Group 16

presents

BATTLESHIP

Ryan Jones\*

Kristine Lee

Sepideh Roghanchi

Tulsi Patel

Kaeyan Jones

\*team leader

April 8, 2015 / v1.0

University of Illinois at Chicago / CS 342 / Professor Ugo Buy

1. **Introduction**

We are developing a Java game that allows the user to play BATTLESHIP against a computer opponent. The user and computer will enter positions which the game system will check to see if the attack has hit, missed, or sunk a ship. The game is turn-based, where the user and computer switch turns after each attack.

**How To Play**

In this game, create two identical game boards and let each player place a number of warships. Each player cannot see the other person’s board. The players then take turns firing at one another by guessing one of the board squares. If the square they guess contains part of a ship, it is a hit. Otherwise it is a miss. They sink a ship when all squares containing that particular ship have been hit. The player wins when all of their opponent’s ships have been sunk.

**Client Specifications**

The application plays against a human user. The game is customizable by the size of the board and the number of the ships; all ships may have the same size.

**Additional Specifications and Goals**

We will assume that the user’s turn is first, and that the board is square with dimensions of at least 4x4 and at most 20x20. Ships may only be placed in a horizontal or vertical orientation and are not allowed to overlap. If time permits, we may also include multiple sizes of ships, different levels of AI difficulty, and a GUI using JFC/Swing.

**Roadmap**

* Phase I: Develop game components
* Phase II: Integrate components together
* Phase III: Debug game
* Phase IV: Add additional components (i.e. additional ships or a GUI)
* Phase V: Final debugging

**Developers**

Our team will develop this program using the following systems & software:

|  |  |
| --- | --- |
| Kaeyan Jones | Apple MBP, OS X 10.9.5 / Windows 7 Ultimate x64, i7 3520m, 16GB DDR3, Java 8.20, Eclipse Luna 4.4.1  Custom workstation, Windows 7 Ultimate x64, i7 4930k 4.6GHz, 32GB DDR3, Java 7  Custom FreeNAS, jail, i3 4370, 16GB DDR3, Java 8 |
| Ryan Jones | Asus UltraBook, Windows 8.1 , i5 -3317U, 4GB DDR3, Java 8  Custom Desktop, Windows 7, i3 2.7GHZ,12GB DD3, Java 8 |
| Kristine Lee | MacBook Air, OS X (10.10.2), i5 1.4GHz, 4GB DDR3, Eclipse Luna 4.4.1, Java 8 |
| Tulsi Patel | Mac OS i5 2.5GHz, 8GB DDR3, Eclipse |
| Sepideh Roghanchi | Mac OS X (10.9.5) i5 2.4GHz, 1600 MHz, 8GB DDR3, Eclipse |

1. **Requirement specifications**

Java-compatible architecture supporting a console either by remote connection or monitor. Windows, OS X or Unix-like operating system. The expected input from the user and AI is a location on the board. The output is the representation of the board on the screen at the beginning of each turn.

**Example run on a 7x7 board with 2 ships**

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

A A A S S S A

B B B B

C C C C

D D D S D

E E E S E

F F F S F

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

x = hit | o = miss | # = sunk

User turn.

What is your move? **B2**

B2 is a miss.

------------------------------------------------------

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

A A A S S S A

B o B B B

C C C C

D D D S D

E E E S E

F F F S F

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

x = hit | o = miss | # = sunk

Computer turn.

Computer entered A5.

A5 is a hit.

------------------------------------------------------

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

A A A S S x A

B o B B B

C C C C

D D D S D

E E E S E

F F F S F

. 1 2 3 4 5 6 7 . . 1 2 3 4 5 6 7 .

x = hit | o = miss | # = sunk

User turn.

What is your move?

1. **Software architecture**

**Main Components**

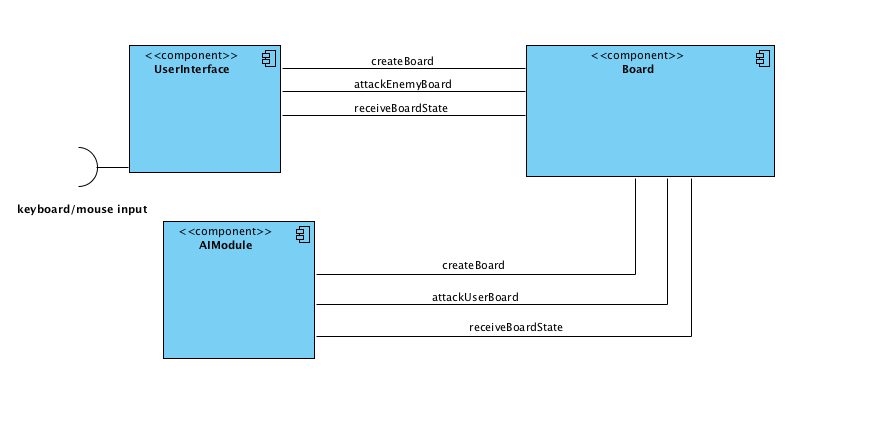
The main components of Battleship are the User Interface, Board, and AI Module.

The User Interface may receive user input via keyboard (in a console implementation) or mouse (in a GUI implementation). The user is responsible for setting up the game (determining the board/ship presets), placing their own ships, and making attacks against the computer.

The AI Module is responsible for placing ships on the computer’s board and calculating locations on the user’s board to attack. This strategy may be calculated with random number generation or a more intelligent algorithm.

The Board is responsible for storing the state of the game. There are two boards that receive input from the players (user & ai): each player sets up ships on their own board and attacks the opponent’s board. If the move is valid, the success or failure is returned back to the player that made the attack. Only the user’s view of their own board and attacks against the computer are displayed on the screen. Board also informs players when their turn has started or ended, or when the game is over.

**Architecture Diagram**

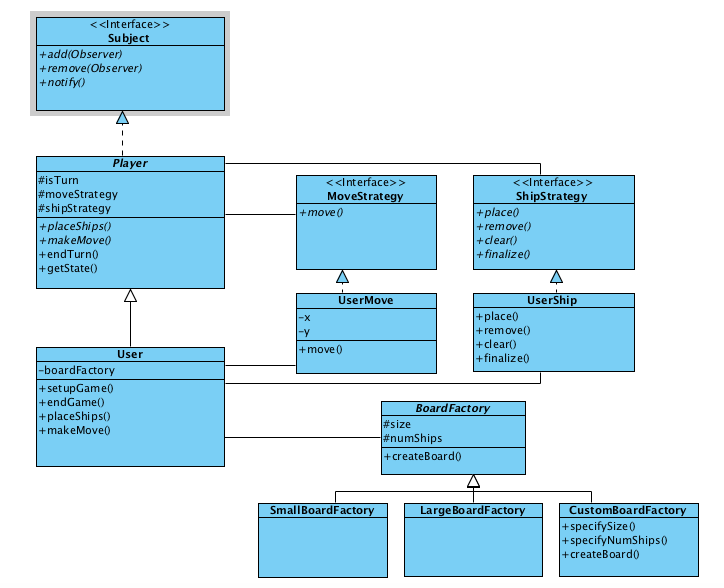


1. **Detailed design**

**Class diagram for the User component**

The User class, along with the Computer class, is a subclass of the abstract class Player. The **Strategy** pattern gives flexibility to how different Players perform semantically similar functions. User moves are determined by console-based user input and sent to the Board component via the **Observer** pattern.

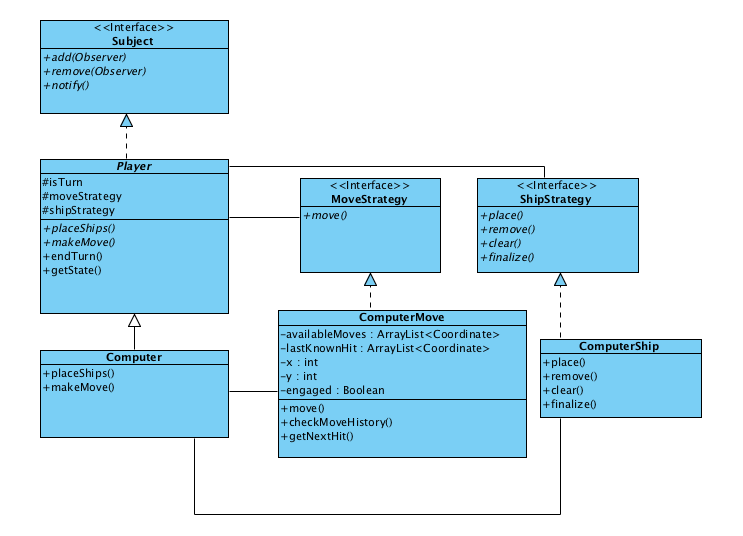
Unlike the computer, the user also has the responsibility of setting up the game and establishing the associations between boards and players. During the setup stage, the user and computer independently place ships on their own board, hidden from the other player. However, the user should control the specifications of both game boards. The user can customize the size of the board or the number of ships with an **Abstract Factory**.



**Class diagram for the AI Module**

Once the player has setup the game and has place his ships on the board, the computer will generate a board for the computer. The computer will create a new instance of the BoardFactory class and randomly place ships in different locations on the board.

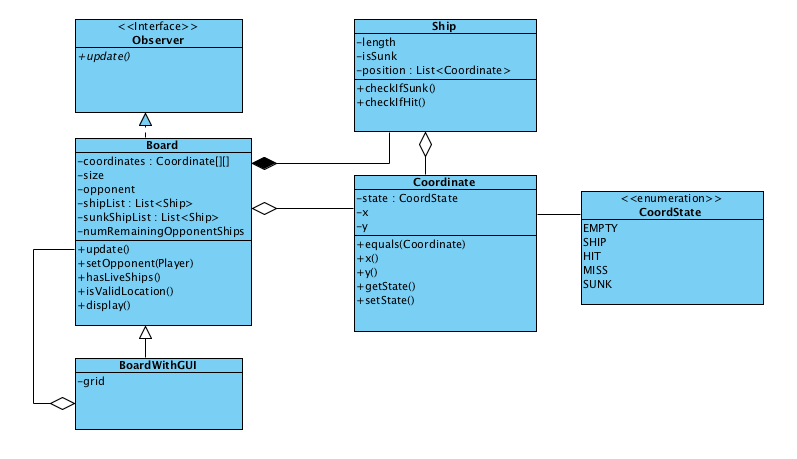
In the AI Module the computer will generate moves based upon couple of factors. It will randomly pick out locations on the board till it gets a hit. It will record every move by removing the location from an arraylist called availableMoves. Once it gets a hit, it will set off a flag called engaged so on its the next turn it will select a location that is one block away from the location that last got hit. The last known hit locations will be stored in a list of coordinates called lastKnownHit. It will continue selecting locations in that area till it gets another hit or till the ship is sunk.



**Class diagram for the Board component**

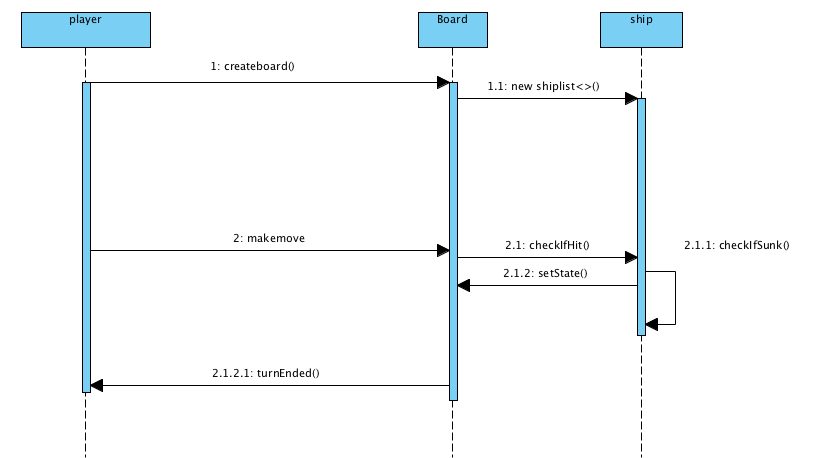
The board contains a 2D array of Coordinates, a list of Ships, and a reference to the opponent that is attacking it. When Player sends an attack location to Board, the board first checks if it is a valid location on the game board (i.e., it is within the board’s dimensions and the location has not already been attacked) and if is not a valid move the board will request the user to enter in another move. If it is, it checks every living ship in shipList to see whether the attack is a hit or a miss and notifies the user that their turn is over.

If the game implements a GUI, the GUI may be a wrapper around the original Board class. BoardWithGUI controls the graphical display and also listens to user input.



**Sequence Diagram: Board interfaces with player**

The player instantiates the board during the setup of the game and also tells the board where to place the ships. There are two instances of Board: one for the user and another for the computer. When the player sends a move to the board, the board relays that coordinate to its ships to see if any of them are hit. If so, the ship then sends a message to the board to set that location to a hit and updates that location in the ship itself. Upon a hit, the ship also checks whether any of its coordinates are alive; if not, it has been sunk. Board then sends a message to the attacking player that their turn is over.



1. **Implementation notes**

**Unresolved issues**

Whether we will get to the GUI is still unknown, so it has not been fully fleshed out. It would be nice to separate game setup from gameplay in a more visible manner. During game setup, each player interfaces with their own board, but during gameplay, the players only interface with the enemy board, which may mean reassigning observers.

**Code snippets**

*// secure? RNG for AI moves*

SecureRandom random = new SecureRandom();

*// snippet for calculating AI moves*

if(move.engaged == true){

move.getNextHit();

}

else {

int randomIndex = random.next(availableMoves.size);

move = availableMoves.remove(randomIndex);

}

return move;

*// snippet for display*

for ( int i=0; i < size; ++i) {

for (int j=0; j < size; ++j) {

CoordState state = coordinates[i][j].getState();

switch (state) {

case EMPTY:

case SHIP: *// display as EMPTY on opponent board*

case MISS: *// display as EMPTY on own board*

case HIT:

case SUNK:

default:

break;

}

}

}

*// snippet for checking whether opponent board has been hit*

for ( Ship ship : shipList) {

for ( Coordinate c : ship) {

if (c.equals(input)) {

if (c.checkIfHit()) {

return -1; */\* invalid: cannot hit twice \*/*

}

else {

c.setState(HIT);

input.setState(HIT);

ship.checkIfSunk();

return 1;

}

}

}

}

board.setState(MISS);

return 1;