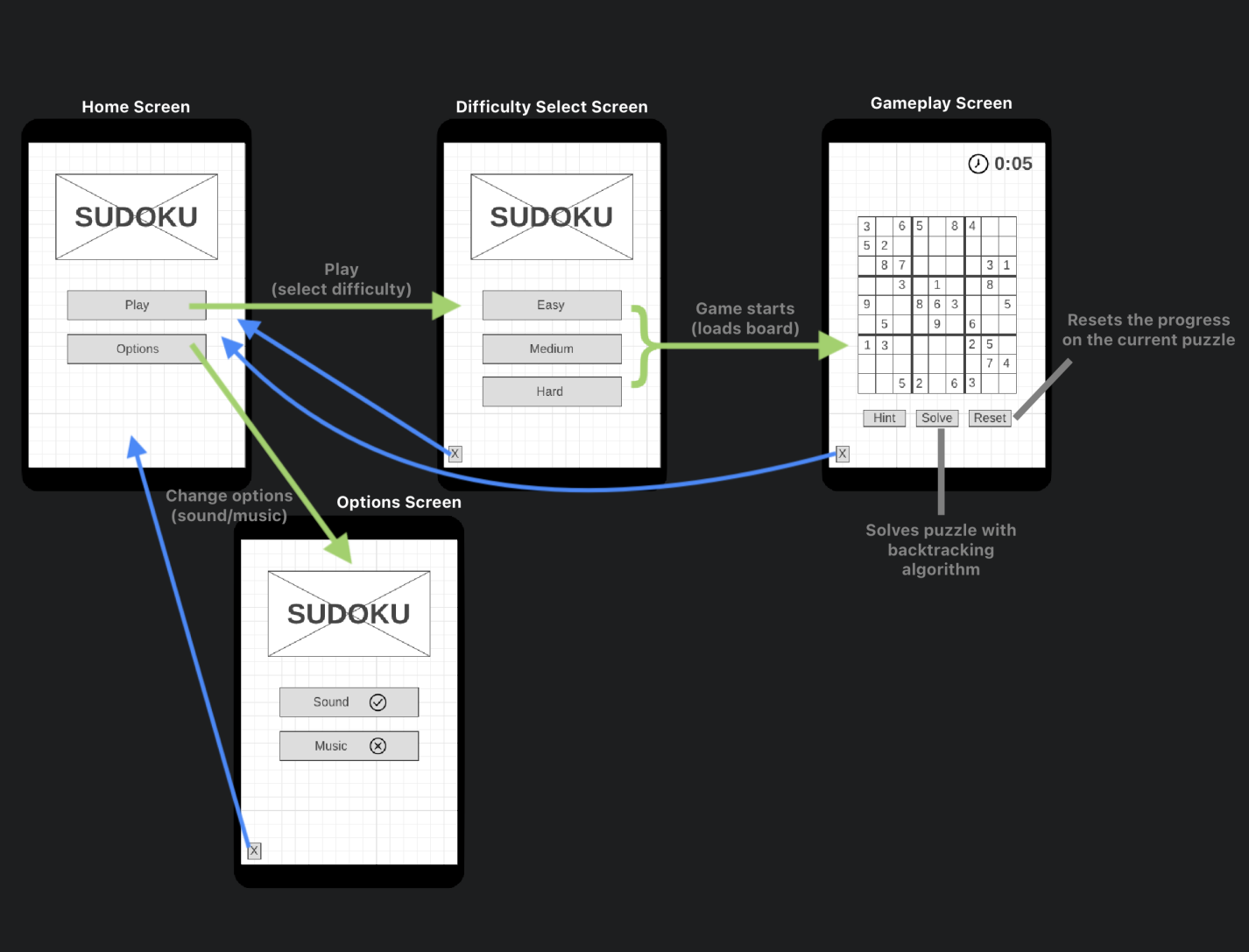
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name of Team:**  Boolean Hooligans | | | | |
| **First name:**  Sheikh Mahdeen | **Last name:**  Islam | **York Email:**  [smi15@my.yorku.ca](mailto:smi15@my.yorku.ca) | **Lecture Section:**  M | **Lab Section:**  1 |
| **First name:**  Michael | **Last name:**  David | **York Email:**  [mikesd@my.yorku.ca](mailto:mikesd@my.yorku.ca) | **Lecture Section:**  N | **Lab Section:**  3 |
| **First name:**  Alexander | **Last name:**  Odorico | **York Email:**  [ao11@my.yorku.ca](mailto:ao11@my.yorku.ca) | **Lecture Section:**  N | **Lab Section:**  4 |
| **First name:**  Charles | **Last name:**  Reed | **York Email:**  [reed226@my.yorku.ca](mailto:reed226@my.yorku.ca) | **Lecture Section:**  N | **Lab Section:**  4 |
| **Project Title:**  Sudoku | | | | |
| **Project Description:**  An android app that features an interface for the user to play a game of sudoku.  Featuring a user friendly menu with various playing difficulties and options.  This app is aimed to heighten one’s skill at the game, through a various of miscellaneous options like hints, timers, and of course good music. You can even visualize how the app solves the puzzle from scratch. Many options are available for puzzles, whether it’d be from a public repertoire, or even handcrafted by our very own puzzle generation algorithm. Our combination of tools makes this an app that anyone would enjoy from beginners trying out a new thing to veterans who have been number crunching for years. | | | | |

**Requirements Definition:**

* - On Launch, the user will hear good background music.
* - The app will display two options, which are play and options.
* - The user can choose to play a game, or go to the options menu.
* - The user can press the back button to go back to the main menu.
* - Upon selecting play, the user will be presented with a menu of difficulty levels. (easy, medium, hard)
* - After choosing a difficulty level, the user will be presented with a grid of 81 cells, a hint button, a solve button, a reset button, and a timer.
* - The timer will record the time it takes the user to solve the puzzle.
* - The user can select a cell to change the value of the cell to any number between 1 to 9.
* - The user can also clear a cell by pressing the clear button.
* - The user can select the hint button to reveal a random empty cell, the solve button to solve the puzzle, and the reset button to reset the puzzle.
* - If the user does a mistake, the app will highlight the cell with the color red, play a sound, and increment the mistakes counter by one.
* - Upon completion of the puzzle, a congratulatory message will be displayed, along with the time it took the user to solve the puzzle.
* - The user can select the options button to go to the options menu.
* - The user can disable the sounds and music through the options menu.
* - The user can enable the “Generate Puzzle” check box to generate a puzzle from scratch instead of loading a pre-generated puzzle.
* - The user can enable the “Visualize Algorithm” check box to visualize the algorithm solving the puzzle, using a backtracking algorithm.

**User Interface:**



**Technologies and Tools:**

- Android Studio

**Libraries/Dependencies:**

- Contexts

- Graphics.Canvas, Graphics.Paint, Graphics.Rect

- Util.AttributeSet, Util.ArrayList

- Views

- Widget.Button

**Design in Pseudocode:**

- Main Activity class -

SudokuBoard gameBoard;

SudokuSolver gameBoardSolver;

private Button solveBTN;

private Button resetBTN;

@Override

protected void onCreate(Bundle savedInstanceState)

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

gameBoard <- findViewById(R.id.SudokuBoard);

gameBoardSolver <- gameBoard.getSolver();

solveBTN <- findViewById(R.id.solveButton);

resetBTN <- findViewById(R.id.resetButton);

Method BTNOnePress:

void BTNOnePress(View view)

gameBoardSolver.setNumberPos(1);

gameBoard.invalidate();

Method BTNTwoPress:

void BTNTwoPress(View view)

gameBoardSolver.setNumberPos(2);

gameBoard.invalidate();

Method BTNThreePress:

void BTNThreePress(View view)

gameBoardSolver.setNumberPos(3);

gameBoard.invalidate();

Method BTNFourPress:

void BTNFourPress(View view)

gameBoardSolver.setNumberPos(4);

gameBoard.invalidate();

Method BTNFivePress:

void BTNFivePress(View view)

gameBoardSolver.setNumberPos(5);

gameBoard.invalidate();

Method BTNSixPress:

void BTNSixPress(View view)

gameBoardSolver.setNumberPos(6);

gameBoard.invalidate();

Method BTNSevenPress:

void BTNSevenPress(View view)

gameBoardSolver.setNumberPos(7);

gameBoard.invalidate();

Method BTNEightPress:

void BTNEightPress(View view)

gameBoardSolver.setNumberPos(8);

gameBoard.invalidate();

Method BTNNinePress:

void BTNNinePress(View view)

gameBoardSolver.setNumberPos(9);

gameBoard.invalidate();

Method solve:

void solve(View view)

gameBoardSolver.getEmptyBoxIndexes();

SolveBoardThread solveBoardThread <- new SolveBoardThread();

new Thread(solveBoardThread).start();

gameBoard.invalidate();

Method reset:

void reset(View view)

gameBoardSolver.resetBoard();

gameBoard.invalidate();

@Override

Method run:

void run()

gameBoardSolver.solve(gameBoard);

- Sudoku Board Class -

int boardColor;

int cellFillColor;

int cellHighlightColor;

int letterColor;

int letterColorSolve;

cellSize;

Paint boardColorPaint = new Paint()

private final Paint cellFillColorPaint = new Paint()

private final Paint cellHighlightColorPaint = new Paint()

Paint letterPaint = new Paint()

Rect letterPaintBounds = new Rect()

SudokuSolver solver = new SudokuSolver()

Method SudokuBoard:

SudokuBoard(Context context, @Nullable AttributeSet attrs)

super(context, attrs);

TypedArray a <- context.getTheme().obtainStyledAttributes(attrs, R.styleable.SudokuBoard,

0, 0);

try

boardColor <- a.getInteger(R.styleable.SudokuBoard\_boardColor, 0);

cellFillColor <- a.getInteger(R.styleable.SudokuBoard\_cellFillColor, 0);

cellHighlightColor <- a.getInteger(R.styleable.SudokuBoard\_cellHighlightColor, 0);

letterColor <- a.getInteger(R.styleable.SudokuBoard\_letterColor, 0);

letterColorSolve <- a.getInteger(R.styleable.SudokuBoard\_letterColorSolve, 0);

finally

a.recycle();

@Override

protected void onMeasure(int width, int height)

super.onMeasure(width, height);

int dimension <- Math.min(this.getMeasuredWidth(), this.getMeasuredHeight()) - 50;

cellSize <- dimension / 9;

setMeasuredDimension(dimension, dimension);

@Override

protected void onDraw(Canvas canvas)

boardColorPaint.setStyle(Paint.Style.STROKE);

boardColorPaint.setStrokeWidth(16);

boardColorPaint.setColor(boardColor);

boardColorPaint.setAntiAlias(true);

cellFillColorPaint.setStyle(Paint.Style.FILL);

cellFillColorPaint.setColor(cellFillColor);

cellFillColorPaint.setAntiAlias(true);

cellHighlightColorPaint.setStyle(Paint.Style.FILL);

cellHighlightColorPaint.setColor(cellHighlightColor);

cellHighlightColorPaint.setAntiAlias(true);

letterPaint.setStyle(Paint.Style.FILL);

letterPaint.setAntiAlias(true);

letterPaint.setColor(letterColor);

colorCell(canvas, solver.getSelectedRow(), solver.getSelectedColumn());

canvas.drawRect(0, 0, getWidth(), getHeight(), boardColorPaint);

drawBoard(canvas);

drawNumber(canvas);

@Override

Method onTouchEvent:

onTouchEvent(MotionEvent event)

boolean isValid;

float x <- event.getX();

float y <- event.getY();

int action <- event.getAction();

if (action = MotionEvent.ACTION\_DOWN) then

solver.setSelectedRow((int) Math.ceil(y / cellSize));

solver.setSelectedColumn((int) Math.ceil(x / cellSize));

isValid <- true;

else

isValid <- false;

return isValid;

Method drawNumber:

void drawNumber(Canvas canvas)

letterPaint.setTextSize(cellSize);

for (int r <- 0; r < 9; r++)

for (int c <- 0; c < 9; c++)

if (solver.getBoard()[r][c] !<- 0) then

String text <- Integer.toString(solver.getBoard()[r][c]);

float width, height;

letterPaint.getTextBounds(text, 0, text.length(), letterPaintBounds);

width <- letterPaint.measureText(text);

height <- letterPaintBounds.height();

canvas.drawText(text, (c \* cellSize) + ((cellSize - width) / 2),

(r \* cellSize + cellSize) - ((cellSize - height) / 2), letterPaint);

letterPaint.setColor(letterColorSolve);

for (ArrayList<Object> letter : solver.getEmptyBoxIndex())

int r <- (int) letter.get(0);

int c <- (int) letter.get(1);

String text <- Integer.toString(solver.getBoard()[r][c]);

float width, height;

letterPaint.getTextBounds(text, 0, text.length(), letterPaintBounds);

width <- letterPaint.measureText(text);

height <- letterPaintBounds.height();

canvas.drawText(text, (c \* cellSize) + ((cellSize - width) / 2),

(r \* cellSize + cellSize) - ((cellSize - height) / 2), letterPaint);

Method colorCell:

void colorCell(Canvas canvas, int row, int column)

if (solver.getSelectedRow() !<- -1 && solver.getSelectedColumn() !<- -1) then

canvas.drawRect((column - 1) \* cellSize, 0, column \* cellSize,

cellSize \* 9, cellHighlightColorPaint);

canvas.drawRect(0, (row - 1) \* cellSize, cellSize \* 9,

row \* cellSize, cellHighlightColorPaint);

canvas.drawRect((column - 1) \* cellSize, (row - 1) \* cellSize,

column \* cellSize, row \* cellSize, cellHighlightColorPaint);

invalidate();

Method drawThickLine:

void drawThickLine()

boardColorPaint.setStyle(Paint.Style.STROKE);

boardColorPaint.setStrokeWidth(10);

boardColorPaint.setColor(boardColor);

Method drawThinLine:

void drawThinLine()

boardColorPaint.setStyle(Paint.Style.STROKE);

boardColorPaint.setStrokeWidth(4);

boardColorPaint.setColor(boardColor);

Method drawBoard:

void drawBoard(Canvas canvas)

for (int c <- 0; c < 10; c++)

if (c % 3 = 0) then

drawThickLine();

else

drawThinLine();

canvas.drawLine(cellSize \* c, 0,

cellSize \* c, getWidth(), boardColorPaint);

for (int r <- 0; r < 10; r++)

if (r % 3 = 0) then

drawThickLine();

else

drawThinLine();

canvas.drawLine(0, cellSize \* r,

getWidth(), cellSize \* r, boardColorPaint);

Method SudokuSolver:

SudokuSolver()

return solver;

- Sudoku Solver Class -

int selected\_row;

int selected\_column;

int[][] board;

ArrayList<ArrayList<Object>> emptyBoxIndex;

SudokuSolver()

selected\_row <- -1;

selected\_column <- -1;

board <- new int[9][9];

for (int r <- 0; r < 9; r++)

for (int c <- 0; c < 9; c++)

board[r][c] <- 0;

emptyBoxIndex <- new ArrayList<>();

Method getEmptyBoxIndexes:

void getEmptyBoxIndexes()

for (int r <- 0; r < 9; r++)

for (int c <- 0; c< 9; c++)

if (this.board[r][c] = 0) then

this.emptyBoxIndex.add(new ArrayList<>());

this.emptyBoxIndex.get(this.emptyBoxIndex.size() - 1).add(r);

this.emptyBoxIndex.get(this.emptyBoxIndex.size() - 1).add(c);

Method check:

check(int row, int col)

if (this.board[row][col] > 0) then

for (int i <- 0; i < 9; i++)

if (this.board[i][col] = this.board[row][col] && row !<- i) then

return false;

if (this.board[row][i] = this.board[row][col] && col !<- i) then

return false;

int boxRow <- row / 3;

int boxCol <- col / 3;

for (int r <- boxRow \* 3; r < 3 \* boxRow + 3; r++)

for (int c <- boxCol \* 3; c < 3 \* boxCol + 3; c++)

if (this.board[r][c] = this.board[row][col] && row !<- r && col !<- c) then

return false;

return true;

Method solve:

solve(SudokuBoard display)

int row <- -1;

int col <- -1;

for (int r <- 0; r < 9; r++)

for (int c <- 0; c < 9; c++)

if (this.board[r][c] = 0) then

row <- r;

col <- c;

break;

if (row = -1 || col = -1) then

return true;

for (int i <- 1; i < 10; i++)

this.board[row][col] <- i;

display.invalidate();

if (check(row, col)) then

if (solve(display)) then

return true;

this.board[row][col] <- 0;

return false;

Method resetBoard:

void resetBoard()

for (int r <- 0; r < 9; r++)

for (int c <- 0; c < 9; c++)

board[r][c] <- 0;

this.emptyBoxIndex <- new ArrayList<>();

Method setNumberPos:

void setNumberPos(int num)

if (selected\_row !<- -1 && selected\_column !<- -1) then

if (this.board[selected\_row - 1][selected\_column - 1] = num) then

this.board[selected\_row - 1][selected\_column - 1] <- 0;

else

this.board[selected\_row - 1][selected\_column - 1] <- num;

Method int[][]:

int[][]()

return this.board;

Method ArrayList<ArrayList<Object>>:

ArrayList<ArrayList<Object>>()

return emptyBoxIndex;

Method getSelectedRow:

int getSelectedRow()

return selected\_row;

Method setSelectedRow:

void setSelectedRow(int row)

selected\_row <- row;

Method getSelectedColumn:

int getSelectedColumn()

return selected\_column;

Method setSelectedColumn:

void setSelectedColumn(int column)

selected\_column <- column;