Implementasi Algoritma Dijkstra pada Game Strategi RPG Berbasis Web dengan Framework Javascript P5

M. Syaiful Amin[1], Pungkas Subarkah[2], Rofiqul Umma[3], Eko Budi Prasetya[4]
Universitas Amikom Purwokerto
Jl. Letjend. Pol Soemarto purwanegara purwokerto utara, Banyumas
Email : [1]syaifulamin@amikompurwokerto.ac.id, [2]subarkah@amikompurwokerto.ac.id,
[3]rofiqulumma@gmail.com [4]ekobp1902@gmail.com

Abstract

Graph theory is a branch of mathematics that studies the graph of a problem that arises, an example of a problem related to graph theory is determining the shortest distance from one point to another. Dijkstra's algorithm is used to determine the shortest path (Shortest Path Problem) for a graph. The author conducted several trials related to the algorithm used. The author also conducted a literature study to determine the existing problems. In this study, we chose the implementation of graph theory in the RPG strategy game that we made with the algorithm used, namely the Dijkstra Algorithm. In this study, the author carried out various stages such as concept generation, display and component design, game creation, testing, analysis, and results. The result of this research is a web-based game created using the Javascript Framework p5 which implements the Dijkstra algorithm so that it can determine the shortest route to a predetermined target.

Keywords: *Graph theory, RPG Game, Dijkstra Algorithm*

Abstraksi

Teori graf adalah cabang matematika yang mempelajari graf dari suatu masalah yang muncul, contoh masalah yang berkaitan dengan teori graf adalah menentukan jarak terpendek dari satu titik ke titik lainnya. Algoritma Dijkstra digunakan untuk menentukan jalur terpendek (Shortest Path Problem) untuk suatu graf. Penulis melakukan beberapa uji coba terkait dengan algoritma yang digunakan. Penulis juga melakukan studi literatur untuk mengetahui permasalahan yang ada. Pada penelitian ini, kami memilih implementasi teori graf pada game strategi RPG yang kami buat dengan algoritma yang digunakan yaitu Algoritma Dijkstra. Dalam penelitian ini, penulis melakukan berbagai tahapan seperti pembuatan konsep, desain tampilan dan komponen, pembuatan game, pengujian, analisis, dan hasil. Hasil dari penelitian ini adalah sebuah game berbasis web yang dibuat dengan menggunakan Javascript Framework p5 yang mengimplementasikan algoritma Dijkstra sehingga dapat menentukan rute terpendek menuju target yang telah ditentukan.

Kata Kunci: *Teori Graf, RPG Game, Algoritma Dijkstra*

1. PENDAHULUAN

Game RPG merupakan sebuah game dimana pemain menjalankan sebuah peran dalam sebuah game untuk dapat menyelesaikan sebuah *Quest* yang disusun sedemikian rupa dalam sebuah cerita tertentu. Pemain memerankan sebuah peran yang biasanya berupa seorang pahlawan atau hero yang secara umum tujuannya adalah mengalahkan penjahat atau menyelesaikan permasalahan tertentu. Sebuah *game RPG* memiliki berbagai macam jenis salah satu yang populer dikalangan *game RPG* tersebut adalah game dengan *genre Strategy*. Game yang memiliki *genre* ini biasanya menggunakan sebuah papan dimana pemain harus menggerakan bidak-bidak yang dimiliki untuk mengalahkan bidak yang dimiliki oleh pemain lawan sama seperti hal nya ketika kita memainkan game seperti catur. Terdapat 2 jenis game *strategy* yaitu *Turn-Based* dan juga *Real-Time*. *Game* yang memiliki tipe *Turn-Based* biasanya memiliki giliran permainan yang bergantian sehingga pemain pertama menjalankan gilirannya terlebih dahulu lalu pemain musuh dalam kasus ini merupakan *bot* menjalankan gilirannya lalu setelah *bot* menjalankan gilirannya pemain menjalankan gilirannya kembali begitu seterusnya hingga terjadi sebuah ending dimana pemain menang, kalah atau seri dengan *bot* musuh.

Dalam implementasi permainan seperti game *RPG* dalam bentuk *Turn-Based Strategy* ini biasanya dapat kita representasikan sebagai graf 2 dimensi yang bersifat tidak-berarah yang memiliki berat atau *weight* disetiap sisinya. Untuk menciptakan sebuah *bot* yang mampu bergerak sesuai dengan pergerakan pemain, *bot* harus bisa mengetahui jarak terdekat dari posisi pemain untuk dapat menentukan pergerakannya [8]. Untuk mengetahui jarak terdekat yang bisa dilalui oleh *bot* dalam sebuah path yang berada dalam environment yang disini dapat kita sebut sebagai 2d grid, kita perlu membuat sebuah algoritma yang mampu menentukan jarak terdekat yang mampu mengetahui posisi bidak pemain dengan cepat dan memiliki efisiensi yang tinggi. Dengan mengimplementasikan salah satu cabang matematika diskrit yaitu teori graf kita bisa membuat sebuah algoritma tersebut dengan efisien dan baik. Algoritma yang kita gunakan dalam implementasi pencarian jarak terdekat dalam *game RPG Strategy* ini adalah *Dijkstra Algorithm* yang terbukti bisa menentukan jarak tercepat dari 2 titik dalam sebuah *node* yang berada dalam *graph* yang memiliki *weight* atau berat yang membedakan jarak antara node satu dengan yang lainnya. Dalam pembuatan

sebuah game rpg ini, penulis menggunakan sebuah *framework javascript* yang bernama P5 yang terbukti memiliki boilerplate yang baik untuk membuat sebuah game berbasis *website*.

Keunikan dari penelitian yang kami buat adalah kami menggunakan sebuah algoritma Dijkstra untuk diimplementasikan pada sebuah game *RPG* Strategi dalam sebuah *framework javascript p5* yang merupakan sebuah framework yang cukup kurang dikenal namun sebenarnya *framework* ini memiliki berbagai macam tool yang *powerfull* untuk membuat sebuah game berbasis *website* [5]. Penelitian ini akan menguji berbagai macam jenis kemungkinan yang bisa dilakukan dalam implementasi tahap awal pembuatan game strategi *RPG* ini dengan membandingkan berbagai macam *variable* yang akan digunakan dalam sebuah proses *development* sebuah game *RPG* Strategi seperti posisi musuh adanya wall atau tembok yang menghalangi *bot* untuk dapat mencari posisi pemain, lalu adanya weight node yang memberikan sebuah beban bagi bot untuk menuju posisi pemain. *Weight node* atau *node* yang memiliki beban merupakan sebuah node yang memiliki nilai lebih atau dengan kata lain memiliki jarak yang lebih besar dari pada node yang tidak memiliki *weight*. Sehingga algortima harus beradaptasi dengan baik ketika menentukan path terdekat yang harus dilalui untuk menuju posisi pemain.

Selanjutnya juga terdapat sebuah implementasi *bomb node* yang merupakan *nod*e yang harus dilalui terlebih dahulu sebelum bisa menuju posisi pemain. Implementasi dari *node* ini dalam sebuah game adalah ketika didalam sebuah permainan *bot* harus mempertimbangkan posisi dari objek lain yang memiliki *value* misalnya saja dalam sebuah *game* dengan judul *Wargroove* dimana terdapat sebuah sistem building yang apabila pemain atau bot mendekati building tersebut lalu melakukan *"conquer"* kepada building tersebut, fungsi yang dilakukan dari building tersebut akan menjadi pemilik kelompok yang mendapatkannya. Penulis akan membandingkan bagaimana performa algoritma ini untuk menyelesaikan permasalahan tersebut dan menganalisis lebih lanjut permasalahan ini.

2. METODE PENELITIAN

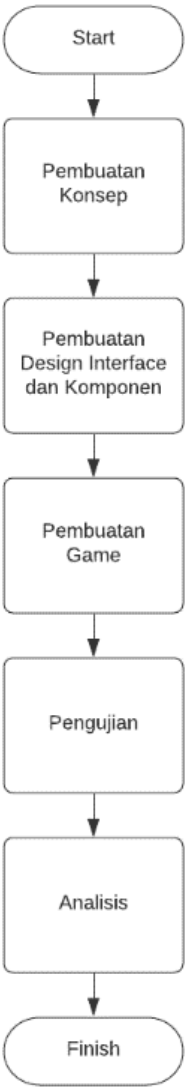
Jenis pendekatan yang peneliti gunakan, subjek penelitian, *variable* penelitian, proses pembuatan sistem, implementasi algoritma dijkstra dan teknik pengumpulan data yang digunakan untuk kemudian diteliti lebih lanjut untuk selanjutnya dianalisis untuk ditarik kesimpulan lebih lanjut.

Pendekatan yang penulis gunakan dalam penelitian ini adalah penelitian kuantitatif eksperimen dimana eksperimen yang dilakukan memiliki sekurang-kurangnya satu variable bebas yang disebut sebagai variable eksperimental yang nilai nya dapat diubah-ubah berdasarkan keinginan peneliti [4]. Penulis menggunakan metode penelitian ini karena semua jenis data yang digunakan dalam penelitian ini menggunakan data dengan jenis data yang bertipe angka sehingga harus menggunakan metode pendekatan kuantitatif dan penulis akan mengganti *variable* yang digunakan untuk menguji algoritma yang diimplementasikan dalam penelitian ini.

Subjek penelitian ini adalah game yang tengah penulis coba buat dengan menggunakan *framework javascript p5* berbasis *web* yang merupakan game yang akan memiliki *genre Turn-Based SRPG*. Penulis akan mencoba menganalisis waktu yang digunakan oleh algoritma dijkstra untuk dapat mencari path atau jalur terdekat untuk dapat mencapai node target dari posisi tertentu dari berbagai macam posisi dan berbagai macam kondisi tertentu. Terdapat beberapa variable yang digunakan yaitu posisi node awal, lalu algoritma yang digunakan untuk mencari rute terdekat, tipe *node* pada game yang memiliki beberapa jenis yaitu *wall node*, *weight node* dan bomb *node*. Selanjutnya adalah kompleksitas dari *wall* yang dibuat pada *game*.

Proses pembuatan sebuah sistem ini memiliki proses yang terdiri dari pembuatan konsep, pembuatan desain user *interface* dari game, pengumpulan materi, pembuatan, pengujian dan analisis.

Pada proses pembuatan konsep peneliti menentukan konsep yang digunakan untuk membuat game untuk kemudian dikembangkan lebih lanjut untuk proses selanjutnya. Konsep yang telah ditentukan oleh penulis adalah sebagai berikut sebuah game berbasis 2D dalam bentuk grid dimana peneliti dapat mengganti berbagai macam variable didalamnya lalu mengambil data berdasarkan hasil yang telah dibuat oleh sistem.



Gambar 1. *Flowchart* Metodologi Penelitian

Pada proses pembuatan desain penulis membuat tampilan *user-interface* untuk kemudian dibuat tampilannya dengan menggunakan HTML dan CSS yang merupakan sebuah markup code yang digunakan untuk membuat kerangka dari tampilan *website*. Penulis juga membuat berbagai macam tampilan animasi yang akan ditampilkan ketika algoritma mencari posisi dari pemain. Selain itu penulis juga membuat desain tampilan *sprite* yang digunakan untuk merepresentasikan *node* pada *game*.

Pada proses pembuatan game terdapat berbagai macam langkah-langkah yang dilakukan seperti pembuatan *markup code* dari desain yang sudah dibuat, pembuatan

basic position untuk bisa menempatkan *node* pada grid yang telah dibuat, pembuatan *backend* pada aplikasi untuk bisa menerapkan berbagai macam algoritma dalam *game*, proses *debugging* dimana peneliti menyelesaikan berbagai macam permasalahan yang ada ketika game sudah selesai dibuat dan terakhir pembuatan *guide* langkah-langkah bermain *game*.

Pada proses pengujian dan analisis kami menguji berbagai fitur yang ada pada *game* untuk kemudian dapat diteliti sedemikian rupa sehingga peneliti dapat melakukan anlisis berdasarkan data yang diambil pada aplikasi.

3. HASIL DAN PEMBAHASAN

Hasil dari penelitian yang kami buat adalah sebuah *prototype* game yang akan dikembangkan lebih lanjut. Berikut adalah tampilan *user interface* dan *sprite* yang nantinya akan digunakan dalam pengembangan game lebih lanjut.

Gambar 2. Tampilan *User Interface Prototype Game Strategi RPG*

Pada tampilan *user interface* tersebut terdapat berbagai macam komponen yang ada diantaranya *Navbar* pada bagian atas yang berfungsi mengatur tampilan dari grid yang pada tahap *prototype* ini akan berfungsi sebagai *map* yang digunakan untuk memposisikan bidak pemain dan musuh dalam game *strategy RPG*. Dalam tampilan game terdapat panduan bagaimana menggunakan *prototype* untuk melihat implementasi algoritma Dijkstra pada *game rpg* untuk menentukan rute terdekat yang harus dilalui untuk mengetahui posisi *node* target pada graf. Terdapat beberapa *sprite* yang kami gunakan dalam game tersebut diantaranya adalah:

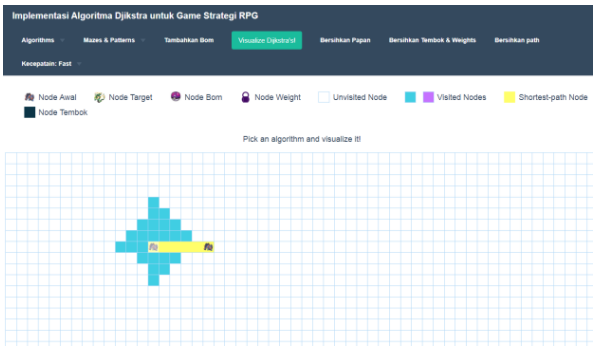
No	Gambar Sprite	Nama Sprite	Keterangan
1		Node Awal	Merupakan node yang merepresentasikan bot pada permainan yang akan mencari posisi node target yang merupakan bidak yang dimiliki oleh pemain.
2		Node Target	Merupakan node yang merepresentasikan posisi pemain dalam permainan dimana node awal akan mencoba mengetahui jarak terdekat yang harus dilalui untuk bisa mencapai node ini untuk kemudian menentukan keputusan dari kecerdasan buatan yang dimiliki oleh bot.
3		Node Bomb	Merupakan node yang merepresentasikan sebuah building atau objective tertentu yang harus dilalui terlebih dahulu sebelum mencapai node akhir.
4		Node Weight	Merepresentasikan sebuah node yang memiliki beban "lebih" untuk dilalui dalam sebuah game misalnya sebuah bush atau air dalam sebuah peta didalam game yang mana node tersebut bisa dilalui namun memiliki jarak yang cukup panjang sehingga perlu diperhatikan dalam perhitungan algoritma.
5		Node Tembok	Merepresentasikan tembok atau sebuah penghalang yang lainnya dalam sebuah game yang tidak dapat dilalui oleh pemain.
6		Unvisited Node	Merupakan representasi node yang belum dilewati oleh node awal atau dengan kata lain node ini belum di jelajahi dan memiliki jarak tak terhingga jika kita mengacu pada algoritma djikstra. Untuk implementasi game strategy RPG biasanya node ini merupakan sebuah node yang tidak dapat dikunjungi karena node tersebut memiliki jarak yang cukup jauh dan node awal tidak memiliki cukup stamina point untuk dapat mengunjunginya.
7		Visited Node	Merupakan representasi node yang telah dikunjungi ketika algoritma dijkstra dijalankan. Biasanya dalam sebuah game RPG node ini merepresentasikan jalur yang dapat dilalui oleh sebuah node berdasarkan stamina point tertentu yang ditentukan.
8		Path Node	Merupakan representasi path yang dilalui oleh graph untuk bisa menuju posisi yang dinginkan yang mana adalah jalur terdekat menuju pemain.

Gambar 3. *Sprite* yang digunakan dalam game

Terdapat berbagai macam hasil yang dapat kami ambil datanya dari implementasi algoritma untuk mencari jarak terdekat pada sebuah graf pada game yang kami buat. Untuk percobaan pertama dimana penulis mencoba melakukan pengambilan data tentang jarak node terdekat serta keterkaitannya dengan posisi *node start* dan *node finish* dengan jumlah *step* yang harus dilakukan untuk melakukan *checking* setiap *neighbour node* dapat direpresentasikan dalam tabel berikut. Untuk pengujian pertama penulis tidak menambahkan *wall* node ataupun weight node pada grid untuk menguji *worst case scenario* dari algoritma dijkstra untuk diimplementasikan pada game dengan sistem yang berbentuk 2D grid.

Pada tabel berikut posisi dari node direpresentasikan dengan 2 angka yang dipisahkan oleh tanda , (koma) untuk membedakan posisi x dan y node pada graph.

Kolom jarak menunjukan jarak terdekat dari path yang harus dilalui untuk menuju posisi pemain untuk pengujian ini perpindahan satu *node* ke *node* yang lain dihitung 1 dan perputaran 900 juga memiliki bobot 1. Untuk variable pada kolom waktu menunjukan waktu yang diperlukan oleh algoritma untuk mencari jarak terdekat. Perlu diingat jika kami memperlambat kecepatan dari proses pencarian untuk keperluan visualisasi sehingga setiap kali node melakukan pengecekan diperlukan waktu sebanyak 10ms untuk mengecek salah satu *node* dan membandingkannya dengan *node* yang lain untuk mencari jarak terdekat.

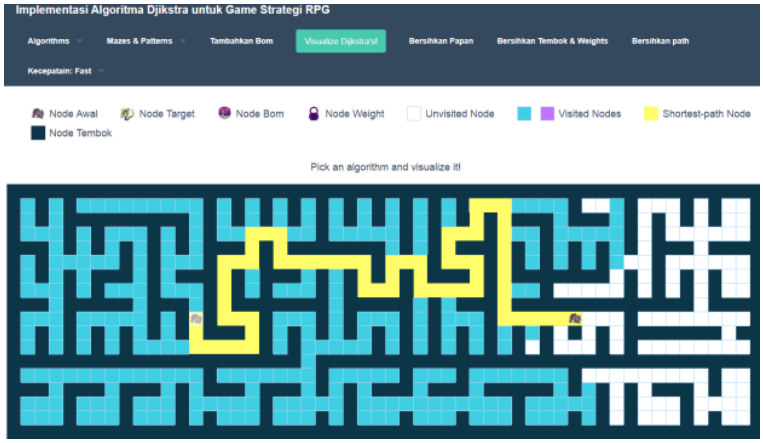


Gambar 4. Proses pengujian visualisasi Algoritma Dijkstra tanpa menggunakan *wall* ataupun *weight*

Tabel 1.
Data Terkait Algoritma Dijkstra Pada Percobaan Tanpa Menggunakan *Wall* Ataupun *Weight*

No	Posisi Awal	Posisi akhir	Jarak Node	Waktu(s)	Jumlah Step
1	14,10	41,10	26	6.40	640
2	14,10	34,4	26	6.9	648
3	14,10	48,16	39	8.2	820
4	14,10	52,1	44	11.45	1450
5	14,10	48,19	44	11.45	1450
6	14,10	6,14	14	3.45	396
7	14,10	18,8	8	0.60	65
8	14,10	36,3	30	11.31	1150
9	14,10	1,2	22	8.27	840
10	14,10	30,18	25	9.69	980
11	1,1	52,19	72	17.02	1950

Dari tabel diatas pada baris nomor 11 merupakan *worst case scenario* pada algoritma dijkstra dimana algoritma harus mengecek semua *node* yang ada pada grid atau papan. Selanjutnya dengan menggunakan prosedur yang sama kami mencoba melakukan pengambilan data lebih lanjut dengan mengimplementasikan adanya tembok pada grid yang kami *generate* menggunakan *Recursive Division* yang merupakan sebuah algoritma yang digunakan untuk membuat sebuah maze untuk pengujian ini [7]. Cara kerja pengujian adalah penulis akan melakukan generate sebuah maze lalu melakukan sebuah pencarian dimana setiap pengujian yang berbeda penulis mengganti bentuk *maze* yang digunakan. Pada pengujian ini penulis tidak mengubah posisi awal dan posisi akhir dari node awal dan node akhir namun hanya mengubah bentuk dari *maze* yang digunakan atau dengan kata lain menghapus papan dan melakukan *generate* kembali *maze* dengan algoritma *recursive division* yang akan menghasilkan bentuk *maze* yang berbeda [3].



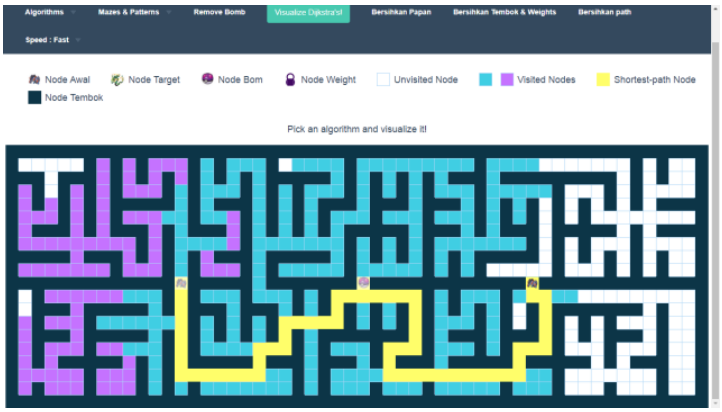
Gambar 5. Proses pengujian Algoritma Dijkstra dengan mengimplementasikan *wall* dalam bentuk *maze*

Tabel 2.
Data terkait implementasi algoritma dijkstra pada *maze* yang memiliki *wall*

No	% wall	Jarak	waktu
1	40	72	6.79
2	44	97	5.59
3	38	53	4.67
4	35	64	3.56
5	45	86	5.39
6	38	66	5.06
7	47	80	6.54
8	35	54	5.15

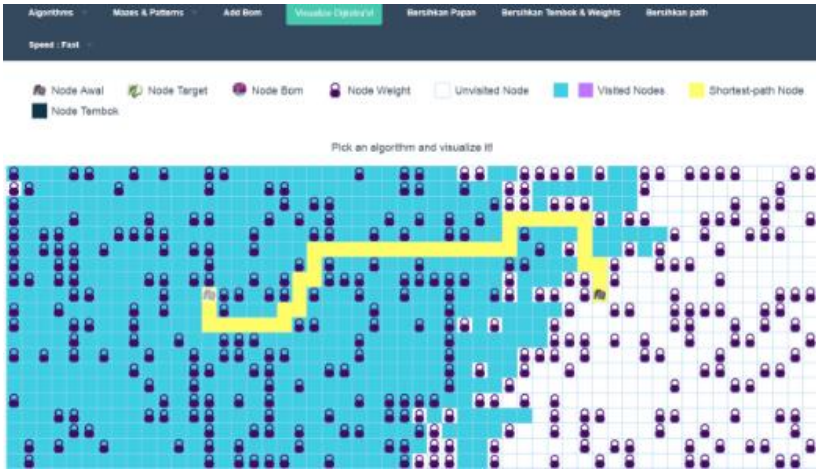
No	% wall	Jarak	waktu
9	38	62	6.11
10	45	70	4.07

Metode selanjutnya adalah percobaan penggunaan *node* bom yang merupakan sebuah *node* yang harus dilalui terlebih dahulu sebelum menuju *node* tujuan. Kami melakukan percobaan hingga 20 kali dan disetiap percobaan, algoritma dijkstra mampu menemukan rute terdekat untuk bisa menjelajahi *path* yang menghubungkan *node* awal, node *bomb* dan juga *node* akhir.



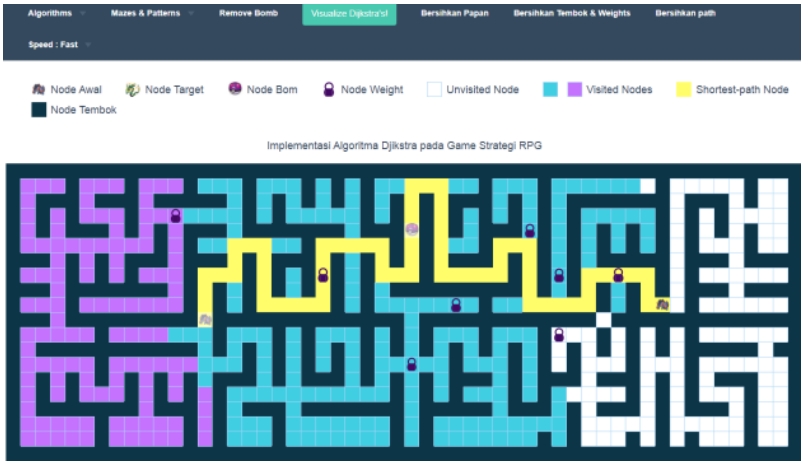
Gambar 6. Proses pengujian Algoritma Dijkstra dengan mengimplementasikan *wall* dalam bentuk *maze*

Pada tahap selanjutnya kami mencoba untuk mengimplementasikan penggunaan algoritma ini untuk diimplementasikan pada sebuah grid yang memiliki *weight node* yang merupakan sebuah *node* yang memiliki berat atau beban lebih dari pada *node* yang lain. *Node* ini bisa dilewati berbeda dengan node weight yang tidak bisa dilewati oleh path namun untuk melewatinya memerlukan beban yang lebih yaitu 15 dari pada *node* biasa yang memiliki beban 1. Dengan menggunakan algoritma *maze Recursive Division* kami mencoba membuat sebuah maze yang memiliki *weight* didalamnya. Pada pengujian ini algoritma dijkstra mampu mencari path terdekat dengan kecepatan yang cukup baik tanpa adanya permasalahan.



Gambar 7. Pengujian penambahan *Weight node* pada graf

Pengujian yang terakhir adalah mencoba menggabungkan antara semua jenis pengujian sebelumnya yaitu penambahan *wall node*, *weight node*, *bomb node* serta implementasi *Recursive Division algorithm* untuk membuat *maze*. Pengujian ini berguna untuk memastikan algoritma dijkstra mampu mencari posisi terdekat dari bidak milik pemain dengan environtment yang biasanya memiliki berbagai macam tembok, building dan sebagainya. Melalui pengujian ini kita bisa menentukan apakah algoritma dijkstra layak digunakan sebagai pathfinding untuk diimplementasikan pada *game Turn-Based Strategy RPG.*



Gambar 8. Pengujian kombinasi antara semua node yaitu node awal, node target, node bom, node weight

Berdasarkan data diatas yang merupakan implementasi algoritma Dijkstra pada *Game RPG Strategy* dengan sistem *Turn-Based* telah terbukti bahwa algoritma tersebut dapat menentukan dengan baik jarak terdekat antara 2 *node* yang ada pada graf.

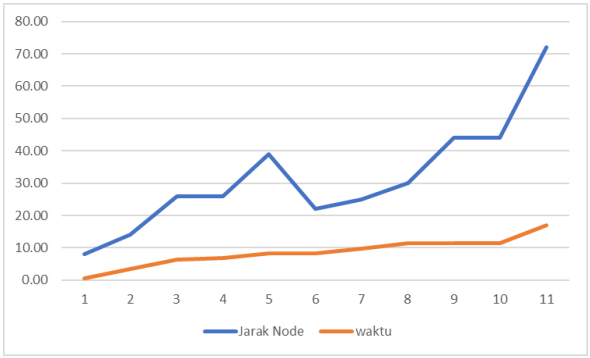
Melalui 5 pengujian diantaranya pengujian kemampuan algoritma untuk mencari rute terdekat ketika terjadi perpindahan posisi node awal dan posisi *node* akhir. Berikut adalah implementasi code javascript untuk menjalankan algoritma Dijkstra pada prototype ini.

```
// function that returns the minimum cost and path to reach Finish
const dijkstra = (graph) => {

  // track lowest cost to reach each node
  const costs = Object.assign({finish: Infinity}, graph.start);

  // track paths
  const parents = {finish: null};
  for (let child in graph.start) {
    parents[child] = 'start';
  }

  // track nodes that have already been processed
  const processed = [];

  let node = lowestCostNode(costs, processed);

  while (node) {
    let cost = costs[node];
    let children = graph[node];
    for (let n in children) {
      let newCost = cost + children[n];
      if (!costs[n]) {
        costs[n] = newCost;
        parents[n] = node;
      }
      if (costs[n] > newCost) {
        costs[n] = newCost;
        parents[n] = node;
      }
    }
    processed.push(node);
    node = lowestCostNode(costs, processed);
  }

  let optimalPath = ['finish'];
  let parent = parents.finish;
  while (parent) {
    optimalPath.push(parent);
    parent = parents[parent];
  }
  optimalPath.reverse();

  const results = {
    distance: costs.finish,
    path: optimalPath
  };

  return results;
};
```

Gambar 9. Implementasi algoritma

Implementasi algoritma *Recursive Division* pada graf atau grid yang pada hasil akhirnya memasang *node* tembok yang menghalangi path menuju node akhir, implementasi *node bomb* yang merupakan *node* yang harus dikunjungi terlebih dahulu sebelum bisa menuju *node* tujuan atau node akhir, implementasi adanya *node* yang memiliki *weight*, implementasi gabungan antara pemasangan dengan menggunakan algoritma *recursive division*, pemasangan *node wall* dan *node bomb*. Melalui berbagai macam tes implementasi ini kita mendapatkan hasil yang cukup baik. Seperti yang dapat

kita lihat pada *nodewall* dari grafik dibawah ini yang menampilkan perbandingan jarak dan waktu yang diperlukan oleh algoritma Dijkstra untuk menentukan path menuju node akhir pada graf.



Gambar 10. Grafik perbandingan jarak node dan waktu implementasi algoritma dijkstra pada graf tanpa *node wall*

Melalui grafik diatas kita bisa memastikan kompleksitas dari algoritma dijsktra untuk diimplementasikan dalam sebuah game strategi ini (Ramadhan & Udjulawa, 2020). Meskipun ketika dalam kondisi worse case scenario algoritma ini cukup memakan waktu yang cukup cepat dan menjamin ditemukannya jalur tercepat [1]. Yang diperlukan dalam pengembangan sebuah game berbasis 2D Grid untuk kemudian diimplementasikan kepada kecerdasan buatan untuk bot pada game tersebut jika dibandingkan dengan algoritma A* (*A star*) [2][6].

4. KESIMPULAN

Algoritma Dijkstra merupakan algoritma yang baik untuk diimplementasikan pada sebuah permasalahan didalam sebuah game dimana sebuah *computer* harus mencari posisi terdekat dari pemain. Melalui prototype game strategy rpg ini kita bisa menyimpulkan bahwa penggunaan algoritma dijkstra dapat diimplementasikan pada berbagai jenis scenario yang mungkin muncul pada sebuah game strategy rpg berbasis grid 2 dimensi seperti mencari rute tercepat pada node yang memiliki berat, mencari rute apabila terdapat halangan didalam node, mencari rute terdekat dari 2 node yang memiliki prioritas yang berbeda, menentukan jalur yang harus dilalui apabila bidak yang digunakan oleh *computer* memiliki batas jarak tertentu yang bisa dilalui. Penggunaan framework javascript p5 juga cukup membantu dalam pembuatan visualisasi animasi serta implementasi algoritma Dijkstra dari pada penggunaan native javascript.

5. SARAN

Meskipun masih dalam bentuk prototype, game ini masih bisa menjadi gambaran tentang implementasi algoritma dijkstra yang kemudian bisa dikembangkan kembali menjadi sebuah kecerdasan buatan yang mampu menentukan keputusan dari *computer* untuk dapat menyesuaikan pergerakan dari pemain.

DAFTAR PUSTAKA

[1] A. D. Hartanto, A. S. Mandala, D. R. P.L., S. Aminudin, and A. Yudirianto, "Implementasi Algoritma Dijkstra Pada Game Pacman," *CCIT Journal*, vol. 12, no. 2, pp. 170–176, 2019.

[2] A. W. Ramadhan and D. Udjulawa, "Perbandingan Algoritma Dijkstra dan Algoritma A Star pada permainan Pac-Man," *Jurnal Algoritme*, vol. 1, no. 1, pp. 12–20, 2020.

[3] C. Adams, H. Parekh, and S. J. Louis, "Procedural level design using an interactive cellular automata genetic algorithm," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017.

[4] Emzir, *Analisis data: metodologi penelitian kualitatif*. Jakarta: Rajawali Pers, 2010.

[5] H. Bohnacker, B. Gross, J. Laub, C. Lazzeroni, and M. Frohling, *Generative design: visualize, program, and create with JavaScript in p5.js*. New York: Princeton Architectural Press, 2018.

[6] M. A. Djojo and K. Karyono, "Pengukuran Beban Komputasi Algoritma Dijkstra, A*, dan Floyd-Warshall pada Perangkat Android," *Jurnal ULTIMA Computing*, vol. 5, no. 1, pp. 13–17, 2013.

[7] P. H. Kim, J. Grove, S. Wurster, and R. Crawfis, "Design-centric maze generation," *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019.

[8] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, "Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm," *Experimental Algorithms*, pp. 303–318.