# Generation of an Arbitrary Shaped Large Maze by Assembling Mazes

Cheong-mok Bae, Eun Kwang Kim, Jongchan Lee, Kyung-Joong Kim* and Joong-Chae Na
Dept. of Computer Science and Engineering, Sejong University, Seoul, Korea
cjdahrl@gmail.com, sdy12123@naver.com, jchlee91@gmail.com, kimkj@sejong.ac.kr, jcna@sejong.ac.kr

***Abstract*—Lots of games have used maze generation for their maps, geographical features or terrains. Previously, they used spanning tree algorithms, growing tree algorithm, and so on. However, the methods placed restrictions on the shape of mazes and used fixed matrices. In this paper, we propose a simple and easy way to generate an arbitrary shaped maze by assembling mazes. Because it's possible to selectively combining mazes with user preference, the final big maze can be expected to be a personalized one. If a database of mazes with gamer's playing logs or preferences is available, the system can dynamically compose an arbitrary new mazes in real time. Furthermore, it can support gamers to create new complex mazes for user-generated contents.**

***Keywords—Maze; Arbitrary Maze; Procedural Contents Generation;***

## I. INTRODUCTION

It's a challenging problem to create large/arbitrary-shaped mazes in real-time considering game player's preferences. It's desirable to automatically generate the mazes based on player's skills and playing styles. However, the traditional maze generation algorithms time/space complexity is proportional to the number of cells. If it's a relatively large number, the real-time generation is not feasible. In mobile games, it's not working only with the small number of cells. Generally, the most widely used methods in maze generation are spanning tree, Prim's, Kruskal's, and Tree Growing algorithms. They generate square or rectangular shaped mazes because it commonly uses square or rectangular matrices to generate results applying Spanning tree algorithm.

In this paper, we propose to store lots of mazes into a database with meta-level information and dynamically assemble them to create large/arbitrary shaped mazes. It creates a new maze by assembling mazes stored in the database. At first, it searches for the mazes suitable as a component of the large maze based on meta-level information of objects. After the search, it assembles them into a single large/arbitrary-shaped maze. If the mazes in the database is enough, they can support different types of request from the games. Because it assembles each maze as it is, it maintains the characteristics (position of walls, inner routes, etc.) of components. In conventional maze algorithms, they're not easily supported.

## II. PROPOSED METHOD

***Standard maze*** (Standard perfect maze): A maze generated by using Spanning tree algorithm (Depth first/Breadth first spanning tree) It has only one way to reach from one random location to another random location, no circular ways, and no closed spaces or no inaccessible spaces.

***Fully connected maze***: It has some difference with the Standard maze. It can have plural ways to reach from one random location to another random location, circular ways, but no closed spaces or no inaccessible spaces.

In our proposal, a method for generating arbitrary mazes executes the following procedure.

1. Locate plural First mazes and let their at least whole or partial one-side outlines are overlapped. (Figure 1, overlapped outline is represented by bold line)
2. Determine at least one location which links mazes on overlapped outline in Step 1.
3. Make Second maze creating link-ways and remove outlines determined in Step2.

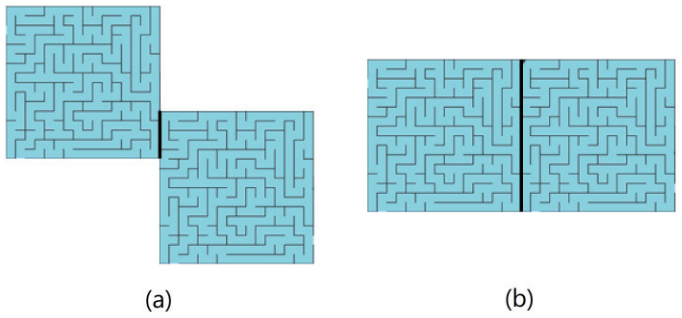

Figure 1. Examples of overlapped outline in Step 1 (a) partial outlines are overlapped (b) whole outlines are overlapped.

The First mazes can be mazes generated by Spanning tree algorithms, fully connected mazes, or selected mazes from a Maze-Database stores plural mazes generated by this procedure. The Second mazes can be atypical mazes. Also, Second mazes can be standard mazes, fully connected mazes. Besides, Second mazes can be stored in the Maze Database. The locations to link are determined randomly on overlapped outline or selected by user. Figure 2 shows a pseudo code of the algorithm.

* Corresponding author

```
struct {
    Point (x, y): coordinate
    AbstractData maze-inner-information
    Value evaluation value (if it's necessary)
} Maze;
record Maze[] MazeDatabase;
function GenerateMaze(Maze m1, Maze m2) {
    set m1's location to any place in two dimensional plane
    set m1's coordinate from x-y axis
    set m2's relative location for m1 at least one-side outlines are overlapped.
    select will-removed-walls[] from overlapped wall(s)
    link mazes by remove wall of will-removed-walls[]
    store linked-maze in MazeDatabase
}
```

Figure 2. A pseudo code of maze assembly (Assume that cell's floors are square and every cell has the same size as 1 and maze walls have no thickness.)

## III. RESULTS

In our proposal, if we have enough storage to store mazes, we can generate mazes in a time efficient way. Because our approach assembles mazes stored in database, it only needs time to combine them. For example, if you want to create a maze by using four 10 × 10 mazes, it assembles them in constant time with 4 ×$10^2$ memory cells. On the other hands, Kruskal's, Prim's and Growing Tree algorithms require memory space proportional to the number of cells ($N^2$). Their time complexity also depends on the number of cells [1]. For example, the conventional algorithms need $20^2$ memory cells and time proportional to the $20^2$ cells to create a 20 × 20 maze.

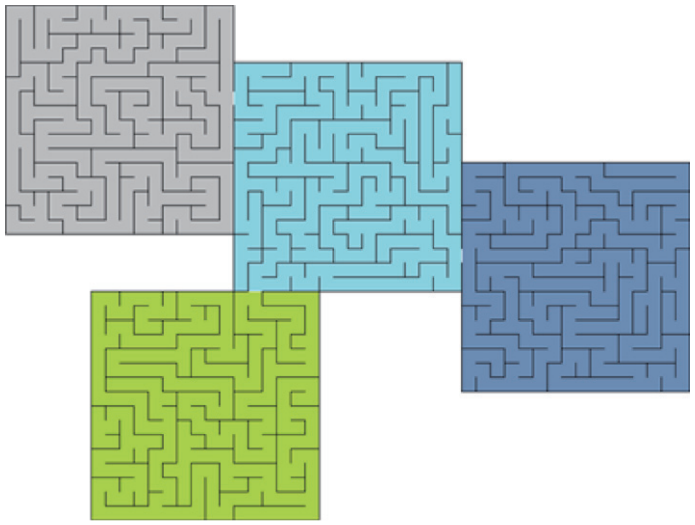

Figure 3. A maze generated by assembling four 16 × 16 mazes

In addition, the final outcome can maintain each component mazes' characteristics with their original shape. Simply, it's possible to estimate the favorability of assembled maze based on the reputation of each sub-mazes. Figure 3 shows an example of maze assembled four different sub mazes. If user evaluated this maze's level of difficulty as high, a maze generator reflect the value to control the levels of game when it selects mazes to assemble.

We used the maze assembly algorithm to create a maze for Unity-based games. Our game was inspired by the movie titled as "Maze Runner." In the movie, the boys were trapped in a maze. Figure 4 shows an example of 100 × 100 maze with 8 × 8 empty space in the center. It assembled four different 54 × 46 mazes.

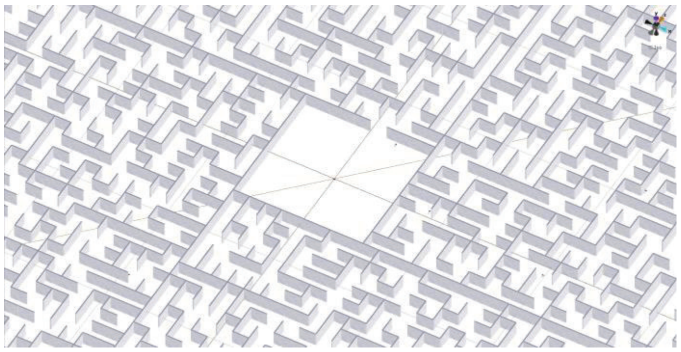

Figure 4. An example of mazes used in a Unity-based game (It has an empty space in the center of the maze)

## IV. CONCLUSIONS AND FUTURE WORK

Maze generation is used not only to construct a simple maze, but also backgrounds, an extended shape of maze. For example, maze-generating method is used to construct random dungeon in some Role-Playing Game (especially, Rogue-like games). They construct a huge dungeon by replacing maze's one cell with one room (space or terrain) or locating many rooms connected to passageways in the remaining space using maze generating algorithms [2]. "Mystery Dungeon series," *Spike Chunsoft* is a representative game with the method.

In our proposal, the method assembles mazes in database to construct a large maze. If it reuses the new large maze as a component, it can create more complex maze step by step. If the assembly is used for automatic contents generation, it can provide with a number of arbitrary mazes. The maze database stores lots of different types of mazes with metadata (user preferences, maze's properties and so on). We demonstrated our approach on the Unity-based games motived by the movie "Maze Runner." On the other hands, it's also interesting to define unapproachable cells in the matrix to create arbitrary shaped mazes.

## ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2013 R1A2A2A01016589).

## REFERENCES

[1] Maze Classification, http://www.astrolog.org/labyrnth/algrithm.htm

[2] Rooms and Maze: A Procedural Dungeon Generator, http://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/