Penerapan Algoritma Particle Swarm Optimization (PSO) untuk Optimisasi Pembangunan Negara dalam Turn Based Strategy Game

Andre Setiawan[1], Leo Wiliyanto Santoso[2], Rudy Adipranata[3]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 - 131 Surabaya 60236
Telp. (031) - 2983455, Fax. (031) - 8417658
Email : setiawanandre06@gmail.com[1], leow@peter.petra.ac.id[2], rudya@peter.petra.ac.id[3]

ABSTRAK

Bermain *Video Game* merupakan salah satu hiburan yang menjadi pilihan untuk mengisi waktu luang. Salah satu *genre game* yang cukup digemari oleh pemain adalah *Turn Based Strategy. Game Turn Based Strategy* memerlukan *AI* yang dapat membuat keputusan dengan optimal. Pada penelitian ini, *AI* dapat membuat keputusan untuk menentukan jenis dan lokasi bangunan, sehingga pembangunan menjadi lebih optimal.

Untuk dapat mengambil keputusan dengan optimal, dibuatlah sebuah sistem algoritma *AI*. Sistem algoritma *AI* yang digunakan adalah *Particle Swarm Optimization* dalam mengambil keputusan untuk menentukan jenis dan lokasi bangunan.

Hasil pengujian menunjukkan bahwa algoritma *Particle Swarm Optimization* dapat dikembangkan untuk dapat menentukan keputusan dengan baik berdasarkan *fitness* yang berpengaruh pada penentuan jenis dan lokasi bangunan. Hasil pengujian juga menunjukkan bahwa dengan batas jumlah turn yang mencukupi, *Particle Swarm Optimization* dapat memberikan hasil yang lebih baik daripada metode random.

Kata Kunci: *Kecerdasan Buatan, Particle Swarm Optimization dan Turn Based Strategy*

ABSTRACT

Playing Video Games is one of the entertainment choices to fill your free time. One genre of games that is quite popular with players is Turn Based Strategy. Turn Based Strategy requires an AI that can make optimal decisions. In this study, AI can make decisions to determine the type and location of buildings, so that development becomes more optimal.

To be able to make optimal decisions, an AI algorithm system is created. The AI algorithm system used is Particle Swarm Optimization in making decisions to determine the type and location of buildings.

The test results show that the Particle Swarm Optimization algorithm can be developed to be able to determine decisions well based on fitness which has an effect on determining the type and location of buildings. The test results also show that with sufficient number of turn, Particle Swarm Optimization can make better decisions than random methods.

Keywords: *Artificial Intelligence, Particle Swarm Optimization and Turn Based Strategy.*

1. INTRODUCTION

Game merupakan salah satu hiburan yang menjadi pilihan untuk mengisi waktu luang. *Video Game* sendiri terbagi mejadi beberapa genre, seperti *Action, RPG, Shooter, Racing, Fighting, RTS*, dll. Salah satu genre Video Game yang banyak digemari oleh berbagai kalangan adalah *Turn Based Strategy (TBS). Turn based strategy (TBS) game* merupakan permainan strategi (biasanya jenis permainan perang) di mana pemain bermain bergiliran mengatur strategi sesuai keinginannya dan

akan berganti giliran setelah pemain selesai. Salah satu hal yang menarik dari *genre Turn based strategy (TBS)* adalah kita dituntut untuk berpikir secara matang mengenai keputusan yang kita ambil, semisal apakah kita akan lebih memusatkan untuk pembangunan ekonomi atau sarana militer, serta merencanakan langkah selanjutnya yang harus diambil.

Salah satu aspek yang penting dari *game Turn Based Strategy (TBS)* adalah adanya lawan / musuh. Untuk dapat mengontrol lawan dalam *game*, tentunya butuh kecerdasan buatan / *Artificial Intelligence (AI). AI* merupakan salah satu fitur utama di dalam *game Turn Based Strategy (TBS)*, karena tanpa *AI*, permainan akan terasa hambar. *AI* yang baik tentu saja dapat merencanakan strategi, melakukan tindakan sesuai kondisi di dalam *game*, serta memberikan tantangan bagi pemain, seperti saat bermain dengan pemain lain.

Terdapat penelitian sebelumnya pada paper dengan judul "*Optimization in Strategy Games:Using Genetic Algorithms to Optimize City Development in FreeCiv*" (Azhar, 2005). Paper ini menjelaskan mengenai penerapan algoritma genetika untuk optimasi pembangunan kota pada *game FreeCiv. Particle Swarm Optimization (PSO)* sendiri memiliki kesamaan dengan algoritma genetika. *PSO* dan algoritma genetika dimulai dengan *generate* populasi secara random, menghitung nilai *fitness* untuk mengevaluasi populasi dan melakukan *update* populasi berdasarkan nilai *fitness* yang didapat. Perbedaan utama *PSO* dengan algoritma genetika adalah *PSO* tidak memiliki proses *crossover* dan mutasi seperti pada algoritma genetika, sehingga proses algoritma *PSO* lebih sederhana dibanding algoritma genetika.

Penelitian ini bertujuan untuk *membuat game Turn Based Strategy (TBS)* dengan *AI* yang memadai di dalamnya. *AI* dalam *game* ini akan menggunakan algoritma *Particle Swarm Optimization (PSO)* untuk mengoptimasi pembangunan negara lawan..

2. LANDASAN TEORI

2.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) adalah algoritma pencarian yang menggunakan banyak individu, atau partikel, dikelompokkan dalam segerombolan. Masing-masing partikel ini mewakili solusi kandidat untuk optimasi masalah[4]. Secara konsep, penggunaan pbest dan gbest pada particle swarm mirip dengan operasi crossover pada *Genetic Algorithm (GA). Particle Swarm Optimization (PSO)* juga menggunakan konsep nilai *fitness*, sesuai dengan paradigma komputasi evolusioner[7]. Langkah-langkah algoritma *Particle Swarm Optimization (PSO)* dalam penyelesaian suatu masalah adalah sebagai berikut:

1.Menentukan jumlah partikel yang akan digunakan
2.Menentukan posisi dan kecepatan partikel secara random
3.Evaluasi nilai fitness menggunakan rumus yang telah ditentukan sebelumnya dari masing-masing partikel berdasarkan posisinya
4.Tentukan partikel dengan *fitness* terbaik untuk dijadikan sebagai Gbest
5.Pbest awal sama dengan posisi awal
6.Menggunakan *Pbest* dan *Gbest* yang ada, perbarui kecepatan partikel dengan rumus berikut :

$$V_i(t) = V_i(t-1)C_1R_1\left(X_i^L - X_i(t-1)\right) + C_2R_2(X^G - X_i(t-1))$$

Keterangan :
V = kecepatan partikel
Vi = kecepatan partikel pada suatu indeks
t = iterasi ke-t
i = indeks partikel
V = kecepatan partikel
X = posisi partikel
R1 dan R2 = nilai *random* dengan range antara 0 sampai 1
C1 dan C2 = konstanta yg bernilai positif yang biasanya disebut sebagai *learning factor*
XL = local best dari suatu partikel
XG = global best dari seluruh kawanan

7.Perbarui posisi masing-masing partikel dengan rumus berikut :

$$X_i(t) = V_i(t) + X_i(t-1)$$

8.Evaluasi kembali nilai *fitness* dari tiap partikel
9.Tentukan partikel dengan fitness terbaik untuk dijadikan sebagai *GBest*. B Untuk tiap partikel, tentukan *PBest* dengan membandingkan posisi sekarang dengan *PBest* pada iterasi sebelumnya
10.Cek apakah semua posisi partikel konvergen. Jika ya, berhenti. Partikel dinyatakan konvergen apabila semua posisi partikel berada posisi yang sama atau sangat berdekatan dengan partikel gBest dan tidak memiliki perkembangan posisi pada iterasi berikutnya. Posisi partikel yang konvergen menandakan bahwa solusi telah ditemukan.

2.2. Strategy Game

Strategy Game merupakan permainan yang mengandalkan pemikiran dan perencanaan yang baik untuk dapat memenangkan permainan. serangkaian *action* terhadap satu musuh atau lebih dan berusaha melemahkan lawan dengan seefisien dan secepat mungkin. Sebagian besar dari *strategy game* melibatkan unsur peperangan, dan mengutamakan kombinasi dari pertimbangan taktik dan strategi Pemain harus membuat rencana untuk *Game* strategi sendiri terbagi menjadi 2 subgenre, yaitu *Turn-Based Strategy* dan *Real-Time Strategy*. *Turn-Based Strategy* merupakan subgenre dari *strategy game* dimana *player* hanya dapat menjalankan strateginya ketika *player* mendapat giliran sebelum giliran berpindah kepada *player* berikutnya. Kelebihan dari *Turn-Based Strategy* adalah umumnya *player* tidak diberikan batasan waktu untuk mengatur strateginya. Kalaupun diberi waktu, umumnya waktu yang diberikan cukup lama, sehingga player dapat memikirkan strateginya secara matang. *Real-Time Strategy* merupakan kebalikan dari *Turn-Based Strategy* dimana permainan berjalan tanpa giliran, sehingga *player* harus berpikir cepat untuk merencanakan strateginya[5].

2.3. A* Pathfinding

A* adalah algoritma pencarian yang dapat digunakan untuk menemukan solusi untuk banyak masalah, *pathfinding* merupakan salah satunya. Algoritma ini menyatukan fitur *uniform-cost search* dan *heuristic search*. Untuk pathfinding, algoritma A * berulang kali memeriksa lokasi yang belum dijelajahi yang paling menjanjikan yang dimilikinya terlihat. Ketika suatu lokasi dieksplorasi, proses algoritma dinyatakan selesai jika lokasi itu adalah tujuannya; jika tidak, algoritma A* akan mengingat atau mencatat lokasi sekitar agar dapat menjelajah lokasi lebih jauh. A * mungkin merupakan algoritma *pathfinding* yang paling populer dalam *game AI* (Kecerdasan Buatan)[2].

2.4. Perbedaan Particle Swarm Optimization dan Genetic Algorithm

Particle Swarm Optimization (PSO) dan *Genetic Algorithm (GA)* memiliki beberapa persamaan. Baik *PSO* maupun *GA* termasuk ke dalam komputasi evolusioner. Komputasi evolusioner merupakan sistem pencarian solusi yang menggunakan model komputasi dari proses evolusi. Beberapa persamaan proses pada *PSO* maupun *GA* yaitu diawali dengan melakukan generate populasi secara *random*, menghitung nilai fitness untuk melakukan evaluasi terhadap masing-masing partikel dan melakukan *update* populasi berdasarkan nilai *fitness* yang didapat.

Selain persamaan, *PSO* dan *GA* juga memiliki beberapa perbedaan. Perbedaan yang pertama yaitu *GA* memiliki proses *crossover* dan mutasi, sedangkan *PSO* tidak memiliki proses tersebut[6]. Sehingga proses dari *PSO* lebih sederhana dibanding *GA*. Perbedaan yang kedua adalah *Genetic Algorithm* bersifat diskrit, yaitu mengubah variabel menjadi biner 0 dan 1, sehingga dapat dengan mudah menyelesaikan masalah yang bersifat diskrit, sedangkan variabel dalam *PSO* dapat mengambil nilai berdasarkan posisi mereka saat ini di ruang partikel[3].

2.5. Video Game

Video Game merupakan sebuah permainan elektronik yang memerlukan interaksi degan *user interface* untuk menghasilkan *feedback* secara visual, seperti layar *TV* maupun monitor komputer. *Video Game* juga merupakan sebuah cara untuk menghilangkan kebosanan dan juga memiliki beberapa kelebihan yang lain, seperti perkembangan otak, meningkatkan konsentrasi, dan lain-lain[1].

3. DESAIN SISTEM

3.1 Deskripsi Permainan

Tujuan dari permainan ini adalah dengan menyerang kota-kota musuh dengan mengerahkan unit-unit yang tersedia sambil berusaha untuk membangun unit dan bangunan pada kota pemain

secara bergantian. Jika pemain sukses menyerang pusat kota lawan, maka kota tersebut akan hancur, beserta dengan seluruh bangunan-bangunan yang ada. Tujuan akhir dari permainan ini adalah untuk menghancurkan ibu kota lawan. Jika ibu kota lawan hancur, maka pemain yang berhasil menghancurkan pusat ibu kota dinyatakan sebagai pemenang.

Peraturan dari permainan ini adalah setiap *tile* hanya boleh ditempati oleh 1 bangunan, 1 *unit military* dan 1 *unit civilian*. Bangunan dan unit dapat menempati 1 tile secara bersamaan. Karena genre game ini adalah turn-based, maka pemain hanya dapat membangun bangunan dan unit, serta menyerang lawan secara bergantian.

Pada awal permainan, masing-masing *player* akan diberi 1 unit settler, dimana *settler* merupakan sebuah unit yang berfungsi untuk membuat kota. Setiap *player* dibebaskan memilih lokasi kota yang akan ditempati.

Setelah *player* membangun kota, pada awal turn *player* dapat membangun 1 unit yang dibutuhkan. Setiap *player* akan mendapatkan *gold* yang sama pada awal permainan. *Gold* pada *game* ini bertujuan untuk membayar biaya untuk membangun *unit* maupun bangunan. *Gold* pada *game* ini didapat dari pajak yang dibayar oleh tiap penduduk, menambang, menemukan *resource* pada *map*, maupun membunuh unit lawan.

Terdapat 5 indikator pada *game* ini, yaitu *food*, *production*, *resource*, *exp* dan *income*. *Food* merupakan elemen yang berguna untuk meningkatkan jumlah penduduk. Terdapat istilah *food basket*, yang berguna untuk menampung *food* yang dihasilkan. Jika nilai *food* lebih besar dari *food basket*, maka nilai penduduk akan bertambah dan nilai *food* akan dikurangi 100. *Production* bertujuan untuk mempercepat peningkatan level. *Production* yang lebih banyak akan mempercepat peningkatan level unit. Jika *worker* menerima peningkatan level, pembangunan yang dilakukan *worker* juga akan lebih cepat. *Exp* digunakan untuk menambah nilai *exp* pada unit saat pertama kali *unit* di *spawn*. *Income* akan meningkatkan pemasukan dari kota.

Unit merupakan pasukan yang dimiliki negara dan memiliki tujuan tertentu. Setiap *unit* memiliki fungsi tersendiri. Untuk membangun sebuah *unit*, diperlukan beberapa *turn* agar unit selesai dibangun. Setiap kota hanya dapat membangun 1 *unit* saja. Jika ingin membangun unit yang lainnya, harus menunggu hingga beberapa *turn* sampai *unit* telah selesai dibangun. Setiap *turn*, *player* dapat menggerakan setiap *unit* yang dimiliki *player*. *Unit warrior* maupun *shooter* dapat menyerang unit lawan.

Jika seorang *player* telah memiliki *unit worker*, *player* dapat memulai untuk membangun sebuah bangunan. Bangunan merupakan sebuah aset yang terdapat pada suatu negara. Bangunan juga menjadi indikator seberapa maju suatu negara. Berbeda dengan *unit*, setiap kota dapat membangun beberapa bangunan pada setiap *turn*, tergantung dari jumlah *worker* yang tersedia.

Player dapat menekan tombol *end turn* apabila *player* ingin mengakhiri gilirannya. Setelah giliran player berhenti, giliran akan berganti dengan player lainnya.

3.2 Implementasi *Particle Swarm Optimization*

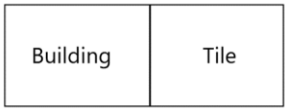
Game Build and Conquer menggunakan *Artificial Intelligence* (AI) sebagai lawan dari player. *AI* pada game ini berfungsi untuk menentukkan unit dan bangunan apa saja yang perlu dibangun, serta mengarahkan unit-unit lawan untuk bergerak ke suatu *tile* tertentu dan menjalankan fungsi unit tersebut. *AI* pada game ini menggunakan algoritma *Particle Swarm Optimization* (*PSO*).

3.2.1 Implementasi pada Jenis Bangunan dan Lokasi Bangunan

Pada langkah ini, metode *Particle Swarm Optimization* digunakan untuk menentukan jenis bangunan yang akan dibangun dan lokasi tempat dibangun. Untuk mendapatkan hasil yang diinginkan, ada beberapa parameter yang perlu diperhatikan:

- Statistik Kota
- Jenis Terrain
- Resource pada suatu tile

Statistik kota menunjukkan kondisi kota saat ini. Jenis Terrain menunjukkan jenis tanah dari suatu *hex*. Terdapat 3 jenis tanah pada *game* ini, yaitu *grassland*, *plains* dan *desert*. *Grassland* bagus untuk membangun *farm* dan *mine*. *Plains* bagus untuk membangun *barracks*, *castle* dan *mine*. Sedangkan desert merupakan jenis tanah yang paling buruk untuk semua jenis bangunan. Beberapa *tile* memiliki *resource* yang berguna untuk menambah nilai dari *farm* maupun *mine*.



Gambar 1. Desain Partikel

Gambar 1 menunjukkan representasi desain partikel yang akan digunakan. Sebuah partikel memuat *index* jenis bangunan dan *index* lokasi *tile*. Pada kasus ini, terdapat 2 ruang dimensi, yang berfungsi sebagai wadah untuk menempatkan partikel. Batas ruang dimensi 1 antara 0 hingga 4. Dimensi 1 merupakan ruang untuk *index* jenis bangunan. Sedangkan batas ruang dimensi 2 antara 0 hingga total tile yang tersedia. Dimensi 2 merupakan ruang untuk index lokasi tile. *Index* jenis bangunan terdiri dari :

Index 0-0.99 : *Farm*

Index 1-1,99 : *Mine*

Index 2-2,99 : *Barracks*

Index 3-3,99 : *Workshop*

Index 4-4,99 : *Castle*

Langkah-langkah *Particle Swarm Optimization* untuk *deploy* suatu bangunan dan lokasi tempat dibangun :

1. Inisialisasi partikel. Pada tahap ini akan menentukan jumlah partikel yang akan digunakan, inisialisasi posisi dan kecepatan partikel. Pada penelitian ini, jumlah partikel yang digunakan sebanyak 40. Menggunakan jumlah partikel yang terlalu sedikit menyebabkan nilai *fitness* yang didapat tidak optimal, sedangkan jumlah partikel yang terlalu banyak menyebabkan proses yang diperlukan menjadi lebih lama. Posisi partikel pada iterasi ke-0 (t=0) di-generate secara acak. Sedangkan kecepatan setiap partikel pada iterasi ke-0 (t=0) adalah 0.

2. Dilakukan perhitungan nilai *fitness* untuk setiap partikel dengan menggunakan rumus fitness berikut :

fitness= Statistic Score + Terrain Score + Resource Score

Formula *fitness* dijabarkan sebagai berikut :

- *Statistic Score* : Pada turn awal, diprioritaskan untuk menambah pemasukan kota, sehingga kota memiliki cukup *gold* untuk membangun unit maupun bangunan. *Production*

juga penting untuk dibangun pada turn awal, sehingga kota akan mengambil keuntungan yaitu pembangunan yang lebih cepat pada pembangunan berikutnya. Jika pemasukan dan production kota telah berkembang saatnya untuk membangun *farm*. *Farm* akan berguna untuk menambah penduduk pada kota. *Barracks* akan dibangun setelah pertumbuhan penduduk kota berkembang. Terakhir, *castle* dibangun pada *tile* yang masih tersisa untuk menambah *HP* kota. Rumus untuk *statistic score* ibu kota dijabarkan sebagai berikut :

- *Farm* : 8 - *food* / 8
- *Mine* : 12 – *Income Total* / 15
- *Workshop* : 10 - (*production* - 4) / 4
- *Barracks* : 5 - *exp* / 40
- *Castle* : 5 - *HP* / 50

Sedangkan untuk kota kecil dijabarkan sebagai berikut :

- *Farm* : 6 - *food* / 8
- *Mine* : 9 – *Income Total* / 15
- *Workshop* : 7 - (*production* - 4) / 4
- *Barracks* : 3 - *exp* / 40
- *Castle* : 2 - *HP* / 35

- *Terrain Score* : Pada bangunan *farm*, *terrain score* akan lebih tinggi jika dibangun di lahan *grassland*, sedangkan *workshop*, *mine* dan *castle* memiliki *score* yang lebih tinggi jika dibangun di lahan *plains*. Lahan *desert* merupakan lahan yang dihindari oleh semua bangunan. *Workshop* dan *castle* ideal jika dibangun di dekat *town hall*, sehingga nilai production dari *workshop* maupun *HP* dari *castle* akan bertambah. Berikut penjabaran terrain score :

  Farm :
  - *Grassland* : 4
  - *Plains* : 3
  - *Desert* : 2

  Barracks, *Mine* & *Castle*
  - *Grassland* : 3
  - *Plains* : 4
  - *Desert* : 2

- *Resource Score* : Beberapa *tile* dari *map* memiliki *resource*. Jika membangun *farm* maupun *mine* yang memiliki *resource*, akan menambah nilai dari *farm* maupun *mine*. Berikut penjabaran mengenai *resource score* :

  Farm :
  - *Resource Score* : 2 (jika *resource* : *grape*, *potato*, *wheat* dan *deer*)
  - *Resource Score* : -1 (jika *resource* : *bronze*, *silver*, *gold*, dan *diamond*)

  Mine :
  - *Resource Score* : 2 (jika *resource* : *bronze*, *silver*, *gold*, dan *diamond*)
  - *Resource Score* : -1 (jika *resource* : *grape*, *potato*, *wheat* dan *deer*)

Maka, ketiga nilai tersebut akan dijumlahkan untuk memperoleh nilai *fitness*. Semakin tinggi nilai *fitness* dari suatu partikel, semakin bagus kualitas partikel tersebut.

3. Inisialisasi *pBest* dan *gBest*. Pada tahap ini, karena masih berada pada iterasi ke-0 (t=0), maka nilai pBest sama dengan posisi awal partikel. Sedangkan gBest ditentukan dari nilai fitness terbaik yang dimiliki oleh semua partikel.

4. Memasuki iterasi selanjutnya, setiap partikel akan di-*update* kecepatannya.

5. Perbarui posisi setiap partikel.

6. Perbarui nilai *fitness* dari setiap partikel menggunakan rumus *fitness* pada nomor 5.

7. Perbarui *pBest* pada setiap partikel. *pBest* didapat dari nilai *fitness* terbaik dari suatu partikel. Cara memperbarui *pBest* adalah dengan membandingkan nilai *fitness* sekarang dengan nilai *fitness* terbaik dari partikel tersebut. Jika nilai *fitness* sekarang lebih baik, maka nilai *fitness* sekarang menjadi nilai *fitness* terbaik partikel tersebut. Partikel dengan nilai *fitness* terbaik menjadi *pBest*.

8. Perbarui *gBest* pada keseluruhan partikel. *gBest* didapat dari nilai *fitness* terbaik dari keseluruhan partikel. Cara memperbarui *gBest* adalah dengan membandingkan nilai fitness pBest dengan nilai fitness terbaik dari keseluruhan partikel. Jika nilai *fitness* sekarang lebih baik, maka nilai *fitness* sekarang menjadi nilai *fitness* terbaik dari keseluruhan partikel. Partikel dengan nilai *fitness* terbaik menjadi *gBest*.

9. Proses akan diulangi dari langkah 6 hingga mencapai jumlah iterasi yang telah ditentukan atau posisi partikel menjadi konvergen..

4. PENGUJIAN

4.1 Pengujian Sistem Particle Swarm Optimization

AI dengan metode *Particle Swarm Optimization (PSO)* akan dibandingkan dengan metode *AI random* yang telah dibuat. Perbandingan yang dilakukan meliputi aspek-aspek berikut :

- Perbandingan dari *income* / *gold* negara yang didapat oleh player hingga akhir permainan
- Perbandingan statistik, seperti *food*, *production*, *resource*, *exp growth* dan *income*
- Perbandingan jumlah *unit* serta building yang mampu dihasilkan oleh *player*

4.1.1 Perbandingan dengan AI Random

Subbab ini menjelaskan mengenai perbandingan dengan metode *AI random*. Parameter pengujian akan didasarkan oleh aspek yang telah dijelaskan pada bab 5.2. Pengujian dilakukan dengan memberikan komando untuk menentukan jenis bangunan dan lokasi bangunan kepada *unit worker* yang bertugas untuk membuat bangunan pada kota. Jumlah uang awal yang dimiliki oleh kedua pemain adalah sebesar 1000. Setiap pergantian *turn*, masing-masing pemain akan memperoleh sejumlah uang yang jumlahnya tergantung bagaimana pemain mengelola negaranya.

Tabel 1. Tabel hasil partikel *PSO* yang terpilih pada pengujian melawan *AI Random*

Turn	Building dan Hex
2	Building : Mine Hex : 63, 13 Terrain : Plains (dibangun di ibu kota) Ada Resource Income: Ya Effect : +20 Income Fitness : 16.333 Selesai pada turn 9
10	Building : Mine Hex : 63, 12 Terrain : Plains (dibangun di ibu kota)

Tabel 1. Tabel hasil partikel *PSO* yang terpilih pada pengujian melawan *AI Random* (lanjutan)

Turn	Building dan Hex
	Ada Resource Income: Tidak Effect : +15 Income Fitness : 14 Selesai pada turn 17
18	Building : Mine Hex : 64, 12 Terrain : Plains (dibangun di ibu kota) Ada Resource Income: Tidak Effect : +15 Income Fitness : 13 Selesai pada turn 25
26	Building : Workshop Hex : 62, 14 (sebelah pusat kota) Terrain : Plains (dibangun di ibu kota) Effect : +5 Production Fitness : 13 Selesai pada turn 34
28	Building : Mine Hex : 63, 20 Terrain : Plains (dibangun di kota kecil) Ada Resource Income : Tidak Effect : +15 Income Fitness : 12.333 Selesai pada turn 40

Tabel 2. Tabel hasil perbandingan jenis bangunan dan lokasi bangunan dengan AI Random

AI PSO	*AI Random*
Mine | 63, 13 (ibu kota)	Mine | 13, 16 (ibu kota)
Mine | 63, 12 (ibu kota)	Barracks | 16, 14 (ibu kota)
Mine | 64, 12 (ibu kota)	Mine | 16, 12 (ibu kota)
Workshop | 62, 14 (ibu kota)	Mine | 11, 11 (kota kecil)
Mine | 63, 20 (kota kecil)	Mine | 13, 14 (ibu kota)
Mine | 62, 13 (ibu kota)	Mine | 12, 16 (ibu kota)
Mine | 64, 15 (ibu kota)	Farm | 12, 14 (ibu kota)
Mine | 67, 19 (kota kecil)	Mine | 9, 11 (kota kecil)
Farm | 61, 15 (ibu kota)	Barracks | 14, 15 (ibu kota)
Farm | 61, 16 (ibu kota)	Farm | 16, 13 (ibu kota)
Workshop | 62, 15 (ibu kota)	Mine | 10, 11 (kota kecil)
Workshop | 63, 15 (ibu kota)	Workshop | 15, 15 (ibu kota)
Farm | 64, 20 (kota kecil)	Castle | 14, 12 (ibu kota)
Mine | 65, 14 (ibu kota)	Barracks | 15, 13 (ibu kota)
Mine | 62, 16 (ibu kota)	Workshop | 7, 14 (kota kecil)
Farm | 65, 12 (ibu kota)	Mine | 15, 12 (ibu kota)

Tabel 2. Tabel hasil perbandingan jenis bangunan dan lokasi bangunan dengan AI Random (lanjutan)

AI PSO	*AI Random*
Workshop | 66, 18 (kota kecil)	Mine | 14, 16 (ibu kota)
Farm | 61, 14 (ibu kota)	Barracks | 10, 10 (kota kecil)
Barracks | 65, 13 (ibu kota)	Mine | 12, 17 (ibu kota)
Workshop | 63, 16 (ibu kota)	Mine | 13, 12 (ibu kota)
Mine | 66, 17 (kota kecil)	Workshop | 16, 15 (ibu kota)
Mine | 65, 15 (ibu kota)	Workshop | 14, 17 (ibu kota)
Farm | 63, 11 (ibu kota)	Workshop | 16, 11 (ibu kota)
Mine | 67, 17 (kota kecil)	Workshop | 12, 13 (ibu kota)
Farm | 60, 16 (ibu kota)	Barracks | 15, 11 (ibu kota)
Barracks | 61, 13 (ibu kota)	Workshop | 11, 16 (ibu kota)
Barracks | 66, 13 (ibu kota)	
Workshop | 62, 12 (ibu kota)	
Workshop | 65, 18 (kota kecil)	
Workshop | 66, 12 (ibu kota)	
Workshop | 66, 11 (ibu kota)	
Mine | 61, 17 (ibu kota)	
Farm | 65, 17 (kota kecil)	
Castle | 66, 14 (ibu kota)	
Castle | 64, 13 (ibu kota)	

Tabel 1. menunjukkan hasil partikel yang terpilih pada setiap giliran. Hasil ini didapatkan dari proses metode algoritma *AI Particle Swarm Optimization* dengan formula *fitness* yang telah ditentukan untuk penentuan jenis dan lokasi bangunan seperti yang tertera pada (ada di bab 3).

Tabel 2. menunjukkan perbandingan jenis bangunan dan lokasi bangunan yang dihasilkan oleh kedua *AI*. *AI PSO* cenderung menempatkan lokasi bangunan yang optimal untuk jenis bangunan tersebut. Semisal *farm* cenderung akan ditempatkan pada lahan *grassland* dan memprioritaskan untuk dibangun pada *tile* yang memiliki *resource food*. Hal ini akan membuat *farm* menghasilkan nilai *food* yang lebih banyak dibandingkan berada pada lahan plains maupun *desert*. Sedangkan untuk bangunan *mine*, *barracks* dan *castle*, *AI PSO* akan cenderung menempatkannya pada lahan *plains*.

Berbeda dengan *AI Random* yang cenderung acak dalam penentuan jenis bangunan dan lokasi bangunan. Dimana *AI Random* cenderung membangun *mine*, sehingga memiliki jumlah *income* yang lebih banyak, namun memiliki statistik lain yang lebih rendah.

Tabel 3. Tabel hasil pengujian *AI PSO* melawan *AI Random*

Aspek	*AI PSO*	*AI Random*
Jumlah Gold yang Dihasilkan	19365	14125
Jumlah Food	52	8
Jumlah Production	36	23
Jumlah Income dari Mine	210	165
Jumlah Exp Growth	120	200
Jumlah Unit yang Dibangun	30	12
Jumlah Building yang Dibangun	35	27
Jumlah Kota Tersisa	2	0
Jumlah Turn	130	
Winner	*AI PSO*	

Tabel 3. memperlihatkan hasil pengujian antara *AI PSO* dengan *AI Random*. Pada aspek jumlah *gold*, terlihat *AI PSO* lebih banyak memperoleh *gold* dibandingkan *AI Random*. Hal ini mengakibatkan *AI PSO* bisa lebih mudah dalam membangun unit maupun *building*. Pada aspek jumlah food, *AI PSO* menghasilkan jauh lebih banyak *food* dibandingkan *AI Random*. Hal ini akan berdampak pertumbuhan penduduk yang dihasilkan *AI PSO* lebih banyak. Salah satu alasan mengapa *AI Random* kalah adalah karena pertumbuhan penduduk yang lambat dibandingkan AI *PSO*. Pada aspek jumlah *resource*, *resource* yang dihasilkan oleh *AI PSO* sedikit lebih banyak dibandingkan *AI Random*. Pada aspek jumlah *production*, production yang dihasilkan *AI PSO* jauh lebih banyak dibandingkan *AI Random*. *Production* yang lebih banyak akan mempercepat pembangunan. Nilai *production* yang lebih sedikit mengakibatkan *AI Random* tertinggal dalam kecepatan pembangunan. *Exp growth* pada *AI PSO* lebih sedikit daripada *AI Random*. *Exp growth* akan menambah nilai *exp unit* saat pertama kali di *spawn*.

Hal ini menunjukkan bahwa *AI PSO* memiliki pembangunan yang lebih optimal daripada *AI Random*. Jumlah *turn* menunjukkan jumlah turn yang dibutuhkan untuk memenangkan pertandingan. Dalam test case ini, *AI PSO* berhasil mengalahkan *AI Random*.

4.1.2. Perbandingan dengan Tester

Subbab ini akan menjelaskan mengenai pengujian sistem *AI PSO* dalam penentuan jenis dan lokasi bangunan melawan tester. Dalam hal ini, *game* akan dimainkan oleh *tester* untuk pengujian *AI PSO*. Jumlah uang awal yang dimiliki oleh kedua pemain adalah sebesar 1000. Setiap pergantian *turn*, masing-masing pemain akan memperoleh sejumlah uang yang jumlahnya tergantung bagaimana pemain mengelola negaranya. *Tester* bebas untuk menentukan jenis dan lokasi bangunan yang akan dibangun. Tester diberi 2 kali kesempatan untuk melakukan pengujian.

Tabel 4. Tabel hasil pengujian pertama *AI PSO* melawan *Tester*

Aspek	*AI PSO*	*Tester*
Jumlah Gold yang Dihasilkan	19880	8930
Jumlah Food	43	14
Jumlah Production	36	8
Jumlah Income dari Mine	235	55
Jumlah Exp Growth	120	90
Jumlah Unit yang Dibangun	28	10
Jumah Building yang Dibangun	38	11
Jumlah Kota Tersisa	2	0
Jumlah Turn	133	
Winner	*AI PSO*	

Pada Tabel 4. ditampilkan hasil pengujian pertama dari *AI PSO* melawan *Tester*. *AI PSO* mampu menentukan keputusan yang optimal dengan memperhatikan statistik kota dan kondisi *terrain* / *tile*. Pemilihan jenis dan lokasi bangunan menjadi hal yang penting, dimana pada giliran awal, *AI PSO* cenderung membangun jenis bangunan yang meningkatkan *income* dari kota, meningkatkan *production*, serta *food* dari kota. Ketiga aspek tersebut penting untuk menunjang pembangunan bangunan mau *Game* berlanjut hingga *AI PSO* mempu menghancurkan kota kecil hingga ke ibu kota lawan. Kemenangan *AI PSO* sebenarnya juga tidak luput dari *tester* yang kerap lupa untuk membangun *building* dan *unit*, serta pemilihan jenis dan lokasi bangunan yang kurang optimal.

Tabel 5. Tabel hasil pengujian kedua *AI PSO* melawan *Tester*

Aspek	*AI PSO*	*Tester*
Jumlah Gold yang Dihasilkan	22045	15672
Jumlah Food	57	39
Jumlah Production	35	9
Jumlah Income dari Mine	205	163
Jumlah Exp Growth	160	0

Tabel 5. Tabel hasil pengujian kedua *AI PSO* melawan *Tester* (lanjutan)

Aspek	*AI PSO*	*Tester*
Jumlah Unit yang Dibangun	28	14
Jumah Building yang Dibangun	39	21
Jumlah Kota Tersisa	2	0
Jumlah Turn	139	
Winner	*AI PSO*	

Pada Tabel 5. ditampilkan hasil pengujian kedua dari *AI PSO* melawan *Tester*. *Tester* mempu memperbaiki permainannya di pengujian kedua. Terbukti dari statistik yang lebih baik dari pengujian pertama. Meskipun begitu, *tester* belum mampu menemukan pembangunan yang optimal, sehingga secara statistik, masih tertinggal dari *AI PSO*, dan *tester* harus mengakui kekalahan dari *AI PSO*.

5. KESIMPULAN

Selama perencanaan, desain, program hingga pengujian, terdapat beberapa kesimpulan yang dapat ditarik dari penelitian ini.Kesimpulan yang diperoleh antara lain :

- Pada batas 30, 50 dan 100 *turn*, belum bisa dibuktikan apakah *PSO* lebih baik daripada random akibat jumlah *turn* yang terbatas. Hasil *PSO* lebih baik dari *random* terlihat pada batas mulai dari 150 *turn*.
- Perubahan nilai variabel dari rumus *fitness* akan menghasilkan output hasil permainan yang berbeda.
- Perubahan nilai W, C1 dan C2 menghasilkan output hasil permainan yang berbeda.
- Responden menjawab bahwa permainan ini menarik untuk dimainkan.
- Particle Swarm Optimization bergantung kepada statistik dari jenis bangunan maupun lokasi bangunan.

6. DAFTAR PUSTAKA

[1] Kurniati, Agnes. 2015. "*Game development 'tales of mamochi' with role playing game concept based on android*", International Conference on Computer Science and Computational Intelligence (ICCSCI 2015). Procedia Computer Science 59 (2015): 392 – 399.

[2] Barnouti, N. H. "*Pathfinding in strategy games and maze solving using A* search algorithm*". Journal of Computer and Communications (2016), 4, 15-25.

[3] Sabir, S. 2016. "*A Comparative study of genetic algorithm and the particle swarm optimization*". International Journal of Electrical Engineering 9.2 (2016): 215-223.

[4] Cunha, A. A. 2015. "*Swarm intelligence in strategy games*". Lisboa : Instituto Superior Técnico.

[5] Sanjaya, A. L. 2016, "*Aplikasi game strategi menggunakan metode algoritma genetika untuk pembuatan pasukan*". Surabaya : Petra Christian University.

[6] Jones, Karl O. 2006. "*Comparison of genetic algorithms and particle swarm optimisation for fermentation feed profile determination*". International Conference on Computer Systems and Technologies – CompSysTech (2006).

[7] Kennedy, J. & Eberhart. R. 1995. "*Particle Swarm Optimization*". Purdue School of Engineering and Technology:Indianapolis..