CS 3110 Project Milestone 1

Battleship

November 12, 2015

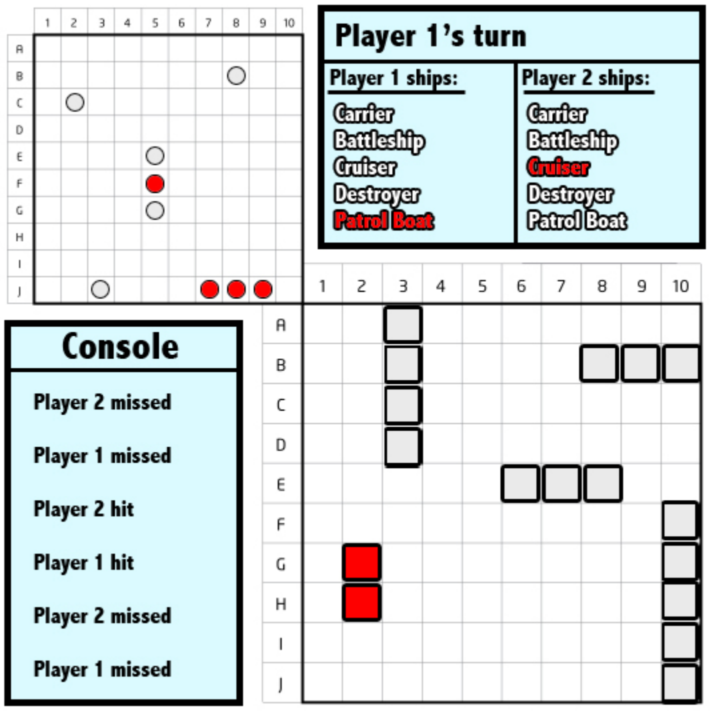
Team Members:

Daniel Etter	`dje67`
Ian Hoffman	`ijh6`
Rob Barrett	`rpb83`

Meetings:

Tuesdays	3:00-4:30pm
Thursdays	3:00-4:30pm
Weekends	As Needed

Mockup



System Description

System Proposal

We intend to build an implementation of Battleship, including a GUI.
Feature list:

- Local Multiplayer
- AI with multiple levels of difficulty
- Graphical User Interface
- Standard 10 x 10 Battleship Board

Narrative Description

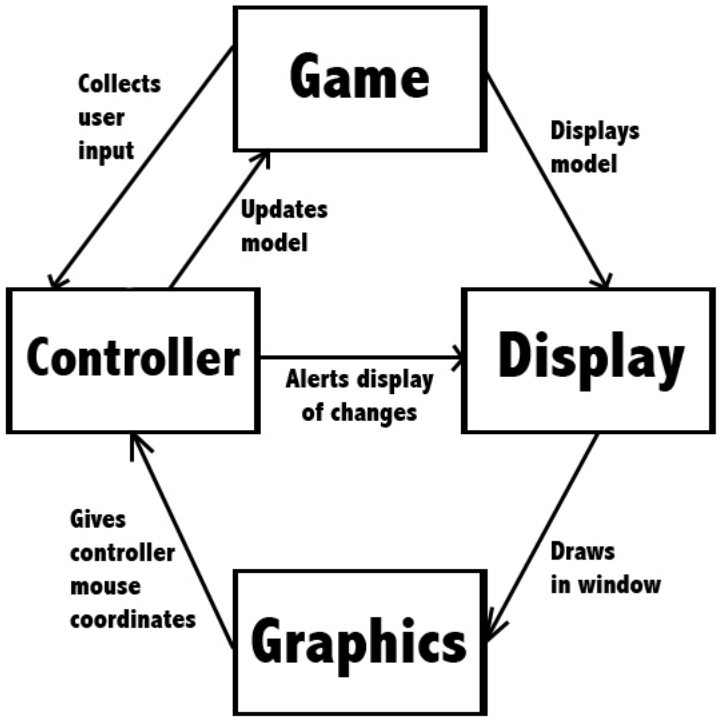
The game being designed will be played between a player and the computer or another player. Each side will have access to their grid and their guesses about an opponents grid. Each grid is made up of a 10 x 10 array, positions on the grid are indicated by a letter A-J, and a number 1-10. Each player gets 5 ships to place:

- An **Aircraft Carrier** of length 5
- A **Battleship** of size 4
- A **Cruiser** of size 3
- A **Destroyer** of size 3
- A **Patrol Boat** of size 2

These are placed across the board, and must be completely contained by the 10 x 10 grid with no overlaps between ships. The game is played by each player guessing about the location of their opponents ships with a (letter, number) pair. If they miss, they will be notified visually and textually. If they hit, they will be notified the same way, but with a different color. Taking out a ship completely results in a notification of the type of ship eliminated. When one player has all their ships sunk, they are the loser! There will be a GUI for the player showing their board and their guesses, allowing for easy placement of ships at the beginning of the game, and smooth, straightforward play.

Architecture

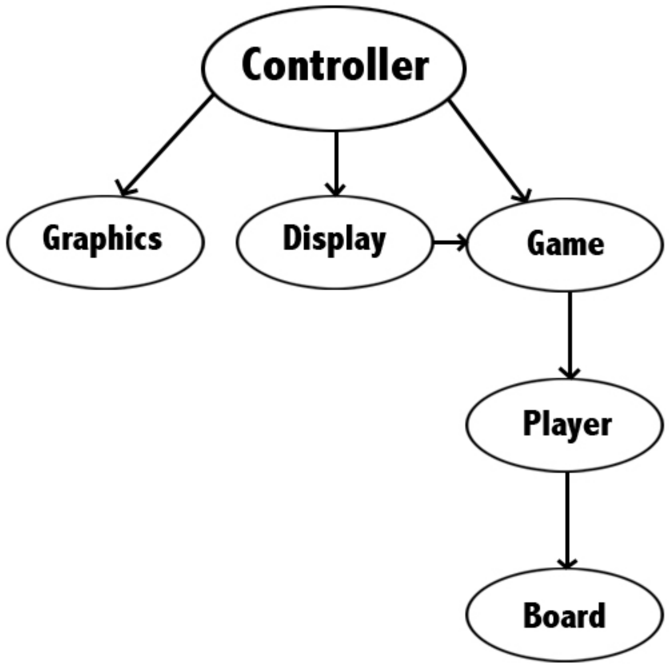
Our system is loosely based around a Model-View-Controller architecture.



System Design

Important modules include:

- `Controller`: Collects user input through `Graphics` and updates `Game` model accordingly
- `Display`: Uses `Graphics` and draws the current state from `Game` model
- `Game`: Represents the state of a Battleship game, with players, boards, and other information
- `Player`: Keeps track of ship and peg board
- `Board`: A board structure of either pegs or ships at positions in the board



Module Design

Please see comments in `*.mli` in `interfaces.zip`.

Data

The data will be stored in our `Game` model according to the following:

Game

- Turn Number: `int` representing the number of turns elapsed
- Current Player: `int` representing the current player
  - 0: Initialization State
  - 1: Player 1's turn
  - 2: Player 2's turn
  - 3: Player 1 has won
  - 4: Player 2 has won
- Player 1: `Player` representing the first player
- Player 2: `Player` representing the second player

Player

- Name: `string` representing the name of this player for display purposes
- IS_AI: `boolean` representing whether this is a player or AI
- AI_LEVEL `int` representing the AI difficulty level
  - 0: None (Human player 2)
  - 1: Easy
  - 2: Normal
  - 3: Hard
- Guesses: `Position list` representing the guesses previously made by this player
- Ship Board: `Board` representing this player's ship board
- Peg Board: `Board` representing this player's peg board (of guesses)

Board

An association list of `Positions` and `Squares`

Squares

A `Square` is either empty, a `Peg`, or a `Ship`

Peg

A `Peg` is a `boolean`, with true or false meaning a red (hit) or white (miss) peg, respectively

Ship

A `Ship` is a `boolean`, representing the statement *this ship has been hit*

Position

A `Position` is a (`Character`,`Int`)

External Dependencies

No external libraries needed, only the oCaml `graphics` module will be used.

Testing Plan

The model will be unit tested, as it is easy to know the output given an input. The controller and display modules, however, cannot be truly unit tested, and will therefore be eyeballed and ensured their behavior is correct. Team members will write and document their own tests and results and share with the rest of the team.