CS 3C Final - Battleship using Pygame

Brian Wen and Jerome Guan
All work is our own unless otherwise specified

April 28, 2022

I Analysis

We were able to meet most of the requirements of this project. Our main program holds classes and functions necessary to run a Battleship game that displays two boards and allows for user interactivity through hits/misses. Furthermore, we were able to develop a version of the program where the computer could guess locations intelligently once a hit has been made. The ship data is loaded in from a JSON that is already written and read at the start of the game.

One fault of our game play is that the user does not know when a ship has been sunk entirely. We might have been able to accomplish this by building an additional ship class that held information on whether or not ships within the fleet have been sunk.

Our code could've also been improved by creating functions that validated for overlap and out of bounds rather than rewriting the same code within multiple functions. Another improvement could have been to make the display more fun by showing actual ships instead of red or blue circles.

A major source of help came from the ImageSprite class provided by the textbook. We made four subclasses of this class to quickly generate the necessary sprites for our game. Another was a Stack Overflow post (https://stackoverflow.com/questions/46390231/how-can-i-create-a-text-input-box-with-pygame) that provided an example as to how to get a input text box onto the screen and how to obtain the text that is inside of the the text box when return is pressed. Before this, we only knew how to display a constant block of text to the screen. This example allowed us to move on to create the attack functionality of the game.

II Reflection

Something that we are most proud of is the bot functionality and the fact that the ships are accurately displayed onto the screen. When initially approaching this problem, we had no idea how we were to build artificial bot intelligence, but the article provided in the project pages helped a lot. We ended up using the method where nearby cells of a hit are added to a stack of potentials. Whenever this stack is empty, the computer will guess randomly.

Something we hoped to have done differently is to add a better user interface and include a game theme. Other examples in class had a water background and displayed real ships and shots, while our program used images that were basic and had no real meaning. Using a better design makes a better playing experience for the user. If we had more time to work on this, we would remove the grid and design one that is not just itself a sprite.

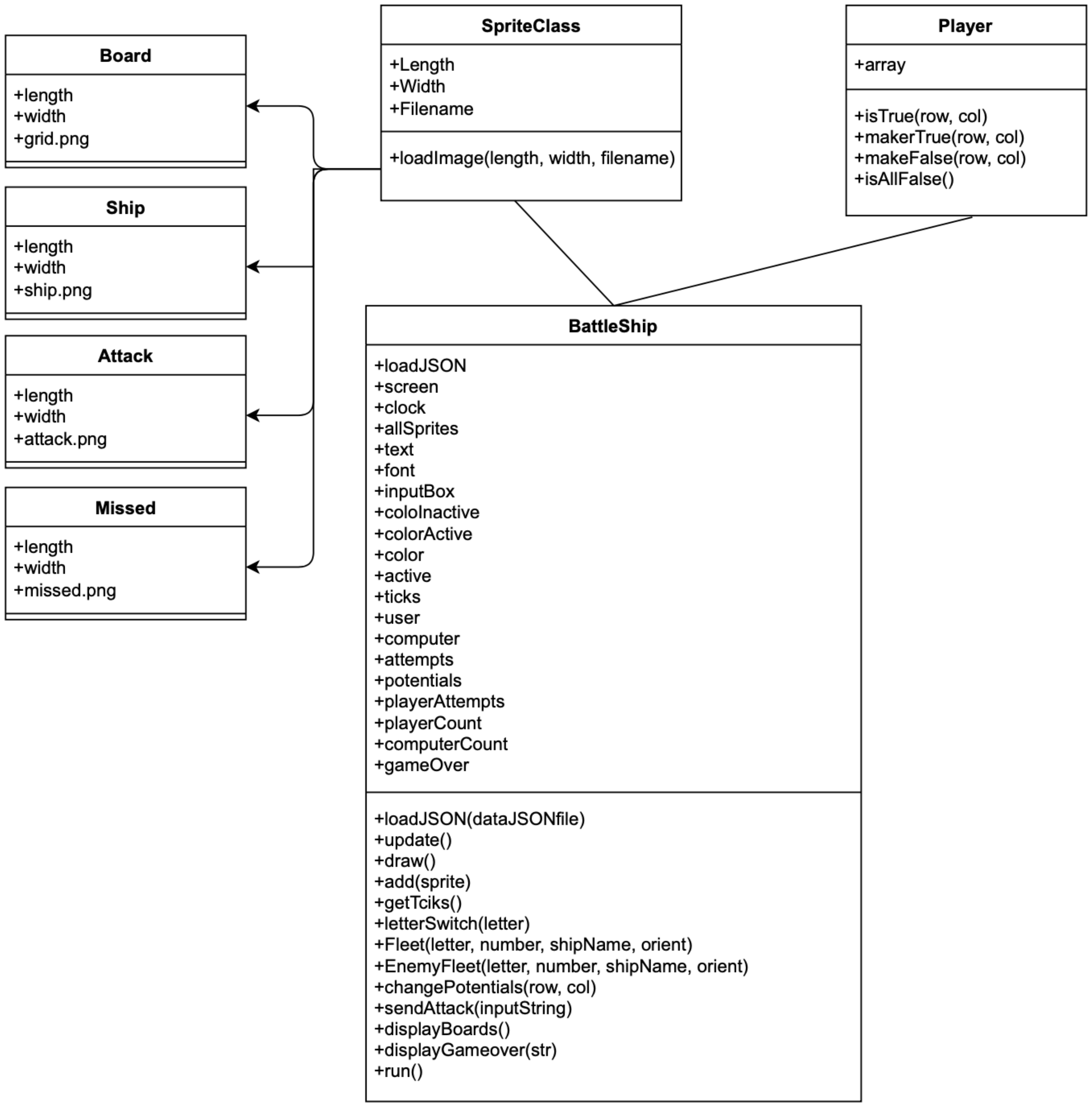
III Test Plan

1. In order to validate that the game properly ends, play the game with knowledge on where the enemy ships are. When the last enemy ship is hit, the game should end.

2. Validate that incorrect input is handled. (Ex: 13,m ; r,2 ; o,17)

3. Validate that when an already guessed cell is passed, nothing happens
4. Validate that hits/misses are correctly placed by referring the the JSON file
5. Validate that the program quits when placed ships overlap or are out of bounds.

IV UML Diagram



Board
+length
+width
+grid.png
Ship
+length
+width
+ship.png
Attack
+length
+width
+attack.png
Missed
+length
+width
+missed.png
SpriteClass
+Length
+Width
+Filename
+loadImage(length, width, filename)
Player
+array
+isTrue(row, col)
+makerTrue(row, col)
+makeFalse(row, col)
+isAllFalse()
BattleShip
+loadJSON
+screen
+clock
+allSprites
+text
+font
+inputBox
+coloInactive
+colorActive
+color
+active
+ticks
+user
+computer
+attempts
+potentials
+playerAttempts
+playerCount
+computerCount
+gameOver
+loadJSON(dataJSONfile)
+update()
+draw()
+add(sprite)
+getTciks()
+letterSwitch(letter)
+Fleet(letter, number, shipName, orient)
+EnemyFleet(letter, number, shipName, orient)
+changePotentials(row, col)
+sendAttack(inputString)
+displayBoards()
+displayGameover(str)
+run()