Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 10, Oktober 2018, hlm. 3827-3834 http://j-ptiik.ub.ac.id

Implementasi *Multi-Agent Path Finding* Menggunakan Algoritma *Conflict-Based Search* Pada Game Bergenre *Adventure-Puzzle*

Fathony Teguh Irawan[1], Eriq Muh. Adams Jonemaro[2], Muhammad Aminul Akbar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]fathony.teguh@gmail.com, [2]eriq.adams@ub.ac.id, [3]muhammad.aminul@ub.ac.id

Abstrak

Penggunaan *artificial intelligence* pada *video game* bukanlah merupakan hal baru lagi. Pada umumnya, kebanyakan *video game* sekarang memanfaatkan *artificial intelligence* pada entitas di *video game*-nya. Permasalahan yang terjadi adalah pengaruh dari implementasi *artificial intelligence* terhadap performa dari suatu *video game.* Permasalahan dari *multi-agent pathfinding* adalah untuk mencari jalur untuk lebih dari satu agen. *Constraint* yang dimiliki oleh *multi-agent pathfinding* adalah tidak boleh ada lebih dari satu agen yang menempati tempat yang sama pada waktu yang sama. Pada penelitian ini, peneliti mengajuan untuk menggunakan algoritma *conflict-based search* dalam menyelesaikan *multi-agent pathfinding*. Perancangan *multi-agent pathfinding* adalah dengan merancang algoritma *Conflict-based search* pada *high-level searching*, dan *A* search* pada *low-level search*. Implementasi *multi-agent pathfinding* berdasarkan perancangan yang sudah didefinisikan, diimplementasikan pada *game engine* Unity3D dengan menggunakan bahasa pemrograman C#. Proses pengujian dilakukan setelah implementasi selesai dan didapati hasil dimana tidak ada lebih dari satu agen yang menempati tempat yang sama pada waktu yang sama. Penggunaan *resource* dari *conflict-based search* menunjukkan bahwa penggunaan *resource* yang stabil. FPS yang didapatkan pada *multi-agent pathfinding* dapat dikatakan *playable*. Namun, waktu *cycle* yang dihasilkan terus bertambah seiring bertambahnya jumlah agen. Dengan implementasi ini diharapkan pengembang *game* tidak kesulitan dalam membangun *multi-agent pathfinding* pada *video game*-nya.

Kata kunci: *Video game, Artificial intelligence, Multi-agent pathfinding, Conflict-based search.*

Abstract

Artificial intelligence usage in video games is not new thing anymore. In general, most of video games now take advantage of artificial intelligence on entities in their video games. The problem that occurs is the impact of the implementation of artificial intelligence on the performance of video game. Multi-agent pathfinding is there to try to find path for the agents registered. Multi-agent pathfinding has some constraint that doesn't let more than one agent occupying same place at the same time. In this study, researcher propose to use conflict-based search algorithm to solve multi-agent pathfinding constraint. The design of multi-agent pathfinding is by designing conflict-based search for high-level search and A search for low-level search. Implementation of multi-agent pathfinding is based on the design that has been defined, implemented on Unity3D game engine using C# programming language. Testing process is conducted after the implementation is completed and found the result is there is no condition where more than one agent occupy same place at the same time. Resource usage of conflict-based search also shown to be at stable rate. FPS that obtained from multi-agent pathfinding is still considered as playable. However, cycle time that generated while running conflict-based search continuous to growth as the number of agents increases. With this implementation is expected to help game developer to implement multi-agent pathfinding in their video game.*

Keywords: *Video game, Artificial intelligence, Multi-agent pathfinding, Conflict-based search.*

1. PENDAHULUAN

Pada *video game* ber-*genre adventure* dan turunannya, setidaknya terdapat *NPC (non-player character)* yang ada pada dunia tersebut. Pada *video game*, agar *NPC* dapat memiliki perilaku yang diinginkan perlu diberikan

artificial intelligence yang diatur sesuai fungsi dan tujuan dari NPC itu sendiri. *Artificial Intelligence* adalah teknik yang digunakan dalam *video game* yang menerapkan tiruan dari kecerdasan manusia terhadap perilaku dari *NPC.* Terdapat beberapa langkah dalam membangun *artificial intelligence* pada *video game*. Pertama adalah dengan melakukan tindakan evaluasi perilaku *NPC* dalam *video game* tersebut. Setelah itu adalah dengan menentukan teknik *artificial intelligence* yang akan diimplementasikan pada *NPC.* (Millington & Funge, 2009)

Salah satu tugas *artificial intelligence* dari *NPC* yang berada pada sebuah dunia *video game adventure-puzzle* adalah melakukan perpindahan tempat dari tempat satu ke tempat lainnya. Dalam implementasinya, agar *NPC* melakukan tindakan tersebut algoritma *path finding* digunakan dalam implementasinya.

Namun, permasalahan terjadi saat melakukan implementasi algoritma *path finding* ke banyak agen. Berdasarkan tersebut, maka perlu dirumuskan suatu solusi untuk membuat sistem *path finding* yang dapat menyelesaikan permasalahan *path finding* terhadap banyak agen pada suatu simulasi.

Multi-agent path finding (MAPF) adalah sistem yang dapat digunakan untuk menyelesaikan permasalahan diatas. Langkah yang perlu dilakukan adalah dengan menggunakan *Conflict-based search* sebagai *high-level searching* dan menggunakan algoritma *A* search* sebagai *low-level searching*. (Boyarski, et al., 2015)

Penelitian ini akan membahas bagaimana membangun *NPC* pada *video game* ber-*genre adventure-puzzle*. Diharapkan dengan adanya penelitian ini dapat memberikan panduan kepada pengembang *video game* untuk membangun *NPC* pada *video game* ber-*genre adventure-puzzle* secara optimal dan sesuai perancangan.

2. DASAR TEORI

2.1. *Multi-agent pathfinding*

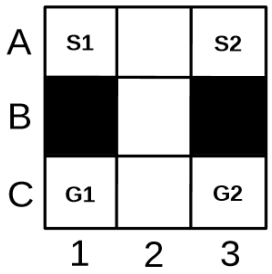
Sistem *multi-agent path finding* adalah sistem untuk pencarian jalur terhadap banyak agen. Kondisi yang harus dipenuhi adalah agen tidak boleh menempati tempat yang sama pada waktu yang sama. (Wang & Botea, 2008)

Algoritma yang dapat digunakan untuk membangun sistem ini adalah dengan menggunakan algoritma *conflict-based search* pada *high-level searching*-nya.

2.2. *Conflict-based search*

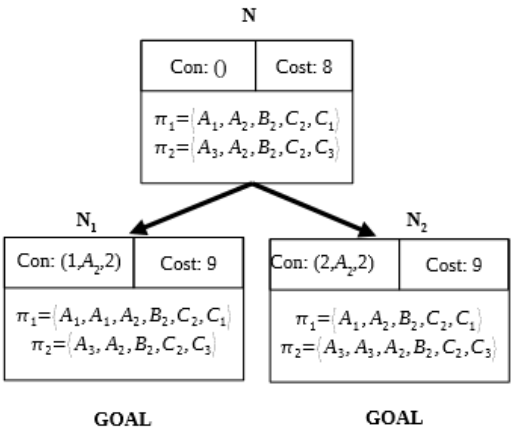
Conflict-based search adalah algortima yang diusulkan untuk dapat menyelesaikan permasalahan *multi-agent path finding*. Fokus dari algoritma *conflict-based search* itu sendiri adalah untuk mencari sebuah *conflict* dari *input* yang diberikan kemudian menghasilkan *constraint*. Dalam mencari *conflict* pada *input* yang diberikan, algoritma mencarian *conflict* juga harus didefinisikan.

Hubungan antara *high-level searching* dan *low-level searching* adalah *output* dari *low-level searching* yang pada penelitian ini menggunakan *A* search*, dijadikan *input* pada *high-level searching* yang pada penelitian ini menggunakan *Conflict-based search*, yang kemudian *output* dari *Conflict-based search* digunakan sebagai jalur untuk agen tersebut.



Gambar 1. *Contoh permasalahan multi-agent pathfinding*

Penyelesaian dari permasalahan diatas adalah dengan cara menjalankan *low level* dahulu, kemudian hasil dari salah satu agen, dijadikan *constraint* pada agen lain sehingga jalur agen lainnya dapat beradaptasi dengan memanfaatkan inputan dari agen lainnya. Berdasarkan permasalahan pada gambar 1, dapat ditemukan sebuah solusi yang tidak memiliki *conflict* pada jalur yang dibuat, bentuk solusi dalam bentuk *conflict tree* dapat dilihat pada gambar 2.



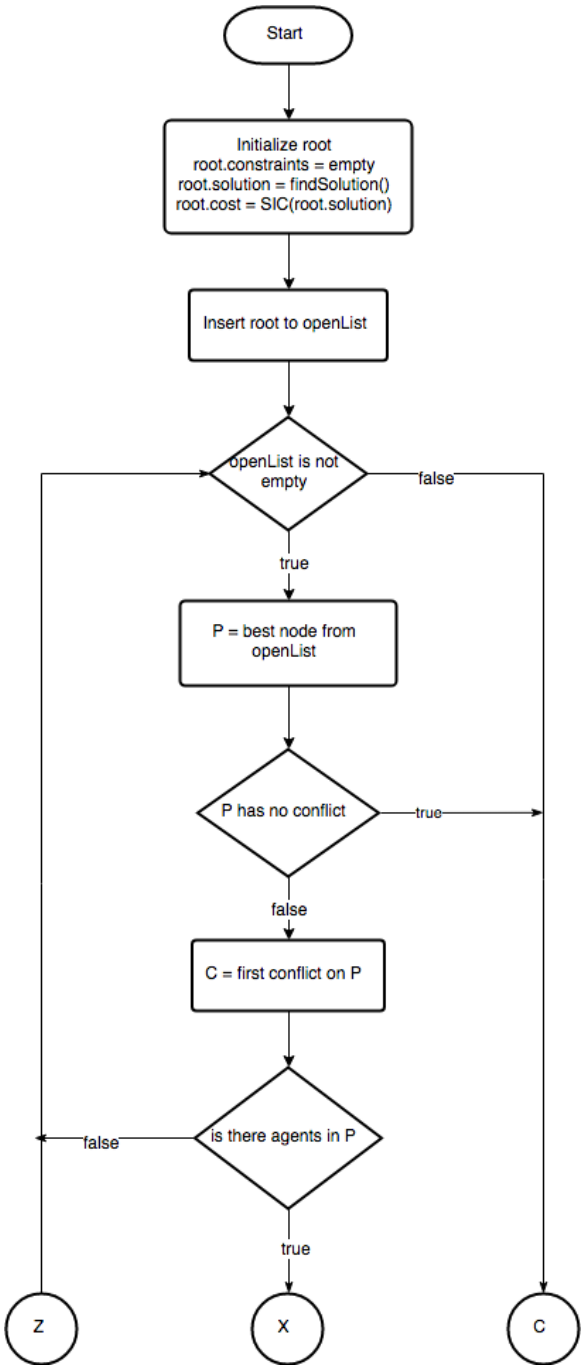
Gambar 2. *Solusi dari multi-agent pathfinding menggunakan CBS*

3. PERANCANGAN

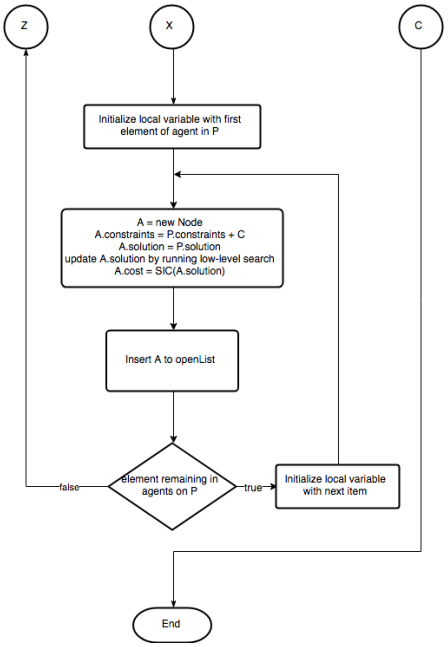
3.1. Perancangan *high-level conflict-based search*

Pada penelitian ini, algoritma *conflict-based search* digunakan untuk melakukan penyelesaian *conflict* pada *high-level*. Pencarian secara *high-level* dilakukan untuk mencari *conflict* dari *solution* yang disediakan pada *node solution* saat itu, yang kemudian hasil pencarian *conflict* tadi digunakan sebagai *constraint* dalam *low-level searching* yang akan memanfaatkan algortma *A* searching*. Algoritma dari *conflict-based search* akan digambarkan dalam bentuk flowchart pada gambar 3 dan gambar 4.

Berdasarkan flowchart yang ada pada gambar 3 dan gambar 4, dapat dilihat flow dari *conflict-based search*. Proses pertama adalah menginisialisasi *root* dari *CT (conflict tree)*. Sebuah *node* memiliki data berupa *constraint* untuk menunjukkan *constraint* berupa waktu, tempat, dan agen apa yang tidak boleh menempati tempat tersebut pada waktu tersebut, data *solution*, dan data *cost*. Kemudian *root* dimasukkan ke dalam openList. Lalu dilakukan pemeriksaan apakah openList sedang tidak kosong, apabila benar, maka variabel *P* diisi dengan *best node* yang ada d dalam openList. Setelah itu, variabel *P* diperiksa apakah memiliki *conflict* atau tidak. Apabila tidak terdapat *conflict* maka nilai *P* dapat dikembalikan dan dapat diasumsikan bahwa nilai *P* adalah *node* yang tidak memiliki *constraint* dan memiliki nilai *cost* terkecil. Metode pengecekan *conflict* adalah dengan memeriksa *solution* pada *node* tersebut. Kemudan, apabila terdapat *conflict* pada nilai *P*, *conflict* digunakan untuk membuat *constraint*. Kemudian diinisialisasi *node* baru berdasarkan *constraint* yang ada pada nilai *P*, nilai *solution* didapatkan dengan menjalankan *low-level search* menggunakan *A* search* dengan mengacu pada *constraint* yang ada. Setelah tu, *node* yang baru dibuat tersebut dimasukkan ke dalam openList dan proses pengecekan apakah openList sedang kosong dan dilakukan kembali seperti pada awal paragraf.



Gambar 3. *Flowchart conflict-based search*

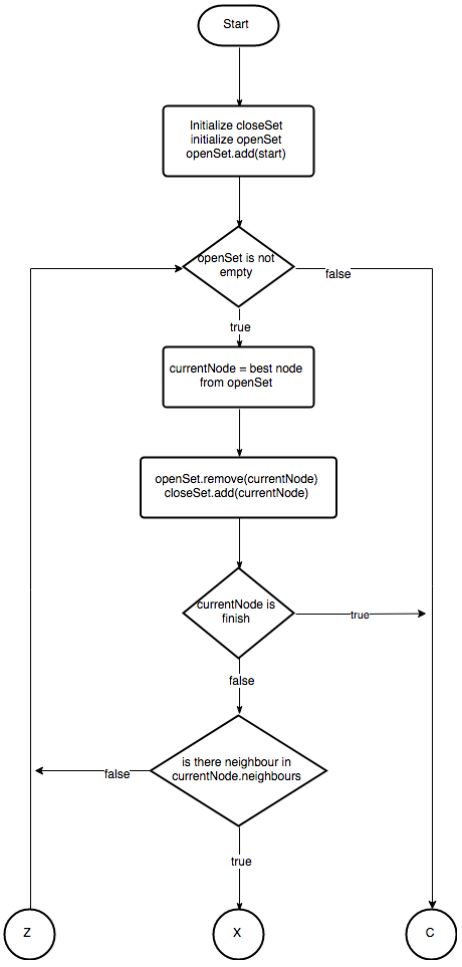


Gambar 4. *Flowchart conflict-based search (lanjutan)*

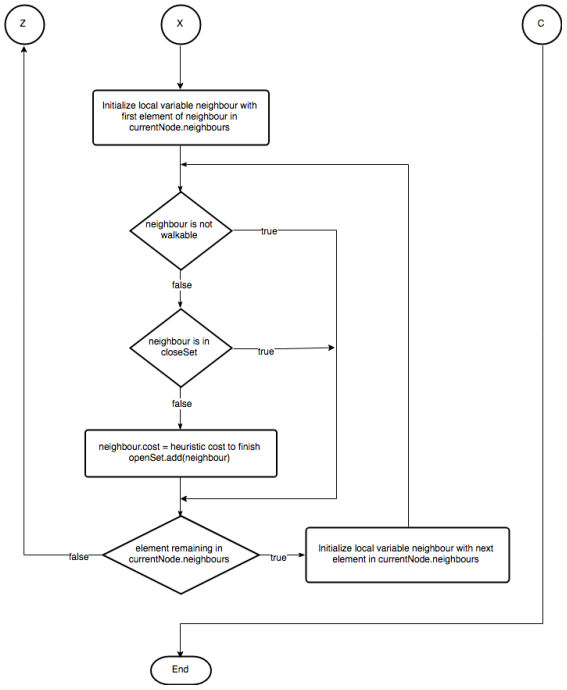
3.2. Perancangan *low-level A* search*

Pada penelitian ini, algoritma *A* search* digunakan untuk melakukan penyelesaian *pathfinding* pada low-*level*. Sesuai yang telah digambarkan pada flowchart dari *conflict-based* search dimana fungsi dari *conflict-based search* berguna untuk mendeteksi adanya *conflict* pada suatu node, penggunaan *A** search digunakan untuk mencari suatu jalur dari suatu *cell* ke *cell* lainnya pada suatu *grid.* Cara kerja dari algoritma ini adalah, diberikan agen, posisi awal, dan posisi akhir, algoritma *A* searching* akan melakukan pencarian rute terdekat tengan mempertahikan peta dimana suatu *node* dapat dilewati atau tidak. *Algoritma* dari *A* search* akan digambarkan pada gambar 5 dan gambar 6.

Berdasarkan flowchart yang ada pada gambar 5 dan gambar 6, dapat dilihat flow dari *A* search* sebagai *low-level search*. Sebelum dilakukan *A* search*, representasi *grid* pada level harus sudah ada karena proses *A* search* menggunakan representasi *grid* pada level tersebut. Nilai yang dimasukkan ke *A* search* adalah agen yang ingin dicari *path*-nya, dan representasi *grid* pada level. Proses awal adalah dengan menginisialisasi openSet, dan closeSet. Kemudian dicari lokasi agen pada representasi *grid* kemudian lokasi tersebut dimasukkan pada openList. Kemudian dilakukan pemeriksaan apakah openSet tidak kosong, apabila openSet tidak kosong, nilai currentNode diambil dari *node* terbaik pada *openSet*. Cara mencari *node* terbaik pada openSet adalah dengan mengambil dari sekumpulan *node* yang ada pada openSet dengan nilai *cost* terkecil. Kemudian currentNode dihapus dari openSet dan closeSet. Dilakukan pemeriksaan apakah currentNode adalah titik akhir dari suatu agen maka *path* telah ditemukan. Apabila currentNode bukan merupakan tujuan akhir dari agen, maka dilakukan pemeriksaan *neighbours* dari currentNode. Pemeriksaan *neighbours* dilakukan dengan mendapatkan *node* tetangga pada currentNode. Pada *2D space*, *neighbours* dari suatu *node* didapatkan dari posisi atas, kiri, kanan, bawah, kiri-atas, kiri-bawah, kanan-atas, dan kanan-bawah dari suatu *node.* Kemudian dilakukan iterasi terhadap masing-masing *neighbours* dan memeriksa apakah *neighbour* sudah terdapat pada closeList atau tidak bisa ditempati. Setelah itu dilakukan kalkukasi *cost* pada *neighbour* tersebut dengan menggunakan metode heuristic dan memasukkan *neighbour* tersebut ke dalam openList dan proses dilakukan kembali seperti pada awal paragraf hingga bertemu dengan *path* yang diinginkan.



Gambar 5. *Flowchart A* searching*



Gambar 6. *Flowchart A* searching (lanjutan)*

4. IMPLEMENTASI

Implementasi keseluruhan dari komponen yang telah didefinisikan diimplementasikan pada *game engine* Unity3D dengan menggunakan bahasa pemgrograman C#.

Dasar dari implementasi *conflict-based search* adalah berdasarkan hasil perancangan *conflict-based search* mengacu pada flowchart gambar 3 dan gambar 4. Implementasi *conflict-based search* diperlukan untuk menyelesaikan *conflict* apabila terdapat *conflict* yang terjadi.

Kemudian dasar dari implementasi *A* search* adalah berdasarkan dari hasil perancangan *A* search* seperti pada flowchart gambar 5 dan gambar 6. Implementasi *A* searching* dilakukan untuk dapat mencari jalur bagi agen menuju titik berhentinya.

Setelah itu, implementasi dari keseluruhan sistem yang nantinya digunakan sebagai media dalam pengujian dan analisis. Hasil dari implementasi dapat dilihat pada gambar 7.

Gambar 7. *Implementasi keseluruhan*

5. PENGUJIAN DAN ANALISIS

3.1. Pengujian *path* masing-masing agen

Pada pengujian ini, dilakukan validasi dari *path* yang dihasilkan oleh *conflict-based search*. Hasil pengujian dapat dilihat pada tabel 1. Dapat dilihat bahwa tidak ada 2 agen atau lebih yang menempati nilai x dan y yang sama pada suatu waktu pada timestep.

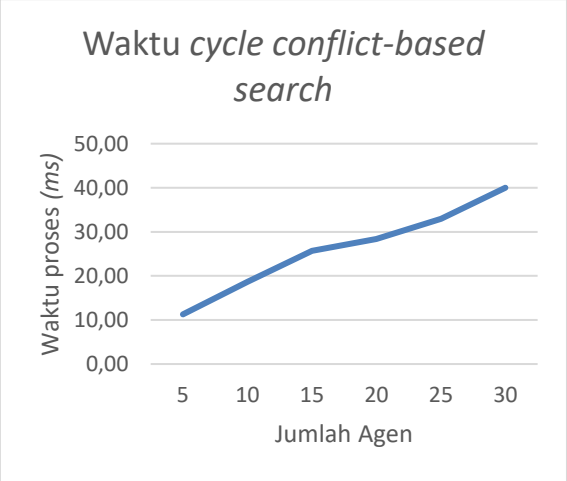
Tabel 1. *Hasil pengujian path dari masing-masing agen*

Timestep	Agent							
	1		2		3		4	
	x	y	x	y	x	y	x	y
1	8	20	28	20	18	19	18	2
2	9	19	27	19	18	18	18	3
3	10	18	26	18	18	17	18	4
4	11	17	25	17	18	16	18	5
5	12	16	24	16	18	15	18	6
6	13	15	23	15	18	14	18	7
7	14	14	22	14	18	13	18	8
8	15	13	21	13	18	12	18	9
9	16	12	20	12	18	11	18	10
10	17	11	19	11	17	10	18	11
11	18	11	18	10	18	9	18	12
12	19	10	17	9	18	8	18	13
13	20	9	16	8	18	7	18	14
14	21	8	15	7	18	6	18	15
15	22	7	14	6	18	5	18	16
16	23	6	13	5	18	4	18	17

17	24	5	12	4	18	3	18	18
18	25	4	11	3	18	1	18	19
19	26	3	10	2	18	0	18	20
20	27	2	9	1	-	-	18	21
21	28	1	8	0	-	-	-	-
22	29	0	7	0	-	-	-	-

3.2. Pengujian waktu *cycle* dari *conflict-based search*

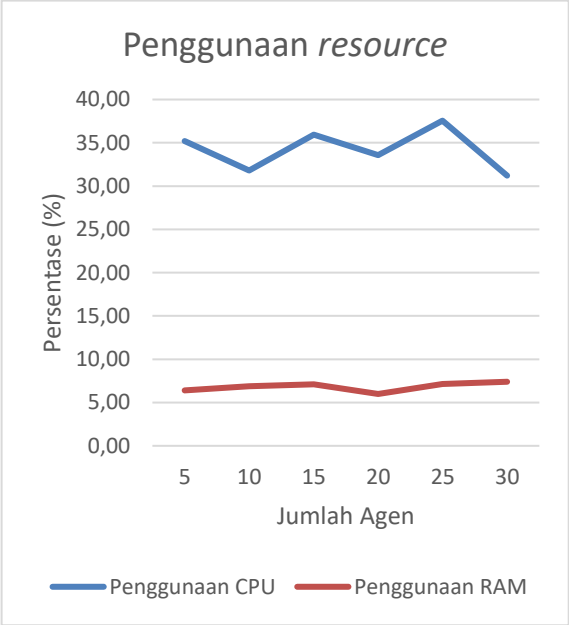
Untuk mendapatkan data waktu eksekusi dari algoritma *conflict-based search*, dilakukan pengambilan waktu yang dibutuhkan dalam menyelesaikan 1 *cycle* dari *conflict-based search*. Data yang diambil adalah rata-rata dari waktu eksekusi *conflict-based search* pada 10 detik pertama, yang kemudian data tersebut dapat digambarkan menjadi grafik garis pada gambar 8.



Gambar 8. *Grafik garis waktu cycle conflict-based search*

3.3. Pengujian penggunaan *resource conflict-based search*

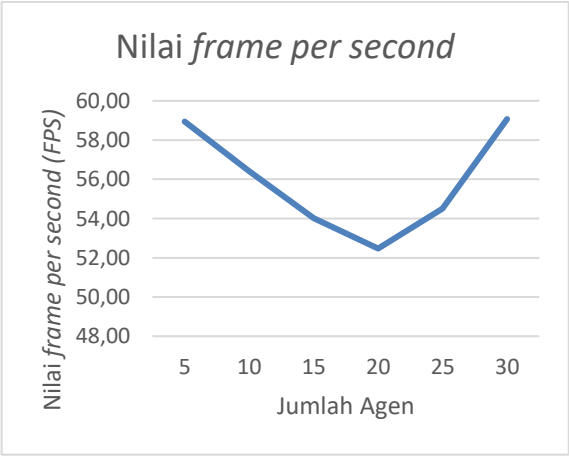
Untuk mendapatkan data *resource* yang dibutuhkan dari algoritma *conflict-based search*, dilakukan pengambilan *usage* dari CPU dan RAM. Data yang diambil adalah nilai *usage* CPU dan RAM pada 10 detik pertama, data diambil setiap detik kemudian data tersebut dapat gambarkan menjadi grafik garis pada gambar 9.



Gambar 9. *Grafik garis penggunaan resource*

3.4. Pengujian *frame per second conflict-based search*

Untuk mendapatkan data *frame per second* yang dibutuhkan dari algoritma *conflict-based search*, dilakukan pengambilan nilai *frame per second* itu sendiri. Data yang diambil adalah nilai *frame per second* pada 10 detik pertama, data diambil setiap detik yang kemudian data tersebut dapat digambarkan menjadi grafik garis pada gambar 10.



Gambar 10. *Grafik garis nilai frame per second*

6. KESIMPULAN DAN SARAN

Perancangan dan membangun *multi-agent patfinding* dilakukan dengan perancangan *dataset* dari level pada *video game*, dan perancangan atribut dan aksi dari agen yang ada pada *video game*. *Dataset* dari level adalah

bentuk representasi level pada *video game* tersebut dalam bentuk *grid*. Kemudian dilakukan implementasi berdasarkan perancangan yang telah didefinisikan pada *game engine* Unity3D dengan menggunakan bahasa pemrograman C#.

Untuk merancang dan membangun *conflict-based search* dilakukan proses perancangan algoritma dari *conflict-based search* itu sendiri. Karena *conflict-based search* hanya digunakan untuk menyelesaikan *conflict* pada *high-level*, maka algoritma lain perlu digunakan untuk menyelesaikan permasalahan pada *low-level*. Perancangan algoritma dari *A* search* diperlukan untuk menyelesaikan permasalahan pada *low-level*. Setelah perancangan selesai, proses implementasi dilakukan berdasarkan perancangan yang telah didefinisikan. Implementasi dilakukan dengan menggunakan bahasa pemrograman C#.

Multi-agent pathfinding diuji dengan mendefinisikan skenario pengujian terlebih dahulu. Setelah pengujian dilakukan berdasarkan skenario pengujian yang didefinisikan, hasil dari pengujian menampilkan bahwa jalur yang dibuat oleh algoritma *conflict-based search* sebagai *high-level searching* dan *A* search* sebagai *low-level searching* menunjukkan bahwa tidak terdapat suatu *conflict* dimana terdapat lebih dari satu agen menempati tempat yang sama pada waktu yang sama. Pada skenario pengujian selanjutnya yaitu dengan mengambil nilai *resource* yang dibutuhkan menujukkan bahwa meskipun jumlah agen yang digunakan terus bertambah, kebutuhan akan *CPU Usage* dan *RAM Usage* tetap stabil, hal ini menunjukkan bahwa penggunaan *conflict-based search* sangatlah evisien dalam menyelesaikan *multi-agent pathfinding*. Pada skenario selanjutnya, nilai FPS pada hasil pengujian menunjukkan bahwa jumlah agen tidak terlalu mempengaruhi dari nilai FPS karena dapat dilihat bahwa jarak FPS yang muncul adalah berkisar dari 52 FPS hingga 59 FPS. Hal ini dianggap normal karena performa pemain pada *video game* mencapai pada titik maksimal pada *rate* 30 FPS. (Claypool & Claypool, 2009) Namun, pada skenario pengujian untuk menghitung waktu yang dibutuhkan dalam menjalankan *conflict-based search* dapat ditemukan bahwa seiring bertambahnya jumlah agen yang diuji, waktu yang dibutuhkan terus bertambah. Hal ini disebabkan karena pada proses *low-level searching*, *A* search* untuk masing-masing agen dilakukan secara sekuensial.

Conflict-based search dapat digunakan untuk menyelesaikan permasalahan *multi-agent pathfinding*. Masih banyak ruang untuk dikembangkan lagi pada sisi *low-level searching* dimana pemrosesan *A* searching* untuk masing-masing agen masih dilakukan secara sekuensial. Saran yang dapat diberikan adalah dengan mengimplementasikan *parallel processing* untuk *low-level searching*-nya karena pada dasarnya tidak ada hubungan kuat antar agen pada level *low-level searching*.

7. DAFTAR PUSTAKA

Lee, J. H., Clarke, R. I., Karlova, N., Thornton, K., & Perti, A. (2014). Facet Analysis of Video Game Genres. *iConference 2014*, 131.

Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games Second Edition*. Burlington, Massachusetts: Morgan Kaufmann.

Prato, G. D., Feijoo, C., & Simon, J.-P. (2014). Innovations in the Video Game Industry: Changing Global Markets. *Digiworld Economic Journal*, 17-18.

Cui, X., & Shi, H. (2011). A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*, 125-130.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., & Schaeffer, J. (2012). A* Variants for Optimal Multi-Agent Pathfinding. *Association for the Advancement of Artificial*, 19-25.

Wang, K.-H. C., & Botea, A. (2008). Fast and Memory-Efficient Multi-Agent Pathfinding. *Association for the Advancement of Artificial*, 1-8.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., & Shimony, E. (2015). ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. *International Joint Conference on Artificial Intelligence*, 740-746.

Claypool, M., & Claypool, K. (2009). Perspectives, Frame Rates and Resolutions: It's all in the Game. *FDG '09 Proceedings of the 4th International Conference on Foundations of Digital Games*, 42-49.

Banerjee, B., & Davis, C. E. (2016). Multi-agent Path Finding with Persistence Conflicts

. *IEEE Transactions on Computational Intelligence and AI in Games*, 402-409.

Subakti, H. (2013). *Rancang Bangun NPC (Non-Player Characters) Pada Game Bergenre Tower Defense*. Universitas Brawijaya, Informatika/Ilmu Komputer, Malang.