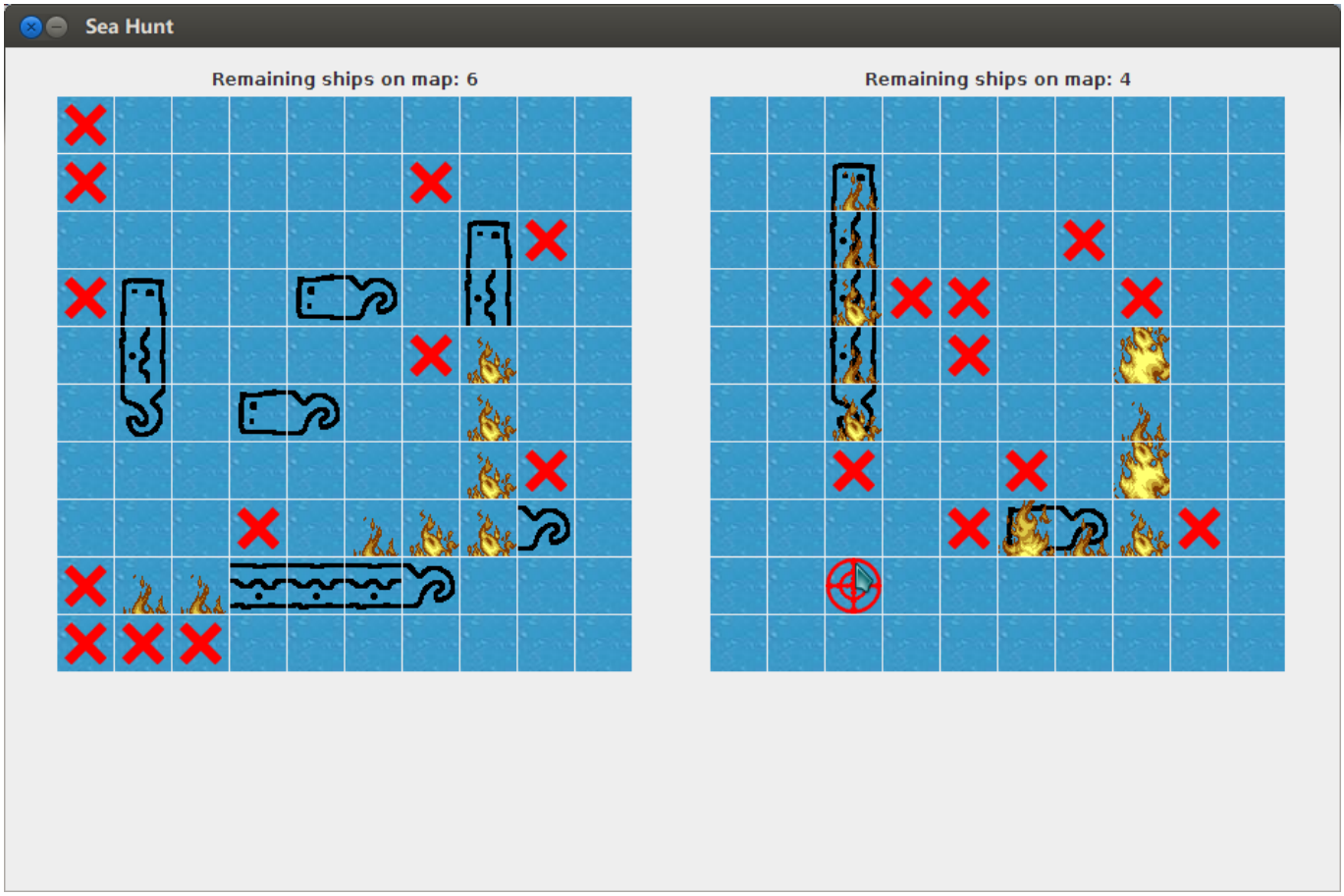
# *Assgn. 2 – BattleShip*

Student name: Bui Thanh Nhan

Student ID: s3360610

Lecturer: Mr. Kevin Jackson

Course: Programming 2



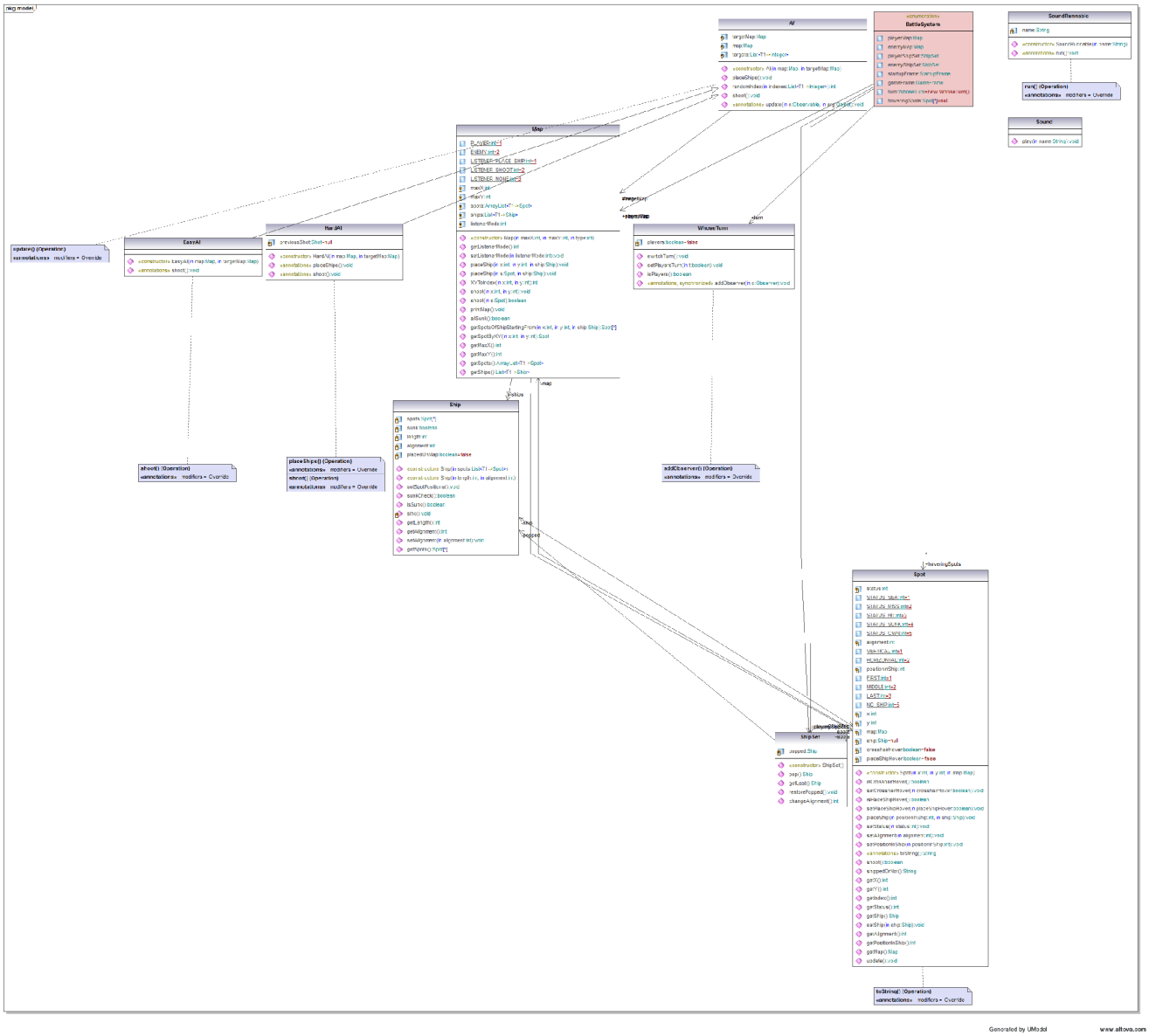
# 1. The Game

The basics are similar to any classic Battleship game, though there are some noteworthy points :

- Although this is a Battleship game, I chose the sea monster concept instead of ships
- The game only has single player mode against the computer.
- Each side has 6 ship with the respective lengths: 2, 2, 3, 4, 5, 6
- In each turn the player can only shoot 1 bullet no matter what.
- In easy mode, the player can only place horizontal ships. Vertical alignment is only possible in hard mode: press **Space** before placing a ship to change its alignment.
- In hard mode the AI will try to shoot places around the previously hit spot.

# 2. Class Design

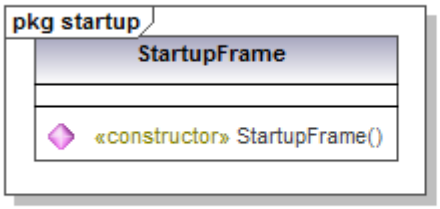
## Models: (zoom to view details)



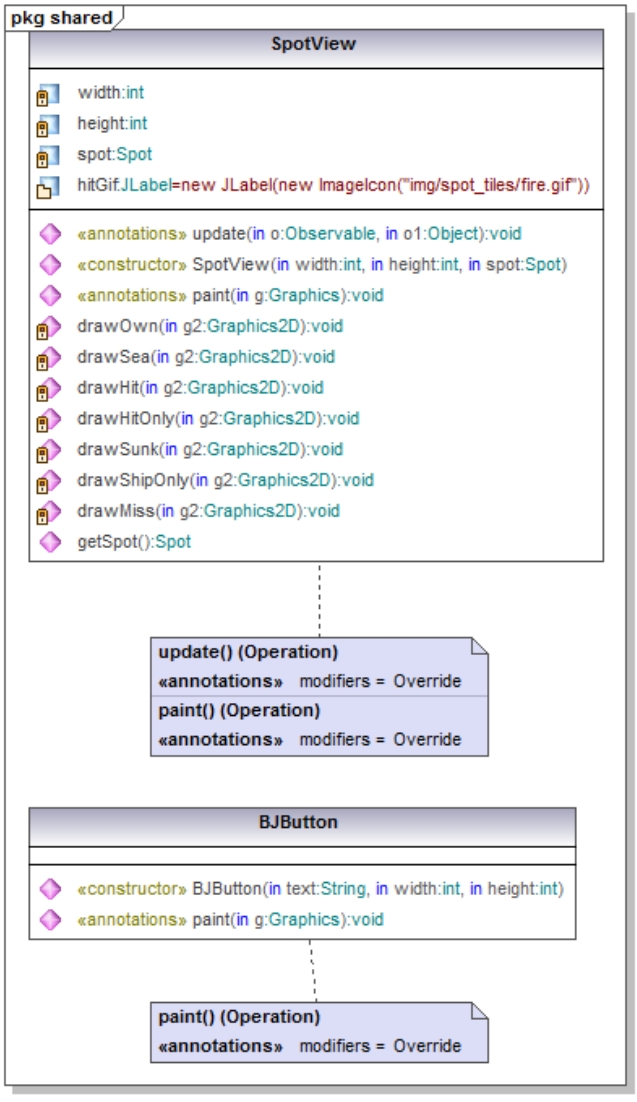
## Views:

There are 3 sub-packages:

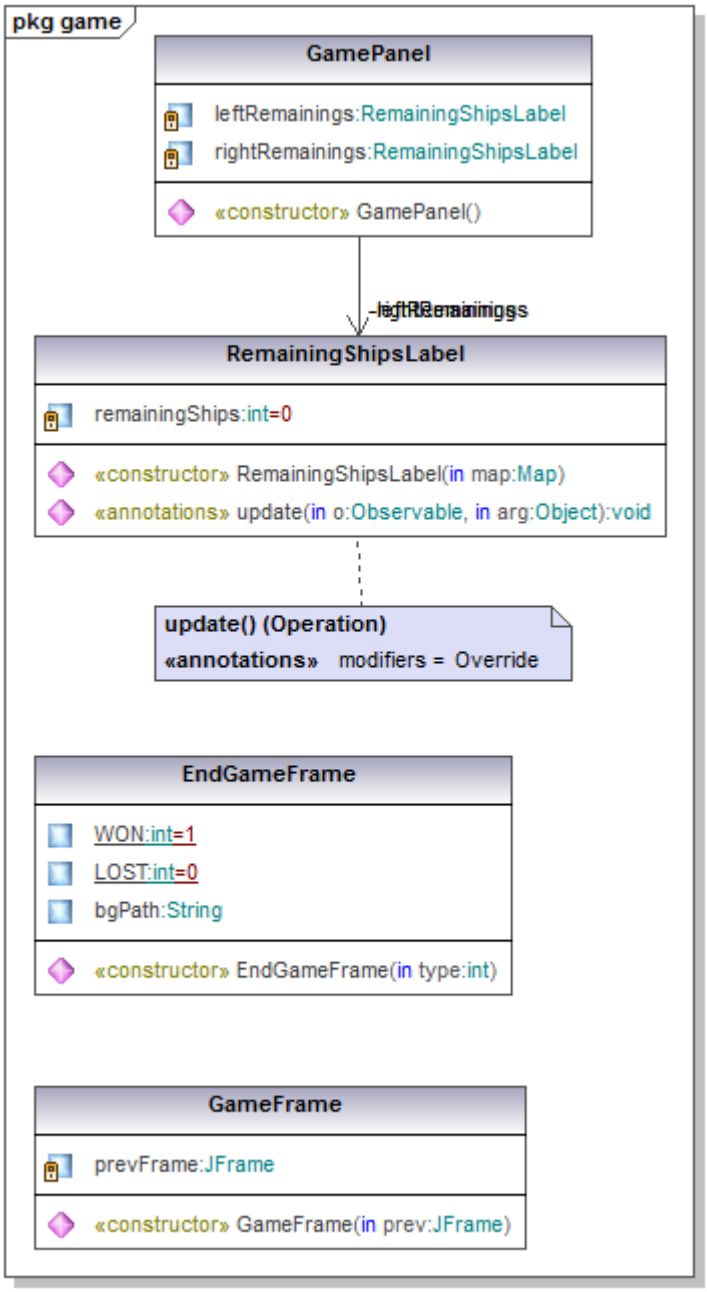
- Startup: for the game startup screen
- Game: for the main “GameFrame”, which is where all the playing happens
- Shared: generic views




Generated by UModel www.altova.com

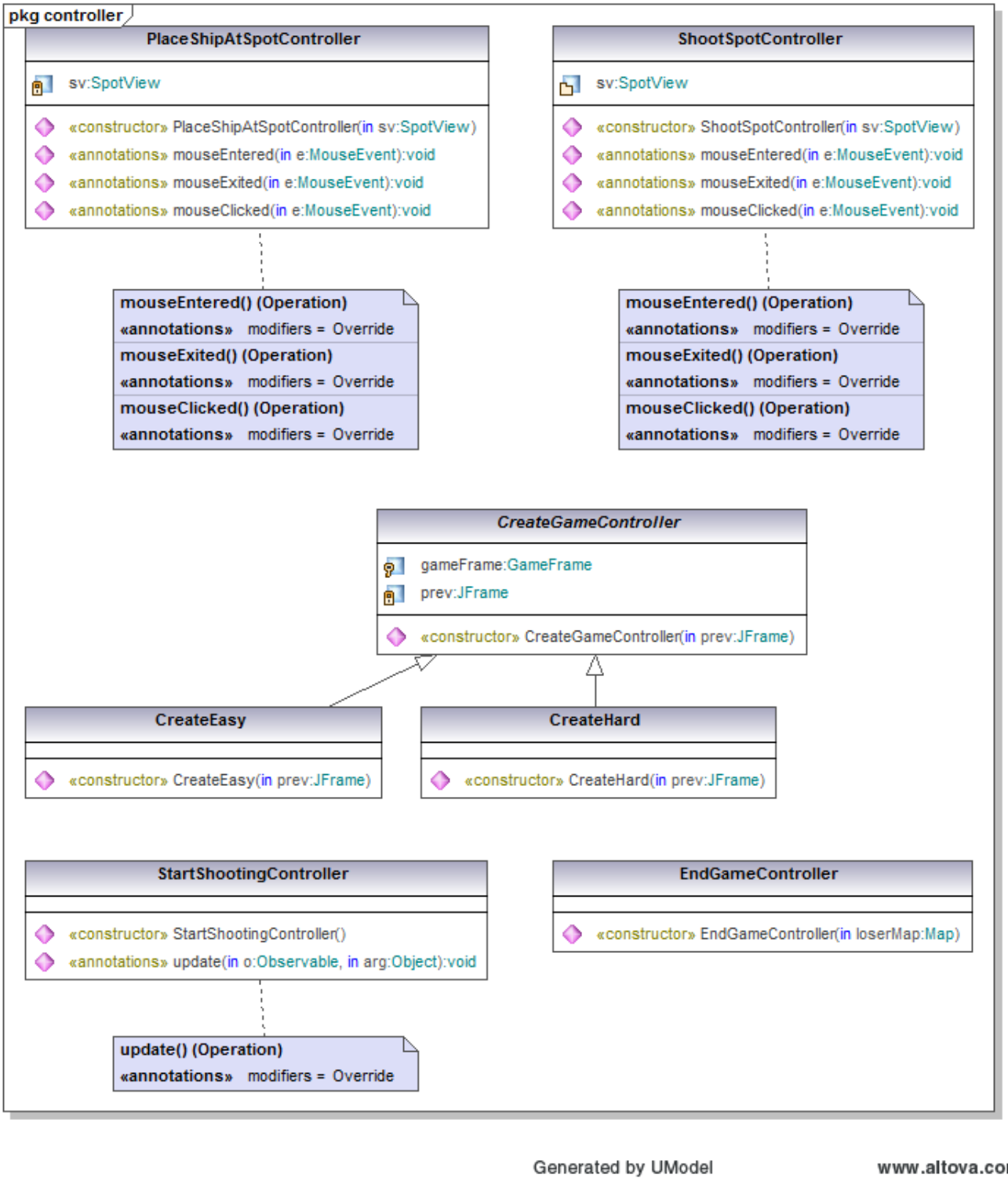



Generated by UModel www.altova.com




Generated by UModel www.altova.com

## Controllers:




## 3. Design explanation

I used the Spot class as the smallest yet most important part of both the game logic and UI.

For game logic: each Ship instance is just a list of Spot instances where that ship resides. The ship's status – where it has been it, its place on the map, whether it has been sunk… - is entirely demonstrated by its Spots.

For UI: since Spots represent everything of the Ship, I let them be the main Observable, letting every major UI component observe them to make relevant UI updates.

The flow of the game is controlled by an instance of class WhoseTurn. This class itself is an Observable. The AI player and the StartShootingController both observe this WhoseTurn instance to know when to take turn.

# 4. Implementation

## The ships

I didn't render the ship on screen as a single unit. Instead, I rendered each spot conditionally based on its location in the ship so that when they are rendered together on the screen, they make up a whole ship. To be more specific, there are 3 kinds of sprite the ship can have: first, middle or last. All spots that are neither at the beginning nor the end of the ship will be rendered with the "middle" sprite, which is a recursive one, so I can dynamically "scale" the ship to any length.

This approach rids me of having to organize multiple layers of JPanels, but limited my choice of sprite. In fact, I gave up looking for a compatible sprite set and had to draw my own, which did not look really nice.

## BattleSystem, the singleton

To avoid making a mess passing references between the classes, I decided to put all frequently used objects in the publically accessible BattleSystem singleton. This may not be an elegant solution, but saves me from making tons of parameters.

## The Hard mode

The basic Easy AI does everything randomly: placing ships and shooting. The Hard one was extended so it can place either vertical or horizontal ships. It starts off shooting at a random spot, but if it detects a hit, it will look for the surround spots and shoot one of those. Unfortunately I did not have enough time to extend it any further (I did extend it, but it just did not work the way I wanted and I still have not figured out where I did wrong).

## **Sound**

Somehow it crashed on my Ubuntu machine so I disabled it.

## Others' works used:

- Fire sprite at http://www.gifandgif.eu/animated_gif/Fire/index.php
- Free sounds at http://www.freesound.org/people/knarmahfox/sounds/59008/