****

**About High Seas Racers**

High Seas Racers is a sailing game designed to introduce users to the basics of sailing in a quick and fun way. It has 4 tutorials and a race level to gently introducer users to the game. Sailing is widely regarded as a complicated sport, and it is thus necessary to have detailed instructions and tutorial stages before one can begin racing properly. This drives the rationale behind having 4 tutorial stages, 3 pages of detailed instructions and only 1 race stage.

High Seas Racers was designed using Python and Pygame, making heavy use of object-oriented programming.

**Modules**

Other modules written in the course of designing High Seas Racers. In the lib folder, we have three other modules: Scenes, Boat and Buoys. These are the three distinct classes that were required to implement the game realistically. The program is then run from Main.py.

**Scenes**

It was thus necessary to have different scenes containing these stages.

Each of these scenes are subclasses of SceneBase and contain three important functions that are repeatedly called in Main.py to update the screen continuously. They are timerFired(), processInput() and updateScreen(). This is in line with the Movement, View, Controller approach. timerFired() is a “controller” that calls other functions to that will compute the latest position speed and direction of the boat and sail each time the screen is refreshed. These other functions control movement. processInput() is another “controller”. It processes what should be done on key and mouse presses that happen in each scene. Lastly, updateScreen() is a “view” function that paints the screen with the latest changes with each tick of the game clock.

*Menu Scene*

The different scenes in High Seas Racers can be classified into 3 broad categories. The menu scenes, instruction scenes and game scenes (tutorial and race stages). The background image for the menu scenes was designed using Paint, while buttons were drawn using Microsoft Word. Hovering over a button in any of the scenes makes it turn a shade lighter, to let the user know what happens if he clicks on it.

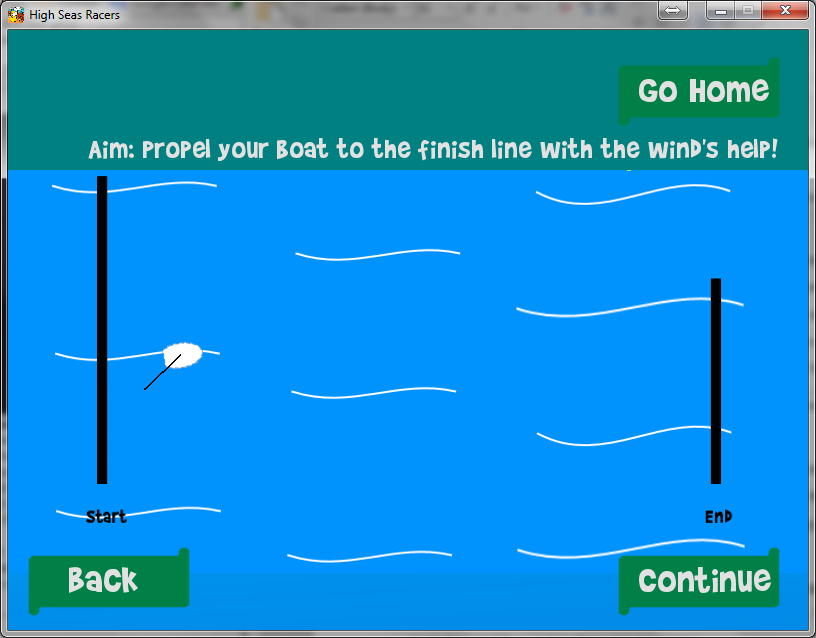


*Main menu scene where hovering over a button changes color*

*Instructions Scenes*

There are 4 stages to the instructions. They are as detailed as possible, because based on studies of other available racing simulators out there, I discovered that there exists no racing simulators out there which explains sailing properly to the user before the race starts. The result is that the only consumers of the game are sailors in real life.

We have player controls to detail what are the keys the user has control of, a demo of a boat sailing across the screen to give the user a better idea of what lies ahead, a diagram to illustrate what direction the boat is allowed to sail in. This can be quite complicating for new sailors, since the boat has a specific optimum sail angle for each direction he can travel in. Some directions are also impossible to move in, such as sailing directly into the wind due to the mechanics involved in the sport. It is thus vital that the user understands these issues before playing the game. Lastly, we have the credits scene, for the game would not have been possible without the people listed there.



*One of many scenes in the instructions page*

These instructions scenes contain a basic layout: they all have continue and/or go back buttons at the same location on the screen. To make my code more compact, I thus made the 2nd,3rd , and 4th pages subclasses of the first instructions scene so that the code for drawing out these buttons were not repeated unnecessarily.

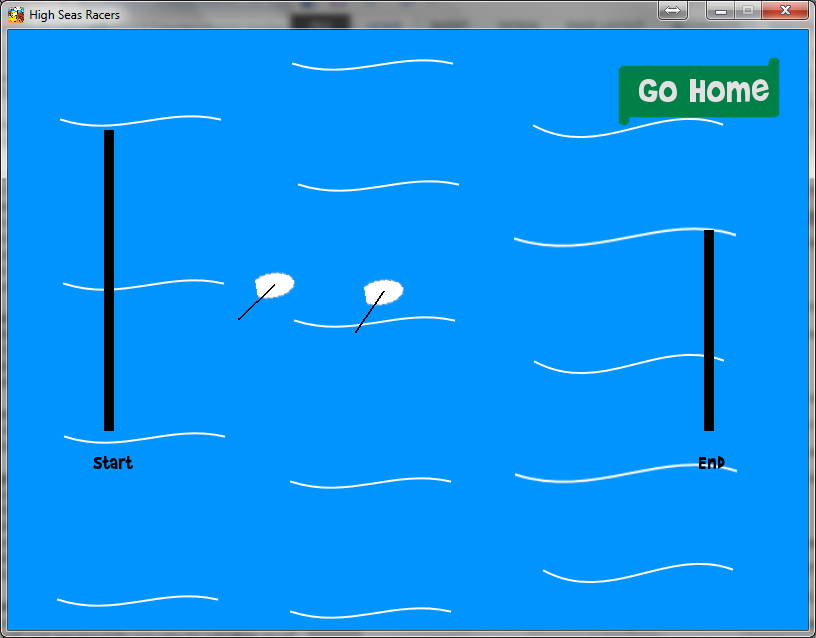
To make matters even easier for the user, he can also push the backspace and return keys on the keyboard to navigate between the instruction scenes and the main menu. This makes for greater efficiency in navigation and thus, a smoother user experience.

*Game Scenes*

All game scenes were subclasses of DemoScene. DemoSceme contains code that are common throughout all the stages of gameplay. These include the drawBoat(), drawSail(), drawBackground() functions, timerFired() that calls updatePosition() and processInput(), since the same 4 keys were used in every stage of gameplay.

The sailAngles and boatAngles are not updated directly during each key press. Rather, Boolean variables are updated so that the user can push and hold the arrow keys, enabling for a smoother experience without having the jab at the keyboard repeatedly.

Following user feedback during the Dec 2 Study-a-thon, it was decided that I should implement a demo boat moving across the screen during the tutorial stages so that the user can have a better idea of what should be done. This was done by printing the coordinates and time as I tried moving the boat across the screen in the original version, saving them in a .dat file and iterating through each line in the file to derive the latest position, speed and direction of the demoBoat.



*One of the tutorial scenes*

In addition, some users kept navigating the boat off the screen by accident. Previously, there was nothing to account for this possibility, so I implemented a “game end” screen that shows whenever his coordinates are greater than the screen dimensions. On the other hand, if a user gets past a stage, a “congratulations” screen will appear, prompting the user to move on to the next stage of gameplay.



*Congratulations image*

The race scene was slightly different from the originals. It has been titled EasyRace because of my original intention to have a moderately difficult and difficult level, but these were scrapped due to time restrictions that arose due to the time spent dealing with bugs in boat control.

**Boat**

The Boat module contains the class Boat. It’s most important function is updatePosition(), which contains a top secret algorithm to compute the speed and direction of the boat given a sail angle and boat direction.

Boat is a subclass of the Sprite class in Pygame. This enables easy collision detection between boats and buoys. The boat itself is an image drawn using Paint. This image can be rotated using the functions available in the Pygame module. The sail contains a line drawn using Pygame functions. The start point is permanently fixed at the center of the sail image. End point is then calculated using sail the sail angle.

It has been decided that the secret algorithm in updatePosition() has to be revealed due to the indecisiveness deposition of the person writing this document. While boat movements *might* appear buggy on the surface, this function will assure you that it is all according to plan. First, it takes into account the Boolean values that were changed with processInput() in the scenes class and gets the latest sailAngle and boatAngle, which is normalized by 360 to make numbers more manageable for debugging. It then calculates the boatAngle and sailAngle from other frames of reference so that the latest velocity can be calculated.

There are three main forces at play with deciding the speed of the boat. They are lift, a push force from the wind and a drag force that acts on the boat as it moves. Lift is derived from the sailAngle relative to the wind. It is smallest when this angle is smallest. It is largest when the sail is 45 degrees to the wind.

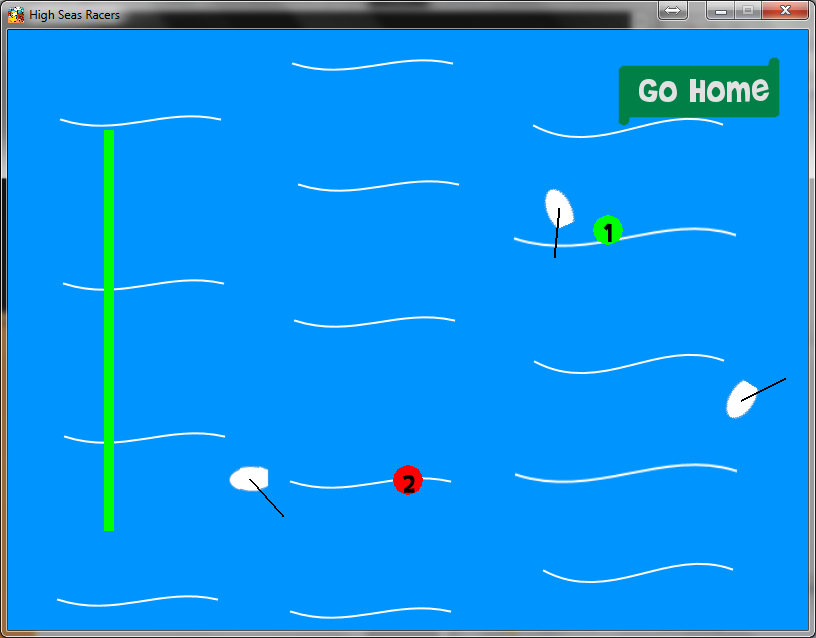
Push is another force generated by the wind pushing on the sail. Factors are arbitrarily decided and it computes an additional component to the vertical velocity of the boat from the sailAngle.

Lastly, drag is a resistive force acting on the boat that is determined using the boatAngle and an arbitrary dragFactor, which is 0.9 in the case.

Autoboat is a subclass of Boat. This Autoboat class contains the code to run the AI boats in the game. For the race scene, 2 Autobo(a)ts are generated for the race scene. One of them is a “smart” boat, completing the race in good time and doubling as an example for new racers to follow. The other is a “not so smart” boat that is unable to complete the race and gets trapped halfway along the course. This remodels real-life sailing, which due to its difficult nature, has many new sailors unable to complete the first few races that they participate in due to the complexity involved.

**Buoys**

Lastly, the Buoy module contains the Buoy class, which contains the draw functions for each buoy. It has another top secret algorithm to detect buoy-rounding so that users cannot cheat even when race officials are not watching closely. In the initializing of each buoy, 4 Boolean values are passed in. These correspond to the 4 points on the compass: North, South, East and West. In order to be considered as rounding a buoy, each boat will have to pass through the invisible lines from the buoy to the North, South, East or West of the screen. These lines are determined arbitrarily based on the positon of the buoy in the direction in which the boat is expected to approach the buoy from. It is only when all four variables are True that the user has rounded the buoy, and this buoy color is changed from red to green to let the user know he is one race leg closer to the finish line.



*Buoys change color as they are rounded. The start line changes too after it is passed*