SOEN 6441 - Advanced Programming Practices
Risk Game
Architecture Description[1]

Nasim Adabi[1],Michael Hanna[2],Pinkal Shah[3] and Wasim Alayoubi[4]

[1]40079444 nasim.adabi@gmail.com
[2]40075977 michael_baligh@yahoo.com
[3]40102983 pinkalshah270594@gmail.com
[4]40053306 wasim.alayoubi@mail.concordia.ca

Group ID Group_U_D

1) Introduction - Risk

Risk is a strategy board game of diplomacy, conflict and conquest [1], it was invented by French film director Albert Lamorisse and originally released in 1957 as La Conquête du Monde. In this project, which is a requirement for SOEN 6441 Advanced Programming Practices course for computer science and software engineering graduate students, we attempt to write the known game as a computer program using the Java language and some other tools. The purpose of this project is not to learn how to write Java code, it aims to introduce the team to some industrial techniques for producing commercial software.
To read more about the game, you can visit https://en.wikipedia.org/wiki/Risk_(game)

2) Tools

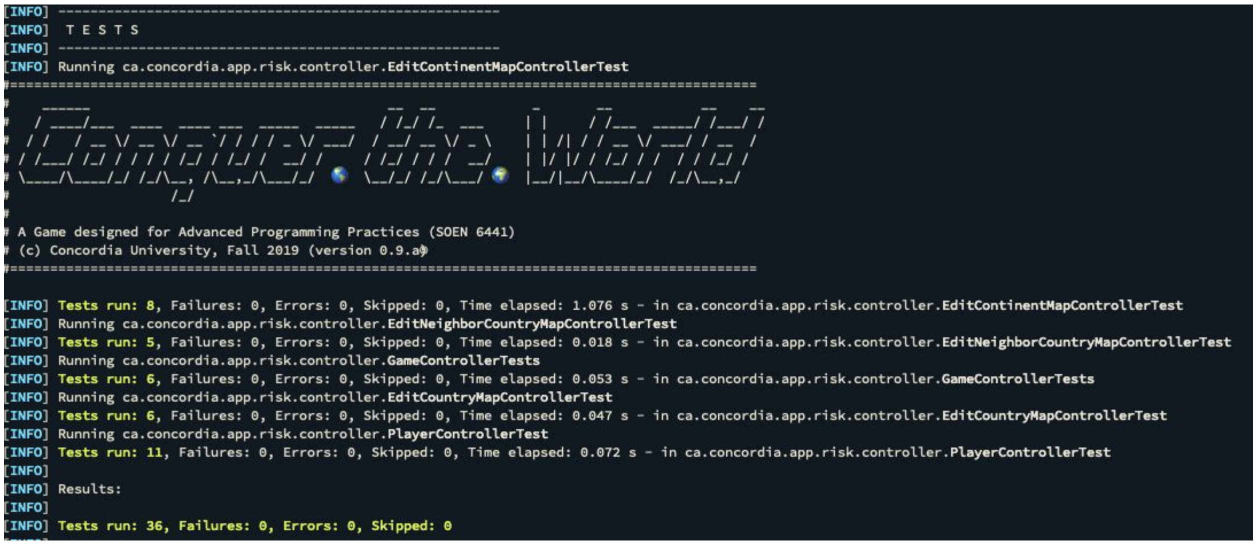
Before going into the implementation details and architecture decision and constraints, we will introduce the process, tools and software that helped us during the course of this project first. We will not go into the details of each and every tool and process, the purpose here is to give the reader an idea what we were using during the project.

a) Unit Testing - Junit 5

JUnit is an open source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. Erich Gamma and Kent Beck initially develop it. As the name implies, it is used for Unit Testing of a small chunk of code[2]. We are using the latest version of JUnit which JUnit 5.

[1] Updated 2019-12-01 (Delivery 3)

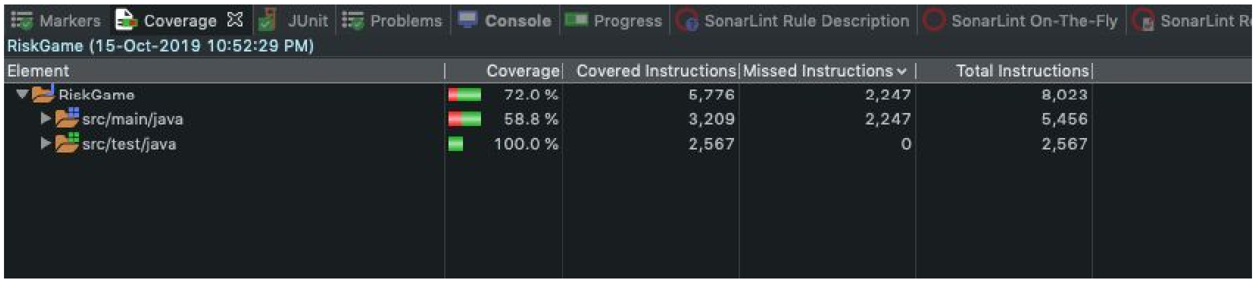
Until the moment of writing this report, we have written 5 Test classes covering 36 test cases and almost 58.5% of the game source as it shows in the following figures



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running ca.concordia.app.risk.controller.EditContinentMapControllerTest
#==================================================================================
#
#    ______                               __  __          _       __            __   __
#   / ____/___  ____  ____ ___  _____  _____   / /_/ /_  ___     | |     / /___  _____/ /___/ /
#  / /   / __ \/ __ \/ __ `/ / / / _ \/ ___/  / __/ __ \/ _ \    | | /| / / __ \/ ___/ / __  /
# / /___/ /_/ / / / / /_/ / /_/ /  __/ /     / /_/ / / /  __/    | |/ |/ / /_/ / /  / / /_/ /
# \____/\____/_/ /_/\__, /\__,_/\___/_/  🌎  \__/_/ /_/\___/ 🌎  |__/|__/\____/_/  /_/\__,_/
#                     /_/
#
# A Game designed for Advanced Programming Practices (SOEN 6441)
# (c) Concordia University, Fall 2019 (version 0.9.a
#==================================================================================

[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.076 s - in ca.concordia.app.risk.controller.EditContinentMapControllerTest
[INFO] Running ca.concordia.app.risk.controller.EditNeighborCountryMapControllerTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.018 s - in ca.concordia.app.risk.controller.EditNeighborCountryMapControllerTest
[INFO] Running ca.concordia.app.risk.controller.GameControllerTests
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.053 s - in ca.concordia.app.risk.controller.GameControllerTests
[INFO] Running ca.concordia.app.risk.controller.EditCountryMapControllerTest
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.047 s - in ca.concordia.app.risk.controller.EditCountryMapControllerTest
[INFO] Running ca.concordia.app.risk.controller.PlayerControllerTest
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.072 s - in ca.concordia.app.risk.controller.PlayerControllerTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 36, Failures: 0, Errors: 0, Skipped: 0
```

Fig 1- Number of Test Cases



RiskGame (15-Oct-2019 10:52:29 PM)

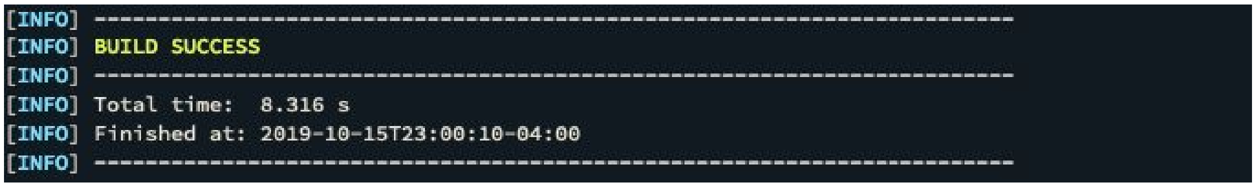
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
RiskGame	72.0 %	5,776	2,247	8,023
src/main/java	58.8 %	3,209	2,247	5,456
src/test/java	100.0 %	2,567	0	2,567

Fig 2 - JUnit Testing Coverage as of milestone one delivery

b) Build Tool - Maven

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies [3].

We are using maven to build, test and package the project, it is also used to manage the third party dependencies of the project.

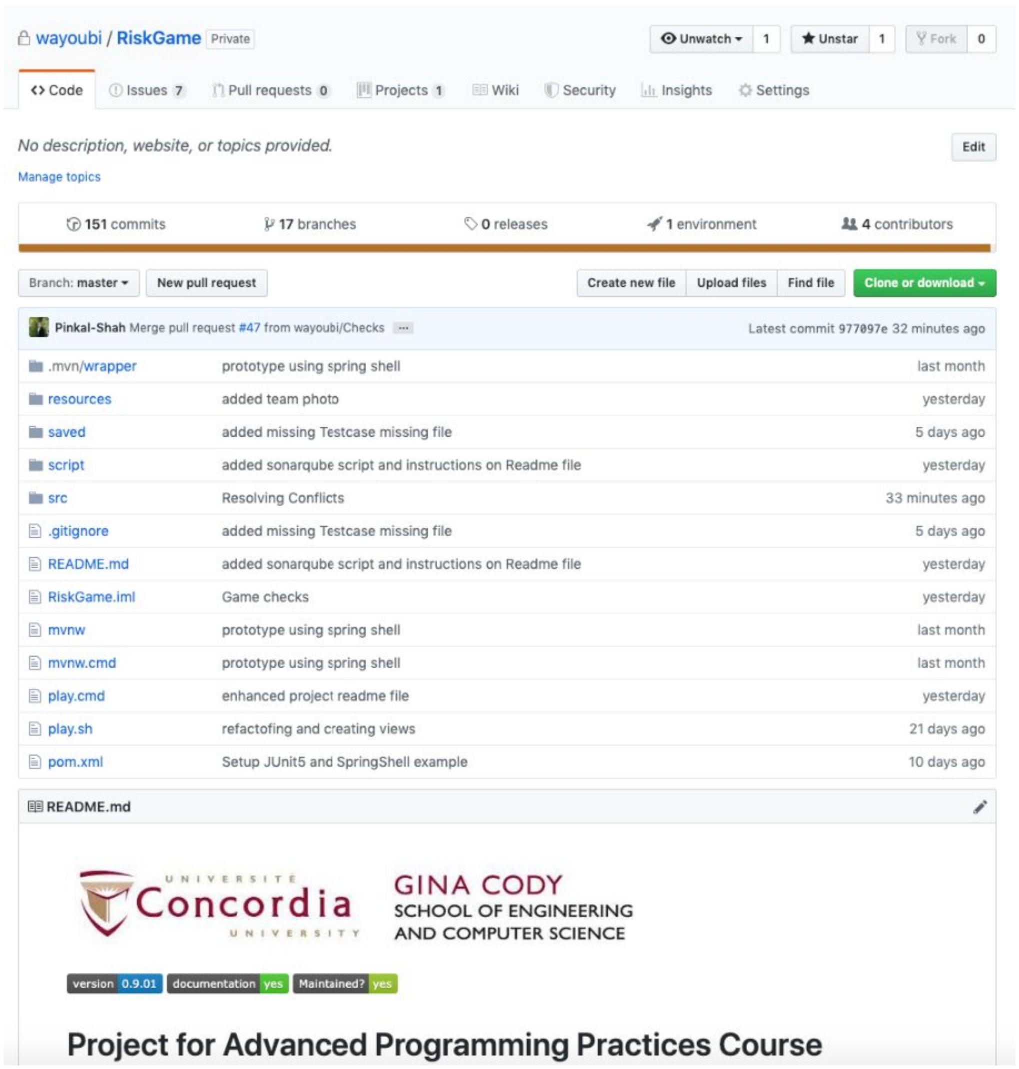


```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.316 s
[INFO] Finished at: 2019-10-15T23:00:10-04:00
[INFO] ------------------------------------------------------------------------
```

Fig 3- Showing Maven build output

c) **Integrated Development Environments - IDEs**
Because of using maven as project building tool, the source code was packaged in a maven project architecture. Maven project structure is supported by all popular Java IDEs including Eclipse and IntelliJ. This gave the team members to use freely whatever IDE they feel comfortable with, some were using eclipse, others intelliJ and some using Visual Studio Code.

d) **Distributed Repository - GitHub**
Git and Github were used for code sharing and collaboration management, Our project is currently private "as required by the course" and available on https://github.com/wayoubi/RiskGame



No description, website, or topics provided.

151 commits | 17 branches | 0 releases | 1 environment | 4 contributors

Pinkal-Shah Merge pull request #47 from wayoubi/Checks … Latest commit 977097e 32 minutes ago

.mvn/wrapper	prototype using spring shell	last month
resources	added team photo	yesterday
saved	added missing Testcase missing file	5 days ago
script	added sonarqube script and instructions on Readme file	yesterday
src	Resolving Conflicts	33 minutes ago
.gitignore	added missing Testcase missing file	5 days ago
README.md	added sonarqube script and instructions on Readme file	yesterday
RiskGame.iml	Game checks	yesterday
mvnw	prototype using spring shell	last month
mvnw.cmd	prototype using spring shell	last month
play.cmd	enhanced project readme file	yesterday
play.sh	refactofing and creating views	21 days ago
pom.xml	Setup JUnit5 and SpringShell example	10 days ago

README.md

UNIVERSITÉ Concordia UNIVERSITY   GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Project for Advanced Programming Practices Course

Fig 4 - Snapshot from Github Repository

e) **Java Docs**
Code documentation is generated using Javadocs and hosted on github on https://wayoubi.github.io/RiskGameDocs/

3) Engineering Process

As a team of 4 developers (we were 5 at the beginning, one guy dropped) and because of the nature of the project like unclear requirements, continuous change, not sure which technologies to use and for the sake of practicing agile, we followed the XP Software development modeling process.

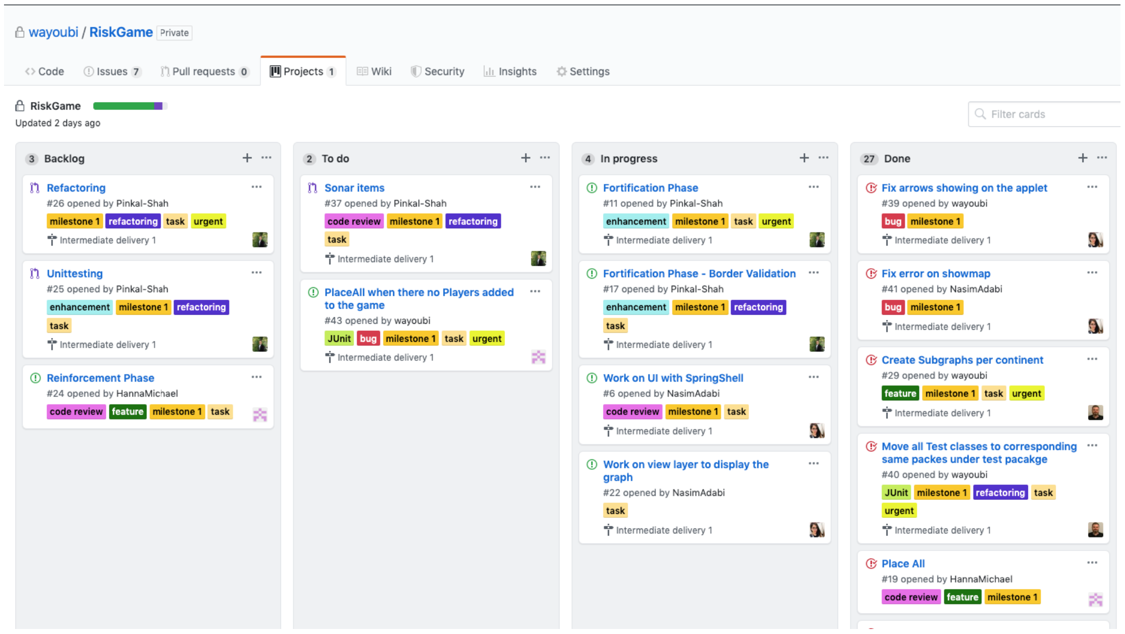
a) eXtreme Programming

During the project, we used the following practices.

i) *Pair Programming*
ii) *Continuous Integration*
iii) *Unit tests.*
iv) *system metaphor.*
v) *Create spike solutions to reduce risk.*
vi) *Refactor*
vii) *collective ownership.*

b) Kanban Board

For managing our tasks and stores, we used Kanban board for simplicity. Actually we were not interested in calculating team velocity and playing the planning game because of the nature of the project. A simple Kanban board was efficient to follow up on our day to day tasks
Board is available on https://github.com/wayoubi/RiskGame/projects/1



RiskGame
Updated 2 days ago

3 Backlog

Refactoring
#26 opened by Pinkal-Shah
milestone 1 refactoring task urgent
Intermediate delivery 1

Unittesting
#25 opened by Pinkal-Shah
enhancement milestone 1 refactoring task
Intermediate delivery 1

Reinforcement Phase
#24 opened by HannaMichael
code review feature milestone 1 task

2 To do

Sonar items
#37 opened by Pinkal-Shah
code review milestone 1 refactoring task
Intermediate delivery 1

PlaceAll when there no Players added to the game
#43 opened by wayoubi
JUnit bug milestone 1 task urgent
Intermediate delivery 1

4 In progress

Fortification Phase
#11 opened by Pinkal-Shah
enhancement milestone 1 task urgent
Intermediate delivery 1

Fortification Phase - Border Validation
#17 opened by Pinkal-Shah
enhancement milestone 1 refactoring task
Intermediate delivery 1

Work on UI with SpringShell
#6 opened by NasimAdabi
code review milestone 1 task
Intermediate delivery 1

Work on view layer to display the graph
#22 opened by NasimAdabi
task
Intermediate delivery 1

27 Done

Fix arrows showing on the applet
#39 opened by wayoubi
bug milestone 1
Intermediate delivery 1

Fix error on showmap
#41 opened by NasimAdabi
bug milestone 1
Intermediate delivery 1

Create Subgraphs per continent
#29 opened by wayoubi
feature milestone 1 task urgent
Intermediate delivery 1

Move all Test classes to corresponding same packes under test pacakge
#40 opened by wayoubi
JUnit milestone 1 refactoring task urgent
Intermediate delivery 1

Place All
#19 opened by HannaMichael
code review feature milestone 1

Fig 5 - Project Kanban Board (Using GitHub)

4) **Domain Model.**

We started the project by defining objects in our Domain Model (The Game), we initially elicited the main objects and the relationships including cardinality and aggregation.

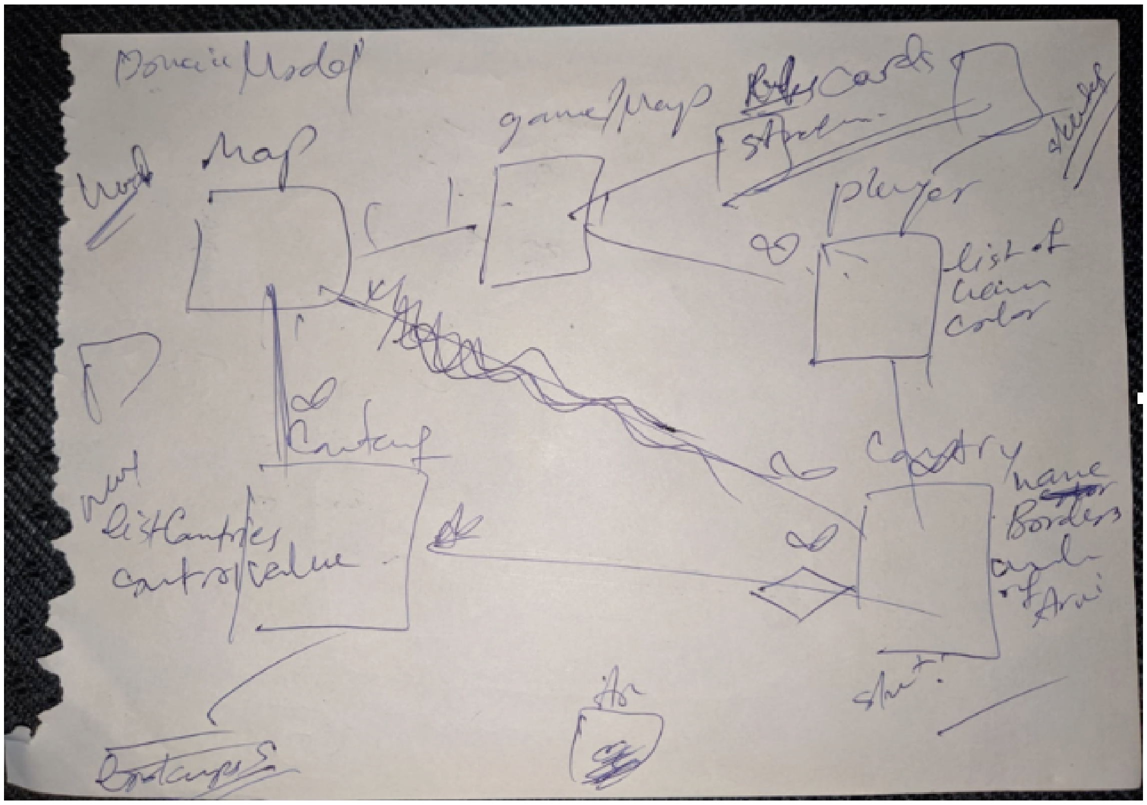


Fig 6 - Domain Model diagram (first version)

a) **Contient**
Is a placeholder for countries.

b) **Country**
Is the unit which players will try to conquer.

c) **Player**
Who is going to play the game (maximum 5)

d) **Game**
In the first version it shows as the Map and in code is the RunningGame object.

e) **Border**
Is the unit that will help us build the Game Graph and be used in calculating the game rules and restrictions.

5) Model-View-Controller

MVC Pattern stands for Model-View-Controller Pattern. We used the MVC design pattern (Architectural Style) to help us maintain a high coherent and loosely coupled code, Using MVC for example makes it possible to change the technology used in implementing the Controllers (SpringShell) and replace it with REST APIs to promote the game to be network distributed game without changing any single line of code in the Model and the View (Command Line and GUI Interface).

The following diagram illustrates the different tiers and components of the game

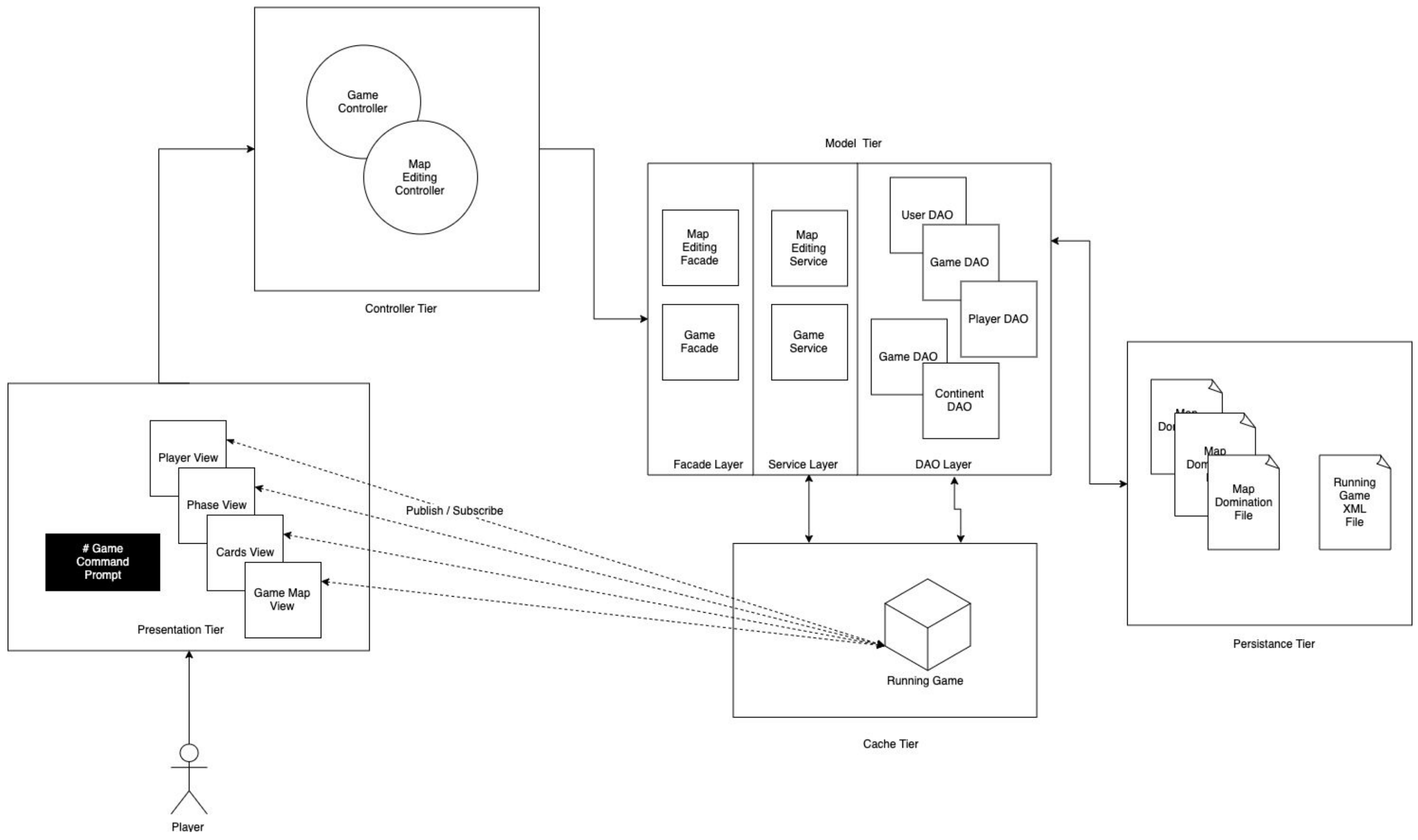
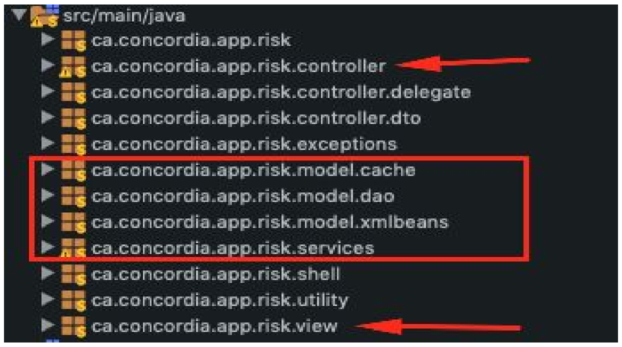


Figure 6, Logical Architecture View

In our coding convention the project packages were also structured to follow the MVC Architectural style, This made it easy on our team to know exactly where to find different classes of the game



```
src/main/java
  ca.concordia.app.risk
  ca.concordia.app.risk.controller
  ca.concordia.app.risk.controller.delegate
  ca.concordia.app.risk.controller.dto
  ca.concordia.app.risk.exceptions
  ca.concordia.app.risk.model.cache
  ca.concordia.app.risk.model.dao
  ca.concordia.app.risk.model.xmlbeans
  ca.concordia.app.risk.services
  ca.concordia.app.risk.shell
  ca.concordia.app.risk.utility
  ca.concordia.app.risk.view
```

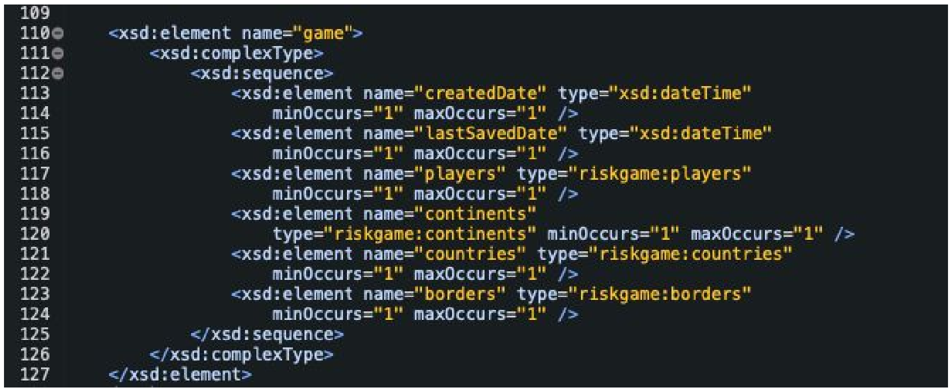
Fig 7 - Project Packages

6) Model

Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

a) XML and XML Schema

We first realized the Domain object as XML elements and generated an xsd schema for them



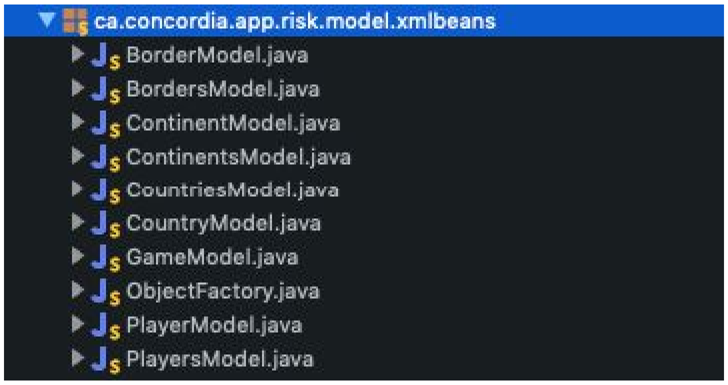
```

    <xsd:element name="game">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="createdDate" type="xsd:dateTime"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="lastSavedDate" type="xsd:dateTime"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="players" type="riskgame:players"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="continents"
                    type="riskgame:continents" minOccurs="1" maxOccurs="1" />
                <xsd:element name="countries" type="riskgame:countries"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="borders" type="riskgame:borders"
                    minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

Fig 8 - Snippet from the Game xml schema (XSD)

b) JAXB - Java API for XML Binding

We used the Java API for XML Binding to create the POJOs that will hold the state of the Game



```
ca.concordia.app.risk.model.xmlbeans
  BorderModel.java
  BordersModel.java
  ContinentModel.java
  ContinentsModel.java
  CountriesModel.java
  CountryModel.java
  GameModel.java
  ObjectFactory.java
  PlayerModel.java
  PlayersModel.java
```

Fig 9 - JAXB POJOs generated by JAXB Compiler

c) **Cache**

RunningGame Class

The Running Game class is one of the most important classes in the game, it holds the state of the game that is being played. It handles the entire state of the game starting from Graph Editing until game end.The graph state, the currently allowed commands, the current player Id and all other global data is part of this class. There is only one RunningGame instance in the JVM while the game is loaded. The RunningGame class uses the singleton design pattern to guarantee that.

Player Class

The Player class is introduced in version two as part of our code refactoring practice, all current player functions were moved to this class to build more coherent objects.

d) **JGraphT**

The graph in the game is created and managed using the open-source JGraphT library [4]. The instance is created in the cache(RunningGame) and all the services which get the graph data from the instance. The purpose of using this library is that it handles all the surprises that graphs usually come along with, thereby, reducing the chances of human-induced errors.

e) **Services**

The Service classes are the pistons of the game, they represent the the business layer, they hold the business logic of the game and they are responsible to change the state of the game

f) **Data Access Object - DAO**

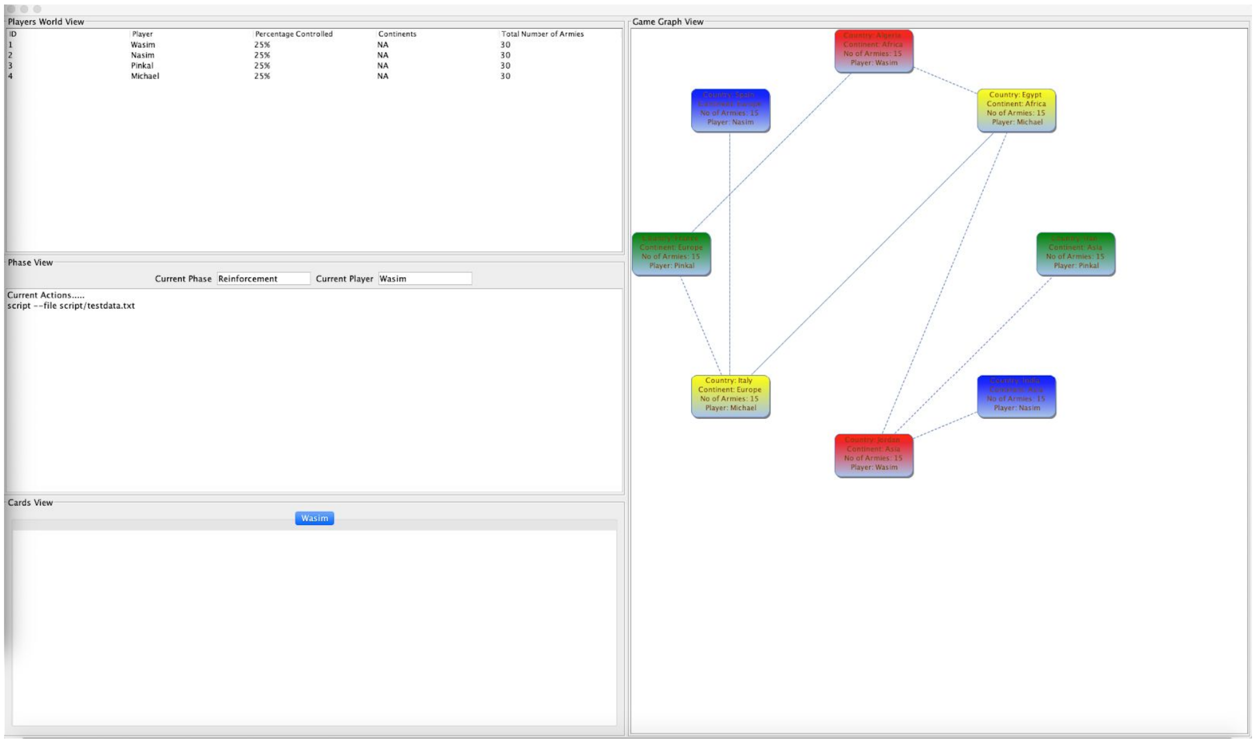
The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API [5]. In our project we use the DAO Objects to fetch information from the RunningGame object, they hide the complexity of finding different object

7) **View**

To make the game more appealing, we decided to build a GUI dashboard using Java GUI swing components.

a) **The Game Dashboard**

We used the Observer Design pattern to help building an interactive GUI dashboard for the game, The main frame is built using JFrame. The different views [Player View, Phase View, Cards View and Game Graph view] are built using JPanels, every panel is an observer that is subscribed to the model [Running Game], whenever the game state is changed the views will be notified to reflect the changes on the corresponding panels.


Players World View

ID	Player	Percentage Controlled	Continents	Total Number of Armies
1	Wasim	25%	NA	30
2	Nasim	25%	NA	30
3	Pinkal	25%	NA	30
4	Michael	25%	NA	30

Phase View

Current Phase: Reinforcement
Current Player: Wasim

```
Current Actions.....
script --file script/testdata.txt
```

Fig 10 - Application Dashboard showing different views

8) Controller

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps the view and model separate.

a) SpringShell

Users of the Spring Shell project can easily build a full featured shell (aka command line) application by depending on the Spring Shell jars and adding their own commands (which come as methods on spring beans). Creating a command line application can be useful e.g. to interact with your project's REST API, or to work with local file content [6].

b) Data Transfer Objects

Transfer of data between the controller and the services is done through Data Transfer Objects(DTOs). This is not a mandatory requirement but this falls inline with the best practices of passing data between different layers of the application. The DTO can also include the properties which are part of the model and the DAO copies those properties from the DTO and operates on them.

c) Business Delegate

The business delegate acts as an interface between the controller and the service classes. Each request that comes to the controller is carefully mapped to the delegate which in turn calls the required service to handle that request. The main purpose behind using the Business Delegate Design pattern is to separate the controller from the domain classes.

9) References

[1] Wikipedia contributors. Risk (game). Wikipedia, The Free Encyclopedia. October 16, 2019, 00:33 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Risk_(game)&oldid=921485153. Accessed October 16, 2019.
[2] https://www.guru99.com/junit-tutorial.html, Last accessed October 15, 2019
[3] Wikipedia contributors. Apache Maven. Wikipedia, The Free Encyclopedia. September 12, 2019, 02:49 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Apache_Maven&oldid=915250670. Accessed October 16, 2019.
[4] JGraphT, https://jgrapht.org/, Last accessed October 15, 2019
[5] The DAO Pattern in Java ,https://www.baeldung.com/java-dao-pattern, Last accessed October 15, 2019
[6] Spring Shell, https://projects.spring.io/spring-shell/, Last accessed October 15, 2019