1. PENDAHULUAN

1.1. Latar Belakang Permasalahan

Game merupakan suatu aktivitas paling menyenangkan dan *enjoyable* apabila *game* tersebut memberikan tantangan yang cukup untuk pemainnya. Contoh tantangan tersebut misalnya, mempelajari cara bermainnya, memecahkan masalah, atau menemukan hal-hal baru ketika bermain game tersebut (Korhonen & Koivisto, 2006). Oleh karena itu, *game* membutuhkan perilaku musuh yang tepat agar *game* tersebut dapat dinikmati oleh pemain-pemainnya.

Video games merupakan hiburan dan tantangan. *Video games* tidak akan menyenangkan jika tidak ada tantangannya. Apabila tantangan tersebut terlalu mudah, maka *game* tersebut akan menjadi membosankan. Tetapi jika sebaliknya, *game* tersebut dapat membuat pemain frustasi. Hal ini berhubungan dengan *flow-state*, yaitu ketika kemampuan pemain dan tantangan dari game setara (Sofyan, Akbar, & Afirianto, 2019). Di dalam seri permainan *turn-based tactics game,* biasanya musuh AI memiliki fitur tambahan seperti *unit* yang lebih banyak dan *power* yang lebih kuat. Hal ini membuat *player* yang ingin mendapatkan pengalaman bermain yang optimal tanpa mengeluarkan waktu lebih banyak menjadi enggan untuk bermain permainan tersebut meskipun menyukai *genre* tersebut, karena hal-hal tersebut mengharuskan *player* untuk melakukan aktivitas bernama *grinding*. *Grinding* adalah sebuah aktivitas dimana seorang *player* melakukan *simple action* yang sama berulang kali untuk mendapatkan *resources* (Karlsen, 2011).

Goal-Oriented Action Planning adalah suatu metode *decision-making* yang dapat membuat suatu karakter tidak hanya melakukan apa yang akan dia lakukan, tetapi juga menentukan bagaimana cara ia melakukannya. Dengan struktur GOAP, GOAP mampu memfasilitasi suatu karakter dengan cara mempertahankan dan menggunakan ulang *behavior* tersebut disesuaikan dengan situasi dimana karakter tersebut berada. Sistem GOAP tidak akan mengganti kebutuhan akan Finite-state machine, tetapi lebih menyederhanakan FSM yang dibutuhkan (Orkin, 2003). Di dalam sisi pengimplementasian, dengan menggunakan GOAP, apabila ada *action* yang ingin ditambahkan, pembuat hanya tinggal menambahkan *action* ke dalam program, tanpa perlu mengganti *action* lainnya. Dengan menggunakan metode *Goal-Oriented Action Planning*, *agent* AI akan memiliki aksi yang cukup bervariasi dan adaptif pada *state* yang dia alami, sehingga *player* dapat bermain dengan *agent* AI yang memiliki *resources*

yang sama dengan kesulitan yang cukup menantang untuk *player* sehingga *player* dapat menikmati permainan secara optimal tanpa harus melakukan *grinding*.

Penelitian tentang teori pengimplementasian GOAP ke dalam *game* pernah dilakukan oleh Jeff Orkin pada tahun 2003. Jeff Orkin menganalisa sebuah game bernama *No One Lives Forever 2: A spy in H.A.R.M.'s Way (NOLF2)*, karena *game* tersebut adalah salah satu contoh yang memiliki *goal-directed autonomous characters*, tanpa kemampuan untuk *planning*. Karakter di dalam *NOLF2* secara konstan melakukan evaluasi ulang tujuan mereka, dan memilih tujuan yang paling relevan untuk mengontrol perilaku mereka. Di dalam penelitiannya, Jeff Orkin mengatakan bahwa sebuah karakter akan membuat sebuah rencana secara *real-time* dengan cara memasukkan tujuan di dalamnya kepada sebuah sistem bernama *planner*. *Planner* tersebut akan mencari *action* yang dibutuhkan untuk sebuah *sequence* yang akan membawa sebuah karakter dari *starting state* hingga *goal state*. Jika *planner* tersebut sukses, *planner* akan menghasilkan sebuah rencana untuk diikuti oleh karakter tersebut untuk pengarahan perilakunya. Jika tidak, maka karakter akan meninggalkan rencana sekarang dan akan membuat yang baru.

Magnusson dan Hall pernah melakukan penelitian serupa pada sebuah *Real-time Strategy Game* pada tahun 2010. Magnusson membuat sebuah AI bernama AI Ice, yang bertujuan untuk mengalahkan musuh di dalam *game* tersebut. Untuk melakukan hal tersebut, AI Ice akan mencari informasi di dalam *gameplay* untuk menentukan, membuat, dan menjalankan sebuah tugas sesuai prioritas yang paling tinggi. Hasil dari penelitian tersebut mengatakan bahwa AI Adaptive dengan metode GOAP yang memiliki fitur yang sama dengan pemain manusia (*disabled fog of war, no extra resource, etc*) mampu mengalahkan AI statis yang memiliki fitur yang lebih dari fitur pemain manusia.

Tujuan yang diangkat dari penelitian ini adalah untuk kontribusi pengetahuan lebih lanjut tentang metode *Goal Oriented Action Planning* pada dataset yang berbeda, khususnya *Turn-based Tactics Video Game*, dengan tujuan untuk membuat *Agent* AI yang adaptif sesuai dengan state yang dialami AI pada *genre game* tersebut. Dengan aksi AI yang variatif dan adaptif, AI tidak membutuhkan *resources* tambahan, sehingga *player* tidak memerlukan *grinding* untuk mendapatkan *progress* di dalam *game* tersebut, tetapi masih mendapatkan kesulitan yang cukup menantang ketika melawan AI tersebut. Agar permainan tidak berjalan dengan monoton, maka permainan diberikan sisi *gambling* berupa *hidden personality* di tiap karakter agar gerakan dari karakter bisa berbeda dari satu pertandingan ke pertandingan lainnya. Hasil dari penelitian ini akan berupa uji coba dari pemain melawan *agent* AI GOAP dan

uji coba AI GOAP melawan AI Finite-State Machine pada umumnya, untuk mengetahui apakah metode GOAP lebih baik daripada Finite-State Machine pada *game Turn-based tactics*.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang ada, permasalahan dapat dirumuskan sebagai berikut :

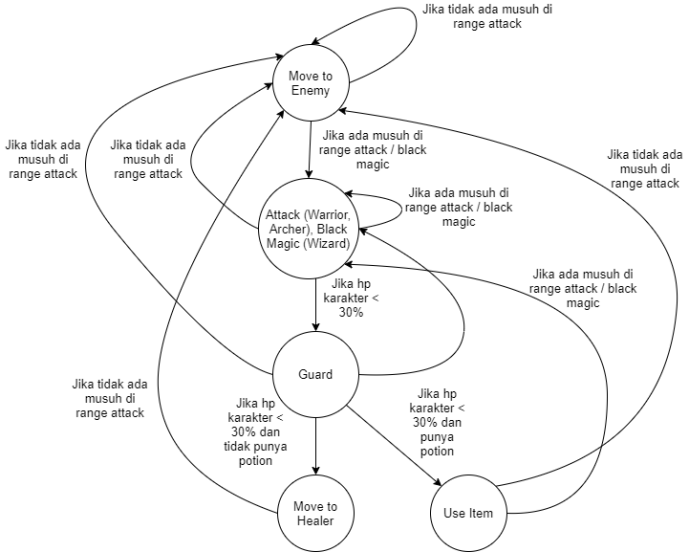
1. Apa saja yang akan dilakukan oleh AI GOAP untuk mencapai tujuannya di dalam situasi yang berbeda-beda?
2. Seberapa pengaruh performa agent AI GOAP yang tidak memiliki *resources* tambahan terhadap tingkat kepuasan pemain jika dibandingkan dengan AI FSM yang memiliki *resources* tambahan?
3. Seberapa baik performa agent AI GOAP jika dibandingkan dengan agent AI FSM yang memiliki *resources* tambahan?

1.3. Ruang Lingkup

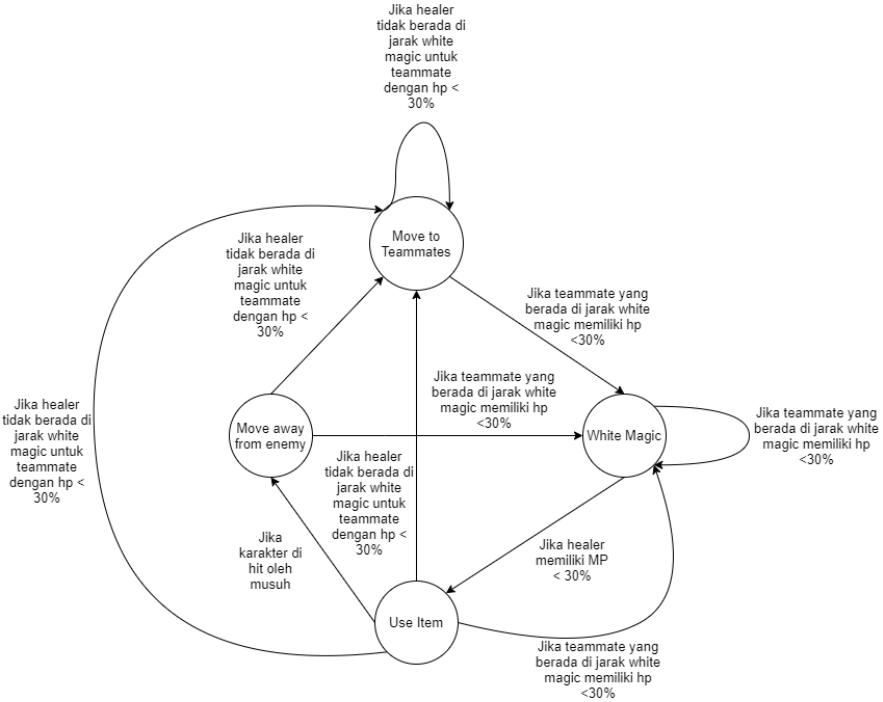
Ruang lingkup skripsi ini dibatasi pada :

1. *Game* dibuat menggunakan Unreal Engine 4.
2. *Game* memiliki 4 jenis karakter berbeda, yaitu Warrior, Wizard, Archer, dan Healer. Tiap karakter memiliki tingkat serangan dan pertahanan yang berbeda-beda, serta aksi yang berbeda-beda. Setiap karakter akan memiliki *personality* (*aggressive, neutral, or non-aggressive*) yang tidak diketahui oleh musuh. *Personalities* tersebut memiliki algoritma gerakan yang berbeda-beda untuk mencegah permainan bersifat monoton.
3. *Game* memiliki 2 jenis serangan dan pertahanan, yaitu *physical* dan *magical*.
4. *Game* bersifat *Tile-based system*, dimana *player* dan AI dapat menggerakan karakter ke *tile* tertentu sesuai keinginan dan batasan karakter tiap putaran.
5. Action yang ada di *game* ini untuk dilakukan karakter-karakter ada enam, yaitu :
    a. Move
    b. Attack
    c. Guard
    d. Black Magic
    e. White Magic
    f. Use Item

6. Menggunakan metode *Goal-Oriented Action Planning* dalam pembuatan agent AI untuk melawan agent AI FSM dan pemain manusia. State yang dimiliki AI FSM adalah sebagai berikut :



Gambar 1.1. *Finite-State Machines* untuk karakter *non-healer*



Gambar 1.2. *Finite-State Machines* untuk karakter *Healer*

7. Output dari penelitian ini adalah program agent AI dengan metode *Goal-Oriented Action Planning* menggunakan A* search.

8. Tiap *move* yang dilakukan oleh agent AI GOAP akan dicatat sebagai perbandingan dengan move yang dilakukan oleh metode FSM.
9. Pengujian seberapa adaptif agent AI GOAP berupa sepuluh kali pertandingan melawan *player* yang memiliki pengalaman bermain yang berbeda-beda.
10. Pengujian keberhasilan performa agent AI GOAP berupa perbandingan *winrate* terhadap melawan agent AI FSM dalam 10 pertandingan dengan *resources* yang berbeda-beda.
11. Pengujian kepuasan pemain melawan agent AI GOAP dilakukan dengan cara kuisioner.

1.4. Tujuan Skripsi

Tujuan dari skripsi ini adalah pembuatan agent AI yang adaptif terhadap berbagai situasi menggunakan metode *Goal-Oriented Action Planning* pada *Turn-based Tactics Video Games* dengan harapan dapat memberikan kesulitan yang cukup menantang meski agent AI tidak memiliki *resources* tambahan ketika melawan *human player* sehingga *player* tidak memerlukan *grinding* untuk mendapatkan *progress* di dalam *game* tersebut.

1.5. Metodologi Penelitian

Langkah-langkah dalam mengerjakan skripsi :

1. Studi Literatur
   - 1.1 *Goal-Oriented Action Planning & Finite-State Machine*
   - 1.2 *Unreal Engine 4*
2. Perencanaan dan Pembuatan Program
   - 2.1 Pembuatan karakter-karakter serta aksinya menggunakan Unreal Engine 4.
   - 2.2 Pembuatan GOAP Planner beserta *cost* dari aksi-aksinya untuk tiap jenis karakter.
   - 2.3 Pembuatan *gameplay* dan fitur menggunakan *Unity*.
3. Pengujian dan Analisis Program
   - 3.1 Melakukan uji coba agent AI GOAP terhadap agent AI FSM yang memiliki *resources* yang bermacam-macam dan *human player*.
   - 3.2 Melakukan survey agent AI GOAP dan agent AI FSM terhadap kepuasan pemain.
   - 3.3 Analisa hasil pengujian dari aplikasi.

4. Pengambilan kesimpulan

   4.1 Membuat kesimpulan tentang hasil penelitian dari analisa yang sudah dilakukan.

   4.2 Membuat saran untuk penelitian serupa kedepannya.

5. Pembuatan Laporan

1.6. Sistematika Penulisan

Penulisan laporan Skripsi ini dibagi menjadi beberapa bab, yaitu :

BAB I: PENDAHULUAN

Bab I berisikan judul, latar belakang, perumusan masalah, ruang lingkup, tujuan skripsi, metodologi penelitian, dan sistematika penulisan yang akan digunakan.

BAB II: LANDASAN TEORI

Bab II berisikan teori-teori serta metode-metode yang digunakan dalam pembuatan skripsi.

BAB III: ANALISIS DAN DESAIN SISTEM

Bab III berisikan analisis dan desain sistem yang dibuat

BAB IV: IMPLEMENTASI SISTEM

Bab IV berisikan tentang implementasi sistem berdasarkan desain sistem seperti pada Bab III.

BAB V: PENGUJIAN SISTEM

Bab V berisikan pengujian sistem yang telah dibuat pada Bab IV.

BAB VI: KESIMPULAN DAN SARAN

Bab VI berisikan kesimpulan yang dapat diambil terhadap hasil yang dicapai, dan saran–saran yang berguna bagi pengembangan selanjutnya.

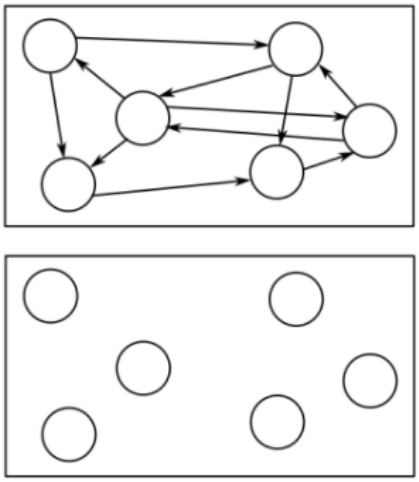
2. LANDASAN TEORI

Pada bab ini akan dijelaskan mengenai teori yang dipakai untuk menyelesaikan masalah dalam pembuatan agent AI *Goal-Oriented Action Planning* di dalam *Turn-based Tactics Video Game*.

2.1. Tinjauan Pustaka

2.1.1. Goal-Oriented Action Planning

Goal-Oriented Action Planning adalah suatu metode *decision-making* yang dapat membuat suatu karakter tidak hanya melakukan apa yang akan dia lakukan, tetapi juga menentukan bagaimana cara ia melakukannya. Dengan struktur GOAP, GOAP mampu memfasilitasi suatu karakter dengan cara mempertahankan dan menggunakan ulang *behavior* tersebut disesuaikan dengan situasi dimana karakter tersebut berada. Sistem GOAP tidak akan mengganti kebutuhan akan Finite-state machine, tetapi lebih menyederhanakan FSM yang dibutuhkan (Orkin, 2003). GOAP merupakan teknik untuk *decision making* yang akan menghasilkan rantai aksi yang bernama *plan* untuk mencapai suatu *goal state* yang sudah ditentukan sebelumnya (Studiawan et al, 2018). GOAP biasanya memiliki 3 komponen utama dalam melakukan tugasnya, yaitu *goals* dan *actions*. *Goals* atau tujuan disini adalah sebuah kondisi yang harus dicapai oleh suatu *agent*, dengan melakukan *action-action* yang dapat mencapai kondisi tersebut.



Gambar 2.1. *Simplification* dari GOAP

Sumber : Studiawan, R., Hariadi, M. & Sumpeno, S. (2018). *Tactical Planning in Space Game using Goal-Oriented Action Planning.* Journal on Advanced Research in Electrical Engineering, 2(1), 5. https://doi.org/10.12962/j25796216.v2.i1.32

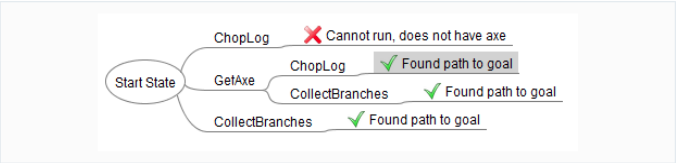
Contoh dari pemakaian GOAP adalah ketika ada seorang agen yang ingin membuat sebuah kayu bakar. Untuk membuat sebuah kayu bakar, agen tersebut memiliki tiga tindakan yang sudah ditetapkan, yaitu :

a. *getAxe.* Biaya: 2. Prekondisi: "*An axe is available*", "*doesn't have an axe*". Efek: "*has an axe*".
b. *chopLog*. Biaya: 4. Prekondisi: "*has an axe*". Efek: "*make firewood*".
c. *collectBranches.* Biaya: 8. Prekondisi: (tidak ada) Efek: "*make firewood*".

World state yang dimiliki agen tersebut adalah "*doesn't have an axe*" dan "*an axe is available*". Berikut adalah GOAP *Planner* di dalam menyusun action sesuai *world state* yang ada.

```
GOAL: "make firewood"
Current State: "doesn't have an axe", "an axe is available"
Can action ChopLog run?
    NO - requires precondition "has an axe"
    Cannot use it now, try another action.
Can action GetAxe run?
    YES, preconditions "an axe is available" and "doesn't have
an axe" are true.
    PUSH action onto queue, update state with action's effect
New State
    "has an axe"
    Remove state "an axe is available" because we just took one.
Can action ChopLog run?
    YES, precondition "has an axe" is true
    PUSH action onto queue, update state with action's effect
New State
    "has an axe", "makes firewood"
We have reached our GOAL of  "makes firewood"
Action sequence: GetAxe -> ChopLog
```

Planner tersebut akan berjalan melalui *action* yang lain juga, dan akan mencari *action* yang memiliki *lowest cost* yang mampu mencapai ke *goal* tersebut. *Tree* yang dibuat *planner* tersebut akan berbentuk seperti ini :



Gambar 2.2. *Tree* yang dibuat GOAP *Planner*

Sumber : Owens, B. (2014, April 23). *Goal Oriented Action Planning for a Smarter AI*. gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793

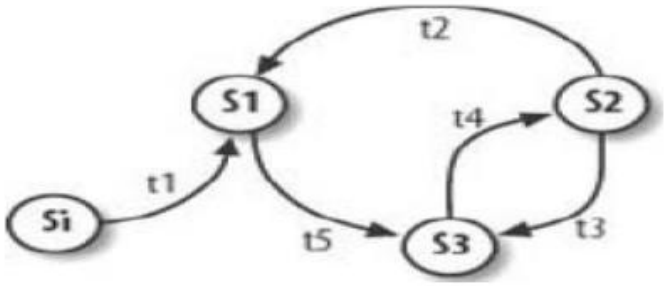
Melalui *planner* tersebut kita dapat melihat bahwa ada tiga rangkaian *action* yang dapat dijalankan, yaitu :

a. *getAxe -> chopLog* (membutuhkan biaya 6)
b. *getAxe -> collectBranches* (membutuhkan biaya 10)
c. *collectBranches* (membutuhkan biaya 8)

Dengan ini, *planner* akan mengambil *action* yang memiliki *cost* yang paling rendah, yaitu *action* pertama (*getAxe -> chopLog)* (Owens, 2014).

2.1.2. Finite State Machine

Finite state machine adalah sebuah metodologi perancangan sistem control yang menggambarkan tingkah laku atau prinsip kerja sistem dengan menggunakan tiga hal berikut: *state, event,* dan *action* (Rahadian et al, 2016). Sistem dapat berpindah dari satu *state* ke *state* lain apabila suatu *event* terjadi. Biasanya transisi antar *state* disertai dengan aksi yang bisa dilakukan, baik aksi yang sederhana maupun kompleks.



Gambar 2.3. Struktur *Finite State Machine*

Sumber : Rostianingsih, S., Budhi, G., S. & Wijaya, H. K. (2013). *Game Simulasi Finite State Machine Untuk Pertanian dan Peternakan*. http://repository.petra.ac.id/id/eprint/16408

Pada gambar 2.3 terdapat 4 *state* {Si, S1, S2, S3} yang mungkin terjadi, setiap *state*-nya dapat berpindah *state* jika kondisi terpenuhi. Sebagai contoh *state* S1 dapat berpindah jika kondisi t5 terpenuhi (Rostianingsih, Budhi, & Wijaya, 2013).

2.1.3. Turn-Based Strategy Games (TBS)

Game yang memiliki *genre turn-based strategy* (TBS) mendorong pemain untuk mengambil keputusan mengenai *action* jenis apa yang perlu dilakukan oleh pion-pion atau *unit-unit* yang dimiliki (Chandra, 2018). Turn-based Strategy Games bersifat putaran, dimana tiap pemain memiliki waktunya sendiri untuk menggerakkan *unit-unit* yang dimiliki oleh pemain tersebut. Tiap pemain memiliki objektif, yaitu mengalahkan musuh dengan *unit-unit* yang mereka miliki.

Gambar 2.4 *Advance Wars: Days of Ruin*

Sumber : Chandra, A. V. (2018). *Perbandingan Performa Turn-based Game Menggunakan Algoritma Genetika dan Logika Fuzzy*.
https://dewey.petra.ac.id/catalog/digital/detail?id=42374

2.1.4. A* Pathfinding

Pathfinding adalah salah satu masalah dasar yang didapatkan ketika membuat *game*. Biasanya, *pathfinding* digunakan untuk menangani ke arah mana karakter harus berjalan untuk mencapai tujuannya. Metode yang digunakan untuk menangani *pathfinding* ada bermacam-macam, salah satunya adalah A* (*A-star*).

A* adalah *generic search algorithm* yang bisa digunakan untuk mencari solusi untuk berbagai masalah, *pathfinding* adalah salah satunya. Metode yang digunakan menunjukkan bahwa cara ini akan mencari suatu *path* dengan *exploring minimum number of nodes* dengan *minimum cost solution*. Karena *simplicity* yang dimilikinya, A* hampir selalu menjadi pilihan *search method*. Ini karena A* menjamin untuk mencari rute terpendek di graf (Mathew, 2015). Menurut Dalem, I., B., G., W., A. (2018), notasi yang dipakai oleh algoritma A* adalah sebagai berikut :

$$f(n) = g(n)+h(n) \quad (2.1)$$

f(n) = biaya estimasi terendah

g(n) = biaya dari *node* awal ke *node* n

h(n) = perkiraan biaya dari *node* n ke *node* akhir

2.2. Tinjauan Studi

2.2.1 ***Applying Goal-Oriented Action Planning to Games*** (Orkin, 2003)

- Masalah yang diangkat pada penelitian tersebut adalah peneliti ingin menganalisa cara kerja agent AI di sebuah *First-person shooter game* bernama *No One Lives Forever 2: A Spy in H.A.R.M.'s Way*, yang menggunakan metode *Goal-Oriented Action Planning.*
- Hasil penelitian menunjukkan bahwa karakter di dalam NOLF2 memiliki *goal* sebanyak 25. Pada suatu waktu, satu *goal* akan aktif, mengontrol tingkah laku karakter. *Goal* di dalam NOLF2 jatuh ke dalam tiga kategori, yaitu *relaxed goals*, *investigative goals*, dan *aggressive goals*. *Relaxed goals* menyangkut *Sleep, work,* dan *patrol*. *Investigative Goals* menyangkut *suspicious investigate* dan *search*. *Aggressive Goals* menyangkut *combat situations*, seperti *chase, charge*, dan *attack from cover*. Meski secara konsep hamper sama, tetapi tujuan di *NOLF2* dan GOAP pada umumnya memiliki satu perbedaan. NOLF2 memiliki *embedded plan*. Di dalam penelitiannya, *search* yang digunakan adalah A* *algorithm*, yang membutuhkan jumlah *cost* dari *node*, dan *heuristic distance* dari sebuah *node* ke sebuah *goal*.
- Perbedaan penelitian yang bersangkutan dengan penelitian ini terletak pada cara meneliti dan dataset yang diteliti. Pada penelitian yang bersangkutan, peneliti hanya menganalisa agent AI yang ada di dalam *game* NOLF2, dan dataset yang digunakan adalah dataset *first-person shooter game*.

2.2.2 *Adaptive Goal Oriented Action Planning for RTS Games* (Magnusson & Hall, 2010)

- Masalah yang diangkat pada penelitian tersebut adalah peneliti ingin mengimplementasi *adaptive goal-oriented* pada AI di dalam sebuah *Real-time strategy game* karena peneliti melihat bahwa AI di dalam *game* RTS itu memiliki "fitur" lebih dan bergerak secara statis melalui *state-machines* dan semacamnya.
- Metode yang digunakan pada penelitian ini adalah metode simple *goal-oriented system*, dengan *adaptive behavior*, yang berarti metode tersebut akan membuat sebuah task yang akan *counter* perilaku musuh.
- AI Ice adalah sebuah AI yang dibuat di dalam penelitian ini. AI Ice membuat rencana-rencana, yang secara praktek sama dengan membuat variasi tugas yang mengarahkan kemenangan kepada dirinya. Untuk melakukan hal tersebut, AI Ice harus mendapatkan informasi di dalam *game* untuk menentukan, membuat, dan menjalankan *tasks* dengan *highest priority*. AI Ice akan selalu membuat *units* berdasarkan tipe senjata dan armor *units* musuh. AI Ice dibuat menggunakan Spring Engine
- Hasil penelitian menunjukkan bahwa adaptive AI yang dibuat oleh peneliti, yang memiliki "fitur" yang sama dengan *human player*, dapat mengalahkan *static* AI yang memiliki "fitur" lebih. Tetapi, prioritas *tasks* harus lebih diperhatikan, dan bisa dibilang akan menjadi susah untuk membuat sebuah prioritas yang bekerja pada "semua" situasi.
- Perbedaan penelitian yang bersangkutan dengan penelitian ini terletak pada dataset yang diteliti. Pada penelitian yang bersangkutan, peneliti menggunakan AI buatannya untuk diuji coba dengan AI yang ada di dalam *game* RTS tersebut, tidak ada pengujian dengan *human player*. Genre *game* yang diteliti juga berbeda.

2.2.3 *Tactical Planning in Space Game using Goal-Oriented Action Planning* (Studiawan, Hariadi, & Sumpeno, 2018)

- Masalah yang diangkat pada penelitian tersebut adalah peneliti ingin membuat sebuah AI di dalam sebuah *game* bergenre *real-time tactics*. Peneliti ingin mencari tahu apakah AI menggunakan *Goal-Oriented Action Planning* dapat digunakan pada *space tactical game*.

- Metode yang digunakan pada penelitian ini adalah metode *Goal-Oriented Action Planning* untuk mengatur perilaku AI yang akan dibuat, berdasarkan *tujuan* yang ingin dicapai dan *state* AI tersebut berada. Penelitian ini juga menggunakan *Blackboard system* untuk memanage *data communication* berdasarkan *blackboard architecture model* dimana *general knowledge base* yang bernama *blackboard* digunakan dan diperbaharui oleh komponen-komponen di dalam sistem. Dengan menggunakan *blackboard system*, *module* di dalam *game* dapat lebih fokus ke fungsi daripada fokus untuk berkomunikasi dengan *module* lainnya.
- Hasil penelitian menunjukkan bahwa di dalam pembuatannya, GOAP membantu mengurangi kompleksitas ketika men-*design* AI. Menggunakan definisi yang ada di dalam penelitian tersebut, komposisi yang dinamis dapat terlihat. Di sisi lain, AI *designer* harus mengevaluasi semua aksi yang berhubungan dengan parameter ketika ada pergantian jenis parameter di dalamnya, dan hal tersebut diakui sebagai salah satu kesulitan menggunakan GOAP.
- Perbedaan penelitian yang bersangkutan dengan penelitian ini terletak pada tujuan penelitian dan dataset yang diteliti. Pada penelitian yang bersangkutan, tujuan penelitian yang dilakukan adalah bagaimana sebuah AI melakukan tugasnya ketika dihadapkan dalam berbagai kasus dan berbagai komponen.

3. ANALISIS DAN DESAIN SISTEM

3.1 Analisis

Di berbagai *game* bergenre *Turn-based Strategy* seperti *Final Fantasy Tactics series*, *player* diberikan *resources* (*unit* pasukan) yang terbatas untuk melakukan *progress*. Pada *series final fantasy* tactics, apabila *player* tidak meningkatkan *level unit* yang dimiliki, mereka tidak dapat melakukan *progress* karena *resources* yang dimiliki oleh musuh jauh lebih besar, sebagai contoh, *level unit* yang dimiliki musuh bisa memiliki perbedaan sangat jauh dengan *level unit* yang dimiliki oleh *player*. Agar *player* bisa melanjutkan petualangannya, *player* diharuskan melakukan aktivitas bernama *grinding*, yaitu melakukan aktivitas yang sama berkali-kali agar *resources* yang dimiliki *player* menjadi banyak. *Grinding* di sini berarti melawan musuh yang sama untuk mendapatkan *level unit* pasukan yang dimiliki *player*. Diharapkan dengan menggunakan AI GOAP, meski *resources* yang dimiliki oleh musuh seimbang dengan *resources* yang dimiliki oleh *player*, AI GOAP dapat memberikan kesulitan yang cukup menantang sehingga *player* tidak perlu melakukan *grinding* untuk mendapatkan *progress* di dalam *genre game* ini.

3.2 Desain Program

Jenis program yang dibuat adalah *game* bergenre *Turn-based Strategy*. *Game* akan dijalankan dengan 2 *mode*, yaitu *mode player* melawan AI GOAP, dan AI FSM melawan AI GOAP. Semua pihak di dalam *game* ini memiliki satu tujuan, yaitu untuk mengalahkan semua *unit* yang musuh punya dengan *unit* yang mereka punya. *Map* yang ada di *game* ini berupa petak atau *tiles* yang berbentuk persegi dan biasanya dikenal dengan nama *grid-based map*. Untuk mencapai tujuan tersebut, *player* maupun AI dapat menggerakkan *unit*nya masing-masing. Sebelum permainan dimulai, tiap kubu dapat memilih *unit* yang digunakan untuk melawan musuh nantinya.

Game ini dibuat dengan tujuan yaitu untuk melakukan penelitian AI mana yang lebih cocok untuk dipakai untuk *Turn-based Strategy*. AI yang akan diuji adalah AI *Finite State Machine* dan AI *Goal-Oriented Action Planning*. Pengujian performa AI akan dilakukan melalui AI FSM melawan AI GOAP, dan kepuasan pemain akan dilakukan melalui *player* melawan AI GOAP, dan hasilnya akan diambil melalui kuesioner. Untuk mendapatkan performa AI masing-masing, akan dilakukan pendataan langkah-langkah yang telah dibuat tiap AI selama pertandingan berjalan.

3.2.1 Elemen *Game*

Game ini memiliki elemen-elemen atau komponen yang sebagian besar sama seperti *game* lainnya yang ber*genre* sama. Elemen-elemen yang ada diantaranya, yaitu :

a. *Map* : *Map* yang digunakan adalah *map* yang berisi petak-petak berbentuk persegi / *tiles*. Ukuran *map* yang digunakan adalah 16 baris x 20 kolom.

b. *Unit* : *Unit* yang digunakan adalah *unit* pasukan (karakter) yang digunakan untuk melakukan penyerangan terhadap *unit* pasukan musuh. *Unit* pasukan memiliki atribut. Atribut-atribut tersebut adalah :

- *Health* : Nyawa yang dimiliki oleh *unit* pasukan tersebut. Jumlah nyawa tergantung pada tipe unit tersebut.
- *Damage* : Kekuatan serangan dari *unit* tersebut untuk mengurangi *unit* pasukan musuh.
- *Move* : Jarak yang digunakan untuk menentukan berapa kotak yang bisa ditempuh *unit* pasukan untuk berpindah *tile*.
- Range : Jarak yang digunakan untuk menentukan berapa jauh kotak yang bisa ditempuh *unit* pasukan untuk menyerang *unit* musuh.
- AP : AP (*Action Point*) digunakan untuk menentukan *action* yang dapat diambil. *Default* AP tiap unit adalah 2. Ketika *command move* digunakan, AP akan di *set* menjadi 1. Jika AP yang dimiliki *unit* lebih dari sama dengan 1, maka *unit* dapat melaksanakan *command attack* atau *move*. Apabila *command attack* dilaksanakan, maka AP menjadi 0.
- Attack : Ada 2 jenis, yaitu Physical dan Magical. Physical menunjukkan serangan *single target* dengan *medium damage*, sedangkan *magical* menunjukkan serangan *area target* dengan *low damage*.
- Type : Tipe *unit*. Ada 2 jenis, yaitu *melee* dan *range*.
- Job : Job yang ada di dalam *game* ini ada 4, yaitu Warrior, Archer, Mage, dan Healer. Warrior memiliki

serangan *single target* jarak dekat, Archer memiliki serangan *single target* jarak jauh, Mage memiliki serangan *area target* jarak jauh, dan Healer berguna untuk menambah *Health friendly unit*.

c. *Actions* : *Command* atau perintah yang bisa diarahkan ke *unit* pasukan untuk melakukan sesuatu sesuai dari tipe *unit*.

Berikut adalah tabel atribut per *unit* berdasarkan tipe yang dimiliki *unit*.

Tabel 3.1

Tabel atribut untuk seluruh tipe *unit*

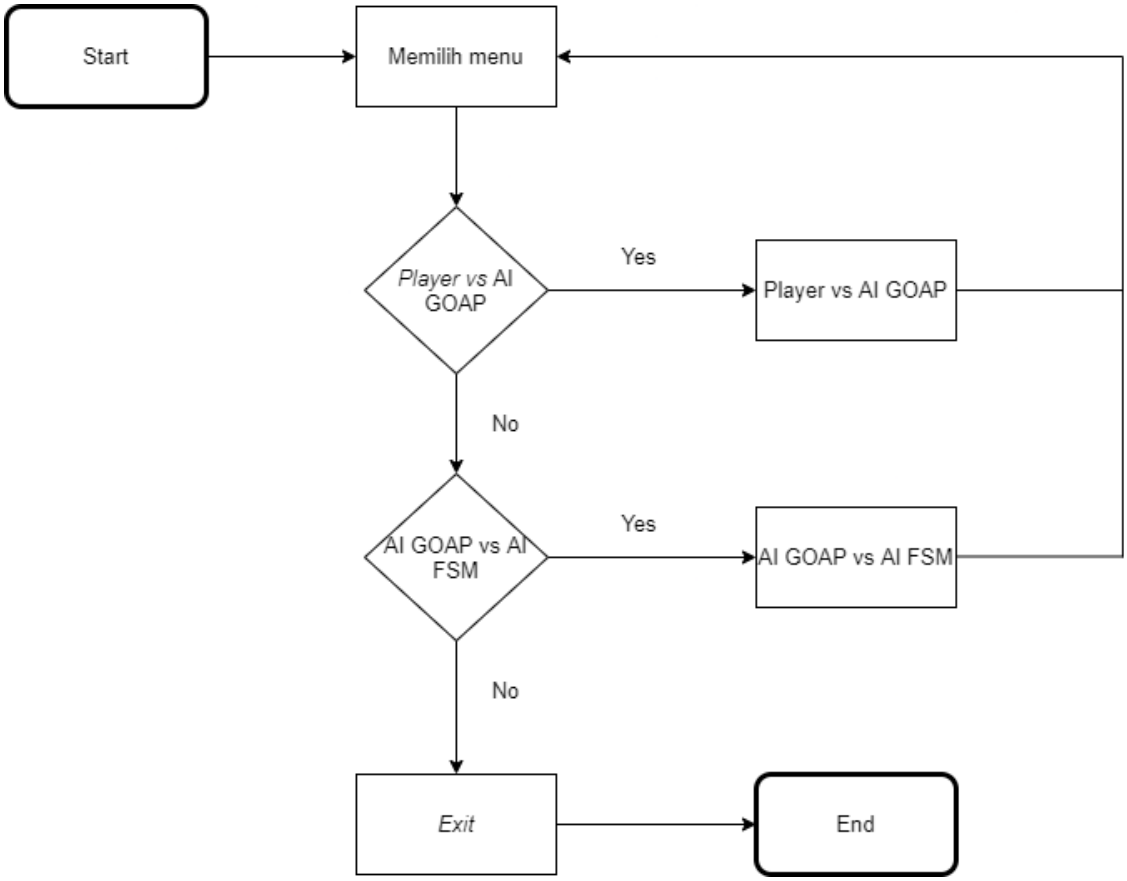
Tipe / Atribut	Health	Damage	Range	Move
Archer	60	30	6	4
Warrior	100	25	1	6
Mage	60	20	3	5
Healer	65	0	4	3

3.2.2 Jenis *Game*

Untuk mendapatkan data yang dibutuhkan dari penelitian ini, maka *game* akan dibagi menjadi 2 jenis, yaitu *player* melawan AI GOAP, dan AI FSM melawan AI GOAP. Di AI FSM melawan AI GOAP, tiap langkah-langkah masing-masing AI akan dicatat sebagai data agar bisa diteliti lebih lanjut.

3.2.3 Desain *Main Menu*

Pada saat pertama kali masuk ke dalam *game*, *user* akan diberikan 3 pilihan, yaitu *player* vs AI, AI FSM vs AI GOAP, dan *exit*. Jika *user* memilih *player* vs AI, maka *user* dapat bermain melawan AI GOAP dengan fitur-fitur yang ada di dalam *game*. Jika *user* memilih AI FSM vs AI GOAP, maka *user* dapat melihat pertandingan antara AI FSM melawan AI GOAP, dan jika *user* memilih *exit*, maka *program* akan otomatis tertutup. Berikut adalah *flowchart* alur *program*.



Gambar 3.1 *Flowchart* menu

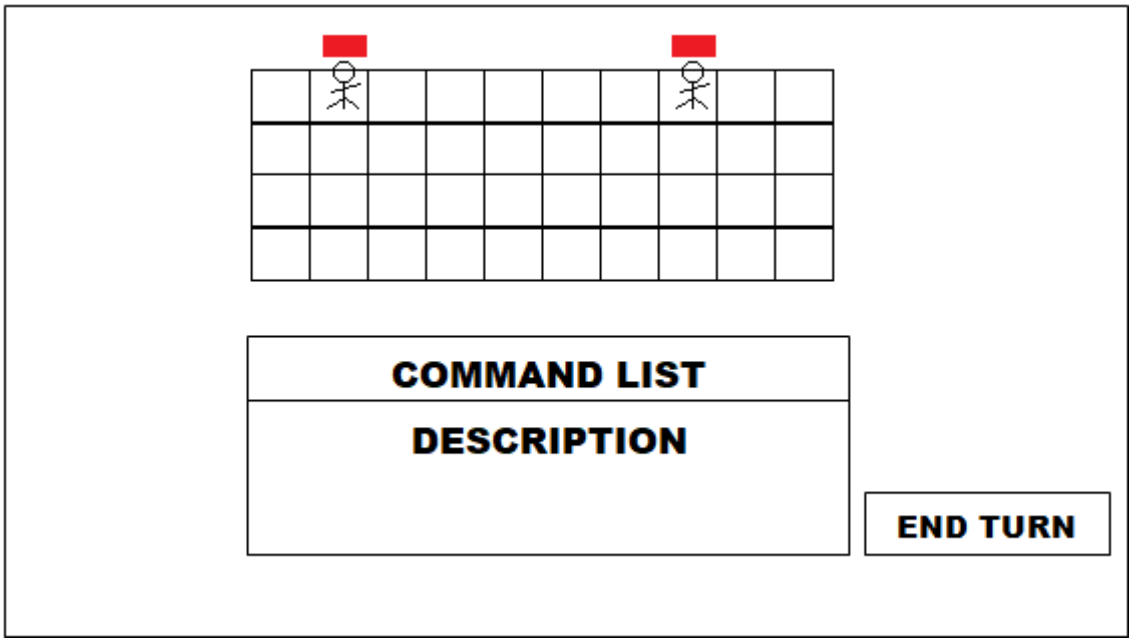
3.2.4 Tampilan Main Menu

Tampilan *main* menu yang ada di dalam program ini terdiri dari text dan *button* untuk memilih menu seperti *Player* vs AI GOAP, AI FSM vs AI GOAP, dan *exit*. Berikut adalah gambar tampilan *menu* secara garis besar.

Gambar 3.2 Tampilan *main* menu

3.2.5 Tampilan Gameplay

Ketika *user* memilih *player* vs AI GOAP, *user* akan diberi tampilan *gameplay*. Di dalam tampilan *gameplay* akan ditampilkan karakter yang akan dimainkan, nyawa karakter yang tersisa, serta di *tile* mana karakter tersebut berada. Di bawah layar, terdapat *window command list* beserta *description* yang berisi *command-command* yang tersedia untuk *user* pakai kepada karakter tersebut. Di bawah kanan, terdapat tombol *End Turn* apabila *user* sudah selesai melakukan gerakannya.



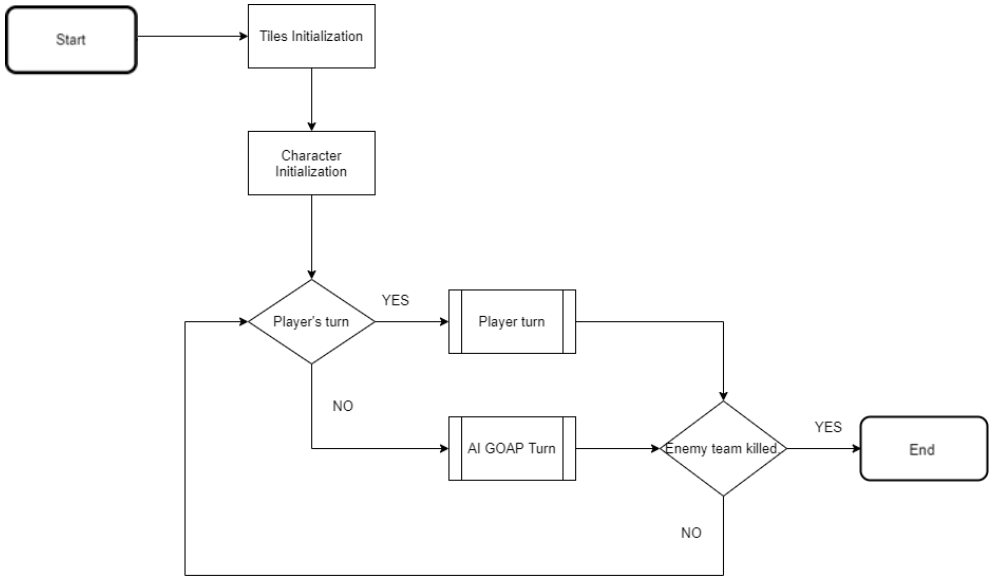
Gambar 3.3 Tampilan *Gameplay*

3.3 Alur *Game*

3.3.1 Alur *Player* vs AI GOAP

Setelah *user* memilih *player vs* AI GOAP di dalam *main* menu, *program* akan melakukan inisialisasi *game*, yaitu *tiles initialization* dan *character initialization*. Setelah inisialisasi selesai, maka sistem mengecek apakah sekarang giliran *player* atau bukan. Jika iya, maka akan masuk ke proses *player turn*. Jika tidak, maka sistem akan memasuki proses AI GOAP *turn*. Ketika *player turn* atau AI GOAP *turn* selesai, maka mereka akan mengecek, apakah *unit* pasukan musuh sudah terbunuh semua atau belum. Jika belum, maka akan kembali mengecek apakah sekarang giliran *player* lagi atau tidak.

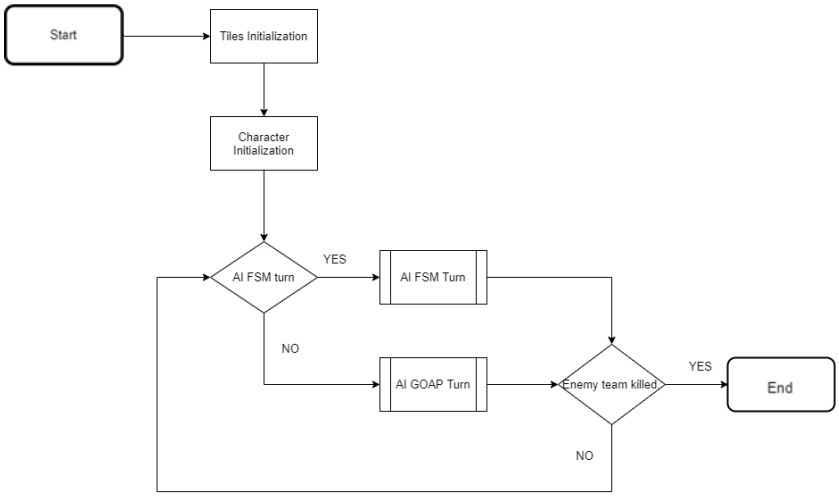
Berikut adalah *flowchart* bagaimana putaran di dalam alur *player* vs AI GOAP berjalan.



Gambar 3.4 Alur di dalam *player* vs AI GOAP

3.3.2 Alur AI FSM vs AI GOAP

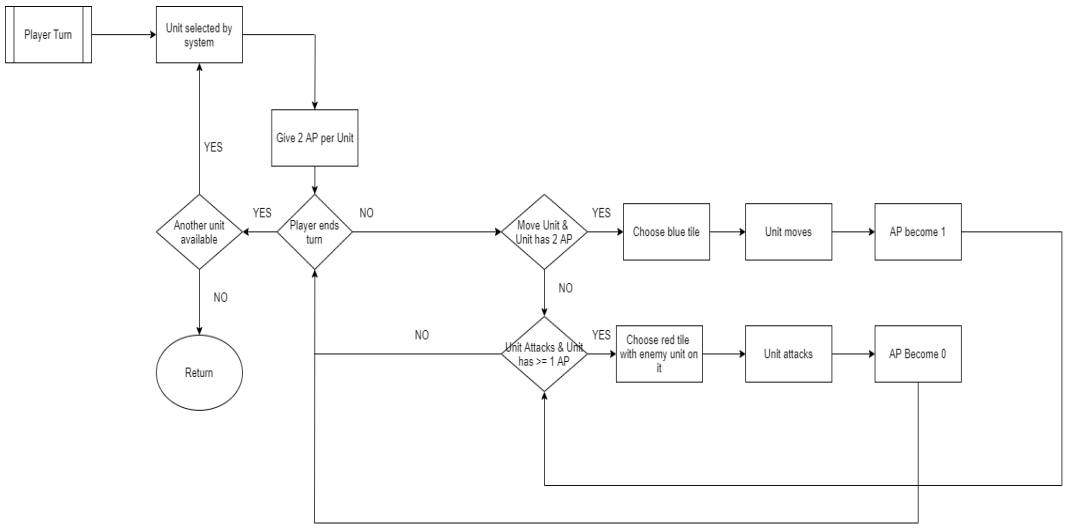
Setelah *user* memilih AI FSM *vs* AI GOAP di dalam *main* menu, *program* akan melakukan inisialisasi *game*, yaitu *tiles initialization* dan *character initialization*. Setelah inisialisasi selesai, maka sistem mengecek apakah sekarang giliran AI FSM atau bukan. Jika iya, maka akan masuk ke proses AI FSM *turn*. Jika tidak, maka sistem akan memasuki proses AI GOAP *turn*. Ketika AI FSM *turn* atau AI GOAP *turn* selesai, maka mereka akan mengecek, apakah *unit* pasukan musuh sudah terbunuh semua atau belum. Jika belum, maka akan kembali mengecek apakah sekarang giliran AI FSM lagi atau tidak. Berikut adalah *flowchart* bagaimana putaran di dalam alur AI FSM vs AI GOAP berjalan.



Gambar 3.5 Alur di dalam AI FSM vs AI GOAP

3.3.3 *Player Turn*

Setelah memasuki *player* vs AI GOAP*,* ketika *game* berada di putaran *player*, sistem akan memilih *unit* yang memiliki speed yang paling cepat diantara semua *unit* pasukan milik *player*. Setelah sistem memilih *unit* pasukan, *unit* pasukan tersebut akan diberikan 2 AP (*Action Point*). *Action points* tersebut berguna untuk menentukan apakah *unit* tersebut dapat melaksanakan *command* yang dipilih atau tidak. Contohnya, untuk melakukan *command move*, *unit* tersebut harus memiliki minimal 2 AP. Apabila *move command* berhasil dijalankan, maka AP *unit* tersebut menjadi 1. Ketika sebuah *unit* memiliki hanya memiliki 1 AP, *unit* tersebut hanya dapat melakukan *command-command* seperti menyerang atau mengeluarkan sebuah *skill*. Ketika *command* menyerang berhasil dijalankan, maka AP dari *unit* tersebut akan menjadi 0. Sebuah *unit* yang memiliki AP 0 hanya bisa melaksanakan perintah *end turn*. Ketika sebuah perintah *end turn* dilaksanakan, sistem akan mengecek, apakah ada *unit* pasukan yang masih bisa bergerak atau tidak. Jika ya, maka sistem akan memilih *unit* pasukan tersebut agar bisa diberi *command* oleh *player*. Jika tidak, maka giliran akan berpindah ke musuh. Berikut adalah *flowchart* tentang bagaimana putaran *player* berjalan.



Gambar 3.6 *Flowchart* putaran *player*

3.3.4 AI GOAP Turn

Setelah memasuki *player* vs AI GOAP*,* ketika *game* berada di putaran AI GOAP, sistem akan memilih *unit* yang memiliki speed yang paling cepat diantara semua *unit* pasukan milik AI GOAP. Setelah sistem memilih *unit* pasukan, *unit* pasukan tersebut akan diberikan 2 AP (*Action Point*). *Action points* tersebut berguna untuk menentukan apakah *unit* tersebut dapat

melaksanakan *command* yang dipilih atau tidak. Contohnya, untuk melakukan *command move*, *unit* tersebut harus memiliki minimal 2 AP. Apabila *move command* berhasil dijalankan, maka AP *unit* tersebut menjadi 1. Ketika sebuah *unit* memiliki hanya memiliki 1 AP, *unit* tersebut hanya dapat melakukan *command-command* seperti menyerang atau mengeluarkan sebuah *skill*. Ketika *command* menyerang berhasil dijalankan, maka AP dari *unit* tersebut akan menjadi 0. Sebuah *unit* yang memiliki AP 0 hanya bisa melaksanakan perintah *end turn*. Ketika sebuah perintah *end turn* dilaksanakan, sistem akan mengecek, apakah ada *unit* pasukan yang masih bisa bergerak atau tidak. Agar AI GOAP dapat melakukan sesuatu, sistem AI GOAP harus mengecek *world state game* tersebut. Contoh dari *world state* tersebut adalah "is not on low health", "attacking target X", "at target X", "no enemies remain" dan "is on low health". Apabila *world state* terpenuhi, maka suatu *action* dapat dilakukan agar *goals* terpenuhi. Berikut adalah tabel-tabel berisi *action* yang dapat dilakukan beserta kebutuhan *world state* dan *goals*.

Tabel 3.2

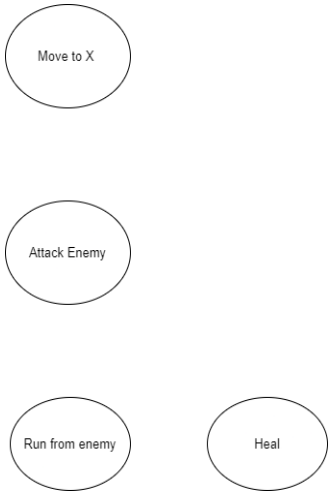
Action yang ada pada AI GOAP

Actions	Satisfies World State	Requires World State
Attack	Attacking target X	At target X Is not on low health
Move to X	At target X Going to target X	Is not on low health
Run from enemy	Run from enemy X	Is on low health
Heal	Is not on low health	Is on low health

Tabel 3.3

Goals yang ada pada AI GOAP

Goals	Desired World State
Kill enemy	Attacking target X
Recover	Run from enemy X Is on low health

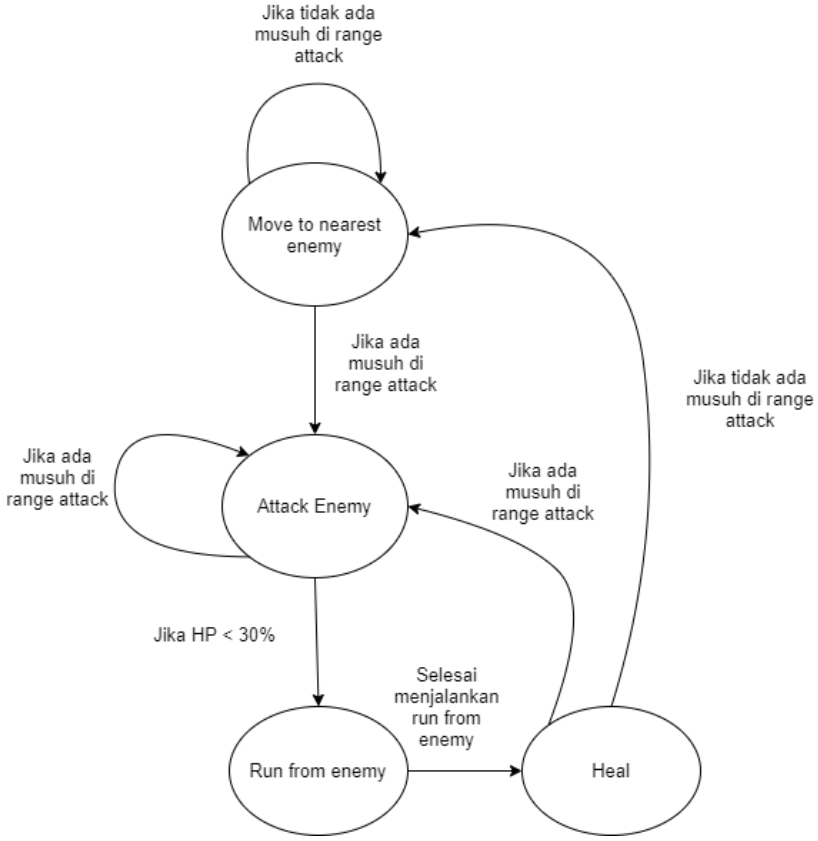


Gambar 3.7 State AI GOAP

Berdasarkan penulisan paper “Tactical Planning in Space Game using Goal-Oriented Action Planning”, state GOAP adalah state FSM yang disederhanakan.

3.3.5 AI FSM Turn

Setelah memasuki AI FSM vs AI GOAP*,* ketika *game* berada di putaran AI FSM, sistem akan memilih *unit* yang memiliki speed yang paling cepat diantara semua *unit* pasukan milik AI FSM. Setelah sistem memilih *unit* pasukan, *unit* pasukan tersebut akan diberikan 2 AP (*Action Point*). *Action points* tersebut berguna untuk menentukan apakah *unit* tersebut dapat melaksanakan *command* yang dipilih atau tidak. Contohnya, untuk melakukan *command move*, *unit* tersebut harus memiliki minimal 2 AP. Apabila *move command* berhasil dijalankan, maka AP *unit* tersebut menjadi 1. Ketika sebuah *unit* memiliki hanya memiliki 1 AP, *unit* tersebut hanya dapat melakukan *command-command* seperti menyerang atau mengeluarkan sebuah *skill*. Ketika *command* menyerang berhasil dijalankan, maka AP dari *unit* tersebut akan menjadi 0. Sebuah *unit* yang memiliki AP 0 hanya bisa melaksanakan perintah *end turn*. Ketika sebuah perintah *end turn* dilaksanakan, sistem akan mengecek, apakah ada *unit* pasukan yang masih bisa bergerak atau tidak. Berikut adalah *state* AI FSM.



Gambar 3.8 *State* dari AI FSM

3.4 Desain Pengujian

Metode pengujian yang digunakan untuk penelitian ini yaitu dengan menggunakan AI FSM untuk melawan AI GOAP, dan membuat *player* melawan AI GOAP. Pengujian AI FSM vs AI GOAP akan dilakukan sebanyak 10 kali dengan pasukan yang berbeda-beda, dengan giliran dimana AI GOAP mendapatkan giliran pertama sebanyak 5 kali, dan AI FSM mendapatkan giliran pertama sebanyak 5 kali. Pada pengujian tersebut, akan dicatat ukuran-ukuran sebagai berikut :

1. Langkah-langkah yang diambil tiap AI.
2. Winrate dari masing-masing AI, kemudian dibandingkan *winrate* ketika mendapatkan giliran pertama dan *winrate* ketika tidak mendapatkan giliran pertama untuk menguji AI mana yang lebih baik untuk *game genre* ini.
3. *Unit* yang dimiliki tiap-tiap AI.

Pengujian *player* vs AI GOAP akan dilakukan sebanyak 10 kali. Pengujian akan dilakukan kepada orang-orang yang memiliki pengalaman berbeda bermain *game genre* ini. Tingkatan pengalaman akan berupa : 1) Tidak pernah bermain sama sekali, 2) Pernah bermain *game*

genre ini, dan 3) Sering bermain *game genre* ini. Tiap orang akan bermain sebanyak 2 kali, dan bisa menggunakan *resources* yang berbeda.