Concordia University
Engineering and Computer Science

SOEN 6441- Advanced Programming Practices

Summer 2019

Architectural Design Document for BattleShip Game Project

Submitted to: Prof Nagi Basha

Group 3

Ankur Malhotra 40041056

Raghav Kaushal 40075268

Subhannita Sarcar 40059367

Kritika Kritika 40068982

Shriyans Athalye 40037637

Table of Contents

1. Introduction

1.1 Purpose of the Document

This document's purpose is to build an architecture design model implementing a predefined architectural model. The implemented design model is Event Driven Architecture. In EDA, the components do not invoke each other explicitly, but generate signals called events. To receive events, objects can receive events at ports (statically or dynamically bound) or can register for event notification (e.g. via callback). This architecture supports the Java Swing provision of user interface related functionality. The Swing API uses "ActionListener" and "ActionEvent" to organize event concerns. Class that needs to be aware of a particular event simply implements the appropriate listener and overrides the inherited methods and is then added to the object that fires the event. The following concepts are also implemented:

- **Programming Concepts**: Some of the well-known programming practices were followed to maintain effectiveness of project development such as pair programming where two programmers worked on the same workstation to maintain productivity.
- **Simple Design**: Simple workflow of design made it easier to read and avoid faults.
- **Continuous Integration**: Github was used as version control for the project where all programmers were made to work with single branch and commit accordingly. Maintenance of frequent changes, rollbacks helped increase the productivity by automatic integration.
- **Coding standards**: Simple and understandable coding standards were followed including naming and file organization conventions.
- **Refactoring**: The following refactorings were done after Build 1:
    - Computer.java : improved AI implementation
    - SelfBoard.java : selfboard and FleetSetup.java (from build1) are merged together in a single file and drag-and-drop functionality is implemented to place ships
    - Game.java : game flow is changed depending upon mode(2 player or play against computer) and variation(normal or Salva) selection
    - AttackBoard.java : on mouse click, program flow was dependent on player's name. This has been changed to make universal operation for both players. This is made possible by including a simple method (Game.getOppo()) that return the opponent player.

- Player.java : Several methods are combined to form a single method called checkPossible() that checks whether a ship can be placed at specified coordinates. This has reduced lines of code as well as combined business logic under a single method.

- Screen.java : The action listener has direct game flow to either of the two methods (setUpScreen() or gamePlayScreen()) depending on phase of the game. Earlier in build 1, the entire logic was implemented in one single complex action listener method.

1.2 Scope of the Document

This design covers the development of simple battleship game using different builds covering different releases of the game. Detailed explanations of design are covered in different sections below. The motives of design decisions are as follows:

- Different subsystems processing the same event
- Minimum time lag with processes running like real time
- Supporting component reuse as minimum coupling between modules
- Adding new consumers is easy due to point to point integration
- Respond to events is easy on arrival by consumers
- Every subsystem has an independent view of its event streams

Functional requirements:

Ship Placement:

- User-driven placement of ships on self board.
- Users are able to place both horizontal and vertical ships.
- Users are able to drag and drop ships to valid positions

- Board constraints such as invalid placement on edges or collision with existing ship or placement right next to existing ship (in same alignment) are respected.

Game Play:

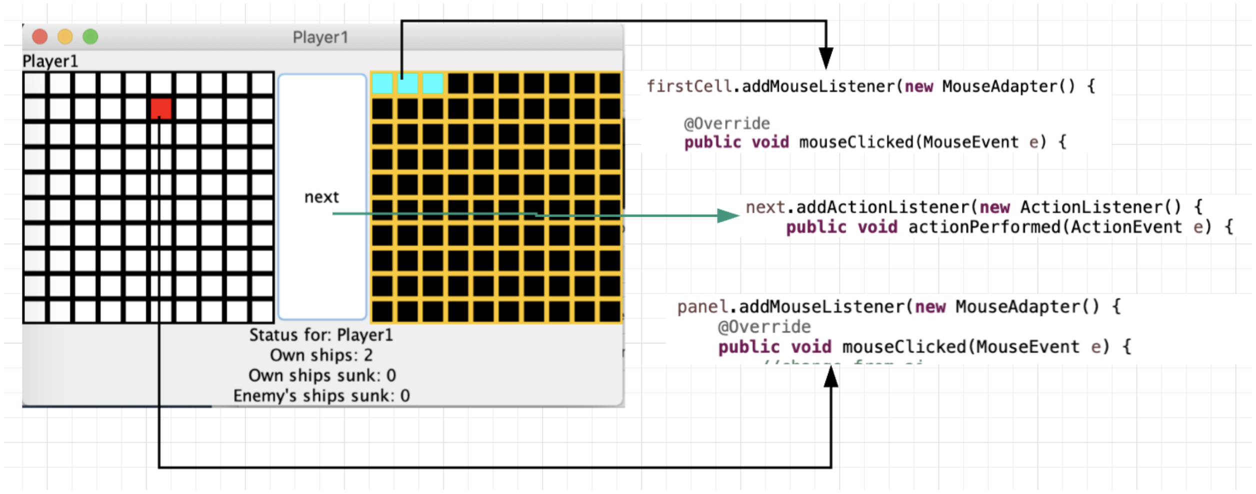
- The button "next" triggers the other player's turn.
- Players are unable to change anything in their self boards after set up phase.
- Players are unable to hit more than the maximum allowable shots every turn on attack board.
- Players are unable to hit on cells that are already hit before (denoted by colored cells).
- Player 2 is the AI engine against which a human player (Player1) will be playing
- The strategy of the AI engine is as follows :

  Unless it gets a hit, random shots are selected to be hit. Once a hit is found, the probability of getting a hit on each cell is calculated. The cells with highest probability are selected as targets.
- At the end of the game, the winner is declared.
- Timer keeps a counter of the time taken by player to select a target.
- Game has a salva variation where each player takes 5 shots per turn initially. The number of maximum allowable shots per turn is decreased by 1 every time a ship sinks in the player's fleet.

1.3 Definitions

Self Board : Part of each player's screen where his own ships are placed
Attack Board : Part of each player's screen where he will select cell to attack opponent
Setup phase : Beginning phase of the game when each player are setting up self ships locations
Game play : Phase of the game after setup phase is complete

2. Event Driven Architecture



```
firstCell.addMouseListener(new MouseAdapter() {

    @Override
    public void mouseClicked(MouseEvent e) {
```

```
next.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
```

Figure 1 event driven architecture model

The listeners used in various source files are listed below and perform the following functionalities:

FILE	EVENTS	ACTION
AttackBoard.java	addMouseListener	Handles event triggered by mouse click on attack board for corresponding player when isAttackBoardListener is true
AttackBoard.java	isAttackBoardListener	Controls activity of attack board, i.e., it should be set to false during set up phase and true during game play
Screen.java	addActionListener	Handles event triggered by mouse click on “next” button to change the screen from one player to another (switching turns)
SelfBoard.java	isSelfBoardListener	Controls activity of self board, i.e., it should be set to true during setup phase and false during game play
AttackBoard.java	addMouseListener	Handles event triggered by mouse click on self board for corresponding player when isSelfBoardListener is set true

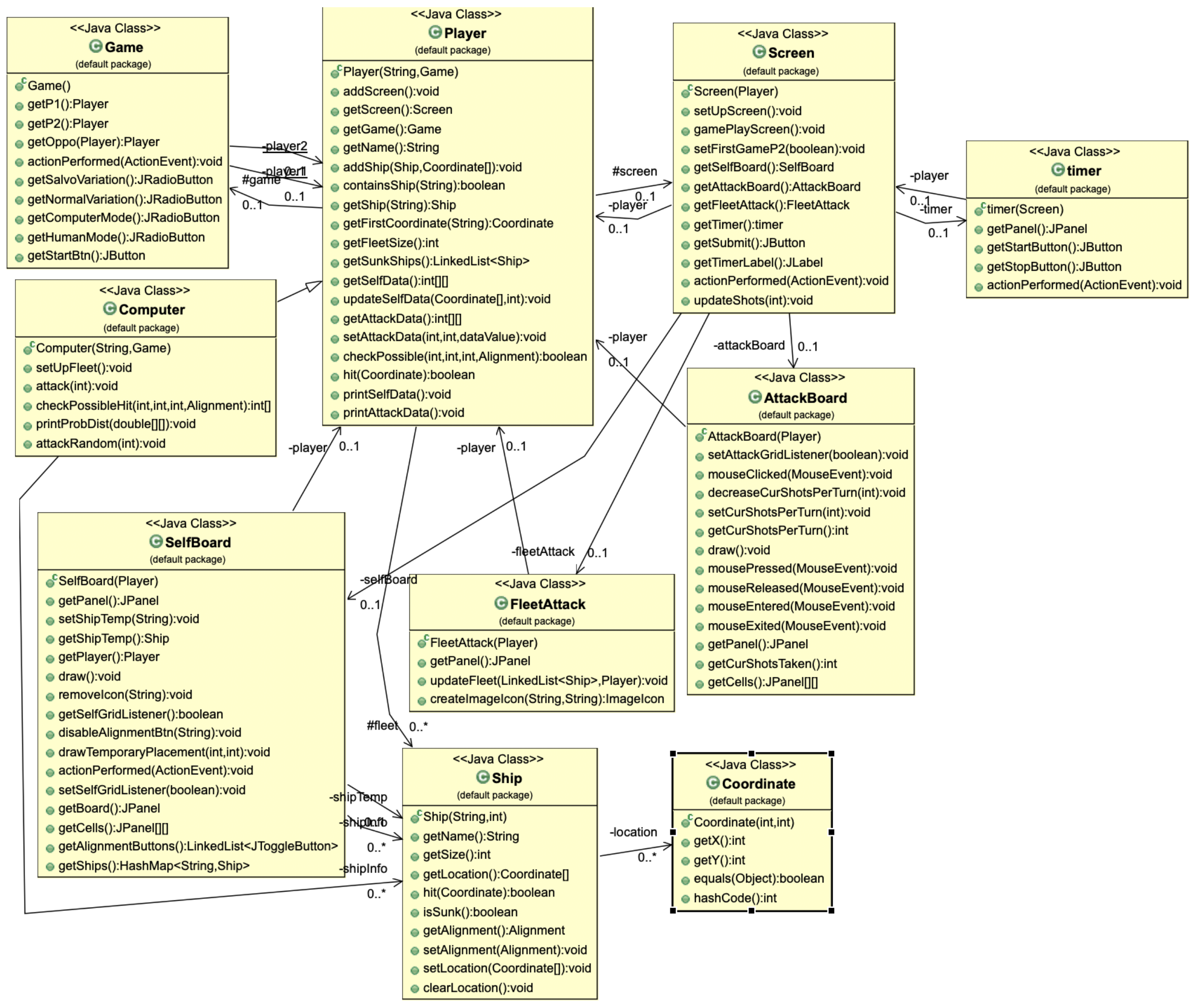


Figure 2. Class Diagram

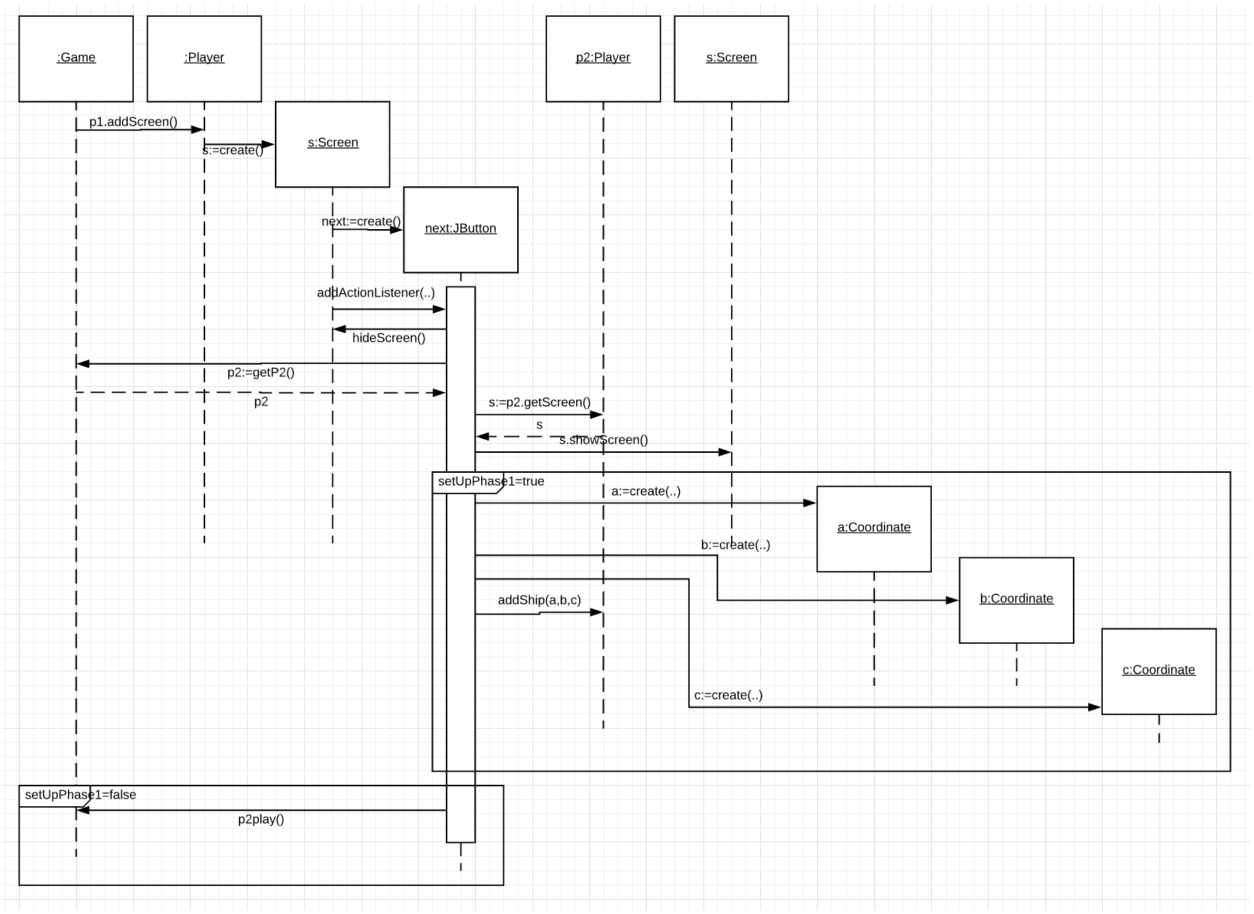


Figure 3. Interaction Diagram when “next” button is pressed

3. Technology and Tools:

Technology and Tools	Description
Eclipse, Intellij	IDE for game development
JDK 1.8.0	Java Development Kit Version
Swing	Library to control UI components
GitHub	Version Control