Table of Contents

Class Documentation

Battleship Class Reference

* void getGameMode ()
* void setGameModeOn ()
* void setGameModeOff ()
* void getSetMode ()
* void setSetModeOn ()
* void setSetModeOff ()
* void setPlayerShip ()
* void setGrid ()
* void setCompShip ()
* boolean checkInGrid ()
* void setShipInGrid ()
* void createAndPlaceAShip ()
* int randomInt ()

BattleshipGUI Class Reference

* updateGameTable()
* updateSetTable
* actionPerformed()
* mouseClicked ()
* mousePressed ()
* mouseReleased ()
* mouseEntered ()
* mouseExited ()

BattleShipStatisticsFileIO Reference

* Int getUserHighScore(String)
* Int getOverallHighScore()
* writeUserHighScore()

GameLoadSave Reference

* + - openGame ()
* saveGame ()

1. Design Notes

* Using Java swing class

1. Test Strategies

Modular Internal Design

To represent the battle ships of the player and computer, we used the two 1D arrays. One is called healthComp[] for the health [] of the computer, and health[] for the user. Each ship is represented by a certain indices and its health is stored in each indice. For example, Aircraft will be represented by health [0] and will equal 5.

|  |  |  |
| --- | --- | --- |
| GRID STATE | REPRESENTATION | HEALTH |
| AIRCRAFT CARRIER | 0 | 5 |
| BATTLESHIP | 1 | 4 |
| CRUISER | 2 | 3 |
| DESTROYER 1 | 3 | 2 |
| DESTROYER 2 | 4 | 2 |
| SUBMARINE 1 | 5 | 1 |
| SUBMARINE 2 | 6 | 1 |
| EMPTY WATER | 7 | - |
| BAD HIT (MISS) | 8 | - |
| GOOD HIT | 9 | - |

**2. Class Documentation**

***2.1 BattleShip Class Reference***

*getGameMode*

**IN:** NONE

**UPDATES:** NONE

**OUT:** gameMode

getGameMode

**return** gameMode

*setGameModeOn*

**IN:** NONE

**UPDATES:** gameMode

**OUT:** NONE

SetGameModeOn sets the gameMode to true (ON)

*setGameModeOff*

**IN:** NONE

**UPDATES:** gameMode

**OUT:** NONE

SetGameModeOff sets the gameMode to false (OFF)

|  |  |
| --- | --- |
|  | Depends On |
| getGameMode | setGameModeOn  gameMode = TRUE |
| setGameModeOff  gameMode = FALSE |

***getSetMode***

**IN:** NONE

**UPDATES:** NONE

**OUT:** setMode

getSetMode

**return** setMode

***setSetModeOn***

**IN:** NONE

**UPDATES:** setMode

**OUT:** NONE

Set the setMode to true (ON)

***setSetModeOff***

**IN:** NONE

**UPDATES:** setMode

**OUT:** NONE

Set the setMode to false (OFF)

|  |  |
| --- | --- |
|  | Depends On |
| getSetMode | setSetModeOn  setMode = TRUE |
| setSetModeOff  setMode = FALSE |

**setPlayerShip()**

IN: int x, int y, int shipType

UPDATES: grid[x][y]

OUT: NONE

Takes in x and y coordinate from the user and places the corresponding ship type on the player grid

Pseudocode:

**if** grid[x][y] == 7

grid[x][y] = shipType

**else**

NONE

|  |  |
| --- | --- |
| grid[x][y] == 7 | grid[x][y] = shipType |
| grid[x][y] != 7 | NONE |

***setGrid()***

IN: NONE

UPDATES: grid[][], gridComp[][]

OUT: NONE

Sets the all cells in player and computer grid to 7 (Empty water)

Pseudocode:

for i = 0 to 9

for j = 0 to 9

grid[i][j] = 7

**setCompShips**

IN: NONE

UPDATES: gridComp[][]

OUT: NONE

Calls createAndPlaceAShip 7 times with corresponding predefined parameters for ship placement on the computer grid

Pseudocode:

createAndPlaceAShip(SUBMARINE\_2, SUBMARINE\_LENGTH, 1)

createAndPlaceAShip(SUBMARINE, SUBMARINE\_LENGTH, 1)

createAndPlaceAShip(DESTROYER\_2, DESTROYER\_LENGTH, 1)

createAndPlaceAShip(DESTROYER, DESTROYER\_LENGTH, 1)

createAndPlaceAShip(CRUISER, CRUISER\_LENGTH, 1)

createAndPlaceAShip(BATTLESHIP, BATTLESHIP\_LENGTH, 1)

createAndPlaceAShip(CARRIER, CARRIER\_LENGTH, 1)

***checkInGrid ()***

IN: int x, int y, int length, int direction

UPDATES: NONE

OUT: boolean

Check whether the defined ship position is available in the grid. Making sure ships don’t overlap

Pseudocode:

for i = 0 to ship length

if direction == NORTH

check availability for cell at (x - i,y) in the grid

return false if its not available

else if direction == EAST

check availability for cell at (x,y + 1) in the grid

return false if its not available

else if direction == SOUTH

check availability for cell at (x + i,y) in the grid

return false if its not available

else if direction == WEST

check availability for cell at (x,y - i) in the grid

return false if its not available

else

“invalid direction”

return true

***setShipInGrid ()***

IN: int x, int y, int shipType, int length, int direction

UPDATES: gridComp[][]

OUT: NONE

This method is called by createAndPlaceAShip for ship placement

Pseudocode:

if direction == NORTH

put a ship of type shipType at starting at cell (x,y) pointing north direction with length = length

else if direction == EAST

put a ship of type shipType at starting at cell (x,y) pointing east direction with length = length

else if direction == SOUTH

put a ship of type shipType at starting at cell (x,y) pointing south direction with length = length

else if direction == WEST

put a ship of type shipType at starting at cell (x,y) pointing west direction with length = length

else

“invalid direction”

***createAndPlaceAShip ()***

IN: int shipType, int length, int quantity

UPDATES: gridComp[][]

OUT: NONE

Create and place the defined ship on the computer grid

Pseudocode:

generate random coordinate number (x,y)

generate random number from 1 to 4 for the direction

for i = 0 to quantity

while checkInGrid(x,y,length,direction) != true

generate random coordinate number (x,y)

generate random number from 1 to 4 for the direction

setShipInGrid(x, y, shipType, length, direction)

***randomInt ()***

IN: int in

UPDATES: NONE

OUT: int out

Generate a random number from 0 to in

Pseudocode:

Generate random double from 0.0 to 1.0

Multiply the double by in

Convert the double to integer

Return integer

***checkMove ()***

**IN:** x:integer, y:integer

**UPDATES:** NONE

**OUT:** TRUE, FALSE

// check the x, y for valid input. compare to computer's grid

checkMove(x:integer,y:integer)

if 0<=x=<9 AND 0<=y=<9

if the spot gridComp[x][y] is being hit/Miss before

**print *invalidInput***

return false

else

**print *validInput***

return true

else

**print *Invalid input***

return false

|  |  |  |
| --- | --- | --- |
| checkMove | | RESULT |
| 0<=x=<9 AND 0<=y=<9 | gridComp[x][y]<8 | VALID INPUT(TRUE) |
| gridComp[x][y]>=8 | INVALID INPUT(FALSE) |
| Not(0<=x=<9 AND 0<=y=<9) |  | INVALID INPUT(FALSE) |

***checkPlacement ()***

**IN:** x:integer, y:integer

**UPDATES:** NONE

**OUT:** TRUE, FALSE

checkPlacement

**if** the spot choose by player is **empty water** that is grid at x, y is 7

***Good Place***

return true

**else**

***Bad Place***

return false

|  |  |
| --- | --- |
| checkPlacement | RESULT |
| grid[x][y]==7 | **GOOD PLACE**(TRUE) |
| not(grid[x][y]==7) | **BAD PLACE**(FALSE) |

***playerTurn ()***

**IN:** x:integer, y:integer

**UPDATES:** gridComp[x][y](Computer’s Grid)

**OUT:** NONE

Pseudocode:

playerTurn

**switch** Computer’s Grid at xy

***case emptyWater***

change the computer’s grid at xy = 8 //BAD HIT

***case Submarine2***

decrease health of Submarine2 by 1

Change the grid value at xy = 9 //GOOD HIT

***case Submarine1***

decrease health of Submarine1 by 1

Change the grid value at xy = 9 //GOOD HIT

***case Destroyer2***

decrease health of Destroyer2 by 1

Change the grid value at xy = 9 //GOOD HIT

***case Destroyer1***

decrease health of Destroyer1 by 1

Change the grid value at xy = 9 //GOOD HIT

***case Cruiser***

decrease health of Cruiser by 1

Change the grid value at xy = 9 //GOOD HIT

***case Battleship***

decrease health of Battleship by 1

Change the grid value at xy = 9 //GOOD HIT

***case Aircraft Carrier***

decrease health of Aircraft Carrier by 1

Change the grid value at xy = 9 //GOOD HIT

|  |  |
| --- | --- |
| playerTurn | |
| gridComp[x][y]=0  (AIRCRAFT CARRIER) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=1  (BATTLESHIP) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=2  (CRUISER) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=3  (DESTROYER 1) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=4  (DESTROYER 2) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=5  (SUBMARINE 1) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=6  (SUBMARINE 2) | healthComp[x][y]=healthComp – 1  gridComp[x][y] = 9(**GOOD HIT**) |
| gridComp[x][y]=7  (EMPTY WATER) | gridComp[x][y] = 8(**BAD HIT**) |

***compTurn ()***

**IN:** NONE

**UPDATES:** grid[x][y](Player’s Grid)

**OUT:** NONE

**USES:** java.util.Random

Pseudocode:

**while** move is true

**if** 0=<grid[x][y]<= 7 **there is new move**

**switch** Player’s Grid at xy

***case emptyWater***

change the computer’s grid at xy = 8 //BAD HIT

move = false

***case Submarine2***

decrease health of Submarine2 by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Submarine1***

decrease health of Submarine1 by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Destroyer2***

decrease health of Destroyer2 by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Destroyer1***

decrease health of Destroyer1 by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Cruiser***

decrease health of Cruiser by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Battleship***

decrease health of Battleship by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

***case Aircraft Carrier***

decrease health of Aircraft Carrier by 1

Change the grid value at xy = 9 //GOOD HIT

move = false

**else**

print ***move already done. move again***

|  |  |  |
| --- | --- | --- |
| compTurn | | |
| 0=<grid[x][y]<= 7 | grid[x][y]=0  (**AIRCRAFT CARRIER**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=1  (**BATTLESHIP**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=2  (**CRUISER**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=3  (**DESTROYER 1**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=4  (**DESTROYER 2**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=5  (**SUBMARINE 1**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=6  (**SUBMARINE 2**) | health[x][y]=health – 1  grid[x][y] = 9(**GOOD HIT**) |
| grid[x][y]=7  (**EMPTY WATER**) | grid[x][y] = 8(**BAD HIT**) |
| not(0=<grid[x][y]<= 7) | OLD MOVE | |

***getGameResult()***

**IN:** NONE

**UPDATES:** NONE

**OUT:** INT

for 0 to 7

totalHealthPlayer = totalHealthPlayer + health[i];

totalHealthComp = totalHealthComp + healthComp[i];

if(totalHealthComp == 0)

return 1;

else if (totalHealthPlayer == 0)

return 2;

else

return 0;

}

|  |  |
| --- | --- |
| getGameResult | |
| totalHealthComp = 0 | Return 1 |
| totalHealthPlayer = 0 | Return 2 |
| totalHealthPlayer != 0 | Return 0 |

***MODULE* BattleShipGUI**

updateSetTable()

*Updates game grid during player grid setting*

IN: None

UPDATES: gameTable

OUT: None

updateGameTable()

*Updates game grid during game time*

IN: None

UPDATES: gameTable

OUT: None

actionPerformed()

IN: ActionEvent me

UPDATES: System.out.println

OUT: None

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| me.getActionCommand() |  | | | | System.out.println |
| “New” | |  | | Pressed: New |
| setCounter = 0 | | Set counter reseted |
| updatesetTable | | Set grid updated |
| Battleship() | gameInstance=battleship | Game instance created |
| setGrid() | Blank grid generated for player and computer |
| setCompShips() | Computer ship placement done  Computer grid copied |
| setGameModeOff() | Game mode 0FF |
| setSetModeOn() | Game mode 0FF |
| setEnabled(false) | Go button disabled |
| updateSetTable() | Set grid updated  Cell color rendered |
| “Save” | | | | Pressed: Save |
| “Load” | | | | Pressed: Load |
| “High Score” | | | | Pressed: High Score |
| “Quit” | | | | Pressed: Quit |
| “Go” |  | | | Pressed: Go |
| setGameModeOn() | | | Game mode ON |
| setSetModeOff() | | | Set Mode OFF |
| setEnabled(false) | | | Go button disabled |
| updateGameTable() | | | Game grid updated |
| "How to play" | | | | Go to start and create a New game by pressing new… |
|  | | | |  |
|  | | | | |

mouseClicked(()

IN: MouseEvent me

UPDATES: System.out.println

OUT: None

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gameInstance.getGameMode() == false && gameInstance.getSetMode() == true && gameInstance.checkPlacement(row,column) == true |  | | | | Placed…ship out of 18 |
| UpdateSetTable() | | | | Set grid updated |
| setCounter >= 18 |  | | | Player ship placement done |
| setCounter= 0 | | | Set counter reset |
| setGameModeOff() | | | Game Mode OFF |
| setSetModeOff() | | | Set Mode OFF |
| setEnabled(true) | | | Go button enabled |
| setCounter<18 |  | | | Input next piece please |
|  |  | | |  |
| gameInstance.getGameMode() == true && gameInstance.getSetMode() == false && gameInstance.checkMove(row,column) == true | playerTurn(row,column) | | | Player input computed successfully | |
| compTurn() | | | Computer made its move | |
| updateGameTable() | | | Game grid updated | |
| For i = 0; i < 7 | |  | [Player] Ship " + i + " healthComp: " + gameInstance.healthComp[i]);  Computer] Ship " + i + " healthComp: " + gameInstance.healthComp[i]);  } | |
| getGameResult() == 1 | You win | |
| getGameResult() == 2 | Computer win | |
|  | Play on | |
| For >=7 | |  | Nothing happens | |

mousePressed(()

IN: MouseEvent me

UPDATES: setEnabled()

OUT: None

|  |  |  |
| --- | --- | --- |
|  | | setEnabled() |
| health[0] | 0 | playerShip0.setEnabled(false) |
| 5 | playerShip0.setEnabled(true) |
| health[1] | 0 | playerShip1.setEnabled(false) |
| 4 | playerShip1.setEnabled(true) |
| health[2] | 0 | playerShip2.setEnabled(false) |
| 3 | playerShip2.setEnabled(true) |
| health[3] | 0 | playerShip3.setEnabled(false) |
| 2 | playerShip3.setEnabled(true) |
| health[4] | 0 | playerShip4.setEnabled(false) |
| 2 | playerShip4.setEnabled(true) |
| health[5] | 0 | playerShip5.setEnabled(false) |
| 1 | playerShip5.setEnabled(true) |
| health[6] | 0 | playerShip6.setEnabled(false) |
| 1 | playerShip6.setEnabled(true) |
| healthComp[0] | 0 | compShip0.setEnabled(false); |
| 5-1 | compShip0.setEnabled(true); |
| healthComp[1] | 0 | compShip1.setEnabled(false); |
| 4-1 | compShip1.setEnabled(true); |
| healthComp[2] | 0 | compShip2.setEnabled(false); |
| 3-1 | compShip2.setEnabled(true); |
| healthComp[3] | 0 | compShip3.setEnabled(false); |
| 2-1 | compShip3.setEnabled(true); |
| healthComp[4] | 0 | compShip4.setEnabled(false); |
| 2-1 | compShip4.setEnabled(true); |
| healthComp[5] | 0 | compShip5.setEnabled(false); |
| 1-1 | compShip5.setEnabled(true); |
| healthComp[6] | 0 | compShip6.setEnabled(false); |
| 1-1 | compShip6.setEnabled(true); |

mouseReleased()

IN: MouseEvent me

UPDATES: JOptionPane.showMessageDialog()

OUT: None

|  |  |  |
| --- | --- | --- |
|  |  | Result |
| getGameMode() == true && gameInstance.getSetMode() == false | getGameResult() == 1 | You win |
| getGameResult() == 2 | You loose |
| otherwise |  |  |

***MODULE* BattleShipStatisticsFileIO**

Provides functionality for storing and retrieving individual and overall user high scores

USES: java.io

java.util.Scanner

java.util.ArrayList

**getUserHighScore(String userName) : retrieves user high score from list of HighScores**

IN: String userName, HighScores.txt

UPDATES:

OUT: int value

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Result | Returns |
| Successful retrieval of high scores from file | Find userName in high scores list | Expected execution | CorrespondingUser high score |
| Fail to find userName in high scores list | User name not in HighScores list | -1 |
| Failure to retrieve high scores from file | | IO Error | -2 |

**getOverallHighScore() : retrieves overall high score from list of HighScores**

IN: HighScores.txt

UPDATES:

OUT: int value

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Result | Returns |
| Successful retrieval of high scores from file | Overall high score exists | Expected Execution | Overall high score |
| No overall high score exists | No overall high score | -1 |
| Failure to retrieve high scores from file | | IO Error | -2 |

**getOverallUserName() : retrieves userName of overall high score from list of HighScores**

IN: HighScores.txt

UPDATES:

OUT: String value

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Result | Returns |
| Successful retrieval of high scores from file | Overall high score exists | Expected Execution | User Name of overall high score |
| No overall high score exists | No overall high score | “-1” |
| Failure to retrieve high scores from file | | IO Error | “-2” |

**clearUser(String userName) : removes user data from HighScores list**

IN: String userName, HighScores.txt

UPDATES: HighScores.txt

OUT: int value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | Result | Returns |
| Successful retrieval of high scores from file | Find user name in HighScores list | Successful attempt to write back new HighScores list | Expected Execution | 0 |
| Failure to write back new HighScores list | IO Error | -2 |
| Fail to find user name in HighScores list | | Name not in High Scores list | -1 |
| Failure to retrieve high scores from file | | | IO Error | -2 |

**writeUserHighScore(String userName, int userScore) : writes new user score to** **HighScores list**

IN: String userName, int userScore

UPDATES: HighScores.txt

OUT: int value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | Result | Returns |
| Successful retrieval of high scores from file | Replace overall high score with user high score if user high score is larger than overall highscore | Successful attempt to write back new HighScores list | Expected Execution | 0 |
| Failure to write back new HighScores list | IO Error | -2 |
| Failure to retrieve high scores from file | | | IO Error | -2 |

***MODULE GameLoadSave***