**Front End:**

1. Create a user interface where user can see ocean in form of grid. New Game button(control) to start a fresh game.
2. User should be able to create game objects like Columbus Ship, Slow Pirate Ship, Fast Pirate Ship, Shark, Island, Treasure.
3. User should be able to create objects by dragging the selected image from Menu option available at left side of window.
4. User should be able to move the columbus ship by pressing the arrow keys.
5. User can see if who the winner is through a pop-up window and then start a new game.
6. User can only create sharks in the pre-defined creature container area.

**Back End:**

1. Create a HTTP Request handler class (App.java) that can receive requests and call the respective methods in Game controller class.
2. Create a Game controller class that is responsible for maintaining rules of the game. Game class will contain static array of characters object called grid.
3. This array object is singleton, i.e., all actors of game will get single grid object throughout the game. If grid is already available, return the available grid to requestors.
4. Elements in grid will represent the type of object(CC/pirate/treasure/shark/whirlpool/island) in specific cell. CC/Pirate/Shark cannot enter island cell.
5. Create a ColumbusShip class that extends Observable and observers are notified of any change in Columbus Ship location
6. CC should have the ability to gain shield when CC enters shield cell. if CC gains a shield, it can kill a Pirate Ship if Pirate ship captures CC.
7. Create methods in ColumbusShip class to update location of CC when user clicks arrow keys.
8. Once CC location is updated, check if CC falls into Whirlpool and if CC falls into Whirlpool, get the new location from Whirlpool and update.
9. If CC reaches treasure island location, set the winner of game as Columbus and stop the game.
10. Once location of CC is updated, CC might be in shark location. In this case, CC is killed and game is over. Delegate the logic of collision check with shark to Visitor by calling accept() method as part of Visitor design pattern.
11. CC will act as Visitee here so CC will also implement VisiteeInterface. Visitee will only have to define accept method which will call visit method of visitor.
12. If after all the operations of CC is completed, notify observers i.e., Pirate ships about the change in CC location.
13. Create an interface Defense as part of Decorator pattern implementation. This interface will have methods for add shield and reduce shield.
14. Defense interface will also have getDefense method to indicate if either CC or wrapper shield has a shield or not.
15. Create a concrete class Shield that implements Defense and its methods. A shield object might contain another Defense reference which might act as wrapper to earlier shield.
16. CC should also implement Defense interface since will have ability to gain/reduce shields. The number of Defense wrappers of CC will indicate number of pirate ships CC can kill if captured.
17. Game (controller) class will check if there is a collision of CC and pirate. If there is collision and CC has shield, then game class will remove Pirate Ship from game and update grid.
18. Create a Pirate Ship Factory abstract class that is responsible for creating new Pirate Ship based on the type provided.
19. Pirate Ship Factory will have abstract method createPirateShip(type) which will be implemented by concrete Pirate Ship Factory class.
20. Other tasks like adding the newly created pirate ship as observer to CC and adding pirate ship to list of pirate ships in game is done in Pirate Ship Factory class.
21. Create a ConcretePirateShipFactory class that extends PirateShipFactory, CPSF will provide implementation for abstract method createPirateShip(type).
22. Create a strategy interface called PirateShip that defines move method. PirateShip will also extend VisiteeInterface since Pirate ship might enter shark location and get killed by shark. So Pirate Ship will act as Visitee.
23. While creating Pirate Ship, user can select type of pirate ship(fast/slow) and this sets the strategy for movement of pirate ship.
24. Once pirate ship is notified of CC location change, Fast Pirate ship(concrete strategy class) can chase CC and slow Pirate Ship can randomly roam around the ocean.
25. If pirate moves to CC location, it can get killed if CC has shield or game gets over, so delegate the collision check with CC to Game(controller).
26. If pirate ship enters Whirlpool, then Pirate ship can be thrown to random Whirlpool location, so get random location from Whirlpool Container.
27. If pirate ship enters shark location, pirate gets killed. Delegate the logic of collision with shark to Visitor by calling accept() method as part of Visitor design pattern. Accept method will call the visit method of visitor.
28. Create an interface Creature and define common functionalities of Sea Monsters/Creatures like move, getLocation. This acts as Component in Composite design pattern.
29. Create a Composite class called CreatureContainer that implements Creature interface. This class holds the boundaries(xCeil, xFloor, yCeil, yFloor) of Leaf nodes in Composite object.
30. CreatureContainer class will contain addMonster() method that can add new Creature to Composite node. Also it should have move method which calls move method of all leaf nodes available in CreatureContainer.
31. CreatureContainer will also act as Visitor, if a CC/pirate ship enters a Shark cell, CC/pirate should be killed. This collision check is handled by Visitor which is CreatureContainer. CreatureContainer will check if coordinates provided matches with any of the shark’s location and kill if matches.
32. CreatureContainer will have 2 visit methods, one for visiting CC and another for visiting Pirate Ship.
33. Create class called GameState to represent the state of game. This class will have array of cell objects and each cell’s text can tell us if it has CC/pirate/island/treasure/shark/whirlpool.
34. Every time user performs action, after all rules are handled, get the grid[][] from game object and create JSON as string.
35. Return the state of the game, i.e., what object is present in which cell in form of JSON string. This JSON is used by front end to map cells with objects.