* **Algorithm used for creating the maze:**
  + I use a (slightly modified) Wilson’s Algorithm for generation the maze. The reason why I modify the algorithm is because Wilson’s Algorithm is originally designed to generate mazes whose wall do not occupy the maze space. However, in the assignment, the wall (tree) itself occupies one grid of the maze.
  + In short, the maze will be initially full of obstacles (trees), and the game uses Wilsons algorithm to break obstacles to create walkable grids .
* **Random and Dynamic:**
  + At the initialization of the game, the game generates 16 different mazes and store them as a set of two-dimensional arrays. So that once the player moves (time advances), the game compare the current maze with the maze at the next time frame to destroy / create walls (trees) as needed. (This improves the performance significantly compared with generating the whole maze again).
  + I use a single Random object based on current system time for all the maze generations, this ensures that each maze will be different. Because the sequence of numbers generated by one random object only repeats after a very long time. Since it is very likely to have a relatively long-time interval (at least seconds) between runs of games, this makes sure that the Random object will not be created with the same seed accidently.
* **Track the path.**
  + Before the game generates a set of mazes, it creates a solution from boundary to boundary by random walk (the boundary of 8 \* 8 part, not the actual boundary surrounding the maze area). The length of the path is set to from 13 to 15 for more variations of paths. The reason why the maximum length is one smaller than the maximum time frame is because later a boundary will be added to the 8 \* 8 maze area, and there is one more step to the exit at the actual boundary.
  + After that, to make the maze solving experience more interesting, instead of reserve the whole path, the game reserve one segment of the path when generating each maze. For example, at the first maze generation, the first 5 grids on the solution path will be reserved to be empty during the maze generation initialization. At the second maze generation, the 2 – 6 grids on the solution path will be reserved to be empty, etc.
  + This ensures there will always be a path connecting the entrance and the exit in the 3D maze (2D + 16 time)
* I use a set of model and texture assets in order to make the maze looks more like a forest, along with some pictures (for aim with projectile) and fonts (since the build-in font is buggy :).) for the UI. Specifically, they are:
  + Tree models: <https://assetstore.unity.com/packages/3d/vegetation/trees/free-trees-103208>
  + Textures:
    - <https://assetstore.unity.com/packages/2d/textures-materials/floors/stylized-terrain-texture-153469>
    - <https://assetstore.unity.com/packages/2d/textures-materials/glass/stylized-grass-texture-153153>
  + Key Model: <https://assetstore.unity.com/packages/3d/handpainted-keys-42044>
  + Aim Picture: <https://www.iconfinder.com/icons/183741/aim_crosshairs_hunting_target_icon>
  + Fonts: Inconsolata and TurretRoad from Google Fonts
* The key challenge is to create the 2D represented maze in a 3D fashion. As described above, I had to do a sort of “pre-processing” before actual generation the maze. The actual implementation part is more entertaining, though. Also, in order to achieve the discrete movement in the maze, I chose to use a grid-based movement system in the maze, which I spent sometime though about. The actual switching between two movement system is not that hard. To simplify the problem, I created the maze from (0, 0, 0) of the unity space, and let one tree occupies one grid. This simplifies the transition from the abstract maze representation to actual maze representation. However, this can be easily customized by adding some offset to the actual maze representation.