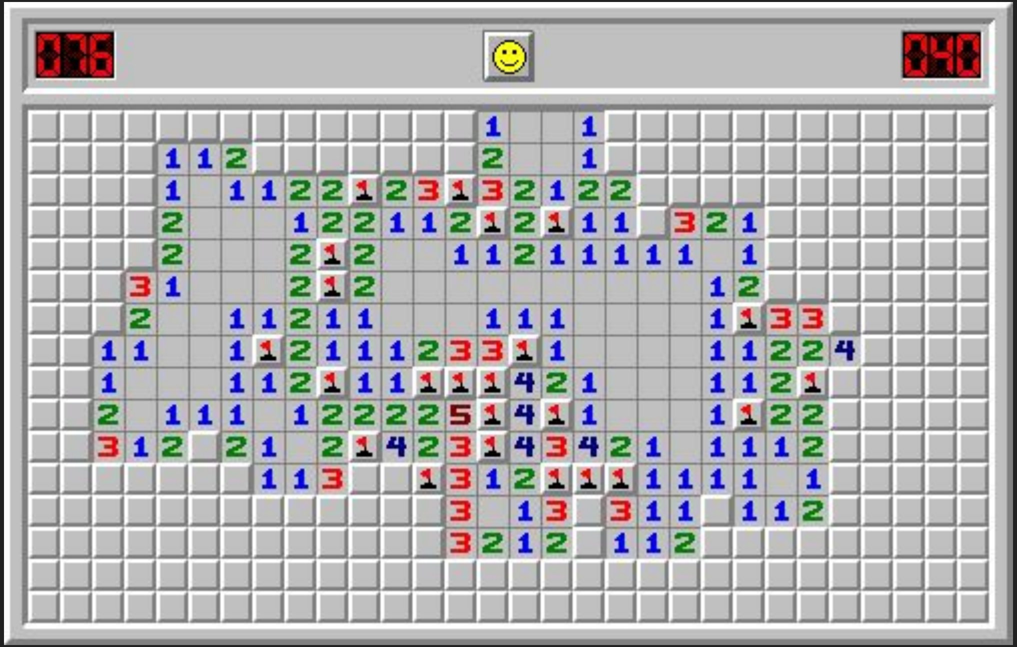
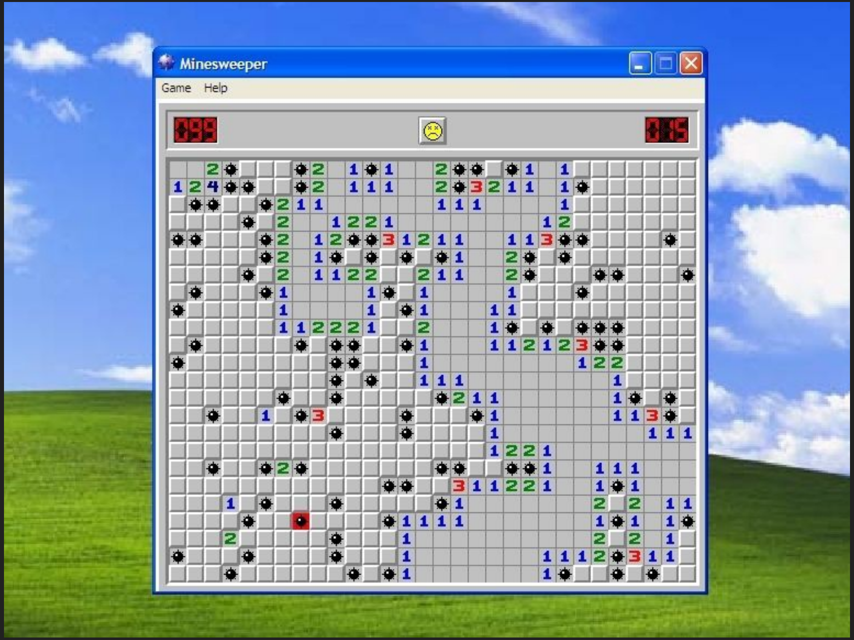
Minesweeper

Given Suman
UNCA CSCI182

What is Minesweeper?



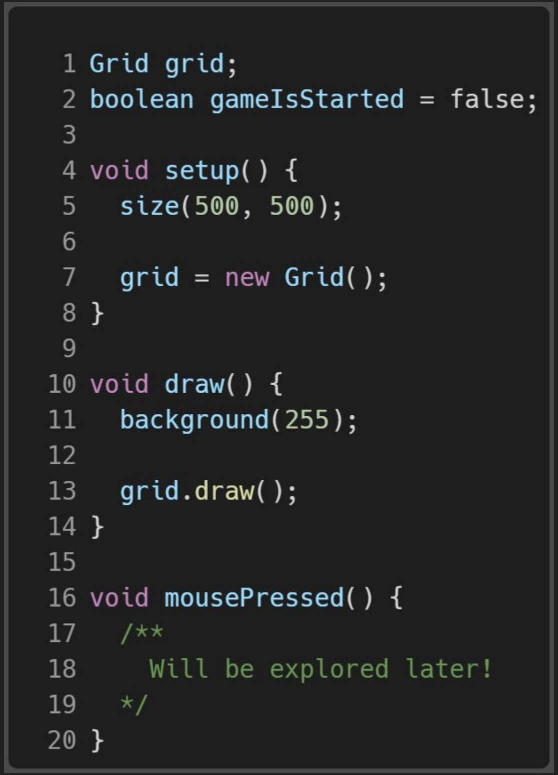


Program Structure

The program logic of the game is split into three classes:

- Minesweeper
  - A driver class that instantiates a grid and draws it to screen, as well as user interaction and game state
- Grid
  - A container for the game grid, which is a two dimensional array of cells. Contains methods to handle creating a randomized board state, overseeing cell management, and validating game state
- Cell
  - An individual game component with internal state management

Setting up the driver class



```
Grid grid;
boolean gameIsStarted = false;

void setup() {
  size(500, 500);

  grid = new Grid();
}

void draw() {
  background(255);

  grid.draw();
}

void mousePressed() {
  /**
    Will be explored later!
  */
}
```

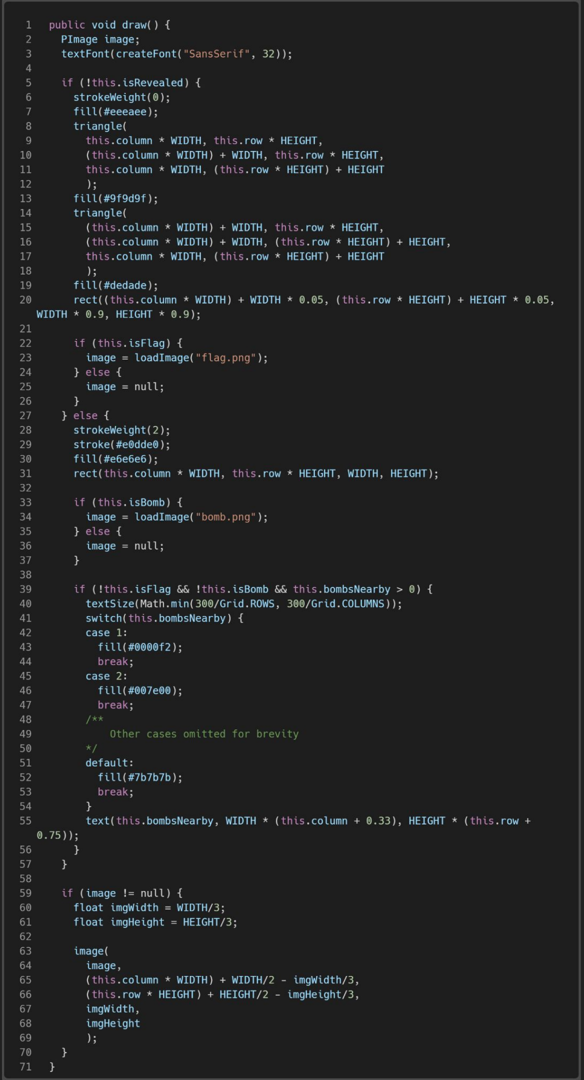
Creating and drawing the grid

```java
class Grid {
  public static final int ROWS = 10;
  public static final int COLUMNS = 10;
  public static final int BOMBS = 10;

  private Cell[][] grid;

  Grid() {
    this.grid = new Cell[ROWS][COLUMNS];
    for (int row = 0; row < Grid.ROWS; row++) {
      for (int column = 0; column < Grid.COLUMNS; column++) {
        this.grid[row][column] = new Cell(row, column);
      }
    }
  }

    public void draw() {
    for (int row = 0; row < Grid.ROWS; row++) {
      for (int column = 0; column < Grid.COLUMNS; column++) {
        this.grid[row][column].draw();
      }
    }
  }
}
```

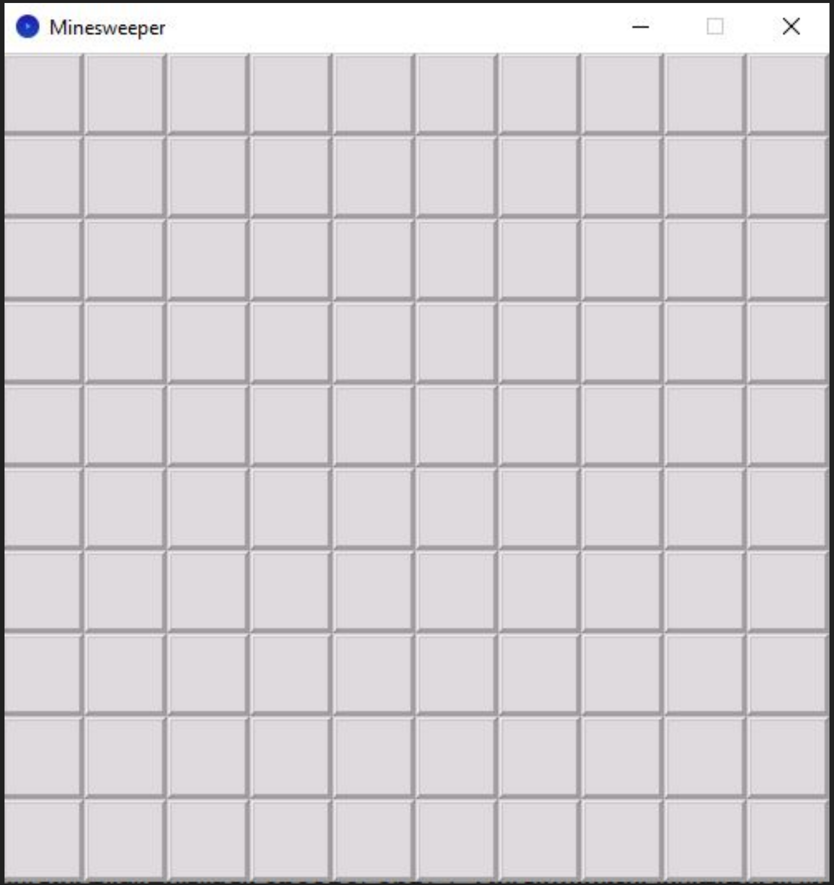
Creating the cells

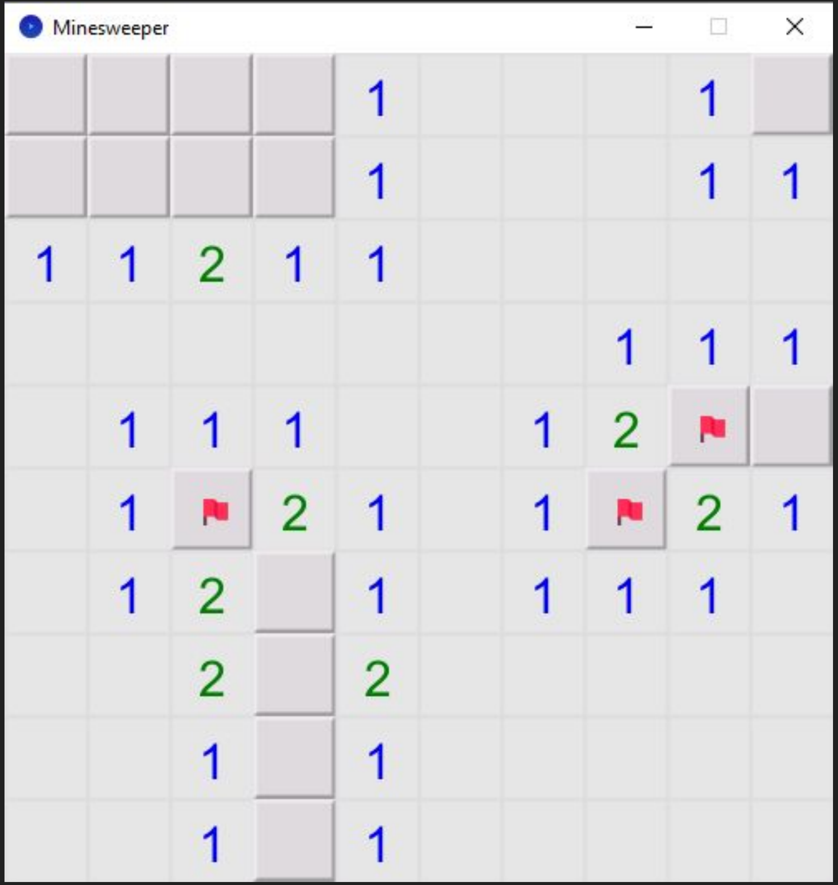
```java
class Cell {
  public final float HEIGHT = height/Grid.ROWS;
  public final float WIDTH = width/Grid.COLUMNS;

  private int row;
  private int column;
  private int bombsNearby = 0;
  private boolean isBomb = false;
  private boolean isFlag = false;
  private boolean isRevealed = false;

  Cell(int row, int column) {
    this.row = row;
    this.column = column;
  }
}
```

Drawing the cells



```java
public void draw() {
  PImage image;
  textFont(createFont("SansSerif", 32));

  if (!this.isRevealed) {
    strokeWeight(0);
    fill(#eeeaee);
    triangle(
      this.column * WIDTH, this.row * HEIGHT,
      (this.column * WIDTH) + WIDTH, this.row * HEIGHT,
      this.column * WIDTH, (this.row * HEIGHT) + HEIGHT
      );
    fill(#9f9d9f);
    triangle(
      (this.column * WIDTH) + WIDTH, this.row * HEIGHT,
      (this.column * WIDTH) + WIDTH, (this.row * HEIGHT) + HEIGHT,
      this.column * WIDTH, (this.row * HEIGHT) + HEIGHT
      );
    fill(#dedade);
    rect((this.column * WIDTH) + WIDTH * 0.05, (this.row * HEIGHT) + HEIGHT * 0.05, WIDTH * 0.9, HEIGHT * 0.9);

    if (this.isFlag) {
      image = loadImage("flag.png");
    } else {
      image = null;
    }
  } else {
    strokeWeight(2);
    stroke(#e0dde0);
    fill(#e6e6e6);
    rect(this.column * WIDTH, this.row * HEIGHT, WIDTH, HEIGHT);

    if (this.isBomb) {
      image = loadImage("bomb.png");
    } else {
      image = null;
    }

    if (!this.isFlag && !this.isBomb && this.bombsNearby > 0) {
      textSize(Math.min(300/Grid.ROWS, 300/Grid.COLUMNS));
      switch(this.bombsNearby) {
      case 1:
        fill(#0000f2);
        break;
      case 2:
        fill(#007e00);
        break;
      /**
          Other cases omitted for brevity
      */
      default:
        fill(#7b7b7b);
        break;
      }
      text(this.bombsNearby, WIDTH * (this.column + 0.33), HEIGHT * (this.row + 0.75));
    }
  }

  if (image != null) {
    float imgWidth = WIDTH/3;
    float imgHeight = HEIGHT/3;

    image(
      image,
      (this.column * WIDTH) + WIDTH/2 - imgWidth/3,
      (this.row * HEIGHT) + HEIGHT/2 - imgHeight/3,
      imgWidth,
      imgHeight
      );
  }
}
```


Minesweeper

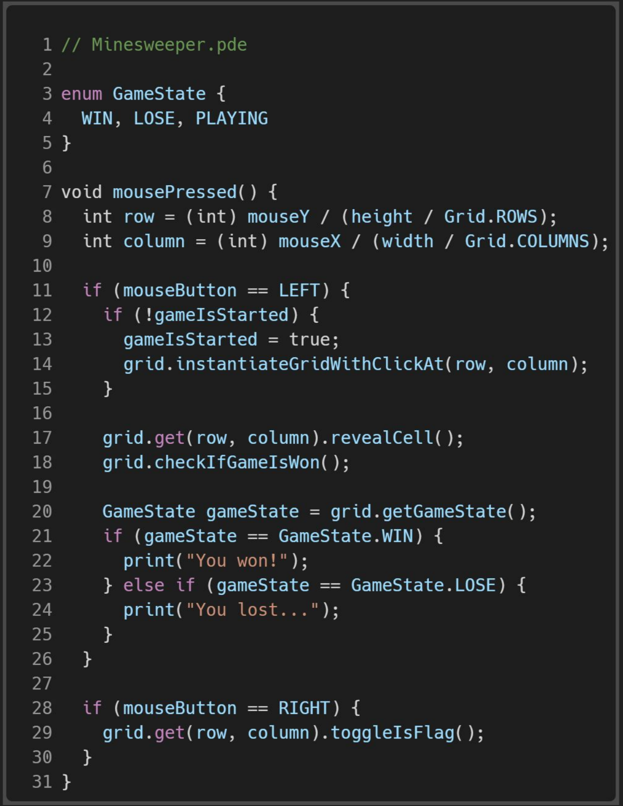

Minesweeper

Right click to toggle flags

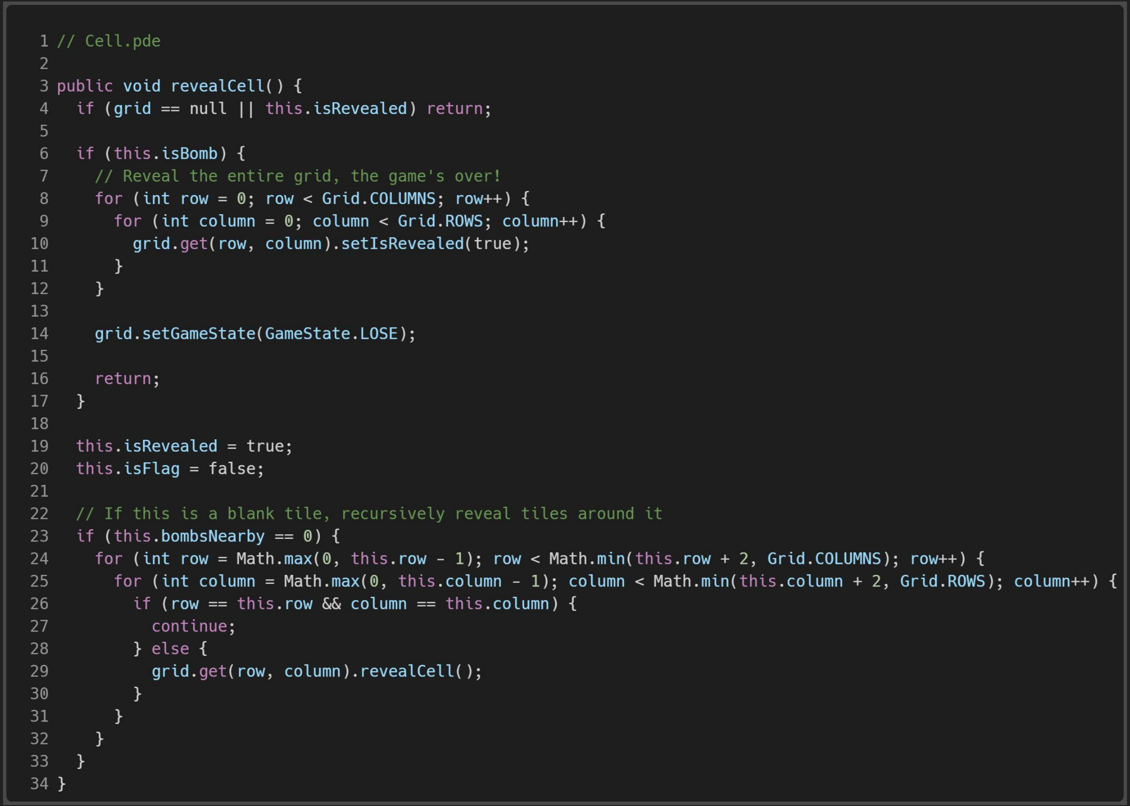
```
// Cell.pde

private boolean isFlag = false;

public void toggleIsFlag() {
  this.isFlag = !this.isFlag;
}
```

```
// Minesweeper.pde

void mousePressed() {
  int row = (int) mouseY / (height / Grid.ROWS);
  int column = (int) mouseX / (width / Grid.COLUMNS);

  if (mouseButton == RIGHT) {
    grid.get(row, column).toggleIsFlag();
  }
}
```

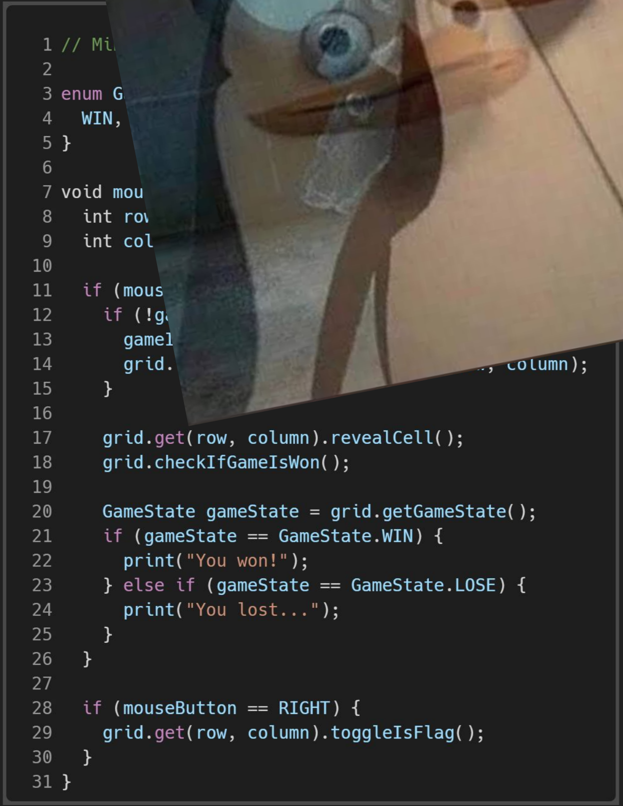
Left click to reveal cells



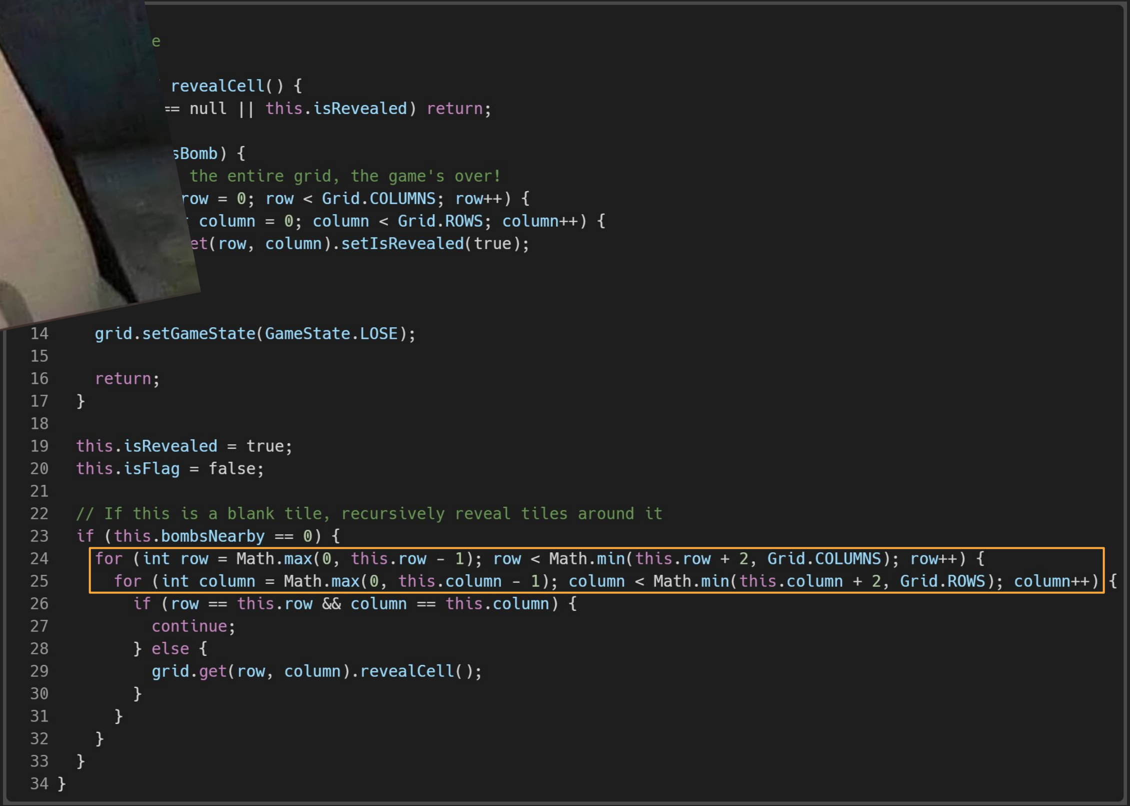
```
// Minesweeper.pde

enum GameState {
  WIN, LOSE, PLAYING
}

void mousePressed() {
  int row = (int) mouseY / (height / Grid.ROWS);
  int column = (int) mouseX / (width / Grid.COLUMNS);

  if (mouseButton == LEFT) {
    if (!gameIsStarted) {
      gameIsStarted = true;
      grid.instantiateGridWithClickAt(row, column);
    }

    grid.get(row, column).revealCell();
    grid.checkIfGameIsWon();

    GameState gameState = grid.getGameState();
    if (gameState == GameState.WIN) {
      print("You won!");
    } else if (gameState == GameState.LOSE) {
      print("You lost...");
    }
  }

  if (mouseButton == RIGHT) {
    grid.get(row, column).toggleIsFlag();
  }
}
```



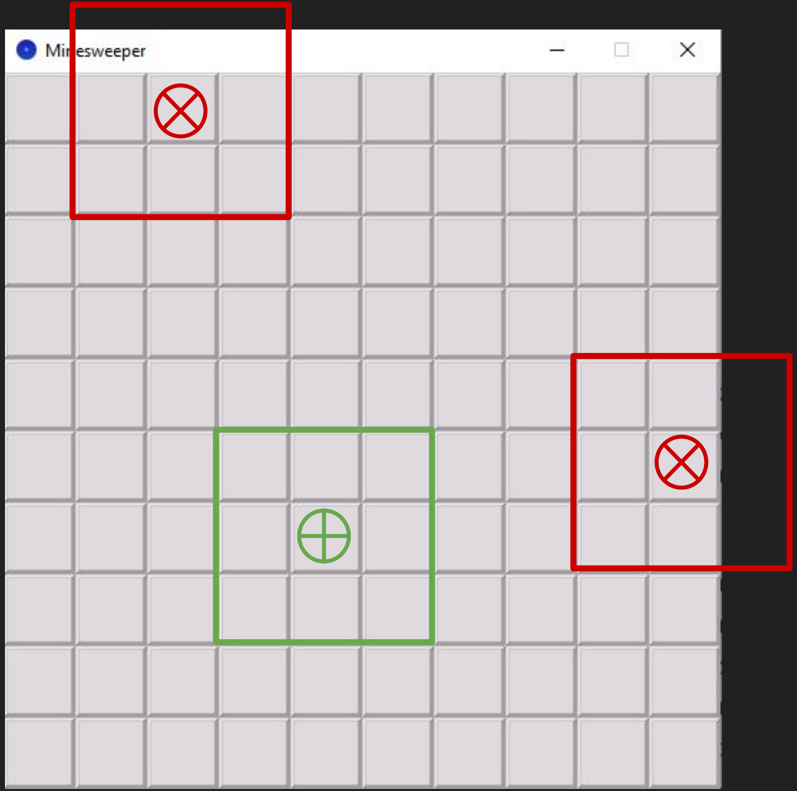
```
// Cell.pde

public void revealCell() {
  if (grid == null || this.isRevealed) return;

  if (this.isBomb) {
    // Reveal the entire grid, the game's over!
    for (int row = 0; row < Grid.COLUMNS; row++) {
      for (int column = 0; column < Grid.ROWS; column++) {
        grid.get(row, column).setIsRevealed(true);
      }
    }

    grid.setGameState(GameState.LOSE);

    return;
  }

  this.isRevealed = true;
  this.isFlag = false;

  // If this is a blank tile, recursively reveal tiles around it
  if (this.bombsNearby == 0) {
    for (int row = Math.max(0, this.row - 1); row < Math.min(this.row + 2, Grid.COLUMNS); row++) {
      for (int column = Math.max(0, this.column - 1); column < Math.min(this.column + 2, Grid.ROWS); column++) {
        if (row == this.row && column == this.column) {
          continue;
        } else {
          grid.get(row, column).revealCell();
        }
      }
    }
  }
}
```



```
// Mi

enum G
  WIN,
}

void mou
  int ro
  int col

  if (mous
    if (!g
      gamel
      grid.                  column);
    }

    grid.get(row, column).revealCell();
    grid.checkIfGameIsWon();

    GameState gameState = grid.getGameState();
    if (gameState == GameState.WIN) {
      print("You won!");
    } else if (gameState == GameState.LOSE) {
      print("You lost...");
    }
  }

  if (mouseButton == RIGHT) {
    grid.get(row, column).toggleIsFlag();
  }
}
```



```
e

 revealCell() {
== null || this.isRevealed) return;

sBomb) {
 the entire grid, the game's over!
row = 0; row < Grid.COLUMNS; row++) {
 column = 0; column < Grid.ROWS; column++) {
et(row, column).setIsRevealed(true);



    grid.setGameState(GameState.LOSE);

    return;
  }

  this.isRevealed = true;
  this.isFlag = false;

  // If this is a blank tile, recursively reveal tiles around it
  if (this.bombsNearby == 0) {
    for (int row = Math.max(0, this.row - 1); row < Math.min(this.row + 2, Grid.COLUMNS); row++) {
      for (int column = Math.max(0, this.column - 1); column < Math.min(this.column + 2, Grid.ROWS); column++) {
        if (row == this.row && column == this.column) {
          continue;
        } else {
          grid.get(row, column).revealCell();
        }
      }
    }
  }
}
```

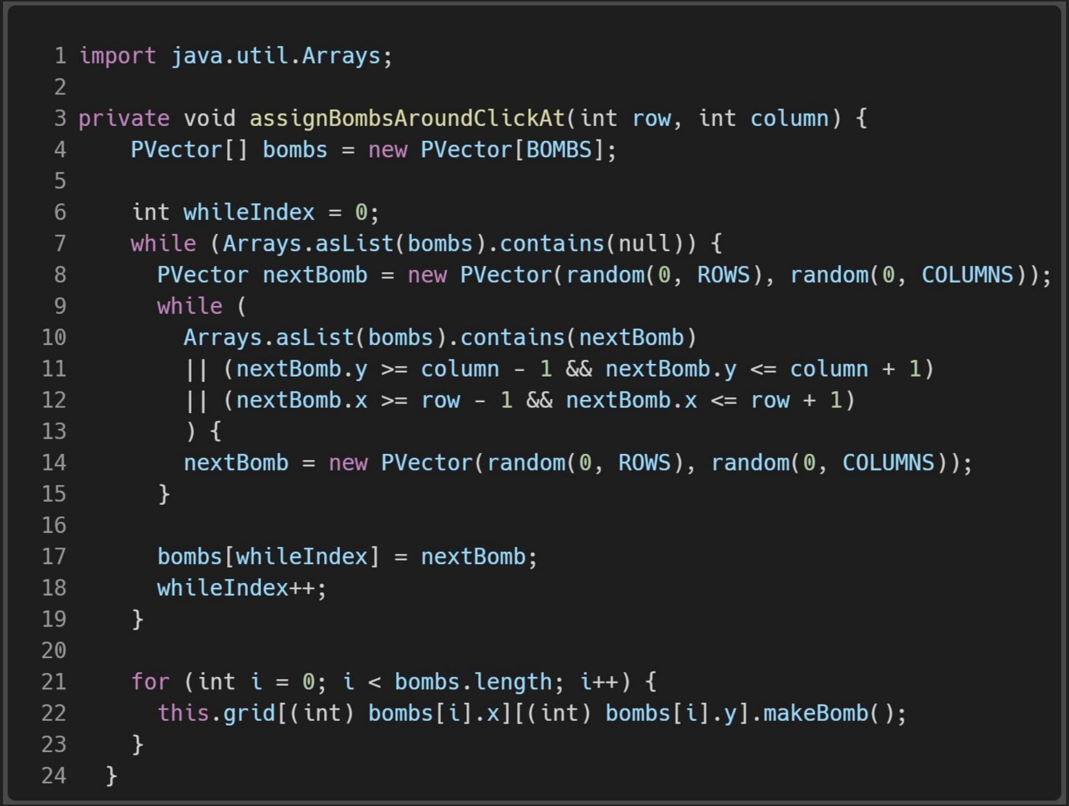


```
for (int row = Math.max(0, this.row - 1); row < Math.min(this.row + 2, Grid.COLUMNS); row++) {
  for (int column = Math.max(0, this.column - 1); column < Math.min(this.column + 2, Grid.ROWS); column++) {
    // Omitted for brevity
  }
}
```

Counting bombs

```
// Grid.pde

private void calculateNearbyBombs() {
  for (int row = 0; row < Grid.ROWS; row++) {
   for (int column = 0; column < Grid.COLUMNS; column++) {
    Cell cell = this.grid[row][column];

    if (!cell.getIsBomb()) {
     int nearbyBombs = 0;

     for (int subRow = Math.max(0, cell.getRow() - 1); subRow < Math.min(cell.getRow() + 2, Grid.ROWS); subRow++) {
      for (int subColumn = Math.max(0, cell.getColumn() - 1); subColumn < Math.min(cell.getColumn() + 2, Grid.COLUMNS); subColumn++) {
       if (this.grid[subRow][subColumn].getIsBomb()) nearbyBombs++;
      }
     }

     cell.setBombsNearby(nearbyBombs);
    }
   }
  }
 }
```

Assigning bombs



```java
import java.util.Arrays;

private void assignBombsAroundClickAt(int row, int column) {
  PVector[] bombs = new PVector[BOMBS];

  int whileIndex = 0;
  while (Arrays.asList(bombs).contains(null)) {
   PVector nextBomb = new PVector(random(0, ROWS), random(0, COLUMNS));
   while (
    Arrays.asList(bombs).contains(nextBomb)
    || (nextBomb.y >= column - 1 && nextBomb.y <= column + 1)
    || (nextBomb.x >= row - 1 && nextBomb.x <= row + 1)
    ) {
    nextBomb = new PVector(random(0, ROWS), random(0, COLUMNS));
   }

   bombs[whileIndex] = nextBomb;
   whileIndex++;
  }

  for (int i = 0; i < bombs.length; i++) {
   this.grid[(int) bombs[i].x][(int) bombs[i].y].makeBomb();
  }
 }
```

More grid methods

```
// Grid.pde

public GameState getGameState() {
  return this.gameState;
}

public void instantiateGridWithClickAt(int row, int column) {
  assignBombsAroundClickAt(row, column);
  calculateNearbyBombs();
}

public void checkIfGameIsWon() {
  for (int row = 0; row < Grid.ROWS; row++) {
    for (int column = 0; column < Grid.COLUMNS; column++) {
      Cell cell = this.grid[row][column];

      if (!cell.getIsBomb() && !cell.getIsRevealed()) return;
      if (cell.getIsBomb() && !cell.getIsFlag()) return;
    }
  }

  this.gameState = GameState.WIN;
}
```

```
// Minesweeper.pde

enum GameState {
  WIN, LOSE, PLAYING
}

void mousePressed() {
  int row = (int) mouseY / (height / Grid.ROWS);
  int column = (int) mouseX / (width / Grid.COLUMNS);

  if (mouseButton == LEFT) {
    if (!gameIsStarted) {
      gameIsStarted = true;
      grid.instantiateGridWithClickAt(row, column);
    }

    grid.get(row, column).revealCell();
    grid.checkIfGameIsWon();

    GameState gameState = grid.getGameState();
    if (gameState == GameState.WIN) {
      print("You won!");
    } else if (gameState == GameState.LOSE) {
      print("You lost...");
    }
  }

  if (mouseButton == RIGHT) {
    grid.get(row, column).toggleIsFlag();
  }
}
```

Thanks!

To view the code, go to:

github.com/givensuman/unca/tree/main/minesweeper