BAB 4 PERANCANGAN

4.1 *Dataset* Karakter *video game*

Sebagaimana yang sudah dijelaskan sebelumnya pada bab sebelumnya, kasus yang diangkat pada penelitian ini adalah untuk menyelesaikan permasalahan *multi-agent pathfinding* beserta *constraint* yang dimiliki oleh permasalahan tersebut. Setiap agen yang akan bergerak pada dunia memiliki atribut dan sejumlah aksi. Penjelasan mengenai atribut dan aksi yang dapat dilakukan masing-masing agen dapat dilihat pada tabel 4.1 dan tabel 4.2.

Tabel 4.1 Tabel Atribut

Atribut	**Penjelasan**
ID	*Identifier* bagi agen AI. Nilai ID tiap agen diatur sebelum simulasi dijalankan. ID digunakan untuk membedakan agen satu dengan agen lainnya.
Move speed	Atribut *move speed* digunakan untuk mendefinisikan kecepatan gerak dari agen.
Rotation speed	Atribut *rotation speed* digunakan untuk mendefinisikan kecepatan rotasi dari suatu agen.
Patrol list	Atribut *patrol list* mendefinisikan daftar lokasi yang akan digunakan sebagai lokasi *patrol* dari agen.
Active patrol	Atribut ini digunakan untuk menentukan *patrol* mana yang sedang aktif yang berdasarkan pada *patrol list.*

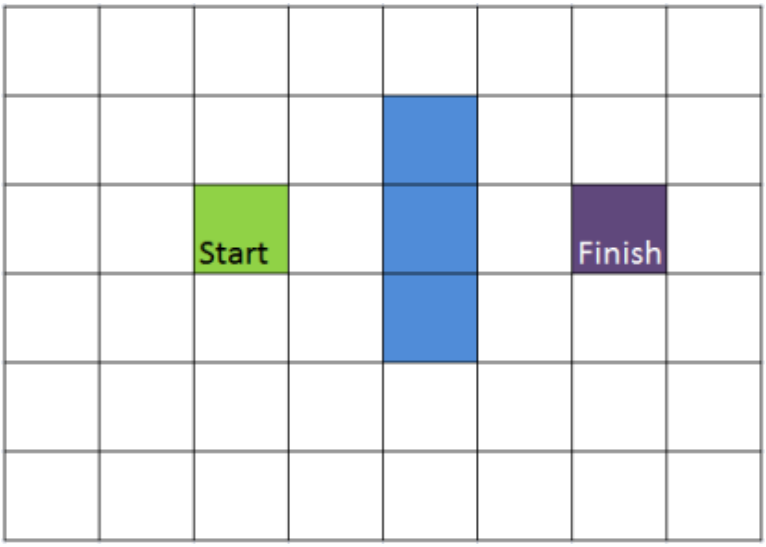
Tabel 4.2 Tabel Aksi

Aksi	**Penjelasan**
Find path	Aksi ini digunakan untuk mencari jalan menuju lokasi *patrol* yang sedang aktif.
Move	Aksi ini digunakan untuk menggerakkan agen menuju *patrol* yang aktif berdasarkan *path* yang dihasilkan dari aksi *Find path.*
Next patrol	Aksi ini digunakan untuk merubah *patrol* aktif menuju *patrol* selanjutnya yang telah didefinisikan pada atribut *patrol list.*

4.2 Perancangan level grid *game*

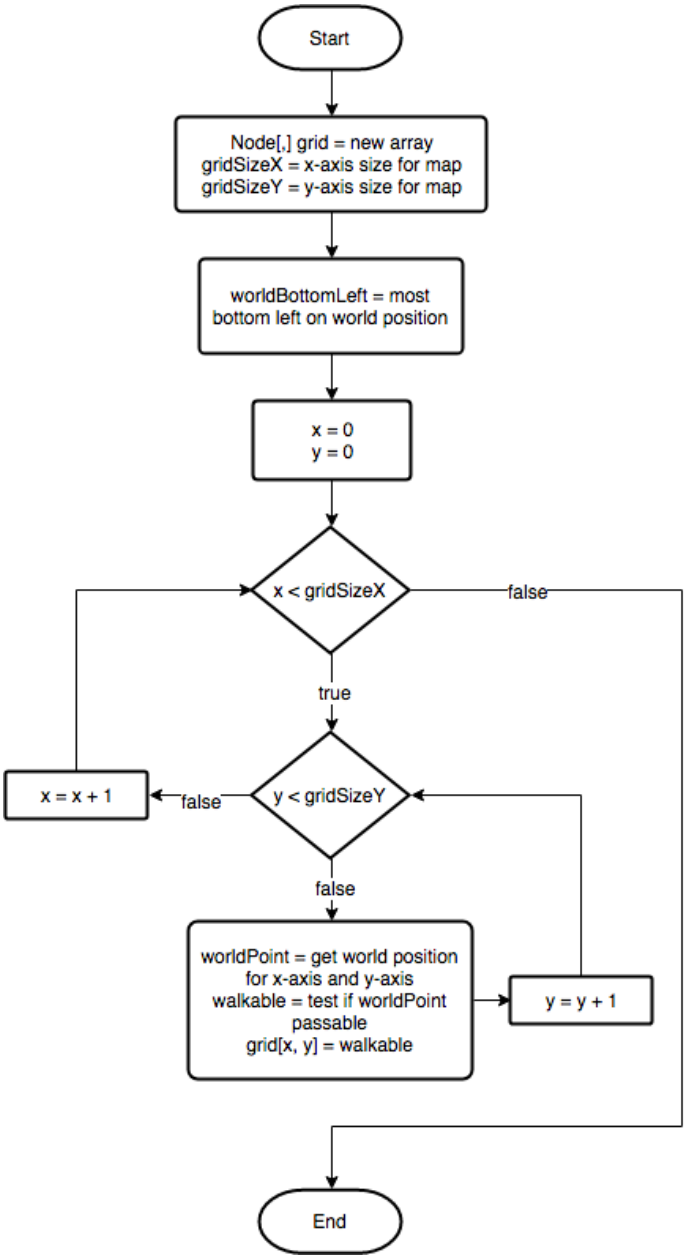
Pada penelitian ini, pembuatan desain level dari *game* didapatkan dari level yang tersedia, kemudian dilakukan pembuatan level dalam representasi *grid*. Representasi *grid* adalah bentuk representasi level 3D dalam dunia 2D yang tersusun atas kotak yang ukurannya telah ditentkan sebelumnya. Terdapat dua *state* dari suatu *cell* pada *grid*, yaitu *passable* dan *unpassable*. *State passable* berarti suatu *cell* pada *grid* tersebut dapat diduduki oleh suatu agen, sedangkan *state unpassable* adalah suatu *state* dimana *cell* pada *grid* tersebut tidak dapat diduduki oleh suatu agen. Apabila pada suatu level memiliki ukuran 500x300 unit sedangkan ukuran cell adalah 1x1, maka dapat didapatkan sebuah *grid* dengan ukuran 500x300 dengan ukuran *cell* sebesar 1x1.

Berikut adalah gambar 4.1 yang menampilkan contoh suatu representasi level dalam bentuk *grid* pada *video game*. Dapat dilihat pada gambar 4.1, bahwa representasi level yang berawal dari level 3D dirubah menjadi dalam bentuk *grid*, dimana karakteristik level berupa *grid* adalah berbentuk persegi yang disusun sedemikian rupa sehingga dapat menggambarkan level semula. Pada gambar dibawah, diberikan kode warna biru muda untuk kotak yang tidak bisa ditempati oleh agen dan diberikan warna putih untuk kotak yang bisa ditempati oleh agen. Pada penelitian ini, masing-masing agen akan memiliki lokasi *grid* mulai untuk mendefinisikan lokasi awal dari agen, dan agen juga memiliki lokasi *grid* akhir untuk mendefinisikan lokasi akhir dari agen tersebut.



Gambar 4.1 Contoh representasi level dalam bentuk grid

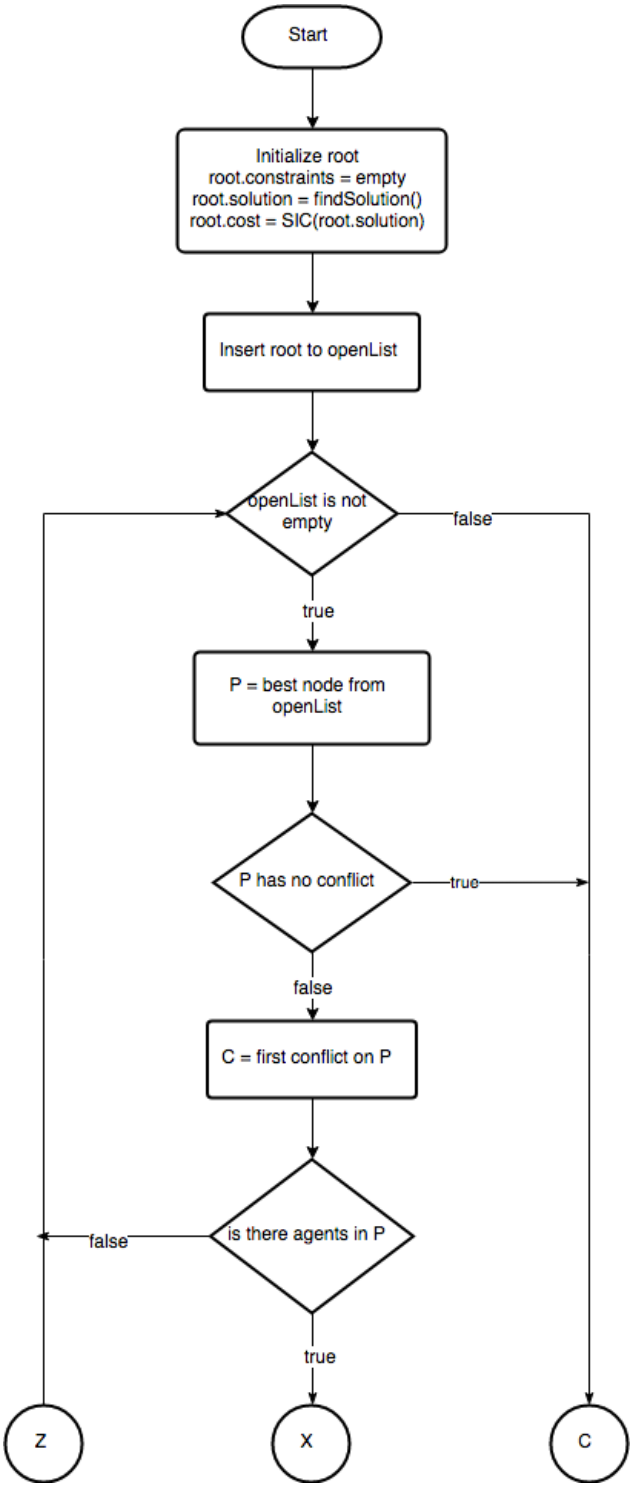
Algoritma dari pembuatan *grid* seperti gambar diatas akan digambarkan dalam bentuk *flowchart* yang ada pada gambar 4.2.



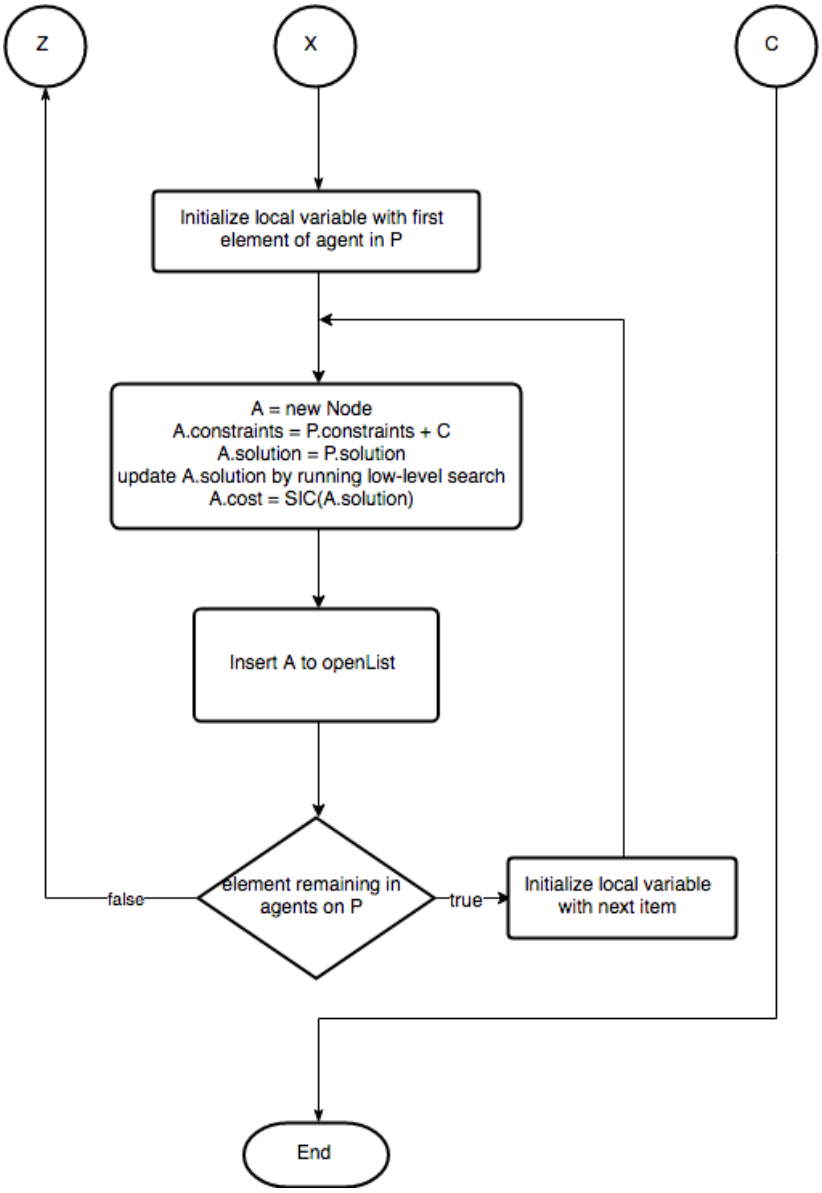
Gambar 4.2 Flowchart buat *gridmap*

4.3 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 4.3 dan gambar 4.4.



Gambar 4.3 Flowchart *conflict-based search*



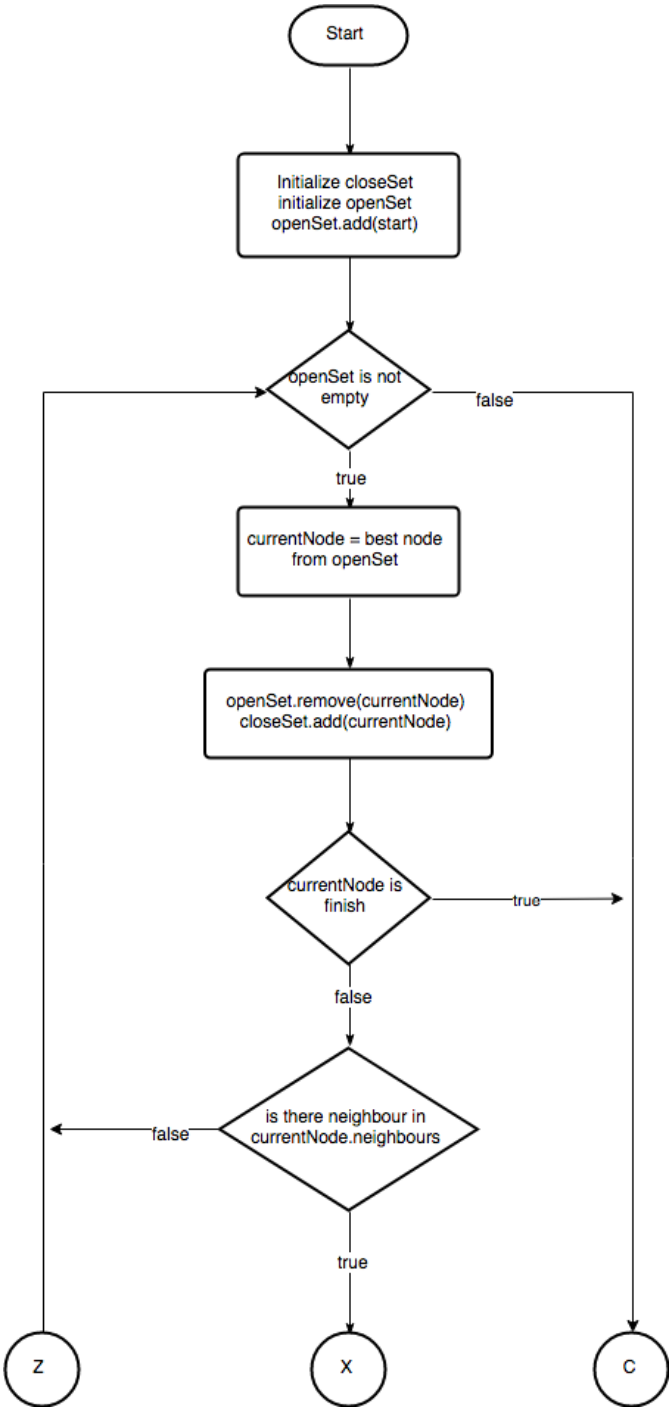
Gambar 4.4 Flowchart *conflict-based search* (lanjutan)

Berdasarkan flowchart diatas, dapat disimpulkan bahwa algoritma *conflict-based search* berfungsi untuk mencari suatu node dalam *nodeList* yang memiliki *cost* terkecil dan tidak terdapat *conflict* pada *solution*-nya. Apabila terdapat *node* dengan *cost* terkecil dan memiliki *conflict* pada *solution*-nya, maka informasi *conflict* tersebut digunakan untuk membuat *constraint* pada *node* selanjutnya. Kemudian *constraint* tersebut dijadikan acuan pada *low-level searching*.

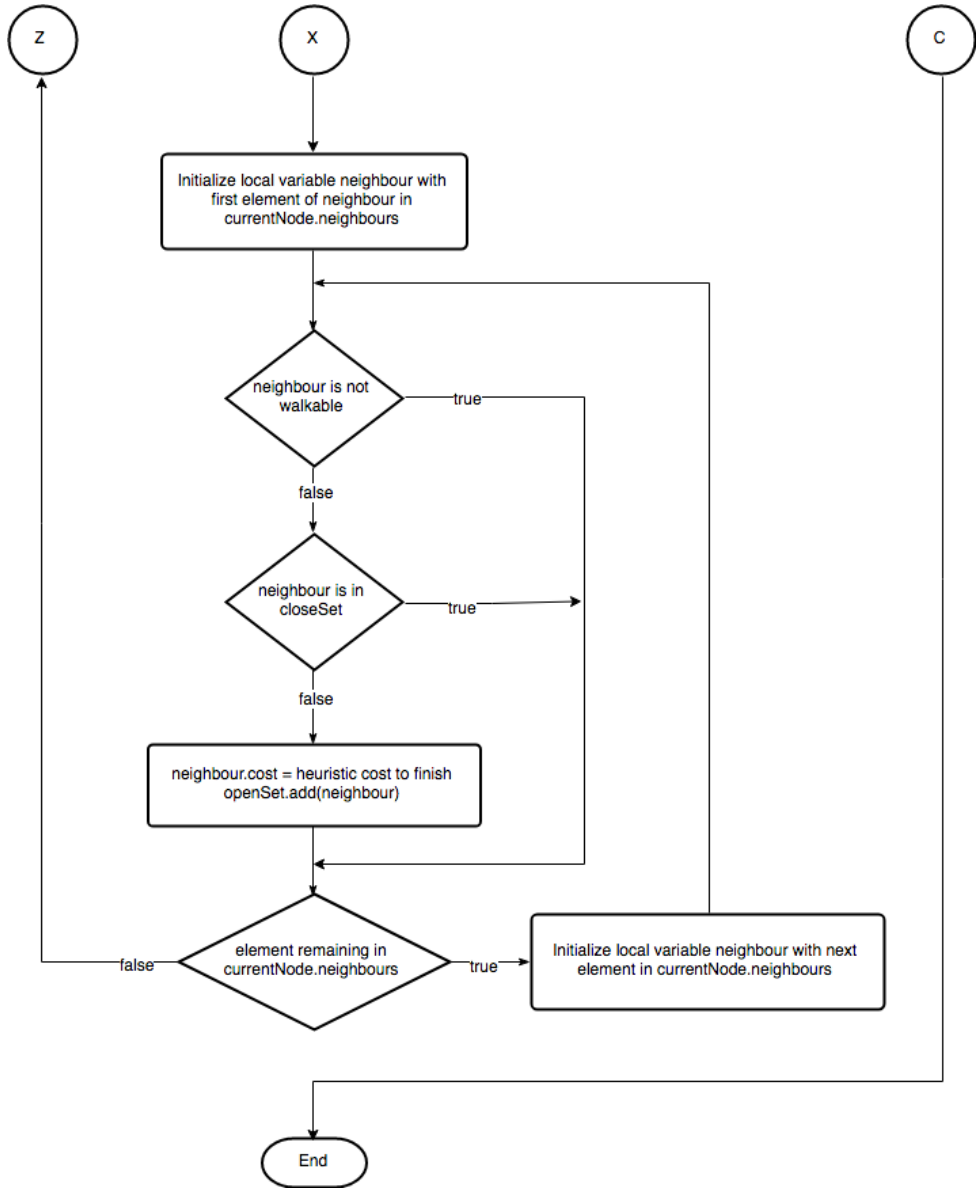
4.4 Perancangan *low-level A* search*

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 4.5 dan gambar 4.6.



Gambar 4.5 Flowchart *A* searching*



Gambar 4.6 Flowchart *A* searching* (lanjutan)

Berdasarkan flowchart diatas, dapat dijelaskan bahwa algoritma *A* searching* berfungsi untuk mencari suatu rute dari suatu *cell* pada *grid* menuju *cell* yang lainnya. Performa dari *A* searching* bergantung dari metode *heuristic* yang digunakan dan jenis struktur data yang digunakan.