**Maze Game**

**Overview**

My game will be a maze game. The game will randomly generate a maze using a recursive algorithm. The play will have a limited time to solve the maze. As the player progresses through levels the timer will decrease but the player speed will increase. The player will realize that with the time limit decreasing every move counts and has to be precise, but the increase in speed will cause more mistakes.

**Rational**

This game is inspired by Snail Maze, a game that came installed on the bios of the Sega Master System. If the player started the system without a cartridge and held up, button 1 and button 2 Snail Maze would start. I played this game for hours. Despite having a good collection of videogames I found myself drawn to the simplicity of this game. Snail Game only had 1 maze and the reason why I liked it was the increasing difficulty.

After the Sega Geneses came out, Sega decided to save the memory in the bios and not include Snail Game with the bios. Snail Game had very little societal impact, but randomly generated mazes might have bigger impact in game design by offering a fresh experience every time the game is played.

For this game I will use a recursive backtracking algorithm and a stack to track which cells have been visited to fully mark out a maze.

<https://en.wikipedia.org/wiki/Snail_Maze>

<https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>

**Gameplay**

The game starts by showing the create maze animation and the solution will be shown for a few seconds. The player will be represented by a mouse. By using the arrow keys or WASD the player can navigate the maze. A timer will start and if it reaches 0 before the player reaches the goal the game will end. The player can also push R to restart or Q to show the solution.

**Objects**

The game will use the following major objects:

**Mouse**

This object represents the player

**Variables**

* x coordinate
* y coordinate
* load sprite (png img)
* x velocity
* y velocity
* KEYDOWN event for mouse turning up, down, left, right
* KEYUP for stopping the mouse
* boolean for if mouse is at goal
* boolean for is mouse is running into wall
* current room stores mouse location

**Functions**

* barrier(mouse): returns true if mouse is running into wall
* spawn (): make mouse appear on the screen
* goal(): check if mouse is at goal
* getVelocity(level): increase the speed of mouse depending on level
* currentRoom(): store current room of mouse

**Timer**

This object assigns time mouse has to finish and starts gameOver() if timer is at 0 and mouse is not at goal.

**Variables**

* Timer start amount
* boolean for T = 0

**Functions**

* startTime(level): Time at start of level
* gameOver(): game over screen if timer runs out before mouse is at goal

**Maze**

This is the heaviest part of the code. Basically the maze will be of a certain size with grid made of rooms and walls. The maze starting point will be defined per level. The end point will be wherever the last room the maze visits. The maze will randomly choose to go either to the room either right, left, up or down from the room it is currently in. The maze will not choose a room that is out of bounds or rooms it has already visited. It will then save in a list every room it visited until it reaches a room where all rooms around it have been visited. It will then go back through the list of saved rooms and decide if it can choose to go to a different room.

The visit room list should by the end only have rooms that are connected to the start and the goal. This list will be saved as solution. As maze removes rooms from the solution list it will store them in a list of connected rooms that are not part of solution. If two rooms are connected maze

will not draw a wall between the rooms.

**Variables**

* cell width
* cell height
* grid stores cells x and y coords
* wall width
* stack
* solution dictionary

**Functions**

* maze(): randomly choose a cell next to current cell and stores cell position
* currentCell(): returns x and y of current room
* update display erasing walls
* update display show solution
* target(): show solution path

**ScoreBoard**

Displays current score, high score, and current level.

* High score
* Current score
* Current Level

**Functions**

* render(): shows current score, level, and high score
* gameOver(): game over screen showing last level current score, high score
* calcScore(current room, level, time remaining): takes level, room, and time remaining to calculate score. If time runs out time remaining adds nothing.