UML for maze_generator

maze_generator

- GenMaze maze; → Holds the Maze
- character dude; → Holds the Character
- int level; → Tracks which level you are on
- int levelsPassed; → Tracks how many levels have been passed
- boolean chosen; → Boolean that shows whether or not you have completed level three
- String liveStr; → To parse the integer into a String.
- int time; → Tracks the time
- int wait; → Time per round (increases)
- int clockCenterX; → X-coordinate of the center of the clock.
- int clockCenterY; → Y-coordinate of the center of the clock.
- float angleIncrement; → Float for angular movement of the timer.
- float timeAngle; → Float for angular movement of the timer.
- int radius = 16; → Radius of the clock.
- Void setup() → Sets up the size of the world and state variables.
- Void draw() → Displays the Maze
- Void keyPressed() → WASD and Up, Down, Left, Right movement
- Void drawClock() → Moves the clock based on the time.
- Void resetClock() → Resets the clock when a level is repeated or begun

Character

- color c; → Color
- int lives; → Number of lives
- int xpos; → X-coordinate
- int ypos; → Y-coordinate
- int xperm; → Original xpos
- int yperm; → Original ypos
- int arrx; → x-coordinate of the maze array
- int arry; → y-coordinate of the maze array
- Void printChar() → Prints the circle
- Void up() → Upward movement
- Void down() → Downward movement
- Void left() → Leftward movement
- Void right() → Rightward movement
- Void validDirection(int i, cell[][] maze) → Checks if the next square is available
- int getArrX() → Accessor Method for arrX value

- int getArrY() → Accessor Method for arrY value
- int getX() → Accessor Method for x-coordinate
- int getY() → Accessor Method for y-coordinate
- int getLives() → Accessor Method for lives
- Void reset() → Returns to my old position
- Void die() → Lose a life and reset

Cell

- protected color c
  - The color of the cell, which plays a role in what each cell represents separately. For example, a green cell would denote the path while the black cells would denote a wall.
- protected boolean unvisited
  - Marks whether or not a cell was visited in the maze generation algorithm.
- protected int x
  - X-coordinate of the center of the cell.
- protected int y
  - Y-coordinate of the center of the cell.
- boolean dropped
  - Whether or not the cell has been dropped
- color getColor()
  - Accessor method for the color of the cell.
- void setColor(color col)
  - Mutator method for the color of the cell.
- int getX()
  - Accessor method for the x-position.
- int getY()
  - Accessor method for the y-position.
- void visit()
  - Changes the color of the cell to blue, which shows that it is in the midst of being a part of the maze generation.
  - Sets the unvisited boolean to false, because it is now visited.
- void backTrack()
  - Changes the color of the cell to green, in order to confirm that it is part of the finalized maze.
- boolean unvisited()
  - Accessor method for the unvisited boolean.
- Boolean equals(cell c)
  - Checks for equality of two different cells based on each of their colors.
- Void drop()
  - Sets dropped to true

- Void undrop()
    - Sets dropped to false
- Void displayCell()
    - Method that prints out the cell.

Wall extends Cell

- x and y are set to inputted values in the constructor.
- The color is set to black.
- The wall starts and remains unvisited.

interface GenMaze

- void generate() → to generate the maze
- void displayMaze() → to print each cell in the maze
- boolean generated() → to return whether or not a maze is generated yet
- void makeExit() → Uses the lower right corner to find a randomly-generated escape square.
- cell[][] getMaze() → accessor method for the maze
- cell getExit() → accessor method for the exit square

MazeDepth implements GenMaze

- import java.util.Stack → Uses a stack in the algorithm for maze generation.
- cell[][] maze; → holds the maze
- Stack<cell> path; → holds the current path
- cell current; → current cell
- character dude; → utilized character
- int x; → current xcor within array [y][x]
- int y; → current ycor within array [y][x]
- int newX; → used for transitions with midX
- int newY; → used for transitions with midY
- int midX; → used for walls in between cells
- int midY; → used for walls in between cells
- int numRow; → number of rows
- int numCol; → number of columns
- cell exit; → exit square

====================*FROM THE INTERFACE*=======================

- void generate() → to generate the maze
- void displayMaze() → to print each cell in the maze
- boolean generated() → to return whether or not a maze is generated yet

- void makeExit() → Uses the lower right corner to find a randomly-generated escape square.
- cell[][] getMaze() → accessor method for the maze
- cell getExit() → accessor method for the exit square

==

- boolean hasNeighbors() → Checks whether or not a cell has unvisited neighbors.

MazeEllers implements GenMaze

- int row → number of rows
- int col → number of columns
- int rowMaze → row of the maze you are currently on
- int colMaze → column of the maze you are currently on
- boolean generated → whether or not the maze is generated
- cell exit → exit square
- cell[][] Maze → contains the maze
- int newSetVal → tracks the subdivisions
- int rowGen → to mark the set of rows that should be used for the actual maze
- void randomlyJoinHorizontal(int r) → Randomly subdivided rows into sections
- void joinVertical(int r) → Joins two squares that are one wall apart together
- void joinBottom() → Joins the bottom row to this maze

=================*FROM THE INTERFACE*===================

- void generate() → to generate the maze
- void displayMaze() → to print each cell in the maze
- boolean generated() → to return whether or not a maze is generated yet
- void makeExit() → Uses the lower right corner to find a randomly-generated escape square.
- cell[][] getMaze() → accessor method for the maze
- cell getExit() → accessor method for the exit square

==

- void join(cell cellA, cell cellB) → Joins two cells together
- void dropSet(color c) → drop all of the cells of a certain color
- void undropSet() → undrop all cells
- void turnGreen() → Turns all non-wall green

MazePrim implements GenMaze

- import java.util.ArrayList
- cell[][] maze → holds our maze
- ArrayList<cell> path; → holds the current path we have taken
- cell current; → current cell we are on in maze
- character dude → utilized character
- int x; → current xcor within array [y][x]
- int y; → current ycor within array [y][x]
- int newX; → used for transitions with midX
- int newY; → used for transitions with midY

- int midX; → used for walls in between cells
- int midY; → used for walls in between cells
- int numRow → number of rows
- int numCol → number of columns
- cell exit → exit square
- int level → tracks the number of levels
- boolean hasNeighbors() → check if any neighbors have not been travelled to
- void getNext() → returns next logical path

===================*FROM THE INTERFACE*=====================

- void generate() → to generate the maze
- void displayMaze() → to print each cell in the maze
- boolean generated() → to return whether or not a maze is generated yet
- void makeExit() → Uses the lower right corner to find a randomly-generated escape square.
- cell[][] getMaze() → accessor method for the maze
- cell getExit() → accessor method for the exit square

==