Introduction

Significant classes

**Desktop application**

* Game: The class acts as the main body of the gaming,
* App: Oversees the shifting between screens/ different stages of the game.
* Player: The player class contains all details related to the player ship including their name, number, location, and the direction they are facing
* Attack: Handles all the attacking mechanics for the game. Performs the necessary calculations to find the winner and loser of each attack and determines what to do based on this result.
* Tile: Super Class to IslandTile, PortTile, PlayerTile and OceanTile. Contains the image for each tile of the board and assigns properties for each separate tile e.g., Can you attack the tile object?

|  |  |
| --- | --- |
| Function Requirement | Classes that meet the requirement |
| FR1 | CharacterScreenController |
| FR2 | Game |
| FR3 | CrewPack |
| FR4 | ChanceCard |
| FR5 | Treasure, Player |
| FR6 | Player, Game |
| FR7 | PortTile, Game |
| FR8 | FlatIsland |
| FR9 | GameScreenController, OceanTile, Player,Tile, PortTile, IslandTile |
| FR10 | Game |
| FR11 | NextPlayerScreen |
| FR12 | Attack |
| FR13 | IslandTile, TreasureIsland |
| FR14 | IslandTile, FlatIsland |
| FR15 | PortTile |
| FR16 | PortTile, AnchorBay |
| FR17 | EndGame |

5. DETAILED DESIGN

Use Case 1.1:

Diagram, box and whisker chart

Description automatically generated

Use Case 1.2:

Diagram, box and whisker chart

Description automatically generated

Use Case 1.3:

Chart, diagram, box and whisker chart

Description automatically generated

Use Case 1.5:

Diagram, box and whisker chart

Description automatically generated

Use Case 1.6:

Diagram, box and whisker chart

Description automatically generated

Use Case 1.8:

Diagram

Description automatically generated with medium confidence

**6. Interface Description**

6.1 **App**

The **App** class contains the main methods that initially launch the program. It contains the methods that enable FXML to work.

|  |
| --- |
| public class App extends Application {   // Is the main method of the project. Starts everything.  public static main(String[] args) { ; }   // Starts the main FXML stage  public static start(Stage) { ; } } |

6.2 **CrewCard**

The **CrewCard** class holds all the data necessary regarding a single Crew card in the game. It holds both a value and a color.

|  |
| --- |
| public class CrewCard {    // The constructor for the class. Creates a new instance with a card value (int) and   // a card color (String).  public CrewCard(int, String) { ; }   // Returns the card value  public int getValue() { return int; }   // Returns the color of the card  public String getColor() { return String; } } |

6.3 **CrewHand**

The **CrewHand** class holds an array of object CrewHand that acts as the player’s current hand of crew hands within the game. Contains functionality to calculate the necessary values for combat and movement.

|  |
| --- |
| public class CrewHand {  public CrewCard[] cards; // Contains all the cards for a CrewHand.  public int totalCards; // Contains the amount of cards currently in the CrewHand.   // The constructor for the class. Creates a new instance.  public CrewHand() { ; }   // Adds a card to the Hand.  public void addCard(CrewCard) { ; }   // Returns the total value of black color cards.  public int getBlackValue() { return int; }    // Returns the total value of red color cards.  public int getRedValue() { return int; }    // Returns the combat value of the player's hand. (red - black or black - red)  public int getCombatValue() { return int; }    // Returns the amount the player can move. The total of all CrewCards.  public int getMoveAbility() { return int; }    // Returns the amount of total cards in the Hand.  public getTotalCards() { return int; } } |

6.4 **CrewPack**

The **CrewPack** class holds and keeps track of all CrewCards in the game and handles handing out cards to players and keeping them all in one place within the Game object. Acts as a card pack, hands out cards on the top of the stack.

|  |
| --- |
| public class CrewPack {  public int totalSize; // Stores the total size of the pack of crew cards.   // The constructor for the class. Creates a new instance.  public CrewPack() { ; }   // Returns a Card at a given index.  public CrewCard getCard(int) { return CrewCard; }   // Returns all cards in the Pack.  public CrewCard[] getCards() { return Crewcard[]; }   // Gives a player a card from the top of the deck.  public CrewCard givePlayerCard(Player) { return CrewCard; }   // Creates a new card pack.  public void newCrewPack() { ; }   // Shifts all cards after one is dealt from the top of the pack  public void shift(CrewCard[]) { ; }   // Shuffles all cards.  public void shuffle() { ; } } |

6.5 **Game**

The **Game** class is the main backend class that handles the entire game. It holds the gameBoard and the player objects. It executes most of the game logic.

|  |
| --- |
| public class Game {  public Player[] players; // Contains all the players (4)  public int turn; // Contains the Player's index of who's turn it is (1-4)  public Tile[][] gameBoard; // Contains all data for the actual board   // The constructor for the class. Creates a new instance.  public Game() { ; }   // Gets the current player (who's turn it is)  public Player getCurrentPlayer() { return Player; }   // Gets a player  public Player getPlayer() { return Player; }   // Gets the current turn index (4)  public int getTurn() { return int; }   // Populates all the tiles with the relevant data  public void populateTiles() { ; } } |

6.6 **GameHandler**

The **GameHandler** class handles all of the object persistence within our game. It saves and loads all player and game data.

|  |
| --- |
| public class GameHandler {   public Player[] players; // Holds all the players (4)   // The constructor for the class. Creates a new instance.  public GameHandler() { ; }   // Continues the game if possible, if not returns false  public boolean ContinueGame() { return boolean; }   // Gets all the players and returns them in an Array  public Player[] getAllPlayers() { return Player[]; }   // Gets all the player's data directly from .json files, returns an array of object Player.  public Player[] getAllPlayersFromFile() { return Player[]; }   // Gets a particular player from file using their playerNumber.  public Player getPlayerFromFile(int) { return Player; }   // Gets all the board's data from file and returns it.  public Game loadBoard() { return Game; }   // Creates a new game, resets files etc.  public void newGame() { ; }   // Saves all players directly to .json files, returns if successful.  public boolean saveAllPlayers() { return boolean; }   // Saves the entire board.  public void saveBoard(Game) { ; }   // Saves a particular player  public boolean savePlayer(Player) { return boolean; } } |

6.7 **GameScreenController**

The **GameScreenController** handles loading FXML scenes for the main game.

|  |
| --- |
| public class GameScreenController {   // Initializes the class  public void initialize() { ; } } |

6.8 **NextPlayerScreen**

The **NextPlayerScreen** handles switching FXML scenes. It contains no public methods.

|  |
| --- |
| public class NextPlayerScreen {  // No public methods } |

6.9 **Player**

The **Player** class holds all the data about a player. It holds data regarding their hands of both crew and chance cards, their treasure, their position and other data.

|  |
| --- |
| public class Player {  public int[] coordinate;  public CrewHand crewHand;  public String playerName;  public int playerNumber;  public Image shipImage;   // The constructor for the class. Creates a new instance.  public Player() { ; }   // The constructor for the class. Creates a new instance with player name and number.  public Player(String, int) () { ; }   // Returns the player's current column coordinate (x).  public int getColCoordinate() { return int; }   // Returns the player's current coordinates.  public int[] getCoordinate() { return int[]; }   // Returns the player's ship image.  public Image getIcon() { return Image; }   // Returns the player's name.  public String getPlayerName() { return String; }   // Returns the player's number.  public int getPlayerNumber() { return int; }   // Returns the player's row coordinate (y).  public int getRowCoordinate() { return int; }   // Sets the player's column coordinate.  public void setColCoordinate(int) { ; }   // Sets the player's coordinate (x, y).  public void setCoordinate(int, int) { ; }   // Sets the player's ship image.  public void setIcon(Image) { ; }   // Sets the player's row coordinate (y).  public setRowCoordinate(int) { ; } } |

6.10 **PlayerTile**

The **PlayerTile** implements **Tile** as it is much easier to implement game tiles this way. Since all tiles have the same functionality, it works out much nicer creating an interface.

This Tile indicates that there is currently a player in this current position.

|  |
| --- |
| public class PlayerTile implements Tile {   // Returns the OceanTile icon.  public Image getIcon() { return Image; }   // Returns if this tile is attackable by a player.  public boolean isAttackAble() { return boolean; }   // Returns if this tile is an island.  public boolean isIsland() { return boolean; }   // Returns if a player can travel to this tile.  public boolean isTraversable() { return boolean; }   // Sets the tile's image.  public void setIcon(Image) { ; } } |

6.11 **Tile**

The **Tile** interface is set up to make using Tiles in the game much easier. It is an interface in which PlayerTile and OceanTile implements. Sets the required details regarding each tile, such as its image and ability to attack.

|  |
| --- |
| public interface Tile {   // Sets a tile icon.  public void setIcon(Image icon);    // Get's a tile icon.  public Image getIcon();    // Defines if a tile is attackable by a player.  public boolean isAttackAble();    // Defines if a tile is traversable by a player.  public boolean isTraversable();    // Defines if the tile is part of an island.  public boolean isIsland(); } |