Risk Board Game

Team 7 - 2023/04/20

Contributors:
Richard Costa, Hunter Masur, Aidan Mazany, Evan Sellers

Table of Contents

Introduction

This document serves as a guide and overview of our Risk Board Game software for both users and future developers. It is a Java project (meaning your machine will need to be able to run Java to use it) that began as a class project for the Rose-Hulman Software Quality Assurance course and was developed further in the Software Construction and Evolution course. During the course of the development for Risk, we took a pre-existing game, refactored it, and added new features. For those unfamiliar with the board game, Risk, a good first place to quickly gain an understanding of the game is to read the basic rules of the World Domination game mode set out by Hasbro.

For users who simply wish to enjoy the game, we only recommend viewing the “Guides” section of this document, where you’ll receive in-depth installation instructions, gameplay tutorials, and explanations of important gameplay rules and terminology. For developers, we recommend reviewing this entire document prior to contributing to the repository. This includes an overview of our requirements, which includes our features and use cases for the game, a discussion of our current design, including UML diagrams and important design decisions, and finally an overview of our testing results and methodology.

For users interested in downloading and playing the game or developers interested in contributing further, the following is a link to our GitHub repo:
https://github.com/SellersEvan/375-Project-Risk

User Guide/Manual/Tutorial

Manual

Risk is an easy-to-learn and play strategy game for 2 to 6 players, in which each player tries to conquer the world by invading territories and eliminating their opponents. The game is played on a board divided into continents and then further divided into territories, where players take turns moving armies, attacking and defending territories, and trying to dominate the world.

At the beginning of the game, players are given a select number of armies, which they can place on territories they claim. Players then take turns reinforcing their own territories and attacking neighboring enemy territories, in the hopes of dominating the world.

To attack an enemy territory, a player must first have a neighboring territory. They will then move their armies to the territory they are attacking. Players will roll dice to determine the outcome of the battle. If the attacker wins the battle, they gain control of the enemy's territory and can move armies into it, occupying it.

Players will receive reinforcements at the beginning of each turn, based on the number of territories they occupy. Players will continue to attack each other and claim territory until there is only one dominant player left standing, who occupies all the territory on the map.

Overall, the key to winning the game is to make strategic decisions about placing, moving, and attacking.

Game Rules (There may be slight differences in rules that will be outlined at a high level below)

If you run into issues or bugs while using or developing for this game, you can report those issues in the repository's "Issues" section.

Terminology & Player Actions

Risk is an extremely easy game to pick up for newcomers, however, there is some important terminology. If you have never played Risk before ensure you familiarize yourself with the terms below before you start to read the instructions or begin the game. This ensures everyone playing the game is on the same page.

Term	**Definition**
Continent	A group of connected territories. If a player controls all territories in a specific continent they receive a reinforcement troop bonus, which they can place at the beginning of their turn.
Territory	A region with a specific continent on the game board, which a single player can control at a time. Players can use this location to attack neighboring territories if they occupy it.
Claim/Conquer/ Occupy	At the beginning of the game, players take turns selecting territories they will begin with. Players can attack a territory and if they win, can claim it.
Reinforcement	Additional troops that players can place on territories they occupy at the beginning of their turn or after conquering new territory.
Attack	When a player moves troops from one of their territories to a neighboring enemy territory, in the hopes of conquering it.
Defend	When a player uses troops stationed in a territory to resist an attack from an enemy opponent.
Combat/Battle	The process that occurs when one player attacks another player's territory. A set of dice are rolled to determine the results of the battle.
Card	A game card represents a specific type of troop like infantry, cavalry, or artillery unit, which can be used for additional reinforcement troops.
Trade-In	During the car trading phase, players can trade in a set of 3 cards of the same type, or a set of 3 with 1 of each unique type. Doing so will take your cards and give you troops that can be placed in the reinforcement phase.
Elimination	When a player loses all their territories they are removed from the game.
Victory	When a player conquers all the territories or (if selected) completes their secret mission, they win the game.
Secret Mission	An optional alternative way to win the game. If enabled, players will be given a secret task that is only known to them. If they complete their task at any point in the game they can reveal it and the game will end.

Board

The basic layout of the game board is overviewed in Figure 1.

In the top left-hand corner (Figure 2), you can see the current player's name and flag displayed, along with the number of armies available for placing and the phase the player is currently in. Each territory on the board is either labeled "Unclaimed" or has a name/color associated with the player who currently occupies the territory. The territory also displays the number of armies deployed to the specific territory.

When in the Attacking Phase you can preview your attack (Figure 4), by first clicking the territory you would like to send troops from and then clicking the territory you would like to attack. You can similarly do this in the Fortifying Phase.

At any time you can click "View Cards" at the top right-hand corner of the screen to view your current cards (Figure 3). These cards can be obtained through the conquering of territories and traded in for additional armies in the Trade-In Card Phase.

When you have completed the actions for the current phase you can click the "End Phase" button, which will take you to the next phase or player.

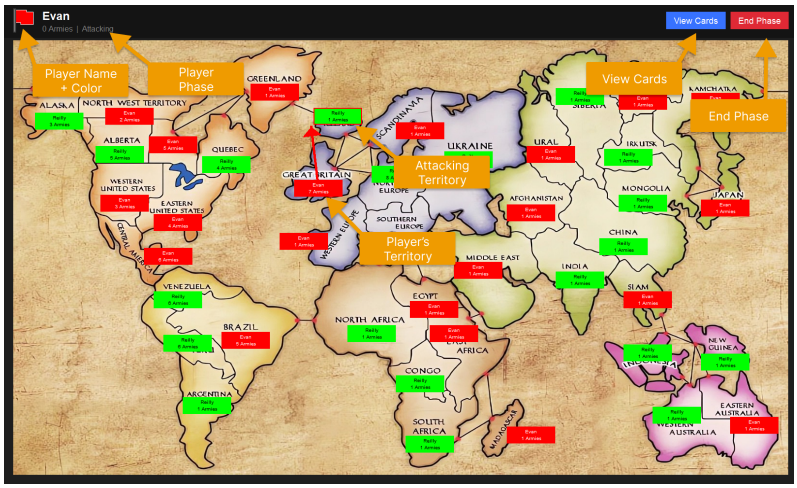


Figure 1 - Risk Application at High-level

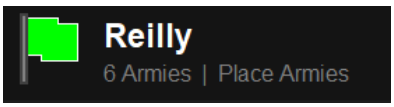


Figure 2 - Player Details

Figure 3 - Player Cards

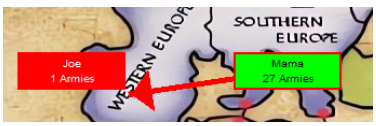


Figure 4 - Attacking

Phases

Risk consists of several different phases. These phases are repeated by each player until one player has achieved victory or their secret mission. The phases are as follows:

> **Trade-In Phase** - At the beginning of each turn players are allowed to trade the cards in to receive more armies. To turn in cards you must need either: a set of 3 cards of the same type, or a set of 3 cards of different unique types. When these cards are traded in, you will receive more armies that can be placed in the reinforcement phase.
>
> **Reinforcement Phase** - After trading in cards, each player will receive a specific number of armies (based on the number of territories they occupy) they can then place the armies in a territory they already occupy.
>
> **Attack Phase** - After placing reinforcement troops, a player may choose to attack an adjacent enemy territory, to a territory they control. Both players (attacker and defender) involved in the battle will then choose how many dice to roll, to determine the outcome of the battle. If the attacker wins the battle, they can then select how many troops to move to the territory they just claimed.
>
> **Fortification Phase** - After completing all the desired attacks, a player may choose to move some of their armies from one territory they occupy to another territory they occupy. This fortification of a specific territory can help protect territories from future attacks.

After a player completes all the phases in their turn, the next player will begin with the reinforcement phase, and so on, until a player has won the game.

Objective

When playing Risk the primary way to win the game is to conquer every territory on the board. However, when you are setting up the game you will be given the option to have secret missions. If enabled, each player will be given a secret mission at the beginning of the game. If the player completes their specific secret mission they will automatically win. For example, let's say a player receives a secret mission to conquer Africa. If the player conquers (occupies all the territories in the continent) Africa at any point, they automatically win the game.

Setup

In the User Tutorial section below you will see these steps in more detail. Setting up Risk is super easy and the game will walk you through each step in the process. First, users will start by selecting the languages, the world map, and the number of players. Each user will enter their name and select their color. The board will start completely empty, with none of the territories claimed. The first player to go will then select the territory they would like to claim. Then the next player will claim the territory they would like. This will continue until all the territories have been claimed. Finally, players, in the same order they claimed territories, will place their initial armies on the territories they occupy. Now the game can get started. Good luck commander!

Supported Languages

Risk currently supports only a select few languages. We hope to expand our language support in the future, but for the time being, feel free to submit a support ticket to let us know what language you would like to see support added to.

- English (EN)
- French (FR)

Tutorial

Risk is a super easy game to play, but if you are new to the game of Risk you might take some time to check out the User Manual. Each user is associated with a name and player, which the player selects in the Setup of the game. The goal of each player is to conquer the world (or complete a secret mission, see Objective section).

Step 1

Launch the Game

Start by ensuring you have downloaded the newest version of the game, see Installation Guide for more information. Navigate to the directory you have stored the .jar file for the game, and double-click the .jar file, see Figure 5. If you're having an issue launching the application see Troubleshooting.

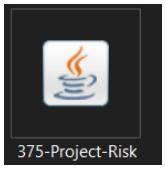


Figure 5 - JAR File

Step 2

Select Language

After launching the game you will be asked to select the language you would like to use, see Supported Languages for more information. Click the dropdown menu, select your desired language, then press "OK" to finalize your selection.

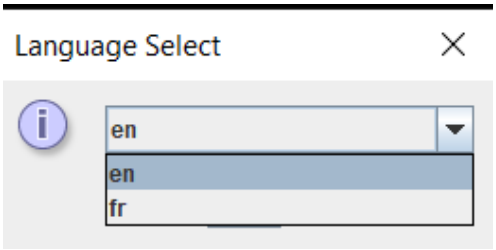


Figure 6 - Select Language Dialog

Step 3

Select World

Next you will select the world you would like to play on. The default selection world is “Earth” (which is the standard Risk board), but there are other possibilities in which you can play. Different worlds help add variety to gameplay and require you to develop a new strategy. Click the dropdown menu, select your desired world, then press “OK” to finalize your selection.

Figure 7 - Select World Dialog

Step 4

Setup Players

Next you will be prompted about the number of players who would like to play the game. The game supports anywhere from 2-6 players. Click the dropdown menu, select the desired number of players, then press “OK” to finalize your selection. See Figure 8.

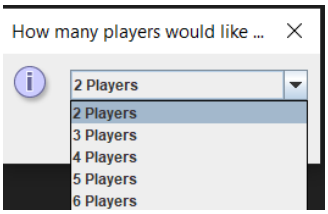


Figure 8 - Select Player Count

Each player will then be prompted to choose their name and color, starting with the first player. After the first player has entered their information pass the computer to the next player and continue, until all the players have entered their information. See Figure 9 and Figure 10.

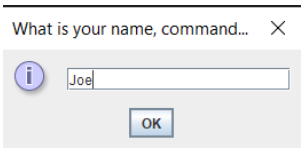


Figure 9 - Select Player Name

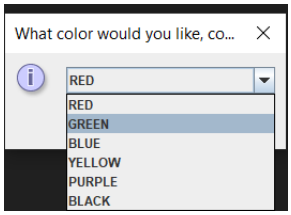


Figure 10 - Select Player Color

Step 5

Secret Mission Mode

Normally the main objective of the game is to conquer all territories in the world, but you can enable an optional secret mission mode, see Objective. Click the dropdown and select your desired choice, see Figure 11. If “YES” is selected, secret mission mode will be enabled.

The game will then prompt you to pass the computer to a specific player to give them their secret mission, see Figure 12. When the player has gotten the computer, they will click “OK”, and then read their secret mission privately. See Figure 13. After the player has finished reading, click “OK” and pass the computer onto the appropriate player. This process will continue until all the players have received their secret mission.

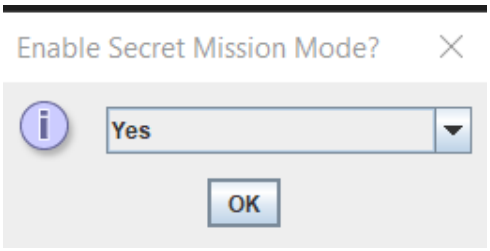


Figure 11 - Enable Secret Mission Mode

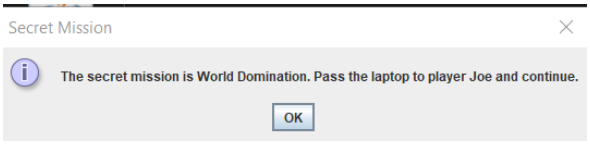


Figure 12 - Pass Laptop to Next Player

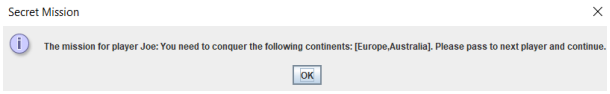


Figure 13 - Display Secret Mission

Step 6

Claim Initial Territory

The game will indicate which player will be starting. The player may click on any territory they would like to claim. The board will then update to reflect this, see Figure 14.

Now the next player will select their territory to claim. This process will continue until all the territories have been claimed.

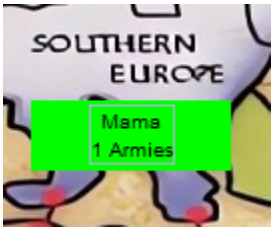


Figure 14 - Claimed Territory by “Mama”

Step 7

Place Initial Armies

Once all the territories have been claimed, players will then have the opportunity to reinforce the territories they occupy.

Players have a specific amount of armies they can place, which can be seen in the top-left corner, see Board. Start by clicking the first territory you would like to place amies at and enter the number of armies you would like to place, see Figure 15. Repeat this procedure until you have no more armies left. Then click “End Phase” and allow the next player to complete this process.

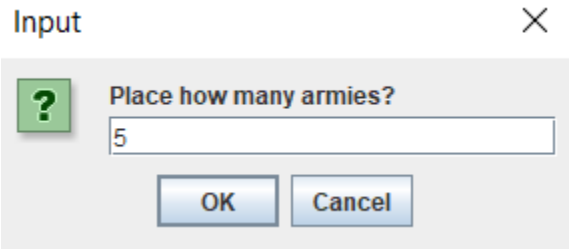


Figure 15 - Prompt of the number of armies to place

Step 8

Begin the Game

Are you ready to start commander? Because whether you are ready or not, the conquest must begin. Good luck commander!

Step 9

Trade-In Card Phase

This phase happens first for every player's turn, *except this phase is skipped for the first round*. Cards are acquired by the player when they conquer a territory. During this phase players are allowed to trade the cards in to receive more armies. To turn in cards you must need either:

- A set of 3 cards of the same type
- A set of 3 cards of different unique types

The extra armies received will be able to be placed during the Placing Armies Phase. If at any time a player has more than 5 cards they must discard any extras. After completing all transactions click “End Phase”.

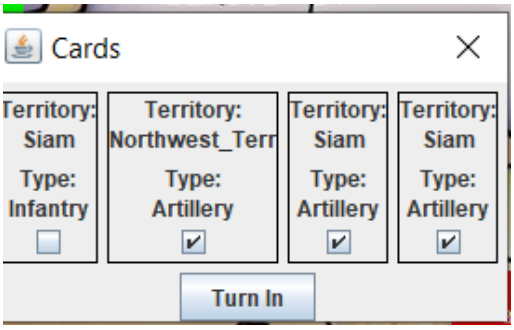


Figure 16 - Select cards to “Turn In”

Step 10

Placing Armies Phase

Before attacking each player will receive a number of additional armies based on the number of territories controlled by the player. Similar to the Place Initial Armies phase, the player can place their armies within their own territories. Once all armies have been placed, the display will update to show the user. Click “End Phase”.

Figure 17 - Prompt of the number of armies

Step 11

Attacking Phase

The player can now attack another enemy territory that is adjacent to their occupied territory; as long as they own at least one territory on that continent.

A player can initiate an attack by first clicking on their own territory they would like to attack from, see Figure 18. Then click the territory they would like to attack, Figure 19. Now the attack has started, and you will choose how many armies or “dice” you want to send. You can send up to 3 at a time. The defender can then select how many armies they would like to defend with, up to 2. The results of the battle will be displayed. If the attacker loses the roll, their armies die, and they can keep attacking or end their phase.

If the attacker wins the roll, they conquer the territory and can send as many armies from their attacking territory as they want (given they keep at least one in the existing territory), see Figure 20.

The player can continue attacking as many times as they can and please. Click “End Phase”, when done attacking. If at any time the player meets the Objectives, they win and the game is over.

Figure 18 - Selection of Attacker

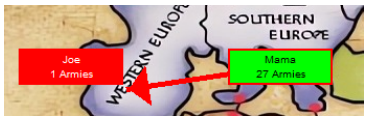


Figure 19 - Selection of Defender

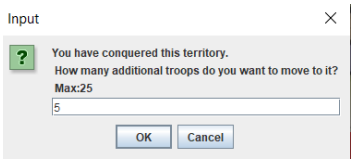


Figure 20 - Troops to Move

Figure 21 - Finalized Board

Step 12

Fortifying Phase

During the fortification phase, the player has one opportunity to move armies from one occupied territory to another adjacent occupied territory.

The player can do this by clicking a territory they occupy and then clicking another adjacent territory they occupy. The player can then select how many armies they would like to move from one to another territory, see Figure 22.

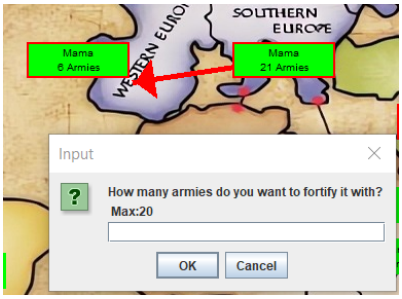


Figure 22 - Troops to Move

After fortifying (or if the player decides to skip this phase), click “End Phase” to continue to the next phase.

Step 13

End of Turn

If, during the player's turn, they conquered any territories they will receive cards. These cards can be used during their next turn in the Trade-In Card Phase.

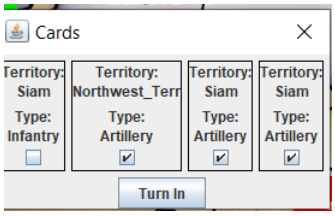


Figure 23 - Cards Received

Step 14

Winning the Game

You will repeat steps 9-13 for each player until the win Objectives have been achieved. After one player either claims all the territories or accomplished their secret mission (if secret mission mode is enabled), the winning player will be notified and the game will end.

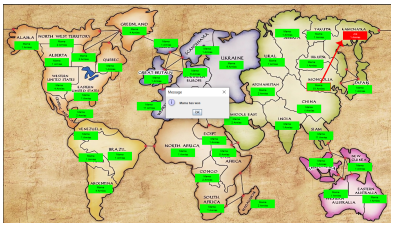


Figure 24 - Winning Board

Installation Guide

Installation of the Risk board game is extremely easy, this guide will walk through the installation process for both players and developers. Deployment process of the application to players is outlined in the Development & Release Process. Configuration of the application is limited because of the size of the application.

Players

If you have any issues with the installation process see Troubleshooting or submit a support ticket to support.

1. Verify Java 8 is installed on your computer (install it, if not)
2. Navigate to the Risk Release Page
3. Download the latest version's associated JAR file (What is a JAR?)
4. Move the JAR file to the desired location (We suggest the Desktop for convenience)
5. Launch the Game
6. Dominate the World!

Developers

1. Verify Java JDK 18 or greater is installed, if not install it (Check your JDK Version?)
2. Verify IntelliJ by JetBrains is installed. Other IDEs are acceptable, we just recommend IntelliJ and our whole team uses IntelliJ.
3. Clone a copy of the repository using a git client. We use GitHub (git) as our version control system. We recommend using GitHub Desktop. (How to clone a repository with GitHub Desktop?)
4. Import the repository that was just cloned into IntellJ as a Gradle project. (How to open Gradle projects in IntelliJ?)
5. Your IDE should look similar to Figure 25. If you have any issues see Troubleshooting or IntelliJ Documentation.
6. Verify Gradle (version 7.1 or greater) is set up in your IDE. Intellij should have it set up by default, otherwise, instructions can be found here.
7. Run the gradle build action to install all project dependencies.
8. Familiarize yourself with the Maintenance Guide.

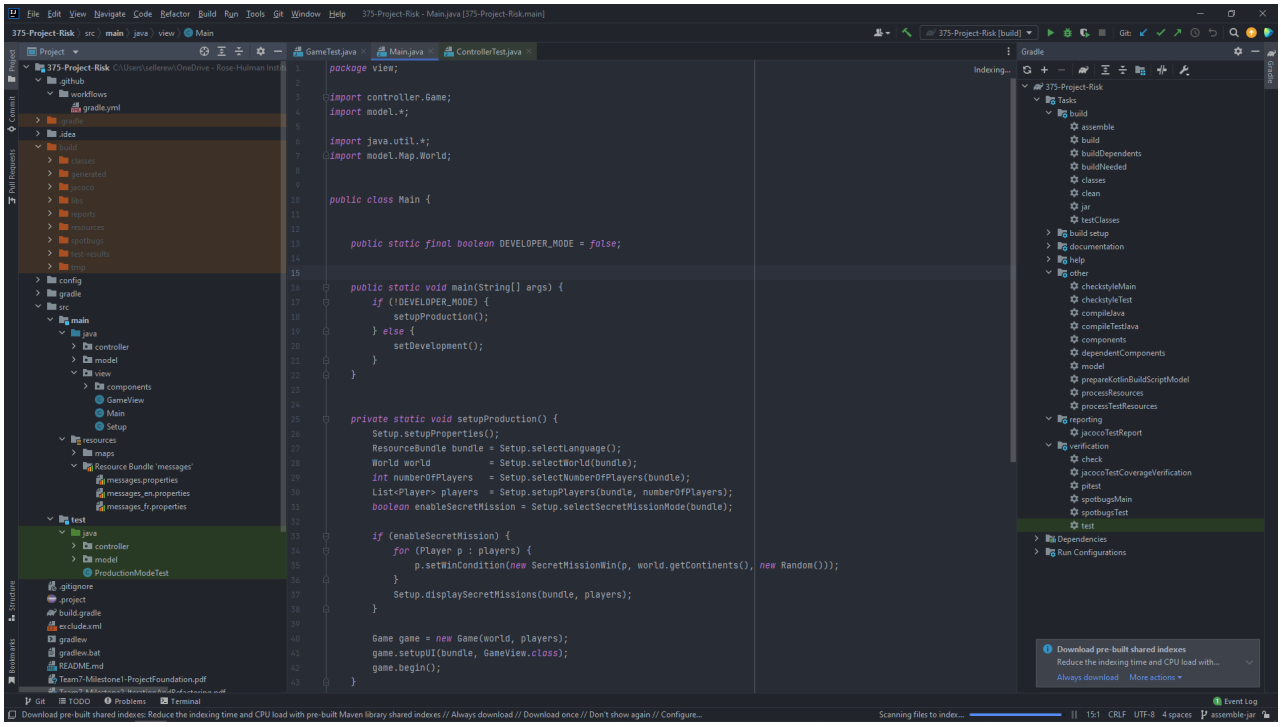


Figure 25 - IntelliJ Setup with Risk Repository and Gradle

Development/Maintenance Guide

Revision History

The revision history should be kept up-to-date and reflect the GitHub Release History.

Version	Date	Description
1.0.0	3/16/23	The original build of the system was developed by Andrew Sullivan, Aidan Mazany, Dixon Ramey, and Jadon Brutcher for a CSSE376 class.
2.0.0	3/30/23	Added ability for players to select name and color. Added ability to support adding additional world files in the future. Added wild cards to the game, which allow you to choose if it's infantry or cavalry. *Backend code cleanup: Improved model, view, and controller pattern for the game. Removed bad code smells in the code base.*
3.0.0	4/20/23	Added Secret Mission Mode - which allows players to select this particular game mode at the beginning, they will be handed a mission - like dominating two specific continents - if they complete this goal they automatically win. New visuals. More information is now provided to the user on the game screen and an indication is provided of territories selected with an attack line. *Backend code is cleaned up: Improved coupling between the user interface and game logic. Removed bad code smells in testing.*
4.0.0	5/11/23	Added additional integration and unit testing for the game logic class.

Management Practices

The development process of this application used the Scrum agile framework along with test-driven development to deliver continuous improvement to our customers. Scrum allows us to break down the work into separate sprints, in our case about 2 weeks long. Since software and factors that influence development are constantly changing, we can easily readjust sprints as needed.

Ensuring the team is always informed about the progress of any sprint is extremely important, because of this we employed Trello, which allows us to share the project kanban board with the whole team. The work was broken into the standard Scrum swimlanes - backlog, up next, development, and completion. With an additional swimlane for “grab bag” quality of life improvements backlogs could be done at any time.

Each ticket is broken up into two separate categories either a milestone for the ticket to be completed on or a Grab bag - these are not connected to a required feature and can be grabbed to complete at any time. From there the tasks are labeled into one (or multiple depending on scale) categories:

- **Feature** - Planned Feature that adds additional features to the game.
- **Bad Code Smell** - These are bad code smells that must be fixed to implement.
- **Refactor Change** - These are changes that are not bad code smells, are not good.
- **QoL Improvements** - These are Quality of Life Improvements and small items like renaming unclear variable names or moving a method.
- **Administrative** - These are tasks that are not code-based and are admin actions.

Development & Release Process

GitHub will be used as both the release platform and the version control system. The repository can be accessed publicly here and the developer has to be given permission to perform some of these steps like creating a pull request, approving a pull request, merging, and creating a release.

1. **Sprint Preparation** - the team will come together, set a timeline for the sprint, and select the features/tasks they would like to be included in this sprint. Tickets will then be created and assigned to the developer within the project management software. Read more about this in the Management Practices section.
2. **Development of Features** - the developer will then start work on their assigned tickets. Developers will update the status of each ticket in the project management software to reflect the task's current state; either up-next, development, or completed.
3. **Create Pull Request** - after a ticket has been completed by a developer, they will commit their code to the remote GitHub server and submit a pull request. How to create a pull request on GitHub?
4. **Automatic Github Testing** - when a pull request is created, GitHub will automatically run a set of continuous integration tasks. These tasks include all of the Gradle verification tasks, specifically related to quality assurance such as JUnit tests, PiTests, and SpotBug. Read more about this in the Quality Assurance Tools and Gradle Actions section. All the tasks must pass to approve the pull request.
5. **Approval and Merging** - the master branch has a branch restriction, so before the pull request can be merged into the master repository the code must be approved by at least one other team member. They merge the branch into the master branch.
6. **Compile to JAR** - after all merges are complete and the game is ready for deployment, compile the game into a single jar file, using the Gradle build action.
7. **Create Release** - create a new release in the GitHub repository. Label the version of the release with an incremental version from the last release, while following Semantic Versioning. Add the compiled JAR file to the release and publish the release. Learn more about GitHub Releases?
8. **Update Documentation** - update the documentation to reflect any changes made in the newest release.

Libraries

We implement libraries (using Maven and Maven Repositories) for a variety of reasons but mainly to ensure the code delivered to our customers is to the highest standards. Learn about the specific libraries we use for Quality Assurance. But we also implement libraries to enable our team to access high-quality tooling, so our team is not reinventing the wheel. For example, instead of building a YAML importer in-house, we opted to use the highly refined SnakeYaml library. Below you can see a list of every library we use (managed by Gradle), the version, and what the library is used for.

Library	Version	Description
SnakeYaml	2.0.0	Import YAML serialized data into java
EasyMock	4.2.0	Mock objects, not under test for unit testing of specific class
JUnit	5.8.1	Testing framework for JVM
SpotBug	5.0.6	Identify common bug patterns in system logic
Checkstyle	*ANY*	Linter to enforce specific code style standards
Jacoco	0.8.8	Coverage reporting to ensure tests cover system logic
PiTest	1.5.1	Mutation testing to ensure logic is justified by test cases

Quality Assurance Tools

We employ a multitude of quality assurance tools to ensure the code produced by our team is top-notch and each commit meets the level of quality required by our customers. This includes checking for common error patterns, coverage tests, and standard code syntax across the codebase. All these tools are integrated into our Testing Suite. Additionally, when a developer creates a pull request, GitHub will automatically run (SpotBug + CheckStyle) using the Gradle test bench on the cloud, see Testing Automation with GitHub. These tests must pass to allow developers to merge.

Library	Version	Description
SpotBug	5.0.6	Identify common bug patterns in system logic
Checkstyle	*ANY*	Linter to enforce specific code style standards
Jacoco	0.8.8	Coverage reporting to ensure tests cover system logic
PiTest	1.5.1	Mutation testing to ensure logic is justified by test cases

Gradle Actions

We use Gradle to standardize build automation procedures and manage software dependencies on the project.

Gradle should be automatically set up when using IntelliJ or Eclipse, if not you can learn more about setting up Gradle. If you have not used Gradle before, ensure that you are to familiarize yourself with Gradle.

In Figure 26 you can see what the Gradle action menu looks like in IntelliJ. All the Gradle actions are listed by category, you can double-click an action to execute it. Below are listed some of the most important actions, including running all tests, compiling a single fat JAR file, and running Quality Assurance Tools.

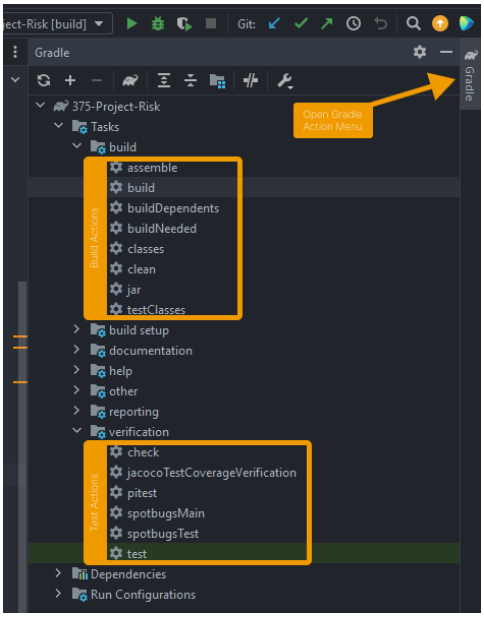


Figure 26 - IntelliJ Gradle Menu

Name	Action	Output
Assemble JAR File	*build/jar*	*build/libs/*.jar*
All Build Actions	*build/build*	N/A
All Verifications Actions	*verification/check*	N/A
JUnit Tests	*verification/test*	*build/reports/tests/test/index.html*
PiTest Mutation Testing	*verification/PiTest*	*build/reports/pitest/index.html*
Spot Bug on Main	*verification/SpotBugMain*	Terminal Output
Spot Bug on Tests	*verification/SpotBugTests*	Terminal Output
Checkstyle Checks	*reporting/checkstyleMain*	*build/reports/checkstyle/index.html*
Jacoco Coverage	*reporting/jacocoTestReport*	*build/reports/jacoco/test/html/index.html*

Bypass Game Setup During Development (Developer Mode)

When developing the game, developers have the option to launch the game in developer mode. This mode enables developers to quickly run acceptance and manual testing to test new features during developement. It does this by by-passing regular game setup. The game will normally ask the player to select the map and set up players - by enabling development mode these steps will be skipped, stepping the developer straight into the game. Developer mode will also enable territories that are unclaimed to have the territory name, to help with debugging world files. This option can be enabled by changing the DEVELOPER_MODE flag in the class view/Main. Setting this flag to false will bypass the regular player setup. *Application testing will not pass while development mode is enabled (due to a JUnit test that validates development mode is disabled).*

Adding World File

World files are different maps that users can select from when playing. These world files have different board layouts, including backgrounds, continents, and territories. Switching these worlds can add a lot of variety to the game and change how players strategize and play the game.

1. To add a new world file start by creating a new background image in JPEG format. We suggest an aspect ratio of 16:9. Add this image to the ./resources/map/ directory, with the name of the file being the name of the world, and ".jpg" being the file extension.
2. Then create a configuration file with the same name, but a ".yaml" file extension. This config file will store all the important information about the world, like continents and territories (Learn to Setup World File). This config file will be again located in the ./resources/map/ directory.
3. After adding these two resources we will update the world index. Edit the ./resources/map/index text file, and add a new line with the name of the world file. This should be the same name as the image and config file.

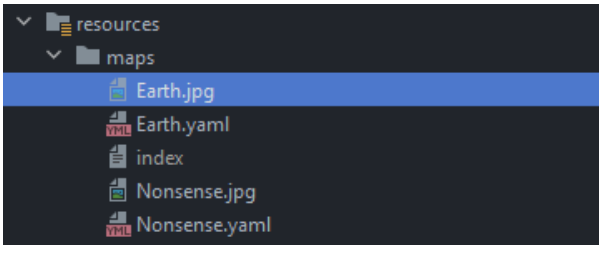


Figure 27 - Resources Directory with Maps (Earth & Nonsense)

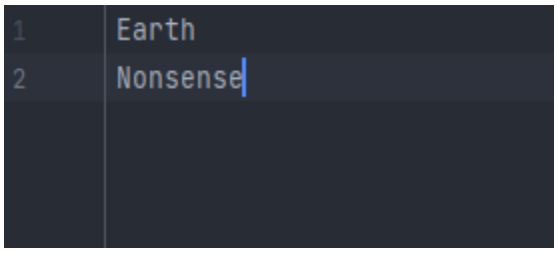


Figure 28 - "index" text file, for list of worlds

World File Format

The world file, is the data that represents the world file map. It is stored in YAML format and parsed by the World Manager Class, using Snake YAML.

The file consists of a top-level property continents key, below this is a list of continent objects. These continent objects consist of a name, and bonus - provided to the player if they capture all the territories in that continent and territories. The territories key contains a list of territories. The territories object consists of a name, coordinates - which is an array of two floats that represents percent from the top left relative to the world image, and neighbors. The neighbor property is a list of strings, each string should correspond to the string name of another territory it neighbors.

In Figure 29 you can see an example of what this file should look like.

```yaml
continents:
    - name: North America
      bonus: 5
      territories:
          - name: Alaska
            coordinates: [ 0.02813, 0.16667 ]
            neighbors:
                - Northwest_Terr
          - name: Northwest_Terr
            coordinates: [ 0.11875, 0.15556 ]
            neighbors:
                - Alaska
                - Greenland
          - name: Greenland
            coordinates: [ 0.30625, 0.1 ]
            neighbors:
                - Northwest_Terr
    - name: South America
      bonus: 2
      territories:
          - name: Venezuela
            coordinates: [ 0.18125, 0.57778 ]
            neighbors:
                - Peru
          - name: Peru
            coordinates: [ 0.19375, 0.67778 ]
            neighbors:
                - Venezuela
```

Figure 29 - Example World Format File

Key	Type	Description
continents	Array of …	Array of continents
- name	String	Name of the continent
- bonus	Integer	Bonus if capture all territories in the continent
- territories	Array of …	Array of territories
- name	String (No Spaces)	Name of territory
- coordinates	Array of floats [0, 1)	Coordinates of territory position, the percentage from top left, relative to map image
- neighbors	Array of strings (territories.name)	Territory neighbors on the map, mapped by a string related to other territory names

Adding Language File

This application uses the standard Java Resource Bundle to support different languages (supported languages). The resource bundle files are named “messages” and located in the ./resources directory. Each resource follows the syntax conventions of the properties file format. The application falls back to English using the “messages.properties” resource bundle.

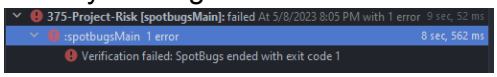
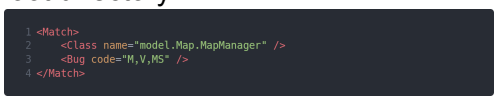
To create a new language add a new file in the ./resources directory, named “messages_<code>.properties”. The code represents the language code (ISO639 - 2 digit) of the language the resource bundle will provide. Ensure that each key property has a translation for that specific bundle, otherwise, the application will fall back to English.

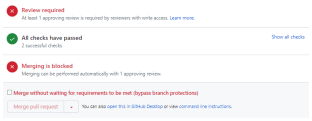
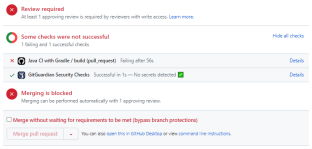
After creating the properties file, you will add the language code to the **SUPPORTED_LANGUAGES** array, which is the list of supported languages located in ./view/Setup class. After this is completed, the new language has been successfully added.

Troubleshooting

Below we have listed some of the issues that we either encountered or expect someone to encounter during the development or usage of the application. If you have any issue the troubleshooting is not able to solve, feel free to create a support ticket.

Symptom Observed	Possible Issue	Resolution
Throws Error - Unable to find world image	Image file for the world that was selected is not present in the resources directory.	Add a world image in the map resources directory for the associated world file following the Adding World File procedures.
Throws Error - Unable to find world config	Config file for the world that was selected is not present in the resource directory.	Add a world config in the map resources directory for the associated world file following the Adding World File and World File Format procedures.
Throws Error - Unable to load World	Config file for the world that was selected is not set up properly.	Ensure the config file for the specific world that was selected is following the correct World File Format.
Added World Map is not appearing as an Option	World map identification code that has not been added to the index listing in the resource directory.	Add a new entry with the name of the new world map to the “index” file in the maps directory of the resources. These procedures are prescribed in more detail in the Adding World File section.
The system claims two territories are not adjacent but they are, when attacking or fortifying	Territories are not labeled properly in the world configuration file.	Enable Development Mode, this enables you to see the names of territories on unclaimed territories and identify issues with any labeled territories. Ensure you turn development mode off when you're done. After you have identified the territories that are causing the issue, update them, while following the correct World File Format.
Added Language is not appearing as an Option	Language code has not been added to the list of supported languages.	Add the languages code to the array list SUPPORTED_LANGUAGES, in the ./view/Setup class, as prescribed by the Adding Language File procedures.

Throw Error - Can't find bundle for base name	There is a missing properties file for the language selected.	Add the properties file for the languages selected by following the Adding Language File procedures.
Throw Error - Can't find a resource for the bundle	There is a missing property key from a specific language that was selected.	The error report will tell you the specific property that is missing, go to the specific language bundle the property was missing from and add it. Check the other bundles to verify what the property was supposed to represent.
Spot Bug Main/Test Errors	There was a possible bug detected by the Spot Bug Quality Assurance Tool.	Start by clicking on the list of errors.  It should give a bug classification code (for example M V MS), class name, method name, and description of the possible bug. Attempt to correct the issue first, however, if the issue is a false positive we will disable the error. Add the following code to the “exclude.xml” file in the root directory…  With the appropriate class name, method name, and bug classification code.
Throws Error - java.io.IOException: Unable to delete directory “...binary…”, during testing	Attempting to run multiple Gradle actions at a time.	Start by executing the build/clean Gradle action. Then re-run each of the actions you wanted to execute one at a time, waiting for the previous to finish execution before starting the next action.
Skipping User Setup Stage	Developer Mode is enabled	Disable developer mode by following the procedures in Development Mode.
Throws Error - duplicate but no duplicate handling strategy has been set	Duplicated resources in the build directory.	Start by executing the build/clean Gradle action. Then re-run each of the actions you wanted to execute one at a time.

GitHub Preventing Merge 	GitHub branch protection requires at least one other person to approve the pull request.	Have another person on the team review your pull request and if everything looks good approve the pull request. To learn more about this process see Development & Release Process.
GitHub Preventing Merge 	GitHub branch protection requires all testing and verification for Quality Assurance Tools and Unit Tests to pass to merge.	Starting by running the tests and verification on your own local machine (Gradle Actions). See which tests or assurance tools are failing. Correct those errors and commit the updates.
Gradle Actions are not visible in IntelliJ	Gradle is not set up properly	Review Gradle documentation for IntelliJ
Gradle Actions are not visible in Eclipse	Gradle is not set up properly	Review Gradle documentation for Eclipse
Unable to execute JAR File	Issue with Environment or Dependencies	Review JAR execution help guide

Planned Features

During the previous development cycles there were additional features we would have liked to add, but did not have enough time to implement. Here are just a few of the features that we hope to implement in future iterations of the application.

Feature	**Description**
Additional Languages	Currently, we only support English and French. We hope to support some more additional popular languages like Mandarin, Hindi, Spanish, Arabic, and Russian.
Additional Integration Testing	Because of the massive overhaul the main game class went through, we did not get enough time to thoroughly develop integration test cases and we would like to develop additional tests.
Upgraded Interface for View Cards	During the development cycles we got the chance to make major visual improvements to the user interface, however, we did not get the time to improve the view cards user interface. In future iterations, we would like to improve the card list menu and make it a lot more visually appealing and easier to understand, and group cards together.
Additional Cards	We added a wild card option on top of the basic infantry, cavalry, and artillery card units. But we would have liked to add additional cards that are rarer but count as multiple cards or are wild cards for not only the card type but all the territory.
Additional Secret Mission Types	We added secret missions that require players to capture a specific set of continents to win the game but we would have liked to also add additional secret missions like occupying at least one territory on each continent or occupying a select couple of specific territories.
Additional Worlds	During the development process we added the ability to add additional worlds but designing the worlds is an extremely time-consuming process. Because of this we only had time to create one world map. In future iterations, we would like to add additional worlds.

Software Requirements Specification

This section outlines the cumulative features decided upon from both academic courses of development on this codebase. Seeing as this game was designed for learning purposes, the only stakeholders who played a role in deciding what would be done were the developers in both courses and their respective professors.

Features

Below are the notable features completed in the current version of the game, followed by a number of use cases showcasing how users interact with specific features in our game, our Use Cases should reflect each of these features. Note that newly added features are in bold.

1. Ability to support anywhere between 2 and 6 players
2. Ability to detect when a player wins the game:
    a. Control of all territories (World Domination)
    b. Captures their secret target continents (Secret Mission Mode)
3. Ability to detect when a player has lost the game (Player loses all territories)
4. Ability to place armies on territories during the start of game
5. Ability to attack adjacent territories (Attack Phase)
6. Ability to move armies to adjacent territories (Fortify Phase)
7. Ability to draw a card after capturing territory
8. **Ability to draw a wild card that represents any card type after capturing territory**
9. Ability to trade in cards on a turn, to receive additional armies
10. Ability to view the state of the map
11. Ability to gain and place armies at the start of your turn
12. Ability to set the number of armies given at the start of game depending on number of players
13. Ability to take cards from another player when you capture their final territory
14. **Ability to have a choice of multiple, mechanically different maps**
    a. Earth World Map
    b. Nonsense World
15. **Ability to choose your player's color and name**
16. **Ability to enable secret mission mode (Win by capturing target continents)**
17. **Ability to view the preview for attack and fortification actions**
18. **Ability to view a summary of current player**

Use Cases

Use cases describe the intended interactions between the user and the game without getting into how the game will actually implement these interactions and run things behind the scenes. They serve several useful purposes in the design process. First off, in the case of a project with a client, they can be used to confirm that our understanding of what the project should do is the same as theirs before we actually code anything. Second, it helps to break the project up into features so that we know what needs to be done. Finally, those features tell us what we need to test in Acceptance Testing to verify that the project is complete and functional.

Setup Game

Actor: Whoever happens to be using the device.
Preconditions: None.
Postconditions: Game is ready for players to place their armies.
1. User runs the executable file.
2. System prompts the user for the language they would like to use.
3. User selects a language.
4. System prompts the user for the map they would like to use.
5. User selects a map.
6. System prompts the user for how many players there will be (between 2 and 6).
7. User selects a number of players.
8. System prompts a player for their name.
9. Player enters their name.
10. System prompts that player for the color they'd like to use.
11. Player selects a color.
12. Repeat steps 8 through 11 until all players have been registered.
13. System prompts the user for whether or not they would like secret missions to be in effect.
14. User selects no.
15. System displays the board and army setup can begin.

Alternate Flow - Secret Missions (begin at step 14):

14.a User selects yes.
14.b System tells the user to pass the device to the current player. (Or people can just walk around accordingly if it's a desktop)
14.c The selected player takes the device and presses okay to see what their mission was.
14.d Repeat steps 14.b and 14.c until all players have learned their secret missions.
14.e Resume main flow at step 15.

Setup Armies

Actor: The players
Precondition: Game has been started and settings have been selected.
Postcondition: Game is ready to begin.
1. The active player selects an unclaimed territory.
2. The system highlights it in their color and adds their name to it along with a single army then indicates whose turn it is now.
3. Users take turns repeating steps 1 and 2 until all territories have been claimed.
4. The active player selects a territory that they own.
5. The system asks the user how many territories they want to place there.
6. The user enters a number of armies less than or equal to how many they currently have.
7. The system increases the number of armies on that territory and takes away that number from the user.
8. Repeat steps 4-7 until that player has placed all their armies.
9. Repeat steps 4-8 until all players have finished placing their armies.

Exception Flow - Invalid Claim (begin at step 1)

1.a The active player selects a claimed territory.
1.b The system informs the player that they need to select an unclaimed territory.
1.c The user closes the popup and resumes at step 1.

Exception Flow - Invalid Location (begin at step 4)

4.a The active player selects a territory that they don't own.
4.b The system informs the player that they need to select a territory they own.
4.c The user closes the popup and resumes at step 4.

Exception Flow - Invalid Number of Armies (begin at step 6)

6.a The user enters a number of armies that is either negative, not an integer, or greater than the number of armies they have left to place.
6.b The system informs the player that the number of armies they've attempted to place is invalid.
6.c The user closes the popup and resumes at step 4.

Trade-In Cards Phase

Actor: The active player.
Precondition: That player's turn has just started (and it's not the first turn).
Postcondition: The phase has ended and the player can start placing armies.
1. The user selects end phase.
2. The system grants that player a number of armies based on how many territories they control then indicates that it is now the placing armies phase.

Alternate Flow - Three Identical Cards to Trade In (start at step 1)

1.a The user selects View Cards (and it's not the first turn).
1.b The system displays all cards available to that player.
1.c The user selects three cards of the same type and then selects trade-in.
1.d The system takes away those cards and grants that player a number of armies based on how many sets have already been traded in.
1.e Continue at step 1.

Alternate Flow - Three Different Cards to Trade In (start at step 1)

1.a The user selects View Cards (and it's not the first turn).
1.b The system displays all cards available to that player.
1.c The user selects three cards of different types and then selects trade-in.
1.d The system takes away those cards and grants that player a number of armies based on how many sets have already been traded in.
1.e Continue at step 1.

Exception Flow - Invalid Combination (start at step 1)

1.a The user selects View Cards (and it's not the first turn).
1.b The system displays all card available to that player.
1.c The user selects cards that don't fit one of the above patterns and attempt to hand them in.
1.d The system informs the user that this isn't a valid combination.
1.e Continue at step 1.

Place Armies Phase

Actor: Active player
Precondition: Trading cards phase has ended.
Postcondition: Attacking phase can begin.
1. The user selects a territory that they control.
2. The system asks the user how many armies they wish to assign to that territory.
3. The user enters a positive number of armies less than or equal to the number at their disposal.
4. The system takes away that many armies from the user and adds them to the selected territory.
5. Repeat steps 1-4 until all armies have been allocated.
6. The user selects the end phase.
7. The system displays that it is now in the attacking phase.

Exception Flow - End Phase Prematurely (begin at step 1 (or anywhere else))

1.a The user selects End Phase before having finished allocating their armies.
1.b The system informs the user that they must place all their armies.
1.c The user closes the popup and resumes at step 1.

Exception Flow - Invalid Location (begin at step 1)

1.a The active player selects a territory that they don't own.
1.b The system informs the player that they need to select a territory they own.
1.c The user closes the popup and resumes at step 1.

Exception Flow - Invalid Number of Armies (begin at step 1)

1.a The user selects a territory they control.
1.b The system asks the user how many armies they want to add.
1.c The user enters a number of armies that is either negative, not an integer, or greater than the number of armies they have left to place.
1.d The system informs the player that the number of armies they've attempted to place is invalid.
1.e The user closes the popup and resumes at step 1.

Attack Phase

Actor: Active player
Precondition: Assign army step completed.
Postcondition: Attacks performed, ready to move to fortify phase.
1. The user selects a territory they control and then an adjacent territory that an opponent controls.
2. The system prompts the user for how many dice they want to use.
3. The user indicates how many dice they will use.
4. The system prompts the opponent who was attacked for how many dice they want to use.
5. The opponent indicates how many dice they will use.
6. The system indicates the rolls for both players and removes armies based on the rolls.
7. Return to step 1 as many times as desired.
8. The user selects the end phase.
9. The system indicates that it is now in the Fortification Phase.

Alternate Flow - End Turn Without Attacking (begin at step 1)

1.a Continue at step 8.

Alternate Flow - Attack Conquers Territory (begin at step 6)

6.a The system indicates the rolls for both players, removes armies based on the rolls, and in doing so removes the last of the defender's armies in that territory. The system prompts the user for how many armies they wish to move into that territory.
6.b The user selects how many.
6.c The system changes that territory to the user's color and moves the selected number of armies to it.
6.d Continue at step 1.

Alternate Flow - Attack Eliminates Opponent (begin at step 6)

6.a The system indicates the rolls for both players, removes armies based on the rolls, and in doing so removes the last of the defender's armies in that territory and that was the last territory controlled by that opponent. The system prompts the user for how many armies they wish to move into that territory.
6.b The user selects how many.
6.c The system changes that territory to the user's color and moves the selected number of armies to it. The opponent will no longer get turns.
6.d Continue at step 1.

Exception Flow - Attack Conquers World (begin at step 6)

6.a The system indicates the rolls for both players, removes armies based on the rolls, and in doing so removes the last of the defender's armies in that territory and that was the last territory not controlled by the player. The system informs the player that they have won.

Exception Flow - Attack Achieves Secret Mission (begin at step 6)

6.a The system indicates the rolls for both players, removes armies based on the rolls, and in doing so removes the last of the defender's armies in that territory and gains that territory gives the player control of all territories required for their secret mission. The system informs the player that they have won their secret mission.

Exception Flow - Invalid Attacker (begin at step 1)

1.a The user selects a territory that they do not own.
1.b The system informs the user that they must control the territory they attack from.
1.c Continue at step 1.

Exception Flow - Insufficient Armies (begin at step 1)

1.a The user selects a territory that they own but that only has one army in it.
1.b The system informs the user that they must have multiple armies in a territory to attack from it.
1.c Continue at step 1.

Exception Flow - Invalid Target (begin at step 1)

1.a The user selects a territory that they own and then a territory that is not adjacent to it.
1.b The system informs the user that they must target an adjacent territory.
1.c Continue at step 1.

Exception Flow - Friendly Target (begin at step 1)

1.a The user selects a territory that they own and then another territory that they own.
1.b The system informs the user that they cannot attack their own territories.
1.c Continue at step 1.

Fortification Phase

Actor: Active player
Preconditions: Attack phase has concluded.
Postconditions: user's turn is over, it's now the next player's trade cards phase.
1. The user selects a territory that they own and then another adjacent territory that they own.
2. The system asks the user how many armies they want to move.
3. The user enters a positive integer that is less than the number of armies in the reinforcing territory minus 1.
4. The system moves that many armies between the territories.
5. The user selects the end phase.
6. The system displays whose turn it is now.

Alternate Flow - Decline to Fortify (begin at step 1 (or anywhere else))

1.a Continue at step 5.

Exception Flow - Invalid Fortifier (begin at step 1)

1.a The user selects a territory that they do not own.
1.b The system informs the user that they must control the territory they fortify from.
1.c Continue at step 1.

Exception Flow - Insufficient Armies (begin at step 1)

1.a The user selects a territory that they own but that only has one army in it.
1.b The system informs the user that they must have multiple armies in a territory to fortify it.
1.c Continue at step 1.

Exception Flow - Invalid Target (begin at step 1)

1.a The user selects a territory that they own and then a territory that is not adjacent to it.
1.b The system informs the user that they must target an adjacent territory.
1.c Continue at step 1.

Exception Flow - Enemy Target (begin at step 1)

1.a The user selects a territory that they own and then another territory that they do not own.
1.b The system informs the user that they cannot fortify other people's territories.
1.c Continue at step 1.

Exception Flow - Invalid Quantity (begin at step 3)

3.a The user enters a number of armies that is either not a positive integer or is greater than the number of armies in the reinforcing territory minus one.
3.b The system informs the user that this is not a valid number of armies to reinforce with.
3.c Continue at step 1.

Software and Architecture Design Specification

System Architecture

System Architecture describes the overall structure of the codebase and how various classes are tied together. UML diagrams provide an easy way to visualize this and keep track of everything in one place. We made major changes to the architecture during our project in order to accommodate new features as well as just make it easier to work with as can be seen below. These changes can be seen in the difference between Figure 30 and Figure 31.

Old UML

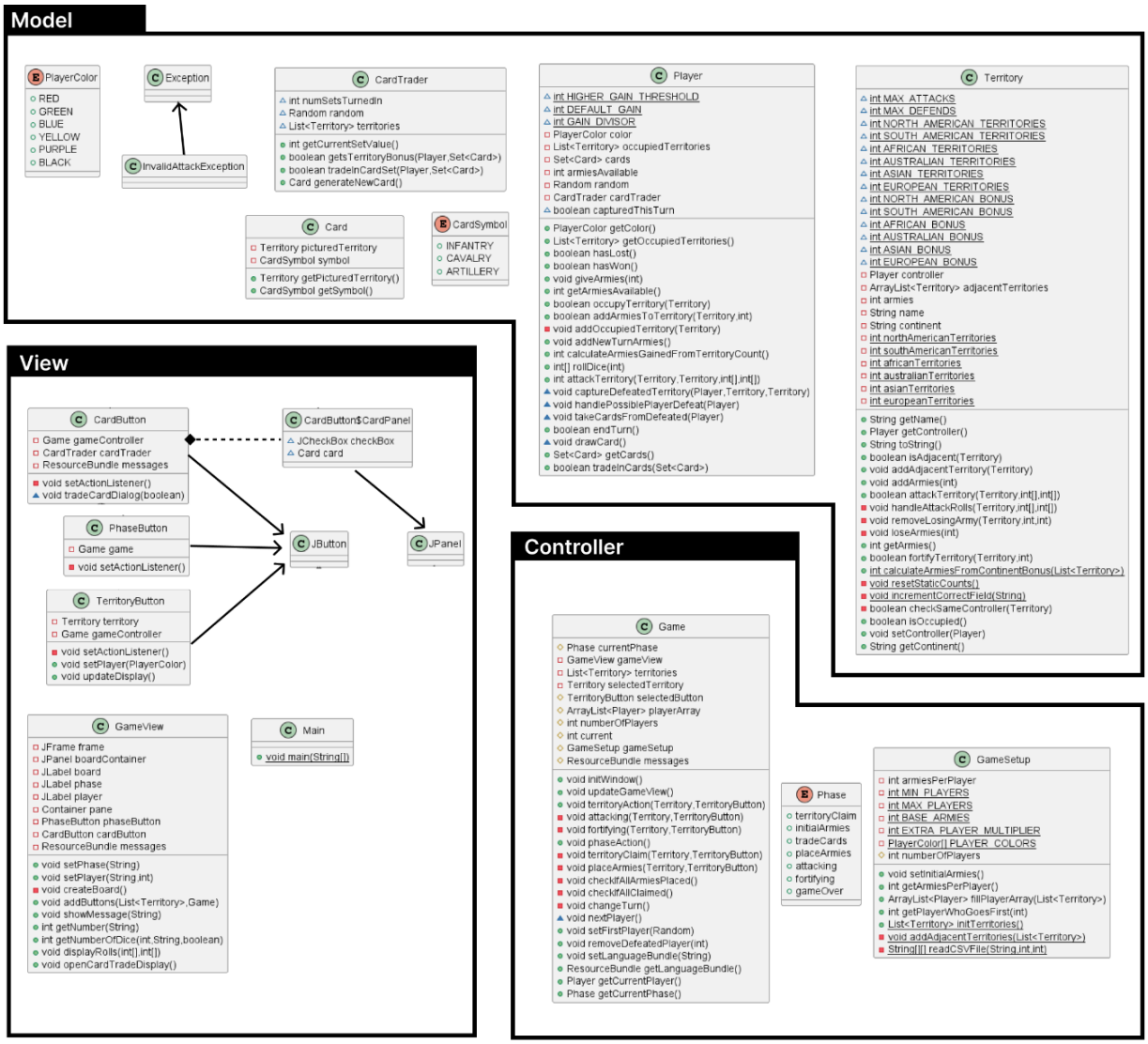


Figure 30 - UML Diagram Before Modifications

New UML

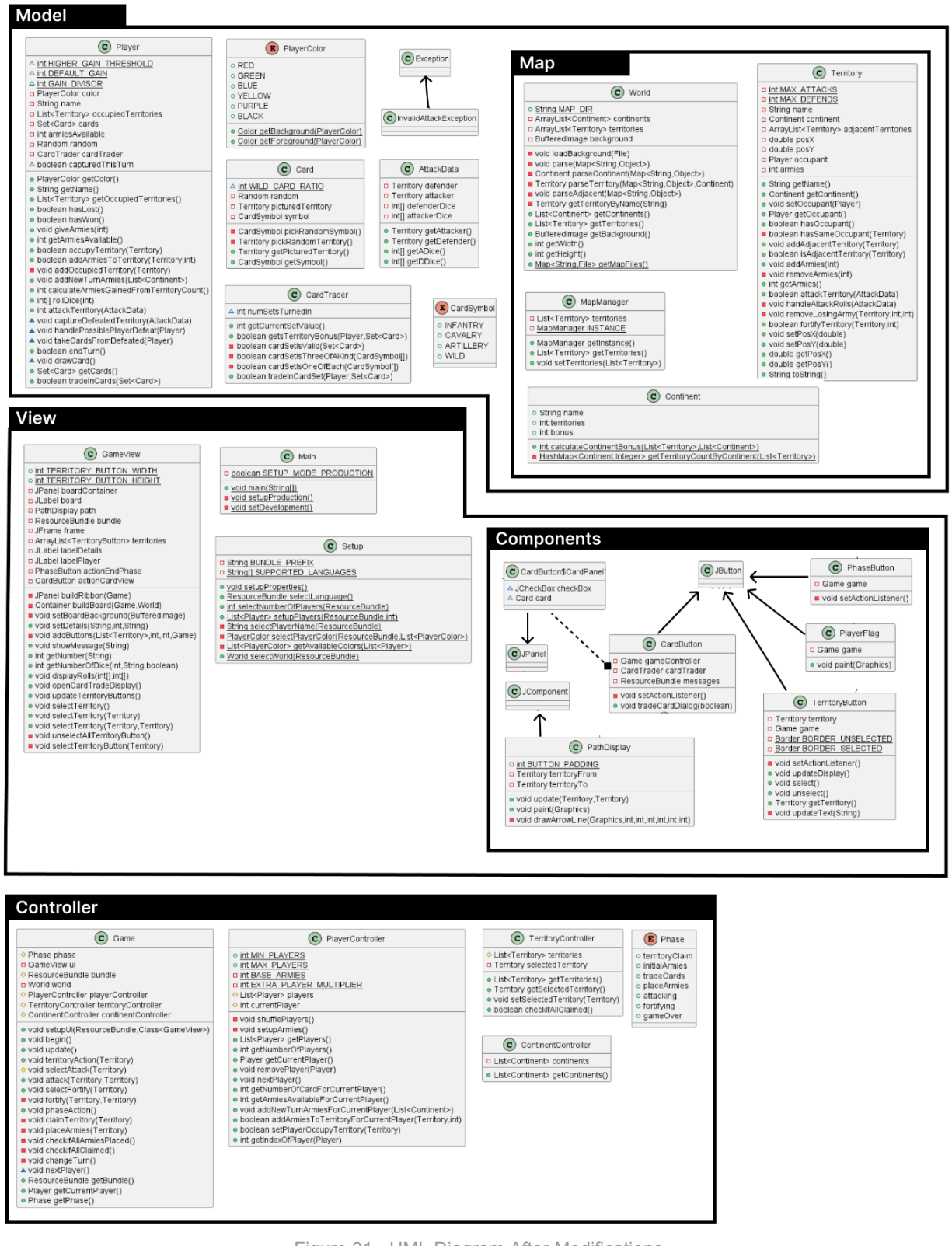


Figure 31 - UML Diagram After Modifications

Notable Design Changes

- Procedures for setting up the game have been moved to their own class. This process used to be short but now that we're adding things like missions, maps, and names, there's a lot more going on here so it gets its own class to avoid concentrating too much responsibility in one place.
- Map-related classes have been moved to their own package including a new class for continents. This organizes things better to help developers and allows us to support Secret Missions that care about conditions like controlling an entire continent.
- Territories now keep track of coordinates themselves rather than having a separate class for them. Coordinates don't really mean anything outside of being two numbers so giving them their own class just makes things more complicated for no reason.
- Map takes information via parameters instead of being hard-coded. This allows us to define different maps depending on the image and other data files that we pass to the game.
- Win Condition is an interface that supports conditions like World Domination (the default) and Secret Missions. This means that new win conditions can be created, added, or removed with great ease using strategy patterns.
- Card now contains the logic responsible for generating itself as well as new behavior to support wild cards. This allows for wild cards to work as well as making it more flexible because if someone wants to change how cards work in the future they only need to modify cards rather than modify the parts responsible for creating them. (Or if they want to change when they're created they change that spot but don't change the Card class.)
- Game has been broken up and many of its responsibilities are now handled by controller classes. This makes the code easier to test and understand by not having massive amounts of functionality crammed into the same class. It's also helpful for flexibility because there's no longer one single class that has a ton of reasons why it might need to be changed.
- View has additional classes responsible for creating a more descriptive UI (consolidated into the Components package). This helps with the organization by separating them from everything else but its main job is to make the game easier to play by making it more clear what's going on at any given time.

Testing Plan/Strategy/Suite

Testing Suite

Testing using JUnit with the Gradle build pipeline to allow for continuous development by our developers, ensuring consistent results. Additionally, restrictions are placed on our master repository, requiring all tests in our testing suite to pass before merging - see Testing Automation with GitHub.

Testing Automation with GitHub

When a new pull request is created, GitHub will automatically run our set of continuous integration tasks to validate the build. This includes running Test Cases as well as the prescribed Quality Assurance Tools. If any of these tasks/tests fail, you will be prevented from merging the pull request into the main branch, see troubleshooting. The results of these continuous integration tasks can be seen on GitHub Actions Menu and the action script can be accessed at .github/workflows/gradle.yaml in the repository. To learn more about GitHub Actions.

Testing Plan & Strategy

Our testing strategy lays out a high-level plan for continuous integration activities to design high-quality software. Utilizing JUnit most of our testing will be accomplished through Unit Testing, with most of our Integration Testing and Acceptance Testing being done manually. As part of our testing strategy, our code will also undergo a subscribed set of quality assurance checks, to ensure the code meets our high standards.

Unit Testing

As this game was initially developed for the Software Quality Assurance term project, the majority of our code was written using test-driven development (TDD). Furthermore, boundary value analysis was used to determine which unit tests should be written in that class. Mocking (specifically the EasyMock library) was used to ensure our test cases were testing only the behavior localized to the method in question. All dependencies to other classes or methods were mocked in these unit tests.

Integration Testing

As portions of the codebase were finished with unit tests during SQA, integration tests were created to ensure the code would still function correctly when the overall logic spanned the actual behavior of multiple methods or classes, rather than that of the preprogrammed mocks.

Acceptance Testing

Acceptance testing is a technique used to determine whether or not a system meets the software requirements specification, set by the development team. In our case, this type of testing is performed mainly manually because of the non-deterministic nature of the game. The tester walks through a set of predefined test cases or an alternative script, to determine if a feature is working as intended and meets the design specifications.

Alternate Languages

1. Select the desired language under test, and click “OK” to finalize the selection. See Figure 32.
2. Ensure subsequent instructions, Step 3 to Step 5, are in the selected language. See Figure 33.
3. Ensure the main game interfaces use the selected language - including player details (Figure 35), game control interface (Figure 36), game modals, cards, and territory information (Figure 34).

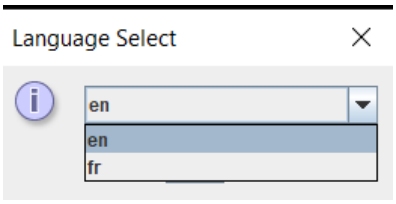


Figure 32 - Language Selection Dialog

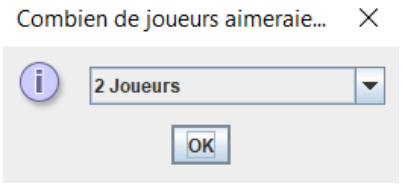


Figure 33 - Player Selection Dialog

Figure 34 - Territories Map

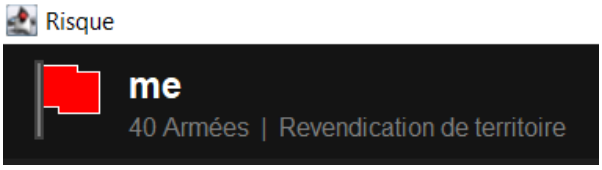


Figure 35 - Player Details

Figure 36 - Game Control Interface

Manual Gameplay Testing

When conducting acceptance testing of the actual gameplay of the program, one has a wide variety of options in doing so depending on what interactions they'd specifically like to test. Most commonly, the number of players or map isn't important to the interaction in question. In cases like these, the use of development mode is recommended to reduce the overhead of repeatedly going through the setup options.

Generally speaking, as this game was created based on vanilla Risk rules (the only material difference being the randomized generation of cards and the option to play alternate maps/game modes), one can verify the correctness of the following interactions by consulting the official Risk rules, specifically the first 10 pages of the document that relate only to the classic World Domination game mode.

To test a specific interaction, one can simply control both players to claim territories and play the game in a way to bring about the desired initial game state for the manual test. It's important to note that one must keep track of certain information, such as the number of turns taken or card sets turned in, to accurately assess whether interactions are actually playing out correctly. For instance, if one gains 10 armies after trading in a card set, that is only correct if there were no territory bonuses from the cards and it was the 4th set turned in of the game.

Seeing as Risk games take a considerable number of turns to complete and relatively few options are afforded to players, playing a game from start to finish with 2 or more people actually trying to win will almost certainly end up testing all important player interactions multiple times over. By tracking certain game information independently from the program, one can easily spot discrepancies with the vanilla game this way.

Test Cases

Our test cases are built using JUnit and consist of both Unit Tests and Integration Tests. We have just about 260 tests that cover both the controller and model aspect of our application. As prescribed by development practices, we aim to have tests to back all features in our system. Additionally, before branches can be merged into our master branch, all test cases must pass, see Testing Automation with GitHub. Below is an up-to-date listing of all 260 unit & integration tests and their status. Newly added test cases are underlined, these test cases were added to either better validate previous features or validate new features.

GameTests

Test	Duration	
testDefaultPlayers	0.146s	✅
testInitGamePlayerArray	0.169s	✅
testNextTurn	0.145s	✅
testNextTurn1	0.529s	✅
testNextTurn5	0.229s	✅
testSetPlayers	0.160s	✅
testAttack	0.143s	✅
testFortify	0.132s	✅
testInvalidFortify	0.139s	✅
testGetBundle	0.101s	✅
testBegin	0.160s	✅
testTerritoryActionTerritoryClaim	0.086s	✅
testTerritoryActionPlaceArmies	0.161s	✅
testTerritoryActionPlaceArmiesNotOwner	0.193s	✅
testTerritoryActionPlaceArmiesTooMany	0.206s	✅
testTerritoryActionAttackNotOccupied	0.146s	✅
testTerritoryActionAttackNotEnoughArmies	0.275s	✅

testTerritoryActionAttackFirst	0.141s	✅
testAttackLose	0.096s	✅
testAttackWin	0.140s	✅
testAttackWinInvalidMove	0.104s	✅
testAttackWinInvalidMove2	0.183s	✅
testAttackSame	0.092s	✅
testAttackNotAdjacent	0.127s	✅
testTerritoryActionFortifyNotOccupied	0.163s	✅
testTerritoryActionFortifyNotEnoughArmies	0.081s	✅
testTerritoryActionFortifyFirst	0.146s	✅
testFortifyValid	0.105s	✅
testFortifyDifferentOccupant	0.105s	✅
testFortifyNotAdjacent	0.135s	✅
testPhaseActionTerritoryClaim	0.114s	✅
testPhaseActionInitialArmies	0.168s	✅
testPhaseActionPlaceArmies	0.176s	✅

ProductionModeTest

Test	**Duration**	
validateProductionMode	0.0s	✅

CardTraderTest

Test	Duration	
testAddsTerritoryBonus	0.002s	✅
testAllTerritoriesCanBeGenerated	0.011s	✅
testCalculateValueFiveSetsTurnedIn	0.001s	✅
testCalculateValueFourSetsTurnedIn	0.001s	✅
testCalculateValueMaxTurnedIn	0.001s	✅
testCalculateValueNoSetsTurnedIn	0.0s	✅
testCalculateValueOneSetTurnedIn	0.001s	✅
testCalculateValueSixSetsTurnedIn	0.001s	✅
testCalculateValueTooManyTurnedIn	0.002s	✅
testGenerateArtilleryCard	0.004s	✅
testGenerateCavalryCard	0.007s	✅
testGenerateInfantryCard	0.003s	✅
testGenerateWildCard	0.004s	✅
testInvalidCombination1	0.001s	✅
testInvalidCombination2	0.001s	✅
testInvalidSetSizeTrade	0.001s	✅
testInvalidSizeIncludingWildCard	0.051s	✅
testNullCardSetTrade	0.002s	✅
testNullPlayerTrade	0.002s	✅
testTerritoryBonusMultipleMatches	0.002s	✅
testTerritoryBonusNoMatches	0.002s	✅
testTerritoryBonusNoTerritories	0.001s	✅
testTerritoryBonusNullCardSet	0.004s	✅

testTerritoryBonusNullPlayer	0.002s	✅
testTerritoryBonusOneMatch1	0.002s	✅
testTerritoryBonusOneMatch2	0.002s	✅
testTerritoryBonusSetSizeFour	0.002s	✅
testTerritoryBonusSetSizeTwo1	0.002s	✅
testTerritoryBonusSetSizeTwo2	0.001s	✅
testValidCombinationAllOne1	0.002s	✅
testValidCombinationAllOne2	0.001s	✅
testValidCombinationAllOne3	0.001s	✅
testValidCombinationAllOneWithWildCard	0.001s	✅
testValidCombinationMultipleWildCards	0.002s	✅
testValidCombinationOneEach1	0.001s	✅
testValidCombinationOneEach2	0.001s	✅
testValidCombinationOneEachWithWildCard	0.002s	✅

ContinentTest

Test	**Duration**	
testCalculateContinentBonusMissingOneTerritoryFromEachContinent	0.004s	✅
testCalculateContinentBonusWithAllTerritories	0.007s	✅
testCalculateContinentBonusWithEmptyList	0.002s	✅
testCalculateContinentBonusWithSingleContinent	0.005s	✅

ContinentTest

Test	Duration	
testPlayerCapturesAndDrawsAndTakesDefeatedCard	0.002s	✅
testPlayerCapturesAndDrawsNoDefeat	0.002s	✅
testPlayerDoesNotCaptureNoDraw	0.002s	✅
testPlayerWinsAttackButDoesNotCaptureNoDraw	0.002s	✅

PlayerTest

Test	Duration	
testAddArmiesToUnoccupiedTerritory1	0.135s	✅
testAddArmiesToUnoccupiedTerritory1Integration	0.121s	✅
testAddArmiesToUnoccupiedTerritory2	0.138s	✅
testAddArmiesToUnoccupiedTerritory2Integration	0.123s	✅
testAddMoreArmiesThanAvailable	0.227s	✅
testAddMoreArmiesThanAvailableIntegration	0.097s	✅
testAddNegativeArmies	0.087s	✅
testAddNegativeArmiesIntegration	0.140s	✅
testAddNewArmiesWith15Territories10ArmiesAlready	0.098s	✅
testAddNewArmiesWith15Territories10ArmiesAlreadyIntegration	0.089s	✅
testAddNewArmiesWith19Territories3Continents	0.085s	✅
testAddNewArmiesWith19Territories3ContinentsIntegration	0.110s	✅
testAddNewArmiesWith1Territory	0.118s	✅
testAddNewArmiesWith1TerritoryIntegration	0.088s	✅
testAddNewArmiesWith9Territories1Continent	0.085s	✅
testAddNewArmiesWith9Territories1ContinentIntegration()	0.196s	✅

testAddToNullTerritory	0.091s	✅
testAddZeroArmies	0.092s	✅
testAddZeroArmiesIntegration	0.102s	✅
testArmiesGainedElevenTerritoryCount	0.102s	✅
testArmiesGainedElevenTerritoryCountIntegration	0.087s	✅
testArmiesGainedFifteenTerritoryCount	0.093s	✅
testArmiesGainedFifteenTerritoryCountIntegration	0.093s	✅
testArmiesGainedOneTerritoryCount	0.127s	✅
testArmiesGainedOneTerritoryCountIntegration	0.085s	✅
testArmiesGainedThirteenTerritoryCount	0.087s	✅
testArmiesGainedThirteenTerritoryCountIntegration	0.089s	✅
testArmiesGainedTwelveTerritoryCount	0.129s	✅
testArmiesGainedTwelveTerritoryCountIntegration	0.114s	✅
testArmiesGainedZeroTerritoryCount	0.084s	✅
testAttackOwnTerritory	0.091s	✅
testAttackOwnTerritoryIntegration	0.089s	✅
testAttackTerritoryCapture1	0.086s	✅
testAttackTerritoryCapture1Integration	0.129s	✅
testAttackTerritoryCapture2	0.086s	✅
testAttackTerritoryCapture2Integration	0.113s	✅
testAttackTerritoryInvalidRolls1	0.092s	✅
testAttackTerritoryInvalidRolls1Integration	0.088s	✅
testAttackTerritoryInvalidRolls2	0.091s	✅
testAttackTerritoryInvalidRolls2Integration	0.088s	✅
testAttackTerritoryNotCapture1	0.116s	✅

testAttackTerritoryNotCapture1Integration	0.089s	✅
testAttackTerritoryNotCapture2	0.136s	✅
testAttackTerritoryNotCapture2Integration	0.090s	✅
testAttackWithUncontrolledTerritory1	0.166s	✅
testAttackWithUncontrolledTerritory1Integration	0.115s	✅
testAttackWithUncontrolledTerritory2	0.107s	✅
testAttackWithUncontrolledTerritory2Integration	0.124s	✅
testDoesNotHaveCards	0.114s	✅
testEndTurnDidCapture	0.112s	✅
testEndTurnNoCapture	0.086s	✅
testHandlePossiblePlayerDefeatedIsDefeated	0.086s	✅
testHandlePossiblePlayerDefeatedNotDefeated	0.084s	✅
testInvalidSet	0.112s	✅
testOccupyNullTerritory	0.092s	✅
testPlayerColors	0.092s	✅
testPlayerHasInsufficientCards	0.086s	✅
testPlayerHasLost1	0.143s	✅
testPlayerHasLost2	0.116s	✅
testPlayerHasLost2Integration	0.096s	✅
testPlayerHasWon1	0.089s	✅
testPlayerHasWon1Integration	0.124s	✅
testPlayerHasWon2	0.091s	✅
testPlayerHasWon2Integration	0.138s	✅
testPlayerOccupyOccupiedTerritory1	0.192s	✅
testPlayerOccupyOccupiedTerritory1Integration	0.123s	✅

testPlayerOccupyOccupiedTerritory2	0.154s	✅
testPlayerOccupyOccupiedTerritory2Integration	0.124s	✅
testPlayerOccupyUnoccupiedTerritory1	0.086s	✅
testPlayerOccupyUnoccupiedTerritory1Integration	0.144s	✅
testPlayerOccupyUnoccupiedTerritory2	0.089s	✅
testPlayerOccupyUnoccupiedTerritory2Integration	0.121s	✅
testPlayerOccupyUnoccupiedTerritoryNotEnoughArmies	0.084s	✅
testPlayerOccupyUnoccupiedTerritoryNotEnoughArmiesIntegration	0.089s	✅
testRollNegativeDice	0.087s	✅
testRollOneDie	0.105s	✅
testRollTwoDice	0.119s	✅
testRollZeroDice	0.105s	✅
testStartsWithNoTerritories	0.089s	✅
testSuccessfulCardTrade1	0.121s	✅
testSuccessfulCardTrade2	0.127s	✅
testSuccessfullyAddMultipleArmies	0.140s	✅
testSuccessfullyAddMultipleArmiesIntegration	0.087s	✅
testSuccessfullyAddOneArmy	0.090s	✅
testSuccessfullyAddOneArmyIntegration	0.096s	✅
testTakeCardsBothEmpty	0.140s	✅
testTakeCardsMultipleEach	0.090s	✅
testTakeCardsOneEach	0.093s	✅
testTakeEmptyCardsAlreadyHasMultiple	0.096s	✅
testTakeEmptyCardsAlreadyHasOne	0.095s	✅
testTakeMultipleCardsHasNone	0.085s	✅

testTakeMultipleCardsHasOne	0.110s	✅
testTakeOneCardHasMultiple	0.235s	✅
testTakeOneCardHasNone	0.114s	✅
testTooLargeSet	0.085s	✅
testTooSmallSet	0.098s	✅

TakingCardIntegrationTest

Test	**Duration**	
testPlayerCapturesButNotDefeatTakesNoCards	0.001s	✅
testPlayerDefeatsAndNoCardsToTake	0s	✅
testPlayerDefeatsAndTakesTenCards	0.002s	✅
testPlayerDefeatsAndTakesThreeCards	0.001s	✅

TerritoryTest

Test	**Duration**	
testFortifyTerritoryInvalidNumberOfUnits3	0s	✅
testFortifyUnownedTerritory	0.001s	✅
testGetters	0s	✅
testIsOccupiedIsOccupied	0s	✅
testIsOccupiedNotOccupied	0.001s	✅

TerritoryTest - AdjacencyTestNet

Test	Duration	
test20ValidAdjacentTerritories	0.001s	✅
testInvalidTerritoryIdIsAdjacentLargerList	0.001s	✅
testInvalidTerritoryIdNotAdjacent	0s	✅
testMultipleValidAdjacentTerritories	0.001s	✅
testValidTerritoryIdIsAdjacent	0.001s	✅
testValidTerritoryIdIsAdjacentLargerList	0.001s	✅
testValidTerritoryIdNotAdjacent	0.001s	✅

TerritoryTest - FortifyTestNet

Test	Duration	
testFortifyTerritoryCallerDoesntHaveEnoughUnits	0.001s	✅
testFortifyTerritoryInvalidNumberOfUnits	0s	✅
testFortifyTerritoryInvalidNumberOfUnits2	0s	✅
testFortifyTerritoryMove20Units	0.001s	✅
testFortifyTerritoryMove5Units	0s	✅
testFortifyTerritoryMoveAllUnitsFromTerritory	0s	✅

TerritoryTest - AttackTestNet

Test	Duration	
testAttackTerritoryAttackerHas0Armies	0.002s	✅
testAttackTerritoryAttackerLoses1	0.001s	✅
testAttackTerritoryAttackerLoses1DefenderLoses1	0s	✅
testAttackTerritoryAttackerLoses1DefenderLoses1OrderMatters	0.002s	✅
testAttackTerritoryAttackerLoses1OrderMatters	0.001s	✅
testAttackTerritoryDefenderHas0Armies	0.001s	✅
testAttackTerritoryDefenderInvalidRoll	0.001s	✅
testAttackTerritoryDefenderLoses1	0s	✅
testAttackTerritoryDefenderLoses1AndIsEmpty	0s	✅
testAttackTerritoryDefenderLoses1OrderMatters	0s	✅
testAttackTerritoryDefenderLoses2	0s	✅
testAttackTerritoryDefenderTooManyRollValues	0.001s	✅
testAttackTerritoryInvalidRolls	0.001s	✅
testAttackTerritoryOneArmyShort	0.001s	✅
testAttackTerritoryTooManyRollValues	0.001s	✅

TradingCardsIntegrationTest

Test	Duration	
testInvalidSet	0.002s	✅
testTradeInFirstValidSetNoBonus	0.002s	✅
testTradeInNinthSet	0.003s	✅
testTradeInSecondValidSetWithBonus	0.001s	✅
testTradeInTenthSet	0.001s	✅
testTwoPlayersTurnInValidSets	0.003s	✅

WinTest

Test	Duration	
testPlayerDoesNotWinSecretMission1	0s	✅
testPlayerDoesNotWinSecretMission2	0s	✅
testPlayerDoesNotWinSecretMissionControlsWrongContinents	0s	✅
testPlayerDoesNotWinSecretMissionOnlyPartialControl	0s	✅
testPlayerDoesWinSecretMission1	0.001s	✅
testPlayerDoesWinSecretMission2	0s	✅
testPlayerHasWonDomination1	0.003s	✅
testPlayerHasWonDomination1Integration	0.003s	✅
testPlayerHasWonDomination1Integration	0.003s	✅
testPlayerHasWonDomination2Integration	0s	✅

ControllerTest

Test	Duration	
testAddArmiesTerritoryPlayer	0.001s	✅
testAllClaimedTerritory	0.146s	✅
testAmountPlayer2	0.003s	✅
testAmountPlayer6	0.002s	✅
testClaimTerritoryPlayer	0.004s	✅
testCurrentIndexPlayer3	0s	✅
testCurrentIndexPlayer6	0.002s	✅
testCurrentPlayer	0.005s	✅
testCurrentPlayerArmies	0.001s	✅
testCurrentPlayerCards0	0s	✅
testGetContinent	0.165s	✅
testGetTerritories	0.173s	✅
testNextPlayer1	0.173s	✅
testNextPlayer2	0.001s	✅
testNextPlayer3	0.001s	✅
testNoneClaimedTerritory	0.222s	✅
testRemovePlayer	0.001s	✅
testSelectTerritory	0.153s	✅
testSetupPlayer3	0.698s	✅
testSetupPlayer6	0.224s	✅
testSomeClaimedTerritory	0.680s	✅
testUnselectTerritory	0.158s	✅

WorldTest

Test	Duration	
GetMaps	0.001s	✅
LoadedBackground	0.098s	✅
LoadedContinentAfrica	0.090s	✅
LoadedContinentAustralia	0.097s	✅
LoadedCoordinatesBrazil	0.096s	✅
LoadedCoordinatesScandinavia	0.108s	✅
LoadedCoordinatesWAustralia	0.108s	✅
LoadedTerritoryBrazil	0.093s	✅
LoadedTerritoryScandinavia	0.108s	✅
LoadedTerritoryWAustralia	0.093s	✅