The Forbidden Island Project

10.08.2021

By Jonathan Yang, Zijian Guo, Colby Frison, Keerthana Krishnan, Eric Zhu

Team Jonathan

SLHS

Katy, TX 77494

Edition 4

Table of Contents:

Section I: About Us & Overview

Jonathan Yang: Team Leader

Keerthana Krishnan: Team Presenter & Editor

James(Zijian) Guo: Mascot

Colby Frison: UI Design

Eric Zhu: Secretary

We are a group of students attending Seven Lakes High School who have been commissioned to create a digital adaptation of the board game *Forbidden Island*™ for educational purposes. This is our outline for the work, which we will complete in the upcoming weeks.

Our goal is to complete the game by the end of the next six weeks, and commence to perfect the digital game interface in the following six weeks before presenting the product to our classmates. We have distributed certain jobs amongst ourselves in order to carry out the functions needed to complete planning, presenting, programming, and testing our final product.

We planned our code for this project by creating a UML diagram which details the interfaces, enumerations, classes, variables, and methods required for this project to run. We also summarized the UML so that the criteria for each class is clearly defined for each member of the group. We met on Zoom during the weekend to ensure that all participants have a clear understanding of what their role is in making sure that this project is a success. We then wrote a technical report detailing our findings and organization of the program. We have outlined classes and functions for the game, player, cards, tiles, and treasures, and we have planned for the game to smoothly carry out functions such as drawing the island, rotating the turns, keeping track of treasures captured, and flooding and sinking islands.

We will, by the end of the fourth six weeks, finish all aspects of the program to be ready for use.

Section II: Distribution of Work

- The workload for the prospectus and presentation of the program were distributed as follows:
    - The UI/Game visual design and processes are done by **Colby Frison**.
    - The Class UML/hierarchy and interactions are done by **Jonathan Yang** and **Zijian Guo**.
    - The test-cases and problem finding are done by **Jonathan Yang** and **Eric Zhu**.
    - The revising and editing is done by **Keerthana Krishnan**.
    - Formatting is done by **Zijian Guo** and **Eric Zhu**.
    - Additional features of the program will be implemented by **Zijian Guo**.
    - The Google Slides presentation is created by **Everyone**.

- The workload for coding and debugging of the program will be distributed by classes as follows:
    - All enumerations are to be implemented by **Keerthana Krishnan** and **Jonathan Yang**
    - IslandTile and IslandGrid are to be implemented by **Eric Zhu**
    - Treasure and Game are to be implemented by **Jonathan Yang**
    - Player is to be implemented by **Keerthana Krishnan**
    - GamePanel and Application are to be implemented by **Colby Frison**

Section III: Time Outline

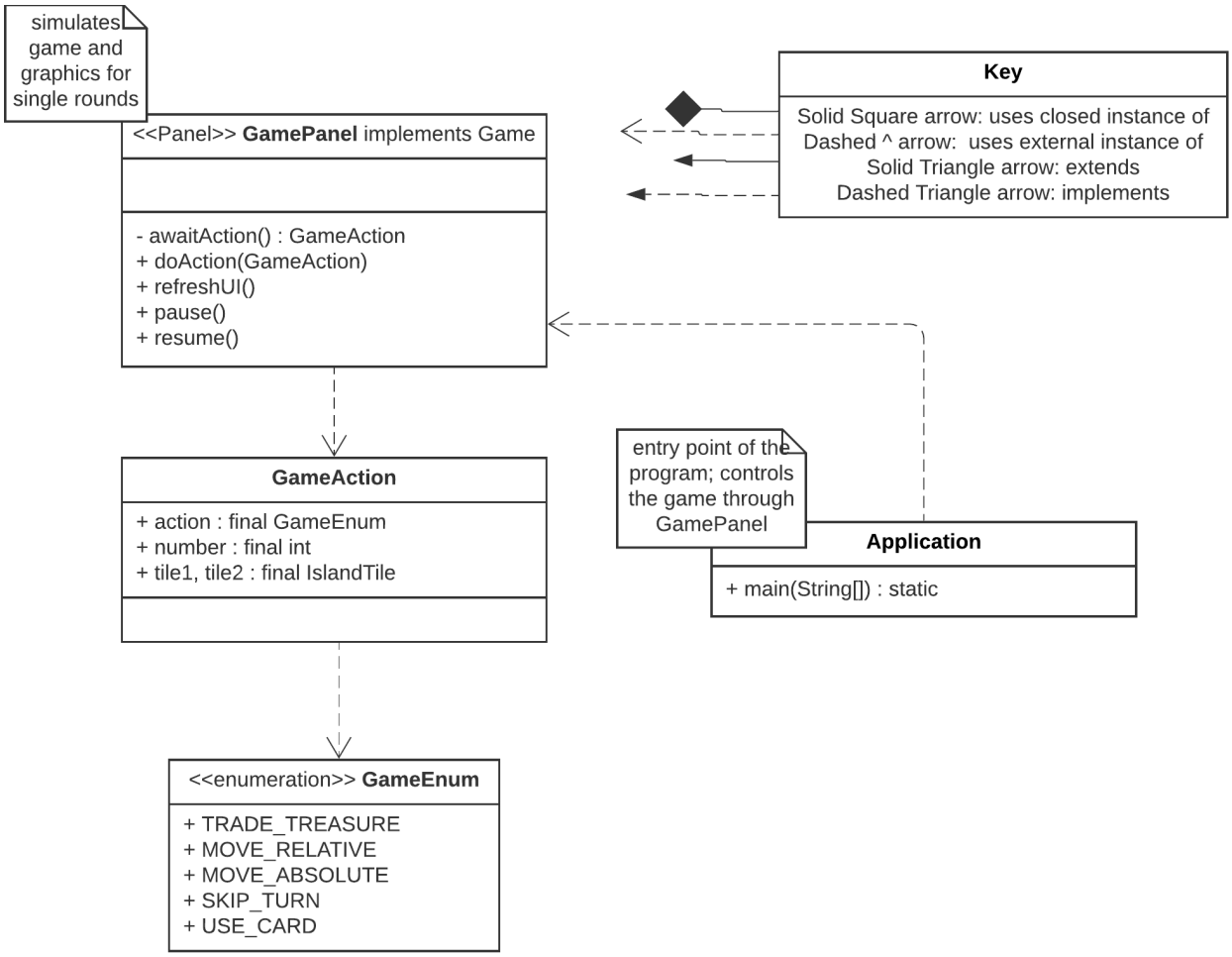
All members of the team were assigned roles at the beginning of the project, which were implemented individually, but reviewed over as a team

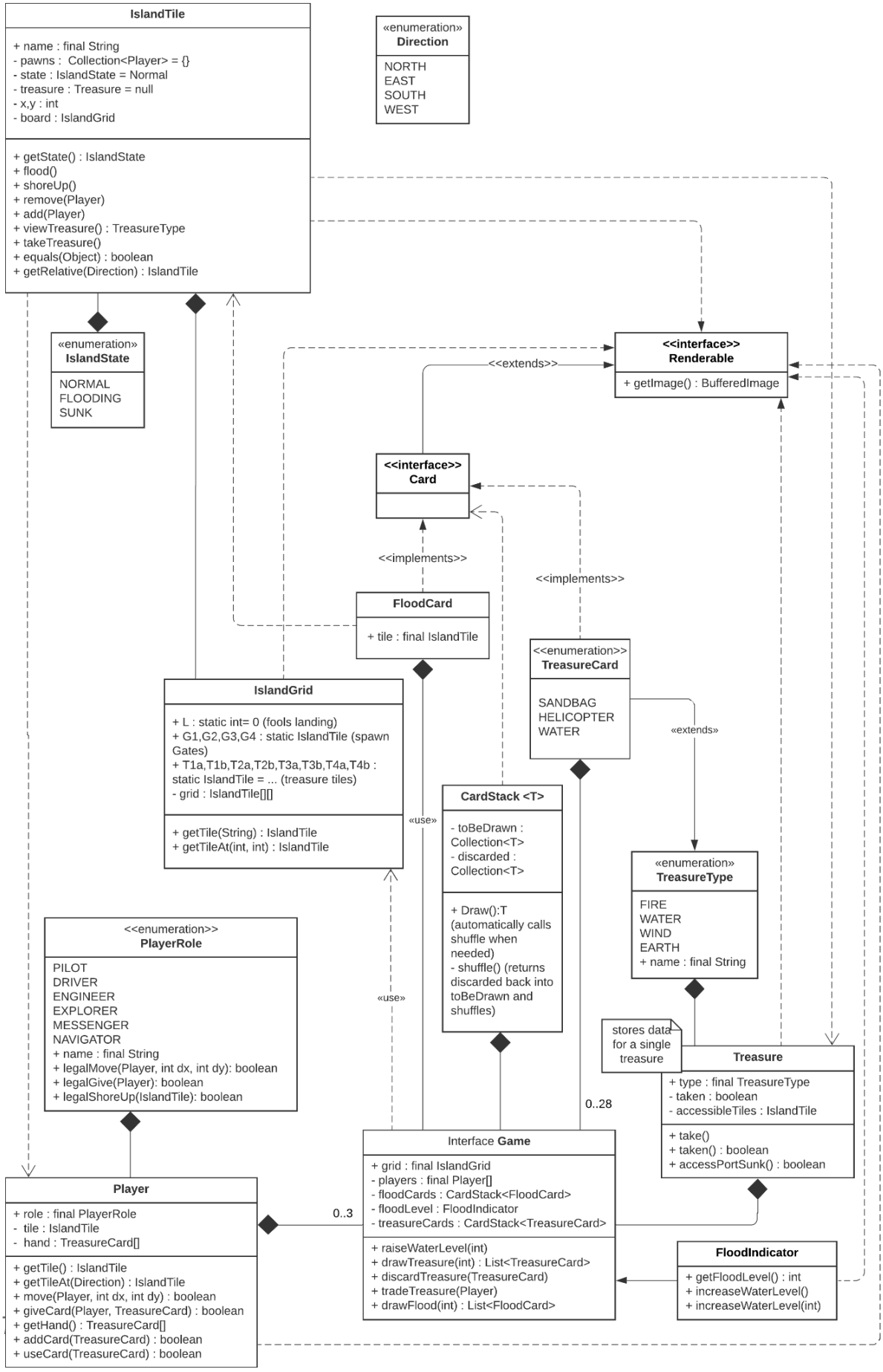
- **Completed Milestones:**
    - On **October 12, 2021**, Zijian Guo and Jonathan Yang started the outline for the technical paper.
    - On **October 13, 2021**, the entire team viewed the rules of the game and brainstormed how we would approach the game.
    - On **October 14, 2021**, we created and distributed roles and responsibilities to each member of the team.
    - From **October 15-October 17, 2021,** Jonathan Yang worked on the UML diagram, completing it by the **October 17** due date (many revisions were made after, but the basic structure was created).
    - On **October 18, 2021**, Keerthana Krishnan started inputting the information from the completed UML into a Google Slides presentation.
    - From **October 20, 2021** to **October 23, 2021**, the GUI was designed by Colby Frison**.**
    - Planned to finish the rough draft of the prospectus by **October 20, 2021**, but completed on **October 22, 2021**.
    - On **October 23, 2021**, the entire team held our first outside-of-school meeting, and discussed the slideshow.
    - Over the next week, from **October 23-30, 2021**, we refined our data.
    - On **October 29, 2021**, we wrote the gameplay, test data, and edited what we had of the paper.
    - On **October 30, 2021**, we held our second meeting in order to finalize the slideshow and presentation.

- From **October 28 to November 3, 2021**, we reviewed and finalized our prospectus and presentation, ensuring its accuracy, detail, and presentability.
- On **November 1, 2021,** we held a session over Zoom to review the technical paper.
- On **November 3, 2021,** Keerthana Krishnan updated the presentation and the entire team reviewed the technical paper.
- On **November 4, 2021,** we presented our report and slideshow.

- **Projected:**
  - On **November 5, 2021**, we will hold a meeting to set up code sharing within our team and begin coding
  - From **November 5, 2021** to **February 12, 2021**, we plan to complete the code of the project
  - On **January 4, 2021**, we will complete the classes and begin debugging
  - On **February 12, 2021**, we will complete the project in its entirety
  - On **February 14, 2021**, we will submit the final project

Section IV: Current UML:




IslandTile
+ name : final String
- pawns : Collection<Player> = {}
- state : IslandState = Normal
- treasure : Treasure = null
- x,y : int
- board : IslandGrid
+ getState() : IslandState
+ flood()
+ shoreUp()
+ remove(Player)
+ add(Player)
+ viewTreasure() : TreasureType
+ takeTreasure()
+ equals(Object) : boolean
+ getRelative(Direction) : IslandTile
«enumeration»
Direction
NORTH
EAST
SOUTH
WEST
«enumeration»
IslandState
NORMAL
FLOODING
SUNK
<<interface>>
Renderable
+ getImage() : BufferedImage
<<extends>>
<<interface>>
Card
<<implements>>
<<implements>>
FloodCard
+ tile : final IslandTile
<<enumeration>>
TreasureCard
SANDBAG
HELICOPTER
WATER
«extends»
IslandGrid
+ L : static int= 0 (fools landing)
+ G1,G2,G3,G4 : static IslandTile (spawn Gates)
+ T1a,T1b,T2a,T2b,T3a,T3b,T4a,T4b : static IslandTile = ... (treasure tiles)
- grid : IslandTile[][]
+ getTile(String) : IslandTile
+ getTileAt(int, int) : IslandTile
«use»
CardStack <T>
- toBeDrawn : Collection<T>
- discarded : Collection<T>
+ Draw():T (automatically calls shuffle when needed)
- shuffle() (returns discarded back into toBeDrawn and shuffles)
«enumeration»
TreasureType
FIRE
WATER
WIND
EARTH
+ name : final String
<<enumeration>>
PlayerRole
PILOT
DRIVER
ENGINEER
EXPLORER
MESSENGER
NAVIGATOR
+ name : final String
+ legalMove(Player, int dx, int dy): boolean
+ legalGive(Player): boolean
+ legalShoreUp(IslandTile): boolean
«use»
stores data for a single treasure
Treasure
+ type : final TreasureType
- taken : boolean
- accessibleTiles : IslandTile
+ take()
+ taken() : boolean
+ accessPortSunk() : boolean
0..28
Interface Game
+ grid : final IslandGrid
- players : final Player[]
- floodCards : CardStack<FloodCard>
- floodLevel : FloodIndicator
- treasureCards : CardStack<TreasureCard>
+ raiseWaterLevel(int)
+ drawTreasure(int) : List<TreasureCard>
+ discardTreasure(TreasureCard)
+ tradeTreasure(Player)
+ drawFlood(int) : List<FloodCard>
Player
+ role : final PlayerRole
- tile : IslandTile
- hand : TreasureCard[]
+ getTile() : IslandTile
+ getTileAt(Direction) : IslandTile
+ move(Player, int dx, int dy) : boolean
+ giveCard(Player, TreasureCard) : boolean
+ getHand() : TreasureCard[]
+ addCard(TreasureCard) : boolean
+ useCard(TreasureCard) : boolean
0..3
FloodIndicator
+ getFloodLevel() : int
+ increaseWaterLevel()
+ increaseWaterLevel(int)

Section V: Classes, Methods, and Variables

The **Game** *interface* will contain all the processes of the entire simulation.

- ❖ Public Instance Variables
  - ➢ The **final IslandGrid** grid will create the **Island.**
- ❖ Private Instance Variables
  - ➢ The **final Player array** players will ensure that the **Game** rotates between one **Player** and another in a consistent manner.
  - ➢ FloodCards is an **ArrayList** of **FloodCard** values, and it will represent the deck of Flood Cards.
  - ➢ TreasureCards is an **ArrayList** of **TreasureCard** values, and it will represent the deck of Treasure Cards.
  - ➢ The **integer** floodLevel will represent the current value of the **FloodMeter.**
- ❖ Methods
  - ➢ The raiseWaterLevel *method* receives an integer value representing the amount to increase the water level by (no need to check if treasure cards contain flood cards, only need to pass in the "Waters Rise" card count)
  - ➢ The drawTreasure *method* receives an integer value of the number of cards to draw and returns a corresponding List of TreasureCards
  - ➢ The discardTreasure *method* receives a **TreasureCard** value and adds the card to the discarded collection.
  - ➢ The tradeTreasure *method* receives a **Player** value and attempts to exchange the cards in the **Player**'s hand with the respective **Treasure**.
  - ➢ The drawFlood *method* receives an integer value of the number of cards to draw and returns a corresponding List of FloodCards, or if no arguments are given, will default to the water level of the game.

The **IslandTile** *class* will represent each islet tile of the bigger island.

- ❖ Private Instance Variables
  - ➢ The final **String** name holds the name of the IslandTile
- ❖ Public Instance Variables

- ➢ The **Player Collection** players will hold the values of the players situated on the tile.
- ➢ The **IslandState** will be set to Normal at the beginning of the game and change to Flooded if it is flooded by a card.
- ➢ The **Treasure** value treasure will be set to null.
- ➢ The **integer** values x and y will represent the location of the **IslandTile**.
- ➢ The **IslandGrid** board represents the current **IslandGrid** layout.

❖ Methods

- ➢ The method viewTreasure will return the **TreasureType** (which stores a representation of the treasure) available on the tile if the **IslandTile** corresponds to a certain untaken treasure, otherwise null if there is no treasure.
- ➢ The method getState will return an **IslandState** Normal, Flooded, or Sunk to communicate the state of the **Island**.
- ➢ The methods flood and shoreUp will change the **IslandState** to more or less flooded, respectively.
- ➢ The methods addPlayer and removePlayer will receive a **Player** and will allow the **Player** to move on and off the **IslandTile.**
- ➢ The equals *method* compares **IslandTile**s in order to allow the **IslandTile** to be stored in a Set.
- ➢ The method takeTreasure will remove the **Treasure** from the tile and set the state of the corresponding **Treasure** to "captured" (validity check required by caller).

The **IslandGrid** class will create a grid of all the **IslandTile**s, which will represent the game board.

❖ Private Instance Variables

- ➢ The public **IslandTile** fields (i.e. L, G1, T1a) correspond to important island positions

❖ Public Instance Variables

- ➢ The internal 2D **Array** is initialized with **IslandTiles** in a random order.

❖ Methods

- ➢ The **getTile** *methods* return **IslandTile**s based on provided information.

The **Treasure** *class* will represent the items the players can capture. It will be used to determine the outcome of the game.

- ❖ Public Instance Variables
  - ➢ The public final **TreasureType** corresponds to the type of treasure (Earth Stone, Statue of Wind, Crystal of Fire, Ocean's Chalice).
- ❖ Methods
  - ➢ The take *method* changes an assigned boolean value to reflect whether the **Player** has captured the treasure.
  - ➢ The taken *method* returns whether the piece has been taken.
  - ➢ The accessPortSunk *method* returns whether the piece can still be captured. If this method returns a value of false, the game is over.

The **Card** *interface* will organize the different types of cards for the **CardStack** *class*

The **CardStack** *class* will model a stack of cards.

The **FloodCard** *class* implements **Card** and will link to an island, automatically causing it to enter a more flooded state when drawn (exists solely for organizational purposes).

- ❖ Private Instance Variables
  - ➢ The public final **IslandTile** tile stores the **IslandTile** which it floods.

The **TreasureCard** *enumeration* will either hold a treasure value OR a special value (Waters Rise, Sandbag, Helicopter Lift).

The **FloodIndicator** *class* will store the flood level and generate the image

The **Player** *method* controls and organizes the different functions of the **Players**.

- ❖ Private Instance Variables
  - ➢ The TreasureCard Array Hand represents the **Player's** hand
  - ➢ The IslandTile
- ❖ Public Instance Variables
  - ➢ The **Player** will store a final **PlayerRole** enumeration of: Pilot, Driver, Engineer,

Navigator, Messenger, and Explorer; the enumeration will have methods that function as role-specific condition checkers, including LegalMove, LegalGive, and LegalShoreUp (which returns true if the specified action is legal).

- Methods
  - The getTile *method* returns the **Island** the **Player** is on.
  - The getTileAt *method* receives a **Direction** *enumeration* and returns an **Island** to the corresponding **Direction** in relation to the location of the **Player**.
  - The move method will return true if the **Player** can move (the **Player** is passed as an argument to account for Navigator moving others)
  - The giveCard method receives another **Player** and a card, and returns whether the give is executed (it will be executed if legal)
  - The getHand *method* returns an Array of **TreasureCards** which will show the **Player**'s hand (will be a subarray of the **Player's** hand).
  - The addCard *method* receives a **TreasureCard**, and if the **Player**'s hand is less than 5, the next null value in their hand will become the **TreasureCard**, a boolean value will be returned specifying whether the action was successful**.**
  - The useCard *method* will remove a **TreasureCard** from the **Player**'s hand and will return true if successful.

The **Renderable** *interface* will require children classes to implement a method that allows the element to be drawn onto the screen

The enumeration GameEnum will hold the possible moves in a game

- Public Instance Variables
  - TRADE_TREASURE holds the method to trade **TreasureCards**.
  - MOVE_RELATIVE holds the method to move to legal **IslandTile**s relative to the **Player.**
  - MOVE_ABSOLUTE holds the method to move to another **IslandTile**.
  - SKIP_TURN allows the **Player** to skip or end their turn.
  - USE_CARD allows the **Player** to discard 4 **TreasureCards** and capture a **Treasure.**

The **GameAction** *class* will execute all the processes of the **Game**

- ❖ Public Instance Variables
  - ➢ The final GameEnum action will allow the class to access the values in the enumeration listed above.
  - ➢ The final integer number will store any needed number value to communicate an action (mainly, if move absolute contains more than 1 player on the starting tile, it will store a bit collection of players that are moved)
  - ➢ The final IslandTile values tile1 and tile2 will indicate any tiles needed to communicate an action (specifically, starting and ending tiles)

The **GamePanel** *class* will implement the **Game** *interface* and handle all user interfaces and **GameAction**s.

- ❖ Methods
  - ➢ The awaitAction() *method* returns the next **GameAction** of a user.
  - ➢ The doAction *method* receives a **GameAction** and allows for the moves to be executed.
  - ➢ The refreshUI() *method* updates the graphics data and repaints.
  - ➢ The pause() *method* disables players from moves.
  - ➢ The resume() *method* undoes pauses and allows the players to move again.

The **Application** *class* will obtain information like the seed needed to generate the board and run the simulation.

Section VI: Mock Up of the Visual Interface

Menu/Starting Panel

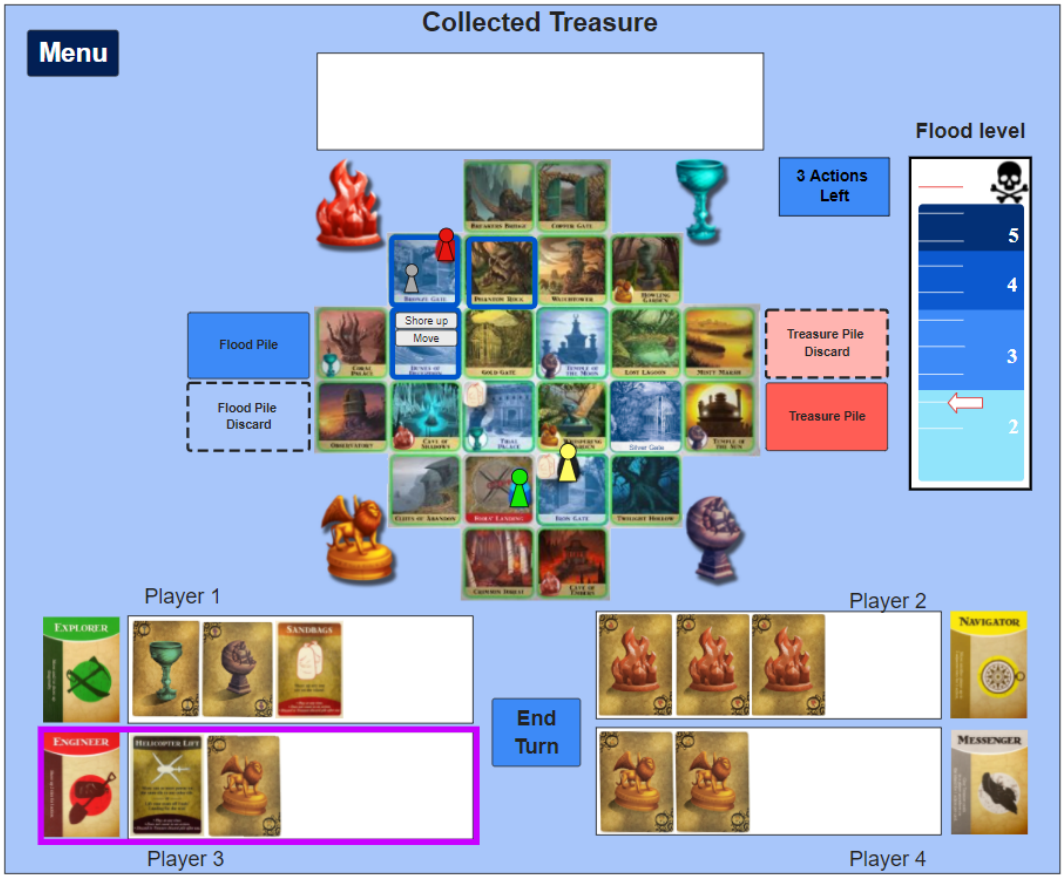
The menu allows the player to either start the game, learn the rules, or quit the game.

Starting Value Selection

This display uses buttons, so that the player cannot enter an invalid response. They can choose to play with 2, 3, or 4 players. They can also choose a novice, normal, elite or legendary level of difficulty.

Game

The cards and treasures will be laid out in the pattern seen below. On the top left corner, a button for the pause menu is displayed. On the upper center area, there is a space on which when treasures are captured, they will be displayed. The rectangle and arrow on the extreme right will show the current flood level. On the bottom, the players, their roles, hands, and inventories will be depicted. There is also a box to tell how many more actions are player can execute in their turn and a button to prematurely end a player's turn.



Pause Menu

If a player wishes to pause the game, they must click the menu button. On the pause screen, they can choose to resume play, review the rules, or end the game.

Win Screen

When all four treasures are captured and the players escape Fool's Landing, this graphic will be displayed.

Lose Screen

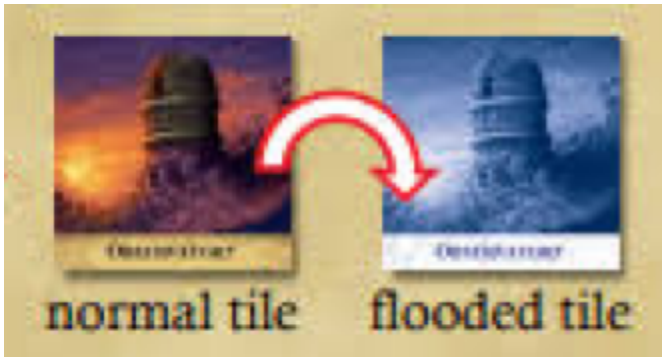
If a player cannot escape a sunk tile, Fool's Landing sinks, or all the treasures are not captured, this graphic will be displayed.

Section VII: Game Components

Game Tile

The tile graphic represents a piece of the board with which the players can interact. A tile can have three states: these are, in order of increasing states of flooding: Normal, Flooded, and Sunk.

A flooded tile can be reverted to the normal state by the process of shoring up. However, if a tile reaches the Sunk state, it cannot change its state nor interact with players (except for diver) for the remainder of the game.



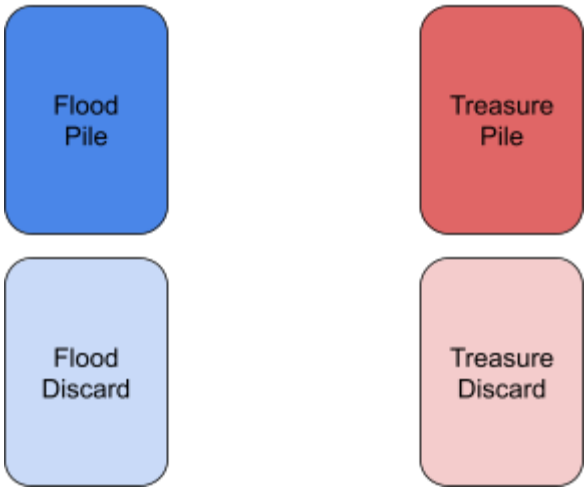
Game Board

A two dimensional array of independent island tiles, arranged like the adjacent image, will be generated randomly.

Cards

Flood cards are drawn from the flood pile and afterwards discarded in the flood discard pile. Treasure cards are drawn from the treasure pile and afterwards discarded in the treasure discard pile

Treasure and Flood Card Stacks



Flood Card

Flood Cards correspond to tiles. When drawn, the tile enters a more flooded state (In increasing order: Normal, Flooded, and Sunk).

Treasure card

Each corresponds to either treasures (traded in on specific tiles and 4 cards of the same type needed to capture each treasure), or special actions, which include "Waters Rise!" (increases flood level, resets the flood deck, and redraws flood cards), "Helicopter Lift" (used to end the game or to move any number of players from one tile to one other tile), and "Sandbag" (used to shore up any one tile).

Treasure

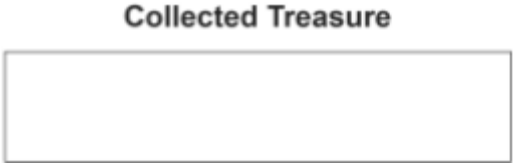
Special

Treasure

The four treasures are the Earth Stone, the Crystal of Fire, the Statue of the Wind, and the Ocean Chalice. Part of the goal of the game is to capture treasures by trading in 4 cards of the same treasure in a designated tile. Collected treasures will be displayed in the "Collected Treasure" box. The players win when all four treasures are captured.

Treasure Piece

Collected Treasure



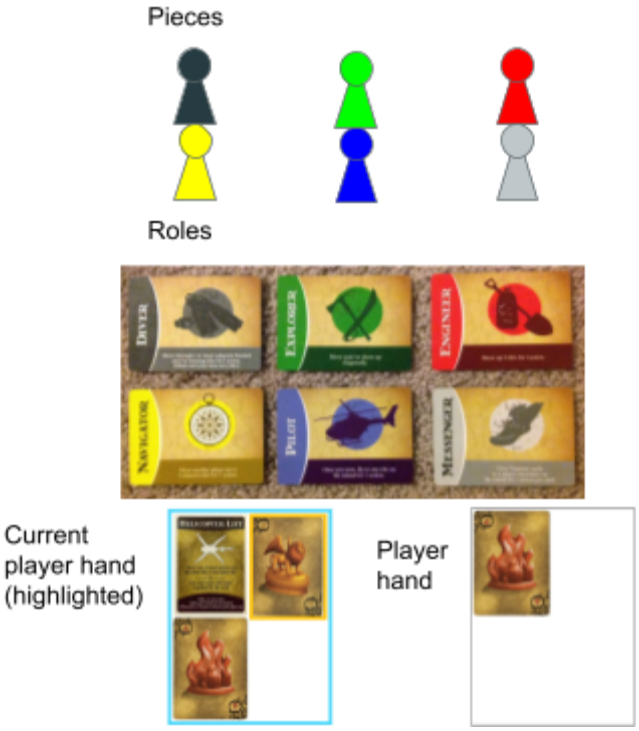
Flood Indicator

As the game progresses, the water level will rise, and this will be displayed in the flood indicator. The number the water is raised to on the flood indicator shows the number of flood cards to draw at the end of a player's turn.

If the indicator reaches the skull and crossbones, the players lose.

Players

The players are parts of the game that users of the game interact through. They are represented in the game by pawns, which mark the position they are at on the island, roles, which describe special abilities of each player, and their hands, which shows the cards they currently have. A player's hand is limited to five cards.



Section VIII: GamePlay

Initialization

1. To start the game, show an option panel with rules, which allows the users to select the number of players, seed to shuffle the cards, and game difficulty. This will be a JOptionPane and the values returned will provide the information needed to instantiate the Game.
2. Afterwards, the correct number of Players are instantiated with random and unique roles in the Game, stored in an ordered collection.
3. The IslandGrid is instantiated, which will initialize the board by shuffling and distributing copies of 24 pre-generated IslandTiles
4. The set of pre-generated FloodCard and TreasureCard collections are copied and shuffled. The six FloodCards drawn from the deck will be enacted, flooding the corresponding Islands, and two TreasureCards will be dealt to each player. If a "Waters Rise!" card is drawn, it will be returned to the deck, and the TreasureCard collection will be shuffled again.

Game Loop

5. Now the TopLevel class will progress through the game, specifically, providing a button to view options and the Game itself. The TopLevel will call functions in the Game to get and execute every move through the Game while the Game is only responsible for providing these functions (see UML), so that the data for WIN/LOSS can be intercepted and displayed by the TopLevel.
6. The TopLevel container will also display the number moves made in a player's round.
7. Game will display and return all processes that occur as part of the actual game (if legal, otherwise will return nothing) To avoid any possible misclicks, the player will be made to confirm their move:
    a. When moving a player, clicking the game piece causes it to be highlighted, then selecting its destination piece will complete the move.

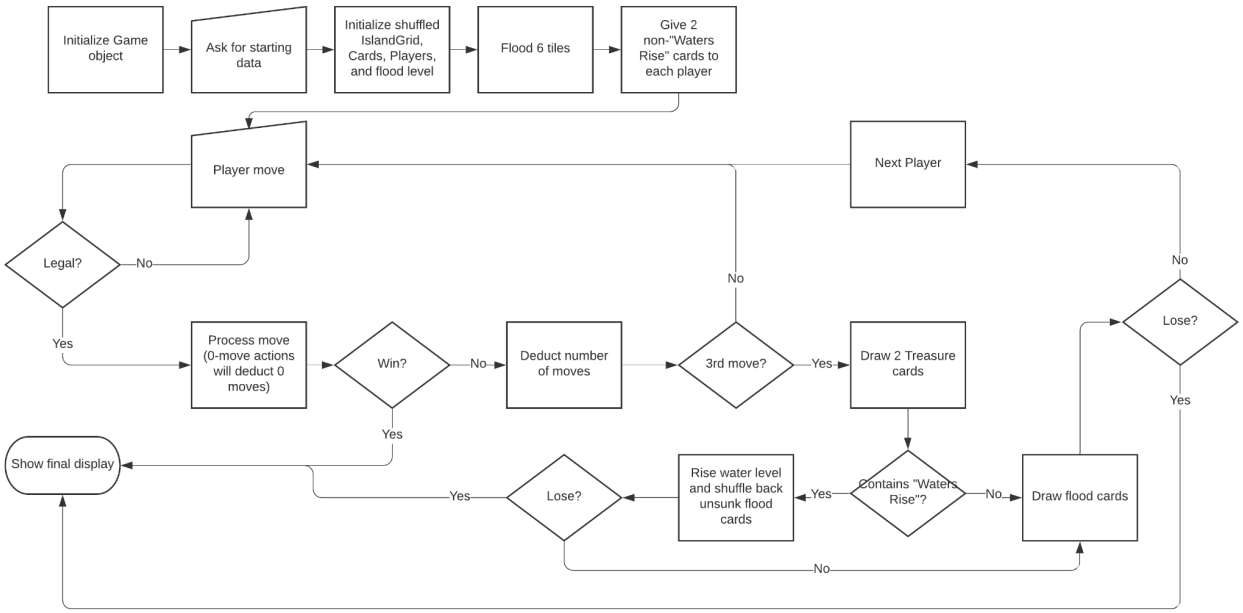
b. When shoring up a tile, right clicking on the tile will create a dropdown menu, and selecting the “Shore Up” option will complete the move.
c. If a player is on a treasure tile and has four cards matching the tile, a button will appear to the bottom right that allows the player to capture the treasure.
d. Clicking a card from the current player’s inventory allows the player to give cards to a player who has fewer than 5 cards (and if it is a special card, the entire board is highlighted so that the tile(s) that it will be used on can be selected), clicking the card again will deselect it:
    i. To use a “Sandbag” card, select it out of the inventory, and select the tile it will be used on.
    ii. To use a “Helicopter Lift” card, select it out of the inventory, and select the start and destination tiles.

8. Ensure that once all treasures have been collected, the “Collected Treasures” box will be highlighted, and if the players are all on Fool's Landing, any “Helicopter Lift” cards will be highlighted (the getImage function of some cards will be written to return different values depending on the stage of the game).

Endgame

9. Once a game ends, the option to reset the game along with the outcome of the game will be displayed.

Logical Flow



Section IX: Test Data

Initialization

Case occurrence	Appropriate game response
When the option panel is displayed, we will play the game with 2, 3, or 4 players by selecting the respective option and a “Start Game” button.	This ensures that roles are randomly assigned only once to each player. And the program will draw the IslandGrid, Players, Cards, and FloodMeter. This will also ensure that the difficulty level is reflected on the FloodMeter.
Six FloodCards are drawn, and two TreasureCards are dealt out to each player.	This will test the methods that detect a “Waters Rise!” card and shuffle it back in the deck in addition to testing the flood() method.

Game Loop

The player executes their turn.	This ensures that the simulation allows for the player to execute no more than three legal moves and that moves unique to certain roles only function for the player with that role.
The player chooses to end their turn.	This ensures that the turn will automatically be rotated to the next player.
The player chooses to play a special card.	This ensures that the special cards of “Helicopter Lift” and “Sandbag” may be played at any time by any player.
The player plays a Sandbag Card.	This will ensure that the tile the card is placed upon will be deterred from changing to a Flooded state. If the tile is already in a Flooded state, it will be deterred from entering a Sunk state. This should be allowed at any time in the game.
The player plays a “Helicopter Lift” card	This will move all the players on one tile to another tile. This will also allow players situated on “Fool’s Landing” to escape the

	Island, which is required to win.
The player moves to an adjacent tile.	This will ensure that the player's pawn will change position to another tile by exactly one, or in the case of the Navigator, 2 tile length(s) east, west, north, south, or in the case of the explorer, diagonally. Additionally, if the player is the Navigator, this move will test whether said player, and said player only, is able to move another pawn up to two adjacent tiles. If the player is the Diver, they should be able to skip over tiles that return a state of Flooded. We will also ensure that the pilot may move anywhere.
A player escapes a sunk tile by swimming to an adjacent, unsunk tile.	This ensures that game allows the player to swim to an adjacent tile if possible. If not, the game should automatically end.
A player shores up an island.	This will ensure that a player, during a turn, can only shore up one, or in the case of the Engineer, 2, tiles that return a state of Flooded, that are adjacent, or in the case of the Explorer, diagonal, to the player.
A player discards four TreasureCards	If all four TreasureCards have an equal TreasureType, we will test whether this action will result in the respective Treasure's taken method returning a value of true and changing position into the box labeled "Captured Treasures." If not, nothing should happen.
A player decides to give a card to another player.	This will test whether the players can carry out those actions with the condition that players may not move to or shore up tiles out of reach, capture non-matching treasures, nor give to themselves. As we play the round, we must check that the players can only hold five cards at a time. This should only be possible if both players are on the same Island.
The player successfully executes three moves.	At the end of the turn, we must ensure two TreasureCard and an amount of FloodCards corresponding to the number on the FloodMeter are automatically drawn.

The player draws a “Waters Rise!” card during gameplay.	This ensures that any “Waters Rise!” cards are automatically used to raise the water level on the flood meter before being shuffled back into the deck of used flood cards to reintroduce all the discarded cards into a new deck.
The TreasureDeck runs out.	All discarded cards are immediately and automatically reshuffled into a new deck.

Endgame related

Players use a “Helicopter Lift” card on Fool's Landing without having collected all 4 treasures.	The game is lost, and the lost game graphic will be displayed.
All the treasures are collected, all players returned to Fool's Landing, and a “Helicopter Lift” card is played.	The game is won only when these conditions are satisfied. The win screen is displayed.
Both locations pertaining to a treasure are sunk.	The game is lost, and the lost game graphic will be displayed.
The game has reached an outcome.	The game ends after, and only after, it has reached a definitive outcome, and the appropriate outcome will be displayed.

Section X: Possible Hardships and Solutions

During this endeavor, it is very likely that our team will encounter hardships and shortcomings. Fortunately, we have planned out solutions in the case of the most likely contingencies.

Special abilities, such as the Explorer and Navigator roles may prove challenging due to the specific abilities afforded to each role. The difficulty stems from allowing a certain role to access a certain ability while simultaneously barring the other players from accessing said ability. This will be remedied by legal checks for each role. For that of the Navigator role, the checks will accept the player being able to move others when it is the Navigator's turn. The Navigator will be afforded those abilities by a boolean method which will return a value of true for the Navigator only.

A player will be able to give to another player despite the turn because receiving a card will not be restricted by the strict cycle of turns. Similarly, a player will have the ability to use a special card out of turn because it will be initiated by selecting the card from the inventory box rather than the player's hand. This will be triggered at the start and end of a turn when a boolean will be invoked to check if the player has more than five cards. If so, the player will trigger the discard action, where they have to use or discard cards until they have five or fewer cards, and they will not be able to receive additional cards without doing so.

We may have trouble communicating while coding, which can be remedied by a GitHub for our group, our Discord server, and our group's page on Canvas. This will ensure that we can discuss aspects of the project at any time.