**Project Design Document**

Abhinav Reddy Gopidi (Z1862225)

Anurag Gudipati (Z1862977)

Venkata Aravind Puppala (Z1859953)

Pooja Reddy Poddutoori (Z1888584)

Csci 513

Prof Mona Rahimi

30 April 2020

**Objective:**

Our objective is to create a fully working Christopher Columbus Adventure game which uses Java and JavaFx following the principle of Software Engineering. The game starts by generating three ships (Christopher Columbus ship and two different pirate ships), a treasure chest, some islands and monsters. The user can select the strategy (Easy, Normal and Hard) that can be used in the game by the pirate ships. The game ends when the Columbus ship reaches the treasure (Win) or the monster catches the Columbus ship (Lose) or either one of the pirate ships catches the Columbus ship. The number of ships, islands and monsters generated are constant. The following design patterns are used in the game.

**Functionalities:**

* Created a large fixed grid.
* We used a **Win** notification when the Columbus ship finds the treasure and a **Lose** notification when the pirate ship or the monster catches the Columbus ship.
* We randomly placed sea monsters which move in a certain area of the ocean.
* We implemented three different search strategies (Easy, Normal and Hard) which can be used by the pirate ships. The user has to select the strategy once the game starts, else it will be set to Easy.
* We implemented Junit test cases for two classes **Shiptest** and **OceanMapTest**.
* We added a **Reset** button which resets the game.

**Design Patterns:**

**Observer**:

The program uses the observer pattern to notify the pirate ships about any movement of Christopher Columbus ship (CCS), here the ship class is to be observed so it implements the subject class which has methods to register, remove and notify the observers and the pirate ship which observes the movements of the ship implements the observer.

**Singleton**:

This program uses the singleton design pattern (singletonMap class) where we can access this class elements from any class without passing any object. By making the method static there is no need for an instance of the class to call it. We use the singleton class to get the dimensions of the grid, to update the location of the Columbus ship, to get the coordinates for the islands and monsters.

**Factory**:

We used the factory design pattern (PirateFactory class) to create the pirate ships in the main class which have different images and origins when generated in the map. Once the start method is executed two different pirate ships are created “normal” and “abnormal” which have different origins and images.

**Strategy:**

We used the strategy design pattern (MoveStrategy interface) to implement the movement of the pirate ships. We have three strategies which implement the MoveStrategy interface (Easy, Normal and Hard) that can be used by the pirate ships. When the game starts the strategy is selected manually and then according to the strategy the pirate ships chase the Christopher Columbus ship.

**Composite:**

We used the composite design pattern (Monster, AreaorMonster classes) to create a group of monsters that move in a certain area. For every movement Monster’s check if the Columbus ship is a child of them. And if the Columbus ship is the child of any then the game is over

**Junit Testing:**

Junit testing was done for two classes. One is for the OceanMap and the other is for the Ship class. OceanMapTesting is done to check for the generation of the map and the dimensions, while the ShipTesting is done to check for the movement of the ship and to check for the updated location in the map.

**Working:**

The main class (OceanExplorer) starts the GUI using JavaFx, draws the ocean with a fixed size. The dimensions are got from the singletonMap class (which implements the **Singleton Design Pattern)**. Then we create two pirate ships using the pirateshipfactory class (which implements the **Factory Design Pattern)** and add them into an arraylist using the pirate ship objects. Then these pirate ships are added as observers. Later we call the methods to load the images of Pirate ships, Treasure chest, Islands, Monsters and Columbus ship. The coordinates for these are bought from the singletonMap class. We didn’t use the random function to generate the coordinates for the ships, monsters and islands, instead we stored the numbers in an array list and shuffled them to avoid repetitions of the numbers. We created three buttons through which we can select the strategy which will be used by the pirate ships. The MoveStrategy interface and the classes: Easy, Normal and Hard implement the **Strategy Design Pattern**. Then the method to start sailing is called where the ship uses the Ship class which implements the subject to generate and check the next point to move the ship, once there are no problems (Pirate ship or the Monster doesn’t catch the Columbus ship) the ship is moved and the imageview is updated in the OceanMap class. For each move the Columbus ship does the pirates ships (which implement the observer class) and the monsters also move accordingly. The Columbus ship and the Pirate ship implement the **Observer Design Pattern.** The monsters don’t move closer to the Columbus ship rather they move in a certain area. The Monster, AreaorMonster classes implement the **Composite Design Pattern.** That is for every move of the Columbus ship the Monsters check if the Columbus ship is a child of any of the Monsters. Whenever the Columbus ship reaches the treasure or is caught either by the pirate ships or the monster a new image pops up with “Win” or “Lose”. The reset button can be used to restart the game.