BAB 2
LANDASAN TEORI

2.1 Pengertian Game Tower Defense

Definisi 1. *Tower Defense* adalah strategi game yang dimainkan di mana pemain memberhentikan atau melambatkan musuh dengan meletakkan rintangan pada medan perang [9].

Definisi 2. Tower Defense game adalah permainan *Real-Time Strategy* (RTS) yang fokus dalam alokasi sumber daya dan penempatan unit (*tower*) untuk mencegah lawan untuk mencapai lokasi tertentu [1].

Dari kedua definisi tersebut, dapat menyimpulkan bahwa *game tower defense* merupakan *game* dengan pemain memiliki tujuan untuk mencegah lawan mencapai suatu area tertentu dengan segala cara. Pemain dapat menggunakan segala cara untuk mencegah lawan dan pada cara tersebut kreativitas serta keahlian pemain diuji.

Sebuah *game* Tower Defense memiliki element-element terbesar atau ternama sebagai berikut [1]:

1. Terrain: Sebuah TD Map memiliki batasan dalam bagaimana pemain dapat menempatkan sumber daya. Ada 4 TD Map yang dapat dibedakan yaitu:
   - Linear dan Branching Paths: Musuh menempuh jalur yang telah diberikan.
   - Free-form Path: Musuh menempuh jalur menyesuaikan dengan sumber daya yang ditempatkan pemain.
   - Survival: Musuh memiliki *spawn point* yang teracak dan mencari jalur yang paling terdekat. Pemain diberikan sumber daya yang terbatas untuk bertahan dengan tower.
   - Parallel Lines: Musuh datang secara bersamaan tetapi hanya melalui garis lurus. Sebuah contoh adalah Plants Vs. Zombies.
2. Towers: Sumber daya yang dapat ditempatkan pemain. Tower dapat ada dalam berbagai bentuk dan dapat memiliki *firing rate*, *damage*, dan *effects* yang berbeda-beda.

3. Creeps: Musuh yang akan menyerang tower pemain. Dapat memiliki atribut berbeda.

4. Reward System: Penghargaan yang didapat oleh pemain. Bisa dengan bentuk uang dalam permainan, *experience*, *upgrade tower*, dan seterusnya.

2.2 Pathfinding

Pathfinding adalah komponen fundamental dalam berbagai ranah seperti GPS, *video game, robotic*, logistik, dan *crowd simulation* [10]. Masalah pathfinding dalam game harus dipecah secara *real-time*, dengan memori dan sumber daya CPU yang terbatas [11]. Untuk mengevaluasi efisiensi sebuah algoritma, harus dipertimbangkan waktu untuk menjalaninya, memori yang telah dipakai, dan apakah lingkungan sistem adalah statis, dinamis, atau *real-time* '*deterministic*'. Pathfinding juga harus memberi solusi yang mencakupi tujuan berikut [12]:

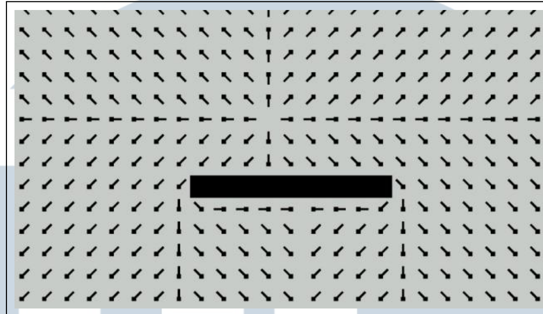
1. Cara untuk jalan dari A ke B.
2. Cara untuk menghindari rintangan yang berada di map.
3. Cara untuk mencari jalur yang paling pendek.
4. Cara untuk mencari jalur dengan cepat.

2.3 Kompleksitas Waktu

Kompleksitas waktu adalah tingkat efisiensi atau waktu yang diambil untuk sebuah fungsi program memproses atau menjalankan sebuah input [13]. Kompleksitas algoritma biasanya diekspresikan secara asimtotik menggunakan notasi big-O. Jika kompleksitas waktu untuk menjalankan suatu algoritma dinyatakan sebagai $T(n)$, dan memenuhi $T(n) \leq C(f(n))$ untuk $n \geq 0$, maka kompleksitas dapat dinyatakan dengan $T(n) = O(f(n))$ [14].

UMN UNIVERSITAS MULTIMEDIA NUSANTARA

2.4 Algoritma Flow Field Pathfinding



Gambar 2.1. Vector Fields

Sumber: [5]

Gambar 2.1 merupakan visualisasi *flow field*. Nama Flow Field diambil dari konsep dinamika fluida yaitu studi mengenai pergerakan permukaan air. Sebelum algoritma *flow field* digunakan untuk *game* komputer, algoritma ini digunakan dalam bidang dinamika fluida. Konsep *flow field* juga dapat diartikan sebagai *vector fields* (bidang vektor). Cara kerja algoritma flow field akan lebih mudah dijelaskan dalam vektor. Dalam bidang vektor, setiap vektor menunjuk ke simpul tetangga yang paling dekat dengan tujuan. Ketika sebuah unit melewati suatu sel, unit tersebut akan mengetahui vektor yang dimiliki oleh node tersebut, sehingga sesuai dengan namanya, vektor-vektor dalam medan aliran tersebut akan terlihat seperti aliran air.

2.4.1 Kompleksitas Waktu Flow Field Pathfinding

Algoritma Flow Field Pathfinding mempunyai kompleksitas waktu dengan notasi $O(n^2)$, notasi ini terbentuk karena dalam pembuatan kode pengecekan terhadap semua node pada grid dilakukan dalam *nested if*.

2.5 Algoritma A* Pathfinding

Algoritma A* merupakan algoritma yang digunakan untuk mencari rute terpendek dengan menggunakan perhitungan nilai terendah dari jalur titik awal menuju titik akhir. Perhitungan nilai terendah dilakukan dengan menggunakan rumus heuristik sebagai berikut [15]:

$$f(n) = g(n) + h(n) \tag{2.1}$$

Atribut-atribut di Rumus 1 dapat memiliki definisi sebagai berikut [16]:

a) g(n) merupakan nilai total dari titik awal hingga titik n

b) h(n) merupakan nilai heuristik, merupakan nilai estimasi yang diperlukan dari titik n hingga titik tujuan.

c) f(n) merupakan jumlah dari nilai g(n) dan h(n). Dengan nilai terkecil f(n) merupakan jalur terpendek untuk menuju titik tujuan.

Berikut adalah *pseudocode* untuk Algoritma A*:

```
1  Create a node containing the goal state node_goal
2  Create a node containing the start state node_start
3  Put node_start on the open list
4  while the OPEN list is not empty
5  {
6  Get the node off the open list with the lowest f and call it node_current
7  if node_current is the same state as node_goal we have found the solution; break from the while loop
8     Generate each state node_successor that can come after node_current
9     for each node_successor of node_current
10    {
11       Set the cost of node_successor to be the cost of node_current plus the cost to get to node_successor from node_current
12       find node_successor on the OPEN list
13       if node_successor is on the OPEN list but the existing one is as good or better then discard this successor and continue
14       if node_successor is on the CLOSED list but the existing one is as good or better then discard this successor and continue
15       Remove occurences of node_successor from OPEN and CLOSED
16       Set the parent of node_successor to node_current
17       Set h to be the estimated distance to node_goal (Using the heuristic function)
18        Add node_successor to the OPEN list
19    }
20    Add node_current to the CLOSED list
21}
```

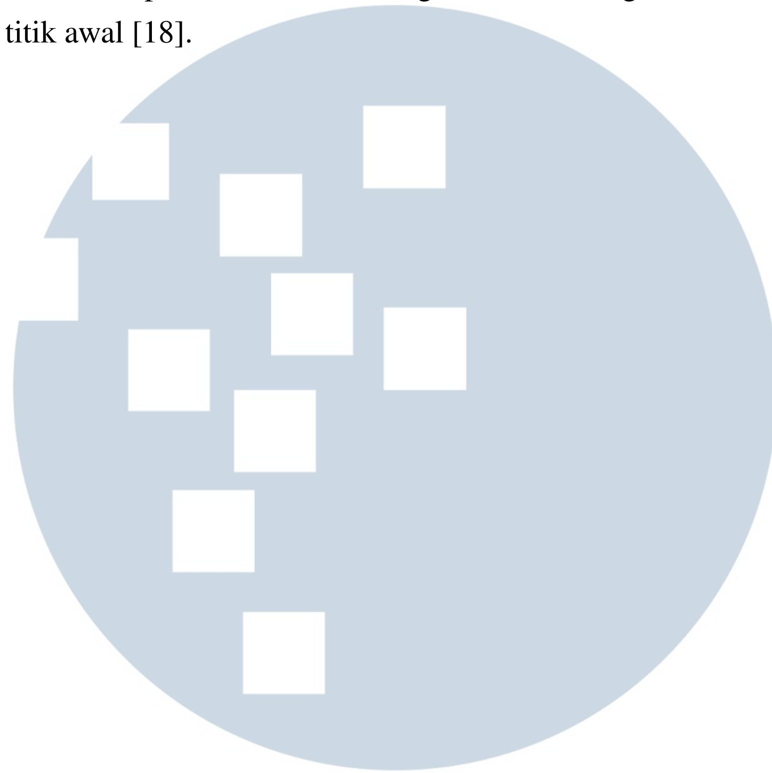
Gambar 2.2. Pseudocode Algoritma A*
Sumber: [17]

Dari Gambar 2.2, dapat disimpulkan bahwa cara kerja Algoritma A* adalah untuk menghitung jarak salah satu lintasan, menyimpannya, dan lanjut untuk menghitung lintasan lainnya yang ada. Ketika seluruh lintasan telah dihitung, Algoritma A* akan memilih lintasan yang terpendek.

2.5.1 Kompleksitas Waktu A* Pathfinding

Algoritma A* Pathfinding mempunyai kompleksitas waktu yang bergantung dengan heuristik. Dalam skenario *worst case*, kompleksitas waktu yang diperoleh

adalah dengan notasi O(b^d) atau O(E). Hal ini dikarenakan, melakukan perjalanan ke seluruh node merupakan sebuah kemungkinan untuk algoritma mencapai titik tujuan dari titik awal [18].


UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA