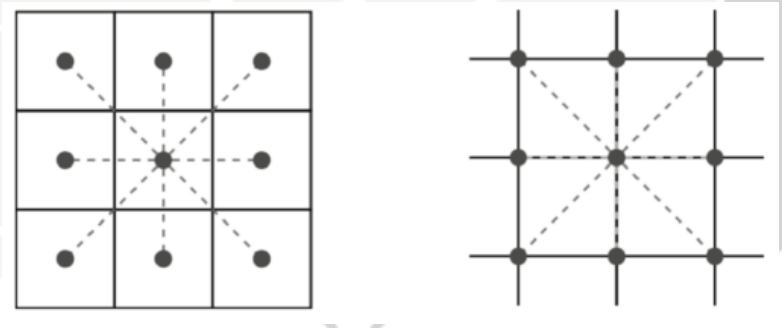
BAB III

LANDASAN TEORI

A. Hexagon-Based Environment

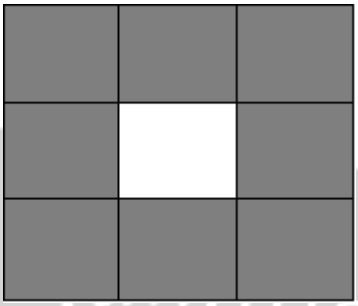
Sebuah lingkungan pada dunia nyata ataupun dunia virtual dapat dimodelkan dalam bentuk sebuah tabel. Lingkungan semacam ini sering ditemui pada dunia robotik ataupun game. Sebuah lingkungan yang dimodelkan ke dalam bentuk tabel dapat dibagi menjadi dua, yaitu *center-based grid environment* dan *corner-based grid environment* (Yakovlev, et al., 2015).



Gambar 1 *Center-based* (kiri) dan *Corner-based* (kanan) grid environment

Gambar 1 menjelaskan perbedaan antara *center-based* dan *corner-based environment*. *Center-based* akan menempatkan titik-titik nodenya di tengah kotak. Hal ini menyebabkan NPC yang bergerak akan selalu berhenti di tengah kotak. *Corner-based* akan menempatkan titik nodenya pada sudut kotak, sehingga NPC yang bergerak pada lingkungan ini akan bergerak di sepanjang garis kotak.

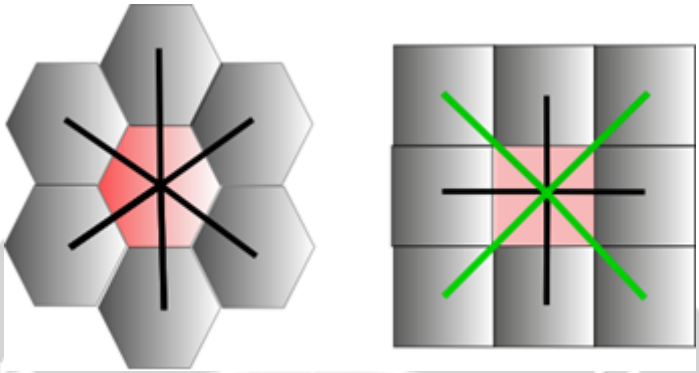
NPC yang diletakkan pada sebuah lingkungan dengan tipe *centre-based* memiliki delapan kemungkinan arah pergerakan, yaitu utara, timur laut, timur, tenggara, selatan, barat daya, barat, dan barat laut (lihat Gambar 2).



Gambar 2 Kemungkinan pergerakan NPC pada lingkungan berbasis tabel (diarsir)

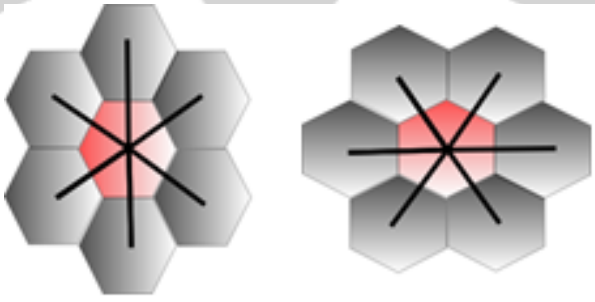
Kelemahan dari *grid-based environment* adalah pergerakan tiap NPC yang terlihat kurang natural terutama ketika NPC bergerak secara diagonal. Hal ini disebabkan oleh perbedaan jarak untuk pergerakan secara vertikal ataupun horizontal dengan pergerakan diagonal. Sebagai contoh, jika *p* merupakan jarak untuk tiap titik, maka untuk pergerakan vertikal ataupun horizontal akan membutuhkan 1**p* sedangkan untuk pergerakan diagonal akan membutuhkan 1√2**p*. Untuk mengatasi ini para pengembang game akhirnya membuat map atau lingkungan baru dengan segi enam sebagai dasar atau biasa disebut dengan *hexagon-based environment*.

Hexagon-based environment sering digunakan dalam game yang memiliki *turn-based gameplay*. Hal ini disebabkan karena keunggulan dari *hexagon-based environment* yang memiliki jarak tiap titik yang konstan. Sedikit lebih sederhana daripada *grid-based environment* yang memiliki 9 kemungkinan pergerakan, *hexagon-based environment* hanya memiliki 6 kemungkinan pergerakan (lihat Gambar 3).



Gambar 3 Perbandingan pergerakan *hexagon-based* (kiri) dan *grid-based* (kanan) *environment*

Pergerakan NPC pada *hexagon-based environment* akan bergantung pada bagaimana *hexagon* itu sendiri. Ada dua tipe bagaimana *hexagon-based environment* dibuat, yaitu "*pointy at top*" atau "*pointy at side*". Jika *hexagon-based environment* yang dibuat menggunakan tipe yang pertama, maka kemungkinan pergerakan dari NPC adalah utara, timur laut, tenggara, selatan, barat daya, dan barat laut. Jika *hexagon-based environment* yang dibuat menggunakan tipe yang kedua, maka kemungkinan pergerakan NPC adalah timur, tenggara, barat daya, barat, barat laut, dan timur laut. Perbedaan kemungkinan pergerakan ini dijelaskan pada Gambar 4.



Gambar 4 "Pointy at top" dan "Pointy at side" hexagon

Penelitian ini akan menerapkan *hexagon-based environment* dengan titik-titik yang digunakan diletakkan di tengah *hexagon* (*center-based)*. Tiap titik akan dihubungkan dan dibentuk dengan ukuran MxN, sehingga membentuk sebuah labirin. Labirin ini akan dimodelkan dengan sebuah matriks $A_{MxN} = \{a_{ij}\}$ dimana i dan j merupakan posisi baris dan kolom, sedangkan M dan N adalah jumlah baris dan kolom dari labirin yang dibuat (Reuter, et al., 2012).

B. Algoritma A*

Algoritma A* adalah sebuah algoritma yang dikenalkan pertama kali oleh (Hart, et al., 1968). A* merupakan salah satu algoritma pencarian heuristik karena pada proses pencarian algoritma ini akan memasukkan nilai perkiraan ke dalamnya. A* akan memilih nilai terkecil dari persamaan $f(n) = g(n) + h(n)$ dimana *n* merupakan titik-titik atau sering disebut node, $g(n)$ merupakan nilai yang dibutuhkan dari titik awal untuk mencapai node *n*, dan $h(n)$ merupakan nilai heuristik atau nilai perkiraan untuk mencapai node akhir.

Algoritma A* akan selalu menemukan jalur terpendek meskipun ada rintangan pada proses pencarian. Hal ini menarik para peneliti untuk terus melakukan riset dan pengembangan pada algoritma ini.

```
function AStar(StartNode, GoalNode)
{
  closed_list = {};
  open_list = {};
  open_list.add(StartNode);
  path = {};
  while open_list.isNotEmpty()
  {
    //Find the node with lowest value of
    //f(n)=g(n)+h(n) from neighbor.
    CurrentNode = FindLowestValue();
    open_list.remove(CurrentNode);
    if(CurrentNode == FinishNode)
    {
      //Path found, stop iteration
      break;
    }
    for(each NeighborNode of CurrentNode)
    {
      current_cost = NeighborNode.getGValue()+
      findDistanceCost(CurrentNode,NeighborNode);
      if(NeighborNode is in open_list)
      {
        if(NeighborNode.getGCost()<=current_cost)
          continue;
      }
      else if (NeighborNode is in closed_list)
      {
        if(NeighborNode.getGValue()<=current_cost)
          continue;
        closed_list.remove(NeighborNode);
        open_list.add(NeighborNode);
      }
      else
      {
        open_list.add(NeighborNode);
        NeighborNode.setHvalue();
      }
      NeighborNode.setGValue(current_cost);
      path.add(CurrentNode);
    }
    closed_list.add(CurrentNode);
  }
}
```

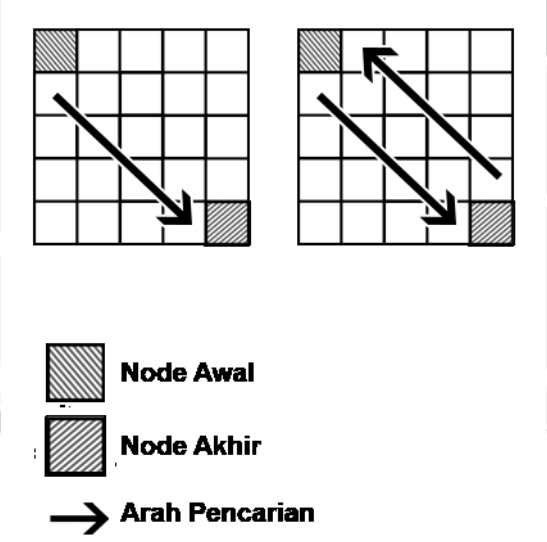
Gambar 5 Pseudocode A*

Gambar 5 merupakan pseudocode dari algoritma A*. Algoritma A* akan menggunakan *open_list* dan *closed_list* untuk memisahkan node-node yang sudah pernah ataupun belum pernah dikunjungi. A* akan mengunjungi node-node yang berdekatan dengan node yang ditunjuk, sambil terus mencari nilai *f(n)* terkecil dari node-node di sekitarnya. Hal ini akan terus menerus diulang sampai node yang

dikunjungi merupakan node tujuan. Jalur terpendek akan didapatkan melalui node-node yang sudah dipilih pada proses sebelumnya.

C. Parallel Bidirectional Search

Parallel Bidirectional Search (PBS) dengan *Bidirectional Search* (BS) merupakan salah satu teknik optimisasi dari pencarian jalur terpendek pada umumnya.



Gambar 6 Perbedaan pencarian jalur klasik dengan PBS

Gambar 6 menjelaskan tentang gambaran umum dari perbedaan pencarian jalur terpendek pada umumnya dibandingkan dengan pencarian jalur terpendek menggunakan BS ataupun PBS.

Gambar sebelah kiri merupakan arah pencarian pada algoritma pencarian jalur terpendek pada umumnya. Pencarian jalur akan dimulai dari posisi awal node menuju ke posisi akhir node. Sebagai contoh, algoritma A* juga menerapkan konsep ini dalam pencarian jalur terpendeknya. Algoritma A* akan mencari jalur

terpendeknya dengan berjalan dari satu ubin ke ubin lainnya sambil memperhatikan nilai perkiraan dari jarak antara node saat ini sampai ke node akhir serta jarak yang sudah ditempuh dari node awal.

Gambar sebelah kanan adalah gambaran umum bagaimana BS ataupun PBS akan mencari jalur terpendeknya. Persamaan dari BS dan PBS ada dalam proses pencarian jalurnya. Proses pencarian jalur terpendek BS dan PBS akan dilakukan pada dua tempat yang berbeda, yaitu dari posisi awal node menuju posisi akhir node dan posisi akhir node menuju posisi awal node. Sedangkan perbedaaan mendasar dari BS dan PBS berada pada eksekusi pencariannya. BS akan berjalan secara bergantian, sedangakn pada PBS, kedua proses pencarian akan dilakukan secara bersamaan. Oleh karena itu, PBS dapat juga disebut sebagai teknik optimisasi dari BS.

BS masih menggunakan satu *core* CPU saat melakukan pencarian jalur terpendek, meskipun terlihat terdapat dua arah pencarian. Kedua proses akan secara bergantian dikerjakan oleh *core* CPU meskipun keduanya mengakses memori yang sama. Hal ini yang dasar dikembangkannya sebuah teknik optimisasi yang baru dan mampu menjawab kelemahan dari BS. *Bidirectional Search* yang semula hanya menggunakan satu *core* CPU akan dioptimasi oleh PBS dengan menggunakan dua *core* dari CPU yang berbeda. Oleh karena itu, proses pencarian jalur terpendek dari *Bidirectional Search* dapat berjalan bersamaan. Proses pencarian akan berhenti ketika kedua *core* telah menemukan titik yang sama di antara titik awal dan titik akhir (Brand & Bidarra, 2011).

D. Compute Unified Device Architecture

Pada bulan November tahun 2006, NVIDIA sebagai salah satu pengembang GPU memperkenalkan *Compute Unified Device Architecture* atau dapat disingkat dengan CUDA. CUDA merupakan sebuah *Application Program Interface* (API) yang disediakan oleh NVIDIA agar para pengembang aplikasi dapat menggunakan secara langsung GPU yang ditanam dalam kartu grafis milik NVIDIA. Kehadiran CUDA memberikan dampak yang sangat besar dalam bidang proses paralel dikarenakan terbukanya akses ke GPU secara langung membuka peluang bahwa GPU tidak digunakan untuk melakukan render gambar saja. Hal ini dikarenakan sebuah GPU memiliki komputasi paralel yang sangat tinggi, *multithread*, dan memori yang sangat besar untuk menyimpan proses komputasi (Khoirudin & Shun-Liang, 2015).

CUDA sendiri merupakan sebuah *library* tambahan yang menggunakan bahasa pemrograman C/C++. CUDA akan memberikan akses kepada GPU untuk melakukan proses komputasi yang lebih luas dan lebih umum dibanding sekedar melakukan proses render (NVIDIA, 2016) (Ghorpade, et al., 2012).