REPOSITOR, Vol. 2, No. 3, Maret 2020, Pp. 373-384
ISSN : 2714-7975
E-ISSN : 2716-1382

Algoritma Maze Generator Recursive Backtracking Untuk Membuat Prosedural Labirin Pada Game Petualangan Labirin 3D

Elbert Setiadharma*[1], Lailatul Husniah[2], Ali Sofyan Kholimi[3]
[1,2,3]Teknik Informatika/Universitas Muhammadiyah Malang
elbert_437128@webmail.umm.ac.id*

Abstrak

Perkembangan game diranah industri berkembang pesat, banyak inovasi yang telah diciptakan agar dapat memaksimalkan kinerja sistem yang mulai kian beragam bersamaan dengan banyak hal yang ditawarkan. melihat perkembangan ini banyak developer pintar memilah metode-metode yang diperlukan agar proses pembuatan game tidak terhambat. salah satu komponen dalam game yang perlu diperhatikan lebih adalah environment, dimana environment merupakan unsur game yang sangat penting. kebanyakan developer merasa kesulitan dalam menciptakan environment yang beragam namun dalam waktu yang singkat. salah satu cara menyiasati adalah dengan cara menerapkan metode procedural content Generator. konteks environment yang mudah dalam penerapan untuk skala prototyping adalah labirin, dimana segala unsur komponen game didalamnya dapat ditentukan sesuai variabel yang telah dirancang oleh developer game. metode algoritma penerapan untuk PCG environment beragam terutama pada bidang labirin, salah satunya algoritmanya adalah Recursive Backtracking, dimana algoritma ini banyak digunakan demi mendapatkan environment yang sesuai dengan kehendak developer game serta banyak memiliki keuntungan dibandingkan dengan algoritma PCG dalam bidang labirin lainnya. berbekal penelitian sebelumnya yang mengatakan bahwa algoritma Recursive Backtracking merupakan algoritma yang tepat untuk mengolah konten labirin secara prosedural, maka penelitian ini akan dilakukan untuk membuktikan apakah algoritma Backtracking dalam pembuatan PCG untuk bidang labirin telah sesuai dengan pernyataan penelitian sebelumnya.

Kata kunci: *PCG, Recursive Backtracking, Labirin, Game*

Abstract

Video game on industries scale were growing fast, many innovation behind that been created to get optimized system behavior. Watch this phenomenon, many game developer try get the finest method to achieve the best way on game development. One of the important thing that must to be cared on game development was environment. Many game developer getting trouble to made environment that had many variety but could be developed on short period of time. Solution for this problem is implement Procedural Content Generation method. Maze was one of the option that most easiest way to approach prototyping on game scale, where most of every Component video games could be placed depend on variable that developer design before. Algorithm for PCG environment had much kind, Recursive Backtracking is one of it. This Algorithm been used and much proven for its behavior for getting good result, moreover on Maze context. as its said on previous research, this Algorithm was the best option to be implemented to create Maze on procuderal way. This research been intend to prove is it Recursive Backtrack could be the best way to implement as method to getting good result moreover for Maze.

Keywords: *PCG, Recursive Backtracking, Labirin, Game*

1. Pendahuluan

Game merupakan salah satu esensi dari perkembangan teknologi, mulai dari rupa yang sederhana dengan mekanika yang terbatas, hingga mekanika yang kompleks seperti kebanyakan game pada saat ini. Mekanika permainan menggunakan tema labirin pun tak kalah menarik, dimana pemain akan dituntut untuk memecahkan teka teki sedari mulai garis Start hingga garis finish. Secara definisi labirin merupakan puzzle untuk mencari jalan keluar dimana selama dalam perjalanan menelusuri labirin pemain akan mendapat banyak rintangan/halangan untuk sampai pada tujuan [1].

Perkembangan game ranah industri hiburan telah berkembang secara pesat dalam kurun waktu terakhir, banyak inovasi yang telah dilakukan mengenai konten yang dipakai, hingga metode baru yang diciptakan agar dapat memaksimalkan kinerja sistem permainan yang mulai beranjak semakin kompleks. Perkembangan yang pesat ini menguntungkan dari segi performa sistem permainan, namun berpengaruh banyak dalam konteks konten asset yang dipakai, hal ini akan berdampak pada waktu yang dibutuhkan untuk implementasi, serta dapat berakhir dengan biaya mahal. Solusi untuk hal ini dapat diatasi dengan penggunaan metode PCG (procedural generated content), PCG merupakan metode untuk mendapatkan konten secara otomatis yang diatur langsung oleh sistem. Terutama video games pada komputer, proses development yang dilakukan oleh developer tidak memerlukan semua penataan konten/asset secara manual [2].

Berdasarkan penelitian The Quest for the Perfect Perfect-Maze pada jurnal konferensi untuk game komputer, dalam penelitian tersebut terdapat penjebaran akan struktur labirin yang sempurna. Poin poin tersebut telah dianalisa dan memiliki kesimpulan bahwa sebuah labirin harus terdapat komponen penting didalamnya, seperti Cell, Start Point, EndPoint, Solution Path, Decision Cell Junction, Decision Cell Cross Road, Dead EndAlcove, Forward Dead End, dan Backward Dead End [3]. PCG untuk Maze itu sendiri memiliki beberapa pilihan algoritma yang dapat digunakan, diantaranya Prim's Algorithm, Kruskal's Algorithm, Eller's Algorithm, Sidewinder Algorithm, Recursive Division Algorithm, Binary Tree Algorithm, Growing Tree Algorithm, Recursive Backtracker Algorithm, dan Hunt and Kill Algorithm [4]. Berdasarkan aspek yang dibahas sebelumnya mengenai pemilihan algoritma untuk generated content pada Maze, algoritma Recursive Backtrack memiliki kelebihan diantara algoritma lain [4]. Algoritma Recursive Backtracking bila dibandingkan dengan proses implementasi algoritma lain, recursive backtraking merupakan algoritma generated Maze yang paling mudah untuk diimplementasikan [5]. Secara definisi Algoritma Recursive Backtracking merupakan pengembangan dari metode DFS. Algoritma DFS adalah metode untuk mencari solusi dari akar ke daun (dalam pohon ruang solusi) dengan pencarian mendalam, Simpul-simpul yang sudah diperiksa dinamakan simpul hidup (Node) sedangkan simpul yang diperluas dinamakan simpul Node [6].

Pembeda antara algoritma DFS dan algoritma Recursive Backtracking ialah, ketika proses yang dilakukan oleh algoritma DFS menggunakan proses iterative/perulangan setiap kali runut balik dilakukan, sedangkan proses Recursive Backtracking lebih kepada penggantian proses secara recursive. Hal tersebut tentunya akan mempengaruhi proses ketika berjalan dimana waktu yang dibutuhkan akan lebih singkat. Kedua algoritma bekerja secara mirip, namun masih ada sedikit pembeda diantara keduanya [7].

2. Metode Penelitian

Implementasi yang dilakukan, beralaskan peraturan pada game sebelumnya yang memiliki rules permainan sederhana seperti game labirin pada umumnya dimana pemain akan menelusuri labirin mulai dari titik awal hingga titik akhir. Algoritma yang digunakan akan mengacu pada referensi penelitian sebelumnya, algoritma Recursive Backtracking ini pun harus mengikuti syarat kategori labirin sempurna, dimana labirin harus :

1. Memiliki titik awal serta titik akhir.
2. Memiliki minimal satu jalur yang terhubung mula dari titik awal hingga titik akhir.
3. Labirin memiliki pseudo generated Seed, dimana labirin dapat diciptakan ke banyak pola namun dapat kembali ke pola sebelumnya dengan menambahkan value Seed yang diberikan saat labirin dibuat.
4. Memiliki parameter labirin sempurna didalam komponennya.

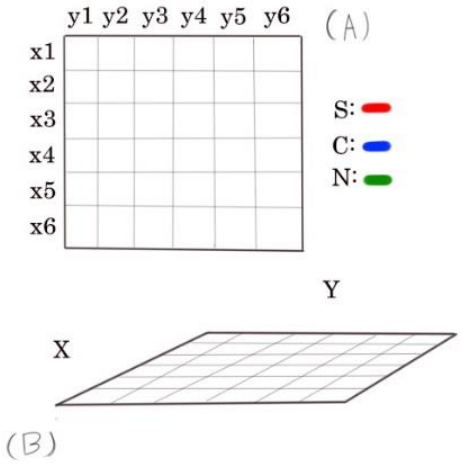
Labirin yang akan dibuat akan direpresentasikan ke dalam ukuran yang telah dibuat sebelumnya, hal ini bertujuan agar hasil pengujian dapat terlihat lebih akurat. Ukuran labirin yang ditetapkan adalah 20*20.

3. Analisa Metode Recursive Backtracking

Algoritma Recursive Backtracking bekerja ketika inisiasi variable ukuran labirin berdasarkan koordinat Size X serta Size Y diciptakan, besarnya nilai kedua variable X serta Y ini akan menjadi faktor penentu kerumitan/ kompleksitas komponen dari labirin itu sendiri. Dikarenakan environment pada permainan yang diinginkan mencakup lingkup 3D, maka variable tambahan seperti wall (Z) serta passsage (Z) akan diciptakan seiringan Node pada plain kosong labirin terbentuk. Visual proses pembentukan sebuah labirin akan dijabarkan satu persatu di

bawah ini, mulai dari proses carving Cell bactrack hingga proses carving wall direction dan passsage direction terbentuk.

Proses awal untuk membentuk labirin dimulai dengan inisiasi plain kosong dimana plain kosong ini didasarkan pada bentuk Graph/grid yang terbagi atas size yang telah ditentukan diawal. Gambar 1 berikut merupakan visual grid Nampak dari atas serta dari samping.



Gambar 1. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Inisiasi Awal Grid Berbentuk Plain Datar

Ketika grid kosong telah diinisiasi proses selanjutnya menentukan/ mengambil kolom pertama sebagai acuan titik mulai algoritma bekerja, titik awal dari implementasi labirin dapat ditentukan, akan tetapi pada proses yang dilakukan, kolom pertama grid dapat dimulai dari kolom manapun. Hal ini bekerja sesuai acuan algoritma pada proses pertama, dimana memilih kolom random sebagai titik mula serta memasukkan data koordinat kolom pertama tersebut pada data Array. Seperti pseudo algoritma yang dikutip, cara kerja algoritma Backtracking maka proses tersebut dimulai seperti [9]:

1. Memilih kolom pertama secara random dan menambahkan data koordinat kolom tersebut kedalam data Array sebagai S (Stack)
2. Cek bila S (Stack) itu memiliki tetangga, maka
   a. Pilih random salah satu kolom sebagai kolom selanjutnya.
   b. Tambahkan sel selanjutnya ke dalam Stack.
   c. Buat jalur antara kolom C(Current) hingga kolom berikutnya.
   d. Tandai kolom berikutnya sebagai kolom C(Current)
3. Bila tidak ada maka tentukan kolom dari urutan dalam S(Stack) sebagai C(Current) kolom
4. Bila kolom pada S(Stack) belum habis.
   a. Ulangi proses nomor 2

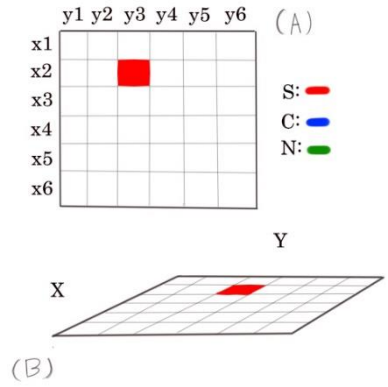
Meninjau dari proses diatas dimana ada 3 variabel penting dalam penjabaran algoritma Recursive Backtracking:

1. S akan direpresentasikan sebagai “Stack” dalam data Array.
2. C akan direpresentasikan sebagai “Current Cell” dalam grid.
3. N akan direpresentasikan sebagai “Not Visited Cell” dalam grid.

Namun dalam kebaruan penelitian ini, diperlukan penambahan variable dari variable utama yang disimpan, diantaranya :

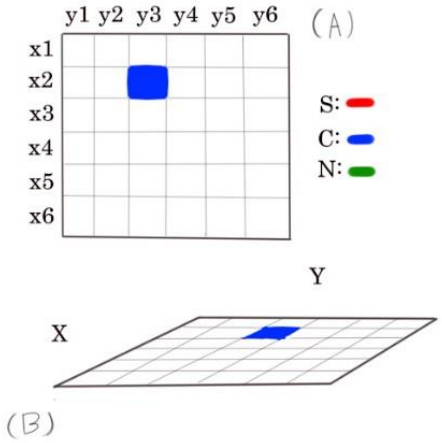
1. Direction value
2. Edges value

Kedua value diatas akan membantu jalannya algoritma utama recursive backtracking sebagai variable yang akan memberi inisiasi letak model wall serta passsage yang berguna sebagai representasi model 3D dalam pola/pattern labirin yang terbuat. Setelah inisiasi variable telah dilakukan, maka proses pembentukan labirin dimulai dari state pertama algoritma Recursive Backtracking hingga seluruh grid size yang telah diinisiasi semuanya ditelusuri.



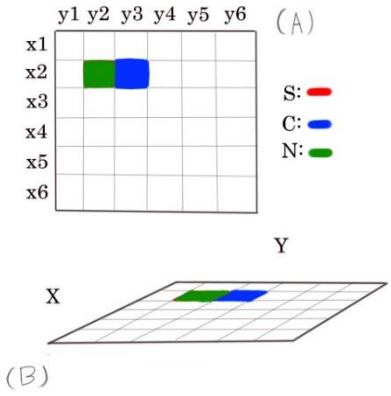
Gambar 2. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Inisiasi Pertama Node Y3, X2

Pada Gambar 2 Node Y3, X2, node didaftarkan ke dalam Node Stack. Simpan value direction dan edge dari cell coordinate X2, Y3.



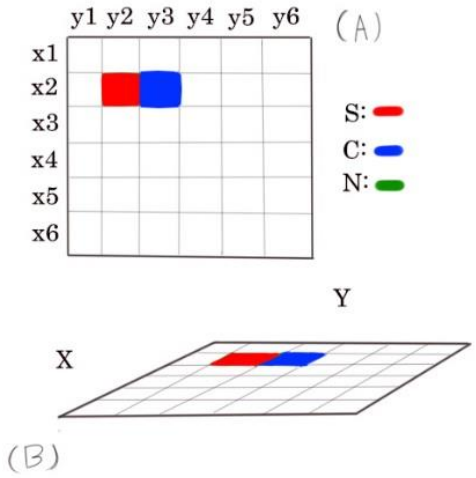
Gambar 3. (A) penampakan Orthographic, (B) Penampakan Persepctive, Node Y3, X2 Diubah Menjadi Current Cell/Node

Setelah inputan Node Y3.X2 menjadi Node yang didaftarkan dalam Node Stack, proses selanjutnya ialah mengubah Node status Node tersebut menjadi Current Node seperti pada Gambar 3.



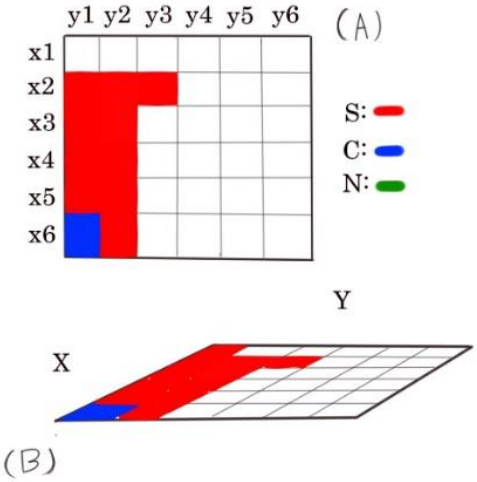
Gambar 4. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Current Cell Memeriksa Neighbor Node Yang Belum Terdaftar Dalam stack/ Not Visited

Pada Gambar 4, bila ada neighbour Node yang belum terdaftar disekeliling Current Cell maka, dari Current Cell terakhir salah satu Node akan dipilih. Dalam ilustrasi, Node yang dipilih adalah Node Y2.X2 sebagai Node yang masih berstatus not visited. Node ini dipilih berdasarkan random, namun direction value akan menentukan apakah sisi edges cell/Node Y2.X3 dapat mengunjungi Node/cell X2.Y2.



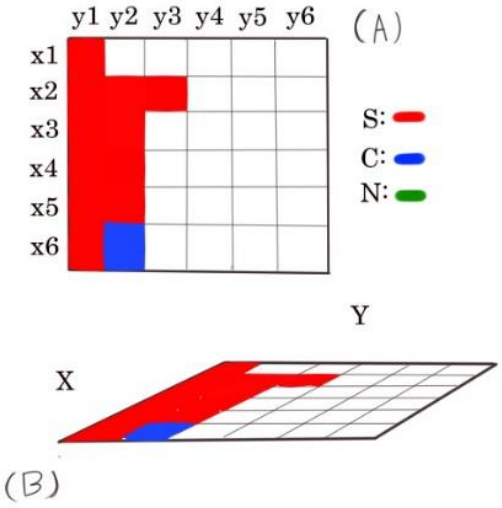
Gambar 5. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Proses Mengecek Neighbour Node serta input Node X2.Y2 ke dalam Stack Node

Setelah status Node X2.Y3 menjadi Current Node, Node tersebut akan memeriksa sekelilingnya, apakah tersedia Node yang belum ditelusuri, bila ada tetangga yang belum terdaftar dalam Node Stack, maka Node tersebut akan didaftarkan ke dalam Node Stack, dimana pada Gambar 5 Node X2.Y2 ditambahkan. Direction value serta edges value X2.Y2 disimpan. Proses diulang hingga keseluruhan node dalam graph tidak menemui tetangga.



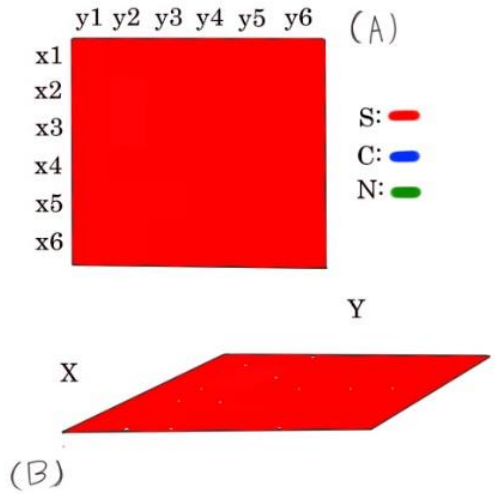
Gambar 6. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Proses Dilakukan Berulang Hingga Current Cell Tidak Menemukan Neighbour Node

Pada Gambar 6, ketika Current Cell tidak berhasil menemukan neighbour yang berstatus not visited, seperti pada illustrasi. Node X6,Y1 tidak memiliki tetangga yang belum terdaftar dalam Stack. Direction value serta edges value berakhir, dimana keduanya memiliki value neighbour == null, ketika hal tersebut terjadi maka proses pencarian Node tetangga untuk Node X6,Y1 berhenti. Edges dari direction value terakhir akan dianggap sebagai wall.



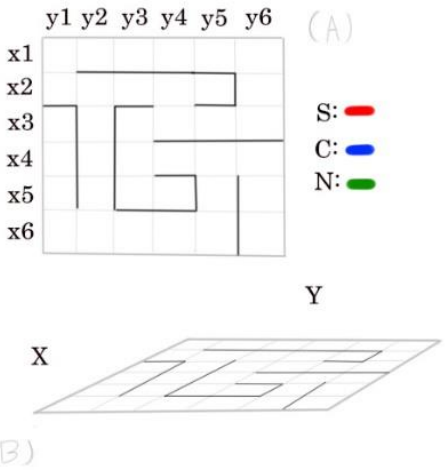
Gambar 7. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Sesuai Urutan Dalam Stack, Node Akan Ditelusuri, Hingga Current Node Memiliki Not Visited Neighbour Node

Proses dipersingkat dalam illustrasi seperti pada Gambar 7, namun proses Backtracking berarti ketika Node yang menjadi Current Cell tidak memiliki tetangga yang belum terdaftar dalam Stack, urutan dalam illustrasi ini Node X6.Y2 merupakan Node dalam Stack yang berpotensi memiliki tetangga, runut balik dilakukan setelah Node dalam illustrasi menginjak pada titik X3.Y1, sedangkan arah runut dari X2,Y2 telah memiliki rute X2,Y1 serta X1,Y1, proses Backtracking bisa saja dimulai dari Node X1,Y1 akan tetapi untuk melihat bagaimana proses Backtracking bekerja, Current Node pada X3,Y1 diilustrasikan sebagai Node terakhir yang tidak memiliki tetangga, sehingga Node yang telah terdaftar dalam Stack di runut kembali seseuai urutan, hingga menemukan Node X6,Y2 sebagai Node yang memiliki tetangga. Node X6, Y2 berganti status dari Stack Node menjadi Current Node. Setelah menemukan Node/cell yang berpotensi memiliki tetangga, value edge serta direction Node tersebut akan berubah, maksud dari perubahan tersebut adalah sisi dari tetangga yang berpotensi akan memiliki value direction serta edges, sehingga pencarian Node tetangga berikutnya dapat diteruskan.



Gambar 8. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Semua Node Telah Terdaftar Dalam Stack Node

Bila semua Node yang ada dalam grid telah terdaftar, maka proses Backtracking berhenti, seperti pada Gambar 8.



Gambar 9. (A) Penampakan Orthographic, (B) Penampakan Persepctive, Hasil Maze Algoritma Backtracking Terbentuk

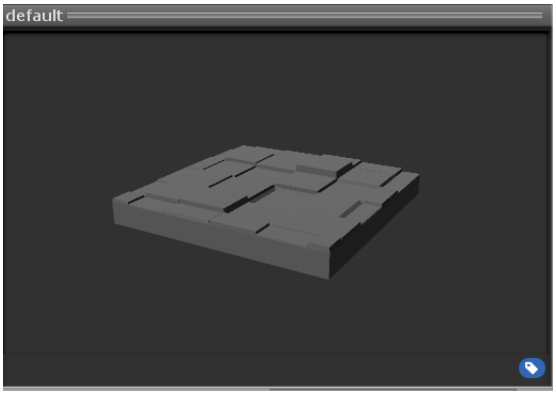
Direction value serta edges value yang telah tersemat dalam masing masing Node akan digantikan dengan mesh 3D, sehingga representasi environment 3D dapat diciptakan, direction akan berguna sebagai variable yang bertugas dimana posisi passsage/rute yang bisa dilewati satu Node terhadap Node lain. Sedangkan Node edges, akan memberi tahu dimana lokasi mesh wall harus diposisikan, ketika Node/cell memiliki 2 passsage didalamnya, maka Node tersebut pasti memiliki 2 value edges yang akan digantikan oleh mesh 3D.

Berdasarkan tahapan tahapan Gambar 9, proses recursive backtracking dapat disederhanakan sebagai script algoritma implementasi Gambar 10, dimana setiap proses mewakili bagaimana maze hasil PCG menggunakan algoritma recursive backtracking terbentuk.

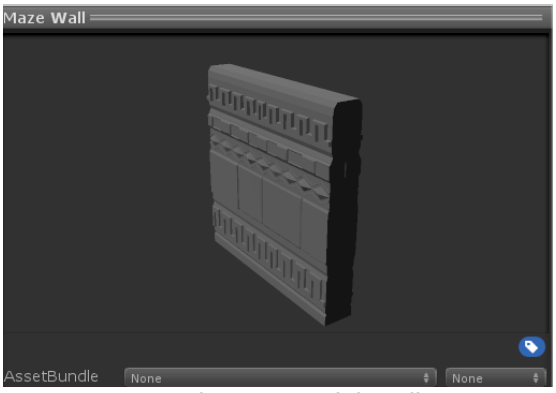
```
public IEnumerator GenerateRecursively()
{
    cells = new MazeCell[size.x, size.z];

    MazeCell initialCell = CreateCell(RandomCoordinates);
    yield return StartCoroutine( doGenerateRecursive(initialCell));
    yield return null;
}

IEnumerator doGenerateRecursive(MazeCell prevCell)
{

    do
    {
        yield return new WaitForSeconds(generationStepDelay);
        MazeDirection direction = prevCell.RandomUninitializeDirection;
        IntVector2 coordinates = prevCell.coordinates + direction.ToIntVector2();
        if (ContainsCoordinates(coordinates))
        {
            MazeCell neighbor = getCell(coordinates);
            if (neighbor == null)
            {
                neighbor = CreateCell(coordinates);
                CreatePassage(prevCell, neighbor, direction);
                yield return StartCoroutine(doGenerateRecursive(neighbor));
            }
            else
            {

                CreateWall(prevCell, neighbor, direction);
            }
        }
        else
        {
            CreateWall(prevCell, null, direction);
        }

    } while (!prevCell.IsFullyInitialized);

}
```

Gambar 10. Script Penerapan Algoritma Recursive Backtracking

Ketika algoritma telah diubah menjadi bentuk script untuk proses membuat maze PCG menggunakan algoritma recursive backtracking seperti hal diatas, maka langkah selanjutnya adalah menambahkan model sebagai representasi 3D untuk maze, seperti pada Gambar 11 dan Gambar 12. Setiap value didalamnya akan digantikan oleh mesh/model 3D. hal tersebut diatur otomatis melalui unity.

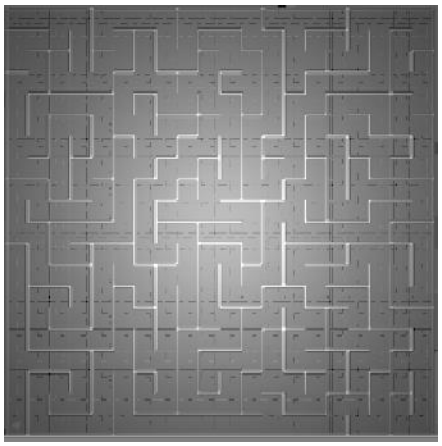


Gambar 11. Model Cell

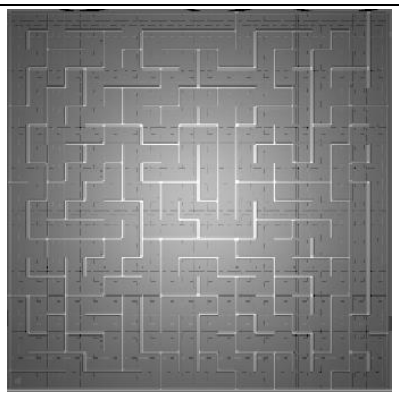


Gambar 12. Model Wall

Sebelumnya, penulis memberi value terhadap masing-masing labirin yang berhasil terbuat, hal ini memiliki arti bahwa tiap Maze yang terbentuk akan mewakili 1 pseudo number Generator (Seed). Seed diimplementasi agar penulis dapat mengakses hasil PCG labirin kembali ketika program dijalankan. Keanekaragaman Seed tergantung dari parameter size X dan Z, bila ukuran semakin massif, maka variasi Seed pun akan semakin banyak. Hasil dari implementasi untuk labirin PCG Recursive Backtracking dengan menggunakan ukuran 20*20 antara lain.

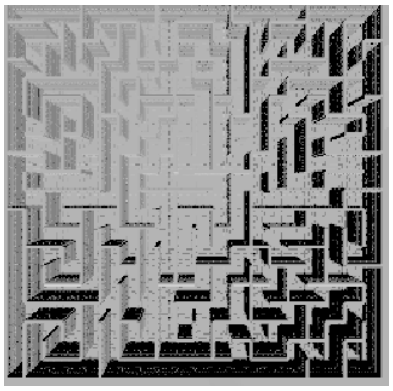


Gambar 13. PCG Seed 0 Orthographic Camera

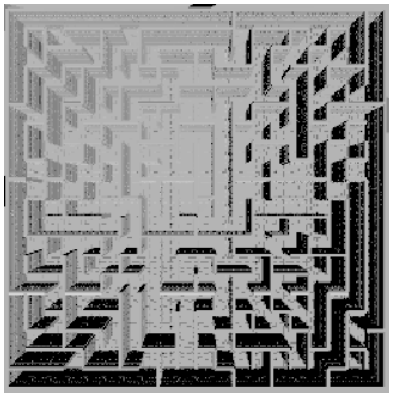


Gambar 14. PCG Seed 20 Orthographic Camera

Dua Gambar 13 dan Gambar 14 merupakan hasil maze berdasarkan tangkapan kamera maze secara orthographic.



Gambar 15. PCG Seed 0 Persepective Camera



Gambar 16. PCG Seed 20 Persepective Camera

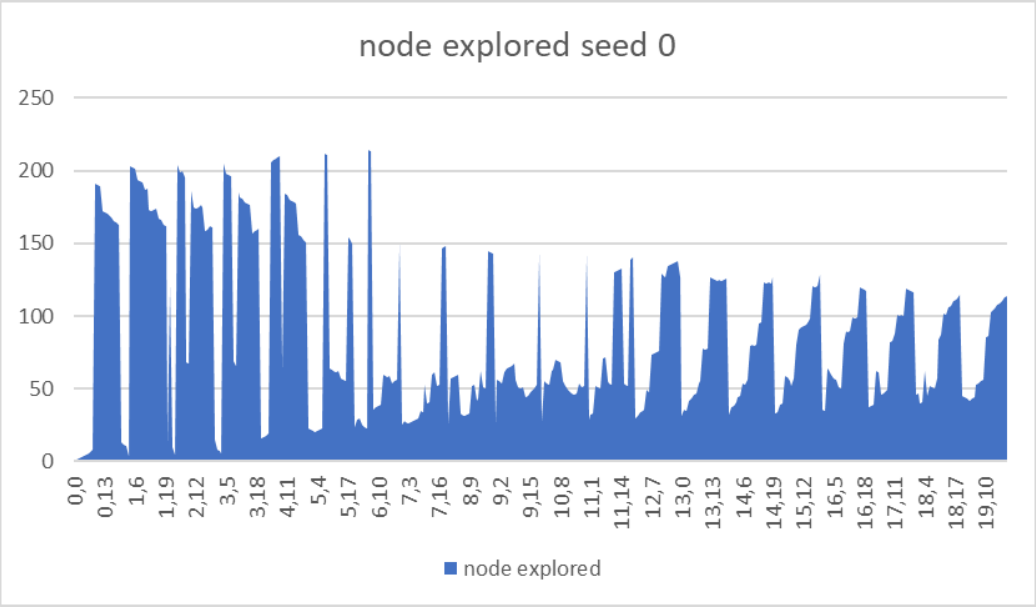
Dua Gambar 15 dan Gambar 16 merupakan hasil maze berdasarkan tangkapan kamera maze secara perspective. Mengenai uji coba scenario apakah maze termasuk dalam kategori perfect maze seperti yang disebutkan oleh Walter D.Pullen bahwa maze sempurna tidak akan memiliki looping cell maupun isolated, serta seluruh node dapat diakses dari arah manapun node awal berasal. Penulis akan memanfaatkan maze untuk ditelusuri menggunakan algoritma maze solver A* dimana node awal masing-masing maze akan ditetapkan pada titik yang sama yakni 0,0 sedangkan node akhir akan diubah setiap kali maze berhasil dihasilkan. Hal tersebut akan dilakukan berulang kali hingga keseluruhan node akhir berhasil ditelusuri. Sebelum hal tersebut dilakukan, penulis memastikan apakah kinerja A* terhadap hasil maze dapat diterapkan. Namun dengan node akhir berada dalam titik yang sama yakni 19,19.

Tabel 1. Tabel Maze Solver A Terhadap Maze Dengan Node Akhir 19,19*

Seed Maze	Node Start Coordinate	End Node Coordinate	Isolated Cell	Pathfinder A* implementation	Total Node	solveable
0	0,0	19,19	No Cell Isolated	Work Well	113	Yes
1	0,0	19,19	No Cell Isolated	Work Well	89	Yes
2	0,0	19,19	No Cell Isolated	Work Well	136	Yes
3	0,0	19,19	No Cell Isolated	Work Well	91	Yes
4	0,0	19,19	No Cell Isolated	Work Well	94	Yes
5	0,0	19,19	No Cell Isolated	Work Well	117	Yes
6	0,0	19,19	No Cell Isolated	Work Well	116	Yes
7	0,0	19,19	No Cell Isolated	Work Well	76	Yes
8	0,0	19,19	No Cell Isolated	Work Well	129	Yes
9	0,0	19,19	No Cell Isolated	Work Well	144	Yes
10	0,0	19,19	No Cell Isolated	Work Well	150	Yes
11	0,0	19,19	No Cell Isolated	Work Well	172	Yes
12	0,0	19,19	No Cell Isolated	Work Well	104	Yes
13	0,0	19,19	No Cell Isolated	Work Well	76	Yes
14	0,0	19,19	No Cell Isolated	Work Well	268	Yes
15	0,0	19,19	No Cell Isolated	Work Well	87	Yes
16	0,0	19,19	No Cell Isolated	Work Well	121	Yes
17	0,0	19,19	No Cell Isolated	Work Well	86	Yes
18	0,0	19,19	No Cell Isolated	Work Well	118	Yes
19	0,0	19,19	No Cell Isolated	Work Well	130	Yes
20	0,0	19,19	No Cell Isolated	Work Well	116	Yes
21	0,0	19,19	No Cell Isolated	Work Well	118	Yes
22	0,0	19,19	No Cell Isolated	Work Well	162	Yes
23	0,0	19,19	No Cell Isolated	Work Well	90	Yes
24	0,0	19,19	No Cell Isolated	Work Well	116	Yes
25	0,0	19,19	No Cell Isolated	Work Well	162	Yes

26	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*150*	*Yes*
27	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*72*	*Yes*
28	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*96*	*Yes*
29	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*136*	*Yes*
30	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*104*	*Yes*
31	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*110*	*Yes*
32	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*100*	*Yes*
33	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*118*	*Yes*
34	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*120*	*Yes*
35	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*104*	*Yes*
36	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*100*	*Yes*
37	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*94*	*Yes*
38	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*94*	*Yes*
39	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*98*	*Yes*
40	*0,0*	*19,19*	*No Cell Isolated*	*Work Well*	*120*	*Yes*

Berdasarkan data Tabel 1, maze solver pathfinding A* dinyatakan berhasil diimplementasi, hal selanjutnya adalah melakukan scenario uji node akhir berada diposisi yang berbeda, seluruh node akan ditelusuri. Apabila ada node akhir yang tidak dapat ditelusuri, maka pengujian dinyatakan tidak berhasil masuk dalam kategori perfect maze.



Gambar 17. Representasi Data Node Akhir Explored Interval Coordinates 0,0 - 19,19 Seed 0

Pengujian skenario Gambar 17, menghasilkan graph dimana setiap node koordinat akhir direpresentasikan sebagai data X, sedangkan total node direpresentasikan sebagai data Y. bila ada node akhir yang tidak menemukan rute, maka value node akhir akan menunjukkan angka 0, namun dalam penelitian ini, mengambil sampel data seed 0, penulis tidak menemukan sama sekali cell yang terisolasi ataupun tidak dapat dikunjungi dari posisi awal node koordinat 0,0. Hal tersebut dapat menyatakan bahwa keseluruhan node dalam maze dapat diakses dari titik manapun.

4. Kesimpulan

Kesimpulan menjelaskan apa yang diharapkan pada bagian pendahuluan, serta kesimpulan dari bab hasil penelitian dan pembahasan. Di dalam kesimpulan hendaknya disampaikan beberapa hal yang telah dicapai. Kesimpulan juga dapat ditambahkan dengan rencana pengembangan penelitian kedep Implementasi sesuai algoritma PCG berhasil diterapkan, dengan beberapa penyesuaian berkaitan dengan objek akhir yang diinginkan harus dapat divisualisasikan dengan model 3D, berkaitan hal tersebut, poin utama yang mencakup algoritma Recursive Backtracking berhasil diimplementasi. Melihat permasalahan kedua tujuan penelitian ini dilakukan, konten labirin yang diciptakan berhasil mencakup kategori komponen labirin sempurna, beserta visualisasi yang diharapkan mampu memperjelas bagaimana kategori Perfect Maze itu terbentuk. Mengacu pada penelitian sebelumnya yang menyatakan definisi kategori sempurna tanpa ada perubahan berhasil diimplementasikan. Hal terakhir yang menjadi tujuan penelitian ini, yakni algoritma A* dapat diimplementasi serta berjalan dalam environment yang dihasilkan oleh PCG. Kesesuaian kebutuhan A* pun tidak menghambat proses implementasi. Pengujian A* ditentukan dengan cara memberi acuan Node awal serta Node akhir pada PCG yang telah dihasilkan oleh algoritma Recursive Backtracking. Posisi kedua Node tersebut tidak menghambat kinerja algoritma A*, hal tersebut dapat dinyatakan karena setiap Node yang dihasilkan pada labirin, dapat diakses dari arah manapun. Tidak adanya looping Cell menambah nilai positif algoritma ini, dan tidak adanya isolated Cell menjadikan algoritma ini bekerja sesuai arah hipotesa penelitian. Dan kesimpulan terakhir algoritma ini berhasil menghasilkan Maze yang sesuai pernyataan deffinisi dari Maze Perfect.

Oleh karena itu penggunaan metode PCG sangat disarankan sebagai metode alternative untuk membuat konten environment secara procedural, terlebih lagi dalam konteks labirin, dimana algoritma Recursive Backtracking terbukti optimal serta mudah diterapkan..

Referensi

[1] R. B. Sirait, “Perancangan Aplikasi Game Labirin Dengan Menggunakan Algoritma Backtracking,” *Pelita Inform. Budi Darma*, vol. Volume 5, no. Nomor 2, p. Halaman 100-103, 2013.
[2] L. H. S. Lelis, W. M. P. Reis, and K. Gal, “Procedural Generation of Game Maps with Human-in-the-Loop Algorithms,” vol. 14, no. 8, pp. 1–10, 2015.
[3] P. H. Kim and R. Crawfist, “The Quest For The Perfect Perfect-Maze,” *CGAMES 2015*, vol. 4, no. 3, pp. 1–28, 2015.
[4] J. Buck, *Mazes for Programmers Code Your Own Twisty Little Passages*.
[5] A. Pech, “Using genetic algorithms to find cellular automata rule sets capable of generating maze like game level layouts,” *Bachelor Thesis, Ed. Cowan Univ.*, 2013.
[6] B. Fahrudin *et al.*, “Penerapan Algoritma Backtracking Pada Permainan Capsa,” vol. 3, no. 6, pp. 30–33, 2016.
[7] M. Foltin, “Automated Maze Generation and Human Interaction,” *Brno Masaryk Univ. Fac. Informatics*, pp. 1–76, 2011.
[8] I. dan W. W. Ahmad, “Penerapan Algoritma A Star (A*) pada Game Petualangan Labirin Berbasis Android,” *J. Ilmu Komput. dan Inform. Univ. Muhammadiyah Surakarta*, vol. 3, no. No 2, pp. 57–63, 2017.
[9] A. Karlsson, “Evaluation of the Complexity of Procedurally Generated Maze Algorithms,” no. June, 2018.