

Game Galore

final project for team 3

Ciarra A. Adam H. Nash M. Jake V.

CSC 150 SA | 2/25/19

# Table of Contents

[Table of Contents 1](#_Toc1900392)

[Part 1: Functional Requirements 2](#_Toc1900393)

[Glossary 2](#_Toc1900394)

[Priority 2](#_Toc1900395)

[Creation, Importing, and Data Storage Behaviors 2](#_Toc1900396)

[Gameplay Behaviors 3](#_Toc1900397)

[User Interaction Behaviors 6](#_Toc1900398)

[Part 2: Objective Coverage 8](#_Toc1900399)

[Part 3: Basic Design 9](#_Toc1900400)

[Bibliography 10](#_Toc1900401)

# Part 0: Project Information

## Team Members

* Ciarra Anderson – Team Lead
* Adam Holt
* Nash Molstad
* Jacob Vella

## Project Description

This project is a large project with mini projects inside it. Our goal is to build an application that has a couple games, large and small, built in that the user can play so that boredom is a thing of the past. The user base can be a wide range from kids sixteen and up. For now, there will be four core games included in this project with the possibily of at least one more if there is time for it: Uno, Blackjack, Risk, and Battleship.

[State goals of games here]

UNO - The project is a card game intended for 2-10 players to play. Each player starts with a hand of seven cards and try to match a card in their deck with the previously played card. The goal of the game is to play every card in your hand.

Blackjack is a casino card game, involving one to seven, (including the dealer) recive 2 cards and try to get to 21 first, If you go over 21, you are out. The winner is the player that Is either still in or has the hand closest to 21. There are a couple of terms to know about blackjack. “Hit” means to ask for another card with the limit being about 3 cards depending on how many players there are. “Stand” means the player skips their turn since they are either at 21 or close to 21. “Double down” double downs the amount of chips you have and you cannot draw anymore cards. Finally we have “fold” where you just surrender because you do not like your hand.

Risk is a board game involving 2 to 6 players, where the players battle to take over a continent using cards, dice, and army figurines. More can be read about it [Here](https://en.wikipedia.org/wiki/Risk_(game)#Territories) and [Here](http://www.ultraboardgames.com/risk/game-rules.php).

Finally, Battlehship is a trial and error game where two players have to guess where the other person’s ships are located on a grid and take them down. The grid has A-J for columns and 1-10 for rows and a player choses which spot on the grid they believe may have a ship; an example being B4. There are five ships in the fleet that have different sizes for how much they take up on a grid. A Carriers take up 5 spaces, a Battleship takes up 4, a Cruiser and Submarine take up 3, and a Destroyer takes up 2. A player can place ships vertically or horizontally anywhere on the grid as long as they are not next to another ship. A player gets one Carrier, two Battleships, two Cruisers, one Submarine, and four Destroyers. Each player takes one shot per turn unless they hit a target. They they have to make another hit until they miss again. If a player loses their fleet of ships, they lose.

# Part 1: Functional Requirements

## Glossary

#### Uno

* “Reverse Card” – Changes the player turn rotation form clockwise to counter-clockwise or vice-versa.
* “Skip Card” – Skips the next player in the turn rotation, going to the following plyer.
* “Wild Card” – Allows the player who places this card to choose any suit to become the active suit.
* “Draw Card” – Makes the next player in the turn rotation draw the specified number of cards and add them to their hand.

#### Blackjack

* “Dealer” – this is the person that will be dealing with the cards or aka the console and code or I could implement an AI player to play as the dealer
* “Player”- These are obviously needed to play the game, I may limit it to players to 1 or 2 but im not sure yet.
* “Game win/Lose conditions” – The dealer or aka the console should be able to realize when a a player has lost or they have lost.
* “Chips/Money” – you do not need to have these in a blackjack game, but this could be an option and would allow the double down method to be used if I want to.
* “Player Profiles”- These are not needed but can be able to hold player chips amounts or player wins and loses. The function could even do both depending on what I would like to implement.

#### Risk

* **Map –** The map contains every country, and every territory inside each country, and serves as the reference to what territories can interact with each other
* **Territory –** The 42 territories are small sections of land inside each country that can be either
  + **unoccupied** – containing to units or
  + **occupied –** containing the units of a player
* **Units –** the base component in the game. Units are used to claim territories, attack, and defend. Units can be
  + **inactive**, where they are not currently on the boardor
  + **active** where they are currently on the board
* **Cards –** risk uses a non-standard 42 card deck, with one card for every territory, with a picture of each territory, and each unit on the card
* **Dice –** risk uses 6 sided dice**,** 3 white dice for the attacker, 2 black dice for the defender, 1 red die for to determine player order.

#### Battle-Ship

* “Ship” – a parent class that contains what’s needed to make a ship.
* “Player” – a class that contains what’s need to make a player
* “BoardSpots” – a parent class that contains information about what goes on in a spot
* “Board” – a child class of BoardSpots that contains what’s need to make a board
* “Driver” – starts the application
* “ConsoleIO” – a class that contains methods for converting user input into storable data
* “GamePlay” – stores whats going on during a game and game logic
* “Display” – displays board and other information

## Requirements List

### Priority

Each requirement has a priority level of [1], [2], or [3].

1. Must-have functionality critical to the problem solution.
2. Highly desirable feature that should be included.
3. Optional requirements that will be completed if time allows.

#### Uno

1. A deck consisting of 108 cards [1]
   1. The deck consists of four suits:
      1. Red, Green, Blue, and Yellow
   2. Each suit consists of the following cards:
      1. One 0
      2. Two 1s, 2s, 3s, 4s, 5s, 6s, 7s, 8s, and 9s
      3. Two Draw Two
      4. Two Skip
      5. Two Reverse
   3. The deck also has four Wild cards and four Wild Draw Four cards.
2. There are between 2-10 players [1].
   1. A player has a name. [2]
   2. A player has a hand that consists of cards. [1]
3. At the start of a game:
   1. The deck is shuffled [2]
   2. Each player is dealt 7 cards [1]
   3. The remaining cards are used as the draw pile [1]
      1. The card on top of the draw pile is used as the first card in the discard pile [1]
         1. If the first card is an action card, the action is carried out on the first player. [2]
   4. The turn rotation begins clockwise [2]
   5. The first player is randomly chosen [2]
4. Gameplay:
   1. During a turn, the player must match one of their cards from their hand with the top card in the discard pile. This can be done in any one of the following ways: [1]
      1. The numbers match
      2. The colors match
      3. The symbols match (Both Reverse, Skip, etc.)
      4. Place a wild card
   2. If a player does not have a card in their hand the matches the top card in the discard pile, then that player must continue to draw cards from the draw pile until they draw a matching card. [2]
   3. Once the draw pile has been completely depleted, all cards but the top are taken from discard pile, reshuffled and used as the new draw pile. [2]
   4. Players take turns matching cards to the top discard pile card until a player has one card remaining in their hand. [1]
      1. The player to reach a single card can click the “Uno!” button to keep them safe, otherwise another player can, making that player draw an additional two cards. [2]
         1. If no one presses the “Uno!” button on the same turn as a player reaching one card, the game carries on as usual. [2]
         2. If a player falsely presses the “Uno!” button, they are penalized by having their turn skipped and having to draw a card. [3]
   5. The game ends once a player has played all of the cards in their hand [1]
   6. Action cards:
      1. Reverse – Changes the turn rotation of the players from clockwise to counter-clockwise or vise-versa. It can only be played on cards of the same color or other Reverse cards. If a Reverse card is the first card, the first player takes their turn as usual, however the turn rotation is reversed. [2]
      2. Skip – Skips the turn of the next player in the turn rotation. It can only be played on cards of the same color or other Skip cards. If a Skip card is the first card, the first player is skipped. [2]
      3. Draw Two – The next player in the turn rotation must draw two cards from the draw pile and forfeit their turn, unless they play a Draw Two card as well. The draw amount will continue to stack until a player does not have a Draw Two card, this player will have to draw the total of all the consecutive Draw Two cards played (4 Draw Two cards played in a row means the player that broke the streak has to draw 8 cards). It can only be played on cards of the same color or other Draw Two cards. If a Draw Two card is the first card, the first player must draw two cards and forfeit their unless they can play a Draw Two card of their own. [2]
      4. Wild – The wild card can be placed on any card. The player gets to choose which color they want the Wild card to represent. The next player in the turn rotation must play a card of that color or another Wild card. Wild cards can be played anytime during a player’s turn, regardless of the other cards in their hand. If a Wild card is the first card, the first player gets to choose the color they wish to play. [2]
      5. Wild Draw Four – This card acts just like a Wild card except that it also makes the next player in the turn rotation draw 4 cards from the draw pile. Wild Draw Four cards can be stacked similarly to how Draw Two cards can, with the total of all the cards being draw by the player who cannot continue the streak. If a Wild Draw Four card is the first card, the first player gets to choose the color they wish to play and does not have to draw four cards. [2]
      6. Add additional Action cards with unique features. [3]
5. Once a player wins:
   1. Display a message stating so with the name of the winner. [2]
   2. Prompt the user if they wish to play again, if so with or without the same players. [2]

#### Blackjack

1. Dealer – Must be functional since that is the AI that the player plays against or known as the house

1.1 – Must be able to draw cards and shuffle them

1.2 – Must be able to Ask the player what they would like to do after they have recvived their first cards

2.1 – Must be able to calculate if the AI is going to win or what it needs to win

1.3 – Must be able to know all the different combinations for what wins beat out the others2

1.2.1 Player – Must be functional so the player can even play the game

1.2.2 – Must be able to draw cards and make choices based on their hand

1.2.3 – Must be able Win the game or lose the game

1.2.4 – Must be able to use the multiple options and have them fuction properly

3. Chips/Money – Is an optional requirement since the money and chips are not needed for the player to be able to get the full experience of blackjack

2.1- have a bank of chips for the correct amount or have a certain amount of money that is the max amount and when a player reaches 0 dollars or no more chips, then the player would lose

2.2 calcualte how much money is given compared to chips so it can be consitant.

2.3 – Have again, player profiles to represent players names, wins, loses and the amount of chips the player had since the last time they saved.

#### Risk

**Creation, Importing, and Data Storage Behaviors**

1. A user can create a game [1]
   1. The system stores the following required information about a game:
      1. Board state [1]
         1. Map [1]
            1. Countries [1]

Territories [1]

Player’s Units [1]

Adjacent Territories [2]

Unit bonus for owning the country [2]

* + - 1. Cards [1]
         1. Drawn cards [1]

Unit [1]

Territory [2]

* + - * 1. Undrawn cards [1]

Unit [1]

Territory [2]

* + - 1. Dice [1]
         1. Attack Dice [1]

Outcome [1]

* + - * 1. Defense Dice [1]

Outcome [1]

* + 1. Players [1]
       1. Name [2]
       2. Inactive Units [1]
       3. Active Units [1]
       4. Cards [1]
          1. Unit [1]
          2. Territory [2]
       5. Units
          1. Color
       6. Owned Territories
    2. Player order [1]
    3. Active player [1]

1. A user can load an existing game [1]
   1. A user can browse for their file [1]
      1. The system validates a user’s file selection [1]
         1. The system warns a user if the file doesn’t contain the game [1]
         2. The system warns a user if the file is corrupted or has errors [1]
         3. The system uses a file chooser [3]
2. A user can save the current game [1]
   1. A user can save the game at any point [2]
   2. A user can browse for where to save the file [1]
      1. The system uses a file chooser to save the file [3]
         1. The system warns a user if the file is the wrong type [1]

**Gameplay Behaviors**

* 1. At the beginning of a game:
     1. The system assigns each user a designated player to use [1]
        1. The system prompts the user for how many players to add [1]
           1. The system allows each user to choose a player [3]

The system allows each user to make a custom player [3]

* + - 1. There may be no more than 6 players [1]
      2. Each player is given an amount of **inactive units** [1]
         1. The amount of inactive units is determined by the number of players
         2. If there are 2 players, each player gets 40 units [2]
         3. If there are 3 players, each player gets 35 units [2]
         4. If there are 4 players, each player gets 30 units [2]
         5. If there are 5 players, each player gets 25 units [2]
         6. If there are 6 players, each player get 20 units [2]
    1. Each player rolls a die [1]
       1. The system makes a special, temporary,order **die** [2]
       2. Each player rolls the orderdie [2]
          1. Their outcome is recorded [2]
          2. The outcomes are compared [2]
    2. The player with the highest outcome goes first [2]
       1. The system marks the player with the highest outcome as the activeplayer [2]
          1. In the event of a tie, the system determines the activeplayer alphabetically, by name [2]

In the event of both players having the same name, the activeplayer is determined pseudo-randomly [2]

* + 1. The system then determines playerorder alphabetically by player name [2]
       1. In the event of both players having the same name, playerorder will be determined pseudo-randomly [2]
    2. The activeplayer selects a territory to place an inactive unit in. [1]
       1. The system checks that the territory is **unoccupied** [1]
       2. Once an inactive unit is placed, it becomes marked as an **active unit** [1] [1]
    3. Upon placing a unit, the system removes that active player marker from that player
    4. The system marks the next player in play order as the active player [1]
    5. The sequence of selecting a territory and placing a unit is repeated unit each player runs out of inactive units [1]
       1. If the end of the play order is reached, the first player in play order is marked as the active player and play order resumes iterating through the players [1]
       2. When there are no unoccupied territories remaining, the active player may place their unit in an occupied territory [1]
          1. The active player must already have a unit in the occupied territory [1]
  1. For the remainder of the game:
     1. The active player’s turn now consists of:
        1. Receiving inactive units
           1. The amount of inactive units received are based on the following:

Territories owned

Each territory owned is worth 1 unit [1]

Countries owned

The player receives a bonus for owning every territory in a country [2]

Cards redeemed

The player receives extra units for redeeming cards [1]

Redeeming cards yields a units equal to five plus one unit for every set of cards redeemed previously. [1]

After the 5th set has been redeemed, each additional set is worth 5 additional armies, instead of 1. [1]

Cards redeemed must be:

All the same unit [1]

One of each unit [1]

The player may earn an additional 2 units

A redeemed card must have a territory on it that the player controls. [2]

The player must redeem their cards if they have 5 or more cards [2]

The system will add the units earned to the players inactive units [1]

* + - 1. Placing inactive units
         1. The player can add inactive units to any territory they control, in any combination

The system will check that the player owns the territory they control [1]

The system will not allow the player to place more units in a territory than they have inactive units for [1]

* + - * 1. The player must place all inactive units before proceeding to the next stages of their turn [1]
      1. Attacking
         1. The player may choose to attack

The player will be prompted if they want to attack [1]

The player selects a territory they control [1]

The territory must enough units currently occupying it [1]

The system will check that there are at least 2 units in the territory [1]

The player selects the territory they want to attack [1]

The player must not own that territory [1]

It must be adjacent to the territory they are attacking from [1]

The player declares how many units are attacking [1]

The user will be prompted for how many units to attack with [1]

The system will validate that:

The player does not try to attack with more than 3 units [1]

The player does not try to attack with less than one unit [1]

There is always at least one unit remaining in the attacking territory [1]

The player that owns the territory being attacked declares how many units are defending [1]

The system checks to see which player owns the territory [1]

The user is prompted for how many units to defend with [1]

The system validates that:

The player does not try to defend with more than 2 units [1]

The player does not try to defend with less than 1 unit [1]

Both players roll dice

The system rolls attack dice equivalent to the number of units attacking for the attacking player [1]

The results are stored [1]

The system rolls attack dice equivalent to the number of units defending player [1]

The results are stored [1]

The system evaluates the dice rolls to see who wins the battle [1]

Dice are compared, highest to lowest [1]

If the Highest attack die is larger than the highest defending die, the defender loses a unit [1]

If the Highest defending die is larger, or equal to, the highest attacking die, the attacker loses a unit [1]

If the second highest attack die is larger than the second highest defending die, the defender loses a unit [1]

If the second highest defending die is larger, or equal to, the second highest attacking die, the attacker loses a unit. [1]

This process is repeated until:

The attacker decides to stop attacking [2]

After each attack, the attacking player is prompted if they want to keep attacking [1]

The system validates that the response [1]

The attacker has less than 2 units in the attacking territory [1]

The system checks that there are always at least 2 units in the attacking territory [1]

If there are less than 2 units in the attacking territory, attacking stops [1]

The defender has no units in the attacked territory [1]

The attacker now owns the attacked territory [1]

The defender loses the attacked territory [1]

The attacker may move as many units into the attacked territory as were attacking (up to three, there must still be one unit in the attacking territory) [1]

The attacking player is prompted for how many units to move [1]

The system validates that the attacking player does not try to move more than 3 units [1]

The system validates that the attacking player does not try to move less than 1 unit [1]

The player may attack as many times as they like in a round [1]

The player is prompted if they want to keep attacking [1]

* + - * 1. The player does not have to attack [1]
        2. The player may receive a card [1]

The player must attack and capture a territory [1]

* + - 1. Moving units
         1. The player may move any number of units [1]

The player is prompted if they want to move units [1]

It must be from one territory they control to another territory they control [1]

The player is prompted for what territory to move units from [1]

The system validates that the player owns that territory [1]

The player is prompted for what territory to move units to [1]

The system validates that the player owns that territory [1]

The system validates that the two territories are adjacent [1]

The player is prompted for how many units to move [1]

The system validates that there will still be at least 1 unit in the territory units are being moved from after the move. [1]

They may only do this for one territory. [1]

* 1. The game ends when one player owns every territory on the map. [1]
     1. The system checks to see how many territories each player owns [1]
        1. If a player doesn’t own any territories, they are removed from the play order [1]
           1. If there is only one player in play order, they win [1]

The users are shown: [1]

The winning player [1]

Time it took to win [3]

Number of units owned by each player [3]

Number of units killed by each player [3]

**User Interaction Behaviors**

1. The user will interact with the application through GUI’s
   1. There will be multiple GUI’s
      1. All GUI’s will use a common view interface
      2. The GUI will change based off what the user is doing.
         1. Main menu GUI for when the user first starts the game
         2. Main GUI for when the user(s) are playing the game
         3. End Game GUI for when the user(s) finish the game
2. The Main Menu GUI
   1. This GUI is displayed with the App is first launched
      1. This GUI displays options to:
         1. Exit the application
         2. Start a new game (See Creation, Importing, and Data Storage Behaviors 1)
         3. Load an existing game (See Creation, Importing and Data Storage Behaviors 2)
3. The main game GUI
   1. This GUI is used for the majority of the game
   2. This GUI is Split into multiple areas
      1. An application control area
         1. Located on the top of the window
         2. Allows the user to save the game
         3. Allows the user to exit the game
         4. Allows the user to return to the main menu
      2. An additional information area (AIA)
         1. Located at the bottom of the window
         2. Displays context-based information
            1. Context is derived from the user’s interaction in the main area
      3. A main area
         1. This section is where most of the information will be presented
            1. This section will display a map

The map will contain all the countries

The countries will contain all the territories

The territories will display their owners

The territories will display a unit count

* + - * 1. This section will also display the players

Each player will be selectable

Upon selection additional information will appear in the additional information area

* + - * 1. This section will also display these options:

Attack

When a player is attacking or defending, it will display:

The options available to the players:

How many units to attack/defend with

How many available units the attacker/defender has

The outcome of the battle (See Gameplay Behaviors 1.2.1.3.1.6.3)

The option for the attacker to stop attacking

Free move

When free moving, it will highlight territories that the active player controls

End turn

See (Gameplay Behaviors 1.2.1)

* + - 1. Most of the user’s interaction will take place here as well
         1. Countries and territories will be selectable from the map

The selected item will display additional information in the additional information area (AIA)

For countries the AIA will display:

Territory count

Unit bonus for owning the country

A percentage breakdown based on which players own territories in the country

Fun facts about the country [3]

A picture of the country [2]

A background based on the country [3]

Each country will have a unique background [3]

For territories the AIA will display:

Unit count

Who owns the territory

If the current player owns the territory, it will display options to attack

Fun facts about the territory [3]

A picture of the country [2]

A background based on the territory [3]

Each territory will have a unique background [3]

For players the AIA will display:

Amount of units owned [2]

Amount of territories owned [2]

Expected amount of units to gain at the beginning of their next turn [3]

How close the player is to winning [2]

How powerful the player is (ranking) [3]

Calculation based off how many units the player controls

Ranked based off other players in the current game

If the active player is attacking (See User Interaction Behaviors 1.1.3.1.3.1)

The AIA displays an ‘attack from’ button

The territory must be eligible to attack (see Gameplay behaviors 1.2.1.3.1.2.1 and 1.2.1.3.1.2.1.1)

Territories that can be attacked will display an ‘attack’ button

1. The End-Game GUI
   1. This GUI displays the stats of the game (See Gameplay Behaviors 1.3.1.1.1.1)
   2. This GUI will also display options to:
      1. Exit the application
      2. Start a new game (See Creation, Importing, and Data Storage Behaviors 1)
      3. Load an existing game (See Creation, Importing and Data Storage Behaviors 2)

#### BattleShip

1. All ships needed for a fleet for each player can be created [1]
   1. The following infomatino is what’s needed to make a ship:
      1. Ship Length [1]
      2. Ship Name [1]
2. The board needs to be created to have ships be put on it [1]
   1. A board has a set length and width of 11x11 [1]
   2. Rows must be numbers 1-10 [2]
   3. Columns must be letter A-J [2]
   4. Another board must be blank to hold firing spots [1]
      1. Take info from other player’s board and hides the ships [1]
3. A board spot must know these things about itself
   1. Is spot occupying a ship? [1]
      1. Has a ship been hit in my spot? [1]
      2. Is the spots around me being occupied by another ship? [2]
   2. Has my spot been fired at? [1]
4. A player can be created and used to store information:
   1. Player name [2]
   2. Stores board information [1]
5. These game mechanics must be implemented:
   1. Set up beginning of the game [1]
      1. Ask for name of both players [2]
      2. Have players set up their boards by placing their ships on the grid [1]
   2. Player can select a spot and have that fire a missle [1]
      1. If player hits, continue letting them make hits until they miss again [1]
      2. If player misses the first time, switch to other player [1]
   3. Players will lose ships if all parts of the ship are hit [1]
      1. Space around the dead ship gets treated like they’ve been targeted and cannot be selected [2]
   4. Player wins when all ships are sunk [1]
      1. Check how many ships are left on the field [1]
         1. If the number is 0, player loses [1]
6. Timer keeps track of how long it takes to beat a person [3]
   1. Can be added to a high score [3]

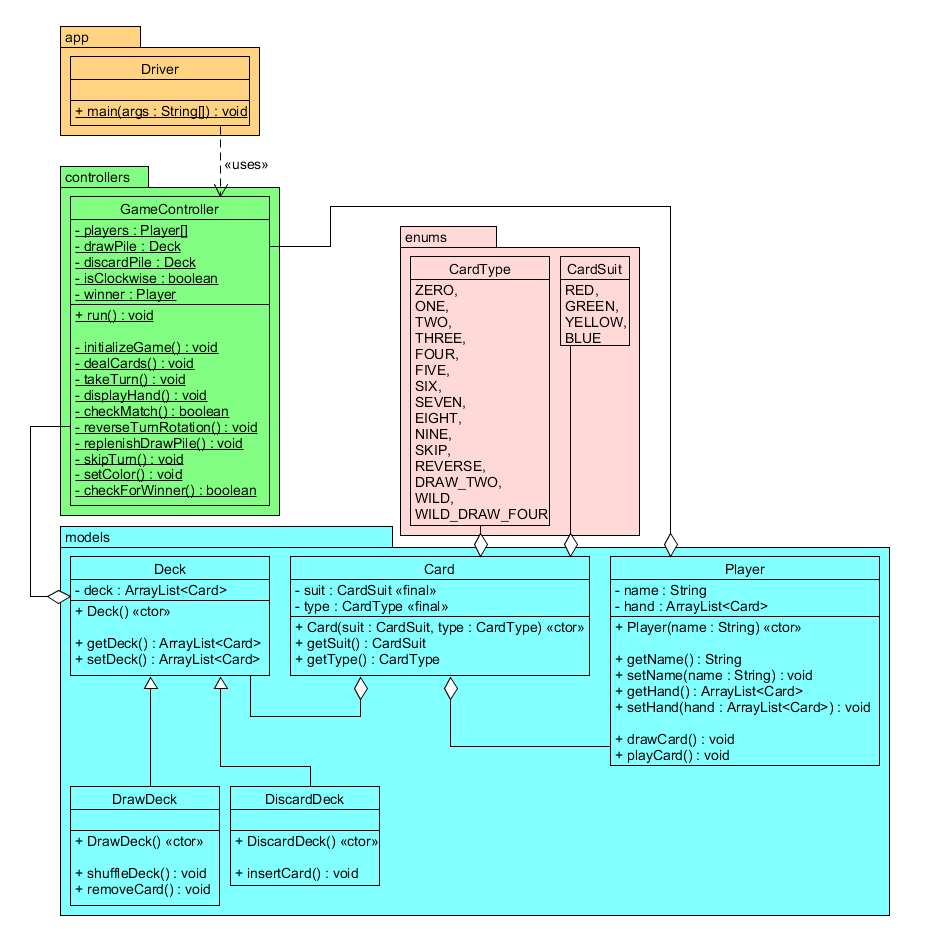
# Part 2: Objective Coverage

This project will cover the following course objectives:

* In-depth topics of Object-Oriented Design and Programming
  + This project will be using an MCV pattern, the usage of design patterns can be considered an in-depth topic.
* Write programs using control structures: arrays, loops, and classes.
  + This project will be using:
    - HashMap for the countries
    - HashMap or Arrays for the territories
    - Array List for owned territories
    - Classes to make Country, Territory, Board, Map, Player, Dice, Card, and View objects
* Build classes with variables and methods.
  + This project will be using 8-12 classes, each containing 4-12 methods.
* Use class composition, inheritance, polymorphism, and interfaces and know the advantages of each in program design
  + This project will be using composition to store data,
  + Polymorphism and inheritance to display information to the user
* Use even handling and exception handling techniques
  + This program will throw and catch all its own exceptions
* Develop and understand object-oriented programs
  + This project is made as an effort to display understanding in object-oriented programs, using Java.

# Part 3: Basic Design

#### Uno



#### Blackjack

#### 

#### Risk

\*\*The UML is subject to change… A LOT\*\*

UML doesn’t include any information on the viewer, as that information will be added when appropriate knowledge is acquired. The controllers may significantly change with the addition of a viewer as the implementation of a viewer is still unknown and said viewer will play a large role in the design, and usage, of the project.



#### BattleShip



# Part 4: Project Plan

#### Uno

* 2/25 – Have the documentation and planning completed.
* 3/1 – Have all of the methods fully pseudo-coded and beginning to actually implement code.
* 3/4 – Have all of the methods fully implemented. Learn how to integrate code with GUI and create graphics for the cards.
* 3/8 – Have the code and GUI interacting correctly and visually appealing graphics.

#### Blackjack

2/28 – stub out and finish UML

3/1 – fully implement models

3/4 - fully implement controllers

3/8 fully implement GUI’s

#### Risk

2/28 – stub out UML

3/1 – fully-learn GUI’s, implement basic Menu GUI

3/2 – fully implement Models

3/4 – fully implement game logic

3/6 – fully implement GUI’s to control everything

3/7 – implement file persistence

3/8 – polish app, include other modules.

#### Battleship

Phase 1: Week 1

Task 1: Get ships made and have the variables properly storeable

Task 2: Get the player made a have variables properly storeable

Task 3: Have a board be able to be created and connected to the players stored

Task 4: Be able to display board and ships with GUI

Phase 2: Week 2

Task 1: Set up beginning of game (setting ships onto the board)

Task 2: Add firing mechanics

Task 3: Add boundaries between ships so that one isn’t next to the other

Task 4: Put in win logic

Phase 3: Week 3

Task 1: Finish any other pieces that aren’t working for functionality

Task 2: If there is time left, add a timer

Task 3: If there is more time left, add a scoreboard

# Bibliography