UNIVERSITY OF LIVERPOOL

2018/19

Student Name: **David Morris**

Student ID: 201084474

Project Title: COMP39X
Year 3 Risk Project

Supervisor: **Primary: Alexi Lisitsa**

Secondary: Martin Zimmermann

DEPARTMENT OF COMPUTER SCIENCE

The University of Liverpool, Liverpool L69 3BX

Contents

Abstract

The projects aim was to implement the board game Risk[1] in a computerised manner, this meant implementing the game and its rules into a computer game. The 2 main parts of the project that needed to be implemented were the front and back ends, the front end needed to display to the user the same information that the board game does, this was done in the forms of a combination of menu GUI's and In-Game Menu's. Whilst the back end kept track of the information during the game as well as having computer opponents that the user would be able to play against.

The first step of the project was designing how the program would work before implementation, as well as creating the designs for the GUI's within the program. Gathering information about the game was necessary and understanding its rules would be crucial to implementing the game.

During the first steps of implementation the first point of call was to create the GUI designs and implement all user input methods as it was necessary to know what user input would be required. Most of the designs were created using the RGE Photoshop [3] and the Java menu items from the IDE NetBeans [4] which is what the program was created in. GUI's were needed for the menu that the user navigates before the game starts as well as providing the user opportunity to select game options. After these designs were created the GUI's were implemented and the components were input the work on the back end began.

The game map was created using the NetBeans IDE [4] and was implemented using a back-ground image of the map and interactable buttons that the user could click to bring up territory information and the ability to invade other territories during their turns, this was quite tricky as the IDE in question does not handle large amounts of items on a GUI very well.

Implementing the back end required creating classes to fulfil purposes regarding the game's rules, a class was made to hold, send, and retrieve information of the territories in the game, such as what territory is selected, what player controls that territory, how many soldiers are placed there, and its name. A class was created for the game turn and to allow players to iterate through their turns. A calculation class was needed to perform the mathematical functions of the program. As well as a deck class for the deck of cards which are handed to players throughout the game. And a parent player class which the human players and computer players will extend from allowing inheritance to become a factor during the implementation.

During this process setbacks did occur, and designs needed to be recreated as limitations of the IDE, and the programming language that was selected, and my Personal Time-Management and Mental Health, created obstacles that delayed the completion of the program. But even with these setbacks the Project was completed implementing most if not all the essential criteria that were set.

Introduction

What problem is addressed?

The aim of this project is to design, create, test, and review a software development project to re-create the board game Risk [1] as a video game. The core aims of which are to implement the game and all its rules using a programming language to allow the user to play the game through a GUI (Graphical User Interface). When playing the game, the user will be able to play against other human players or against an in-game AI (Artificial Intelligence). The Project is aimed for My Supervisor of my course Alexi Lisitsa. For the third-year course COMP 390/393/394/395.

Aims and Objectives

The aim of this project is to create, design, test, and evaluate a software development project to re-create the board game Risk [1] as a video game. This will require implementing the rules of the board game in a programming language to do the same functions as intended in the rules. It will also require a virtual version of the game board that the players can interact with using the same rules and techniques that the board game intends.

This will be done by making each territory on the board an interactive button that the user can select and view the information for that tile allowing them to invade other territories, so the players can achieve victory in the game.

The Objectives of this project are to:

- Implement the Game in a Video game form and all its rules.
- To learn how to implement AI in a Video game to play against a Human player.
- To implement a GUI that the user will be able to easily understand and for players who have played the board game to easily understand aspects of the video game quickly.

Project Challenges

The main challenges that are faced during this project is to implement a game system where both a human and a computer player can interact with the game according to the settings that are chosen by the player allowing various sized games to take place. As well as this it will be necessary to implement an AI into the game for the human players to play against. This will be done with an algorithm as we know that is a set of instructions that the AI will follow.

Another challenge that is posed is the UI which is one of the most important parts of this project as it allows the user to interact with the game and displays the information that the user needs to know to play the game. So, it is essential that the user can interact with each aspect of the UI as intended, meaning that constraints need to be set for the user, so they do not interact with GUI elements that they should not have access to until specified.

Although the language that has been chosen for this project to be implemented in is Java, and it is an Object-Oriented language, and the IDE that has been selected is specialised for use with java, this does mean that the limitations of these decisions will have to be considered during the implementation stage. As a game engine is not being used and a standard JFrame interface was not designed for complex Game structures.

Solution

The solution that has been produced for this project is a set of GUI screens for the user to interact with, objects and data structures to access, contain, and manipulate the data that is needed to perform the functions for the game to run as well as adhering to the rules of the game.

The GUI screens will be divided into 2 parts, the menu GUI's that allow the user to view and select options for the game as well as being the entry point to the program for the user, this was decided to be necessary as other strategy computer games have this feature that allow the user to navigate through the game. The second aspect of the GUI screens will be the game map GUI as this is where the user plays the game and interacts with the game components to play and win the game. A small amount of other GUI screens will be incorporated into this GUI screen as they will be smaller menus that allow the user to see information of the game in more detail than if the information was available to see on the game map. This is because having too much information on the screen at once can be disorienting for the user as it would clutter the game board.

The other classes of the program will be back-end oriented, as the user does not need to witness the computations and functions the program performs and the information that is vital for the user to know will be sent to the GUI screens. This is also useful as the AI does not need to use GUI screens to input data or make decisions, so they can just access the necessary classes and methods that they need to for them to perform actions within the game.

Project Success

The project overall is a success as the core elements of the game have been implemented and the game runs as intended. The menu GUI's are simple, minimalist, and the colour schemes and designs work for the nature of the game, this has been done intentionally as to not confuse the user along with clear unambiguous titles and text meaning that the user is unlikely to misinterpret any information that is displayed.

The Game map GUI did not match the original design that was set at the start of the project, but it has fulfilled most of the requirements that were set for it, as all buttons and most of the menu's work as intended and display the correct information that the user requires. Given the nature of the IDE that has been used implementing more features in the original design was unlikely as adding extra features to the game map

would have disrupted the existing structure and placement of the other UI elements. A clear example of this would be the Card Hand button as it is misplaced in the GUI but due to the IDE's functionality it is impossible to move the button without breaking the placement of each other UI component.

The territory array that has been used works exactly as intended as it is an int array that is 42 by 3 meaning that there are 42 "rows" and 3 "columns" one column to determine which territory the array is referring to, one column to determine which player is in control of that territory, and one column to determine the number of soldiers that currently occupy that territory.

The deck also works as intended as it too is an int array that is 44 by 2 with one "column" determining which card it is, and one "column" for which player is in possession of that card. This also contains a function which adds a card to a player's hand when a card is drawn from the deck, checking to see if the card is owned by a player and if so that card will be unable to be drawn. The only unsuccessful aspect of the deck and hand of cards is that the user is unable to choose when to discard cards for extra troops, instead the program checks to see if the player has a match of cards and spends them immediately.

The AI is not as advanced as it could have been, but it is simple and makes random decisions for every aspect of game play that it can, this is regarding troop deployment, territory selection, invasion target selection, and the amount of moves it makes. But due to its simplicity it does not break the game, nor does it perform any unintended actions.

The functions that make up the game turn work as intended, the initial idea was to use a loop that iterates through and allows a player to take their turn and when they click the end turn button it would cycle the loop and then the next player takes their turn, and this process would be repeated until a player wins the game. However, this was not possible with the use of Java GUI's as if a loop was used the user would be unable to interact with the program as the loop would continuously cycle as there was no way to pause of user input in this manner. So instead a series of method calls is used to represent a "loop" without using the loop functions.

Background

The Problem

A base overview of the game is that Risk [1] is a strategy board game where the players objective is to conquer and occupy all the tiles on the board. The game board is an image of the world map with certain countries or areas of the world segmented into “Territories”. The players conquer these territories using troops that are represented using small cubes where the colour is different based on the nation the player is playing as. The players start with a set amount of territories and soldiers at the start of the game which are deployed in random territories on the map at the start of the game.

Each Player takes turns until all players have taken a turn then starting again from the first player until only one player remains in the game. during the start of each players turn they receive several soldiers that they can place on any territories that they own at the start of that turn. The number of soldiers that each player receives depends on the amount of territories that they own at the start of their turn. After deployment of these reinforcements the player will then decide if they want to invade any of the territories that are adjacent to their own.

When a player declares an attack, each player will roll dice depending on the amount of attacking and defending soldiers. Players will roll a dice for each solder in the invasion up to a maximum of 3 dice for the attackers and a maximum of 2 dice for the defenders, when the dice are rolled the defender allocates their dice to the attackers and then values are compared. If the attackers die values are higher, then they inflict one casualty to the defenders soldiers for each die of a higher value. But if the defenders dice are even or higher than the attackers dice then they inflict one casualty to the attackers soldiers. This process will be repeated until either; the attacker withdraws their invasion, the defender has no more soldiers left, or the attacker only has one soldier left. If the defender has no soldiers left, then the attacker can move any of the remaining troops into the newly conquered territory and claim control of that territory.

If a player conquers at least one territory on their turn then they draw a random card from the deck and add it to their hand. A player may only have a maximum of 5 cards in hand. Each card has either; a foot soldier, a knight, a cannon, or is a wild card which contains all 3 symbols. A player can play these cards only if; a player has “three of a kind” of these symbols (a wild card can only count as one type of symbol when it is played), or one of each of these symbols. When a player plays cards, they will receive an amount of extra reinforcements based on the amount of times players have played cards before them.

Players will then continue this procedure until only one player remains in the game. Then once only one player is in the game they win.

Existing Solutions

There currently exists a small number of computerised versions of the board game Risk, the main one that has been looked at was the Ubisoft Risk Game[12]. This is because it is a game made by a major game developer and publisher meaning that they should use good practice and a good execution of this implementation. There are also several other Strategy computer games that inspiration can be taken from.

Research

The background research that has been done on this project is mainly to do with the rules of the game Risk [1] itself. Since its rulebook and included rules are essential to the workings of the video game version, research has also been done on how to implement these rules into a programming language as some of these rules will be rather tricky to implement due to the nature of using human intellect to anticipate the specific workings of some of the rules. Research has also been done in how to use some of the software required for the completion of this project, such as how to use photoshop [3], NetBeans [4], and ArgoUML [2]. Research has also been done on other strategy games to see how they layout their GUI's to see what they have in common and why they layout their GUI's in that way.

Project Requirements

The list of requirements for this project are as follows:

Essential Requirements	Desirable requirements	Tertiary requirements
- To implement the game board as an interactive GUI - To implement a main menu screen for the user - Have a basic AI that can make decisions randomly - Incorporate the mathematical calculations required for the game - To have a uniform design appearance for all menu's - For the user to be unaware of the back-end functions so they can focus on the UI - To allow multiple human players to play the game on the same machine.	- Incorporate a simple background ambience soundtrack - For the players hand of cards to have the same art as thegame's - Tutorial mission that the user can play to more easily learn the rules and understand the UI - Highlight the territories of the faction whose turn it is. - To close pop-up windows by clicking elsewhere on the screen - display a small text box describing what the component is when the cursor hovers over it - Add screen resolution change options - Incorporate and AI that can make decisions based on the current game state.	- To create an AI that can anticipate future moves and plan accordingly - To include other game types other than that given in the rulebook - To implement a multiplayer option between multiple machines

Data Required

Data will need to be obtained to design and create this project and doing so it will have to be acquired ethically without violating any copyright laws or using data without the owner's permission. So, any data that is externally acquired shall be notified and recorded in the bibliography at the end of this report.

The data that will be obtained during this project will be the use of the Risk board game [1] and its rules as that is what the project is based around, so it will be essential to use the data from the game to help in the creation of this project. It will also be noted that human participants will not be included in this project and will also not be included in the evaluation of the project.

Design

System components

There are essential components for the program that will need to be implemented for the project to fulfil its objectives. The first major component that will be needed will be a UI so that the user can interact with the game, this UI will have interactable components that will allow the user to view and manipulate information within the game. The next major component that will be needed will be an AI or set of AI's for the user to play against as computer opponents. The complexity of the AI will depend on how many of the requirements are completed as the more complex the AI the lower the priority is for this project.

The deck of cards the players draw from is another major component as it is essential to the game rules that they are implemented this will be organised into a multidimensional array to store the cards before they can be drawn into the players hands. The next major component will be the main menu GUI which will allow the user to navigate to the game, edit options, view information and edit game options. Of which all these components will be formed into a menu that can be navigated through by the user.

Used Objects in the system

This section of the design document will show the list of objects that will be used during this project and give a description of the functions the objects will perform.

Main Object

The main object will be the connection point for all the other objects as it will contain the primary main method for the program. It will collect the user inputs and send those requests to the relevant object for query then the object will send the response back to the main method and implement the effect warranted by the request.

GUI Objects

The GUI objects will be the object that connects the rest of the program with the various GUI panels in the program. Giving the buttons effects throughout the

program. The GUI object will not have many functions on its own but will be used to trigger the effects of other objects within the program.

Player Object

The player object will structure the player turns during the game and will send the user the relevant prompts and retrieve the relevant information from the user. The player object will also hold the workings for the Artificial Intelligence for the project as the AI will be used as players in the game. This will mean that the AI will be in an extended player class as to utilise inheritance in the program to make the program more efficient and durable. This will be done by setting a parent player class with two child classes extending from the player class for each a human player and an AI player.

Calculation Object

The calculation object shall hold information for all the numerical processes in the program as during the game “dice” will need to be rolled but to replace dice it will be necessary to calculate a random number and then display its effects to the user. the AI will also have calculations to perform to determine where they place pieces or where they choose to invade during the game. These calculations shall be done in this object then sent to the Player object for the use of the AI.

Deck object

This object in the program will hold the deck of cards that the user and computer opponents will have to call in extra reinforcements in the game. So, the deck will have the cards from the game and the players will draw a random card from the deck when required and that card will not be able to be drawn again.

Territory Object

This object in the program will store all information that relates or interacts with the territories on the game map, information such as which player controls the territory and how many troops are placed there, along with methods that interact and manipulate this data.

Attributes and methods of objects

This section will cover what attributes and methods each object will require for the project and why these are required.

Main Object

The main method will mainly consist of a main method and a few variables for turn counters and method calls to the other objects in the program. It will contain a loop that will ensure the program functions until the user exits the loop by quitting the game.

GUI Object

The GUI Object will have a lot of methods for the use of the GUI interactive components, as most interactive parts of the game will have triggered effects that will

be used in other calculations or methods in the object. The Menu GUI's won't have much except for method calls and method calls to call a new GUI to the screen. The in-game GUI however will require more methods for functions as the territories in the GUI will have attributes, so the players can interact with them.

Player Object

The player object will contain methods for how the user will be able to interact with the GUI's and hold parent information that can be inherited by the user player and the AI player to increase efficiency. The player object will contain a loop which will wait for user input, cycle to the appropriate function perform it if the user has requested it and then return to the top of the loop until the user ends the turn using the button on the GUI screen.

Calculation Object

The calculation object will have a lot of maths functions and so will require methods for each of these functions, as some of the variables will be used to contain the random number generated from the "dice rolls" in the game, for which most of these variables and methods can be kept private until they need to be sent to the user. This object will also calculate the number of soldiers each player gets as reinforcements at the start of the turn checking to see how many soldiers the player can deploy depending on the amount of territories they control, if they control any full regions, and if they have any sets of cards in their hand.

Deck Object

The deck object will contain an array to hold the cards and a mathematical function to randomise the card the player will draw and the ability to once a card is draw it cannot be drawn again until it is discarded and placed back into the deck (array).

Territory Object

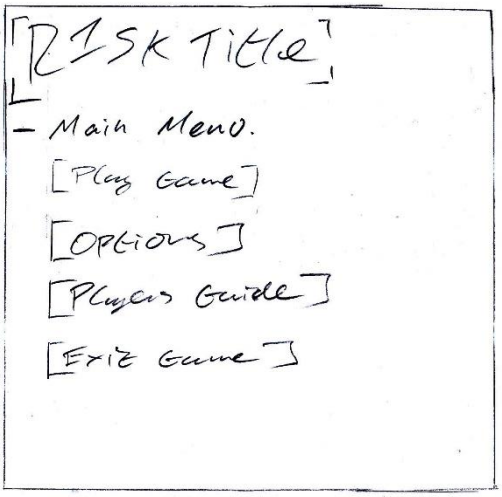
The territory object will contain an array that hold all of the territory information as well as methods that add and remove soldiers and players from that territory, or to acquire information such as how many territories the player controls.

Interface design

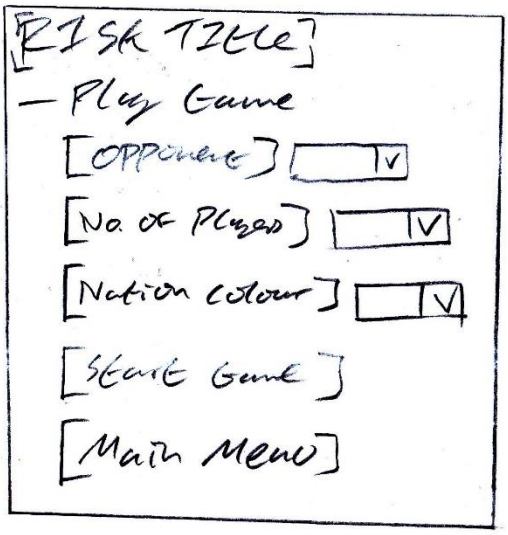
The project will require an interface which the user will interact with, so the program will have a GUI to fulfil that requirement. Within the program there will be multiple GUI's that will display depending on what the user selects. Each of these menu screens will have the same background and have a uniform layout and GUI design.

Starting with the main menu screen which will be the first screen the user will encounter. It will display the main options for the game as a list where each option will be a selectable button which will take the user to the appropriate screen. On the main menu the options will be; Play Game, Options, Players Guide, and Exit Game.

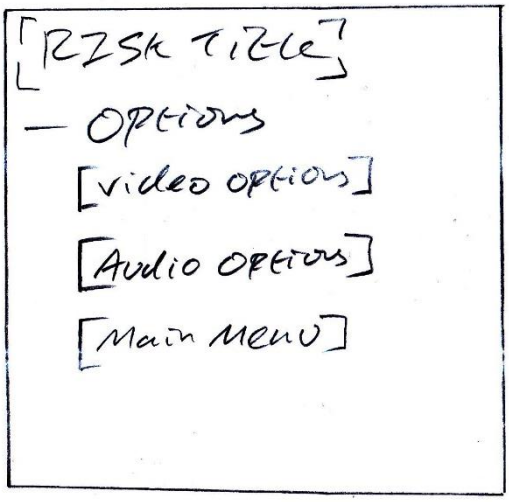
Each menu option will take the user to another menu screen with different options depending on the menu option that was selected.

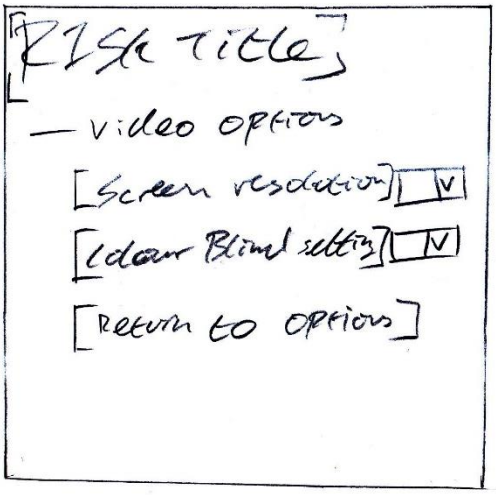


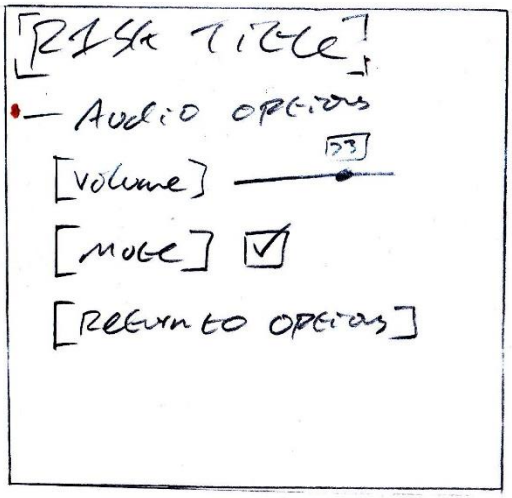
The first option on the main menu screen is Play Game which will take the user to another list of options which displays the following; Opponent, Number of players, Nation colour, Start Game, and Main Menu. The first option will have a dropdown box which displays “Human” or “Computer” this will allow the user to select what type of opponent they will be playing against. The second option will also have a dropdown menu that will allow the user to select the number of players in the game ranging from 2 to 6. The third option will determine what colour their nation and pieces will be for the course of the game. The fourth option will only be available until all the above options have a selected choice, once the conditions are met and the option has been selected the user will be taken to the in-game GUI and a notification window will appear displaying the essential information to the user and then the game will begin. The final option is main menu and will take the user back to the main menu screen, so they may choose another option. In the original design the user would be able to choose what colour they wished to play as however given the nature of how the player colours must be implemented the player colour is now decided by what player they are.



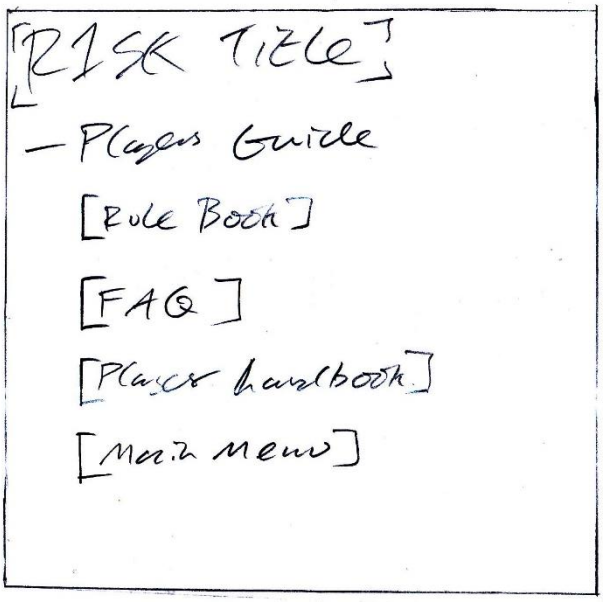
The second option on the main menu screen is Options which will take the user to another list of options which displays the following; Video Options, Audio Options, and Main Menu. The first option will contain a small list of basic changes the user can make to the visual preferences of the game such as screen resolution, colour blind options, and Return to Options which will send the user back to the main options menu. The second option on this screen is Audio Options where the user can do the following; change the volume of the music, mute all sounds, and Return to Options which will send the user back to the main options menu. The final option on the list is main menu which will take the user back to the main menu screen, so they can select another option.







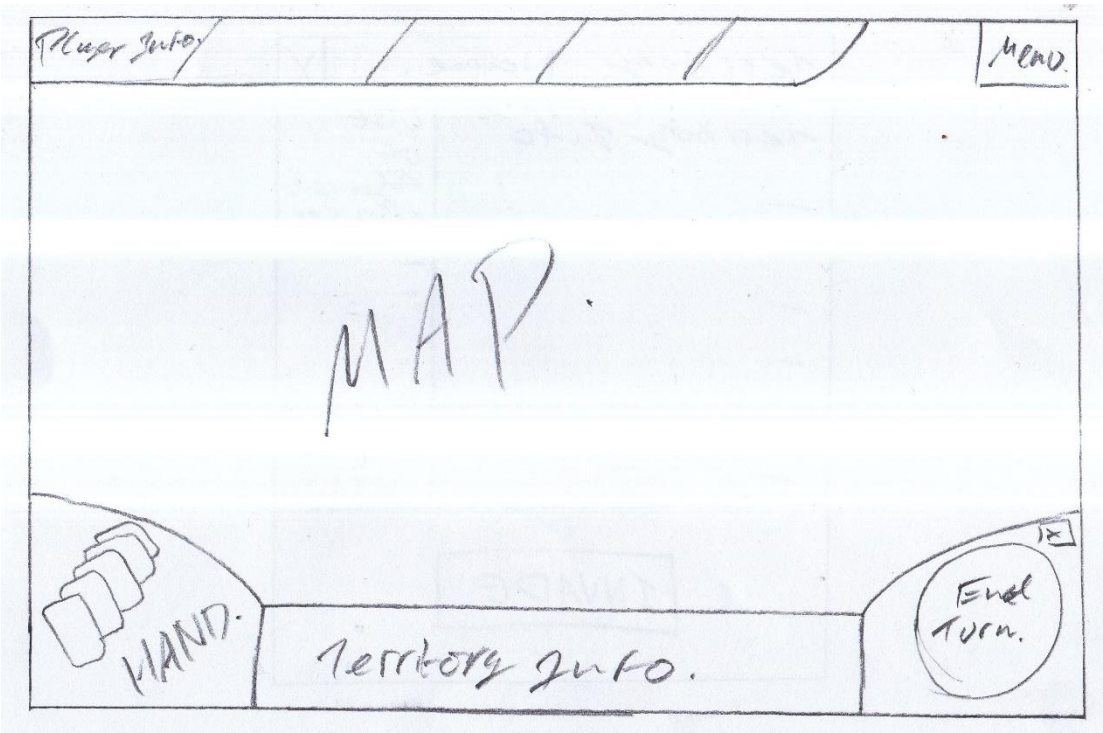
The third option on the main menu screen is Players Guide which will take the user to another list of options that displays the following; Rulebook, FAQ, Player Handbook, and Main Menu. The first option of which will open a window that will allow the user to view the actual Risk Board game rulebook [1], containing all the information that comes with the actual board game. The second opens a list of frequently asked questions that will answer questions about ambiguity on certain



rules and situations for the user to more easily understand the way the game works. And the Third Option Player handbook will give the user basic instructions on how to play and more easily understand the workings of the game. The final option Main Menu will take the user back to the main menu screen, so they can choose another option.

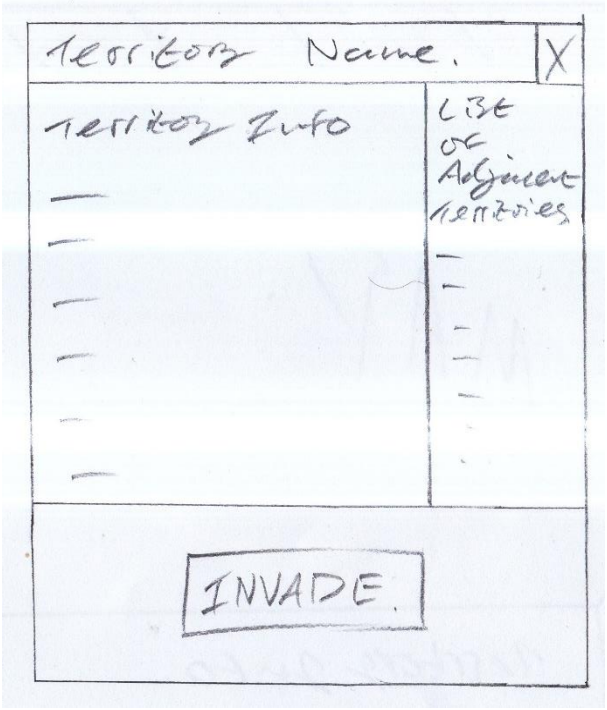
The final option given on the main menu is Exit Game which will close the program down and end all processes.

The next Main GUI design in this project is the in-game GUI the design of the Game GUI will be a top-down view of the map like how one would play the board game as the map is in sight from an elevated point of view. The map itself will be a copy of the game board in the board game with the graphics updated to suit the computerised version. The GUI itself shall have toolbars containing information and functions that the user will be able to interact with. These designs are inspired by a multitude of strategy video games such as Hearts of Iron 4 [5], Civilisation 5 [6], and Dawn of war [7].



At the bottom of the screen starting from the left will be the players hand of cards where the user can view their hand and select the cards to trade them in for additional reinforcements as per the game rules. And in the bottom right of the screen there shall be the end turn button and a turn counter which shall display the amount of turns the game has gone through. The main menu button in the top left will take the user to the main menu of the game.

Finally, when the user selects a territory on the game map a small window will open displaying information of the selected tile. The information is as follows; the name of the territory, the number of troops stationed at that territory, the territories adjacent to this one, their troop count, and their owner. At the bottom of this window the user can select the button which will allow the user to invade one of the adjacent territories using the troops based at the selected territory.



Evaluation design

For evaluating this project, it will be essential to look at the initial requirements and objectives for the project that have been set at the start of the design document and see if they have been met. After determining if the requirements have been met the game will be tested using criteria based on the board game to see if all functions in the program work the same as the board game. For the evaluation process there will have to be 3rd party evaluators for the program to see if all processes work correctly and to see if the game holds true to the original version.

The type of tests that will be undertaken to see if the project meets its criteria will be done in 2 parts, the first part will be to test the front end and see if all interactions have the same effects as those in the risk rules to ensure that the game runs as intended. The second type of testing will be done throughout the implementation process to see if all parts of the program work before the finished product and to ensure that all back-end processes have no unintended effects.

Algorithms and Artificial Intelligence

Although this project is primarily an Object-oriented design project there will be an aspect of algorithm use as the AI players will have to have instructions to follow to play the game.

For a standard AI for the player to play against the AI will decide everything randomly, when deploying troops, it will determine a random number of soldiers to place in a random territory it owns. There will be constraints meaning that they will not be able to deploy more troops than they have and having the AI be unable to deploy troops or invade a territory they do not control.

For the advanced AI, it will have an algorithm to decide what function to make when given a choice, for example the AI will require an algorithm to determine where to place the reinforcements that player gains at the start of that players game turn. The algorithm that is planned for this use will allow the AI player to determine which tiles have the greatest ratio of enemy forces compared to that players and place a soldier there, then will repeat this process until a suitable ration of friendly to enemy troops is more even. The next major algorithm that will be in use by the AI will be how the AI players determine where to invade. Once again, the AI will calculate which enemy territories have the best ratio of friendly troops to enemy troops. And shall try to invade that territory. As a more desirable requirement an algorithm will be put into place to focus the AI to try and conquer the territories in a continent rather than just attacking based on the ratio of strength of soldiers.

This section has been expanded on as the project continued as at the start of the project it was unknown what manner the AI would have been implemented as.

System Components

There are essential components for the program that will need to be implemented for the project to fulfil its objectives. The first major component that will be needed will be a UI so that the user can interact with the game, this UI will have interactable components that will allow the user to view and manipulate information within the game. The next major component that will be needed will be an AI or set of AI's for the user to play against as computer opponents. The complexity of the AI will depend on how many of the requirements are completed as the more complex the AI the lower the priority is for this project.

The deck of cards the players draw from is another major component as it is essential to the game rules that they are implemented this will be organised into a multidimensional array to store the cards before they can be drawn into the players hands. The next major component will be the main menu GUI which will allow the user to navigate to the game, edit options, view information and edit game options. Of which all these components will be formed into a menu that can be navigated through by the user.

Data Structures

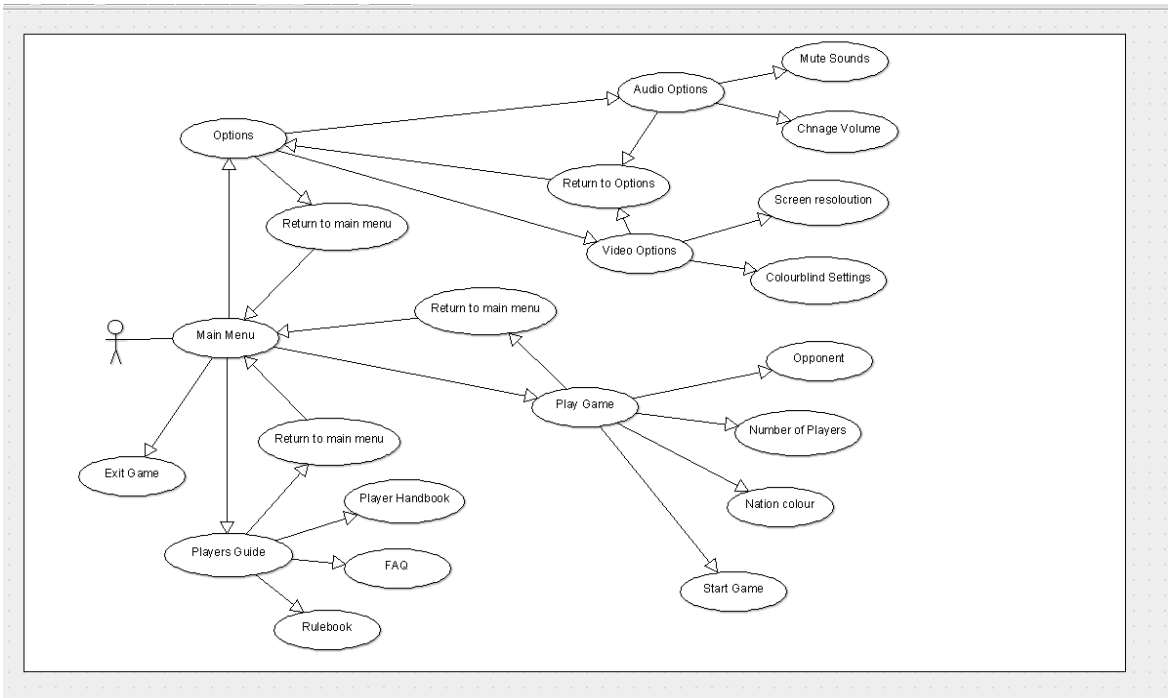
Since a database will not be used, and Lists are not available in Java, during the implementation of the project the main type of data structures that will be used will be arrays and vectors. As an Array will be used to store the deck of cards that the players will receive when they conquer territories in the game. This deck array will also store the information regarding which player is currently in possession of a card with a constraint disallowing anyone to draw a card from the deck if a player has it in hand. An array will also be used for the territories as data will be specific to the territories as players will own, control, and have stationed troops on each territory which means that an array will store all the information for each territory.

Object Oriented design methodology

To help in understanding how the user will interact with the program some Use Case diagrams have been made to understand the user decision process and how it interacts with the GUI's and what outcomes they might have.

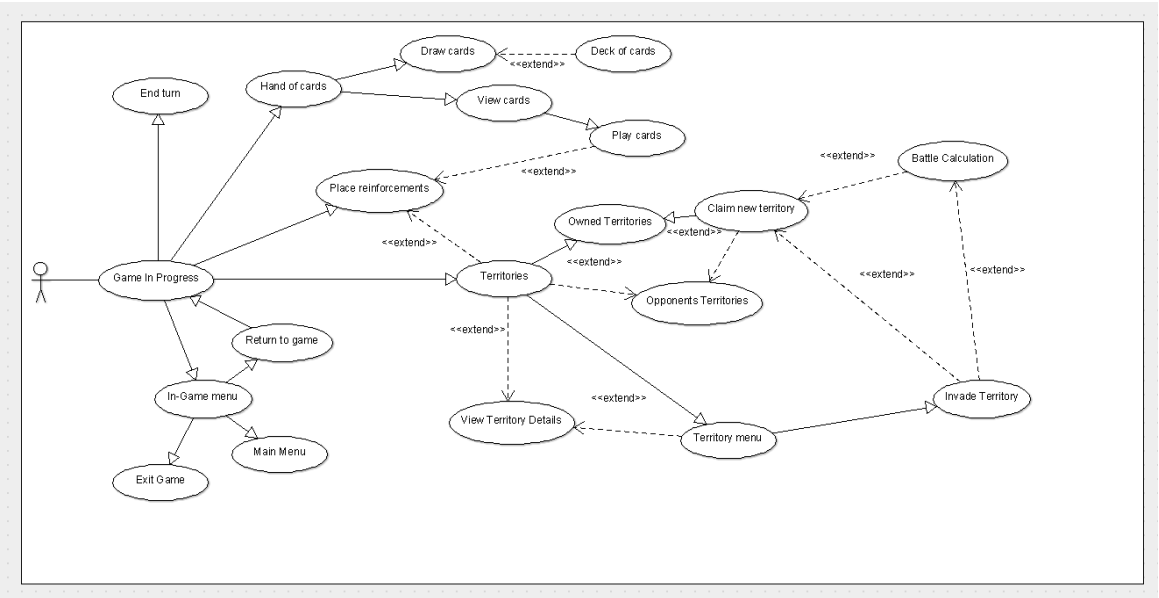
User Menu GUI Use case

Here is a GUI User menu Use Case Diagram made in ArgoUML [2]:



User Game GUI Use case

Here is a Game GUI Use Case Diagram made in ArgoUML [2]:



Realisation

Implementation Description

Initially at the start of the implementation stage of the project the first aspect of the program that was implemented was the menu GUI screens, this was because the users first point of entry into the program would be the main menu GUI, and from there the user would be able to navigate through various menu's that would display information to the user. The menu's that follow off from the main menu GUI have a uniform design in background and button aesthetic. These menu GUI's did not differ from the initial designs in both layout and functionality.

After these menu GUI's were implemented the focus of development moved on to the game map GUI. This is the main GUI screen that the user would interact with and access information regarding the game. Since this is the case it would be important for the implementation stage to know exactly what the user will be able to interact with and what user input would be needed and what information needed to be output for the user to use in the game.

The start game GUI is the screen that the user interacts with before the game starts and allows the user to select the options for the game, starting with the type of game the user can either play against other human players or against computer opponents. The other option that the user can select are the number of players that are in the game ranging from 2 to 6 players. These options are selected using a combo box, and when the user clicks the start game button the game process is initiated taking the options selected by the user to be able to set the preferences for the game.

Before the GUI's could be implemented it was necessary to create the images and backgrounds that the program would need. This was done by using the RGE Photoshop [3] to create and edit images that were made using Photoshops [3] tools and resources. The buttons for the menu and map GUI's were created using this method, along with the game map itself.

Once these images were created it was possible to now implement the appearance for the GUI screens. This was done by importing the Java buttons using the IDE and setting the icons to these images that were created and imported to the project. once this was done the next step was placing the map on a JFrame with a transparent JPanel on-top of this image frame. Once this was completed it was now possible to place the JButtons onto the GUI and place them onto the map in the places that represent the territories.

Once the user interfaces were complete it was then necessary to create the territory menu that the user would interact with when the territory buttons were clicked and display the information regarding what territory was selected, which territories can be invaded from the selected territory, along with the soldiers and players that are in possession of the territories selected. This was done by placing a combo-box to

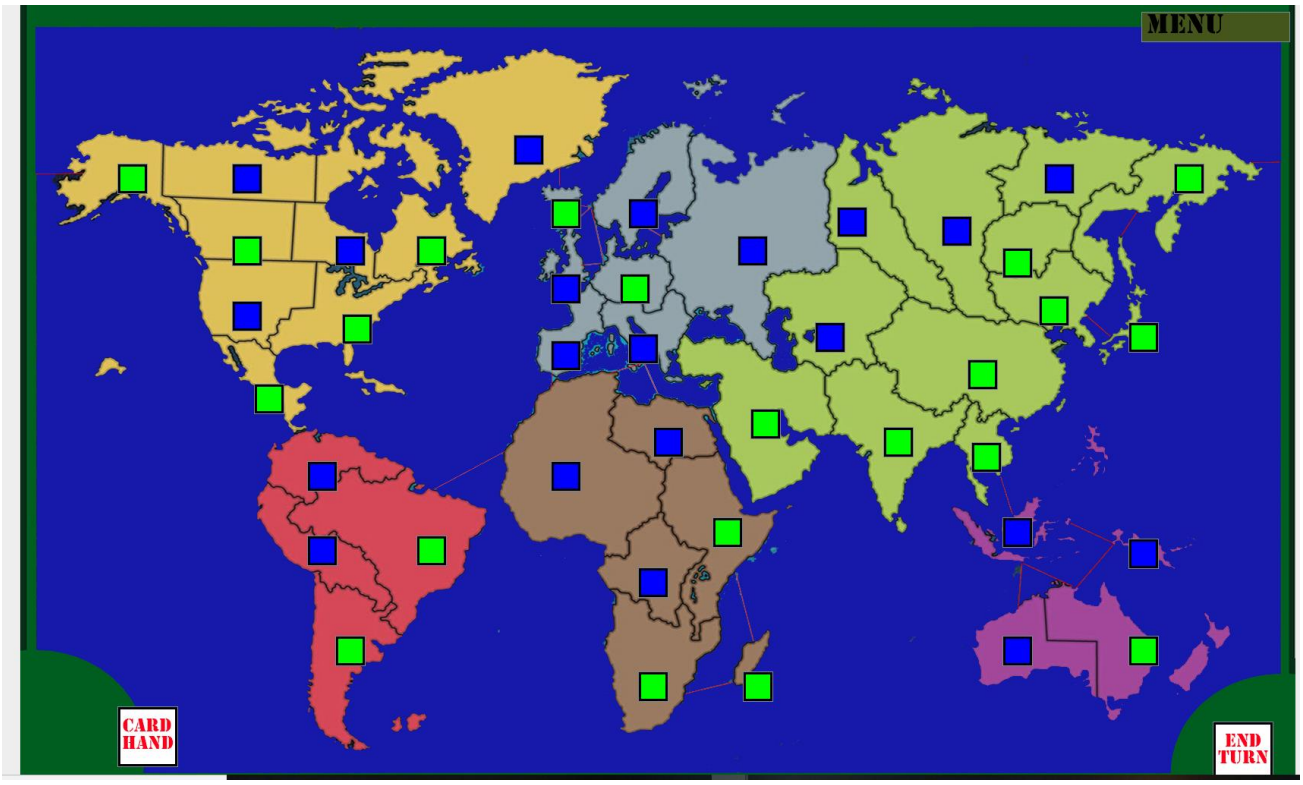
select the territory that the user wishes to invade, a button to close the menu, a button to invade the selected territory, and text boxes to display information to the user such as the number of soldiers, the players, the territory names, and the dice roll results if the invade button is clicked.

Next the card hand GUI was made, this is a simple GUI which just display the cards that are currently in possession of the current player and what those cards are. This is done using a text box which outputs the names of the cards that the user has.

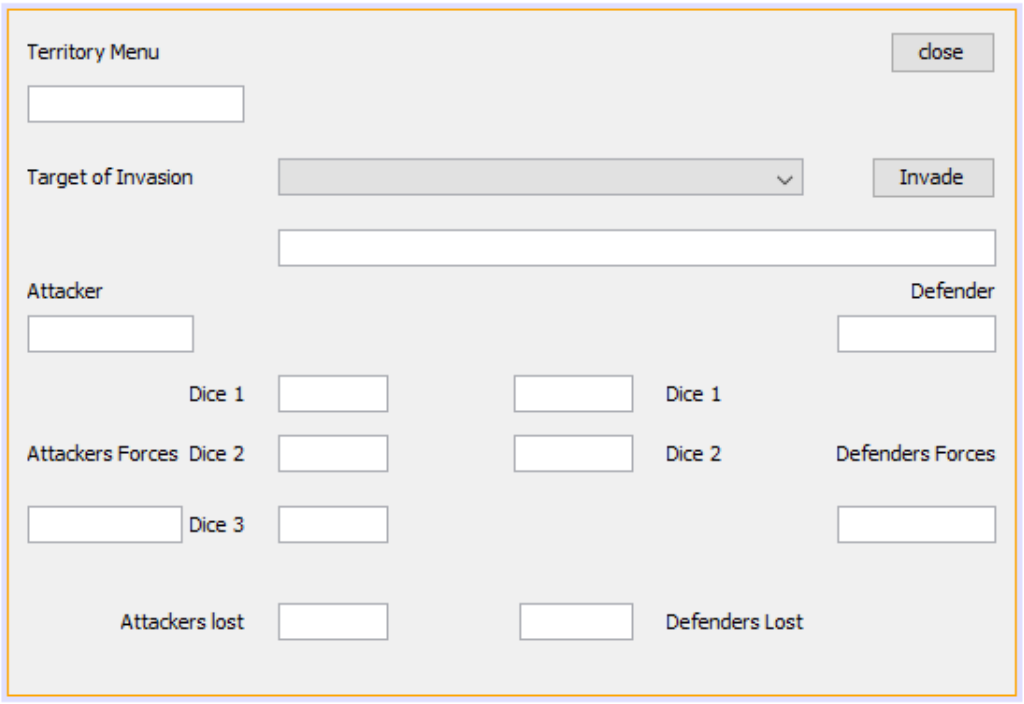
The last main GUI that was implemented was the reinforcement menu. This menu allowed the human players to decide what territories the user can reinforce with soldiers that are generated at the start of the turn. It also displays the possible territories that can be reinforced using a combo box to display them. Along with a number slider that the player can choose the number of soldiers to place on that territory. And lastly two buttons were placed on the menu, one to reinforce the territory once user input has been selected, and one to close the menu.

Images for the GUI screens are found in the Appendix and the image for the Game map and territory Menu is here as it differs slightly from the original design. This is because of the nature of the IDE used to implement these features and other problems with GUI design that will be mentioned in the problems section of the report.

Game GUI screen:



Territory Menu GUI screen:



After the GUI screens were implemented the next stage was to implement the territory class which contains the territoryArray to contain all the information that the territories should have from what territory it is using a for loop to set the values from 0 to 41, what player owns the territory, and the number of soldiers that are currently stationed at each territory.

Once this was set up the next step was to set up the game turn methods, starting with the startGame method which is the method that is called when the player starts the game in the play game GUI. This method collects the settings that are given by the user and sets the number of players and the type of players that will be in the game. After this was completed the next step was calling a function to set what territories the players own, this is determined randomly which is calculated in the calculation class and then checks to see if a player currently owns the selected territory and tries the process again until each territory is distributed evenly among the players in the game. After this process the game sets the game map GUI to visible meaning that the player can now view the game map. Along with a method in the game menu GUI class which sets the territory buttons to the colours that each player controls.

After this process is complete the gameTurn method begins and gives the first player the option to reinforce their territories using the territory GUI and calls the method to calculate the number of soldiers the current player can have depending on the amount of territories and regions they control, along with checking to see if the current player has the correct cards to qualify for acquiring extra soldiers. Once this is complete and the user has distributed their soldiers and close the reinforcement

GUI, the player is now ready to start the conquest aspect of the game. Then if the user ends their turn the end turn button calls the endGame method which checks to see if the win condition has been achieved by the current player and if it is then a small pop up appears stating which player has won along with a button to take the user back to the main menu. If no player has won at the current time then it iterates the current player variable and starts the game turn method again for the next player to take their turn. It also checks to see if the current player has conquered a territory and adds a card to their hand if they did.

Once this stage was accomplished the next focus was performing the battle sequence this starts when the player invades a territory by selecting the territory to invade with, selecting the invasion target, and then clicking the invade button. Once the invade button has been selected it checks to see if the target is owned by the current player and if so the user is unable to invade there. A constraint is also added in case a user selects a territory that they do not control and tries to invade with it then the user will be unable to do so as they do not control that territory.

Once the invade button is pressed and the conditions are met, the battleMethod Is called and this method retrieves the attackers and defenders details and rolls dice according to the number of soldiers on each side of the battle. Then a calculation method is called to compare the dice roll results and calculate the number of casualties for each player. Then these details are sent to the user on the territory GUI, and the lost troops are taken away from the current soldiers in the affected territories in the Array and displays the updated information to the user in the GUI.

If the defenders soldiers reaches zero during this process then the attacker has conquered the province and the control of the territory changes to the invading player and one soldier moves over to the newly conquered territory.

After this the next section of the project that was implemented was the deck class, this contains an array with set values ranging from 0 to 43 with another column of the array to determine which player has that card in their hand. A for loop is then set in the constructor to set 14 of the cards to value 1 (soldier), 14 cards to value 2 (knight), 14 cards to value 3 (cannon), and 2 cards to value 4 (joker). And a method which is performed when a player draws a card, this method chooses a card at random from the deck and checks to see if that card is owned by a player and if it is not then it gives the player that card and if it is then it repeats the random process until a card is given to the player.

The final aspect of implementation that was to be done was the implementation of the parent PlayerClass and the human and AI child classes. The player class is called at the start of the game to determine which players need to be activated in the game depending on what the user’s settings are before the game starts. Once this has been achieved the players are initialised and the number of territories each

player owns is calculated and used for reinforcement calculation and for checking the win condition.

The human and AI classes are very similar except the AI class has the functions that the AI performs whereas the human class just contains information relevant to that human player. The AI class has the functions which determine what its actions are on its turn. Since only one type of AI was implemented in the game all of its non-mandatory functions are chosen at random, when reinforcing the AI has a method which randomly determines one of the territories that is controls and places a random number of reinforcements ranging between 1 and the number of reinforcements it has left to deploy.

Once the AI has completed this function it performs the conquest sequence where it randomly determines a territory to invade from, checking to see if it controls the selected territory and recalling the randomisation method if this is not the case. After this has been done it determines a random valid territory to invade and performs the battleMethod to try and conquer the selected territory. Then the AI will end their turn by calling the end turn method and the game cycle continues to the next AI or human player.

Implementation Problems

During the implementation process several issues and problems occurred due to the sheer size of the program, unoptimized code, limitations due to the used software, and due to time constraints.

The first large problem that occurred was due to the NetBeans IDE [4], this problem occurred as the Game Map GUI was being implemented. The issue that came up was that due to fixed distances that each component on the GUI has, if one component was placed and another was placed within proximity if these fixed distances it would move all other component the minimum distance required so the fixed distances did not intersect. This resulted in an infuriating process where the components would move when another was added resulting in a lot of time taken to place these individual components on the game board. This issue caused a lot of the GUI design time to be lost and this resulted in a knock-on effect where less time was given to other areas of the project as a result of the time lost in this process. Even at the end of the implementation the view card Hand button is not in the correct position on the HUD which is undesirable but means that the other components work as intended without more time being spent on this issue.

The next issue that occurred was due to the auto-generated code the IDE provides as it helped with certain parts to do with the combo-box use, but hindered optimisation of the code as in one instance there exists a 600-line long switch statement that was unavoidable given the nature of the listeners for the JButtons on the GUI. This switch statement has a very simple function as it defines which image is set as the icon for the territory buttons depending on which player controls the

territory. This does mean that this function is unable to be implemented in a more efficient manner given the restrictions of being unable to edit the auto-generated code.

The last major problem that occurred was one that was caused by some un-optimised code that does not affect the way the game runs but does mean that this function cannot be edited without changing a significant portion of the program to fix. This was caused by a switch statement that determined what the selected territories names are, at the start of the implementation this was not considered an issue, until later it was realised that it would have been more efficient and good practice to place these String values into a String Array as this enforces the DRY principle. But by the time that this was realised it was too late and multiple functions depended on this switch statement and it would have taken too long to change this function and the other functions that interact with this switch statement. This has been taken as a point of learning that can aid in the development of future large projects.

A few other minor issues and bugs occurred throughout the process small things such as the AI would often prioritise invading provinces that were recently lost by that AI player even though the choice should be totally random. Another small issue is that the user must select the option in the combo-box each time they wished to interact with a territory for whatever reason, otherwise after the first selection the program would select the first option in the combo box regardless of which option is displayed after being selected by the user. This is an issue that reduces the efficiency of the interaction but does not violate the games function. Lastly the next bug that is in the program, sometimes when the AI chooses a location to invade it will pick a random territory that is not adjacent to any of their owned territories. These minor issues have not been fixed due to time constraints but are not “game breaking” as they seem to only occur under very specific conditions that are currently unknown which is interesting, and leaves room for improvement and learning how these problems occur and how to avoid them in the future.

Design Changes and Justification

The main design changes that were made were to the game map GUI as it was initially intended to display more readily available information to the user to reduce the amount of button clicks that the user must do to view territory information. This was not possible with the IDE that was used to implement the GUI’s as explained in the problems section above. This resulted in a slightly different design which is shown above.

The next main design change was the layout of the territory GUI as it displayed more information than was anticipated at the start of the design stage as it was unsure what information would be displayed on that GUI at the time of specification and design.

In the initial design the AI that was required was an advanced AI that could make decisions based on the current board state comparing the territories adjacent to the ones that were owned by that computer player invading the ones with the weakest ratio of soldiers compared to the AI's deployed troops. This was however decided before implementing the AI that this would not be viable with the given timeframe due to other setbacks in the implementation process. This original AI design was then moved to a desirable requirement and a simpler AI was designed to just make random decisions with parameters stopping the random selection to break the game rules.

Testing Description

This section displays some of the Major parts of testing and a more comprehensive list of the tests can be found in the Appendix.

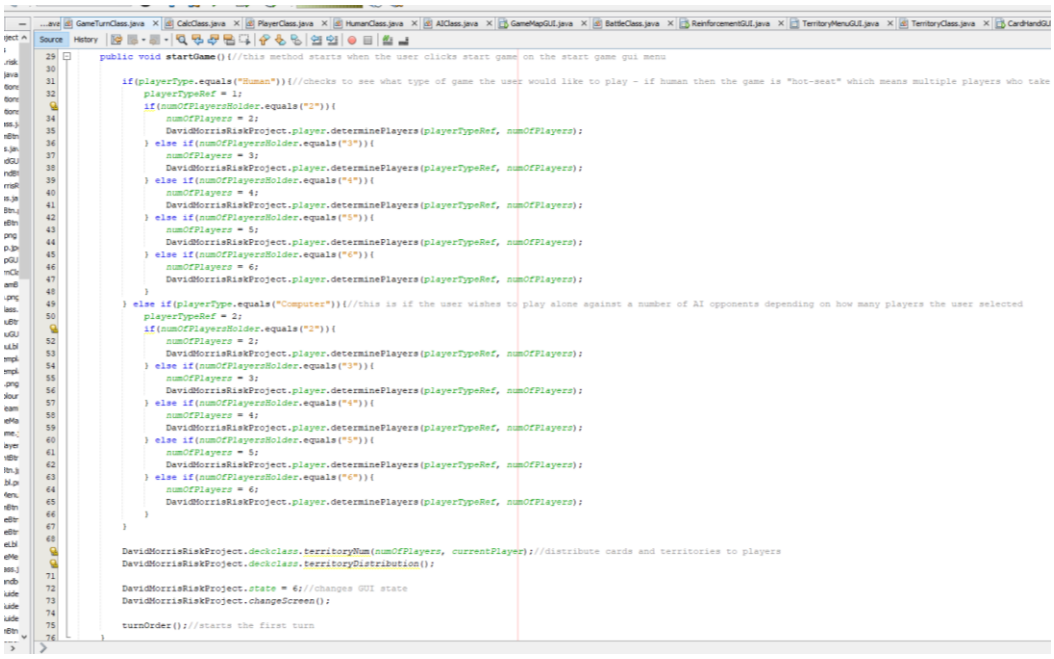
Test	Was the Test Successful?	Comments
1. Can a player invade a Territory?	YES	This Function works as intended.
2. Does the Territory Change control if the attacker reduces defending troops to zero?	YES	This Function works as intended.
3. Are the correct number of troops calculated during the reinforcement step?	YES	This Function works as intended.
4. Does the AI only attack viable territories from its own territories?	NO	The AI does follow the rules most of the time, but a Bug allows the AI to attack non-viable territories in rare circumstances.
5. If someone owns 42 territories do they win the game?	YES	This Function works as intended.

These tests are the Most important as they are all very essential for the Game to function, as one of the tests is not a complete success this leaves room to improve the program and understand the nature of why that test has failed. The remaining test log can be found in the Appendix with more tests that are less crucial to the workings of the program.

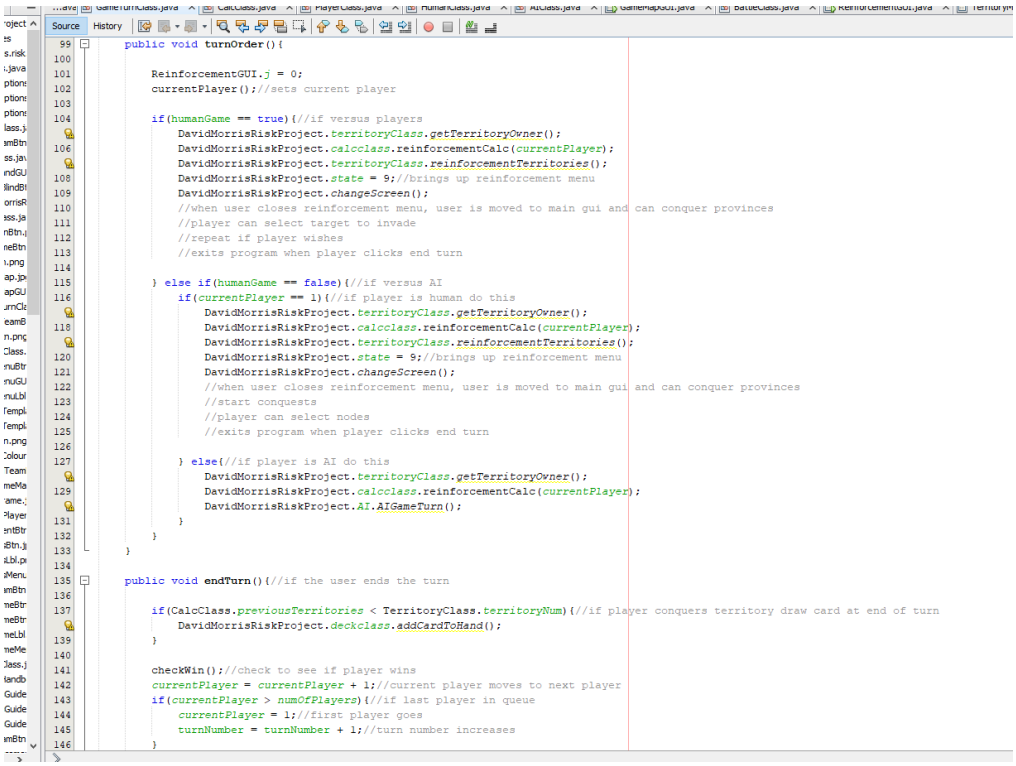
Code and Method Listing

This section displays some of the Major parts of the program code and a brief explanation of what it does, a more comprehensive list of the code can be found in the Appendix.

Game Turn Class:



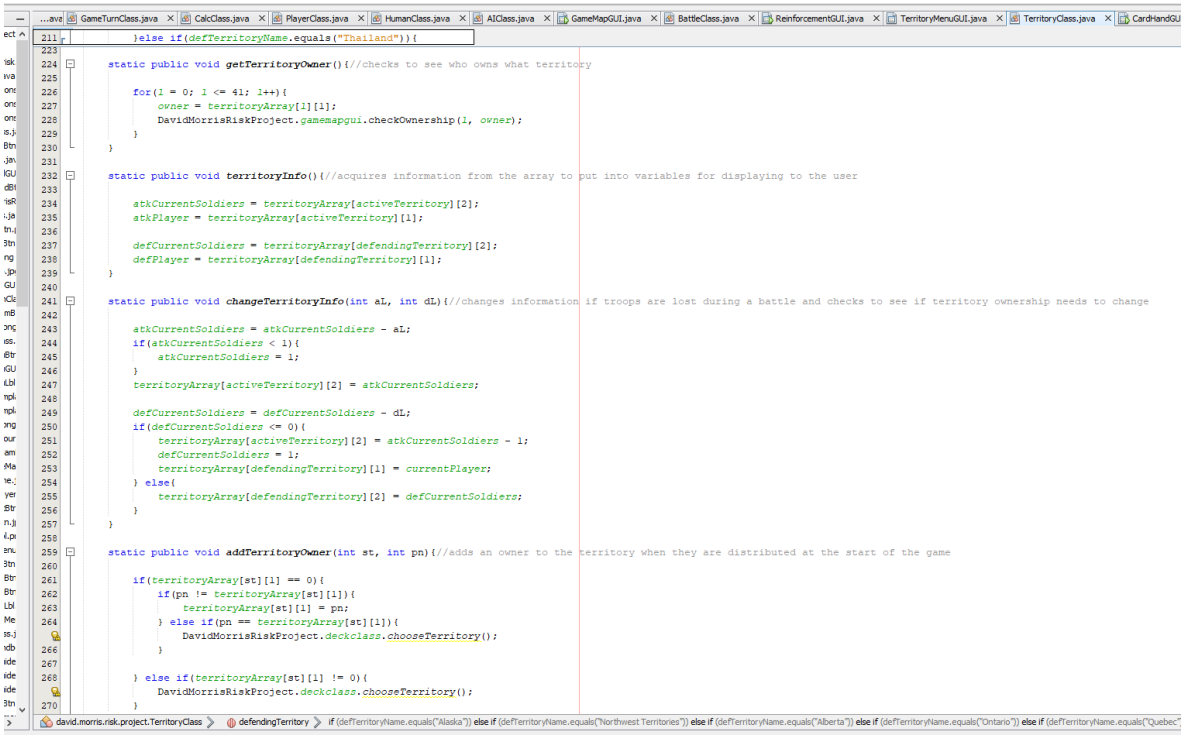
```
public void startGame(){//this method starts when the user clicks start game on the start game gui menu

    if(playerType.equals("Human")){//checks to see what type of game the user would like to play - if human then the game is "hot-seat" which means multiple players who take
        playerTypeRef = 1;
        if(numOfPlayersHolder.equals("2")){
            numOfPlayers = 2;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("3")){
            numOfPlayers = 3;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("4")){
            numOfPlayers = 4;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("5")){
            numOfPlayers = 5;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("6")){
            numOfPlayers = 6;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        }
    } else if(playerType.equals("Computer")){//this is if the user wishes to play alone against a number of AI opponents depending on how many players the user selected
        playerTypeRef = 2;
        if(numOfPlayersHolder.equals("2")){
            numOfPlayers = 2;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("3")){
            numOfPlayers = 3;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("4")){
            numOfPlayers = 4;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("5")){
            numOfPlayers = 5;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        } else if(numOfPlayersHolder.equals("6")){
            numOfPlayers = 6;
            DavidMorrisRiskProject.player.determinePlayers(playerTypeRef, numOfPlayers);
        }
    }

    DavidMorrisRiskProject.deckclass.territoryNum(numOfPlayers, currentPlayer);//distribute cards and territories to players
    DavidMorrisRiskProject.deckclass.territoryDistribution();

    DavidMorrisRiskProject.state = 6;//changes GUI state
    DavidMorrisRiskProject.changeScreen();

    turnOrder();//starts the first turn
}
```



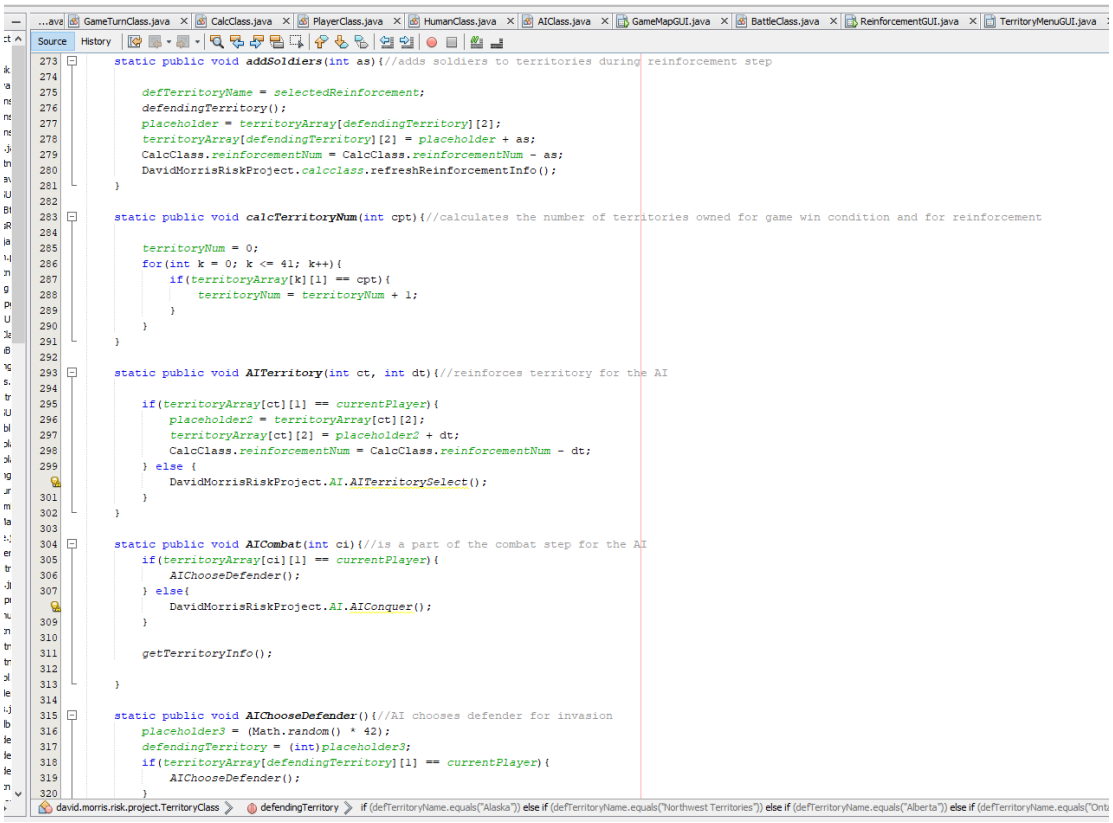
```
public void turnOrder(){

    ReinforcementGUI.j = 0;
    currentPlayer();//sets current player

    if(humanGame == true){//if versus players
        DavidMorrisRiskProject.territoryClass.getTerritoryOwner();
        DavidMorrisRiskProject.calcclass.reinforcementCalc(currentPlayer);
        DavidMorrisRiskProject.territoryClass.reinforcementTerritories();
        DavidMorrisRiskProject.state = 9;//brings up reinforcement menu
        DavidMorrisRiskProject.changeScreen();
        //when user closes reinforcement menu, user is moved to main gui and can conquer provinces
        //player can select target to invade
        //repeat if player wishes
        //exits program when player clicks end turn

    } else if(humanGame == false){//if versus AI
        if(currentPlayer == 1){//if player is human do this
            DavidMorrisRiskProject.territoryClass.getTerritoryOwner();
            DavidMorrisRiskProject.calcclass.reinforcementCalc(currentPlayer);
            DavidMorrisRiskProject.territoryClass.reinforcementTerritories();
            DavidMorrisRiskProject.state = 9;//brings up reinforcement menu
            DavidMorrisRiskProject.changeScreen();
            //when user closes reinforcement menu, user is moved to main gui and can conquer provinces
            //start conquests
            //player can select nodes
            //exits program when player clicks end turn

        } else{//if player is AI do this
            DavidMorrisRiskProject.territoryClass.getTerritoryOwner();
            DavidMorrisRiskProject.calcclass.reinforcementCalc(currentPlayer);
            DavidMorrisRiskProject.AI.AIGameTurn();
        }
    }
}

public void endTurn(){//if the user ends the turn

    if(CalcClass.previousTerritories < TerritoryClass.territoryNum){//if player conquers territory draw card at end of turn
        DavidMorrisRiskProject.deckclass.addCardToHand();
    }

    checkWin();//check to see if player wins
    currentPlayer = currentPlayer + 1;//current player moves to next player
    if(currentPlayer > numOfPlayers){//if last player in queue
        currentPlayer = 1;//first player goes
        turnNumber = turnNumber + 1;//turn number increases
    }
```

These screenshots display the startGame method, the TurnOrder method, and the EndTurn method as these are the main methods in the GameTurn Class.

Territory Class:



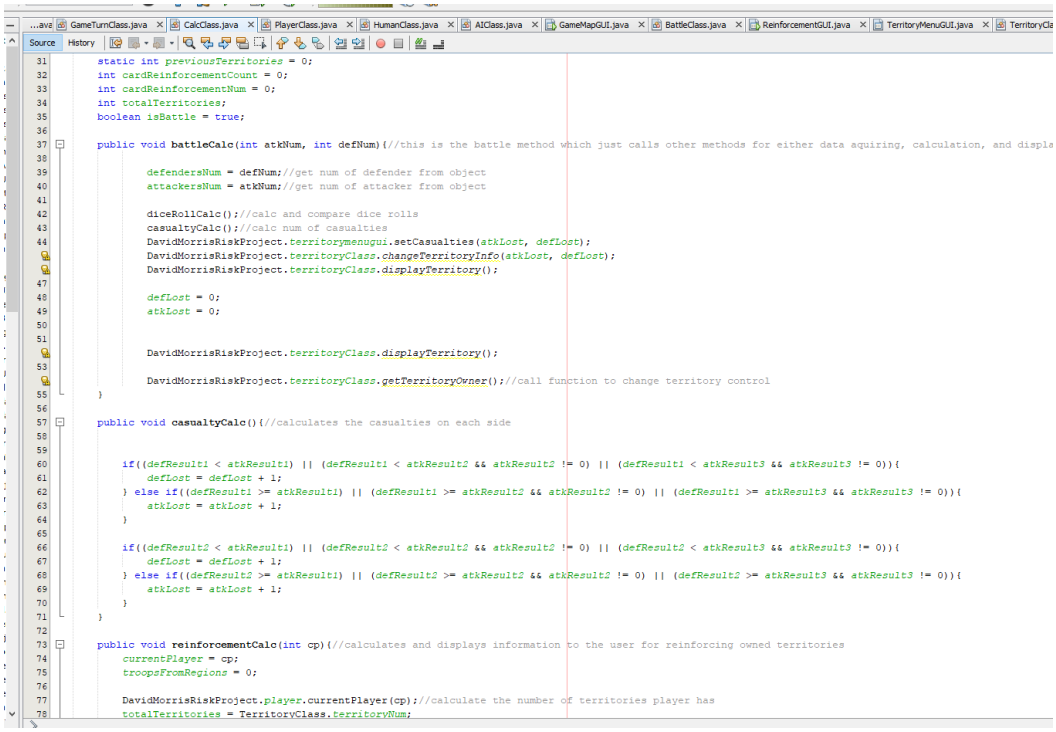
```
}else if(defTerritoryName.equals("Thailand")){

static public void getTerritoryOwner(){//checks to see who owns what territory

    for(i = 0; i <= 41; i++){
        owner = territoryArray[i][1];
        DavidMorrisRiskProject.gamemapgui.checkOwnership(i, owner);
    }
}

static public void territoryInfo(){//acquires information from the array to put into variables for displaying to the user

    atkCurrentSoldiers = territoryArray[activeTerritory][2];
    atkPlayer = territoryArray[activeTerritory][1];

    defCurrentSoldiers = territoryArray[defendingTerritory][2];
    defPlayer = territoryArray[defendingTerritory][1];
}

static public void changeTerritoryInfo(int aL, int dL){//changes information if troops are lost during a battle and checks to see if territory ownership needs to change

    atkCurrentSoldiers = atkCurrentSoldiers - aL;
    if(atkCurrentSoldiers < 1){
        atkCurrentSoldiers = 1;
    }
    territoryArray[activeTerritory][2] = atkCurrentSoldiers;

    defCurrentSoldiers = defCurrentSoldiers - dL;
    if(defCurrentSoldiers <= 0){
        territoryArray[activeTerritory][2] = atkCurrentSoldiers - 1;
        defCurrentSoldiers = 1;
        territoryArray[defendingTerritory][1] = currentPlayer;
    } else{
        territoryArray[defendingTerritory][2] = defCurrentSoldiers;
    }
}

static public void addTerritoryOwner(int st, int pn){//adds an owner to the territory when they are distributed at the start of the game

    if(territoryArray[st][1] == 0){
        if(pn != territoryArray[st][1]){
            territoryArray[st][1] = pn;
        } else if(pn == territoryArray[st][1]){
            DavidMorrisRiskProject.deckclass.chooseTerritory();
        }

    } else if(territoryArray[st][1] != 0){
        DavidMorrisRiskProject.deckclass.chooseTerritory();
    }
```



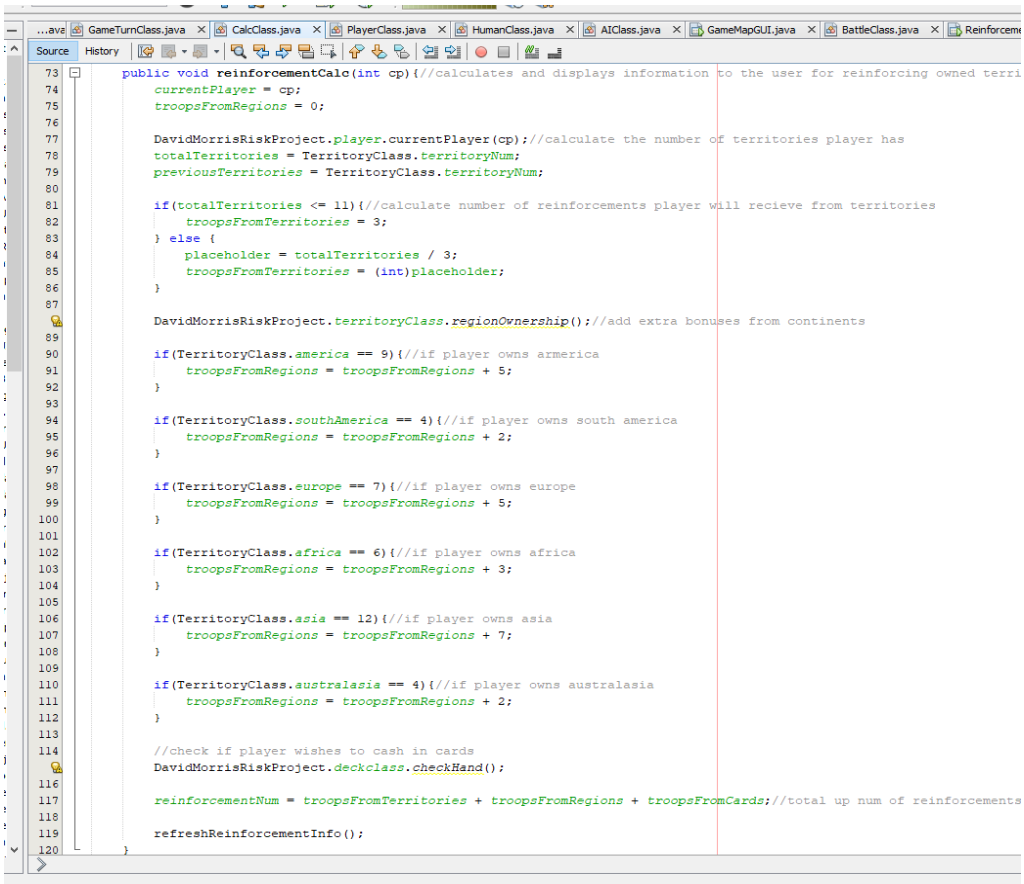
```
static public void addSoldiers(int as){//adds soldiers to territories during reinforcement step

    defTerritoryName = selectedReinforcement;
    defendingTerritory();
    placeholder = territoryArray[defendingTerritory][2];
    territoryArray[defendingTerritory][2] = placeholder + as;
    CalcClass.reinforcementNum = CalcClass.reinforcementNum - as;
    DavidMorrisRiskProject.calcclass.refreshReinforcementInfo();
}

static public void calcTerritoryNum(int cpt){//calculates the number of territories owned for game win condition and for reinforcement

    territoryNum = 0;
    for(int k = 0; k <= 41; k++){
        if(territoryArray[k][1] == cpt){
            territoryNum = territoryNum + 1;
        }
    }
}

static public void AITerritory(int ct, int dt){//reinforces territory for the AI

    if(territoryArray[ct][1] == currentPlayer){
        placeholder2 = territoryArray[ct][2];
        territoryArray[ct][2] = placeholder2 + dt;
        CalcClass.reinforcementNum = CalcClass.reinforcementNum - dt;
    } else {
        DavidMorrisRiskProject.AI.AITerritorySelect();
    }
}

static public void AICombat(int ci){//is a part of the combat step for the AI
    if(territoryArray[ci][1] == currentPlayer){
        AIChooseDefender();
    } else{
        DavidMorrisRiskProject.AI.AIConquer();
    }

    getTerritoryInfo();

}

static public void AIChooseDefender(){//AI chooses defender for invasion
    placeholder3 = (Math.random() * 42);
    defendingTerritory = (int)placeholder3;
    if(territoryArray[defendingTerritory][1] == currentPlayer){
        AIChooseDefender();
    }
```

These images of the Territory Class display the methods for aquiring territory information, modifying the territory information and setting the territory owners from distribution or conquest. The second image shows some of the AI functions that interact with the Territory Array as well to do with territory selection and conquest.

Calculation Class:



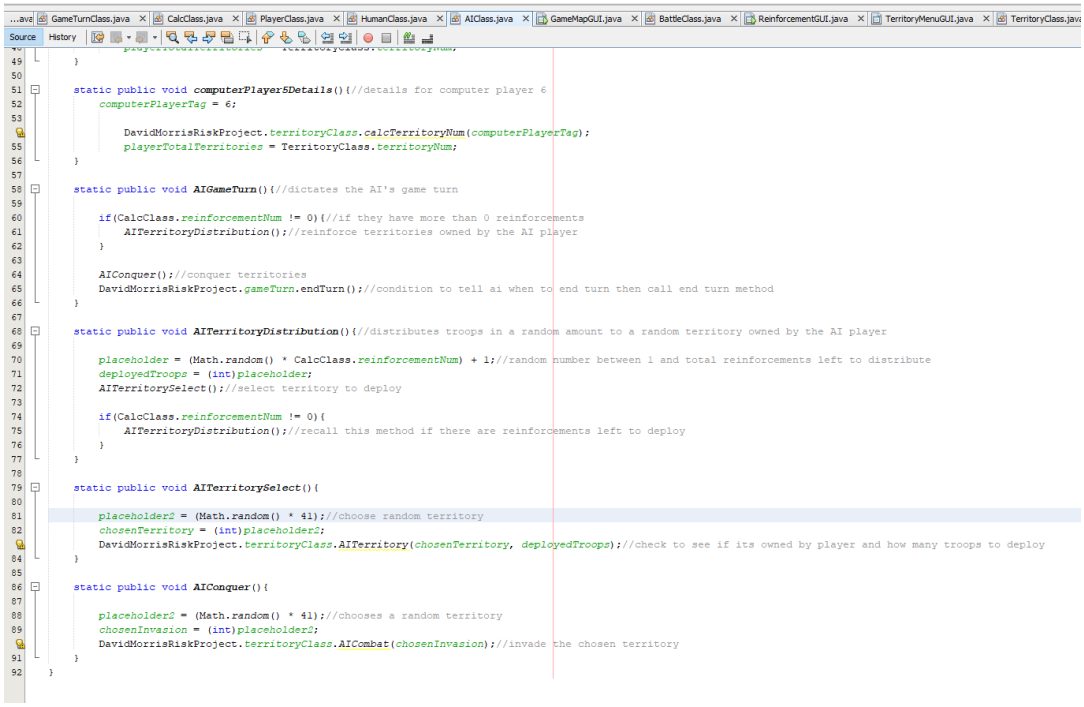
```
31      static int previousTerritories = 0;
32      int cardReinforcementCount = 0;
33      int cardReinforcementNum = 0;
34      int totalTerritories;
35      boolean isBattle = true;
36
37      public void battleCalc(int atkNum, int defNum){//this is the battle method which just calls other methods for either data aquiring, calculation, and displa
38
39              defendersNum = defNum;//get num of defender from object
40              attackersNum = atkNum;//get num of attacker from object
41
42              diceRollCalc();//calc and compare dice rolls
43              casualtyCalc();//calc num of casualties
44              DavidMorrisRiskProject.territorymenugui.setCasualties(atkLost, defLost);
45              DavidMorrisRiskProject.territoryClass.changeTerritoryInfo(atkLost, defLost);
46              DavidMorrisRiskProject.territoryClass.displayTerritory();
47
48              defLost = 0;
49              atkLost = 0;
50
51
52              DavidMorrisRiskProject.territoryClass.displayTerritory();
53
54              DavidMorrisRiskProject.territoryClass.getTerritoryOwner();//call function to change territory control
55      }
56
57      public void casualtyCalc(){//calculates the casualties on each side
58
59
60          if((defResult1 < atkResult1) || (defResult1 < atkResult2 && atkResult2 != 0) || (defResult1 < atkResult3 && atkResult3 != 0)){
61              defLost = defLost + 1;
62          } else if((defResult1 >= atkResult1) || (defResult1 >= atkResult2 && atkResult2 != 0) || (defResult1 >= atkResult3 && atkResult3 != 0)){
63              atkLost = atkLost + 1;
64          }
65
66          if((defResult2 < atkResult1) || (defResult2 < atkResult2 && atkResult2 != 0) || (defResult2 < atkResult3 && atkResult3 != 0)){
67              defLost = defLost + 1;
68          } else if((defResult2 >= atkResult1) || (defResult2 >= atkResult2 && atkResult2 != 0) || (defResult2 >= atkResult3 && atkResult3 != 0)){
69              atkLost = atkLost + 1;
70          }
71      }
72
73      public void reinforcementCalc(int cp){//calculates and displays information to the user for reinforcing owned territories
74          currentPlayer = cp;
75          troopsFromRegions = 0;
76
77          DavidMorrisRiskProject.player.currentPlayer(cp);//calculate the number of territories player has
78          totalTerritories = TerritoryClass.territoryNum;
```



```
73      public void reinforcementCalc(int cp){//calculates and displays information to the user for reinforcing owned terri
74          currentPlayer = cp;
75          troopsFromRegions = 0;
76
77          DavidMorrisRiskProject.player.currentPlayer(cp);//calculate the number of territories player has
78          totalTerritories = TerritoryClass.territoryNum;
79          previousTerritories = TerritoryClass.territoryNum;
80
81          if(totalTerritories <= 11){//calculate number of reinforcements player will recieve from territories
82              troopsFromTerritories = 3;
83          } else {
84              placeholder = totalTerritories / 3;
85              troopsFromTerritories = (int)placeholder;
86          }
87
88          DavidMorrisRiskProject.territoryClass.regionOwnership();//add extra bonuses from continents
89
90          if(TerritoryClass.america == 9){//if player owns armerica
91              troopsFromRegions = troopsFromRegions + 5;
92          }
93
94          if(TerritoryClass.southAmerica == 4){//if player owns south america
95              troopsFromRegions = troopsFromRegions + 2;
96          }
97
98          if(TerritoryClass.europe == 7){//if player owns europe
99              troopsFromRegions = troopsFromRegions + 5;
100         }
101
102         if(TerritoryClass.africa == 6){//if player owns africa
103             troopsFromRegions = troopsFromRegions + 3;
104         }
105
106         if(TerritoryClass.asia == 12){//if player owns asia
107             troopsFromRegions = troopsFromRegions + 7;
108         }
109
110         if(TerritoryClass.australasia == 4){//if player owns australasia
111             troopsFromRegions = troopsFromRegions + 2;
112         }
113
114         //check if player wishes to cash in cards
115         DavidMorrisRiskProject.deckclass.checkHand();
116
117         reinforcementNum = troopsFromTerritories + troopsFromRegions + troopsFromCards;//total up num of reinforcements
118
119         refreshReinforcementInfo();
120     }
```

These screenshots of code in the calculation method display the battleCalc and casualtyCalc methods which are used in the battle process. Along with the method for calculating reinforcements for the players.

AI Class:




```
    }

    static public void computerPlayer5Details(){//details for computer player 6
        computerPlayerTag = 6;

            DavidMorrisRiskProject.territoryClass.calcTerritoryNum(computerPlayerTag);
            playerTotalTerritories = TerritoryClass.territoryNum;
    }

    static public void AIGameTurn(){//dictates the AI's game turn

        if(CalcClass.reinforcementNum != 0){//if they have more than 0 reinforcements
            AITerritoryDistribution();//reinforce territories owned by the AI player
        }

        AIConquer();//conquer territories
        DavidMorrisRiskProject.gameTurn.endTurn();//condition to tell ai when to end turn then call end turn method
    }

    static public void AITerritoryDistribution(){//distributes troops in a random amount to a random territory owned by the AI player

        placeholder = (Math.random() * CalcClass.reinforcementNum) + 1;//random number between 1 and total reinforcements left to distribute
        deployedTroops = (int)placeholder;
        AITerritorySelect();//select territory to deploy

        if(CalcClass.reinforcementNum != 0){
            AITerritoryDistribution();//recall this method if there are reinforcements left to deploy
        }
    }

    static public void AITerritorySelect(){

        placeholder2 = (Math.random() * 41);//choose random territory
        chosenTerritory = (int)placeholder2;
        DavidMorrisRiskProject.territoryClass.AITerritory(chosenTerritory, deployedTroops);//check to see if its owned by player and how many troops to deploy
    }

    static public void AIConquer(){

        placeholder2 = (Math.random() * 41);//chooses a random territory
        chosenInvasion = (int)placeholder2;
        DavidMorrisRiskProject.territoryClass.AICombat(chosenInvasion);//invade the chosen territory
    }
}
```

The AI class contains 3 methods for AI function, a reinforcement distribution method, a territory selection method, and the AI invasion method.

Evaluation

Methods of Evaluation

This section is dedicated to the evaluation of the project noting the successful and unsuccessful aspects of the project as well as stating the criteria that will be used to evaluate how and why the project is successful.

Evaluation Criteria

The most important criteria that the project must meet will be the essential requirements that were set at the start of the project as these requirements were detrimental to the workings of the project and set the basic goals that needed to be achieved for the project. As well as these requirements the game also needs to follow its rules as close as possible for it to still be the game that it was intended to be.

Assessment of criteria

Overall the criteria will be assessed by whether the program performs the functions that were set in the requirements stage. Starting with the essential requirements are met and then moving to the desirable and tertiary requirements that the program can perform.

Evaluation Party Involvement

The evaluation shall be completed by Myself as No 3rd party evaluation is necessary since the project is a game and if the game follows the game rules and flows as the board game does then the criteria will be met according to the specifications set at the beginning of the project.

Evaluation of Results

Evaluation of requirements

Checking through the requirements that were set at the start of the project the program seems to meet all the essential requirements, as a GUI was implemented for the user to interact with and play the game. A set of menu GUI's was created for the user to be able to navigate through the initialisation of the program. Even though the AI is not very advanced in the program the essential requirements stated that a basic AI was needed and could be improved upon later to enhance the program. The mathematical calculations of the program are all contained within the calculation class this is to do with the invasion steps as dice need to be "rolled" for the players to conquer territories, as well as calculating the amount of reinforcements to deploy at the start of each players turn. Lastly the players do not see any of the background functions and are displayed the information that is necessary for them to play the game.

Unfortunately, due to set backs and a lack of time the more advanced requirements such as a proactive AI, or Desirable options were not implemented as these additions to the program would have taken too much time to add to the project at the current time, however it could be possible to implement these functions in the future.

Evaluation of outcome

Overall the game itself works as intended as the functions of the rules are followed, and the restrictions of some implementation decisions and software used to create the program did mean that the game itself could be polished and be more user friendly. As users do not receive notifications if other players have performed actions, the game just displays the results of the actions of other players.

Future improvement

With this project during the implementation stage settings and designs were left open to improvement, such as a function to allow the player to select what colour they wished to play as, or implementing an advanced AI to play against, or being able to have different types of AI in one game, or a mix of human players and AI players.

Improvements could be made to raise the standard of the game compared to official computer games like the addition of animations, or music/ambience, or audio and video settings that can change depending on user decisions. These functions were accounted for in the design stage but were not considered crucial to the game so were given a lower priority over the games function.

These are just some of the things that are available for improvement for the program and have been left open as the program is robust enough that adding a lot of these features would be possible given enough time.

Summary

To conclude, the program has fulfilled the basic requirements that were set for the game and its rules to be followed, as well as leaving room for improvement and addition to the program and its functions.

Aspects of Project Evaluation

NetBeans

A large aspect of the weaknesses of the project was the choice of development environment as the NetBeans IDE [4] was chosen for use during this project as the project was decided to be developed in java and so it made sense that an IDE to suit the chosen language would be a wise decision to help during the implementation stage. The IDE did provide some positive aspects for the development such as having listeners in the auto-generated code and having useful features for designing the smaller GUI screens and its ability to easily import external images so set icons for the various GUI designs.

However the IDE drastically reduced the amount of optimisation that the program had as the auto-generated code was itself a ‘double-edged sword’, meaning that although the code generated was useful in some minor ways it also drastically reduced the amount of code reusability and made implementing the DRY principle impossible to follow in certain areas, this led to unoptimized code and extremely long winded condition statements that were unavoidable given the nature of the IDE, the best example of this is a 600 line switch statement to change the button image for

the territory buttons as there was no other way to override the IDE's methods of changing image icons for certain UI elements. This along with another large switch statement in the program that set the options for the combo boxes was unavoidable because of the workings of NetBeans and its components.

The game map GUI for the program was also extremely difficult to implement using the IDE as the GUI was larger than the other menu's, as well as the amount of interactable buttons on the GUI. It appears that in the NetBeans [4] GUI editors that each GUI item has a distance around it in vertical and horizontal directions that prevent other items from being within its vicinity. This proved extremely frustrating when one item was added to the GUI and all the other components either broke, relocated, or were deleted from the JFrame, this drastically increased the amount of time spent creating and editing the GUI screens. Considering the amount of game objects that would've been needed in hindsight it would've been a better decision to use a game engine like the Unity Game Engine [13] and use an object-oriented language such as C# or C++ as they are better suited to game design than Java. This would've also meant that the game objects could've held information meaning that the individual territories would have the information unique to them rather than having an array hold the information and to constantly reference the array each time something needed to be checked.

Given the nature of the project during the design stage the original design intended to use a while loop for the main game turn iterating the player turns until a winner was decided but however when using Java GUI's it was impossible to implement while loops for the program, because every time the program entered the loop the user would be unable to interact with the UI as the program would be stuck in a loop and would not accept user input until the loop ends, which in this instance required user input to exit the loop, resulting in an infinite loop that the user cannot escape unless they close the program manually.

Time-management

Another aspect that hindered the projects development especially near the beginning of the implementation was that due to circumstance the front-end GUI's took a lot longer to implement than expected and this had a knock-on effect throughout the project, as this meant that it was difficult to follow the timetable set at the start of this project using the Gantt chart.

This pushed back implementation back by a couple of weeks and resulted in some of the more Minor GUI's having a very simple design as no more time could have been dedicated to the GUI designs as other aspects of the program were given higher priority as the duration of the project increased.

Player Interaction

One of the Strongest aspects of the project is the user's ability to interact with the right components at the necessary times, with constraints added so that they cannot

make decisions that are against the game rules. This is enforced through the back-end functions that checks the conditions for certain user interactions or calculations. Then this information is displayed to the user using the GUI's and they can make further decisions based on the displayed information.

GUI's

Another Strong aspect of this project are the usability of the GUI screens as all the screens have a somewhat minimalist design, this is intentional as it reduces the risk that the user will navigate the menu's incorrectly or misinterpret any displayed information. As mentioned earlier the IDE did not make advanced GUI implementation easy, but this comment is only relevant for the game map GUI.

Code Layout

The layout of the code itself is one of the strong aspects of the project, each class, method, and variable are all declared in the correct places with indentation of all non-auto-generated code. As each of these components are named appropriately and indented as it should be in a professional environment with comments explaining what the functions do so it is easier for someone to read the code and understand the process that the functions perform.

Learning Points

Throughout the duration of this project It must be said that there were obstacles that needed to be overcome, whether it was having difficulty using the software to create the program, or lacking certain programming skills, or having trouble balancing the workload between different aspects of the project fairly. However, even though setbacks did occur, I learnt from these experiences and from my mistakes during this project.

One of the most important learning points that I learnt was that it is important to balance the amount of time dedicated to the development of different components of the program. As Initially I spent a lot of time working on the design of the GUI's and the game map, and due to a lack of knowledge of the workings of the IDE's GUI designer. This meant that a lot of time got spent doing one part of the program which meant that other areas of the program are not as optimised or as functional as it could have been.

I also learned how to use an RGE to create, edit, and manipulate images using various tools and resources provided by the editing software, as I think this will be useful experience in the software development industry.

I also learnt certain limitations of the IDE and how auto-generated code can have upsides and downsides, because of this code in the program it has led to optimisation and some code that would be considered "bad practice" that was unavoidable due to the un-editable auto-generated code. The IDE's GUI designer is also very poor when designing a complex GUI with multiple interactable components. This has given me experience using this IDE and as a result I now know how I can avoid some of the issues that comes with the software, as well as understanding different ways that things can be implemented around or using the auto-generated code.

Another lesson that this project has taught me is how it is essential to plan the code for future use in such a large program, as some parts were un-optimised which at the time of creation seemed insignificant, then turned out to cause several problems whether it was an increased amount of un optimised code, or the creation of a redundant part of the program. An example of this in the project is in the territory class, as it is not possible to create an array with different data types, I created an int array for the territory information, and then rather than create a string array to hold the names for the territories I created a switch statement that was 80 lines long that chose the name depending on what value was stored in the array to represent what territory was selected. This along with redundant variables and methods that essentially did the same thing meant that the program could have been optimised in a more efficient manner to increase the robustness of the code as well as enforcing the DRY principle.

Professional Issues

Code of conduct

Looking through the British Computer Society Code of Conduct I have highlighted points that are relevant to the work that I have done and the way it has been implemented. Here are some of the codes that I have followed:

"only undertake to do work or provide a service that is within your professional competence." [11] – The work that I have done for this project is based upon the computational aspects rather than the artistic aspects of the project as I have limited artistic abilities due to my lack of experience in that area. I did however do limited image manipulation and creation as it was necessary for the game to be implemented in this manner.

"develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field." [11] – This has been applied during this project as I have had to learn new techniques and methodologies to aid in completing the project, and as a result my knowledge of the project development process has been expanded.

Code of practice

Looking through the British Computer Society Code of Practice Conduct I have highlighted points that are relevant to the work that I have done and the way it has been implemented. Here are some of the codes that I have followed:

"Strive to produce well-structured code that facilitates testing and maintenance." [10] – I have tried to adhere to this code to the best of my ability when it was possible to do so. The nature of this program and some of the constraints of the software and functions, have limited my ability to achieve this in some areas of the project as I have mentioned earlier in this report.

"Produce code that other programmers will find easy to maintain; use meaningful naming conventions and avoid overly complex programming techniques, where these are not strictly necessary." [10] – throughout the project I have tried to give appropriate names to classes, methods, and variables throughout the project as to more easily identify what each function does. This technique along with code indentation and clear informative comments have allowed the program to be more easily read by other programmers.

"Encourage re-usability; consider the broader applications of your designs and, likewise, before designing from new seek out any existing designs that could be re-used." [10] – I tried to encourage code re-usability where I could, and I did this by having classes that inherited information from parent classes, this was mostly used regarding the player classes, as both the human and AI player classes had information that was relevant for both the types of players to have.

Bibliography

This is the bibliography for this report containing links and information for all aspects of this report that are not my property or external data that has been acquired for reference in this report [8].

[1] Hasbro, Risk board game, https://shop.hasbro.com/en-us/product/risk-game:2C7C6F52-5056-9047-F5DD-EB8AC273BA4C, accessed 26th September 2018.

[2] Softonic, ArgoUML, https://argouml.en.softonic.com , accessed 17th October 2018.

[3] Adobe: Creative, Marketing, and Document management solutions, Photoshop, https://www.photoshop.com/, accessed 9th November 2018.

[4] Oracle Corporation, NetBeans, NetBeans IDE, https://netbeans.org/, accessed and downloaded 26th September 2018.

[5] Paradox interactive, Hearts of Iron IV, https://www.paradoxplaza.com/hearts-of-iron-iv-waking-the-tiger/HIHI04ESK0000010-MASTER.html, accessed 28th October 2018.

[6] 2016 Take-Two Interactive Software and its subsidiaries. Sid Meier's Civilization, Civilization, Civ 2K, Firaxis Games, Sid Meier's Civilisation V, https://civilization.com/civilization-5/, accessed 28th October 2018.

[7] Games Workshop Limited, Relic Entertainment, SEGA, Dawn of War III, https://www.dawnofwar.com/, accessed 28th October 2018.

[8] Ullrich Hustadt, Professional Skills in Computer Science, Department of Computer Science School of Electrical Engineering, Electronics, and Computer Science University of Liverpool, http://cgi.csc.liv.ac.uk/~ullrich/COMP110/notes/Bibliographies_and_Referencing.pdf, accessed 7th May 2019.

[9] Hasbro, Risk Board Image, Google Images, https://www.google.co.uk/search?q=risk+board&safe=off&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi24arRjNXeAhWKXMAKHVaWBuIQ_AUIDigB&biw=958&bih=879#imgrc=zR845MHjY-xW9M:, accessed 10th November 2018.

[10] The British Computer Society Code of Good Practice, https://www.bcs.org/upload/pdf/cop.pdf, accessed 9th May 2019.

[11] CODE OF CONDUCT FOR BCS MEMBERS, https://www.bcs.org/upload/pdf/conduct.pdf, accessed 9th May 2019.

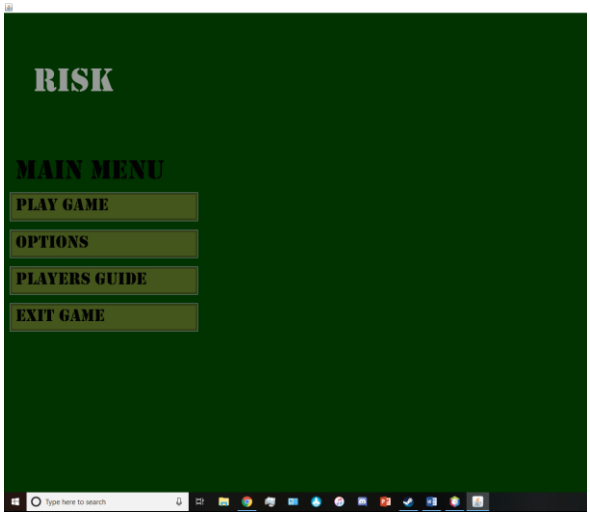
[12] Ubisoft, Risk Game, https://www.ubisoft.com/en-gb/game/risk/, accessed 6th May 2019

[13] Unity Game Engine, https://unity.com/, accessed 7th May 2019.

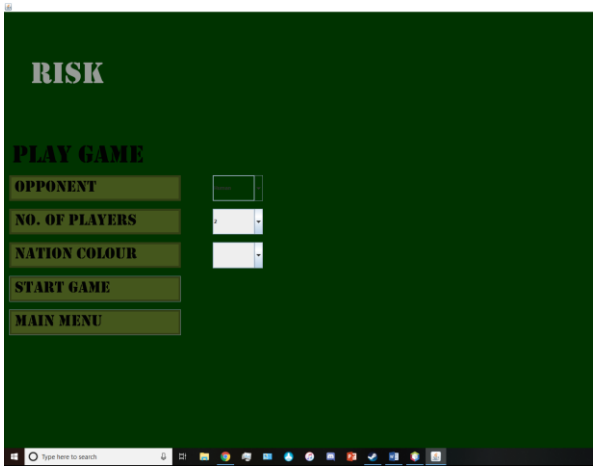
Appendices

Appendix A – GUI Implemented designs

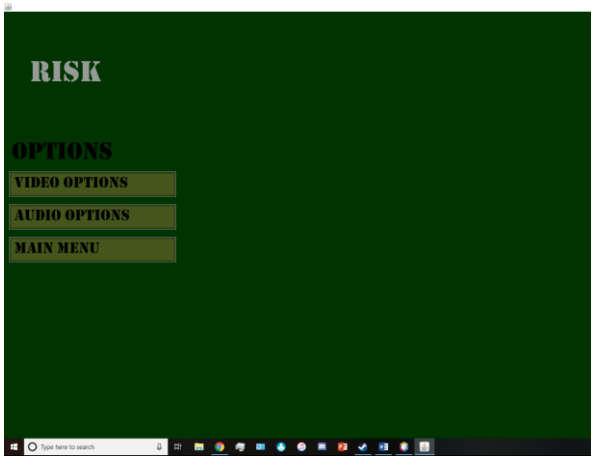
Main Menu GUI



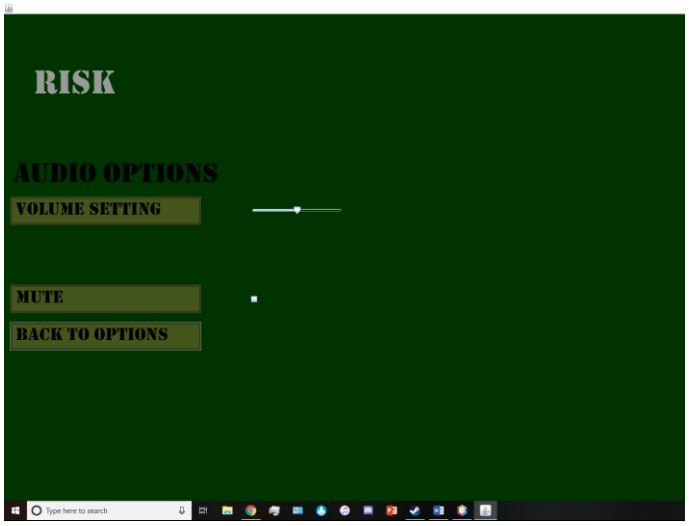
Play Game GUI



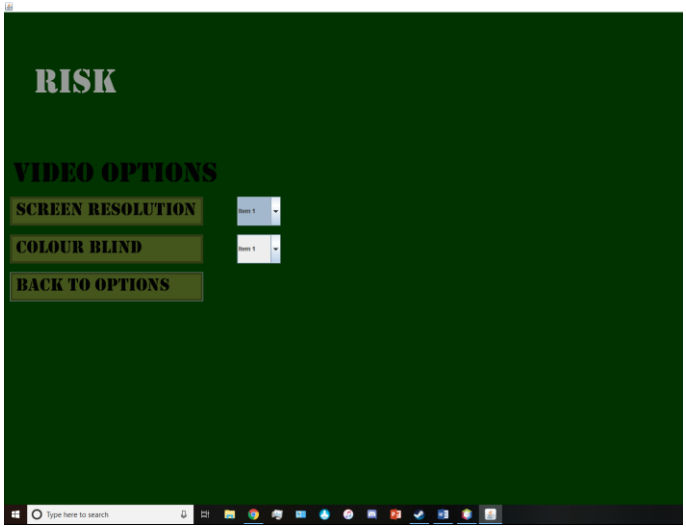
Options GUI



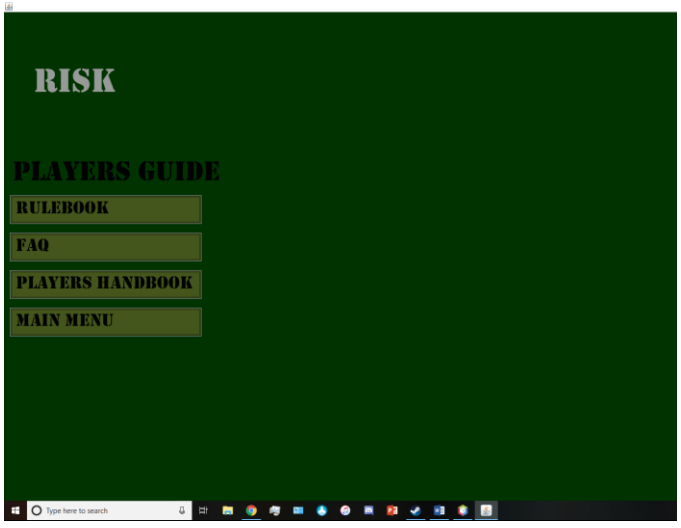
Audio Options GUI



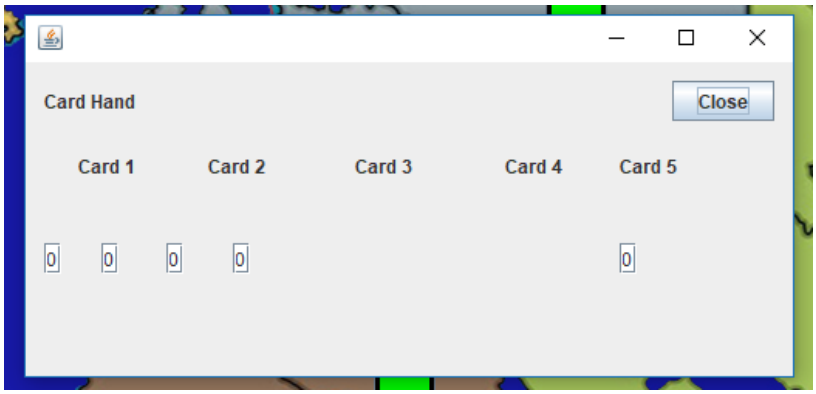
Video Options GUI



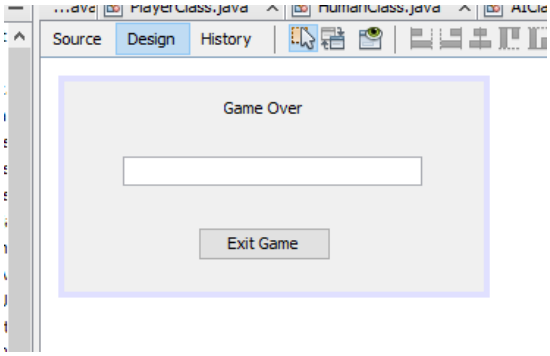
Players Guide GUI



Card Hand GUI



Win Game GUI



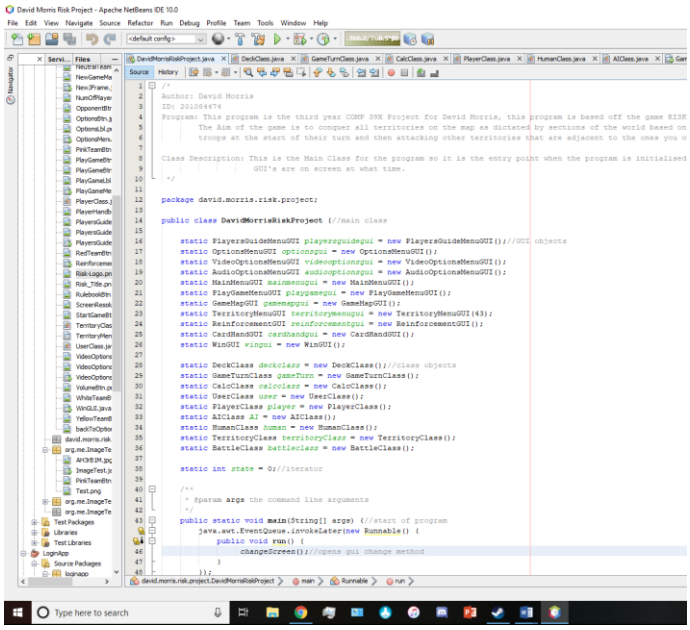
Appendix B – Testing Results

This section of the appendix displays a more detailed testing table:

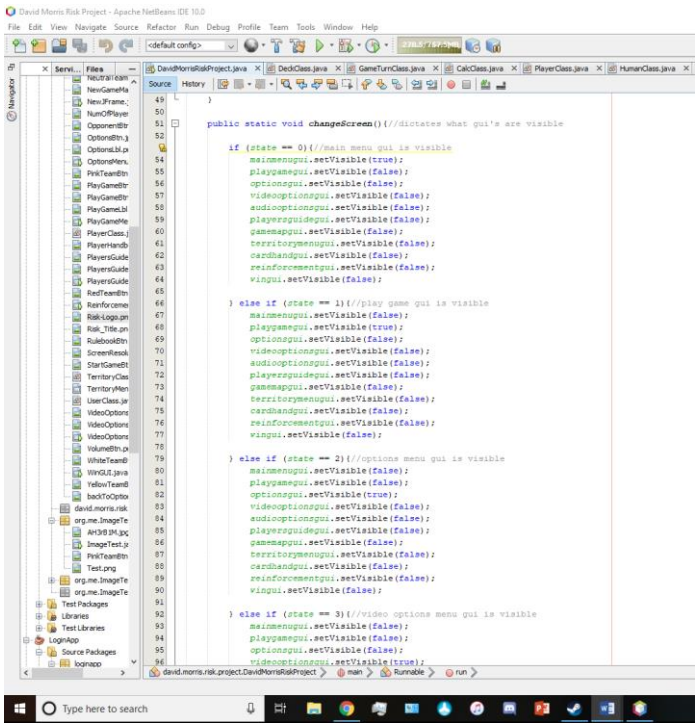
Test	Was the test successful?	Comments
6. Does the game turn iterate through different players?	YES	This Function works as intended.
7. Is the user able to view cards in the card GUI?	NO	
8. Can the user only invade using a territory they control?	YES	This Function works as intended.
9. Does the game disallow invading territories the current player controls?	YES	This Function works as intended.
10. Are the correct troops stationed on the correct territories?	YES	This Function works as intended.
11. Can the game run with multiple human players?	YES	This Function works as intended.
12. Can the game run with multiple AI players?	YES	
13. Do the territory buttons display the correct player's colour?	YES	This Function works as intended.
14. Does the game turn go back to player 1 after the last player ends their turn?	YES	This Function works as intended.
15. Are the dice results representative of a 6 sided die?	YES	This Function works as intended.

Appendix C – Code Screenshots

Main Class

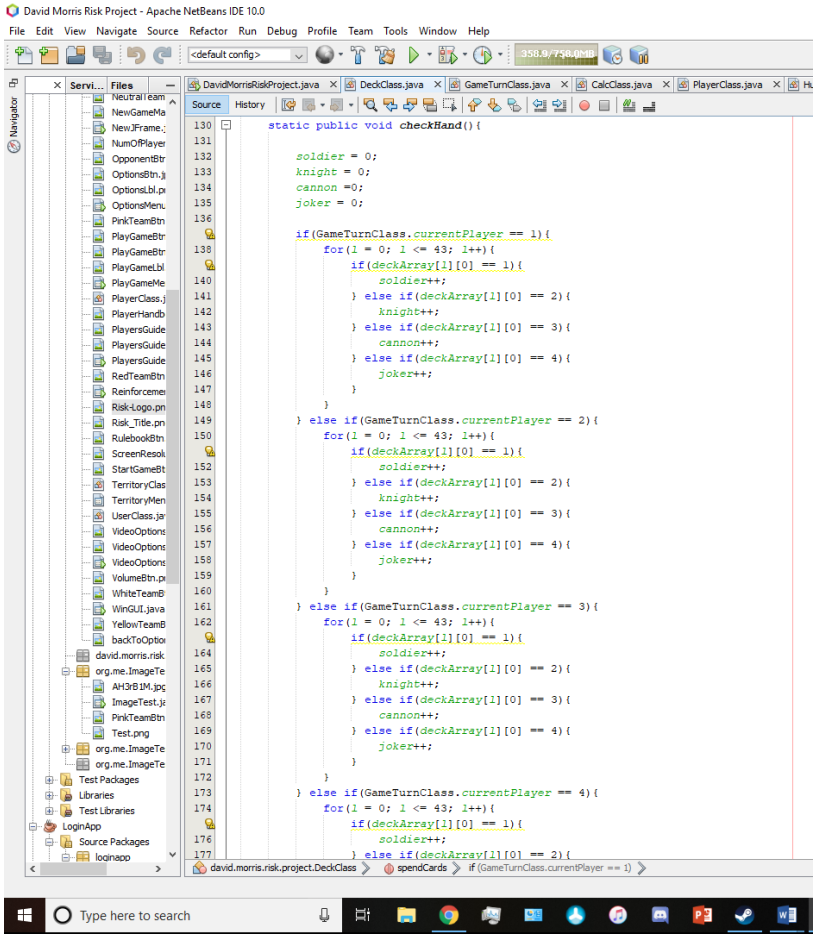


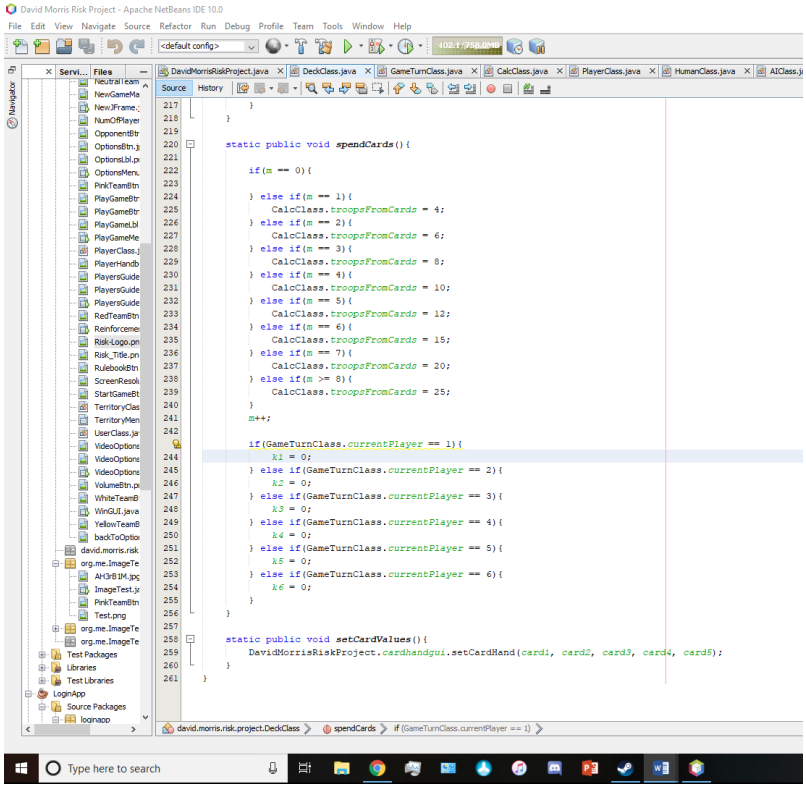
```
/*
 Author: David Morris
 ID: 201004474
 Program: This program is the third year COMP 39X Project for David Morris, this program is based off the game RISK
          The Aim of the game is to conquer all territories on the map as dictated by sections of the world based on
          troops at the start of their turn and then attacking other territories that are adjacent to the ones you o

 Class Description: This is the Main Class for the program so it is the entry point when the program is initialised
                    GUI's are on screen at what time.
 */

package david.morris.risk.project;

public class DavidMorrisRiskProject {//main class

    static PlayersGuideMenuGUI playersguidegui = new PlayersGuideMenuGUI();//GUI objects
    static OptionsMenuGUI optionsgui = new OptionsMenuGUI();
    static VideoOptionsMenuGUI videooptionsgui = new VideoOptionsMenuGUI();
    static AudioOptionsMenuGUI audiooptionsgui = new AudioOptionsMenuGUI();
    static MainMenuGUI mainmenugui = new MainMenuGUI();
    static PlayGameMenuGUI playgamegui = new PlayGameMenuGUI();
    static GameMapGUI gamemapgui = new GameMapGUI();
    static TerritoryMenuGUI territorymenugui = new TerritoryMenuGUI(43);
    static ReinforcementGUI reinforcementgui = new ReinforcementGUI();
    static CardHandGUI cardhandgui = new CardHandGUI();
    static WinGUI wingui = new WinGUI();

    static DeckClass deckclass = new DeckClass();//class objects
    static GameTurnClass gameTurn = new GameTurnClass();
    static CalcClass calcclass = new CalcClass();
    static UserClass user = new UserClass();
    static PlayerClass player = new PlayerClass();
    static AIClass AI = new AIClass();
    static HumanClass human = new HumanClass();
    static TerritoryClass territoryClass = new TerritoryClass();
    static BattleClass battleclass = new BattleClass();

    static int state = 0;//iterator

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {//start of program
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                changeScreen();//opens gui change method
            }
        });
```



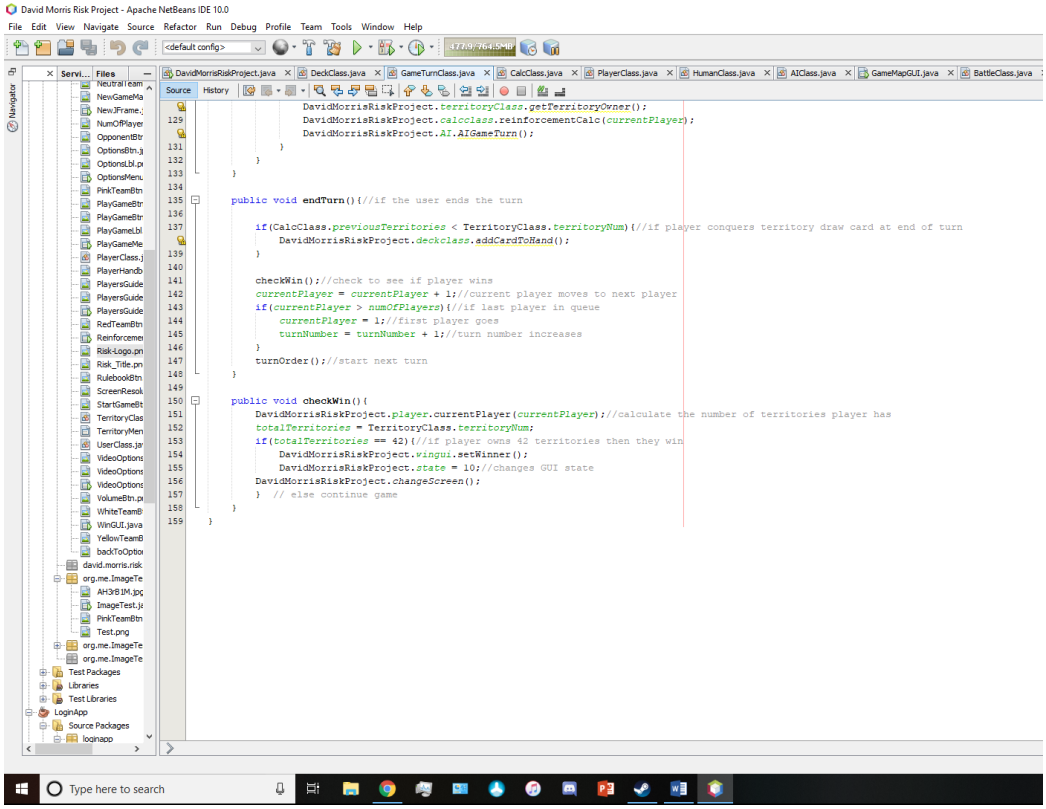
```
    }

    public static void changeScreen(){//dictates what gui's are visible

        if (state == 0){//main menu gui is visible
            mainmenugui.setVisible(true);
            playgamegui.setVisible(false);
            optionsgui.setVisible(false);
            videooptionsgui.setVisible(false);
            audiooptionsgui.setVisible(false);
            playersguidegui.setVisible(false);
            gamemapgui.setVisible(false);
            territorymenugui.setVisible(false);
            cardhandgui.setVisible(false);
            reinforcementgui.setVisible(false);
            wingui.setVisible(false);

        } else if (state == 1){//play game gui is visible
            mainmenugui.setVisible(false);
            playgamegui.setVisible(true);
            optionsgui.setVisible(false);
            videooptionsgui.setVisible(false);
            audiooptionsgui.setVisible(false);
            playersguidegui.setVisible(false);
            gamemapgui.setVisible(false);
            territorymenugui.setVisible(false);
            cardhandgui.setVisible(false);
            reinforcementgui.setVisible(false);
            wingui.setVisible(false);

        } else if (state == 2){//options menu gui is visible
            mainmenugui.setVisible(false);
            playgamegui.setVisible(false);
            optionsgui.setVisible(true);
            videooptionsgui.setVisible(false);
            audiooptionsgui.setVisible(false);
            playersguidegui.setVisible(false);
            gamemapgui.setVisible(false);
            territorymenugui.setVisible(false);
            cardhandgui.setVisible(false);
            reinforcementgui.setVisible(false);
            wingui.setVisible(false);

        } else if (state == 3){//video options menu gui is visible
            mainmenugui.setVisible(false);
            playgamegui.setVisible(false);
            optionsgui.setVisible(false);
            videooptionsgui.setVisible(true);
```

Deck Class

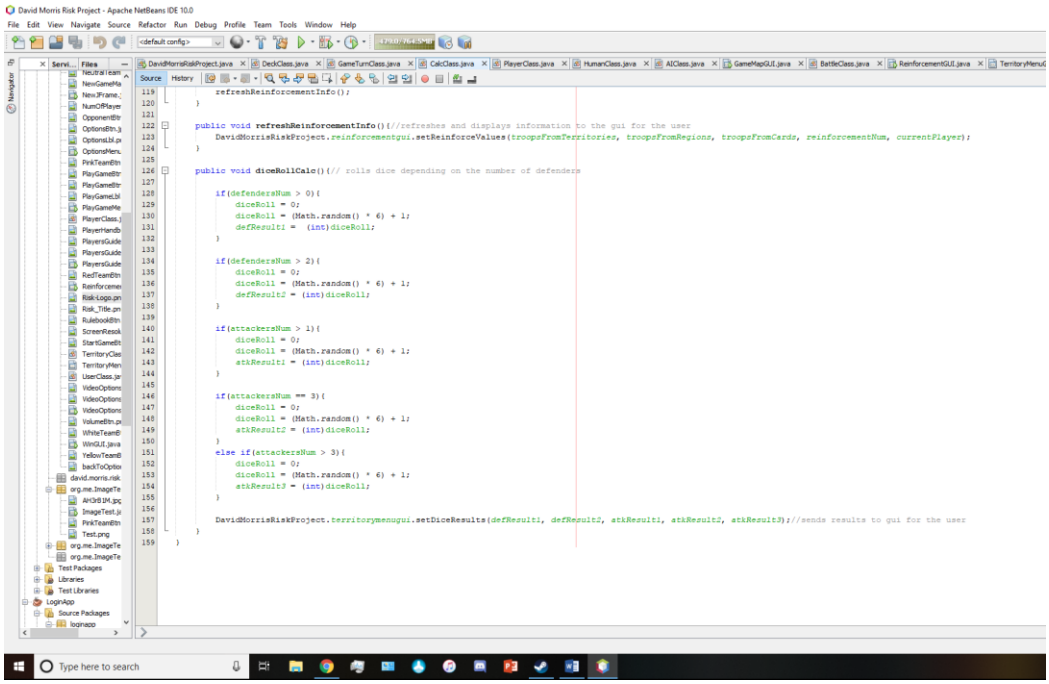

David Morris Risk Project - Apache NetBeans IDE 10.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
DavidMorrisRiskProject.java
DeckClass.java
GameTurnClass.java
CalcClass.java
PlayerClass.java
Source History
static public void checkHand(){
soldier = 0;
knight = 0;
cannon =0;
joker = 0;
if(GameTurnClass.currentPlayer == 1){
for(l = 0; l <= 43; l++){
if(deckArray[l][0] == 1){
soldier++;
} else if(deckArray[l][0] == 2){
knight++;
} else if(deckArray[l][0] == 3){
cannon++;
} else if(deckArray[l][0] == 4){
joker++;
}
}
} else if(GameTurnClass.currentPlayer == 2){
for(l = 0; l <= 43; l++){
if(deckArray[l][0] == 1){
soldier++;
} else if(deckArray[l][0] == 2){
knight++;
} else if(deckArray[l][0] == 3){
cannon++;
} else if(deckArray[l][0] == 4){
joker++;
}
}
} else if(GameTurnClass.currentPlayer == 3){
for(l = 0; l <= 43; l++){
if(deckArray[l][0] == 1){
soldier++;
} else if(deckArray[l][0] == 2){
knight++;
} else if(deckArray[l][0] == 3){
cannon++;
} else if(deckArray[l][0] == 4){
joker++;
}
}
} else if(GameTurnClass.currentPlayer == 4){
for(l = 0; l <= 43; l++){
if(deckArray[l][0] == 1){
soldier++;
} else if(deckArray[l][0] == 2){
david.morris.risk.project.DeckClass
spendCards
if (GameTurnClass.currentPlayer == 1)
Type here to search


David Morris Risk Project - Apache NetBeans IDE 10.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
DavidMorrisRiskProject.java
DeckClass.java
GameTurnClass.java
CalcClass.java
PlayerClass.java
HumanClass.java
AIClass.j
Source History
}
}
static public void spendCards(){
if(m == 0){
} else if(m == 1){
CalcClass.troopsFromCards = 4;
} else if(m == 2){
CalcClass.troopsFromCards = 6;
} else if(m == 3){
CalcClass.troopsFromCards = 8;
} else if(m == 4){
CalcClass.troopsFromCards = 10;
} else if(m == 5){
CalcClass.troopsFromCards = 12;
} else if(m == 6){
CalcClass.troopsFromCards = 15;
} else if(m == 7){
CalcClass.troopsFromCards = 20;
} else if(m >= 8){
CalcClass.troopsFromCards = 25;
}
m++;
if(GameTurnClass.currentPlayer == 1){
k1 = 0;
} else if(GameTurnClass.currentPlayer == 2){
k2 = 0;
} else if(GameTurnClass.currentPlayer == 3){
k3 = 0;
} else if(GameTurnClass.currentPlayer == 4){
k4 = 0;
} else if(GameTurnClass.currentPlayer == 5){
k5 = 0;
} else if(GameTurnClass.currentPlayer == 6){
k6 = 0;
}
}
static public void setCardValues(){
DavidMorrisRiskProject.cardhandgui.setCardHand(card1, card2, card3, card4, card5);
}
}
david.morris.risk.project.DeckClass
spendCards
if (GameTurnClass.currentPlayer == 1)
Type here to search

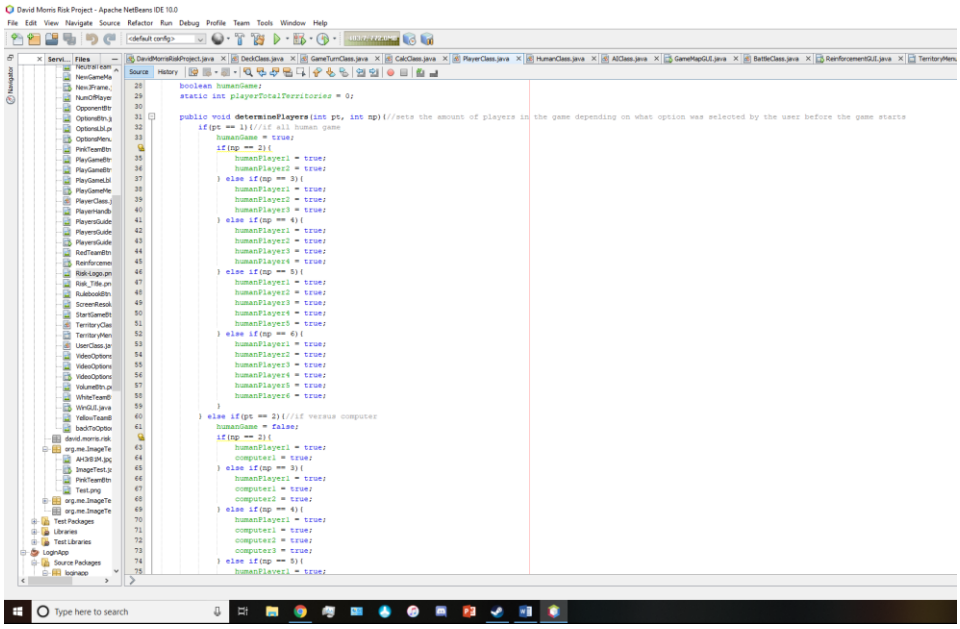
Game Turn Class

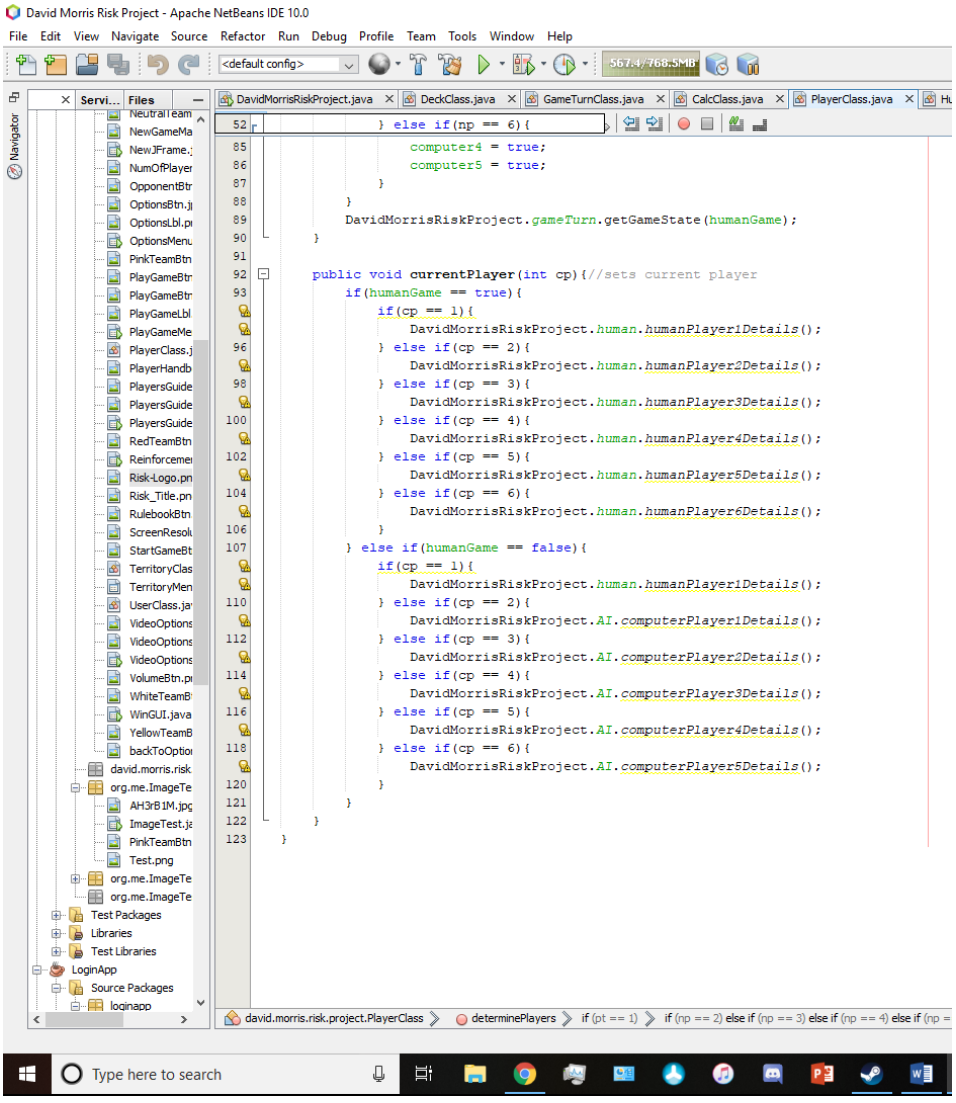


Calculation Class

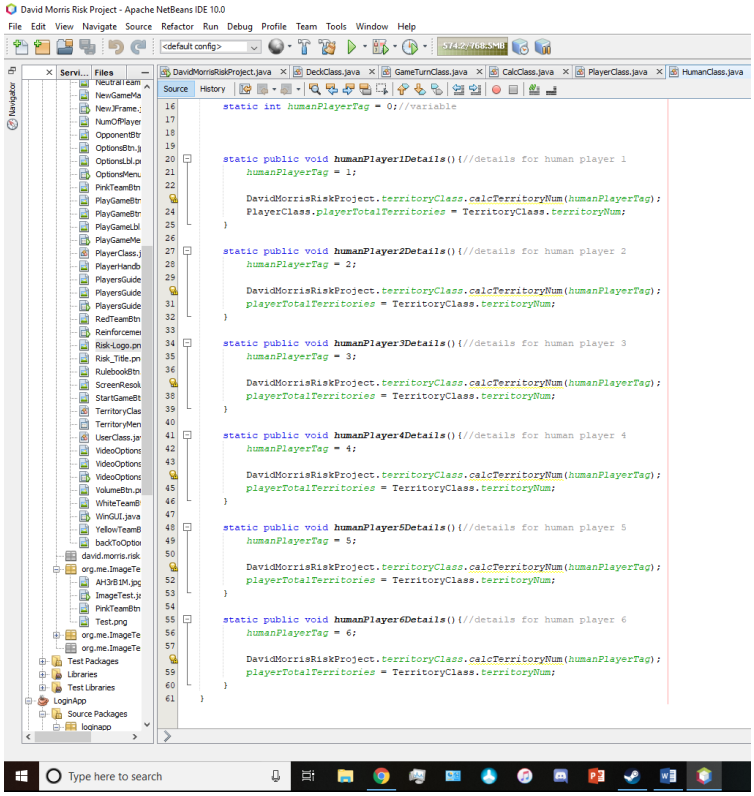


Player Class

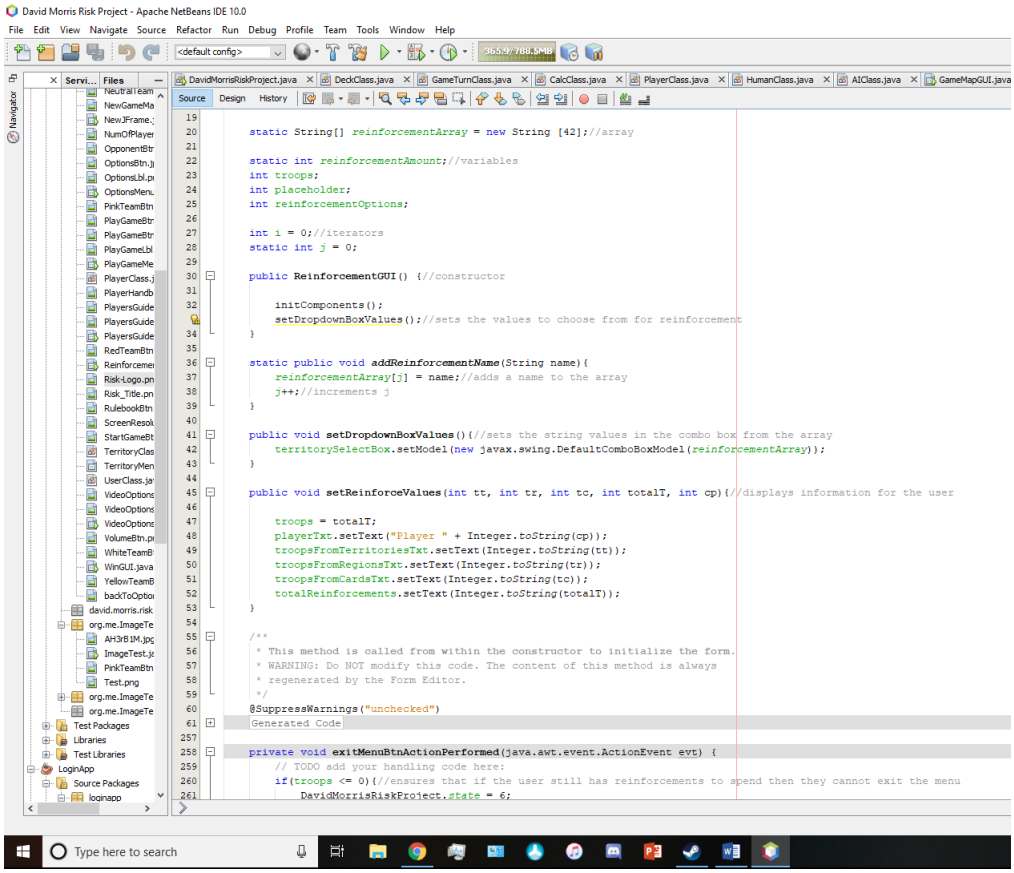




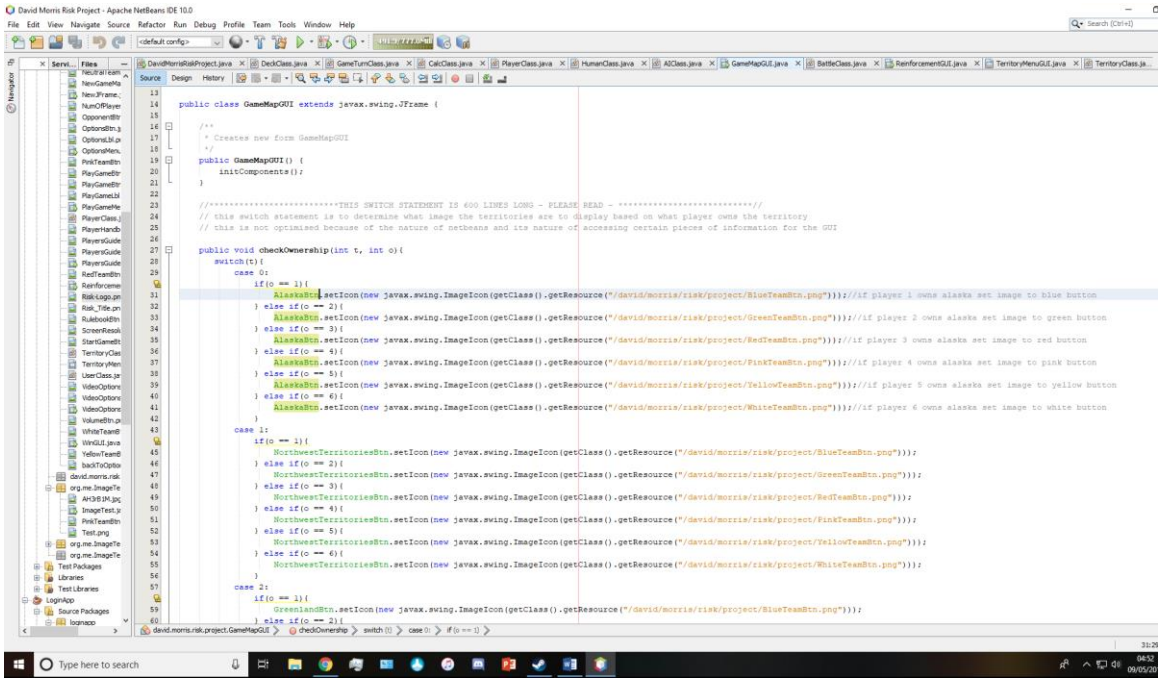
Human Class



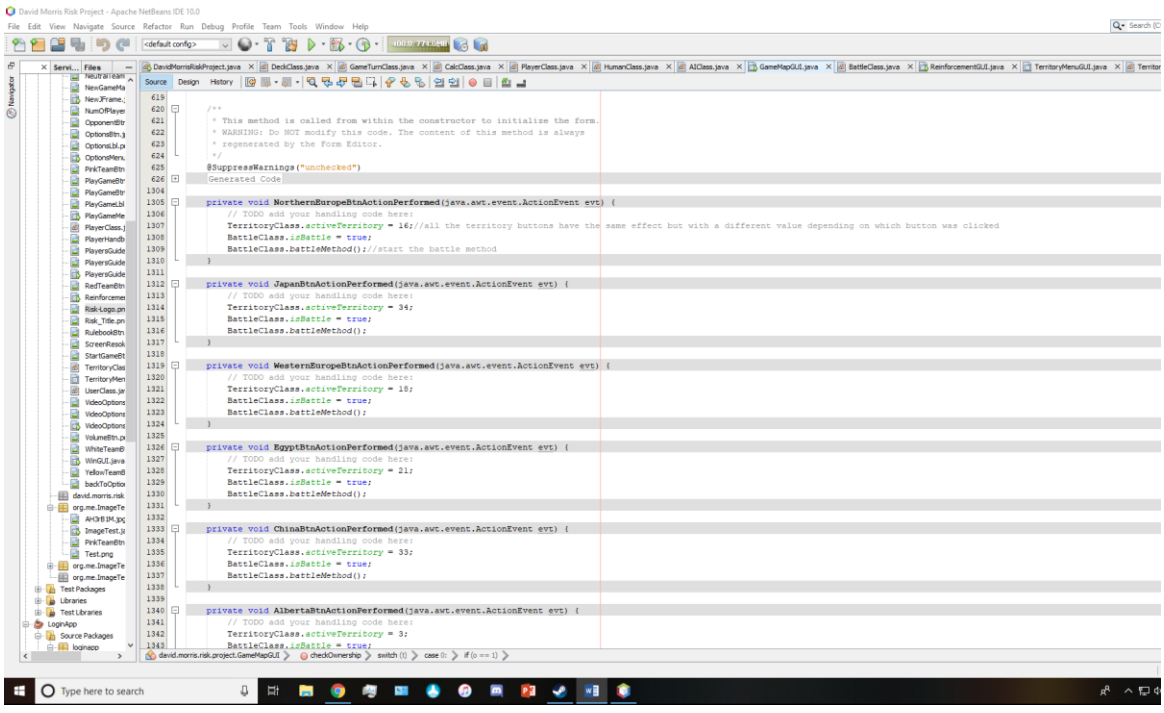
Reinforcement GUI



Game Map GUI Class

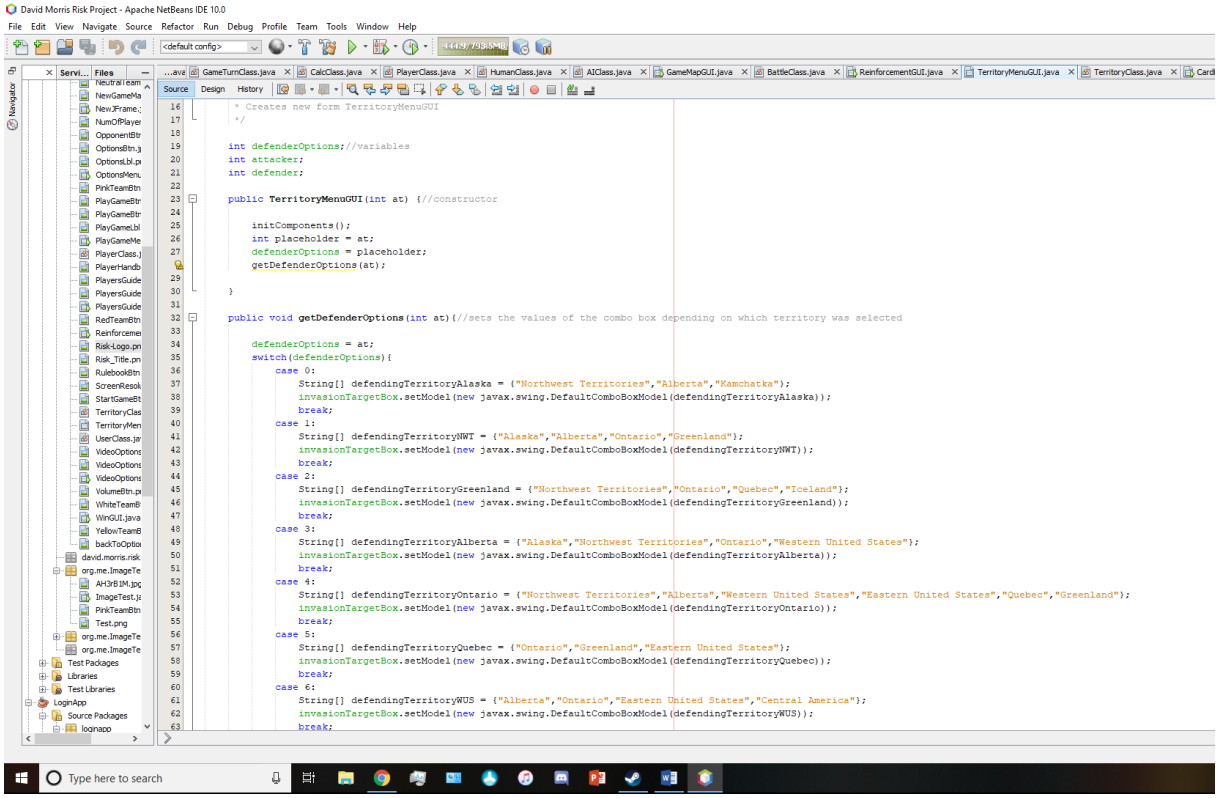


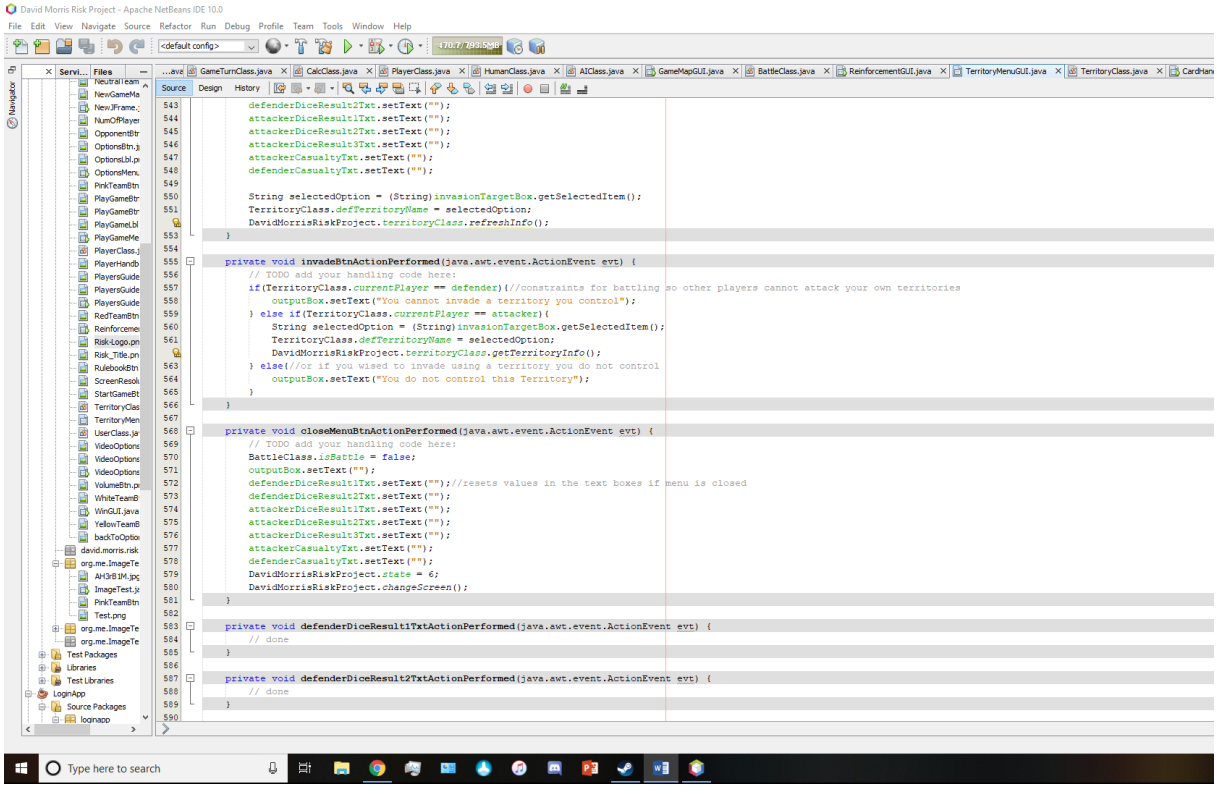
```
public class GameMapGUI extends javax.swing.JFrame {

    /**
     * Creates new form GameMapGUI
     */
    public GameMapGUI() {
        initComponents();
    }

    //*************************THIS SWITCH STATEMENT IS 600 LINES LONG - PLEASE READ - *************************//
    // this switch statement is to determine what image the territories are to display based on what player owns the territory
    // this is not optimised because of the nature of netbeans and its nature of accessing certain pieces of information for the GUI

    public void checkOwnership(int t, int o){
      switch(t){
          case 0:
              if(o == 1){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/BlueTeamBtn.png")));//if player 1 owns alaska set image to blue button
              } else if(o == 2){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/GreenTeamBtn.png")));//if player 2 owns alaska set image to green button
              } else if(o == 3){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/RedTeamBtn.png")));//if player 3 owns alaska set image to red button
              } else if(o == 4){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/PinkTeamBtn.png")));//if player 4 owns alaska set image to pink button
              } else if(o == 5){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/YellowTeamBtn.png")));//if player 5 owns alaska set image to yellow button
              } else if(o == 6){
                  AlaskaBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/WhiteTeamBtn.png")));//if player 6 owns alaska set image to white button
              }
          case 1:
              if(o == 1){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/BlueTeamBtn.png")));
              } else if(o == 2){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/GreenTeamBtn.png")));
              } else if(o == 3){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/RedTeamBtn.png")));
              } else if(o == 4){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/PinkTeamBtn.png")));
              } else if(o == 5){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/YellowTeamBtn.png")));
              } else if(o == 6){
                  NorthwestTerritoriesBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/WhiteTeamBtn.png")));
              }
          case 2:
              if(o == 1){
                  GreenlandBtn.setIcon(new javax.swing.ImageIcon(getClass().getResource("/david/morris/risk/project/BlueTeamBtn.png")));
              } else if(o == 2){
```



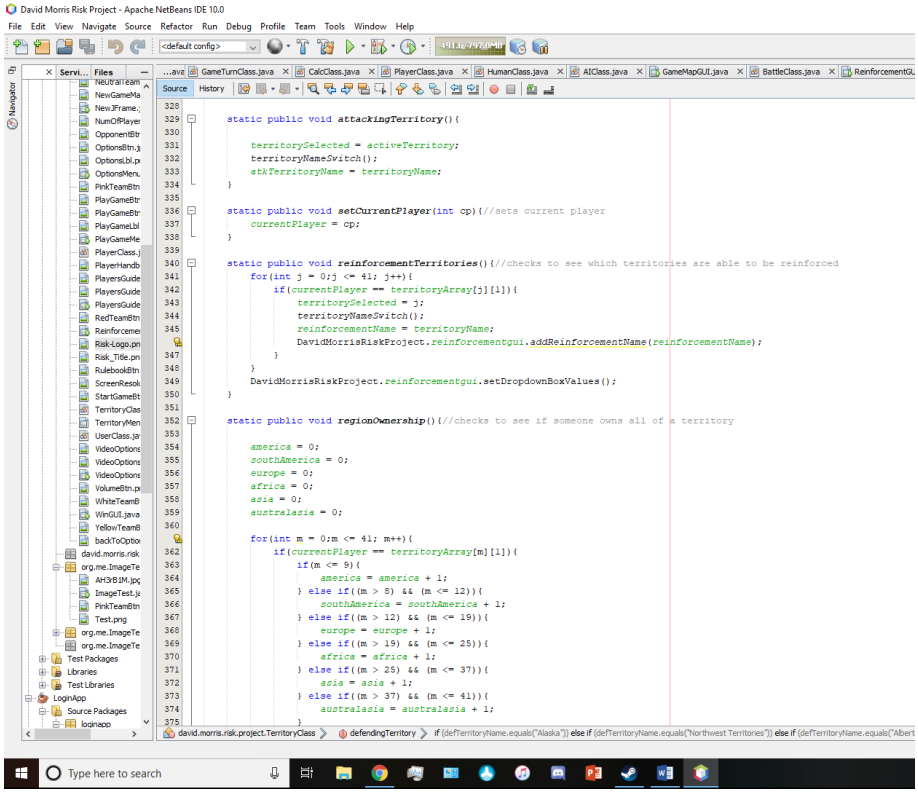
```
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void NorthernEuropeBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 16;//all the territory buttons have the same effect but with a different value depending on which button was clicked
        BattleClass.isBattle = true;
        BattleClass.battleMethod();//start the battle method
    }

    private void JapanBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 34;
        BattleClass.isBattle = true;
        BattleClass.battleMethod();
    }

    private void WesternEuropeBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 18;
        BattleClass.isBattle = true;
        BattleClass.battleMethod();
    }

    private void EgyptBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 21;
        BattleClass.isBattle = true;
        BattleClass.battleMethod();
    }

    private void ChinaBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 33;
        BattleClass.isBattle = true;
        BattleClass.battleMethod();
    }

    private void AlbertaBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        TerritoryClass.activeTerritory = 3;
        BattleClass.isBattle = true;
```

Territory GUI

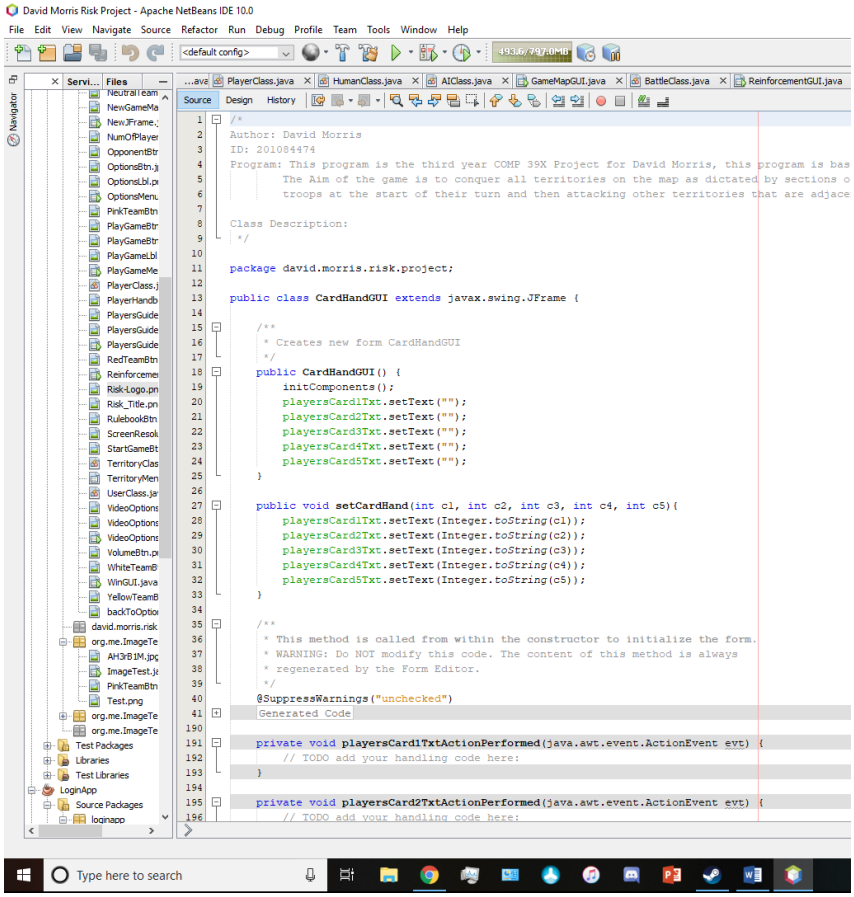




Territory Class



Card Hand GUI Class



Win Game Class