Aplikasi *Game* yang Menerapkan Metode *Autonomous NPC* untuk Mengkoordinasi *AI* dari *Enemy*

Evan Sanjaya[1], Gregorius Satia Budhi[2]
Program Studi Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra.
Jln. Siwalankerto 121-131 Surabaya 60236
Telp. (031)-2983455, Fax. (031)-8417658
E-mail: sanjayaevan1@gmail.com[1], greg@petra.ac.id[2]

ABSTRAK: Dalam *action game* belakangan ini, biasanya *player* akan dihadapkan dengan banyak *AI* secara bersamaan. Agar dapat memberikan tantangan terhadap *player* akan dibutuhkan kelakuan yang kompleks. Untuk menciptakan kesan ini, *AI* tersebut harus dapat berkoordinasi dengan yang lain dalam menyerang *player* tanpa menghilangkan *gameplay* yang baik dan seimbang.

Untuk dapat membuat *AI* tersebut, maka akan ditambahkan mekanisme *attack slot, exclusion zone, trigger system, enemy event, player mutex, allowed zone,* dan *blackboard* untuk membantu *AI* saling berkomunikasi dan mengatur sikap dari *AI* dalam menghadapi sebuah situasi.

Dari hasil pengujian, bisa dilihat bahwa *player* dapat melawan sejumlah *AI* sekaligus dan memiliki kesempatan untuk menang. *AI* tersebut juga bisa terlihat berkoordinasi dalam perlawanan mereka terhadap *player*. Masalah yang dapat terjadi adalah dalam penggunaan *attack slot* dan *exclusion zone* harus memperhitungkan jumlah *enemy* yang bisa menempati *slot* yang sama karena bila terlalu banyak dapat megakibatkan *enemy* saling berebut tempat dan menghilangkan kesan berkoordinasi. Dalam penggunaan *player mutex* harus dipastikan pengembaliannya oleh *enemy* yang mengambil. Kegagalan dalam mengembalikan *mutex* tersebut akan membuat semua *enemy* lain tidak dapat menyerang.

Kata Kunci: *Game, AI,* Koordinasi

ABSTRACT: In recent action game, a player will usually be faced against many AI s simultaneously. To give a challange for the player, a complex behaviour will be needed. To give this impression, the AI needs to coordinate with the other in attacking the player without losing a good and balanced gameplay.

To make that AI, it will be given some mechanisms which are attack slot, exclusion zone, trigger system, enemy event, player mutex, allowed zone, and blackboard to help the AI in communicating with each other and control the behaviour of the AI in dealing against a situation

From the test result, it can be seen that the player can fight against numerous AI at once and still have a chance to win. The AI can also be seen coordinating in their fight against the player. Problem that can happen is that in the usage of attack slot and exclusion zone, the number of enemy that occupy a slot need to be considered because if there is too many, it can cause them to fight for the same place and lost the impression of coordination. In the usage of player mutex, returning it need to be assured by the enemy that had taken it. Failure to return the mutex will cause all enemy unable to attack.

Keywords: Game, AI, Coordination

1. LATAR BELAKANG

Pada *action game* belakangan ini, pemain biasanya akan menghadapi banyak musuh yang dikendalikan oleh *AI.* Saat pemain semakin menikmati tantangan yang ada dalam *game*, dibutuhkan kelakuan yang semakin kompleks. Tetapi ini tidak cukup hanya dengan sebuah individu yang kompleks, melainkan dibutuhkan juga untuk mengambil sebuah tindakan pada waktu dan tempat yang tepat. Jika koordinasi ini tidak dilakukan dengan benar maka *NPC (Enemy)* akan bertindak tanpa memperhatikan *NPC* yang lain.

Saat sebuah tindakan akan diambil, ada berbagai faktor yang harus dipertimbangkan seperti jumlah peluru, jenis senjata, dan sisa nyawa sebuah karakter.

Tetapi untuk mengambil sebuah tindakan juga perlu diperhitungkan tindakan dari *NPC* lain. Dengan begitu mereka bisa lebih bekerja sama dan mengurangi kemungkinan untuk mengganggu satu sama lain seperti menghalangi tembakan yang lain atau berusaha menyerang dari posisi yang sama.

Fokus skripsi ini adalah membuat *AI* yang akan dapat mengontrol *NPC* untuk lebih berkoordinasi dengan *NPC* lainnya tanpa meninggalkan *gameplay* yang baik dan seimbang. Yang dimaksud dengan *gameplay* seimbang adalah dimana *AI* (*Enemies*) tidak menyerang pemain bersamaan namun tidak menghilangkan rasa dikeroyok.

2. *FINITE STATE MACHINE*

Finite State Machine merupakan alat yang mempunyai beberapa *state* yang dapat beroperasi berdasarkan *input* untuk melakukan transisi dari satu *state* ke yang lain atau mengakibatkan sebuah *output* atau tindakan terjadi. Sebuah *finite state machine* hanya bisa berada dalam satu *state* setiap saat. [1]

3. *HIERARCHICAL FINITE STATE MACHINE*

Hierarchical Finite State Machine (*HFSM)* merupakan *FSM* yang terdiri dari beberapa *FSM.* Sebuah *state* merupakan tujuan utama yang akan dilakukan dalam *state* tersebut dan di dalamnya terdapat beberapa *state* dengan tujuan sendiri untuk memenuhi tujuan utama tersebut.[2]

4. KOORDINASI MENGGUNAKAN *AUTONOMOUS NPC*

Untuk menunjukkan bahwa *enemy* memiliki kepandaian, maka *enemy* tidak dapat melakukan perhitungan secara individu, tetapi juga melakukan koordinasi terhadap tindakan *enemy* lainnya.
Koordinasi tersebut bisa diciptakan dengan menggunakan beberapa tambahan mekanisme sebagai berikut [3] :

4.1. Attack Slots

Untuk melakukan sebuah serangan, *enemy* akan mencari lokasi terbaik untuk menyerang. Tanpa sebuah pengaturan, *enemy* akan mengambil posisi terdekat terhadap *player*.
Supaya semua *enemy* tidak melakukan hal tersebut dan mengalami kemungkinan menghalani jalan *enemy* yang lain atau bahkan jalan *player* sepenuhnya, maka diberikan sebuah mekanisme *attack slots*. Yang dimaksudkan sebagai *slot* di sini adalah sebuah tanda untuk mengetahui jarak minimum yang boleh diambil *enemy* untuk menyerang *player*. Setelah *enemy* melakukan sebuah serangan maka akan diambil *slot* lain yang lebih jauh dari jarak sekarang dan mengizinkan *Enemy* lain untuk mengambil *slot* yang telah dipakai.

4.2. Exclusion Zones

Dengan menggunakan *attack slots, enemy* akan mengambil jarak terhadap *player*, tetapi bila hanya seperti itu, terdapat kemungkinan bahwa dua atau lebih *enemy* akan mengambil jarak dan posisi yang sama.

Untuk mengatasinya, *exclusion zone* dibuat agar *enemy* melihat posisi *enemy* lain didekatnya. Bila ada *enemy* lain yang terlalu dekat atau berada dalam lintasan serangan dari *enemy* lainnya maka *enemy* tersebut akan mencari posisi lain yang lebih sesuai.

4.3. Trigger System

Sebuah *trigger* adalah sebuah tanda yang akan direspon *enemy* seperti suara serangan dan ledakan. Jika ada *enemy* yamg mendengar suara tersebut, maka *enemy* tersebut akan berjalan menuju sumber suaranya untuk melihat apa yang terjadi. Ini juga bisa berubah tubuh sebuah *enemy* yang telah dijatuhkan oleh *player* sehingga memberitahu *enemy* yang melihatnya bahwa ada penyusup. *Enemy* bisa disetting *trigger* yang mana yang akan menarik perhatiannya. [5]

4.4. Enemy Events

Mekanisme ini digunakan agar *enemy* dapat berkomunikasi dengan *enemy* lainnya. *Enemy* akan berkomunikasi dengan mengriimkan sebuah pesan. Pesan dapat berupa arahan seperti meminta *enemy* lain untuk bergeser bila pandangannya terhalangi. Pesan ini digunakan untuk *enemy* dapat merespon dengan pantas. [6]

4.5. Player Mutex

Supaya semua *enemy* tidak menyerang *player* secara bersamaan, maka diberikan sebuah batasan jumlah *enemy* yang bisa menyerang. *Mutex* di sini adalah sebuah tanda yang memberikan akses kepada *player*. Saat sebuah *enemy* ingin menyerang, *enemy* tersebut harus memiliki *mutex* ini. Jika ada *enemy* yang mau menyerang saat *mutex* tersebut dimiliki *enemy* lain maka dia harus menunggu *enemy* tersebut menyelesaikan serangannya.

4.6. Allowed Zone

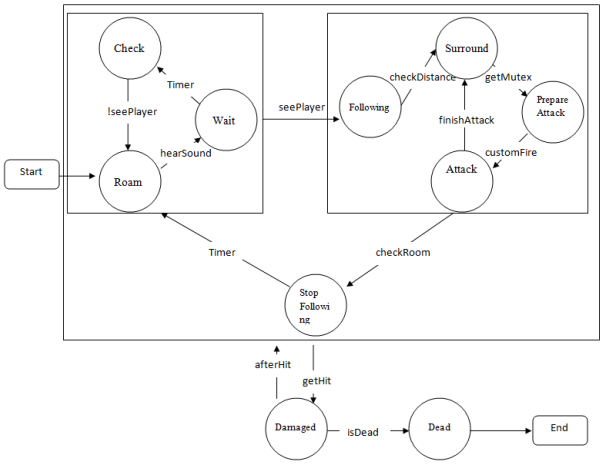
Merupakan daerah yang boleh dilalui oleh sebuah *enemy*. *Enemy* bisa berjalan ke mana saja selama berada dalam daerah ini. *Enemy* bisa memasuk daerah lain hanya bila rute mereka melalui daerah tersebut. Tetapi tujuan dari *enemy* tersebut harus berakhir dalam daerah yang telah ditentukan tadi.

4.7. Blackboards

Sebuah alat untuk bertukar informasi antara *enemy. Enemy* bisa menaruh sebuah informasi di sini yang dapat diakses oleh *enemy* lain dan akan mempengaruhi tindakan yang diambil mereka.[4]

5. Desain Sistem

Pada pembuatan *game* ini, digunakan sebuah *finite state machine* untuk *AI* yang mencakup semua keadaan untuk digunakan konsep koordinasi yang telah dibuat. Diagram dari *FSM* tersebut dapat dilihat pada Gambar 1.



Gambar 1. Diagram *FSM* dari *enemy*

Saat *game* dimulai, *enemy* akan memasuki *state* '*Roam*' di mana *enemy* akan mengelilingi peta sesuai *waypoint* yang tersedia untuk berpatroli. Jika *enemy* mendengar suara (*hearSound*) seperti serangan *player* atau panggilan dari *enemy* lain (*hearCall)* maka *enemy* akan memasuki *state* '*Wait*' untuk melihat keadaan sekitar dan setelah beberapa saat akan memasuki *state* '*Check*' di mana *enemy* akan pergi berjalan menuju sumber suara tersebut untuk melihat apakah ada *player* di sana. Seandainya tidak menemukan *player* maka *enemy* akan kembali ke posisi asalnya dan kembali ke *state* '*Roam*' untuk berpatroli lagi.

Jika dalam salah satu *state* tersebut *enemy* melihat *player* (*seePlayer*), *enemy* tersebut akan menjalankan fungsi *findAttackSlots()* bila bertipe *melee* dan *findAttackSlotsRange()* bile bertipe *range* dari *Black Board* untuk mendapatkan *attackSlot.*

Setelah mendapat *attackSlot, enemy* akan memasuki *state* '*Following*' di mana musuh akan mengejar *player* tersebut. Dengan menggunakan fungsi *checkDistance(),* jika *enemy* telah mencapai jarak tertentu terhadap *player*, *enemy* akan memasuki *state* '*Surround*' untuk mengelilingi *player* dan mengambil jarak sesuai *attackSlot* yang didapatkan sebelumnya.

Selama dalam state ini, *enemy* akan selalu menjalankan fungsi *maintainDistance()* untuk menjaga jarak terhadap *player* bila *player* bergerak dan fungsi *exclusionZone()* untuk menjaga jarak terhadap *enemy* lain agar tidak terlalu dekat satu sama lain.

Beberapa saat setelah mengambil posisi, bila *enemy* bertipe *melee*, maka *enemy* tersebut akan melakukan pengecekan apakah memiliki *attack slot* yang pertama. Bila tidak dalam *slot* yang pertama maka *enemy* tersebut akan menjalankan fungsi *checkSlot()* untuk melihat apakah *slot* pertama atau yang lebih dekat dari yang dimiliki tersedia dan mengambilnya bila iya.

Setelah semua syarat terpenuhi, *enemy* akan melakukan fungsi *getMutex()* dari *Black Board* untuk mengecek apakah tersedia posisi untuk menyerang. Jika tersedia, *enemy* akan memasuki *state* '*PrepareAttack*' untuk mendekat ke *player* untuk tipe *melee* dan menyiapkan anak panah untuk tipe *range*.

Ketika persiapan selesai, *enemy* akan memasuki *state* '*Attack*' untuk melakukan gerakan menyerang terhadap *player* dan kembali kepada *state* '*Surround*' setelah selesai menyerang dan mengembalikan *mutex* agar bisa diambil oleh *enemy* yang lain.

Dalam pengejaran ini, *enemy* akan terus menjalankan fungsi *checkRoom()* dan bila *enemy* telah pindah zona sebanyak dua kali maka *enemy* akan berhenti mengejar, mengembalikan *attackSlot* yang dimilikinya dan memasuki *state* '*StopFollowing*' dan kembali ke posisi awalnya di mana *enemy* akan kembali ke *state* '*Roam*' untuk berpatroli lagi.

Jika *enemy* terkena *damage* dari *player*, *enemy* akan memasuki *state* '*damaged*' di mana enemy menghentikan seluruh proses sementara dan mengembalikan *mutex* bila memilikinya. Kemudian dilakukan pengecekan apabila *enemy* masih hidup, maka *enemy* akan kembali ke *state* sebelumnya dan memasuki *state* '*Dead*' bila *Health* dari *enemy* tersebut 0 dan mengakhiri proses dari *enemy*.

6. PENGUJIAN

6.1. *Pengujian Attack Slot*

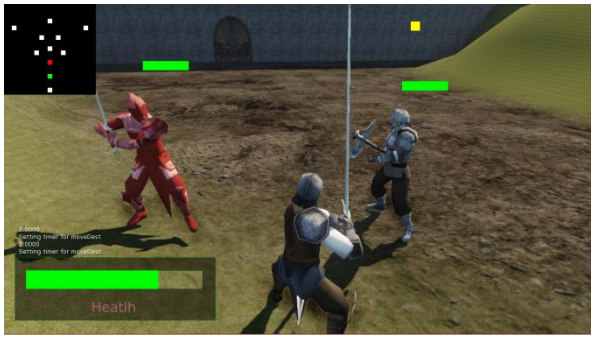
Dalam *game* nantinya, pada saat *enemy* melihat *player* make *enemy* tersebut akan mengambil sebuah *slot* dan mengejar *player*. Setelah mencapai jarak tertentu terhadap *player*, *enemy* akan berhenti mengejar dan beruasaha mengelilingi *player* dengan mengambil jarak tertentu sesuai *slot* yang diambilnya. Pada Gambar 2 dapat dilihat dua *enemy* yang bertipe *melee* dan bertipe *range* di mana yang *melee* mengambil *slot* pertama dan yang *range* mengambil *slot* ketiga dan kotak indikator yang memiliki warna merah dan biru untuk menandakan bahwa *slot* tersebut telah terpakai. *Enemy* yang bertipe *range* mengambil jarak yang lebih jauh terhadap *player* karena *slot* miliknya merupakan *slot* ketiga

Gambar 2 Pengujian *attack slot*

6.2. *Pengujian Exclusion Zone*

Dalam usaha mengelilingi *player*, *enemy* bisa memiliki *slot* yang sama dan hanya memikirkan jarak terhadap *player*. Saat ada dua *enemy* atau lebih yang saling berdekatan dalam pengambilan jarak ini, maka *enemy* tersebut akan saling bergeser. Dapat dilihat pada Gambar 3 dan 4.

Gambar 3 Pengujian *Exclusion Zone 1*.



Gambar 4 Pengujian *Exclusion Zone 2*.

6.3. *Pengujian Trigger System*

Pengujian ini dilakukan dengan menampilkan dua *enemy* di mana yang satu menghadapi *player* dan yang lainnya sedang berjalan menjauh karena sedang berada dalam *state Roaming* dengan tujuan tempat yang lain. Setelah *player* menyerang dan mengenai *enemy* sehingga menimbulkan suara, *enemy* yang berjalan tadi akan bereaksi dan datang melihat sumber suara tersebut. Dapat dilihat pada gambar 5, 6, dan 7.

Gambar **Error! No text of specified style in document.** Pengujian *Trigger System* 1.

Gambar 6 Pengujian *Trigger System* 2.

Gambar 7 Pengujian *Trigger System* 3.

6.4. *Pengujian Enemy Event*

Pengujian ini dilakukan dengan dua keadaan yang berbeda. Yang satu merupakan keadaan di mana hanya satu *enemy* yang melihat *player* dan meminta bantuan kepada *enemy* yang lain. Dapat dilihat pada Gambar 8 dan 9.

Gambar 8 Pengujian *Enemy Event* 1.

Gambar 9 Pengujian *Enemy Event* 2.

Keadaan satunya merupakan keadaan di mana terdapat *enemy* bertipe *melee* dan *range* dan yang bertipe *melee* berada di jalur serangan *enemy* bertipe *range*. *Enemy* bertipe *range* akan meminta *enemy* bertipe *melee* untuk bergeser. Dapat dilihat pada Gambar 10 dan 11.

Gambar 10 Pengujian *Enemy Event* 3.

Gambar 11 Pengujian *Enemy Event* 4

6.5. *Pengujian Player Mutex*

Pada saat *player* dalam keadaan dikelilingi oleh banyak *enemy*, nantinya tidak semua *enemy* akan menyerang secara bersamaaan. *Enemy* akan melihat apakah *mutex* masih tersedia atau sudah diambil oleh *enemy* lain. *Enemy* yang sudah mengambil *mutex* tersebut akan menyerang *player* sedangkan *enemy* yang tidak berhasil mengambilnya akan diam saja dan tetap dalam posisinya. Contoh kejadian ini dapat dilihat pada Gambar 12 dan 13.

Gambar 12 Pengujian *Player Mutex* 1.

Gambar 13 Pengujian *Player Mutex* 2.

6.6. *Pengujian Allowed Zone*

Pada saat *player* menghadapi *enemy*, *enemy* akan selalu berusaha mengejar *player*. Tetapi dengan menggunakan *allowed zone*, bila *player* memasuki sebuah ruangan yang bukan lorong maupun ruangan tempat pertama kali *enemy* tersebut muncul, maka *enemy* tersebut akan berhenti mengejar dan mengelilingi peta sekali lagi. Contoh kejadian ini dapat dilihat pada Gambar 14 dan 15.

Gambar 14 Pengujian *Allowed Zone 1*.

Gambar 15 Pengujian *Allowed Zone*

7. KESIMPULAN

Berdasarkan hasil pengujian, desain yang sudah dirancang dapat sesuai yang diinginkan. *Player* dapat merasakan perlawanan dengan banyak *enemy* tanpa menutup kemungkinan untuk memenangkannya. *Enemy* juga bisa terlihat bekerja sama satu dengan yang lain. Kendala yang dapat terjadi adalah dalam penggunaan *attack slot* dan *exclusion zone* harus memperhitungkan jumlah *enemy* yang bisa menempati *slot* yang sama karena bila terlalu banyak dapat megakibatkan *enemy* saling berebut tempat dan menghilangkan kesan berkoordinasi. Dalam penggunaan *player mutex* harus dipastikan pengembaliannya oleh *enemy* yang mengambil. Kegagalan dalam mengembalikan *mutex* tersebut akan membuat semua *enemy* lain tidak dapat menyerang.

8. REFERENSI

[1] Buckland, M. (2005). *Programming Game AI by Example.* Woodware Publishing.

[2] Adams, E. (2010). *Fundamentals of Game Design second edition.* New Riders.

[3] Garcés, D. (2006). *"Achieving Coordination with Autonomous NPC." Game Programming Gems 6.* Charles River Media.

[4] Isla, D., & Blumberg, B. (2002). *"Blackboard Architectures." AI Game Programming Wisdom.* Charles River Media.

[5] Orkin, J. (2002). *"A General-Purpose Trigger System." AI Game Programming Wisdom.* Charles River Media.

[6] Rabin, S. (2002). • *Rabin, Steve. "Enhancing a State Machine Language Through Messaging." AI Game Programming Wisdom.* Charles River Media.