

LAPORAN PRAKTIKUM KECERDASAN BUATAN



2022

Jobsheet Informed

2141720019

Bima Bayu Saputra

TI-2C

Daftar Isi

PRAKTIKUM 1	3
A) BUATLAH SEBUAH CLASS GRAPH, YANG DIDALAMNYA TERDIRI ATAS BEBERAPA FUNGSI ATAUPROSEDUR. MULAI DARI PROSEDUR INIT UNTUK DEKLARASI ADJACENT LIST MULAI DARI NODE START,PROSEDUR GET_NEIGHBORS UNTUK ADJACENT LIST DENGAN NODE-NODE TETANGAA MENUJU GOAL,FUNGSI HEURISTIC YANG MEMBERIKAN NILAI SAMA UNTUK SEMUA NODE YANG ADA, DAN FUNGSI-FUNGSI YANG AKAN DI JELASKAN PADA TAHAP SELANJUTNYA. UNTUK KODE PROGRAM AWAL TAHAP INI DENGAN PYTHON ADALAH SEBAGAI BERIKUT:	3
B) MASIH DALAM CLASS YANG SAMA, BUATLAH FUNGSI ALGORITMA A* SEBAGAI BERIKUT :	4
C) TAHAP SELANJUTNYA ADALAH BUATLAH ADJACENCY LIST, KEMUDIAN CARI SOLUSINYA DENGAN ALGORITMA A* DENGAN CODE PROGRAM SEBAGAI BERIKUT.	5
RUN	5
PERTANYAAN.....	5
PRAKTIKUM 2	7
A) UNTUK MEMPERDALAM PENGERTIAN ALGORITMA GREEDY, PADA PRAKTIKUM MAHASISWA AKAN MENGIMPLEMENTASIKAN ALGORITMA YANG TELAH DIJELASKAN PADA BAGIAN SEBELUMNYA KE DALAM KODE PROGRAM. CONTOH KODE PROGRAM AKAN DIBERIKAN DALAM BAHASA PEMROGRAMAN PYTHON. SEBAGAI LANGKAH AWAL, TERLEBIH DAHULU HARUS MEREPRESENTASIKAN GRAPH. PADA IMPLEMENTASI INI, GRAPH DIREPRESENTASIKAN DENGAN MENGGUNAKAN DICTIONARY DI DALAM DICTIONARY, SEPERTI BERIKUT:	7
B) SELANJUTNYA KITA AKAN MEMBUAT FUNGSI ALGORITMA GREEDY SEBAGAI BERIKUT:	7
C) DATA HEURISTIK (STRAIGHT-LINE DISTANCE) DARI MASING-MASING SIMPUL KE GOAL STATE, B SEBAGAI BERIKUT:	8
RUN	8
PERTANYAAN.....	8
PRAKTIKUM 3	10
A) DENGAN MENGGUNAKAN PYTHON, KITA DAPAT MENYELESAIKAN MASALAH KNAPSACK DIATAS. BUATLAH SEBUAH CODING LIST PYTHON, YANG DI DALAM NYA TERDIRI DARI PENGINISIALISASIAN ITEM SERTA FUNGSI ATAU PROSEDUR KNAPSACK. UNTUK CODE PROGRAM AWAL TAHAP INI DENGAN PYTHON, SEBAGAI BERIKUT:	10
B) SETELAH ITU BUATLAH LIST CODING YANG BERISI FUNGSI KNAPSACK SEBAGAI BERIKUT:	10
RUN	11
TUGAS	12
LINK GITHUB	14

Praktikum 1

- a) Buatlah sebuah class graph, yang didalamnya terdiri atas beberapa fungsi ataupun prosedur. Mulai dari prosedur init untuk deklarasi adjacent list mulai dari node start, prosedur get_neighbors untuk adjacent list dengan node-node tetangga menuju goal, fungsi heuristic yang memberikan nilai sama untuk semua node yang ada, dan fungsi-fungsi yang akan di jelaskan pada tahap selanjutnya. Untuk kode program awal tahap ini dengan Python adalah sebagai berikut:

```
J2 > venv > Praktikum1.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
1 # Praktikum 1
2
3 class Graph:
4     def __init__(self, adjacency_list):
5         self.adjacency_list = adjacency_list
6
7     def get_neighbors(self, v):
8         return self.adjacency_list[v]
9
10    def h(self, n):
11        H = {
12            'A': 1,
13            'B': 1,
14            'C': 1,
15            'D': 1
16        }
17        return H[n]
```

b) Masih dalam class yang sama, buatlah fungsi algoritma A* sebagai berikut :

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help J2 - D:\Document\Pycharm\J2\venv\Praktikum1.py
J2 > venv > Praktikum1.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
18
19 def a_star_algorithm(self, start_node, stop_node):
20     open_list = set([start_node])
21     closed_list = set([])
22
23     g = {}
24
25     g[start_node] = 0
26     parents = {}
27     parents[start_node] = start_node
28
29     while len(open_list) > 0:
30         n = None
31
32         for v in open_list:
33             if n == None or g[v] + self.h(v) < g[n] + self.h(n):
34                 n = v;
35
36         if n == None:
37             print('Path does not exist!')
38             return None
39
40         if n == stop_node:
41             reconst_path = []
42
43             while parents[n] != n:
44                 reconst_path.append(n)
45                 n = parents[n]

```

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help J2 - D:\Document\Pycharm\J2\venv\Pr
J2 > venv > Praktikum1.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
48
49         reconst_path.reverse()
50
51         print('Path found: {}'.format(reconst_path))
52         return reconst_path
53     for (m, weight) in self.get_neighbors(n):
54         if m not in open_list and m not in closed_list:
55             open_list.add(m)
56             parents[m] = n
57             g[m] = g[n] + weight
58         else:
59             if g[m] > g[n] + weight:
60                 g[m] = g[n] + weight
61                 parents[m] = n
62
63             if m in closed_list:
64                 closed_list.remove(m)
65                 open_list.add(m)
66         open_list.remove(n)
67         closed_list.add(n)
68
69     print('Path does not exist!')
70     return None
71
72     adjacency_list = {
73         'A': [('B', 1), ('C', 3), ('D', 7)],
74         'B': [('D', 5)],
75         'C': [('D', 12)]

```

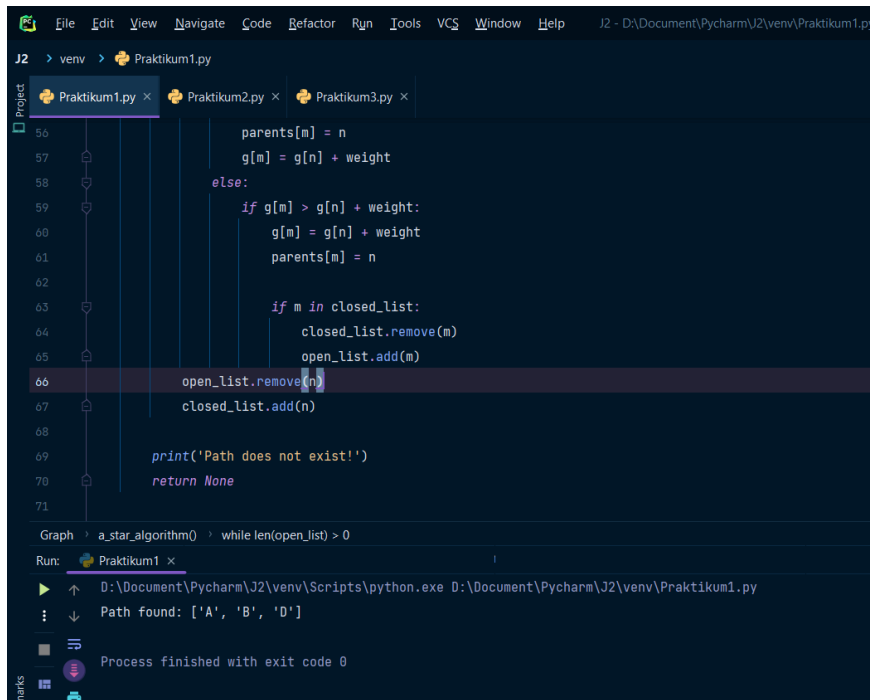
- c) Tahap selanjutnya adalah buatlah adjacency list, kemudian cari solusinya dengan algoritma A* dengan code program sebagai berikut.

```

71
72 adjacency_list = {
73     'A': [('B', 1), ('C', 3), ('D', 7)],
74     'B': [('D', 5)],
75     'C': [('D', 12)]
76 }
77 graph1 = Graph(adjacency_list)
78 graph1.a_star_algorithm('A', 'D')

```

RUN



```

56 parents[m] = n
57 g[m] = g[n] + weight
58 else:
59     if g[m] > g[n] + weight:
60         g[m] = g[n] + weight
61         parents[m] = n
62
63     if m in closed_list:
64         closed_list.remove(m)
65         open_list.add(m)
66 open_list.remove(n)
67 closed_list.add(n)
68
69 print('Path does not exist!')
70 return None
71
Graph > a_star_algorithm() > while len(open_list) > 0
Run:
D:\Document\Pycharm\J2\venv\Scripts\python.exe D:\Document\Pycharm\J2\venv\Praktikum1.py
Path found: ['A', 'B', 'D']
Process finished with exit code 0

```

Pertanyaan

- Amati output pada percobaan 1, dan jelaskan bagaimana hasilnya?
- Jelaskan tahapan-tahapan pada fungsi algoritma A* di percobaan 1 langkah ke-2 yang sudah dijelaskan di atas.
- Apakah tujuan pembuatan fungsi heuristik?
- Bagaimanakah kompleksitas waktu pada algoritma A*? Jelaskan dan beri contoh!
- Jelaskan maksud dari code program di bawah ini:

Jawab:



- a) Outputnya akan menjadi : $A \rightarrow B \rightarrow D$, kenapa begitu? karena setiap kali Lookup neighbor dari suatu node, akan di check mana neighbours dengan F_SCORE paling rendah, ie : $A \rightarrow [B - 1, C - 3, D - 7] \rightarrow$ next node yang akan di proses lagi adalah B
- b) Heuristic hanyalah sebuah angan fungsi untuk menghitung perkiraan jarak antara current node dan end node yang nantinya akan di gunakan untuk decide mana node yang ter-dekat (sedekat aku dan dia sebelumnya) yang akan di set sebagai current node
- c) In short, tak ada bedanya A^* dan dijkstra, bedanya cuman di penggunaan Heuristic Function doang, jadi Time complexity dari si A^* depends on Heuristic function nya seperti apa, jika kita ber-asumsi Heuristic Function dari $A^* \rightarrow O(1) \rightarrow$ seperti di contoh code, maka Time Complexity Dari $A^* ==$ Dijkstra $\rightarrow O(|E| + |V| \log |V|)$
- d) Menunjukkan hubungan graf dan nilai.

Praktikum 2

- a) Untuk memperdalam pengertian algoritma greedy, pada praktikum mahasiswa akan mengimplementasikan algoritma yang telah dijelaskan pada bagian sebelumnya ke dalam kode program. Contoh kode program akan diberikan dalam Bahasa pemrograman python. Sebagai langkah awal, terlebih dahulu harus merepresentasikan graph. Pada implementasi ini, graph direpresentasikan dengan menggunakan dictionary di dalam dictionary, seperti berikut:

```
J2 > venv > Praktikum2.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
1 # Praktikum 2
2
3 DAG = {
4     'A': {'C': 4, 'G': 9},
5     'G': {'E': 6},
6     'C': {'D': 6, 'H': 12},
7     'D': {'E': 7},
8     'H': {'F': 15},
9     'E': {'F': 8},
10    'F': {'B': 5}
11 }
12
```

- b) Selanjutnya kita akan membuat fungsi algoritma Greedy sebagai berikut:

```
13 def shortest_path(graph, source, dest):
14     result = []
15     result.append(source)
16
17     while dest not in result:
18         current_node = result[-1]
19         local_max = min(graph[current_node].values())
20         for node, weight in graph[current_node].items():
21             if weight == local_max:
22                 result.append(node)
23     return result
24
```

- c) Data heuristik (straight-line distance) dari masing-masing simpul ke goal state, B sebagai berikut:

RUN

```

1 # Praktikum 2
2
3 DAG = {
4     'A': {'C': 4, 'G': 9},
5     'G': {'E': 6},
6     'C': {'D': 6, 'H': 12},
7     'D': {'E': 7},
8     'H': {'F': 15},
9     'E': {'F': 8},
10    'F': {'B': 5}
11 }
12
13 def shortest_path(graph, source, dest):
14     result = []
15     result.append(source)
16
17 shortest_path()
18
19 Run:
20 D:\Document\Pycharm\J2\venv\Scripts\python.exe D:\Document\Pycharm\J2\venv\Praktikum2.py
21 ['A', 'C', 'D', 'E', 'F']
22
23 Process finished with exit code 0
  
```

Pertanyaan

- Amati output pada percobaan 1, dan Jelaskan bagaimana hasilnya? Jangan lupa tambahkan line-code untuk menampilkan jalur pencarian dengan A sebagai titik awal dan F sebagai tujuan.
- Jelaskan tahapan-tahapan pada fungsi algoritma Greedy di percobaan 1 langkah ke 2 yang sudah dijelaskan di atas.
- Bagaimanakah kompleksitas waktu pada algoritma Greedy? Jelaskan dan beri contoh!
- Tampilkan urutan simpul dari lintasan yang dipilih oleh algoritma greedy search dan hitung cost-nya.

Jawab:

- Outputnya akan menjadi : [A, B, D, E, F]


```

24
25 print(shortest_path(DAG, 'A', 'F'))

```

shortest_path()

Version Control Run Python Packages TODO Py Python

Enable Auto Reset Color Scheme?: Do you know that you can automatically

- b) Buat sebuah list, katakanlah visited yang nantinya menampung keluh kesah node yang sudah di kunjungi
 - Lalu, apppend first node ke visited
 - Lalu, lakukan looping dengan kondisi selama destination node belum ada di visited node
 - Lalu, dapatkan hatinya neighbor dari current node dengan cost terkecil (sekecil harapkanu mendapatkannya)
 - Oke, masukkan node dengan cost terkecil tersebut ke visited node
- c) Time complexity dari Greedy Algorithm adalah $O(N)$ linear
- d) $A \rightarrow C : 4 \mid A \rightarrow G : 9 \mid C \rightarrow D : 6 \mid C \rightarrow H : 12 \mid D \rightarrow E : 7 \mid E \rightarrow F : 8 \mid F \rightarrow A \rightarrow C \rightarrow E \rightarrow F$

Praktikum 3

- a) Dengan menggunakan python, kita dapat menyelesaikan masalah knapsack diatas. Buatlah sebuah coding list python, yang di dalam nya terdiri dari penginisialisasian item serta fungsi atau prosedur knapsack. Untuk code program awal tahap ini dengan Python, sebagai berikut:

```

1  # Praktikum 3
2
3  item = [[3, 4], [4, 5], [1, 2], [7, 5], [6, 5], [8, 8], [9, 11]]
4
5  from operator import itemgetter, attrgetter
6
7  w = [3, 4, 1, 7, 6, 8, 9]
8  p = [4, 5, 2, 5, 5, 8, 11]
9  item = [[3, 4], [4, 5], [1, 2], [7, 5], [6, 5], [8, 8], [9, 11]]
  
```

- b) Setelah itu buatlah list coding yang berisi fungsi knapsack sebagai berikut:

```

10 i=0
11 while i < len(item):
12     hasil = item[i][1]/item[i][0]
13     item[i].append(hasil)
14     i+=1
15 data = sorted(item, key=itemgetter(2), reverse=True)
16
17 def knapsack(data, cap, flag):
18     total=0
19     tres = ""
20     if(flag==0):
21         dataS = sorted(data, key=itemgetter(flag), reverse=True)
22         tres="bobot prioritas : "
23     elif(flag==1):
24         dataS = sorted(data, key=itemgetter(flag), reverse=True)
25         tres="keuntungan prioritas : "
26     elif(flag==2):
27         dataS = sorted(data, key=itemgetter(flag), reverse=True)
28         tres="p prioritas : "
29
30     j=0
31     hasil=0
32     # print("sini")
33     cek=0
34     weight=0
35     while j < len(dataS):
36         if(cek+dataS[j][0] <= cap):
37             hasil=hasil+dataS[j][1]
38         j+=1
39     return total, tres
  
```



```

J2 > venv > Praktikum3.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
25 tres="keuntungan prioritas : "
26 elif(flag==2):
27     dataS = sorted(data, key=itemgetter(flag), reverse=True)
28     tres="p prioritas : "
29
30     j=0
31     hasil=0
32     # print("sini")
33     cek=0
34     weight=0
35     while j < len(dataS):
36         if(cek+dataS[j][0] ≤ cap):
37             hasil=hasil+dataS[j][1]
38             weight=weight+dataS[j][0]
39             print(dataS[j][0])
40             cek=weight
41             j+=1
42             #print("here")
43             return("optimal dalam "+str(tres)+str(hasil))
44
45     print(knapsack(data, 20, 0))
46     print(knapsack(data, 20, 1))
47     print(knapsack(data, 20, 2))

```

RUN

```

J2 > venv > Praktikum3.py
Project
Praktikum1.py x Praktikum2.py x Praktikum3.py x
1 # Praktikum 3
2
3 item = [[3, 4], [4, 5], [1, 2], [7, 5], [6, 5], [8, 8], [9, 11]]
4
5 from operator import itemgetter, attrgetter
6 w = [3, 4, 1, 7, 6, 8, 9]
7 p = [4, 5, 2, 5, 5, 8, 11]
8 item = [[3, 4], [4, 5], [1, 2], [7, 5], [6, 5], [8, 8], [9, 11]]
9
10 i=0
11 while i < len(item):
12     hasil = item[i][1]/item[i][0]
13     item[i].append(hasil)
14     i+=1
15 data = sorted(item, key=itemgetter(2), reverse=True)
16
Run: Praktikum3 x
D:\Document\Pycharm\J2\venv\Scripts\python.exe D:\Document\Pycharm\J2\venv\Praktikum3.py
9
optimal dalam bobot prioritas : 11
9
optimal dalam keuntungan prioritas : 11
1
optimal dalam p prioritas : 2
Process finished with exit code 0

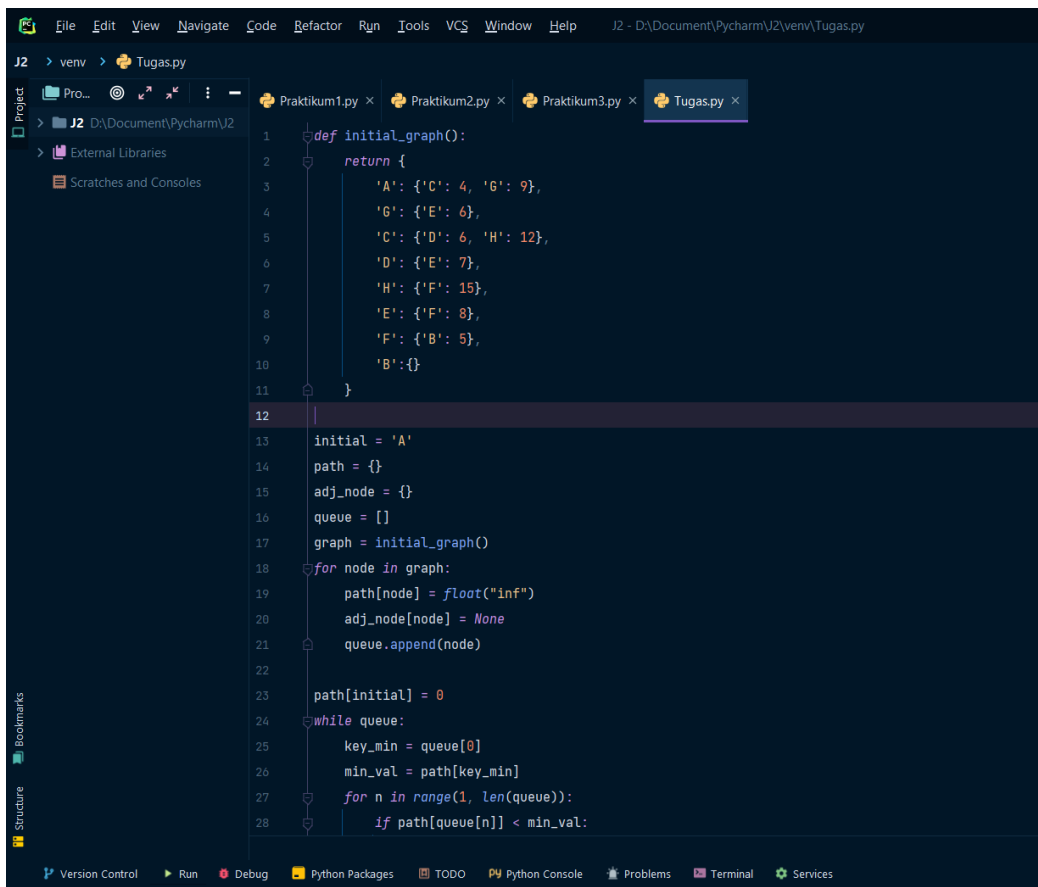
```

Tugas

- Ubahlah kode program pada Percobaan 2 menggunakan metode pencarian jarak yang lain, selain Euclidean distance!
- Jelaskan perbedaan hasilnya pada tugas No. 1!
- Dapatkah algoritma A* diterapkan untuk graph yang tidak berbobot? Jika tidak, apakah alasannya?
- Carilah kegunaan algoritma A* yang diimplementasikan pada game. Jelaskan tahapanya dan juga tampilkan game atau langkah-langkah pembuatanya!

Jawab:

a)



```

1  def initial_graph():
2      return {
3          'A': {'C': 4, 'G': 9},
4          'G': {'E': 6},
5          'C': {'D': 6, 'H': 12},
6          'D': {'E': 7},
7          'H': {'F': 15},
8          'E': {'F': 8},
9          'F': {'B': 5},
10         'B': {}
11     }
12
13 initial = 'A'
14 path = {}
15 adj_node = {}
16 queue = []
17 graph = initial_graph()
18 for node in graph:
19     path[node] = float("inf")
20     adj_node[node] = None
21     queue.append(node)
22
23 path[initial] = 0
24 while queue:
25     key_min = queue[0]
26     min_val = path[key_min]
27     for n in range(1, len(queue)):
28         if path[queue[n]] < min_val:

```

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help J2 - D:\Document\Pycharm\J2\venv\Tugas.py
J2 > venv > Tugas.py
Project
  > J2 D:\Document\Pycharm\J2
  > External Libraries
  > Scratches and Consoles
Praktikum1.py x Praktikum2.py x Praktikum3.py x Tugas.py x
26 min_val = path[key_min]
27 for n in range(1, len(queue)):
28     if path[queue[n]] < min_val:
29         key_min = queue[n]
30         min_val = path[key_min]
31 cur = key_min
32 queue.remove(cur)
33
34 for i in graph[cur]:
35     alternate = graph[cur][i] + path[cur]
36     if path[i] > alternate:
37         path[i] = alternate
38         adj_node[i] = cur
39
40 x = 'F'
41 print('Jalur A ke F')
42 print(x, end='←')
43 while True:
44     x = adj_node[x]
45     if x is None:
46         print("")
47         break
48     print(x, end='←')

```

```

40 x = 'F'
41 print('Jalur A ke F')
42 print(x, end='←')
43 while True:

```

Run: Tugas x

```

D:\Document\Pycharm\J2\venv\Scripts\python.exe D:\Document\Pycharm\J2\venv\Tugas.py
Jalur A ke F
F←E←G←A←
Process finished with exit code 0

```

- Perbedaannya terletak pada pemilihan data awal parent keseluruhan untuk dibandingkan. Pada kode program baru, $weight \geq local.max$, sehingga kode akan melalui weight yang lebih besar,
- Tidak, karena A* dirancang dengan menentukan jalur dengan nilai palai sedikit atau paling pendek jaraknya.
- Dalam sebuah game, salah satu elemen yang dianggap perlu untuk mendukung jalannya dan realitas game adalah bagaimana NPC (Non-Player Character) dalam game tersebut bergerak. A* (A-star) adalah algoritma yang dapat digunakan untuk melakukan pathfinding. Dalam hal ini, A* digunakan untuk mencari jarak terpendek antara NPC dan karakter pemain. Penerapan algoritma A* dalam game pathfinding dibuat berdasarkan prosedur A* untuk mendapatkan langkah terbaik dari posisi titik awal. Algoritma A* merupakan perbaikan dari metode best-first search (BFS) dengan

menggunakan fungsi heuristic. Fungsi heuristic sering juga disebut $f(n)$ yang merupakan penentuan urutan titik yang akan dikunjungi terlebih dahulu. Fungsi heuristic ini sebenarnya menyimbolkan seberapa baik atau mungkin titik yang dikunjungi untuk mencapai titik tujuan. A^* akan meminimumkan total biaya lintasan dan akan memberikan solusi yang terbaik dalam waktu yang optimal. Algoritma A^* selalu dapat menemukan jalur musuh untuk menangkap pemain dan 73% diantaranya adalah jalur yang optimal.

Link Github

https://github.com/BimaBayuUWUUU/KECERDASAN_BUATAN/tree/main/J2