IMPLEMENTASI *LINKED LIST* PADA INTERAKSI ANTAR *MARKER AUGMENTED REALITY* UNTUK *OPERAND* DAN *OPERATOR* ARITMETIKA

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Hermawan Wijaya
NIM: 145150200111205

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI *LINKED LIST* PADA INTERAKSI ANTAR *MARKER AUGMENTED REALITY* UNTUK *OPERAND* DAN *OPERATOR* ARITMETIKA

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Hermawan Wijaya
NIM: 145150200111205

Skripsi ini telah diuji dan dinyatakan lulus pada
15 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Wibisono Sukmo Wardhono, S.T., M.T.	Issa Arwani, S.Kom., M.Sc.
NIK: 2010088204041001	NIP: 198309222012121003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 15 Januari 2018

Hermawan Wijaya

NIM: 145150200111205

KATA PENGANTAR

Puji Syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, karena atas rahmat dan bimbingan-Nya penulis dapat menyelesaikan skripsi mengenai "Implementasi *Linked List* pada Interaksi antar *Marker Augmented Reality* untuk Operand dan Operator Aritmetika".

Skripsi ini merupakan tugas akhir yang diajukan untuk memenuhi syarat dalam memperoleh gelar Sarjana Komputer (S.Kom.) pada Fakultas Ilmu Komputer (FILKOM) Universitas Brawijaya Malang.

Pada kesempatan ini penulis juga menyampaikan rasa terima kasih kepada pihak-pihak yang telah membantu penulis selama penyusunan skripsi ini, diantaranya:

1. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Bapak Wibisono Sukmo Wardhono, S.T., M.T. selaku Dosen Pembimbing Satu yang telah meluangkan waktu untuk membimbing penulisan skripsi.
4. Bapak Issa Arwani, S.Kom., M.Sc. selaku Dosen Pembimbing Dua yang telah meluangkan waktu untuk membimbing penulisan skripsi.
5. Bapak Adam Hendra Brata, S.Kom., M.T., M.Sc. selaku Dosen Fasilitator yang telah menjadi fasilitator Seminar Hasil penulis.
6. Bapak Muhammad Aminul Akbar, S.Kom., M.T. dan Bapak Tri Afirianto, S.T., M.T. selaku penguji skripsi yang sudah meluangkan waktu untuk menguji.
7. Teman- teman mahasiswa angkatan 2014 yang telah banyak membantu penulis dan memberikan dukungan dalam menyusun skripsi ini hingga selesai.
8. Semua pihak yang telah membantu menyelesaikan skripsi ini yang tidak dapat penulis sebut satu persatu.

Penulis menyadari masih banyak kekurangan dalam penyusunan Skripsi ini baik dalam teknik penyajian materi maupun pembahasan. Demi kesempurnaan skripsi ini, saran dan kritik yang sifatnya membangun sangat penulis harapkan. Semoga karya tulis ini bermanfaat dan dapat memberikan sumbangan yang berarti bagi pihak yang membutuhkan.

Malang, 15 Januari 2018

Penulis

hermawanwijaya21@gmail.com

ABSTRAK

Augmented Reality (AR) merujuk kepada suatu persepsi langsung terhadap lingkungan sekitar yang sudah ditambahkan dengan komponen virtual komputer. Pada umumnya tidak terdapat mekanisme bawaan untuk menyimpan urutan pada *framework* AR, yang pada konteks ini adalah Vuforia SDK. *Linked List* merupakan salah satu metode dalam Algoritma dan Struktur Data yang digunakan untuk mengorganisasi struktur data dalam sebuah sistem. Urutan *operand* dan *operator* sangat diperhatikan dalam operasi aritmetika agar dapat melakukan kalkulasi aritmetika dengan benar, terutama untuk operasi yang tidak bersifat komutatif maupun asosiatif. Penelitian ini dilakukan untuk menerapkan penggunaan *Linked List* ke dalam AR sebagai sebuah metode untuk menyimpan urutan sementara *marker* yang berhasil dibaca perangkat untuk kemudian diolah lagi berdasarkan urutannya sebagai operasi aritmetika sederhana menggunakan bantuan *engine* Unity 3D. Dari penelitian ini juga dilakukan pengujian *whitebox* dengan nilai hasil *Cyclomatic Complexity* adalah 9, yang menyatakan sistem sudah memenuhi kebutuhan sesuai rancangan dengan kompleksitas yang rendah, serta pengujian *usability* dengan nilai hasil SUS rata-rata adalah 55 yang berarti sistem belum layak untuk digunakan pada pengguna awam.

Kata kunci: *Augmented Reality, Linked List, Struktur Data.*

ABSTRACT

Augmented Reality (AR) refers to a direct perception of the surrounding environment that has been added to the virtual component of the computer. There is generally no default mechanism for storing sequences in the AR framework, which in this context is Vuforia SDK. Linked List is one of the methods in Algorithm and Data Structure used to organize data structures in a system. The operand and operator sequences are highly considered in arithmetic operations in order to perform arithmetic calculations correctly, especially for non-commutative or non-associative operations. This research was conducted to apply the use of Linked List into AR as a method to store the temporary sequence of marker that successfully read by the device then processed again in sequence as simple arithmetic operation using Unity 3D engine aid. From this research also conducted whitebox testing with value of result of cyclomatic complexity is 9, which states that the system meets the needs according to the design with low complexity, and usability testing with average SUS yield value is 55 which means the system is not yet feasible for use on the novice user.

Keywords: *Augmented Reality, Linked List, Data Structure.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Augmented Reality (AR) merujuk kepada suatu persepsi langsung terhadap lingkungan sekitar yang sudah ditambahkan dengan komponen virtual komputer. Beberapa contoh penggunaan AR dalam kehidupan sehari-hari yang sudah tidak asing lagi didengar antara lain *Google Glass* dan HTC Vive. Perkembangan pesat pada AR memungkinkan perkembangan di banyak sektor, antara lain investasi ekonomi, aplikasi, ekspektasi komersial, dan perkembangan sains (Raja, 2017).

Dalam implementasinya, *tracking* pada AR dibagi menjadi *Marker-based* dan *Markerless Tracking* (Petersen, 2015). *Marker-based AR tracking* membutuhkan media tambahan untuk digunakan sebagai *marker*nya, seperti kartu atau foto. Penelitian yang dilakukan menggunakan *Marker-based tracking* untuk aplikasi perhitungan aritmetika sederhana.

Pada umumnya tidak terdapat mekanisme bawaan untuk menyimpan urutan pada *framework* AR, yang pada konteks ini adalah Vuforia SDK. *Linked List* merupakan salah satu metode dalam Algoritma dan Struktur Data yang digunakan untuk mengorganisasi struktur data dalam sebuah sistem. *Linked List* memiliki keunggulan dibandingkan *Array* dalam hal menambahkan dan mengurangi elemennya (Ryan, 2008).

Urutan *operand* dan *operator* sangat diperhatikan dalam operasi aritmetika, terutama untuk operasi yang tidak bersifat komutatif maupun asosiatif (Tapson, 2006). Penelitian ini akan menerapkan penggunaan *Linked List* ke dalam AR sebagai sebuah metode untuk menyimpan urutan sementara terbacanya *marker* yang kemudian diolah lagi berdasarkan urutannya sebagai operasi aritmetika sederhana dengan menggunakan bantuan *engine* Unity 3D.

1.2 Rumusan masalah

1. Bagaimana implementasi pembacaan *marker* pada aplikasi operasi aritmetika sederhana?
2. Bagaimana cara menerapkan urutan operasi aritmetika sederhana pada *marker Augmented Reality*?
3. Bagaimana *usability* implementasi *Linked List* pada aplikasi operasi aritmetika sederhana?

1.3 Tujuan

1. Menerapkan *Augmented Reality* ke dalam aplikasi perhitungan aritmetika sederhana.
2. Menerapkan *Linked List* pada interaksi antar *marker Augmented Reality* untuk operand dan operator aritmetika.
3. Melakukan pengujian *usability* pada aplikasi AR aritmetika sederhana untuk mengetahui seberapa mudah aplikasi digunakan pengguna awam.

1.4 Manfaat

Aplikasi ini diharapkan mampu mengimplementasikan *Linked List* dengan baik sebagai metode untuk melakukan organisasi data terstruktur. Penelitian ini akan berguna pada penggunaan *Linked List* pada *Augmented Reality* pada penelitian-penelitian serupa kedepannya.

1.5 Batasan masalah

1. Aplikasi berjalan pada sistem operasi Android saja secara offline.
2. Operasi yang tersedia adalah penjumlahan, pengurangan, perkalian, pembagian, pangkat, dan kurung.
3. Sistem tidak dapat menampilkan duplikat terhadap model yang *marker*nya terbaca oleh kamera.
4. Sistem tidak dapat melakukan operasi terhadap masukan bilangan negatif.
5. Penanganan error pada operasi ditangani secara umum, di mana sistem akan menampilkan notifikasi error ketika gagal memproses operasi.
6. Hasil dari penelitian adalah aplikasi yang bersifat *algorithm-ready*, bukan aplikasi yang *ready-to-use*.
7. Sistem tidak memiliki mekanisme *Undo*, melainkan menggunakan mekanisme *Reset* ketika semua kartu "*equals*" dijauhkan dari kamera.

1.6 Sistematika pembahasan

Sistematika penulisan laporan penelitian ini dijelaskan sebagai berikut.

BAB I. PENDAHULUAN

Pada bab pendahuluan, penulis membahas tentang alasan dilakukannya penelitian dan penulisan.

BAB II. LANDASAN KEPUSTAKAAN

Pada bab landasan pustaka, penulis menjelaskan konsep-konsep yang digunakan untuk merancang aplikasi *Augmented Reality* operasi aritmetika sederhana.

BAB III. METODOLOGI

Pada bab metodologi, penulis menjelaskan metodologi penelitian yang digunakan untuk merancang aplikasi *Augmented Reality* aritmetika sederhana.

BAB IV. PERANCANGAN

Pada bab ini, penulis melakukan proses perancangan sistem agar dapat memenuhi kebutuhan sesuai dengan tujuan dilakukannya penelitian.

BAB IV. IMPLEMENTASI

Berisi hasil implementasi terhadap aplikasi yang dikembangkan.

BAB VI. PENGUJIAN

Berisi pembahasan tentang pengujian dan analisis terhadap aplikasi yang dikembangkan.

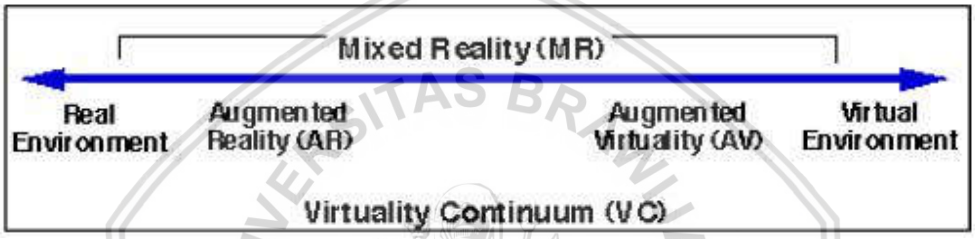
BAB VI. KESIMPULAN

Berisi kesimpulan dan saran dari penelitian yang sudah dilaksanakan oleh penulis.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Augmented Reality

Augmented Reality (AR) merupakan salah satu variasi dari *Virtual Environment* (VE) atau lebih umum disebut sebagai *Virtual Reality* (VR). Teknologi VR ini membuat pengguna seolah-olah berada di dalam lingkungan buatan dan tidak dapat melihat dunia nyata di sekitarnya. Lain halnya dengan AR yang memungkinkan pengguna untuk melihat dunia nyata dengan tambahan objek virtual yang digabungkan dengan lingkungan sekitarnya (Azuma, 1997). AR dapat diartikan sebagai garis tengah antara *Virtual Environment* (sepenuhnya sintetis) dan *telepresence* (sepenuhnya nyata) yang dapat dilihat melalui konsep *virtuality continuum* pada Gambar 2.1 (Milgram, 1994).



Gambar 2.1 Representasi sederhana "virtuality continuum"

Sumber: Milgram (1994)

Konsep *virtuality continuum* berhubungan dengan percampuran kelas objek yang ditampilkan pada situasi tampilan apapun (seperti pada gambar), dimana lingkungan nyata ditunjukan di salah satu ujung *continuum*, dan lingkungan virtual pada ujung satunya. Pada ujung sebelah kiri mendefinisikan lingkungan yang terdiri sepenuhnya dari benda nyata seperti contohnya apa yang kita lihat pada tampilan video konvensional tentang objek dunia nyata. Semakin ke kanan, maka lingkungan yang dilihat dan dirasakan akan semakin maya / tidak terdapat benda dunia nyata

Beberapa kelas tampilan hybrid yang mendukung MR :

1. *Monitor Based Video Display* (*non-immersive*)
2. *Video Display* seperti kelas 1, namun menggunakan *immersive Head Mounted Display* (HMD)
3. HMD yang dilengkapi dengan kemampuan tembus pandang
4. Sama dengan kelas 3, tetapi menggunakan video (kelas 3 bekerja secara optik).
5. *Completely graphic display environment*.
6. *Completely graphic*, dimana objek fisik di lingkungan pengguna memiliki peran dengan skenario buatan komputer.

Selain itu, lingkungan tambahan komputer inklusif lainnya tengah dikembangkan, dimana data asli diterima dan digunakan untuk memodifikasi

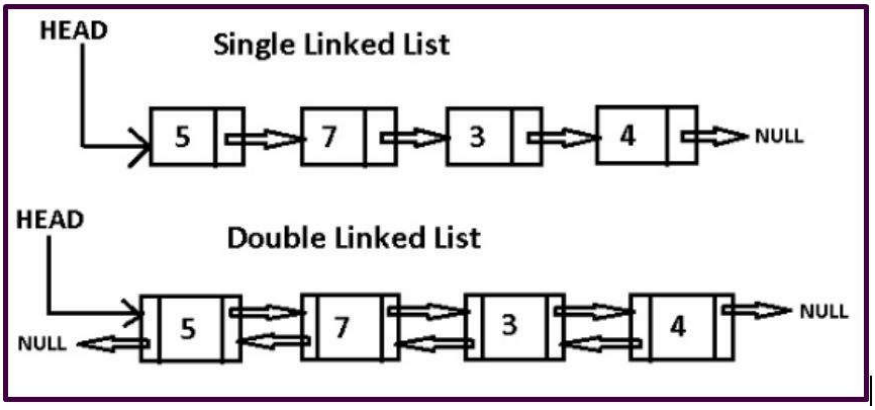
interaksi user dengan dunia yang dijembatani komputer melampaui tampilan visual konvensional.

Definisi AR secara operasional sendiri adalah sesuatu dimana lingkungan asli ditambahkan dengan objek virtual (Milgram, 1994). AR paling menonjol mendukung tampilan kelas 3 (HMD *see through*). Milgram juga menemukan bahwa kelas 1, 2, dan 4 memungkinkan AR dapat diterapkan. Kelas 5 menampilkan sedikit masalah terminologi, karena hal yang ditambahkan (*augmented*) bukanlah representasi langsung dunia nyata, tetapi dunia virtual. Milgram mengajukan display ke 5 sebagai *Augmented Virtuality* (AV). Kelas 6 melampaui kelas 1, 2, 4, dan 5 dalam hal mencantumkan objek dunia nyata yang terlihat secara langsung. Pengalaman melihat tangan nyata sendiri dari seseorang cukup berbeda dengan melihat gambar dari tangan pada monitor. Solusi alternatif dari masalah terminologi untuk kelas 6 adalah istilah *Hybrid Reality* (HR), sebagai sesuatu yang mengarah ke konsep mencampurkan banyak jenis media tampilan.

2.2 *Linked List*

Linked List memiliki kemiripan dengan *array* karena keduanya digunakan untuk menyimpan kumpulan data. Berbeda dengan *array* yang mengalokasikan memori untuk seluruh elemennya ke dalam satu blok memori, *Linked List* mengalokasikan ruang untuk tiap elemen secara terpisah di dalam bloknya sendiri yang dinamakan *"Linked List Element"* atau *"Node". List* mendapatkan keseluruhan strukturnya dengan menggunakan *pointer* yang menghubungkan semua *node*nya seperti hubungan pada rantai. Tiap *node* memiliki 2 *field*, yaitu *data field* dan *next field*. *Data field* digunakan untuk menyimpan tipe elemen apapun yang disimpan pada *list*, sedangkan *next field* digunakan sebagai *pointer* untuk menghubungkan suatu *node* ke *node* berikutnya (Parlante, 2001). *Linked List* ini yang kemudian akan digunakan pada penelitian ini untuk menyusun struktur urutan terbacanya *marker Augmented Reality* pada aplikasi operasi aritmetika sederhana yang dikembangkan.

Terdapat 3 jenis *Linked List* yang ada dalam struktur data, antara lain *Single Linked List, Double Linked List,* dan *Circular Linked List*. *Single Linked List* terdiri dari kumpulan *node* di mana tiap *node* memiliki elemen data dan sebuah *pointer* ke elemen berikutnya di dalam *list*. *Double Linked List* terdiri dari kumpulan *node* di mana tiap *node* memiliki elemen data dan dua *pointer* ke elemen berikutnya dan sebelumnya di dalam *list*. Dan *Circular Linked List* merupakan *list* sederhana di mana *node* terakhirnya mengarah kepada *node* pertama di dalam *list* (Csegeek, 2013). Untuk representasi gambar *Single Linked List* dan *Double Linked List* dapat dilihat pada Gambar 2.2.



Gambar 2.2 Representasi *Single* dan *Double Linked List*

Sumber: Csegeek.com (2013)

Tabel 2.1 Contoh Penggunaan *Linked List* pada Bahasa C#

```
1   public class Node {
2       public Node next;
3       public Object data;
4   }
5
6   public class LinkedList {
7       private Node head;
8       public void printAllNodes() {
9           Node current = head;
10          while (current != null) {
11              Console.WriteLine(current.data);
12              current = current.next;
13          }
14      }
15
16      public void AddFirst(Object data) {
17          Node toAdd = new Node();
18          toAdd.data = data;
19          toAdd.next = head;
20          head = toAdd;
21      }
22
23      public void AddLast(Object data) {
24          if (head == null) {
```

```
25              head = new Node();
26              head.data = data;
27              head.next = null;
28          }
29          else {
30              Node toAdd = new Node();
31              toAdd.data = data;
32              Node current = head;
33              while (current.next != null) {
34                  current = current.next;
35              }
36              current.next = toAdd;
37          }
38      }
39  }
40
41  class Program {
42      static void Main(string[] args)  {
43          Console.WriteLine("Add First:");
44          LinkedList myList1 = new LinkedList();
45          myList1.AddFirst("Hello");
46          myList1.AddFirst("Magical");
47          myList1.AddFirst("World");
48          myList1.printAllNodes();
49          Console.WriteLine();
50          Console.WriteLine("Add Last:");
51          LinkedList myList2 = new LinkedList();
52          myList2.AddLast("Hello");
53          myList2.AddLast("Magical");
54          myList2.AddLast("World");
55          myList2.printAllNodes();
56          Console.ReadLine();
57      }
58  }
```

Sumber: Dmytro (2013)

Berdasarkan contoh pada Tabel 2.1, pembuatan kelas *Node* pada *Linked List* terdapat pada baris 1 sampai 4. Elemen yang terdapat di dalam *Node* adalah *next field* yang terdapat pada baris 2, dan *data field* yang terdapat pada baris 3. Baris 6 hingga 14 merupakan kelas LinkedList yang digunakan untuk inisialisasi awalan

/ *head* suatu *Linked List* yang di dalamnya juga terdapat *method* untuk mencetak seluruh isi dari *Linked List* melalui *method* printAllNodes. *Method* printAllNodes akan terus mencetak *Node* yang dijelajahi sampai *pointer* saat ini kosong / *null. Method* AddFirst dengan parameter *data* pada baris 16 sampai 21 merupakan contoh penggunaan *method* yang digunakan untuk menambahkan *node* baru di awal *Linked List.* Cara kerjanya adalah dengan membuat *node* baru dengan nama toAdd, lalu mengisi *data field* dari *node* toAdd tadi sesuai dengan *parameter* yang dilewatkan. Setelah itu, memberi nilai pada *next field* dengan nilai *head*, hal ini dilakukan agar posisi *node* toAdd terletak sebelum *node head*. Dan terakhir, menjadikan *node* toAdd sebagai *head*. Baris 23 sampai 38 menjelaskan tentang *method* AddLast dengan melewatkan *parameter data* yang digunakan untuk menambahkan *node* baru di akhir *Linked List*. Cara kerja dari *method* ini adalah dengan membuat *node* baru saat nilai *head* adalah *null* (*Linked List* masih kosong), serta mencari ujung *Linked List* untuk menambahkan *node* baru ketika *Linked List* tidak kosong. Kelas program pada tabel diatas menjelaskan bagaimana pemanggilan *method* untuk memodifikasi konten dari *Linked List*.

2.3 Operasi Aritmetika

Operasi aritmetika sederhana terdiri dari penjumlahan, pengurangan, perkalian dan pembagian. Operasi ini kemudian dikembangkan lebih lanjut sebagai dasar dari manipulasi operand seperti persentase, akar, pangkat, dan fungsi logaritma. Aritmetika dihitung berdasarkan urutan operasinya (Tapson, 2006).

2.3.1 Penjumlahan

Penjumlahan merupakan sebuah operasi dasar dari operasi aritmetika. Pada bentuk paling sederhananya, penjumlahan menambahkan dua angka *addend* menjadi satu angka hasil penjumlahan. Menambahkan lebih dari dua angka dapat disebut dengan penjumlahan berulang. Penjumlahan bersifat komutatif dan asosiatif, yang berarti urutan penjumlahan tidak akan mempengaruhi hasil dari operasi. (Tapson, 2006)

2.3.2 Pengurangan

Pengurangan merupakan kebalikan dari penjumlahan. Tujuan dari pengurangan adalah mencari selisih terhadap dua angka, yaitu *minuend* dikurangi dengan *subtrahend*. Jika *minuend* lebih besar dari *subtrahend*, maka hasil dari operasi adalah bilangan positif. Sebaliknya jika *subtrahend* lebih besar dari *minuend*, maka hasil dari operasinya adalah bilangan negatif. Jika keduanya sama, maka selisihnya adalah 0. Pengurangan tidak bersifat komutatif maupun asosiatif, yang berarti urutan operasi akan mempengaruhi hasil dari operasi (Tapson, 2006).

2.3.3 Perkalian

Perkalian merupakan operasi sederhana yang kedua dari aritmetika. Operasi ini menggabungkan dua bilangan, *multiplier* dan *multiplicand* menjadi satu hasil

perkalian. Perkalian bersifat komutatif, asosiatif, serta distributif terhadap penjumlahan dan perkalian. Perkalian dengan angka 1 akan menghasilkan bilangan itu sendiri (Tapson, 2006).

2.3.4 Pembagian

Pembagian merupakan kebalikan dari operasi perkalian yang digunakan untuk mencari hasil bagi dari dua bilangan, dividend dan *divisor*. Semua *dividend* yang dibagi dengan 0 tidak terdefinisi. Jika nilai *dividend* lebih besar dari *divisor*, maka hasil baginya adalah lebih dari 1. Sebaliknya jika *divisor* lebih besar dari *dividend*, maka hasil baginya akan kurang dari 1 dan berlaku untuk bilangan positif maupun bilangan negatif. Operasi ini tidak bersifat komutatif maupun asosiatif (Tapson, 2006).

2.4 Kalkulator *Postfix*

Dalam penelitian ini, sistem yang dikembangkan menggunakan masukan data berupa notasi *infix* yang kemudian akan diubah menjadi notasi *postfix* untuk dioperasikan. Notasi *postfix* ini digunakan agar pengecekan prioritas operasi dapat diotomatisasi dan dapat mengolah operasi yang lebih kompleks dibandingkan menggunakan notasi *infix*. Tabel 2.2 merupakan program untuk melakukan kalkulasi aritmetika yang digunakan pada penelitian ini :

Tabel 2.2 Contoh Implementasi Kalkulator *Postfix* pada Bahasa C#

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace calculator
7   {
8       public enum OperatorType { MULTIPLY, DIVIDE, ADD,
9       SUBTRACT, EXPONENTIAL, OPAREN, CPAREN };
10      public interface Element
11      {
12      }
13
14      public class NumberElement : Element
15      {
16          double number;
17          public Double getNumber()
18          {
19              return number;
```

```
20          }
21
22          public NumberElement(String number)
23          {
24              this.number = Double.Parse(number);
25          }
26
27          public override String ToString()
28          {
29              return ((int)number).ToString();
30          }
31      }
32
33      public class OperatorElement : Element
34      {
35          public OperatorType type;
36          char c;
37          public OperatorElement(char op)
38          {
39              c = op;
40              if (op == '+')
41                  type = OperatorType.ADD;
42              else if (op == '-')
43                  type = OperatorType.SUBTRACT;
44              else if (op == '*')
45                  type = OperatorType.MULTIPLY;
46              else if (op == '/')
47                  type = OperatorType.DIVIDE;
48              else if (op == '^')
49                  type = OperatorType.EXPONENTIAL;
50              else if (op == '(')
51                  type = OperatorType.OPAREN;
52              else if (op == ')')
53                  type = OperatorType.CPAREN;
54          }
55
56          public override String ToString()
57          {
```

```
58              return c.ToString();
59          }
60      }
61
62      public class Parser
63      {
64          List<Element> e = new List<Element>();
65          public List<Element> Parse(String s)
66          {
67              StringBuilder sb = new StringBuilder();
68              for (int i = 0; i < s.Length; i++)
69              {
70                  char c = s[i];
71                  if (Char.IsDigit(c))
72                      sb.Append(c);
73                  if (i + 1 < s.Length)
74                  {
75                      char d = s[i + 1];
76                      if (Char.IsDigit(d) == false &&
77  sb.Length > 0)
78                      {
79                          e.Add(new
80  NumberElement(sb.ToString()));
81                          //clears stringbuilder
82                          sb.Remove(0, sb.Length);
83                      }
84                  }
85
86                  if (c == '+' || c == '-' || c == '*' || c
87  == '/' || c == '^'
88                          || c == '(' || c == ')')
89                      e.Add(new OperatorElement(c));
90              }
91              if (sb.Length > 0)
92                  e.Add(new NumberElement(sb.ToString()));
93
94              return e;
95          }
```

```
96      }
97
98      public class InfixToPostfix
99      {
100         List<Element> converted = new List<Element>();
101         int Precedence(OperatorElement c)
102         {
103             if (c.type == OperatorType.EXPONENTIAL)
104                 return 2;
105             else if (c.type == OperatorType.MULTIPLY ||
106 c.type == OperatorType.DIVIDE)
107                 return 3;
108             else if (c.type == OperatorType.ADD || c.type
109 == OperatorType.SUBTRACT)
110                 return 4;
111             else
112                 return Int32.MaxValue;
113         }
114
115         public void ProcessOperators(Stack<Element> st,
116 Element element, Element top)
117         {
118             while (st.Count > 0 &&
119 Precedence((OperatorElement)element) >=
120 Precedence((OperatorElement)top))
121             {
122                 Element p = st.Pop();
123                 if (((OperatorElement)p).type ==
124 OperatorType.OPAREN)
125                     break;
126                 converted.Add(p);
127                 if (st.Count > 0)
128                     top = st.First();
129             }
130         }
131         public List<Element>
132 ConvertFromInfixToPostFix(List<Element> e)
133         {
```

```
134             List<Element> stack1 = new List<Element>(e);
135             Stack<Element> st = new Stack<Element>();
136             for (int i = 0; i < stack1.Count; i++)
137             {
138                 Element element = stack1[i];
139                 if
140 (element.GetType().Equals(typeof(OperatorElement)))
141                 {
142                     if (st.Count == 0 ||
143                             ((OperatorElement)element).typ
144 e == OperatorType.OPAREN)
145                         st.Push(element);
146                     else
147                     {
148                         Element top = st.First();
149                         if
150 (((OperatorElement)element).type == OperatorType.CPAREN)
151                             ProcessOperators(st, element,
152 top);
153                         else if
154 (Precedence((OperatorElement)element) <
155 Precedence((OperatorElement)top))
156                             st.Push(element);
157                         else
158                         {
159                             ProcessOperators(st, element,
160 top);
161                             st.Push(element);
162                         }
163                     }
164                 }
165                 else
166                     converted.Add(element);
167             }
168
169             //pop all operators in stack
170             while (st.Count > 0)
171             {
```

```
172                 Element b1 = st.Pop();
173                 converted.Add(b1);
174             }
175 
176             return converted;
177         }
178 
179         public override String ToString()
180         {
181             StringBuilder s = new StringBuilder();
182             for (int j = 0; j < converted.Count; j++)
183                 s.Append(converted[j].ToString() + " ");
184             return s.ToString();
185         }
186     }
187 
188     public class PostFixEvaluator
189     {
190         Stack<Element> stack = new Stack<Element>();
191 
192               NumberElement  calculate(NumberElement  left,
NumberElement right, OperatorElement op)
193         {
194             Double temp = Double.MaxValue;
195             if (op.type == OperatorType.ADD)
196                                 temp  =  left.getNumber()  +
right.getNumber();
197             else if (op.type == OperatorType.SUBTRACT)
198                                 temp  =  left.getNumber()  -
right.getNumber();
199             else if (op.type == OperatorType.MULTIPLY)
200                                 temp  =  left.getNumber()  *
right.getNumber();
201             else if (op.type == OperatorType.DIVIDE)
202                                 temp  =  left.getNumber()  /
right.getNumber();
203             else if (op.type == OperatorType.EXPONENTIAL)
204                 temp = Math.Pow(left.getNumber(),
right.getNumber());
205 
```

```
210             return new NumberElement(temp.ToString());
211         }
212         public Double Evaluate(List<Element> e)
213         {
214             List<Element> v = new List<Element>(e);
215             for (int i = 0; i < v.Count; i++)
216             {
217                 Element element = v[i];
218                 if
219 (element.GetType().Equals(typeof(NumberElement)))
220                     stack.Push(element);
221                 if
222 (element.GetType().Equals(typeof(OperatorElement)))
223                 {
224                     NumberElement right =
225 (NumberElement)stack.Pop();
226                     NumberElement left =
227 (NumberElement)stack.Pop();
228                     NumberElement result = calculate(left,
229 right, (OperatorElement)element);
230                     stack.Push(result);
231                 }
232             }
233             return
234 ((NumberElement)stack.Pop()).getNumber();
235         }
236     }
237
238     class Program
239     {
240         public double Calculate(String s)
241         {
242             Parser p = new Parser();
243             List<Element> e = p.Parse(s);
244             InfixToPostfix i = new InfixToPostfix();
245             e = i.ConvertFromInfixToPostFix(e);
246
247             PostFixEvaluator pfe = new PostFixEvaluator();
```

```
248                 return pfe.Evaluate(e);
249             }
250
251             static void Main(string[] args)
252             {
253                 Program c = new Program();
254                 double d = c.Calculate("4+6+9*8-(5*6+9)^2");
256                 Console.WriteLine(d);
257             }
258         }
259     }
```

Sumber: Eric (2012)

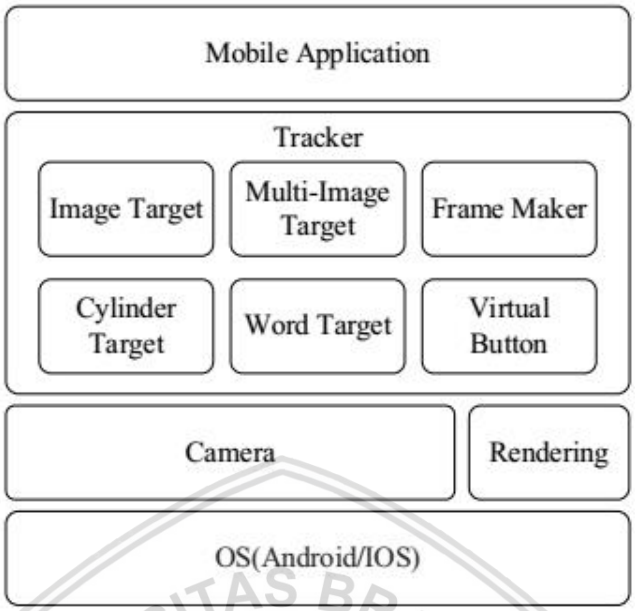
2.5 Vuforia

Vuforia merupakan *Source Development Kit* (SDK) gratis yang digunakan untuk mengimplementasikan *Mobile Augmented Reality*. SDK yang mendukung iOS, Android, dan Unity 3D ini diluncurkan oleh Qualcomm di tahun 2010. *Platform* Vuforia memungkinkan kita untuk merancang satu aplikasi *native* yang dapat diakses sebagian besar pengguna *smartphone* dan *tablet*. Keseluruhan arsitektur Vuforia dapat dilihat pada Gambar 2.3. Untuk menggunakan layanan SDK Vuforia, terdapat beberapa kebutuhan, antara lain;

Tabel 2.3 Tabel Kebutuhan Sistem Vuforia

Sistem Operasi Perangkat Bergerak	Android 4.1.x 32-bit
Sistem Operasi Perangkat Pengembangan	Windows 7+
Versi Unity	Unity 2017.2
API Grafis Android	OpenGL ES 2.0, OpenGL ES 3.x
API Grafis Windows	DirextX 11 di Windows 10

Sumber: Vuforia (2017)



Gambar 2.3 Arsitektur Vuforia

Sumber: Xiao (2014)

Terdapat dua metode pengenalan pada SDK Vuforia, yaitu pengenalan citra dan pengenalan teks. Juga terdapat dua metode untuk menyimpan basis data citra, yaitu basis data perangkat dan basis data *cloud*. Perbedaan mendasar basis data perangkat terletak pada ketidakbutuhan koneksi internet, waktu respon lebih cepat, terbatas untuk 100 target per target basis data perangkat serta tidak terdapat dukungan metadata. Pada basis data *cloud*, waktu respon tergantung dari kecepatan koneksi internet, mendukung lebih banyak target basis data dan metadata (Xiao, 2014). Penelitian kali ini menggunakan basis data perangkat dalam implementasinya.

Proses pengembangan dengan Vuforia dapat dibagi menjadi 4 tahap:

1. Membuat basis data target gambar lokal yang digunakan untuk *Augmented Reality*.
2. Mendefinisikan pengaturan target gambar di *file* XML.
3. Melakukan *load file* XML dan memulai pengenalan gambar pada program.
4. Melakukan *load* dan menampilkan model 3D.

2.6 White Box Testing

White box testing merupakan metode desain kasus uji yang menggunakan struktur kontrol desain prosedural untuk memperoleh kasus uji (Pressman, 2005, hal 485). Dengan metode pengujian ini, kita mampu memperoleh kasus uji sebagai berikut :

1. Menjamin semua jalur independen pada suatu modul telah digunakan setidaknya satu kali
2. Menguji semua keputusan logis pada sisi *true* dan *false*
3. Melakukan eksekusi seluruh perulangan/*loop* pada batasan tertentu
4. Melakukan validasi terhadap struktur data internal

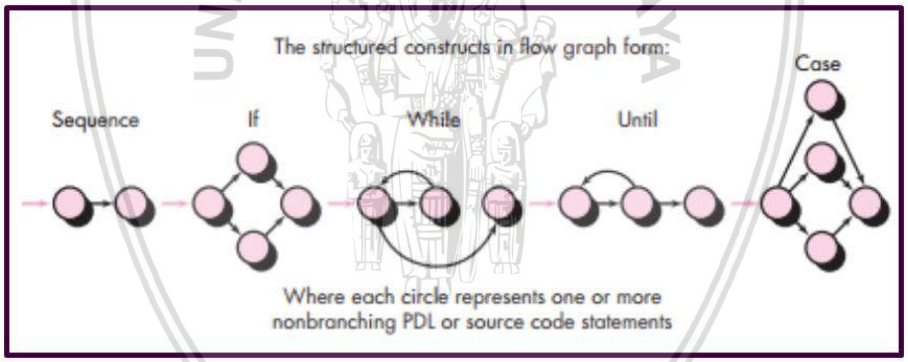
Terdapat dua jenis pengujian pada *White Box Testing,* antara lain Pengujian Jalur Dasar/*Basis Path Testing* dan Pengujian Struktur Kontrol/*Control Structure Testing* (Pressman, 2005).

2.6.1 Basis Path Testing

Basis Path Testing merupakan pengujuan struktural yang dibuat berdasarkan ukuran tingkat kompleksitas dari algoritma hasil perancangan (Pressman, 2005). Langkah-langkah pengujian ini antara lain :

1. Mendefinisikan *flow graph* berdasarkan *mapping* dari *flow chart* atau struktur algoritma.
2. Menentukan ukuran kompleksitas (*Cyclomatic Complexity*).
3. Mendefinisikan kasus uji.

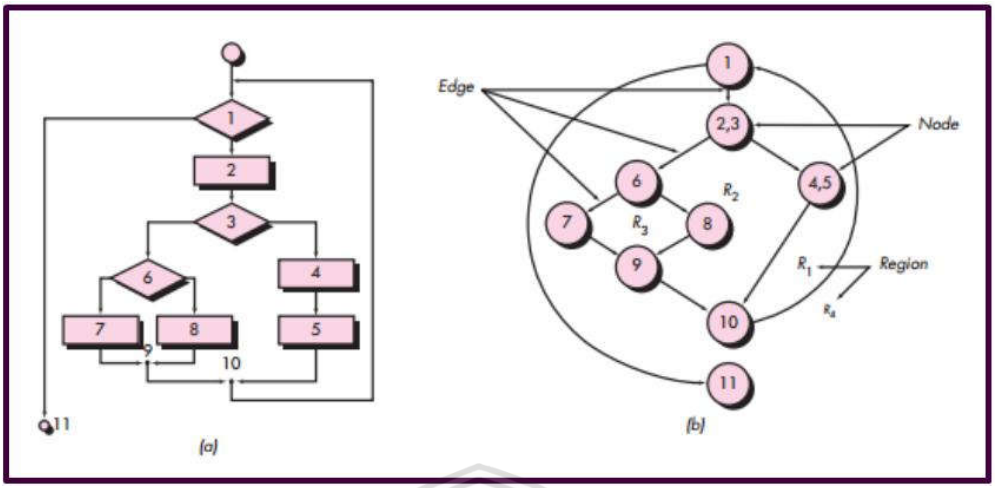
Flow Graph merupakan notasi sederhana yang merepresentasikan aliran kontrol (Pressman, 2005). Notasi tersebut digambarkan pada Gambar 2.4.



Gambar 2.4 Konstruksi Terstruktur dalam bentuk *Flow Graph*

Sumber: Pressman (2005)

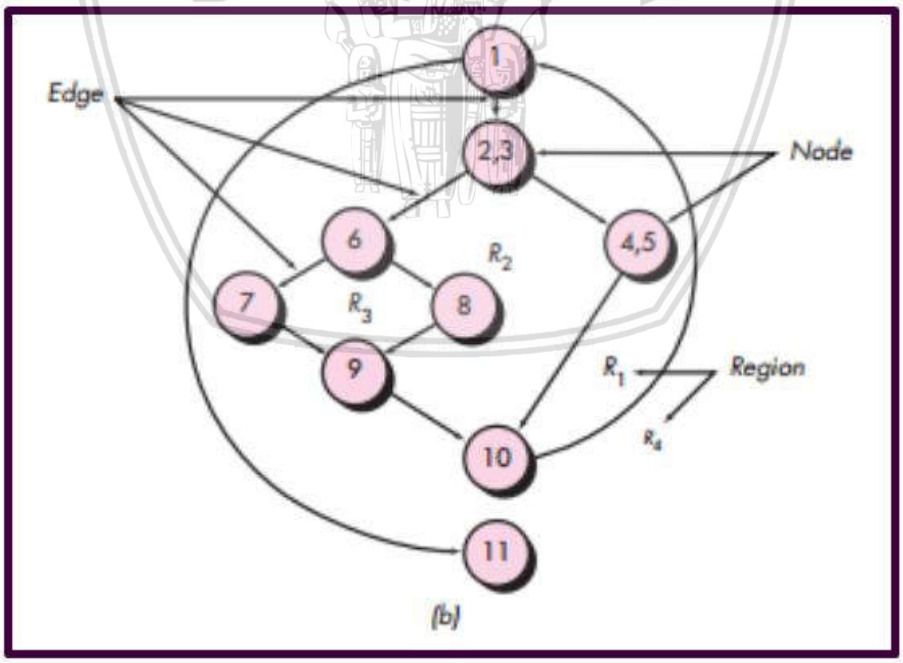
1. Proses dan keputusan yang berurutan dipetakan menjadi satu *node.*
2. Setiap *edge* harus berakhir pada sebuah *node* meskipun tidak merepresentasikan proses apapun.
3. *Region* merupakan daerah yang dibatasi oleh *edge* dan *node*.
4. *Predicate node* merupakan *node* yang merupakan kondisi dimana dua atau lebih *edge* akan keluar dari sini.



Gambar 2.5 Contoh Transformasi *Flow Chart* ke *Flow Graph*

Sumber: Pressman (2005)

Cyclomatic Complexity merupakan angka yang menyatakan jumlah jalur independen/jalur dasar dari sebuah program (Pressman, 2005). Representasi dari kompleksitas program ini menunjukan jumlah pengujian (kasus uji) yang harus dieksekusi. Jalur independen/*independent path* sendiri merupakan tiap jalur dalam program yang memiliki setidaknya satu set pernyataan/*processing statement* atau satu kondisi yang belum digunakan oleh jalur sebelumnya.



Gambar 2.6 Contoh *Flow Graph*

Sumber: Pressman (2005)

Berdasarkan *Flow Graph* pada Gambar 2.6, terdapat 4 jalur independen, antara lain :

1. Jalur 1 : 1-11
2. Jalur 2 : 1-2-3-4-5-10-1-11
3. Jalur 3 : 1-2-3-6-8-9-10-1-11
4. Jalur 4 : 1-2-3-6-7-9-10-1-11
5. Bukan jalur independen : 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

Perhitungan matematis *Cyclomatic Complexity* - V(G), sebagai berikut :

1. V(G) = jumlah *region*
2. V(G) = E - N + 2
3. V(G) = P + 1

Contoh perhitungan matematis *Cyclomatic Complexity* - V(G) berdasarkan *Flow Graph* di atas, sebagai berikut :

1. V(G) = 4, karena terdapat 4 *region* (R1, R2, R3, R4).
2. V(G) = 11 - 9 + 2 = 4, karena terdapat 11 *edges* dan 9 *nodes.*
3. V(G) = 3 + 1 = 4, karena terdapat 3 *predicate node*.

Dari perhitungan matematis *Cyclomatic Complexity* - V(G) berdasarkan *Flow Graph* di atas dapat disimpulkan bahwa nilai V(G) pada *Flow Graph* di atas adalah 4. Nilai V(G) yang diperoleh dapat dievaluasi resikonya yang dapat dilihat pada Tabel 2.4, di mana semakin besar nilainya maka resiko terhadap tingkat kompleksitas juga semakin tinggi dan mempengaruhi kesulitan untuk melakukan modifikasi terhadap program.

Tabel 2.4 Tabel Evaluasi Resiko *Cyclomatic Complexity*

Cyclomatic Complexity	**Risk Evaluation**	**Probability of Bad Fix**
1-10	Low risk, testable code	5%
11-20	Moderate risk	10%
21-50	High risk	30%
>50	Very high risk, untestable code	40%

Sumber: Prabhu K (2014)

2.6.2 Control Structure Testing

Pengujian struktur kontrol digunakan sebagai pelengkap bagi pengujian jalur dasar/*basis path testing.* Terdapat dua jenis pengujian dalam *Control Structure Testing*, antara lain pengujian kondisi/*condition testing* dan pengujian perulangan/*loop testing* (Pressman, 2005).

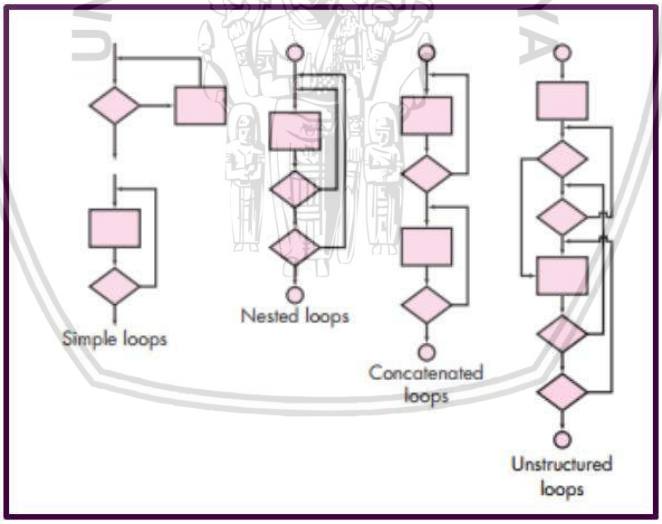
2.6.2.1 Condition Testing

Condition Testing merupakan pengujian *white box* yang digunakan untuk menguji kondisi logika dalam sebuah program. Terdapat dua jenis kondisi, yaitu kondisi sederhana/*simple condition* dan kondisi majemuk/*compound condition*. Kondisi sederhana terdiri dari sebuah ekspresi relasi atau sebuah ekspresi boolean, sedangkan kondisi majemuk terdiri dari dua atau lebih kondisi sederhana, operator boolean, dan tanda kurung (Pressman, 2005).

Strategi pengujian yang digunakan pada pengujian kondisi antara lain adalah pengujian cabang/*branch testing* dan pengujian domain/*domain testing.* Pada pengujian cabang, pengujian dilakukan untuk tiap cabang *true* atau *false* dari kondisi, minimal sekali dilakukan untuk setiap cabangnya. Sedangkan pada pengujian domain, pengujian dilakukan untuk setiap kemungkinan nilai dari ekspresi kondisi. Untuk ekspresi boolean diperlukan 2^n kasus uji untuk tiap *n* variabel (Pressman, 2005).

2.6.2.2 Loop Testing

Loop Testing merupakan pengujian *white box* yang dilakukan untuk menguji validitas dari struktur loop. Terdapat empat kelas *loops* yang berbeda, antara lain *simple loops*, *nested loops*, *concatenated loops*, dan *unstructured loops* seperti pada Gambar 2.7 (Pressman, 2005).



Gambar 2.7 Variasi Kelas Perulangan

Sumber: Pressman (2005)

Pengujian yang dapat diterapkan ke dalam *simple loops*, dimana *n* adalah angka maksimal diperbolehkannya melewati perulangan, antara lain :

1. Kasus uji tidak melewati *loop* sama sekali.
2. Kasus uji *m* kali melewati *loop*, dimana $m < n$.
3. Kasus uji *n*-1, *n*, *n*+1 melewati *loop*.

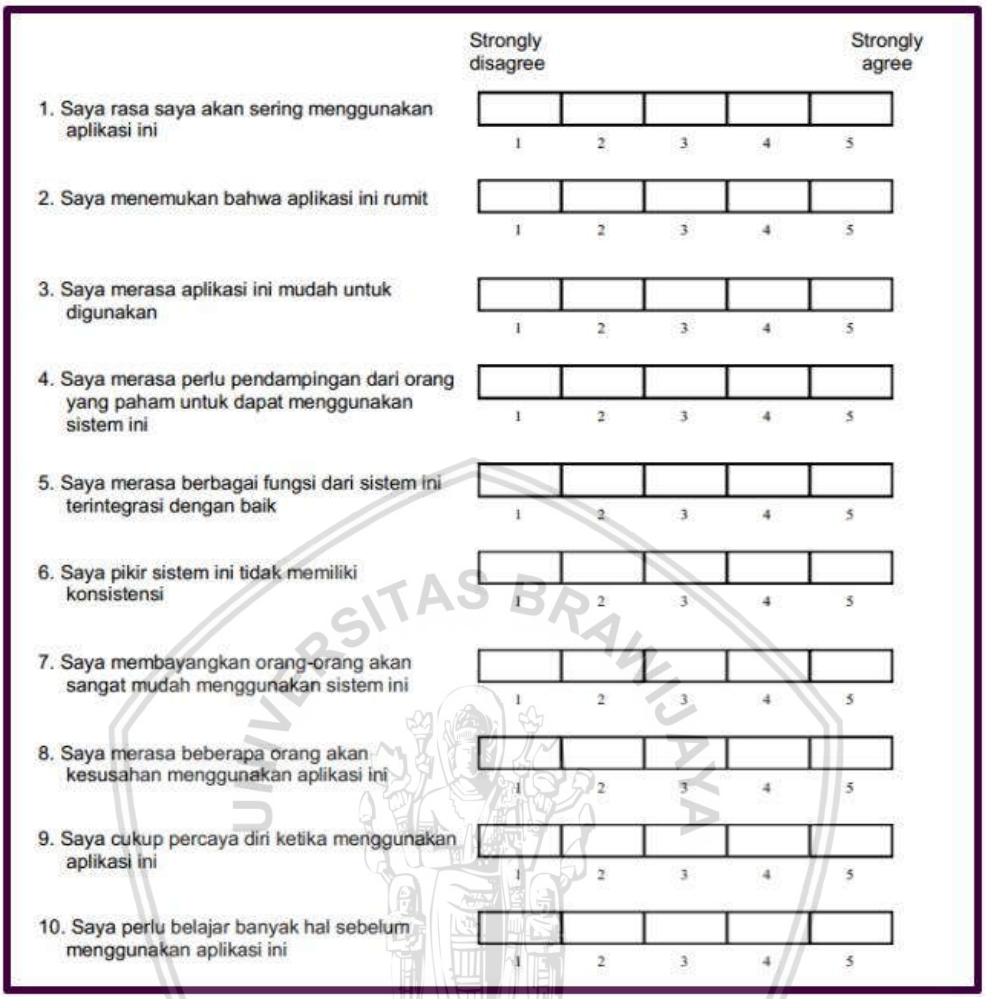
Pengujian yang dapat diterapkan ke dalam *nested loops*, antara lain :

1. Pengujian dimulai dari *loop* paling dalam. Berikan nilai minimum pada *loop iterator* yang lain.
2. Lakukan pengujian *simple loop* untuk *loop* paling dalam, sementara *loop* luarnya di*set* pada *iterator* yang minimum.
3. Pengujian berjalan keluar dengan melakukan pengujian untuk *loop* berikutnya, namun tetap menjaga nilai minimum pada semua *loop* luarnya, serta nilai tipikal untuk *nested loops* lainnya.
4. Lanjutkan pengujian sampai semua *loop* sudah diuji.

Pengujian pada *concatenated loops* dapat disamakan dengan prosedur pengujian *simple loops* jika tiap perulangan tidak bergantung satu sama lain. Jika perulangan bergantung pada satu sama lain, maka gunakan pengujian yang sama dengan pengujian pada *nested loops.* Jika terdapat *unstructured loops*, maka sebisa mungkin lakukan desain ulang untuk merepresentasikan penggunaan konstruksi pemrograman terstruktur. (Pressman, 2005).

2.7 Pengujian *Usability*

Pengujian *usability* digunakan untuk melihat bagaimana kemudahan penggunaan aplikasi oleh pengguna. Pengujian dijalankan dengan menggunakan kuisioner yang terdiri dari 10 pertanyaan serta pilihan jawaban yang terdiri dari Sangat Tidak Setuju (STS), Tidak Setuju (TS), Netral (N), Setuju (S), dan Sangat Setuju (SS) di setiap pertanyaannya. Penyusunan pertanyaan dikondisikan sesuai dengan format *System Usability Scale* (SUS) seperti pada Gambar 2.8. Pengujian dilakukan kepada 6 orang responden, hal ini dilakukan demi efisiensi dan hasil yang optimal (Nielsen, 2000).



Gambar 2.8 Contoh Kuesioner *System Usability Scale*

Sumber: Diadaptasi dari Brooke (1996)

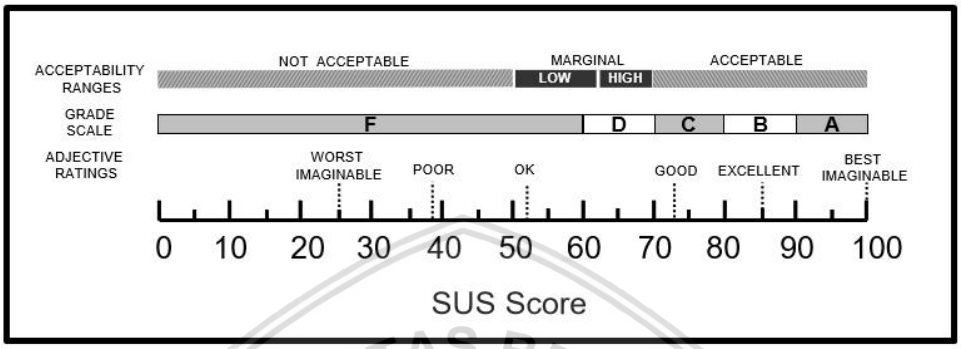
2.7.1 Penggunaan *System Usability Scale*

SUS pada umumnya diajukan setelah responden telah selesai menggunakan sistem yang akan dievaluasi, tetapi sebelum dilakukan diskusi atau penjelasan singkat akan sistem. Responden diarahkan agar langsung mengisi kuesioner tanpa menghabiskan waktu lama untuk berpikir. Tiap pertanyaan harus diisi, apabila responden bingung dalam menjawab pertanyaannya, mereka harus diarahkan untuk mengisi poin tengah dari skala (Brooke, 1996).

2.7.2 Penilaian *System Usability Scale*

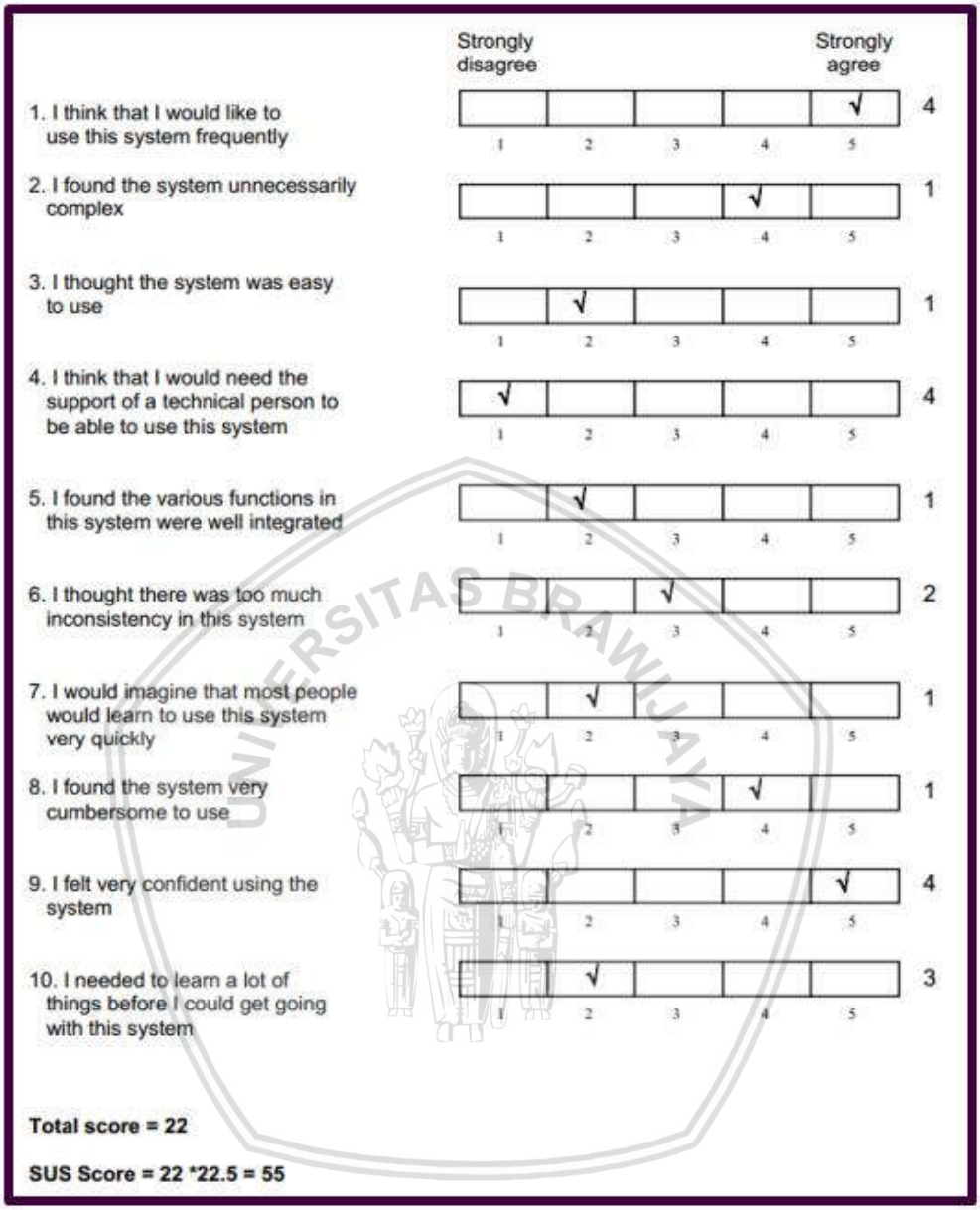
Untuk menghitung nilai SUS, pertama jumlahkan seluruh nilai berdasarkan kontribusinya pada tiap pertanyaan. Kontribusi tiap pertanyaan berada di antara angka 0 sampai 4. Untuk pertanyaan 1, 3, 5, 7, dan 9 kontribusi nilainya adalah posisi skala dikurang dengan 1. Untuk pertanyaan 2, 4, 6, 8, dan 10 kontribusi

nilainya adalah 5 dikurangi dengan posisi skala. Kalikan jumlah nilai tersebut dengan 2.5 untuk memperoleh nilai keseluruhan dari SUS (Brooke, 1996). Contoh pada Gambar 2.10 terdapat sedikit kesalahan pada perkalian (tertulis 22 * 22.5 = 55), yang seharusnya adalah 22 * 2.5 = 55. Pengkategorian skor untuk SUS dapat dilihat pada Gambar 2.9.



Gambar 2.9 Pengkategorian Skor SUS

Sumber: Bangor (2009)

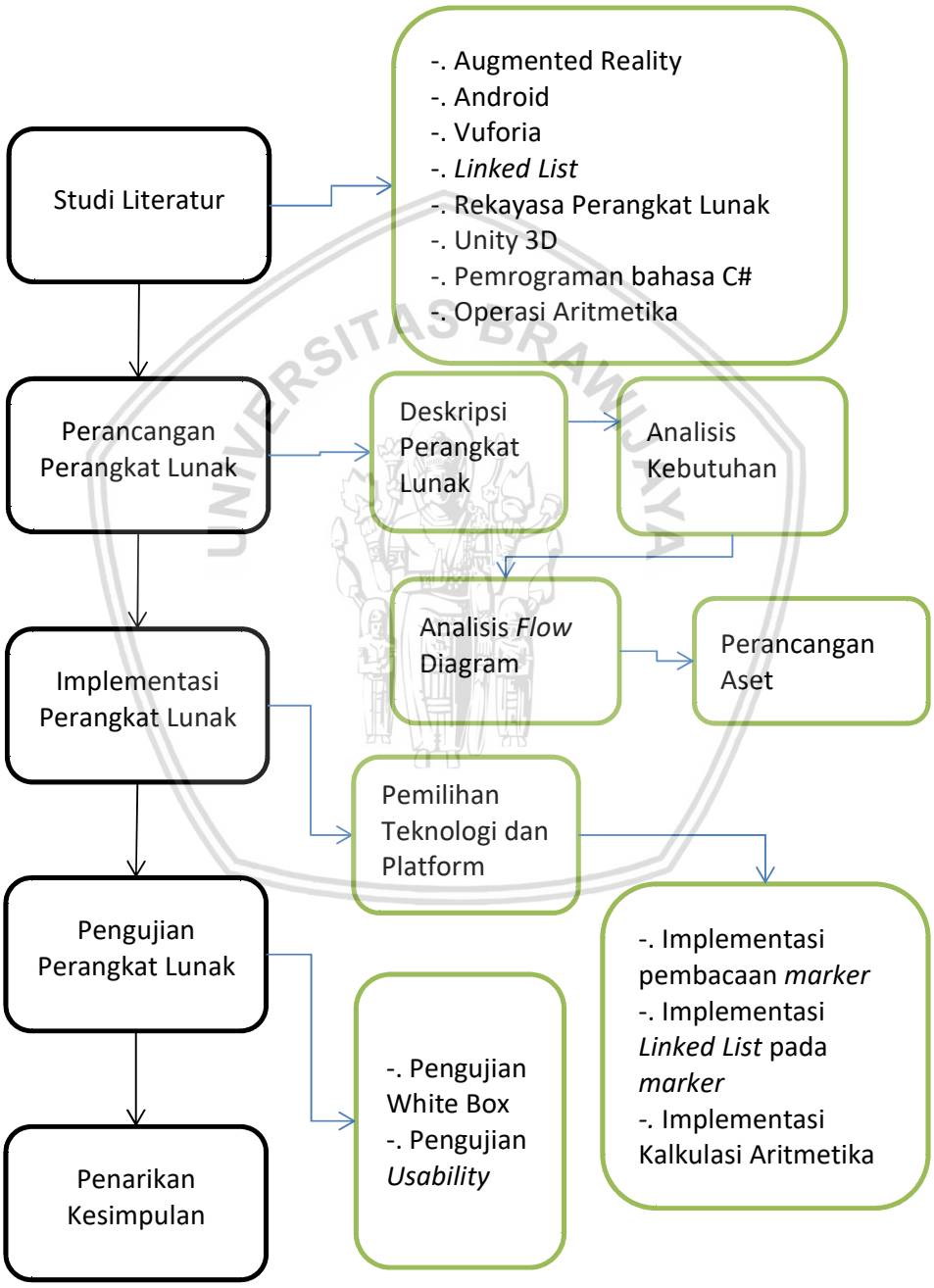


Gambar 2.10 Contoh Penilaian *System Usability Scale*

Sumber: Brooke (1996)

BAB 3 METODOLOGI

Pada bab ini akan dijelaskan mengenai prosedur dan kegiatan-kegiatan yang akan dilakukan dalam pengerjaan skripsi. Dimulai dari studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian perangkat lunak, dan penarikan kesimpulan.



Gambar 3.1 Diagram Blok Penelitian

3.1 Studi Literatur

Metode penelitian ini memerlukan tahapan studi literatur dalam pengerjaannya. Studi Literatur merupakan tahap penelusuran pengetahuan dalam rangka menyusun dasar teori yang digunakan sebagai dasar pengetahuan dalam penelitian ini. Penelusuran pengetahuan ini bersumber dari buku, jurnal, dan internet (website resmi) yang berkaitan dengan informasi yang dibutuhkan dalam mengerjakan penelitian ini. Teori dan daftar pustaka yang akan digunakan sebagai landasan dasar pengetahuan dalam melaksanakan penelitian ini diantaranya adalah literatur sebagai berikut :

1. Augmented Reality
2. Android
3. Vuforia
4. *Linked List*
5. Rekayasa Perangkat Lunak
6. Unity 3D
7. Pemrograman bahasa C#
8. Operasi aritmetika

3.2 Perancangan Perangkat Lunak

Tahap perancangan perangkat lunak dilakukan ketika studi literatur selesai dilakukan. Pada tahap ini akan dilakukan analisis terhadap kebutuhan sistem dan menggunakan analisis *flow* diagram sebagai media untuk melakukan desain terhadap sistem yang dibangun. Selain itu pada tahap ini akan disertakan perancangan aset berupa *marker* 2D dan model 3D dari aplikasi yang akan dikembangkan dengan memanfaatkan beberapa tools desain seperti GIMP, Blender, dan Inkscape.

3.3 Implementasi Perangkat Lunak

Setelah tahap perancangan perangkat lunak telah dilakukan, implementasi dilakukan terhadap perangkat lunak yang dikembangkan. Pada tahap implementasi perangkat lunak akan dilakukan implementasi terhadap hasil rancangan aplikasi sesuai dengan yang telah dilakukan pada tahap perancangan perangkat lunak. Penelitian ini akan menggunakan teknologi *Augmented Reality* pada *game engine* Unity yang digunakan untuk mengembangkan aplikasi AR aritmetika sederhana ini dengan bantuan SDK Vuforia. Tahap ini terdiri dari pemilihan teknologi dan *platform*, implementasi pembacaan *marker*, implementasi *linked list* pada *marker,* dan implementasi kalkulasi aritmetika.

Bagian pemilihan teknologi dan platform menjelaskan kebutuhan perangkat yang digunakan selama proses implementasi. Kebutuhan yang digunakan dibagi menjadi kebutuhan perangkat komputer untuk proses pengembangan dan kebutuhan perangkat bergerak untuk proses pengujian.

Implementasi pembacaan *marker* dan implementasi linked *list* pada *marker* dilakukan dengan menggunakan bahasa pemrograman C# dengan menggunakan

Visual Studio sebagai IDE (*Integrated Development Environment*)nya. Editor Unity 2017.2 juga digunakan selama proses pengembangan aplikasi ini. SDK Vuforia yang digunakan merupakan SDK yang terintegrasi langsung sebagai *plugin* dari Unity 2017.2 itu sendiri.

3.4 Pengujian Perangkat Lunak

Setelah sistem dan aplikasi diimplementasikan, tahapan selanjutnya yang dilakukan adalah pengujian. Pada tahap pengujian akan dilakukan pengujian terhadap aplikasi AR aritmetika sederhana yang dikembangkan untuk menunjukkan bahwa aplikasi yang dikembangkan dapat bekerja sesuai dengan hipotesa dan harapan penelitian yang dilaksanakan. Pengujian yang antara lain *white box testing* dan *usability testing*.

Pengujian secara *white box* perlu dilakukan untuk menguji fungsionalitas sistem, sehingga dapat diketahui apakah sistem yang dikembangkan sudah sesuai dengan kebutuhan dan sesuai dengan rancangan sistem yang sudah dirancang sebelumnya. Pengujian ini dilakukan dengan menentukan kasus uji terlebih dahulu. Jumlah dari kasus uji adalah sebanyak jalur independen yang didapatkan. Selanjutnya pengujian dilakukan sesuai dengan prosedur yang tertera di setiap kasus ujinya.

Pengujian *usability* dilakukan untuk mengetahui apakah sistem yang dikembangkan dapat digunakan dengan mudah atau tidak. Hal ini ditujukan untuk memenuhi rumusan masalah yaitu untuk menguji nilai *usability* dari sistem yang dikembangkan. Pengujian *usability* ini dilakukan dengan menggunakan kuesioner *System Usability Scale* atau SUS kepada 6 orang responden berusia 21 tahun yang dipilih secara acak. Responden dijelaskan secara singkat tentang aplikasi ini sebelum melakukan pengujian. Setelah pengisian kuesioner SUS, penguji menerima masukan dari responden untuk dipertimbangkan sebagai saran penelitian kedepannya.

3.5 Penarikan Kesimpulan

Penarikan kesimpulan dilakukan setelah seluruh tahapan pengembangan aplikasi telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap akhir dari penelitian ini adalah penyertaan saran dari hasil pengujian yang didapatkan agar dapat memperbaiki kekurangan dan kesalahan yang terjadi, serta menyempurnakan penelitian yang akan dilakukan selanjutnya.

BAB 4 PERANCANGAN

4.1 Deskripsi Perangkat Lunak

Aplikasi ini memiliki tujuan untuk membaca urutan *marker* dan mendaftarkan urutan tersebut pada *Double Linked List* untuk diolah lagi. Pemilihan *Double Linked List* ini digunakan agar penelitian kedepannya dapat menerapkan mekanisme *undo* pada sistem. Cara penggunaan aplikasi ini adalah dengan mengarahkan kamera ke arah *marker* satu per satu. Pengguna akan mendapatkan informasi yang ditampilkan pada layar perangkat, antara lain model 3D yang ditampilkan saat *marker* terdeteksi, teks *marker* apa yang terdeteksi saat ini, konten dari *Linked List* saat ini, notifikasi untuk *error* maupun hasil perhitungan, jumlah objek yang terdeteksi saat ini, serta hasil konversi dari notasi *infix* ke *postfix*.

4.2 Analisis Kebutuhan

Berikut merupakan beberapa poin yang dibutuhkan dalam aplikasi yang dikembangkan.

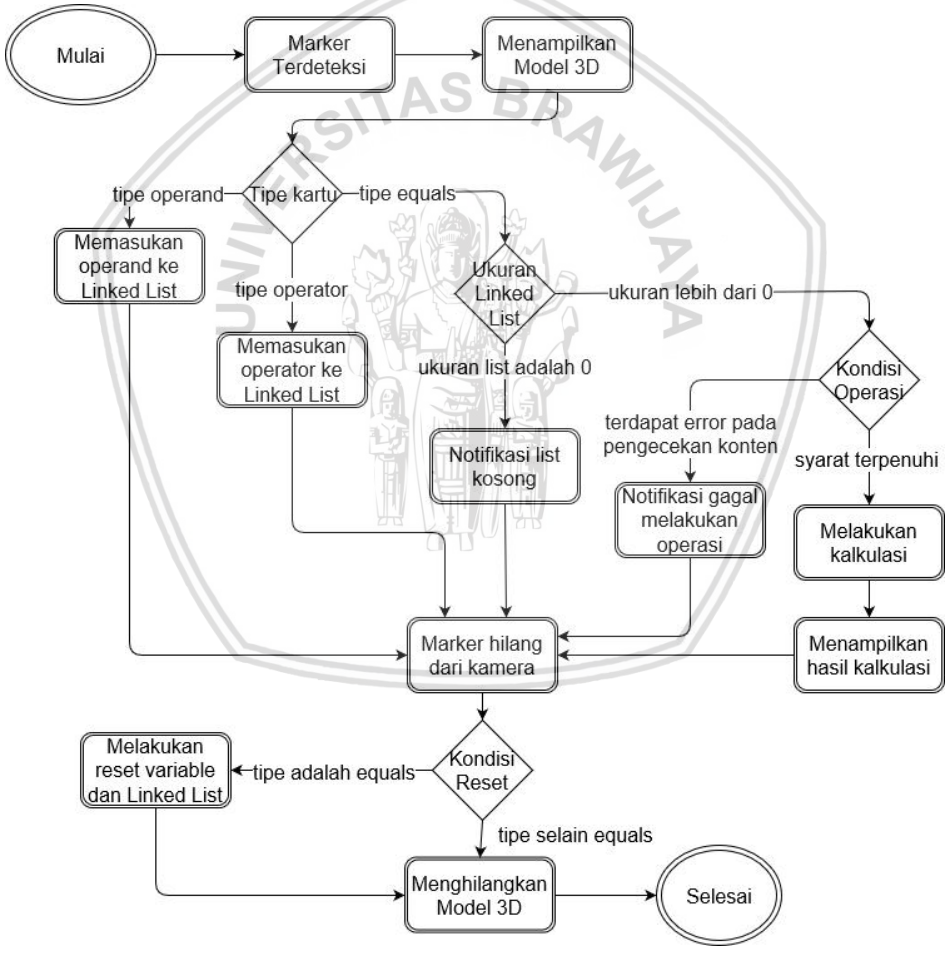
1. Sistem dapat mengakses kamera perangkat yang digunakan.
2. Sistem dapat melakukan deteksi terhadap *marker* dan menampilkan model sesuai *marker*nya.
3. Sistem dapat memasukan nilai dari *marker* operand dan operator ke dalam Linked List.
4. Sistem dapat menampilkan notifikasi ketika *marker equals* terbaca, yang berisi pemberitahuan list kosong, operasi gagal, ataupun hasil dari perhitungan aritmetika.
5. Sistem dapat melakukan kalkulasi dari masukan yang berupa Linked List dengan konten notasi *infix*.
6. Sistem dapat melakukan perhitungan aritmetika dengan melakukan konversi notasi *infix* ke notasi *postfix*.
7. Sistem dapat menghilangkan model 3D apabila *marker* tidak terdeteksi oleh kamera.
8. Sistem mampu melakukan reset terhadap konten Linked List dan variabel-variabel yang digunakan apabila *marker* yang hilang bertipe *equals*.

4.3 Analisis *Flow* Diagram

Gambar 4.1 merupakan analisis *flow* diagram dari sistem yang dikembangkan. Pada awal aplikasi berjalan, kamera perangkat bergerak akan diaktifkan untuk membaca *marker*. Saat *marker* terdeteksi, sistem akan menampilkan Model 3D berdasarkan *marker* yang terdaftar di dalam database lalu membedakan apakah objek yang terdeteksi tergolong *operand*, *operator*, atau *equals*. Saat objek terdeteksi sebagai *operand* ataupun *operator, s*istem akan memasukan nilai dari objek yang terdeteksi ke dalam *Linked List*.

Apabila sistem mendeteksi objek bertipe *equals,* maka sistem akan membedakan apakah *Linked List* sudah terisi atau belum. Jika *Linked List* belum terisi, maka sistem akan menampilkan notifikasi bahwa *list* kosong dan tidak melakukan operasi. Jika *Linked List* sudah terisi, sistem akan melakukan pengecekan untuk syarat operasi aritmetika. Jika terdapat error pada konten dalam *Linked List*, sistem akan menampilkan notifikasi bahwa operasi gagal. Selain itu sistem akan melakukan kalkulasi dan menampilkan hasil dari operasi aritmetika.

Saat *marker* bertipe *equals* dijauhkan dari kamera, sistem akan menjalankan fungsi *reset*. Fungsi ini melakukan *reset* terhadap variabel-variabel dan *Linked List* yang digunakan selama aplikasi berjalan ke kondisi semula. Sistem juga akan menghilangkan model 3D dari tampilan perangkat pengguna apabila *marker* bertipe apapun menghilang dari pandangan kamera perangkat.



Gambar 4.1 Analisis *Flow* Diagram

4.4 Perancangan Aset

Aset dirancang dengan menggunakan software Inkscape untuk *Marker* 2D dan Blender untuk Model 3D. Aset dibuat se-sederhana mungkin agar dapat

mudah diterima oleh pengguna namun tetap memiliki tingkat *augmentability* yang baik.

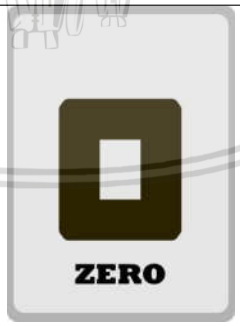
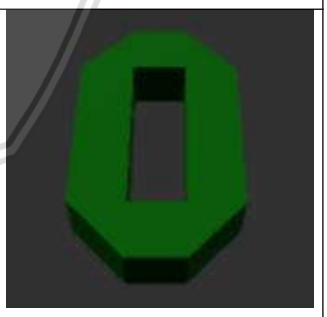
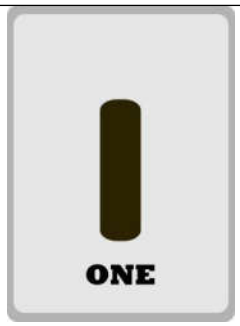
4.4.1 *Marker* 2D

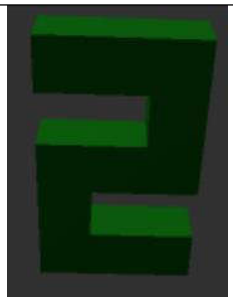
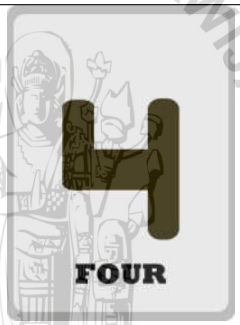
Marker yang akan digunakan ke dalam aplikasi operasi aritmetika sederhana ini menggunakan citra dengan tingkat kedalaman warna 24-bit, yang dapat diperoleh dengan menyimpan citra ke format ekstensi jpeg. Terdapat 10 buah *marker* bertipe operand, 7 buah *marker* bertipe operator, serta 1 buah bertipe untuk *equals*. *Marker* bertipe operand terdiri dari angka 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9. *Marker* bertipe operator terdiri dari operator penjumlahan (+), pengurangan (-), perkalian (*), pembagian (/), pangkat (^), kurung buka, dan kurung tutup. *Marker* bertipe *equals* merupakan *marker* khusus yang berisi nilai sama dengan (=). *Marker* yang sudah dirancang dapat dilihat di Tabel 4.1.

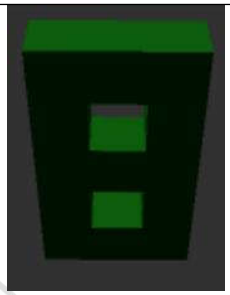
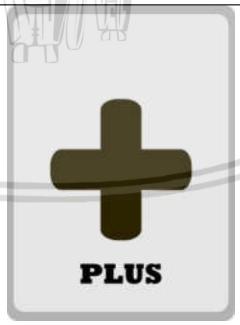
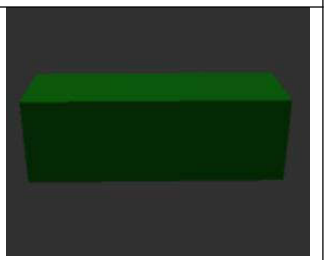
4.4.2 Model 3D

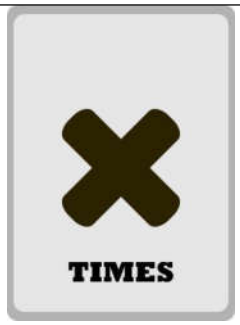
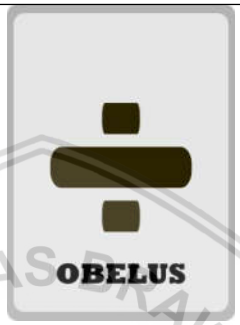
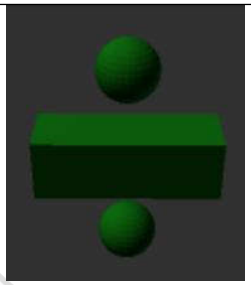
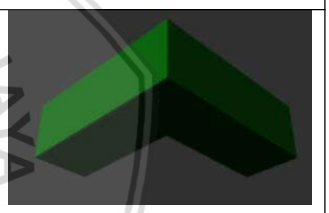
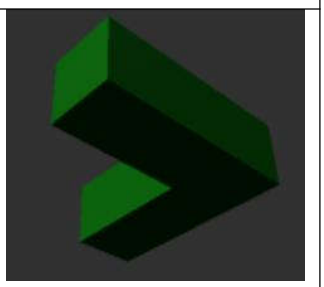
Berikut merupakan rancangan model 3D yang akan digunakan ke dalam aplikasi operasi aritmetika sederhana ini. Bentuk yang digunakan berdasar dari objek kubus yang ditransformasi sedemekian rupa hingga merepresentasikan operand dan operator aritmetika sesuai dengan *marker*nya. Pastikan model memiliki nilai normal yang benar, atau dapat diatur dengan melakukan Ctrl+N saat berada di *edit mode* dengan melakukan select ke semua objek. Hal ini dilakukan agar saat model blender diimpor di unity semua *face* nya menghadap ke posisi yang benar dan tidak ada *mesh* yang hilang atau tidak terlihat. Model yang sudah dirancang dapat dilihat di Tabel 4.1.

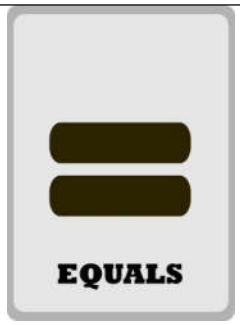
Tabel 4.1 Tabel *Marker* dan Model

Tipe	Nilai	*Marker* 2D	Model 3D
Operand	0		
Operand	1		

Operand	2		
Operand	3		
Operand	4		
Operand	5		
Operand	6		

Operand	7		
Operand	8		
Operand	9		
Operator	+		
Operator	-		

Operator	*		
Operator	/		
Operator	^		
Operator	(		
Operator	)		

Equals	=		

BAB 5 IMPLEMENTASI

5.1 Pemilihan Teknologi dan Platform

Proses implementasi aplikasi ini dilakukan menggunakan komputer dengan spesifikasi sebagai berikut :

Tipe Perangkat : Lenovo Ideapad 310 80TU

Sistem Operasi : Windows 10 Home Single Language 64-bit

Prosesor : Intel Core i5-7200U CPU @ 2.50GHz (4 CPU), ~2.7GHz

Memory : 4096MB RAM

DirectX Version : DirectX 12

GPU : NVIDIA GeForce 920MX

VRAM : 2020MB

Perangkat Android yang digunakan selama proses implementasi aplikasi ini memiliki spesifikasi sebagai berikut :

Tipe Perangkat : Asus Zenpad 8 P024

Sistem Operasi : Android 6.0.1 32-bit (Marshmallow)

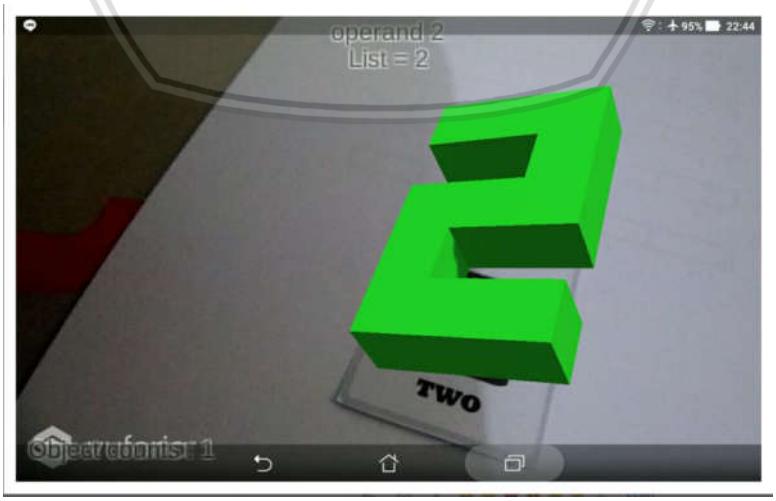
Prosesor : 8 core ARMv7 Processor (VFPv4, NEON), 200 ~ 1362.2MHz

Memory : 2927 RAM

GPU : Adreno^{TM} 405, OpenGL ES 3.1 V@140.0

Resolusi Layar : 800 x 1280

5.2 Implementasi Pembacaan *Marker*



Gambar 5.1 *Marker* operand 2 saat terdeteksi kamera

Bagian ini menjelaskan proses pengiriman *marker* ke vuforia untuk kemudian dijadikan sebuah database *marker* yang dapat diimpor ke engine Unity 3D. *Marker* yang sudah dirancang di*upload* ke server Vuforia untuk dimasukan ke dalam database. Setelah semua *marker* berhasil di*upload*, lakukan *download* terhadap database dan lakukan impor database tersebut ke unity. Gambar 5.1 merupakan hasil yang ditampilkan saat *marker* operand bernilai 2 terdeteksi kamera AR.

Tabel 5.1 merupakan kelas MathARTrackableEventHandler yang merupakan kelas yang melakukan kontrol terhadap perubahan *state* pada *marker* yang terdaftar dalam aplikasi. Kelas ini juga akan melakukan perintah untuk memasukan data, melakukan perhitungan, serta melakukan reset terhadap *Linked List.* Perintah untuk memasukan data dapat dilihat pada baris 32 untuk data operand dan baris 37 untuk data operator. Perintah untuk melakukan perhitungan dapat dilihat di baris 50 di mana sistem akan memanggil fungsi calculatePostfix dari kelas MathARLinkedList. Untuk melakukan *reset*, pada baris 67 sampai 70 sistem akan menjalankan perintah *reset* saat *marker* bertipe *equals* hilang setelah ditampilkan. Tabel 5.2 berisi penjelasan tiap barisnya untuk kelas MathARTrackableEventHandler.

Tabel 5.1 Kelas MathARTrackableEventHandler

```
1   using UnityEngine;
2   using Vuforia;
3
4   public class MathARTrackableEventHandler : MonoBehaviour,
5   ITrackableEventHandler {
6       public string cardValue, cardType;
7       public MainController main;
8
9       protected TrackableBehaviour mTrackableBehaviour;
10
11      protected virtual void Start() {
12          mTrackableBehaviour =
13  GetComponent<TrackableBehaviour>();
14          if (mTrackableBehaviour)
15  mTrackableBehaviour.RegisterTrackableEventHandler(this);
16      }
17
18      public void OnTrackableStateChanged(
19          TrackableBehaviour.Status previousStatus,
20          TrackableBehaviour.Status newStatus) {
```

```
21          if (newStatus ==
22  TrackableBehaviour.Status.DETECTED ||
23              newStatus ==
24  TrackableBehaviour.Status.TRACKED ||
25              newStatus ==
26  TrackableBehaviour.Status.EXTENDED_TRACKED) {
27              Debug.Log("Trackable " +
28  mTrackableBehaviour.TrackableName + " found");
29              OnTrackingFound();
30              main.uitext.text = cardType +" "+ cardValue;
31              if (cardType == "operand") {
32                  main.list.insertLast(cardValue,
33  cardType);
34                  main.detectedOperand++;
35              }
36              else if (cardType == "operator") {
37                  main.list.insertLast(cardValue,
38  cardType);
39                  main.detectedOperator++;
40              }
41              else if (cardType == "equals") {
42                  if (main.list.getSize() != 0) {
43                      if (main.list.isError()) {
44                          main.txtNotif.text = "Unable to
45  operate";
46                      }
47                      else {
48                          main.txtPostFix.text = "Postfix =
49  " + main.list.getPostFixValue();
50                          main.txtNotif.text = "Result : "
51  + main.list.calculatePostfix().ToString();
52                      }
53                  }
54                  else {
55                      main.txtNotif.text = "Zero list
56  detected";
57                  }
58              }
```

```
59              main.addDetectedObjCount();
60          }
61          else if (previousStatus ==
62  TrackableBehaviour.Status.TRACKED &&
63              newStatus ==
64  TrackableBehaviour.Status.NOT_FOUND) {
65              Debug.Log("Trackable " +
66  mTrackableBehaviour.TrackableName + " lost");
67              if (cardType == "equals") {
68                  main.list.clearList();
69                  main.resetVariables();
70              }
71              main.txtNotif.text = "";
72              main.substractDetectedObjCount();
73              OnTrackingLost();
74          }
75          else {
76              OnTrackingLost();
77          }
78      }
79
80      protected virtual void OnTrackingFound() {
81          var rendererComponents =
82  GetComponentsInChildren<Renderer>(true);
83          var colliderComponents =
84  GetComponentsInChildren<Collider>(true);
85          var canvasComponents =
86  GetComponentsInChildren<Canvas>(true);
87          // Enable rendering:
88          foreach (var component in rendererComponents)
89              component.enabled = true;
90          // Enable colliders:
91          foreach (var component in colliderComponents)
92              component.enabled = true;
93          // Enable canvas':
94          foreach (var component in canvasComponents)
95              component.enabled = true;
96      }
```

```
97
98      protected virtual void OnTrackingLost() {
99          var rendererComponents =
100 GetComponentsInChildren<Renderer>(true);
101         var colliderComponents =
102 GetComponentsInChildren<Collider>(true);
103         var canvasComponents =
104 GetComponentsInChildren<Canvas>(true);
105         // Disable rendering:
106         foreach (var component in rendererComponents)
107             component.enabled = false;
108         // Disable colliders:
109         foreach (var component in colliderComponents)
110             component.enabled = false;
111         // Disable canvas':
112         foreach (var component in canvasComponents)
113             component.enabled = false;
114     }
115 }
```

Tabel 5.2 Pembahasan Kode MathARTrackableEventHandler

1-2	Perintah untuk melakukan impor *library* unity dan vuforia.
4-5	Pembuka kelas dengan MonoBehaviour dan ITrackableEventHandler.
6	Variabel bertipe string untuk tipe kartu dan nilai kartu.
7	Variabel untuk objek MainController yang merupakan kontroler dari aplikasi ini.
9	Variabel mTrackableBehaviour yang digunakan untuk menggunakan behaviour deteksi vuforia.
11-16	*Method* Start yang dijalankan saat kelas dipanggil. *Method* ini akan mendaftarkan behaviour deteksi vuforia.
18-78	*Method* yang dijalankan saat ada perubahan state pada deteksi objek.
21-60	*Method* ini akan dijalankan saat ada *marker* yang terdeteksi

	kamera.
31-35	Percabangan if ketika tipe kartu yang terdeteksi adalah operand. Sistem akan menambahkan *node* baru pada linked *list* kontroler main.
36-40	Percabangan if ketika tipe kartu yang terdeteksi adalah operator. Sistem akan menambahkan *node* baru pada linked *list* kontroler main.
	Percabangan if ketika tipe kartu yang terdeteksi adalah *equals*.
41-58	Sistem akan melakukan kalkulasi terhadap konten linked *list* apabila ukuran linked *list* bukan 0 dan tidak terjadi error dalam pengecekan isError.
61-74	*Method* yang akan dijalankan saat terdapat perubahan state dari objek terdeteksi menjadi tidak terdeteksi. *Method* ini akan menghilangkan model 3D dan akan melakukan *reset* variabel dan linked *list* apabila tipe kartu adalah *equals*.
75-77	Percabangan else yang dijalankan saat kondisi deteksi selain berhasil deteksi objek dan berpindah state.
80-96	*Method* yang dijalankan saat deteksi menemukan objek yang terdaftar di dalam database. *Method* ini akan menampilkan objek 3D sesuai dengan objek yang berhasil dikenali melalui kamera.
98-114	*Method* yang dijalankan saat deteksi gagal menemukan objek yang terdaftar di dalam database. *Method* ini akan menonaktifkan objek 3D dari tampilan perangkat.

5.3 Implementasi *Linked List* pada *Marker*

Gambar 5.2 Memasukan *marker* secara urut

Bagian ini menjelaskan implementasi *Linked List* pada *marker* dengan cara kerja sesuai dengan rancangan pada bab perancangan. Pada Gambar 5.2, *marker* dibaca secara urut dari operand 2, operator perkalian, operator kurung buka, operand 8, operator pengurangan, operator kurung tutup, operator pangkat, dan operand 3.

Tabel 5.3 merupakan kelas MathARNode yang digunakan sebagai *node* untuk *linked list* MathARLinkedList. Kelas ini menyimpan dua *field* data bertipe string, yaitu data dan type. Tabel 5.5 merupakan kelas MathARLinkedList yang merupakan *Linked List* yang digunakan sistem. Kelas ini berisi konstruktor, *method* clearList untuk melakukan *reset* terhadap *list*, *method* insertFirst untuk memasukan *node* ke posisi *head*, *method* insertLast untuk memasukan *node* ke posisi *tail, method* deleteLast untuk menghapus *node* dari posisi *tail*, *method* getSize untuk mendapatkan ukuran *list*, *method* getLastNode untuk mendapatkan *node* pada posisi *tail, method* getContent yang digunakan untuk mengembalikan nilai string dari konten *linked list*, *method* getPostfixValue yang digunakan untuk mengembalikan hasil konversi notasi infix pada string content menjadi notasi postfix, *method* bool isError yang akan bernilai *true* jika terjadi error saat eksekusi operasi aritmetika dan akan bernilai *false* jika operasi dapat dieksekusi, serta *method* calculatePostfix yang digunakan untuk mengembalikan nilai perhitungan operasi aritmetika dengan menggunakan program kalkulator postfix. Tabel 5.7 merupakan kelas MainController yang digunakan untuk mendaftarkan *game object* yang pada sistem untuk dapat dimanipulasi dengan engine unity. Kelas ini juga berisi *method* untuk mereset variabel, mengurangi, menambahkan, dan mengembalikan jumlah objek yang terdeteksi saat ini.

Tabel 5.3 Kelas MathARNode

```
1   public class MathARNode {
2       public MathARNode next, prev;
3       public string data, type;
4       public MathARNode(string data, string type) {
5           this.data = data;
6           this.type = type;
7       }
8   }
```

Tabel 5.4 Pembahasan Kode MathARNode

1	Pembuka kelas MathARNode
2	Deklarasi variabel MathARNode next dan prev untuk penghubung antar *node*
3	Deklarasi variabel data dan type yang akan menyimpan tipe dan nilai *marker* yang terdeteksi.
4-7	Constructor MathARNode yang akan memberi nilai data dan type berdasarkan parameter yang dilewatkan.

Tabel 5.5 Kelas MathARLinkedList

```
1   using calculator;
2   using System;
3   using System.Collections.Generic;
4   using UnityEngine;
5   using Node = MathARNode;
6
7   public class MathARLinkedList {
8       private Node head, tail, pointer, tmp;
9       private int size = 0;
10      public string content;
11
12      public MathARLinkedList() {
13          this.head = this.tail = null;
```

```
14      }
15
16      public void clearList() {
17          this.head = this.tail = null;
18          content = "";
19          size = 0;
20      }
21
22      public void insertFirst(string data, string type) {
23          tmp = new Node(data, type);
24
25          if (size == 0) {
26              tmp.next = tmp.prev = null;
27              tail = tmp;
28          }
29          else {
30              tmp.next = head;
31              tmp.next.prev = tmp;
32          }
33          head = tmp;
34          pointer = head;
35          while (pointer.next != null) {
36              pointer = pointer.next;
37          }
38          tail = pointer;
39          size++;
40      }
41
42      public void insertLast(string data, string type) {
43          if (size == 0) {
44              insertFirst(data, type);
45          }
46          else {
47              tmp = new Node(data, type);
48              pointer = tail;
49              pointer.next = tmp;
50              tmp.prev = pointer;
51              tail = tmp;
```

```
52              size++;
53          }
54          content += data;
55      }
56
57      public void deleteLast() {
58          tail = tail.prev;
59          tail.next = null;
60          size--;
61      }
62
63      public int getSize() {
64          return size;
65      }
66
67      public Node getLastNode() {
68          if (size == 0){
69              return null;
70          }
71          else {
72              return tail;
73          }
74      }
75
76      public string getContent() {
77          if (size != 0) {
78              pointer = head;
79              string tmpContent = "";
80              while (pointer != null)
81              {
82                  tmpContent += pointer.data;
83                  pointer = pointer.next;
84              }
85
86              return tmpContent;
87          }
88          else
89              return "none";
```

```
90      }
91
92      public string getPostFixValue() {
93          Program c = new Program();
94          return c.getPostFix(this.content);
95      }
96
97      public bool isError() {
98          try {
99              calculatePostfix();
100             return false;
101         }
102         catch (Exception) {
103             return true;
104         }
105     }
106
107     public double calculatePostfix() {
108         Program c = new Program();
109         return c.Calculate(this.content);
110     }
111 }
```

Tabel 5.6 Pembahasan Kode MathARLinkedList

1-5	Perintah untuk melakukan impor *library* unity dan merefer kelas MathARNode sebagai Node dan juga melakukan include ke program kalkulator *postfix*.
7	Pembuka kelas MathARLinkedList
8	Deklarasi head, tail, pointer, dan tmp dengan tipe data Node sebagai *node* yang akan digunakan dalam kelas ini.
9	Deklarasi variabel size untuk menyimpan ukuran linked *list*
10	Deklarasi variabel content yang digunakan untuk menyimpan isi linked *list*.
12-14	Constructor linked *list* MathARLinkedList.
16-20	*Method* clearList yang digunakan untuk melakukan *reset* variabel

	dan *reset* linked *list* ke kondisi semula.
22-40	*Method* insertFirst yang akan digunakan untuk menambahkan *node* baru dari posisi kepala linked *list*.
42-55	*Method* insertLast yang akan digunakan untuk menambahkan *node* baru dari posisi ekor linked *list*.
57-61	*Method* deleteLast yang digunakan untuk menghapus *node* dari posisi ekor linked *list*.
63-65	*Method* getSize yang akan mengembalikan nilai size linked *list*
67-74	*Method* getLastNode yang akan mengembalikan *node* di posisi ekor linked *list*
76-90	*Method* getContent yang digunakan untuk mengembalikan nilai string dari konten linked *list*.
92-95	*Method* getPostfixValue yang digunakan untuk mengembalikan hasil konversi notasi *infix* pada string content menjadi notasi *postfix*.
97-105	*Method* bool isError yang akan bernilai true jika terjadi error saat eksekusi operasi aritmetika dan akan bernilai false jika operasi dapat dieksekusi.
107-111	*Method* calculate Postfix yang digunakan untuk mengembalikan nilai perhitungan operasi aritmetika dengan menggunakan program kalkulator *postfix*.

Tabel 5.7 Kelas MainController

```
1   using UnityEngine;
2   using UnityEngine.UI;
3   using List = MathARLinkedList;
4
5   public class MainController : MonoBehaviour {
6       public Text uitext, txtLL, txtCount, txtNotif,
7       txtPostFix;
8       private int detectedObj;
9       public int detectedOperator = 0, detectedOperand = 0;
10      public List list = new List();
```

```
11
12      private void Update() {
13          txtLL.text = "List =
14  "+list.getContent().ToString();
15          txtCount.text = "Object counts :
16  "+getDetectedObjCount().ToString();
17      }
18
19      public void resetVariables() {
20          txtNotif.text = txtPostFix.text = "";
21          detectedOperator = detectedOperand = 0;
22      }
23
24      public void addDetectedObjCount() {
25          detectedObj++;
26      }
27
28      public void substractDetectedObjCount() {
29          detectedObj--;
30      }
31
32      public int getDetectedObjCount() {
33          return this.detectedObj;
34      }
35  }
```

Tabel 5.8 Pembahasan Kode MainController

1-3	Melakukan impor *library* untuk unity dan merefer MathARLinkedList sebagai List.
5	Pembuka kelas MainController
6	Deklarasi variabel bertipe Text untuk uitext yang menampilkan nilai dan tipe objek yang terdeteksi, txtLL yang menampilkan konten linked *list*, txtCount yang menampilkan jumlah objek yang terdeteksi saat program berjalan, txtNotif yang menampilkan notifikasi, serta txtPostFix yang menampilkan teks konversi *infix*

	ke *postfix*.
8	Deklarasi variabel detectedObj untuk menyimpan nilai jumlah objek yang terdeteksi saat program berjalan
9	Deklarasi variabel detectedOperator dan detectedOperand untuk menyimpan nilai jumlah operator dan operand yang berada di linked *list* saat program berjalan
10	Instansiasi objek *list* bertipe List untuk membuat linked *list* baru
12-17	*Method* update yang akan berjalan setiap frame. *Method* ini akan melakukan pembaharuan konten untuk txtLL dan txtCount setiap framenya.
19-22	*Method reset*Variables yang digunakan untuk melakukan *reset* jumlah operator yang berada dalam linked *list*.
24-26	*Method* addDetectedObjCount yang digunakan untuk menambahkan nilai ke variabel detectedObj saat ada objek yang terdeteksi kamera AR.
28-30	*Method* substractDetectedObjCount yang digunakan untuk mengurangi nilai ke variabel detectedObj saat ada objek yang hilang dari deteksi kamera AR.
32-34	*Method* getDetectedObjCount yang digunakan untuk mengembalikan nilai variabel detectedObj.

5.4 Implementasi Kalkulasi Aritmetika

Gambar 5.3 Membaca *marker equals*

Perhitungan akan dilakukan apabila kamera perangkat menangkap *marker* bertipe *equals* yang dapat dilihat pada Tabel 5.1 baris 50. Pada Gambar 5.3, *marker equals* diarahkan ke kamera untuk memulai perhitungan dari 2*(8-5)^3. Hasil yang ditampilkan adalah 54 dengan nilai *postfix* 285-3^*. Hasil perhitungan dan nilai *postfix* yang ditampilkan sudah diperiksa dengan menggunakan program kalkulator biasa dan program perubah nilai *infix* ke *postfix* secara online.

BAB 6 PENGUJIAN

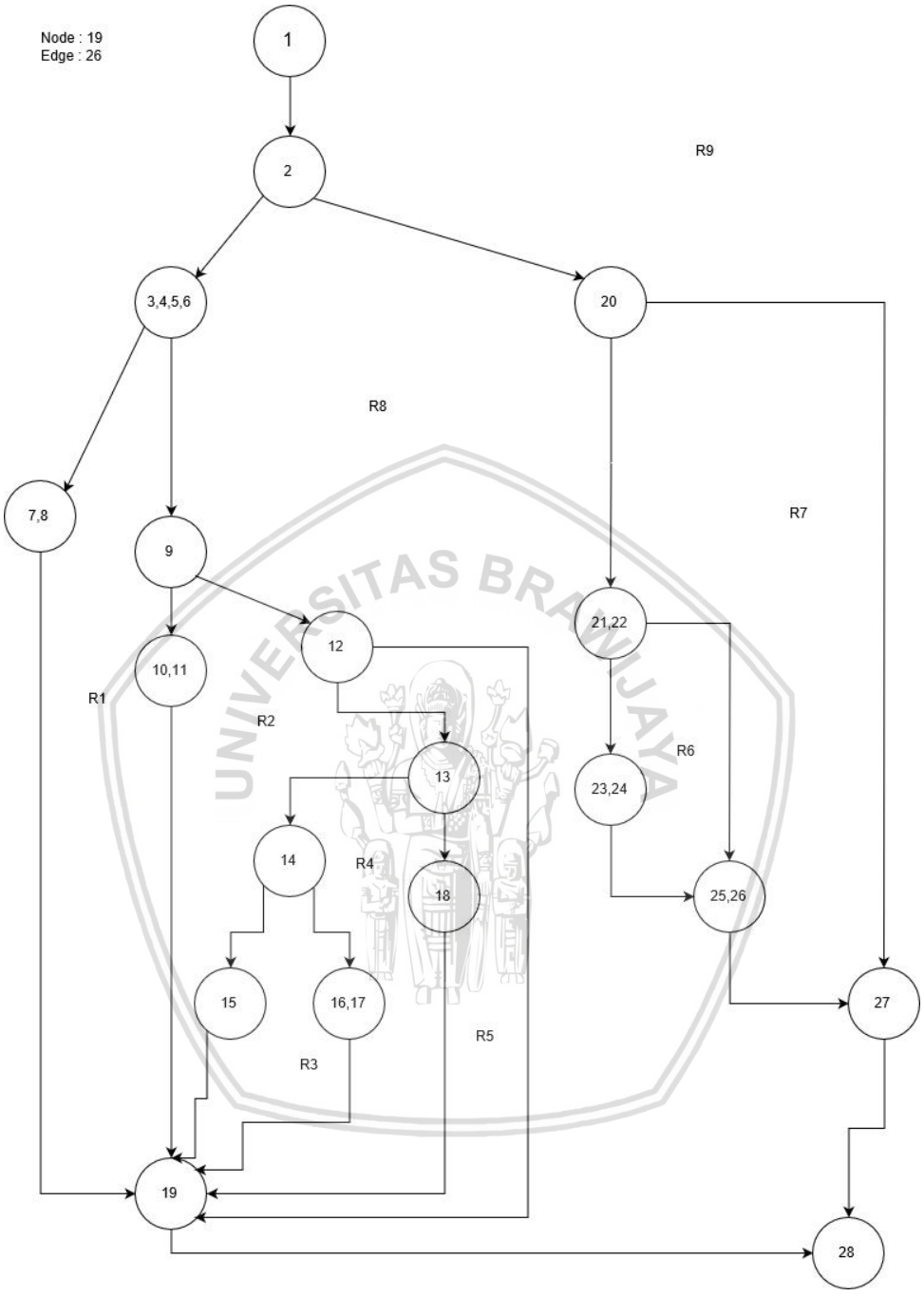
6.1 Basis Path Testing

Pengujian ini merupakan pengujian Whitebox dilakukan untuk mengetahui nilai *Cyclomatic Complexity* dari *method* perubahan state OntrackableStateChanged pada class MathARTrackableEventHandler yang digunakan sebagai dasar dari proses pembacaan dan pemrosesan *marker*. Terdapat beberapa istilah dalam pengujian ini, antara lain *edge* yang merupakan penghubung antar *node*, *region* yang merupakan daerah yang dibatasi oleh *node* dan *edge*, serta *predicate node* yang merupakan *node* yang mengeluarkan dua atau lebih *edge*. Tabel 6.1 merupakan *method* OnTrackableStateChanged yang akan dikonversi menjadi Flow Graph pada Gambar 6.1.

Tabel 6.1 Method OnTrackableStateChanged

1	1	public void OnTrackableStateChanged(
	2	TrackableBehaviour.Status previousStatus,
	3	TrackableBehaviour.Status newStatus) {
2	4	if (newStatus ==
	5	TrackableBehaviour.Status.DETECTED ||
	6	newStatus ==
	7	TrackableBehaviour.Status.TRACKED ||
	8	newStatus ==
	9	TrackableBehaviour.Status.EXTENDED_TRACKED) {
3	10	Debug.Log("Trackable " +
	11	mTrackableBehaviour.TrackableName + " found");
4	12	OnTrackingFound();
5	13	main.uitext.text = cardType +" "+
	14	cardValue;
6	15	if (cardType == "operand") {
7	16	main.list.insertLast(cardValue,
	17	cardType);
8	18	main.detectedOperand++;
	19	}
9	20	else if (cardType == "operator") {
10	21	main.list.insertLast(cardValue,
	22	cardType);
11	23	main.detectedOperator++;
	24	}
12	25	else if (cardType == "*equals*") {

```
13  26                  if (main.list.getSize() != 0) {
14  27                      if (main.list.isError()) {
15  28                          main.txtNotif.text = "Unable
    29  to operate";
    30                      }
    31                      else {
16  32                          main.txtPostFix.text =
    33  "Postfix = " + main.list.getPostFixValue();
17  34                          main.txtNotif.text = "Result :
    35  " + main.list.calculatePostfix().ToString();
    36                      }
    37                  }
    38                  else {
18  39                      main.txtNotif.text = "Zero list
    40  detected";
    41                  }
    42              }
19  43              main.addDetectedObjCount();
    44          }
20  45          else if (previousStatus ==
    46  TrackableBehaviour.Status.TRACKED &&
    47              newStatus ==
    48  TrackableBehaviour.Status.NOT_FOUND) {
21  49              Debug.Log("Trackable " +
    50  mTrackableBehaviour.TrackableName + " lost");
22  51              if (cardType == "equals") {
23  52                  main.list.clearList();
24  53                  main.resetVariables();
    54              }
25  55              main.txtNotif.text = "";
26  56              main.substractDetectedObjCount();
27  57              OnTrackingLost();
    58          }
    59          else {
27  60              OnTrackingLost();
    61          }
28  62      }
```



Gambar 6.1 *Flow Graph Method* OnTrackableStateChanged

Pada *Flow Graph* Gambar 6.1, terdapat 9 jalur independen antara lain :

1. Jalur 1 : 1-2-3-4-5-6-7-8-19-28
2. Jalur 2 : 1-2-3-4-5-6-9-10-11-19-28
3. Jalur 3 : 1-2-3-4-5-6-9-12-13-14-15-19-28
4. Jalur 4 : 1-2-3-4-5-6-9-12-13-14-16-17-19-28
5. Jalur 5 : 1-2-3-4-5-6-9-12-13-18-19-28

6. Jalur 6 : 1-2-3-4-5-6-9-12-19-28
7. Jalur 7 : 1-2-20-21-22-23-24-25-26-27-28
8. Jalur 8 : 1-2-20-21-22-25-26-27-28
9. Jalur 9 : 1-2-20-27-28

Perhitungan matematis *Cyclomatic Complexity* - V(G), sebagai berikut :

1. V(G) = jumlah *region*
2. V(G) = E - N + 2
3. V(G) = P + 1

Contoh perhitungan matematis *Cyclomatic Complexity* - V(G) berdasarkan *Flow Graph* di atas, sebagai berikut :

1. V(G) = 9, karena terdapat 9 *region* (R1, R2, R3, R4, R5, R6, R7, R8, R9).
2. V(G) = 26 - 19 + 2 = 9, karena terdapat 26 *edges* dan 19 *node.*
3. V(G) = 8 + 1 = 9, karena terdapat 8 *predicate node*.

Dari perhitungan matematis *Cyclomatic Complexity* - V(G) berdasarkan *Flow Graph* di atas dapat disimpulkan bahwa nilai V(G) pada *Flow Graph* di atas adalah 9, yang berarti resiko kompleksitasnya tergolong rendah dan dapat dilakukan pengujian masing-masing jalur independen. Pengujian masing-masing jalur independen terdapat pada Tabel 6.2.

Tabel 6.2 Tabel Kasus Uji

No. Jalur	Node yang dilewati	Hasil yang Diharapkan	Hasil yang didapatkan	Status Valid
1	1-2-3-4-5-6-7-8-19-28	Model 3D dengan *marker* bertipe operand berhasil ditampilkan dan dimasukan ke dalam *linked list*.	Model 3D dengan *marker* bertipe operand berhasil ditampilkan dan dimasukan ke dalam *linked list*	Valid
2	1-2-3-4-5-6-9-10-11-19-28	Model 3D dengan *marker* bertipe operator berhasil ditampilkan dan dimasukan ke dalam *linked list*.	Model 3D dengan *marker* bertipe operator berhasil ditampilkan dan dimasukan ke dalam *linked list*	Valid
3	1-2-3-4-5-6-9-12-13-14-15-19-28	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi gagal melakukan operasi karena nilai *method*	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi gagal melakukan operasi karena nilai *method*	Valid

		isError adalah *true*.	isError adalah *true*.	
4	1-2-3-4-5-6-9-12-13-14-16-17-19-28	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi berupa hasil perhitungan.	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi berupa hasil perhitungan.	Valid
5	1-2-3-4-5-6-9-12-13-18-19-28	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi berupa bahwa list kosong.	Model 3D dengan *marker* bertipe equals berhasil ditampilkan, muncul notifikasi berupa bahwa list kosong	Valid
6	1-2-3-4-5-6-9-12-19-28	Model 3D yang tipe *marker*nya tidak diketahui dimunculkan di layar.	Model 3D yang tipe *marker*nya tidak diketahui dimunculkan di layar.	Valid
7	1-2-20-21-22-23-24-25-26-27-28	Model 3D dengan *marker* bertipe *equals* berhenti ditampilkan, sistem melakukan *reset* nilai variabel dan isi *linked list.*	Model 3D dengan *marker* bertipe *equals* berhenti ditampilkan, sistem melakukan *reset* nilai variabel dan isi *linked list.*	Valid
8	1-2-20-21-22-25-26-27-28	Model 3D dengan *marker* bertipe selain equals berhenti ditampilkan.	Model 3D dengan *marker* bertipe selain equals berhenti ditampilkan.	Valid
9	1-2-20-27-28	Sistem menghilangkan model 3D saat tidak ada perpindahan *state*.	Sistem menghilangkan model 3D saat tidak ada perpindahan *state*.	Valid

6.2 Pengujian *Usability*

Pengujian *usability* digunakan untuk melihat bagaimana kemudahan penggunaan aplikasi oleh pengguna. Pengujian dijalankan dengan menggunakan kuisioner yang terdiri dari 10 pertanyaan serta pilihan jawaban yang terdiri dari Sangat Tidak Setuju (STS), Tidak Setuju (TS), Netral (N), Setuju (S), dan Sangat Setuju (SS) di setiap pertanyaannya. Penyusunan pertanyaan dikondisikan sesuai dengan format *System Usability Scale* (SUS).

Pada pengujian ini, responden ditentukan sebanyak 6 orang dengan klasifikasi sebagai berikut :

1. Tidak asing dengan menggunakan ponsel Android.
2. Tidak asing dengan operasi kalkulator konvensional.
3. Tidak ikut andil dalam proses pengembangan sistem yang dibuat.
4. Tidak mengerti secara pasti cara kerja sistem yang dibuat.

Dengan klasifikasi di atas, responden diharapkan mampu mengisi kuesioner yang diberikan dengan tepat dan sesuai dengan yang dirasakannya saat pengujian berlangsung. Berikut adalah soal-soal yang diberikan saat pengujian :

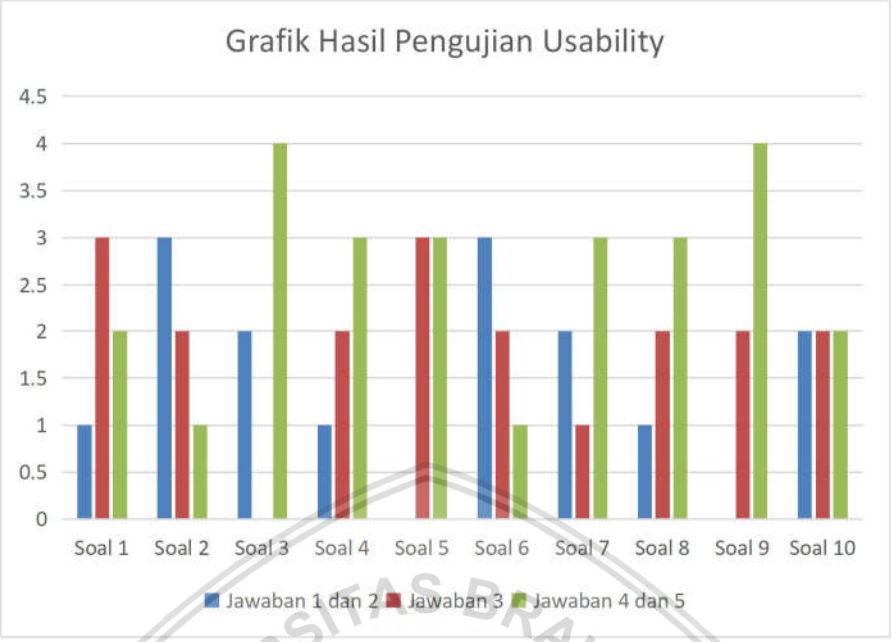
1. Saya rasa saya akan sering menggunakan aplikasi ini.
2. Saya menemukan bahwa aplikasi ini rumit.
3. Saya merasa aplikasi ini mudah untuk digunakan.
4. Saya merasa perlu pendampingan dari orang yang paham untuk dapat menggunakan sistem ini.
5. Saya merasa berbagai fungsi dari sistem ini terintegrasi dengan baik.
6. Saya pikir sistem ini tidak memiliki konsistensi.
7. Saya membayangkan orang-orang akan sangat mudah menggunakan sistem ini.
8. Saya merasa beberapa orang akan kesusahan menggunakan aplikasi ini.
9. Saya cukup percaya diri ketika menggunakan aplikasi ini.
10. Saya perlu belajar banyak hal sebelum menggunakan aplikasi ini.

6.2.1 Analisis Hasil Pengujian Usability

Tabel 6.3 dan Gambar 6.2 merupakan hasil pengujian yang sudah dilakukan kepada responden :

Tabel 6.3 Tabel hasil pengujian *usability*

Nomor Soal	Jumlah berdasarkan Nilai				
	1	**2**	**3**	**4**	**5**
1	1		3	2	
2	1	2	2	1	
3		2		3	1
4	1		2	2	1
5			3	3	
6		3	2	1	
7		2	1	3	
8		1	2	3	
9			2	3	1
10	1	1	2		2



Gambar 6.2 Grafik Hasil Pengujian *Usability*

Poin pertama mayoritas bernilai 3, berarti pengguna masih ragu-ragu apakah akan sering menggunakan aplikasi ini atau tidak. Pada poin kedua, mayoritas responden memilih nilai 2 dan 3, berarti pengguna cenderung menilai aplikasi yang dikembangkan tidak rumit. Pada poin ketiga, mayoritas responden memilih nilai 4, berarti responden menilai aplikasi ini mudah untuk digunakan. Pada poin keempat, mayoritas responden memberi nilai 3 dan 4, yang berarti responden cenderung merasa perlu pendampingan untuk menggunakan aplikasi ini. Pada poin kelima, mayoritas responden memberi nilai 3 dan 4, berarti responden menilai bahwa fungsi dari sistem ini terintegrasi dengan cukup baik. Pada poin keenam, mayoritas responden memberi nilai 2, yang berarti responden merasa sistem memiliki konsistensi yang cukup baik. Pada poin ketujuh, mayoritas responden memilih nilai 4, berarti responden merasa aplikasi ini cukup mudah untuk digunakan orang-orang. Pada poin kedelapan, mayoritas responden memilih nilai 4, yang berarti beberapa orang juga akan kesusahan menggunakan aplikasi ini. Pada poin kesembilan, mayoritas responden memberi nilai 4, berarti responden cenderung paham apa yang dia lakukan selama menggunakan aplikasi ini. Pada poin yang terakhir, mayoritas responden memilih poin 3 dan 5, yang berarti responden masih perlu belajar banyak hal sebelum menggunakan aplikasi ini.

Nilai yang didapatkan pada soal bernomor ganjil adalah semakin besar semakin baik, sedangkan pada nomor soal genap semakin besar semakin buruk. Dari hasil pengujian *usability* yang dilakukan, didapatkan nilai rata-rata SUS sebesar 55 yang berarti nilai *usability* dari sistem yang dikembangkan masih tergolong “Not Acceptable” atau masih belum layak untuk digunakan masyarakat awam.

BAB 7 KESIMPULAN

7.1 Kesimpulan

Berdasarkan hasil pengujian *whitebox* dan pengujian *usability* yang dilakukan, dapat diambil beberapa kesimpulan antara lain :

1. Pembacaan *marker* dilakukan dengan menggunakan bantuan Vuforia SDK dan *game engine* Unity 3D. Sistem yang dikembangkan telah memenuhi kebutuhan sesuai dengan rancangan dengan tingkat kompleksitas yang rendah.
2. Algoritma untuk memodifikasi konten *Linked List* terdapat pada method OnTrackableStateChanged pada Vuforia SDK. *Linked List* pada sistem bekerja dengan baik sesuai dengan fungsinya untuk menyimpan urutan masukan operand dan operator aritmetika.
3. Berdasarkan pengujian *usability* yang dilakukan, nilai rata-rata SUS yang didapatkan adalah 55. Nilai ini berarti sistem masih tergolong belum layak digunakan kepada masyarakat awam.

7.2 Saran

Berikut merupakan beberapa saran terkait hasil penelitian yang sudah dilakukan untuk kemudian diperbaiki atau dikembangkan pada penelitian berikutnya :

1. Menambahkan variasi operator ke dalam sistem.
2. Memproses perhitungan dengan cara langsung, dengan menampilkan semua *marker* ke layar lalu melakukan eksekusi perhitungan.
3. Memperbaiki tingkat terbacanya *marker* dengan menggunakan pola yang tidak serupa tiap *marker*nya, karena pada kasus tertentu *marker* yang terbaca oleh sistem tidak sesuai dengan yang diarahkan pengguna.
4. Menambahkan unsur permainan dalam sistem, sehingga dapat menarik untuk digunakan.
5. Memperbaiki sistem agar model yang ditampilkan tidak berpindah-pindah.
6. Menambahkan fitur *undo* apabila pengguna membaca *marker* yang salah. Baik berupa *marker* atau berupa tombol.
7. Menggunakan model 3d sebagai tampilan hasil operasi.
8. Menguji dan membandingkan performa sistem apabila menggunakan *array* atau *arraylist* dalam penyimpanan urutan datanya.
9. Menggunakan *markerless AR* untuk membandingkan dengan penelitian ini yang menggunakan *marker-based AR.*

DAFTAR PUSTAKA

Azuma, R.T., 1997. A survey of augmented reality. Presence: Teleoperators and virtual environments, 6(4), pp.355-385.

Bangor, A., Kortum, P. and Miller, J., 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies, 4(3),* pp.114-123.

Brooke, J., 1996. SUS-A quick and dirty usability scale. *Usability* evaluation in industry, 189(194), pp.4-7.

Csegeek, 2013. Types of *Linked List* [online] <http://www.csegeek.com/ csegeek/ view/ tutorials/ algorithms/linked_list/list_intro.php> (Diakses 8 Oktober 2017)

Dmytro, 2013. Creating a Very Simple Linked List [online] <https://stackoverflow.com/questions/3823848/creating-a-very-simple-linked-list> (Diakses 6 September 2017)

Eric, 2012. Postfix Calculator in C# [online] <https://randcode.wordpress.com/2012/10/03/*postfix*-calculator-in-c/> (Diakses 8 Desember 2017)

Milgram, P. dan Kishino, F., 1994. A taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems, 77(12), pp.1321-1329.

Nielsen, J., 2000. Why You Only Need to Test with 5 Users [online] <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> (Diakses 13 Desember 2017)

Parlante, N., 2001. Linked *list* basics. Stanford CS Education Library, 1, p.25.

Petersen, N. dan Stricker, D., 2015. Cognitive augmented reality. Computers & *Graph*ics, 53, pp.82-91.

Prabhu K, Gautham, 2014. Software Metrics [online] <https://randcode.wordpress.com/2012/10/03/*postfix*-calculator-in-c/> (Diakses 27 Desember 2017)

Pressman, R.S., 2005. Software engineering: a practitioner's approach. Palgrave Macmillan.

Raja, V. dan Calvo, P., 2017. Augmented reality: An ecological blend. Cognitive Systems Research, 42, pp.58-72.

Ryan, 2008. Array versus linked-list [online] <https://stackoverflow.com/questions/166884/array-versus-linked-list> (Diakses 10 Januari 2018)

Tapson, F., 2006. The Oxford mathematics study dictionary. Oxford University Press.

Vuforia, 2017. Vuforia Supported Versions [online] <https://*library*.vuforia.com/articles/Solution/Vuforia-Supported-Versions> (Diakses 6 September 2017)

Xiao, C. and Lifeng, Z., 2014, June. Implementation of mobile augmented reality based on Vuforia and Rawajali. In Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on (pp. 912-915). IEEE.