BAB 5 IMPLEMENTASI

5.1 Spesifikasi Sistem

Sistem yang digunakan pada implementasi metode penelitian ini akan ditunjukkan pada tabel 5.1 dan tabel 5.1. Tabel 5.1 akan menunjukkan spesifikasi perangkat lunak sedangkan tabel 5.2 akan menunjukkan spesifikasi perangkat keras.

Tabel 5.1 Spesifikasi Perangkat Lunak

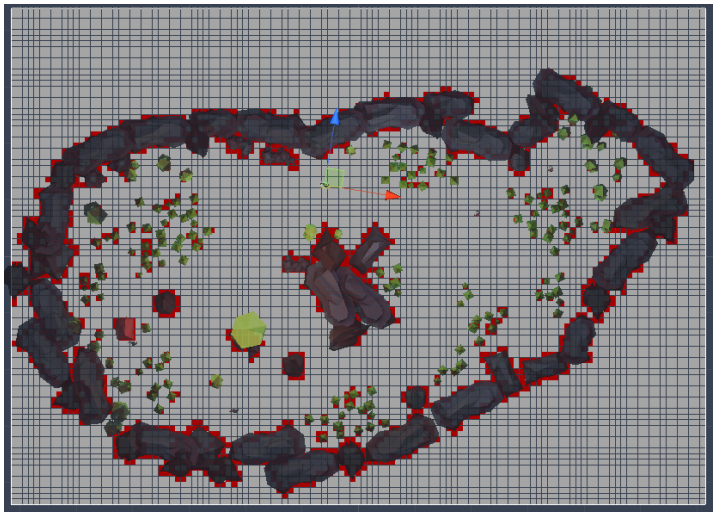
OS (Operating System)	macOS High Sierra
Programming Language	C#
SDK (Software Development Kit)	Unity 2017.2.0f3
Integrated Development Environment	Visual Studio for Mac

Tabel 5.2 Spesifikasi Perangkat Keras

Processor	Intel i5-5287U CPU @ 2.90 GHz
RAM (Memory)	8 GB
VGA (Graphic Card)	Intel Iris Graphics 6100 1.5GB

5.2 Implementasi *dataset* level *grid*

Pada level yang ada pada *game,* dibuatlah representasi *grid* dari level. Gambar dibawah akan menampilkan hasil representasi level *grid. Cell* berwarna putih adalah *cell passable, dan cell* berwarna merah adalah *cell unpassable* seperti pada gambar 5.1.



Gambar 5.1 Level *grid game*

Data diatas akan disimpan pada sebuah array 2D dari Node yang nantinya diolah oleh algoritma *Conflict-based search*, dan *A* search*.

5.2.1 Pseudocode untuk membuat representasi level dalam *grid*

Implementasi dari kode untuk membuat grid berdasarkan level 3D untuk *video game* akan dijelaskan dalam bentuk pseudocode seperti pada tabel 5.3.

Tabel 5.3 Pseudocode CreateGrid

Penerapan CreateGrid pada Level	
1	def Node[,] CreateGrid():
2	Node[,] _grid = new Node[gridSizeX, gridSizeY]
3	Vector3 worldBottomLeft = transform.position - Vector3.right *
4	gridWorldSize.x / 2 - Vector3.forward * gridWorldSize.y / 2
5	for (int x = 0; x < gridSizeX; x++)
6	for (int y = 0; y < gridSizeY; y++)
7	Vector3 worldPoint = worldBottomLeft + Vector3.right * (x *
8	nodeDiameter + nodeRadius) + Vector3.forward * (y * nodeDiameter +
9	nodeRadius)
10	bool walkable = !Physics.CheckSphere(worldPoint, nodeRadius,
11	unwalkableMask)
12	_grid[x, y] = new Node(walkable, worldPoint, x, y)
13	end for
14	end for
15	return _grid

Penjelasan:

1. Definisi method terdapat pada baris 1
2. Pada baris 2 dilakukan inisialisasi array dua dimensi grid
3. Pada baris 4 - 14 kode akan melakukan looping sebanyak gridSizeX dan gridSizeY untuk mendeteksi pada suatu *node* apakah *node* tersebut *passable* atau *unpassable.*

5.3 Implementasi *High-level Conflict-based search*

Pada implementasi *Conflict-based search* terdapat berbagai macam algoritma yang digunakan. Selain menggunakan *conflict-based search*, beberapa algoritma digunakan untuk mencari suatu *conflict* pada suatu *solution*. Beberapa algortima tersebut akan dibahas pada subbab dibawah.

5.3.1 Pseudocode untuk mencari *conflict* dari suatu *solution*

Implementasi dari kode untuk menemukan *conflict* pertama yang ada pada suatu *solution* dar suatu *node* akan dijelaskan dalam bentuk pseudocode seperti pada tabel 5.4. Pada pseudocode tabel 5.4 terdapat baris untuk memeriksa suatu *conflict* dari suatu *solution* pada suatu waktu, implementasi dari kode tersebut akan dijelaskan dalam bentuk pseudocode seperti pada tabel 5.5.

Tabel 5.4 Pseudocode FindFirstConflict

Penerapan FindFirstConflict	
1	def Conflict FindFirstConflict():
2	int maxTime = GetMaxTime()
3	Conflict firstFoundConflict = null
4	for (int currentTime = 0; currentTime < maxTime; currentTime++)
5	irstFoundConflict = CheckConflictAt(currentTime)
6	if (firstFoundConflict != null) break
7	end for
8	return firstFoundConflict

Penjelasan:

1. Definisi method terdapat pada baris 1
2. Pada baris 2 - 3 dilakukan inisialisasi variabel yang dibutuhkan
3. Pada baris 4 - 7, kode akan melakukan looping sebanyak waktu maksimal yang ada pada *solution*, dan pada masing-masing waktu tersebut, akan dilakukan pemeriksaan *conflict,* apabila terdapat *conflict*, maka *conflict* yang pertama kali ditemukan yang akan menjadi nilai kembalian dari method ini.

Tabel 5.5 Pseudocode CheckConflictAt

Penerapan CheckConflictAt	
1	def Conflict CheckConflictAt(int time):
2	Agent previousAgent = null
3	Node previousNode = null
4	foreach (KeyValuePair<Agent, Path> entry in pathMap)
5	if (previousNode != null && (previousNode ==
6	entry.Value.GetNodeAt(time) || previousNode ==
7	entry.Value.GetNodeAt(time + 1)))
8	return new Conflict(previousAgent, entry.Key,
9	entry.Value.GetNodeAt(time), time)
10	end if
11	previousAgent = entry.Key;
12	previousNode = entry.Value.GetNodeAt(time)
13	end for
14	return null

Penjelasan:

1. Definisi method terdapat pada baris 1
2. Pada baris 2 - 3 dilakukan inisialisasi variabel yang dibutuhkan
3. Pada baris 4 - 13 terdapat pemeriksaan dari suatu waktu tertentu berdasarkan parameter yang diberikan, kemudian kode akan melakukan looping sebanyak agen yang terdaftar dan memeriksa apakah terdapat *conflict* pada waktu tersebut.

5.3.2 Pseudocode *Conflict-based search*

Implementasi dari kode untuk menjalankan *conflict-based* search dan mendapatkan *solution* yang tidak memiliki conflict akan dijelaskan dalam bentuk pseudocode seperti pada tabel 5.6.

Tabel 5.6 Pseudocode GetSolution

Penerapan GetSolution	
1	def Solution GetSolution():
2	Dictionary<Agent, Path> agentsPath = new Dictionary<Agent,
3	Path>()
4	foreach (Agent agent in agents)
5	Path path = new Path(aStarSolver.Solve(agent))
6	agentsPath.Add(agent, path)
7	end for
8	Solution solution = new Solution(agentsPath)
9	CBSNode _root = new CBSNode(new Dictionary<Agent,
10	List<Constraint>>(), solution, solution.cost)
11	
12	List<CBSNode> openSet = new List<CBSNode>()
13	openSet.Add(_root)
14	
15	while (openSet.Count > 0)
16	CBSNode bestNode = openSet[0]
17	for (int i = 1; i < openSet.Count; i++)
18	if (openSet[i].solution.cost < bestNode.solution.cost)
19	bestNode = openSet[i]
20	end if
21	end for
22	
23	openSet.Remove(bestNode)
24	
25	Conflict bestConflict = bestNode.solution.FindFirstConflict()
26	if (bestConflict == null)
27	return bestNode.solution
28	
29	CBSNode leftNode = GenerateNode(bestNode, bestConflict.agentA,
30	bestConflict.node, bestConflict.time)
31	CBSNode rightNode = GenerateNode(bestNode,
32	bestConflict.agentB, bestConflict.node, bestConflict.time)
33	
34	openSet.Add(leftNode)
35	openSet.Add(rightNode)
36	end while
37	return null

Penjelasan:

1. Definisi method terdapat pada baris 1
2. Baris 2 digunakan untuk inisialiasi variabel yang dibutuhkan

3. Pada baris 4 - 7, dilakukan penyelesaian jalur masing-masing agen menggunakan *low-level searching*, pada penelitian ini menggunakan *A* searching* yang akan dibahas dibawah ini.
4. Pada baris 17 - 21, dilakukan algoritma untuk mendapatkan *node* dari openSet yang memiliki *cost* terendah.
5. Pada baris 26 - 27, dilakukan pemeriksaan apabila pada *solution* saat ini tidak terdapat *conflict*, maka *solution* tersebut dapat digunakan.
6. Selanjutnya, pada baris 29 - 35, apabila terdapat *conflict*, data *conflict* tersebut digunakan untuk bahan acuan dalam pembuatan *constraint* yang nantinya akan digunakan pada iterasi selanjutnya.

5.4 Implementasi *Low-level A* search*

Seperti yang telah dijelaskan sebelumnya bahwa ada *conflict-based search* memerlukan bantuan dari *low-level searching*, maka pada penelitian ini *A* search* digunakan sebagai algoritma dalam melakukan *low-level searching*. Algoritma dari *A* search* akan dibahas pada tabel dibawah ini.

5.4.1 Pseudocode *A* Search*

Implementasi dari kode untuk menjalankan *A* searching* untuk mendapatkan jalur dari suatu titik mulai menuju titik selesai akan dijelaskan dalam bentuk pseudocode seperti pada tabel 5.7.

Tabel 5.7 Pseudocode SolvePath

Penerapan SolvePath	
1	def List<Node> SolvePath(Node[,] map, Node startNode, Node
2	targetNode):
3	levelGrid.ResetGrid(map)
4	
5	List<Node> openSet = new List<Node>()
6	HashSet<Node> closedSet = new HashSet<Node>()
7	openSet.Add(startNode)
8	
9	while (openSet.Count > 0)
10	Node currentNode = openSet[0]
11	for (int i = 1; i < openSet.Count; i++)
12	if (openSet[i].fCost < currentNode.fCost || openSet[i].fCost
13	== currentNode.fCost && openSet[i].hCost < currentNode.hCost)
14	currentNode = openSet[i]
15	end if
16	end for
17	
18	openSet.Remove(currentNode)
19	closedSet.Add(currentNode)
20	
21	if (currentNode == targetNode)
22	return RetracePath(startNode, targetNode)
23	end if
24	
25	foreach (Node neighbour in levelGrid.GetNeighbours(map,

```
26  currentNode))
27        if (!neighbour.walkable || closedSet.Contains(neighbour))
28          continue
29        end if
30
31        int newMovementCostToNeighbour = currentNode.gCost +
32  GetDistance(currentNode, neighbour)
33        if (newMovementCostToNeighbour < neighbour.gCost ||
34  !openSet.Contains(neighbour))
35          neighbour.gCost = newMovementCostToNeighbour
36          neighbour.hCost = GetDistance(neighbour, targetNode)
37          neighbour.parent = currentNode
38
39          if (!openSet.Contains(neighbour))
40            openSet.Add(neighbour)
41          end if
42        end if
43      end for
44    end while
45    return new List<Node>()
```

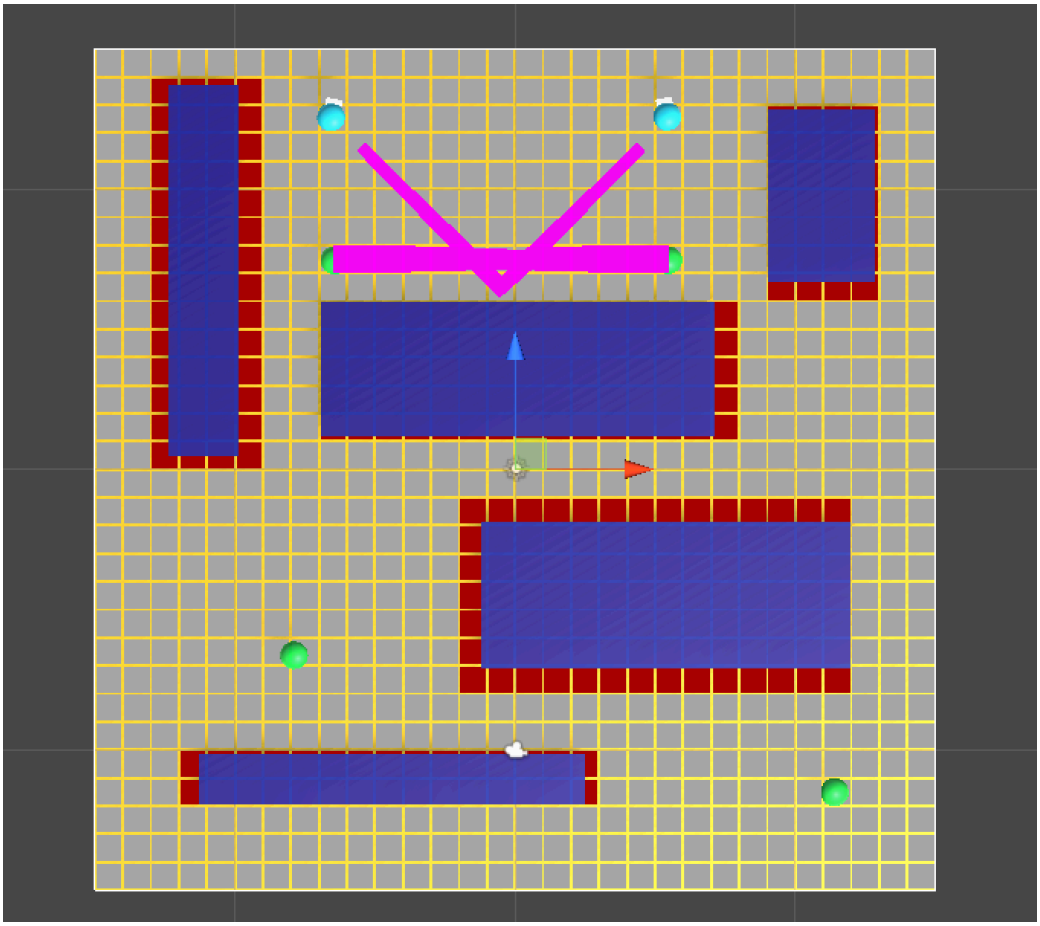
Penjelasan:

1. Definisi method terdapat pada baris 1
2. Baris 2 - 11 digunakan untuk inisialisasi variabel yang dibutuhkan
3. Pada baris 12 - 15, dilakukan pemilihan terhadap *node* yang ada pada openSet yang memiliki jarak heuristik terendah.
4. Pada baris 25 - 43, kode program akan menjalankan pemeriksaan untuk masing-masing tetangga dari *node* yang sedang aktif, dilakukan penghitungan jarak beradasarkan metode heuristik, dan memasukkannya ke openSet apabila belum terdapat pada closeSet.

5.5 Implementasi Sistem Keseluruhan

Pada subbab ini, implementasi keseluruhan dari sistem yang telah dirancang berdasarkan pseudocode yang telah didefinisikan, dilakukan pemrograman menggunakan bahasa pemrograman C# dengan memanfaatkan SDK dari Unity, maka dihasilkanlah seperti gambar dibawah ini.

5.5.1 Implementasi pada *level simple*



Gambar 5.2 Hasil pathfinding *simple grid* game

Urgensi dari melakukan implementasi pada level *simple* adalah untuk memastikan algoritma yang diimplementasikan benar-benar berjalan seperti seharusnya sebelum dilakukan implementasi pada level yang lebih kompleks.

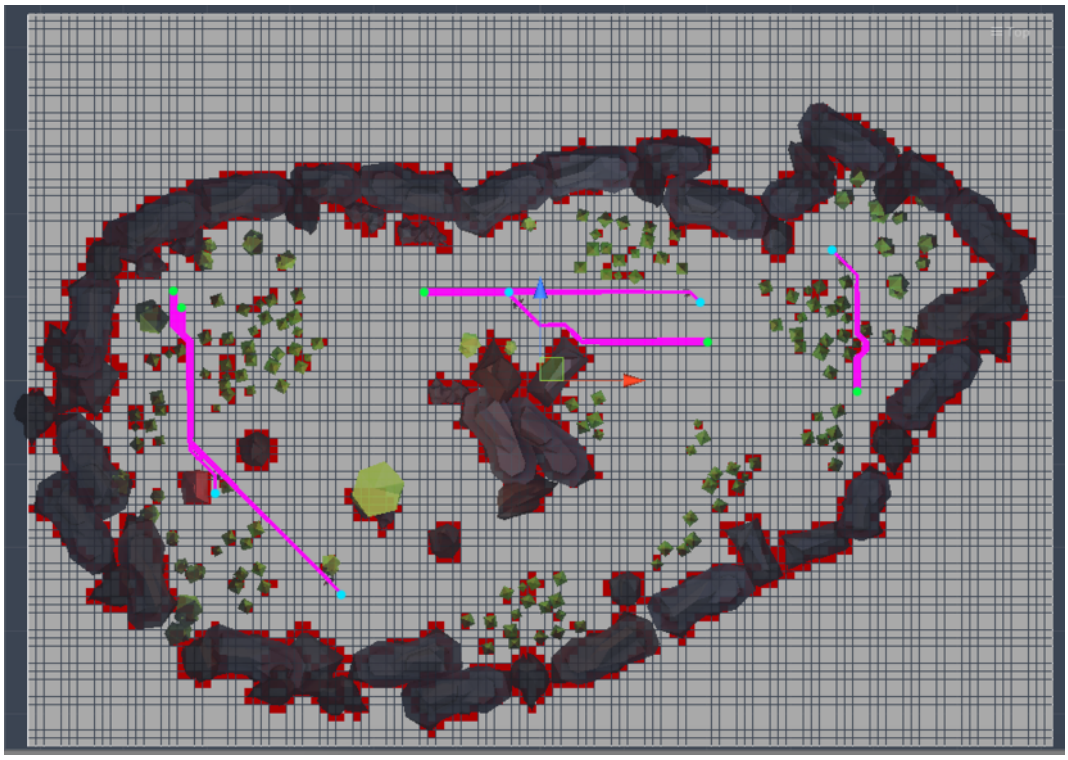
Pada gambar 5.2 diatas dapat dilihat salah satu agen mencoba untuk mencari jalan lain sehingga *constraint* yang telah didefinisikan pada masalah penelitian dapat terhindar sehingga tidak terjadi *conflict* pada saat pembuatan rute perjalanan untuk masing-masing agen.

Dapat dilihat bahwa grid yang dihasilkan memiliki dua *state*, yaitu berwarna merah dan berwarna putih. Node yang berwarna putih berarti node yang dapat ditempati oleh agen dan node yang berwarna merah adalah node yang tidak bisa ditempati oleh agen.

Pada gambar 5.2, agen direpresentasikan dengan lingkaran berwarna biru muda, sedangkan lingkaran yang berwarna hijau adalah lokasi dimana agen akan berjalan. Setiap agen akan memiliki set lokasi kemana agen tersebut akan berjalan. Proses *multi-agent path finding* akan berjalan yang dimulai dengan dijalankannya *low-level searching* pada masing-masing agen, kemudian *output* dari *low-level searching* akan dijadikan *input* oleh *high-level searching* untuk

dicari *conflict*-nya. Apabila *solution* tidak memiliki *conflict* maka hasil *low-level searching* pada *solution* yang tidak memiliki *conflict* akan dijadikan rute perjalanan masing-masing agen yang digambarkan dengan garis berwarna merah muda.

5.5.2 Implementasi pada *level complex*



Gambar 5.3 Hasil pathfinding complex grid game

Urgensi dalam melakukan implementasi pada level yang lebih kompleks adalah untuk memenuhi batasan masalah dimana level dilakukan pengujian adalah level dengan ukuran 500x300 unit.

Pada gambar 5.3 diatas dapat dilihat terdapat banyak agen secara langsung mencoba untuk mencari jalan agar mereka tetap memenuhi *constraint* yang telah ditetapkan. Hal tersebut dibuktikan dengan tidak adanya agen yang bertabrakan pada suatu waktu yang sama.

Titik berwarna biru adalah titik lokasi dimana agen berada, titik berwarna hijau adalah lokasi tujuan agen tersebut, dan garis berwarna merah muda adalah garis jalur dari agen tersebut.