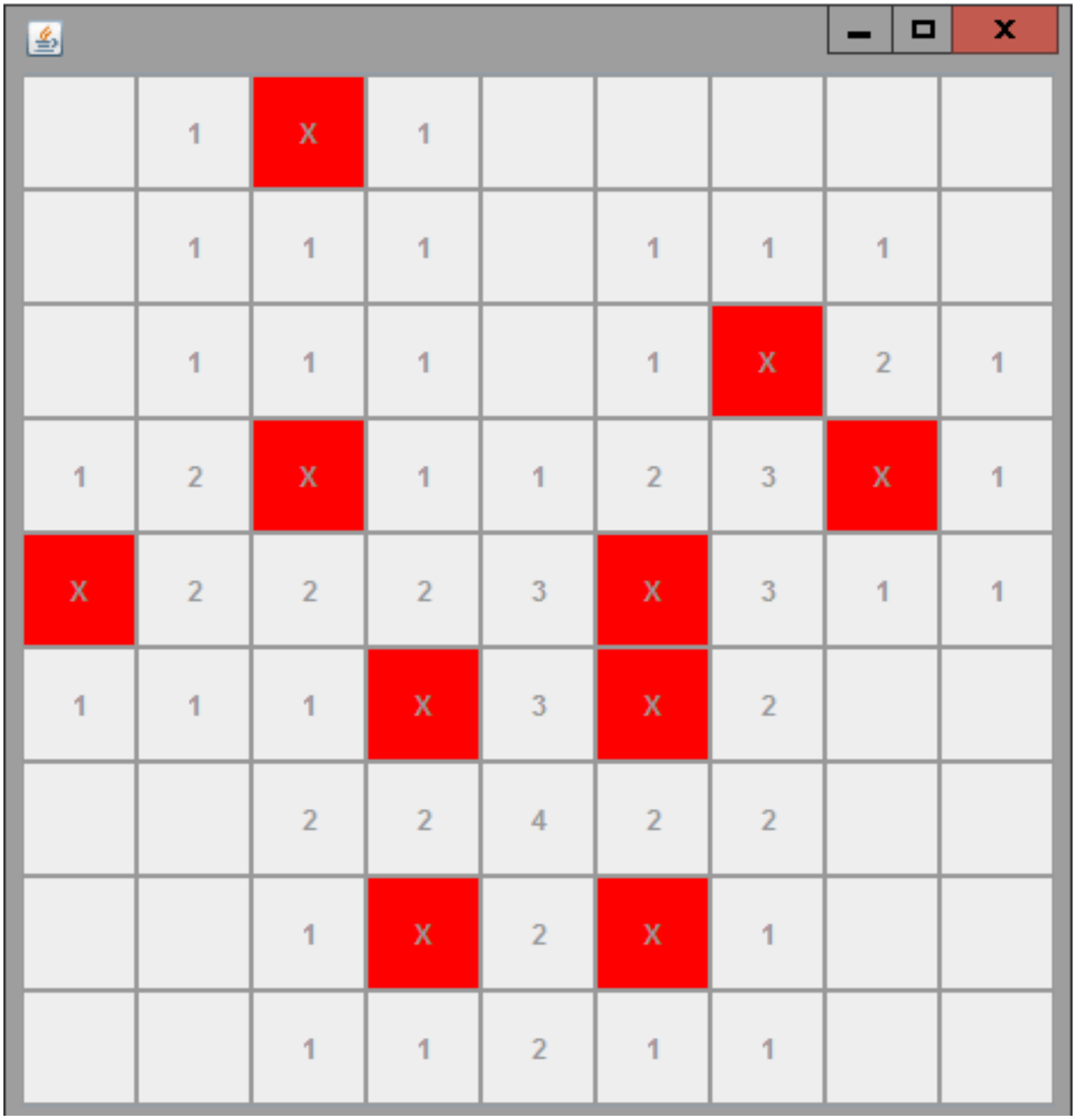
ICS4U1 CPT June 2019 Report

Minesweeper Game by: David, Vishwa, Sami, and Zeeshan

MINESWEEPER GAME



Authors: David, Vishwa, Sami, and Zeeshan

Course Code: ICS4U1-02

Last Updated: June 14th, 2019

Date Started: May 28th, 2019

Date Delivered: June 14th, 2019

Customer Requirements:

The client wants a game that is a variation of the classic Windows game, Minesweeper, where there are certain placements of mines and a timer implemented as well. It has the same aspects of the normal Minesweeper like being able to flag the tiles if you think there is a mine there, and the numbers on the tiles that help you indicate how many mines are around it however, an extra feature is that you can either play it or you can enable a bot to play it instead ensuring a win every time.

Project Timeline:

	Requirements	Design	Implementation	Testing	Maintenance
Proposed	May 28 - 29	May 29 - June 3	June 3 - 14	June 14 - 17	Jan 17 - 18
Completed	May 28 - 29	May 29 - June 7	June 3 - 13	June 13	June 14 - 17

Requirements

We took the proposed amount of time (2 days) to complete going over the CPT requirements and have a full understanding of what needed to be completed in order to achieve a 4+ in all sections.

Design

The designing took a little over 6 days which promptly was completed after completing a stepwise refinement and beginning some basic flowcharts, which would later be modified after implementation.

Implementation

Implementation took a little longer than expected, as we first had to create a game that would function with a player, and then add a bot that is able to successfully win the game. When opening group of non-numbered tiles, we ran into a bug where it not reveal the closest numbered tiles as it should. This was a bug that cost us some time correcting.

Testing

When we first planned out our timeline, we considered that after one step is done, the next begins. However, many of the steps in the process overlap each other, in which we started testing the game much earlier, during implementation to test pieces of our code before putting it all together. It took much longer than expected to test, as there were many components, rather than what we first initially thought: testing just the final product.

Maintenance

Lastly came the maintenance, in which we fixed any bugs, and commented on our code. As we were simultaneously working, and could not all be on the same code at the same time, one of us updated the design, refining the final flowcharts for our program, and again, making sure that the program covers all things which need to be completed for a 4+ in all sections, while others continued to implement and test.

Design Proposal:

We are planning to design a game called “Minesweeper” which satisfies all the customer requirements mentioned above. Mr. Reid, G. needs variables, sequence, selection, repetition, subprograms, classes, objects, and all other things needed to display our understanding of the ICS 4U1 course material. It is further visualized through our UMLs, stepwise refinement and flowcharts later within this document.

The main idea of the entire game is to have playable versions of the Minesweeper game where anyone can click their on the tiles they wish to open as well as a bot or CPU, that plays the game and wins for you. The mines are randomized every game and an end screen is revealed once the game is one and all the tiles not containing mines are all opened, either through the player or the bot.

We will use variables for keeping track of certain number values, such as the value of the tile based on the number of mines within the radius of the tile. We also used my multiple methods to code efficiently as shown in our multiple classes shown below in our UML diagrams. The use of classes allowed us to organize and breakdown the different aspects of our game into multiple large components each further broken down into subprograms made up of variables.

UML's: They outline each class' attributes and behaviours, and the instances of them (objects):

Class Diagram: Board

Attributes:
Tiles: Tiles[][] Mines : int Width : int Height : int
Behaviours:
void setBoard() void createButtons() void createMines() setNumMines() scanForEmptyTiles() botClick() gameOver()

Class Diagram: BoardTest

Attributes:
Tiles: Tiles[][] Width : int Height : int Limit: int BoardValues: int[][]
Behaviours:
void setBoardTest() void createButtons() void createMines() setNumMines() scanForEmptyTiles() botClick() gameOver()

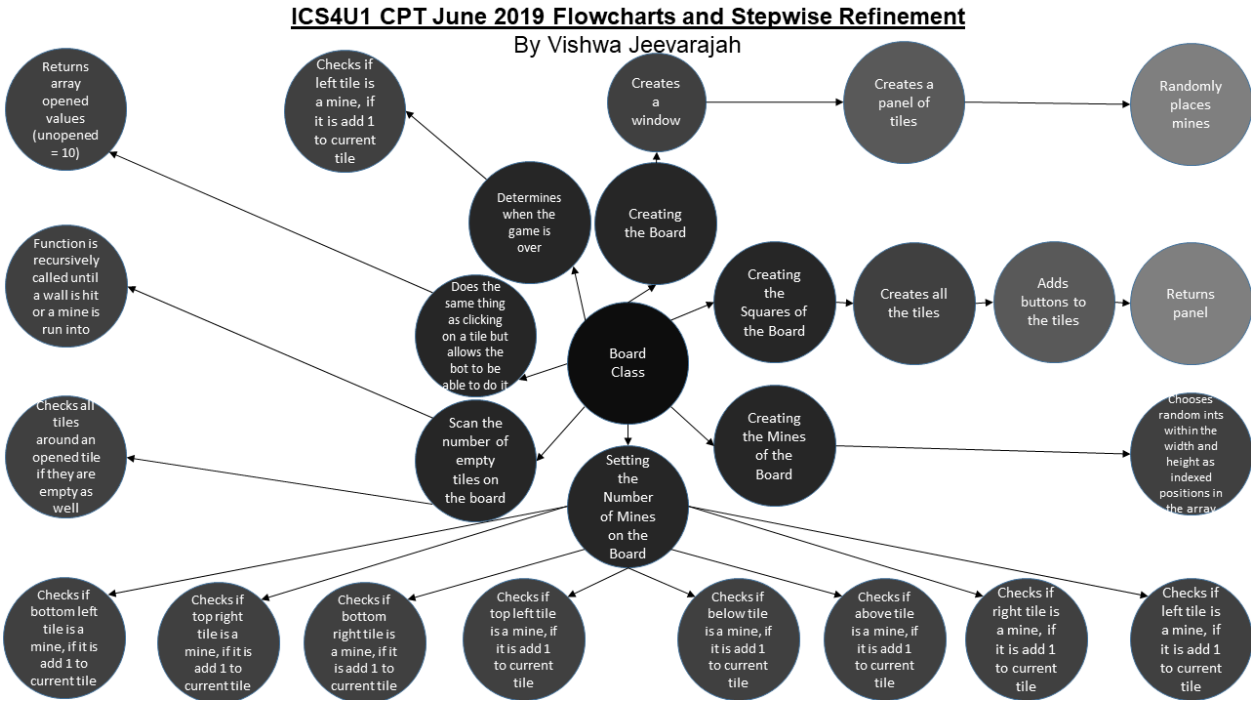
Class Diagram: Bot

Attributes: TileStatus: String Columns: int Rows: int GameOver: boolean
Behaviours: getTileStatus() checkGameOver() chooseTile() allHidden()

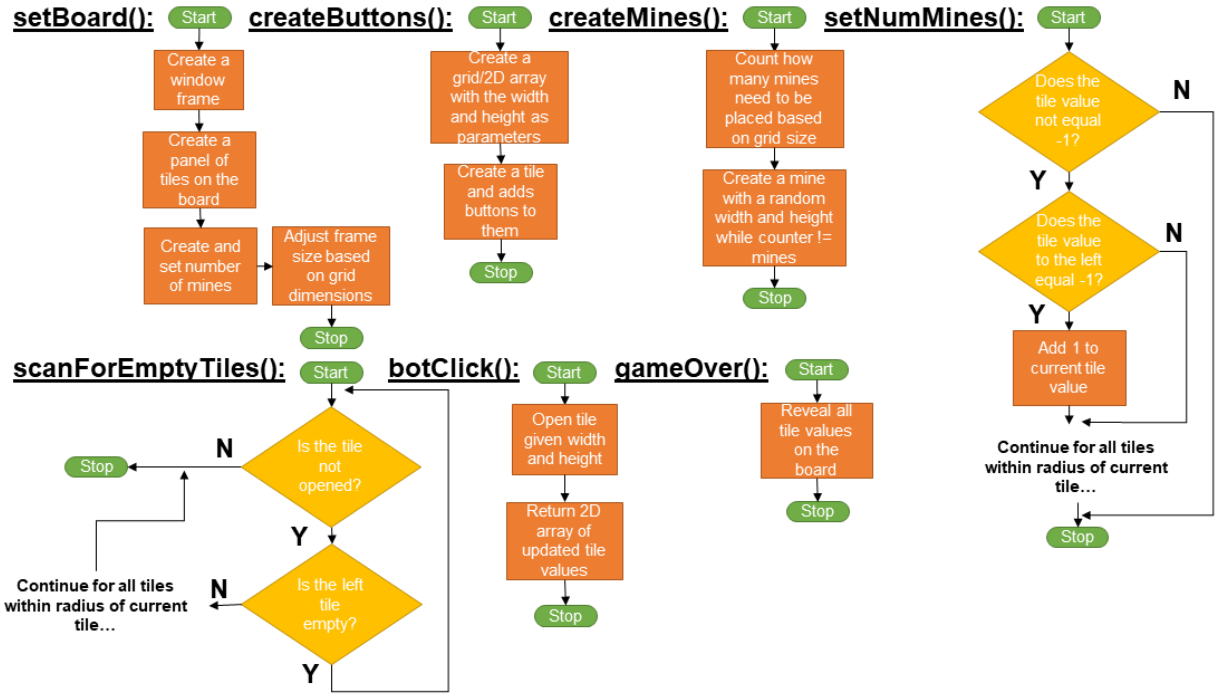
Class Diagram: Tile

Attributes: button : JButton board : Board opened : boolean numNeighbors : int
Behaviours: getButton() getValue() setMine() isMine() setValue() displayValue() openTile() incrementValue() isNotOpened() isEmpty() reveal()

Stepwise Refinement: A brief mind map of how we initially decided the break the game down into components/smaller pieces:

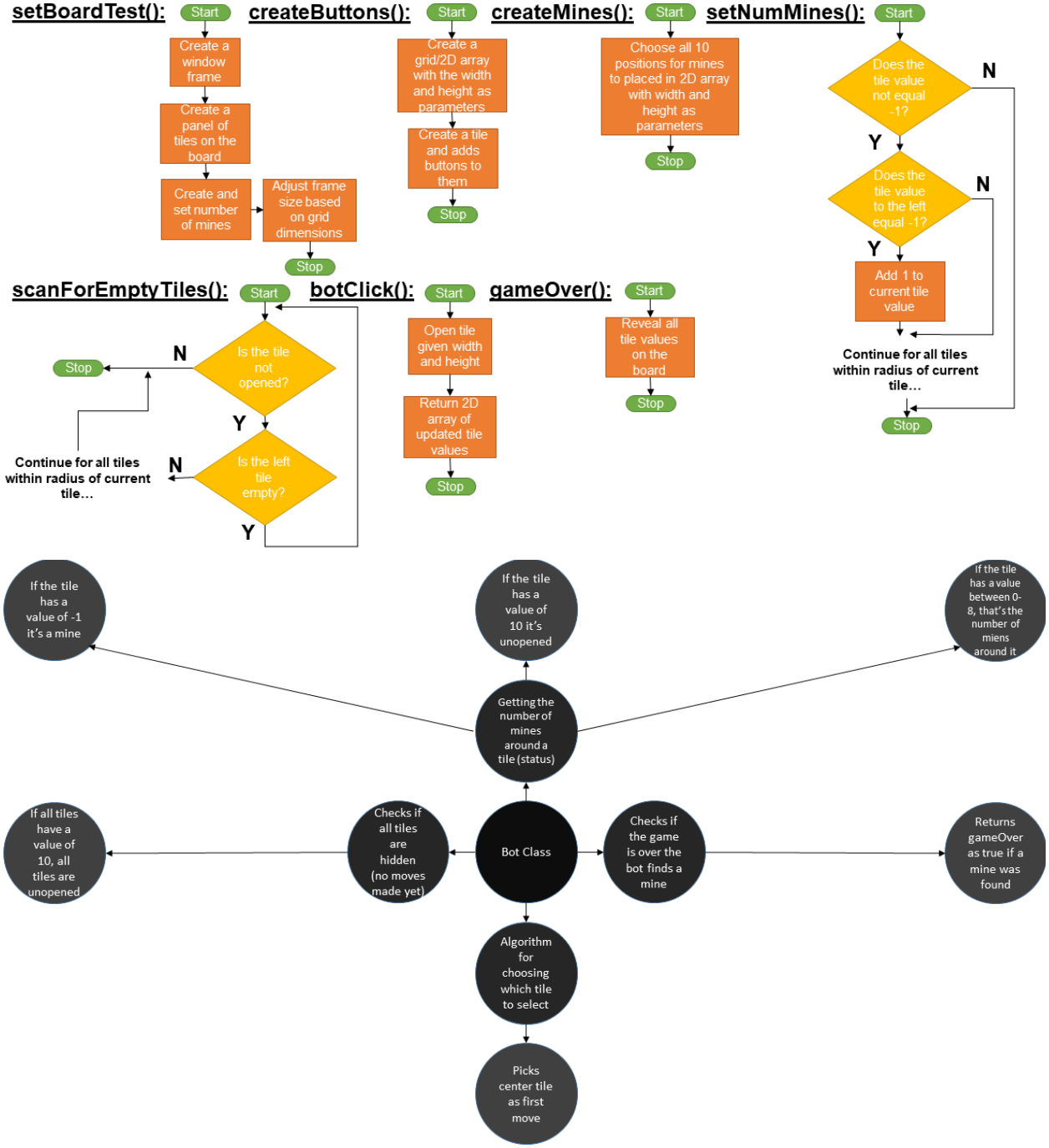


Board Class:

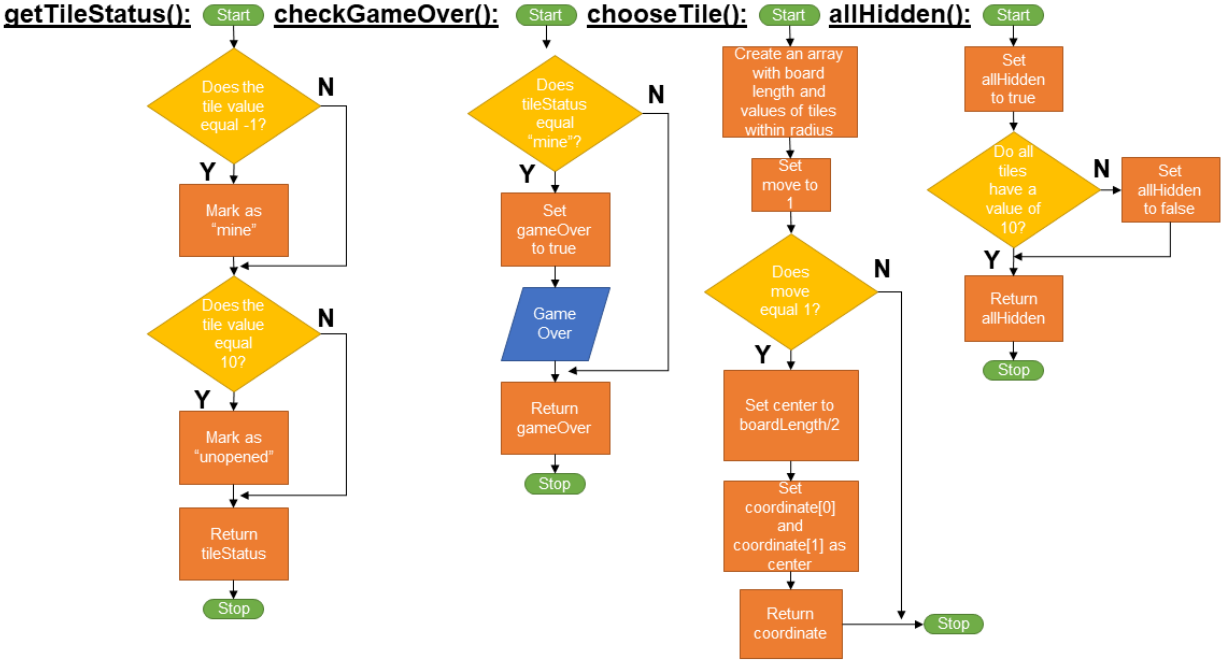

setBoard():
Start
Create a window frame
Create a panel of tiles on the board
Create and set number of mines
Adjust frame size based on grid dimensions
Stop
createButtons():
Start
Create a grid/2D array with the width and height as parameters
Create a tile and adds buttons to them
Stop
createMines():
Start
Count how many mines need to be placed based on grid size
Create a mine with a random width and height while counter != mines
Stop
setNumMines():
Start
Does the tile value not equal -1?
N
Y
Does the tile value to the left equal -1?
N
Y
Add 1 to current tile value
Continue for all tiles within radius of current tile…
Stop
scanForEmptyTiles():
Start
Is the tile not opened?
N
Stop
Y
Is the left tile empty?
N
Continue for all tiles within radius of current tile…
Y
botClick():
Start
Open tile given width and height
Return 2D array of updated tile values
Stop
gameOver():
Start
Reveal all tile values on the board
Stop

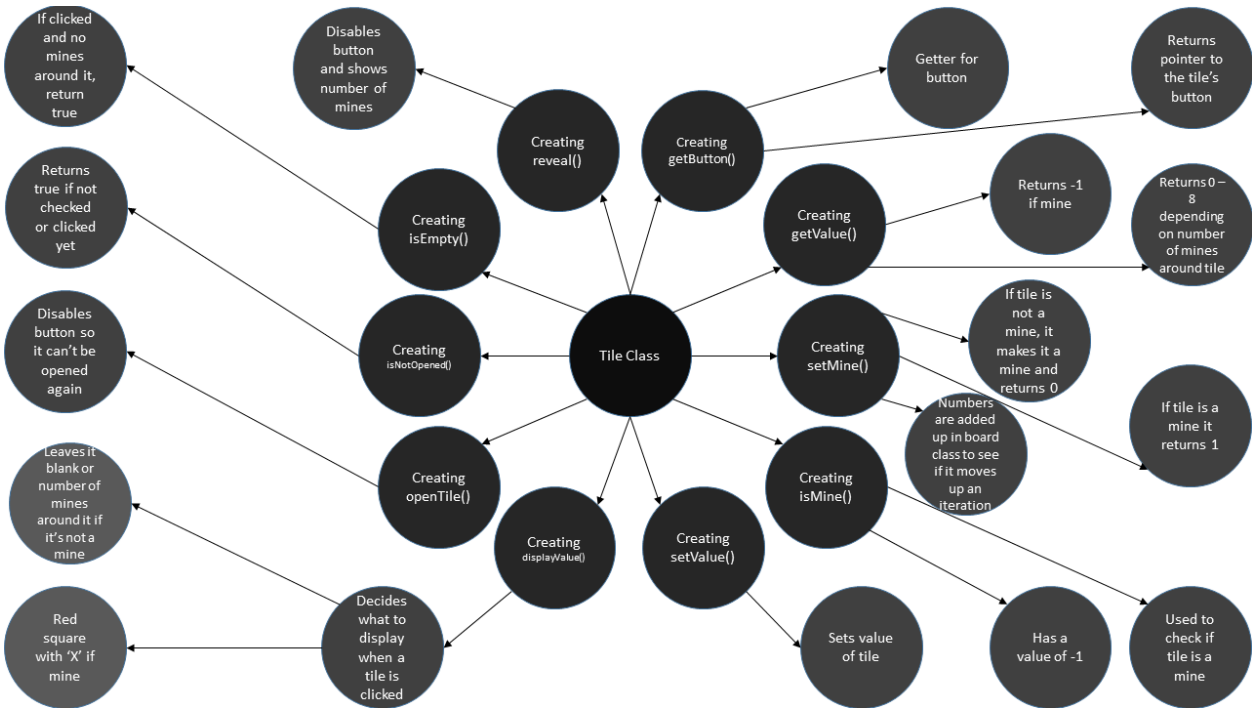

Returns array opened values (unopened = 10)
Checks if left tile is a mine, if it is add 1 to current tile
Creates a window
Creates a panel of tiles
Randomly places mines
Determines when the game is over
Creating the Board
Function is recursively called until a wall is hit or a mine is run into
Does the same thing as clicking on a tile but allows the bot to be able to do it
BoardTest Class
Creating the Squares of the Board
Creates all the tiles
Adds buttons to the tiles
Returns panel
Checks all tiles around an opened tile if they are empty as well
Scan the number of empty tiles on the board
Creating the Mines of the Board
Chooses specific 'positions' in the 2D array for bombs to see if the bot can solve it
Setting the Number of Mines on the Board
Checks if bottom left tile is a mine, if it is add 1 to current tile
Checks if top right tile is a mine, if it is add 1 to current tile
Checks if bottom right tile is a mine, if it is add 1 to current tile
Checks if top left tile is a mine, if it is add 1 to current tile
Checks if below tile is a mine, if it is add 1 to current tile
Checks if above tile is a mine, if it is add 1 to current tile
Checks if right tile is a mine, if it is add 1 to current tile
Checks if left tile is a mine, if it is add 1 to current tile

BoardTest Class:

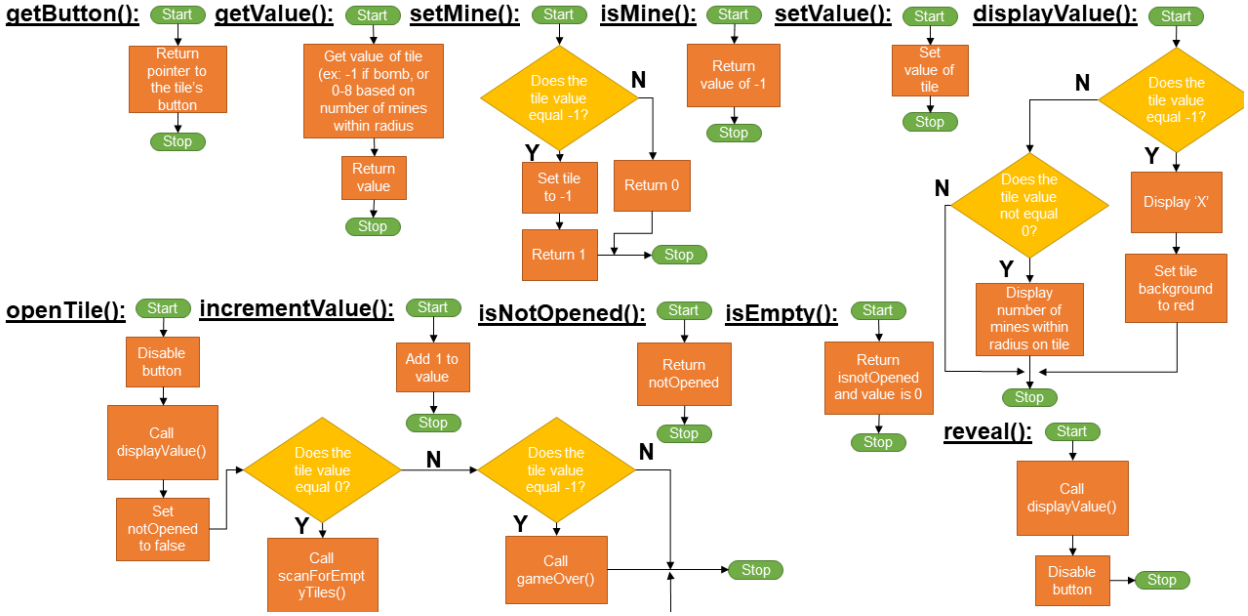


Bot Class:


getTileStatus():
Start
Does the tile value equal -1?
N
Y
Mark as "mine"
Does the tile value equal 10?
N
Y
Mark as "unopened"
Return tileStatus
Stop
checkGameOver():
Start
Does tileStatus equal "mine"?
N
Y
Set gameOver to true
Game Over
Return gameOver
Stop
chooseTile():
Start
Create an array with board length and values of tiles within radius
Set move to 1
Does move equal 1?
N
Y
Set center to boardLength/2
Set coordinate[0] and coordinate[1] as center
Return coordinate
Stop
allHidden():
Start
Set allHidden to true
Do all tiles have a value of 10?
N
Set allHidden to false
Y
Return allHidden
Stop


Tile Class
Creating reveal()
Disables button and shows number of mines
If clicked and no mines around it, return true
Creating isEmpty()
Returns true if not checked or clicked yet
Creating isNotOpened()
Disables button so it can't be opened again
Creating openTile()
Leaves it blank or number of mines around it if it's not a mine
Creating displayValue()
Decides what to display when a tile is clicked
Red square with 'X' if mine
Creating setValue()
Sets value of tile
Creating isMine()
Has a value of -1
Numbers are added up in board class to see if it moves up an iteration
Creating setMine()
If tile is not a mine, it makes it a mine and returns 0
If tile is a mine it returns 1
Used to check if tile is a mine
Creating getValue()
Returns -1 if mine
Returns 0 – 8 depending on number of mines around tile
Creating getButton()
Getter for button
Returns pointer to the tile's button

Tile Class:



Implementation Details and Deliverables:

Now that the game has been basically completed the final product should be playable. The game is similar to minesweeper but has different features to it. We have added unlimited moves and no timer, we also included a bot that plays it. For the bot we made it so you give it a coordinate and it sends it to the bot and that bot clicks the tile. You input the column and row and send it to the bot, -1 means it is a mine, 0-8 tells us how many mines are around that specific tile, and 10 means it is an unopened tile. It checks the tile to see if the tile is a mine and has an algorithm for determining which tile to select. The board class is used to create the board, we set the amount of mines we want to display is 10, and kept the dimensions to 9x9 and added a jframe for the user to see with tiles are clickable buttons, when you open a tile it displays a number from to tell you how many surrounding tiles there are. The tile class is used for adding tiles to the board, these tiles are clickable and when clicked on a mine it ends the game. What we learned from this was a lot like GUI interface and advanced arrays, how to add a bot that plays it itself which was probably the most challenging. We had to learn how to use swing to make the interface of the gamer and add the mouse clicking functionality.

Finally, if a user wanted to run the game it wouldn't be complicated at all. If you want to play the game yourself then Download the main class, the tile class, and the board class then just run it and you will be able to play the game. If you want to use the bot functionality then download the bot along with the other run it and input the coordinates.

```java
/*
 * David Szczecina 4/6/2019
 * ICS4UC
 * Main class, used to run program, creates
 */

public class Main {
    public static void main(String[] args){
        Board board = new Board();
        board.setBoard();
        //Bot bot = new Bot();
        //bot.play(board());
    }
}
```

```java
public class Tile implements ActionListener{
    private JButton button;
    private Board board;
    private int value;//-1 is mine, 0-8 is the number mines aroun
    private boolean notOpened;//Whether or not tile is clicked or



    public Tile(Board board){
        button = new JButton();//sense if clicked
        button.addActionListener(this);//sense if clicked
        button.setPreferredSize(new Dimension(50,50));//setting s
       // button.setMargin(new Insets(0,0,0,0));//space between b
        this.board = board;
        notOpened = true;//starts at no tile is clicked or checke
    }



    public JButton getButton() {//getter for button, just returns
        return button;
    }

    public int getValue() {//returns -1 if bomb, or 0-8 depending
        return value;
    }

    int setMine() {
        if (!isMine()) {//if its not a mine, it makes it a mine,
            setValue(-1);
            return 1;//added up in board class, so if it created
        }
        return 0;//if it didn't add a mine(already was a mine) it
    }
```

Testing:

For testing we created a test case to test our program by hard coding the mines to test the logic of the bot for specific mine locations. We wanted to also test to see what would happen if we hard coded the mines to see if our program would still run.

```java
private void createMines() {//randomly p
    //Random random = new Random();//cou
    int counter = 0;
    //while (counter != mines)
  {
        //counter += tiles[random.nextIn
    counter += tiles[0][1].setMine();
    counter += tiles[1][0].setMine();
    counter += tiles[1][1].setMine();
    counter += tiles[2][1].setMine();
    counter += tiles[3][1].setMine();
    counter += tiles[4][1].setMine();
    counter += tiles[5][1].setMine();
    counter += tiles[6][1].setMine();
    counter += tiles[7][1].setMine();
    counter += tiles[8][1].setMine();
    }
}
```

Maintenance:

The entire project in creating this product was quite successful, considering that this was our first attempt to create a game which demonstrated all that we had learned over a period of 5 months along with an attempt to make a working bot.

Within this game, we were able to effectively implement the use of classes and objects, for our tiles. The logic used in the game was far more complex than initially planned in our stepwise refinement. For instance, the idea of the bot being able to read the updated board and determine the best move based on logic wasn't as simple as initially thought. Another piece of logic was needed, to perfect it and there just wasn't enough time for it to be completed:

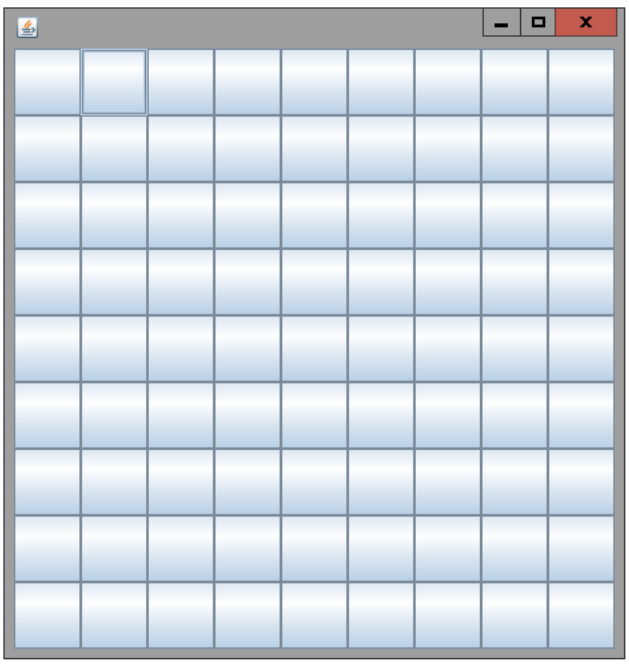
```java
 * Tile Values
 * -1 = Mine
 * 0 = no mines around
 * 1-8 = number of mines around tile
 * 10 = covered/hidden tile
 * 15 = 'flagged' tile
 *
 * private boolean allHidden()
     //checks if all the tiles are covered (no moves done yet)
     boolean allHidden = true;
     for(int i = 0; i>9; i++)
         for(int j = 0; j>9; j++)
             if( board[i][j] != 10)
                 allHidden = false;
     return allHidden;
```

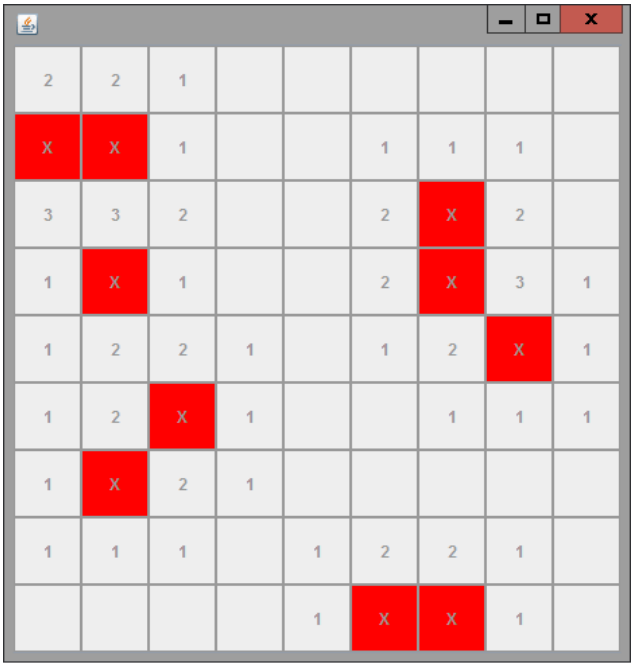
```java
int[] play(int[][] newBoard)
    for(int i = 0; i>9; i++)
        for(int j = 0; j>9; j++)
            if (board[i][j] != 15)
                board[i][j] = newBoard[i][j];
     int coordinate[];
     boolean foundMove = false;
    //checks if the board is empty
    if(allHiden())
        int center = 9 / 2;
        coordinate[0] = center;
        coordinate[1] = center;
        return coordinate;
```

Some components were quickly completed, with little to no difficulties, while others required more thinking which required a lot of testing and retesting to ensure that there were no bugs and things were working as designed.

In the process of making this product, what we found went very well was implementation of classes and objects into our program. It allowed for many objects to be produced while keeping the code clean (tiles). We are also proud of our use of variables, and how we were able to manage all of them within our program. We also used Java Swing to be able to make a 9x9 board as well as the tiles/buttons that go on this board. One last thing that went exceptionally well would be indicating to the player that the game was over by showing all the mines with red ‘X’s when one was clicked by the player.

If we had more time, we would try to focus more on perfecting of the bot on winning the game a near 100% of the time as well as general quality of life improvements. For example, with the bot, we could have a main screen giving the player the option to choose whether they want to actually play the game or have the bot beat it for them. This would introduce a design for a home screen as well as background music and sound effects to give the effect the real Minesweeper game, not just something put together by a group of amateur coders.





Resource Allocations:

Group Member	Responsibility	Description
David	UMLs	*Came up with the UMLs for the game constantly updating with further understanding of the scope of our project.*
Vishwa	Flowcharts/Stepwise Refinement	*Computed the flowcharts for the methods within the Board, BoardTest, Bot, Tile and Main Class. Also did most of the stepwise refinement.*
David	Design	*Came up with the logic behind the Board class, Tile class and the Bot class.*
Sami	Implement Bot	*Coded the Bot Class and found how to give the bot an updated board after every move.*
Zeeshan	Test Cases	*Coded the BoardTest Class and tested for preset positions of mines and see if the would succeed.*
David	Implement Game	*Coded the Board and Tile Class, and most functions of the game (i.e., randomizing bomb placement, displaying number of mines in radius, etc.).*
Vishwa	Report	*Wrote half of the report (title page, project timeline, design proposals, resource allocations, maintenance, etc.).*
Zeeshan	Report	*Wrote the other half of the report (customer requirements, implementation details and deliverables, testing, etc.).*