BAB 1 PENDAHULUAN

1.1 Latar Belakang

Pada saat ini game bukan hanya sarana untuk bermain. Melainkan juga digunakan untuk simulasi. Simulasi peperangan militer dan game tembak-menembak sangat populer pada saat ini, baik karena tampilan visual yang bagus, ataupun karena kecerdasan buatan tinggi (Wei, Xiao-fang, & Shuai, 2010). Simulasi sangat berguna untuk membantu dalam pelatihan peperangan. Selain itu juga simulasi dalam game atau dunia *virtual* dapat menghemat biaya berlebih, dan juga efisien. Oleh karena itu perlu adanya pengembangan game simulasi.

Menciptakan tingkah laku manusia ke dalam Simulasi peperangan masih menjadi tantangan tersendiri untuk saat ini. Dimana ini melibatkan berbagai macam ruang lingkup penelitian dalam merancang simulasi (Shen, & Zhou, 2006). Dalam game peperangan, sekelompok prajuruit pasti memiliki banyak tingkah laku dalam sebuah permainan. Contohnya dalam pergerakan secara berkelompok, sebuah kawanan kelompok harus bergerak secara terkoordinasi atau membentuk sebuah formasi dalam kondisi tertentu. Menurut Marcel, Bakkes, dan Spronck(2008) menjelaskan bahwa formasi dalam sebuah game terdapat 2 model yang khas yaitu, formasi yang sudah ditetapkan (*fixed formation*) dan formasi yang muncul karena interaksi antara setiap unit kelompok (*emergent formation*). Tentunya itu akan menjadi studi kasus yang harus diselesaikan dalam perancangan sebuah game.

Pada umumnya dalam pengembangan game membutuhkan AI (*Artificial Intelligence)* untuk implementasi tingkah laku pada NPC (*Non Player Character*) agar NPC akan nampak terlihat cerdas dan tercipta suasana yang lebih realistis dalam game. Game AI juga harus berimbang dalam pembuatannya. Tidak perlu terlalu sulit untuk dimainkan, ataupun terlalu mudah dimainkan. Karena tujuan dari pembuatan game sendiri adalah supaya menyenangkan ketika dimainkan (Kyaw, Peters, & Naing Swe, 2013). Oleh karena itu untuk menciptakan pergerakan prajurit secara berkelompok atau formasi dalam kawanan, game membutuhkan AI (*Artificial Intelligence)* untuk implementasi pada NPC.

Steering Behavior adalah metode yang sering digunakan dalam pergerakan atau movement untuk NPC (*Non Player Character*). Dimana perilaku pergerakan metode ini menjelaskan suatu kawanan dapat bergerak tanpa diberi perintah atau input dari pemain (Reynolds, 1999b). Metode *Steering Behavior* yang digunakan dalam skripsi ini adalah : pergerakan menuju target tujuan (*Seek),* pergerakan menjauhi target atau gerakan menghindar *(Flee),* pergerakan menuju target tujuan dengan batasan-batasan speed radius target *(Arrival).* Untuk menunjang dalam pergerakan NPC (*Non Player Character*) tentunya dibutuhkan path atau lintasan dalam melakukan pergerakan. Oleh karena itu Game AI *(Artificial*

intelligence) juga membutuhkan metode *Pathfinding* sebagai pencarian jarak atau jalur terdekat dalam lintasan. Pada skripsi ini, rekan kelompok project Game *First Person Military Simulation.* Telah membantu dalam pembuatan pathfinding dalam game dengan menggunakan Algoritma HPA* *(Hierarchial Pathfinding A*).* Dimana pathfinding ini nantinya digunakan dalam navigasi NPC (*Non Player Character*) untuk melakukan pergerakan menuju tujuannya.

Tidak cukup hanya menggunakan *Steering Behavior* dan *Pathfinding* saja, dimana pergerakan prajurit yang berkelompok belum bisa terkoordinasi dengan baik. Algoritma *Flocking* adalah algoritma yang umum digunakan untuk menyelesaikan permasalahan dalam perilaku pergerakan secara berkelompok. *Flocking* merupakan penerapan perilaku agent atau NPC (*Non Player Character*) yang menyerupai pergerakan kawanan burung yang terbang, sekelompok ikan yang berenang, ataupun koloni dari bakteri, dalam mempertahankan kehidupannya (Zongyao, & Dongbing, 2007). Algoritma ini berperan supaya suatu kawanan dapat bergerak secara terkoordinasi antara NPC (*Non Player Character*) satu dengan yang lainnya (Fathy, 2014). *Flocking* adalah algoritma yang mudah, sederhana untuk dipahami. Karena hanya menggunakan perhitungan matematika sederhana.

Oleh karena itu, dalam skripsi ini dilakukan implementasi *Squad Formation Behavior* untuk NPC dalam game *First Person Military Simulation* dengan menggunakan Algoritma *Flocking*, sehingga perilaku pergerakan berkelompok kawanan prajurit dalam game *First Person Military Simulation* dapat bergerak secara terkoordinasi antara NPC (*Non Player Character*) satu dengan yang lainnya. Untuk hasil dari implementasi skripsi ini nantinya akan digabungkan dalam satu project game *First Person Military Simulation,* dimana dalam satu project terdapat rekan yang bertugas untuk pembuatan desain game, animasi beserta tampilan visual yang digunakan untuk game *First Person Military Simulation.*

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang, maka dapat dirumuskan permasalahan pada penelitian ini yaitu :

1. Bagaimana melakukan perancangan algoritma *Flocking* pada NPC di *First Person Military Simulation Game*?
2. Bagaimana mengimplementasikan algoritma *Flocking* pada NPC di *First Person Military Simulation Game*?
3. Bagaimana mengetahui kinerja algoritma *Flocking* pada NPC di *First Person Military Simulation Game*?

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan lebih terfokus, maka penelitian ini dibatasi oleh hal-hal sebagai berikut :

1. Lingkungan implementasi system bukan pada tempat yang ramai dengan NPC (*non crowded environment*).
2. Implementasi algoritma *flocking* ini akan diintegrasikan dengan HPA* (*Hierarchial Pathfinding A**).
3. Dalam Implementasi Algoritma Flocking, NPC pada game dibatasi sebanyak 4 NPC dalam satu grup. Karena penyesuaian terhadap game design.

1.4 Tujuan

Tujuan yang ingin dicapai dalam implementasi skripsi ini adalah untuk menyelesaikan masalah *Squad Formation Behavior* pada game *First Person Military Simulation* menggunakan Algoritma *Flocking*.

1.5 Manfaat

Manfaat yang diharapkan dari implementasi skripsi ini adalah :

1. Agar perilaku pergerakan NPC pada game *First Person Military Simulation* menjadi bagus dan terkoordinasi.
2. Dapat dijadikan sebagai pembanding atau literatur dalam penelitian yang menerapkan algoritma *Flocking* dimasa yang akan datang.

1.6 Sistematika Penulisan

Sistematika isi dan penulisan dalam skripsi ini antara lain:

BAB I : Pendahuluan

Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan dan manfaat dari penelitian serta sistematika penulisan.

BAB II : Dasar Teori

Menguraikan tentang dasar teori dan teori penunjang yang berkaitan dengan Algoritma *Flocking.*

BAB III : Metodologi Penelitian dan Perancangan

Berisi tentang gambaran umum langkah-langkah dalam perancangan dan implementasi Algoritma *Flocking* dimulai

dari tahap perancangan, implementasi serta pengujian secara umum.

BAB IV : Hasil

Memuat hasil dari implementasi Squad Formation Behavior, serta melakukan pengujian terhadap Algoritma *Flocking*.

BAB V : Pembahasan

Memuat pembahasan hasil terhadap Algoritma *Flocking* yang telah diimplementasikan. Berupa analisa dari pengujian yang dilakukan

BAB VI Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian real-time dari implementasi program, serta saran-saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Implemnetasi Algoritma Flocking dalam *First Person Military Simulation Game.*

2. 1 Kecerdasan Buatan Pada Game

Kecerdasan buatan adalah membuat komputer sanggup untuk melaksanakan tugas berfikir seperti apa yang dilakukan oleh hewan dan manusia (Milington, & Funge, 2009). Begitu juga pada industri game yang menggunakan kecerdasan buatan untuk mendukung pengembangan game. Game AI juga harus berimbang dalam pembuatannya. Tidak perlu terlalu sulit untuk dimainkan, ataupun terlalu mudah dimainkan. Karena tujuan dari pembuatan game sendiri adalah supaya menyenangkan ketika dimainkan (Kyaw, Peters, & Naing Swe, 2013). Kecerdasan buatan pada kebenyakan *modern game* memiliki 3 kebutuhan utama, yaitu : pergerakan (*movement*), pengambil keputusan (*decision making*), dan strategi (*strategy*). Tiga kebutuhan utama ini sering digunakan di game manapun (Milington, & Funge, 2009).

2.2 Non Player Character

Non Player Character atau yang biasa disebut NPC merupakan sebuah karakter dalam permainan game atau simulasi yang perilakunya tidak dapat dikendalikan oleh player/ user. Contoh yang paling banyak ditemui dalam sebuah permainan game adalah musuh yang dihadapi ketika bermain game. Keberadaan *Non Player Character* yang menarik merupakan salah satu faktor yang menyebabkan user memainkan game tersebut terus-menerus (Julius, 2011). Banyak teknik yang bisa digunakan untuk merancang atau membuat NPC menjadi berperilaku realistis. Contohnya bisa menggunakan *Finite State Machine(FSM), Pathfinding, Steering Behavior,* dan masih banyak lagi lainnya. Penerapan metode-metode tersebut sudah sering digunakan dalam perkembangan game *modern* saat ini.

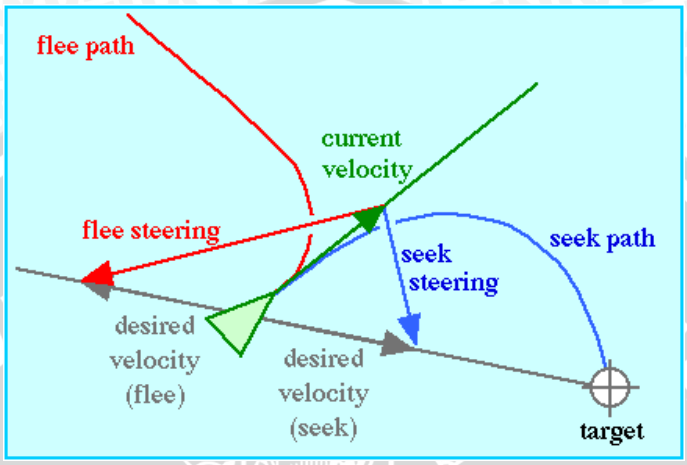
2.3 Steering Behavior

Perilaku dalam pergerakan pasukan atau sekelompok NPC akan menggunakan *Steering Behavior* untuk bergerak dalam keadaan tertentu. Dimana perilaku pergerakan metode ini menjelaskan suatu kawanan dapat bergerak tanpa diberi perintah atau input dari pemain (Reynolds, 1999b). Berikut metode *Steering Behavior* yang digunakan pada skripsi ini :

2.3.1 Seek dan Flee

Seek adalah sebuah aksi dari NPC untuk bergerak menuju posisi tertentu yang telah ditetapkan. Sedangkan *Flee* merupakan sebuah aksi dari NPC

untuk bergerak menjauhi posisi yang telah ditetapkan. *Seek* dan *Flee* merupakan perilaku paling dasar dari *steering behavior* (Wijanarko, 2010). Perbedaan *Steering Vector* antara *desired velocity (seek & flee)* dan *character current velocity* dapat dilihat pada gambar 2.1. Dimana current velocity adalah velocity dari NPC, sedangkan garis merah (flee) dan biru (seek) menunjukkan perubahan vektor ketika terjadi flee atau seek.

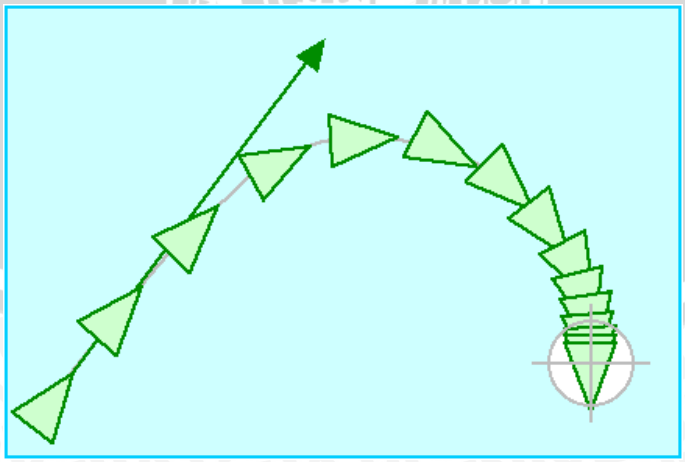


Gambar 2.1 Seek dan Flee

Sumber : Reynolds (1999b)

2.3.2 Arrival

Arrival behavior identik dengan seek behavior ketika target jauh dari karakter. *Arrival* bergerak menuju target dengan kecepatan penuh, tetapi ketika berada dekat dengan target karakter akan menurunkan kecepatannya. dan ketika dalam radius paling dekat karakter akan berhenti (Reynolds, 1999b). Penggambarannya dapat dilihat pada gambar 2.2.



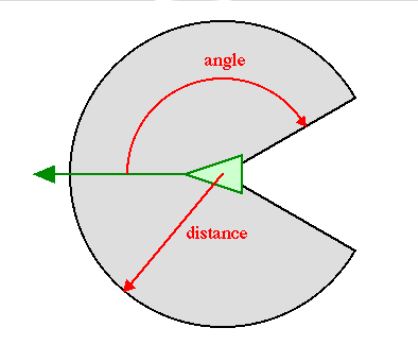
Gambar 2.3 Arrival

Sumber : Reynolds (1999b)

2.4 Flocking Behavior

Algoritma *Flocking* adalah algoritma yang umum digunakan untuk menyelesaikan permasalahan dalam perilaku pergerakan secara berkelompok. *Flocking* merupakan perilaku agent atau NPC yang menyerupai pergerakan kawanan burung yang terbang, sekelompok ikan yang berenang, ataupun koloni dari bakteri, dalam mempertahankan kehidupannya (Zongyao, & Dongbing, 2007). Algoritma ini berperan supaya suatu kawanan dapat bergerak secara terkordinasi antara NPC (*Non Player Character*) satu dengan yang lainnya.

Untuk mencapai atau menerapkan *flocking*, karakter harus bisa mendeteksi karakter lain yang berada di sekitar mereka (*neighborhood*). Karakter yang berada diluar *neighborhood* akan diabaikan. *Neighborhood* memberikan spesifikasi jarak antar karakter satu dengan yang lain, sehingga bisa didefinisikan karakter terdekat dalam satu lingkungan (Reynolds, 1999b). Skema *Neighborhood radius* yang diterapkan pada setiap NPC dapat dilihat pada gambar 2.4.

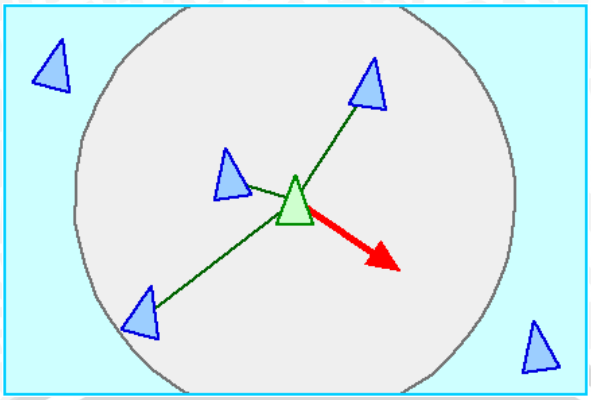


Gambar 2.4 Neighborhood

Sumber : Reynolds (1999b)

Berdasarkan dari makalah Reynolds (Reynolds, 1987a). Flocking berdasar atas 3 aturan/model, yaitu:

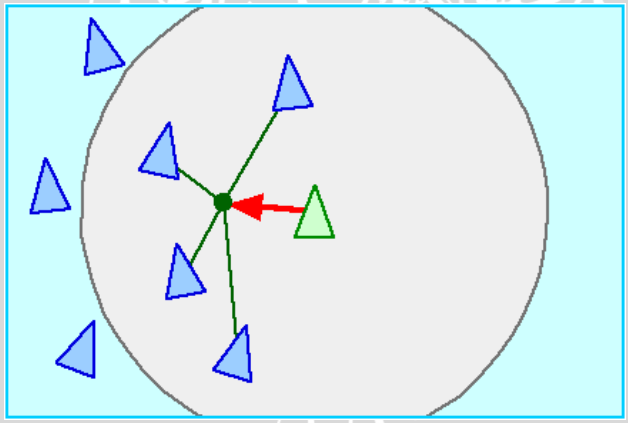
1. ***Separation*** **:** Setiap boid akan bergerak menjauhi boid lainnya. Aturan ini merupakan lawan dari perilaku *cohesion,* dan dapat dianggap seperti perilaku *collision avoidance* dalam *flocking* (Reynolds, 1999b). Pertama tama karakter akan mendeteksi seluruh karakter yang berada dalam lingkungan. Kemudian bila ada karakter yang terdeteksi dalam lingkungan, karakter akan bergerak menghindar (mengatur jarak) pada karakter lain dalam satu lingkungan. Untuk menghindar karakter dapat menggunakan *flee behavior* atau kebalikan dari *seek behavior.* Untuk penggambaran Algoritma / lebih jelasnya dapat dilihat pada gambar 2.5.



Gambar 2.5 Separation

Sumber : Reynolds (1999b)

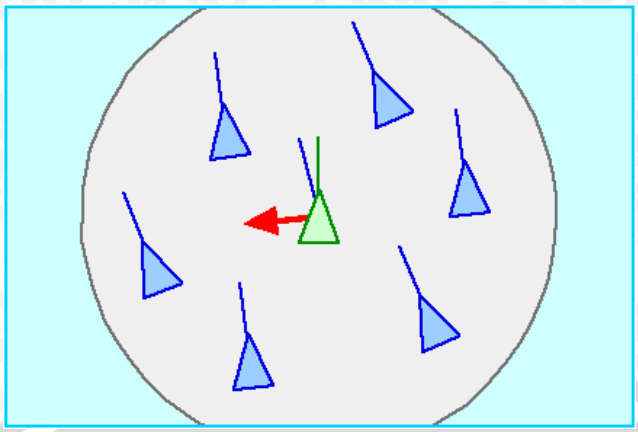
2. ***Cohesion***: *Boid* bergerak menuju pusat kerumunan untuk bergabung dengan *boid* lainnya (Reynolds, 1999b). Pertama tama karakter akan mendeteksi seluruh karakter yang berada dalam lingkungan. Kemudian melakukan rata-rata posisi (*center of gravity*) dari karakter terdekat. Untuk menuju titik massa, karakter dapat menggunakan *seek behavior.* Untuk penggambaran Algoritma / lebih jelasnya dapat dilihat pada gambar 2.6.



Gambar 2.6 Cohesion

Sumber : Reynolds (1999b)

3. ***Alignment***: Menerapkan arah yang sama pada masing-masing boid (Reynolds, 1999b). Sama halnya dengan yang lain, Pertama tama karakter akan mendeteksi seluruh karakter yang berada dalam lingkungan. Kemudian karakter melakukan rata-rata direksi/arah dari karakter terdekat, hasil rata-rata arah digunakan karakter sebagai arah pergerakannya. Untuk penggambaran Algoritma / lebih jelasnya dapat dilihat pada gambar 2.7.



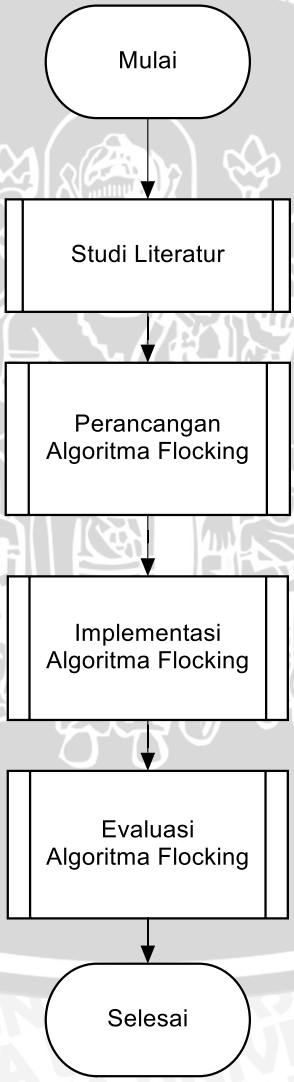
Gambar 2.7 Alignment

Sumber : Reynolds (1999b)

Flocking behavior sendiri adalah penambahan atau penggabungan dari separation, cohesion, dan alignment behavior. Dimana penggabungan tersebut akan menciptakan *boids model of flocks, herds and schools* (Reynolds, 1987a).

BAB 3 METODOLOGI

Bab ini menjelaskan tentang langkah-langkah yang ditempuh dalam penelitian yang diusulkan. Metodologi yang dilakukan dalam penelitian ini dilakukan melalui beberapa tahapan, yakni studi literatur, perancangan Algoritma *Flocking* pada *First Person Military Simulation Game*, implementasi, dan evaluasi. Tujuan implementasi Algoritma *Flocking* ini nantinya akan digabungkan dalam satu project game *First Person Military Simulation*. Dimana dalam implementasi program hanya sebatas implementasi algoritma, bukan membuat game *First Person Military Simulation* secara utuh. Dalam hal ini penulis mengarahkan bagaimana tahap – tahap yang ditempuh hingga algoritma *flocking* dapat bekerja dengan baik. Gambar 3.1 merupakan diagram alir metode yang berisi tahapan-tahapan yang dilakukan dalam penelitian yang diusulkan.



Gambar 3.1 Diagram Alir Runtutan Metode Penelitian

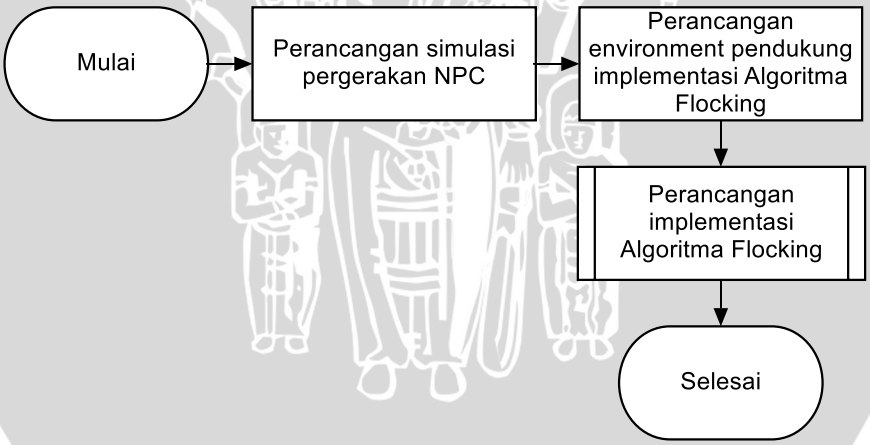
3.1 Studi Literatur

Studi literatur merupakan cara yang digunakan dalam pengumpulan informasi-informasi yang diperlukan dengan sumber yang digunakan berupa data-data yang terkait. Dalam studi literatur dikumpulkan data-data dari sumber yang terpecaya yang menjadikan sumber tersebut menjadi dasar dari teori-teori yang akan diimplementasikan lebih lanjut pada aplikasi yang dibuat. Studi literatur yang digunakan adalah sebagai berikut :

1. Steering Behavior
2. Algoritma Flocking
3. Bahasa Pemrograman C#

3.2 Perancangan Algoritma Flocking

Perancangan terdiri dari tiga tahap, yaitu perancangan simulasi pergerakan NPC, perancangan environment pendukung implementasi, dan perancangan implementasi Algoritma *Flocking*. Tahap-tahap perancangan dijabarkan dalam diagram alir gambar 3.2. Dimana pada perancangan simulasi pergerakan NPC akan dijelaskan behavior yang akan diimplementasikan, sedangkan untuk perancangan environment pendukung implementasi menjelaskan lingkungan pendukung sebelum implementasi algoritma, dan untuk perancangan implementasi menjelaskan langkah-langkah dalam implementasi Algoritma.



Gambar 3.2 Diagram Alir Perancangan Implementasi

3.2.1 Perancangan Simulasi pergerakan NPC

Pada awal perancangan akan dirancang simulasi pergerakan NPC pada lingkungan 3D. Pada awal simulasi disediakan sejumlah 4 NPC sebagai prajurit. Prajurit akan berjalan menuju tujuan dengan bantuan navigator atau player. Navigator nantinya bergerak menuju tujuan dengan mengikuti path yang sudah

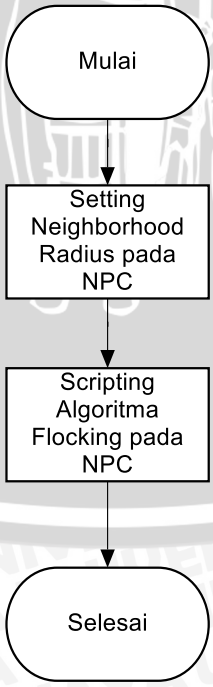
disediakan. Untuk pergerakan menuju player NPC menggunakan *Seek Behavior,* sedangkan Untuk navigasi NPC menggunakan *Arrival Behavior.* Ataupun bisa berbalik penggunaan sesuai dengan model peta Implementasi pada *game design.*

3.2.2 Perancangan Environment Pendukung Implementasi Algoritma Flocking

Environment pendukung implementasi berguna untuk menunjang dari implementasi Algoritma Flocking. Dalam perancangan membutuhkan peta atau latar untuk pergerakan NPC. dalam perancangan ini menggunakan *Plane* 3D yang sudah disediakan oleh Unity3D. *Plane* 3D ini kemudian akan diberikan *script* Map yang menjadikan *Plane* 3D sebagai peta permainan dengan tambahan grid untuk pengolahan perhitungan *Pathfinding*. Untuk *pathfinding* disini menggunakan Algoritma HPA* (*Hierarchial Pathfinding A**). Dimana *script* Map beserta HPA* sudah disediakan oleh rekan satu tim dalam pegerjaan skripsi kali ini. Sehingga pada pengerjaan hanya terfokus pada *movement behavior* saja.

3.2.3 Perancangan Implementasi Algoritma Flocking

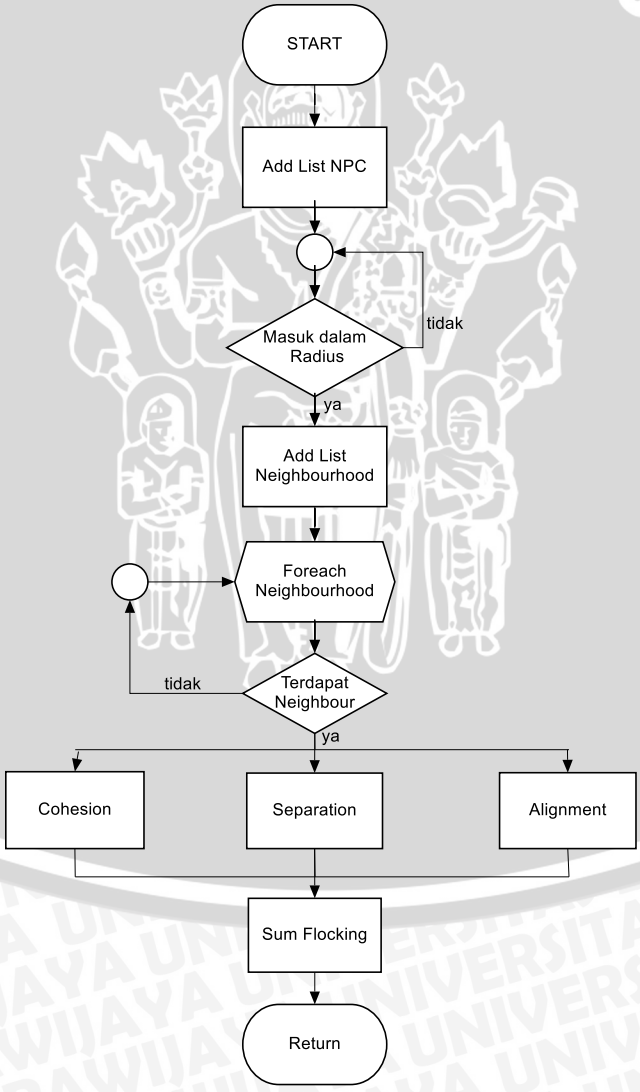
Perancangan Implementasi berisi tentang langkah-langkah dalam melakukan implementasi. Tahap-tahap perancangan terdiri atas : setting *neighborhood, scripting* algoritma *flocking*. Setting *neighborhood* adalah pemberian radius terhadap NPC, sedangkan untuk algoritma *flocking* terdiri dari : *Separation, Cohesion*, dan *Alignment* Tahap-tahap perancangan Implementasi Algoritma *Flocking* dijabarkan dalam diagram alir gambar 3.3.



Gambar 3.3 Perancangan Implementasi Algoritma Flocking

Setiap NPC atau *squad* pertama kali akan diberikan radius berupa *sphere* yang berguna untuk membantu pendeteksian karakter lain dalam *Neighborhood*. *Neighbourhood* berguna untuk pendukung jalannya implementasi algoritma Flocking, dimana algoritma ini melakukan pengecekan pada setiap NPC yang berada di sekitarnya. Sehingga Flocking bisa berjalan jika ada karakter lain di sekitar NPC yang berada dalam *Neighborhood* mereka. Jika berada diluar *Neighborhood* karakter lain akan diabaikan. Kemudian pada setiap NPC juga diberikan *Arrival Behavior* yang berguna untuk membantu para Squad mengikuti leader atupun navigator. *Arrival behavior* hampir sama dengan *seek behavior.* Penggunaan *arrival behavior* bertujuan agar NPC yang mengikuti *leader* bila sudah sampai bisa berhenti, dan apabila jarak melebihi radius dari leader maka kecepatan berubah lebih cepat. Sebagai pembantu terdapat *Navigator* yang bergerak mengikuti *path* yang sudah disediakan HPA*.

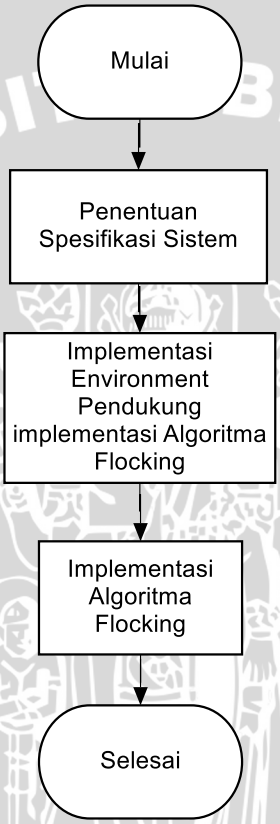
Cara bekerja dari algoritma Flocking akan dijabarkan dalam diagram alir pada gambar 3.4.



Gambar 3.4 Diagram Alir algoritma Flocking

3.3 Implementasi Algoritma Flocking

Bab ini membahas mengenai implementasi algoritma berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, implementasi environment pendukung implementasi, dan implementasi Algoritma Flocking pada *First Person Military Simulation Game*. Tahap-tahap implementasi dijabarkan dalam diagram alir dalam gambar 3.5.



Gambar 3.5 Diagram Alir Implementasi

3.3.1 Penentuan Spesifikasi Sistem

Hasil analisis kebutuhan dan perancangan perangkat lunak yang telah dijelaskan pada bab 3 menjadi acuan untuk melakukan implementasi *Squad Formation Behavior* pada *First Person Military Simulation Game* menggunakan Algoritma *Flocking* dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem untuk implementasi algoritma dijelaskan pada spesifikasi perangkat keras dan perangkat lunak.

3.3.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang dipakai dalam proses pengembangan dijelaskan dalam tabel 3.1.

Tabel 3.1 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
Processor	Intel® Core™ i5-3317U CPU @1.70GHz
Memory	4.00 GB RAM
Hardisk	500GB
Graphic Card	NVIDIA GeForce GT 635M, 2GB

3.3.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang dipakai dalam proses pengembangan dijelaskan dalam tabel 3.2.

Tabel 3.2 Spesifikasi Perangkat Lunak

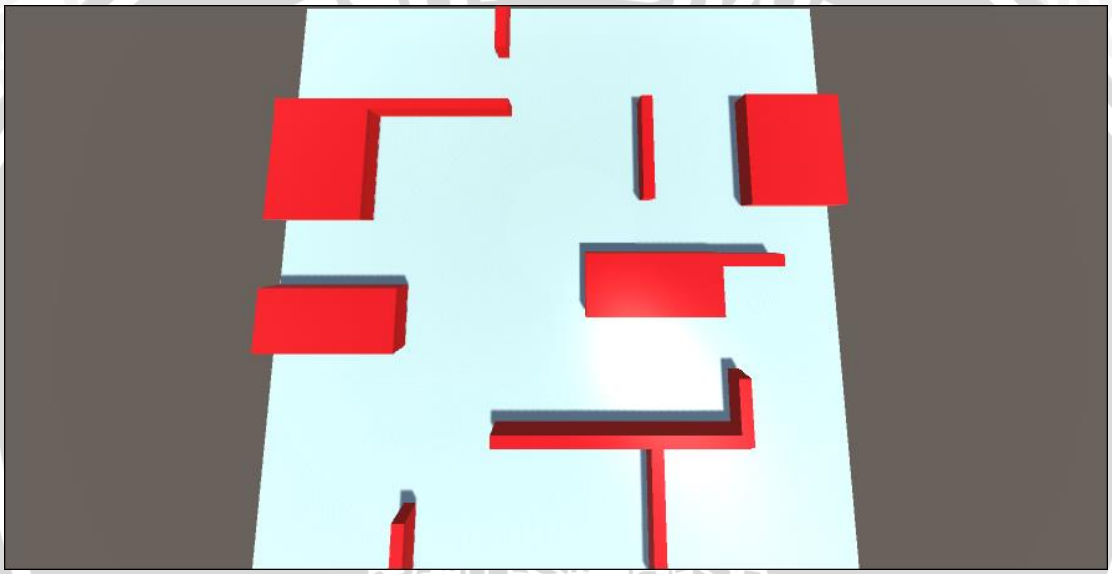
Perangkat Lunak	Spesifikasi
Operating System	Windows 8.1 Pro 64-bit
DirectX Version	DirectX 11
Programing Language	C#
Software Development Kit	Unity Version 5.0.0f4

3.3.2 Implementasi Environment Pendukung Implementasi Algoritma Flocking

Environment pendukung berguna sebagai sarana pendukung sebelum Implementasi Algoritma flocking. Environment pendukung disini antara lain : Implementasi Peta Simulasi, Implementasi Penggunaan HPA* *(Hierarchial Pathfinding A*)* dalam peta simulasi.

3.3.2.1 Implementasi Peta Simulasi

Implementasi peta permainan untuk simulasi *pathfinding* akan langsung dirancang dalam Unity3D pada *editor scene view*. Objek-objek yang akan digunakan antara lain adalah *plane* dan *cube. Plane* akan digunakan sebagai lantai. *Cube* akan digunakan sebagai halangan statis. Hasil implementasi dari masing-masing peta ditunjukkan pada gambar 3.5. Selain implementasi peta, pemberian script Map.cs yang sudah disediakan sebagai media perhitungan atau berguna untuk pemrosesan Algoritma HPA* *(Hierarchial Pathfinding A*)*. Dimana map script akan memberikan penambahan Grid, Graph, dan juga untuk penentuan node radius dan cluster radius, serta Grid World Size untuk penyamaan terhadap plane yang sudah diimplementasikan. Berikut rancangan implementasi peta simulasi dapat dilihat pada gambar 3.6.



Gambar 3.6 Implementasi Peta Simulasi

3.3.2.2 Implementasi Penggunaan HPA*

Penggunaan HPA* sendiri sudah dimudahkan karena dalam game sudah disediakan dengan menggunakan library. Implementasi ini berguna untuk NPC dalam melakukan pencarian jarak terdekat. Berikut script code dari penggunaan Implementasi HPA* dijabarkan pada tabel 3.3.

Tabel 3.3 Sript Code Penggunaan HPA*

```
using HPA;
Deklarasi
Map Mapa;
public GameObject HPA;
public Transform seeker;
public Transform target;
public int speed = 3;
public List<Vector3> Path = new List<Vector3>();
```

Tabel 3.3 (Lanjutan)

```
1   Void Awake() {
2   Mapa = HPA.GetComponent<Map>();
3   }
4   void Start () {
5   Mapa.path = Mapa.map.HPAfindPath(seeker.position,target.position);
6   }
7   void Update () {
8   if (Path.Count > 0) {
9   seeker.position = Vector3.MoveTowards (seeker.position, Path [0], speed *
10  Time.deltaTime);
11                      if (seeker.position == Path [0]) {
12                              Path.Remove (Path [0]);
                        }}
    }
```

Penjelasan :

1. Pada header sebelumnya harus diberikan “using HPA*; “ sebagai penambahan library.
2. Untuk deklarasi terdapat : penentuan Object seeker/ target, penenuan speed dan List Array.
3. Baris 1 merupakan pemanggilan method Awake yang hanya dijalankan sekali pada awal *load scene.*
4. Baris 2 merupakan method untuk pengambilan komponen dari script pada class Map.
5. Baris ke 4 merupakan pemanggilan method Start yang dijalankan dahulu pada saat *load scane.*
6. Baris ke 5 merupakan penentuan jalur pathfinding dimana batasan jarak dibatasi oleh 2 object yaitu : seeker.position dan target.position.
7. Baris ke 7 merupakan pemanggilan method Update yang dijalankan berkali kali pada saat program berjalan, berbeda halnya dengan method start/ awake.
8. Baris ke 8-13 merupakan logika sederhana untuk *movement* mengikuti path. Dimana pada awal akan berjalan bila jumlah path lebih dari 0 (terdapat jalur path), kemudian seeker akan bergerak menuju path ke 0, ketika sampai path 0 path 0 akan di remove. Dengan otomatis seeker akan berjalan lagi ke path selanjutnya.

3.3.3 Implementasi Algoritma Flocking

Implementasi algoritma AI diperlukan untuk menjadikan NPC dapat Bergerak dengan kordinasi yang baik saat berkelompok. Oleh karena itu algoritma AI yang sudah dirancang akan diterapkan ke *First Person Military Simulation Game.*

3.3.3.1 Pra Pengolahan Algoritma Flocking

Sebelum pengolahan algoritma flocking ada script pembantu dalam penyempurnaan algoritma diantara lain : Proses memasukkan data NPC dalam list array untuk perhitungan algoritma. Penentuan velocity tiap NPC, pembuatan face orientation untuk orientasi NPC. Dan berikut script code dari *Add List Array, Set Velocity, dan Set Face Orientation* dijabarkan pada tabel 3.4, 3.5, dan 3.6.

Tabel 3.4 Sript Code Add List Array

Script code Add List Array	
Deklarasi public static List<steer> simpan = new List<steer>();	

1
2
3
	void Start () {
	simpan.Add (this);
	}

Penjelasan :

1. Pada deklarasi dideklarasikan list simpan sebagai *new list.*
2. Baris ke 1 merupakan pemanggilan method Start yang dijalankan dahulu pada saat *load scane.*
3. Baris ke 2 merupakan method penambahan list, dan secara otomatis NPC yang sudah diberikan script ini akan ditambahakan dalam list.

. **Tabel 3.5 Set Velocity**

Script code Set Velocity	

Deklarasi
public float Mass = 10;
public float Friction = .05f;
public Vector3 currentVelocity;
public Vector3 previous = Vector3.zero;
public Vector3 velocity;	

1
2
3
4
5
6
	Vector3 accel = Vector3.zero;
	currentVelocity = Vector3.zero;
	currentVelocity = (transform.position - previous)/Time.deltaTime;
	velocity += currentVelocity / Mass;
	velocity -= currentVelocity * Friction;
	previous = transform.position;

Penjelasan :

1. Pada deklarasi dideklarasikan Mass, friction, currentVelocity, velocity, prev
2. Baris 3-6 merupakan pemberian nilai pada velocity, dengan logika pembagian Mass, Friction, dan waktu.

Tabel 3.6 Set Face Orientation

Script code untuk Penggunaan HPA*	

Deklarasi
public float MaxVelocity = 1;
public bool RotateSprite = true;	

1	if (velocity.magnitude > MaxVelocity)
2	velocity = velocity.normalized * MaxVelocity;
3	if (RotateSprite && velocity.magnitude > 0f){
4	float angle = Mathf.Atan2(velocity.x, velocity.z) * Mathf.Rad2Deg;
5	transform.eulerAngles = new Vector3(transform.eulerAngles.x, angle,
6	transform.eulerAngles.z);
7	}

Penjelasan :

1. Pada deklarasi dideklarasikan MaxVelocity, bool RotateSprite.
2. Baris 1-2 merupakan logika sederhana untuk penyamaan velocity dengan menggunakan .normalized dikalikan dengan MaxVelocity
3. Baris 3-6 merupkan logika sederhana bila velocity.magnitude > 0 , maka melakukan rotasi dengan menggunakan Mathf.Atan2 sebagai perhitungan angle(sudut rotasi).

3.3.3.2 Pengolahan Algoritma Flocking

Untuk pengolahan *Algoritma Flocking* pertama kali dilakukan *set Neighborhood*, yang berguna untuk mendeteksi setiap NPC yang berada di lingkungan *radius.* Kemudian dilakukan penambahan *Algoritma Flocking* yang terdiri dari : *Separation, Cohesion, Alignment.* Dan berikut *script code* dari *Set Neighborhood, Separation, Cohesion, Alignment yang* dijabarkan pada tabel 3.7, 3.8, 3.9, dan 3.10.

Tabel 3.7 Set Neighborhood

Script code untuk Set neighborhood()	
1	void updateNeighbours (){
2	neighbour.Clear ();
3	foreach (steer data2 in steer.simpan) {
4	float distance =
5	Vector3.Distance(data2.transform.position,transform.position);
6	if (distance < nRadius) {
7	neighbour.Add (data2);
8	}
9	}}

Penjelasan :

1. Pada baris 2 untuk pertama kali harus menghapus data sisa, berguna untuk set neighborhood yang hanya ada dalam radius bila diluar maka akan dihapus.

2. Pada baris 3 merupakan method pengambilan data dari list array “simpan” yang sudah di atur pada pra pengolahan.
3. Pada baris 4-5 merupakan pembuatan jarak antara NPC lain dengan karakter.
4. Dan pada baris ke 6 merupakan logika sederhana bila jarak kurang dari nradius karakter (radius tetangga) maka dilakukan add list array neighbor yang sudah disiapkan.

Tabel 3.8 Separation

Script code untuk Separation()	
1	Vector3 seperation(){
2	Vector3 direction = Vector3.zero;
3	Vector3 move = Vector3.zero;
4	foreach (steer data in neighbour)
5	direction += (Vector3)transform.position-
6	(Vector3)data.transform.position ;
7	move += direction.normalized* (speed * Time.deltaTime);
8	move.y = 0;
9	transform.position += move;
10	return transform.position;}

Penjelasan :

1. Pada baris 2-3 merupakan deklarasi vecktor3 direction, move
2. Pada baris 4 merupakan method pengambilan data di list neigbour
3. Pada baris 5 merupakan pembuatan arah jalan, dengan cara mengurangkan posisi karakter dengan posisi npc neighbour.
4. Pada baris 7-9 merupakan cara untuk transformasi posisi, dengan cara direction dikalikan dengan waktu dan kecepatan. Untuk penambahan pada baris 8 merupakan pengaturan agar vector.y selalu dalam kondisi 0.
5. Pada baris 10 merupakan pengembalian nilai transformasi.

Tabel 3.9 Cohesion

Script code untuk Cohesion()	
1	Vector3 cohesion(){
2	Vector3 averagePosition = Vector3.zero;
3	Vector3 direction = Vector3.zero;
4	Vector3 move = Vector3.zero;
5	foreach (steer data in neighbour)
6	averagePosition += ((Vector3)data.transform.position);
7	averagePosition /= (neighbour.Count);
8	averagePosition.y = 0;
9	direction += (Vector3)averagePosition -
10	(Vector3)transform.position;
11	move = direction.normalized* (speed * Time.deltaTime);
12	move.y = 0;
13	transform.position += move;
14	return transform.position.normalized;}
15	

Penjelasan :

1. Pada baris 2-4 merupakan deklarasi vecktor3 direction, move, dan averagePosition
2. Pada baris 5 merupakan method pengambilan data di list neigbour
3. Pada baris ke 6-7 merupakan cara untuk pemberian nilai pada averagePosition dengan penambahan posisi data yang terdapat pada list neighbor. Kemudian dari hasil penambahan akan dilakukan pembagian terhadap banyaknya list neighbour. Sehingga mengahasilkan nilai tengah berupa rata-rata posisi dari data list neighbour.
6. Pada baris ke 8 merupakan pengaturan agar vector.y selalu dalam kondisi 0.
7. Pada baris ke 9-10 merupakan pembuatan arah jalan, dengan cara mengurangkan posisi dari averagePosition dengan posisi karakter.
8. Pada baris 11-14 merupakan cara untuk transformasi posisi, dengan cara direction dikalikan dengan waktu dan kecepatan.
9. Pada baris 15 pengembalian nilai transform position.

Tabel 3.10 Alignment

Script code untuk Alignment()	
1	Vector3 alignment()
2	{
3	Vector3 averageDirection = Vector3.zero;
4	if (neighbour.Count == 0)
5	return averageDirection;
6	
7	foreach (steer data in steer.simpan)
8	averageDirection += data.velocity;
9	averageDirection /= (neighbour.Count);
10	return averageDirection.normalized;}

Penjelasan :

1. Pada baris 3 merupakan deklarasi vecktor3 averageDirection
2. Pada baris 4-5 merupakan logika ketika jumlah neighbor 0 maka pengembalian nilai juga 0.
4. Pada baris 7 merupakan method pengambilan data di list simpan
5. Pada baris 8-9 merupakan pemberian nilai pada averagePosition dengan cara penambahan pada data.velocity, kemudian dibagi dengan banyaknya jumlah neighbour pada list neighbour.
6. Pada baris 10 merupakan pengembalian nilai dari averageDirection.

Setelah algoritma berjalan kawanan melakukan seeking menuju navigator, dimana navigator akan bergerak menuju target tujuan dengan menggunakan pathfinding HPA*. Berikut *script code Seek behavior* yang dijabarkan pada tabel 3.11.

Tabel 3.11 Seek

```
Script code untuk seek()
1   void Seek(){
2         Vector3 direction = Vector3.zero;
3         Vector3 move = Vector3.zero;
4   direction = (Vector3)target.transform.position -
5   (Vector3)transform.position ;
6         move = direction.normalized * (speed * Time.deltaTime);
7         transform.position += movel;
8         }
```

Penjelasan :

1. Pada baris 2-3 merupakan deklarasi vecktor3 direction, dan move.
2. Pada baris 4 merupakan pembuatan arah jalan, dengan cara mengurangkan posisi dari NPC dengan posisi karakter
3. Pada baris 6 merupakan cara untuk transformasi posisi, dengan cara direction dikalikan dengan waktu dan kecepatan.
4. Pada baris 7 pengembalian nilai transform position.

Pada implementasi NPC pasti akan bertabrakan dengan obstacle. Dalam implementasi dibuat logika sederhana, ketika menabrak obstacle NPC akan melakukan pathfinding ke player/navigator. Dengan begitu secara otomatis akan mengarah ke jalan yang benar atau tidak bertabrakan. Ketika sampai ke leader atau navigator NPC akan melakukan seek kembali.

Agar implementasi berjalan dengan baik dibutuhkan semacam perpindahan *state* sehingga NPC berjalan dengan satu behavior saja dan tidak ada penggandaan behavior. Kemudian untuk pendeteksian menggunakan “on trigger enter” ketika trigger state akan pindah menjadi pathfinding ke navigator. Berikut *Script Code* untuk trigger yang dijabarkan pada tabel 3.12.

Tabel 3.12 On Trigger Enter

```
Script code untuk trigger
1   public void OnTriggerEnter(Collider other) {
2               if (other.gameObject.tag == "avoid") {
3                  FsmCommand.toFindCapt = true;
4                   steerSeek = false;
5               }
```

Penjelasan :

1. Pada baris 2-3 merupakan logika sederhana jika NPC collider dengan object dengan tag “avoid” maka bool untuk pemindah state dari FsmCommand akan aktif (true).
2. Kemudian untuk baris ke 4 seek behavior tidak diaktifkan (false).

3.4 Pengujian dan Analisis

Akan dilakukan pengujian berdasarkan implementasi yang telah dibuat berdasarkan metriks pengujian yang telah ditentukan, yaitu panjang jalur dan waktu pencarian. Alur dari pengujian dan analisis dapat dilihat pada gambar 3.7.

Gambar 3.7 Diagram Alir Pengujian dan Analisis

3.4.1 Menentukan Skenario Pengujian

Skenario pengujian algoritma dilakukan dengan melakukan pengecekan pada FPS setiap dilakukan penambahan NPC *squad,* dan Pengujian pada 3 sampel peta dengan ukuran yang berbeda. Disini bertujuan untuk melihat peforma dari implementasi apakah sudah memenuhi serta layak untuk dijalankan atau dimainkan.

3.4.2 Melakukan Pengujian

Pengujian dilakukan penambahan NPC atau agen dalam *Squad,* kemudian dilakukan pencatatan rata-rata FPS (*Frame Per Second*) pada setiap penambahan. Data kemudian dimasukkan kedalam tabel supaya mudah dalam proses analisis. Untuk pengujian peta, peta ukuran kecil, sedang, dan besar masing-masing dilakukan pengujian dengan memberikan beberapa NPC untuk menguji implementasi program.

3.4.3 Menganalisa Hasil Pengujian

Hasil dari tabel pengujian skenario akan dianalisis pada setiap penambahan NPC. Sehingga nantinya akan nampak, pada penambahan keberapa FPS mengalami penurunan (*down*). Jika FPS mengalami penurunan (*down*) maka implementasi game masih belum layak untuk dimainkan atau dijalankan. Untuk

pengujian peta nantinya akan di analisa bagaimana peforma dari implementasi apakah sudah berjalan baik atau tidak, disertai dengan pendataan masing-masing sampel peta.

BAB 4 HASIL

Pada bab ini dilakukan proses pengujian terhadap implementasi *Squad Formation Behavior* pada *First Person Military Simulation Game* menggunakan Algoritma *Flocking*. Pengujian dilakukan dengan menguji tiap skenario dan pencacatan hasil berdasarkan parameter pengujian.

4.1 Spesifikasi Pengujian

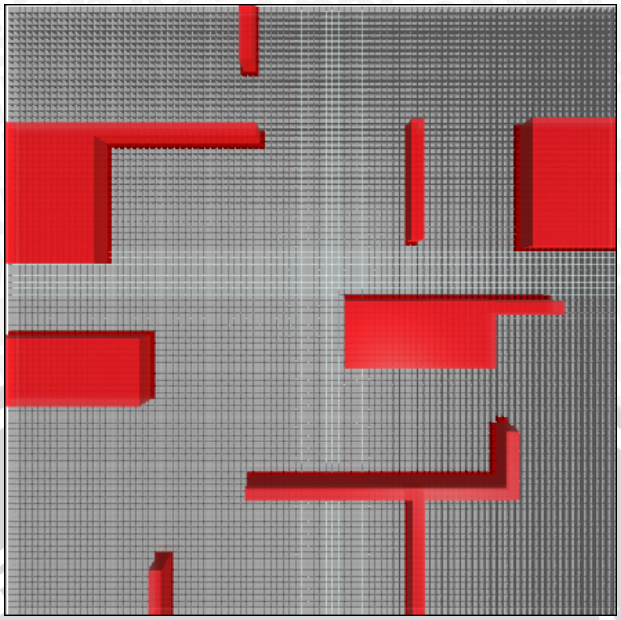
Spesifikasi perangkat keras yang digunakan dalam uji coba adalah procesor Intel Core i5, harddisk 500GB, memori 4GB RAM, kartu grafis NVIDIA GeForce GT 635M 2GB. Sedangkan spesifikasi perangkat lunak yang digunakan adalah Windows 8.1 Pro 64-bit, *game engine* Unity v5.0.

4.2 Menentukan Skenario Pengujian

Pengujian dilakukan untuk mengetahui valid atau tidaknya *behaviour* yang sudah dirancang dengan implementasinya. Selain itu, pengujian juga dilakukan untuk mengetahui efisiensi dari sistem. Skenario peta untuk pengujian berdasar pada ukurannya yaitu: kecil, sedang, besar. Dari ketiga ukuran tadi akan diuji dengan jumlah NPC mulai dari sedikit, sedang, dan banyak. Untuk kategori sedikit terdapat 4 buah NPC, untuk kategori sedang terdapat 12 NPC, dan untuk kategori banyak terdapat 20 NPC. Selain itu juga dilakukan pengecekan FPS dari setiap penambahan NPC, apakah layak atau tidaknya jika di implementasikan pada game, dan untuk mengetahui sampai penambahan ke berapa NPC FPS sudah tidak layak untuk dimainkan dalam game.

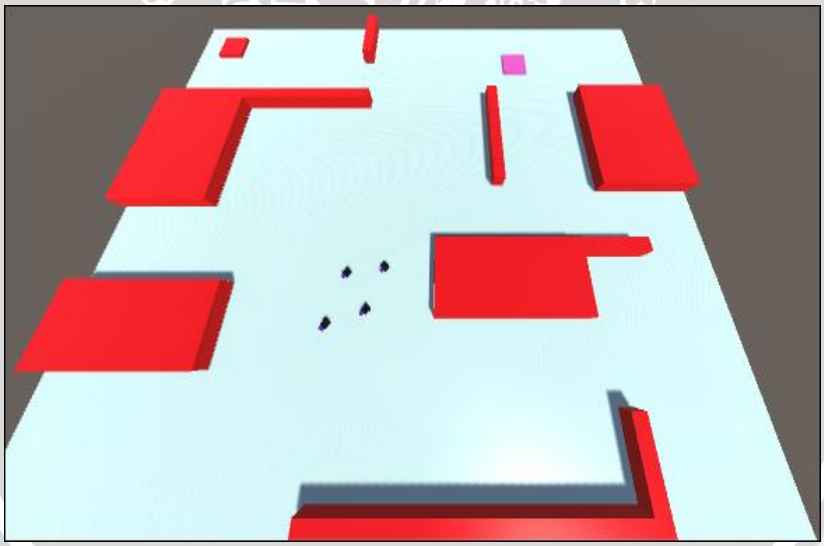
4.3 Melakukan Pengujian

Pada subbab ini akan dijabarkan uji coba dan hasil pengujian dari penggunaan *cluster* dengan ukuran 100x100, 150x150, 200x200 *grid*. Berikut merupakan peta yang digunakan untuk pengujian dapat dilihat pada gambar 4.1, dimana peta sesuai dengan perancangan peta yang sudah dirancang sebelumnya. Penggunaan peta untuk pengujian ini disamakan, hanya berbeda ukuran *cluster* atau besar peta.



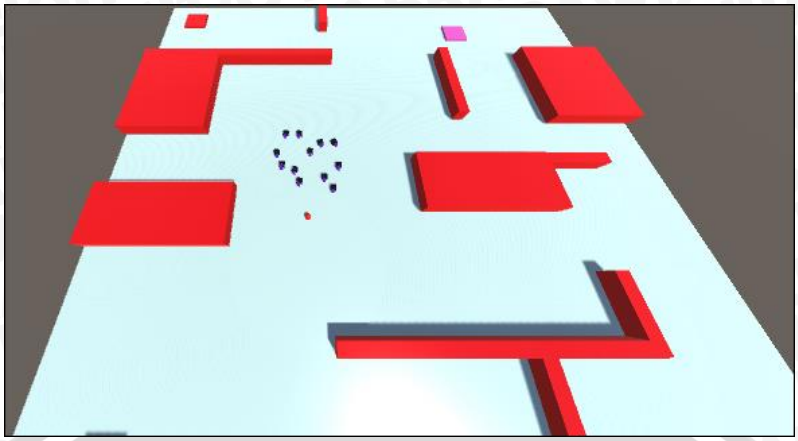
Gambar 4.1 Contoh Peta Pengujian

Berikut contoh pengujian terhadap FPS pada gambar 4.2. dengan melakukan penambahan sebanyak 4 NPC.



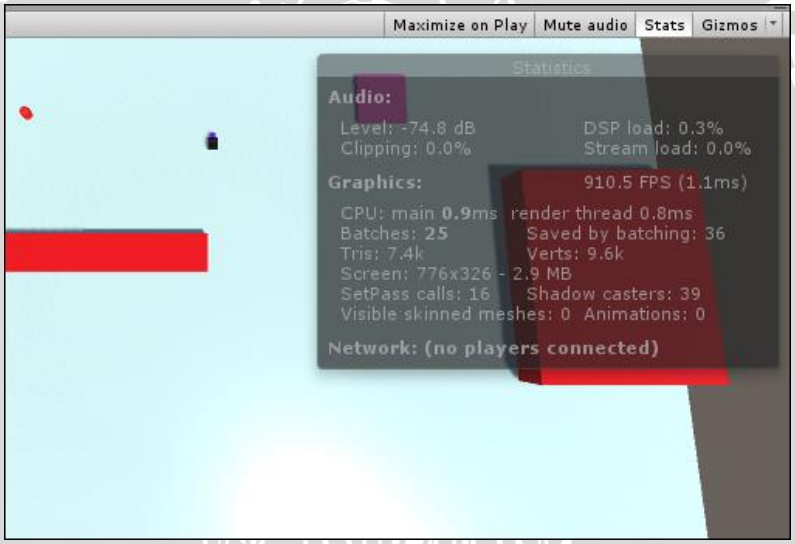
Gambar 4.2 Contoh Pengujian Terhadap FPS 1

Berikut contoh pengujian terhadap FPS pada gambar 4.3. dengan melakukan penambahan sebanyak 12 NPC.



Gambar 4.3 Contoh Pengujian Terhadap FPS 2

Untuk pendataan pada pengujian FPS, dalam pengujian menggunakan penambahan tampilan Stats dalam Unity, yang berguna untuk menampilkan data Statistik dalam simulasi. Berikut adalah tampilan Stats dalam Unity dapat dilihat pada gambar 4.4. dan untuk pendataan FPS terdapat dalam bagian Graphics Stats.



Gambar 4.4 Stats Unity

Berikut hasil pengujian simulasi terhadap FPS, dimana pada pengujian dilakukan pencatatan nilai FPS untuk setiap penambahan NPC, untuk penjabaran hasil pengujian dapat dilihat pada tabel 4.1.

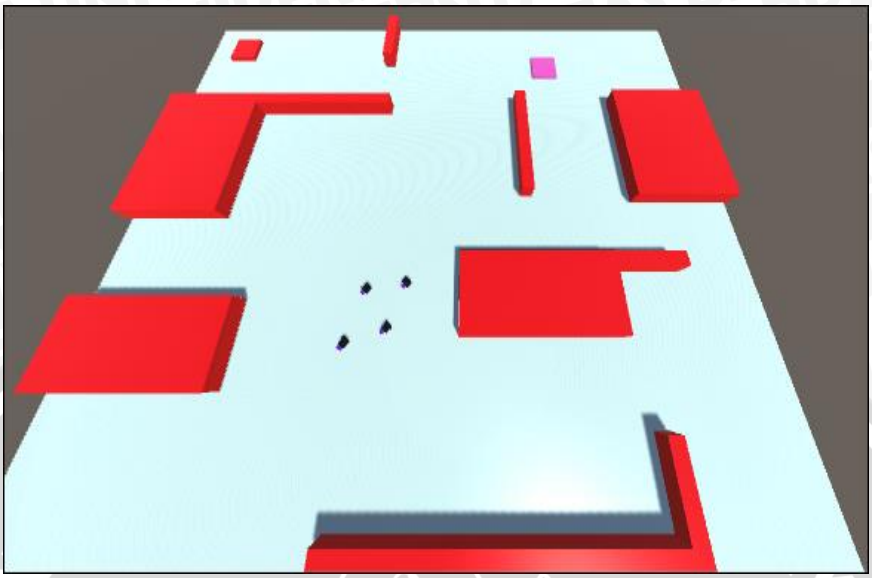
Tabel 4.1 Hasil Pengujian Terhadap FPS

Jumlah NPC	FPS
1	67
2	65

Tabel 4.1 (Lanjutan)

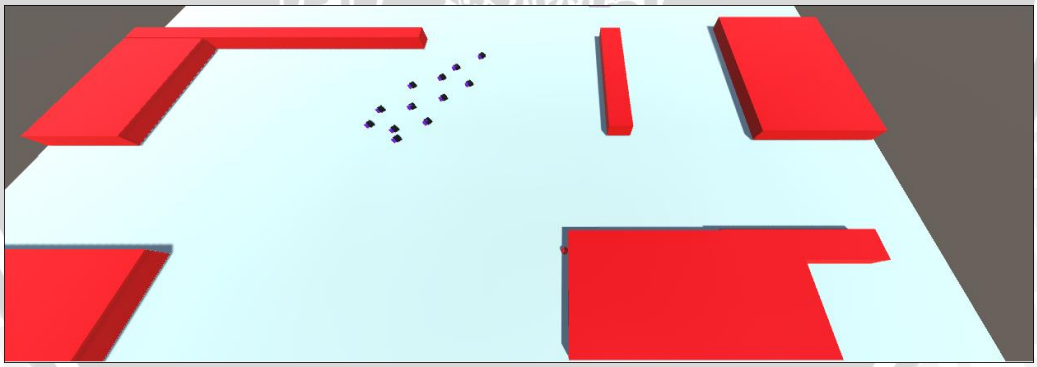
3	63
4	64
5	55
6	46
7	40
8	34
9	31
10	28
11	26
12	24
13	22
14	20
15	19
16	17
17	16
18	16
19	15
20	14

Berikut contoh pengujian pada gambar 4.5. dengan menggunakan 4 NPC (sedikit) dengan ukuran peta kecil.



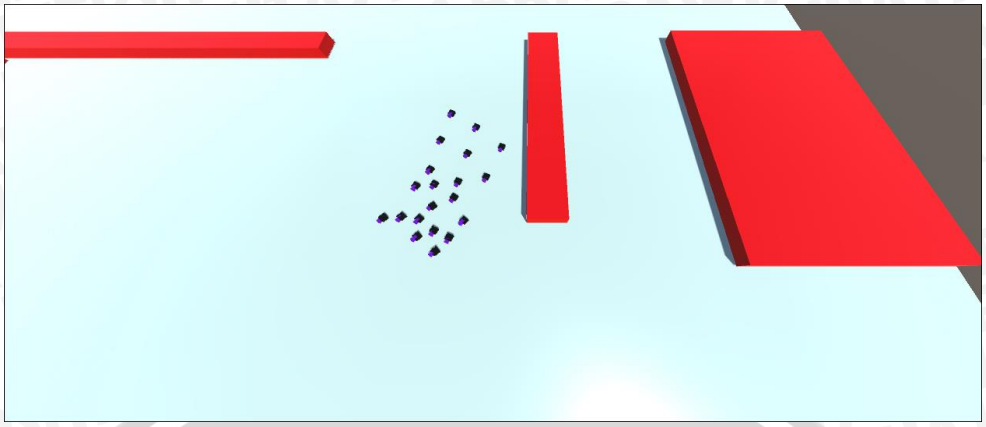
Gambar 4.5 Contoh Pengujian Algoritma Flocking 1

Berikut contoh pengujian pada gambar 4.6. dengan menggunakan 12 NPC (sedang) dengan ukuran peta sedang.



Gambar 4.6 Contoh Pengujian Algoritma Flocking 2

Berikut contoh pengujian pada gambar 4.7. dengan menggunakan 20 NPC (banyak) dengan ukuran peta besar.



Gambar 4.7 Contoh Pengujian Algoritma Flocking 3

Berikut hasil pengujian Jumlah *Stack Collision* terhadap 3 sampel peta beserta kuantitas jumlah NPC dengan kategori : sedikit, sedang, dan banyak dimana detail jumlah sesuai dengan scenario pengujian. *Stack Collision* adalah kondisi dimana sebuah NPC mengalami tabrakan pada sebuah dinding atau halangan yang mengakibatkan NPC berhenti bergerak. Untuk penjabaran hasil pengujian dapat dilihat pada tabel 4.2.

Tabel 4.2 Hasil Pengujian Terhadap Sampel Peta

Ukuran Peta	Banyak NPC	Jumlah Stack Collision
Kecil	Sedikit	0
Kecil	Sedang	0
Kecil	Banyak	0
Sedang	Sedikit	0
Sedang	Sedang	0
Sedang	Banyak	0
Besar	Sedikit	0
Besar	Sedang	0
Besar	Banyak	0

BAB 5 PEMBAHASAN

Pada Bab ini berisi mengenai pembahasan terhadap pengujian yang telah dilakukan. Sehingga mendapatkan evaluasi dan kesimpulan terhadap implementasi program.

5.1 Analisis Hasil Pengujian

Analisis bertujuan untuk mendapatkan kesimpulan dari hasil pengujian implementasi Algoritma *Flocking*. Proses analisis terhadap hasil pengujian yaitu dilihat dari FPS dalam game serta pengujian pada beberapa sampel peta.

Pada pengujian FPS dapat dilihat bahwa pada pengujian pada NPC ke 1-5 jumlah FPS masih tinggi, sehingga pada game tidak ada gerakan lambat yang mengganggu dalam permainan. Sedangkan pada pengujian NPC ke 6-7 sudah mengalami penurunan yaitu 40-50 FPS, meskipun demikian Game masih baik untuk dimainkan. Sedangkan pada pengujian 8-10 jumlah FPS sudah banyak mengalami penurunan yaitu diantara 28-35 FPS, sehingga pergerakan dari permainan mengalami penurunan, meskipun demikian Game masih lancar digunakan, hanya saja tidak sebagus 40 FPS ke atas. Untuk pengujian NPC ke 11-14 jumlah FPS berkisar antara 20-26 FPS, sehingga dalam permainan peforma menjadi berkurang. Sedangkan pada NPC ke 15-20 jumlah FPS kurang dari 20 FPS, sehingga game berjalan lamban dan dapat mengganggu dalam permainan. Dapat disimpulkan bahwa dalam permainan game dibutuhkan minimal FPS lebih dari 40 FPS untuk mendapatkan peforma terbaik. Sedangkan untuk FPS dibawah 30 sudah mempengaruhi kecepatan dalam permainan. Menurut Sarkar (2014) mengatakan bahwa 30 FPS adalah dasar yang umum para pembuat game untuk tidak jatuh lebih kebawah. Dibawah ambang batas tersebut, gambar akan terlihat terlihat berombak atau lag. Bakan untuk FPS yang kurang dari 20 sudah sangat tidak bagus untuk digunakan. Oleh karena dalam desain implementasi *First Person Military Simulation Game* ini hanya membutuhkan 4 NPC untuk *Squad Formation Behavior* maka Implementasi Algoritma *Flocking* pada program sudah memenuhi standart bagus untuk di Implementasikan dalam game.

Untuk Pengujian pada sampel peta dapat dilihat dalam tabel pengujian untuk sampel peta Kecil, Sedang, Besar dengan jumlah NPC yang berbeda-beda dalam pengujian, Pergerakan NPC bisa sampai tujuan semua. Hal yang paling mempengaruhi sampai atau tidaknya NPC ke target tujuan adalah penggunaan Navigasi, dimana navigasi ini adalah sebuah obyek yang di dalamnya di berikan *Script* penggunaan *pathfinding* HPA* sehingga membantu menavigasi NPC untuk sampai ke target tujuan. Bukan hanya itu pada peta tentunya ada Wall atau Avoider yang mengganggu NPC dalam mengikuti Navigasi. Oleh karena itu setiap NPC diberikan kondisi ketika mengalami tabrakan, NPC akan melakukan pathfinding menuju *navigator* dan kondisi kembali *Seek* atau *Arrival*. Sehingga dapat terlihat seperti agen cerdas. Untuk pergerakan *Flocking* ketika menggunakan *Seek* menuju *Navigator* dibanding dengan *Seek* menuju *Player*

berbeda Kinerjanya. Pada *Seek* Navigator, *Flocking* tidak berjalan optimal dikarenakan seringnya melakukan tabrakan dengan dinding *obstacle* sehingga sering melakukan *pathfinding* ke *navigator*. Karena *navigator* cenderung mencari jalan tersingkat, yang kebanyakan berdekatan dengan dinding, dan otomatis gerakan NPC dibelakangnya sering mengalami benturan terhadap dinding atau rintangan. Meskipun NPC dapat melakukan pathfinding menuju *navigator*, Algoritma *Flocking* sebenarnya bisa berjalan opimal ketika berada dalam lahan yang lapang dimana *Group Movement* akan tampak lebih indah kordinasinya. Berbeda ketika NPC bergerak mengikuti *player*, gerakan bisa berjalan baik karena *player* tidak terpusat dengan pathfinding, karena *player* dijalankan manual oleh pemain dalam game.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang dilakukan terhadap implementasi *Squad Formation Behavior* pada *First Person Military Simulation* menggunakan *Algoritma Flocking* dapat diambil kesimpulan sebagai berikut :

1. Implementasi Squad Formation Behavior berhasil pada *First Person Military Simulation Game* berhasil dilakukan dengan menggunakan algoritma *Flocking.*
2. Penggunaan HPA* untuk membantu navigasi dalam Pathfinding berhasil dilakukan. Sehingga dapat membantu dalam pencarian lokasi terdekat dalam sebuah Peta.
3. Implementasi Algoritma *Flocking* sudah memenuhi syarat Standart minimal FPS. Sehingga dapat berjalan dengan baik.
4. Algoritma Flocking dapat bekerja baik pada tempat atau latar yang lapang.

6.2 Saran

Untuk meningkatkan hasil yang telah dicapai dari penelitian ini dapat dilakukan beberapa perbaikan sebagai berikut:

1. Agar Flocking bisa berjalan dengan optimal, dalam pembuatan peta akan lebih bagus bila ada sebuah lapangan kosong, sehingga algoritma dapat berjalan dengan baik.
2. Untuk Penelitian selanjutnya, penggunaan algoritma *Flocking* bisa ditambahkan dengan Algoritma *Steering Behavior* yang lain, seperti obstacle *avoidance , pursuit, evasion, wander* yang dapat menjadikan NPC menjadi lebih cerdas.

DAFTAR PUSTAKA

Milington, Ian. & Funge, John. 2009. *Artificial Intelligence for Games Second Edition*. Morgan Kaufman. Burlington : Massachusetts.

C. W. Reynolds. 1987a. *Flocks, Herds, and Schools : A Distributed Behavioral Model.* California : ACM SIGGRAPH Conference 25–34.

C. W. Reynolds. 1999b. *Steering Behavior for Autonomous Characters*. California : Game Developers Conference 763-782.

Z. Shen. & S. Zhou. 2006. *Behavior representation and simulation for military operations on urbanized terrain*. International Journal of Computer Games Technology, vol. 82, no. 9, 593–607.

Van der Heijden, Marcel., Bakkes, Sander., & Spronck, Pieter. 2008. *Dynamic Formations in Real-Time Strategy Games*. IEEE Symposium on Computational Intelligence and Games.

Wei, Zhao., Xiao-fang, Xie. & Shuai, Tu. 2010. *Virtual Soldier Battle Efficiency Evaluation* , Yantai : Department of Ordnance Science and Technology.

Wang, Zongyao. & Gu, Dongbing. 2007. *Fuzzy Control of Distributed Flocking System*. Colchester : Department of Computer Science, University of Essex.

Fathy, Heba., Raouf, Osama A. & Abdelkader, Hatem. 2014. *Flocking Behaviour of Group Movement in Real.* Egypt : Faculty of Computer and Information Benha University & Menoufiya University.

Kyaw, Aung Sithu., Peters, Clifford. & Naing Swe, Thet. 2013. *Unity 4.x Game AI Programming*. Birmingham : Packt Publishing Ltd.

Pamungkas, Wijanarko S. & Lili, Suhadi. 2010. *Implementasi Automated Path Dan Route Finding Untuk Unit Behavior Pada Real-Time Strategy Game Dengan Menggunakan Fuzzy Logic*. Surabaya : Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember.

Bata, Julius., 2011. *Agent Cerdas Dan Game Komputer*. [online] tersedia di: <https://juliusbata.wordpress.com/tag/npc/> [Diakses 11 Nopember 2015].

Biefeld, Nathan., 2014. *Flocking*. [online] tersedia di : <https://www.youtube.com/watch?v=RI2mViPlJzc&list=PLZMkqlPED79wLRniY_8-sxGx7TuNPUhts> [Diakses 14 Juni 2015].

Bevilacqua, Fernando., 2012. *Understanding Steering Behaviors: Seek*. [online] tersedia di : <http://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-seek--gamedev-849> [Diakses 12 Mei 2015].

Simplistic Production., 2014. *Steer2D*. [online] tersedia di : <https://www.assetstore.unity3d.com/en/#!/content/21381> [Diakses 20 April 2015]

Attila., 2011. *Working on a AI Steering Behavior script*. [online] tersedia di : <http://forum.unity3d.com/threads/working-on-a-ai-steering-behavior-script.127655/> [Diakses 21 Mei 2015]

Sarkar, Samit., 2014. *Why frame rate and resolution matter: A graphics primer.* [online] tersedia di : <http://www.polygon.com/2014/6/5/5761780/frame-rate-resolution-graphics-primer-ps4-xbox-one> [Diakses 12 Desember 2015]