SoulS - Android Game Horror

Oleh : Wahyu Santoso

Agustus 2017

Download the game : http://intip.in/souls
"Pembangunan game horror Android dengan menerapkan algoritma A* path finding pada NPC"

1 About This Game

SoulS merupakan game horror 2D per-platform Android yang dibuat menggunakan Unity. Ru, karakter utama di dalam game ini, mempunyai tujuan untuk menemukan kekasihnya yang hilang, yaitu Neng. Untuk itu dia pergi ke Bang-Bang, labirin tempat tinggal Bang untuk berusaha menemukan di mana kekasihnya berada, karena tempat itu merupakan tempat terakhir yang dikunjungi kekasihnya itu. Masalahnya, Bang merupakan musuh Ru yang akan berusaha membunuh Ru, karena Bang menganggap Ru telah mengambil Neng darinya. Berikut ini merupakan karakter yang ada di dalam game ini.

1. Ru, merupakan seorang Blus (golongan Blue Soul). Pemain akan berperan sebagai Ru dalam game ini. Kemampuan Ru adalah dia bisa berpindah tempat secara acak dalam durasi tertentu. Selain itu, dia juga dapat mengetahui masa lalu yang tidak dia ketahui saat mendapatkan beberapa Red Soul, tapi resikonya akan mengurangi setengah dari cahaya dan kecepatan Ru dalam beberapa detik.

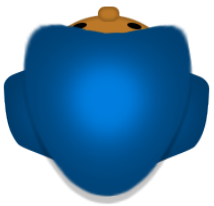


Figure 1: Ru, the Blus

2. Bang, merupakan seorang Res (golongan Red Soul). Merupakan musuh yang harus dihindari oleh pemain dalam game ini. Bang dapat berjalan dengan kecepatan acak dalam durasi tertentu, kadang pelan kadang cepat. Kecepatan Bang dapat melebihi kecepatan Ru saat Ru mendapatkan Blue Soul dalam jumlah tertentu, tapi kesempatannya hanya 30

persen saja dalam durasi tertentu. Kemampuan paling penting Bang di dalam game ini adalah Bang-Bang, di mana dia sudah mengetahui semua jalannya sehingga dia dapat mencari jalur tercepat untuk ke tempat Ru.

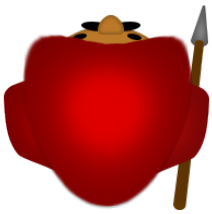


Figure 2: Bang, the Res

3. Neng, merupakan seorang Yels (golongan Yellow Soul). Kekasih Ru dalam game ini.

Figure 3: Neng, the Yels

4. Soul, di dalam game terdapat 3 soul, yaitu Blue Soul, Red Soul, dan Yellow Soul

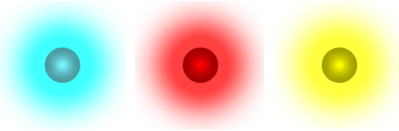


Figure 4: Soul

2 Apa Itu A* Path Finding?

Karakter Bang mempunyai AI untuk mencari jalur dari posisinya ke posisi Ru. Salah satu algoritma yang dapat digunakan adalah A*. A* (baca: A star) adalah algoritma pencarian simpul (node) yang mencari jalur dari titik awal ke titik yang ditentukan. Algoritma ini pertama kali ditemukan pada tahun 1968 oleh Peter Hart, Nils Nilsson dan Bertram Raphael. Dalam tulisan mereka, algoritma ini dinamakan algoritma A. Penggunaan algoritma ini dengan fungsi heuristik yang tepat dapat memberikan hasil yang optimal, maka algoritma inipun disebut A*. Berikut merupakan istilah di dalam algoritma A*.

a. **Starting point** adalah node awal (dalam game ini merupakan posisi Bang).

b. **Current node** adalah node yang sedang dijalankan.

c. **Simpul (node)** adalah petak-petak kecil sebagai representasi dari area path finding. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.

d. **Harga (cost)** adalah nilai yang diperoleh dari penjumlahan nilai **step** (jumlah nilai dari starting point ke current node) dan **distance** (jumlah nilai perkiraan dari sebuah current node ke node tujuan).

e. **Open list** adalah tempat menyimpan data node yang bisa digunakan untuk membandingkan node dengan cost terkecil.

f. **Closed list** adalah tempat menyimpan data node sebelum current node yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.

g. **Rintangan** adalah sebuah atribut yang menyatakan bahwa sebuah node tidak dapat dilalui oleh current node.

Algoritma A* dapat dijelaskan dengan pseudocode di bawah ini :

1. Masukkan starting point (node awal) ke open list
2. Ulangi langkah-langkah di bawah ini :
   2.1. Keluar dari perulangan jika open list kosong (jalur tidak ditemukan), atau current node adalah node tujuan (jalur ditemukan).
   2.2. Cari node dengan nilai cost yang paling kecil di open list. Jadikan sebagai current node.
   2.3. Hapus current node dari open list dan masukkan ke close list.
   2.4. Untuk setiap node yang bertetangga dengan current node, lakukan ini :
      - Jika node tetangga tidak dapat dilalui atau sudah ada dalam close list, maka abaikan.
      - Jika node tetangga belum ada di open list, maka jadikan current node sebagai parent dari node tetangga tersebut. Simpan nilai cost dan step dari node tetangga tersebut.
      - Jika node tetangga tersebut sudah ada di open list, maka gunakan nilai step sebagai perbandingan. Jika nilai step dari node yang ada di open list lebih kecil, maka jadikan current node sebagai parent dari node tersebut. Lalu perbarui nilai cost dan step dari node tetangga tersebut.
3. Simpan node yang ada di close list secara 'backward', urut mulai dari node tujuan ke parent-nya terus sampai mencapai node awal. Simpan node-node tersebut ke path list

3 Penerapan A* Path Finding

Pertama-tama kita harus membuat map yang terdiri dari node-node. Node tersebut merupakan matriks 2 dimensi yang akan digunakan untuk kalkulasi path finding. Matriks tersebut berukuran Width x Height. Pada game ini, ukuran map-nya adalah 20 x 20 sehingga membutuhkan matriks ukuran 20 x 20, kita sebut map [20, 20]. Selanjutnya kita beri atribut untuk setiap anggota (node) dari map [20, 20]. Berikut atribut yang dibutuhkan :

a. **isOpen**, bernilai true jika node tersebut bisa dilalui dan bernilai false jika tidak dapat dilalui (halangan).

b. **cost**, nilai dari penjumlahan nilai step dan distance, digunakan untuk perbandingan antar node.

c. **step**, jumlah nilai dari starting point ke current node, digunakan untuk perbandingan.

d. **parent**, untuk menyimpan node sebelumnya, atau masternya.

e. **position**, untuk menyimpan koordinat node.

   (pada game ini antar node mempunyai jarak koordinat sebanyak 2, sehingga koordinatnya dapat diketahui dengan mengalikan indeks node dengan 2, misal node dengan indeks map [3, 2] berposisi pada koordinat $x = 6$, $y = 4$).

Pseudocode untuk membuat map.

- for $y = 0$ to $mapHeight - 1$:
  - for $y = 0$ to $mapWidth - 1$:
    - $map[x, y].isOpen =$ if (node is not wall) ? true : false,
    - $map[x, y].position = ((x * tileSize), (y * tileSize))$,
    - $map[x, y].cost = 9999$,
    - $map[x, y].step = 9999$,
    - $map[x, y].parent = (x, y)$

Pada atribut cost dan step, inisialisasi awalnya adalah tak hingga (diberi nilai 9999 untuk mewakili tak hingga), karena semua node belum pernah dikunjungi. Pada atribut parent inisialisasi awalnya adalah node dia sendiri (parent boleh tidak diinisialisasi awal atau dibiarkan saja).

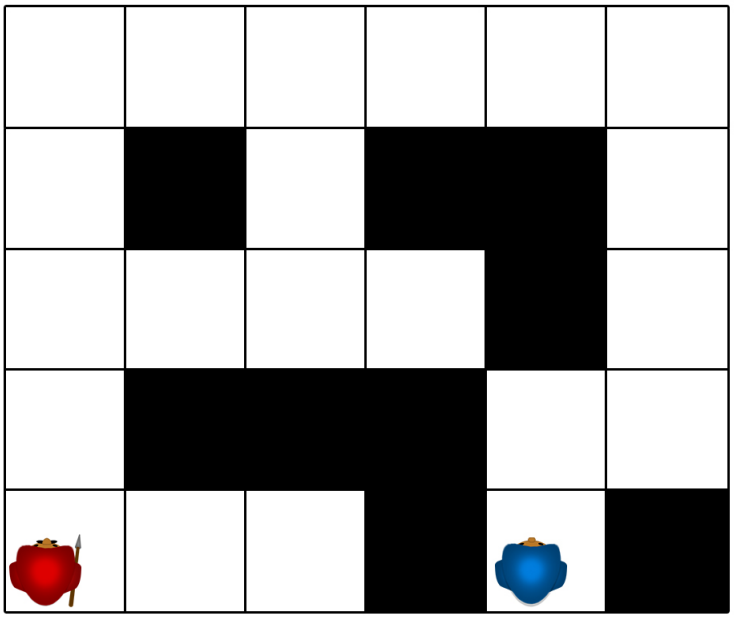
Berikutnya adalah menentukan posisi node di mana Bang (node awal) dan Ru (node tujuan) berada. Karena pada game ini jarak antar node adalah 2, maka tinggal dibagi saja koordinat posisi Bang maupun Ru dengan 2 (jika hasilnya desimal, maka dibulatkan), sehingga akan didapatkan indeks posisi node-nya. Misal Bang berada pada koordinat $x = 10$ dan $y = 5$, maka Bang

berada pada node dengan indeks map [5, 3].

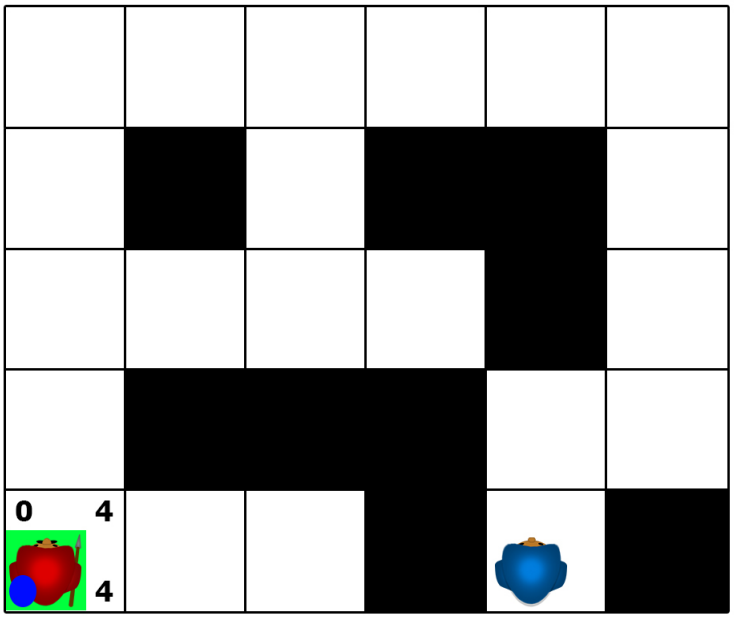
Berikut ini merupakan langkah-langkah pencarian jalur dengan menggunakan A* path finding.

- Node awal adalah Bang
- Node tujuan adalah Ru
- Kotak warna hijau adalah open list
- Kotak warna merah adalah close list
- Lingkaran warna putih adalah path list
- Nilai di sudut kiri atas adalah step
- Nilai di sudut kanan atas adalah cost
- Nilai di sudut kanan bawah adalah distance
- Checklist adalah current node
- Tanda panah menunjukkan parent dari suatu node

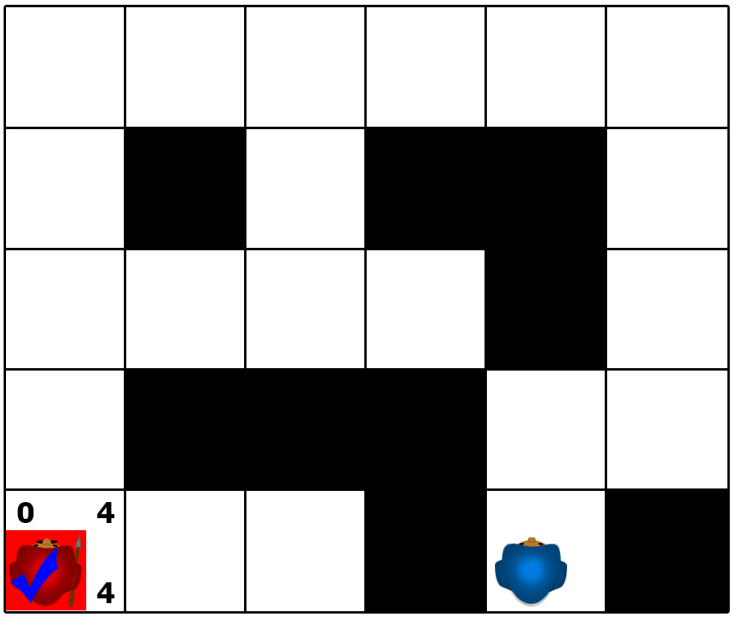
1. Buat map dan tentukan node awal (Bang) dan node tujuan (Ru).



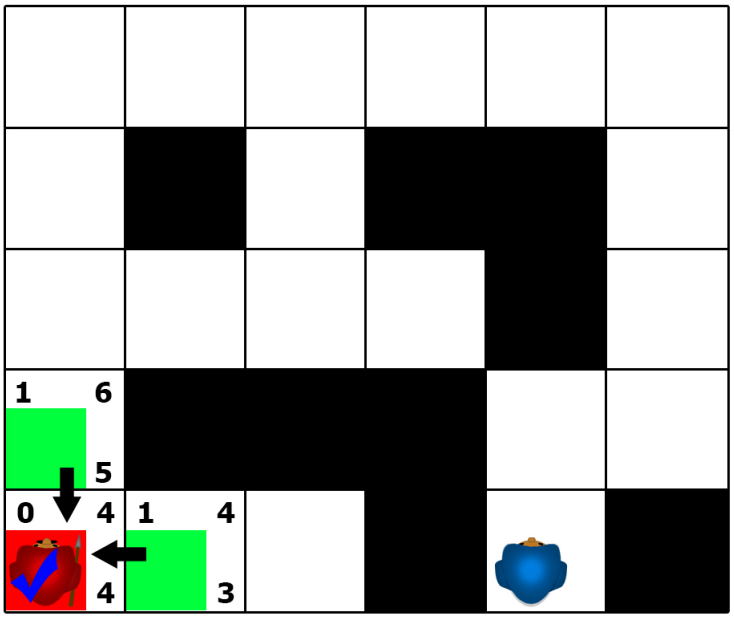
2. Masukkan node awal ke open list. Dan perbarui nilai cost dan step node tersebut. $cost = step + distance$. Step didapatkan dari banyaknya langkah yang ditempuh dari node awal, distance didapatkan dari node Ru dikurangi node Bang. Jadikan sebagai current node. Perbarui step dan cost dari current node.



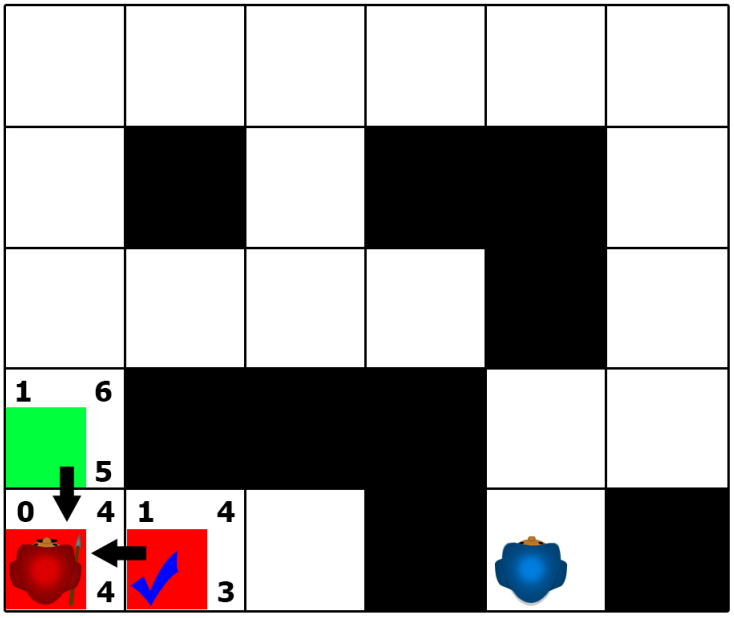
3. Cek, apakah current node adalah node tujuan, jika tidak, cari node dengan cost terkecil di open list, jadikan node itu sebagai current node. Hapus dari open list dan masukkan ke close list



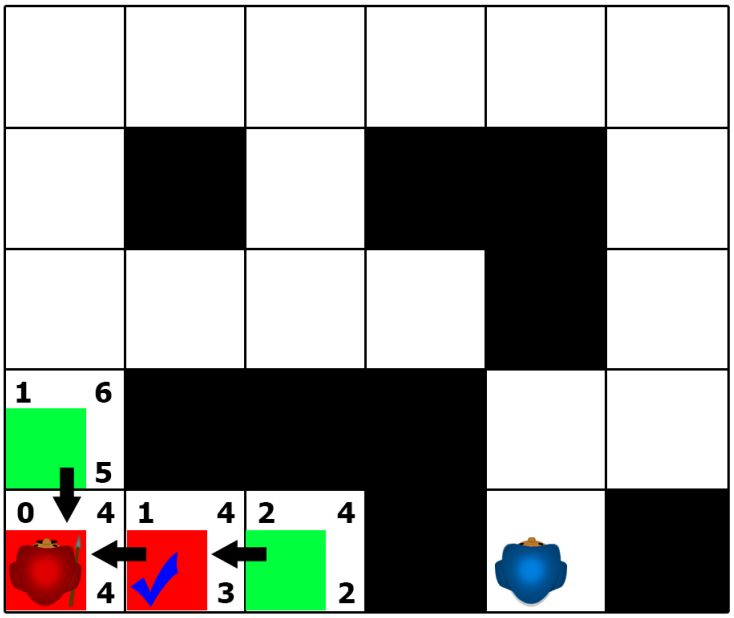
4. Perbarui step, cost, dan parent dari masing-masing node yang bertetangga dengan current node. Masukkan ke open list.



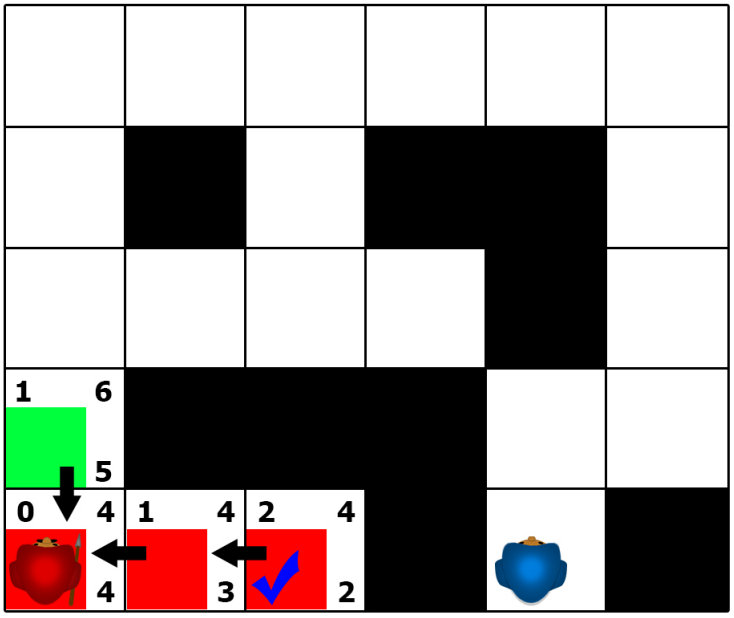
5. Ulangi langkah ke 2.



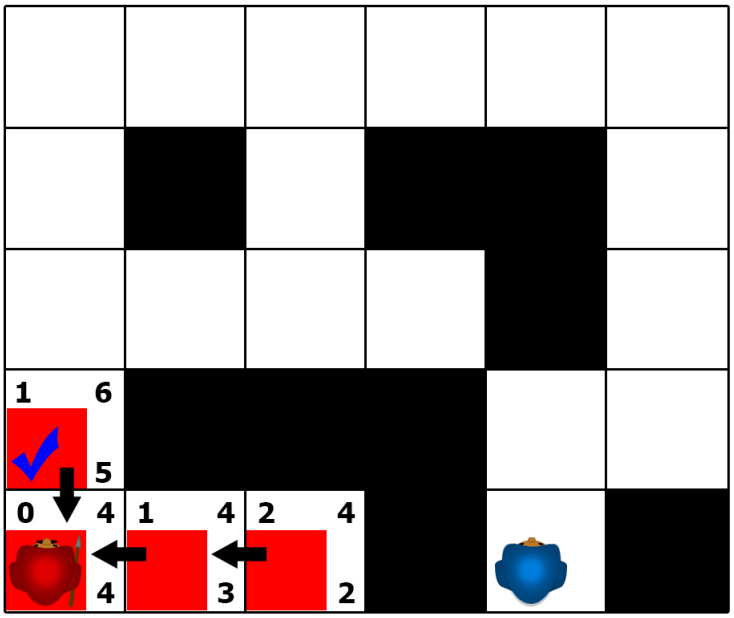
6. Ulangi langkah ke 3.



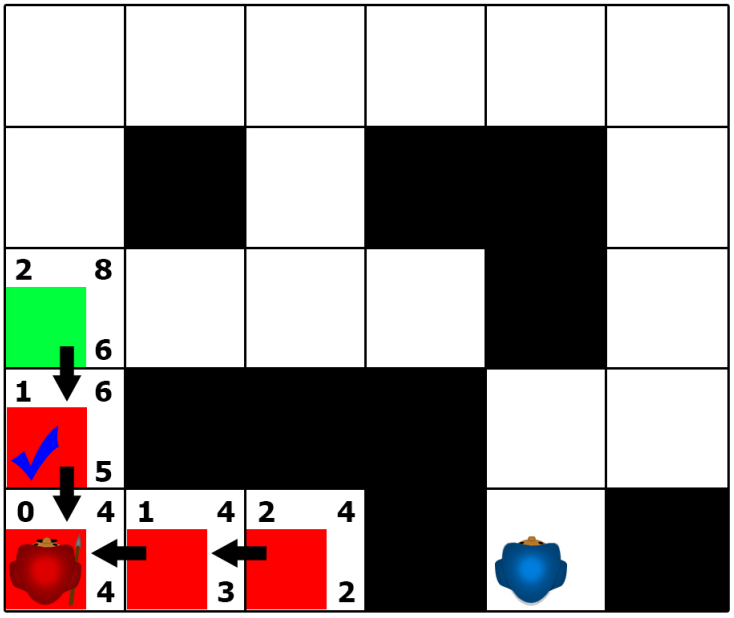
7. Ulangi langkah ke 2.



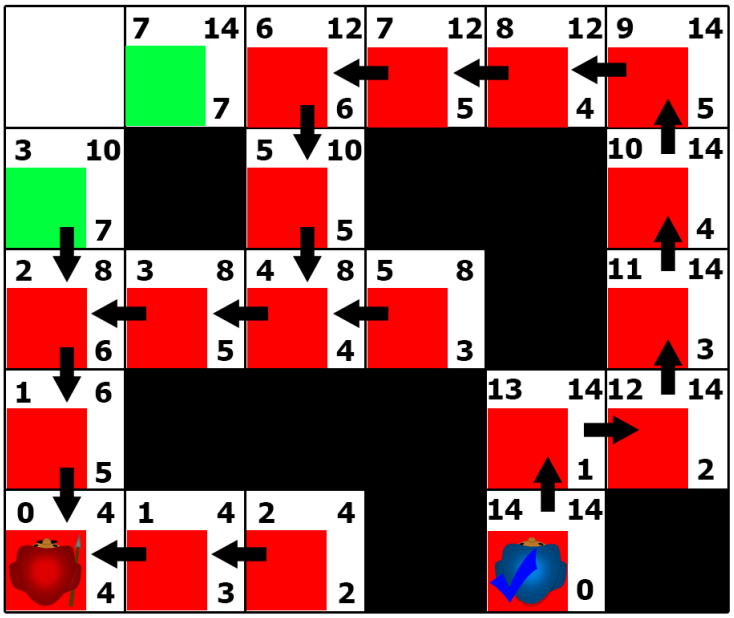
8. Ulangi langkah ke 3. Karena tidak ada yang bertetangga dengan current node, maka diabaikan. Ulangi langkah ke 2.



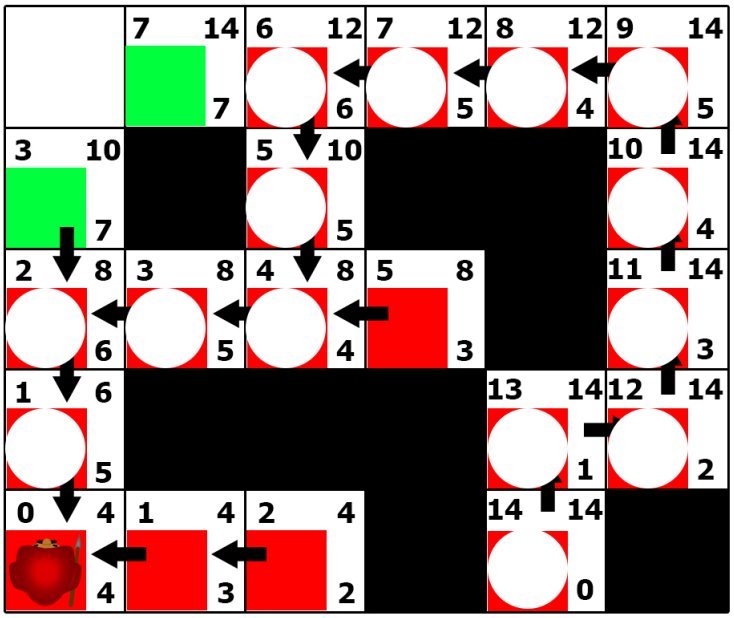
9. Ulangi langkah ke 3.



10. Seterusnya sampai current node adalah node tujuan (jalur ditemukan).



11. Simpan node yang ada di close list secara ‘backward’, urut mulai dari current node yang merupakan node tujuan ke parent-nya terus sampai mencapai node awal. Simpan node-node tersebut ke path list.



4 Hasil Uji Coba A* Path Finding pada Game