Contents

Module Guide

Class Name	Description	Secret
BShipMainMenu	The graphical object shown where the main menu is displayed	JFrame frame as the root container for the game.
BShipMainMenuClient	Runs BShipMainMenu (starts the game).	None.
FleetPanel	The graphical object where the user can create/organize their battleship fleet.	JPanel panel as the graphical object.
GamePanel	The graphical object through which the user interacts with the computer AI in a game of Battleship. The user's ship selection, current progress and opponents map are displayed in an interactive grid. The turn based gameplay is facilitated by the user clicking on their target cell in the grid. The grid responds graphically to display a hit, miss or entire ship sunk.	JPanel panel as the graphical object.
GameSound	A sound object that streams file information from .wav files to the computer's audio out.	None.
InputNamePanel	The graphical object where the user can enter their name after starting a new game.	JPanel panel as the graphical object.
JBackgroundPanel	A Paint Object that creates an image icon in the requested JPanel.	None.
Opponent	This representation of the computer's map (fleet). This class also defines the behavior of the computer (artificial intelligence) in terms of guessing where the user's ships are placed.	Using 2-dimensional array as the data structure for the computer's fleet.

PausePanel	The graphical object shown where the user pauses the game while in the GamePanel panel.	JPanel panel as the graphical object.
PlayerMap	This representation of the user's map (fleet). Elements in the array with value 0 represent unoccupied spaces, and elements with values x ∈ {1, 2, 3, 4, 5} represent spaces which are occupied by a ship of length x.	Using 2-dimensional array as the data structure for the computer's fleet.
ResultsPanel	The graphical object where the user can see the results of the game.	JPanel panel as the graphical object.
RulesPanel	The graphical object where the rules of battleship are displayed.	JPanel panel as the graphical object.
StatsIO	This class is responsible for storing high score data.	Storing the high score data in arraylist (data structure).

Design Decisions

The other screens the user can go to (create their fleet, play the game, see the stats) were designed as panels so that one class (BShipMainMenu) could contain all the panels and control the game by changing its visible panels. This mega class was given a public static field theGame which contained all the other panels so that the other panel classes could access theGame without needing access to a BShipMainMenu object. This would allow the other panel classes to show different panels when one of their buttons is pressed by mutating theGame panel. As a result, each of the other panel classes can be independently responsible for handling user input with it.

The RulesPanel object was created in the BShipMainMenu constructor while the rest of the Panels were created in the BShipMainMenu listener method because the panels in the listener will be created each time their corresponding button is pressed. This allows them to update their information (reset the shipButtons matrix in FleetPanel, update the StatsPanel) after a game is finished and their corresponding buttons are clicked. The RulesPanel object does not need to update its information when a game is finished so it can be created when the BShipMainMenu object is created.

MIS/MID

Class: BShipMainMenu

This class represents the graphical object shown where the main menu is displayed. This class is the root container for the game.

Interface

Uses
- InputNamePanel
- FleetPanel
- GamePanel
- PausePanel
- ResultsPanel
- RulesPanell
- StatsPanel
- CardLayout

Types
- None

Extends/Inherits
- JFrame class

Implements
- ActionListener interface

Access Programs
- BShipMainMenu(): BShipMainMenu
  - Constructs a new BShipMainMenu object
- actionPerformed(event: ActionEvent): void
  - Specifies how the program reacts to user interaction described by event

Implementation

Variables
- newGameButton, statsButton, rulesButton, quitButton: JButton
  - buttons labeled “New Game”, “Stats”, “Rules” and “Quit” respectively
- mainMenu: JPanel
  - panel consisting of the main menu buttons
- rulesPanel: RulesPanel
  - object of RulesPanel class

-inputNamePanel: InputNamePanel
    -object of InputNamePanel class
-fleetPanel: FleetPanel
    -object of FleetPanel class
-gamePanel: GamePanel
    -object of GamePanel class
-pausePanel: PausePanel
    -object of PausePanel class
-resultsPanel: ResultsPanel
    -object of ResultsPanelclass
-statsPanel: StatsPanel
    -object of StatsPanel class
-cardLayout: CardLayout
    -layout used by theGame panel to show different panels within theGame
-theGame: static JPanel
    -panel composed of other panels used to cycle through different panels when the game changes screens

Access Programs
-BShipMainMenuPanel(): BShipMainMenu
    - The main menu buttons ("New Game", "Stats", "Rules" and "Quit") are created and placed inside a main menu panel. A RulesPanel object is also created. The layout of theGame panel is set to a new CardLayout object. The main menu panel and the rulesPanel are added to theGame panel by calling on the addLayoutComponent method of the CardLayout object. The main menu buttons are registered with the current BShipMainMenu object as a listener. Since the main menu panel was added first, it is shown.

-actionPerformed(event: ActionEvent): void

    -If the "New Game" button is clicked, InputNamePanel, FleetPanel, GamePanel, PausePanel and ResultsPanel objects are created and added to theGame and the CardLayout object. The program shows the InputPanel panel of theGame panel. If the "Stats" button is clicked, a StatsPanel object is created and added to theGame and the CardLayout object. The program shows the StatsPanel panel of theGame panel. If the "Rules" button is clicked, the program shows the RulesPanel panel of theGame panel previously created in the BShipMainMenu constructor. If the "Quit" button is pressed, the GUI closes and the program terminates.

Class: InputNamePanel

This class represents the graphical object shown where the user can enter their name after starting a new game.

Interface

Uses
-BShipMainMenu

Types
-None

Extends/Inherits
-JPanel class

Implements
-ActionListener interface

Access Programs
-InputNamePanel(): InputNamePanel
    -Constructs a new InputNamePanel object

-static getInputName(): string
    -returns the current user's input name

-actionPerformed(event: ActionEvent): void
    -Specifies how the program reacts to user interaction described by event

Implementation

Variables
-name: static String
    -static variable which keeps track of the current user's name
-msg: JLabel
    -information message about what the user has to do
-inputField: JTextField
    -text field where user enters their name
-okButton: JButton
    -button labeled "OK" where user clicks after entering their name

Access Programs
-InputNamePanel(): InputNamePanel

- Creates GUI componets (information message, input field and “OK” button) and adds them to the panel. Registers the “OK” button (source) with the InputNamePanel object (listener).

-static getInputName(): string

- Returns static field name

-actionPerformed(event: ActionEvent): void

- When the user clicks the “OK” button (event = okButton), the static field name is assigned to the text the user entered in the input field and the program shows the FleetPanel panel in theGame panel of BShipMainMenu.

Class: FleetPanel

This class represents the graphical object shown where the user can create/organize their battleship fleet.

Interface

Uses
-BShipMainMenu
-PlayerMap
-GamePanel

Types
-None

Extends/Inherits
-JPanel class

Implements
-ActionListener interface

Access Programs
-FleetPanel(): FleetPanel
    -Constructs a new FleetPanel object
-actionPerformed(event: ActionEvent): void
    -Specifies how the program reacts to user interaction described by event

Implementation

Variables
-shipLength: int
    -keeps track of current ship's length
-orientation: String
    -keeps track of current ship's orientation
-msg: JLabel
    -information message that tells the user what to do and displays errors
-shipButtons: JButton[][]
    -matrix of buttons representing the fleet of ships
-horizButton: JCheckBox

-check box representing horizontal orientation
-vertButton: JCheckBox
-check box representing vertical orientation
-orientationPanel: JPanel
-panel consisting of vertButton and horizButton
-doneButton: JButton
-“Done” button where user clicks after organizing their fleet

Access Programs
-FleetPanel(): FleetPanel
- Initiliazes shipLength to 5 and orientation to horizontal. Creates GUI components (an information message, matrix of ship buttons, orientation buttons and a “Done” button) and adds them to the panel. Registers the ship buttons, orientation buttons and “Done” button (sources) with the FleetPanel object (listener).

-actionPerformed(event: ActionEvent): void
- If the user clicks one of the orientation buttons, the current orientation is assigned to the appropriate orientation. If the user clicks the “Done” button, the program shows the gamePanel panel in theGame panel of BShipMainMenu. If the user clicks one of the ship buttons, an information message is displayed about the size of the current ship. The shipButtons matrix’s column and row indices of the ship button clicked are calculated and the static checkOverlap method of the PlayerMap class is called to see if the ship selected overlaps any other previously placed ships. If overlap occurs, an error message is displayed. If no overlap occurs, the program attempts to place the current ship starting at the ship button clicked in the fleet. Depending on the orientation and the starting point, a horizontal or vertical segment of ship buttons of the current ship length is highlighted. If a ship is too big to start at the starting point, an ArrayIndexOutOfBounds exception is thrown and caught (an appropriate error message is displayed and the segment is unhighlighted). The playerMap is updated by calling the static method setMap of the PlayerMap class and the shipLength is decremented by 1. If shipLength = 0, a message notifying the user that their fleet is complete is displayed and the player map of the GamePanel class is updated by calling the static methods setPlayerMap and colourPlayerMap.
-*Pseudocode*:
```
if event = horizButton
 orientation = “horizontal”
else if event = vertButton
 orientation = “vertical”
else if event = doneButton
 BShip.theGame.show(gamePanel)
else
 column = (event.getX()-170)/45
 row = (event.getY()-100)/45
 if shipLength >= 1
   msg.setText("Select orientation, then place ship of length " + (shipLength-1))
 if PlayerMap.checkOverlap(row, column, shipLength, orientation)
   msg.setText("ERROR: Overlap occured. Click another starting position.")
else
 try
```

```
   if orientation = horizontal
    for i = 0 ... shipLength
      shipButtons[row][column+i].setBackground (orange)
   else
    for i = 0 ... shipLength
      shipButtons[row+i][column].setBackground (orange)
   PlayerMap.setMap(row, column, shipLength, orientation)
   shipLength--
   if shipLength = 0
    doneButton.setVisible()
    msg.setText("Fleet organization complete. Click Done to proceed to game.")
    GamePanel.setPlayerMap(PlayerMap.getMap())
    GamePanel.colourPlayerMap()
 catch (ArrayIndexOutOfBoundsException e)
   msg.setText("ERROR: Ship starting in row " + (row+1) + " column " + (column+1) + "
of           length"  + shipLength + " is too big. Click another starting position")
   while (column < 10 ^ row < 10)
          shipButtons[row][column].setBackground(white)
          if orientation = horizontal
            column++
          else
            row++
```

Class: GamePanel

This class represents the graphical object through which the user interacts with the computer AI in a game of Battleship. The user's ship selection, current progress and opponents map are displayed in an interactive grid. The turn based gameplay is facilitated by the user clicking on their target cell in the grid. The grid responds graphically to display a hit, miss or entire ship sunk.

Interface

Uses

- Opponent
- BShipMainMenu
- ResultsPanel
- PlayerMap
- InputNamePanel

Types

- None

Extends/Inherits

- JPanel class

Implements

- ActionListener interface

Access Programs

- GamePanel(): GamePanel
  - Constructs a new GamePanel object
- actionPerformed(event: ActionEvent): void
  - Specifies how the program reacts to user interaction described by event

- colourPlayerMap():void
  - colors the cells where the user has placed their ships
- setPlayerMap(int[][] map)
  - Initializes Player map

Implementation

Variables

- opponent: Opponent
  - generates opponent's ship orientation and opponent attacks on users fleet
- playerTurn: static boolean
  - boolean value keeps track of players turn
- computerTurn : static boolean
  - boolean value keeps track of computers turn (!playerTurn)

- compHitRow:int
  - stores the row index of previous hit from AI on the user
- compHitColumn: int
  - stores the row index of previous hit from AI on the user
- compHit : private boolean
  - true when a selected cell is hit
- compMapButtons:static JButton[][]
  - Multi Dimensional Array of buttons representing the computers hidden fleet in the form of a grid
- playerMapButtons: static JButton[][]
  - Multi Dimensional Array of buttons representing the users fleet in the form of a grid
- pause:JButton
  - suspends gameplay
- playerMap: int[][]
  - An integer array that stores ship locations based on shipLength (ie.1,2,3,4,5) and unknowns on 0's and previous hits on -1's
- opponentMap:static int[][]
  - An integer array that stores ship locations based on shipLength (ie.1,2,3,4,5) and unknowns on 0's and previous hits on -1's
- shipCounter :int[]
  - stores the number of hits on a given ship, index is based on shipLength - 1.
- playerScore:int
  - stores the players score
- computerScore:int
  - stores the computers score
- progressBar:JProgressBar
  - displays players score visually in the form of a progress bar
- progressBar_1:JProgressBar
  - displays computers score visually in the form of a progress bar
- shipSunk:static JLabel
  - Label that informs the player they have sunk a ship
- lblPlayer:static JLabel
  - Label for players progress bar
- lblPlayer_1:static JLabel
  - Label for the computers progress bar

- lblBtlshp:static JLabel
    - Large Battleship logo
- horizontalLabels:JLabel[]
    - Array of labels for the columns of the opponents grid
- verticalLabels:JLabel[]
    - Array of labels for the rows of the opponents grid

Access Programs

- GamePanel(): GamePanel
    - Constructs a new GamePanel object with two labeled matricies of buttons representing the users fleet and the opponents fleet. Labeled progress bars, a large title, a pause button and ship sunk label.

- colourPlayerMap():void
    - Taking a playerMap, the algorithm loops through the multidimensional array using a nested for loop. When a cell holds an integer value greater than 0, it means there is a ship there. The algorithm then colors the appropriate cell blue.

```
function COLOURPLAYERMAP()
    for $i = 0\ i \to 9$ do
        for $j = 0\ j \to 9$ do
            if playerMap[i][j]>0 then
                Set Button to BLUE
            end if
        end for
    end for
end function
```

- setPlayerMap(int[][] map)
    - Initializes Player map

- actionPerformed(event: ActionEvent): void
    - Specifies how the program reacts when the user clicks one of the buttons

Algorithm: When the user clicks their opponents uncovered grid the coordinate of the button is used to check the index of the corresponding opponentMap[][]. Depending on the value stored at index i, j the shot will register as a hit on a ship of length l, or a miss.

Case 1a: HIT

when there has been a hit, the number of hits on that ship of lenght l and the playersScore are incremented by 1. These values are stored in shipCounter[] and playerScore respectively. The Color of the respective button is changed to red and he progress bar is updated. playersTurn becomes boolean false to allow for the computer to make a move.

Case 2a: HIT & SINK

Identical to Case 1, except shipCounter[] for the target ship equals the ships length. Displays “Computer Ship Sunk!” via the user interface. playersTurn becomes boolean false to allow for the computer to make a move.’

Case 3a:HIT & SINK & WIN GAME
the program moves onto the next panel (ResultsPanel) of BShipMainMenu. Using ResultsPanel, methods setWinner(InputNamePanel.getInputName()) and setLoser(“Computer”) are set.

Case 4a: MISS

Clicked button becomes no longer visible. playersTurn becomes boolean false to allow for the computer to make a move.’

During the computersTurn the GamePanel uses the Opponent class to generate an opponent guess. generateOpponentGuess takes the value of the last hit (if there was one) and the current misses on the PlayerBoard.

Case 1b: HIT

When there has been a hit, the minimap of the players ships turns red in the respective cell that was hit. The computers score is incremented by 1 and the progress bar is updated.

Case 2b:HIT & SINK & LOSE GAME
the program moves onto the next panel (ResultsPanel) of BShipMainMenu. Using the class ResultsPanel, methods setWinner(“Computer”) and setLoser(InputNamePanel.getInputName() are set.

Case 3b: MISS

targeted cell becomes no longer visible. playersTurn becomes boolean true to allow for the player to make a move.’

```
function ACTIONPERFORMED($ActionEvent\ e$)
    if $pausebutton$ then
        set cardLayout = BShipMainMenu.pause
    else
        if $playerTurn$ then
            get index of button (row,column)
            if $opponentMap[row][column] > 0$ then
                Set Button clicked to RED
                PlayerScore++
                Refresh ProgressBar
                if $ShipHits == ShipLength$ then
                    shipSunk is visible
                end if
            end if
            if $opponentMap[row][column]\ is\ 0$ then
                Set Button clicked to invisible
            end if
            if $PlayerScore\ is\ 15$ then
                Set visible panel to results
                designate Player as the winner
            end if
        end if
        $playerTurn \leftarrow false$
        if !$playerTurn$ then
            if $playerMap[row][column] > 0$ then
                Set cell on Player Map to RED
                computerScore++
                Refresh ProgressBar
            end if
            if $playerMap[row][column]\ is\ 0$ then
                Set cell on Player Map to invisible
            end if
            if $ComputerScore\ is\ 15$ then
                Set visible panel to results
                designate Computer as the winner
            end if
            $playerTurn \leftarrow true$
        end if
    end if
end function
```

Class: PausePanel

This class represents the graphical object shown when the user pauses the game while in the GamePanel panel

Interface

Uses
-BShipMainMenu

Types
-None

Extends/Inherits
-JPanel class

Implements
-ActionListener interface

Access Programs
-PausePanel(): PausePanel
    -Constructs a new PausePanel object

-actionPerformed(event: ActionEvent): void
    -Specifies how the program reacts to user interaction described by event

Implementation

Variables
-msg: JLabel
    -information message notifying the user that the game is paused
-resumeButton, quitButton: JButton
    -buttons labeled “Resume” and “Quit”, respectively

Access Programs
-PausePanel(): PausePanel
    - Creates GUI components (an information message, a “Resume” button and a “Quit” button) and adds them to the panel. Registers the “Resume” and “Quit” buttons (sources) with the PausePanel object (listener)

-actionPerformed(event: ActionEvent): void
    - If the “Resume” button was clicked, the program shows the GamePanel panel in theGame panel of BShipMainMenu. If the “Quit” button was clicked, the program shows the mainMenu panel of theGame panel in BShipMainMenu.

Class: ResultsPanel

This class represents the graphical object shown where the user can see the results of the game.

Interface

Uses
-BShipMainMenu
-InputNamePanel
-StatsIO

Types
-None

Extends/Inherits
-JPanel class

Implements
-ActionListener interface

Access Programs
-ResultsPanel(): ResultsPanel
    -Constructs a new ResultsPanel object
-static setWinner(winner: String): void
    -updates the current winner
-static setLoser(loser: String): void
    -updates the current loser
-addStats(winnner: String): void
    -stores current player's stats
-actionPerformed(event: ActionEvent): void
    -Specifies how the program reacts to user interaction described by event

Implementation

Variables
-winner, loser: static String
    -the winner and loser of the game (either "Computer" or input player's name), respectively
-title: JLabel
    -title of the panel shown to the user
-winnerLabel' loserLabel: JLabel
    -information message showing the winner and loser of the game, respectively
-mainMenuButton: JButton
    -button labeled "Main Menu"

Access Programs
-ResultsPanel(): ResultsPanel

- Creates GUI components (a title, winner information message, loser information message and a “Main Menu” button) and adds them to the panel. Registers the “Main Menu” button (source) with the ResultsPanel object (listener)

-static setWinner(winner: String): void
- Updates the current winner by assigning the input winner to the static field winner and updates the winnerPanel accordingly.

-static setLoser(loser: String): void
- Updates the current loser by assigning the input loser to the static field loser and updates the loserPanel accordingly.

-addStats(winner: String)
- If input player won (computer lost), win = 1 and loss = 0. If computer won, win = 0 and loss = 1. Calls static method storeCurrentStats(win, winner, loss) of StatsIO class.
-*Pseudocode*:

```
if winner = InputNamePanel.getInputName()
 win = 1
 loss = 0
else
 win = 0
 loss = 1
StatsIO.storeCurrentStats(win, winner, loss)
```

-actionPerformed(event: ActionEvent): void
- When the “MainMenu” button is clicked, the program adds the current players stats by calling this.addStats(winner) and the program shows the mainMenu panel in theGame panel of BShipMainMenu.

Class: RulesPanel

This class represents the graphical object shown where the rules of Battleship are displayed.

Interface

Uses
-BShipMainMenu

Types
-None

Extends/Inherits
-JPanel class

Implements
-ActionListener interface

Access Programs
-RulesPanel(): RulesPanel
    -Constructs a new RulesPanel object
-actionPerformed(event: ActionEvent): void
    -Specifies how the program reacts to user interaction described by event

Implementation

Variables
-rule1, rule2, rule3: JLabel
    -the first, second and third rules of the game respectively
-mainMenuButton: JButton
    -button labeled "Main Menu"
Access Programs

-RulesPanel(): RulesPanel
    - Creates GUI components (3 rules and a "Main Menu" button) and adds them to the panel. Registers the "Main Menu" button (source) with the RulesPanel object (listener).

-actionPerformed(event: ActionEvent): void
    -*Algorithm*: When the mainMenuButton was clicked, the program shows the mainMenu panel in theGame panel of BShipMainMenu.

Class: Opponent

This class represents the computer's map (fleet) as a 2-dimensional integer array (10x10). This class also defines the behavior of the computer (artificial intelligence) in terms of guessing where the user's ships are placed.

Interface

Uses
-None

Types
-None

Extends/Inherits
-None

Implements
-None

Access Programs
-Opponent(): void
 -Constructs a new Opponent array object as well as a random number generator.
-generateOpponentMap(): int [ ] [ ]
 -Randomly generates a 2-dimensional integer array representing the computer's map (fleet). Elements in the array with value 0 represent unoccupied spaces, and elements with values x ∈ {1, 2, 3, 4, 5} represent spaces which are occupied by a ship of length x. The generated map will contain 5 ships, each of unique length (1-5). A space on the map cannot belong to more than one ship.
-generateOpponentGuess(boolean: prevHit, int: prevHitRow, int: prevHitColumn, int[ ][ ] Map): int [ ]
 -Guesses indices of the location of player ships using information from previous guesses (prevHit, prevRow, prevHitColumn) and returns the indices as an array where the first element is the row and the second element is the column.

Implementation

Variables
-opponent: int [ ] [ ]
 -Array representation of user's map.
-generator: Random
 -Generates random integers for generateOpponentMap().

Access Programs
-Opponent(): void
 -Constructs a new PlayerMap object. This involves creating the array playerMap, where all elements are initialized to 0.
-generateOpponentMap(): int [ ] [ ]

-To generate a random opponent map, ships are placed individually in increasing order of length. When a ship is placed, first orientation (vertical/horizontal) is determined by generating a bit randomly (0 = horizontal, 1 =vertical). After orientation is determined, random number are generated such that the ship will fit in the map based on orientation and ship length. For example, if a ship of length 3 is being placed and the orientation is determined to be vertical, then the row of the first unit of ship will be determined by generating a random number between 0 and 7 (row cannot exceed 7 or else the ship will be placed off the grid), and the column of the first unit of ship will be determined by generating a random number between 0 and 9 (there are no restrictions on the column because the orientation is vertical). After these indices have been determined, each element containing the ship is checked to determine if another ship has been placed in that index. If so, new indices are generated until the ship does not overlap with any other ships.

Pseudocode

```
generateOpponentMap()
    opponentMap = new int[10][10]
    boolean failcheck
    for(i = 1 to 5){
                failcheck = true
                int oppRow
                int oppColumn
                int oppOrientation
                while(failcheck){
                failcheck=false;
                 oppOrientation = randomBit();
                  if(oppOrientation == 0){
                        oppRow = generator(10)
                        oppColumn = generator(10-i)
                  }
                  else{
                        oppRow = generator(10-i)
                        oppColumn = generator(10)
                  }
                  for(j = 0 to i){
                        if(oppOrientation == 0){
                              if(opponentMap[oppRow][oppColumn+j]>0){
                                    failcheck =true
                              }
                        }
                        else{
                              if(opponentMap[oppRow+j][oppColumn]>0){
                                    failcheck =true
                              }
                       }

                  }

                  if(!failcheck){
                    for(int j = 0 to i){
                        if(oppOrientation == 0){
                          opponentMap[oppRow][oppColumn+j]=i+1
                        }

                        else{
                          opponentMap[oppRow+j][oppColumn]=i+1
                        }
                    }

                }
            }
        }
    return opponentMap
```

-generateOpponentGuess(boolean: prevHit, int: prevHitRow, int: prevHitColumn, int[ ][ ] Map): int [ ]

-This method keeps track of past indices which were indicated as "hits". The row of the last hit is remembered as prevHitRow, and the column of the last hit is remembered as prevHitColumn. The array Map is used to keep track of which indices have already been guessed. The Boolean variable prevHit determines whether the method has "hit" an enemy ship. If the method has not "hit" an enemy ship, it randomly generates two numbers between 0 and 9, one representing the row to guess, and the other representing the column to guess. The indices are checked in Map to determine if they have already been guessed. If so, two new indices are generated until they satisfy the requirement of having not been guessed before. If the method has "hit an enemy ship, it's first alternative is to guess the position immediately west of hit (at position (prevHitRow, prevHitColumn -1)). If this position is off the grid or has already been guessed, then the method chooses to guess the position immediately north of the hit (at position (prevHitRow - 1, prevHitColumn)). If this position is off the grid or has already been guessed, then the method chooses to guess the position immediately east of the hit (at position (prevHitRow, prevHitColumn +1)). If this position is off the grid or has already been guessed, the method chooses to guess the position immediately south of the hit (at position (prevHitRow +1, prevHitColumn)). If this position is off the grid or has already been guessed, the method randomly generated indices in the same fashion as it would if the method had not "hit" an enemy ship until it "hits" an enemy ship

<u>Pseudocode</u>

```
generateOpponentGuess(boolean prevHit, int prevHitRow, int prevHitColumn,
   int[][] Map)
     int[ ] indices = new int[10]
    int compRow
    int compColumn
    if(!prevHit){


    do{
            compRow = generator(10)
```

```
          compColumn = generator(10)
          }while(Map[compRow][compColumn]==alreadyGuessed)
}
else{
   if(((prevHitColumn-1)>=0) &
            (playerMap[prevHitRow][prevHitColumn-1]!=alreadyGuessed)){
      compRow = prevHitRow
      compColumn = prevHitColumn-1

   }
   else if(((prevHitRow-1)>=0) &
            (playerMap[prevHitRow-1][prevHitColumn]!=alreadyGuessed)){
      compRow = prevHitRow-1
      compColumn = prevHitColumn

   }
   else if(((prevHitColumn+1)<=9) &
       (playerMap[prevHitRow][prevHitColumn+1]!=alreadyGuessed)){
      compRow = prevHitRow
      compColumn = prevHitColumn+1

   }
   else if(((prevHitRow+1)<=9)&
       (playerMap[prevHitRow+1][prevHitColumn]!=alreadyGuessed)){
      compRow = prevHitRow+1
      compColumn = prevHitColumn

   }
   else{
      do{
                compRow = generator(10)
               compColumn = generator(10)
         }while(playerMap[compRow][compColumn]==alreadyGuessed)

   }
}
indices[0]=compRow
indices[1]=compColumn
return indices
```

Class: PlayerMap

This class represents the user's map (fleet) as a 2-dimensional integer array (10x10). Elements in the array with value 0 represent unoccupied spaces, and elements with values x ∈ {1, 2, 3, 4, 5} represent spaces which are occupied by a ship of length x.

Interface

Uses
-None

Types
-None

Extends/Inherits
-None

Implements
-None

Access Programs
-PlayerMap(): void
-Constructs a new PlayerMap array object.
-setMap(int: startRow, int: startColumn, int: length, String: Orientation): void
-Sets elements in PlayerMap from position (startRow, startColumn) to (startRow, startColumn + length) to value length if Orientation is "horizontal", and sets elements from position (startRow, startColumn) to (startRow + length, startColumn) to value length if Orientation is "vertical".
-resetMap(): void
-Sets all elements in the PlayerMap array to zero.
-getMap(): int[ ][ ]
-Returns the PlayerMap array.
-checkOverlap(int: startRow, int: startColumn, int: length, String: Orientation): boolean
- If Orientation is "horizontal", checks if any elements in PlayerMap from position (startRow, startColumn) to (startRow, startColumn + length) have value not equal to zero. If Orientation is "vertical", checks if any elements in PlayerMap from position (startRow, startColumn) to (startRow + length, startColumn) have value not equal to zero.

Implementation

Variables
-playerMap: int [ ] [ ]
-Array representation of user's map.

Access Programs

-Playermap(): void

-Constructs a new PlayerMap object. This involves creating the array playerMap, where all elements are initialized to 0.

-setMap(int: startRow, int: startColumn, int: length, String: Orientation): void

-Sets elements in PlayerMap from position (startRow, startColumn) to (startRow, startColumn + length) to value length if Orientation is “horizontal”, and sets elements from position (startRow, startColumn) to (startRow + length, startColumn) to value length if Orientation is “vertical”.

<u>Pseudocode:</u>

```
setMap(int: startRow, int: startColumn, int: length, String: Orientation)
   if(Orientation is horizontal) {
     for(i = 1:length){
        playerMap(startRow, startColumn + i) = length
     }
   }
  if(Orientation is vertical) {
     for(i = 1:length){
        playerMap(startRow + i, startColumn) = length
     }
   }
```

-getMap(): int[ ][ ]

-Returns the PlayerMap array.

-checkOverlap(int: startRow, int: startColumn, int: length, String: Orientation): boolean

- If Orientation is “horizontal”, checks if any elements in PlayerMap from position (startRow, startColumn) to (startRow, startColumn + length) have value not equal to zero. If Orientation is “vertical”, checks if any elements in PlayerMap from position (startRow, startColumn) to (startRow + length, startColumn) have value not equal to zero.

<u>Pseudocode:</u>

```
checkOverlap(int: startRow, int: startColumn, int: length, String: Orientation)
  Boolean overlap = false
   if(Orientation is horizontal) {
     for(i = 1:length){
        if(playerMap(startRow, startColumn + i) > 0) {
          overlap = true
        }
     }
   }
  if(Orientation is vertical) {
     for(i = 1:length){
        if(playerMap(startRow + i, startColumn) > 0) {
          overlap = true
        }
     }
   }
  Return overlap
```

Class: GameSound

This class creates a sound object

<u>Interface</u>

Uses

- None

Types

- None

Extends/Inherits

- None

Implements

- AudioSystem

- AudioInputStream
- Clip

Access Programs

- music(): void
  - Constructs a new thread and outputs the background music
- explosion(): void
  - Constructs a new thread and outputs an explosion sound.

<u>Implementation</u>

Variables

- explosion: boolean
  - if true the explosion sound will be created
- on: boolean
  - if true background music will play

Access Programs

- music(): void

    - Creates a new Clip using the AudioSystem and Clip class. The AudioInputStream is gathered from the song.wav file, and is called on using song.start().
- explosion():void
    - Creates a new Clip using the AudioSystem and Clip class. The AudioInputStream is gathered from the bomb2.wav file, and is called on using bomb.start().
    - The explosion bomb.start() is only called when the boolean value of explosion is true.

Class: StatsPanel

Interface

Uses

- None

Types

- None

Extends/Inherits

- JPanel

Implements

- ActionListener

Access Programs

-populateStats(): void

    -Fills the Stats GUI with the top three stored players, sorted by number of wins.

Implementation

Uses:

- StatsIO
- javax.swing
- java.awt
- java.io
- java.util

Variables:

- lblFirstPlace, lblSecondPlace, lblThirdPlace, wins1, losses1, wins2, losses2, wins3, losses 3: JLabel
    - Labels for first, second, and third places, and their respective wins and losses
- button, button_1: JButton
    - Two button names for the "Clear Stats" buttons on the GUI

Access Programs

-populateStats(): void

    Inputs: None
    Outputs: None

Updates: lblFirstPlace, lblSecondPlace, lblThirdPlace, wins1, losses1, wins2, losses2, wins3, losses3: JLabel

Pseudocode:

```
 populateStats()
  StatsIO.readStats()
  statsFiller()
  bubbleSort(StatsIO.storedWins, StatsIO.storedWins.size())

   Set the text of lblFirstPlace, lblSecondPlace, lblThirdPlace, wins1, losses1,
wins2,       losses2, wins3, losses3 from the statistics that have been read, in increasing
order
end function
```

Local Programs

-bubbleSort(a: ArrayList<Integer>; n: Integer): void

Inputs: a: ArrayList<Integer>; n: Integer

Outputs: None

Updates: StatsIO.storedNames: ArrayList<String>; StatsIO.storedWins: ArrayList<Integer>; StatsIO.storedLosses<Integer>

Pseudocode:

```
bubbleSort(ArrayList<Integer> a, int n)
   for (i=0 to n)
      for (j=0 to n-i)
         if (a(j-1) > a(j))
            t = a(j-1);
            t2 = StatsIO.storedNames(j-1);
            t3 = StatsIO.storedLosses(j-1);
            set a(j) to a(j-1);
            set StatsIO.storedNames(j) to  StatsIO.storedNames(j-1);
            set StatsIO.storedLosses(j) to StatsIO.storedLosses(j-1);
            set a(j) to t;
            set StatsIO.storedNames(j) to t2;
            set StatsIO.storedLosses(j) to t3;
      end for
   end for
end function
```

-statsFiller(): void

Inputs: None

Outputs: None

Updates: StatsIO.storedNames: ArrayList<String>; StatsIO.storedWins: ArrayList<Integer>; StatsIO.storedLosses<Integer>

Pseudocode:

```
 arrayFiller()
            if no player statistics exist
```

```
                add three “No Player” entries to StatsIO.storedNames and
                three “0” entries to     StatsIO.storedWins and
                StatsIO.storedLosses;
        else if only 1 player has stored statistics
                add two “No Player” entries to StatsIO.storedNames and two
                “0” entries to     StatsIO.storedWins and StatsIO.storedLosses;
        else if only 2 players have stored statistics
                add one “No Player” entry to StatsIO.storedNames and one “0”
                entry to     StatsIO.storedWins and StatsIO.storedLosses;
end function
```

Class: StatsIO

Interface

Uses
- None

Types
- None

Extends/Inherits
- None

Implements
- None

Access Programs

-storeCurrentStats (wins: IN integer; name: IN String; losses: IN integer): void
Stores the statistics of the player that is currently playing the game along with their win or loss for that game.

-readStats (): void
Retrieves and updates the stored statistics of all the players

-playerExists (name: IN String): boolean
Checks whether a player has previously played the game. The method returns true if the player exists and false if it is the first time the player is playing the game.

-storeStats (name: IN String): void
Stores the statistics of all the players to include the player that is currently playing the game.

-removePlayer (name: IN String): void
Removes a player and their statistics given their name.

-updateStatsFile(): void
Writes the stored statistics for storage to be retrieved in later instances of the game.

Implementation

Uses
- java.io
- java.util

Variables
- storedNames: ArrayList

    - Stores names (String) of players in an array list
- storedWins: ArrayList
    - Stores number (Integer) of wins in an array list
- storedLosses: ArrayList
    - Stores number (Integer) of losses in an array list

Access Programs

-storeCurrentStats (wins: IN integer; name: IN String; losses: IN integer): void

Inputs: wins: integer; name: String; losses: integer
Outputs: None
Updates: currentStats: Arraylist
readStats (): void
Inputs: None
Output: None
Updates: storedNames: ArrayList (String) , storedWins: ArrayList (Integer), storedLosses: ArrayList (Integer)
Pseudocode:

```
readstats()
        clear storedNames, storedLosses, and storedWins;
        read file “stats.txt”
         while(a next line exists)
           add first element to storedWins;
                add second element to storedNames;
                add third element to storedWins;
   close file
```

-playerExists (name: IN String): boolean

Inputs: name: String
Outputs: playerExists: boolean
Updates: None
Pseudocode:

```
playerExists (String name)
           boolean playerExists = false;
        for i=0 to size of storedNames
                      if storedNames[i] = name
                                 playerExists = true;
           end for
           return playerExists;
end function
```

-storeStats (name: IN String): void

Inputs: name: String
Outputs: None
Updates: storedNames: Arraylist (String); storedWins: Arraylist (Integer); storedLosses: Arraylist(Integer);
Pseudocode:

```
storeStats (String name)
        call readStats();
        if playerExists(name) = true
```

```
                int playerIndex = getPlayerIndex(name);
               set storedWins to currentStats;
                    set storedLoses to currentStats;
                    for i=0 to size of storedNames
                            write (storedWins at i + "|" + storedNames at i + "|" +
                            storedLosses at i) to stats.txt;
                            close file;
                      end for
             else
                    increment size of storedNames, set storedNames to currentStats[1];
                    increment size of storedNames, set storedWins to currentStats[1];
                    increment size of storedNames ,set storedLosses to currentStats[1];
                    for i=0 to storedNames.size()
                            write (storedWins at i + "|" + storedNames at i + "|" +
                            storedLosses at i) to stats.txt;
                            close file;
                    end for
       end function
```

-removePlayer (name: IN String): void

```
       Inputs: name: String
       Outputs: None
       Updates:  storedNames: Arraylist; storedWins: Arraylist; storedLosses: Arraylist
       Pseudocode:
       remove(String name)
          for  i=0 to storedNames.size()
             if name == storedNames[i]
                storedNames.remove(i)
                storedWins.remove(i)
                storedLosses.remove(i)

                Trim the array to its new size
             end if
          end for
       end function
```

-updateStatsFile(): void

```
       Inputs: None
       Outputs: None
       Updates: storedNames: ArrayList (String); storedWins: ArrayList (Integer);
       storedLosses: ArrayList (Integer)
       Pseudocode:
       updatesStatsFile()
          read "stats.txt";
          for (i=0 to size of storedNames)
             write (storedWins[i] + "|" + storedNames[i] + "|" + storedLosses[i]);
             move to next line;
```

```
        end for
        close file;
    end function
```

Local Programs

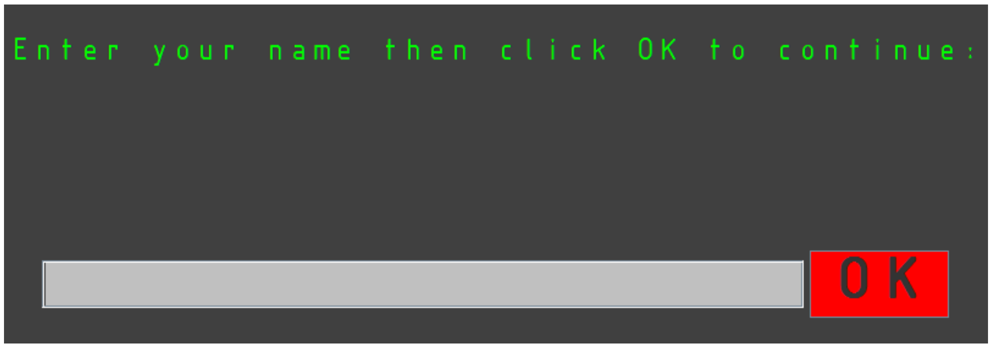
-getPlayerIndex (name: IN String): int

```
    Inputs: name: String
    Outputs: index: integer
    Updates: None
    Pseudocode:
    getPlayerIndex (String name)
            for i=0 to size of storedNames
               if storedNames[i] = name
                          return i;
            end for
        return -1;
    end function
```

Testing

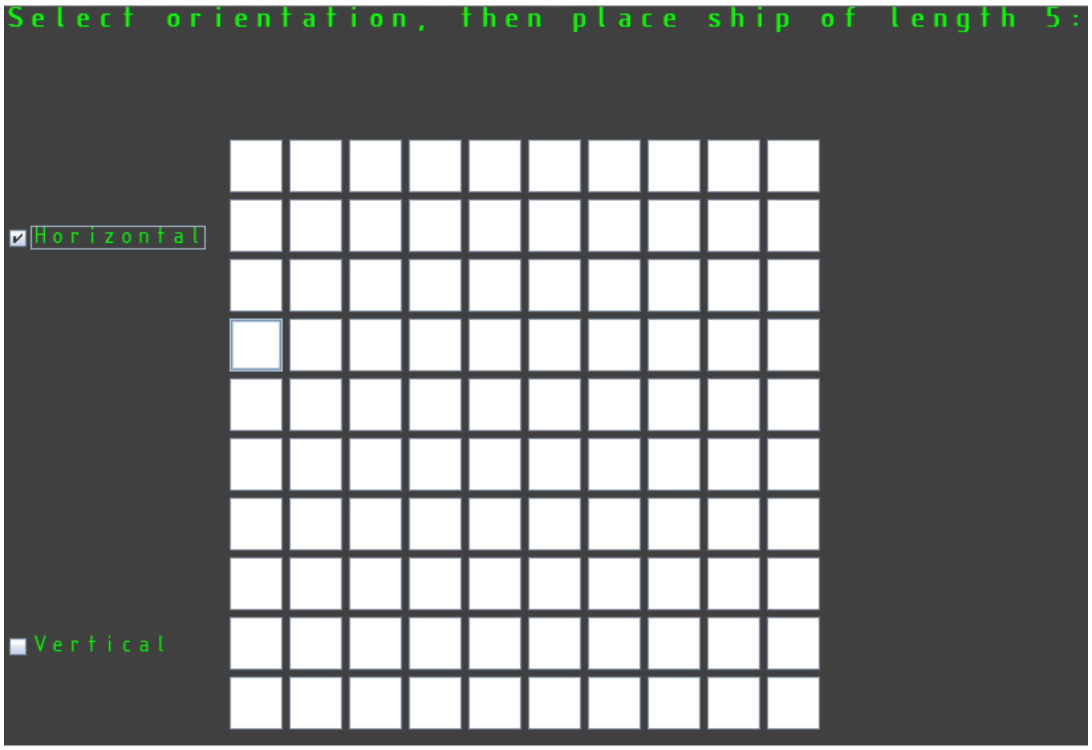
InputNamePanel Test

Constructor Call Test
-Create a new InputNamePanel panel
-Expected Output: Input Panel panel shown
-Actual Output:



-Result: Pass

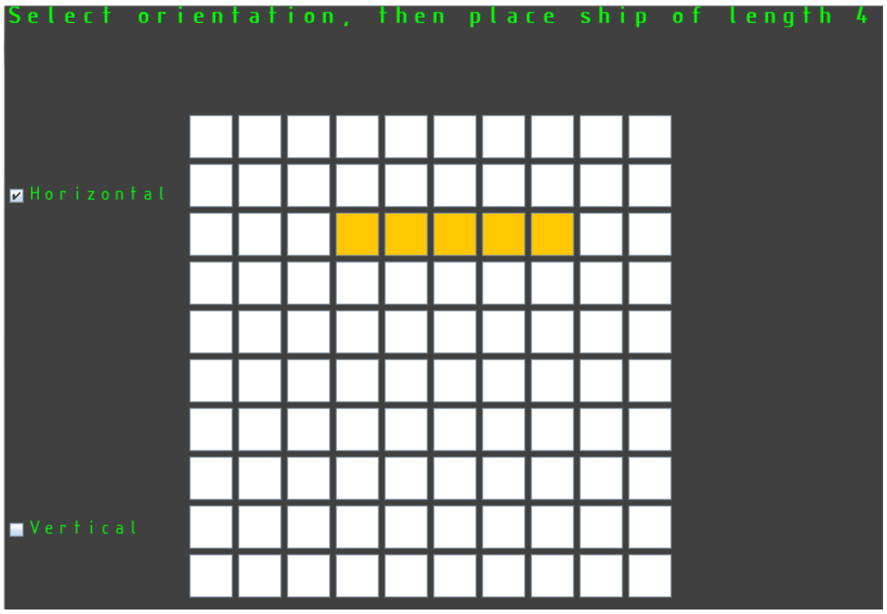
Ok Button Test
-Click OK button after inputting name
-Expected Output: FleetPanel panel shown
-Actual Output:



-Result: Pass

FleetPanel Test

Horizontal ship test
-place a horizontal ship of length 5 in the fleet starting at row 3 column 4
-Expected Output: Highlighted horizontal segment from row 3 column 4 to row 3 column 8
-Actual Output:



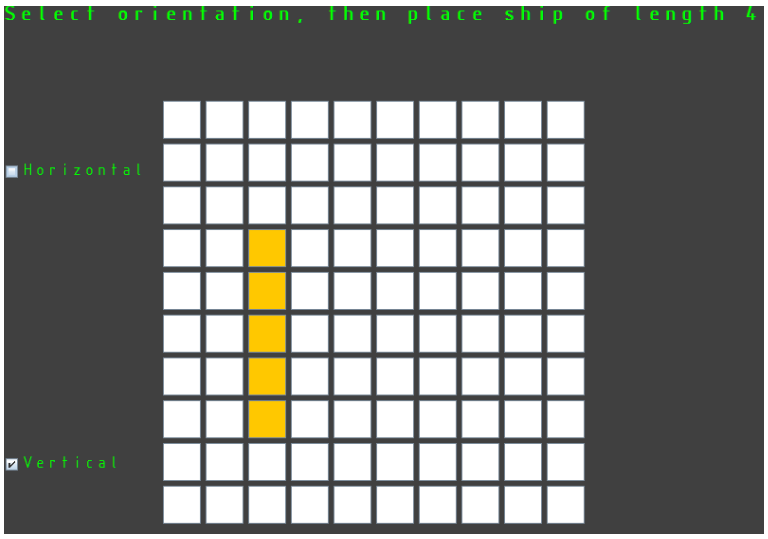
-Result: Pass

Vertical ship test

-place a vertical ship of length 5 in the fleet starting at row 4 column 3

-Expected Output: Highlighted vertical segment from row 4 column 3 to row 8 column 3

-Actual Output:



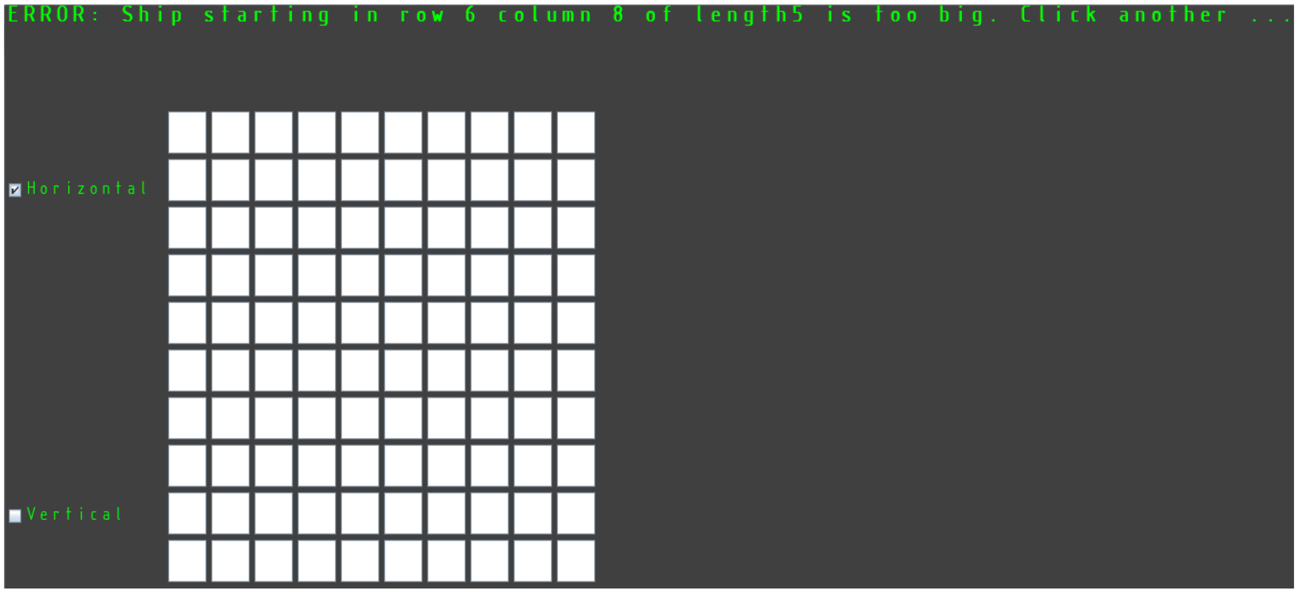
-Result: Pass

Invalid Ship Placement Test

-place a horizontal ship of length 5 starting in row 6 column 8

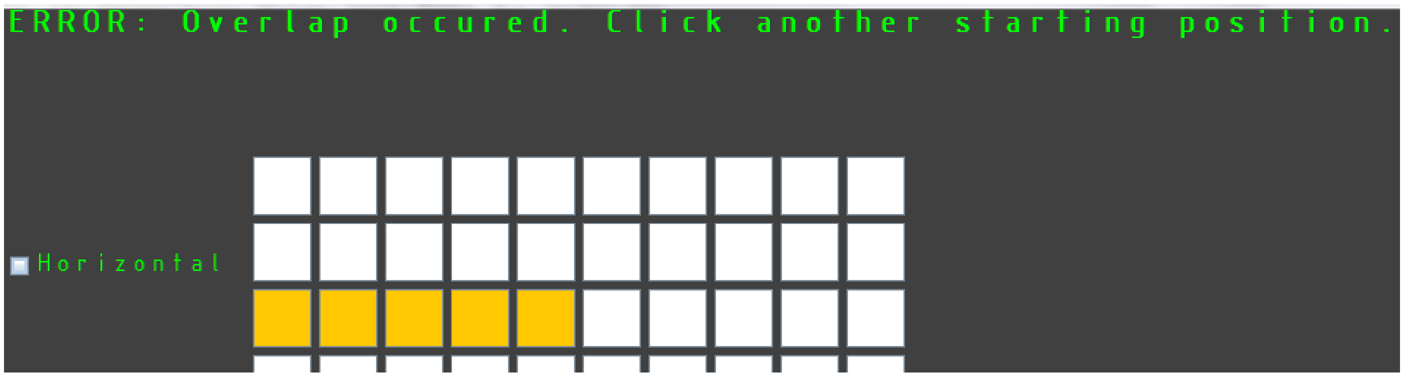
-Expected Output: Error message displayed

-Actual Output:


ERROR: Ship starting in row 6 column 8 of length5 is too big. Click another ...

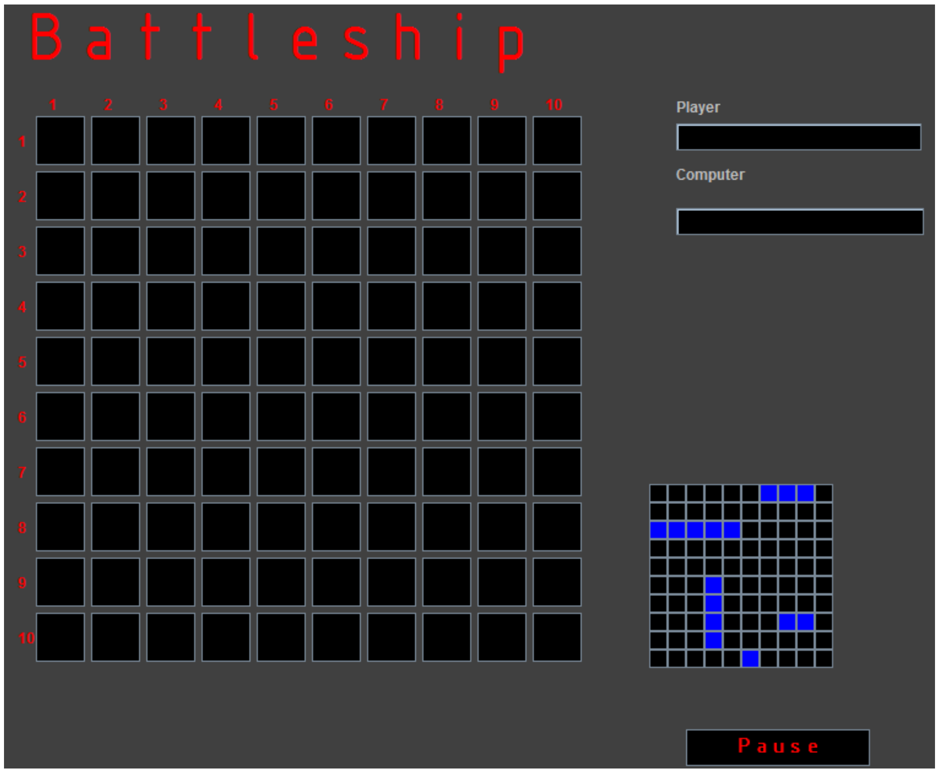
Result: Pass

Overlapping Ship Test
-place a vertical ship of length 4 starting at row 3 column 3 after placing a horizontal ship of length 5 starting at row 3 column 1
-Expected Output: Overlap error message displayed
-Actual Output:


ERROR: Overlap occured. Click another starting position.

Result: Pass

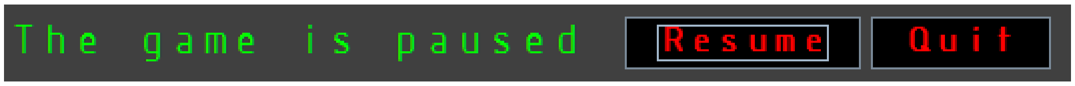
Done Button Test
-after organizing a valid fleet of 5 ships, click on the “Done” button
-Expected Output: GamePanel panel shown
-Actual Output:



-Result: Pass

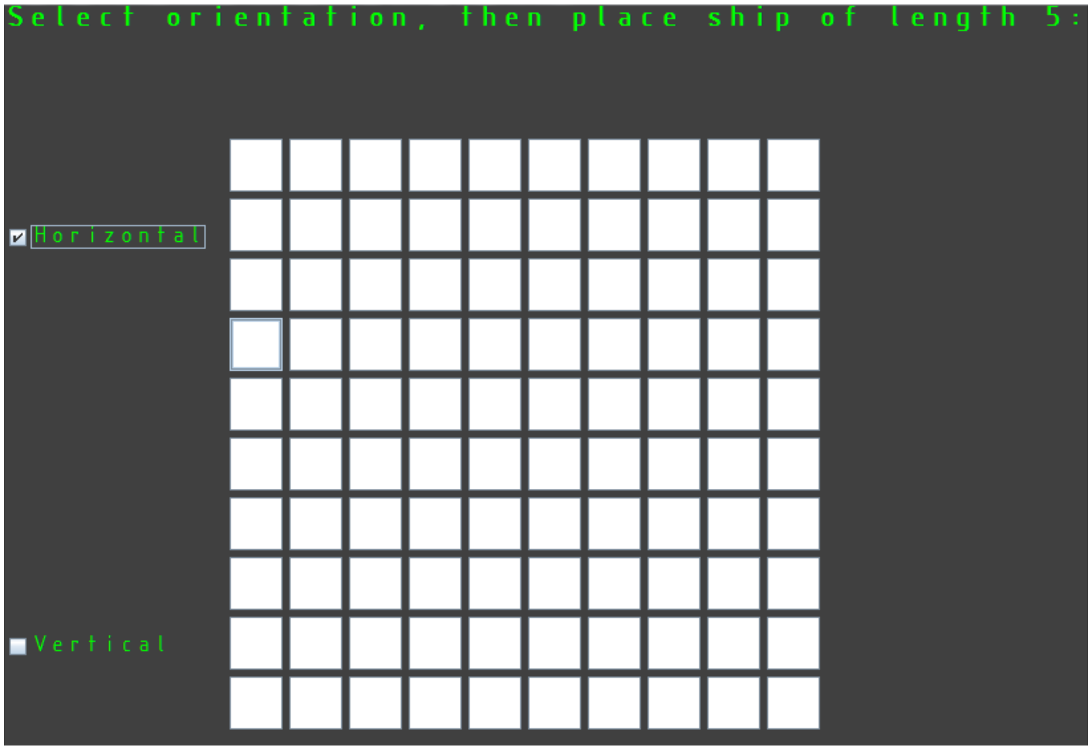
PausePanel Test

Constructor Call Test
-Create a new PausePanel panel
-Expected Output: Input Panel panel shown
-Actual Output:



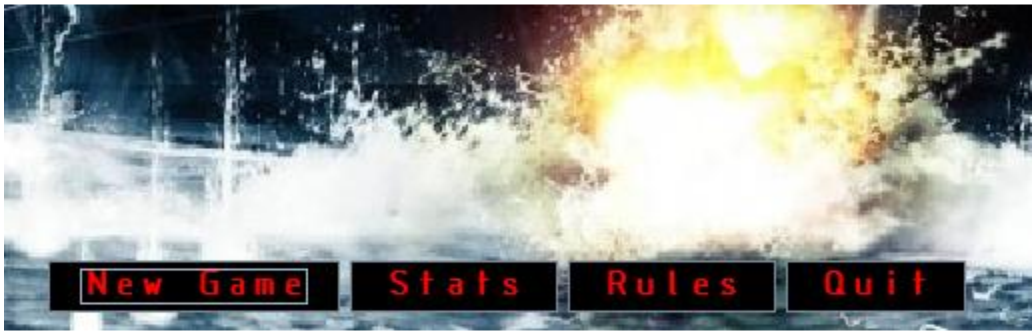
-Result: Pass

Resume Button Test
-Click Resume button
-Expected Output: FleetPanel panel shown
-Actual Output:


Select orientation, then place ship of length 5:

-Result: Pass

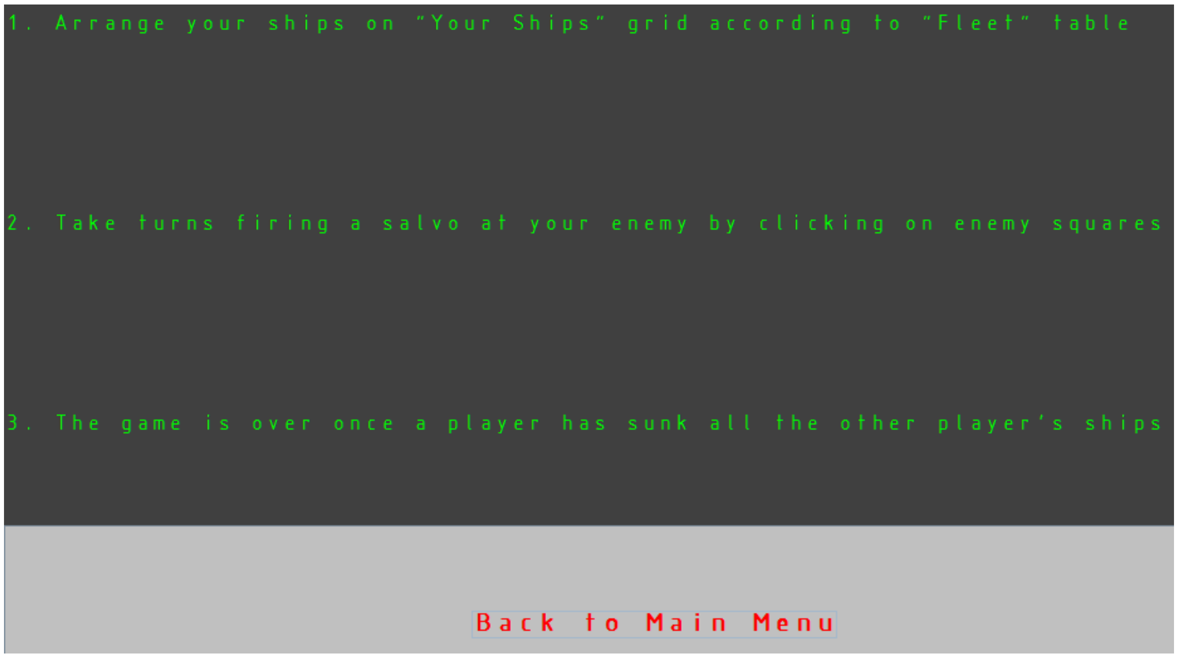
Quit Button Test
-Click Quit button
-Expected Output:main menu panel shown
-Actual Output:



-Result: Pass

RulesPanel Test

Constructor Call Test
-Create a new RulesPanel panel
-Expected Output: RulesPanel panel shown
-Actual Output:



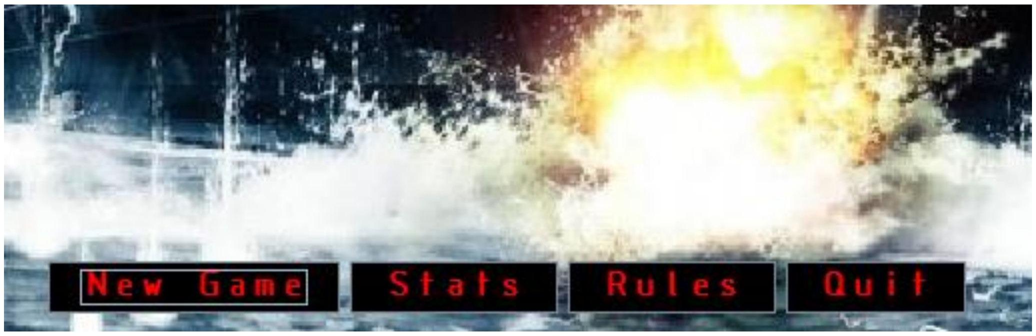
1. Arrange your ships on "Your Ships" grid according to "Fleet" table

2. Take turns firing a salvo at your enemy by clicking on enemy squares

3. The game is over once a player has sunk all the other player's ships

-Result: Pass

Back to Main Menu Button Test
-Click Back to Main Menu Button
-Expected Output: main menu panel shown
-Actual Output:

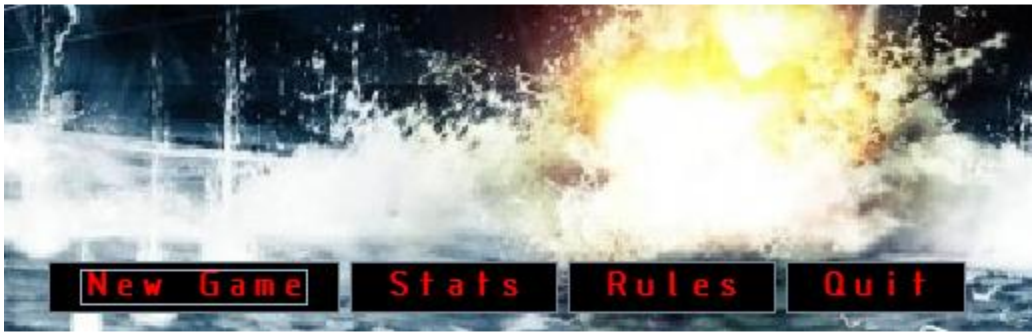


-Result: Pass

BShipMainMenu Test

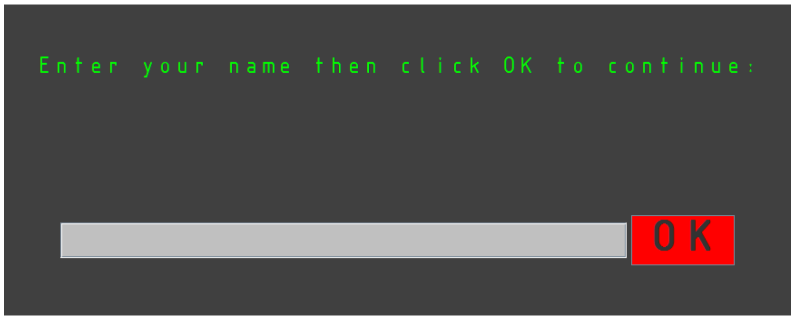
These tests run BShipMainMenuClient to test the BShipMainMenu class.

Constructor Call Test
-create new BShipMainMenu frame
-Expected Output: main menu panel shown
-Actual Output:



-Result: Pass

New Game Button Test
-click new game button
-Expected Output: InputNamePanel panel shown so user can enter their name to start a new game
-Actual Output:

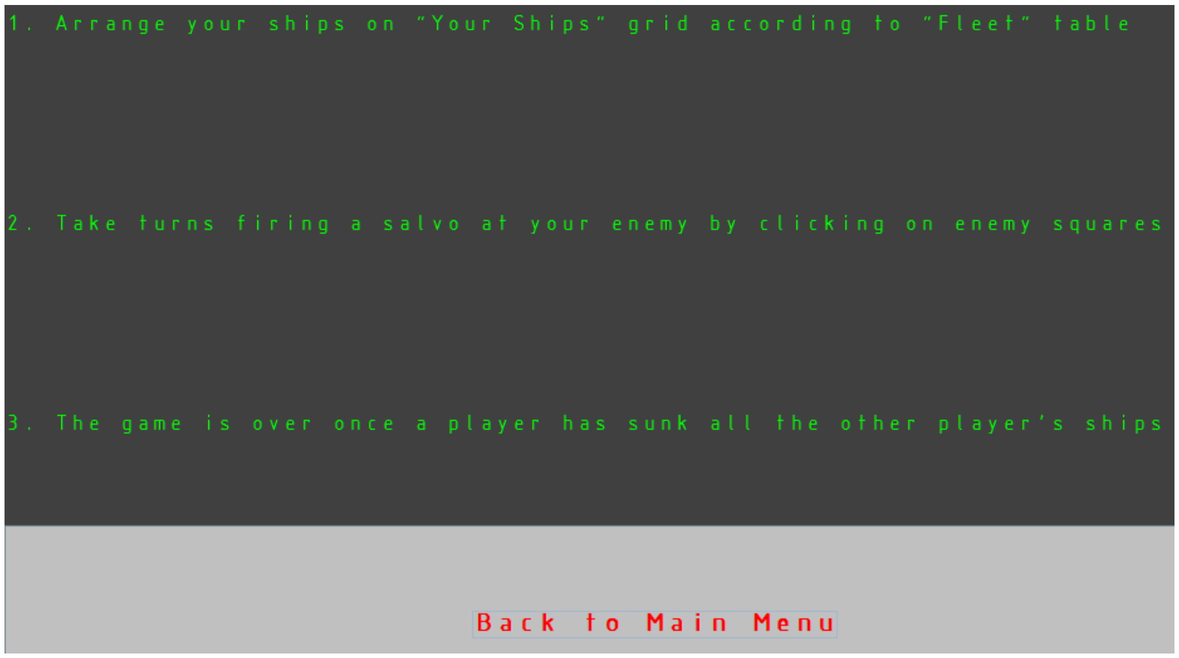


-Result: Pass

Stats Button Test
-click new game button
-Expected Output: StatsPanel panel shown
-Actual Output:

-Result: Pass

Rules Button Test
-click rules button
-Expected Output: RulesPanel panel shown
-Actual Output:



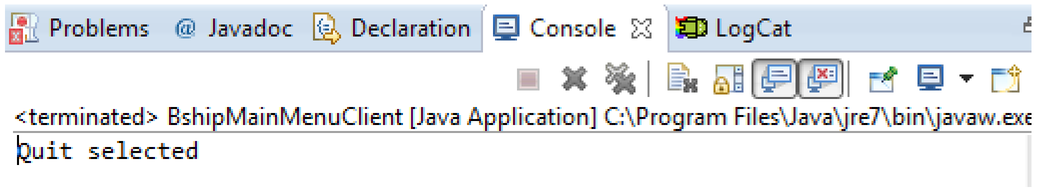
1. Arrange your ships on "Your Ships" grid according to "Fleet" table

2. Take turns firing a salvo at your enemy by clicking on enemy squares

3. The game is over once a player has sunk all the other player's ships

Result: Pass

Quit Button Test
-click rules button
-Expected Output: GUI closed and program terminated
-Actual Output:



```
<terminated> BshipMainMenuClient [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Quit selected
```

Result: Pass

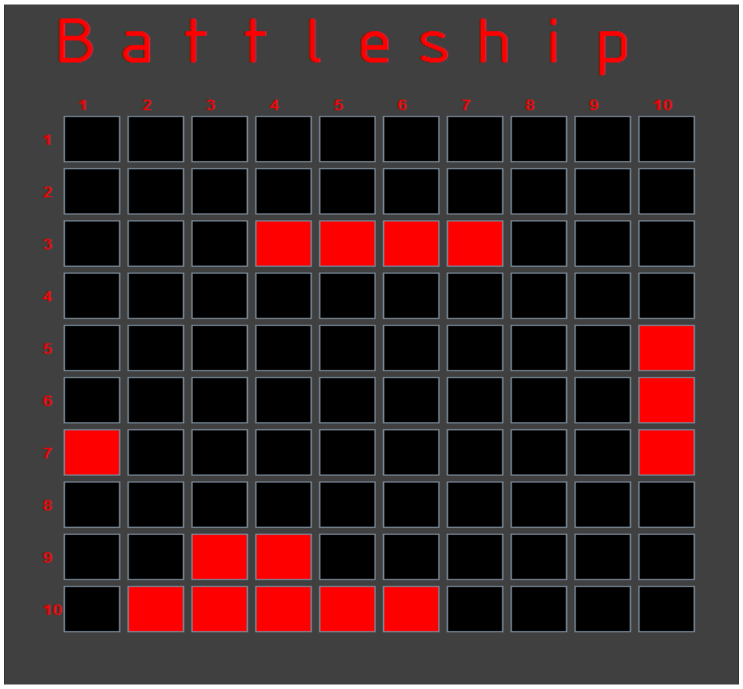
Opponent Test

generateOpponentMap( ) test

-Each generated map should contain 5 ships of unique length x ∈ {1, 2, 3, 4, 5}, and no ships can overlap spaces.

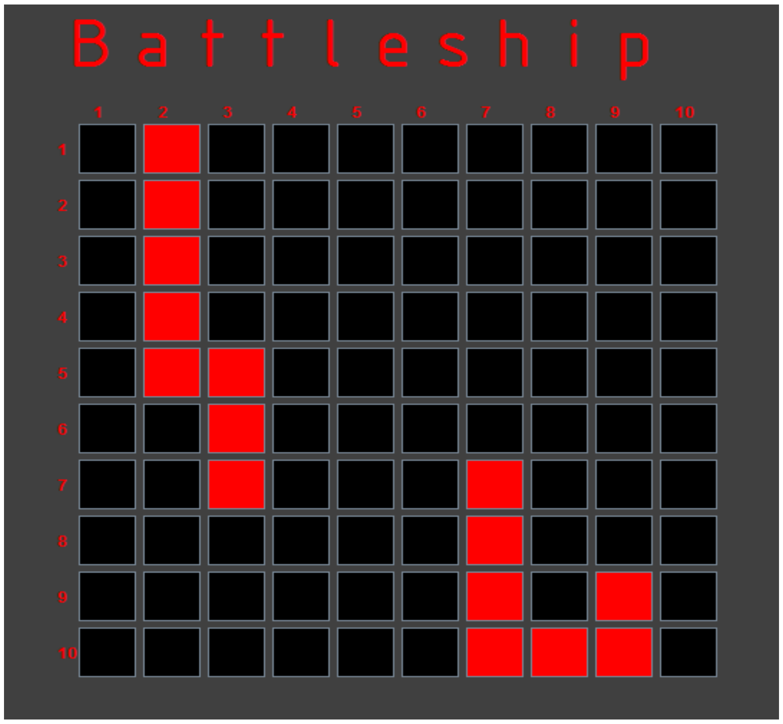
-For the sake of testing, the generated opponent map will be visible (opponent ship locations coloured red) on initialization of gamePanel.

Test 1



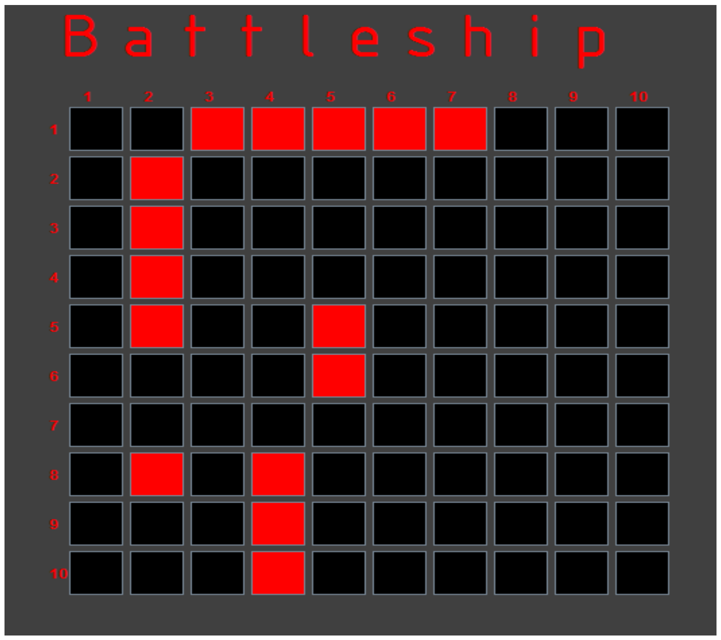
Result: Pass

Test 2



Result: Pass

Test 3

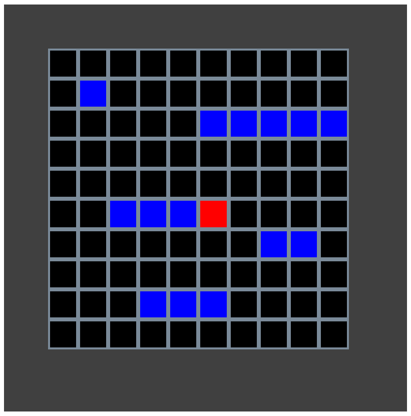


Result: Pass

generateOpponentGuess( ) test

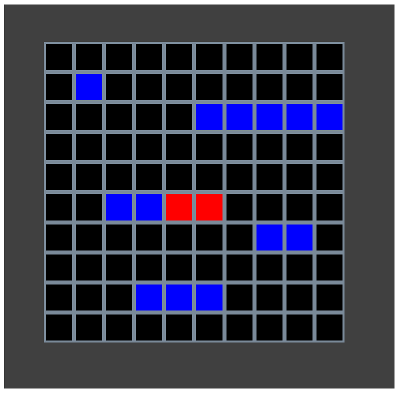
-In this section, we will examine the first 20 opponent guesses to test our guessing algorithm. Expected guess indices, actual guess indices and results will be established after every guess.

<u>Guess 1</u>



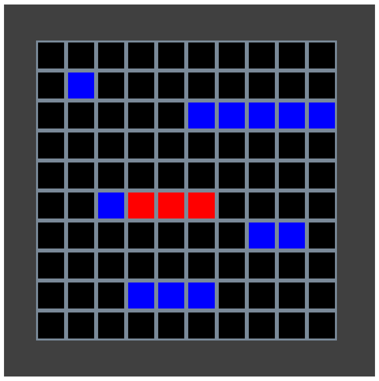
Expected guess indices: (random, random)
Actualy guess indices: (5, 5)
Result: Pass

<u>Guess 2</u>



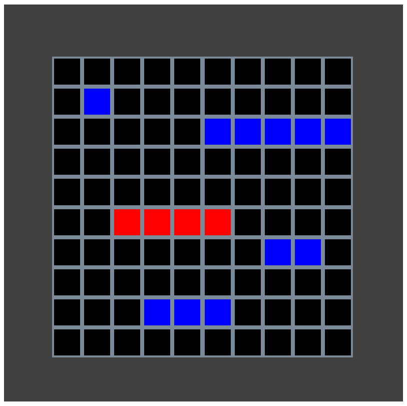
Expected guess indices: (5, 4)
Actualy guess indices: (5, 4)
Result: Pass

<u>Guess 3</u>



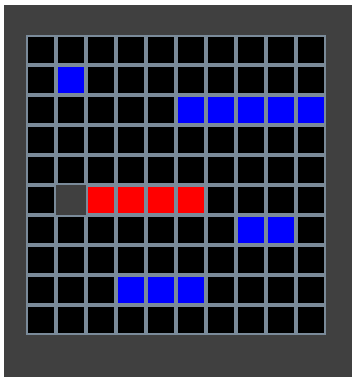
Expected guess indices: (5, 3)
Actualy guess indices: (5, 3)
Result: Pass

Guess 4



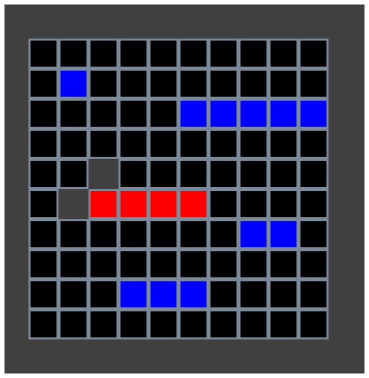
Expected guess indices: (5, 2)
Actualy guess indices: (5, 2)
Result: Pass

Guess 5



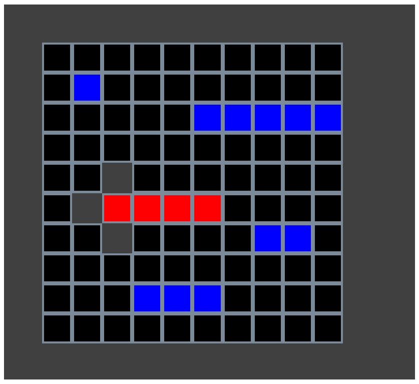
Expected guess indices: (5, 1)
Actualy guess indices: (5, 1)
Result: Pass

Guess 6



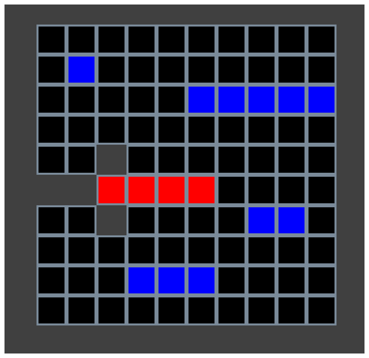
Expected guess indices: (4, 2)
Actualy guess indices: (4, 2)
Result: Pass

Guess 7



Expected guess indices: (6, 2)
Actualy guess indices: (6, 2)
Result: Pass

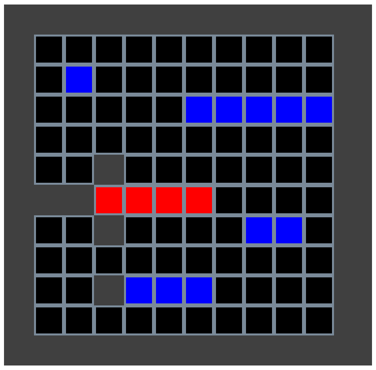
Guess 8



Expected guess indices: (random, random)

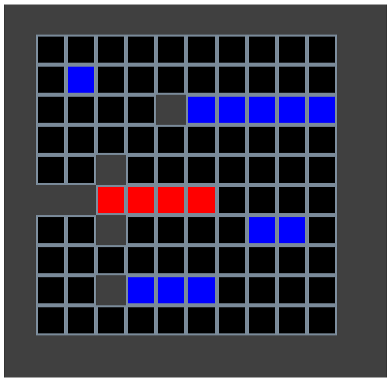
Actualy guess indices: (5,0)
Result: Pass

Guess 9



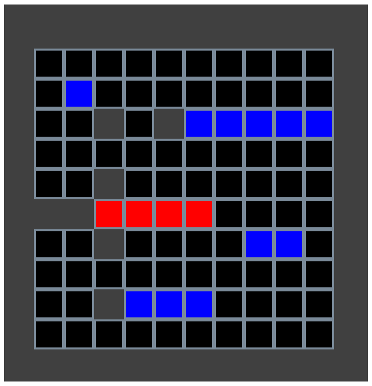
Expected guess indices: (random, random)
Actualy guess indices: (8,2)
Result: Pass

Guess 10



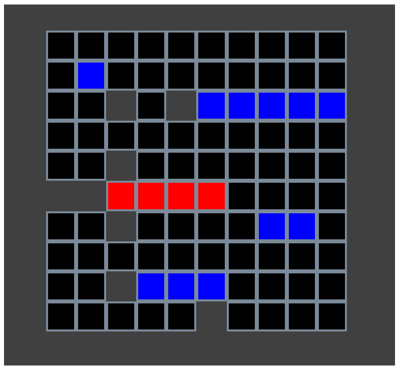
Expected guess indices: (random,random)
Actualy guess indices: (2,4)
Result: Pass

Guess 11



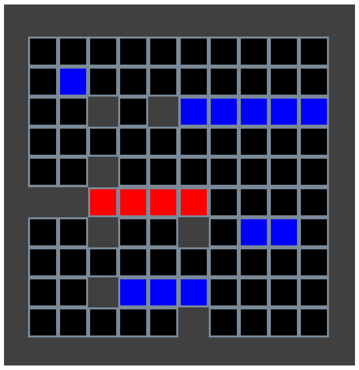
Expected guess indices: (random, random)
Actualy guess indices: (2, 2)
Result: Pass

Guess 12



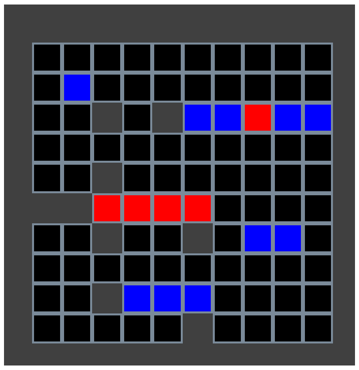
Expected guess indices: (random, random)
Actualy guess indices: (9, 5)
Result: Pass

Guess 13



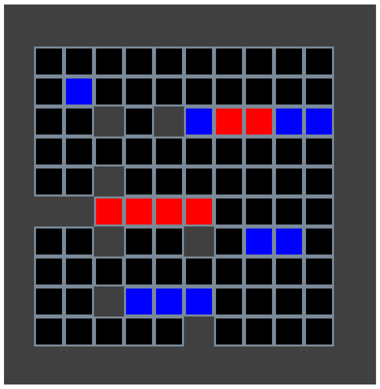
Expected guess indices: (random, random)
Actualy guess indices: (6,5)
Result: Pass

Guess 14



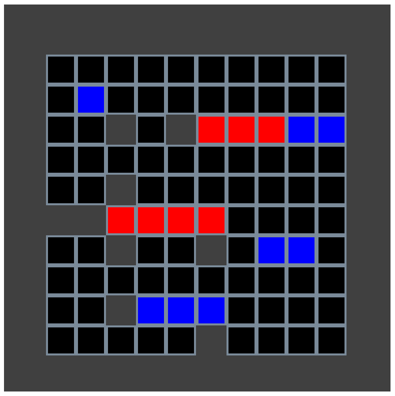
Expected guess indices: (random,random)
Actualy guess indices: (2, 7)
Result: Pass

Guess 15



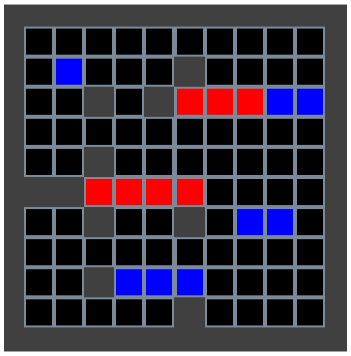
Expected guess indices: (2, 6)
Actualy guess indices: (2, 6)
Result: Pass

<u>Guess 16</u>



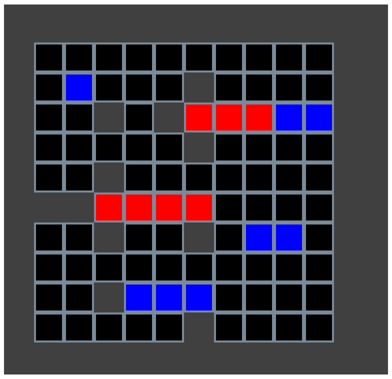
Expected guess indices: (2, 5)
Actualy guess indices: (2, 5)
Result: Pass

<u>Guess 17</u>



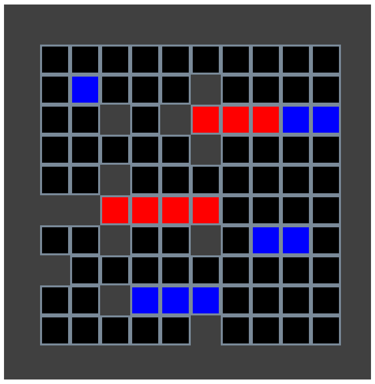
Expected guess indices: (1, 5)
Actualy guess indices: (1, 5)
Result: Pass

<u>Guess 18</u>



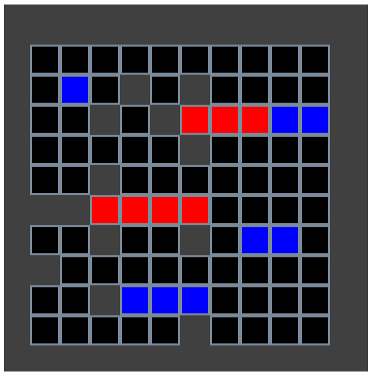
Expected guess indices: (3, 5)
Actualy guess indices: (3, 5)
Result: Pass

<u>Guess 19</u>



Expected guess indices: (random, random)
Actualy guess indices: (7, 0)
Result: Pass

<u>Guess 20</u>



Expected guess indices: (random, random)
Actualy guess indices: (1, 3)
Result: Pass

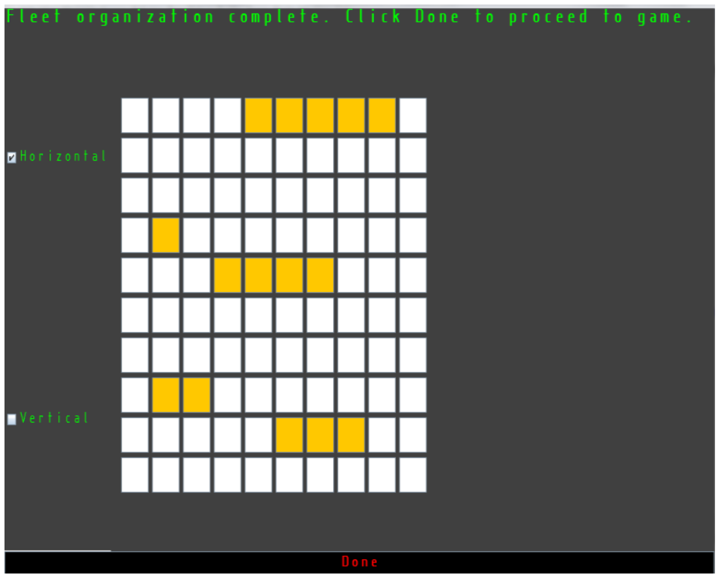
PlayerMap Test

setMap( ) test

-The setMap() method is used in the fleetPanel class to initialize the player Map. The player Map is displayed on the gamePanel class panel in the bottom right corner. To test the setMap() method, 3 different maps will be initialized in fleetPanel and compared to the map displayed in the gamePanel panel.

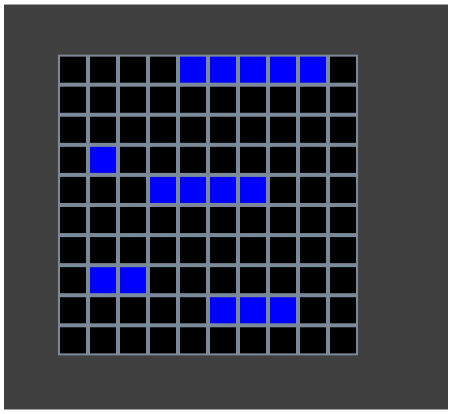
Test 1

Initialization:



Fleet organization complete. Click Done to proceed to game.

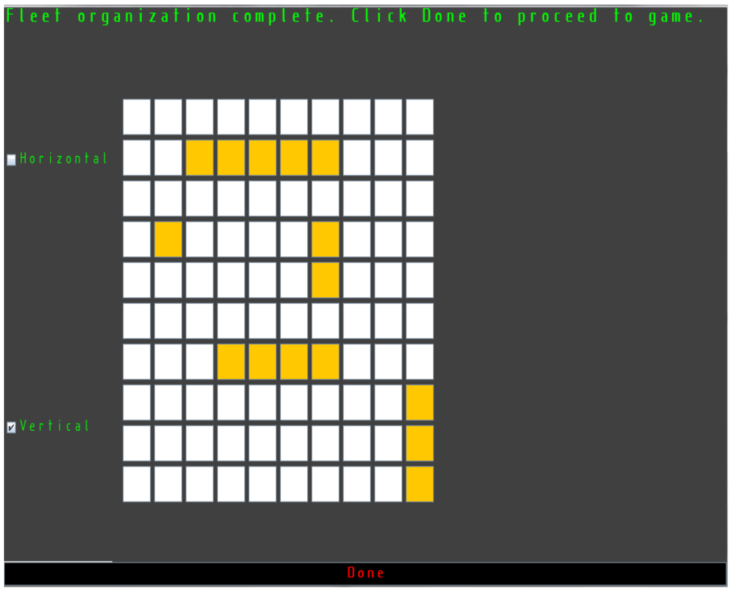
Output:



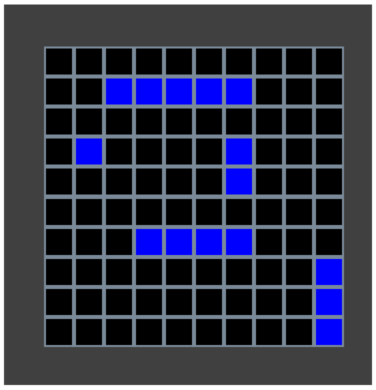
Result: Pass

Test2

Initialization:


Fleet organization complete. Click Done to proceed to game.

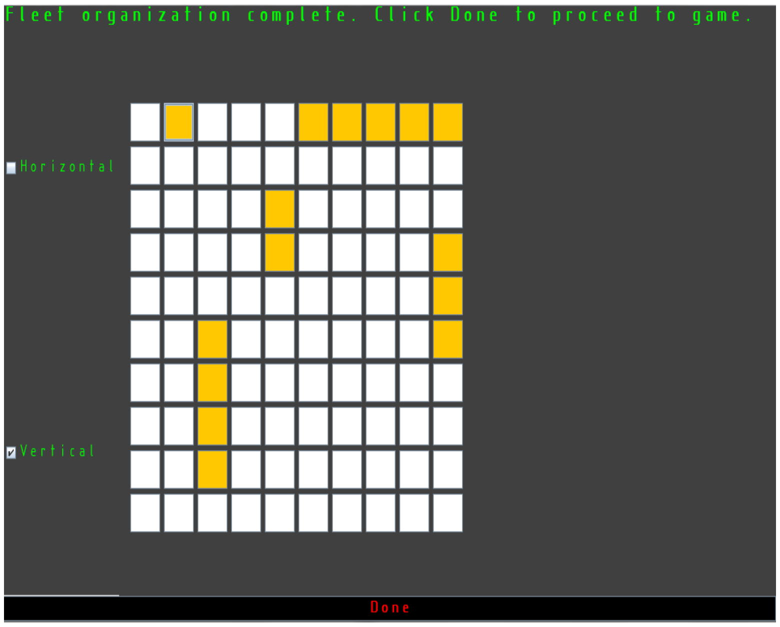
Output:



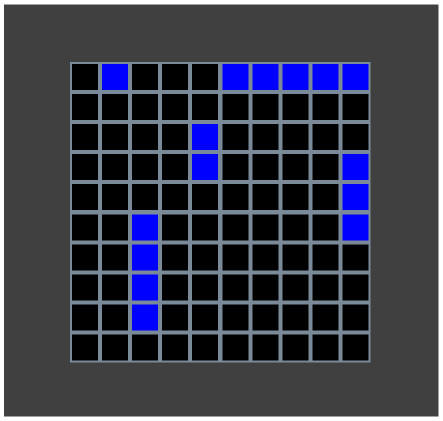
Result: Pass

<u>Test3</u>

Initialization:


Fleet organization complete. Click Done to proceed to game.

Output:



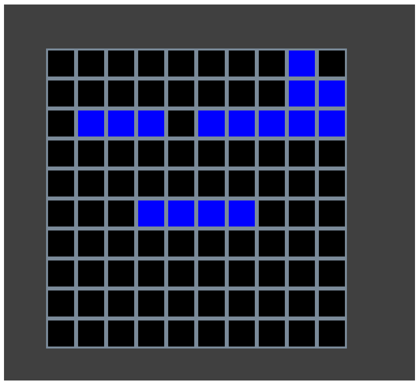
Result: Pass

resetMap( ) test

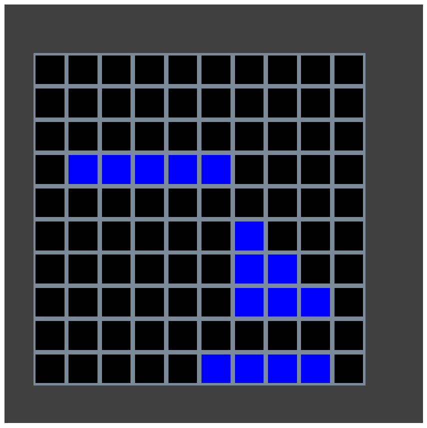
-The resetMap() method is used on initialization of FleetPanel to clear previously created maps. To test this method, 2 consecutive new games will be run and the second map that is created will be check to ensure it was cleared before it was initialized.

<u>Test 1</u>

Map 1:



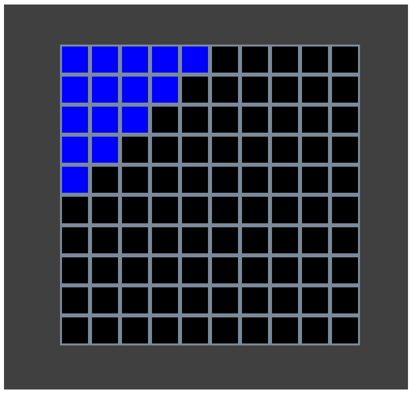
Map 2:



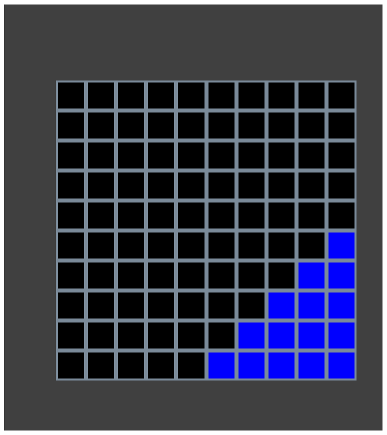
Result: Pass

<u>Test 2</u>

Map 1:



Map 2:



Result: Pass

checkOverlap( ) test

-The checkOverlap() method is used in the fleetPanel class to determine ship overlap when ships are being placed. If ships overlap, a message is displayed at the top of the panel. This method has already been tested in the FleetPanel overlapping ship test.

StatsIO Test

readStats()

Contents of "stats.txt":

```
2|Fady|0
5|Nick|2
1|Josh|4
0|Steve|90
4|James|2
1|323232|0
```

Expected Output (Printed Array):

```
Fady    2    0
Nick    5    2
Josh    1    4
Steve    0    90
James    4    2
323232    1    0
```

Actual Output (Printed Array):

```
Fady    2    0
Nick    5    2
Josh    1    4
Steve    0    90
James    4    2
323232    1    0
```

Result:
Pass

storeCurrentStats()

Input:
storeCurrentStats(1, Jake, 0)

Expected Output:

```
Jake    1    0
```

Actual Output:

```
Jake    1    0
```

Result:
Pass

playerExists()

Contents of “stats.txt”:
2|Fady|0
5|Nick|2
1|Josh|4
0|Steve|90
4|James|2
1|323232|0

Method Call:
playerExists(“Josh”)

Expected Output:
true

Actual Output:
true

Result:
Pass

Method Call 2:
playerExists(“fdssdfsdf”)

Expected Output:
false

Actual Output:
false

Result:
Pass

storeStats()

Original contents of “stats.txt”:
2|Fady|0
5|Nick|2
1|Josh|4
0|Steve|90
4|James|2
1|323232|0

Method Call 1:
stats.storeStats("Jake");

Expected Output:

```
Fady    2    0
Nick    5    2
Josh    1    4
Steve    0    90
James    4    2
323232    1    0
Jake    1    0
```

Actual Output:

```
Fady    2    0
Nick    5    2
Josh    1    4
Steve    0    90
James    4    2
323232    1    0
Jake    1    0
```

Result:
Pass

Method Call 2:

Original contents of “stats.txt”:

```
2|Fady|0
5|Nick|2
1|Josh|4
0|Steve|90
4|James|2
1|323232|0
```

Method Call:

```
stats.storeCurrentStats(1, "Nick", 0)
```

Expected Output:

```
Fady    2    0
Nick    6    2
Josh    1    4
Steve    0    90
James    4    2
323232    1    0
```

Actual Output:

```
Fady    2    0
Nick    6    2
Josh    1    4
Steve    0    90
James    4    2
```

```
323232    1    0
```

Result:
Pass

getPlayerIndex()

Original contents of “stats.txt”:

```
2|Fady|0
6|Nick|2
1|Josh|4
0|Steve|90
4|James|2
1|323232|0
```

Method Call:
stats.getPlayerIndex("Fady")

Expected Output:
0

Actual Output:
0

Result:
Pass

Method Call Test 2:

Original contents of “stats.txt”:

```
2|Fady|0
6|Nick|2
1|Josh|4
0|Steve|90
4|James|2
```

Method Call:
stats.getPlayerIndex("George")

Expected Output:
-1

Actual Output:
-1

Result:
Pass

removePlayer()

Original contents of “stats.txt”:
```
2|Fady|0
6|Nick|2
1|Josh|4
0|Steve|90
4|James|2
```

Method Call 1:
```
stats.removePlayer("Steve");
```

Expected Output:
```
Fady    2    0
Nick    6    2
Josh    1    4
James    4    2
323232    1    0
```

Actual Output:
```
Fady    2    0
Nick    6    2
Josh    1    4
James    4    2
323232    1    0
```

Result:
Pass

Method Call 2:
```
stats.removePlayer("aaaaaaa");
```

Original contents of “stats.txt”:
```
2|Fady|0
6|Nick|2
1|Josh|4
0|Steve|90
4|James|2
```

Expected Output:
```
2|Fady|0
6|Nick|2
1|Josh|4
0|Steve|90
4|James|2
```

Actual Output:
```
2|Fady|0
```

6|Nick|2
1|Josh|4
0|Steve|90
4|James|2

Result:
Pass

updateStatsFile()

Original contents of “stats.txt”:
2|Fady|0
6|Nick|2
1|Josh|4
4|James|2

Method Call:
stats.updateStatsFile();

Expected Output:
2|Fady|0
6|Nick|2
1|Josh|4
4|James|2
0|f|1

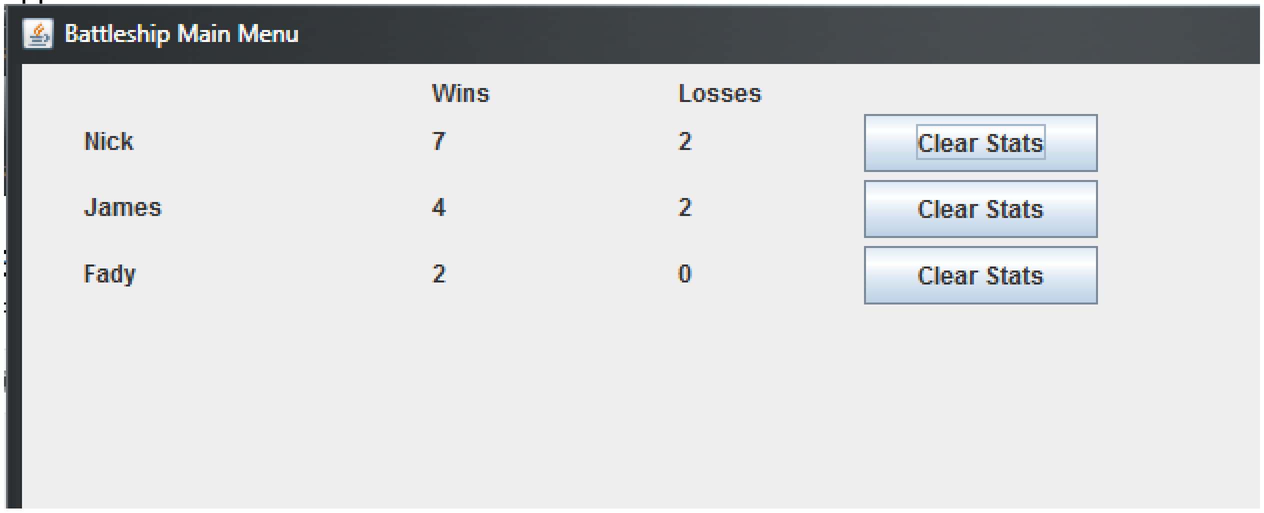
Actual Output:
2|Fady|0
6|Nick|2
1|Josh|4
4|James|2
0|f|1

Result:
Pass

StatsPanel Test

Contents of stats.txt:
7|Nick|2
4|James|2
2|Fady|0
1|Josh|4

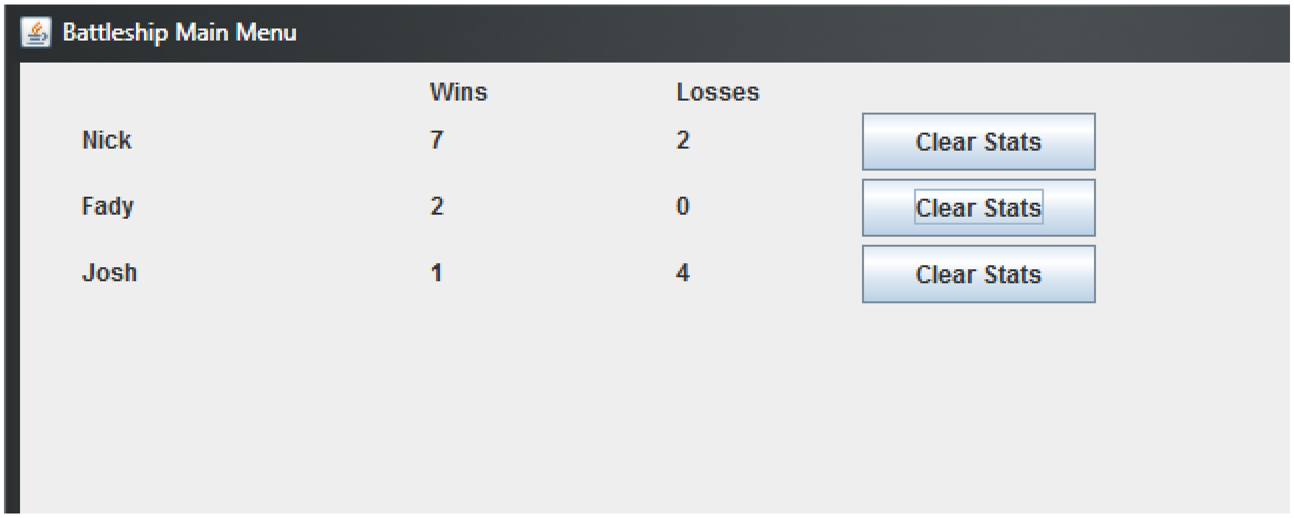
Appearance of Stats GUI:



Battleship Main Menu

	Wins	Losses	
Nick	7	2	Clear Stats
James	4	2	Clear Stats
Fady	2	0	Clear Stats

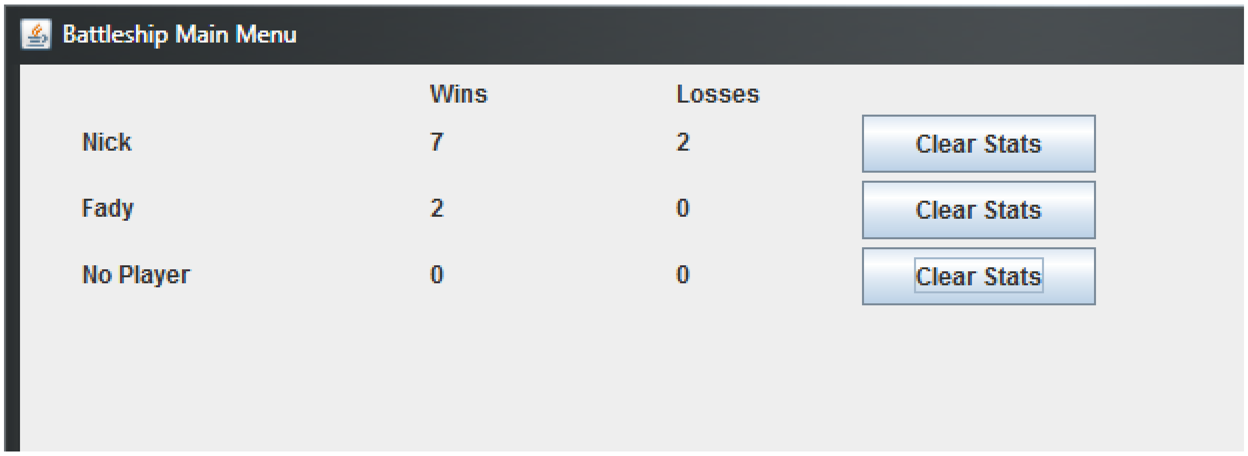
When “Clear Stats” for James is clicked:



Battleship Main Menu

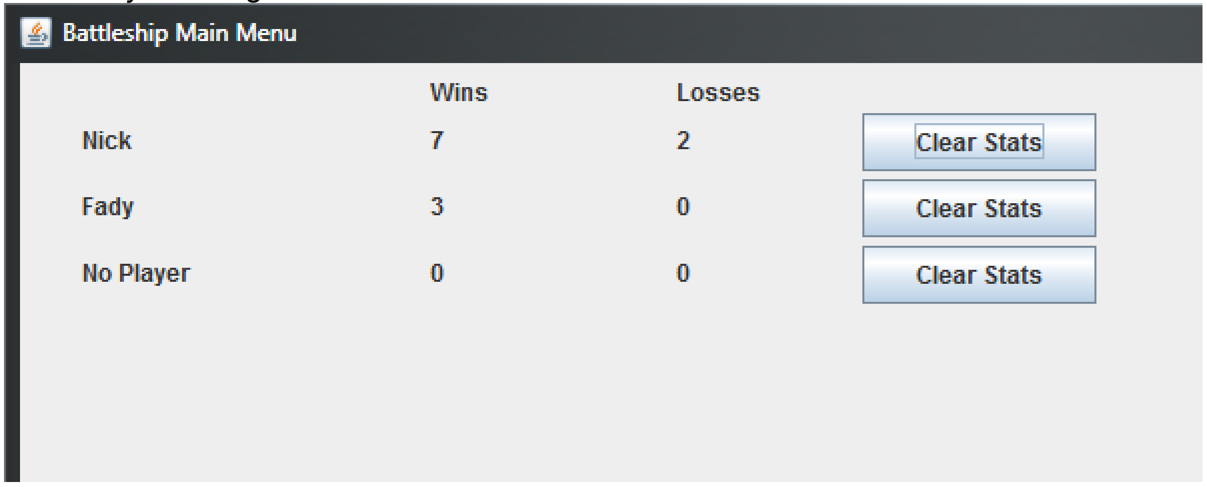
	Wins	Losses	
Nick	7	2	Clear Stats
Fady	2	0	Clear Stats
Josh	1	4	Clear Stats

When “Clear Stats” for Josh is clicked:



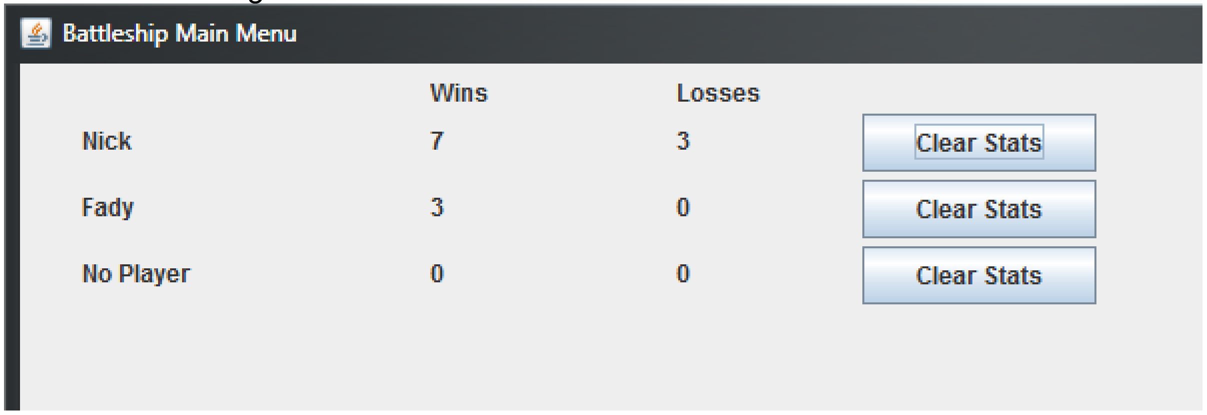
	Wins	Losses	
Nick	7	2	Clear Stats
Fady	2	0	Clear Stats
No Player	0	0	Clear Stats

After *Fady* wins a game:



	Wins	Losses	
Nick	7	2	Clear Stats
Fady	3	0	Clear Stats
No Player	0	0	Clear Stats

After Nick loses a game:

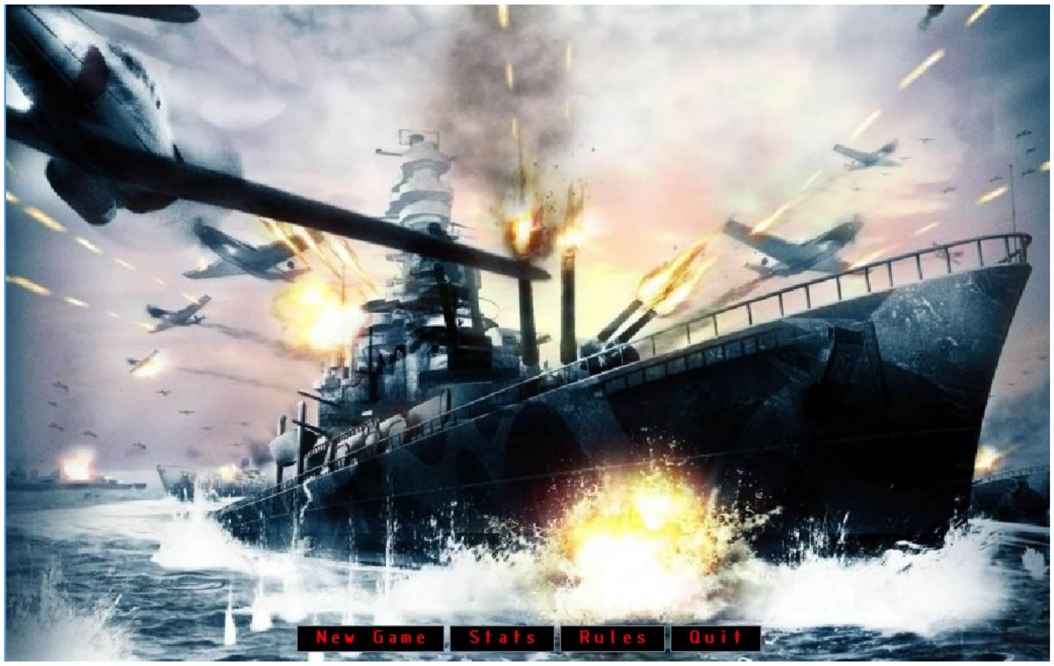


	Wins	Losses	
Nick	7	3	Clear Stats
Fady	3	0	Clear Stats
No Player	0	0	Clear Stats

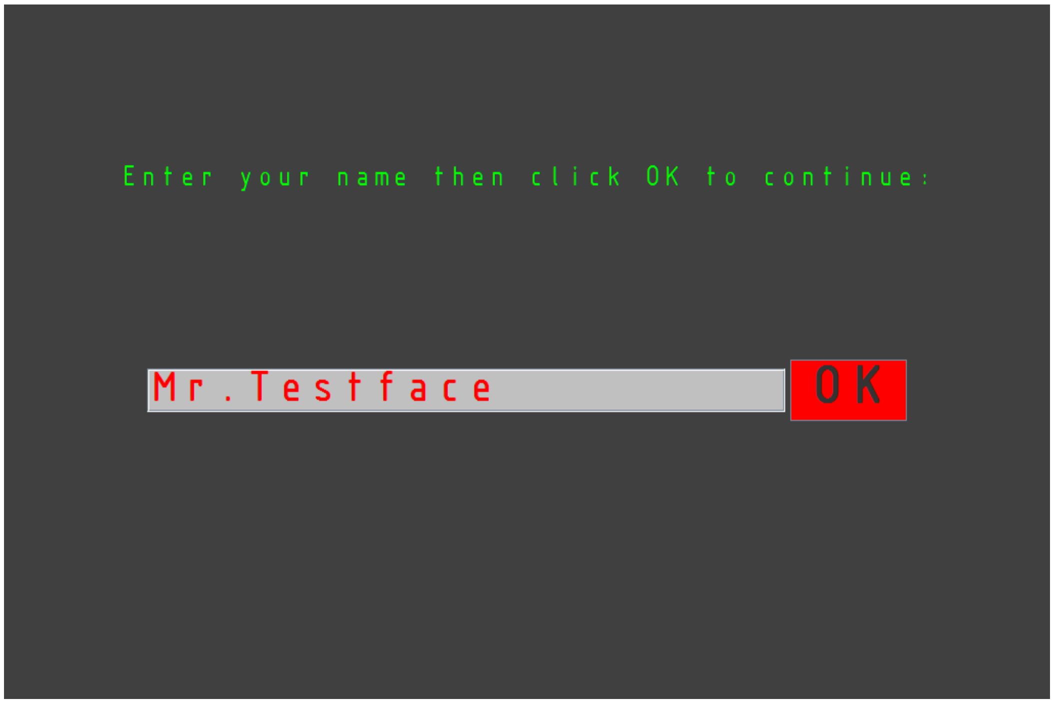
Game Walkthrough

For full test case see http://youtu.be/Rza4A8zeUmY

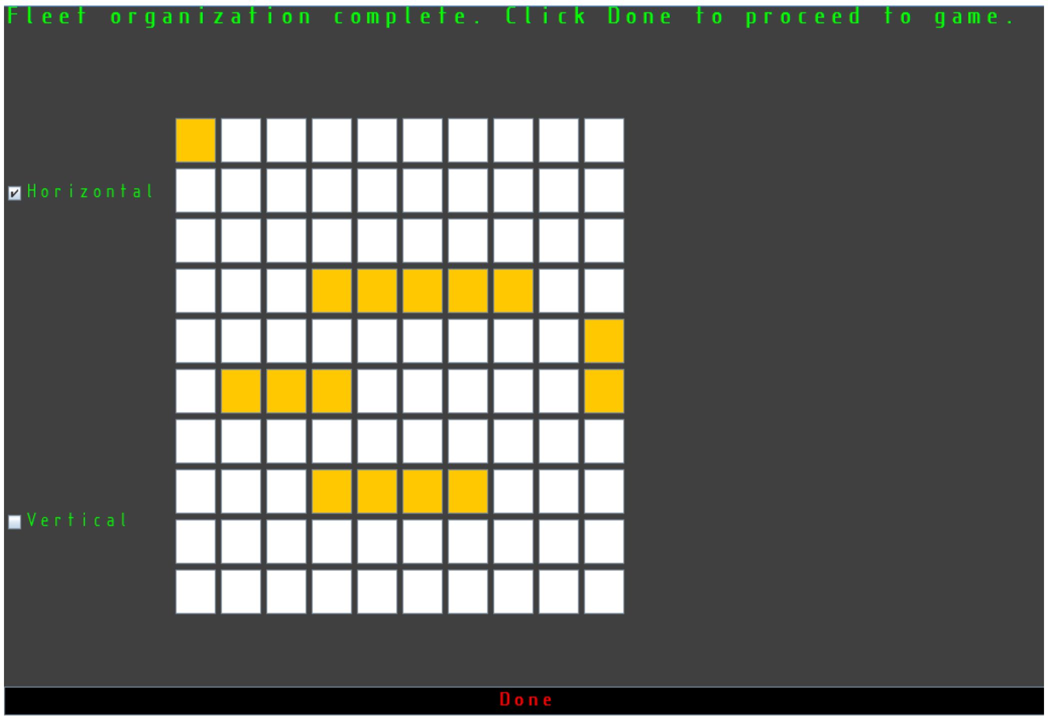
Main Menu:



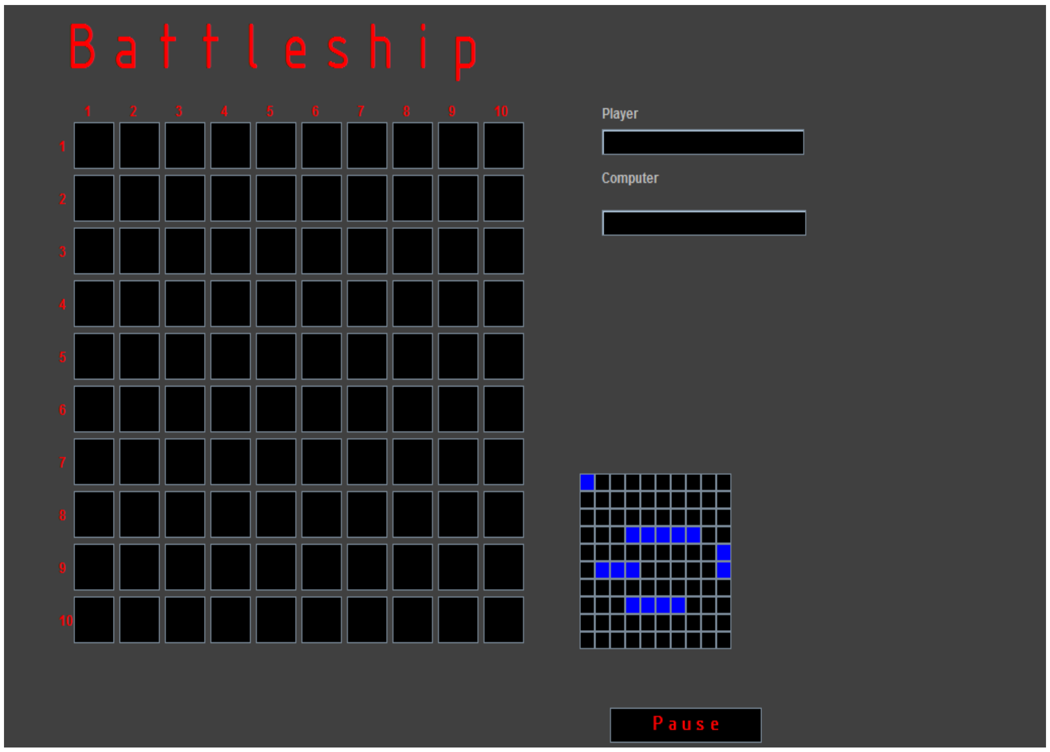
Name Initialization:



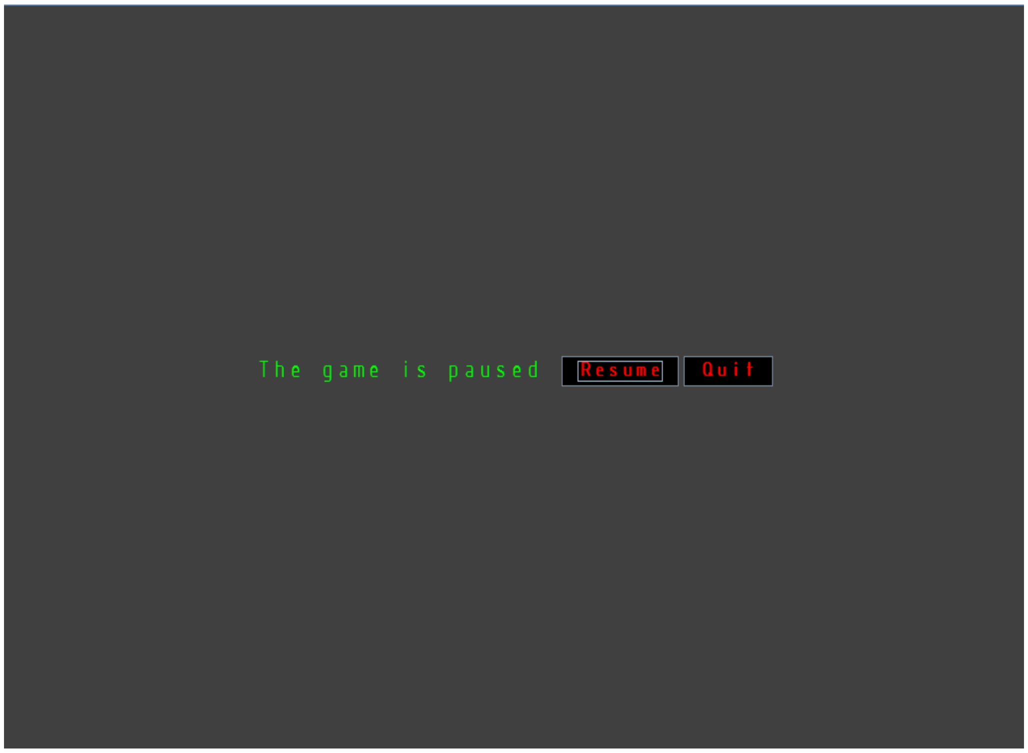
User Fleet Initialization:



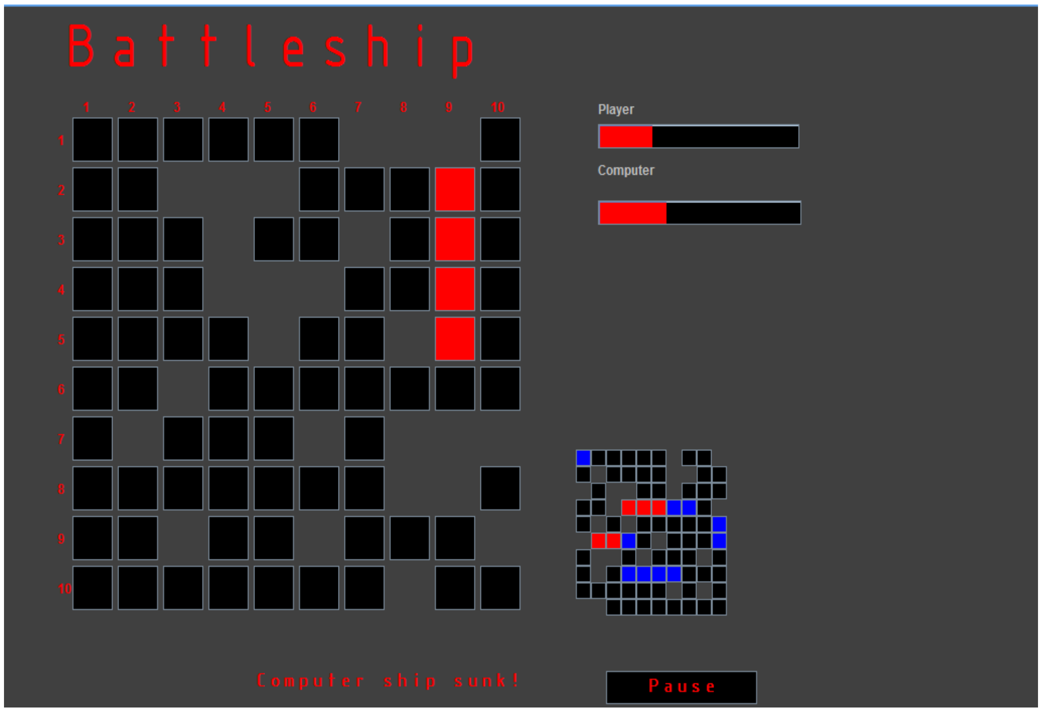
Initial Game State:



Paused Game:

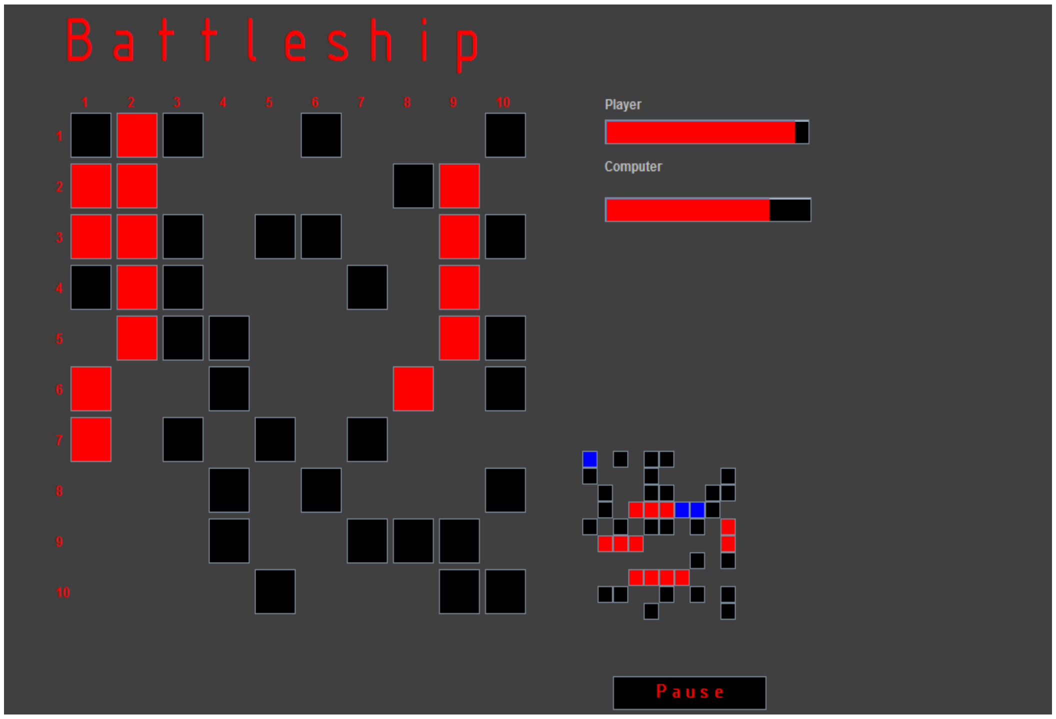

The game is paused

Mid-Gameplay 1:

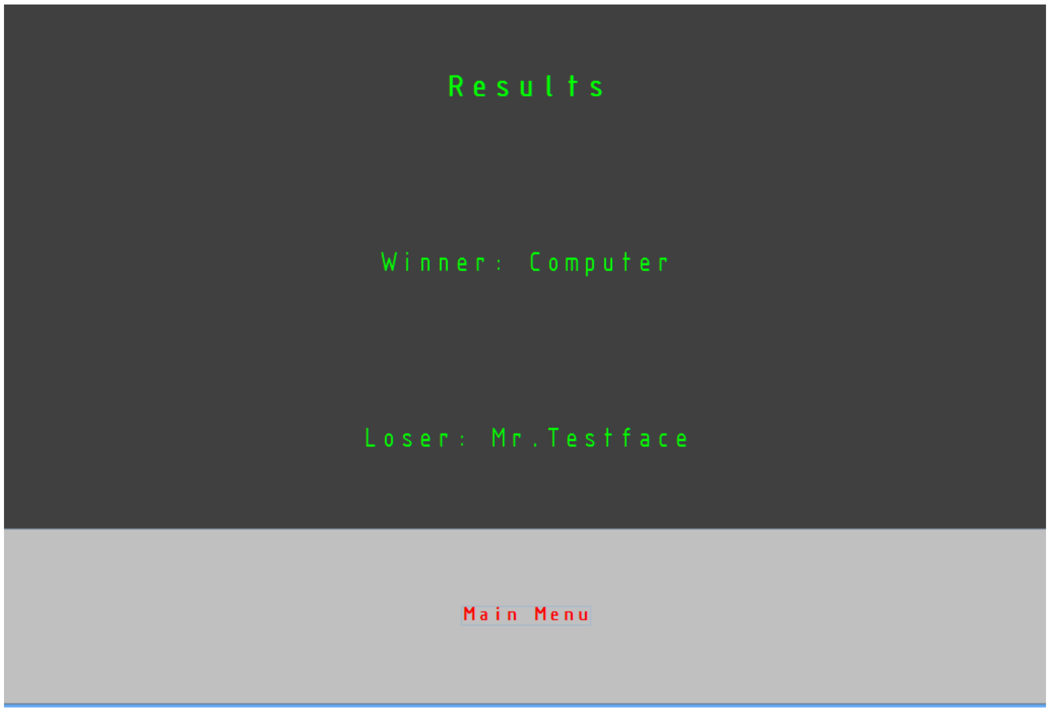


Computer ship sunk!

Mid-Gamplay 2



Results Panel:


Results
Winner: Computer
Loser: Mr.Testface
Main Menu