A Dynamic Terrain-Spaced Maze Generation Algorithm

Metin Turan[1], Kenan Aydın[2]

GJCST Classification
I.2.1, I.2.2

***Abstract*-Maze algorithms are generally developed to create mazes in a game board, which consist of single cell of passages and nearly all cells are accessible. However, it would be useful if some group of cells randomly separated for terrain design and some passages was randomly irregular in width or contains rooms for arrangement of game objects.In this study a randomly irregular and terrain-spaced maze generation algorithm has been developed. The randomly produced rooms within the generated passages can be used for planning game strategy. On the other hand the cells which are not used for passages may be used for terrain design. This algorithm only needs boundary check to prevent getting out of the game board. Moreover maze complexity can be identified by a ratio which is defined as the ratio of passage cells number over total number of cells on the game board.**

I. Introduction

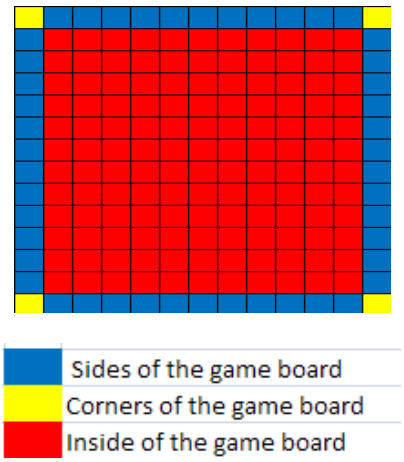
There are various kinds of maze algorithms in the literature; most of them use tree structure to generate a maze and others use sets to generate mazes. Maze generation algorithms generally focus on generating perfect mazes. A maze is perfect if every cell in the maze is accessible and there is exactly one path to another cell. Well known perfect algorithms using tree structure is Prim algorithm [1] where using sets is Kruksal algorithm [2]. Both use same approach to create a random maze which constructs cell walls using a random chosen side of the cell to create a wall.Maze algorithms (perfect, braid, unicursal, sparse, partial braid) [3,4,5,6,7] generate passages in a given two or n- dimensional game board which is composed of a number of cells. Passage width is generally regular for all passages through the entire game board. These algorithms aim is only to create a solvable maze. They haven't got ability to generate rooms within the passages. On the other hand, sparse algorithms [8] create passages where some cells are left uncreated, resulting in an irregular maze with wide passages and rooms. However the maze shapes not eligible to design rooms for game characters and to produce amusing maze games. Moreover it is still impossible saving empty spaces out of passages which would be used for designing terrains.However, it should be noted that although there are lots of tools available to make mazes more challenging when generating them on a computer, often the most interesting mazes are those designed and created by hand – "it is hard to have a random algorithm which will also generate a psychologically interesting maze"[9]. In other words, the passages of lots of two dimensional popular games (maze, labyrinth and racing game categories) have been designed statically. However, the proposed algorithm generates dynamic (random) passages for each new play which is more interesting for players.The proposed algorithm assumes game board is a wall initially. The aim is to construct randomly passages through the wall. This is a reverse approach if it is compared with wall adding techniques used in maze algorithms. The complexity of maze can be controlled by a parameter (filled ratio) given to proposed algorithm. It also creates loops and dead-ends in the maze. Loops are connected to at least one passage. On the other hand limited dead-ends help to create traps in the artificial games.Proposed algorithm starts with a random point on the board. Then it finds a random direction and a random passage length (number of cells) to construct passages in the maze. This step is repeated recursively until it reaches the given maze complexity target which is defined as the ratio of passage cells number over total number of cells on the game board.Unlike other known algorithms, proposed algorithm does not require extreme memory except an n-dimensional matrix to simulate the game board. A pixel or a predetermined area size (e.g. 20x20 pixels) could be represented with a cell in the matrix of size maxCOLUMN x maxROW, where for two dimensional matrix n shows x coordinate and m shows the y coordinate.The rest of the article is designed so to describe the work in Section II, statistical findings in Section III, algorithm analysis in section IV, game example(using proposed algorithm) in Section V, conclusion in Section VI and further works in section VII.

II. Work

The proposed algorithm (***generateMaze()***) depends on random values as the previous algorithms did. Due to the randomization generates same values over and over again a special randomize function is needed to construct reliable mazes. This general randomize function is constructed by adding CPU time on the current time which gives highly better results.First of all, ***generateMaze()*** finds a random starting point at one side of the board. It may be on horizontal (x is between 1 and maxCOLUMN) or vertical(y is 1 between maxROW) boundary which is determined by a parameter(boardside).Then generateMaze***()*** creates the passages. Before a new passage is created, it checks if the maze complexity target is obtained. The maze complexity is provided by a parameter (complexity). New passage direction is produced by general randomize function. The board is divided into three regions

About[1] - Kültür University, Istanbul, Turkey m.turan@iku.edu.tr
About[2]- Kültür University, Istanbul, Turkey heret1que@hotmail.com

where each cell has different possible directions to route. First region is the corner cells, where routing is lowest and there are two possible directions to move since they are on the corners of the game board. Second region is the side cells excluding the corner cells, where routing has three possible directions. Third region is the inner section of the board, where routing has four directions. This is outlined at Picture I.



Picture I: The regions of a game board

By the time the direction is selected, ***generateMaze()*** uses a random length (number of cells) to go forward in the selected direction. It checks the case of exceeding boundaries. A generation method which doesn't use a random length after direction decider will not produce a complex maze. However it will generate different type of terrain as in Picture II.

Picture II: A maze generated by proposed algorithm without "how many cells it will go" function.

generateMaze() called recursively until it reaches maze complexity.

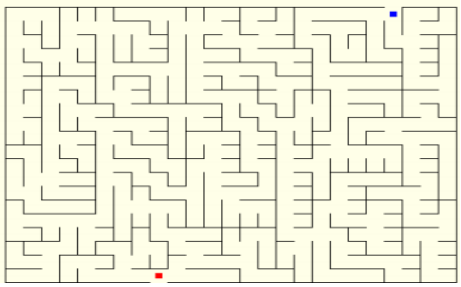
Checking the complexity procedure calculates the ratio of passage cells number over total number of cells on the game board

Unlike other known maze-generation algorithms, the ***generateMaze()*** doesn't create passages cell by cell, instead use number of cells. This speeds up the construction time of maze in the case of big boards. The ***generateMaze()*** pseudo-code is given below.

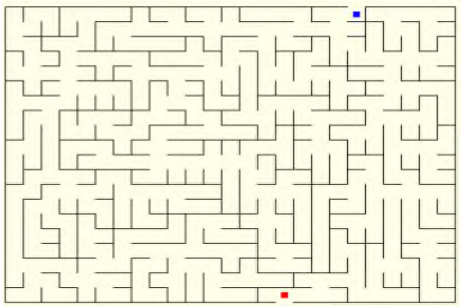
```
void generateMaze( boardside, complexity)
{
col,row=call startPoint(boardside);
call generatePassage(col,row,complexity);
}
col,row startPoint(boardside)
{
return random row and column value using boardside;
/* on the vertical or on the horizontal side of the board /*
}
void generatePassage(currentRow,currentColumn,
complexity)
{
direction=decideDirection(randomize());

passageLength=randomize()
if (checkBoundary())
currentRow,currentColumn=createPassage(passageLength);
else
currentRow,currentColumn=createPassage(Length to the
boundary);
if (checkComplexity(complexity))
generatePassage(currentRow,currentColumn,complexity);
}
```

Example outputs of Kruskal, Prim and ***generateMaze***() is given on Picture III, Picture IV and Picture V respectively.



Picture III: A maze generated by Kruskal Algorithm

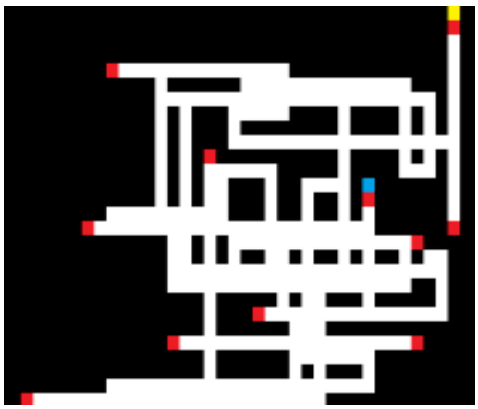


Picture IV: A maze generated by Prim Algorithm

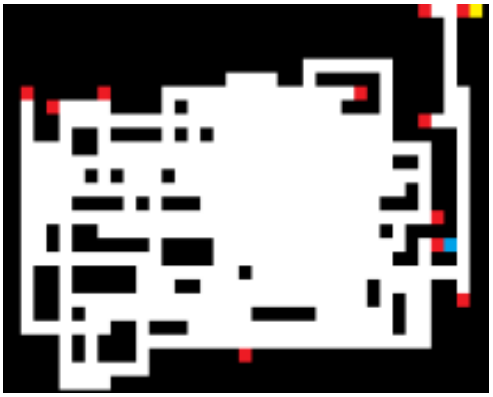
Picture V: A maze generated by Proposed Algorithm

Locating and storing the coordinates of the dead ends is also possible by findDeadEnds() algorithm.
After the generation of the maze is completed, findDeadEnds() will be called and simply looking for the cells which only have one neighbor cell to access these cells.

```
void findDeadEnds()
{
for each counting numbers between 0 and maxROW , called ROW
for each counting numbers between 0 and maxCOLUMN , called COL
If(NeighborCellsCount(ROW,COL)= =1)
DeadEnds[ROW][COL]=true;
}
```



Picture VI: A Maze generated by Proposed Algorithm by using 30% Filled Ratio

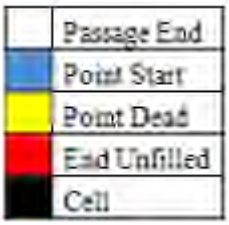


Picture VII: A Maze generated by Proposed Algorithm by using 45% Filled Ratio

Picture VIII: A Maze generated by Proposed Algorithm by using 60% Filled Ratio

Picture IX: A Maze generated by Proposed Algorithm by using75% Filled Ratio

Table I
Visual information of simulated mazes.



Above example maze outputs is given in pictures Picture VI, Picture VII, Picture VIII and Picture IV per different filled ratio values (0.3, 0.45, 0.6, 0.75). The meaning of the colors in the mazes is given in Table I.

III. STATISTICAL FINDINGS

Statistical work has been done for step, dead end, turn and collision properties of ***generateMaze() algorithm*** by running four different filled ratio values, each time it has been executed 200 times (Table II).

Table II
Average values of Steps, Dead Ends, Turns and Collisions per filled ratio

Filled Ratio	Step	Dead Ends	Turn	Collision
30%	1420	24	469	2546
45%	2851	23	935	6520
60%	5876	20	1914	16661
75%	16784	16	5474	57130

Step means that the number of passage construction movement (for each movement a number of cells are processed- reverse movement on the existing passages is also considered) produced by ***generateMaze().***

- Dead ends are the cells which only have one neighbor cell to access these cells.
- Turn is defined as changing previous direction.
- Filled ratio is the ratio of passage cells number over total number of cells on the game board.
- Collision is the number of how many times passages re-filled by reversing or overlapping.

In Figure I, it can be viewed that the average step values gets higher as the filled ratio gets higher. As a result average turn values gets higher. Nearly in every 3 steps ***generateMaze()*** makes a turn.

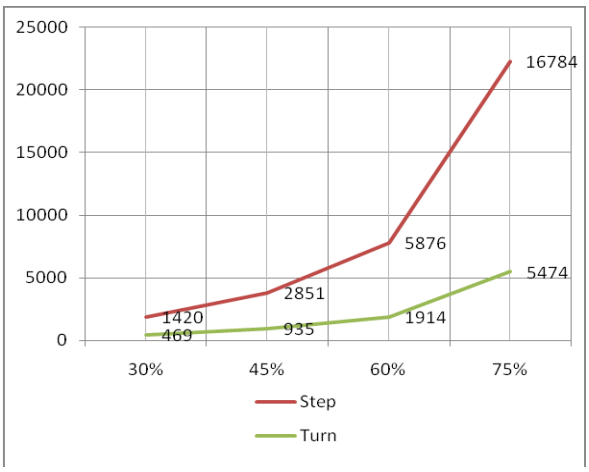


Figure I: Average Step and Turn Values per Filled Ratio

In Figure II, it can be seen that the dead ends are gets lower as the filled ratio gets higher. This means that there is a negative correlation between these two properties. It is an expected result of the ***generateMaze().*** It makes various kinds of turns which eliminate existing dead ends by connecting them with other new passages that has been intersected by the existing dead ends, since a dead end occurs at cells which connected with only one neighbor cell.

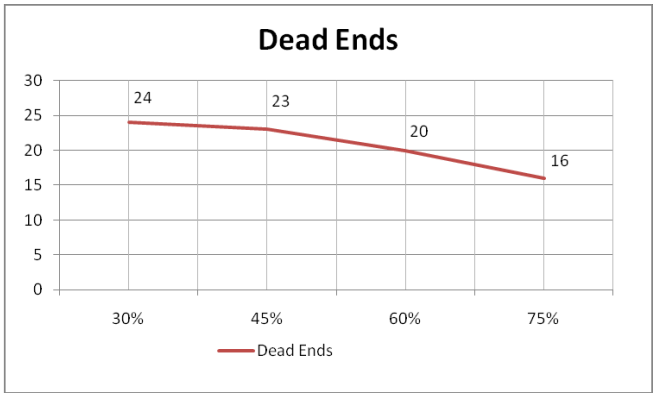


Figure II: Number of Dead Ends versus Filled Ratio

Table III
Correlation values between Turn – Collision and Dead End

Correlation Values	30%	45%	60%	75%
Turn – Collision	~1	~1	~1	~1
Dead End – Turn	~0	~0	~0	~0
Dead End - Collision	~0	~0	~0	~0

On the other hand, there is a positive correlation between turn and collision. This is an expected result due to the ***generateMaze()*** runs to create passages to reach to the filled ratio(when steps increases, turns increases also Figure I), while makes a lot of turns in random directions which causes more collisions. However, Dead End is independent property from turn and collision. It can be concluded from Table III.

IV. ALGORITHM ANALYSIS

Both Prim and Kruksal algorithm create vertices and edges to construct their mazes. Their time complexity nearly same and can be defined as O(E log V), where E and V implies edges and vertices respectively[10]. However, generateMaze() doesn't create edges or vertices. It randomly produces passages. Passages are constructed on a board which is represented by a matrix (maxCOLUMN is ***x***, maxROW is ***y***). Moreover the ***filled ratio*** (sparse area percentage) and random ***length*** (number of cells to fill for each step) are also determines the time complexity. Using these parameters we calculate the following game board construction complexities for both the worst and the best conditions under assumption of no collision [11].For the worst condition, the random length is 1. This time ***generatePassage()*** is called **x*y** times.
filled ratio* (max(**x**,**y**)²)*(4+**length**) where 4 shows the number of steps executed in auxiliary functions and inside the ***generatePassage().*** If the matrix NxN size then the worst game board construction complexity is;
O(5*filled ratio*N²) For the best condition, the random length is max(**x**,**y**). This time ***generatePassage()*** is called only min(**x**,**y**) times. **filled ratio*** (min(**x**,**y**)*(4+max(x,y)))
And if the matrix NxN size then the shortest game board construction complexity is; O (filled ratio*(N²+4N)) This complexity values are acceptable and applicable if they compared with the Kruskal's (O(E log V)) and Prim's (O(E + V log V)) algorithms, where E and V shows edges and vertices respectively.

V. AN EXAMPLE GAME

A simple 2-Dimensional Windows based game developed by using the ***generateMaze()*** (Picture X)[12]. The goal of the game is to collect all the stars and proceed through levels (1 to 10) to complete the game. Player has to collect more stars for each next level. By the time, player shouldn't collect skulls. It decreases lives of the player (4). If player collect first-aid bags then it increases player lives. Player character is chased by an AI controlled character while collecting the stars. If AI character catches the player the live is decreased by one. ***generateMaze()*** allows creating dead end passages which make this game more amusing. Also an algorithm is developed so dispersing collectable items on dead-end passages to challenge the game.

Picture X: An example game created by proposed algorithm.

VI. CONCLUSION

The ***generateMaze()*** is a flexible(parametric) algorithm and if it is compared with the sparse algorithms in literature, it is truly random (not stable) as a result of the used technique. By the way, statistics from the hundreds of experiments have shown that created mazes are not related to each other and well-shaped. Moreover, computational complexity is less than any sparse algorithm. The worst case is to check only next matrix location O(1). On the other hand, memory requirements depends on the game board matrix sized N column and M rows. The worst case is not more than O(M x N). It decreases by the dimension of the passage width.The only problem using ***generateMaze()*** is the stack overflow error above the 0.75 filled ratio. When the filled ratio gets higher, collision increases and recursive call of ***generatePassage()*** reaches a value which causes stack overflow error. When filled ratio is greater than the 0.75 then generated maze is not well (Picture IX). So this is not a big problem for algorithm acceptance.

VII. FURTHER WORKS

Algorithm optimization is needed to control dead end generation and reduce the collision values which will speed up the process of maze generation.This algorithm may also be generalized to use in 3D terrain-spaced maze generation easily.

VIII. REFERENCES

1) [R. C. Prim: Shortest connection networks and some generalizations. In: Bell System Technical Journal, 36 (1957), pp. 1389–1401
2) Joseph. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", Proceedings of the American Mathematical Society, vol. 7, no. 1 (Feb, 1956), pp. 48–50
3) Fisher A., "The Amazing Book of Mazes", Harry R. Abrams Inc.,2006

4) http://www.conceptispuzzles.com
5) Kern H., “Through the Labyrinth: designs and meanings over 5000 years”, Prestel , 2000
6) http://www.astrolog.org/labyrnth
7) T.H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms, MIT Press and McGraw-Hill, 2001
8) Hans Pedersen and Karan Singh, “Organic labyrinths and mazes”, Proceedings of the 4th international symposium on non-photorealistic animation and rendering NPAR ’06, ACM Press, June 2006
9) Danny Kodicek, “ Mathematics and Physics for Programmers”, Charles River Media , 2005
10) Anany Levitin, “Introduction to The Design & Analysis of Algorithms”, Addison Wesley, International Edition, 2003.
11) Oliver Kullman, CS-232 algorithms and complexity notes, October 2005.
12) K. Aydın, M. Turan, “A Simple Windows Based Game”, Student Final Project, İstanbul Kültür University, Dept. Of Comp. Science, January 2010.