Journal of Animation and Games Studies, Vol.3 No.1 – April 2017
ISSN 2460-5662

Metode *Trigger Detection* Untuk Gerakan Kendaraan NPC Dalam Game

Wandah Wibawanto

Program Studi Desain Komunikasi Visual, Jurusan Seni Rupa,
Fakultas Bahasa dan Seni, Universitas Negeri Semarang
e-mail: wandah@mail.unnes.ac.id

Abstrak

Artikel ini membahas tentang penggunaan metode trigger detection untuk membuat simulasi lalulintas yang melibatkan kendaraan NPC (*Non Playable Character*) untuk keperluan game. Dalam sebuah game bertipe *racing* atau *open world* seperti Need For Speed, Grand Theft Auto, Watch Dog dan sejenisnya, simulasi kendaraan NPC diperlukan untuk menghasilkan kesan realistis. Dalam membuat alogaritma gerakan kendaraan NPC pada umumnya digunakan metode *pathfinding* dan metode *waypoint*. Alternatif yang dapat digunakan adalah metode *trigger detection*, yaitu dengan menempatkan sejumlah sensor di sekeliling kendaraan. Selanjutnya sensor akan mendeteksi tumbukan antara sensor dengan objek yang berada di lingkungan virtual dalam game. Hasil dari tumbukan tersebut diolah lebih lanjut melalui proses posisi, deteksi, reaksi dan negosiasi agar menghasilkan gerakan yang dinamis. Metode ini selanjutnya diujicoba untuk mengetahui keefektifannya.

Kata kunci: algoritma, kendaraan NPC, metode trigger detection.

Abstrak

This article discusses the use of trigger detection method to simulate traffic involving vehicles NPCS (Non-Playable Character) for the game. In a game of racing or open world like Need For Speed, Grand Theft Auto, Watch Dog and the like, simulating vehicle NPC required to produce a realistic impression. In making the complex algorithms of vehicle motion used method of NPC pathfinding and waypoint method. An alternative that can be used is the method of trigger detection, ie by placing a number of sensors around the vehicle. The next sensor will detect the collision between the sensor and the objects that are in the virtual environments in the game. The result of the collision the further process of reaction, detection, positioning and negotiating in order to generate a dynamic movement. These methods are further tested to know their effectiveness.

Keywords: algorithm, vehicle NPC, trigger detection

Pendahuluan

Permainan elektronik atau game telah menjadi bagian integral kehidupan sehari-hari bagi banyak orang. Game sering menyajikan situasi di mana "pemain terlibat dalam konflik buatan, yang didefinisikan oleh peraturan untuk mencapai tujuan tertentu" (Salen & E. Zimmerman, 2004). Konflik buatan semacam itu sering ditampilkan dalam bentuk teka-teki atau tantangan, dan setelah teka-teki atau tantangan dipecahkan, maka pemain game akan mendapatkan *reward* atau penghargaan tertentu (Liu, Alexandrova, & T. Nakajima, 2011). Dalam praktiknya sebagian besar game menggunakan kecerdasan buatan (AI) untuk memberikan tantangan khusus sebagai lawan (musuh) atau sebagai *non player character* (NPC).

Beberapa penelitian substansial yang berfokus pada pengembangan komponen sistem permainan yang menggunakan AI, dimana AI tersebut mendekati atau meniru gaya permainan manusia. Komponen ini sering disebut sebagai "bots." Motivasi untuk mengembangkan "bot" adalah untuk memberikan pengalaman bermain game yang lebih tinggi karena pemain dituntun untuk percaya bahwa lawan dalam permainan adalah manusia/pemain lain. Contoh dari jenis game ini adalah game *multiplayer first person shooter* Counter-Strike yang sangat populer, dimana tujuannya adalah untuk mengalahkan semua pemain di tim musuh atau lawan. (Salen & E. Zimmerman, 2004)

Contoh algoritma AI lainnya yang diadopsi oleh game antara lain *finite state machines*, *fuzzy state machines*, and *decision trees*. Penggunaan alogaritma AI tersebut dapat ditemukan pada game sederhana seperti permainan catur, game-game klasik seperti pacman, sampai dengan game kompleks (*serious game*) seperti seri Grand Theft Auto. Penggunaan AI tersebut bervariasi tergantung kebutuhan yang diperlukan oleh game, sebuah game dapat menggunakan satu alogaritma atau mengkombinasikan beberapa alogaritma sekaligus.

Seperti halnya game yang lain, pada game bertipe *racing* atau balap beberapa sistem alogaritma AI diterapkan untuk mendapatkan efek realistis. Pemain berharap bisa mengendalikan mobil mereka seperti mereka mengendalikan mobil dalam kehidupan nyata. Sebagai elemen NPC, "bot mobil" atau kendaraan NPC harus melakukan penyesuaian terus menerus terhadap lintasan / jalan dan menyesuaikan secara terus menerus terhadap mobil pemain dan mobil NPC lainnya.

Untuk memastikan "bot mobil" balap dapat meniru perilaku manusia maka diperlukan penerapan beberapa alogaritma AI.

Dalam beberapa game bertipe *open world* seperti seri Grand Theft Auto, Crazy Taxi, Watch Dog, Need For Speed dan sejenisnya memiliki kompleksitas dalam gerakan *Non Playable Character* (NPC) berupa kendaraan NPC. Kendaraan NPC dalam game-game tersebut bergerak dalam ruang terbuka sebagaimana kendaraan umum di jalanan kota untuk menghadirkan kesan realistis. Sebagai contoh dalam game GTA V menyediakan sebuah lingkungan virtual berupa lingkungan kota yang kompleks. GTA V memiliki peta seluas 259 kilometer persegi (100 mil persegi) (Algorithm Design - Best First Search) dengan total populasi 4.000.000 orang. *Game engine* yang digunakan oleh GTA mensimulasikan 262 jenis kendaraan yang berbeda (Vehicles - GTA V wiki guide , 2016), 1.167 model pejalan kaki dan hewan. Lingkungan virtual dalam game GTA V memiliki 77.934 ruas jalan dan 74.530 simpul jalan (Liu, Alexandrova, & T. Nakajima, 2011) yang membentuk jaringan jalan jembatan, terowongan, jalan bebas hambatan, dan persimpangan di perkotaan, pinggiran kota, pedesaan, padang pasir dan lingkungan hutan. Selain itu, GTA5 memiliki 14 jenis simulasi cuaca dan mensimulasikan kondisi pencahayaan selama 24 jam sehari. Terkait dengan NPC berupa mobil, game GTA mensimulasikan lalu lintas dan tanda rambu lalu lintas, masing-masing sesuai standar negara Amerika Serikat. Secara visual, lingkungan virtual yang dihasilkan di GTA V tampil realistis dan sebanding dengan dunia nyata.

Menyusun sebuah alogaritma AI yang kompleks seperti pada game GTA V tentusaja membutuhkan waktu yang lama dan sumber daya yang mahal, sehingga apabila terdapat pengembang game lokal yang ingin mengembangkan game bertipe *open world* seperti GTA V akan menemui beberapa kendala dalam proses pembuatan alogaritma AI, salah satunya alogaritma AI untuk mensimulasikan lalulintas kendaraan NPC. Meskipun demikian, simulasi sistem lalulintas kendaraan dalam game GTA V dapat dijadikan sebagai salah satu referensi metode yang efektif dalam sebuah game bertipe *open world*. Artikel ini secara khusus membahas metode *triger detection* untuk membuat simulasi lalulintas kendaraan NPC, dimana metode ini dapat dilakukan dengan lebih sederhana.

Artificial Intelligence dalam Simulasi Lalu Lintas

Dalam game, kecerdasan buatan digunakan untuk menghasilkan perilaku cerdas terutama pada karakter non-pemain (NPC), yang sering mensimulasikan kecerdasan mirip manusia. Teknik yang digunakan biasanya memanfaatkan metode yang ada dari bidang kecerdasan buatan (*artificial intelligence* / AI). Istilah game AI sering digunakan untuk merujuk pada seperangkat algoritma yang luas yang juga mencakup teknik dari teori kontrol, robotika, grafis komputer dan ilmu komputer pada umumnya. AI memungkinkan teknologi beroperasi dengan berbagai cara, mengembangkan kepribadiannya sendiri dan menerapkan instruksi yang rumit dari pengguna (Eastwood, 2017).

Seperti pada penjelasan sebelumnya, bahwa game bertipe *openworld* melibatkan penggunaan alogaritma AI yang kompleks untuk membentuk sebuah lingkungan virtual yang realistik. Salah satu simulasi yang umum ditampilkan adalah simulasi lalulintas. Dalam pengembangan game yang melibatkan simulasi lalulintas beberapa alogaritma AI yang paling umum digunakan antara lain:

1. Sistem *pathfinding*

   Pathfinding adalah sebuah metode pencarian rute terpendek antara dua titik. metode *pathfinding* mencari rute dengan memulai pada satu titik dan menjelajahi nodus yang berdekatan sampai simpul tujuan tercapai, umumnya dengan maksud untuk menemukan rute yang terpendek. Alogaritma *pathfinding* pada umumnya diaplikasikan pada lingkungan virtual yang berbasis sitem grid (*tiled based*). Contoh umum dari alogaritma *pathfinding* adalah alogaritma *disjkstra*, *breadth-first* dan A* (*A star*).

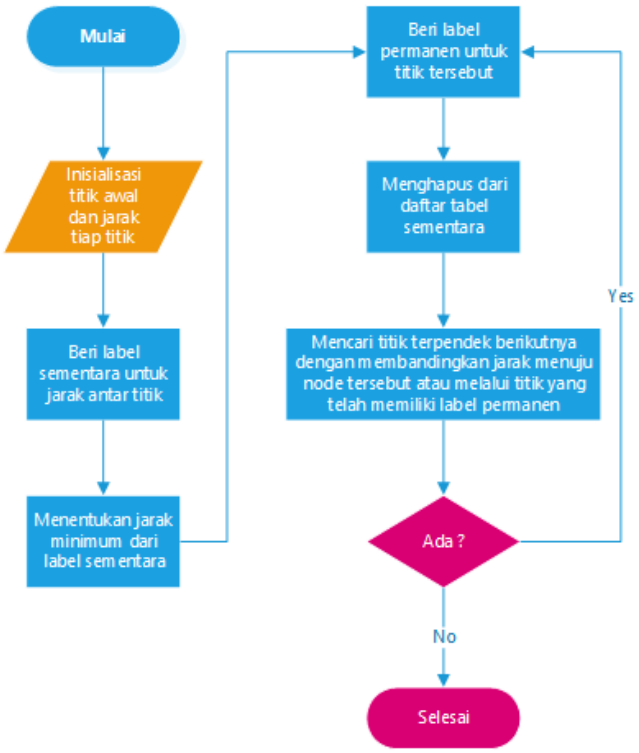
   a. Alogaritma Djiskstra

      Algoritma Dijkstra adalah algoritma untuk menemukan jalur terpendek antara node dalam graf/pohon. Alogaritma ini ditemukan oleh ilmuwan komputer Edsger W. Dijkstra pada tahun 1956 dan diterbitkan tiga tahun kemudian (Sedgewick & Wyne)(Djikstra Algorithm). Djikstra merupakan salah satu varian bentuk algoritma popular dalam pemecahan persoalan terkait masalah optimasi pencarian lintasan terpendek sebuah lintasan yang

mempunyai panjang minimum dari verteks a ke z dalam graf berbobot, bobot tersebut adalah bilangan positif jadi tidak dapat dilalui oleh node negatif. Namun jika terjadi demikian, maka penyelesaian yang diberikan adalah infiniti (Tak Hingga).

Langkah dari metode Djikstra adalah sebagai berikut :

(1) Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi).
(2) Set semua node "Belum Terjamah" dan set node awal sebagai "Node keberangkatan".
(3) Dari node keberangkatan, pertimbangkan node tetangga yang belum terjamah dan hitung jaraknya dari titik keberangkatan. Sebagai contoh, jika titik keberangkatan A ke B memiliki bobot jarak 6 dan dari B ke node C berjarak 2, maka jarak ke C melewati B menjadi 6+2=8. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru.
(4) Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah terjamah sebagai "Node terjamah". Node terjamah tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
(5) Set "Node belum terjamah" dengan jarak terkecil (dari node keberangkatan) sebagai "Node Keberangkatan" selajutnya dan lanjutkan dengan kembali ke langkah 3.



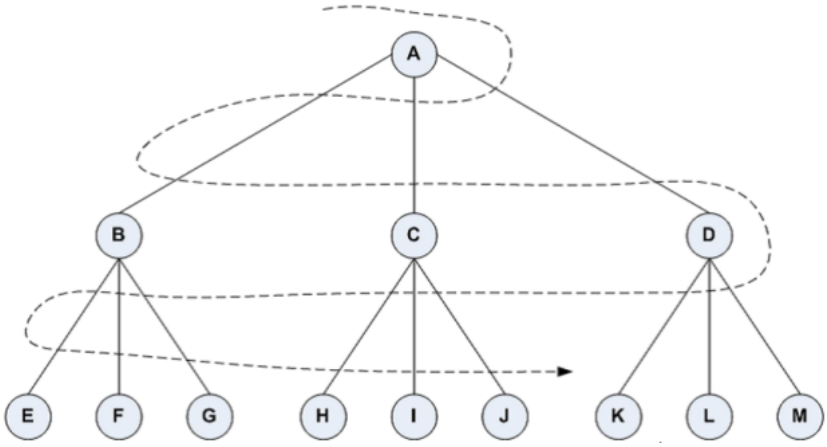
Gambar 1. Flowchart alogaritma Djisktra

b. Alogaritma *Breadth First Search*

Algoritma *Breadth First Search* (BFS) melakukan pencarian secara melebar yang mengunjungi simpul secara *preorder* yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu. Selanjutnya, simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi , demikian seterusnya. Algoritma ini memerlukan sebuah antrian q untuk menyimpan simpul yang telah dikunjungi. Simpul-simpul ini digunakan sebagai acuan untuk mengunjungi simpul-simpul yang bertetanggaan dengannya. Algoritma ini juga membutuhkan table Boolean untuk menyimpan simpul yang telah dikunjungi sehingga tidak ada simpul yang dikunjungi lebih dari satu kali. *Breadth-first search* (BFS) melakukan proses pencarian pada semua node yang berada pada level atau hirarki yang sama terlebih dahulu sebelum melanjutkan proses pencarian pada node di level berikutnya (Algorithm Design - Best First Search).

Langkah dari metode *Breadth frist* adalah sebagai berikut :

(1) Masukkan simpul ujung (akar) ke dalam antrian.
(2) Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi.
(3) Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
(4) Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) ke dalam antrian.
(5) Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan.
(6) Ulangi pencarian dari langkah kedua sampai simpul tujuan ditemukan.



Gambar 2. Alur pencarian rute dengan alogaritma BFS

c. Alogaritma A* (A Star)

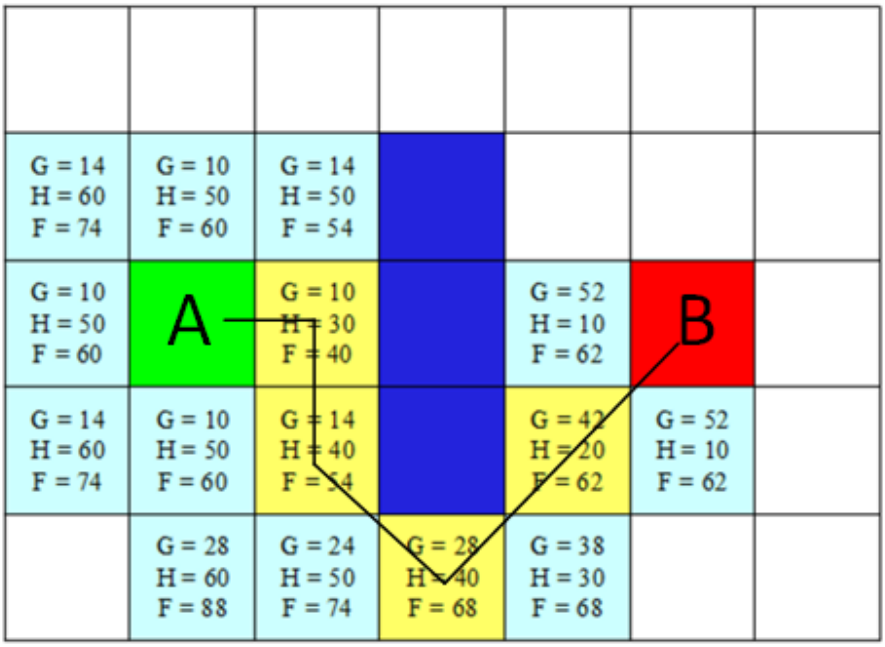
Algoritma A* menggunakan pendekatan heuristik h(x) yang memberikan peringkat ke tiap-tiap titik x dengan cara memperkirakan rute terbaik yang dapat dilalui dari titik tersebut. Setelah itu tiap-tiap titik x tersebut dicek satu-persatu berdasarkan urutan yang dibuat dengan pendekatan heuristik tersebut [2]. Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point*, simpul (*nodes*), A, *open list*, *closed list*, harga (*cost*), halangan (*unwalkable*).

A* memperhitungkan *cost* dari *starting point* ke tujuan dengan fungsi heuristic, selanjutnya *cost* yang telah ditempuh selama ini dari *initial state*

ke *current state* dihitung untuk ditentukan jarak terpendek dengan *cost* terkecil.

Langkah dari metode *A** adalah sebagai berikut :

(1) Masukkan tujuan awal dan tujuan akhir.
(2) Tentukan biaya pergerakan (*cost*) antar simpul, selama simpul tidak terbentur dengan halangan (*unwalkable*)) disimbolkan dengan G..
(3) Menentukan estimasi gerakan, yaitu menghitung biaya pergerakan (*cost*) antar simpul awal dan tujuan tanpa memperhatikan halangan (*unwalkable*) disimbolkan dengan H.
(4) Menentukan nilai tiap pergerakan antar titik yang dilalui (F = G+H).
(5) Ulangi langkah nomor dua sampai empat sampai dengan tujuan ditemukan, selanjutnya ditentukan nilai F terkecil.



Gambar 3. Pencarian rute dari titik A ke B menggunakan alogaritma A*

Contoh penerapan metode *pathfinding* dalam game antar lain adalah simulasi gerakan kendaraan darat dalam game Transport Tycoon. Pada game tersebut bus dan truk dapat bergerak dari satu terminal ke terminal lain dengan metode *pathfinding*.

Gambar 4. Penerapan metode *pathfinding* dalam game Transport tycoon
(Sumber : www.pockettactic.com)

2. Sistem *Waypoints*

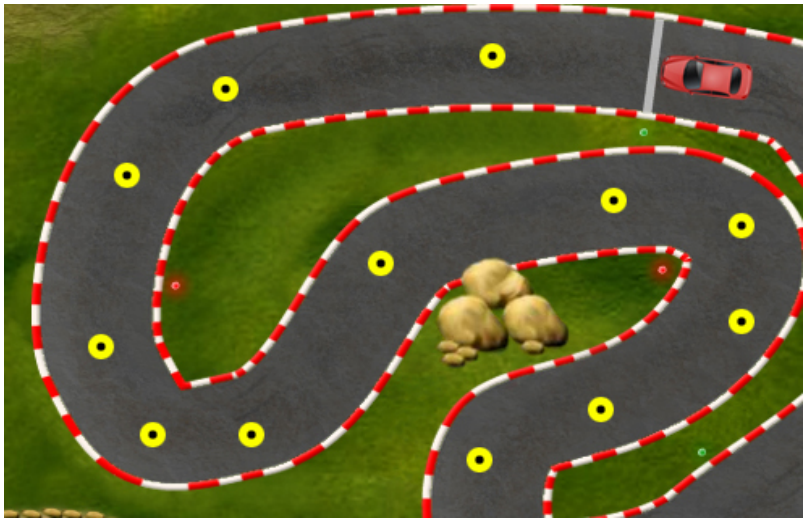
Sistem *waypoint* adalah meletakkan seperangkat titik koordinat dalam ruang 2D/3D yang mewakili posisi kunci di arena lintasan. Sistem ini digunakan oleh banyak game balap mobil generasi awal karena efektifitas dan kesederhanaannya. Untuk mengatur gerakan mobil menuju titik arah (*waypoint*), sistem permainan melakukan serangkaian perhitungan vektor, yang menentukan jumlah rotasi kemudi yang dibutuhkan sistem permainan untuk diberikan ke mobil. Perhitungan ini didasarkan pada vektor awal yang dibuat oleh kedua posisi *waypoint* yang dituju dengan posisi mobil. (T, Christine, & Craig., 2015).

Keunggulan dari metode *waypoint* adalah kemudahan dalam menghitung gerakan mobil di suatu lintasan, serta kecepatan memproses data. Hal ini dikarenakan proses perhitungan yang dilakukan hanya mengacu pada dua titik yang berada di lintasan. Langkah dari metode *waypoint* adalah sebagai berikut:

(1) List posisi kordinat masing-masing *waypoint* ke dalam sebuah data bertipe *array*.

(2) Mulai gerakan dengan perhitungan *waypoint* pertama yang terbaca. Dari perhitungan antara posisi mobil dan *waypoint* akan ditemukan sudut untuk rotasi kemudi mobil.

(3) Gerakkan mobil sesuai dengan pengaturan kemudi yang diperoleh dari perhitungan sebelumnya.
(4) Dengan metode pengukuran jarak (*radius*), apabila mobil mendekati titik kordinat *waypoint* maka perhitungan dilakukan kembali dengan menghitung *waypoint* selanjutnya.
(5) Ulangi langkah nomor tiga sampai empat sampai dengan *waypoint* terakhir.



Gambar 5. Peletakan titik *waypoint* dalam lintasan 2 D
(Sumber : dokumentasi penulis, 2017)

Salah satu contoh penerapan metode *waypoint* adalah pada game Froyo Taxi (http://www.froyogames.com/FroyoTaxi.html). Pada game tersebut, masing-masing kendaraan NPC memiliki *waypoint* mengelilingi kota, sehingga kendaraan NPC akan bergerak secara terus menerus tanpa tumbukan dengan mobil NPC lain. Ketika mobil NPC bertabrakan dengan mobil pemain, maka dengan mudah mobil kembali ke posisi yang benar.

Kelemahan metode *waypoint* adalah gerakan mobil yang sudah terdefinisikan sejak awal, sehingga tidak memiliki variasi gerakan, dapat diprediksi dan dalam beberapa kasus akan terlihat monoton.

Gambar 6. Penerapan metode *wayfinding* pada game Froyo Taxi
(Sumber : dokumentasi penulis, 2017)

Metode *Trigger Detection* Untuk Gerakan Mobil NPC Dalam Game

Metode alternatif yang ditawarkan untuk simulasi kendaraan NPC di lingkungan virtual adalah metode *trigger detection*. Metode ini menggunakan sensor pemicu (*triggrer*) untuk mendeteksi keberadaan objek ketika berada di lingkungan virtual. Mekanisme yang mendukung fitur ini adalah area pemicu (sensor) yang ditambahkan di sekeliling mobil. Sensor yang ditambahkan akan dihitung secara matematis untuk menentukan apakah sensor mengenai objek lain seperti tembok (*unwalkable object*), kendaraan lain, atau objek-objek lain yang berada di lingkungan virtual. Data yang diperoleh akan menentukan gerakan selanjutnya, sebagai contoh ketika sensor bagian depan mengenai mobil lain, maka secara otomatis kecepatan mobil akan dikurangi, sehingga tabrakan akan dapat dihindari.

Dalam praktik pembuatan alogaritma AI untuk mobil NPC hal utama yang harus ditekankan adalah dua tujuan mengemudi yaitu (1) tujuan utama pengemudi adalah "jangan sampai tabrakan", dan (2) “Jalan mana yang akan saya pilih agar sampai di tujuan". Sistem yang dibuat harus mengadaptasi sistem yang berlaku di kehidupan nyata. Sebuah kendaraan NPC dalam sebuah simulasi lalulintas di ruang virtual secara umum harus melakukan beberapa prosedur berikut:

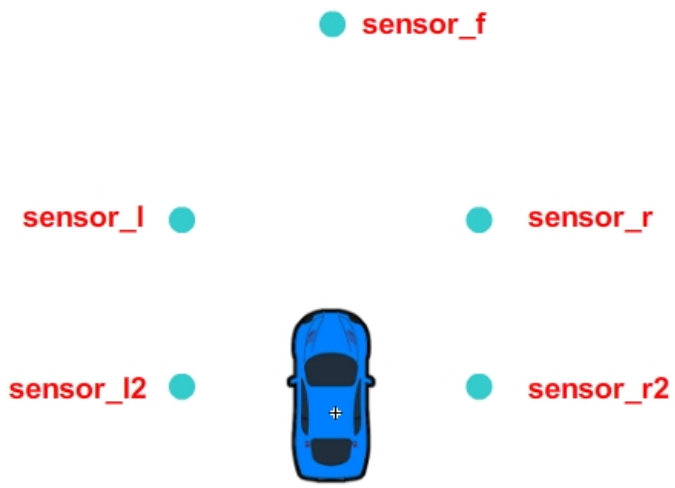
a. Posisi

Kendaraan NPC harus mengetahui kordinat/posisinya terhadap lingkungan virtual. Dalam game bertipe *tile based*, kordinat x, y dapat diketahui dengan mudah sehingga perhitungan kendaraan dan sensornya dapat dilakukan dengan mudah. Sebagai contoh dalam game Sim Taxi dan Ace Gangster baik versi 2D maupun 3D – keduanya menggunakan metode *tile based*, kendaraan NPC yang ada masing-masing membaca posisi *tile X* dan *tile Y*, sehingga dapat mengetahui posisinya terhadap ruas jalan. Ketika kendaraan NPC berada di ruas jalan lurus, tikungan, pertigaan maupun perempatan keputusan dalam mengemudi dapat ditentukan dengan sederhana (Wibawanto, 2015).

Gambar 7. Sim Taxi 2D dan Editor Ace Gangster menggunakan metode *tilebased*
(Sumber : dokumentasi penulis, 2017)

b. Deteksi

Posisi peletakan sensor harus diperhatikan agar melingkupi seluruh area penting yang harus dideteksi. Dalam metode ini, sensor kendaraan NPC diset memiliki 5 sensor, yaitu (1) sensor bagian depan, (2) sensor bagian kanan depan, (3) sensor bagian kiri depan, (4) sensor bagian kanan dan (5) sensor bagian kiri. Perhatikan gambar :



Gambar 8. Peletakan posisi sensor

Perhitungan sensor dapat dilakukan dengan perhitungan trigonometri sederhana sebagai berikut :

```
function find_point(ob:Object):Object {
      var dist:Number = Math.sqrt(ob.x * ob.x + ob.y *
ob.y);
      var rad:Number = Math.atan(ob.y / ob.x);
      var rot:Number = 0;
      if (ob.x > 0) {rot = rad * 180 / Math.PI;}
      else {rot = rad * 180 / Math.PI + 180;}
      return {a:dist, b:rot};
}
car.front_dist = find_point(car.sensor_f).a;
car.front_rotation = find_point(car.sensor_f).b;
```

c. Reaksi

Reaksi adalah respon yang terjadi ketika sensor menyentuh salah satu objek . Reaksi umum yang selalu terjadi dalam simulasi gerak kendaraan NPC dalam lingkungan virtual antara lain (1) sensor tidak menyentuh objek lain (2) sensor mendeteksi persimpangan jalan (3) sensor mendeteksi mobil lain di depannya (4) sensor mendeteksi rintangan (*unwalkable*). Perhatikan contoh reaksi berikut :

```
var px_front = ob.x + ob.front_dist * Math.cos((ob.front_rot
+ ob.rotation) * Math.PI / 180);
var py_front = ob.y + ob.front_dist * Math.sin((ob.front_rot
+ ob.rotation) * Math.PI / 180);
```

```
//jika sensor depan mendeteksi adanya mobil lain maka mobil
mulai mengerem
if (other_car.hitTestPoint(px_front,py_front,true)){....  }
```

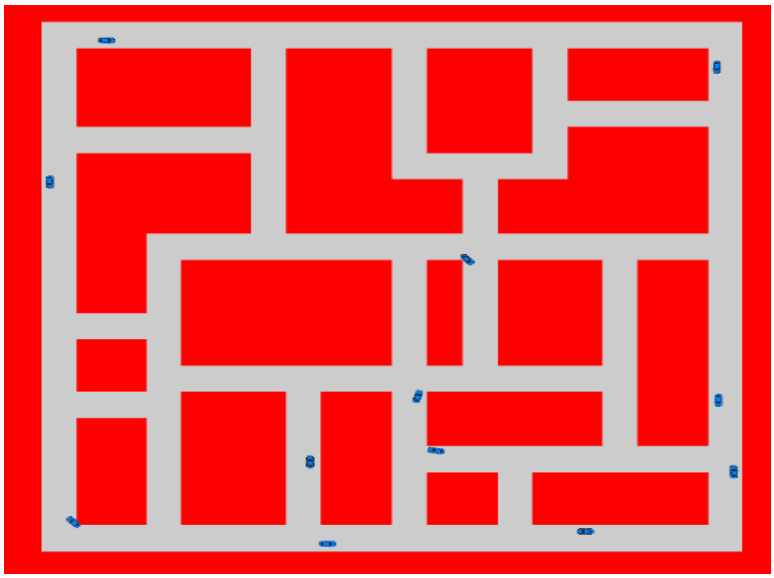
d. Negosiasi

Adalah prosedur ketika kendaraan NPC dihadapkan pada beberapa alternatif pilihan. Sebagai contoh apabila terdapat kondisi mobil mendeteksi adanya persimpangan jalan, maka harus diputuskan apakah mobil bergerak lurus atau belok. Pengambilan keputusan ini dapat dilakukan dengan metode *pathfinding*, akan tetapi karena kendaraan NPC tidak harus memiliki tujuan khusus atau dalam artian kendaraan NPC cukup berputar-putar di dalam lingkungan kota, maka metode *pathfinding* tidak diperlukan. Pada akhirnya, metode pengambilan keputusan dapat dilakukan dengan metode *fuzzy logic*.

Dalam percobaan simulasi lalu lintas, menggunakan metode *trigger detection* beberapa kondisi negosiasi dapat ditetapkan sebagai berikut:

(1) Mobil selalu berjalan di ruas kanan (standar Amerika).
(2) Jika mobil mendeteksi adanya persimpangan jalan maka akan dilakukan pengacakan angka (random) dengan kemungkinan 1:2. Dengan logika *fuzzy* ditentukan mobil bergerak lurus atau belok.
(3) Jika mobil mendeteksi adanya tembok di depan maka sensor kanan dan kiri akan aktif dan mendeteksi adanya area kosong. Maka mobil akan belok menuju ke ruas jalan yang terbuka.
(4) Jika mobil mendeteksi adanya mobil lain di depannya maka negosiasi yang terjadi adalah mobil menghitung kecepatan mobil yang ada di depannya. Jika mobil di depannya lebih lambat, maka mobil akan mendeteksi kondisi jalan di kanan dan kiri untuk kemungkinan menyalip. Akan tetapi jika menyalip tidak memungkinkan, maka mobil akan melakukan pengereman.
(5) Jika mobil keluar lintasan (jalan), maka mobil akan berhenti sampai kondisi tertentu untuk mengaktifkan kembali mobil.

Dengan beberapa kondisi negosiasi tersebut sebuah simulasi lalulintas sederhana yang melibatkan 12 kendaraan NPC dalam sebuah peta 2D menghasilkan gerakan yang dinamis dan kemungkinan tabrakan antar kendaraan NPC yang sangat kecil (T, Christine, & Craig., 2015).



Gambar 9. Simulasi lalulintas dengan metode trigger detection
(Sumber : www.wandah.org)

Penerapan Metode *Trigger Detection* dalam Game

Metode *trigger detection* diterapkan oleh penulis dalam game Ace Gangster 2. Game Ace Gangster 2 dikembangkan menggunakan Actionscript 3 dan Flare 3D. Game tersebut menggunakan metode *tilebased* dalam membentuk lingkungan virtual. Selanjutnya game menjalankan 40 kendaraan NPC dalam lingkungan 100 x 100 *grid* (10K X 10 K *pixel*). Penerapan metode *trigger detection* dalam game tersebut cukup efektif karena hanya meningkatkan penggunaan memory sebesar 13 *KiloByte*.

Gambar 10. Penerapan metode trigger detection dalam game Ace Gangster 2
(Sumber : www.wandah.org)

Kesimpulan

Dari beberapa metode alogaritma kecerdasan buatan (*Artificial Intellegence*) untuk menggerakkan kendaraan NPC (*Non Playable Character*) dalam game, secara umum developer menggunakan metode pathfinding dan waypoint. Kedua metode tersebut memiliki kelebihan dan kelemahan masing-masing. Alternatif lain yang ditawarkan dalam penelitian ini adalah metode trigger detection, yaitu dengan menempatkan sejumlah sensor di sekeliling kendaraan. Selanjutnya sensor akan mendeteksi tumbukan antara sensor dengan objek yang berada di lingkungan virtual dalam game. Hasil dari tumbukan tersebut diolah lebih lanjut melalui proses posisi, deteksi, reaksi dan negosiasi untuk menghasilkan kesan realistik dalam game.

Metode tersebut telah diujicobakan dalam 2 bentuk simulasi : (1) simulasi dengan metode artbased dengan melibatkan 12 kendaraan NPC, dimana simulasi

menghasilkan sebuah sistem lalulintas sederhana, kendaraan NPC bergerak secara dinamis dan tidak bertabrakan sepanjang proses ujicoba, (2) simulasi lalu lintas dengan metode tilebased dalam game Ace Gangster 2 dengan melibatkan 40 kendaraan NPC pada lingkungan virtual 100 x 100 grid. Kedua uji coba tersebut menunjukkan bahwa pemakaian metode trigger detection dapat dilakukan dengan sederhana, menggunakan sedikit memory dan menghasilkan simulasi lalulintas yang dinamis/realistis.

Referensi

Salen, K., & E. Zimmerman. (2004). *Rules of Play: Game Design Fundamentals* (Vol. 1 ed.). MIT Press.

Counter-Strike. (n.d.). Retrieved from Counter-Strike: http://www.valvesoftware.com/games/css.html.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). "Engineering route planning ...algorithms". In *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation* (pp. 117-139). Springer.

Eastwood, G. (2017). *How video game AI is changing the world.* CIO.

GTA V Wiki Guide. (2016). Retrieved 2016, from IGN: http://www.ign.com/wikis/gta-5

Sedgewick, R., & Wyne, K. (n.d.). *Algorithm, Fourth Edition.* Retrieved from Algorithm, Fourth Edition, Shortest Path: http://algs4.cs.princeton.edu/44sp/

Djikstra Algorithm. (n.d.). Retrieved from Djikstra Algorithm: http://www.ms.unimelb.edu.au/~moshe@unimelb/620-261/dijkstra/dijkstra.html

Paths (GTA V). (2016). Retrieved from Paths (GTA V): http://gta.wikia.com/wiki/Paths_(GTA_V)

Vehicles - GTA V wiki guide . (2016). Retrieved from IGN: http://www.ign.com/wikis/gta-5/Vehicles

T, T. M., Christine, W., & Craig., G. (2015). Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques. *nternational Journal of Computer Games Technology. , 2015*.

Liu, Y., Alexandrova, T., & T. Nakajima. (2011). *Gamifying Intelligent Environments.* Retrieved from Gamifying Intelligent Environments: ttp://www.dcl.cs.waseda.ac.jp/~yefeng/yefeng/pubs/2011/ubimui11_yefeng.pdf

Algorithm Design - Best First Search. (n.d.). Retrieved from http://ww3.algorithmdesign.net/: http://ww3.algorithmdesign.net/handouts/BFS.pdf

Wibawanto, W. (2015). *Kecerdasan Buatan - Simulasi Lalu Lintas dengan Flash AS3*. Retrieved from http://www.wandah.org/AI-gerakan-mobil-otomatis: http://www.wandah.org/AI-gerakan-mobil-otomatis