# Game technology final report

**Game design plan:**

We designed a ball game with Unity and we focus on level generation and a simple AI for path finding. The rules for the game are very simple: the player must navigate the ball in a maze to find the win cube within the time limit and avoid potential traps. The game can give player a potential path to the win cube if the player click on “hints” button. The maze can rotate at a specific speed and the ball can only jump once every time it touches the wall. The wall becomes “penetrable” if it has the same color as the ball. Other than the algorithms, we customized the particle system and did two shaders.

**Algorithm:**

1. We used recursive backtracking for each level generation. The algorithm takes in width, height and difficulty of a maze.
   1. The algorithm keeps a 2D array “maze[,].” After it generates a maze, it will randomly generate the wall. The wall are divided into two categories: pure wall (impenetrable) and walls with different colors. The algorithm takes in difficulty and use it to generate the rate for impenetrable walls. If the difficulty is larger, the possibility for not impenetrable walls will be larger.
2. Used A star for path finding
   1. The algorithm keeps the distance from current cell to the end cell and the distance from starting cell to current cell.
   2. It gives priority to cell with smaller total distance, which equals distance from starting cell plus distance to the end cell.
   3. The algorithm does not use the “change color” function.
   4. The algorithm uses depth first search
   5. The algorithm use the ball’s current position as starting position

**Shaders:**

1. We did two shaders, one for the ball and one for the wall. The ball shader has diffuse and specular. The wall shader can “glow” and we did bump map.

**Difficulties:**

1. Customize the maze generation and path finding algorithm
   1. Since our maze is always rotating, it is a little difficult to find which cell the ball is currently in. The way I solve it is to get the quaternion for the current rotation and invert it. Then round it to the nearest integer. However, the process has some truncation error. Sometimes, if the maze is inclined a lot, the hint can contain some errors for the starting position.
   2. It takes a little time to let the algorithms fit in our game
2. Learn shader is a little difficult. Unity shader uses shader lab and it has many built in arguments that I am not aware of.

**Possible improvements for the future**

1. Get a better way to find the starting position
2. Adopt the algorithm so that it can properly penetrate the wall and detect color change