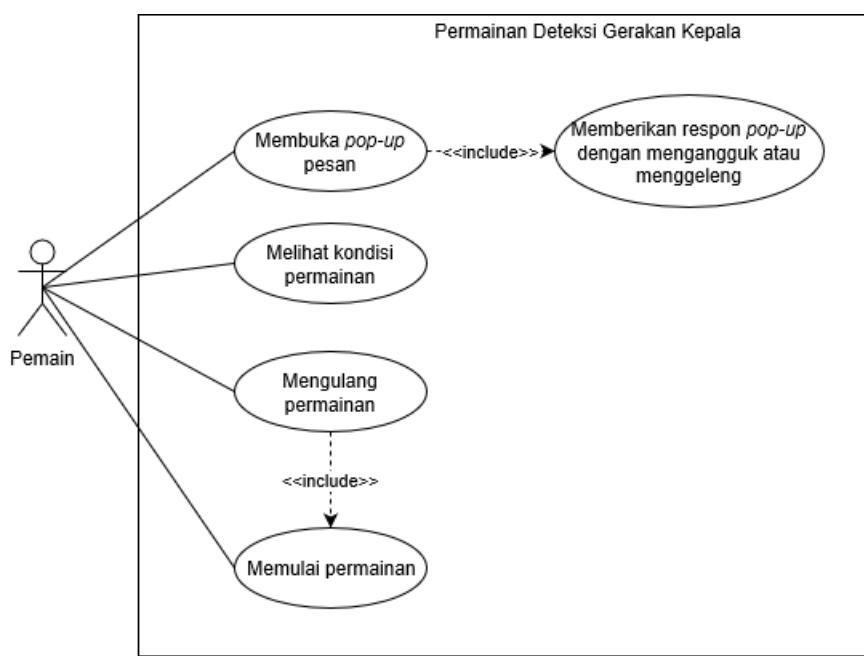
dari batas-batas simpangan dan kecepatan sudutnya. *List* ini dikosongkan kembali jika salah satu *List*-nya sudah lebih dari dua atau kedua *List* tersebut sudah memiliki isi sebanyak dua, atau lebih.

Untuk pengecekan jarak waktu antara *valley* dengan *hill* dilakukan sebanyak dua kali. Pengecekan pertama yaitu untuk pengecekan jarak waktu antara *valley* atau *hill* pertama dengan kedua. Pengecekan kedua yaitu untuk pengecekan jarak waktu antara *valley* atau *hill* kedua dengan ketiga. Perhitungan jarak waktu antara *valley* atau *hill* pertama dengan kedua dapat dilakukan dengan mengurangi waktu dimulainya *valley* atau *hill* kedua dengan waktu berakhirnya *valley* atau *hill* pertama. Begitu pula dengan perhitungan jarak waktu antara *valley* atau *hill* kedua dengan ketiga.

3.5 Analisis Aplikasi Permainan dengan Algoritma Deteksi Gerakan Kepala

Pada skripsi ini dibuat suatu aplikasi yang mengimplementasikan algoritma deteksi gerakan kepala. Aplikasi ini merupakan suatu permainan dengan menggunakan gerakan kepala sebagai salah satu masukan pada permainan ini. Masukkan dari gerakan kepala hanyalah mengangguk dan menggeleng. Masukkan ini hanya dapat digunakan ketika aplikasi membutuhkan jawaban dialog yang hanya dapat dijawab dengan ya atau tidak saja. Oleh karena itu permainan ini menggunakan pertanyaan dialog sebagai inti dari permainannya.

Aplikasi ini memiliki beberapa fungsi yang dimiliki. Fungsi utama dalam permainan ini yaitu dengan menggunakan anggukan dan gelangan sebagai masukan pada permainan. Fungsi-fungsi lainnya pada permainan ini dijelaskan dengan menggunakan diagram *use case* pada Gambar 3.14 dan skenario pada Tabel 3.1 hingga Tabel 3.1.



Gambar 3.14: Diagram *use case* aplikasi permainan deteksi gerakan kepala

Algoritma pendeksi gerakan kepala pada subbab 3.5 tidak memiliki desain kelas yang baik. Algoritma tersebut mengerjakan beberapa tugas berbeda dalam waktu yang sama, sehingga algoritma tersebut melanggar prinsip desain SOLID pertama yaitu Single Responsibility Principle.

Nama	Memulai Permainan
Deskripsi	Memulai permainan dari kondisi awal
Aktor	Pemain
Pre-kondisi	Aplikasi dimulai
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Sistem memuat aplikasi. 2. Sistem memulai permainan.

Tabel 3.1: Tabel skenario memulai permainan.

Nama	Membuka <i>pop-up</i> pesan
Deskripsi	Memunculkan pesan berupa <i>pop-up</i> pada pengguna dalam VR
Aktor	Pemain
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Pengguna melihat notifikasi baru yang dimunculkan oleh sistem. 2. Pengguna mengklik notifikasi. 3. Sistem memunculkan <i>pop-up</i> pesan.

Tabel 3.2: Tabel skenario membuka *pop-up* pesan.

Nama	Memberikan respons <i>pop-up</i> dengan menggeleng atau mengangguk
Deskripsi	Memberikan jawaban terhadap pesan di <i>pop-up</i> yang diberikan dengan menggunakan gerakan kepala, yaitu mengangguk dan menggeleng.
Aktor	Pemain
Pre-kondisi	<i>Pop-up</i> telah muncul.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Pengguna membaca pesan <i>pop-up</i> yang telah muncul. 2. Pengguna memberikan respons dengan mengangguk. 3. Pesan pada <i>pop-up</i> diubah menjadi "yes".
Alur Skenario Alternatif	<ol style="list-style-type: none"> 2a. Pengguna memberikan respons dengan menggeleng. 3a. Pesan pada <i>pop-up</i> diubah menjadi "no".

Tabel 3.3: Tabel skenario memberikan respons *pop-up* dengan menggeleng atau mengangguk.

Nama	Melihat kondisi permainan
Deskripsi	Melihat status kondisi permainan.
Aktor	Pemain
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Pengguna mengarahkan posisi kepala ke panel kondisi permainan. 2. Panel kondisi berubah-ubah seiring pemain memainkan permainannya.

Tabel 3.4: Tabel skenario melihat kondisi permainan.

Nama	Mengulang permainan
Deskripsi	Mengulang permainan kembali ke kondisi awal ketika permainan telah usai.
Aktor	Pemain
Pre-kondisi	Permainan telah usai.
Alur Skenario Utama	<ol style="list-style-type: none"> 1. Pengguna mengklik (Menarik pelatuk pada Google Cardboard). 2. Mengulang permainan dari awal.

Tabel 3.5: Tabel skenario mengulang permainan.

Sehingga perlu dibuat kelas-kelas baru untuk memisahkan tugas yang berbeda ke dalam suatu kelas baru tersebut. Oleh sebab itu perlu dibuat diagram kelas untuk menjelaskan kelas-kelas yang diperlukan.

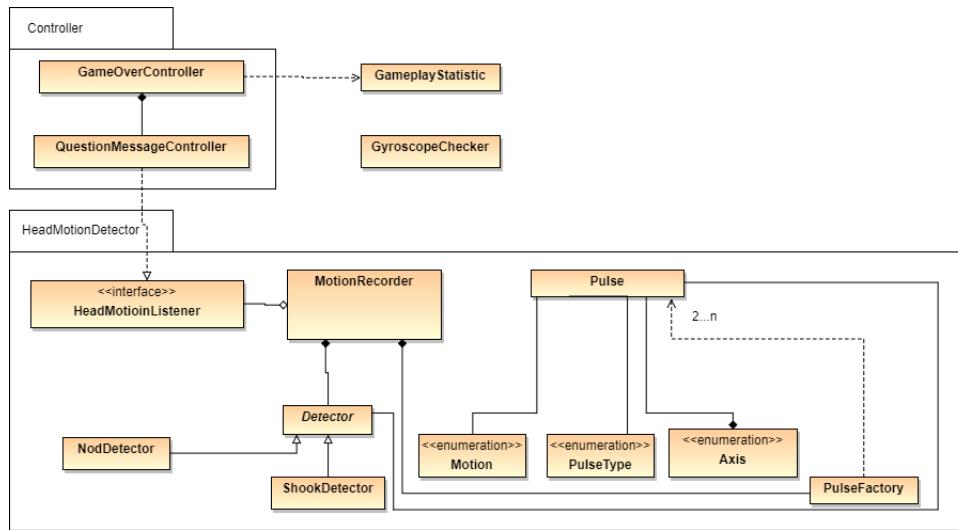
Pada subbab ini dibuat suatu diagram kelas sederhana untuk menjelaskan kelas-kelas yang dibutuhkan dalam membuat aplikasi permainan yang mengimplementasikan algoritma deteksi gerakan kepala. Aplikasi ini dibuat dengan menggunakan Game Engine Unity. Game Engine Unity menggunakan *design pattern* MVC (Model-View-Controller) dalam pengimplementasianya. Hal ini menyebabkan diagram kelas yang dibuat memiliki bentuk seperti *design pattern* MVC. Meski diagram kelas yang dibuat memiliki bentuk MVC, namun diagram kelas ini tidak memiliki bagian *view*-nya karena bagian *view* sudah diimplementasikan pada Game Engine Unity dengan menggunakan GameObject Hierarchy yang telah dijelaskan 2.4.1. Oleh karena itu diagram kelas ini hanya memiliki bagian *controller* dan bagian *model* saja. *Package* "Head Motion Detector" merupakan bagian yang serupa dengan *model* pada *design pattern* MVC. Gambar 3.15 merupakan diagram kelas permainan yang menggunakan algoritma deteksi gerakan kepala.

Berikut ini adalah penjelasan dari kelas-kelas pada diagram di Gambar 3.15.

- Kelas GameplayStatistic

Kelas ini bertujuan untuk menyimpan seluruh atribut-atribut statis pada keseluruhan permainan.

- Kelas GyroscopeChecker



Gambar 3.15: *Screenshot* task yang diberikan aplikasi InMind VR.

Kelas ini digunakan untuk mengecek apakah perangkat yang sedang menjalankan permainan ini memiliki sensor gyroscope.

- Package Controller
 - Kelas GameOverController

Kelas ini bertujuan untuk mendeteksi kondisi *game over* dan menampilkan pesan *game over*.
 - Kelas QuestionMessageController

Kelas ini digunakan untuk mengatur tampilan pesan pertanyaan pada permainan.
- Package HeadMotionDetector
 - interface HeadMotionListener

Interface ini di-*implement* oleh kelas-kelas yang menggunakan hasil dari pendekripsi gerakan kepala.
 - Kelas MotionRecorder

Kelas ini digunakan untuk merekam, menentukan, dan mendeteksi gerakan kepala.
 - Kelas PulseFactory

Kelas ini berguna untuk mendeteksi *Pulse* yang terjadi pada grafik data gyroscope.
 - Kelas Detector

Kelas ini merupakan kelas yang mendefinisikan pendekripsi gerakan kepala. Sub-class kelas ini adalah NodDetector dengan ShookDetector
 - Kelas NodDetector dengan ShookDetector

Kelas ini berisi algoritma untuk mendekripsi gerakan mengangguk atau menggeleng. Kelas ini merupakan turunan dari kelas Detector
 - Kelas Pulse

Kelas ini digunakan hanya untuk menyimpan karakteristik-karakteristik dari suatu *Pulse* yang terbentuk oleh kelas PulseDetector.

BAB 4

PERANCANGAN

Pada bab ini dijelaskan mengenai perancangan aplikasi yang dibangun meliputi perancangan kelas algoritma pendekripsi gerakan kepala, *GameObject*, *Gameplay*, dan perancangan antarmuka.

4.1 Perancangan Kelas Algoritma Pendekripsi Gerakan Kepala

Pada subbab 3.4 telah dijelaskan algoritma pendekripsi gerakan kepala secara prosedural. Untuk merapihkan algoritmanya, dibuatlah diagram kelas rinci untuk memenuhi kebutuhan dalam pembangunan aplikasi. Desain diagram kelas ini pada dasarnya membagi masalah-masalah pada algoritma di subbab 3.4 menjadi masalah-masalah yang lebih sederhana. Kemudian setiap masalah disatukan kembali untuk menyelesaikan masalah pendekripsi kepala. Deskripsi kelas beserta fungsi dari diagram kelas dijelaskan sebagai berikut:

- Kelas GameplayStatistic

Kelas ini bertujuan untuk menyimpan seluruh atribut-atribut statis pada keseluruhan permainan. Saat ini kelas ini hanya menyimpan atribut `isAnswered` saja, yang mengindikasikan bahwa suatu pertanyaan yang telah muncul telah dijawab.

- Kelas GyroscopeChecker

Kelas ini digunakan untuk mengecek apakah perangkat yang sedang menjalankan permainan ini memiliki sensor gyroscope. Kelas ini tidak memiliki hubungan kepada kelas lain karena kelas ini berjalan pada Scene yang berbeda dengan kelas lainnya. Pengecekan perangkat dilakukan pada saat aplikasi mulai dijalankan, oleh karena itu implementasi pengecekan perangkat terletak pada method `Awake()`.

- Package Controller
 - Kelas GameOverController

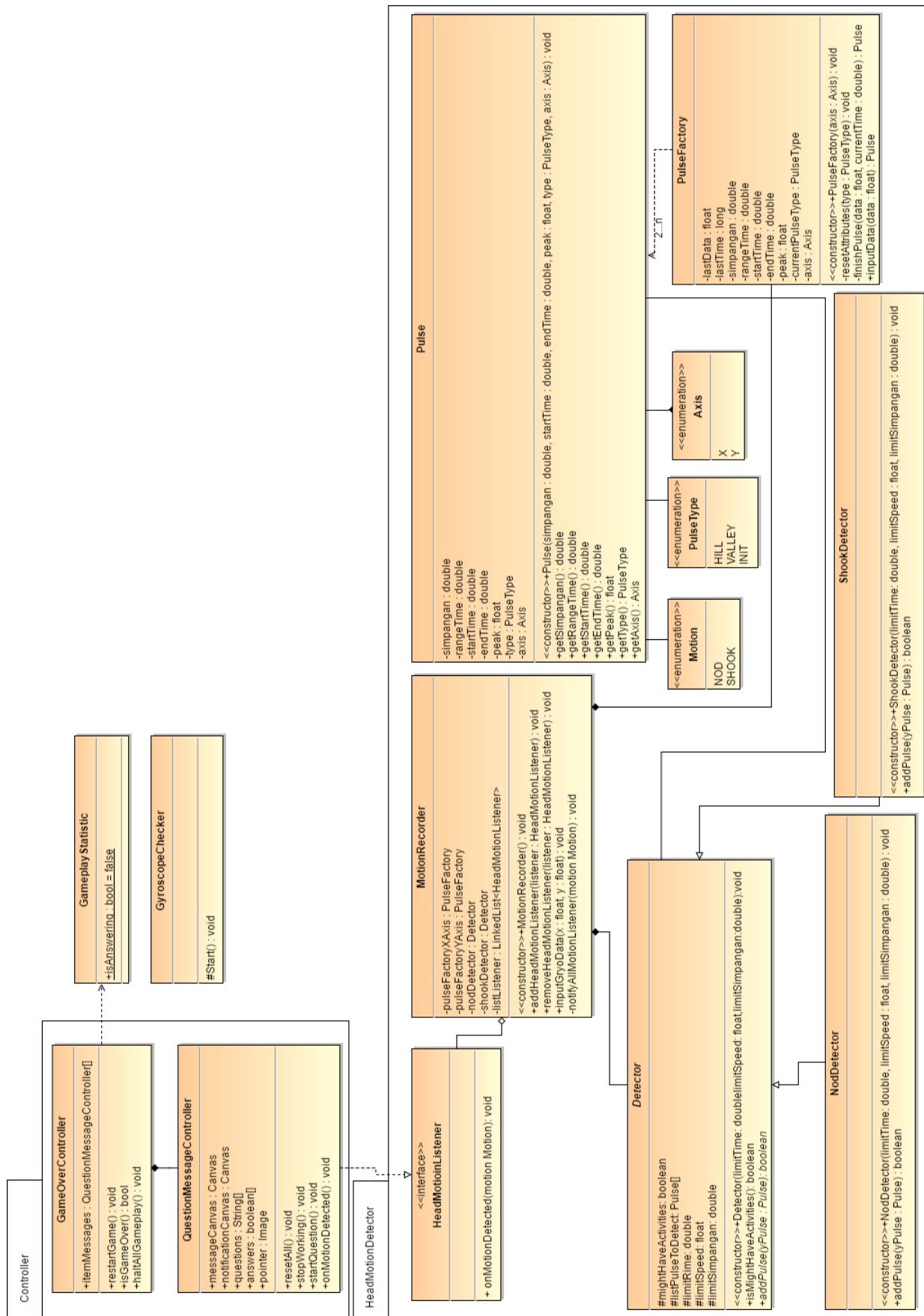
- Kelas GameOverController

Kelas ini bertujuan untuk mendekripsi kondisi *game over* dan menampilkan pesan *game over*. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * `public QuestionMessageController[] itemMessages`

Atribut ini digunakan untuk menyimpan seluruh `QuestionMessageController` yang ada pada permainan agar dapat diberhentikan dan diatur ulang.

Method-method pada kelas ini adalah sebagai berikut:



Gambar 4.1: Diagram Kelas Algoritma Pendekripsi Gerakan Kepala. (Gambar diputar sembilan puluh derajat searah jarum jam)

* public void restartGame()

Method ini berfungsi untuk mengembalikan kondisi permainan ke kondisi awal, untuk mengulang permainan jika permainan telah selesai.

* public boolean isGameOver()

Method ini berfungsi untuk mengecek apakah permainan sudah selesai atau belum.

Method ini dipanggil secara berulang-ulang setiap satu *frame* pada permainan.

* public void haltAllGameplay()

Method ini berfungsi untuk memberhentikan seluruh fungsi permainan. Method ini dipanggil ketika permainan telah selesai.

– Kelas QuestionMessageController

Kelas ini digunakan untuk mengatur tampilan pesan pertanyaan pada permainan. Kelas ini juga mengatur kondisi permainan bergantung dari masukan pengguna, yaitu anggukan dan gelangan kepala. Atribut-atribut pada kelas ini adalah sebagai berikut.

* public Canvas messageCanvas

Atribut ini digunakan untuk mendapatkan Canvas yang menampilkan pesan pertanyaan kepada pengguna.

* public Canvas notificationCanvas

Atribut ini digunakan untuk mendapatkan Canvas yang menampilkan pesan pemberitahuan kepada pengguna.

* public String[] question

Atribut ini digunakan untuk menyimpan pertanyaan-pertanyaan yang ditampilkan pada messageCanvas.

* public boolean[] answers

Atribut ini digunakan untuk menyimpan jawaban-jawaban penentu untuk setiap pertanyaan yang ada.

* public Image pointer

Atribut ini digunakan untuk mendapatkan gambar yang digunakan sebagai pengukur kondisi permainan.

Method-method pada kelas ini adalah sebagai berikut:

* public void resetAll()

Method ini berfungsi untuk mengembalikan kondisi permainan menjadi kondisi awal permainan.

* public void stopWorking()

Method ini berfungsi untuk memberhentikan semua fungsi pada suatu objek QuestionMessageController.

* public void startQuestion()

Method ini berfungsi untuk memulai atau memunculkan suatu pertanyaan pada tampilan.

* public void onMotionDetected(Motion motion)

Method ini dipanggil jika terdeteksi suatu gerakan kepala. Parameter motion diisi dengan spesifikasi gerakan yang terdeteksi.

- Package HeadMotionDetector

- interface HeadMotionListener

Interface ini di-*implement* oleh kelas-kelas yang menggunakan hasil dari pendekripsi gerakan kepala. Method-method abstrak pada interface ini adalah sebagai berikut:

- * public void onMotionDetected(Motion motion)

Method ini dipanggil oleh jika terdeteksi suatu gerakan kepala. Parameter motion diisi dengan spesifikasi gerakan yang terdeteksi.

- Kelas MotionRecorder

Kelas ini digunakan untuk merekam, menentukan, dan mendekripsi gerakan kepala. Atribut-atribut pada kelas ini adalah sebagai berikut:

- * private PulseFactory pulseFactoryXAxis

Atribut ini digunakan untuk menyimpan PulseFactory yang merekam data gyroscope pada sumbu X.

- * private PulseFactory pulseFactoryYAxis

Atribut ini digunakan untuk menyimpan PulseFactory yang merekam data gyroscope pada sumbu Y.

- * private Detector nodDetector

Atribut ini digunakan untuk menyimpan Pendekripsi gerakan mengangguk.

- * private Detector shookDetector

Atribut ini digunakan untuk menyimpan Pendekripsi gerakan menggeleng.

- * private LinkedList<HeadMotionListener> listListener

Atribut ini digunakan untuk menyimpan kelas-kelas yang menjadi *listener* pendekripsi gerakan kepala.

Method-method pada kelas ini adalah sebagai berikut:

- * public MotionRecorder()

Constructor ini digunakan untuk menginisialisasi atribut-atribut yang digunakan.

- * public void addHeadMotionListener(HeadMotionListener listener)

Method ini digunakan untuk menambahkan *listener* baru pada atribut listListener

- * public void removeHeadMotionListener(HeadMotionListener listener)

Method ini digunakan untuk menghapus suatu *listener* pada atribut listListener.

- * public void inputGyroData(float x, float y)

Method ini digunakan untuk memasukkan data gyroscope yang digunakan untuk mendekripsi gerakan mengangguk dengan menggeleng.

- * public void notifyAllMotionListener(Motion motion)

Method ini memberitahu seluruh *listener* ketika terdeteksi suatu gerakan. Jenis gerakan yang diberitahukan adalah jenis gerakan pada parameter tersebut.

- Kelas PulseFactory

Kelas ini berguna untuk mendekripsi *Pulse* yang terjadi pada grafik data gyroscope. Beberapa atribut yang dimiliki oleh kelas ini menjadi kriteria dari suatu *Pulse* yang terdeteksi. Atribut-atribut tersebut adalah:

- * private float lastData
Atribut ini menyimpan data sebelum data yang baru dimasukkan
- * private float lastTime
Atribut ini menyimpan waktu pada pemasukan data sebelum daya yang baru dimasukkan.
- * private double simpangan
Atribut ini menyimpan simpangan terjauh pada *Pulse* terakhir yang terdeteksi.
- * private double rangeTime
Atribut ini menyimpan rentang waktu yang terjadi pada suatu *Pulse*
- * private double startTime
Atribut ini menyimpan waktu mulai dari suatu *Pulse* yang terjadi.
- * private double endTime
Atribut ini menyimpan waktu akhir dari suatu *Pulse* yang terjadi.
- * private float peak
Atribut ini menyimpan puncak nilai tertinggi atau terendah dari suatu *Pulse* bukit atau lembah.
- * private PulseType
Atribut ini menyimpan tipe *Pulse* antara bukit atau lembah.
- * private Axis axis
Atribut ini mendefinisikan sumbu yang direkam datanya.

Method-method pada kelas ini adalah:

- * public PulseFactory(Axis axis)
Constructor ini berfungsi untuk mendefinisikan sumbu pada gyroscope yang didekripsi.
- * private void resetAttributes()
Method ini berfungsi untuk mengembalikan nilai-nilai atribut kembali ke nilai asal.
- * private Pulse finishPulse(float data, double currentTime)
Method ini dipanggil ketika grafik melewati angka nol. Pada pemanggilan ini *Pulse* baru terdeteksi dan resetAttributes() dipanggil untuk pendekripsi *Pulse* selanjutnya.
- * public Pulse inputData(float data)
Method ini digunakan untuk memasukkan data gyroscope baru.

- Kelas Detector

Kelas ini merupakan kelas yang mendefinisikan pendekripsi gerakan kepala. Sub-class kelas ini adalah NodDetector dengan ShookDetector. Atribut-atribut pada kelas ini adalah:

- * public Detector(float limitSpeed, double limitSimpangan)
Constructor ini digunakan untuk meninisialisasi batasan-batasan yang didefinisikan pada atribut-atribut kelas ini.
- * protected boolean mightHaveActivities()
Atribut ini menunjukkan apakah suatu detector memiliki kemungkinan sedang mendekripsi gerakan atau tidak.

* protected Pulse[] listPulseToDetect()

Atribut ini menyimpan beberapa *Pulse* untuk diperiksa karakteristiknya, apakah sesuai dengan algoritma pendekripsi gerakan kepala.

* protected float limitSpeed()

Atribut ini mendefinisikan batas kecepatan angular yang sesuai dengan pendekripsi gerakan kepala.

* protected double limitSimpangan()

Atribut ini mendefinisikan batas Simpangan yang sesuai dengan pendekripsi gerakan kepala.

– Kelas NodDetector dengan ShookDetector

Kelas ini berisi algoritma untuk mendekripsi gerakan mengangguk atau menggeleng. Kelas ini merupakan turunan dari kelas Detector

– Kelas Pulse

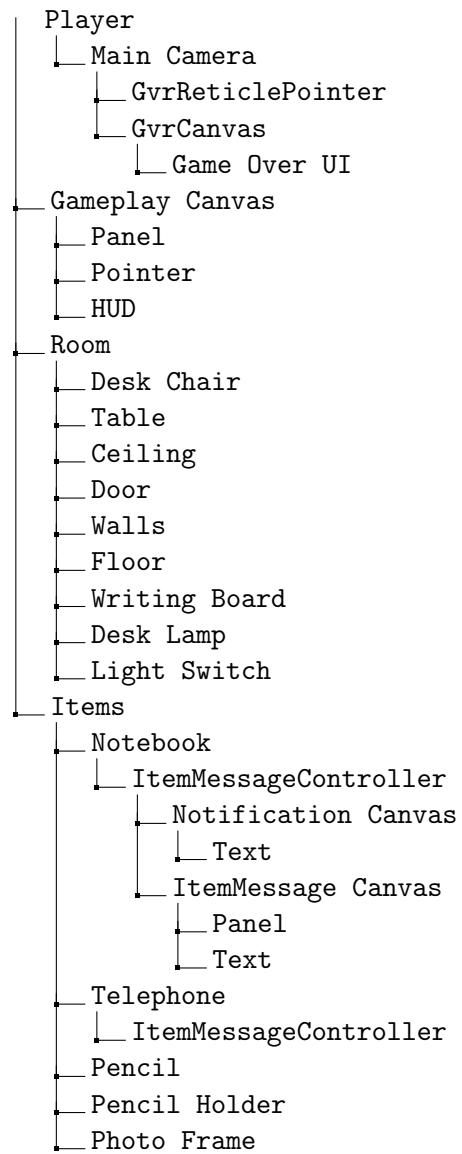
Kelas ini digunakan hanya untuk menyimpan karakteristik-karakteristik dari suatu *Pulse* yang terbentuk oleh kelas PulseDetector. Atribut-atribut pada kelas ini serupa dengan atribut-atribut pada kelas PulseDetector. Method-method pada kelas ini hanyalah Getter dengan Constructor saja.

4.2 Perancangan Hierarki GameObject pada Unity

Perancangan hierarki pada Unity merupakan salah satu perancangan yang cukup penting. Perancangan ini penting karena mempermudah pencarian, pemilihan, perubahan transformasi GameObject pada dunia permainan yang sedang dibuat. Selain itu perancangan ini dapat membantu pengguna Unity untuk membuat suatu GameObject yang memiliki sifat relatif terhadap suatu GameObject yang lain. Oleh Karena itu perancangan hierarki dibahas pada subbab ini.

Permainan ini hanya dibangun dengan menggunakan dua buah scene saja. Scene pertama digunakan hanya untuk mengecek apakah perangkat memiliki sensor gyroscope atau tidak. Scene kedua digunakan untuk implementasi keseluruhan permainan. Permainan ini tidak memerlukan Scene yang banyak karena permainan ini hanya memiliki satu ruangan saja. Sebagian besar pada perancangan ini hanyalah mengelompokkan GameObject-GameObject pada Scene. Perancangan hierarki scene pertama hanyalah satu buah GameObject GyroscopeChecker saja. Gambar 4.2 merupakan hasil dari perancangan hierarki Scene kedua permainan ini.

Pada Perancangan di atas ada beberapa GameObject utama, diantaranya adalah Player, GameplayCanvas, Room, dan Items. GameObject Player digunakan sebagai camera pada dunia permainan tersebut. Camera pada Player ini bergerak mengikuti orientasi pada *smartphone*. Pada GameObject ini terdapat GameObject Main Camera yang berfungsi untuk menjadi kamera pada permainan dan menentukan titik pandang pengguna. Main Camera memiliki dua buah Child Game Object. Kedua Game Object tersebut adalah GvrReticlePointer dan GameOverCanvas. GvrReticlePointer berguna untuk menampilkan titik pandang pengguna. GvrReticlePointer ini menjadi lingkaran ketika objek yang dipandangnya dapat memberikan respons dari masukan tombol pada Google Cardboard. GameOverCanvas berguna untuk menampilkan pesan "Game Over!" ketika permainan berakhir.



Gambar 4.2: Perancangan hierarki Game Object

GameObject GameplayCanvas digunakan untuk menunjukkan kondisi permainan sekarang. GameObject ini memiliki dua buah child yaitu Panel, Pointer, dan HUD. Panel berfungsi sebagai bidang putih pada UI. Pointer berfungsi sebagai menunjuk kondisi permainan pada saat ini. HUD adalah bentuk kurang lebih 1/6 lingkaran yang membatasi Pointer.

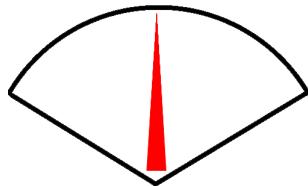
GameObject Room digunakan untuk menyimpan model-model dekorasi dalam ruangan. *Child* dari GameObject hanya merupakan model-model tiga dimensi yang membentuk ruangan pada dunia game.

GameObject Item digunakan untuk menyimpan model-model barang yang ada pada ruangan. Sebagian GameObject pada Item memiliki component yang diberikan script untuk mendeteksi gerakan kepala. GameObject yang diberikan script tersebut adalah GameObject yang memiliki *Child* GameObject ItemMessageController, yaitu Notebook dengan Telephone. Selain kedua GameObject tersebut GameObject lainnya merupakan model-model dekorasi pada meja.

4.3 Perancangan *Gameplay* dan Perancangan Antarmuka

Permainan ini merupakan permainan yang pemainnya berperan sebagai pemimpin dari suatu perusahaan. Tugas dari permainan ini adalah memilih pilihan-pilihan yang sesuai dari kondisi yang sedang terjadi. Pemain diberi pertanyaan dalam mengatur perusahaan tersebut. Pertanyaan tersebut hanya dapat dijawab ya atau tidak. Jawaban ya dilakukan dengan mengangguk dan jawaban tidak dengan menggeleng. Jawaban yang mengacu untuk menguntungkan perusahaan membuat tingkat kesenangan karyawan berkurang. Jawaban yang mengacu untuk menyenangkan karyawan merugikan perusahaan.

Takaran kesenangan karyawan dengan keuntungan perusahaan digambarkan dengan meteran seperti pada gambar 4.3. Jika jarum pada meteran tersebut mengarah ke kiri, berarti karyawan perusahaan sedang memiliki tingkat kesenangan yang tinggi, namun perusahaan tidak mendapatkan keuntungan yang baik. Begitu pula sebaliknya jika jarum mengarah ke kanan.



Gambar 4.3: Meteran pada permainan.

Tidak ada kondisi menang pada permainan ini, karena permainan ini terus berlangsung hingga pemain kalah. Pemain kalah jika meteran terlalu mengarah ke kiri, atau terlalu mengarah ke kanan. Ketika pemain sudah kalah, muncul pesan "Game Over!" (Gambar 4.5). Jika pemain menarik atau menekan tombol pada Google Cardboard-nya, permainan dimulai kembali dari awal.

Antarmuka yang ditampilkan pada permainan ini merupakan pemandangan seseorang pemimpin perusahaan (Pemain) pada meja kerjanya. Ruangan tersebut sangat sederhana, yaitu ruangan

dengan bentuk balok biasa dan diisi dengan beberapa model untuk dekorasi. Pada meja kerja terdapat beberapa alat seperti, pensi, buku catatan, telepon, lampu, dan lain sebagainya. Alat-alat yang ditambahkan *script C#* untuk dapat mendeteksi anggukan dan gelengan adalah telepon dan buku catatan. Ruangan tersebut digambarkan pada gambar 4.4.



Gambar 4.4: Desain ruangan untuk permainan yang dibuat.

Unity Asset Store menyediakan berbagai macam asset seperti *model*, *texture*, *prefab*, animasi, bahkan *scripts*. Perancangan aplikasi ini menggunakan beberapa asset yang diperoleh dari Unity Asset Store. Tabel 4.1 adalah asset-asset yang digunakan pada perancangan antar muka ini.

Untuk menampilkan pertanyaan, pemain harus menunggu suatu alat menampilkan notifikasi yang ditunjukkan dengan memunculkan karakter tanda seru (!) di atas alat tersebut. Ketika notifikasi tersebut muncul, pemain dapat menghadapkan pandangannya ke alat tersebut dan menarik/menelek tombol pada Google Cardboard untuk menampilkan pertanyaan yang diberikan. Pertanyaan yang diberikan diberikan dengan memunculkan suatu *pop-up* di atas alat tersebut yang ditulis pertanyaan yang ditanyakan (Notifikasi dan *pop-up* ditunjukkan pada gambar 4.5). Jumlah pertanyaan yang dapat dimunculkan hanyalah satu pertanyaan saja, tidak dapat memunculkan dua buah pertanyaan secara bersamaan. Setelah pertanyaan dijawab dengan anggukan atau gelengan *pop-up* tersebut menghilang, dan meteran bergerak tergantung dari jawaban pemain.

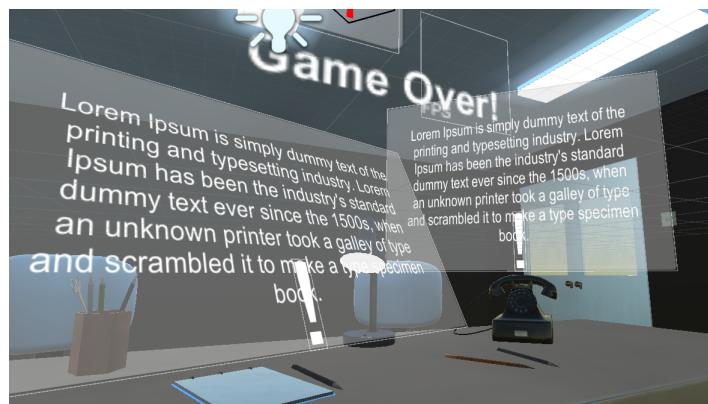
No	Nama Aset	Penerbit
1	Morgue Room PBR ¹	Rokay3D
2	Old Telephone ²	Rokay3D
3	Native Popups for iOS and Android ³	The Next Flow

Tabel 4.1: Tabel Aset-aset yang digunakan.

¹<https://www.assetstore.unity3d.com/en/#!/content/65817>

²<https://www.assetstore.unity3d.com/en/#!/content/62434>

³<https://www.assetstore.unity3d.com/en/#!/content/29566>



Gambar 4.5: Desain User Interface

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas implementasi, pengujian, dan masalah yang dihadapi. Pada bagian implementasi dijelaskan mengenai lingkungan implementasi dan hasil dari implementasi. Pada bagian pengujian berisi hasil dari pengujian. Pada bagian masalah yang dihadapi dijelaskan masalah-masalah menarik yang dihadapi pada saat implementasi.

5.1 Implementasi

Implementasi ini dilakukan dengan menggunakan *Game Engine* Unity. Implementasi dilakukan pada satu buah laptop, dan aplikasi dijalankan pada tiga buah *smartphone* Android.

5.1.1 Lingkungan Implementasi

Berikut adalah spesifikasi laptop yang digunakan untuk implementasi:

1. Processor : AMD A10-5750M Quad-core 2.5 - 3.5 Ghz
2. RAM : 8GB (7.21 Usable)
3. VGA : AMD Radeon HD 8650G + HD 8670M Dual Graphics
4. Sistem Operasi : Windows 10 64-bit
5. Versi Unity : 5.5.1f1 Personal Edition
6. Google VR SDK for Unity : 1.1

Berikut adalah spesifikasi perangkat *smartphone* android yang digunakan untuk implementasi:

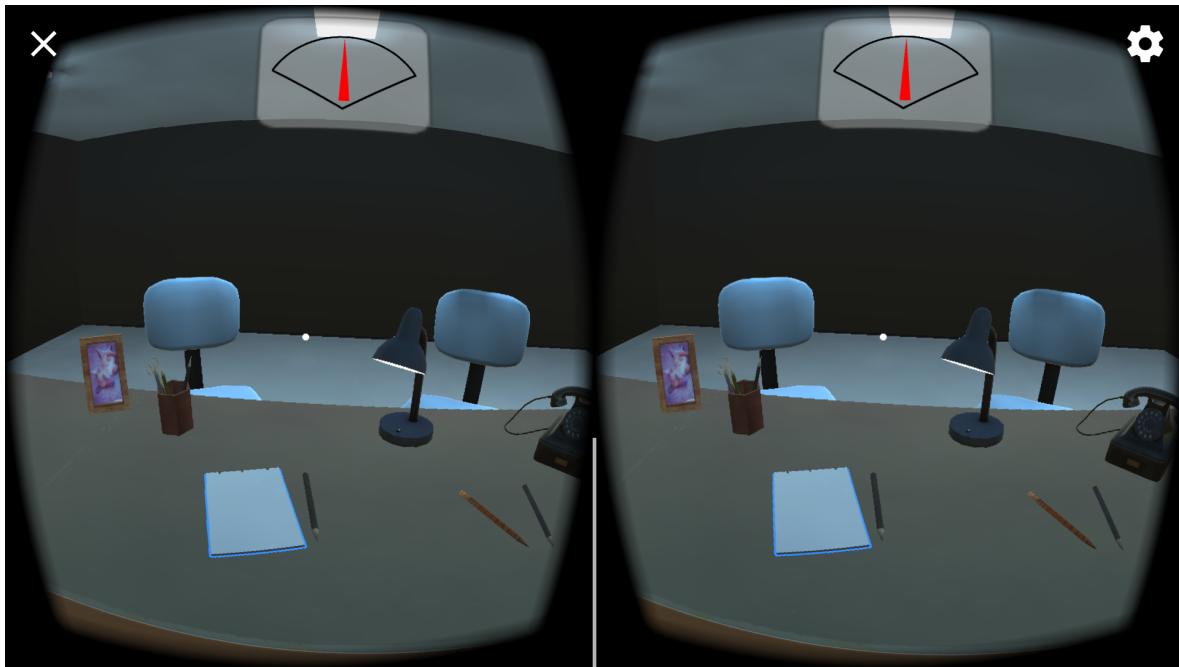
1. Processor : Qualcomm Snapdragon 625 Octa-core 2.0 GHz Cortex-A53
2. Sistem Operasi : Android OS 6.0 (Marshmallow)
3. Sensor-sensor : Accelerometer, gyro, proximity, compass, barometer

5.1.2 Hasil Implementasi

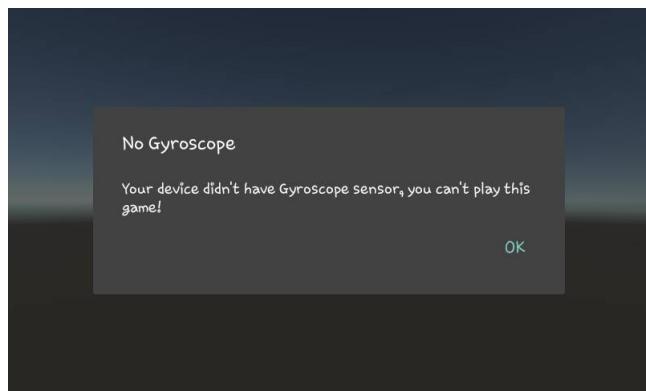
Hasil Implementasi ini adalah aplikasi permainan VR berbasis Android yang menggunakan Game Engine Unity. Aplikasi ini dapat di-*install* dengan menggunakan *Package Installer* yang berekstensi file ".apk" pada Android.

1. Aplikasi saat pertama kali terbuka.

Pada saat pertama kali dijalankan permainan langsung dimulai (Gambar 5.1). Jika perangkat tidak memiliki sensor Gyroscope, aplikasi menampilkan *pop-up* (Gambar 5.2) yang menunjukkan bahwa perangkat ini tidak dapat menjalankan aplikasi ini. Tombol 'X' pada pojok kiri atas selalu ada pada permainan ini dan berguna sebagai tombol untuk keluar dari permainan.



Gambar 5.1: Tampilan ketika aplikasi baru dimulai.



Gambar 5.2: Tampilan ketika perangkat tidak memiliki sensor Gryoscope.

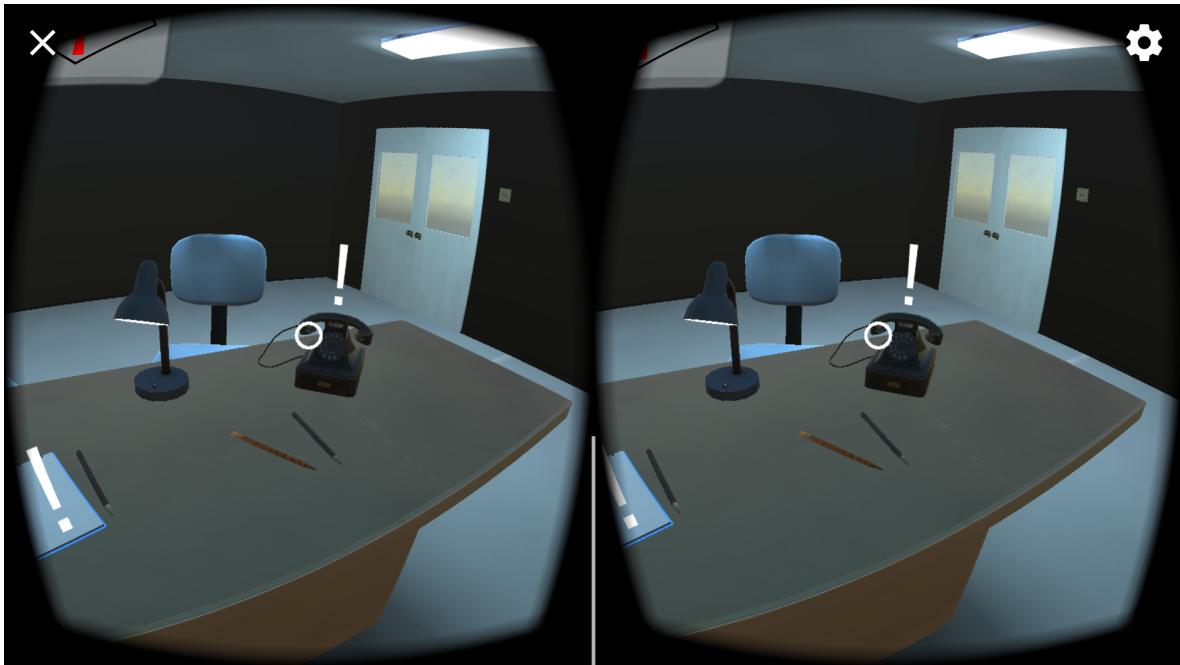
2. Aplikasi saat muncul notifikasi.

Ketika ada notifikasi, muncul tanda seru (!) di atas benda yang memiliki notifikasi seperti pada Gambar 5.3.

3. Aplikasi saat memunculkan pertanyaan.

Ketika pengguna sudah memilih suatu notifikasi yang muncul, muncul jendela pertanyaan di atas benda yang dipilih (Gambar 5.4).

4. Aplikasi setelah memberikan respons mengangguk atau menggeleng.

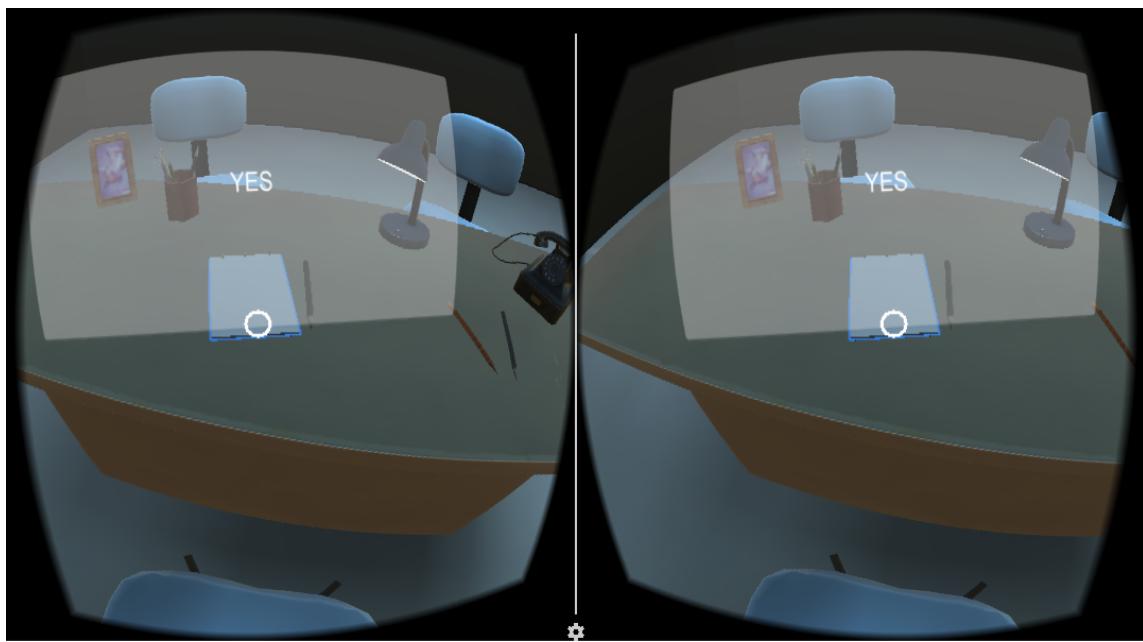


Gambar 5.3: Tampilan ketika notifikasi muncul.



Gambar 5.4: Tampilan ketika *pop-up* pertanyaan muncul.

Ketika pengguna mengangguk pada saat pertanyaan diberikan, tulisan pada panel pertanyaan berubah menjadi kata "YES" (Gambar 5.5) dan "NO" ketika pengguna menggeleng (Gambar 5.6).



Gambar 5.5: Tampilan setelah pengguna mengangguk.

5. **Aplikasi ketika pemain kalah dalam permainan tersebut.** Ketika pengguna kalah, maka muncul tulisan "Game Over!" di depan pandangan pengguna (Gambar 5.7). Tulisan ini mengikuti pergerakan kepala pengguna. Jika pengguna menarik/menekan tombol pada Google Cardboard, permainan diulang menjadi kondisi awal.

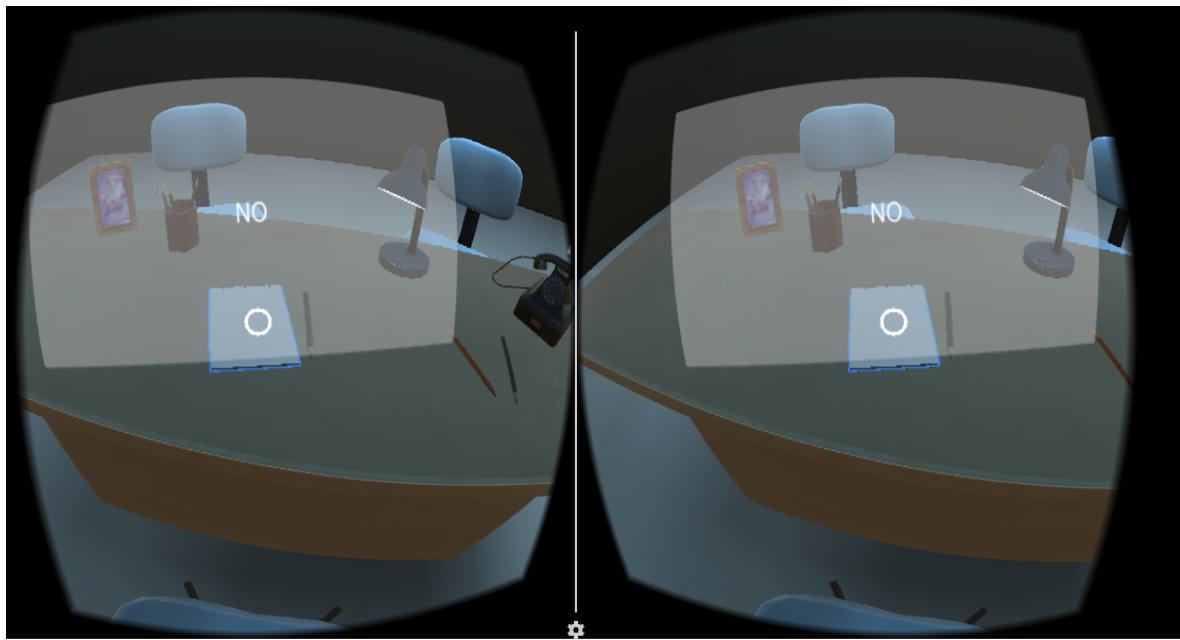
5.2 Pengujian

Pengujian dilakukan dengan menggunakan dua buah metode yaitu pengujian fungsional dan pengujian eksperimental. Pengujian fungsional bertujuan untuk menguji fungsi-fungsi yang disediakan pada aplikasi. Pengujian Eksperimental bertujuan untuk menguji pengalaman pengguna dalam menggunakan aplikasi ini.

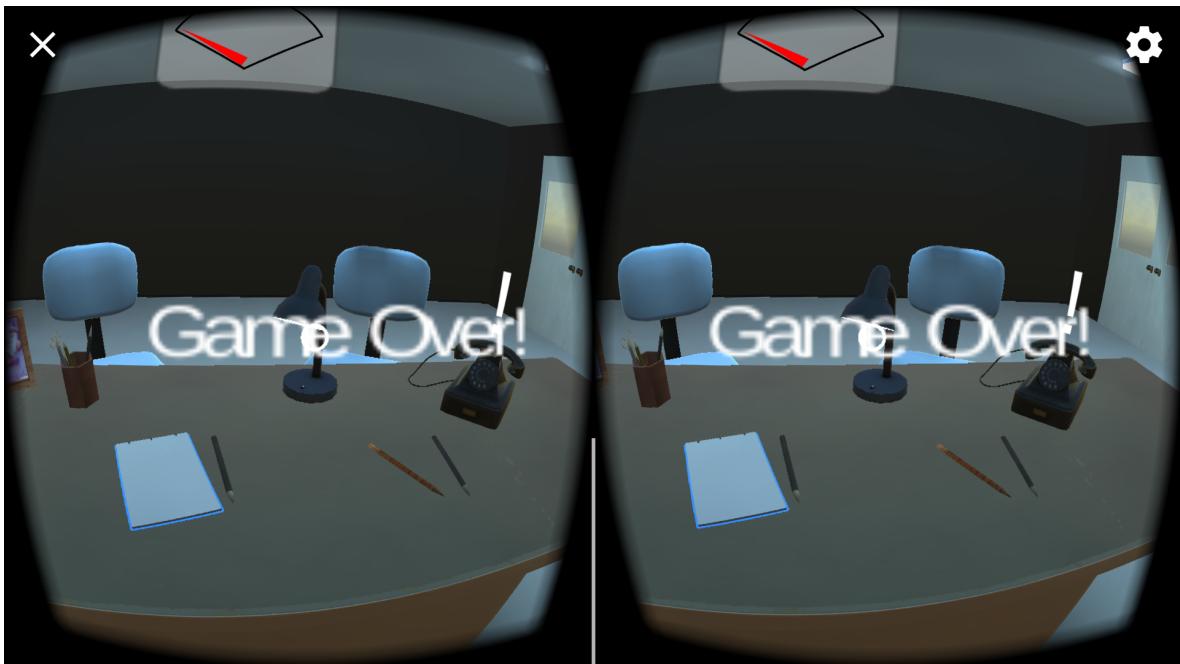
5.2.1 Pengujian Fungsional

Pengujian Fungsional ini dilakukan dengan mencoba menjalankan aplikasi pada perangkat-perangkat yang berbeda spesifikasi-nya. Tabel 5.1 merupakan daftar perangkat-perangkat *smartphone* Android yang digunakan saat Implementasi

Pada *smartphone* 1 terjadi kegagalan pada saat instalasi pada perangkat. Perangkat ini mengeluarkan *error* pada Google VR API for Unity, tetapi perangkat ini dapat berjalan dengan baik ketika menjalankan aplikasi untuk menganalisis algoritma pendekripsi gerakan kepala. Aplikasi-aplikasi Google VR lainnya yang tidak menggunakan Unity dalam membuatnya dapat berjalan dengan baik. Hal ini disebabkan karena pada Google VR SDK for Unity membutuhkan Sistem Operasi minimum pada versi 4.4 (KitKat), sehingga perangkat ini tidak dapat menjalankan aplikasi ini.



Gambar 5.6: Tampilan setelah pengguna menggeleng.



Gambar 5.7: Tampilan setelah permainan usai.

No	Processor	Sistem Operasi	Sensor-sensor
1	Exynos Dual-core 1.4 GHz Cortex-A9	Android OS v4.1.2 (Jelly Bean)	Accelerometer, gyro, proximity, compass, barometer
2	Qualcomm Snapdragon 800 Quad-core 2.3 GHz Krait 400	Android OS v6.0 (Marshmallow)	Accelerometer, gyro, proximity, compass, barometer
3	Qualcomm Snapdragon 625 Octa-core 2.0 GHz Cortex-A53	Android OS v6.0 (Marshmallow)	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass
4	Quad-core 1.2 GHz Cortex-A7	Android OS v4.4 (KitKat)	Accelerometer, proximity
5	1.3 GHz octa-core CPU, MediaTek Helio X10 MT6753 chipset	Android OS v6.0 (Marshmallow)	Accelerometer, Proximity, Compass, Light sensor, Fingerprint, Gyroscope(Software)
6	Qualcomm MSM8916 Snapdragon 410	Android 4.4.4 (KitKat)	Accelerometer, gyro, proximity, compass

Tabel 5.1: Tabel Perangkat yang digunakan

Aplikasi-aplikasi Google Cardboard yang dibuat dengan menggunakan Google Cardboard SDK dapat berjalan dengan baik karena Google Cardboard SDK membutuhkan Sistem Operasi minimum pada versi 4.1 (Jelly Bean).

Pada *smartphone* 2 aplikasi dapat berjalan dengan baik.

Pada *smartphone* 3 aplikasi dapat berjalan dengan baik.

Pada *smartphone* 4 aplikasi dapat di-*install* dengan baik. Ketika membuka aplikasi tersebut menampilkan pesan *error* bahwa perangkat yang digunakan tidak memiliki sensor Gyroscope. Hal ini terjadi karena aplikasi mengecek terlebih dahulu apakah perangkat memiliki sensor Gyroscope. Setelah menampilkan pesan *error* tersebut aplikasi memberhentikan dirinya sendiri.

Pada *smartphone* 5 aplikasi dapat berjalan dengan baik, tetapi pergerakan kamera pada aplikasi terkadang tidak sesuai dengan pergerakan. Hal ini dikarenakan pada perangkat ini terdeteksi memiliki sensor Gryoscope, namun sensor gyroscope yang terbentuk pada perangkat ini merupakan sensor yang menggunakan *software* untuk memenuhi fungsi gyrosopenya. Sensor Gyroscope yang terbentuk oleh *software* lebih tidak akurat dibandingkan dengan sensor Gyroscope yang terbentuk dari hardware. Oleh sebab itu pada perangkat ini aplikasi ini tidak berjalan begitu baik.

Pada *smartphone* 6 aplikasi dapat berjalan dengan baik.

5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan dengan menguji aplikasi ini kepada lima pengguna berbeda. Pada saat pencobaan kelima pengguna tersebut memainkan permainan ini. Kemudian mencatat pendapat, kritik, atau saran dari pengguna tentang permainan ini.

Berikut adalah pengguna-pengguna pada pengujian aplikasi ini:

1. Ricky Setiawan (Gambar 5.8) menggunakan perangkat *smartphone* Android nomor 4

"Aplikasi-nya bagus, untuk masukan anggukan dan gelengan harus memperkirakan terlebih dahulu seberapa simpangan yang dibutuhkan agar dapat terdeteksi."



Gambar 5.8: Pengujian oleh Ricky Setiawan.

2. Harseto Pandityo (Gambar 5.9) menggunakan perangkat *smartphone* nomor 3
"Permainannya bagus dan menarik, khususnya pada fungsi mengangguk dengan menggeleng. Terkadang ada masalah ketika mengangguk tidak terdeteksi, tetapi harus mengangguk dengan lebih jauh lagi simpangan-nya agar dapat terdeteksi. Tidak ada masalah pada saat menggeleng."



Gambar 5.9: Pengujian oleh Harseto Pandityo.

3. Herfan Heryandi (Gambar 5.10) menggunakan perangkat *smartphone* nomor 5
"Untuk geleng sedikit kurang sensitif, tetapi untuk mengangguk sudah sensitif."
4. Kevin Antonius (Gambar 5.11) menggunakan perangkat *smartphone* 3



Gambar 5.10: Pengujian oleh Herfan Heryandi.

"Udah bagus, terkadang ketika mengangguk tidak terdeteksi. Mungkin karena anggukan saya terlalu kecil simpangannya."



Gambar 5.11: Pengujian oleh Kevin Antonius.

5. Antonius Kurnia (Gambar 5.12) menggunakan perangkat *smartphone* 3

"Sensor untuk deteksinya sudah bagus, sudah dapat mendeteksi anggukan dan geleng dengan baik. Hanya saja terkadang susah membaca tulisannya, karena gerakan anggukan dan gelengannya"

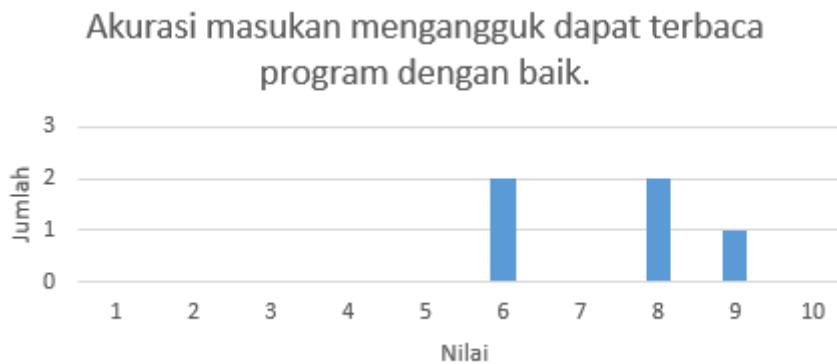
Setelah melakukan pengujian, kelima pengguna diberikan *survey* dengan memilih nilai dari 1 hingga 10 dengan pernyataan-pernyataan berikut:



Gambar 5.12: Pengujian oleh Antonius Kurnia.

1. Akurasi masukan mengangguk dapat terbaca program dengan baik.
2. Akurasi masukan menggeleng dapat terbaca program dengan baik.
3. *Gameplay* pada permainan menarik.
4. Tampilan grafik model tiga dimensi pada permainan memuaskan.
5. Nilai keseluruhan dari permainan.

Nilai 1 diartikan sebagai sangat tidak baik. Semakin besar nilai yang diberikan menunjukkan hasil yang baik. Berarti nilai 10 diartikan sebagai sangat baik.



Gambar 5.13: Grafik hasil *survey* pengujian pernyataan pertama.

Gambar 5.13 merupakan hasil rangkuman dari perolehan data *survey* pada pernyataan pertama. Dari grafik pada Gambar 5.13 dapat disimpulkan bahwa permainan ini dapat mendeteksi gerakan anggukan dengan baik. Dua pengguna menilai dengan nilai 7. Dua orang menilai dengan nilai 6. Satu orang menilai dengan nilai 9. Rata-rata dari nilai-nilai yang diperoleh adalah sejumlah 7,4. Oleh karena itu untuk mendeteksikan gerakan mengangguk rata-rata pengguna berpendapat baik.

Gambar 5.14 merupakan hasil rangkuman dari perolehan data *survey* pada pernyataan kedua. Dari grafik pada Gambar 5.14 dapat disimpulkan bahwa permainan ini dapat mendeteksi gerakan

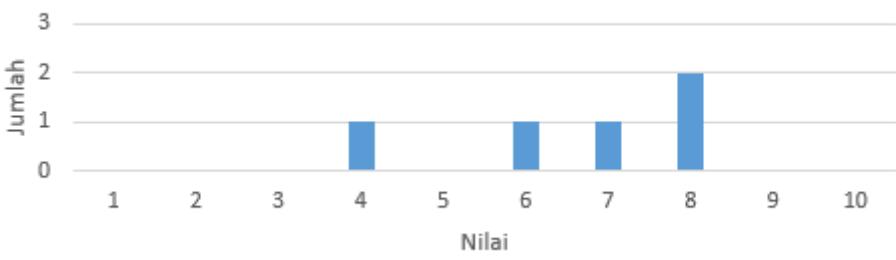
**Akurasi masukan menggeleng dapat terbaca
program dengan baik.**



Gambar 5.14: Grafik hasil *survey* pengujian pernyataan kedua.

anggukan dengan sangat baik. Dua pengguna menilai dengan nilai 7. Nilai 8,9, dan 10 masing-masing berjumlah 1 orang. Rata-rata dari nilai-nilai yang diperoleh adalah sejumlah 8,2. Oleh karena itu untuk pendekripsi gerakan menggeleng rata-rata pengguna berpendapat baik.

Gameplay pada permainan menarik.



Gambar 5.15: Grafik hasil *survey* pengujian pernyataan ketiga.

Gambar 5.15 merupakan hasil rangkuman dari perolehan data *survey* pada pernyataan ketiga. Dari grafik pada Gambar 5.15 dapat disimpulkan bahwa *gameplay* dari permainan ini cukup. Nilai 4,6, dan 7 masing-masing berjumlah 1 orang. Nilai 8 berjumlah 2 orang. Rata-rata dari nilai-nilai yang diperoleh adalah sejumlah 6,6. Oleh karena itu *gameplay* dari permainan ini pengguna berpendapat cukup.

**Tampilan grafik model tiga dimensi pada
permainan memuaskan.**



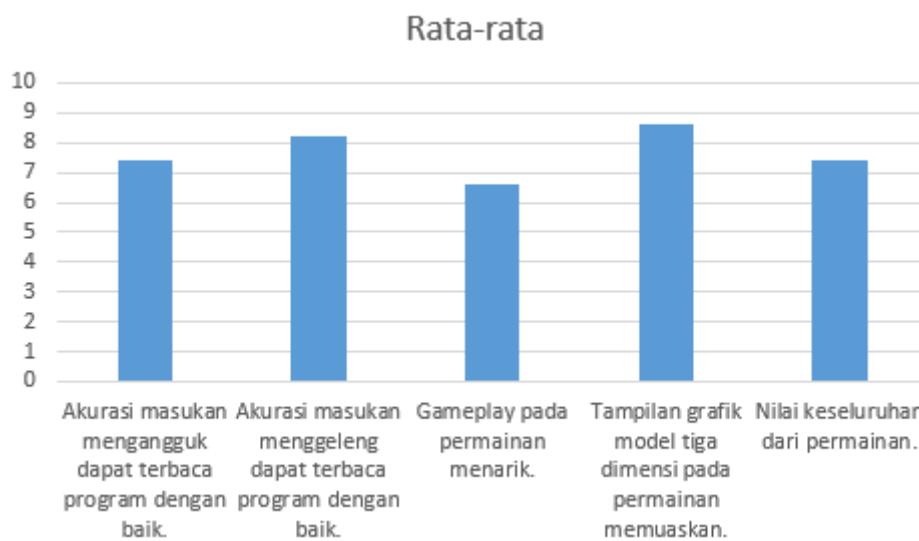
Gambar 5.16: Grafik hasil *survey* pengujian pernyataan keempat.

Gambar 5.16 merupakan hasil rangkuman dari perolehan data *survey* pada pernyataan keempat. Dari grafik pada Gambar 5.16 dapat disimpulkan bahwa permainan ini memiliki tampilan grafik model tiga dimensi yang baik. Seorang pengguna menilai dengan nilai 6. Empat pengguna menilai dengan nilai 8. Rata-rata dari nilai-nilai yang diperoleh adalah sejumlah 8,6. Oleh karena itu untuk tampilan grafik tiga dimensi pengguna berpendapat baik.



Gambar 5.17: Grafik hasil *survey* pengujian pernyataan kelima.

Gambar 5.17 merupakan hasil rangkuman dari perolehan data *survey* pada pernyataan kelima. Dari grafik pada Gambar 5.17 dapat disimpulkan bahwa nilai keseluruhan permainan ini baik. Tiga pengguna memberikan nilai 7 untuk keseluruhan permainan. Dua pengguna memberikan nilai 8. Nilai dari keseluruhan permainan memiliki rata-rata 7,4. Jadi nilai keseluruhan permainan dinilai baik oleh pengguna.



Gambar 5.18: Grafik rata-rata nilai dari hasil *survey* yang dilakukan.

Gambar 5.18 merupakan grafik seluruh rata-rata dari pernyataan-pernyataan yang ada. Pada grafik tersebut terlihat bahwa nilai rata-rata tertinggi adalah tampilan model grafik tiga dimensi. Nilai rata-rata terendah adalah *gameplay* pada permainan. Akurasi pendekripsi mengangguk memiliki rata-rata yang lebih rendah dibandingkan dengan akurasi pendekripsi menggeleng. Berdasarkan dari grafik tersebut kekurangan dari permainan ini terdapat pada *gameplay*-nya, karena

mendapatkan nilai rata-rata paling kecil. Bahkan ada pengguna yang memberikan nilai 4 pada *gameplay*. Dari keseluruhan rata-rata permainan ini dapat disimpulkan mendapatkan respons yang baik dari pengguna.

5.3 Masalah yang Dihadapi pada Saat Implementasi

Berikut adalah beberapa masalah yang dihadapi pada saat implementasi:

1. Sulit dalam mencari permainan *open source* yang telah ada. Rencana awal dalam mengimplementasi algoritma ini adalah dengan mencari permainan *open source* yang telah ada, namun permainan *open source* yang telah mengimplementasi Google VR sangatlah langka. Ketika telah menemukan permainan *open source* yaitu "Doom VR", SDK yang digunakan oleh permainan merupakan SDK dengan versi lama yang sudah tidak dikembangkan lagi oleh Google. Sehingga peneliti tidak dapat menemukan SDK Google VR versi lama. Jadi masalah ini diselesaikan dengan membuat suatu game baru dengan bantuan Game Engine Unity.
2. Penggunaan Quaternion untuk menyimpan orientasi GameObject-GameObject pada Unity membuat bingung pada saat implementasi. Hal ini terjadi karena penggunaan Quaternion yang dikonversi menjadi sudut Euler membuat objek tersebut tidak dapat menyimpan sudut putar yang lebih besar dari 360 derajat dan lebih kecil dari 0 derajat.
3. Google VR for Unity belum sepenuhnya stabil. Pada saat instalasi Google VR for Unity terjadi *compile error*. Kesalahan *compile error* ini tidak diberikan penyelesaiannya pada website resmi Google VR maupun Unity. Sehingga penulis mencari solusi-nya pada forum pengguna Unity. Satu-satunya solusi untuk menyelesaikan solusi ini adalah dengan mengganti satu baris kode pada Google VR for Unity. *Compile error* tersebut terjadi pada script C# "GvrVideoPlayerTexture.cs" pada baris 595. Solusinya yaitu dengan menambahkan kode "yield break;" sebelum perintah "return;". Sekarang masalah ini sudah diperbaiki oleh Google VR sehingga tidak terjadi lagi.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil penelitian yang dilakukan, diperoleh kesimpulan-kesimpulan sebagai berikut:

1. Dalam pembuatan Grafik dapat diselesaikan menggunakan Android API pada Android Studio. API ini digunakan untuk merekam seluruh data sensor-sensor pada perangkat Android ketika pengguna mengangguk dan menggeleng. Data kemudian disimpan pada *file ber-format ".csv"* dan dibuatkan grafiknya menggunakan Aplikasi Microsoft Excel
2. Untuk dapat mendeteksi gerakan kepala khususnya mengangguk dan menggeleng, hanya diperlukan sensor gyroscope saja. Sensor-sensor lainnya tidak diperlukan untuk membantu mendeteksi gerakan kepala. Selain dapat menggunakan gyroscope, berdasarkan pengujian fungsional sensor gabungan accelerometer dan magnetometer dapat juga digunakan sebagai pengganti sensor gyroscope (sensor gyroscope software) pada perangkat-perangkat tertentu.
3. Aplikasi telah dapat membaca gerakan kepala dengan baik. Hal ini ditunjukkan dengan tidak adanya kesalahan masukan pada pengujian eksperimental.
4. Berdasarkan hasil pengujian eksperimental, algoritma pada aplikasi ini dapat berjalan dengan baik. Kelemahan pada algoritma ini berada pada simpangan anggukan dengan gelangan yang dilakukan, karena setiap orang mengangguk dan menggeleng dengan besar simpangan yang berbeda-beda.

6.2 Saran

Berdasarkan hasil penelitian yang dilakukan, berikut adalah beberapa saran untuk pengembangan:

1. Mengimplementasikan fungsi kalibrasi anggukan dan gelenggan, sehingga sesuai dengan kriteria pengguna.
2. Mengimplementasikan fungsi pendekripsi gerakan kepala menggunakan sensor-sensor selain gyroscope, jika perangkat tidak memiliki sensor gyroscope. Sensor-sensor tersebut seperti accelerometer, compass, dan lain-lain. Jika hasilnya kurang memuaskan dengan menggunakan satu buah sensor saja, implementasi dapat menggunakan metode Sensor Fusion. Sensor Fusion adalah metode untuk menggabungkan nilai-nilai yang didapatkan pada setiap sensor untuk mencapai tujuan tertentu.

3. Membuat *library* untuk algoritma pendekripsi gerakan kepala, sehingga dapat dimanfaatkan dan digunakan oleh pengembang lain
4. Membuat fitur pada permainan untuk mencatat simpangan yang terjadi sehingga dapat dia-nalisis kriteria gerakan anggukan dan gelangan kepala pengguna.

DAFTAR REFERENSI

- [1] T. Parisi, *Learning virtual reality: developing immersive experiences and applications for desktop, web, and mobile*. O'Reilly Media, 1 ed., 2015.
- [2] G. J. Kim, *Designing virtual reality systems: the structured approach*. Springer, 2005.
- [3] J. Vince, *Introduction to virtual reality*. Springer, 2004.
- [4] “Google cardboard.” <https://vr.google.com/cardboard/>. [Online; diakses 10-September-2016].
- [5] “Sensor types.” <https://source.android.com/devices/sensors/sensor-types.html>. [Online; diakses 10-September-2016].
- [6] G. Bleser and D. Stricker, “Advanced tracking through efficient image processing and visual–inertial sensor fusion,” *Computers & Graphics*, vol. 33, no. 1, pp. 59–72, 2009.
- [7] “Android 7.0 Nougat!” <https://developer.android.com/>. [Online; diakses 12-September-2016].
- [8] “Google VR | Google Developers.” <https://developers.google.com/vr/>. [Online; diakses 20-September-2016].
- [9] J. B. Kuipers, *Quaternions and Rotation Sequences : A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 1998.
- [10] “Unity - Game Engine.” <https://unity3d.com/>. [Online; diakses 16-Februari-2016].
- [11] “Aldin Dynamics.” <http://www.aldindynamics.com/>. [Online; diakses 22-November-2016].

LAMPIRAN A

KODE PROGRAM APLIKASI UNTUK ANALISIS

Listing A.1: MainActivity.java

```
1 package com.example.egaprianto.testingsensors;
2
3 import android.content.Intent;
4 import android.hardware.Sensor;
5 import android.hardware.SensorManager;
6 import android.support.v7.app.AppCompatActivity;
7 import android.os.Bundle;
8 import android.view.View;
9 import android.widget.TextView;
10
11 public class MainActivity extends AppCompatActivity {
12     TextView mTextView;
13     private SensorManager mSensorManager;
14     private Sensor mSensor;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_main);
20     }
21
22     @Override
23     public void onDestroy() {
24         super.onDestroy();
25         android.os.Debug.stopMethodTracing();
26     }
27
28     public void onClickLightSensorButton(View view){
29         Intent intent = new Intent(this, SensorAccelerometerActivity.class);
30         startActivity(intent);
31     }
32
33     public void onRotVecSensorButton(View view){
34         Intent intent = new Intent(this, QuaternionTheta.class);
35         startActivity(intent);
36     }
37
38     public void recordAllRawsensorData(View view){
39         Intent intent = new Intent(this, RecordAllRawsensorData.class);
40         startActivity(intent);
41     }
42
43     public void onGyroSensorClicked(View view){
44         Intent intent = new Intent(this, SensorGyroscopeActivity.class);
45         startActivity(intent);
46     }
47     public void onMagSensorClicked(View view){
48         Intent intent = new Intent(this, SensorGeomagneticRotationActivity.class);
49         startActivity(intent);
50     }
51 }
```

Listing A.2: RecordAllRawsensorData.java

```
1 package com.example.egaprianto.testingsensors;
2
3 import android.Manifest;
4 import android.content.Context;
5 import android.content.pm.PackageManager;
6 import android.hardware.Sensor;
7 import android.hardware.SensorEvent;
8 import android.hardware.SensorEventListener;
9 import android.hardware.SensorManager;
10 import android.media.Ringtone;
11 import android.media.RingtoneManager;
12 import android.net.Uri;
13 import android.os.Build;
14 import android.os.Bundle;
15 import android.os.CountDownTimer;
16 import android.os.Environment;
```

```

17 import android.support.v4.app.ActivityCompat;
18 import android.support.v7.app.AppCompatActivity;
19 import android.text.format.DateFormat;
20 import android.util.Log;
21 import android.view.View;
22 import android.view.WindowManager;
23 import android.widget.TextView;
24
25
26 import java.io.BufferedWriter;
27 import java.io.File;
28 import java.io.FileWriter;
29 import java.io.IOException;
30 import java.util.ArrayList;
31 import java.util.List;
32
33 public class RecordAllRawSensorData extends AppCompatActivity implements SensorEventListener {
34     private static final String FILENAME = "sensorRecord-";
35     private SensorManager mSensorManager;
36     private ArrayList<Sensor> sensorArrayList;
37     private TextView textViewCountDown;
38     private boolean isCapturing;
39     private File file;
40     private ArrayList<double[]> acceleroData;
41     private ArrayList<double[]> gyroData;
42     private long startCaptureTime;
43     private long runningTime;
44     private ArrayList<double[]> rotData;
45     private ArrayList<double[]> rotVecData;
46
47     @Override
48     protected void onCreate(Bundle savedInstanceState) {
49         super.onCreate(savedInstanceState);
50         this.sensorArrayList = new ArrayList<Sensor>();
51         setContentView(R.layout.activity_record_all_rawsensor_data);
52         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
53         textViewCountDown = (TextView) findViewById(R.id.textViewCountDown);
54         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
55             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_ACCELEROMETER));
56             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_GYROSCOPE));
57             this.sensorArrayList.add(sensorGetter(Sensor.TYPE_ROTATION_VECTOR));
58         }
59
60         int REQUEST_EXTERNAL_STORAGE = 1;
61         String[] PERMISSIONS_STORAGE = {
62             Manifest.permission.READ_EXTERNAL_STORAGE,
63             Manifest.permission.WRITE_EXTERNAL_STORAGE
64         };
65         int permission = ActivityCompat.checkSelfPermission(this, Manifest.permission.
66             WRITE_EXTERNAL_STORAGE);
67         if (permission != PackageManager.PERMISSION_GRANTED) {
68             ActivityCompat.requestPermissions(
69                 this,
70                 PERMISSIONS_STORAGE,
71                 REQUEST_EXTERNAL_STORAGE
72             );
73         }
74         isCapturing = false;
75         for (Sensor sensor :
76             sensorArrayList) {
77             mSensorManager.registerListener(this, sensor, SensorManagerSENSOR_DELAY_GAME);
78         }
79     }
80
81     @Override
82     protected void onResume() {
83         super.onResume();
84         for (Sensor sensor :
85             sensorArrayList) {
86             mSensorManager.registerListener(this, sensor, SensorManagerSENSOR_DELAY_GAME);
87         }
88     }
89
90     @Override
91     protected void onPause() {
92         super.onPause();
93         mSensorManager.unregisterListener(this);
94     }
95
96     @Override
97     public void onSensorChanged(SensorEvent event) {
98         Sensor sensor = event.sensor;
99         if (isCapturing) {
100             float[] copyData = new float[4];
101             System.arraycopy(event.values, 0, copyData, 1, 3);
102             copyData[0] = System.currentTimeMillis() - startCaptureTime;
103             if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
104                 this.acceleroData.add(convertFloatsToDoubles(copyData));
105             } else if (sensor.getType() == Sensor.TYPE_GYROSCOPE) {
106                 this.gyroData.add(convertFloatsToDoubles(copyData));
107             } else if (sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {
108                 copyData = new float[6];
109                 System.arraycopy(event.values, 0, copyData, 1, event.values.length);
110                 copyData[0] = System.currentTimeMillis() - startCaptureTime;
111                 this.rotData.add(convertFloatsToDoubles(copyData));
112                 double theta = (Math.acos(event.values[3]))*2;
113                 double x = event.values[0]/ Math.sin(theta/2);
114                 double y = event.values[1]/ Math.sin(theta/2);

```

```

115    double [] vecData = new double[5];
116    vecData[0] = copyData[0];
117    vecData[1] = x;
118    vecData[2] = y;
119    vecData[3] = z;
120    vecData[4] = theta;
121    this.rotVecData.add(vecData);
122 }
123 runningTime = (System.currentTimeMillis() - startCaptureTime);
124 this.textViewCountDown.setText("Time:" + runningTime / 1000 + "sec");
125 }
126
127
128 @Override
129 public void onAccuracyChanged(Sensor sensor, int accuracy) {
130 }
131
132
133 public void onStartButtonClicked(View view) throws InterruptedException {
134     Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
135     Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), notification);
136     acceleroData = new ArrayList<double[]>();
137     gyroData = new ArrayList<double[]>();
138     rotData = new ArrayList<double[]>();
139     rotVecData = new ArrayList<double[]>();
140     String time = DateFormat.format("MM-dd-yyyy-h|mm|ss-aa", System.currentTimeMillis()).toString();
141     File root = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS)
142         ) + File.separator + "DataSensors");
143     root.mkdirs();
144     file = new File(root, FILENAME + time + ".csv");
145     try {
146         file.createNewFile();
147     } catch (IOException e) {
148         e.printStackTrace();
149     }
150     new CountDownTimer(10000, 1000) {
151
152         public void onTick(long millisUntilFinished) {
153             textViewCountDown.setText("Count_Down=" + millisUntilFinished / 1000);
154         }
155
156         public void onFinish() {
157             textViewCountDown.setText("Capturing_Data!");
158             Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
159             Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), notification);
160             r.play();
161             isCapturing = true;
162             getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
163             startCaptureTime = System.currentTimeMillis();
164         }
165     }.start();
166 }
167
168 private Sensor sensorGetter(int input) {
169
170     List<Sensor> sensorList = mSensorManager.getSensorList(input);
171     Sensor sensor = null;
172     for (int i = 0; i < sensorList.size(); i++) {
173         sensor = sensorList.get(i);
174     }
175     return sensor;
176 }
177
178 private void writeData(BufferedWriter bw, ArrayList<double[]> data, String title, String columnTitle)
179     throws IOException {
180     bw.write(title);
181     bw.newLine();
182     bw.write(columnTitle);
183     bw.newLine();
184     for (int i = 0; i < data.size(); i++) {
185         for (int j = 0; j < data.get(i).length; j++) {
186             bw.write(data.get(i)[j] + ",");
187         }
188         bw.newLine();
189     }
190     bw.newLine();
191 }
192
193 private double[] convertFloatsToDoubles(float[] input) {
194     if (input == null)
195     {
196         return null;
197     }
198     double[] output = new double[input.length];
199     for (int i = 0; i < input.length; i++)
200     {
201         output[i] = input[i];
202     }
203     return output;
204 }
205
206 public void onStopButtonClicked(View view) {
207     isCapturing = false;
208     getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
209     try {
210         BufferedWriter bw = new BufferedWriter(new FileWriter(this.file));
211         writeData(bw, acceleroData, "Accelerometer", "time,x,y,z");

```

```

212     writeData(bw, gyroData, "Gyroscope", "time,x,y,z");
213     writeData(bw, rotData, "Rotation_Vector", "time,x*sin(theta/2),y*sin(theta/2),z*sin(theta/2),
214     cos(theta/2),akurasi_(dalam_radians)_(-1_jika_tidak_ada)");
215     writeData(bw, rotVecData, "Rotation_Vector_by_Vector_and_Angle", "time,x,y,z,theta");
216     bw.newLine();
217     bw.write("Running_Time:" + runningTime + "ms");
218     bw.flush();
219     bw.close();
220     file.createNewFile();
221     textViewCountDown.setText("Data_Captured!_at_" + file.getAbsolutePath());
222     Log.d("File", "File_is_located_at_" + file.getAbsolutePath());
223 } catch (IOException e) {
224     e.printStackTrace();
225 }
226 }
227 }
```

Listing A.3: SensorAccelerometerActivity.java

```

1 package com.example.egaprianto.testingsensors;
2
3 import android.app.Activity;
4 import android.content.Context;
5 import android.hardware.Sensor;
6 import android.hardware.SensorEvent;
7 import android.hardware.SensorEventListener;
8 import android.hardware.SensorManager;
9 import android.os.Build;
10 import android.os.Bundle;
11 import android.widget.TextView;
12
13 import java.util.List;
14
15 public class SensorAccelerometerActivity extends Activity implements SensorEventListener {
16     private SensorManager mSensorManager;
17     private Sensor mAccelSensor;
18     private TextView textViewXData;
19     private TextView textViewYData;
20     private TextView textViewZData;
21     private TextView textViewDebug;
22     private TextView textViewAccuracy;
23
24     @Override
25     public final void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_accelero_sensor);
28         textViewXData = (TextView) findViewById(R.id.textViewXData);
29         textViewYData = (TextView) findViewById(R.id.textViewYData);
30         textViewZData = (TextView) findViewById(R.id.textViewZData);
31         textViewDebug = (TextView) findViewById(R.id.textViewDebug);
32         textViewAccuracy = (TextView) findViewById(R.id.textViewAccuracy);
33         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
34         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM SANDWICH_MR1) {
35             mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
36             List<Sensor> accelerometerSensors = mSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
37             mAccelSensor = null;
38             if (accelerometerSensors != null) {
39                 for (int i = 0; i < accelerometerSensors.size(); i++) {
40                     mAccelSensor = accelerometerSensors.get(i);
41                 }
42             }
43         }
44     }
45
46     @Override
47     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
48     }
49
50     @Override
51     public final void onSensorChanged(SensorEvent event) {
52         float x = event.values[0];
53         float y = event.values[1];
54         float z = event.values[2];
55         textViewXData.setText("x=_" + x);
56         textViewYData.setText("y=_" + y);
57         textViewZData.setText("z=_" + z);
58     }
59
60     @Override
61     protected void onResume() {
62         super.onResume();
63         mSensorManager.registerListener(this, mAccelSensor, SensorManager.SENSOR_DELAY_NORMAL);
64     }
65
66     @Override
67     protected void onPause() {
68         super.onPause();
69         mSensorManager.unregisterListener(this);
70     }
71
72 }
73 }
```

Listing A.4: SensorGyroscopeActivity.java

```
1 package com.example.egaprianto.testingsensors;
```

```

2|
3 import android.app.Activity;
4 import android.content.Context;
5 import android.hardware.Sensor;
6 import android.hardware.SensorEvent;
7 import android.hardware.SensorEventListener;
8 import android.hardware.SensorManager;
9 import android.os.Build;
10 import android.os.Bundle;
11 import android.widget.TextView;
12
13 import java.util.List;
14
15 public class SensorGyroscopeActivity extends Activity implements SensorEventListener {
16     private SensorManager mSensorManager;
17     private Sensor mGryoSensor;
18     private TextView textViewXData;
19     private TextView textViewYData;
20     private TextView textViewZData;
21     private TextView textViewDebug;
22     private TextView textViewAccuracy;
23
24     @Override
25     public final void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_gyroscope_sensor);
28         textViewXData = (TextView) findViewById(R.id.textViewXData);
29         textViewYData = (TextView) findViewById(R.id.textViewYData);
30         textViewZData = (TextView) findViewById(R.id.textViewZData);
31         textViewDebug = (TextView) findViewById(R.id.textViewDebug);
32         textViewAccuracy = (TextView) findViewById(R.id.textViewAccuracy);
33         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
34         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
35             mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
36             List<Sensor> gyroscopeSensors = mSensorManager.getSensorList(Sensor.TYPE_GYROSCOPE);
37             mGryoSensor = null;
38             if (gyroscopeSensors != null) {
39                 for (int i = 0; i < gyroscopeSensors.size(); i++) {
40                     mGryoSensor = gyroscopeSensors.get(i);
41                 }
42             }
43         }
44     }
45
46     @Override
47     public final void onAccuracyChanged(Sensor sensor, int accuracy) {
48     }
49
50     @Override
51     public final void onSensorChanged(SensorEvent event) {
52         float x = event.values[0];
53         float y = event.values[1];
54         float z = event.values[2];
55         textViewXData.setText("x=" + x);
56         textViewYData.setText("y=" + y);
57         textViewZData.setText("z=" + z);
58     }
59
60     @Override
61     protected void onResume() {
62         super.onResume();
63         mSensorManager.registerListener(this, mGryoSensor, SensorManager.SENSOR_DELAY_NORMAL);
64     }
65
66     @Override
67     protected void onPause() {
68         super.onPause();
69         mSensorManager.unregisterListener(this);
70     }
71
72
73 }

```


LAMPIRAN B

KODE PROGRAM APLIKASI *GAME*

Listing B.1: GyroscopeChecker.cs

```

1  iżf
2  using TheNextFlow.UnityPlugins;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class GyroscopeChecker : MonoBehaviour {
7
8      // Use this for initialization
9      void Start()
10     {
11         //Debug.Log("init");
12
13         if (!SystemInfo.supportsGyroscope)
14         {
15             MobileNativePopups.OpenAlertDialog(
16                 "No Gyroscope", "Your device didn't have Gyroscope sensor, you can't play this game!",
17                 "OK", () => { Application.Quit(); });
18         }
19         else
20             SceneManager.LoadScene("Gameplay", LoadSceneMode.Single);
21     }
22 }
```

Listing B.2: GameplayStatistic.cs

```

1  iżf
2  public class GameplayStatistic {
3
4      public static bool isAnswering = false;
5 }
```

Listing B.3: GameOverController.cs

```

1  iżfusing System;
2  using UnityEngine;
3  using UnityEngine.UI;
4
5  public class GameOverController : MonoBehaviour, IGvrGazeResponder {
6
7      public QuestionMessageController[] itemMessages;
8      public Image pointer;
9      public Animator animator;
10
11     public void OnGazeEnter()
12     {
13         //Do nothing
14     }
15
16     public void OnGazeExit()
17     {
18         //Do nothing
19     }
20
21     public void OnGazeTrigger()
22     {
23         //print("trigger");
24         restartGame();
25     }
26     public void restartGame()
27     {
28         //print("restart");
29         foreach(QuestionMessageController itemMessage in itemMessages)
30         {
31             itemMessage.resetAll();
32         }
33         animator.SetBool("poppedOut", false);
34         pointer.transform.Rotate(new Vector3(0,0,(-1*pointer.transform.rotation.eulerAngles.z)));
35     }
36
37     // Use this for initialization
38     void Start () {
```

```

38 }
39 }
40
41 // Update is called once per frame
42 void Update ()
43 {
44     //print(Math.Abs(pointer.transform.rotation.eulerAngles.z));
45     //RectTransform gameOverTransform= this.GetComponent<RectTransform>();
46     //gameOverTransform.rotation.eulerAngles.Set(gameOverTransform.rotation.eulerAngles.x,
47         //gameOverTransform.rotation.eulerAngles.y,0);
48     if (isGameOver ())
49     {
50         animator.SetBool ("poppedOut" , true);
51         haltAllGameplay ();
52     }
53 }
54 void LateUpdate()
55 {
56     GvrViewer.Instance.UpdateState();
57     if (GvrViewer.Instance.BackButtonPressed)
58     {
59         Application.Quit();
60     }
61 }
62 public bool isGameOver()
63 {
64     return Math.Abs(pointer.transform.rotation.eulerAngles.z) >= 59 && Math.Abs(pointer.transform.
65     rotation.eulerAngles.z) <= 309;
66 }
67 public void haltAllGameplay()
68 {
69     foreach (QuestionMessageController itemMessage in itemMessages)
70     {
71         itemMessage.stopWorking();
72     }
73 }

```

Listing B.4: QuestionMessageController.cs

```

1 iż£
2 using System;
3 using TheNextFlow.UnityPlugins;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class QuestionMessageController : MonoBehaviour , IGvrGazeResponder , HeadMotionListener
8 {
9     public Canvas messageCanvas;
10    public Canvas notificationCanvas;
11    public String[] questions;
12    public bool[] answers;
13    public Image pointer;
14    private Animator animatorMessage;
15    private Animator animatorNotification;
16    private Text messageText;
17    private bool startCounting;
18    private bool isDetecting;
19    private bool clickable;
20    private float timeAvailable;
21    private MotionRecorder mMotionRecorder;
22    private int selectedIndexQuestions;
23
24    public void OnGazeEnter()
25    {
26    }
27
28    public void OnGazeExit()
29    {
30    }
31
32    public void OnGazeTrigger()
33    {
34        if (clickable && !GameplayStatistic.isAnswering)
35        {
36            Input.gyro.enabled = true;
37            Input.gyro.updateInterval = 0.00002F;
38            mMotionRecorder = new MotionRecorder();
39            mMotionRecorder.addHeadMotionListener (this);
40            isDetecting = true;
41            selectedIndexQuestions = (int)(UnityEngine.Random.Range(0.0f , 0.9999f) * questions.Length);
42            messageText.text = questions[selectedIndexQuestions];
43            animatorMessage.SetBool ("poppedOut" , !animatorMessage.GetBool ("poppedOut"));
44            animatorNotification.SetBool ("poppedOut" , !animatorNotification.GetBool ("poppedOut"));
45            clickable = false;
46            GameplayStatistic.isAnswering = true;
47        }
48    }
49
50    // Use this for initialization
51    void Start () {
52        timeAvailable = UnityEngine.Random.Range(5.0f , 10.0f);
53        clickable = false;
54        animatorMessage = messageCanvas.GetComponentInChildren<Animator>();
55        animatorNotification = notificationCanvas.GetComponentInChildren<Animator>();
56        messageText = messageCanvas.GetComponentInChildren<Text>();
57        startCounting = true;

```

```

58     }
59
60     // Update is called once per frame
61     void Update() {
62         if (startCounting)
63         {
64             timeAvailable -= Time.deltaTime;
65         }
66         if (timeAvailable < 0)
67         {
68             startQuestion();
69         }
70         if (isDetecting)
71         {
72             this.mMotionRecorder.inputGryoData(Input.gyro.rotationRate.y, Input.gyro.rotationRate.x);
73             if (Input.GetKeyDown("d")) debugOnNodMotionDetected();
74             if (Input.GetKeyDown("f")) debugOnShookMotionDetected();
75         }
76     }
77
78     public void resetAll()
79     {
80         print("resetting _object");
81         animatorMessage.SetBool("poppedOut", false);
82         animatorNotification.SetBool("poppedOut", false);
83         startCounting = true;
84         isDetecting = false;
85         clickable = false;
86         timeAvailable = UnityEngine.Random.Range(5.0f, 10.0f);
87         mMotionRecorder = null;
88     }
89
90     public void stopWorking()
91     {
92         isDetecting = false;
93         clickable = false;
94     }
95     public void debugOnNodMotionDetected()
96     {
97         this.onMotionDetected(Motion.NOD);
98     }
99
100    public void debugOnShookMotionDetected()
101    {
102        this.onMotionDetected(Motion.SHOOK);
103    }
104
105    public void startQuestion()
106    {
107        startCounting = false;
108        animatorNotification.SetBool("poppedOut", !animatorNotification.GetBool("poppedOut"));
109        clickable = true;
110        timeAvailable = UnityEngine.Random.Range(5.0f, 10.0f);
111    }
112
113    public void onMotionDetected(Motion motion)
114    {
115        if (isDetecting)
116        {
117            if (motion == Motion.NOD)
118            {
119                messageText.text = "YES";
120                if (answers[selectedIndexQuestions])
121                {
122                    pointer.transform.Rotate(new Vector3(0, 0, 10));
123                }
124                else
125                {
126                    pointer.transform.Rotate(new Vector3(0, 0, -10));
127                }
128            }
129            else
130            {
131                messageText.text = "NO";
132                if (!answers[selectedIndexQuestions])
133                {
134                    pointer.transform.Rotate(new Vector3(0, 0, 10));
135                }
136                else
137                {
138                    pointer.transform.Rotate(new Vector3(0, 0, -10));
139                }
140            }
141            clickable = false;
142            startCounting = true;
143            GameplayStatistic.isAnswering = false;
144            animatorMessage.SetBool("poppedOut", false);
145            Input.gyro.enabled = false;
146            isDetecting = false;
147            mMotionRecorder.removeHeadMotionListener(this);
148            mMotionRecorder = null;
149        }
150    }
151 }

```

Listing B.5: Axis.cs

```

2| {
3|     X,
4|     Y
5| }
```

Listing B.6: Detector.cs

```

1 iżf
2 public abstract class Detector {
3
4     protected bool mightHaveActivites;
5
6     protected Pulse[] listPulseToDetect;
7     protected double limitTime;
8     protected float limitSpeed;
9     protected double limitSimpangan;
10
11    public Detector(double limitTime, float limitSpeed, double limitSimpangan)
12    {
13        this.limitTime = limitTime;
14        this.limitSpeed = limitSpeed;
15        this.limitSimpangan = limitSimpangan;
16    }
17
18    public bool isMightHaveActivites()
19    {
20        return mightHaveActivites;
21    }
22
23
24    abstract public bool addPulse(Pulse yPulse);
25 }
```

Listing B.7: HeadMotionListener.cs

```

1 iżfpublic interface HeadMotionListener
2 {
3     void onMotionDetected(Motion motion);
4 }
```

Listing B.8: Motion.cs

```

1 iżfpublic enum Motion
2 {
3     NOD,
4     SHOOK
5 }
```

Listing B.9: MotionRecorder.cs

```

1 iżf
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class MotionRecorder {
6
7     private PulseFactory pulseFactoryXAxis;
8     private PulseFactory pulseFactoryYAxis;
9     private Detector nodDetector;
10    private Detector shookDetector;
11
12    private LinkedList<HeadMotionListener> listListener;
13
14    public MotionRecorder()
15    {
16        listListener = new LinkedList<HeadMotionListener>();
17        pulseFactoryXAxis = new PulseFactory(Axis.X);
18        pulseFactoryYAxis = new PulseFactory(Axis.Y);
19        nodDetector = new NodDetector(700, 2, 0.25);
20        shookDetector = new ShookDetector(700, 2, 0.25);
21    }
22
23    public void addHeadMotionListener(HeadMotionListener listener)
24    {
25        listListener.AddFirst(listener);
26    }
27
28    public void removeHeadMotionListener(HeadMotionListener listener)
29    {
30        listListener.Remove(listener);
31    }
32
33    public void inputGryoData(float x, float y)
34    {
35        Pulse xPulse = pulseFactoryXAxis.inputData(x);
36        Pulse yPulse = pulseFactoryYAxis.inputData(y);
37        if (xPulse != null)
38        {
39            Debug.Log("PULSE");
40            bool shookDetected = shookDetector.addPulse(xPulse);
41            if (shookDetected && !nodDetector
42                .isMightHaveActivites())
43                {// terdeteksi geleng tanpa ada pergerakan pada sumbu y
44
```

```

44        notifyAllMotionListener(Motion.SHOOK);
45        //Log.d("MotionDetectedPulsesX", shookDetector.toString());
46    }
47    if (yPulse != null)
48    {
49        boolean nodDetected = nodDetector.addPulse(yPulse);
50        if (nodDetected && !shookDetector
51            .isMightHaveActivites())
52        { // terdeteksi gelengan tanpa ada pergerakan pada sumbu x
53            notifyAllMotionListener(Motion.NOD);
54        }
55    }
56 }
57
58 public void notifyAllMotionListener(Motion motion)
59 {
60     foreach (HeadMotionListener motionListener in listListener)
61     {
62         motionListener.onMotionDetected(motion);
63     }
64 }
65 }
66 }
```

Listing B.10: NodDetector.cs

```

1 ﻿using System;
2
3 public class NodDetector : Detector
4 {
5     /*
6     private double limitTime;// in millisecons
7     private float limitSpeed = 2;
8     private double limitSimpangan = 0.25;
9     */
10    public NodDetector(double limitTime, float limitSpeed, double limitSimpangan) : base(limitTime,
11                           limitSpeed, limitSimpangan)
12    {
13
14        this.mightHaveActivites = false;
15        this.listPulseToDetect = new Pulse[2];
16        this.listPulseToDetect[0] = PulseFactory.STANDARD_PULSE; //firstPulse
17        this.listPulseToDetect[1] = PulseFactory.STANDARD_PULSE; //lastPulse
18
19    public override bool addPulse(Pulse yPulse)
20    {
21        bool result = false;
22        this.listPulseToDetect[0] = yPulse;
23        if (Math.Abs(this.listPulseToDetect[0].getPeak()) > limitSpeed)
24        {
25            mightHaveActivites = true;
26            bool isPassLimitSimpangan = Math.Abs(this.listPulseToDetect[0].getSimpangan()) >
27                limitSimpangan
28                && Math.Abs(listPulseToDetect[1].getSimpangan()) > limitSimpangan;
29            bool isPassLimitTimeRange =
30                listPulseToDetect[0].getRangeTime() < limitTime && listPulseToDetect[1].getRangeTime() <
31                limitTime;
32            bool isDifferentTypePulse = this.listPulseToDetect[0].getType() != listPulseToDetect[1].
33                getType();
34            result = isPassLimitSimpangan && isPassLimitTimeRange && isDifferentTypePulse;
35        }
36        else
37        {
38            if (Math.Abs(listPulseToDetect[1].getPeak()) < limitSpeed)
39            {
40                mightHaveActivites = false;
41            }
42        }
43        this.listPulseToDetect[1] = this.listPulseToDetect[0];
44        return result;
45    }
46 }
```

Listing B.11: Pulse.cs

```

1 ﻿using System;
2
3 public class Pulse
4 {
5     private double simpangan;
6     private double rangeTime;
7     private double startTime;
8     private double endTime;
9     private float peak;
10    private PulseType type;
11    private Axis axis;
12
13    public Pulse(double simpangan, double startTime, double endTime, float peak, PulseType type,
14               Axis axis)
15    {
16        this.simpangan = simpangan;
17        this.startTime = startTime;
18        this.endTime = endTime;
19        this.rangeTime = endTime - startTime;
20        this.peak = peak;
21        this.type = type;
22    }
23 }
```

```

22     this.axis = axis;
23 }
24
25 public double getSimpangan()
26 {
27     return simpangan;
28 }
29
30 public double getRangeTime()
31 {
32     return rangeTime;
33 }
34
35 public double getStartTime()
36 {
37     return startTime;
38 }
39
40 public double getEndTime()
41 {
42     return endTime;
43 }
44
45 public float getPeak()
46 {
47     return peak;
48 }
49
50 public PulseType getType()
51 {
52     return type;
53 }
54
55 public Axis getAxis()
56 {
57     return axis;
58 }
59 }
```

Listing B.12: PulseFactory.cs

```

1 ﻿using System;
2
3 public class PulseFactory {
4
5     private long lastTime;
6     private float lastData;
7     private double simpangan;
8     private long rangeTime;
9     private double startTime;
10    private double endTime;
11    private float peak;
12    private PulseType currentPulseType;
13    private Axis axis;
14
15    public static Pulse STANDARD_PULSE = new Pulse(0, 0, 0, 0, PulseType.HILL, Axis.X);
16
17    public PulseFactory(Axis axis)
18    {
19        this.axis = axis;
20    }
21
22    private void resetAttributes(PulseType type)
23    {
24        this.simpangan = 0;
25        this.peak = 0;
26        this.currentPulseType = type;
27    }
28
29    private Pulse finishPulse(float data, double currentTime)
30    {
31        this.endTime = ((-lastData / (data - lastData)) * (currentTime - lastTime)) + lastTime;
32        double areaBeforeIntersect = ((endTime - lastTime) / 1000) * lastData / 2;
33        this.simpangan += areaBeforeIntersect;
34        Pulse newPulse = new Pulse(this.simpangan, this.startTime, this.endTime, this.peak,
35            this.currentPulseType,
36            this.axis);
37        this.startTime = endTime;
38        return newPulse;
39    }
40
41    public Pulse inputData(float data)
42    {
43        Pulse result = null;
44        long currentTime = DateTime.Now.Millisecond;
45
46        if (currentPulseType == PulseType.HILL)
47        {
48            if (peak < data)
49            {
50                peak = data;
51            }
52            if (data < 0)
53            { //if values intersect data=0
54                result = finishPulse(data, currentTime);
55                resetAttributes(PulseType.VALLEY);
56            }
57            else
```

```

58         {
59             this.simpangan += ((lastData + data) / 1000) * (currentTime - lastTime) / 2;
60         }
61     } else if (currentPulseType == PulseType.VALLEY)
62     {
63         if (peak > data)
64         {
65             peak = data;
66         }
67         if (data > 0)
68         {
69             //if values intersect data=0
70             result = finishPulse(data, currentTime);
71             resetAttributes(PulseType.HILL);
72         }
73     } else
74     {
75         this.simpangan += ((lastData + data) / 1000) * (currentTime - lastTime) / 2;
76     }
77 }
78
79 lastData = data;
80 lastTime = currentTime;
81 return result;
82 }
83 }
```

Listing B.13: PulseType.cs

```

1  ifdef public enum PulseType
2  {
3      HILL,
4      VALLEY,
5      INIT
6 }
```

Listing B.14: ShookDetector.cs

```

1  ifdef using System;
2
3  public class ShookDetector : Detector
4  {
5      /*
6      private double limitTime;// in milliseccons
7      private float limitSpeed = 2;
8      private double limitSimpangan = 0.25;
9      */
10     public ShookDetector(double limitTime, float limitSpeed, double limitSimpangan) : base(limitTime,
11         limitSpeed, limitSimpangan)
12     {
13         mightHaveActivites = false;
14         listPulseToDetect = new Pulse[3];
15         listPulseToDetect[0] = PulseFactory.STANDARD_PULSE;
16         listPulseToDetect[1] = PulseFactory.STANDARD_PULSE;
17         listPulseToDetect[2] = PulseFactory.STANDARD_PULSE;
18     }
19
20     public override bool addPulse(Pulse xPulse)
21     {
22         bool result = false;
23         listPulseToDetect[2] = listPulseToDetect[1];
24         listPulseToDetect[1] = listPulseToDetect[0];
25         listPulseToDetect[0] = xPulse;//currentPulse
26         if (Math.Abs(listPulseToDetect[0].getPeak()) > limitSpeed)
27         {
28             mightHaveActivites = true;
29             bool isPassLimitSimpangan = Math.Abs(listPulseToDetect[0].getSimpangan()) > limitSimpangan
30                 && Math.Abs(listPulseToDetect[1].getSimpangan()) > limitSimpangan
31                 && Math.Abs(listPulseToDetect[2].getSimpangan()) > limitSimpangan;
32             bool isPassLimitTimeBetween =
33                 (listPulseToDetect[1].getStartTime() - listPulseToDetect[2].getEndTime() == 0) &&
34                 (listPulseToDetect[0].getStartTime() - listPulseToDetect[1].getEndTime() == 0);
35             bool isPassLimitTimeRange =
36                 listPulseToDetect[0].getRangeTime() < limitTime && listPulseToDetect[1].getRangeTime() <
37                     limitTime
38                 && listPulseToDetect[2].getRangeTime() < limitTime;
39             bool isDifferentTypePulse =
40                 (listPulseToDetect[1].getType() != listPulseToDetect[2].getType()) && (listPulseToDetect
41                     [0].getType() != listPulseToDetect[1]
42                     .getType());
43             result = isPassLimitSimpangan && isPassLimitTimeBetween && isDifferentTypePulse
44                 && isPassLimitTimeRange;
45         }
46     }
47     else
48     {
49         if (Math.Abs(listPulseToDetect[2].getPeak()) < limitSpeed
50             && Math.Abs(listPulseToDetect[1].getPeak()) < limitSpeed)
51         {
52             mightHaveActivites = false;
53         }
54     }
55     return result;
56 }
```