Implementasi Kombinasi Jaringan Syaraf Tiruan Metode *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* Pada *AI Game Action Adventure*.

Kevin Leonard Tjung[1], Gregorius Satia Budhi[2], Kristo Radion Purba[3]
Program Studi Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jl. Siwalankerto 121 – 131, Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-mail: kevin.tjung827@gmail.com[1], greg@petra.ac.id[2], kristo@petra.ac.id[3]

ABSTRAK

Game merupakan aplikasi multi media yang membutuhkan beberapa unsur utama yaitu, tujuan, tantangan dan *game play* atau biasa dikenal sebagai aturan main dalam sebuah *game*.

Game action adventure akan menerapkan sistem *game play* yang sifatnya *real time*. Sifat *game play* yang *real time* biasanya membutuhkan metode pengambilan keputusan yang cepat sehingga tidak mengganggu sistem *game play* yang sedang berjalan.

Salah satu yang metode yang cepat dalam proses pengambilan keputusan dan beradaptasi adalah *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)*. Penggunaan metode *SOM* dapat digunakan sebagai bagian *decision making* dari *AI* sedangkan *BAM* dapat digunakan untuk menyimpan gerakan terakhir *player* yang digunakan untuk membunuh *AI*.

Berdasarkan hasil pengujian, *SOM* dan *BAM* dapat diterapkan *AI* dalam sistem *game* yang sifatnya *real time* dan berhasil beradaptasi terhadap serangan *player* walaupun tidak sempurna.

Kata Kunci: Jaringan Saraf Tiruan, *Self-Organized Map (SOM)*, *Bidirectional Associative Memory (BAM), AI*, *Action Adventure Game*.

ABSTRACT

Game is a multi media application that need some main parts such as, goal, challenge and game play or usually has been know as rules in a game

.

Action adventure game will apply real time system in its game play. Real time game usually needs fast decision making method for artificial intelligence, so it can't interrupt any system in the game.

Self-Organized Map (SOM) and Bidirectional Associative Memory (BAM) is one of the fastest method that can be used to process data for decision making and for adaptation. SOM method can be used as decision making part for AI while BAM can be use to store player last move that being used to kill the AI.

Based on testing result, SOM and BAM can be applied in real time game system as AI and successfully adapt to player attacks although not perfect.

Keywords: *Neural Network*, *Self-Organized Map (SOM)*, *Bidirectional Associative Memory (BAM), AI*, *Action Adventure Game.*

1. PENDAHULUAN

Perkembangan dunia *game* kini semakin berkembang pesat. Bukan hanya dalam hal jumlah *game* yang dihasilkan pertahun, grafik yang semakin indah, *gameplay* yang semakin mulus, namun *AI* yang digunakan pun semakin pintar. Salah satu jenis *AI* yang sering digunakan pada *game* saat ini adalah *AI* Jaringan Syaraf Tiruan (JST). Oleh karena itu, dalam proposal ini saya ingin menggabungkan dua jenis JST yang berbeda yaitu *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* sebagai *AI*. *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* lebih mudah untuk diterapkan dalam *game* yang *real-time* karena memiliki proses perhitungan yang lebih cepat. Apa itu *game* yang bersifat *real-time*? Suatu *game* disebut *real-time* jika baik *player* maupun *AI* dapat bertindak dalam waktu bersamaan, tidak seperti *game turn-based* dimana player maupun *AI* harus menunggu giliran masing-masing sebelum dapat bertindak. Oleh sebab itu pada skripsi ini akan dibuat *game* yang ber-*genre action adventure* yang menggunakan kombinasi JST *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* sebagai *AI game* tersebut dan akan dibuat dengan menggunakan *game engine Unreal Engine 4*.

Alasan utama untuk memilih *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* sebagai *AI game action adventure* adalah, jenis perkelahian yang ditawarkan oleh *game action adventure* kebanyakan bertipe *fast paced* yaitu melibatkan banyak aksi pergerakan yang sangat cepat untuk mengalahkan musuh dalam *game*. Oleh karena itu, *AI game action adventure* sangat cocok untuk menggunakan kombinasi kedua jenis JST tersebut.

2. LANDASAN TEORI

2.1 Pengenalan *Game Action Adventure*

Game Action Adventure merupakan *game* dengan kombinasi *genre Action* yang menekankan pada tindakan-tindakan *player* yang harus dilakukan untuk mencapai suatu tujuan tertentu dan *genre Adventure* menekankan pada *player* dapat menjelajahi dunia *game* dan menyelesaikan masalah-masalah (*puzzle*) tertentu dangan tindakan tertentu [4]. Contoh *game action adventure* yang populer adalah *Batman: Arkham Asylum*, seri *Grand Theft Auto*, seri *Assassin's Creed*, seri *Resident Evil*, dan seri *Tomb Raider*. Contoh *game action adventure* yang populer dan menggunakan unreal engine adalah *DmC (Devil May Cry) 2013* pada Gambar 1.

Gambar 1. ***DMC (Devil may Cry) 2013.***

2.2 Pengenalan Jaringan Syaraf Tiruan (JST)

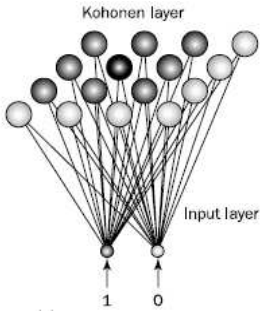
Jaringan Syaraf Tiruan (JST) merupakan suatu sistem pemrosesan informasi yang mempunyai karakteristik menyerupai jaringan syaraf biologi (JSB). JST tercipta sebagai suatu generalisasi model matematis dari pemahaman manusia (*human cognition*) yang didasarkan atas asumsi sebagai berikut :

1. Pemrosesan informasi terjadi pada elemen sederhana yang disebut neuron.
2. Sinyal mengalir diantara sel saraf/neuron melalui suatu sambungan penghubung.
3. Setiap sambungan penghubung memiliki bobot yang bersesuaian. Bobot ini akan digunakan untuk menggandakan / mengalikan sinyal yang dikirim melaluinya.
4. Setiap sel syaraf akan menerapkan fungsi aktivasi terhadap sinyal hasil penjumlahan berbobot yang masuk kepadanya untuk menentukan sinyal keluarannya.

Jaringan syaraf tiruan dapat belajar dari pengalaman, melakukan generalisasi atas contoh-contoh yang diperolehnya dan mengabstraksi karakteristik esensial masukan bahkan untuk data yang tidak relevan [8]. Algoritma untuk JST beroperasi secara langsung dengan angka sehingga data yang tidak numerik harus diubah menjadi data numerik. JST tidak diprogram untuk menghasilkan keluaran tertentu. Semua keluaran atau kesimpulan yang ditarik oleh jaringan didasarkan pada pengalamannya selama mengikuti proses pembelajaran. Pada proses pembelajaran, ke dalam JST dimasukkan pola-pola masukan (dan keluaran) lalu jaringan akan diajari untuk memberikan jawaban yang bisa diterima.

2.3 *Self-Organized Map (SOM)*

Self-Organized Map (SOM) merupakan salah satu jenis JST yang menggunakan teknik *unsupervised learning*, dimana jaringan belajar untuk membentuk klasifikasinya sendiri melalui *data training* tanpa bantuan dari luar. *Self-Organized Map (SOM)* juga merupakan JST yang berdasar pada *competitive learning*, dimana *neuron output* saling berkompetisi satu sama lain untuk diaktifkan dan hanya terdapat satu *neuron* yang teraktivasi pada akhirnya [3]. *Neuron* yang teraktivasi ini disebut *neuron* pemenang. Berikut merupakan contoh model JST *Self-Organized Map (SOM)*.



Gambar 2. Jaringan *Self-Organized Map (SOM)* Kohonen.

Algoritma *Self-Organized Map (SOM):*

1) Tahap Inisialisasi:
Set weight (bobot) awal dengan nilai acak kecil antara 0-1 dan *set* nilai *learning rate* α dengan nilai positif kecil.

2) Tahap Aktivasi dan *Similarity Matching***:**
Mengaktifkan jaringan *Self-Organized Map (SOM)* dengan memasukkan nilai *input vector* X, lalu temukan *neuron* pemenang j_X pada iterasi ke p menggunakan *minimum-distance Euclidean criterion:*

$$j_X(p) = \min \left|\left|X - W_j(p)\right|\right| = \left\{\sum_{i=1}^{n}\left[x_i - W_{ij}(p)\right]^2\right\}^{\frac{1}{2}},$$
$$j = 1, 2, \ldots\ldots, m$$

Dimana *n* merupakan jumlah *neuron* pada *input layer* dan *m* merupakan jumlah *neuron* pada *output layer*.

3) Tahap *Learning***:**
Melakukan *update weight*:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Dimana $\Delta w_{ij}(p)$ merupakan perbaikan *weight* pada iterasi ke *p*. perbaikan *weight* ditentukan oleh *competitive learning rule*:

$$\Delta w_{ij}(p) = \begin{cases} \alpha\left[x_i - W_{ij}(p)\right], & j \in \hat{}j(p) \\ 0, & j \notin \hat{}j(p) \end{cases}$$

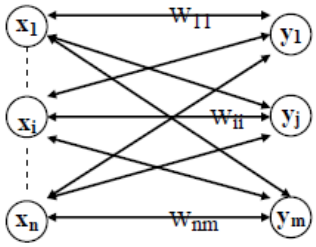
Dimana α merupakan *parameter learning rate* dan $\hat{}j(p)$ merupakan fungsi tetangga yang berpusat pada *neuron* pemenang j_X pada iterasi ke *p*.

4) Tahap Iterasi:

Tambah iterasi *p* sebanyak 1 kali lalu kembali pada tahap kedua hingga *minimum-distance Euclidean criterion* telah terpenuhi atau tidak ada perubahan yang tampak pada matriks *weight*.

2.4 *Bidirectional Associative Memory (BAM)*

BAM merupakan salah bentuk dari jaringan *Heteroassociative memory*, yang dikembangkan oleh Kosko (1988) [7]. *BAM* menyimpan pola pelatihan dalan matriks n x m yang merupakan perkalian dari vektor input dan target pelatihan (*outer product*). Arsitektur *BAM* terdiri dari 2 lapisan yang dihubungkan oleh lintasan koneksi *weight*. Jaringan akan beriterasi, mengirimkan sinyal pulang pergi antara kedua lapisan sampai semua *neuron* menjadi stabil (semua aktivasi *neuron* konstan). *BAM* dapat memberikan respon terhadap *input* dari kedua lapisan [7]. *Weight* bersifat 2 arah (*bidirectional*). Berikut merupakan contoh arsitektur *BAM*.



Gambar 3. Contoh arsitektur *BAM*.

Lapisan x terdiri dari n unit dan lapisan y terdiri dari m unit. Koneksi antara kedua lapisan bersifat 2 arah, yaitu jika matriks *weight* untuk sinyal yang dikirim dari lapisan x ke lapisan y adalah w, maka matriks *weight* untuk sinyal yang dikirim dari lapisan y ke lapisan x adalah w^T.

Dalam algoritma *BAM* terdapat 3 Tahap:

1) Tahap I: *Storage*

BAM membutuhkan M pasang pola untuk disimpan dalam bentuk matriks. *Weight* matriks detentukan dengan rumus:

$$W = \sum_{m=1}^{4} X_m \times Y_m^T$$

Dimana angka 4 menunjukkan jumlah pasangan yang disimpan. X_m merupakan matriks pola yang akan diingat dan Y_m^T merupakan transpose dari pasangan matriks X_m.

2) Tahap II : *Testing*

BAM dapat menerima *vector* apapun dari set X dan mengembalikan *vector* asosiasi dari set Y maupun sebaliknya dengan cara:

$$Y_m = sign(W^T X_m), m = 1,2,\ldots,M$$
$$X_m = sign(W \quad Y_m), m = 1,2,\ldots,M$$

3) Tahap III: *Retrieval*

Menggunakan *vector* tidak dikenal X sebagai *input* pada *BAM* untuk mengambil *vector* asosiasi pada *BAM*. *Vector* asosiasi yang diterima dapat *corrupt* atau tidak lengkap dari set X maupun set Y yang tersimpan dalam *BAM*. Dengan cara,

$$X \neq X_m, \qquad m = 1,2,\ldots,M$$

a) menginisialisasi *BAM retrieval algorithm*:

$$X(0) = X, \qquad p = 0$$

Dan mengkalkulasi hasil output *BAM* pada iterasi ke p,

$$Y(p) = sign[W^T X(p)]$$

b) Meng*update input vector* $X(p)$:

$$X(p+1) = sign[W\, Y(p)]$$

Ulangi iterasi sampai mencapai *equilibrium,* ketika input maupun output *vector* tidak lagi berubah.

3. DESAIN SISTEM

AI pada *game* akan dibuat dengan mengkombinasikan JST *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* murni tanpa perubahan sesuai dengan yang diterangkan dalam landasan teori pada Bab 2 diatas, dimana perubahan yang dilakukan hanya terdapat pada metode *input* dan *ouput* dari *SOM* dan *BAM*. pada *AI* bagian *SOM* akan digunakan dalam proses decision making *AI* dan *BAM* digunakan sebagai penyimpan gerakkan *player* terakhir seperti yang sudah tertulis pada ruang lingkup pada Bab 1 diatas.

Musuh akan terbagi menjadi 3 kelas yaitu, *mob (normal monster), elite mob (elite monster)*, dan *boss*. Kelas *mob* memiliki jumlah *storage (memory)* pada *BAM* sebanyak 8 *slot* (pasang) dengan jumlah *cluster* pada *SOM* yang paling sedikit yaitu 4 (serang, bertahan, menghindar, dan kejar). Kelas *elite mob* memiliki jumlah *storage (memory)* pada *BAM* sebanyak 4 *slot* (pasang) dengan jumlah *cluster* pada *SOM* sebanyak 5 *cluster* (serang, bertahan, menghindar, kejar, dan satu *slot* untuk *special move/skill*). Kelas *boss* memiliki jumlah *storage (memory)* pada *BAM* sebanyak 4 *slot* (pasang) dengan jumlah *cluster* pada *SOM* sebanyak 7 *cluster* (serang, bertahan, menghindar, kejar, dan sisa tiga *slot* untuk *special move/skill*).

Cara implementasi kombinasi *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* sebagai *AI*:

Bagian *SOM* pada *AI* bertugas untuk melakukan proses pengambilan keputusan. *Input (training set)* dari JST *Self-Organized Map (SOM)* merupakan kombinasi dari 5 *parameter* dasar dan *parameter* tambahan yaitu:

- Jarak dengan target/musuh. Nilai *Max*=2000 dan nilai *Min*=0.
- *Health point* yang dimiliki oleh *AI*. Nilai *Max*=1000 untuk Mob, *Max*=2000 untuk Elite Mob, *Max*= 4000 untuk Boss dan nilai *Min* untuk ketiganya sama yaitu *Min*=0.
- *Energy* yang dimiliki oleh *AI*. Nilai *Max*=100 dan Nilai *Min*=0. Nilai ini secara default bernilai 0 untuk kelas *Mob* karena, *Mob* tidak memiliki skill.
- *Health point* yang dimiliki oleh *Player*. Nilai *Max*=2000 dan nilai *Min*=0.
- *Energy* yang dimiliki oleh *Player*. Nilai *Max*=100 dan Nilai *Min*=0.
- Parameter tambahan berupa parameter *float* yang benilai 0 atau 1 berdasarkan gerakan terakhir musuh. Untuk kelas *Mob* terdapat 4 parameter tambahan yang dicek (jika gerakan terakhir *AI* merupakan *Chase* maka hanya parameter *float* tambahan yang bersangkutan yang diset 1 parameter tambahan yang lain diset 0) melalui variabel *boolean Chase, Attack, Defense*, dan *Evade*. Untuk kelas *Elite Mob* terdapat 5 parameter tambahan yang dicek melalui variabel *Chase, Attack, Defense, Evade*, dan *SPA*. Untuk kelas *Boss* terdapat 7 parameter tambahan yang dicek melalui variable *Chase, Attack, Defense, Evade, SPA, SPA2,*dan *SPA3*.
- Sehingga *Input SOM* kelas *Mob* berupa *array* berisi 9 elemen, kelas *Elite Mob* berupa *array* berisi 10 elemen, dan kelas *Boss* berupa *array* berisi 12 elemen.

Matriks *SOM* kelas *Mob* memiliki 9 baris dan 4 kolom dimana jumlah baris disesuaikan dengan banyak input dan jumlah kolom disesuaikan dengan banyak *cluster* (aksi yang dapat dilakukan), Matriks *SOM* kelas *Elite Mob* memiliki 10 baris dan 5 kolom, dan Matriks *SOM* kelas *Boss* memiliki 12 baris dan 7 kolom.

Weight (bobot) dari *SOM* akan ditentukan secara acak (*manual)* dan akan di-*training* menggunakan *training set* yang *parameter*-nya didapat dari *player* dan *AI* selama *game* berlangsung.

Proses kerja *SOM* sebagai berikut, *Input* (berupa *array*) yang didapat selama *game* berlangsung dikalikan dengan matriks *SOM* sehingga memberikan output (disimpan dalam array output)

sebanyak jumlah kolom pada matriks *SOM*. Sebelum dikali tiap parameter dasar akan diset melalui fungsi normalisasi terlebuh dahulu. Setelah itu proses e*uclidean distance* milik *SOM* akan dikalikan dengan koefisien *weight* untuk masing-masing perkalian tiap elemen. Elemen dari parameter dasar memiliki koefisien *weight* yang lebih besar dari parameter tambahan. Selanjutnya dalam *array output* akan dipilih indeks yang memiliki hasil terkecil, indeks yang memiliki hasil tekecil itulah yang akan dipilih sebagai *decision* yang dilakukan oleh *SOM* (ditranslasikan kedalam bentuk *behaviour*), lalu kolom matriks *SOM* yang bersangkutan dengan indeks pemenang itulah yang akan di *update weight*-nya. Matriks *SOM* akan di-*share* untuk digunakan oleh musuh yang memiliki kelas sama tetapi proses *decision making* akan dilakukan oleh setiap individual musuh. Proses kerja *SOM* ini akan di ulang terus selama *game* berlangsung oleh setiap musuh baik yang memiliki kelas berbeda atau sama.

Bagian *BAM* pada *AI* bertugas sebagai tempat penyimpan gerakan *player* terakhir yang digunakan untuk membunuh *AI*. Implementasi *BAM* pada *AI* adalah dengan menggunakan *queue FIFO (First In First Out)* yang menyimpan 5 gerakan terakhir *player* sebagai *set* A (berupa *array*) dan 4 kombinasi angka biner yang sudah manjadi *bipolar* sebagai *set* B (berupa *array*). Telah disiapkan 4 *se*t B, hal ini dikarenakan jumlah *storage BAM* yang terbanyak dimiliki oleh musuh kelas *Mob* yaitu 4. *queue FIFO* akan dimasukkan sebagai *set* A ketika *AI* mati. Matriks *BAM* yang disiapkan untuk masing-masing kelas musuh memiliki jumlah baris 5 dan jumlah kolom 4.

Kombinasi gerakan *player* yang bisa dimasukkan dalam *queue* adalah *Heavy Attack* dan *Light Attack* di *bipolar*-kan sebagai nilai 1 (true), sedangkan *Skill 1, Skill 2, Skill 3* di *bipolar*-kan sebagai nilai -1 (false) hal ini dikarenakan oleh *BAM* hanya dapat menerima *input bipolar true* atau *false* (1/-1). Nilai *parameter set* B juga masing-masing berupa *true* atau *false* (1/-1). Ketika *slot BAM* yang bisa terpakai telah penuh maka *AI* tidak bisa menambah memorinya lagi. Memori dari *BAM* hanya akan di-*share* pada *AI* dengan kelas yang sama.

Proses kerja *BAM* adalah setiap kali *AI* mati maka 5 gerakan terakhir akan dijadikan sebagai *set* A dan dikalikan dengan satu *set* B yang sudah disediakan secara urut lalu hasilnya akan disimpan kedalam matriks *BAM* kelas musuh yang bersangkutan, hal ini akan dilakukan terus sampai jumlah *storage* kelas musuh yang bersangkutan habis. Jika *storage BAM* sudah penuh maka setiap kali musuh melakukan proses *decision making (SOM)* maka *array* yang menyimpan 5 gerakan terakhir musuh itu akan digunakan sebagai *retrieval set*. Jika setelah *BAM* melakukan proses *retrieval* menemukan ada *set* yang *match* (sama/mirip) dari *storage* matriks *BAM* maka *AI* akan memilih untuk menggunakan hasil *SOM* sebesar 2/3 kemungkinan dan *BAM* (*evade*)sebesar 1/3 kemungkinan.

4. PENGUJIAN SISTEM

Cara pengujian *AI game* adalah sebagai berikut, untuk *AI* gabungan *SOM* dan *BAM playe*r akan memainkan *game* melalui *stage* 1 sampai *stage* 3 lalu diulangi terus sampai tidak terlihat perubahan pada ketiga matriks milik *monster* kelas *mob, elite mob,* dan *boss* lagi. Hal ini berarti *learning rate* untuk masing-masing matriks sudah sama dengan nol atau terlalu kecil sehingga tidak tampak dalam matriks yang disebabkan karena tiap elemen dalam matriks hanya akurat sampai lima angka dibelakang tanda koma. Setelah itu, *player* akan bermain menggunakan *AI* gabungan *SOM* dan *BAM* lagi mulai dari *stage* 1 sampai *stage* 3 (hanya 1 kali saja untuk tiap *stage*) yang hasilnya akan dibandingkan dengan hasil *AI rule based*. Khusus untuk *AI* gabungan *SOM* dan *BAM* pengujian akan dilakukan sebanyak 5 kali. Sedangkan untuk *AI rule based player* juga akan memainkan *game* melalui *stage* 1 sampai *stage* 3 namun hanya akan dilakukan sekali untuk tiap *stage*, hal ini disebabkan karena *AI rule based* tidak perlu melakukan proses *training*. Untuk *AI rule based* pengujian akan dilakukan sebanyak 1 kali saja. Proses *training AI* kombinasi *SOM* dan *BAM* serta pengujian *AI* kombinasi *SOM* dan *BAM* dengan *AI rule based* dilakukan semirip mungkin (gerakkan aksi *player*). Perlu diingat bahwa *SOM* menangani bagian *decision making* dalam *AI* sedangkan *BAM* menangani bagian *AI* yang beradaptasi terhadap gerakan serangan *player* (menyimpan 5 serangan *player* terakhir yang digunakan untuk membunuh musuh).

Tabel 1. Hasil pengujian *AI game* (*stage* 1-3)

Tipe *AI*	Total Waktu	Total *Player* Mati	Total *Mob* Yang Dibunuh	Total *Elite Mob* Yang Dibunuh	Total *Boss* Yang Dibunuh
SOM+ BAM (tes 1)	6 menit 45 detik	0	11	7	3
SOM+ BAM (tes 2)	6 menit 57 detik	0	11	7	3
SOM+ BAM (tes 3)	5 menit 49 detik	0	11	7	3
SOM+ BAM (tes 4)	6 menit 21 detik	0	11	7	3
SOM+ BAM (tes 5)	5 menit 57 detik	0	11	7	3
Rule Based	4 menit 33 detik	0	11	7	3

Jika dilihat dari tabel 1 tampak bahwa total waktu yang dibutuhkan untuk menyelesaikan *stage* 1-3 dengan *AI SOM* dan *BAM* lebih lama dibandingkan *AI Rule Based*. Melalui hasil observasi hal ini disebabkan oleh, *monster* dengan *AI SOM* dan *BAM* lebih susah dibunuh karena dapat menghindar melalui serangan yang mirip namun walaupun demikian *monster* dengan *AI rule based* memberikan *damage* yang terbesar dimana *HP player* pernah jatuh sampai 50% pada akhir *stage* ketika menghadapi *monster* dengan *AI rule based*. Sedangkan ketika menghadapi *monster* dengan *AI SOM* dan *BAM HP player* hanya jatuh sampai 70% pada akhir *stage*.

Dengan menggunakan *AI SOM* dan *BAM* masing-masing tipe *monster* jenis aksi yang berbeda dimana melalui pengamatan kelas *mob* melakukan aksi *chase* saja pada awalnya seperti pada gambar 4, lalu ketika *HP monster* berada pada 30% *monster* kelas *mob* melakukan aksi *attack* seperti pada gambar 5.

Kelas *elite mob* hanya melakukan aksi *attack* seperti pada gambar 6, dan memiliki kemungkinan yang kecil untuk melakukan aksi *chase, SPA, defense* (kemungkinan sangat kecil dan acak).

Gambar 4. *Monster bear* kelas *mob* Melakukan aksi *chase*

Gambar 5. *Monster bear* kelas *mob* melakukan aksi *attack*

Gambar 6. *Monster spider* kelas *elite mob* melakukan aksi *attac*k

Kelas *boss* melakukan aksi *attack* seperti pada gambar 7,

Gambar 7. *Monster troll* kelas *boss* melakukan aksi *attac*k

dan melakukan aksi *special attack 1* seperti pada gambar 8,

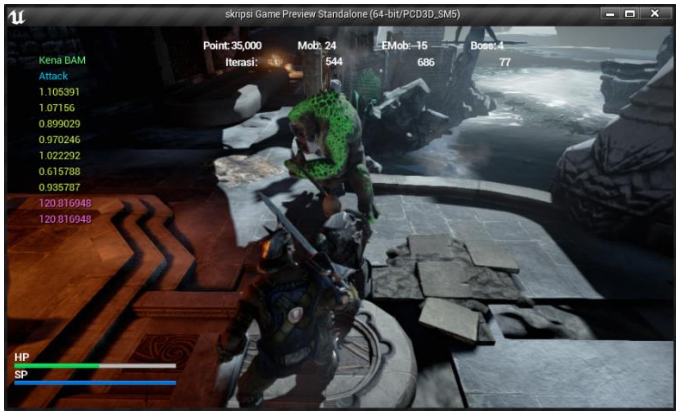
Gambar 8. *Monster troll* kelas *boss* melakukan aksi *SPA*

ketika *HP* berada sekitar 30% . serta memiliki kemungkinan yang kecil untuk melakukan aksi *evade, SPA2, SPA3* (kemungkinan sangat kecil dan acak)

Bagian *BAM* yang menangani penyimpanan gerakkan terakhir berhasil dengan baik. Melalui pengamatan baik kelas *mob, elite mob*, maupun *bos*s dapat melakukan aksi *evade* terhadap gerakan *player* yang sama atau mirip seperti tampak pada gambar 9-11.

Gambar 9. *Monster bear* kelas *mob* melakukan aksi *evade*

Gambar 10. *Monster spider* kelas *elite mob* melakukan aksi *evade*



Gambar 11. *Monster troll* kelas *boss* melakukan aksi *evade*

Setelah training berdasarkan pengamatan, *monster* kelas *mob* melakukan aksi *evade (BAM)* terbanyak yaitu sekitar 6 kali tiap bertarung dengan *player* sedangkan *monster* kelas *elite mob* dan *boss* melakukan aksi *evade* hanya sesekali saja sekitar 3-4 kali tiap pertarungan (probabilitas antara aksi *evade (BA*M) digunakan untuk ketiga kelas *monster* sama yaitu 1/3, 2/3 merupakan milik *SOM*).

Jika aksi yang dilakukan oleh *AI* kombinasi *SOM* dan *BAM* dibandingkan dengan *AI rule based*, maka *AI* kombinasi *SOM* dan *BAM* bisa dibilang mudah, baru lambat laun susah hal ini ditunjukkan melalaui hasil observasi dimana pada awalnya semua *AI* kombinasi *SOM* dan *BAM* hanya melakukan satu gerakan saja misalnya *chase* (kelas mob) namun jika diserang dan *HP*-nya mencapai batas tertentu (sekitar 30% untuk kelas *mob*) maka *AI* kombinasi *SOM* dan *BAM* akan mulai menyerang (kelas *mob*) dan jika *player* pergi ketika *AI* tersebut menyerang *AI* tersebut dapat mengejar *player* lagi (terkadang). Ketika *AI* kombinasi *SOM* dan *BAM* dari suatu kelas sudah mati 4 kali maka proses *retrieval BAM* akan dijalankan sehingga *AI* lebih sering menhindar serangan *player* yang mirip, hal ini yang pada akhirnya membuat *AI* kombinasi *SOM* dan *BAM* memakan waktu lebih untuk dibunuh. Sedangkan pada *AI rule based*, *AI* lebih agresif ketika melihat *player* langsung mengejar *player* lalu ketika sudah dekat langsung melakukan aksi yang diatur menggunakan probabilitas sehingga terkadang aksi yang dilakukan ketika dekat dengan *player* bisa berupa serangan, serangan spesial, bertahan maupun menghindar (terlihat *random*).

Jadi, berdasarkan hasil pengujian dapat dikatakan *AI rule based* lebih sulit (bagi *player*) dibandingkan *AI* gabungan *SOM* dan *BAM* dalam hal *damage output* sedangkan *AI* gabungan *SOM* dan *BAM* lebih sulit dari *AI rule based* dalam hal aksi *evade* yang dilakukan oleh *BAM*. Sehingga dapat dikatakan bahwa hasil pengujian sukses karena bagian *BAM* berhasil melakukan aksi *evad*e terhadap gerakkan *player* yang sama atau mirip sedangkan bagian *SOM* yang menangani bagian *decision making* juga berhasil walaupun tidak sempurna.

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari proses perancangan, pembuatan, dan hasil pengujian Implementasi Kombinasi Jaringan Syaraf Tiruan Metode *Self-Organized Map (SOM)* dan *Bidirectional Associative Memory (BAM)* Pada *AI Game Action Adventure,* dapat diambil kesimpulan sebagai berikut:

- Hasil dari *decision making SOM* tiap kelas *monster* sudah cukup baik namun masih kurang sempurna.
- *SOM* dapat digunakan sebagai bagian *decision making* milik *AI* namun memerlukan banyak *variable input* dan nilai *input SOM* harus bervariasi untuk menghindari *cluster* pemenang yang selalu sama.
- Proses *retrieval BAM* dapat menghasilkan pasangan tidak akurat seperti pasangan yang telah disimpan dalam matriks *BAM*.

5.2 Saran

Beberapa saran untuk mendukung pengembangan aplikasi lebih lanjut di masa yang akan datang adalah:

- Metode input *SOM* dapat diubah dan *data input SOM* dibuat lebih bervariasi.
- Tidak menggunakan *SOM* sebagai bagian *decision making* untuk *AI* tetapi menggunakannya untuk membuat pen-kategorian *rule* milik *AI*.
- Menggabungkan bagian *BAM* dengan *AI Rule Based*.

6. DAFTAR PUSTAKA

[1] Bacao, F., Lobo, V. n.d. *Introduction to Kohonen's Self-Organizing Maps*. Retrieved October 20, 2015, from http://edugi.uji.es/Bacao/SOM%20Tutorial.pdf.

[2] Bhatia, S., Golman, R. 2014. *A Recurrent Neural Network for Game Theoretic Decision Making*. Retrieved October 20, 2015, from https://mindmodeling.org/cogsci2014/papers/337/paper337.pdf.

[3] Bullinaria, J.A. 2004. *Self Organizing Maps : Fundamentals*. Retrieved October 20, 2015, from www.cs.bham.ac.uk/~jxb/NN/l16.pdf.

[4] Lee, J.H., Karlova, N., Clarke, I.R., Thornton, K., & Perti, A. 2014. *Facet Analysis of Video Game Genres*. Retrieved October 20, 2015, from https://www.ideals.illinois.edu/bitstream/handle/2142/47323/057_ready.pdf?sequence=2.

[5] Mänttäri, J., Larsson, J. 2011. *Applications of Artificial Neural Networks in Games; An Overview*. Retrieved October 20, 2015, from http://www.idt.mdh.se/kurser/ct3340/ht11/MINICONFERENCE/FinalPapers/ircse11_submission_5.pdf.

[6] Negnevitsky, M. 2005. *Articial intelligence: a guide to intelligent systems (2nd Ed.)*. England: Pearson Education Limited.

[7] Widagda, I. 2012. Jaringan Syaraf Tiruan. Retrieved October 20, 2015, from https://igawidagda.files.wordpress.com/2012/02/diktat-jst.pdf.

[8] Wuryandari, M.D., Afrianto, I. (2012, March). Perbandingan Metode Jaringan Syaraf Tiruan *Backpropagation* Dan *Learning Vector Quantization* Pada Pengenalan Wajah. *Jurnal Komputer dan Informatika (KOMPUTA)*, 1(1), 46.