Implementation

Encounter pirates:
Ships are generated when a MainScreen object is created, and are associated with a given college, 5 ships per college. These ships roam around the map and can be encountered. The ship array that the ships were previously stored in was changed to an ArrayList variable, as this removed issues with some items within the ship array being of the type null, and causing NullPointerExceptions within the code. ArrayList allows the creation of a size-variable list, but in other respects behaves the same way as a regular Array, hence being a justifiable change.

Encounter obstacles:
A new class under Interactables that is a simple collidable object. If the player simply hits them, they will take no damage but they will be stopped dead, and will have to re-manoeuvre. If the player keeps pushing, however, they will take a singular large chunk of 50 damage and be given a grace period of 2 ticks before they are hit again.

Encounter bad weather:
A new class under the new Interactables category, when the player is colliding with bad weather, displayed as clouds, they gain more XP per second but also take incremental damage, meaning that a player could choose to navigate through the clouds (where they also have an obscured view of their ship, so could be hit by cannonballs or collide with obstacles.) in order to gain more XP to be able to move faster, or shoot farther for example.

Combat Ship:
A new class under the actions package, ShipCannon was added to implement other ships shooting at you, calling and already implemented fireCannon function. Ships were added to getCollision in the MainScreen class, to allow for cannonballs to detect when they have hit ships. CannonballAction was changed to only allow the player to damage ships or colleges, rather than all entities damaging each other, as this free for all approach would create impossible games as there would be too few ships to get enough gold, xp, or allies. It could also allow the player to win without doing anything.

Combat College:
As explained above, colleges cannot damage any entity other than the player to ensure the possibility of completing a game or to ensure the player has to actually play the game to win. Furthermore, a new variable isInvulnerable was added to the GameActor class, to allow the possibility that a college cannot be damaged if it is in the final objective, to force the player to complete previous objectives.

Points via combat:
If a College is captured, gold is added to the player’s gold count, along with gold for all the ships associated with that college. If a ship is sunk, gold is added to the player's gold count. Adding gold is done through a call to the addGold function in the ship class

Points via time:

Experience is gained slowly over time in order to aid a struggling player that can manage to survive with a faster ship to outmanoeuvre incoming cannonballs.

Gold from sinking ship:
If a ship is detected as sunk, gold is added to the player's gold count via a call to the addGold function.

Capturing colleges:
If a college is defeated, a call to the setCollege class for each ship associated with that college is executed. This function sets the college for that ship to the player's college, changing the textures for that ship and also removing its ability to shoot, hence making it impossible for it to damage the player.

Objectives:
A new class under interactables was created, called Objective, to create different objectives. This allowed each objective to be random, either to obtain gold, obtain xp, ally ships with your college or use a powerup. If it was a gold or xp objective it would be a random amount of gold or xp to obtain. Objectives would be generated when instantiating an instance of MainScreen. Each objective could only be completed once the previous objective had been completed. This was done through a for loop which only checked the first active objective in the objectives list, and set it to complete if necessary. To stop multiple objectives being completed at once, for example if two consecutive objectives were "Gain 10 xp" and "Gain 8 xp", they would be completed at the same time, but instead objectives only start tracking the change in the game state after the previous objective, increasing challenge. The final objective is always to defeat a random college, and until all prior objectives have been completed, that college would be invulnerable, hence ensuring that the final objective is random, and cannot be completed instantaneously.

Shop:
A shop was added to give the player options to spend their gold, acquired to destroy colleges. This is an essential item as it offers repairs to the ship, which the player would struggle to complete the game without.

Powerups:
Powerups were added extending the InteractableActor class. There are 5 powerups detailed by an Enum within the class. These powerups all look different and occur randomly around the map. They can add the following effects to the player: making them faster, adding health, adding gold, adding experience, and making them invisible and invulnerable.

Difficulty:
The user has the option of selecting a difficulty when creating a new game, which, as the difficulty increases, decreases the player's health and increases all enemies health. It also increases the amount of objectives to complete in the game.

Savegame:
To save and load a game, two significant function were added, saveGame, to the MainScreen class, and loadGame, to the TitleScreen class. These functions use a Java object Properties, which writes and reads to dot properties files. These files are, in essence,

a representation of HashMap in file format. This allows the saving of specific important data points with an unique key associated with the data point, ensuring easy access later on, which made it a very good fit for the problem. The saveGame function hence writes important information about all the objects which make up a game. For weather and obstacles their position was saved, for colleges and ships, position and health, for powerups, type and position, and finally for objectives, whether it was completed, the final objective, what its type was, the goal of the objective and the text it is displaying. Also, the player's experience, health, position and gold were stored. To load a game, the function loadGame would simply instantiate a MainScreen object and edit its properties to match that of the saved game, removing the need to store a very complicated class, and just holding the important information about a game.

Changes to previous software:

A new class, InteractableActor, was created as GameActor is not basic enough for the simpler acotrs. InteractableActor removes health and unnecessary functions from the GameActor class, as obstacles and weather do not require all of the different variables that colleges and ships require.