# Perbandingan Performa Turn-Based Game Menggunakan Algoritma Genetika dan Logika *Fuzzy*

Vincent Andrian Chandra, Rolly Intan, Kristo Radion Purba
Program Studi Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jln. Siwalankerto 121-131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) - 8417658
m26414021@john.petra.ac.id, rintan@petra.ac.id, kristo@petra.ac.id

## ABSTRAK

Sudah banyak *game* dengan genre tersebut yang beredar di kalangan umum dan sebagian besar *game* tersebut memiliki mode dimana pemain bisa melawan *Artificial Intelligence* atau disingkat *AI*. Namun, *AI* yang diterapkan dalam *game* dengan genre itu sangat beragam jenisnya. Oleh karena itu penelitian ini diharapkan bisa membantu para *developer game* yang ingin membuat game *Turn-Based Strategy* namun tidak bisa memutuskan untuk memakai *AI* yang mana.

Program berupa *game* yang dibuat memiliki dua *AI* yang bisa berfungsi saat permainan dimulai, yaitu algoritma genetika dan logika *fuzzy*. Terdapat pula 3 mode *game* yang bisa dipilih oleh *user*, yaitu *player vs. AI GA*, *player vs. AI Fuzzy*, dan yang terakhir yaitu *AI GA vs. AI Fuzzy*. Kedua metode tersebut akan dipakai untuk menentukan pasukan jenis apa yang perlu dibuat, pasukan harus menyerang dan bergabung dengan siapa maupun untuk kabur dari musuh.

Dari hasil uji coba pertandingan antara logika *fuzzy* dan algoritma genetika, algoritma genetika lebih bisa membuat pasukan yang beragam namun tetap bisa menyesuaikan dengan situasi yang diperlukan. Logika *fuzzy* juga memakan waktu yang lebih lama dari algoritma genetika dikarenakan jumlah *rules* yang cukup banyak dan algoritma genetika yang memiliki batas untuk pengakhiran proses algoritma genetika sehingga tidak terlalu memakan waktu dalam iterasinya. Namun, apabila *fuzzy rules* dan *membership function* nya dioptimasi lebih lanjut akan menghasilkan *output* yang lebih baik.

**Kata Kunci:** *Fuzzy Inference Rules, Genetic Algorithm, Turn-Based Strategy, Video Game, A* Algorithm*

## *ABSTRACT*

*There are a lot of games with said genre that have been published for public and most of the games have a vs. Artificial Intelligence mode or AI for short. But the variations of the AI for Turn-Based Strategy genre are abundant. That is why hopefully this research will be able to help game developers who would like to make a Turn-Based Strategy game but have no idea which type of AI suits this kind of genre the most.*

*The program, which is a game, that is made containts two AIs which are Genetic Algorithm and Fuzzy Logic. There are also 3 type of game modes that user can choose from, which are player vs. GA AI, player vs. Fuzzy AI, and the last one is GA AI vs. Fuzzy AI. The two AIs will be used to determine what type of unit the AI should make, who to attack and retreat from.*

*From the results of the testcase which are a fight between the two AIs, genetic algorithm is more than able to make different kinds of unit, but it's also capable of producing units that are suitable for the situation. Fuzzy logic consumes more time than genetic algorithm because of the huge amount of rules and genetic algorithm having some kind of threshold to get out of its iteration before it's supposed to end that results in not-so time consuming process for each process. But on the other hand, if the fuzzy rules and its membership functions to be optimized, it's almost guaranteed to produce better results.*

***Keywords:*** *Fuzzy Inference Rules, Genetic Algorithm, Turn-Based Strategy, Video Game, A* Algorithm*

## 1. PENDAHULUAN

Penerapan *Artificial Intelligence* atau disingkat *AI* bisa dilakukan di banyak platform, tidak terkecuali dalam *video game* yang keberadaannya sudah sangat umum. Konsekuensi dari keumuman tersebut yaitu kesempatan bagi *AI, Computational Intelligence,* dan *Machine Learning* dalam *game* untuk memainkan peran yang lebih penting dalam pembuatan *game* komputer dan penyampaian pengalaman yang menarik bagi pemain [7]. Tidak sedikit *game* modern yang memiliki mode agar pemain bisa melawan pemain dan juga bisa melawan *AI*. *Game* yang memiliki genre *Turn-Based Strategy (TBS)* sendiri sangat diminati oleh kalangan muda karena membutuhkan pemikiran yang teliti dan cermat tentang langkah yang harus diambil agar bisa menang. Dalam *game* bergenre *turn-based*, tiap giliran satu pemain bisa mengambil tindakan atau beberapa tindakan [9]. *AI* yang dipakai dalam game bergenre tersebut seringkali memiliki serangkaian perintah yang bisa dilakukan, misalnya saja *move* dan *attack*. Ada beberapa keputusan yang harus diambil oleh *AI* dalam genre *game TBS*, antara lain yaitu penentuan penggerakan *unit* miliknya ke posisi tertentu, penyerangan *unit* milik musuh dan penempatan bangunan yang strategis. Ketiga masalah tersebut bisa diselesaikan dengan banyak cara, terutama dengan logika *fuzzy* ataupun dengan algoritma genetika.

Latar belakang pemilihan kedua algoritma tersebut dikarenakan algoritma genetika sudah cukup umum diterapkan pada bidang *AI* pada *game*, dan begitu juga dengan logika *fuzzy* dimana para *game developer* mulai sering menerapkan *Fuzzy State Machine* (FuSM) pada game bergenre *TBS* yang mereka buat belakangan ini. Kedua metode yang telah dipilih juga secara teori tingkatnya sesuai dengan game dengan genre *TBS* yang sederhana dan tidak terlalu rumit, sehingga tidak dibutuhkan metode lain seperti *neural network* yang biasanya diterapkan pada *game* berskala besar dan dibutuhkan *AI* yang bertingkat tinggi.

Namun, pendekatan dengan masing-masing metode tersebut tentu saja memiliki kelebihan dan kekurangan, sehingga memunculkan pertanyaan metode mana yang lebih menguntungkan dalam beberapa aspek untuk diterapkan pada game dengan genre *TBS*. Oleh karena itu, proposal ini mengajukan penelitian dan perbandingan terhadap performa tiap metode.

# 2. LANDASAN TEORI

## 2.1 Algoritma Genetika

Algoritma genetika adalah algoritma yang didasarkan pada seleksi yang natural, dengan mengkombinasi struktur berupa string yang dirandom namun tetap terstruktur yang paling bisa bertahan hidup [2]. Algoritma genetika juga merupakan algoritma pencarian yang bersifat *random* yang didasarkan dari teori evolusi biologis [6]. Tujuan utama dari algoritma genetika adalah untuk menemukan struktur individu yang paling dianggap sempurna. Bagian-bagian dari algoritma genetika antara lain individu atau kromosom kumpulan balok yang membentuk struktur baru sehingga mirip dengan sebuah *string*, *gen* yaitu satu bagian dari sebuah kromosom, populasi yaitu kumpulan dari beberapa individu / kromosom, fungsi *fitness* yang mengindikasikan seberapa bagus individu dan memperlihatkan kemampuan individu tersebut untuk berkompetisi dengan individu lain, *crossover* yang merupakan proses dimana kedua kromosom yang telah dipilih sebelumnya untuk reproduksi akan ditukar beberapa gennya, dimulai dari index tertentu dan diakhiri pada index yang sudah ditentukan sebelumnya, dan *mutation* yang merupakan proses pergantian isi dari gen dengan probabilitas yang biasanya cukup rendah.

## 2.2 Logika *Fuzzy*

*Crisp* set adalah set yang terdiri dari 0 atau 1 dan tiap *crisp* set merupakan sebuah koleksi elemen yang telah dipilih dari domain, dimana elemen-elemen tersebut memenuhi kriteria tertentu [8]. Sebuah elemen dikatakan tergolong sebagai member dari suatu *Crisp* set atau bukan (A = {0,1}). Sedangkan *Fuzzy* set merupakan set yang memiliki sebagian dari suatu elemen, tidak sepenuhnya (A = [0,1]). Sehingga, elemen tersebut memiliki *membership value* di antara 0 yang tidak termasuk sebagai elemen dan 1 yang termasuk sebagai elemen dari suatu set.

Logika *fuzzy* sendiri merupakan logika yang bertujuan untuk menetapkan suatu definisi dari suatu pemikiran yang tidak pasti atau tidak akurat, yang berbeda dengan logika *classical* dimana logika tersebut memakai pemikiran yang sudah jelas definisinya, yaitu ya atau tidak [1]. Logika *fuzzy* memiliki sistem bernama *Inference System*, dimana sistem tersebut memiliki komponen-komponen yaitu *fuzzy linguistic variables* yang merupakan variabel–variabel yang menampung berbagai elemen dalam bentuk *fuzzy* set, *fuzzy rules* yaitu aturan – aturan (*rules*) yang kerangkanya terdiri dari *antecedent* dan *conesequent* (bagian *IF* dan *THEN* dari *rule), Membership function* yaitu fungsi yang menggambarkan region atau area milik suatu *Fuzzy Linguistic Variable* untuk menandakan dari mana sampai mana berlakunya *FLV* tersebut, yang apabila digambarkan mulai dari *degree of membership* 0 hingga menuju puncak yaitu 1 dan berlaku juga sebaliknya, *Degree of Membership* atau *Membership Value* adalah letak suatu nilai di dalam suatu *membership function*, atau dengan kata lain hasil yang didapat jika nilai tersebut dimasukkan ke dalam *membership function*, *fuzzification* yaitu proses dimana terjadi konversi dari *input* yang berupa *crisp* set yang kemudian diubah menjadi *fuzzy* set dengan cara memasukkan bilangan *crisp* ke dalam *membership function* dan setelah itu didapatkan *degree of membership*, *fuzzy inference* merupakan proses yang akan menghasilkan *fuzzy* set pada konklusi *rule THEN*, *defuzzification* dilakukan setelah proses *fuzzy inference* selesai sehingga didapatkan satu *fuzzy* set, dan *defuzzification* berkebalikan dengan *fuzzification*, atau dengan kata lain *deffuzification* merubah sebuah *fuzzy* set menjadi satu nilai *crisp*.

## 2.3 Turn-Based Strategy Game

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 [4]. Hal tersebut termasuk bagian utama dari *TBS game*, yaitu *managing resources*.

Contoh *game* tradisional yang termasuk dalam genre *TBS* yaitu catur. Untuk memenangkan permainan tersebut dibutuhkan keputusan untuk menjalankan pion yang terbaik dan ke posisi yang paling memungkinkan untuk mencapai kemenangan. Contoh *game TBS* lain yaitu *Civilization* dan *Advance Wars*. Kedua *game* tersebut sama-sama mempunyai *AI* yang melakukan pengambilan keputusan untuk mengerahkan *unit* yang dimiliki untuk menyerang lawan dan membuat bangunan [5].

## 2.4 *Unity* Engine

*Unity* adalah sebuah game engine untuk membuat video game yang menargetkan berbagai macam platform [3]. Selain bersifat multiplatform, misalnya untuk *console*, *desktop* dan *mobile*, *Unity* juga menyediakan fasilitas untuk merancang game yang berbasiskan 2D maupun 3D. Untuk membuat suatu *game*, komponen utama yang berupa algoritma dalam bentuk *script* dengan suatu bahasa pemrograman pasti dibutuhkan, dan animasi untuk objek yang telah dibuat.

Untuk *scripting*, *Unity* menggunakan bahasa pemrograman *C#*, *Javascript* dan *Boo*, namun untuk versi *Unity* yang lebih baru tidak mendukung bahasa *Javascript* dan *Boo* lagi. Dalam hal animasi dan *3D modelling*, *Unity* hanya menyediakan fasilitas untuk membuat animasi yang sangat sederhana, namun untuk hasil yang lebih kompleks dan *modelling* bisa menggunakan program lain seperti *Blender* dan hasilnya bisa diimpor ke dalam *Unity* untuk dipakai menjadi asset.

## 2.5 *Scripting* dan *IDE*

Pada *Unity*, *scripting* menggunakan bahasa pemrograman *C#* untuk *Unity* versi baru, merupakan *Object-Oriented Programming (OOP)*, dan hasil dari *scripting* berupa *script* yang dipasangkan sebagai komponen pada *GameObject* tertentu. Beberapa *Integrated Development Environment (IDE)* yang bisa digunakan untuk melakukan proses scripting yaitu *MonoDevelop* dan *Visual Studio*.

*MonoDevelop* merupakan salah satu *IDE built-in* pada *Unity* yang menggunakan bahasa *C#* sebagai alat menulis *script* untuk *Unity* yang mendukung *cross-platform* antara *Linux*, *Windows*, dan *Mac OS X*. *MonoDevelop* versi ini sudah menjadi satu paket dengan *Unity*, tidak mengikuti versi *MonoDevelop* aslinya agar lebih mendukung.

# 3. DESAIN SISTEM

## 3.1 Desain Program

Program yang dibuat berupa *game* dengan genre *Turn-Based Strategy* yang terdiri dari kedua belah pihak saling bertarung untuk menghancurkan *City HQ* milik musuh masing-masing. *Game* akan berakhir ketika salah satu bangunan tersebut telah

dihancurkan. Satu pihak memiliki warna seragam oranye dan terletak di sebelah kiri bawah *map*, sedangkan yang satu lagi memiliki warna seragam biru dan terletak di sebelah kanan atas *map*. *Map* untuk *game* ini berbentuk persegi panjang dengan panjang dan lebar 16 x 18. Tiap petak-petak dari *map* berupa kubus yang memiliki jenis *terrain* yang berbeda dan memberikan efek *defense points* yang beragam dengan jangkauan 0-3. *Terrain* yang tersedia yaitu *mountain, forest, grass, water,* dan *city*.

Agar bisa menghancurkan *City HQ* milik musuh masing-masing, tiap kubu bisa membuat *unit* yang akan bergerak menuju *City HQ* milik musuh yang memiliki 3 tipe utama, yaitu *troop, vehicle,* dan *building*. Untuk *troop* pasukan yang dibuat berupa *infantry* dan *grenadier* yang rata-rata memiliki jarak perpindahan tempat pendek dan *health points* yang rendah. Untuk *vehicle* pasukan yang dibuat berupa tank, *artillery, APC,* dan *fighter jet* yang rata-rata memiliki jarak perpindahan tempat sedang hingga jauh dan *health points* yang tinggi, kecuali *APC* karena tujuannya hanya mengantarkan *troop* agar bisa ke tempat yang lebih jauh dan *vehicle* tersebut tidak bisa menyerang. Tipe utama yang terakhir yaitu *building* yang terdiri dari *armor tower, barrack, city* dan *sentry turret*. *Armor tower* dan *city* bisa memulihkan *health points* milik *troop* dan *vehicle*, sedangkan *sentry turret* akan menyerang musuh yang berada di dalam jangkauannya. *Barrack* digunakan sebagai pabrik pembuatan *unit*, apabila telah dihancurkan oleh musuh maka pemilik *barrack* yang hancur tidak bisa lagi membuat *unit* baru. Selain memulihkan *health points, city* juga memberi keuntungan berupa distribusi uang kepada pemilik *city* tersebut. Khusus untuk *barrack* dan *city*, kedua bangunan tersebut tidak bisa dibangun sendiri dan sudah tersedia dari awal bersamaan dengan *map*.

Masing-masing *troop* dan *vehicle* yang telah dibuat bisa menjalankan perintah sesuai dengan yang diberikan oleh *user*. Satu perintah yang diberikan *user* hanya akan dilaksanakan oleh *unit* yang telah dipilih sebelumnya, sehingga perintah yang diberikan saat itu tidak bersifat global terhadap semua *unit*. Tiap *unit* bisa menjalankan perintah seperti *move, attack, join, capture* dan *wait. Unit* yang sudah melaksanakan perintah *move* sebelumnya bisa memmilih untuk *attack* atau *capture*, namun jika perintah sebelumnya berupa *attack, join*, atau *capture*, maka status *unit* otomatis akan berubah menjadi *wait*. *Wait* adalah perintah untuk mengakhiri giliran *unit* tersebut, sehingga *unit* yang berstatus *wait* tidak bisa melaksanakan perintah lagi sampai giliran berganti. Perintah *capture* bertujuan untuk menguasai city yang kosong atau milik musuh, sedangkan perintah *join* untuk bergabung dengan *trrop* atau *vehicle* milik kubu dan jenis yang sama. Giliran satu kubu akan berakhir apabila kubu tersebut telah mengakhiri gilirannya dan pada saat pergantian giliran, *building* seperti *city, armor tower,* dan *sentry turret* akan menjalankan tugasnya masing-masing.

## 3.2 Artificial Intelligence

*Artificial Intelligence* yang digunakan ada 3, yaitu algoritma genetika, logika *fuzzy*, dan algoritma *A**. Algoritma *A** dipakai ketika *troop* maupun *vehicle* milik *AI* melaksanakan perintah *move* agar *unit* akan mencari jalan dan memilih petak terdekat apabila tujuannya adalah *City HQ* musuh. Kedua *AI* utama, yaitu algoritma genetika dan logika *fuzzy*, dipakaikan dalam kondisi dan situasi yang sama, yaitu pada saat penentuan tipe *unit* yang terbaik saat proses pembuatan *unit*, pemilihan target petak

perpindahan tempat dan *unit* musuh yang perlu diserang ketika *unit AI* akan menjalankan perintah *attack*, pemilihan target penggabungan ketika *unit AI* akan menjalankan perintah *join*, dan pemilihan target petak perpindahan tempat serta target *unit* musuh yang bisa diserang dari petak tersebut. Keempat situasi tersebut akan dianalisa oleh masing-masing metode dan akan memberikan hasil sesuai dari perhitungan tiap algoritma.

Untuk pemilihan perintah apa yang perlu dijalankan memakai *rule base*, seperti persyaratan untuk melakukan perintah *attack* yaitu apabila ada musuh di dalam jangkauan *unit* milik *AI* dan *health points unit AI* lebih dari 40%, sedangkan untuk perintah *join* yaitu apabila ada musuh di dekat dan juga ada *unit AI* lain yang bisa diajak bergabung, Perintah *retreat* dilakukan hanya jika *unit* tidak memenuhi persyaratan *attack* maupun *join.*

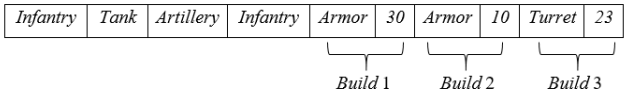
Karena tujuan dari penelitian ini adalah membandingkan antara kedua algoritma tersebut, maka faktor penentu pembuatan keputusan saat menghadapi keempat situasi yang telah dijabarkan sebelumnya pun disamakan. Untuk pembuatan *unit* bertipe *troop* atau *vehicle*, faktor yang mempengaruhi pengambilan keputusan diantaranya yaitu *type advantage* terhadap masing-masing *unit* musuh dan jarak antara spot pembuatan *troop* dengan musuh terdekat. Sedangkan untuk pembuatan *sentry turret* yaitu *health points* milik musuh terdekat dan untuk *armor tower* yaitu jumlah *health points unit* dalam jangkaunnya dan jarak antara spot pembuatan *building* dengan musuh terdekat. Selain itu jumlah uang juga mempengaruhi keputusan yang akan diambil.

Untuk perintah *attack*, faktor-faktornya adalah rasio *health points* antara *unit AI* yang akan menyerang dengan yang akan diserang, *damage points* yang didapat oleh musuh yang akan diserang, dan *range advantage* apabila menyerang target tersebut. Sedangkan perintah *join* dipengaruhi oleh jumlah *health points* setelah penggabungan dan jarak antara *unit* yang diajak bergabung dengan musuh terdekatnya. Perintah *retreat* dipengaruhi oleh *healing points* dan *defense points* yang bisa diterima *unit* yang melakukan *retreat* apabila berpindah ke petak tersebut serta jarak antara *unit* musuh terdekat.

### *3.2.1 Algoritma Genetika*

Langkah algoritma genetika yang dipakai pada *game* ini terdiri dari randomisasi isi kromosom, perhitungan *fitness* masing-masing kromosom dalam populasi, *selection, crossover, mutation,* perhitungan *fitness* baru, dan *elitism*. Dari proses *selection* sampai dengan *elitism* akan diiterasi terus-menerus sampai dengan batas iterasi yang telah ditentukan sebelumnya. Namun, apabila maksimum *fitness* telah stabil selama beberapa generasi, maka iterasi akan dihentikan. Sebelum proses algoritma genetika dimulai, akan ada pengelompokan *unit AI* ke dalam kategori perintah yang akan dilancarkan (*attack, join,* atau *retreat*). Setelah pengelompokan selesai, proses algoritma genetika akan berjalan sesuai masing-masing perintah.

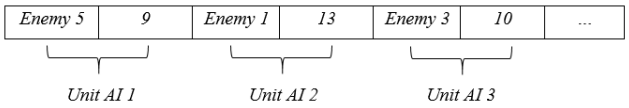
Seperti pada Gambar 1, hasil yang diperoleh saat proses algoritma genetika berjalan dipengaruhi oleh kromosom dan rumus *fitness* yang telah didesain. Untuk kromosom pembuatan *unit*, 4 gen pertama melambangkan *troop* atau *vehicle* yang akan dibuat, dan 3 gen selanjutnya adalah bangunan yang akan dibuat.




**Gambar 1. Desain kromosom *make unit***

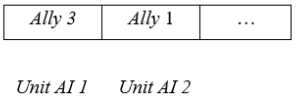
Untuk kromosom perintah *attack*, panjang kromosomnya tergantung dari jumlah *unit AI* yang melancarkan perintah *attack*

dikali 2, karena tiap 2 gen melambangkan 1 *unit AI*. Gen pertama melambangkan musuh yang bisa diserang dan gen kedua melambangkan petak yang mencapai musuh tersebut jika *unit AI* menyerang dari sana. Ilustrasinya bisa dilihat pada Gambar 2.



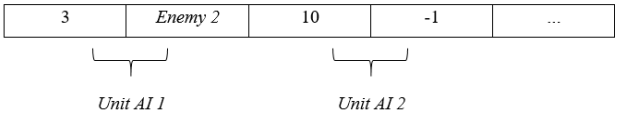

**Gambar 2. Desain kromosom *attack***

Pada Gambar 3, untuk kromosom perintah *join* panjangnya sama dengan jumlah *unit* yang melakukan *join* dan isinya adalah *unit AI* lain yang bisa diajak bergabung.




**Gambar 3. Desain kromosom *join***

Untuk kromosom perintah *retreat*, desain kromosomnya hampir sama dengan *attack*, namun gen pertama adalah petak perpindahan tempat dan gen kedua musuh yang bisa diserang dan bisa dilihat pada Gambar 4. Perbedaannya yaitu musuh yang bisa diserang bisa saja bernilai kosong untuk kromosom *retreat*.




**Gambar 4. Desain kromosom *retreat***

Rumus *fitness* yang digunakan untuk *make troop* mengkalikan *type advantage* (kompabilitas tipe *unit* antara *unit* 1 dengan *unit* lainnya) dan *enemy dist* (jarak dari musuh) dengan konstanta 1.5, sedangkan untuk uang dikalikan dengan konstanta 0.5 agar pembuatan *unit* tidak terlalu terpengaruh oleh uang seperti pada Rumus 1.

$$f = 1.5\,x\left(\frac{type\ adv.}{enemy\ dist.}\right) + (0.5 * money) \quad (1)$$

Pada Rumus 2, perintah *make turret* dan *make armor* memiliki *fitness* yang berbeda, namun faktor selain uang juga tetap dikalikan 1.5 dan hal yang sama diaplikasikan juga pada Rumus 3 yaitu rumus untuk *make armor*. *HP enemy* merupakan presentase nyawa yang dimiliki musuh, *dist* adalah jarak antara *spot* bangun dengan musuh terdekat, dan *HP ally* merupakan presentase nyawa yang dimiliki *unit* teman.

$$f = 1.5\,x\left(\frac{HP\ enemy}{dist}\right) + (0.5 * money) \quad (2)$$

$$f = 1.5\,x\left(\frac{dist}{HP\ ally}\right) + (0.5 * money) \quad (3)$$

Perintah *attack* mengutamakan *attack* yang bisa dilancarkan oleh target *attack* dan rasio *health points* sebaga faktor terpernting kedua seperti pada Rumus 4. *Attack points* merupakan jumlah *damage* yang akan diterima oleh musuh dan *range advantage* merupakan jarak serang dari posisi musuh.

$$f = 3\,x\ attack\ pts. + 2\,x\ HP\ ratio + range\ adv. \quad (4)$$

*Fitness* join lebih mengutamakan total *health points* setelah *join* dilakukan antara 2 *unit AI* yang memiliki tipe sama, kemudian hasilnya ditambahkan dengan *enemy dist* yaitu jarak dari musuh dan bisa dilihat pada Rumus 5.

$$f = 2\,x\ joined\ HP + enemy\ dist. \quad (5)$$

Karena *retreat* lebih mengutamakan keselamatan daripada musuh yang bisa diserang setelah kabur, maka *healing dealt* (jumlah *healing* yang diterima oleh *unit* lain), *enemy distance* (jarak dari musuh terdekat), dan *terrain defense* (*defense points* yan diberikan *terrain*) dikalikan 2, selanjutnya akan dilakukan penambahan hasil dengan *attack points* dan *health points ratio* seoerti pada Rumus 6.

$$f = 2 * (healing\ dealt + enemy\ dist. + terrain\ def.) + attack\ pts. + hp\ rat. \quad (6)$$

### *3.2.2 Logika Fuzzy*

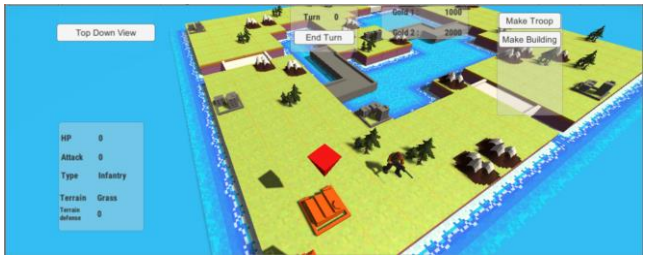
Untuk perintah pembuatan *unit*, logika *fuzzy* mencari presentase tiap *unit* yang kemungkinan bisa dibeli oleh *AI* dengan mendapatkan hasil *defuzzification* dengan metode *centroid*, sehingga *unit* yang memiliki presentase terbesarlah yang akan dibuat. Ketika *unit* hendak melakukan perintah *attack*, ada 2 langkah yang diambil oleh *AI* sebelum menyerang musuh. Langkah pertama yaitu penentuan petak yang bisa mencapai musuh dan langkah kedua yaitu penentuan target *attack* itu sendiri. *Unit AI* akan mengambil pasangan petak dan musuh yang memiliki nilai *centroid* paling besar.

Perintah *join* cukup sederhana karena presentase yang didapat berasal dari semua *ally* (*unit* yang memiliki tipe yang sama dengan *unit AI* yang ingin bergabung) dalam jangkauan *unit AI* yang akan melaksanakan perintah *join*. *Ally* dengan presentase terbesarlah yang akan menjadi sasaran *unit AI* untuk dilakukan *join*. Perintah *retreat* mirip dengan saat *AI Fuzzy* melaksanakan perintah *attack*, hanya saja pada *retreat AI* lebih mengutamakan keamanan daripada penyerangan *unit* musuh. Dari semua petak yang bisa dilangkahi oleh *unit AI*, satu persatu petak tersebut akan dihitung presentase keamanannya dengan *fuzzy rules* yang telah dibuat sebelumnya. Setelah petak yang presentasenya paling besar didapat, dari petak tersebut akan dicari kemungkinan *unit* musuh mana saja yang bisa diserang. Target penyerangan yang akan dituju oleh *unit AI* adalah *unit* musuh yang memiliki hasil *centroid* paling besar.

## 4. PENGUJIAN SISTEM

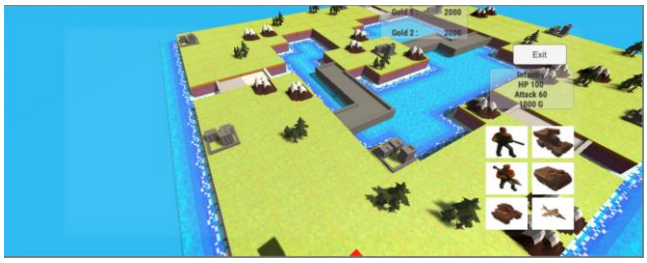
### 4.1 Implementasi Desain *Interface*

Pada Gambar 5, apabila *user* memilih mode *player* melawan *AI*, maka tampilan *map* akan ditampilkan di layar. Pada kanan atas layar terdapat pilihan perintah yang bisa dijalankan oleh *unit* yang dipilih dengan klik kiri tombol *mouse*, sedangkan pada kiri bawah terdapat status *unit* yang dipilih dengan klik kanan tombol *mouse*. Panah yang muncul berperan sebagai indikator *unit* yang dipilih menggunakan klik kiri tombol *mouse*. Tombol *top down view* akan merubah kamera menghadap ke bawah dan tombol *end turn* akan mengganti giliran ke kubu selanjutnya.




**Gambar 5. Tampilan ketika *game* dimulai**

Layar pembelian *unit* akan muncul di kanan layar jika pemain memilih untuk membeli *unit* seperti pada Gambar 6. Di atas gambar *unit* terdapat deskripsi berupa *health points*, *attack*, dan harga apabila *user* melakukan *hover* di dalam gambar.




**Gambar 6. Tampilan ketika membeli *troop***

## 4.2 Pengujian Pertarungan Antara GA dan *Fuzzy*

Pengujian kedua *AI* dilakukan dengan cara mempertandingkan keduanya sebanyak 8 kali. Pada uji coba ke-1 sampai dengan ke -4 *AI* algoritma genetika mendapat giliran pertama, sedangkan pada uji coba ke-5 sampai dengan ke-8 *AI fuzzy* yang maju pertama kali. Hasil uji coba yang dicatat adalah jumlah dan tipe *unit* yang dibuat oleh masing-masing *AI* seperti hasil yang ditunjukkan pada Tabel 1. Begitu halnya dengan pada Tabel 2, jumlah *unit* yang dibuat ditandai dengan simbol M (*made*) dan yang mati ditandai dengan simbol D (*dead*). Baris yang paling bawah merupakan total *unit* dari keseluruhan uji coba.

**Tabel 1. Tabel *unit* yang dibuat dan mati milik *AI GA***

| | Infantry | | Grenadier | | Tank | | Artillery | | Jet | | Builder | | Armor | | Turret | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M | D | M | D | M | D | M | D | M | D | M | D | M | D | M | D |
| 1 | 5 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 4 | 3 | 5 | 1 | 3 | 1 | 2 | 2 | 7 | 4 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 3 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 3 | 3 | 1 | 0 | 4 | 0 | 0 | 0 | 6 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 9 | 8 | 3 | 2 | 3 | 3 | 0 | 0 | 3 | 3 | 4 | 0 | 2 | 0 | 2 | 1 |
| 7 | 5 | 5 | 8 | 6 | 2 | 2 | 0 | 0 | 2 | 2 | 3 | 0 | 1 | 0 | 2 | 1 |
| 8 | 2 | 2 | 0 | 0 | 1 | 1 | 2 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 33 | 28 | 18 | 10 | 14 | 7 | 6 | 3 | 27 | 21 | 8 | 0 | 3 | 0 | 5 | 2 |

**Tabel 2. Tabel *unit* yang dibuat dan mati milik *AI Fuzzy***

| | Infantry | | Grenadier | | Tank | | Artillery | | Jet | | Builder | | Armor | | Turret | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M | D | M | D | M | D | M | D | M | D | M | D | M | D | M | D |
| 1 | 4 | 3 | 3 | 2 | 2 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 4 | 4 | 14 | 11 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 3 | 2 | 1 | 1 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 2 | 1 | 1 | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 9 | 6 | 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 12 | 9 | 13 | 5 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 11 | 9 | 8 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 9 | 4 | 8 | 2 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| | 13 | 10 | 60 | 44 | 43 | 11 | 11 | 5 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

Selain dicatat kedua hal tersebut, pada Tabel 3 juga diperlihatkan pemenang dari pertandingan ke sekian dan rata-rata memori yang terpakai juga ikut didata. Dari hasil pada tabel tersebut algoritma genetika menang sebanyak 3 kali dan semuanya ketika ia maju pertama kali, sedangkan *fuzzy* menang 1 kali saat ia maju kedua dan pada saat maju pertama kali *fuzzy* memenangkan keempat pertandingannya.

**Tabel 3. Tabel kemenangan *AI* dan rata-rata memori**

| Percobaan ke - | Algoritma Genetika | Logika *fuzzy* | Rata-rata Memori |
|---|---|---|---|
| 1 | Menang | Kalah | 167 MB |
| 2 | Menang | Kalah | 158 MB |
| 3 | Kalah | Menang | 160 MB |
| 4 | Menang | Kalah | 167 MB |
| 5 | Kalah | Menang | 164 MB |
| 6 | Kalah | Menang | 165 MB |
| 7 | Kalah | Menang | 161 MB |
| 8 | Kalah | Menang | 169 MB |

## 4.3 Pengujian Pembuatan *Unit*

Pengujian untuk pembuatan *unit* yang diterapkan kepada kedua *AI* dilakukan sebanyak 5 kali dengan masing-masing kondisi *unit* milik musuh yang berbeda. Untuk pengujian ke-1, 4 dan 5 kedua *AI* membuat *unit* yang sama persis, sedangkan perbedaannya terletak pada pengujian ke- 2 dimana *unit* musuh berada di dekat spot *spawn* dan *AI GA* membuat *turret* namun *AI Fuzzy* membuat *artillery*. Begitu pula dengan perocbaan ke-3 dimana musuh mempunyai 1 *fighter jet*, 1 *infantry* dan 1 *grenadier*, *AI GA* membuat 1 *artillery* dan 2 *infantry*, sedangkan *AI Fuzzy* membuat 1 *fighter jet*. Waktu yang diperlukan untuk membuat *unit* pada masing-masing *testcase* juga dicatat pada Tabel 4.

**Tabel 4. Tabel waktu eksekusi untuk perintah *make unit***

| | GA | Fuzzy |
|---|---|---|
| 1 | 0.1363044 | 0.09012699 |
| 2 | 0.2301884 | 0.3821297 |
| 3 | 0.29812695 | 0.4297733 |
| 4 | 0.2806547 | 0.7921388 |
| 5 | 0.3070932 | 0.8978726 |

## 4.4 Pengujian Pemilihan Target untuk *Attack*

Untuk pengujian pemilihan target penyerangan dilakukan sebanyak 4 kali dan uji coba ke-2, 3 dan 4 hasil untuk kedua *AI* sama, yaitu target penyerangannya tidak berbeda. Pada pengujian pertama, *unit artillery* milik *AI GA* menyerang musuh *infantry* yang memiliki *health points* 10. Sedangkan *artillery* milik *AI Fuzzy* menyerang musuh *infantry* yang memiliki *health points 60* dan *defense points* yang sama dengan saat pengujian *artillery* milik *AI GA*. Waktu yang diperlukan untuk menentukan target *attack* dicatat untuk masing-masing metode dan ditampilkan pada Tabel 5.

**Tabel 5. Tabel waktu eksekusi untuk perintah *attack***

| | GA | Fuzzy |
|---|---|---|
| 1 | 0.110569 | 0.4203438 |
| 2 | 0.107688 | 0.2641444 |
| 3 | 0.1064224 | 0.2276373 |
| 4 | 0.1072907 | 1.343616 |

## 4.5 Pengujian Pemilihan Target untuk *Join*

Pengujian pemilihan target *join* dilakukan sebanyak 3 kali dan dari 3 kali uji coba tersebut hasil yang diproduksi saat menguji menggunakan kedua metode sama. Waktu yang diperlukan untuk menentukan target *join* dicatat untuk kedua metode pada Tabel 6.

**Tabel 6. Tabel waktu eksekusi untuk perintah *join***

| | GA | Fuzzy |
|---|---|---|
| 1 | 0.1793966 | 0.04303455 |
| 2 | 0.1154485 | 0.03110933 |
| 3 | 0.1139383 | 0.02980328 |

## 4.6 Pengujian Pemilihan Target untuk *Retreat*

Pengujian untuk perintah *retreat* dilakukan sebanyak 5 kali dan dari kelima pengujian tersebut terdapat 2 uji coba yang hasilnya berbeda antara kedua metode, salah satunya yaitu uji cobe ke-1 dimana *AI GA* memilih untuk berpindah ke pegunungan berjarak 2 dan kemudian menyerang musuh, sedangkan *AI Fuzzy* berpindah ke hutan berjarak 3 saja. Pada uji coba ke-5 *AI GA* berpindah tempat ke pegunungan dengan jarak 4, sedangkan untuk *AI Fuzzy* berpindah tempat ke pegunungan dengan jarak 3. Waktu yang ditempuh untuk menentukan target *retreat* dicatat untuk masing-masing metode seperti pada Tabel 7.

**Tabel 7. Tabel waktu eksekusi untuk perintah *retreat***

| | GA | Fuzzy |
|---|---|---|
| 1 | 0.1436481 | 0.3754439 |
| 2 | 0.2123404 | 0.4526143 |
| 3 | 0.1590271 | 0.4042253 |
| 4 | 0.1605964 | 0.3255949 |
| 5 | 0.1637759 | 0.3566241 |

# 5. KESIMPULAN

Dari hasil analisa saat implementasi dan pengujian program, ditarik beberapa kesimpulan sebagai berikut:

- Waktu yang dilewati saat melakukan perintah dengan menggunakan metode logika *fuzzy* lebih lama, terutama pada perintah *attack* karena di dalam *fuzzy inference* untuk *attack* terdapat looping yang memutari semua kombinasi petak dan musuh yang tersedia pada unit tersebut.
- Ada kemungkinan beberapa petak terlewatkan oleh pengecekan algoritma genetika apabila *unit* nya memiliki *move range* besar, karena jumlah kromosom yang mencakup perpindahan tempat hanya 40 buah, sedangkan ada *unit* yang bisa berpindah tempat sampai dengan 85 kemungkinan tempat.
- Algoritma genetika lebih bisa membuat jenis *unit* yang beragam, sedangkan *fuzzy* membuat *unit* yang sebagian besar jenisnya sama. Hal ini dikarenakan faktor *balancing* dari *game* itu sendiri dan juga *fuzzy rules* yang terlalu sempit jangkauannya, sehingga berpengaruh juga pada pengujian perpindahan tempat saat *retreat*.

# 6. DAFTAR REFERENSI


[1] Chen, G., Trung T. P. 2001. *Introduction to fuzzy sets, fuzzy logic and fuzzy control systems.* Florida: CRC Press LLC

[2] Goldberg, D.E. 1989. *Genetic algorithm in search, optimization, and machine learning*. Reading, MA: Addison-Weasley

[3] Hocking, J., Schell, J. 2015. *Unity in action*: *Multiplatform game development in C# with Unity 5.* Manning Publications

[4] Novac, J. 2012. *Game development essentials: An introduction, third edition.* Delmar: Cengage Learning

[5] Pirovano, M. 2012, December 7. *The use of fuzzy logic for artificial intelligence in games.* Milano, Italy: Department of Computer Science, University of Milano

[6] Purba, K.R., Liliana, Johan, P. 2016. *Optimization of units movement in turn-based strategy game.* Surabaya, Indonesia: Universitas Kristen Petra

[7] Riedl, M.O., Zook, A. 2013. *AI for game production*

[8] Shinghal, R. 2013. *Introduction to fuzzy logic.* Delhi: PHI Learning Private Limited

[9] Yannakakis, G.N., Togelius J. 2018. *Artificial intelligence and games*. New York City: Springer