Projet 2019

Risk

Auteurs

Nom	Adresse
Boursier Louis	louis.boursier@etu.univ-nantes.fr
Filaudeau Éloi	filaudeau.eloi@etu.univ-nantes.fr
Lasherme Loïc	loic.lasherme@etu.univ-nantes.fr
Nantier Matthias	matthias.nantier@etu.univ-nantes.fr

Introduction

Le but de ce projet est de mettre en oeuvre les connaissances développées dans les modules d'analyse et de conception ainsi que de test logiciel. Dans cet objectif, nous devons développer un logiciel réparti adaptant le jeu de société Risk.

Le jeu du Risk est à la base un jeu de plateau qui se joue de 2 à 6 joueurs. À tour de rôle, chaque joueur va définir un territoire sur lequel il placera une unité jusqu'à remplir chaque zone. Une fois cela fait, les joueurs pourront compléter les forces de leur armée sur les zones occupées par celle-ci afin de se libérer des unités en main. Ensuite, le jeu peut enfin commencer et le joueur qui possédera tous les territoires se verra remporter la partie.

Nous avons donc pour objectif de comprendre, analyser, modéliser, implémenter et tester ce jeu décrit brièvement ci-dessus.

Organisation du document

Analyse du domaine

Introduction

Objectif

Organisation du chapitre

Ce chapitre constitue le premier livrable d'une série de trois chapitres destinés à fournir une analyse et une conception par objets complètes répondant au cahier des charges qui nous a été fourni. Ce document présente l'ensemble de la démarche suivie ainsi que les résultats obtenus lors de la phase de l'analyse du domaine de notre système. Il se décompose en plusieurs parties.

Dans la première partie, nous présentons de manière détaillée l'ensemble des cas d'utilisation que nous avons dégagés lors de l'analyse. Nous utiliserons pour cela le canevas proposé par Cockburn que nous compléterons par des instantanés ainsi que par des post-conditions exprimées en OCL (Object Constraint Language) et quelques scenarii. Cette partie constitue une étape clé de la phase de l'analyse du domaine.

Dans la deuxième partie, nous présentons le diagramme de classes métiers (i.e. diagramme de classes au niveau analyse) que nous avons construit à partir de l'analyse réalisée. Ce diagramme fournit une vue statique et synthétique du domaine de notre projet. Cette partie constitue également une étape clé de la phase de spécification des besoins.

Dans la troisième er dernière nous fournissons le dictionnaire des données que nous avons construit suite à l'analyse du domaine. Il s'agit d'un listing de l'ensemble des termes relatifs au domaine étudié ainsi que leur définition précise.

Cas d'utilisation

Mise en place d'un jeu

Use Case Jouer

Use Case Template. Copyright (c) 2004-2005 TechnoSolutions Corporation

Use Case: Jouer

Id: UC- 1

Description Jouer une partie de RISK.

Level: High Level Summary

Primary Actor Jeu

Supporting Actors Joueur

Stakeholders and Interests

Pre-Conditions

Au moins 3 joueurs sont présents.

Post Conditions

Success end condition

Le jeu s'est déroulé suivant les règles jusqu'à l'obtention d'un joueur gagnant.

Failure end condition:

Le serveur tombe en panne. Le réseau tombe en panne. Un des joueurs ne réalise pas d'action et bloque le jeu.

Minimal Guarantee

Les joueurs sont informés par message si une erreur survient.

Trigger

Un groupe de joueurs veut faire une partie.

Main Success Scenario

1. Les joueurs mettent en place la partie (Voir UC2: Initialisation)
2. Les joueurs jouent tour à tour (Voir UC3: Tour)
3. On répète 2. tant qu'il y a plus d'un joueur sur les territoires.
4. Le dernier joueur est le gagnant. La partie est terminée.

Frequency: Une seule fois

Use Case Initialisation

Use Case Template. Copyright (c) 2004-2005 TechnoSolutions Corporation

Use Case: Initialisation

Id: UC- 2

Description Initialisation et mise en place d'une partie de RISK

Level: User Goal

Primary Actor Jeu

Supporting Actors Joueurs

Stakeholders and Interests Pioche Défausse Carte Territoire Continent

Pre-Conditions

Au moins 3 joueurs sont présents.

Post Conditions

Success end condition

Le jeu a été correctement mis en place selon les règles de RISK. Chaque territoire est en possession d'un joueur. Chaque territoire supporte une ou plusieurs armées du joueur possédant ce territoire. La pioche est remplie avec toutes les cartes du jeu dans un ordre aléatoire.

Failure end condition:

Un des joueurs ne place pas une de ses armées lorsque c'est à son tour de le faire. Il bloque l'initialisation.

Minimal Guarantee

La partie ne peut pas continuer tant que le jeu n'a pas été correctement initialisé.

Trigger

Les joueurs sont prêts à jouer.

Main Success Scenario

1. On distribue à chaque joueur des unités d'armé selon le nombre de joueurs total. 35 unités pour 3 joueurs, 30 pour 4, 25 pour 5 et 20 pour 6.
2. L'ordre de jeu des joueurs à l'initialisation et lors des tours qui suivront est déterminé aléatoirement avec un lancé de dé 6. Chaque joueur lance le dé et le joueur ayant le plus grand chiffre commence. Les joueurs jouent ensuite dans le sens des aiguille d'une montre après le premier joueur **Extension 1**.
3. Les joueurs posent tour à tour une de leur unité d'armé sur les territoires non occupés par un autre joueur, avec obligation de placer sur un territorie innocupé si possible. **Extension 2**.
4. Répéter 3. tant qu'il reste des armées aux joueurs.
5. On mélange toutes les cartes du jeu et les met dans la pioche.

Extensions

1. Si il y a une égalité entre les premiers, ces joueurs rejouent jusqu'à ce qu'il y ai un gagnant.

2. Si le territoire était innocupé par une armée, le joueur qui y pose l'armée se l'approprie.

Frequency: Une seule fois

Use Case Tour

Use Case: Tour

Id: UC- 3

Description Le tour d'un joueur lors d'une partie de RISK

Level: User Goal

Primary Actor Joueur

Supporting Actors Jeu

Stakeholders and Interests Pioche Défausse Carte Territoire Continent Main

Pre-Conditions

C'est au tour du joueur de jouer. La partie n'est pas terminée.

Post Conditions

Success end condition

Le joueur a déployé ses armées gagnées dans ce tour lors de la phase de renfort sur son territoire. Il a éventuellement attaqué d'autre joueurs et renforcé ses territoires.

Failure end condition:

Le joueur ne réalise aucune action.

Minimal Guarantee

Le joueur ne peut pas perdre lors de son tour.

Trigger

L'initialisation est terminée. Le joueur précédent le joueur courant a fini son tour.

Main Success Scenario

1. Le joueur entre en phase de renfort.
    a. Le joueur reçoit des armées selon les territoires qu'il possède. Nombre d'armées = Max(3,arrondiInférieur(nombreDeTerritoirePossédés/3)). **extension 1**
    b. Le joueur peut échanger des jeux de 3 cartes de sa main contre des armées. Voir UC4 Echange de cartes.
    c. Le joueur pose ses nouvelles armées sus ses territoires occupés de la façon qu'il souhaite.
2. Le joueur entre en phase d'attaque. **extension 2**
    a. L'attaquant choisi un de ses territoires depuis lequel il attaque, possédant au moins 2 armées, et un territoire d'un autre joueur à attaquer avec lequel il possède une voie maritime directe ou une frontière partagée.
    b. Le joueur attaquant et le joueur défenseur mettent en jeu un nombre de dés correspondant

au nombre d'armées qu'ils veulent faire combattre. Ce nombre ne doit pas dépasser:

i. Le nombre d'armées présentent sur le territoire

ii. 3 pour l'attaquant

iii. 2 pour le défenseur

c. On lancé les dés. Le dé le plus faible de chaque paire de dé attaquant/défenseur fait perdre une armée à son possesseur. Dans le cas d'une égalité, c'est le défenseur qui gagne. Le territoire attaquant ne peut descendre en dessous une armée. **extension 3**

3. Le joueur rentre en phase de fortification. **extension 4**

a. Le joueur peut déplacer une fois entre 1 et n-1 armées d'un de ses territoires à un de ses autres si le chemin entre les deux est composé uniquement de ses propre territoires.

4. Le tour du joueur prend fin et laisse place au suivant.

Extensions

1. Si le joueur possède tous les territoires d'un continent, il reçoit 5 armées supplémentaires.

2. Le joueur peut passer cette phase si il le veut.

3. Si le défenseur perd toutes ses armées sur le territoire attaqué, l'attaquant déplace entre 1 et n-1 armées sur ce terrtoire. Avec n le nombre d'armées présentent sur le territoire attaquant. L'attaquant devient le possesseur de ce territoire. **extension 3.1** p 3.1. Si c'était le dernier territoire du défenseur, celui-ci est éliminé de la partie et l'attaquant récupère les cartes de sa main. Si l'attaquant vient à posséder 5 cartes ou plus grâce à cela, il retourne à l'étape 1.b. avec obligation d'échange de carte.

4. Si le joueur a capturé au moins un territoire lors de la phase d'attaque, il pioche la première carte de la pioche et la met dans sa main. **extension 4.1**

4.1. Si la pioche est vide, on mélange les cartes de la défausse et on les placent dans la pioche.

Frequency: Quelques dizaines de fois par partie pour chaque Joueur.

Use Case Echange de carte

Use Case Template. Copyright (c) 2004-2005 TechnoSolutions Corporation

Use Case: Echange de carte

Id: UC- 4

Description Le procédé par lequel un joueur échange des cartes contre des armées

Level: User Goal

Primary Actor Joueur

Supporting Actors Jeu

Stakeholders and Interests Pioche Défausse Carte Territoire Continent Main

Pre-Conditions

C’est au tour du joueur de jouer. La partie n’est pas terminée.

Post Conditions

Success end condition

Le joueur a pu (ou non) échanger certaines de ses cartes contre des armées et les a placées sur ses territoires.

Failure end condition:

Le joueur ne réalise aucune action.

Minimal Guarantee

Le joueur ne peut que faire des échanges valides.

Trigger

Le joueur est en phase de renfort lors de son tour.

Main Success Scenario

1. Le joueur peut échanger des jeux de 3 cartes de la forme: **extension 1**
    a. 1 carte de chaque type: Fantassin, Cavalerie, Char
    b. 3 cartes du même type
    c. 2 cartes quelconques et une carte de type "Wild"
2. Pour chaque jeu échangé, il gagne un nombre d’armées correspondant au nombre total de jeu de carte échangés depuis le début de la partie par tous les joueurs:
    a. Premier échange: 4 armées
    b. 2: 6
    c. 3: 8
    d. 4: 10
    e. 5: 12
    f. 6: 15
    g. 7: 20
    h. n: de 5 en 5…
3. Une fois les cartes jouées, elles partent dans la défausse.

Extensions

1. Si le joueur a 5 cartes ou plus, il est obligé de faire un échange.

Frequency: Une fois par tour pour chaque Joueur. Quelques dizaines de fois par partie pour

chaque Joueur.

Modèle de classes du domaine

Diagramme de classe du domaine

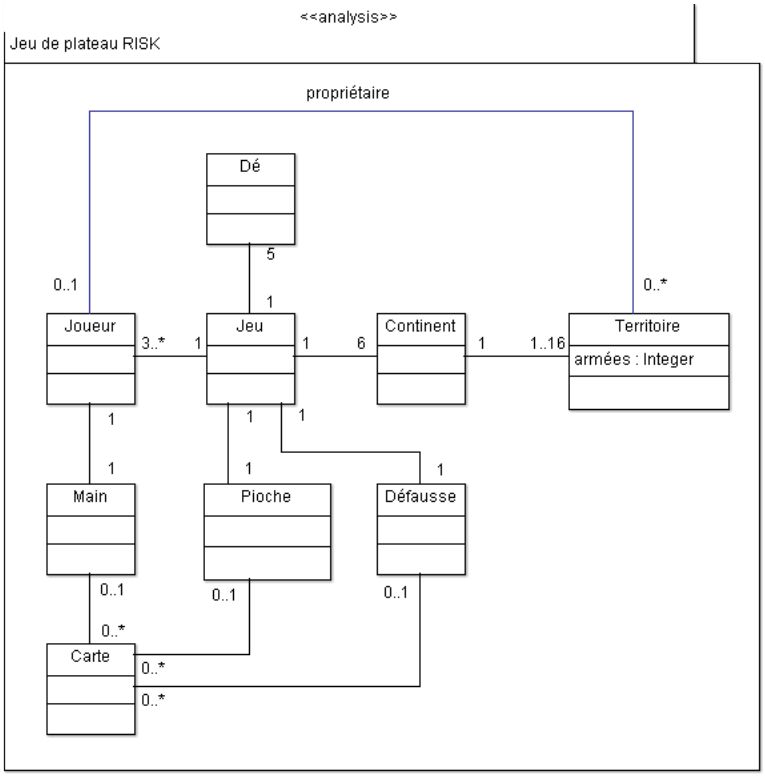
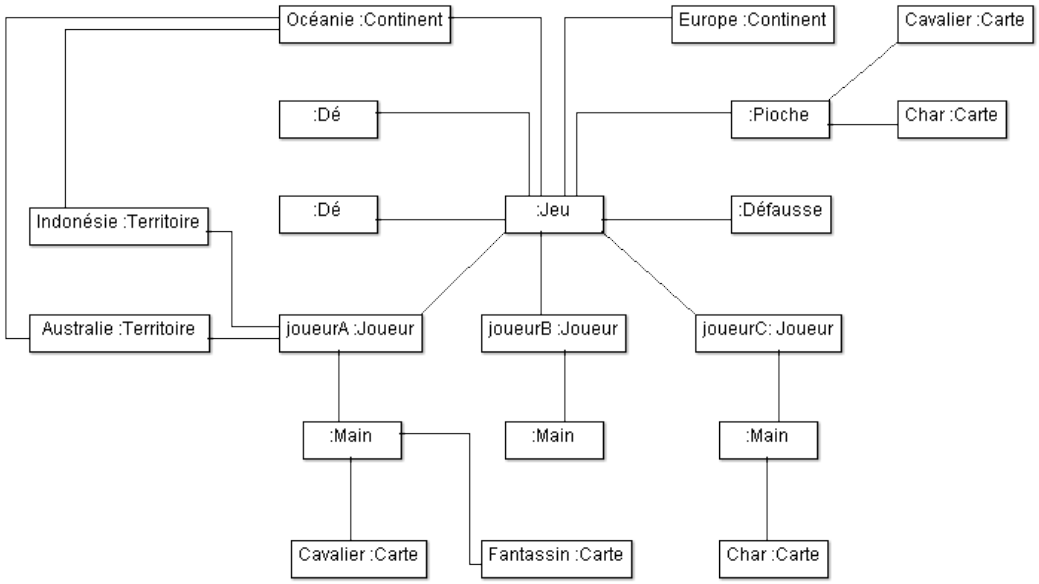
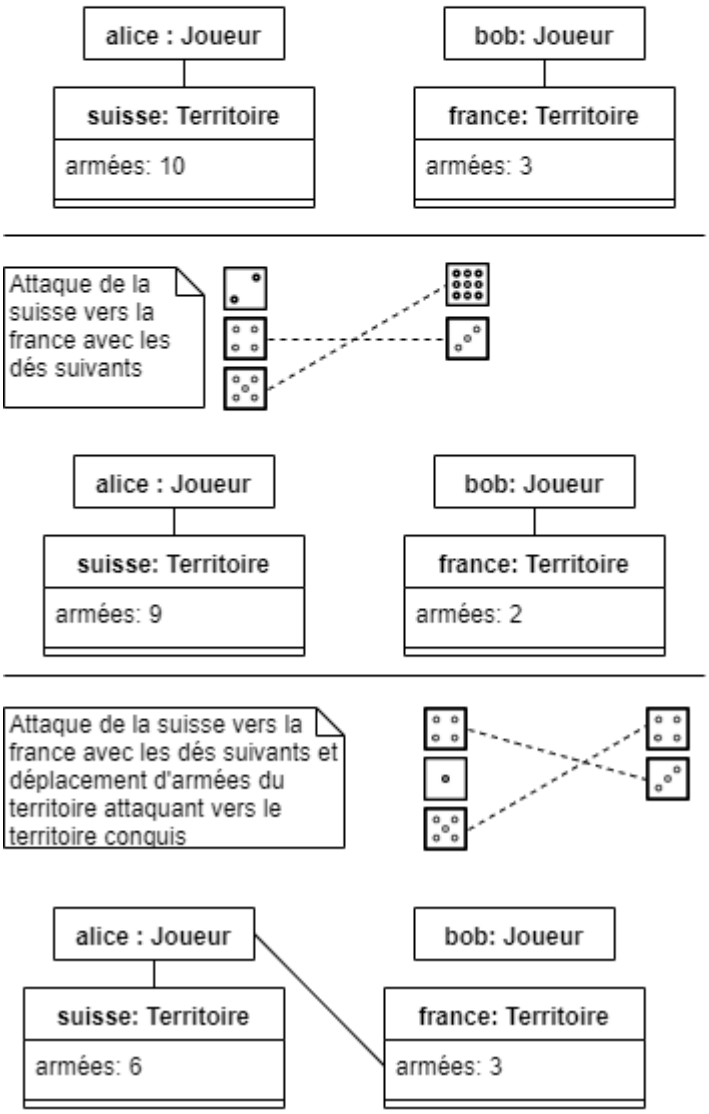


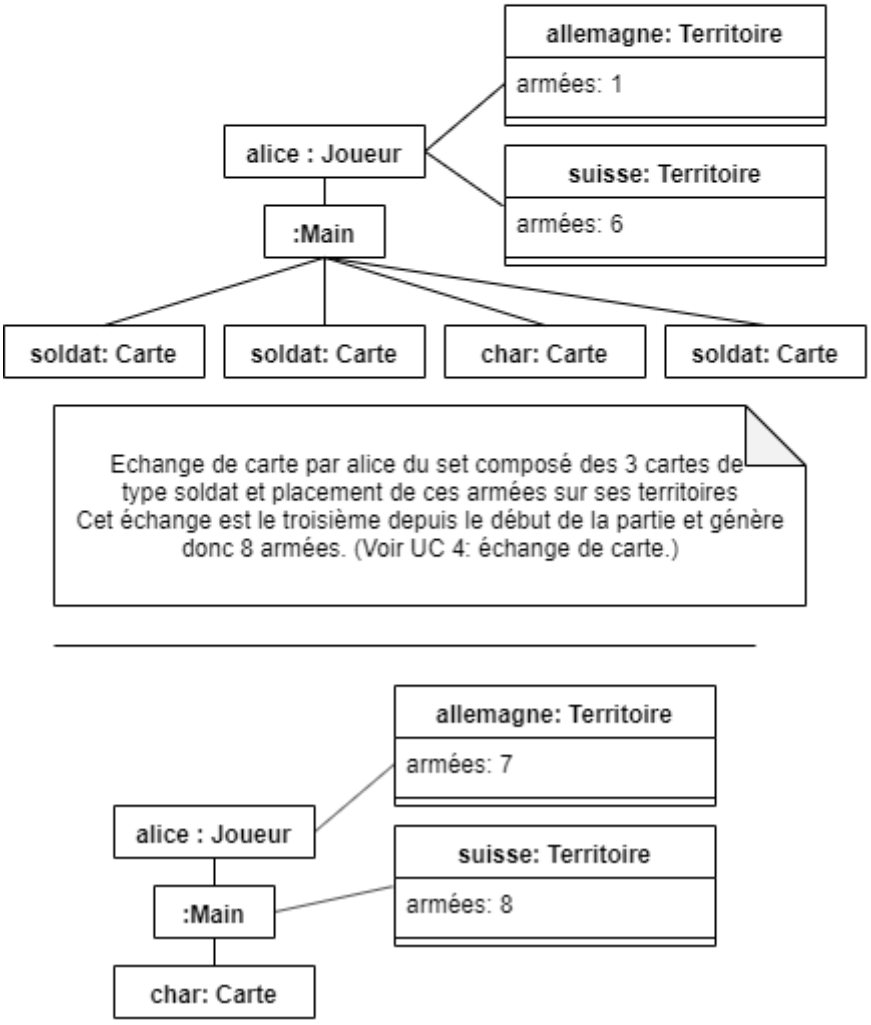
Diagramme d’objet du domaine



Action attaquer - Snapshot



Action échanger carte - Snapshot



Invariants

```
context Jeu inv:
    self.joueur->size() >= 3 and self.joueur->size() <= 6

context Continent inv:
    self.name = "Afrique" implies self.territoire->size() = 6 and
    self.name = "Asie" implies self.territoire->size() = 12 and
    self.name = "Océanie" implies self.territoire->size() = 4 and
```

```
    self.name = "Europe" implies self.territoire->size() = 7 and
    self.name = "Amérique du Nord" implies self.territoire->size() = 9 and
    self.name = "Amérique du Sud" implies self.territoire->size() = 4

context Jeu inv:
    self.joueur->size() >= 3 and self.joueur->size() <= 6

-- Ce sont des dés de 6
context De::lancer(): Integer
    post: result >= 1 and result <= 6

-- Le nombre d'armées par joueur en début de partie dépend du nombre de joueurs
context Joueur::territoire::armées->size() : Integer
    init: 35 - (Jeu::joueur->size() - 3) * 5

-- L'attaquant laissera toujours au moins un pion d'armée sur le territoire attaquant
context Jeu::attaquer(attaquant: Territoire, defenseur: Territoire, nombreAttaquant:
Integer)
    pre: attaquant.voisins->includes(defenseur) and nombreAttaquant > 0 and
         attaquant.propriétaire <> defenseur.propriétaire and
         attaquant->armées->size() > nombreAttaquant and
         nombreAttaquant <= 3
    post:  ( (attaquant->armées@pre->size() <> attaquant->armées@post->size()) or
           (defenseur->armées@pre->size() <> defenseur->armées@post->size()) ) and
           attaquant->armées@post->size() >= 1 and
           let lancesDes : Sequence(OclMessage) =
           De^^lancer() in
           lancesDes->size() = nombreAttaquant

-- Les déplacements d'armées ne laisse jamais un territoire innocupé
context Jeu::fortification(origine: Territoire, destination: Territoire, nombreUnitée:
Integer)
    pre: origine.propriétaire = destination.propriétaire and
         origine.voisins->includes(destination)
         and origine->armées->size() >= 2  and origine->armées->size() > nombreUnitée
and
         nombreUnitée > 0
    post: origine->armées@post->size() >= 1 and
          origine->armées@post->size() =  origine->armées@pre->size() - nombreUnitée
and
          destination->armées@post->size() =  destination->armées@pre->size() +
nombreUnitée

context Jeu::nombreDeRenfort(joueur: Joueur) : Integer
    pre: joueur.estEliminé = false
    post: return = Max(3, jeu->continents->territoires->iterate(each: Territoire;
                        answer: Integer = 0 |
                        if each.proprétaire = joueur then answer = answer + 1))


context Jeu::partieTerminée() : Boolean
```

```
    post: return = jeu->continents->territoires->isUnique(propriétaire)
```

Dictionnaire de données

Term	Short Description	Meaning
Unité	Représente un pion d'armée.	Les unités permettent d'attaquer et de défendre les territoires d'un joueur. On considère qu'il y a autant d'unités disponbile qu'on en a besoin.
Territoire	Zone possédée par un joueur.	Plus petite unité de terrain du jeu. Le joueur possédant des troupes sur cette zone possède donc le territoire. Chaque territoire est accessible par un autre, grâce à un ou plusieurs chemins (terrestres ou maritimes).
Continent	Ensemble de Territoire.	Unité de terrain du jeu. Un continent n'est rien d'autre qu'un ensemble de territoire. Si un joueur possède chaque territoires du continent, alors il possède le continent.
Carte	Cartes : Fantassin, Cavalier, Char, Wild. Les quattres différentes cartes du jeu.	Permettent de faire certains échanges qui octroient un certain nombre d'unités au joueur qui les a marchandées.
Main	Ensemble des cartes que possède le joueur.	Chaque tour le joueur pioche des cartes et les range dans sa main.
Pioche	Ensemble des cartes piochable par les joueurs.	C'est ici que toutes les cartes du jeu qui n'ont pas encore été piochées ou jouées se trouvent. Ces cartes servent à faire des échanges pour gagner des unités. Il y a 44 cartes unités et 2 cartes Joker.

Défausse	Ensemble des cartes jouées par les joueurs.	C’est ici que toutes les cartes du jeu qui ont été précédemment jouées par les joueurs ce trouvent. Une fois la pioche vide, c’est les cartes présentent dans cette défausse qui sont mélangées puis utilisées pour faire la nouvelle pioche.
Dés	Des dés de 6.	Ils permettent de jouer les affrontement entre les joueurs quand un territoire en attaque un autre. C’est là que la partie random du jeu du Risk intervient.
Plateau	Plateau du jeu RISK.	Le plateau contient tous les éléments du jeu tels que la pioche, les joueurs, armées…

Requirements Specification

Introduction

Purpose

The purpose of this document is to describe the requirement specifications for the project «Risk» for software engineering students.

The intended audience of this specification includes the prospective developers of the tool, as well as the technical assessment personnel.

Document Conventions

None so far.

Intended Audience and Reading Suggestions

Project Scope

The software system to be produced is a simplified version of the Hearthstone online game, which will be referred to as «Risk» thorough this document.

The Risk system will allow players from different locations to confront each-other in short and intensive games.

References

1. IEEE Standard 830-1993: IEEE Recommended Practice for Software Requirements Specifications

Overview

The rest of this document contains an overall description of the Risk software system (section Overall Description), the specific functional requirements (section System Features), and the non-functional requirements for the system (see Other Nonfunctional Requirements.

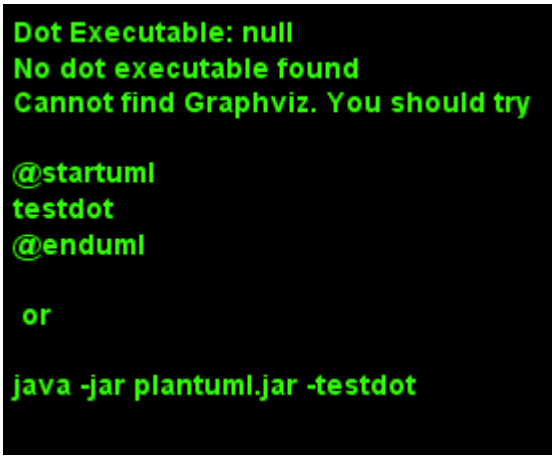
Overall Description

Product Perspective

Hearthstone is a card game where two players confront each-other. The Risk software should allow players that are connected to the Internet to use their connected devices to play. Thus, «Risk» is an online, electronic version of the card game.

While the system is distributed and organized in different components, players should perceive it as a single piece of software. Figure UML Deployment Diagram presents the overall architecture of the software. Players interact with a Web Client, which uses the HTTP protocol to communicate

with (at most) one Game Server. Servers use TCP/IP to communicate with a Database Management Server, which stores all software data.



UML Deployment Diagram

Product Functions

The Risk software must provide two main functions:

1. Game creation: allowing two players to discover each other and start a game.
2. Game play: allowing players to actually play Risk until the victory of one of them.

User Classes and Characteristics

The Risk software has only one class of user: players. Players may have different levels: beginners, intermediate, or expert. However, independently from their level, players should use the same user interface to play against each other.

Operating Environment

The Risk software should operate on any popular and recent operating system: Linux, Windows, or MacOS. The Web Client should operate on any recent web browser: Firefox, Chrome, Safari, or Edge.

Design and Implementation Constraints

1. The Game Server must be developed in Java (version 1.8), using the Spring Framework.
2. The Client must be developed in TypeScript (version 3.1), using the Angular Framework.
3. All software artifacts must use a building too: Maven or Groovy for Java, npm for TypeScript.
4. Dynamic tests must use JUnit (version >= 5.0) and Jasmine (version >= 3.5.0).
5. The code must log its main operations using SLF4J.

Verification Constraints

1. Test Doubles must be used to test each component independently.
2. Each unit test must describe its intention.

User Documentation

No user documentation is required for the first version of the software.

Assumptions and Dependencies

None until now.

External Interface Requirements

User Interfaces

Hardware Interfaces

The software does not interact directly with any hardware device.

Software Interfaces}

The client part of the software must operate on web browsers, whereas the server part must interact with a database through the Java Persistence API (JPA).

Communications Interfaces

Communications between the client and the game server must use Websockets.

System Features

Game initialization

The Risk software must allow the setup of a game with two players and automatically prepare and distribute cards.

Description and Priority

See Chapter [domain] (domain analysis) for further details.

Stimulus/Response Sequences

Functional Requirements

Game play

The Risk software must allow two players to play against each other until a winer is settled. See Chapter [domain] (domain analysis) for further details.

Other Nonfunctional Requirements

Performance Requirements}

1. The game must be *playable*, meaning that users must have fast feedback for their actions and delays due to communications/connection problems must be correctly held.
2. The Web Client must be able to execute on a personal computer with 4GB of RAM.

Safety Requirements

Security Requirements

Software Quality Attributes

Extensibility

The software must be extensible, it must be easy for developers to add new cards and heroes to the game.

Maintainability

1. The software must be readable and easy to maintain.
2. The Java source must respect Google's guidelines: https://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Business Rules

Other Requirements

Appendix A: Glossary

Appendix B: Analysis Models

See Chapter [domain] (domain analysis) for further details.

Appendix C: To Be Determined List

Spécification des composants

Ce livrable correspond à la "conception préliminaire" du projet. Il comprend la division de la solution en différents composants, l'explication des fonctionnalités attendues de chaque composant, la spécification des interfaces fournies par chaque composant, ainsi que des diagramme de séquence qui valident ces interfaces.

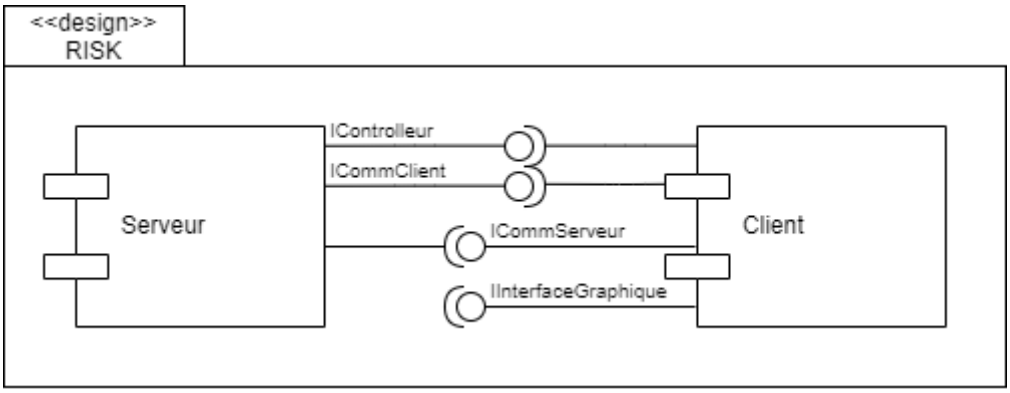
Objectif

Etablir les frontières du système et le diviser en composants. Définir les composants et interfaces, leur portée et leurs responsabilitées. Spécifier les opérations fournies par les compsoants.

Organisation du chapitre

Cette section décrit le contenu du reste du chapitre et explique comment le document est organisé.

Description des composants

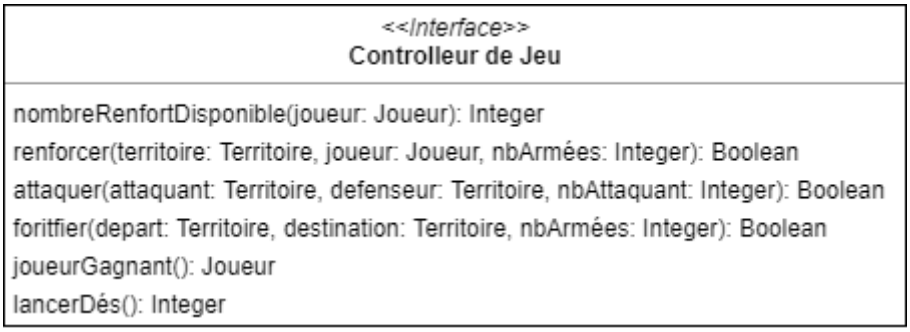


Le composant `Serveur`

Le serveur est responsable de la communication entre client (ordonnancement des tours et distribution des messages). Il vérifie également les opérations de chaque client (est-ce que l'opération est valide ? est-ce bien à ce joueur de jouer ? est-ce que elle respecte les règles de RISK et les cas d'utilisation?). Il assure les parties critiques telles que le lancé de dés (éviter la triche côté client). Il a aussi le rôle de maître de jeu dans le sens où il est chargé de mettre à jour le plateau selon les actions des clients.

Spécification des interfaces

Spécification de l'interface Controlleur de Jeu



Spécification de l'interface Communication Client

Spécification de l'interface B

Le composant `Client`

Le client fourni une GUI aux joueurs pour réprésenter l'état du jeu et effectuer des actions. Cette interface est aussi responsable pour transmettre et réceptionner les messages en direction et en provenance du serveur.

Spécification des interfaces

Spécification de l'interface Interface Graphique

Spécification de l'interface Communication Serveur

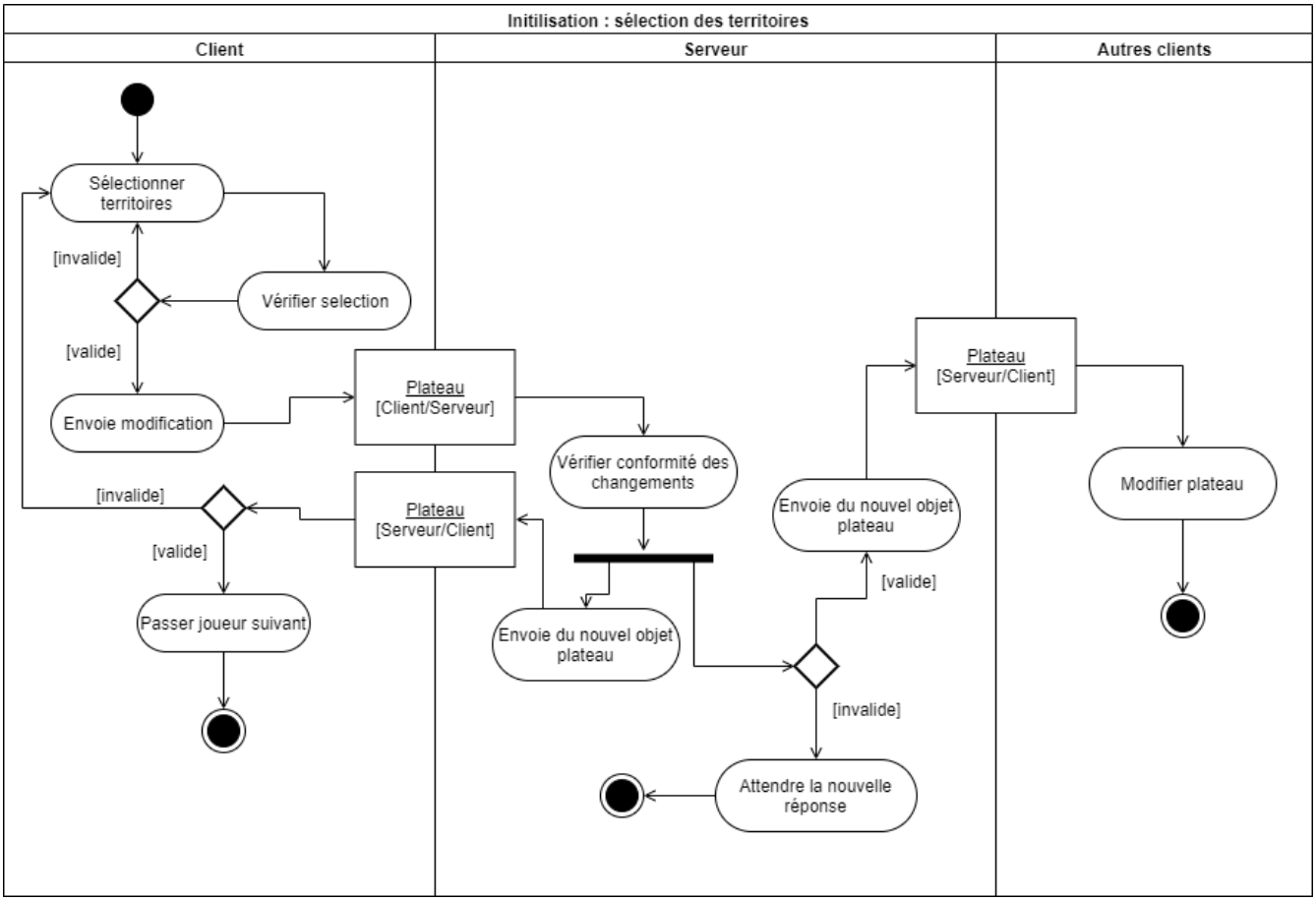
Conception détaillée

Objectif

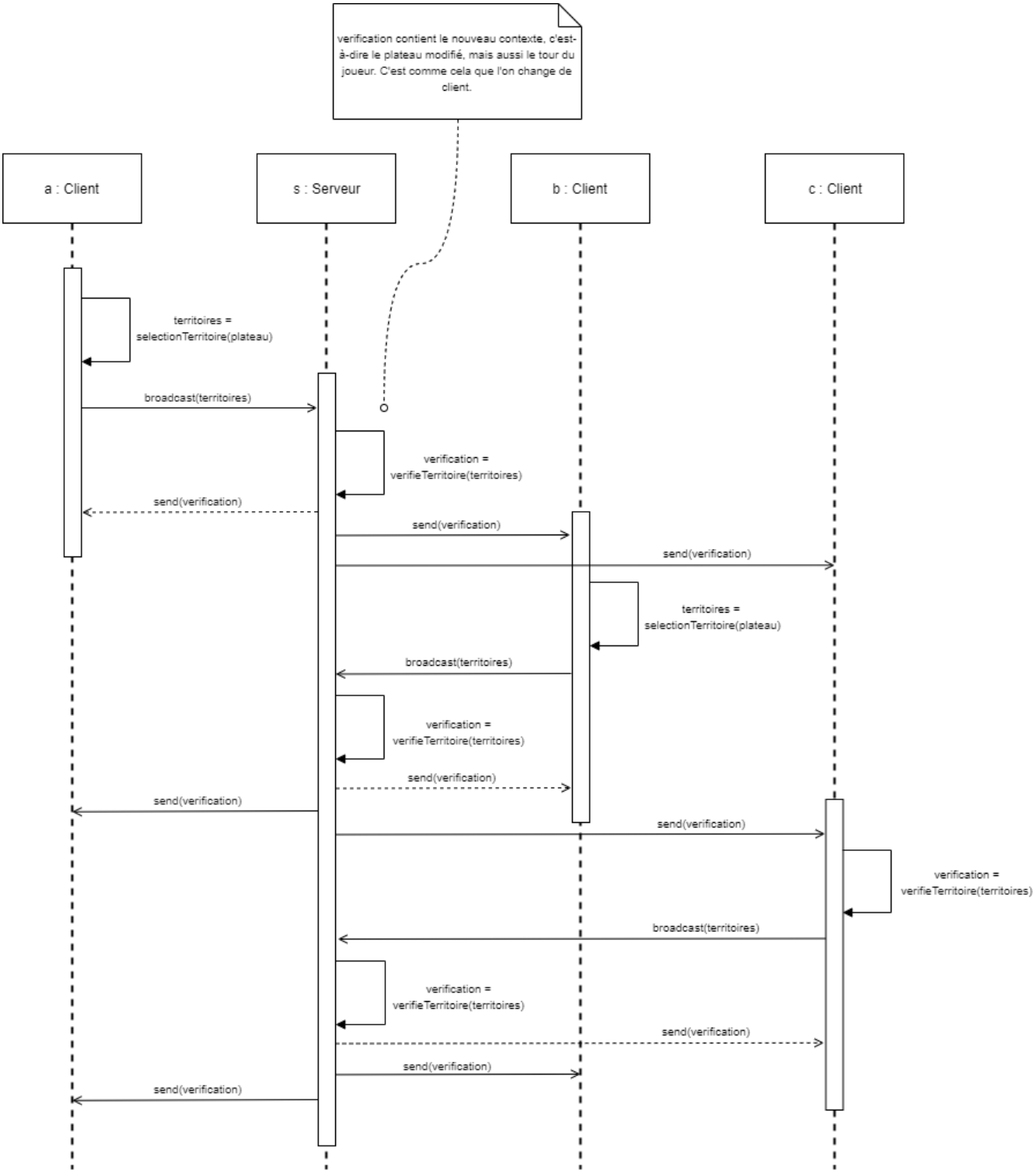
Spécifier la structure interne et le comportement des composants en assurant une certaine qualité (évolution, performance, portabilité, testabilité, etc).

Description de l'initialisation d'une partie

Algorithme

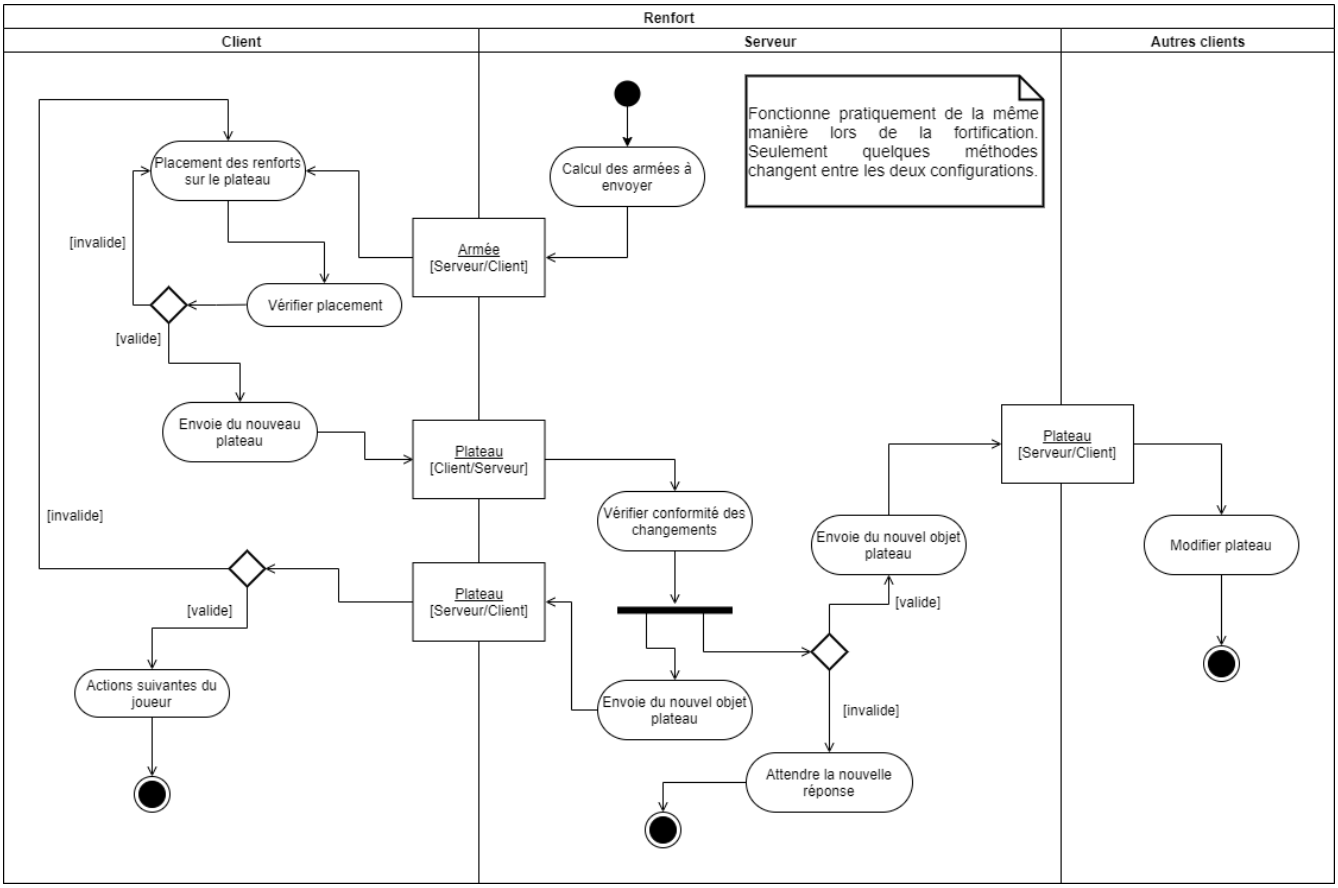


Scenario optimal

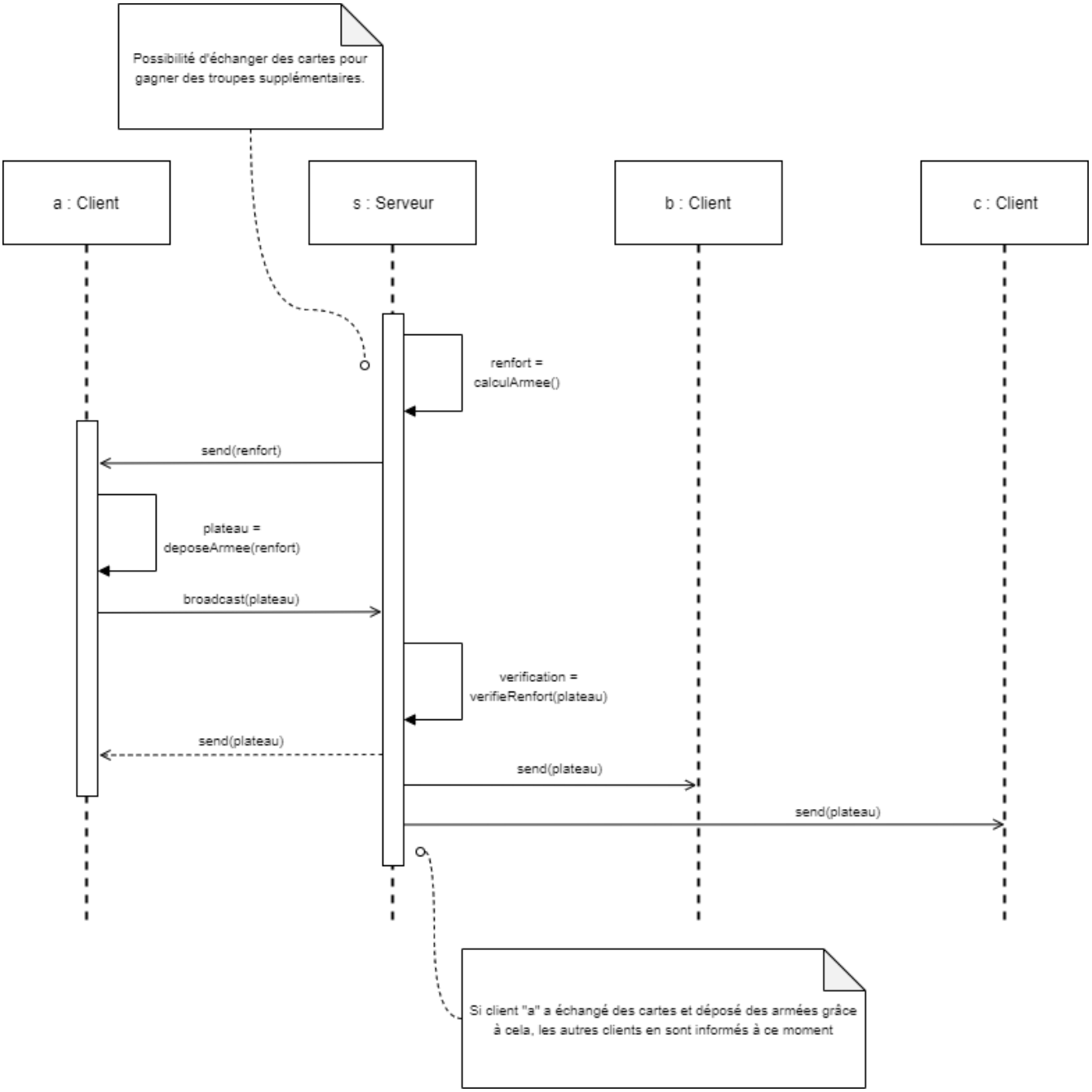


Description de la phase de fortification

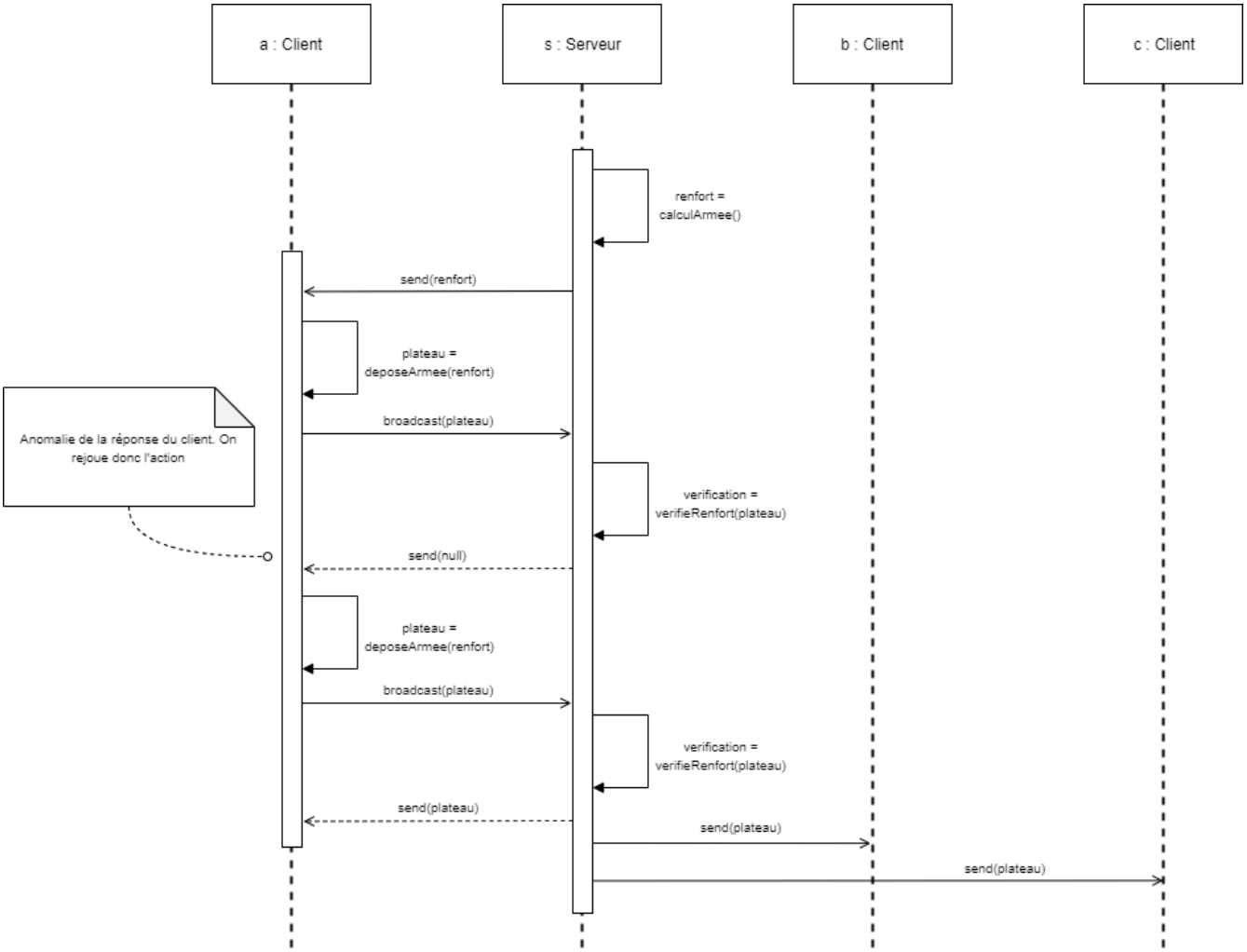
Algorithme



Scenario optimal

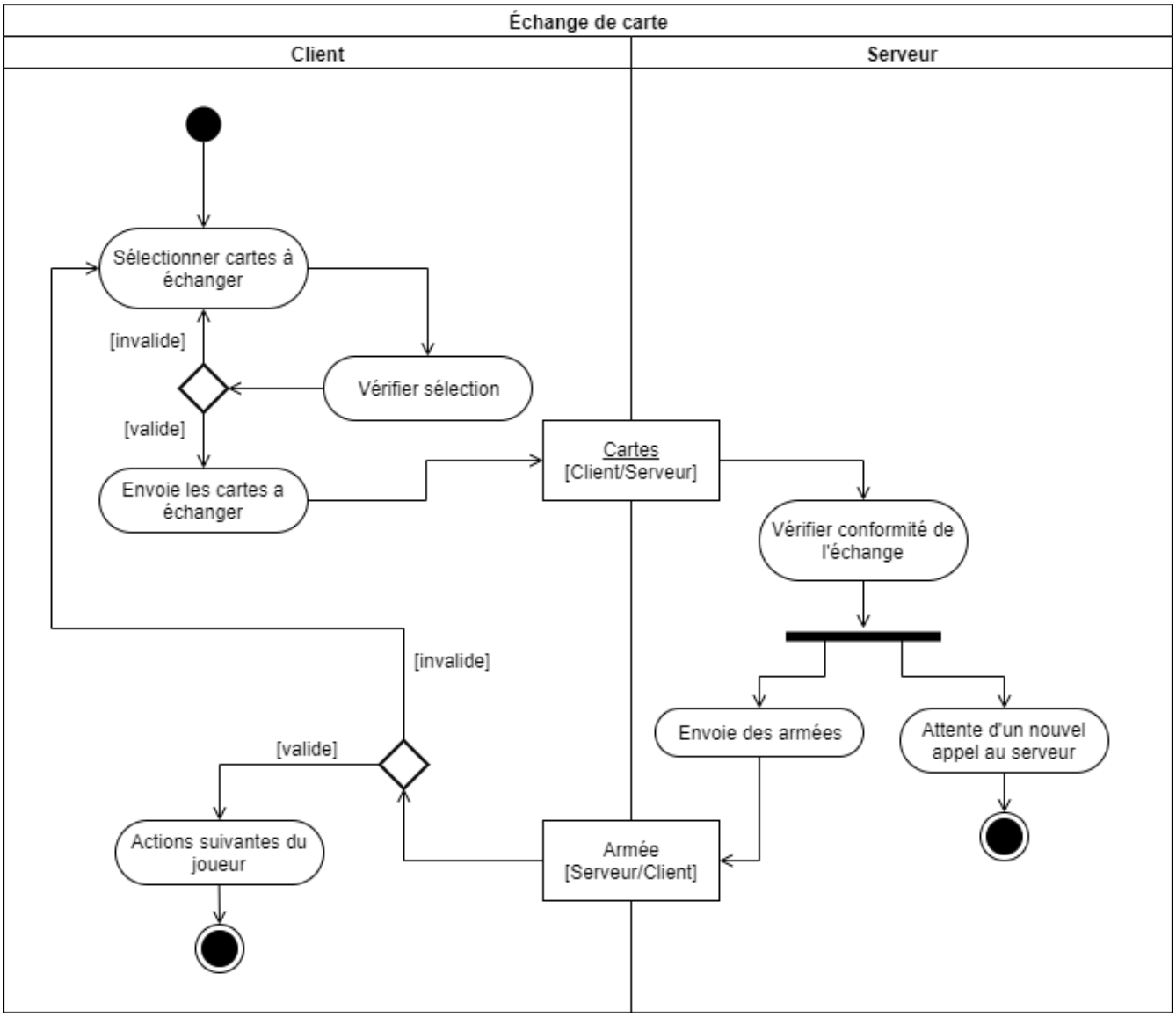


Scenario suboptimal

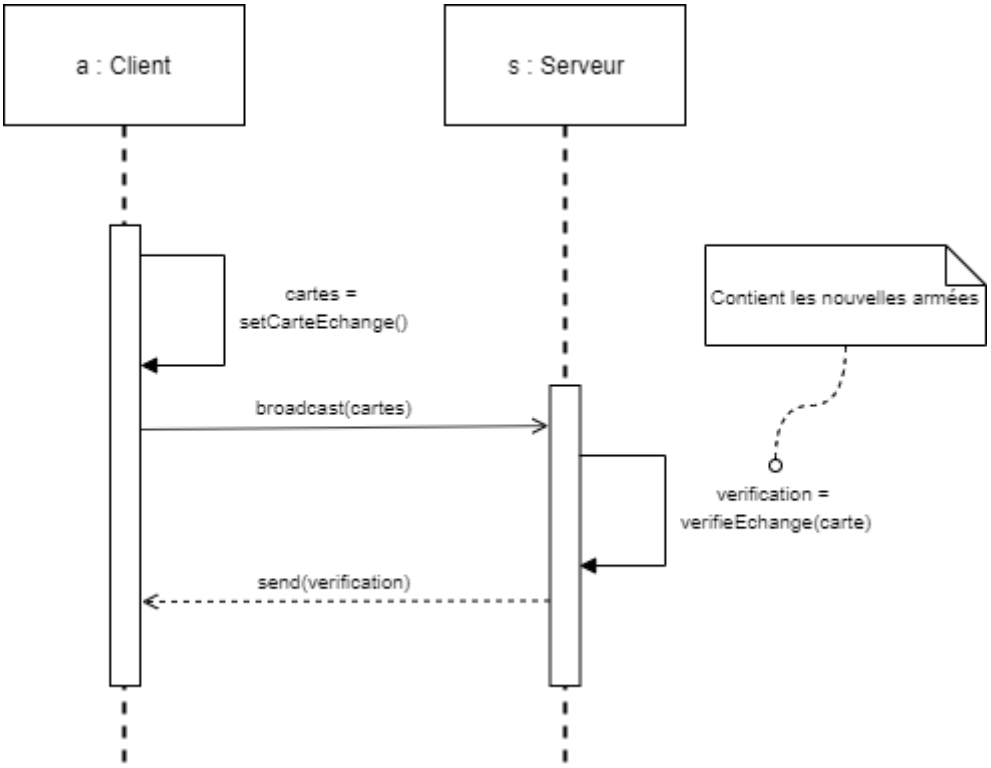


Description de l’échange de carte

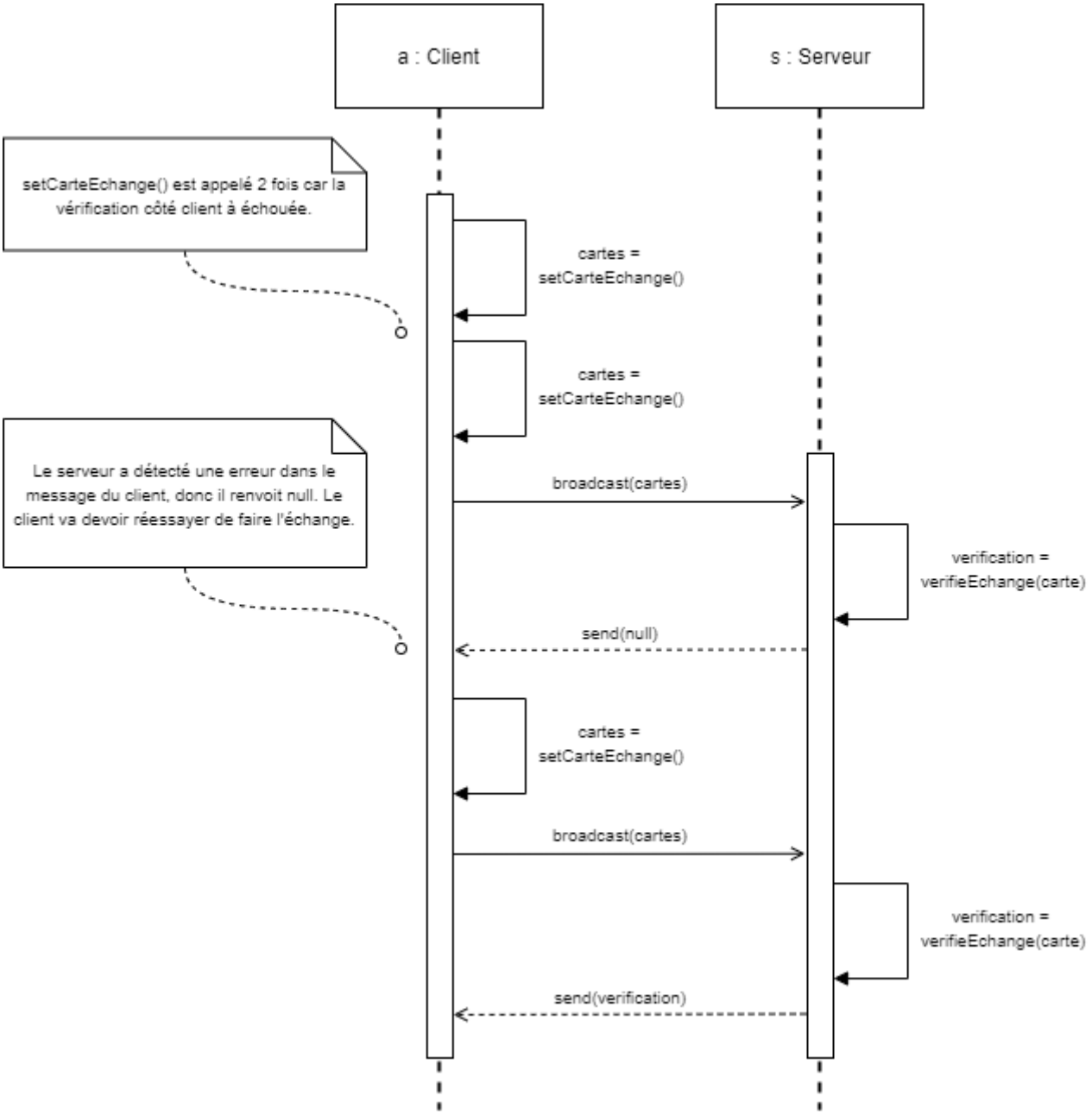
Algorithme



Scenario optimal

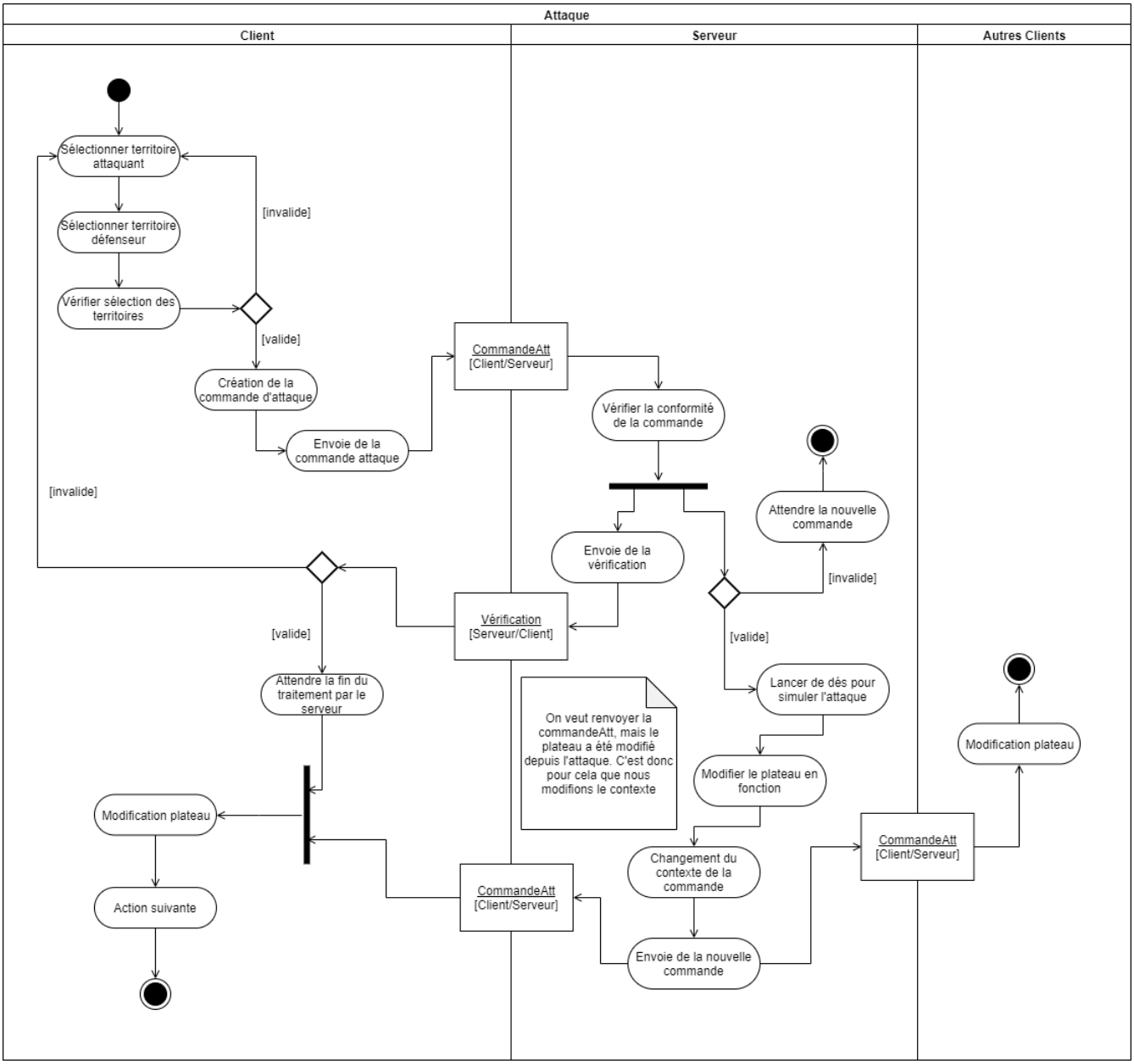


Scenario suboptimal

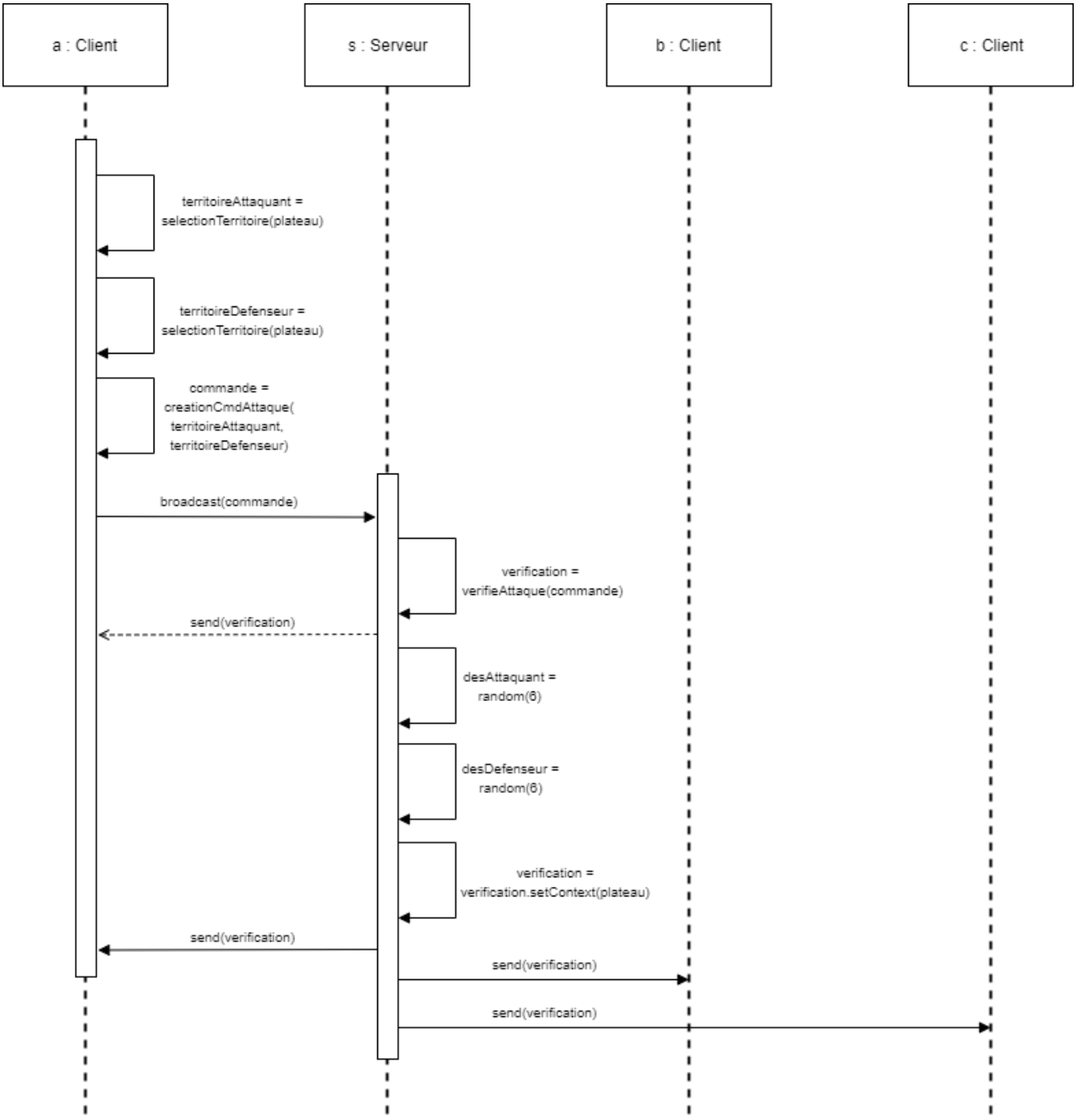


Description de la phase d'attaque

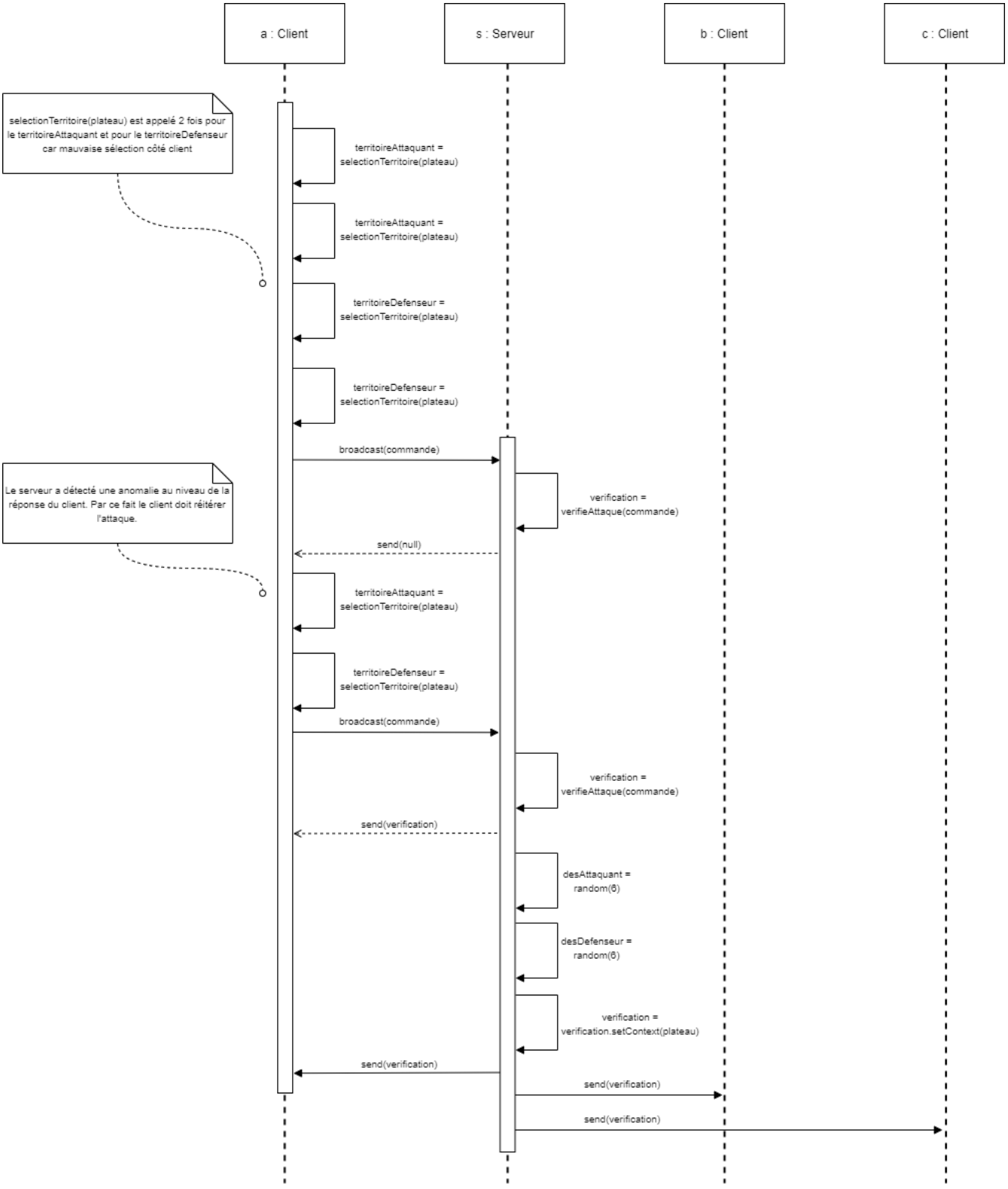
Algorithme



Scenario optimal



Scenario suboptimal



Le composant Serveur

Package

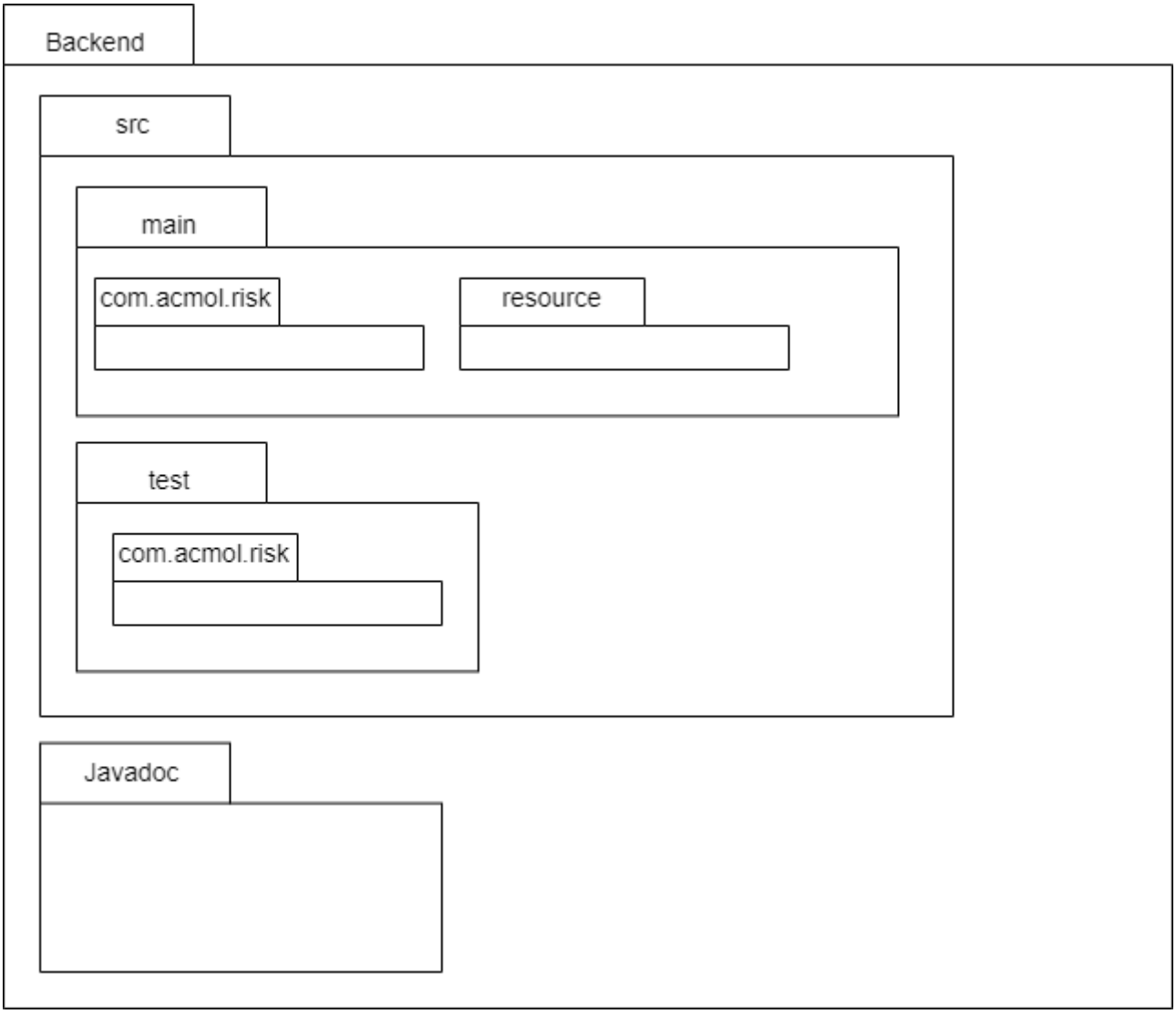
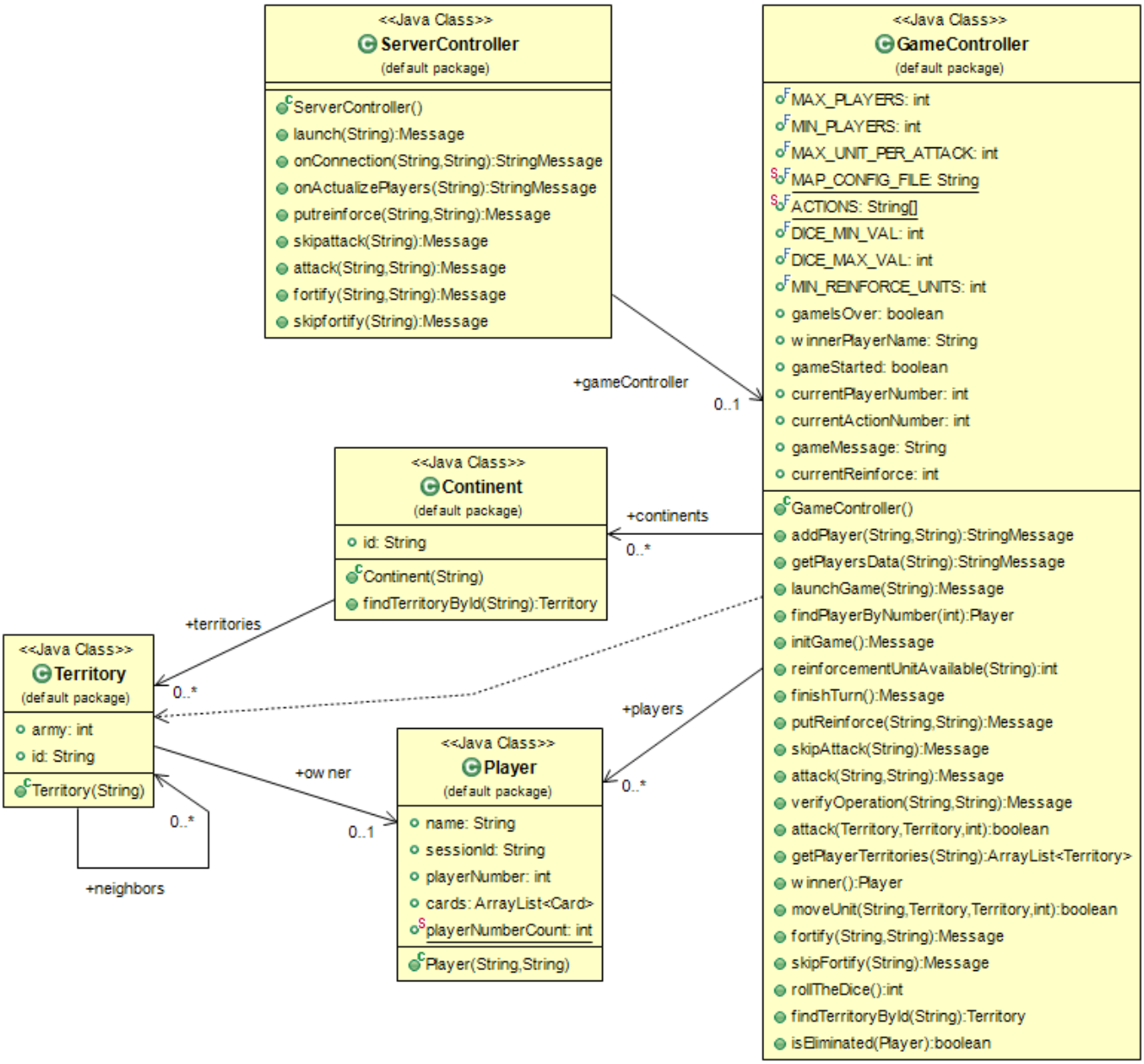
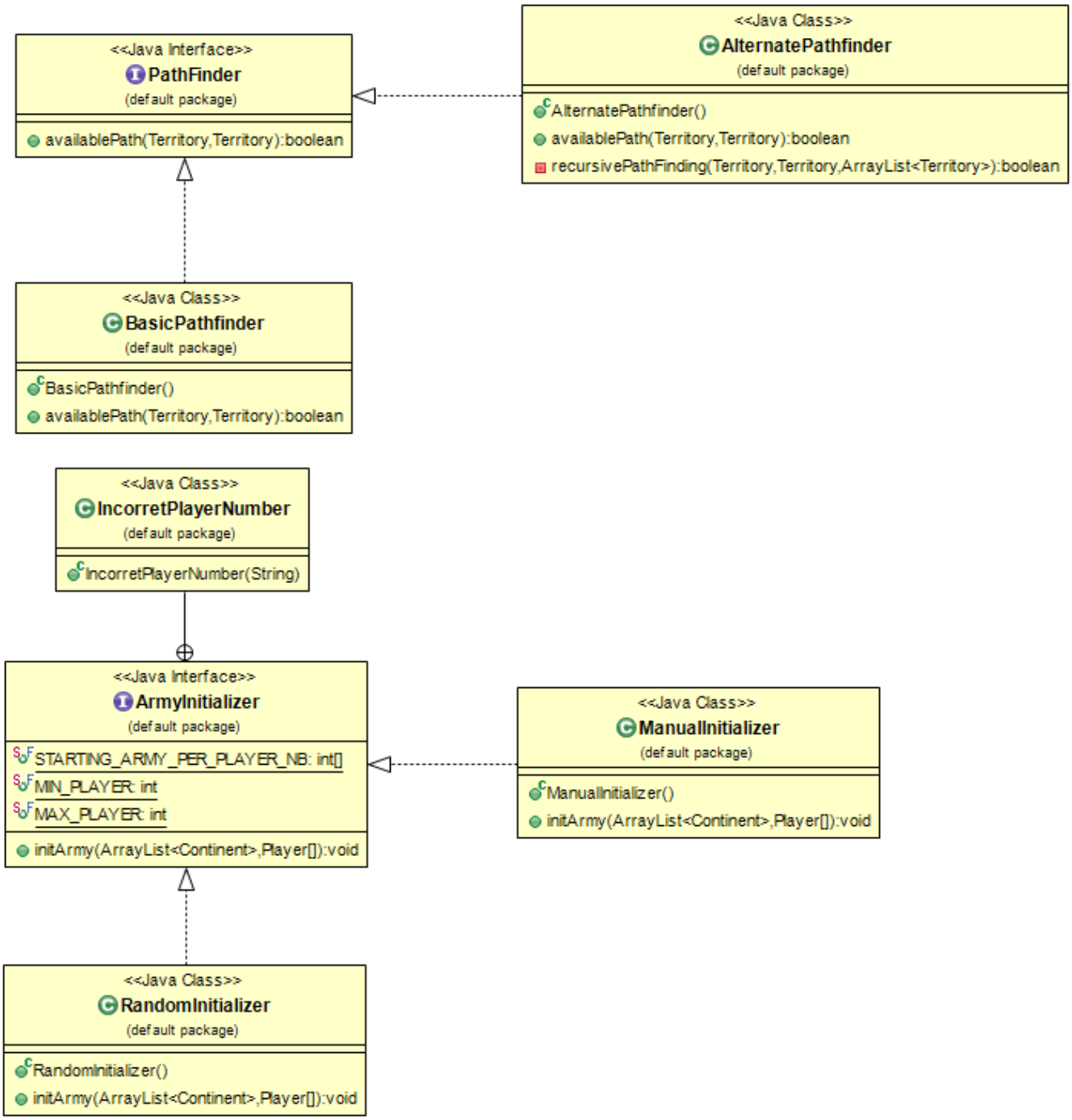


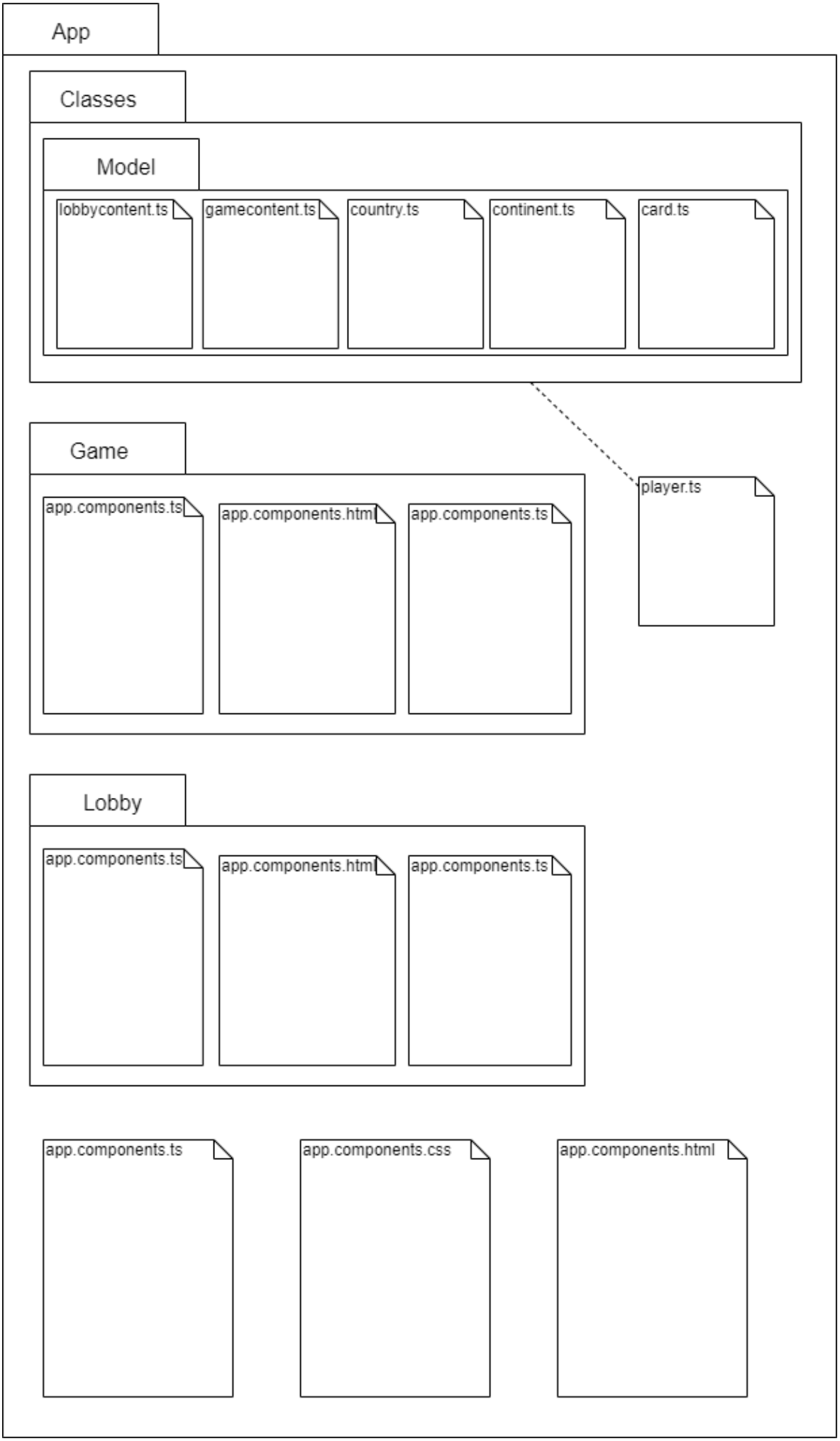
Diagramme de classe


<<Java Class>>
ServerController
(default package)
ServerController()
launch(String):Message
onConnection(String,String):StringMessage
onActualizePlayers(String):StringMessage
putreinforce(String,String):Message
skipattack(String):Message
attack(String,String):Message
fortify(String,String):Message
skipfortify(String):Message
<<Java Class>>
GameController
(default package)
MAX_PLAYERS: int
MIN_PLAYERS: int
MAX_UNIT_PER_ATTACK: int
MAP_CONFIG_FILE: String
ACTIONS: String[]
DICE_MIN_VAL: int
DICE_MAX_VAL: int
MIN_REINFORCE_UNITS: int
gameIsOver: boolean
winnerPlayerName: String
gameStarted: boolean
currentPlayerNumber: int
currentActionNumber: int
gameMessage: String
currentReinforce: int
GameController()
addPlayer(String,String):StringMessage
getPlayersData(String):StringMessage
launchGame(String):Message
findPlayerByNumber(int):Player
initGame():Message
reinforcementUnitAvailable(String):int
finishTurn():Message
putReinforce(String,String):Message
skipAttack(String):Message
attack(String,String):Message
verifyOperation(String,String):Message
attack(Territory,Territory,int):boolean
getPlayerTerritories(String):ArrayList<Territory>
winner():Player
moveUnit(String,Territory,Territory,int):boolean
fortify(String,String):Message
skipFortify(String):Message
rollTheDice():int
findTerritoryById(String):Territory
isEliminated(Player):boolean
+gameController
0..1
<<Java Class>>
Continent
(default package)
id: String
Continent(String)
findTerritoryById(String):Territory
+continents
0..*
+territories
0..*
<<Java Class>>
Territory
(default package)
army: int
id: String
Territory(String)
0..*
+neighbors
+owner
0..1
<<Java Class>>
Player
(default package)
name: String
sessionId: String
playerNumber: int
cards: ArrayList<Card>
playerNumberCount: int
Player(String,String)
+players
0..*



Le composant Client

Package


App
Classes
Model
lobbycontent.ts
gamecontent.ts
country.ts
continent.ts
card.ts
player.ts
Game
app.components.ts
app.components.html
app.components.ts
Lobby
app.components.ts
app.components.html
app.components.ts
app.components.ts
app.components.css
app.components.html

Conclusion

Le temps passé sur la phase d'analyse du domaine a porté ses fruits dans le sens cela a simplifié et accéléré la phase de conception. Une conception précise et adaptée au système étudié nous a également permis d'implémenter le logiciel sans difficulté majeures. Cependant, l'implémentation s'est vue simplifiée en omettant notamment la gestion des cartes. Cela nous a été nécessaire pour respecter les délais impartis. Proportionnellement au relativement peu de temps passé sur le projet (quelques semaines), la configuration d'Angular avec Spring a été très chronophage. Les tests nous ont permis d'expérimenter avec la génération automatique, les outils d'analyse statique et de couverture.