

SYSC3110 Design Project

Group Members: Nikolas Paterson, Adi El-Sammak, Erik Iuhas

RISK: Global Domination

Milestone 1

Design Process:

When designing RISK: Global Domination, we took multiple variables into account and broke it into a few main class groups. The first group is classes containing game variables and references to other objects; these are Game, Player, Territory. The next group of classes are the ones who allow the user to interact with the objects; these classes are Command, CommandParser, CommandWord and Command. Finally, the last group of classes are responsible for handling the game logic behind a Player's attack and generating dice rolls; these classes include Dice and GameEvent.

Data Structure HashMap:

Risk is a board game that includes 42 Territories, each with a differing number of neighbors. We determined that a HashMap would be the best data structure for storing territory information using the territory name (String) as the key and the Territory object as the value. We believe that this was the most effective way to obtain the Territory object when receiving the user's commands. HashMaps are utilized in multiple classes; Player uses it to store its territories and allows it to find neighbors to attack. The Game class also stores the "WorldMap" which is the entire map, including all territories. The GameSetup class creates the "WorldMap" using a CSV file that contains the Territory neighbours and Continent. Lastly, the Territory class contains a HashMap which stores their neighbouring territories.

GameEvent & Dice Class:

The GameEvent class acts as the controller of the Risk game since it is dependent on user input. Thus meaning, that once a Player in Risk decides to "REINFORCE", "ATTACK" or "FORTIFY" the GameEvent class is responsible for handling the outcomes of an event by updating the Player or Territory class based on the action called by the Player. The GameEvent's attack() method uses the Dice class to set up the corresponding rolls for the attacker and defender. The Dice class's primary responsibility is to generate rolls for the attacker and defender and assist the GameEvent class by handling an attack's outcome. The Dice class generates the attacker's rolls based on the number of dice that the attacking Player wants to roll with. Additionally, the Dice class generates the defender's rolls where the number of the dice the defender rolls with is decided based on how many troops the defending territory has.

Milestone 2

Designing the GUI:

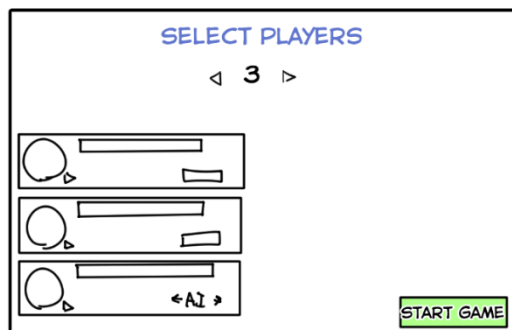
Before creating the GUI, we wanted to create visual prototypes for how the game will look when built in java. We used Clip Studio Paint to break the game into three main phase windows.

Starting Screen



This became the base of the StartUpView; it allowed us to create a screen to inform the user of the game they will be playing and provide rules if they do not yet know how to play the game.

Character Select Screen



This became the base of PlayerSelectView; it was created so that players can select their name and icon. Plans were also made for milestone three, where the player can select whether they want to include an AI player in the game. After the player would select players, it would load the game.

Game Screen



This became the base of GameView; the GUI aimed to create an easy to use and interactive game experience for the players. The key design point for this GUI would be that color represents each player, so when buttons are displayed on the map, it indicates who owns which territory. Lastly, another note worthy GUI component would be the status bar, which shows the game's state and which players turn it currently is.

Action GUI Design:

While the display of the game is necessary, the actions move the game forward. To keep the GUI from becoming cluttered, we opted to create temporary pop-up windows for each player action. Using this method, we only needed to provide the territory objects to the Pop-up windows in initialization.

Below we will go in depth with the current design and purpose of each pop-up window:

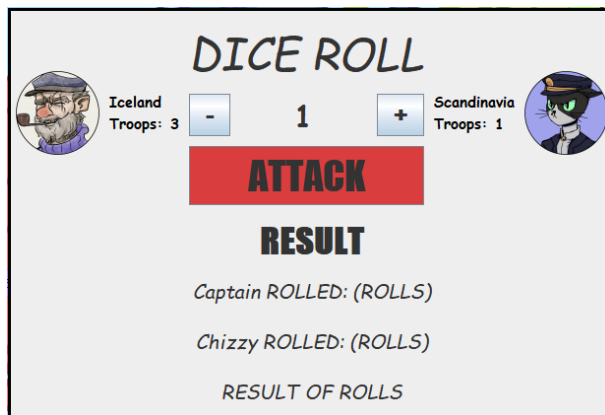
Reinforce



The Reinforce popup interface features a light gray background. At the top left is a circular avatar of a man with a beard and a blue cap. To its right, the word "REINFORCE" is written in a large, bold, black font. Below the avatar, the word "Egypt" is displayed in a smaller, italicized font. To the right of "Egypt" is a green button with the word "DEPLOY" in bold black letters. Below "Egypt" are three buttons: a blue button with a minus sign, a white button with the number "0", and a blue button with a plus sign.

For Reinforce popup, we provide the territory of the player who occupies the territory. They can change the value of troops with the plus and minus buttons. The number of troops is limited to the max deployable troops, which is available in player's model. After deploying the territories are updated.

Attack



The Attack popup interface has a light gray background. At the top, "DICE ROLL" is written in a large, bold, black font. Below this, on the left, is a circular avatar of a man with a beard and a blue cap, with "Iceland Troops: 3" written below it. To the right of the avatar is a blue button with a minus sign, followed by the number "1", and then a blue button with a plus sign. To the right of these buttons is a circular avatar of a cat with a blue cap, with "Scandinavia Troops: 1" written below it. Below the dice roll controls is a red button with the word "ATTACK" in bold black letters. Below the "ATTACK" button is the word "RESULT" in bold black letters. Below "RESULT" are three lines of text: "Captain ROLLED: (ROLLS)", "Chizzy ROLLED: (ROLLS)", and "RESULT OF ROLLS".

Attack popup requires two territories, one occupied by the attacker and one defending neighbor. In this screen, the player can roll between 1 to 3 dice if they have more than three troops, and the max dice roll changes depending on the troop count of the attacker. So, if the attacker has two troops, the dice roll value is locked at 1. After an attack is made, the result screen displays each player's rolls, and the attack outcome and the territory view are updated to consider the loss of troops.

Fortify



The Fortify popup interface has a light gray background. At the top, "FORTIFY" is written in a large, bold, black font. Below this, on the left, is the word "Greenland" in a large, italicized font, and below it is "Troops: 3". To the right of "Greenland" is a blue button with a minus sign, followed by the number "0", and then a blue button with a plus sign. To the right of these buttons is a green button with the word "DEPLOY" in bold black letters. To the right of the "DEPLOY" button is the word "Iceland" in a large, italicized font, and below it is "Troops: 3".

Fortify popup requires two territories, both territories must be neighbors and occupied by the current player. The player can move troops from the left territory to the right territory; troops value is capped at one less troop than the total troop count in the left territory. After deploying, the troops move to the right territory, and the territories are updated.

Unit Testing:

For unit testing, we wanted to test the game's logic with which the player interacts, which mainly pertains to the GameEvent class. Using the game event class, we created scenarios for all three main moves; Reinforce, Attack, and Fortify. We created every possible scenario to ensure the GameEvent is working as designed. Essentially for all actions, we attempted to see if the neighbor rules were enforced, make sure only players that own territories can interact with it, and testing negative and large values.

Changes Made to UML:

For the UML, minimal changes were made to the previously existing classes, and we were mainly adding the GUI elements to interact with model classes. We had to create views for a start-up screen, player select, and game view. Along with each view, we created a subsequent controller for button actions.

When it came to the game view design, we opted to make each model have their own view. For territories, we went with JButtons to display each territory directly on the map and used a JPanel for each player view, which displayed their troop count and color.

Finally, the other significant notable addition would be the popup screens described in the Action GUI Design. Each Popup had its controller, which interacted with the GameEvent class and handled the game logic. Each of these popups are made when interacting with territory buttons and is called from the GameController class.