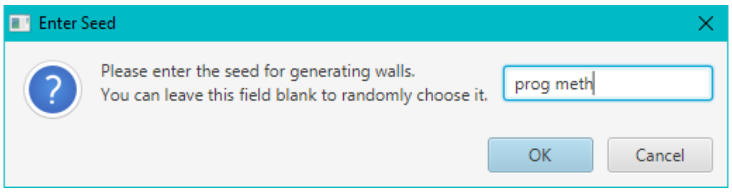
# ->THRU

## Introduction

This project is a simple avoid the obstacle game. But what is not simple is that it has  a 3D appearance and gameplay while every drawing is done on a 2D canvas which dynamically scales with the window. This project is inspired by parallax scrolling and how perspective works in general: the farther the object is, the smaller it appears and moves.

## How to play

At the title screen, left-clicking anywhere in the window to start the game. After clicking, the “Enter Seed” dialog will show up. Any strings could be entered in its text field as a seed for wall generation. It’s also possible to leave the text field blank in order to randomly generate it. Please note that same set of walls will appear if they have the same seed.



Please enter the seed for generating walls.
You can leave this field blank to randomly choose it.

prog meth

*Figure 1: “Enter Seed” dialog*

In the game, use the mouse to move the position of black circle and try to avoid having it collide with gray walls (they will turn red if they are going to hit the black circle). There are 5 different colored cells: red, green, blue, cyan and purple. Any of which can be passed through for an extra score. The game is over when black circle collides with the wall.

After the game is over, it will go to results screen. This screen shows a history of all passed walls and cells in this playthrough and a final score. Click “retry” to play the game again with the same seed or click “back to title” to go back to the title screen.

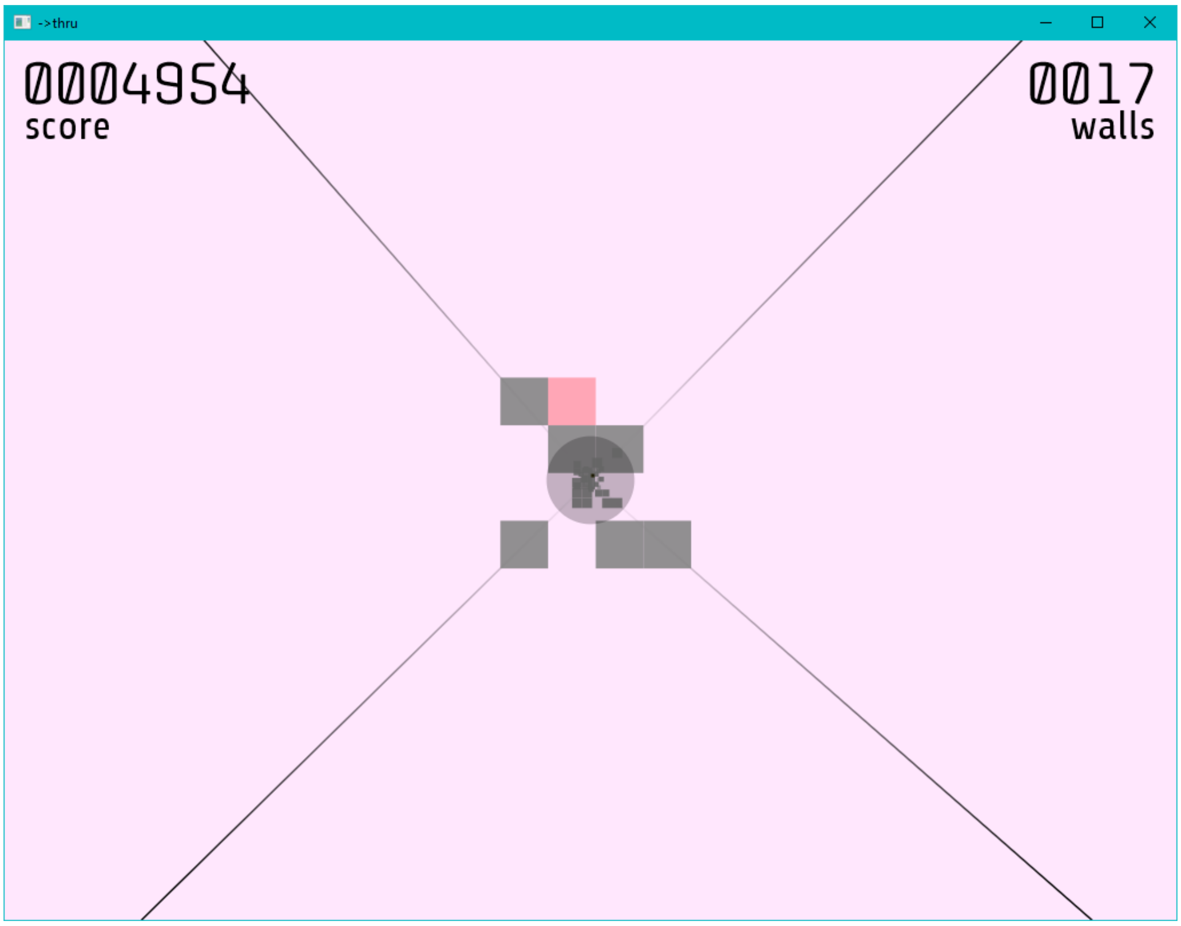


*Figure 2: In-game screen*

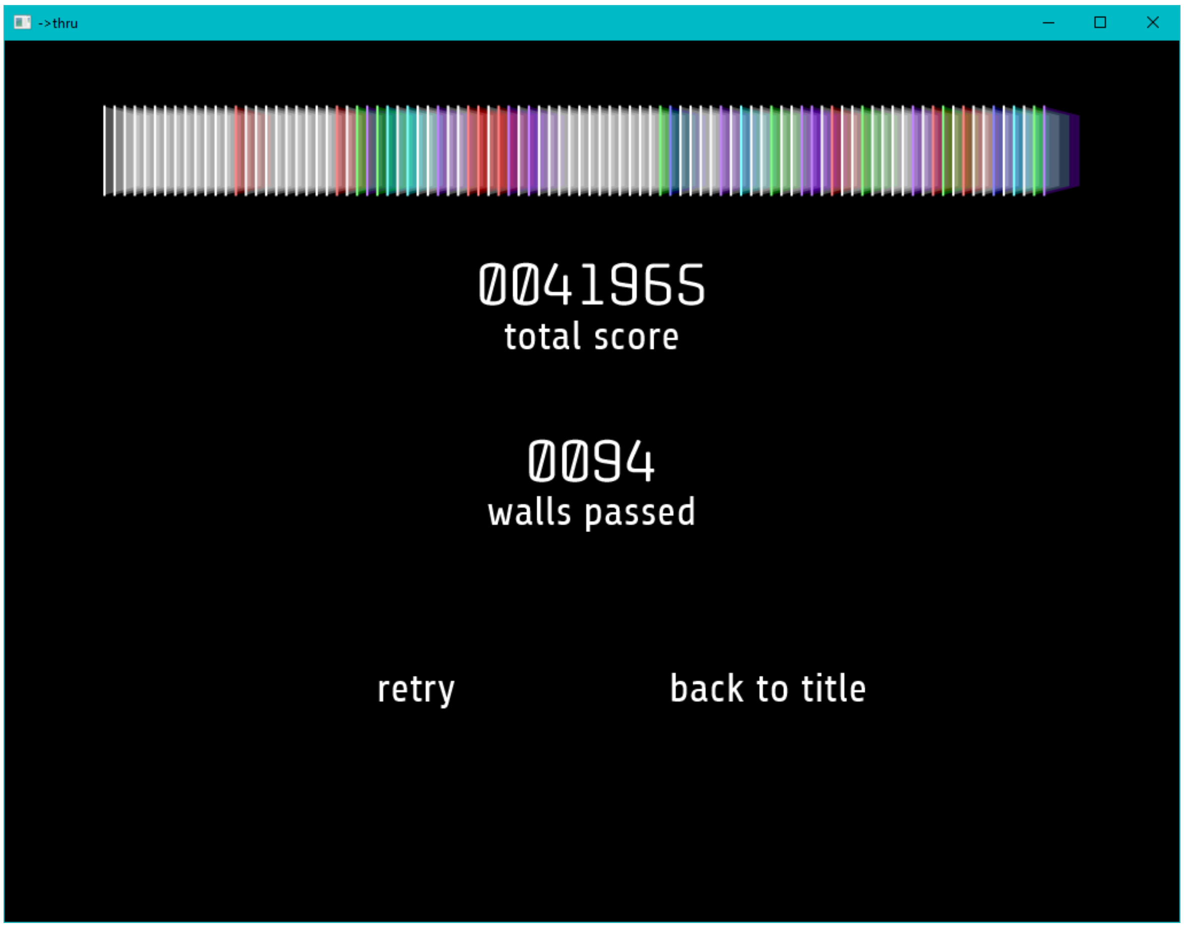


*Figure 3: Results screen*

## Implementation Detail

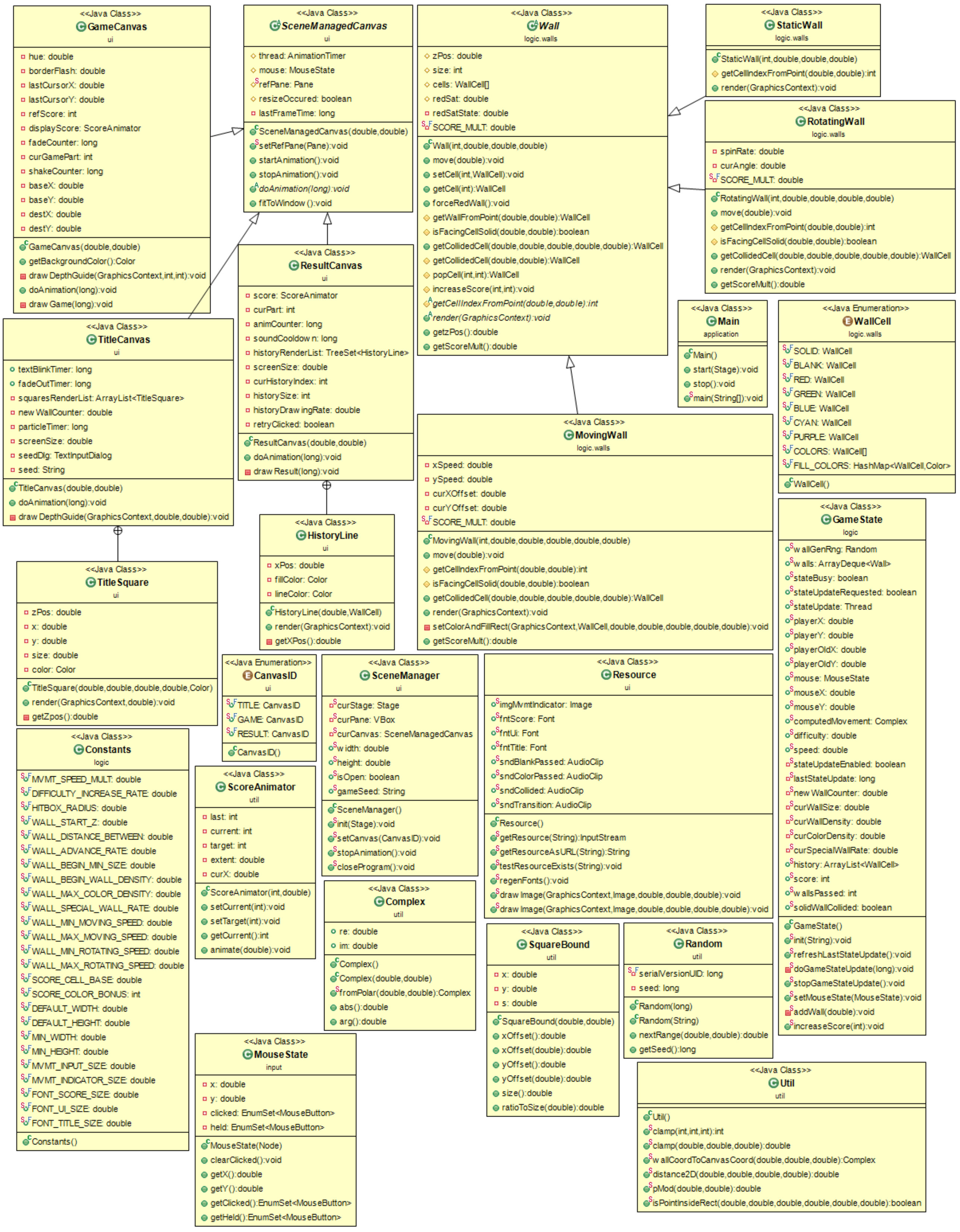


*Figure 4: UML diagram of this application*

# 1. Package application

## 1.1. Class Main extends Application

### *Method*

| | |
|---|---|
| `+ void start()` | Initialize scene manager and set the canvas to title screen canvas |
| `+ void stop()` | Tells scene manager to stop all animations and threads. This is called when the program is closed. |
| `+ void main()` | Entry point of this application. |

# 2. Package input

## 2.1. Class MouseState

This class handles every mouse-related operations such as moving and clicking.

### *Field*

| | |
|---|---|
| `- double x` | X position of mouse related to the top-left corner of the node. |
| `- double y` | Y position of mouse related to the top-left corner of the node. |
| `- EnumSet<MouseButton> clicked` | Set of every clicked mouse buttons. Get cleared every frame. |
| `- EnumSet<MouseButton> held` | Set of every held mouse buttons. |

### *Constructor*

| | |
|---|---|
| `+ MouseState(Node node)` | Initialize x and y to the center of screen, clicked and held to an empty set. Then attach mouse events to node. |

### *Method*

| | |
|---|---|
| `+ clearClicked()` | Clears clicked set. |
| + Getter methods for every field | |

# 3. Package logic

## 3.1. Class Constants

This class contains every important constants used in this application.

### *Field*

| | |
|---|---|
| `+ double MVMT_SPEED_MULT` | How fast the player moves. |
| `+ double DIFFICULTY_INCREASE_RATE` | How fast the difficulty increases. |
| `+ double HITBOX_RADIUS` | Radius of a hitbox, which is currently the radius of black circle. |
| `+ double WALL_START_Z` | Starting Z position of the wall. |

| | |
|---|---|
| + double WALL_DISTANCE_BETWEEN | Distance between walls. |
| + double WALL_ADVANCE_RATE | How fast the wall moves. |
| + double WALL_BEGIN_MIN_SIZE | Minimum wall size at the beginning. |
| + double WALL_BEGIN_WALL_DENSITY | Amount of gray solid cells in one wall. |
| + double WALL_MAX_COLOR_DENSITY | Maximum color intensity. |
| + double WALL_SPECIAL_WALL_RATE | How often moving wall and rotating wall appears. |
| + double WALL_MIN_MOVING_SPEED | Moving wall’s minimum moving speed. |
| + double WALL_MAX_MOVING_SPEED | Moving wall’s maximum moving speed. |
| + double WALL_MIN_ROTATING_SPEED | Rotating wall’s minimum rotating speed. |
| + double WALL_MAX_ROTATING_SPEED | Rotating wall’s maximum rotating speed. |
| + double SCORE_CELL_BASE | Base score for passing through the wall’s hole. |
| + int SCORE_COLOR_BONUS | Bonus score for passing through colored cells. |
| + double DEFAULT_WIDTH | Default screen width in pixels. |
| + double DEFAULT_HEIGHT | Default screen height in pixels. |
| + double MIN_WIDTH | Minimum screen width in pixels. |
| + double MIN_HEIGHT | Minimum screen height in pixels. |
| + double MVMT_INPUT_SIZE | Size of movement area. |
| + double MVMT_INDICATOR_SIZE | Size of movement arrow’s picture. |
| + double FONT_SCORE_SIZE | Size of score font. |
| + double FONT_UI_SIZE | Size of UI font. |
| + double FONT_TITLE_SIZE | Size of title font. |

## 3.2. Class GameState

Most of the game’s logic are in this class, updating in a separate thread from the main thread.

### *Field*

| | |
|---|---|
| + Random wallGenRng | Current random generator used for generating walls. |
| + ArrayDeque<Wall> walls | List of walls currently present in the game. |
| + boolean stateBusy | Both update thread and canvas thread will check this to see if each other is currently updating and wait for it to avoid using shared resources such as wall data at the same time. |
| + boolean stateUpdateRequested | Update thread of this class will not do a state update until this is set to true. Also, it will set this to false when it finishes updating. |
| + Thread stateUpdate | Update thread of this class. |
| + double playerX | Current player's X position in wall coordinate. 0 means at the left edge of the wall and 1 means at the right edge. |
| + double playerY | Current player's Y position in wall coordinate. 0 means at the top edge of the wall and 1 means at the bottom edge. |
| + double playerOldX | Player’s X position in the last frame. |

| | |
|---|---|
| `+ double playerOldY` | Player’s Y position in the last frame. |
| `+ MouseState mouse` | Mouse state to get mouse cursor position from. |
| `+ double mouseX` | X coordinate of mouse cursor. |
| `+ double mouseY` | Y coordinate of mouse cursor. |
| `+ Complex computedMovement` | Converted movement vector. |
| `+ double difficulty` | Current difficulty of the game. |
| `+ double speed` | Current speed of the game. |
| `- boolean stateUpdateEnabled` | Flag to tell the update thread to wait for a state update request. If this is set to false, it will terminate. |
| `- long lastStateUpdate` | System time of when the last state update happened. |
| `- double newWallCounter` | Counter for when to generate and put in a new wall. |
| `- double curWallSize` | Current wall size to generate. |
| `- double curWallDensity` | Current wall density to generate. |
| `- double curColorDensity` | Current colored cell density to generate. |
| `- double curSpecialWallRate` | Current probability of special wall to be generated. |
| `+ ArrayList<WallCell> history` | List of all passed cells in this playthrough. |
| `+ int score` | Current score of the game. |
| `+ int wallsPassed` | Number of passed cells in this playthrough. |
| `+ boolean solidWallCollided` | Flag to tell when solid wall is collided. |

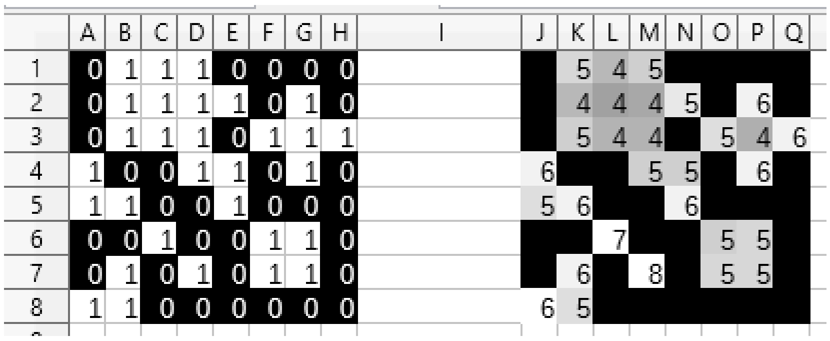
## Method

| | |
|---|---|
| `+ void init(String levelSeed)` | Initialize the game state by doing the following:<br>• Create a new random generator with the given seed.<br>• Set the list of walls to an empty list.<br>• Set the player's position and mouse position to the center.<br>• Set the difficulty and score to 0.<br>• Set the current wall generation parameters to the beginning.<br>• Create and start a new game state update thread. |
| `+ void refreshLastStateUpdate()` | Set the last state update time to when this method is called. |
| `- void doGameStateUpdate(long deltaT)` | This method is called in the update thread when there is a game state update request. This will do the following:<br>• Set the speed to max(difficulty, 1)<br>• Get the mouse position from the reference mouse state and calculate the movement vector.<br>• Update player's position from the resulting movement vector.<br>• Update wall generation parameters alongside the difficulty.<br>• Update newWallCounter and add a new wall if it runs out.<br>• Get the first wall from the walls list and check if it goes past and collides with the player. |

| | |
|---|---|
| | ○ If blank cell (hole) is passed, play blank cell passed sound effect and add it to the history.<br>○ If colored cell is passed, play color cell passed sound effect, add it to the library and increase the score with the color bonus score.<br>○ If collided with solid wall, play solid wall collided sound effect and set the collided flag to true.<br>• Move the first wall, if the resulting Z position is less than 0 then remove it from the list.<br>• Move the rest of the walls. |
| `+ void stopGameStateUpdate()` | Stop the game state update thread. |
| `+ void setMouseState(MouseState m)` | Set the reference mouse state to m. |
| `- void addWall(double zOffset)` | Add a new wall to the list. The type and attributes of a wall will be generated as follows:<br>• The size is randomly generated from curWallSize to curWallSize + 2.<br>• Generate the random number, if it is greater than curSpecialWallRate then it'll be a static wall. Else it'll 50% be a moving wall or 50% be a rotating wall.<br>• Moving / rotating speed is randomly generated between its respective min and max speed. |
| `+ increaseScore(int sc)` | Increase the current score by sc. sc can be negative but the resulting score will never go below 0. |

## 4. Package logic.walls

### 4.1. Class *Wall*



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | | | 5 | 4 | 5 | | | | |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | | | 4 | 4 | 4 | 5 | | 6 | |
| 3 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | | 5 | 4 | 4 | | 5 | 4 | 6 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | | 6 | | | 5 | 5 | | 6 | |
| 5 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | | 5 | 6 | | | 6 | | | |
| 6 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | | | | 7 | | | 5 | 5 | |
| 7 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | | | 6 | | 8 | | 5 | 5 | |
| 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | 6 | 5 | | | | | | |

*Figure 5: An illustration of how scoring works*

Base class for all 3 types of wall which has most of fields and methods for dealing with the wall cells.

#### *Field*

| | |
|---|---|
| `# double zPos` | Current Z position of this wall. |
| `# int size` | Current size in rows and columns of this wall. |
| `# WallCell[] cells` | List of wall cells in this wall. |
| `# double redSat` | Current displaying redness of solid cells in this wall. |

| | |
|---|---|
| `- double redSatState` | |
| `- double SCORE_MULT` | This wall’s score multiplier, defaults to 1. |

## Constructor

| | |
|---|---|
| `+ Wall(int size, double wallDensity, double colorDensity, double zOffset)` | Initialize and generate wall cells by doing the following:<br>• Set the Z position to WALL_START_Z + zOffset.<br>• Create an empty array with a size of size*size for wall cells list.<br>• For each of wall cell in the list:<br>　○ Generate a random number, if it is less than wallDensity then it will be a solid wall cell.<br>　○ If it is not, generate a new random number, if it is lest than colorDensity, then it will be a colored cell. Else, it will be a blank cell. |

## Method

| | |
|---|---|
| `+ void move(double deltaZ)` | Move this wall’s Z position by deltaZ. |
| `+ void setCell(int index, WallCell value)` | Set the wall cell at the given index to the given wall cell. |
| `+ WallCell getCell(int index)` | Get the wall cell at the given index. |
| `+ void forceRedWall()` | Immediately set the solid wall color to the most red. |
| `# WallCell getWallFromPoint(double x, double y)` | Get the wall cell at the given player position by translating it into the list index. |
| `# boolean isFacingCellSolid(double playerX, double playerY)` | Determine if the wall at the given player position a solid wall. |
| `+ WallCell getCollidedCell(double deltaZ, double oldX, double oldY, double newX, double newY)` | Calculate if the wall is going to move past a Z position of 1 where the player is on. If it does, then return the cell that the player will move past and collide. Else, return null. |
| `# WallCell getCollidedCell(double cX, double cY)` | |
| `# WallCell popCell(int x, int y)` | Get the wall cell at the index of y*size+x. Then set the wall cell at that index to a blank cell. |
| `# void increaseScore(int x, int y)` | Calculate the score get from passing through the wall’s blank or colored cell based on this formula:<br><br>SCORE_CELL_BASE * getScoreMult() * size * max(difficulty, 1) / sqrt(hole size)<br><br>For the hole size, it is calculated by counting the number of non-solid cells of and surronding the target cell at x, y coordinate and add them in center cell + adjacent cells + (corner cells) / 4. |

| | |
|---|---|
| | Figure 5 provide a basic illustration of how the scoring work. |
| *# int getCellIndexFromPoint(double x, double y)* | Translate wall coordinate into the list index of wall cells. |
| *+ void render(GraphicsContext gc)* | Abstract method to draw the wall with a respect to its Z position and player's position. |
| + Getter methods for `zPos`, `SCORE_MULT` | |

## 4.2. Class StaticWall extends Wall

Basic, stationary wall.

### *Constructor*

| | |
|---|---|
| `+ StaticWall(int size, double wallDensity, double colorDensity, double zOffset)` | Same as Wall's constructor. |

### *Method*

| | |
|---|---|
| `# int getCellIndexFromPoint(double x, double y)` | Translate player's position into the list index of wall cells by directly turning X and Y coordinate into list index. Any values outside the wall coordinate's 0 to 1 range will be clamped inside. |
| `+ void render(GraphicsContext gc)` | |

## 4.3. Class MovingWall extends Wall

Wall that has moving cells inside of it.

### *Field*

| | |
|---|---|
| `- double xSpeed` | X speed of moving cells. |
| `- double ySpeed` | Y speed of moving cells. |
| `- double curXOffset` | Current X offset of moving cells. |
| `- double curYOffset` | Current Y offset of moving cells. |
| <u>`- double SCORE_MULT`</u> | This wall's score multiplier. Sets to 1.5. |

### *Constructor*

| | |
|---|---|
| `MovingWall(int size, double xSpeed, double ySpeed, double wallDensity, double colorDensity, double zOffset)` | Same as Wall's constructor but will also initialize those added fields. |

### Method

| | |
|---|---|
| `+ void move(double deltaZ)` | Change curXOffset and curYOffset by its respective speed then move this wall's Z position by deltaZ. |
| `# int getCellIndexFromPoint(double x, double y)` | Same as StaticWall's one but any values outside the range will be wrapped around instead. |
| `# boolean isFacingCellSolid(double playerX, double playerY)` | Overridden in order to also account for wrapped around cells. |
| `+ WallCell getCollidedCell(double deltaZ, double oldX, double oldY, double newX, double newY)` | Translate player's position to wall coordinate by subtracting curXOffset/curYOffset from player's position. Then do the calculation like the one in Wall but also account for wrapped around cells. |
| `+ void render(GraphicsContext gc)` | |
| `- void setColorAndFillRect(GraphicsContext gc, WallCell cell, double opacity, double x, double y, double w, double h)` | Extracted instead of included like in StaticWall's one due to more squares have to been drawn for wrapped around cells. |
| `+ double getScoreMult()` | Overridden to return this wall's value instead. |

## 4.4. Class RotatingWall extends Wall

### Field

| | |
|---|---|
| `- double spinRate` | Current rotation speed. |
| `- double curAngle` | Current angle. |
| <u>`- double SCORE_MULT`</u> | This wall's score multiplier. Sets to 1.5. |

### Constructor

| | |
|---|---|
| `+ RotatingWall(int size, double spinRate, double wallDensity, double colorDensity, double zOffset)` | Same as Wall's constructor but will also initialize those added fields. |

### Method

| | |
|---|---|
| `+ void move(double deltaZ)` | Change the current angle by the spin rate then move this wall's Z position by deltaZ. |
| `# int` | Same as StaticWall's one but any values outside the range will return -1 and get considered as a solid wall when this is called in |

| | |
|---|---|
| getCellIndexFromPoint(double x, double y) | getWallFromPoint. |
| # boolean isFacingCellSolid(double playerX, double playerY) | Overridden in order to rotate the player's position before normally continuing to Wall's method. |
| + WallCell getCollidedCell(double deltaZ, double oldX, double oldY, double newX, double newY) | Translate player's position to wall coordinate by rotating player's position by curAngle. Then do the calculation like the one in Wall. |
| + void render(GraphicsContext gc) | |
| + double getScoreMult() | Overridden to return this wall's value instead. |

### 4.5. Enum WallCell

#### *Enum Constant*

| | |
|---|---|
| SOLID | Solid wall cell. |
| BLANK | Blank cell (hole). |
| RED | Red cell. |
| GREEN | Green cell. |
| BLUE | Blue cell. |
| CYAN | Cyan cell. |
| PURPLE | Purple cell. |

#### *Field*

| | |
|---|---|
| + WallCell[] COLORS | List of all colored cells for wall generation. |
| + HashMap<WallCell, Color> FILL_COLORS | List of fill colors for each of wall cell. |

#### *Method*

| | |
|---|---|
| - {...} | Initialize and add each entry of color to FILL_COLORS. |

## 5. Package ui

### 5.1. Class SceneManager

Class for handling switching between the screens and cross-canvas variables.

#### *Field*

| | |
|---|---|
| - Stage curStage | Current stage. |
| - Vbox curPane | Current pane. |
| - SceneManagedCanvas | Current screen. |

| | |
|---|---|
| curCanvas | |
| + double width | Current width of this window. |
| + double height | Current height of this window. |
| + boolean isOpen | Is the current scene is open? |
| + String gameSeed | Current game seed. |

### *Method*

| | |
|---|---|
| + void init(Stage stage) | Initialize the scene manager and set the current stage to the given stage. Also create a new pane and set it as a reference pane to SceneManagedCanvas. |
| + void setCanvas(CanvasID id) | Changes the current screen to the given canvas ID. Also initialize game state if it is in-game canvas. |
| + void stopAnimation() | Stop the animation in the current canvas. |
| + void closeProgram() | Stop the animation and close this application. |

## 5.2. Class *SceneManagedCanvas* extends Canvas

Base class for every canvases in this game. It sets up and handles mouse input, animation timer and resizing.

### *Field*

| | |
|---|---|
| # AnimationTimer thread | Timer for animation and updating. |
| # MouseState mouse | Mouse state for this class. |
| # Pane refPane | Reference pane to get the window size from. |
| # boolean resizeOccured | Set to true when resizing has occurred. |
| - long lastFrameTime | System time of when the last frame happened. |

### *Constructor*

| | |
|---|---|
| + SceneManagedCanvas(double width, double height) | Initialize this class and mouse state and tell the resource to regenerate fonts. |

### *Method*

| | |
|---|---|
| + void setRefPane(Pane pane) | Set the reference pane to the given pane. |
| + void startAnimation() | Start the animation timer. Which will check the window size and clear clicked mouse buttons each frame. |
| + void stopAnimation() | Stop the animation timer. |
| + *void doAnimation(long deltaT)* | Abstract method which gets called every frame. |
| + void fitToWindow() | Checks the reference pane if its size is changed as a result from resizing the window. If it does, update scene manager’s width and height, set resizeOccured to true and tell the resource to regenerate fonts. |

## 5.3. Class TitleCanvas extends SceneManagedCanvas

Canvas for title screen.

### Field

| | |
|---|---|
| `+ long textBlinkTimer` | Timer for blinking “left click to begin” message. |
| `+ long fadeOutTimer` | Counter for fade. |
| `- ArrayList<TitleSquare> squaresRenderList` | List of squares to be rendered. |
| `- double newWallCounter` | Counter for when to put in a new wall. |
| `- long particleTimer` | Counter for when to put in small squares. |
| `- double screenSize` | Current computed screen size for scaling the graphics. |
| `- TextInputDialog seedDlg` | “Enter Seed” dialog. |
| `- String seed` | Seed obtained from the dialog. |

### Constructor

| | |
|---|---|
| `+ TitleCanvas(double width, double height)` | Initialize canvas and fields with the given width and height. Also set the current state to 0. |

### Method

| | |
|---|---|
| `+ void doAnimation(long deltaT)` | Update and draw this canvas, which will do the following:<br>• If there is a left click, show the dialog. Then set the seed to the result, play transition sound effect and begin fading out if the result is not null.<br>• Update blinking timer.<br>• Update wall and particle timer. If timer runs out, then generate and add a new one to the list.<br>• Fill the entire screen with a semi-transparent black background (to simulate a CRT fadeout effect) if the screen is not resized in this frame.<br>• Draw all of 4 depth guides.<br>• Sort every squares by Z position so the farthest square get drawn first.<br>• Draw every squares in squaresRenderList by calling render() in each of them.<br>• Draw the title text and “left click to begin” when appropriate.<br>• If fade out is enabled, fades this screen out to white and then change to in-game screen. |
| `- void drawDepthGuide(GraphicsContext gc, double x, double y)` | Draw a depth guide line from the center to the given x, y corner. |

## 5.4. Class TitleSquare (Inner class of TitleCanvas)

### Field

| | |
|---|---|
| `- double zPos` | Current Z position of this square. |

| | |
|---|---|
| `- double x` | Current X position of this square. |
| `- double y` | Current Y position of this square. |
| `- double size` | Size of this square, 1 is as large as a wall in the game screen. |
| `- Color color` | Color of this square's outline. |

### Constructor

| | |
|---|---|
| `+ TitleSquare(double x, double y, double size, double zOffset, Color color)` | Initialize this square. zOffset is an offset from WALL_START_Z. |

### Method

| | |
|---|---|
| `+ void render(GraphicsContext gc, double deltaZ)` | Render this passed cell into gc and move this square's Z position by deltaZ. |
| - Getter method for zPos | |

## 5.5. Class GameCanvas extends SceneManagedCanvas

Canvas for in-game screen (Figure 2).

### Field

| | |
|---|---|
| `- double hue` | Current background color's hue. |
| `- double borderFlash` | Current border's opacity. |
| `- double lastCursorX` | Mouse cursor's X position in the last frame. |
| `- double lastCursorY` | Mouse cursor's Y position in the last frame. |
| `- int refScore` | Reference score to check with GameState's score. |
| `- ScoreAnimator displayScore` | Score animator for total score in the top-left corner. |
| `- long fadeCounter` | Counter for fades and animations. |
| `- int curGamePart` | Current state of this screen. |
| `- long shakeCounter` | Counter for shaking animation cycle. |
| `- double baseX` | Shaking animation's base X position. |
| `- double baseY` | Shaking animation's base Y position. |
| `- double destX` | Shaking animation's destination X position. |
| `- double destY` | Shaking animation's destination Y position. |

### Constructor

| | |
|---|---|
| `+ GameCanvas(double width, double height)` | Initialize canvas and fields with the given width and height. Also set the current state to 0. |

### Method

| | |
|---|---|
| `+ Color getBackgroundColor()` | Get this screen's current background color. |
| `- void` | Draw a depth guide line from the center to the given x, y corner. |

| | |
|---|---|
| `drawDepthGuide(GraphicsContext gc, int x, int y)` | |
| `+ void doAnimation(long deltaT)` | Update the canvas by calling `drawGame(deltaT)` and then do the following for each curGamePart state:<br>• For curPart of 0: fade in this screen from white. Also reset GameState's last state update time when this part ends.<br>• For curPart of 1: tell GameState to update for this frame.<br>• For curPart of 2: do the shaking animation by manipulating current player position.<br>• For curPart of 3: fade the screen out to white. Then change the screen to results screen. |
| `- void drawGame(long deltaT)` | Draw this canvas, which will do the following:<br>• If GameState is still updating, wait until it finishes.<br>• Check the game state if the solid wall is collided. If it does, stop GameState's update thread and change curGamePart to 2.<br>• Get mouse and movement data from game state.<br>• Fill the entire screen with a background color then update hue.<br>• Draw all of 4 depth guides.<br>• Draw every walls currently present in the game state (except the ones with Z position below 1). Farthest wall gets drawn first.<br>• Draw a movement border then update borderFlash for a flashing effect.<br>• Draw a black circle with a radius of hitbox radius in the middle of the screen.<br>• Draw a wall that has a Z position below 1.<br>• Draw a movement indicator.<br>• Draw a current score, walls passed and texts.<br>• Draw a mouse cursor with a tail. |

## 5.6. Class ResultCanvas extends SceneManagedCanvas

Canvas for results screen (Figure 3).

### *Field*

| | |
|---|---|
| `- ScoreAnimator score` | Score animator for total score. |
| `- int curPart` | Current state of this screen. |
| `- long animCounter` | Counter for fades and animations. |
| `- long soundCooldown` | Cooldown timer for a sound effect to prevent from getting played too much when there is a large number of passed cells. |
| `- TreeSet<HistoryLine> historyRenderList` | Set of passed cells to be rendered. |
| `- double screenSize` | Current computed screen size for scaling the graphics. |
| `- int curHistoryIndex` | Current number of displayed passed cells. |
| `- int historySize` | Total number of passed cells. |
| `- double historyDrawingRate` | How many passed cells to add and display per frame. |

| | |
|---|---|
| `- boolean retryClicked` | Indicate whether “retry” or “back to title” was clicked. |

### *Constructor*

| | |
|---|---|
| `+ ResultCanvas(double width, double height)` | Initialize canvas and fields with the given width and height. Also set the current state to 0. |

### *Method*

| | |
|---|---|
| `+ void doAnimation(long deltaT)` | Update the canvas by calling `drawResult(deltaT)` and then do the following for each curPart state:<br>• For curPart of 0: fade in this screen from white.<br>• For curPart of 1: animate the total score and partially add passed cells from GameState history to historyRenderList and play the sound effect for 3 seconds.<br>• For curPart of 2: wait for mouse input and check if the mouse is clicked in these two button areas to advance to the next state.<br>• For curPart of 3: play transition sound effect and fade out to white. Then change the screen to game screen if “retry” is clicked or title screen if “back to title” is clicked. |
| `- void drawResult(long deltaT)` | Draw this canvas, which will do the following:<br>• If there is a left click during curPart state of 0 and 1, skip all animations and change the state to 2.<br>• Fill the entire screen with black background.<br>• Draw every passed cells in historyRenderList by calling render() in each of them.<br>• Draw the current score and texts. |

## 5.7. Class HistoryLine (Inner class of ResultCanvas)

Class for each passed cells displayed in the top part of results screen.

### *Field*

| | |
|---|---|
| `- double xPos` | X position relative to the start and end of an area for displaying passed cells. |
| `- Color fillColor` | Fill color for this passed cell. |
| `- Color lineColor` | Line color for this passed cell. |

### *Constructor*

| | |
|---|---|
| `+ HistoryLine(double xPos, WallCell cell)` | Initialize this passed cell and set the colors based on a given cell type. |

### *Method*

| | |
|---|---|
| `+ void render(GraphicsContext gc)` | Render this passed cell into gc. |
| - Getter method for xPos | |

## 5.8. Enum CanvasID

### *Enum Constant*

| | |
|---|---|
| `TITLE` | Title screen. |
| `GAME` | Game screen. |
| `RESULT` | Results screen. |

## 5.9. Class Resource

This class contains every resources that needs to be loaded from an external file such as image, fonts and sounds.

### *Field*

| | |
|---|---|
| `+ Image imgMvmtIndicator` | Movement arrow image. |
| `+ Font fntScore` | Font for numbers. |
| `+ Font fntUi` | Font for UI. |
| `+ Font fntTitle` | Font for title screen name. |
| `+ AudioClip sndBlankPassed` | Sound for when passing the wall’s hole. |
| `+ AudioClip sndColorPassed` | Sound for when collecting the colored cell. |
| `+ AudioClip sndCollided` | Sound for when colliding with a solid wall. |
| `+ AudioClip sndTransition` | Sound for a transition between screens. |

### *Method*

| | |
|---|---|
| `- {...}` | Check and load every resources. If there is a missing file, show the alert and close the game. |
| `+ InputStream getResource(String res)` | Load a resource from a given file path, if the file is missing then throw an exception. |
| `+ String getResourceAsURL(String res)` | |
| `+ void testResourceExists(String res)` | Check if the file exists, if the file is missing then throw an exception. |
| `+ void regenFonts()` | Reload every fonts for the new screen size. |
| `+ void drawImage(GraphicsContext gc, Image img, double x, double y, double scale)` | Draw an image into gc with a given scale. x and y position is based on the center of an image instead of top-left corner. |
| `+ void drawImage(GraphicsContext gc, Image img, double x, double y, double sx,` | |

| | |
|---|---|
| double sy) | |

# 6. Package util

## 6.1. Class Util

Class for miscellaneous utility functions.

### *Method*

| | |
|---|---|
| + int clamp(int number, int min, int max) | Limit the number into the range between min and max. If the number is less than min, it will be set to min. If the number is greater than max, it will be set to max. |
| + double clamp(double number, double min, double max) | |
| + Complex wallCoordToCanvasCoord(double x, double y, double z) | Convert the game logic's wall coordinate into canvas coordinate in respect to player's position and z position. Assuming the origin of canvas is translated to the middle of the screen. |
| + double distance2D(double x1, double y1, double x2, double y2) | Calculate the distance between two points in a 2D plane. |
| + double pMod(double a, double b) | Do a % b that always result in a positive number (Euclidean division remainder). Since Java's % operator assumes resulting number's sign from the dividend. |
| + boolean isPointInsideRect(double pointX, double pointY, double rectX, double rectY, double rectW, double rectH) | Check if the given point is in a rectangle. |

## 6.2. Class Complex

Utility class for complex number which can also be used as a 2D vector.

### *Field*

| | |
|---|---|
| + double re | Real part (x value) of the number. |
| + double im | Imaginary part (y value) of the number. |

### *Constructor*

| | |
|---|---|
| + Complex() | Create a complex number of 0. |
| + Complex(double re, double im) | Create a complex number of re+im*i. |

### *Method*

| | |
|---|---|
| + Complex fromPolar(double r, double phi) | Create a complex number from a polar representation r<phi. |
| + double abs() | Returns the absolute value of this number. |

| | |
|---|---|
| `+ double arg()` | Return the argument (phase) of this number. |

## 6.3. Class Random extends java.util.Random

### *Field*

| | |
|---|---|
| `- long seed` | Randomization seed. |

### *Constructor*

| | |
|---|---|
| `+ Random(long seed)` | Equivalent to `new java.util.Random(seed)` |
| `+ Random(String seed)` | Creates a new random number generator using a 64-bit hash of string as a seed. If the string is empty, it will use the string representation of System.currentTimeMillis() instead. |

### *Method*

| | |
|---|---|
| `+ double nextRange(double a, double b)` | Randomly generates a number in the range between a (inclusive) and b (exclusive). |
| + Getter method for seed | |

## 6.4. Class ScoreAnimator

Utility class for animating a score. Amount changed in each update is based on an exponential curve to make it change fast and then become slow afterwards.

### *Field*

| | |
|---|---|
| `- int last` | Initial score at x = 0. |
| `- int current` | Current score at x = curX. |
| `- int target` | Target score at x = +∞. |
| `- double extent` | x value to set current to target. |
| `- double curX` | Current value of x. |

### *Constructor*

| | |
|---|---|
| `+ ScoreAnimator(int init, double extent)` | Initialize all fields. last, current and target will be set to init and curX will be set to 0. |

### *Method*

| | |
|---|---|
| `+ void setCurrent(int value)` | Set current and last to this value. |
| `+ void setTarget(int value)` | Set a new target and reset x position to 0. last and current will be set to the old target. |
| `+ void animate(double deltaX)` | Update x position by adding it to deltaX and set current to the calculated value. If x position gets past extent, set current to target instead. |
| + Getter method for current | |

## 6.5. Class SquareBound

Utility class for computing a centered square area inside a rectangle area, which is used for scaling the game field and GUI.

### *Field*

| | |
|---|---|
| `- double x` | X offset from top-left corner of rectangle. |
| `- double y` | Y offset from top-left corner of rectangle. |
| `- double s` | Length of square sides. |

### *Constructor*

| | |
|---|---|
| `+ SquareBound(double w, double h)` | Initialize and compute the centered square area.<br>• If w is less than h, use w as a length, else use h.<br>• Calculate the offset from the lop-left corner of rectangle. |

### *Method*

| | |
|---|---|
| `+ double xOffset()` | Returns x + a, or just x if a is not given. |
| `+ double xOffset(double a)` | |
| `+ double yOffset()` | Returns y + a, or just y if a is not given. |
| `+ double yOffset(double a)` | |
| `+ double size()` | Returns s. |
| `+ double ratioToSize(double ratio)` | Returns s * ratio, used when translating game logic's 0 – 1 coordinate for drawing. |