Menggunakan BFS dan DFS untuk membantu menyelesaikan puzzle game I Wanna Lockpick

Devinzen - 13522064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13522064@std.stei.itb.ac.id

***Abstrak*—Pada makalah ini akan dibuat program untuk menyelesaikan level-level pada world 1-3 di game puzzle I Wanna Lockpick dengan algoritma Breadth-First Search dan Depth-First Search. Implementasi program saat ini seharusnya, asumsi waktu dan memori tak terhingga, bisa menyelesaikan semua level di world 1-3 kecuali level 2-10.**

Keywords—BFS, DFS, I Wanna Lockpick, Puzzle

I. Pendahuluan

I Wanna Lockpick adalah video game puzzle yang dibuat oleh Lucas A. Watson. Objektif dari game ini adalah mengambil kunci dan membuka pintu sampai ke goal. Game ini bisa didownload secara gratis di itch.io untuk platform Windows. Bagian 1 pada game ini dirilis pada 3 Desember 2021 dan bagian 2 dirilis pada 14 Febuari 2024.

Game ini mempunyai banyak mekanik, dimana level-level yang lebih jauh mempunyai bilangan negatif, bilangan imajiner, dan lain-lain. Untuk kemudahan implementasi, program yang akan dibuat hanya mengimplementasikan mekanik di 3 world pertama, yaitu pada gambar dibawah.

Gambar 1-5. Tangkapan layar bulk keys & doors, master key, blank door, blast door, dan exact keys.

Keys dan Doors adalah kunci dan pintu yang bisa mempunyai berbagai macam warna. Pintu memerlukan kunci dengan warna yang sama untuk dibuka. Kunci bisa saja lebih dari 1 dan pintu juga bisa memerlukan lebih dari 1 kunci untuk dibuka. Master keys adalah kunci yang bisa membuka pintu apa saja, tak peduli berapa banyak kunci yang diperlukan untuk membuka pintu tersebut awalnya. Blank doors adalah pintu yang hanya bisa dibuka jika tidak memiliki kunci dengan warna tersebut. Blast doors adalah pintu yang hanya memerlukan 1 kunci untuk dibuka, tetapi ketika dibuka akan menghabiskan semua kunci dengan warna tersebut. Exact keys adalah kunci yang mengubah (bukan menambah) jumlah kunci yang dimiliki menjadi jumlah tersebut.

II. Landasan Teori

A. *Algoritma Breadth-First Search (BFS)*

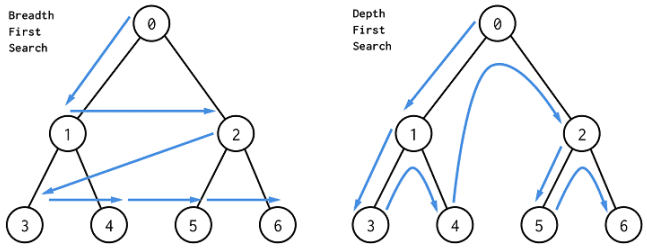
Algoritma ini digunakan untuk menelusuri graf. Pencarian algoritma ini dilakukan secara melebar. Pencarian dilakukan dengan mengunjungi simpul v terlebih dahulu, kemudian mengunjungi simpul-simpul yang bertetangga dengan v, kemudian mengunjungi simpul-simpul yang bertetangga dengan simpul yang tadi dikunjungi, dst.

Jika b adalah branching factor (berapa banyak tetangga simpul) dan n adalah kedalaman, maka kompleksitas waktu adalah O(b^n) dan kompleksitas ruang adalah O(b^n).

B. *Algoritma Depth-First Search (DFS)*

Algoritma ini juga digunakan untuk menelusuri graf. Perbedaannya dengan DFS adalah pada algoritma ini pencarian dilakukan secara mendalam. Kunjungi simpul sedalam mungkin sampai tidak ada lagi simpul bertetangga yang belum dikunjungi, lalu di-backtrack ke simpul terakhir yang dikunjungi sebelumnya yang mempunyai tetangga yang belum dikunjungi.

Kompleksitas waktu algoritma DFS adalah O(b^n) dan kompleksitas ruangnya O(bn).



Gambar 6. Ilustrasi perbedaan penelusuran dengan algoritma BFS dan DFS, Sumber: https://www.freelancinggig.com/blog/2019/02/06/what-is-the-difference-between-bfs-and-dfs-algorithms/

III. IMPLEMENTASI

Pada implementasi penyelesaian puzzle-puzzle di dalam game I Wanna Lockpick, berikut adalah langkah-langkah yang dilakukan:

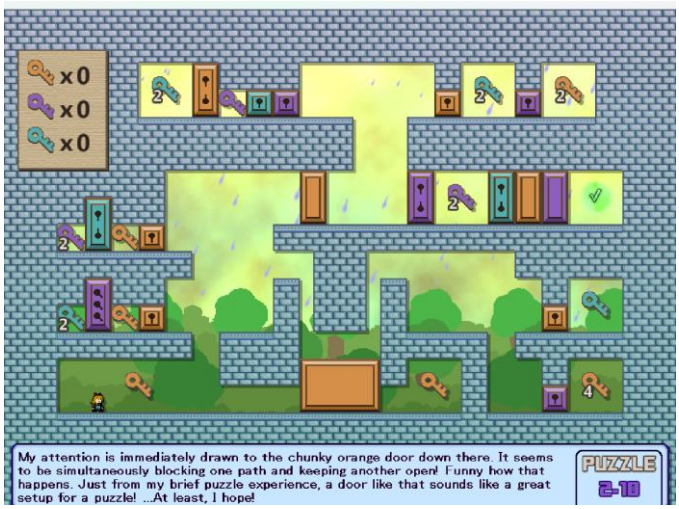
A. Memetakan level menjadi graf

Level-level di dalam game dipetakan menjadi sebuah graf. Simpul graf bisa berupa kunci, pintu, posisi pemain awal, tempat tujuan, atau kosong (untuk menghubungkan simpul lain tanpa menggunakan banyak sisi). Bagian ini dilakukan secara manual, hasil graf akan menjadi input untuk dimasukkan ke program yang akan menyelesaikan puzzle tersebut.

Graf yang dihasilkan di langkah ini berbeda dengan graf penelusuran State yang akan digunakan dalam implementasi algoritma BFS dan DFS. Graf yang ini digunakan untuk menelusuri langkah selanjutnya yang bisa dilakukan. Urutan mengambil kunci/membuka pintu sangat berpengaruh.

Gambar 7. Tangkapan layar level 1-4 dengan menggunakan fitur pencilmarks di dalam game untuk menomori simpul graf dan dianotasi untuk menggambar sisi graf.

Karena di dalam game ada elemen platformer, bisa saja pemain tidak bisa kembali ke posisi sebelumnya setelah jatuh, membuat graf menjadi berarah yang memberitahu dimana pemain tidak bisa kembali lagi. Selain itu, karena pintu juga merupakan block, maka ada beberapa kasus dimana setelah membuka pintu ada jalan yang tidak bisa dilewat lagi, menyebabkan graf berubah. Untuk kemudahan implementasi, implementasi hanya kepada level-level di World 1 sampai 3 yang bisa direpresentasikan dengan graf tak berarah statis.



Gambar 8. Level 2-10 tidak bisa diselesaikan dengan implementasi program saat ini dikarenakan pintu yang memblok jalan ketika dibuka.

B. Penjelajahan Simpul State

Program menyimpan keadaan permainan di dalam kelas State yang memuat inventori kunci yang dimiliki pemain, aksi terakhir yang dilakukan pemain, list untuk menyimpan simpul mana saja pada graf yang sudah dibuka, dan list untuk menyimpan simpul yang bertetangga dengan simpul yang sudah dibuka untuk dijelajahi selanjutnya.

Setiap kali mengambil kunci atau membuka pintu, simpul itu ditandai sudah dibuka, dihilangkan dari list simpul yang bisa dijelajahi, dan semua simpul yang bertetangga dan belum dijelajahi ditambahkan ke list tersebut. Jika ada simpul kosong (yang hanya digunakan untuk menghubungkan simpul lain) maka langsung dibuka secara otomatis.

Dari simpul-simpul yang bisa dijelajahi, ditentukan aksi-aksi yang bisa dilakukan selanjutnya. Aksi bisa berupa membuka pintu atau mengambil kunci, dimana kalau membuka pintu bisa memakai master key (jika ada).

```
    vector<struct Move> possibleNextMove(const vector<boardObject>& b) const {
        vector<struct Move> next;
        for (int i = 0; i < possibleNextNode.size(); i++){
            if (possibleNextNode[i]){
                boardObject test = b[i];
                if (test.getType() == "key"){next.push_back({i, 0});} else {
                    // asumsi door
                    // check normal move
                    if (test.isSpecial() && (inv.getAmount(test.getColor()) > 0)){
                        next.push_back({i, 0}); // cek blast door
                    } else if (test.getAmount() == 0){
                        if (inv.getAmount(test.getColor()) == 0){next.push_back({i, 0});} // cek blank door
                    } else if (inv.getAmount(test.getColor()) >= test.getAmount()){
                        next.push_back({i, 0});
                    }

                    // check master key
                    if ((inv.getAmount("master") > 0) && (test.getColor() != "master")){
                        next.push_back({i, 1});
                    }
                }
            }
        }
        return next;
    }
};
```

Gambar 9, 10. Potongan kode C++ pada sebagian kelas State.

C. Pencarian Solusi dengan Algoritma BFS dan DFS

Pada implementasi algoritma BFS, semua simpul State dimasukkan ke sebuah list yang merepresentasikan pohon. Elemen di dalam list akan diekspansi dan simpul-simpul baru dimasukkan ke belakang list. Karena semua simpul baru dimasukkan ke belakang, jika elemen diiterasi dari depan, maka akan menjelajahi simpul dari kedalaman 1 dulu, lalu 2, 3, dan seterusnya mengikuti urutan penjelajahan algoritma BFS. Jika iterasi sudah mecapai elemen terakhir, artinya tidak ada simpul yang bisa diekspansi lagi, yang berarti tidak ada solusi. Jika solusinya ada, maka iterasi dihentikan.

Langkah-langkah BFS pada implementasi adalah sebagai berikut:

1. Masukkan state awal ke list dan mulai iterasi list.
2. Untuk setiap elemen State, cari aksi yang bisa dilakukan dan masukkan State setelah melakukan aksi tersebut ke belakang list.
3. Jika ada state dimana salah satu simpul yang bisa dijelajahi selanjutnya adalah simpul tempat tujuan, artinya solusi ketemu, hentikan iterasi dan masukkan aksi-aksi untuk mencapai solusi kepada suatu list.
4. Jika iterasi sudah melewati elemen terakhir artinya tidak ada solusi.

```
vector<struct Move> solveBFS(const State& start, const vector<boardObject>& board, const Graph& g){
    vector<struct Move> solution;
    vector<State> stateTree;
    vector<int> previous;
    stateTree.push_back(start);
    previous.push_back(-1);
    for (int i = 0; i < stateTree.size(); i++){
        nodesVisited++;
        vector<struct Move> possibleMoves = stateTree[i].possibleNextMove(board);
        for (int moves = 0; moves < possibleMoves.size(); moves++){
            State newNode(stateTree[i], board, g, possibleMoves[moves]);
            stateTree.push_back(newNode);
            previous.push_back(i);
            if (newNode.getPossibleNextNode()[g.getSize() - 1]){
                int trace = stateTree.size() - 1;
                while (trace > 0){
                    solution.push_back(stateTree[trace].getLastMove());
                    trace = previous[trace];
                }
                return solution;
            }
        }
    }
    return solution;
}
```

Gambar 11. Implementasi algoritma BFS

Pada implementasi algoritma DFS, penjelajahan simpul dilakukan secara rekursif. Langkah-langkahnya adalah sebagai berikut:

1. Mulai mencari dari simpul awal dan mendaftarkan semua aksi yang bisa dilakukan.
2. Untuk setiap aksi, buat State setelah melakukan aksi itu. Cari apakah bisa mencapai tujuan dari State itu secara rekursif.
3. Jika menemukan solusi, maka akan return dari rekursi, menambahkan aksi yang dilakukan untuk mencapai tujuan ke dalam suatu list.
4. Jika pada setiap aksi tidak bisa mencapai tujuan, maka tidak ada solusi.

```
bool recursion(const State& s, const vector<boardObject>& board, const Graph& g, vector<struct Move>& solution){
    nodesVisited++;
    vector<struct Move> possibleMoves = s.possibleNextMove(board);
    if (possibleMoves.size() == 0){return 0;}
    for (int moves = 0; moves < possibleMoves.size(); moves++){
        State newNode(s, board, g, possibleMoves[moves]);
        if (newNode.getPossibleNextNode()[g.getSize() - 1]){
            solution.push_back(possibleMoves[moves]);
            return 1;
        }
        if (recursion(newNode, board, g, solution)){
            solution.push_back(possibleMoves[moves]);
            return 1;
        }
    }
    return 0;
}

vector<struct Move> solveDFS(const State& start, const vector<boardObject>& board, const Graph& g){
    vector<struct Move> solution;
    recursion(start, board, g, solution);
    return solution;
}
```

Gambar 12. Implementasi algoritma DFS

Setelah memasukkan input graf ke program dan program mengeluarkan output solusi, solusinya dilakukan ke level puzzle secara manual.

IV. TESTING

Program yang dibuat menerima input berupa jumlah kunci/pintu di dalam level, lalu untuk setiap kunci/pintu dimasukkan warna, jumlah, dan jika kunci apakah merupakan exact key atau blast door untuk pintu. Tempat awal pemain otomatis menjadi simpul index 0 dan tempat tujuan di index N+1 dimana N merupakan jumlah kunci dan pintu. Setelah itu, program meminta jumlah sisi di dalam graf, diikuti dengan sisinya.

Keluaran program yang dibuat adalah solusi puzzle tersebut berdasarkan aksi-aksi yang dilakukan untuk mencapai tempat tujuan. Program juga menampilkan jumlah simpul yang dikunjungi dan waktu eksekusi program.

```cpp
int main (){
    int N;
    cout << "Jumlah key/door: " << flush;
    string line;
    getline(cin, line);
    istringstream stream(line);
    stream >> N;
    N += 2; // start dan finish
    Graph g(N);
    vector<boardObject> board;
    board.push_back(boardObject("start", "", 0, 0));
    cout << "0 start" << endl;
    for (int i = 1; i <= (N - 2); i++){
        string type;
        string color;
        int amount;
        bool isSpecial;
        cout << i << " " << flush;
        getline(cin, line);
        stream.clear();
        stream.str(line);
        stream >> type;
        if (type == "empty"){
            board.push_back(boardObject("empty", "", 0, 0));
        } else {
            stream >> color >> amount >> isSpecial;
            board.push_back(boardObject(type, color, amount, isSpecial));
        }
    }
    board.push_back(boardObject("finish", "", 0, 0));
    cout << N - 1 << " finish\nJumlah sisi: " << flush;
    int edge;
    getline(cin, line);
    stream.clear();
    stream.str(line);
    stream >> edge;
```

```cpp
    for (int i = 0; i < edge; i++){
        int node1, node2;
        getline(cin, line);
        stream.clear();
        stream.str(line);
        stream >> node1 >> node2;
        g.addEdge(node1, node2);
    }
    State start(g, board);
    string algorithm;
    cout << "Algoritma (BFS/DFS): " << flush;
    getline(cin, line);
    stream.clear();
    stream.str(line);
    stream >> algorithm;
    vector<struct Move> solution;
    auto startTime = chrono::high_resolution_clock::now();
    if (algorithm == "BFS"){
        solution = solveBFS(start, board, g);
    } else {
        solution = solveDFS(start, board, g);
    }
    auto endTime = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(endTime - startTime);
    printMoves(board, solution);
    cout << "Nodes visited: " << nodesVisited << endl;
    cout << "Execution time: " << duration.count() << " ms" << endl;
    return 0;
```

Gambar 13, 14. Fungsi main dalam program yang dibuat.

A. Tangkapan Layar Test Cases

Test case 1: Level 1-4 “Double Door!” (Tangkapan layar level ini ada di bagian sebelumnya)

```
Jumlah key/door: 16
0 start
1 door orange 1 0
2 door orange 1 0
3 door white 1 0
4 door white 1 0
5 key orange 1 0
6 key orange 1 0
7 key white 1 0
8 door white 1 0
9 door green 1 0
10 key white 1 0
11 key white 1 0
12 key green 1 0
```

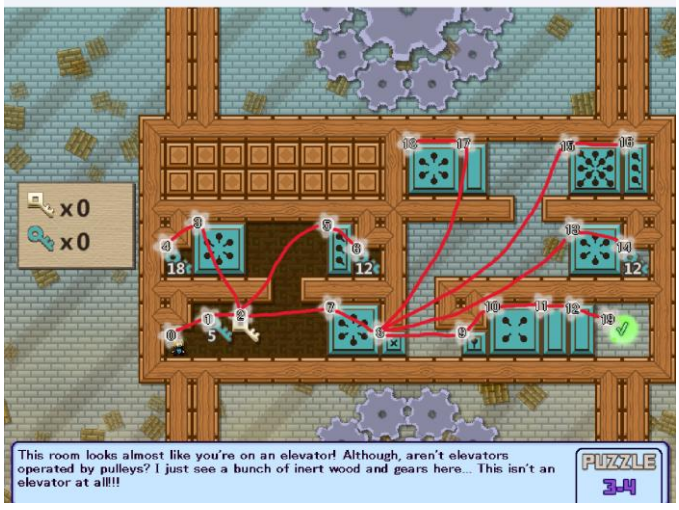
```
13 door green 1 0
14 door green 1 0
15 key orange 1 0
16 empty
17 finish
Jumlah sisi: 18
0 1
1 2
2 17
0 3
3 4
4 5
5 6
0 7
7 8
0 12
```

```
12 13
8 16
13 16
16 9
16 14
9 10
9 11
14 15
Algoritma (BFS/DFS): BFS
ambil key 7 (1 white)
buka door 8 (1 white)
ambil key 12 (1 green)
buka door 9 (1 green)
ambil key 10 (1 white)
buka door 3 (1 white)
ambil key 11 (1 white)
buka door 4 (1 white)
ambil key 5 (1 orange)
buka door 1 (1 orange)
ambil key 6 (1 orange)
buka door 2 (1 orange)
Nodes visited: 120
Execution time: 21 ms
```

```
Algoritma (BFS/DFS): DFS
ambil key 7 (1 white)
buka door 8 (1 white)
ambil key 12 (1 green)
buka door 9 (1 green)
ambil key 10 (1 white)
buka door 3 (1 white)
ambil key 11 (1 white)
buka door 4 (1 white)
ambil key 5 (1 orange)
buka door 1 (1 orange)
ambil key 6 (1 orange)
buka door 2 (1 orange)
Nodes visited: 15
Execution time: 0 ms
```

Gambar 15 – 18. Tangkapan layar tampilan input dan output untuk algoritma BFS dan DFS

Test case 2: Level 3-4 “Cargo Lift”. Level ini mempunyai master key, blank door, dan blast door. Selain master key, satu-satunya warna pintu dan kunci yang ada adalah cyan.



Gambar 19. Tangkapan layar yang dianotasi untuk menggambarkan graf level 3-4

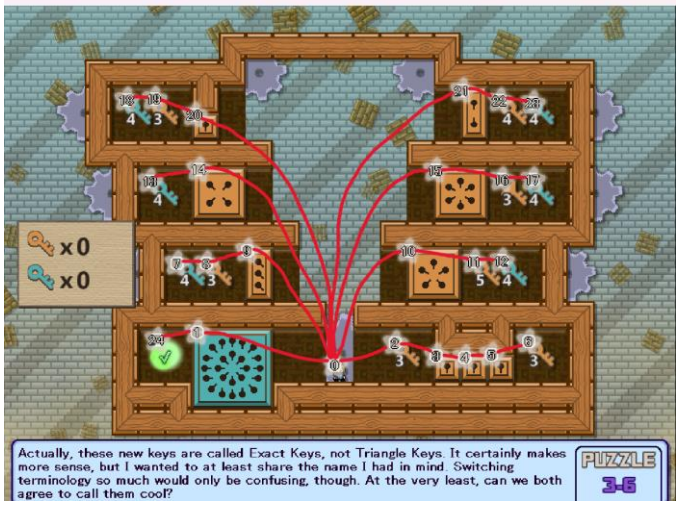
```
Jumlah key/door: 18
0 start
1 key cyan 5 0
2 key master 1 0
3 door cyan 6 0
4 key cyan 18 0
5 door cyan 3 0
6 key cyan 12 0
7 door cyan 6 0
8 door cyan 0 1
9 door cyan 1 0
10 door cyan 4 0
11 door cyan 0 0
12 door cyan 0 0
13 door cyan 6 0
14 key cyan 12 0
15 door cyan 12 0
16 door cyan 3 0
17 door cyan 0 0
18 door cyan 6 0
19 finish
Jumlah sisi: 19
0 1
1 2
2 3
3 4
2 5
5 6
2 7
7 8
8 9
9 10
10 11
11 12
8 13
13 14
8 15
15 16
8 17
17 18
12 19
```

```
Algoritma (BFS/DFS): BFS
ambil key 1 (5 cyan)
ambil key 2 (1 master)
buka door 5 (3 cyan)
ambil key 6 (12 cyan)
buka door 3 (6 cyan)
ambil key 4 (18 cyan)
buka door 7 (6 cyan)
buka blast door 8 (cyan) dengan master key
buka door 9 (1 cyan)
buka door 10 (4 cyan)
buka door 15 (12 cyan)
buka door 16 (3 cyan)
buka door 11 (0 cyan)
buka door 12 (0 cyan)
Nodes visited: 5950
Execution time: 286 ms
```

```
Algoritma (BFS/DFS): DFS
ambil key 1 (5 cyan)
ambil key 2 (1 master)
buka door 5 (3 cyan)
ambil key 6 (12 cyan)
buka door 3 (6 cyan)
ambil key 4 (18 cyan)
buka door 7 (6 cyan)
buka blast door 8 (cyan) dengan master key
buka door 9 (1 cyan)
buka door 10 (4 cyan)
buka door 15 (12 cyan)
buka door 16 (3 cyan)
buka door 11 (0 cyan)
buka door 12 (0 cyan)
Nodes visited: 1464
Execution time: 25 ms
```

Gambar 20 – 23. Tangkapan layar input/output untuk level 3-4.

Test case 3: Level 3-6 “Profit Margin”. Level ini mempunyai satu exact key dengan nilai 3, yang artinya jumlah kunci yang dimiliki pemain dengan warna itu di-set menjadi 3 ketika diambil.



Gambar 24. Tangkapan layar yang dianotasi untuk menggambarkan graf level 3-6

```
Jumlah key/door: 23
0 start
1 door cyan 24 0
2 key orange 3 0
3 door orange 1 0
4 door orange 1 0
5 door orange 1 0
6 key orange 3 0
7 key cyan 4 0
8 key orange 3 0
9 door orange 3 0
10 door orange 5 0
11 key orange 5 0
12 key cyan 4 0
13 key cyan 4 0
14 door orange 4 0
15 door orange 6 0
16 key orange 3 1
17 key cyan 4 0
18 key cyan 4 0
19 key orange 3 0
20 door orange 1 0
21 door orange 2 0
22 key orange 4 0
23 key cyan 4 0
24 finish
Jumlah sisi: 24
0 1
1 24
0 2
2 3
3 4
4 5
5 6
0 9
9 8
8 7
0 10
10 11
11 12
0 14
14 13
0 15
15 16
16 17
0 20
20 19
19 18
0 21
21 22
22 23
Algoritma (BFS/DFS): DFS
ambil key 2 (3 orange)
buka door 3 (1 orange)
buka door 20 (1 orange)
ambil key 19 (3 orange)
buka door 9 (3 orange)
ambil key 8 (3 orange)
ambil key 7 (4 cyan)
ambil key 18 (4 cyan)
buka door 21 (2 orange)
ambil key 22 (4 orange)
buka door 10 (5 orange)
ambil key 11 (5 orange)
ambil key 12 (4 cyan)
buka door 15 (6 orange)
ambil exact key 16 (3 orange)
buka door 4 (1 orange)
buka door 5 (1 orange)
ambil key 6 (3 orange)
buka door 14 (4 orange)
ambil key 13 (4 cyan)
ambil key 17 (4 cyan)
ambil key 23 (4 cyan)
buka door 1 (24 cyan)
Nodes visited: 7731264
Execution time: 141534 ms
```

```
0 21
21 22
22 23
Algoritma (BFS/DFS): BFS
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
```

Gambar 25 – 27. Tangkapan layar input/output untuk level 3-6. Memori tidak cukup untuk menyimpan State-state pada algoritma BFS.

Pada test case ini, algoritma BFS tidak menghasilkan solusi. Ini dikarenakan pada BFS semua simpul harus disimpan, dan memakan banyak memori. Ini menyebabkan

kompleksitas ruang O(b^n). Pada program yang dibuat, simpul-simpul dimasukkan ke dalam list sampai akhirnya program tidak bisa lagi mengalokasikan memori yang cukup, sehingga melempar std::bad_alloc.

B. Hasil Pengujian

Pada semua test case, algoritma BFS lebih lambat daripada algoritma DFS. Hal ini disebabkan karena pada puzzle di game ini, solusi optimalnya kebanyakan memakai hampir semua kunci dan pintu yang tersedia. Algoritma BFS mengecek semua simpul di kedalaman yang sama dulu sebelum lanjut ke kedalaman selanjutnya. Selain itu, pada program yang dibuat urutan penelusuran berdasarkan nomor simpul, yang artinya jika kebetulan beruntung dimana beberapa langkah-langkah awal untuk mencapai solusi urutan nomornya kecil, algoritma DFS bisa memberikan solusi dengan sangat cepat.

Untuk test case yang ketiga, ada banyak jalur yang bisa dilalui. Pencarian solusi dengan algoritma DFS menelusuri 7,7 juta node dan memakan waktu 141 detik. Menyimpan jumlah State sebanyak itu dimana setiap State mempunyai 3 list akan memakan memori yang terlalu besar, itulah mengapa algoritma BFS tidak menghasilkan solusi pada test case ini.

V. KESIMPULAN DAN SARAN

Untuk sekarang, program yang dibuat seharusnya, asumsi mempunyai memori dan waktu tak terhingga, bisa menyelesaikan semua level di World 1-3 kecuali level 2-10. Algoritma DFS pada kasus ini lebih cepat daripada BFS, meskipun kompleksitas waktunya sama.

Dengan beberapa (banyak) perubahan, program bisa diperbaiki menjadi lebih baik dengan menangani graf level yang bisa berarah/berubah dan mekanik yang lebih banyak.

Sayangnya, tidak semua level bisa diselesaikan dengan DFS, dikarenakan ada level yang memiliki negative star master key dan memungkinkan duplikasi pintu sebanyak mungkin, yang artinya kedalaman bisa saja tak terhingga. Jika ada yang ingin mengembangkan program menjadi lebih bagus lagi, bisa dicari algoritma lain untuk menyelesaikan puzzle-puzzle yang ada didalam game ini.

REFERENSI

[1] https://iwannalockpickwiki.miraheze.org/wiki/I_Wanna_Lockpick, diakses pada 12 Juni 2024.

[2] R. Munir dan N. U. Maulidevi. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf, diakses pada 12 Juni 2024.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Devinzen 13522064