DRisk

Software engeneering 2018/2019

Claudio Rota 816050
Simone Paolo Mottadelli 820786
Matteo Paolella 816933
Stefano Zuccarella 816482

Contents

1 Introduction

Risk is a turn-based board game, in which 2 to 6 players fight over the occupation of 42 territories in a political map, trying to achieve a secret mission that generally requires the control of the territories in the map.

More in detail, the aim of the game is to achieve a goal, that is assigned to each player at the beginning of the game, and is kept secret from other players.
Players control armies, which are deployed in occupied territories (and grouped in continents), and can be used to capture territories from other players. The results of attacks between territories is determined by rolling dices.

Unfortunately, Risk has several known playability limitations, for example the map configuration is rigid, in the sense that both the number of territories and the neighboring relation between territories (which determines how the attacks between territories can be performed) are fixed for all game instances. Another limitation is that the granularity of the map (in particular, how many territories constitute it) is also fixed, making some game instances very lengthy, especially when a small number of players is involved.

DRisk is a highly configurable and web based client-server version of the Risk game, where users play the game through a web site and whose goal is to overcome the above mentioned limitations.
In particular, it implements all core functionalities and rules of the standard version of Risk, including rules for obtaining and placing armies, attacking territories, and fortifying, as well as supporting Risk cards.
The user can choose between a single objective or multiple objectives with card missions and, additionally, the game supports greater levels of configuration, through the possibility of choosing the map one wants to play on, and different complexity levels. It comes with three different preloaded maps, each with a different difficulty level. More precisely, DRisk has been developed to:

• allow users to define and upload map configurations, including not only the aesthetic aspects such as images, etc., but also specific numbers of territories, grouping into continents, and neighboring relations (and optionally stronger restrictions on the number of players);

• allow for maps to automatically adjust granularity, by collapsing neighboring territories according to number of players and a user selected complexity for the game (e.g., easy, medium, hard).

2 Requirements Analysis

2.1 Brief introduction

The only actors who interact with the system are the players and we can summarize the things they can do with two main use cases: *play match* and *adjust match settings*.

2.2 Use cases

2.2.1 Brief descriptions

• **Adjust match settings:** The user connects himself to the lobby page and inserts his nickname. Soon afterward, the user chooses the map type as well as the difficulty level and also between single or multiple objectives. If the player chooses a custom map, he must import his own map and its configuration, including the aesthetic parts.

• **Play match:** The player sets himself ready to start the game and waits for the other players to do the same. When everyone is ready, the game starts. The system assigns territories as well as the initial tanks to each player. The system also assigns a mission to each player. The map is now loaded and rendered. The players can now start the use case *Place initial tanks*. At the end, a player will be the winner.

• **Place initial tanks:** The player places his first tanks on his territories. When everyone has completed the initial tanks placement phase, a player is drawn among the players to start the first turn. After that, the drawn player must start the use case *Get tanks*.

• **Get tanks:** The system assigns tanks to the current player, proportionally to the number of territories that he owns. If the player owns a continent, bonus tanks will be assigned to him. The use case *Use cards* starts.

• **Use cards:** The player can choose three cards from his hand to be converted into additional tanks. If a card contains a territory owned by the player, bonus tanks will be assigned. The player can start the use case *Use cards* another time or start the use case *Place tanks*.

• **Place tanks:** The player must choose the territory on which he wants to place the tanks. Then, he chooses the number of tanks to place. If the player has more tanks to place, he starts this use case another time, otherwise the use case *Attack enemy* starts.

• **Attack enemy:** The player chooses the territory from which he wants to attack and the territory to attack, which must be adjacent to the first one. Then, he chooses how many tanks to use to conduct the attack, but at least a tank must remain on that territory. The winner is determined by rolling dices. The attacker rolls a dice for each tank used during the attack and the defender does the same for each tank used during the defense. Both players can roll at most three dices. The dices are sorted and compared. Afterwards, the system determines how many tanks both the defender and the attacker have lost. If the defender loses all the tanks on the territory, then the attacker conquers it and all the tanks used during the attack are moved to the conquered territory. If the player has successfully conquered at least a territory, then the attacker awards a territory card, which can be later used to build a tris. The player can start this use case another time or he can start the use case *Move tanks*.

• **Move tanks:** The player can choose the territory from which he wants to move his tanks and a territory which must be adjacent to the first one. The player chooses the number of tanks to move, but at least a tank must remain on that territory. The player must start the use case *End turn*.

• **End turn:** The player who is playing the turn passes it to the next player, who can start the use case *Get tanks*.

2.2.2 Fully detailed use cases

Adjust match settings

- **Use case name:** Adjust match settings
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The player has joined the lobby.
- **Post conditions:** The match settings are successfully saved: the map is ready and the type of the mission is set.
- **Stakeholder:** The other players. They want that the match settings are set to start the game.
- **Main success scenario:**
  1. The player chooses the type of the mission.
  2. The player chooses the difficulty of the map.
  3. The system updates the match settings.
  4. If the game has not already started, the player can go back to the point 1 or 2.
- **Extensions:**
  0. Anytime, the player can leave the game.
     (a) The system removes the player from the lobby.
  2. The player chooses a custom difficulty.
     (a) The player writes the map components.
        i. The player writes the map components in a wrong format.
        ii. The system notifies an error.
        iii. Return to point 2 of the main scenario.
     (b) The player writes the aesthetic part of the map.
     (c) Return to point 3 of the main scenario.

Play match

- **Use case name:** Play match
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The match settings have been configured.
- **Post conditions:** A player has won the game.
- **Main success scenario:**

  1. The player sets himself ready to start the game.
  2. The player waits for others to do the same.
  3. The game starts.
  4. The system assigns random territories to each player.
  5. The system places a tank on each territory for each player.
  6. The system assigns a mission to each player, according to the match settings.
  7. The use case *Place initial tanks* can start.

- **Extensions:**

  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
     (b) The system assigns a new turn to the next player if the use case *Place initial tanks* has finished.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.
        iii. The system initializes a new match.
  1. The player sets himself not ready.
     (a) The player, when he wants, can go back to point 1.

Place initial tanks

- **Use case name:** Place initial tanks
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The game has started. The player has received his initial territories, tanks and his mission.
- **Post conditions:** The player has placed all his initial tanks on his territories, each with at least a tank.
- **Main success scenario:**

  1. The player chooses a territory of his own.
  2. The player chooses the amount of tanks to place on that territory.
  3. The system updates the number of tanks on that territory.
  4. The player must go back to point 1 until he has available tanks to place.
  5. The player waits for other players to finish this use case too.

- **Extensions:**

  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.
        iii. The system initializes a new match.
  1. The player chooses a territory not of his own.
     (a) The system notifies an error.
     (b) Return to point 1 of the main scenario.
  2. The player tries to place more tanks than the ones he owns.
     (a) The system notifies an error.

(b) Return to point 1 of the main scenario.

5. The player is the last one to place his initial tanks.
   (a) The system let a player start the use case *Play turn*.

Use cards

- **Use case name:** Use cards
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The player is playing his turn.
- **Post conditions:** The player has received additional tanks according to the cards used.
- **Main success scenario:**
  1. The player chooses three cards from the ones he owns.
  2. The system converts the cards into additional tanks.
  3. The system assigns these tanks to his available tanks.
  4. The system discards the cards used from the hand of the player.
  5. The player can go back to point 1 if he wants to use other cards.
- **Extensions:**
  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
     (b) The system assign a new turn to the next player.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.
        iii. The system initializes a new match.
  2. The player chooses three cards that aren't a valid tris.
     (a) The system notifies an error.
     (b) Return to point 1 of the main scenario.
  3. The cards chosen contains territories that the player actually owns.
     (a) The system assigns 2 more tanks to the player for each territory owned printed on the cards.

Place tanks

- **Use case name:** Place tanks
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The player is playing his turn.
- **Post conditions:** The player has placed all his available tanks on his territories.
- **Main success scenario:**

  1. The player chooses a territory of his own.
  2. The player chooses the amount of tanks to place on that territory.
  3. The system updates the number of tanks on that territory.
  4. The player must go back to point 1 until he has available tanks to place.

- **Extensions:**

  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
     (b) The system assign a new turn to the next player.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.
        iii. The system initializes a new match.
  1. The player chooses a territory not of his own.
     (a) The system notifies an error.
     (b) Return to point 1 of the main scenario.
  2. The player tries to place more tanks than the ones he owns.
     (a) The system notifies an error.
     (b) Return to point 1 of the main scenario.

Attack enemy

- **Use case name:** Attack enemy
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Stakeholder:** The other player under attack, who wants to defend the territory being attacked.
- **Preconditions:** The player is playing his turn and all his available tanks have already been placed.
- **Main success scenario:**

  1. The player chooses a territory of his own from which the attack will be conducted.
  2. The player chooses an enemy territory.
  3. The player chooses the number of tanks to use during the attack.
  4. The system rolls a dice for each tank used during the attack, with the constraint that at most three dices can be rolled.
  5. The system rolls a dice for each tank used during the defense, with the constraint that at most three dices can be rolled.
  6. The system determines the tanks lost for both parties and remove them from the game.
  7. The player can go back to point 1 if he wants to attack another time.

- **Extensions:**

  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
     (b) The system allows another player to start his turn.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.

iii. The system initializes a new match.

1. The player chooses a territory not of his own.
   (a) The system notifies an error.
   (b) Return to point 1 of the main scenario.
2. The player chooses an enemy territory which is not adjacent to the one from which he desires to attack.
   (a) The system notifies an error.
   (b) Return to point 1 of the main scenario.
2. The player chooses a territory of his own.
   (a) The system notifies an error.
   (b) Return to point 1 of the main scenario.
3. The player chooses a number of tanks which is greater or equal than the number of tanks placed on the territory from which the attack will take place.
   (a) The system notifies an error.
   (b) Return to point 1 of the main scenario.
6. The defender loses all the tanks on his territory.
   (a) The player attacker conquers that territory.
   (b) The player attacker moves all the tanks survived during the attack to the conquered territory.
      i. The defender loses all of his territories.
      ii. The system removes him from the game.
      iii. The system changes the mission for all those players who had to destroy the defender.
   (c) If the player attacker hasn't been awarded yet with a territory card for a conquest of an enemy territory, then the system gives one to him.
6. The attacker completes his mission.
   (a) The system notifies every player that the attacker has won the game.
   (b) The system initializes a new match.

Move tanks

- **Use case name:** Move tanks
- **Scope:** DRisk
- **Level:** User goal
- **Primary actor:** Player
- **Preconditions:** The player is playing his turn and all his available tanks have been placed.
- **Post conditions:** The tanks have been moved between two adjacent territories owned by the player.
- **Main success scenario:**

  1. The player chooses a territory owned from which to move the tanks.
  2. The player chooses another territory owned on which to move the tanks.
  3. The player chooses the amount of tanks he wants to move.
  4. The system updates the number of tanks of both territories.

- **Extensions:**

  0. Anytime, the player can leave the game.
     (a) The system removes the player from the game.
     (b) The system passes the turn to the next player.
        i. Only one player is remained in the game.
        ii. The system makes him the winner.
        iii. The system initializes a new match.
  1. The player chooses a territory that is not of his own.
     (a) The system notifies an error.
     (b) Return to point 1 of the main scenario.
  2. The player chooses a territory that is not of his own or that is not adjacent to the territory from which he wants to move the tanks.

(a) The system notifies an error.

(b) Return to point 1 of the main scenario.

3. The player chooses an amount of tanks greater or equals then the available tanks on the territory from which he wants to move the tanks.

   (a) The system notifies an error.

   (b) Return to point 1 of the main scenario.

2.2.3 Use cases diagram

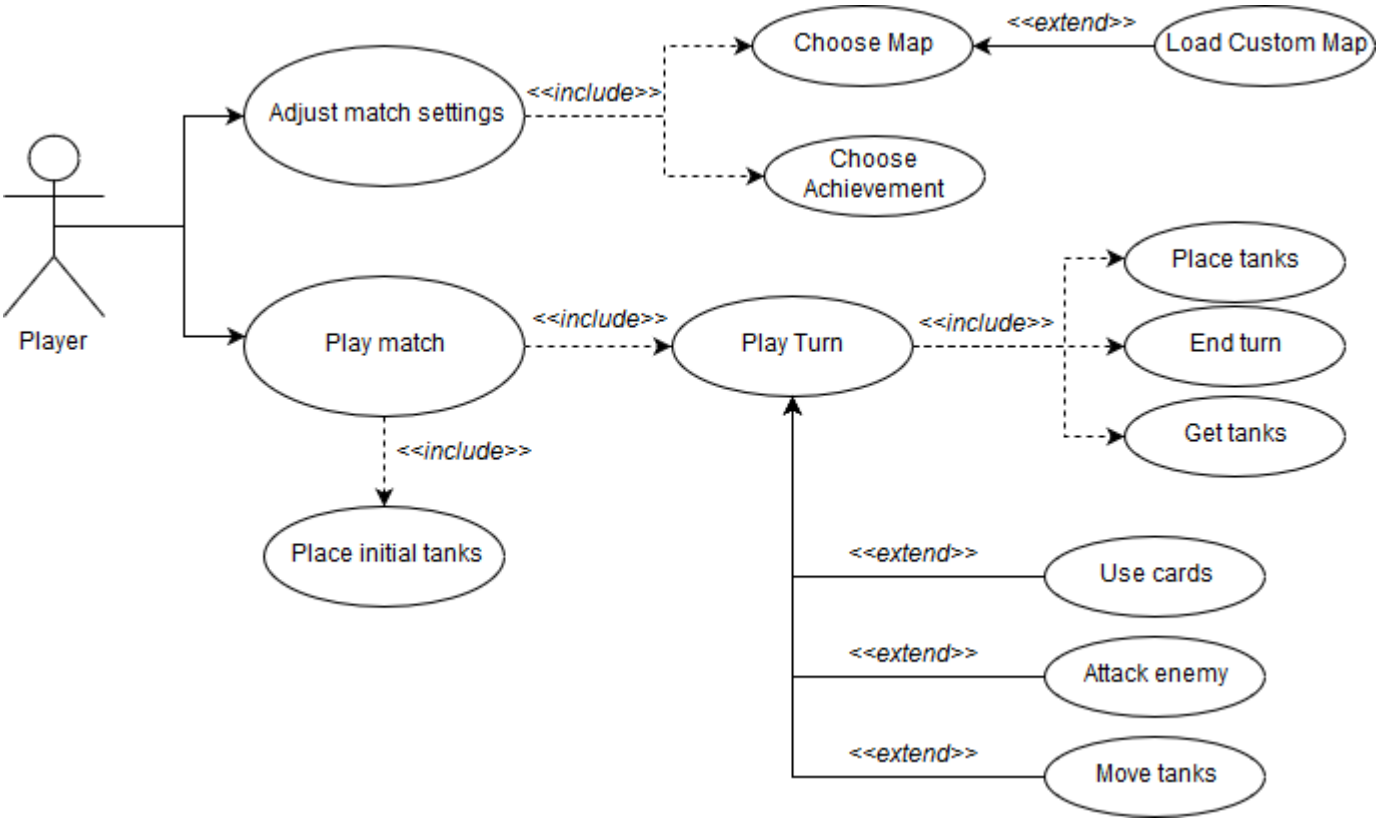

Figure 1: Use cases diagram

3 Domain analysis

3.1 Activity diagrams

3.1.1 Get tanks activity diagram

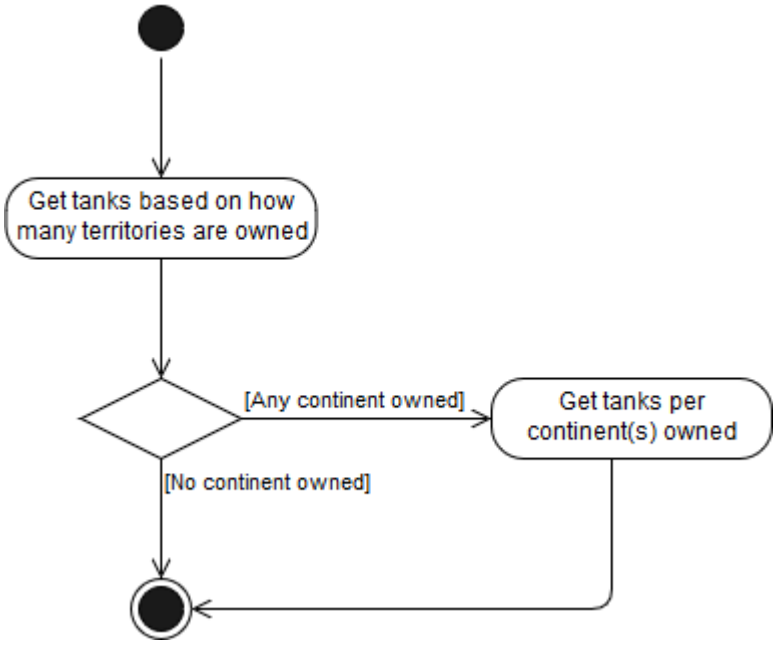

Figure 2: *Get tanks* activity diagram

3.1.2 Use cards activity diagram

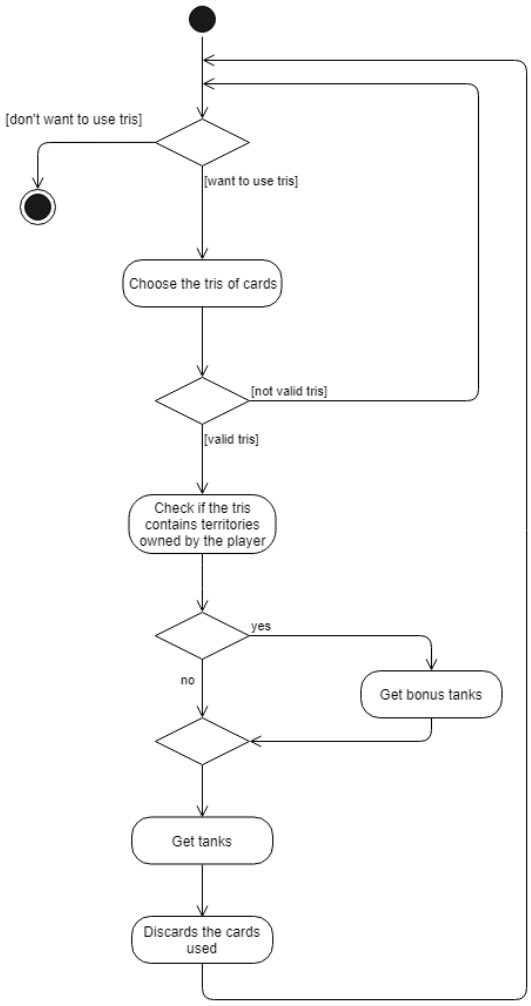

Figure 3: *Use cards* activity diagram

3.1.3 Place tank activity diagram

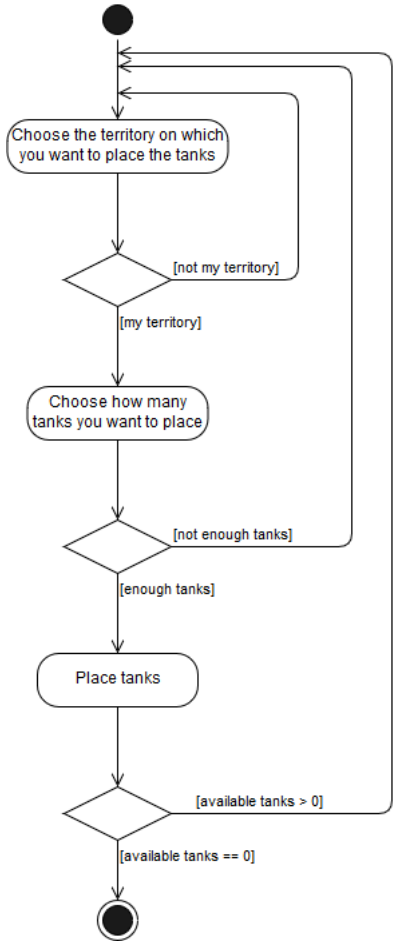

Figure 4: *Place tanks* activity diagram

3.1.4 Attack enemy activity diagram

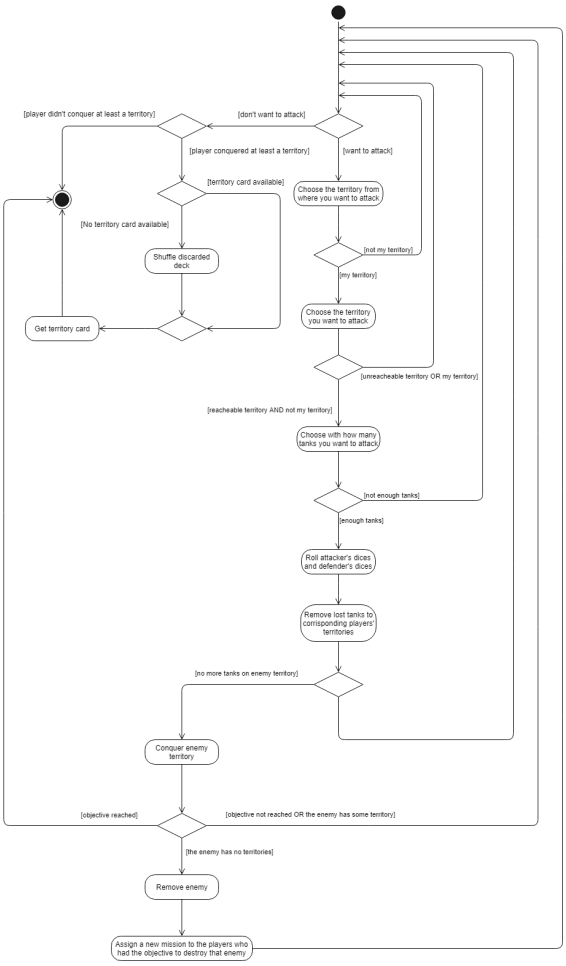

Figure 5: *Attack enemy* activity diagram

3.1.5 Move tanks activity diagram

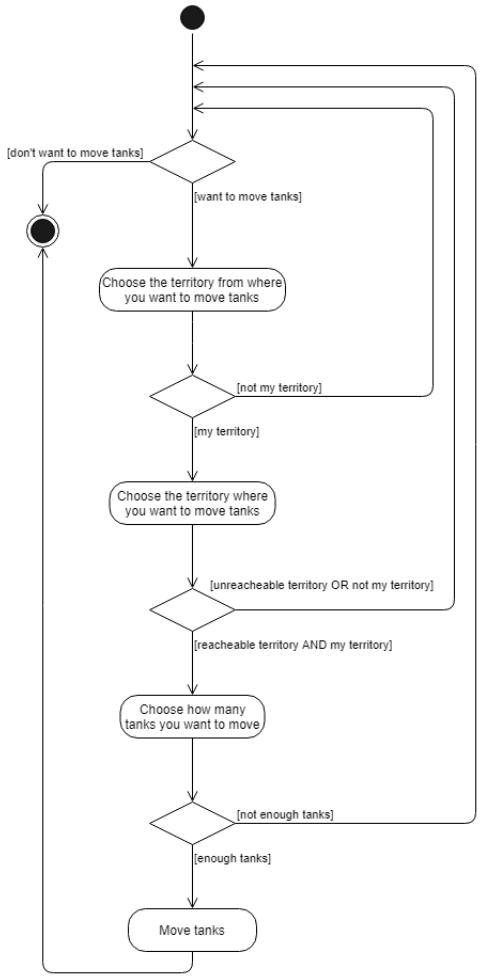

Figure 6: *Move tanks* activity diagram

3.2 Domain diagram

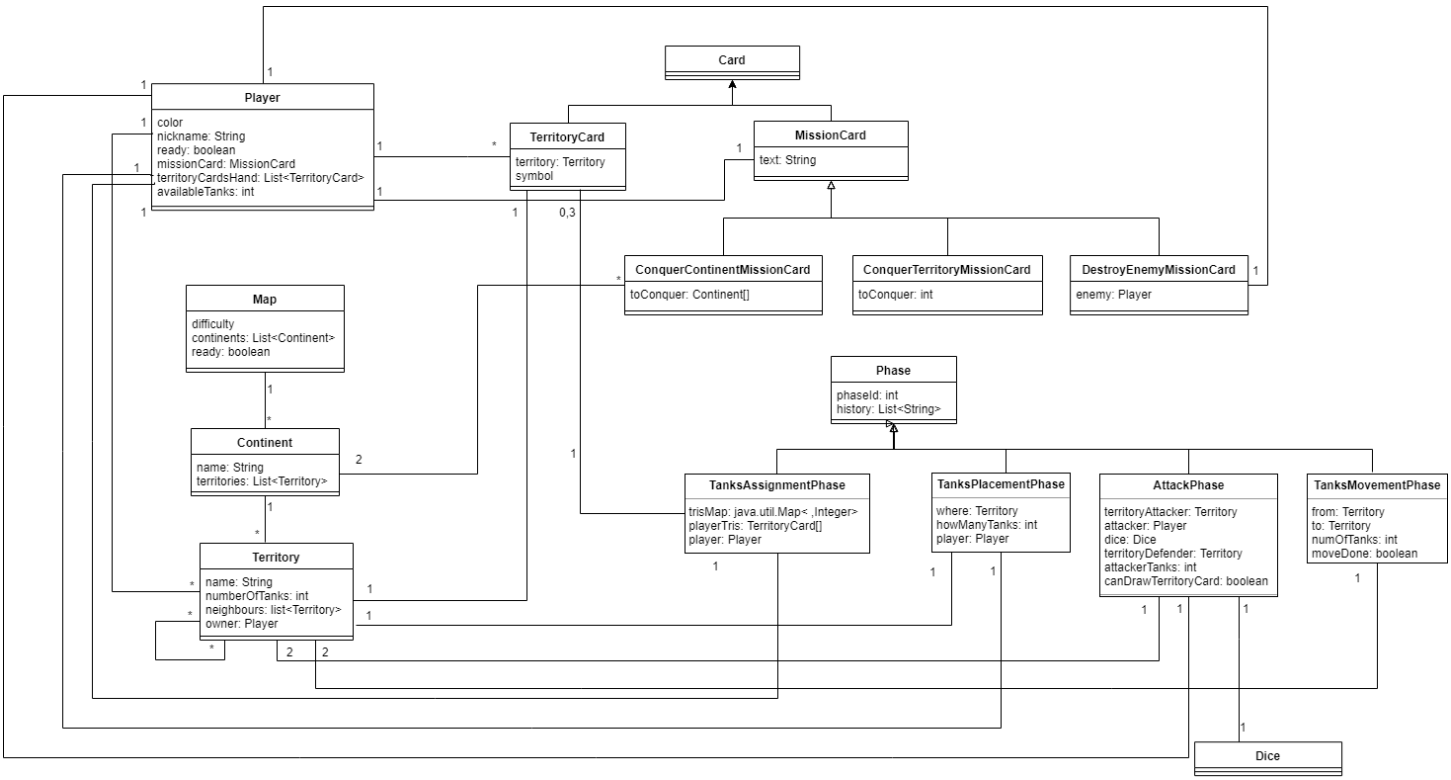

Figure 7: Domain class diagram

4 Design

4.1 Introduction

DRisk implements a multiple tier client-server architecture. More specifically, the clients send requests to the controllers, *LobbyController* and *GameController* located in the package *controller* in the server. The controllers interpret the request payloads and delegate the work to the business classes in the *domain* package. In the *technical service* package there are two classes: *FileLoader* and *JsonHelper*. These classes are used to load the maps of the game and to help the controllers to create and read the JSON responses or requests payloads.

4.2 Package diagram

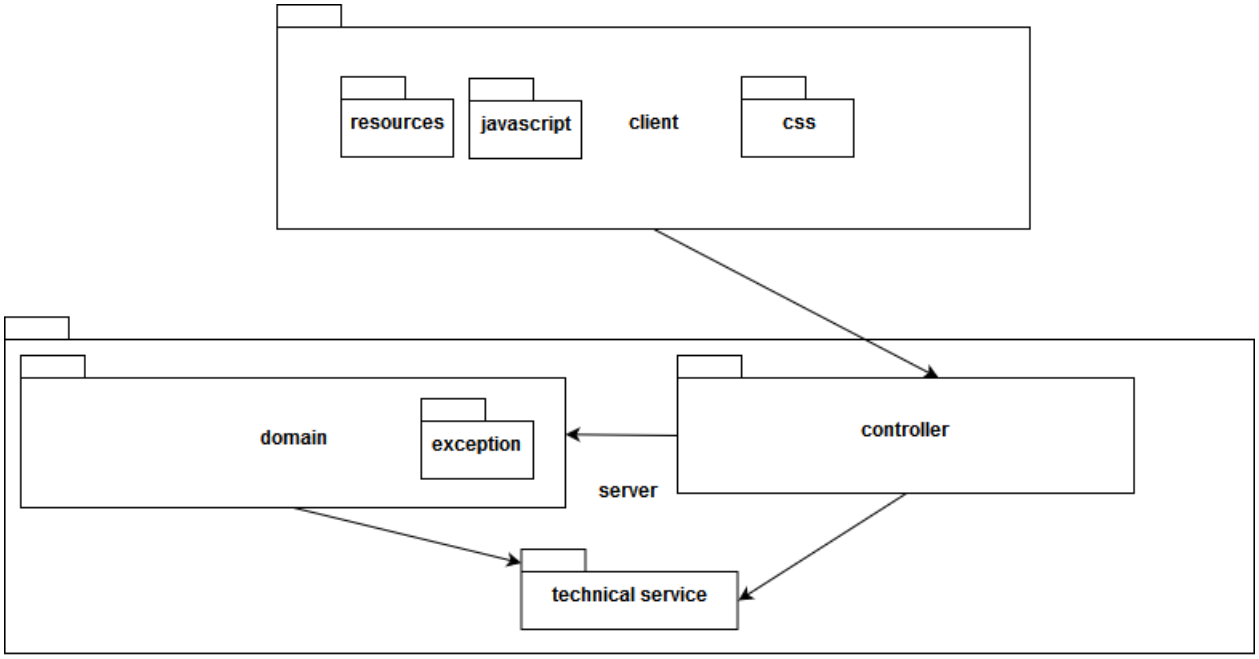

Figure 8: Package diagram

4.3 Project class diagrams

4.3.1 Controller package class diagram

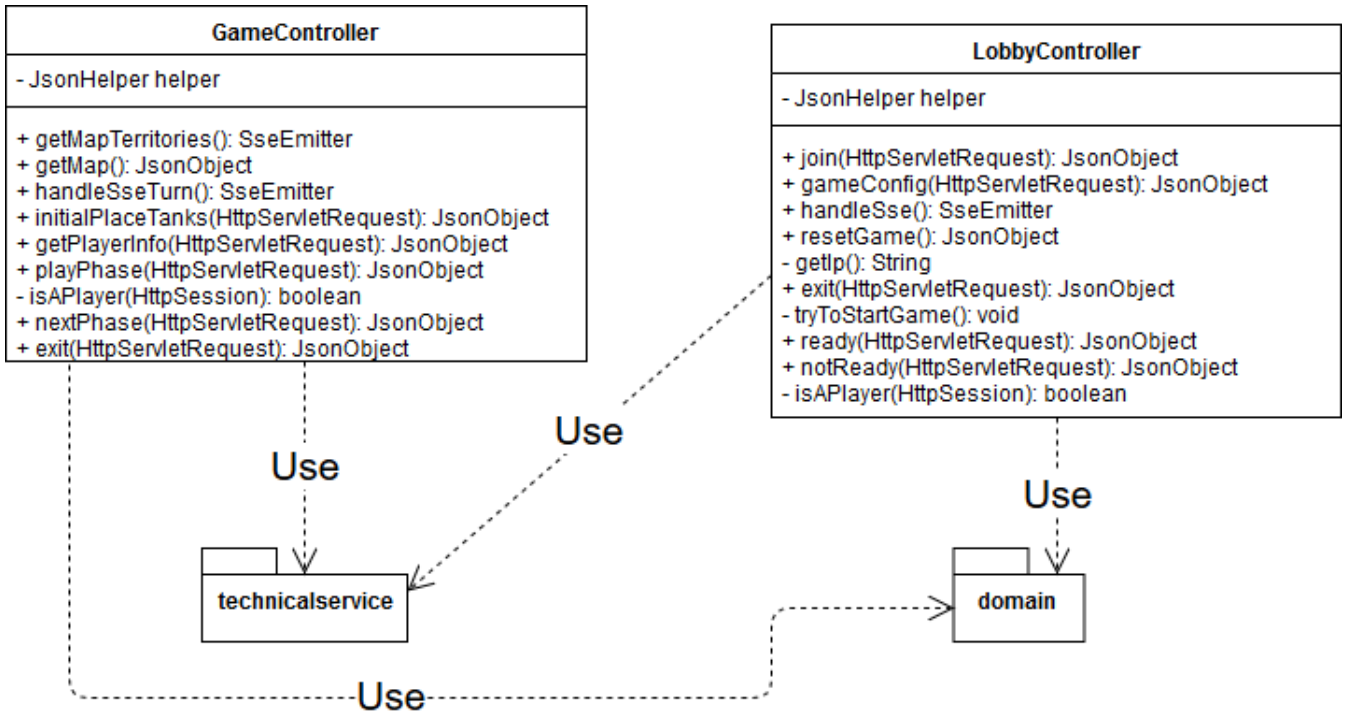

Figure 9: Controller package class diagram

4.3.2 Domain package class diagram

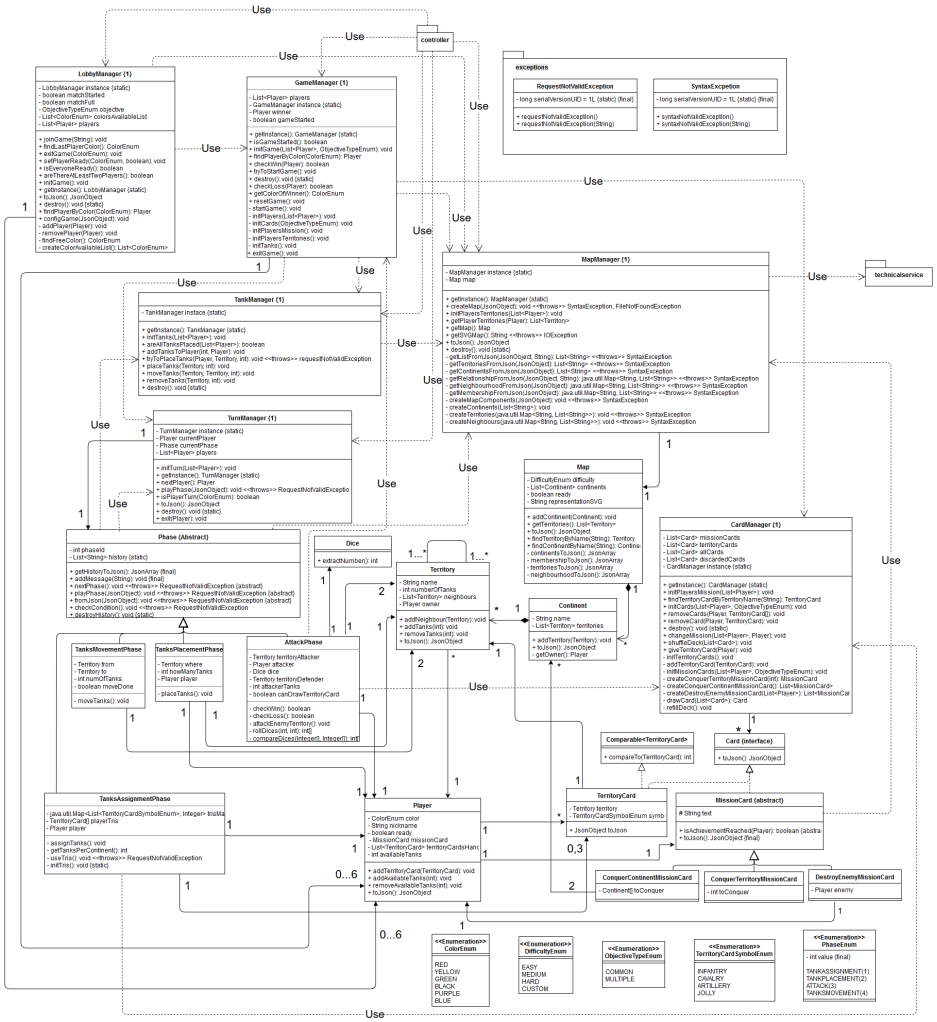

Figure 10: Domain package class diagram

4.3.3 Technical service package class diagram

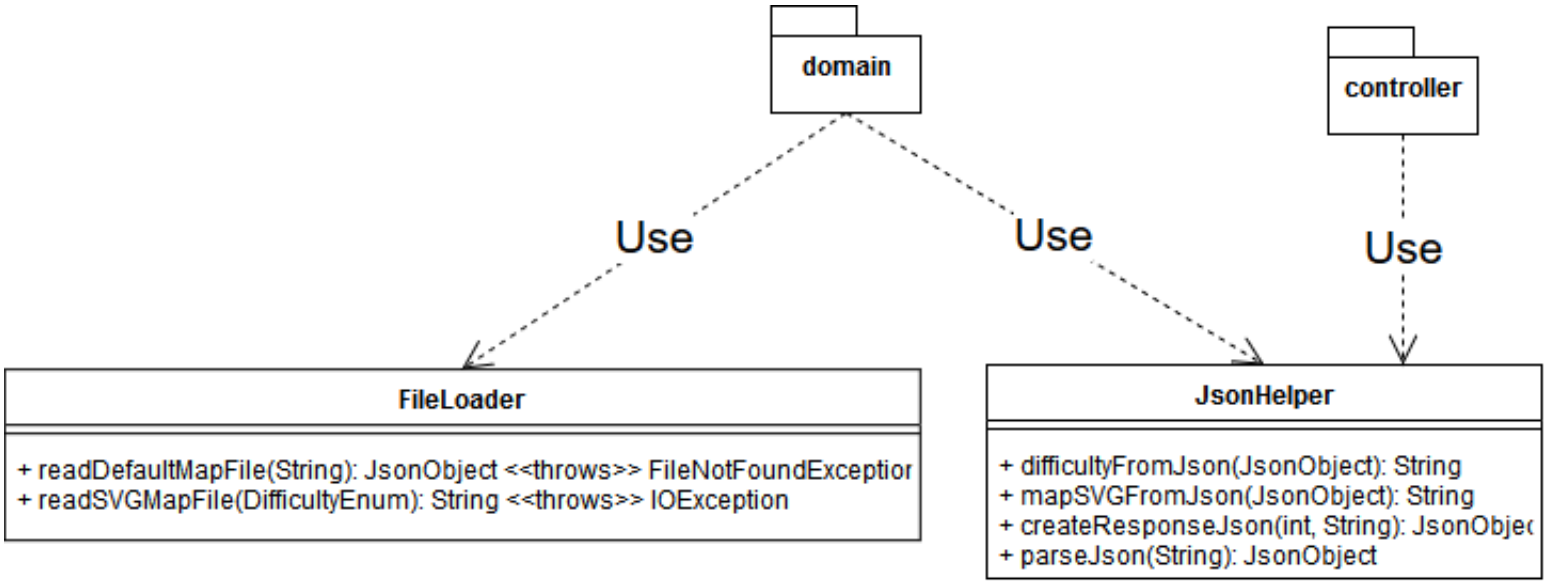

Figure 11: Technical service package class diagram

4.4 Design patterns

For this project, two design patterns have been used:

• **Singleton:** All the manager classes implement this design pattern, because only an instance of them must exist at runtime.
You can find singleton pattern at the following url:

https://sourcemaking.com/design_patterns/singleton

• **State:** This pattern is used in the TurnManager to maintain the correct order of phases of a turn. The phases are sorted in the following way:

1. *TanksAssignmentPhase*
2. *TanksPlacementPhase*
3. *AttackPhase*
4. *TanksMovementPhase*

When the TurnManager is initialized, the current phase is set to *TanksAssignmentPhase*. The current player, who is playing his turn, is the only one who can change between phases.
Note that a player can play a phase multiple times and can go to the next phase only when he wants to. When the current player is playing the *TanksMovementPhase* and tries to go to the next phase, then the current phase is set to *TanksAssignmentPhase* another time and the player passes the turn to the next player.
You can find state pattern at the following url:

https://sourcemaking.com/design_patterns/state

4.5 Sequence diagrams

4.5.1 Join game sequence diagram

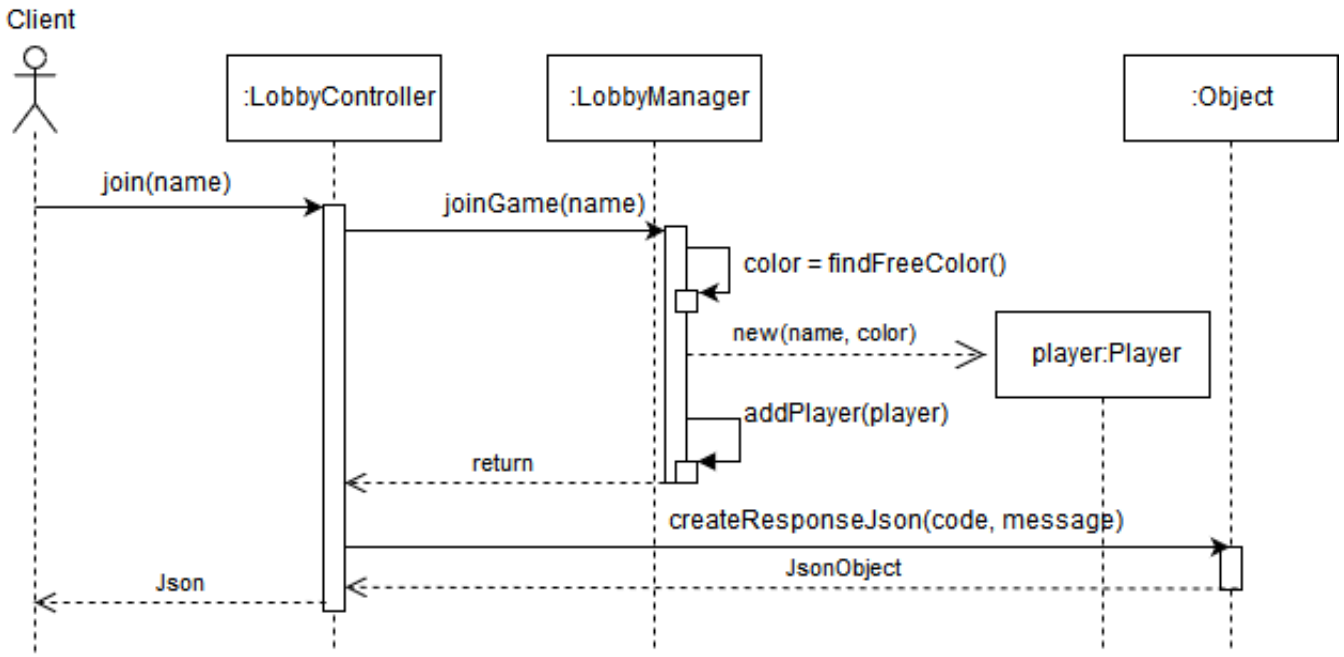

Figure 12: "Join game" sequence diagram

4.5.2 Configure game sequence diagram

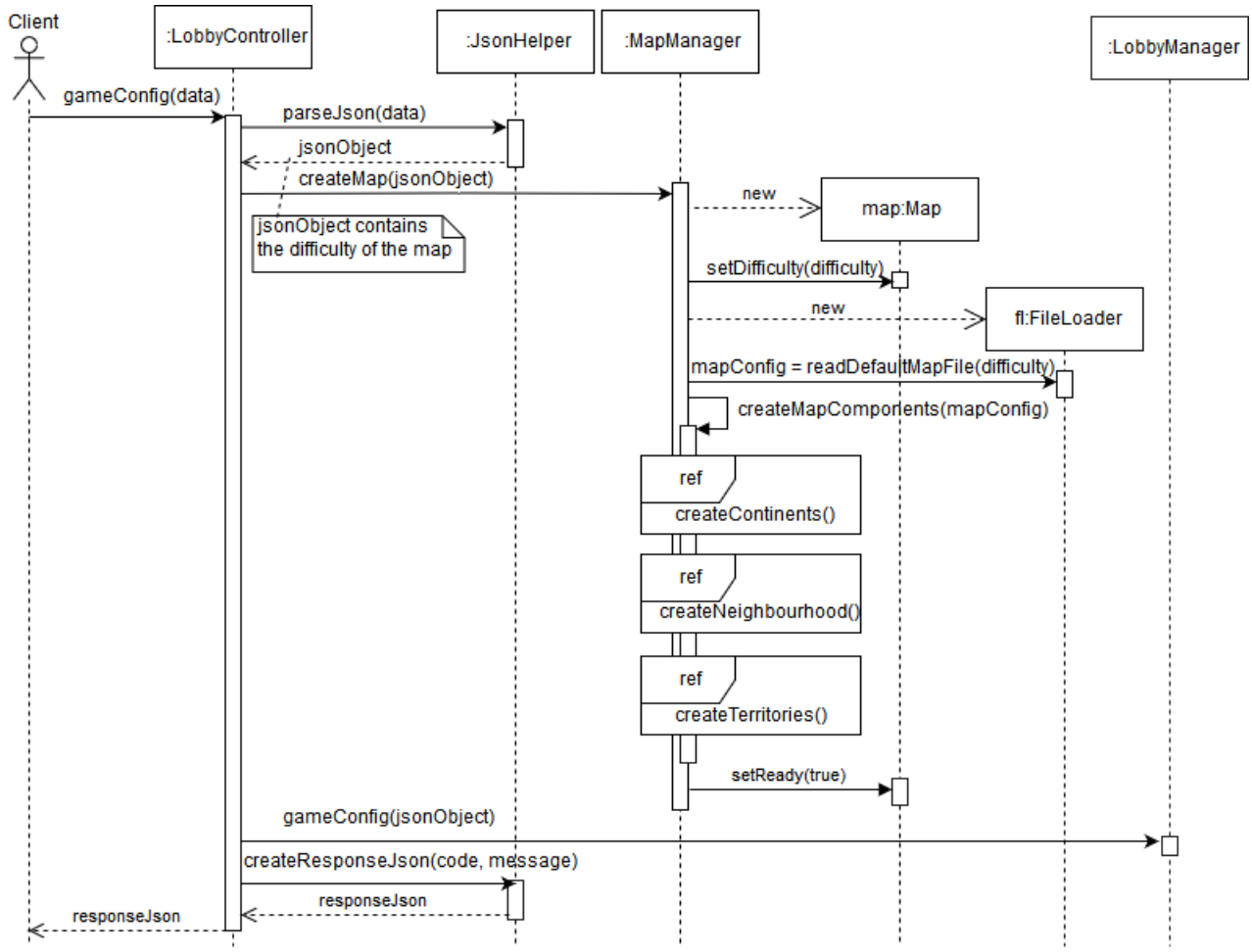

Figure 13: "Configure game" sequence diagram

4.5.3 Create continents sequence diagram

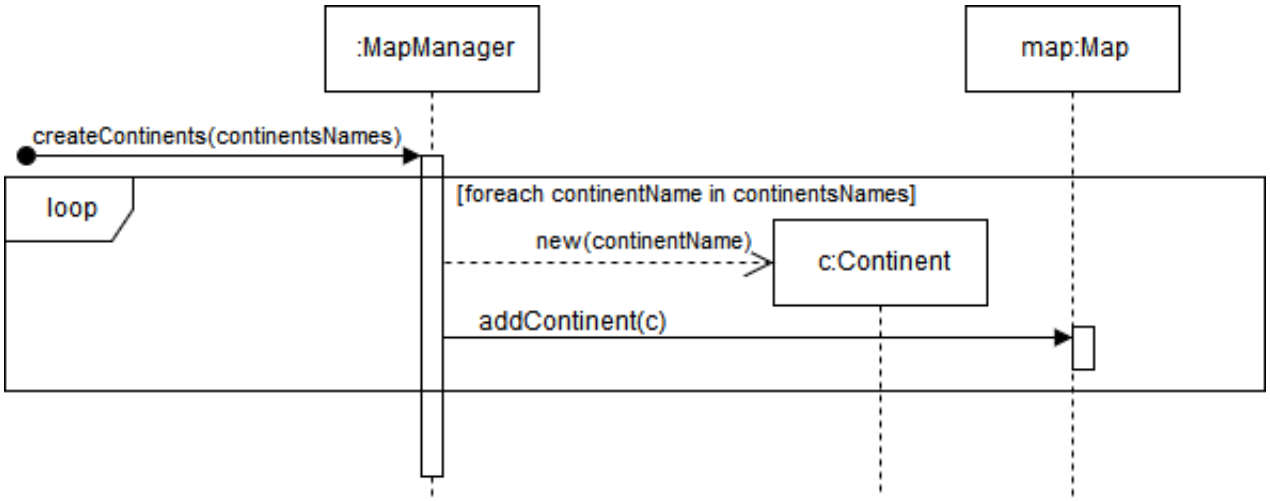

Figure 14: "Create continents" sequence diagram

4.5.4 Create territories sequence diagram

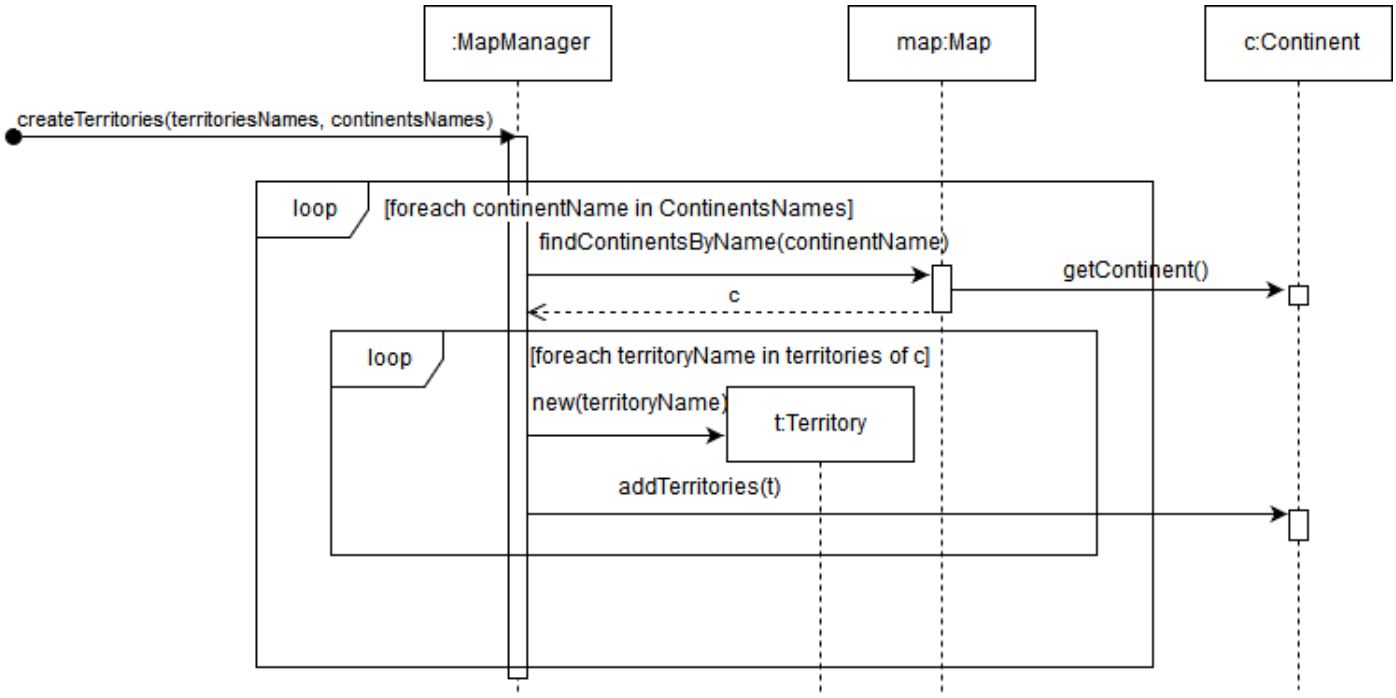

Figure 15: "Create territories" sequence diagram

4.5.5 Create neighbours sequence diagram

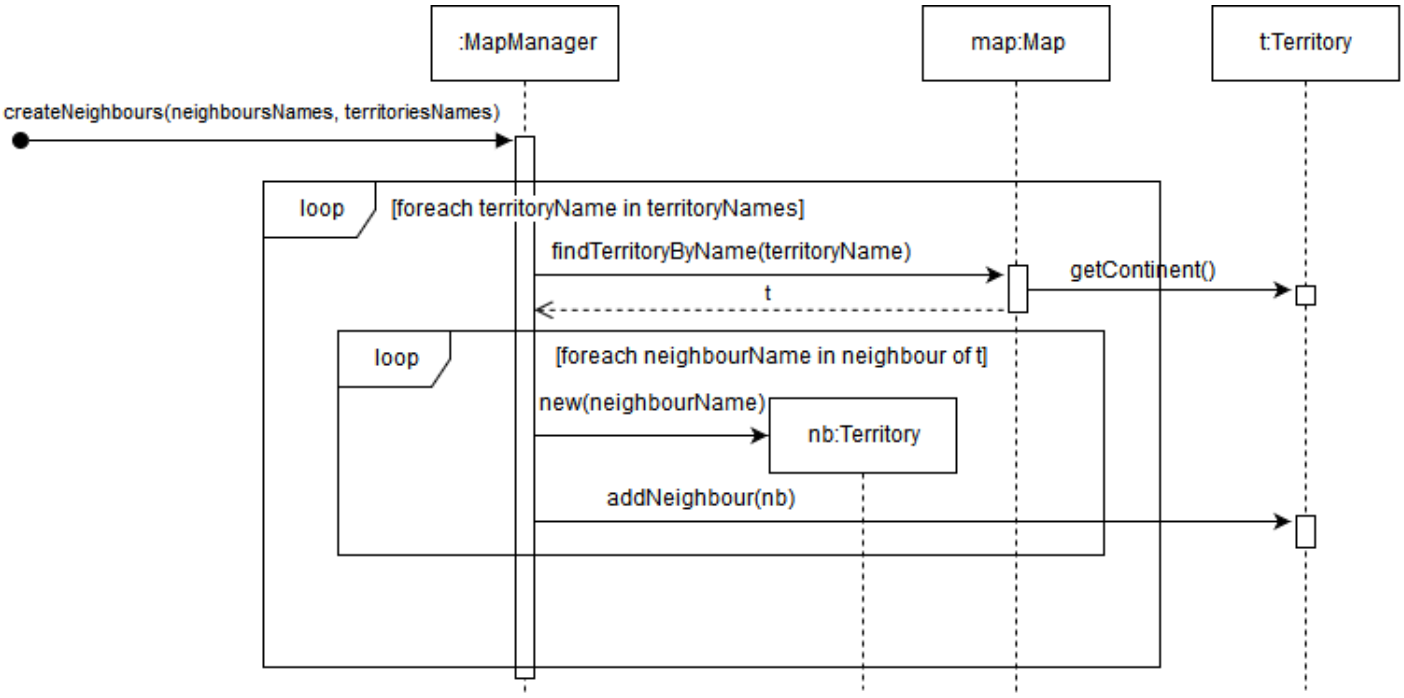

Figure 16: "Create neighbours" sequence diagram

4.5.6 Play phase sequence diagram

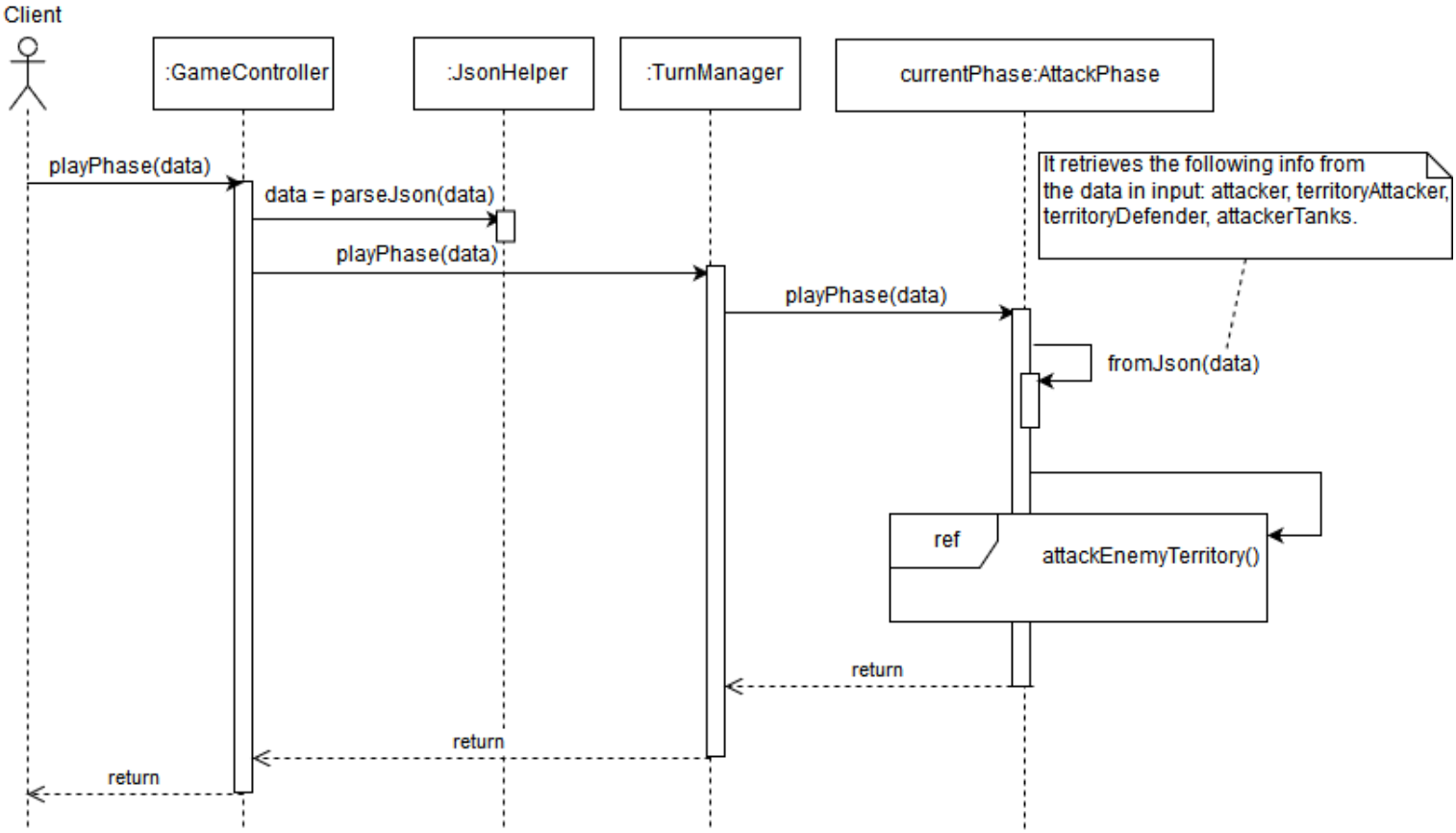

Figure 17: "Play phase" sequence diagram

4.5.7 Attack enemy territory sequence diagram

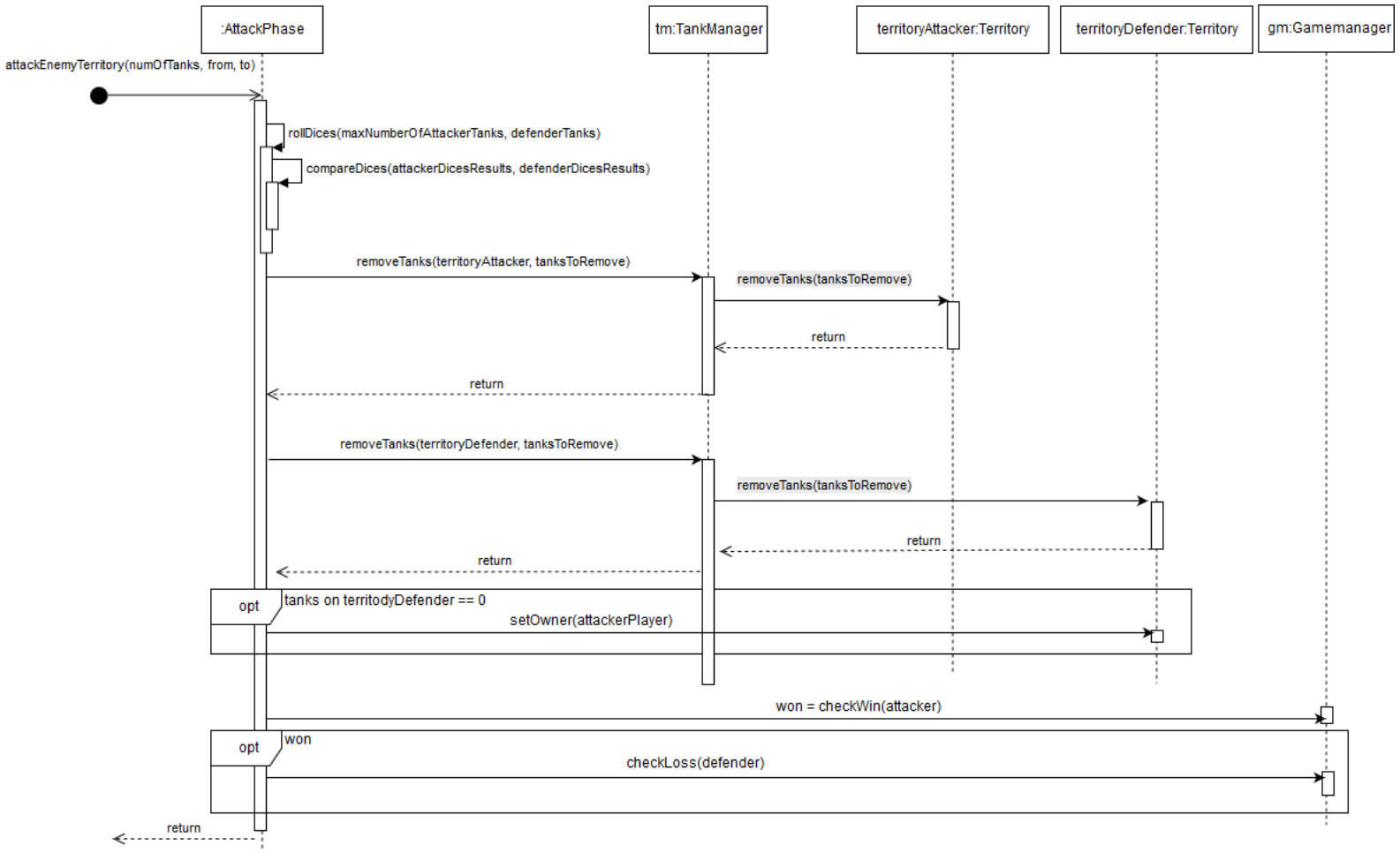

Figure 18: "Attack enemy territory" sequence diagram

4.5.8 Move tanks sequence diagram

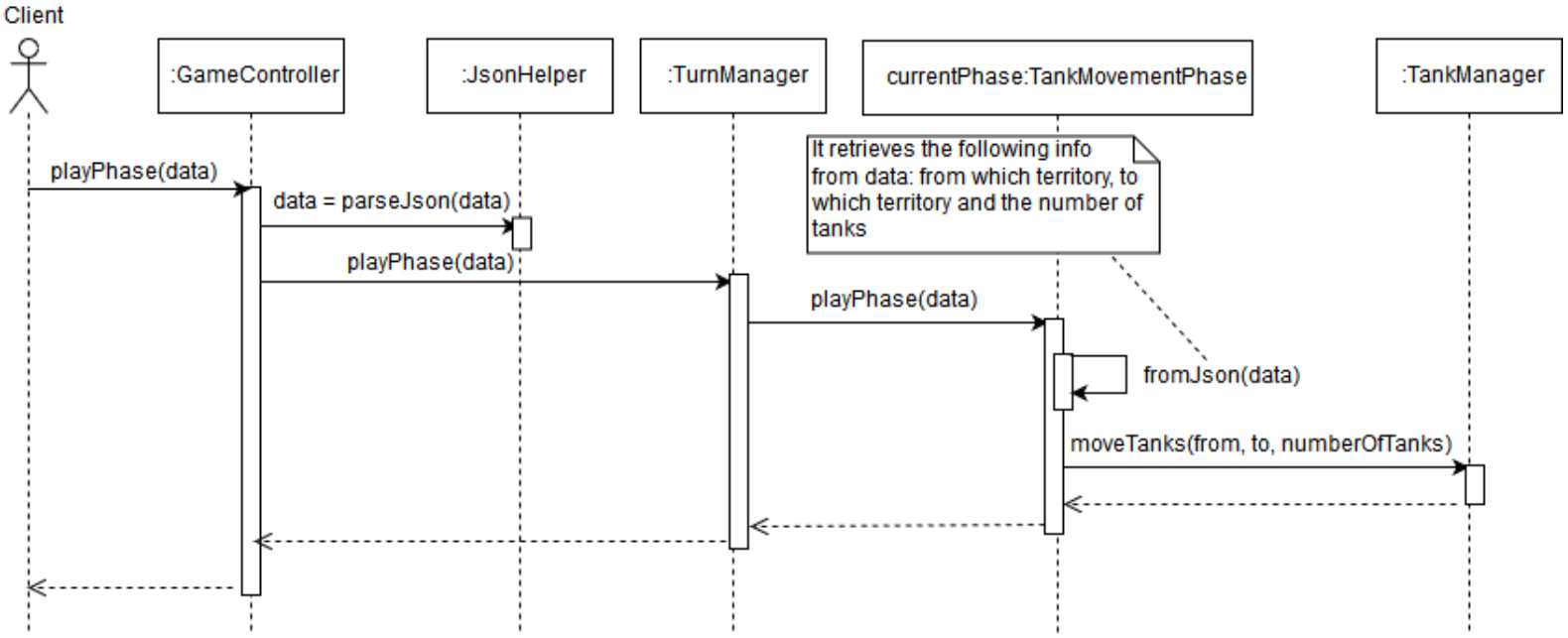

Figure 19: "Move tanks" sequence diagram

4.6 State diagrams

4.6.1 Player ready state diagram

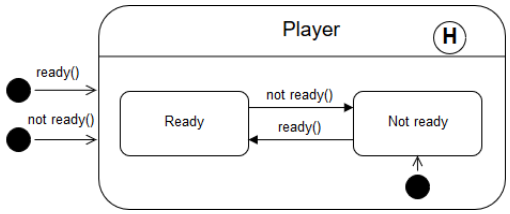

Figure 20: Player state diagram

4.6.2 TurnManager state diagram

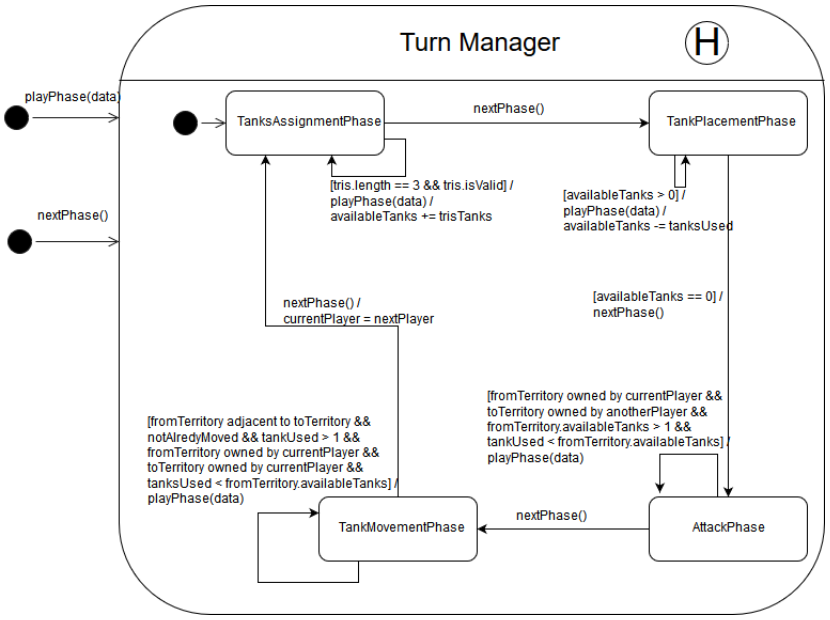

Figure 21: TurnManager state diagram

4.7 Deployment diagram

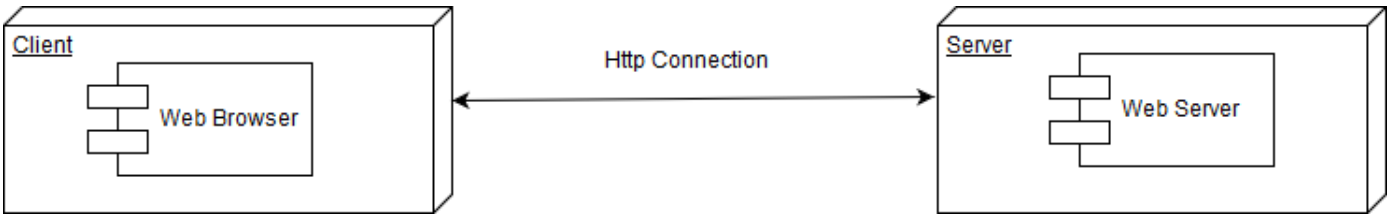

Figure 22: Deployment diagram

5 Implementation

5.1 Introduction

DRisk has been implemented using Java v1.8 on the server side and JavaScript + CSS + HTML on the client side.
For our implementation we decided not to use a database, but JSON files on which the logical maps are represented: *territories*, *continents*, the relationship *neighbourhood* between a territory and its adjacent territories and the relationship *membership* between a continent and its territories. For example,

```
{
'continents' :     ['africa', 'europe'],
'territories' :    ['italy','france', 'egypt', 'north_africa'],
'membership' :     [
                    {'name' : 'europe', 'territories' : ['italy', 'france']},
                    {'name' : 'africa', 'territories' : ['egypt', 'north_africa']}
                   ],
'neighbourhood' : [
                    {'name' : 'italy', 'territories' : ['france, 'egypt']},
                    {'name' : 'north_africa', 'territories' : ['egypt']},
                    {'name' : 'egypt', 'territories' : ['north_africa', italy']},
                    {'name' : 'france', 'territories' : ['italy']}
                   ]
}
```

The aesthetic view of the map is done by SVG files. The territory cards and the maps are dynamically created.
DRisk comes with three pre-loaded maps, each with a different difficulty level (easy, medium, hard), but the user can also upload his custom map. For a better understanding on how to create and upload a custom map, see How to upload a custom map.

5.2 Custom exceptions

In our project, we decided to use two types of custom exceptions:

- **RequestNotValidException:** This exception represents the event of a not valid request sent from the client to the server. More specifically, this exception is thrown when the client tries to do something that it is forbidden. For example, the player tries to place more tanks than the ones available or when the player tries to attack an enemy territory from a territory of his own which has only a tank, et cetera...

- **SyntaxNotValidException:** This exeption represent the event of the player that tries to create a map without following the rules described here.

5.3 Libraries

5.3.1 GSON library

We imported GSON library because it provides a simple way to build a JsonObject to be inserted in the payload of a HttpResponse as well as a way to parse a JsonObject read in the payload of a HttpRequest.

5.3.2 AJAX library

We decided to use this library because it implements an easy way to send POST and GET requests to the front controllers asynchronously.
Moreover, the communication with the server is based on sending JSON messages, which are parsed with the help of the JsonHelper class, located in the Technical Service package.

5.4 Frameworks

5.4.1 Spring MVC

We decided to use Spring MVC because it provides a simple way to implement two front controllers, the LobbyController and the GameController, which receive all the requests from the clients and delegate them to the right components.
This framework also provides a simple way to create a JSON response to be sent back to the client, thanks to Spring annotations.
For example, @ResponseBody maps the HttpRequest body to a transfer object (JsonObject) enabling automatic deserialization of the request body onto a Java object.

5.5 Techologies

5.5.1 Server Send Event (SSE)

As the Risk game is a turn-based game, it is necessary to notify all the clients when something happens in the game. For example, if a player successfully conquers a territory, all the others need to know it. For this reason, the clients must send a request to the server periodically (polling), but as this is inefficient, we decided to use SSE because it enables the clients to receive

automatic updates from a server via HTTP protocol. As an example, the code below shows how to enable this mechanism on the client side

```
function startRequest() {
        var source = new EventSource('endpointURL');
        source.onmessage = function(event) {
            // do something
                }
        }
}
```

while the code below shows how the server sends events to the clients.

```
@GetMapping("/endpointURL")
public SseEmitter yourFunction() {
        SseEmitter emitter = new SseEmitter();
        JsonObject result = new JsonObject();
        // fill RESULT with what you want
        try {
                emitter.send(result);
        } catch (IOException ex) {
                emitter.completeWithError(ex);
        }
        emitter.complete();
        return emitter;
}
```

5.6 SonarQube

We have monitored continuously this project with SonarQube, keeping always ***A*** level in bugs, vulnerabilities and code smells.

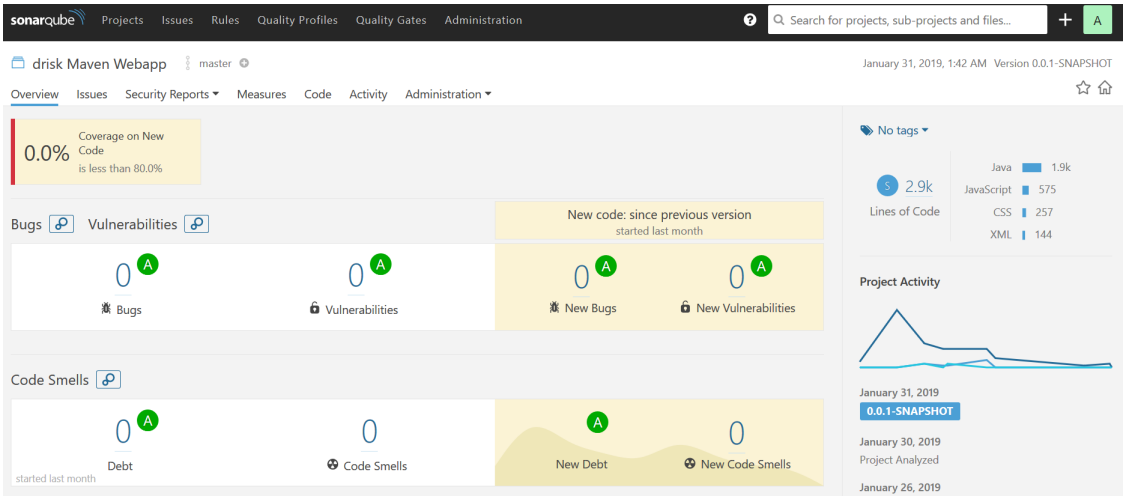

Figure 23: SonarQube screenshot

The are four bugs which are false positive, because there is no need to implement hash code methods for those classes, as they are never used in data structures which are based on hash functions.

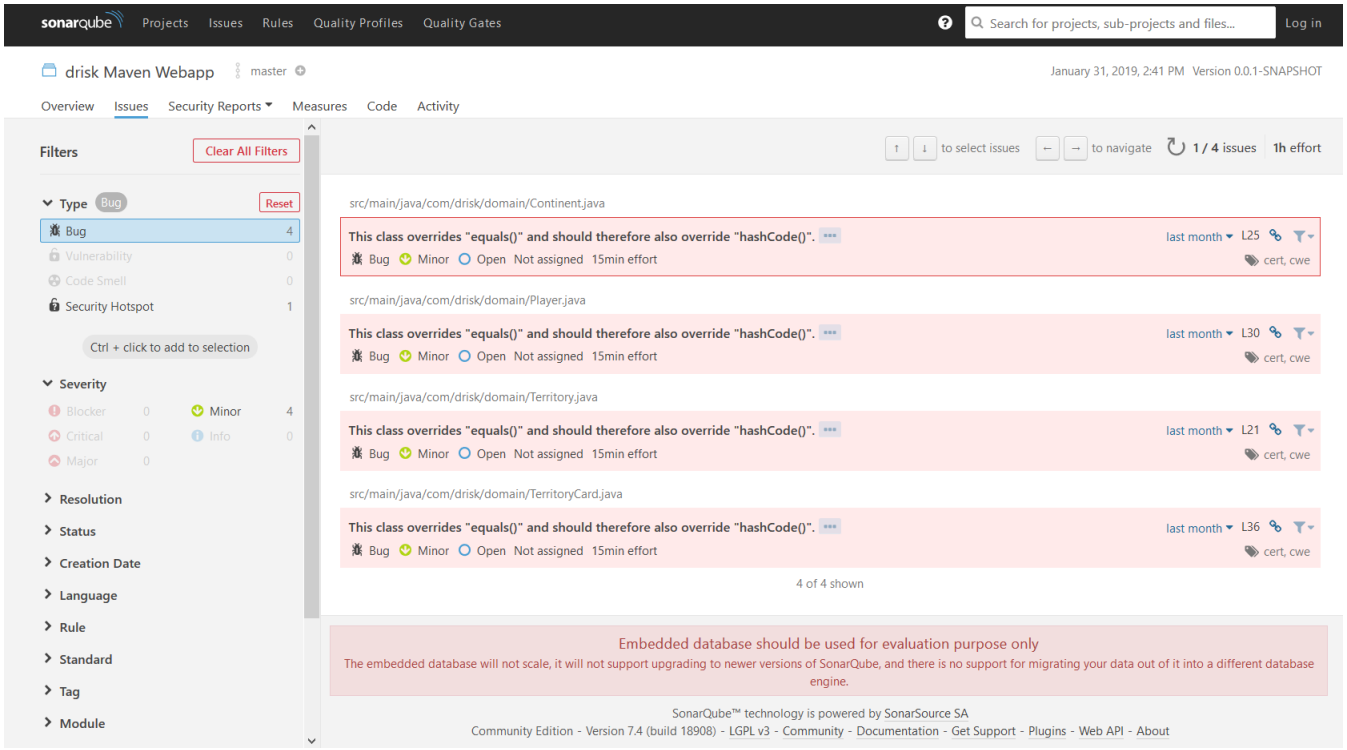

Figure 24: SonarQube screenshot with bugs

6 User manual

6.1 How to play

The game can be played in a Local Area Network or directly online, using a modern browser, but Microsoft Edge and Internet Explorer do not work properly.
The game supports only a match at once. So there can't be multiple users playing in different matches.
If two players want to play on a single computer, they must use two different browsers (e.g Chrome, Firefox), because a player is identified using a sessions mechanism.
The game page **MUST NOT** be reloaded using the browser button or F5, otherwise the player will leave the game and, consequently, lose.

6.1.1 Online

The game supports an online version at *https://drisk.herokuapp.com/*.
Even if there should not be any problems with the server, if something goes wrong, please contact Matteo Paolella *(m.paolella1@campus.unimib.it)* who will restart the server for you.

6.1.2 LAN

To play the game locally, you have to clone the project at *https://github.com/UnimibSoftEngCourse1819/progetto-risiko-1-team1.git* (using HTTP) or at *git@github.com:UnimibSoftEngCourse1819/progetto-risiko-1-team1.git* (using SSH) and import it on Eclipse. Then, follow these steps to start the server with Eclipse:
Right click on the project, move the mouse over **Run as** and then click on **Run Configurations...**.

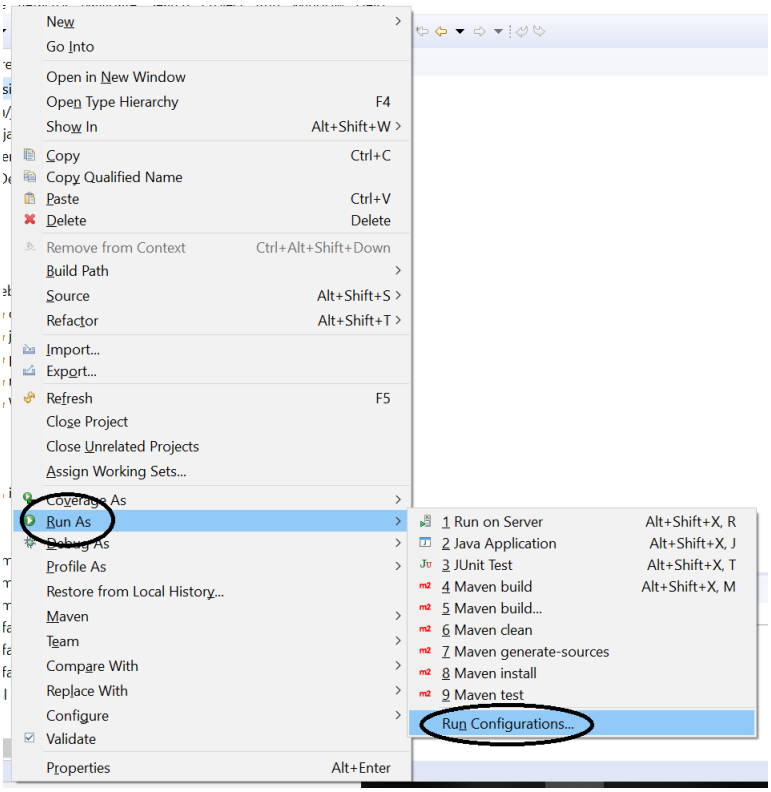

Figure 25: Run project in Eclipse

Now, write *tomcat7:run* in **Goals**, and then click on **Run**.

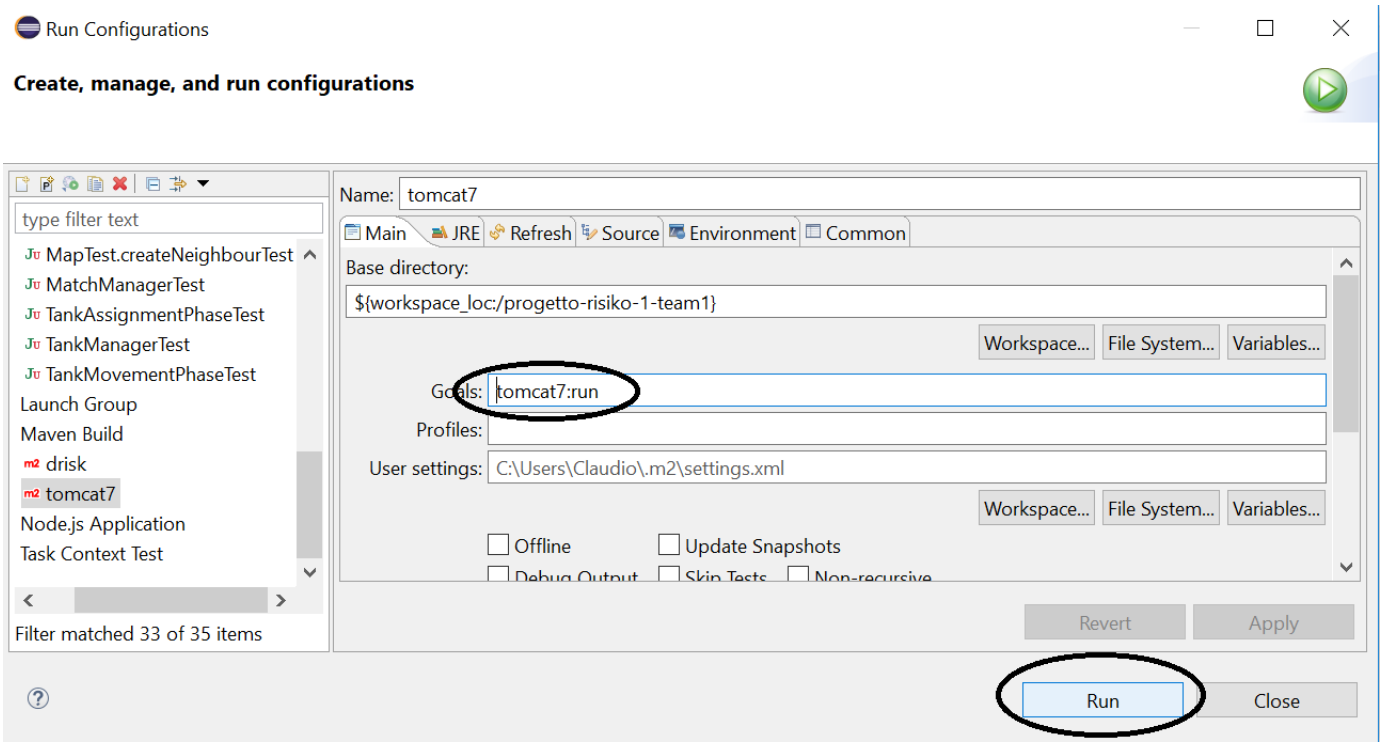

Figure 26: Run Configurations in Eclipse

After that, the following screen will appear in the console. Wait until the server is up.

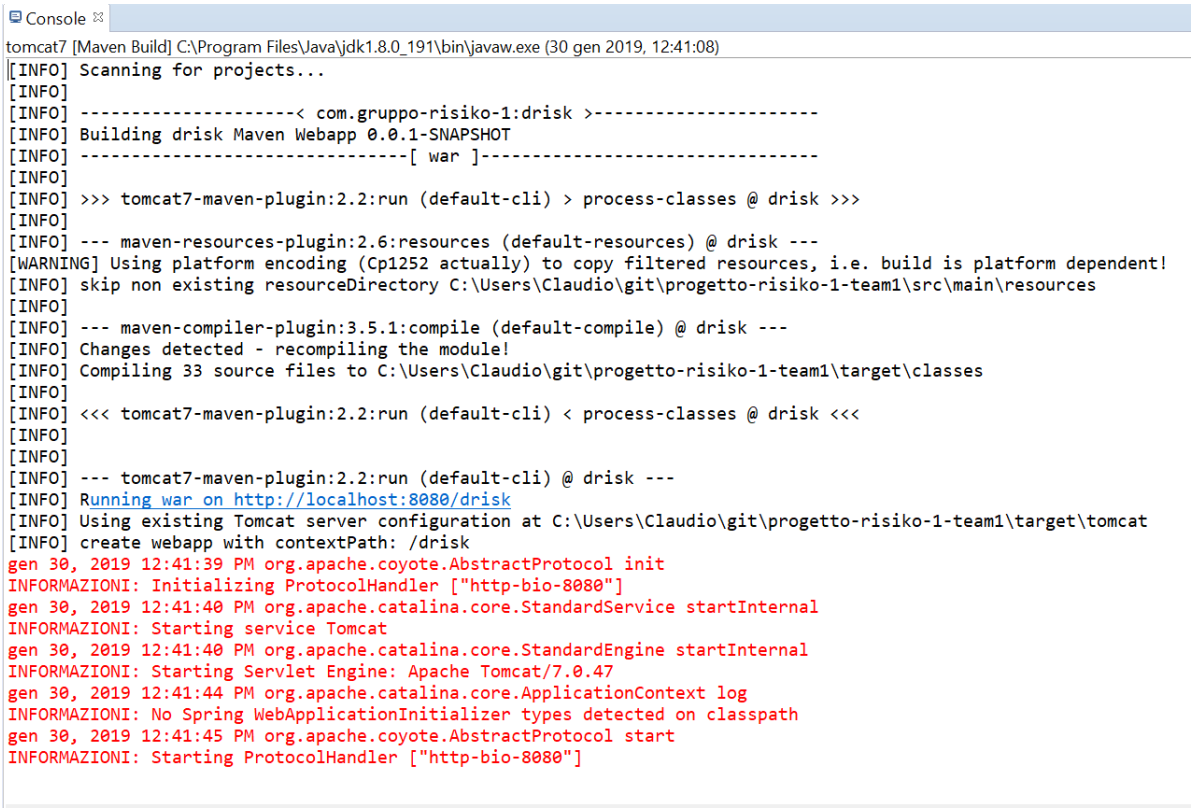

Figure 27: Tomcat7 is now ready

When the server is ready, open *cmd.exe*, run the command *ipconfig* and search for your IPv4 address.

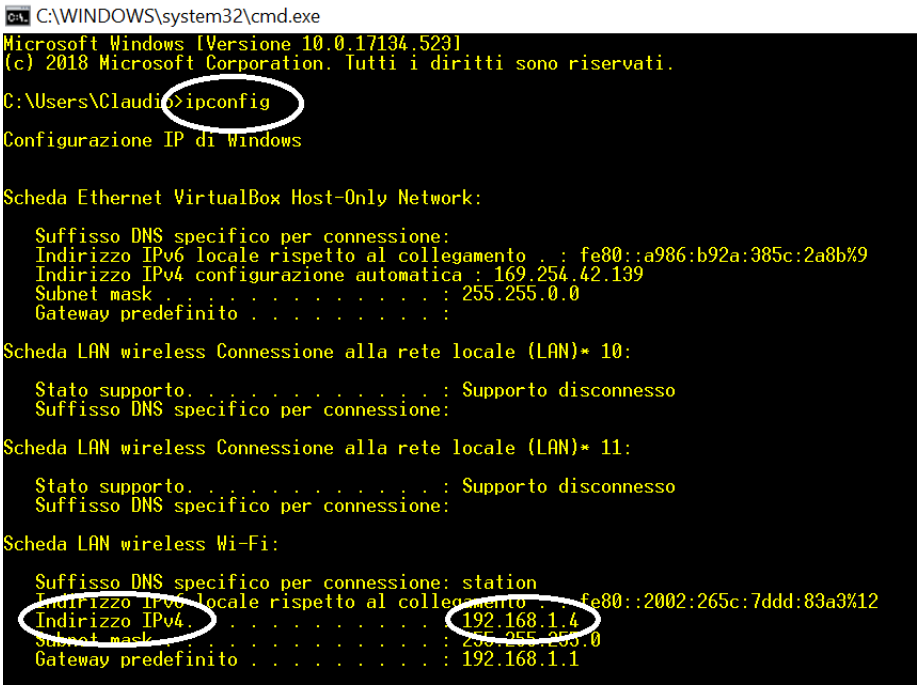

Figure 28: Cmd display with local ip

Then open Firefox or Chrome and paste the following URL:

yourIPv4Address:8080/drisk

6.2 How to upload a custom map

If the user wants to upload a custom map, he must know how to write an SVG file and a JSON file.

In particular, the JSON file must follow these schema as example:

```
'continents' :    ['africa', 'europe'],
'territories' :   ['italy','france', 'egypt', 'north_africa'],
'membership' :    [
                   {'name' : 'europe', 'territories' : ['italy', 'france']},
                   {'name' : 'africa', 'territories' : ['egypt', 'north_africa']}
                  ],
'neighbourhood' : [{'name': 'italy', 'territories': ['france','egypt']},
                   {'name' 'france': , 'territories': ['italy']},
                   {'name': 'egypt', 'territories': ['north_africa', 'italy']},
                   {'name': 'north_africa', 'territories': ['egypt']}
                  ]
```

Pay attention NOT to insert the begin and end curly brackets of the JSON file.

The SVG file is used to create the aesthetic components of the map and it must follow some rules. Each territory is represented with a PATH tag of class "country", in which there's an attribute ID with the same name of the territory specified in the JSON file. This attribute ID is used by the client page to fill the territory with the same color of the owner. For example:

```
<path class="country" id="italy" d="..."></path>
```

In addition, for each territory there should exist a tag TEXT with an attribute ID with the same name of the territory specified in the JSON file concatenated with "_text". This tag is used by the client page to show the number of tanks on that territory. For example:

```
<text id="italy_text" x="..." y="..."></text>
```

Note that DRisk checks if the logical map is written correctly, but it **does NOT** check the consistency between the SVG file and the JSON file. It is assumed that they are consistent with each other.

7 Future ideas

In a newer version of DRisk, there could be the following improvements and features implemented:

- Allow multiple matches to be played at the same time concurrently.
- An updated game page with better graphics.
- Allow players to use a chat embedded in the game page to talk to each other.
- A quicker and smoother interaction with the game page.
- Faster responses by the server.
- Allow users to log in to maintain their statistics about losses and wins, as well as a ranking for the best players.
- Allow the users to create a map using an embedded visual design editor instead of an external editor.
- Allow the users to create the logical map without forcing them to write a JSON text by hand.
- The game has sounds effects and a music in the background.
- Make it a hazard game where players can bet their wins if they want.