Rapport de projet

Rachel BLIN,
Enora GICQUEL,
Claire LOVISA,
Claire MOLINIER,
Pierrick RANDRIAMANANTSOA,
Maëva SALLANDRE

11 octobre 2017

1 Introduction

Au cours de ce semestre et dans le cadre de l'enseignement d'informatique répartie notre groupe a travaillé sur un projet consistant à réaliser un Bomberman distribué. Ce projet avait pour objectif de nous introduire aux applications distribuées en implémentant un serveur contenant les fonctionnalités principales de notre jeu et un client qui communique avec ce serveur pour pouvoir utiliser cette application.

Nous retrouverons ici une description détaillée de notre projet notamment ses objectifs, les fonctionnalités principales et les règles du jeu.

1.1 Contexte et objectifs

Dans le cadre de notre formation à l'INSA de Rouen Normandie et au sein de l'EC Informatique Répartie, nous sommes amenés à mettre en œuvre une application de jeu distribué dans le but de mettre en pratique les connaissances acquises tout au long du semestre.

1.2 Règles du jeu

Le projet reprend le principe du jeu vidéo Bomberman.

Le jeu se joue de 2 à 4 joueurs. Il est possible de jouer avec des Intelligences Artificielles (IA) mais il faut au minimum un joueur humain. Le jeu est constitué d'un plateau contenant des murs destructibles et indestructibles. Chaque joueur est placé dans un coin du plateau et au cours de la partie peut se déplacer aux endroits où il n'y a pas de mur et y poser des bombes. Une fois posée, une bombe attend un certain délai puis produit une déflagration. Cette déflagration en forme de croix a une certaine portée, si elle atteint un mur destructible, celui-ci est détruit et si elle atteint un joueur, celui-ci est éliminé.

Lorsqu'un mur est détruit, un bonus peut apparaître pouvant avoir différents effets, entre autres :

— *Bomb Up* : augmente le nombre de bombes pouvant être posé par le joueur ;
— *Bomb Down* : diminue le nombre de bombes pouvant être posé par le joueur (minimum 1) ;
— *Fire Up* : augmente la portée de la déflagration des bombes du joueur ;
— *Fire Down* : diminue la portée de la déflagration des bombes du joueur (minimum 1).

1.3 Fonctions principales

Les fonctions principales du jeu sont :

— Jeu multijoueurs ;
— Personnage(s) virtuel(s) avec avatar(s) ;
— Enregistrement des actions sous la forme de la sauvegarde d'un fichier de scores ;
— Possibilité de récupérer les scores réalisés durant des parties précédentes ;
— Compter les points des joueurs.

1.4 Utilisateurs et caractéristiques

L'application est destinée à tout joueur possédant et sachant utiliser un minimum un ordinateur. Pour une partie, 2 à 4 joueurs sont requis. S'il y a cependant moins de joueurs disponibles, les places restantes seront complétées par des IA.

1.5 Contraintes matérielles et logicielles

Il est nécessaire d'avoir à disposition un ordinateur fonctionnant avec le système d'exploitation Linux.

2 Besoins détaillés

Pour mener à bien ce projet, il est important de cerner précisément les différentes fonctionnalités à fournir.

2.1 Spécifications fonctionnelles

Les spécifications fonctionnelles sont triées en premier temps en fonction de leur priorité (celle-ci allant de 1 à 5, 1 étant la priorité maximale) puis en fonction de leur version. La version 1 est primordiale pour que le projet réponde aux fonctions principales demandées. La version 2 sera réalisée si le temps le permet, elle agrandira les possibilités de jeu.

ID	Fonctionnalité	Prédécesseurs	Priorité	Version
1	Présence d'un menu "Jouer - Scores" (au minimum)	-	2	1
2	Jouer contre une ou plusieurs IA (IA très simple, de type "l'IA se déplace aléatoirement")	1	1	1
3	Jouer contre d'autres joueurs	1	1	1
4	Se déplacer sur des cases libres uniquement	-	1	1
5	L'utilisateur peut poser une bombe	-	1	1
6	Présence de murs indestructibles	-	1	1
7	Créer une partie	-	1	1
8	Rejoindre une partie	-	1	1
9	L'utilisateur peut récupérer des bonus	24	5	2
10	Chaque personnage a un nombre limité de bombes	-	2	1
11	Les scores de l'utilisateur peuvent être sauvegardés	8	2	1
12	L'utilisateur peut choisir son avatar	1	3	2
13	Un des bonus permet d'augmenter la portée des bombes	5,9,24	5	2
14	Un des bonus permet de diminuer la portée des bombes	5,9,24	5	2
15	Un des bonus permet de pouvoir poser un nombre plus important de bombes simultanément	5,9,24	5	2
16	Un des bonus permet de pouvoir poser un nombre moins important de bombes simultanément	5,9,24	5	2
17	Possibilité de consulter les scores	1	2	1
18	Plusieurs niveaux sont disponibles	1,6	3	2
19	L'utilisateur peut se connecter	1	1	1
20	Lorsqu'un personnage est touché par une déflagration, celui-ci est éliminé	5,21	2	1
21	Au bout d'un certain délai, une bombe posée produit une déflagration	5	1	1
22	Présence de murs destructibles	-	4	2
23	L'utilisateur peut faire exploser certain mur avec une bombe	22	3	2
24	Apparition de bonus en cassant certains murs	5,22,23	5	2

Table 1 – Spécifications Fonctionnelles

2.2 Spécifications d'interface

L'interface du jeu doit correspondre aux différentes spécifications fonctionnelles, c'est pourquoi nous avons plusieurs versions. Nous avons créer la maquette de l'écran principale du jeu, celui pour jouer.

Nous avons prévu de faire plusieurs versions de notre jeux et nous allons présenter les diagrammes de navigation des différentes versions.

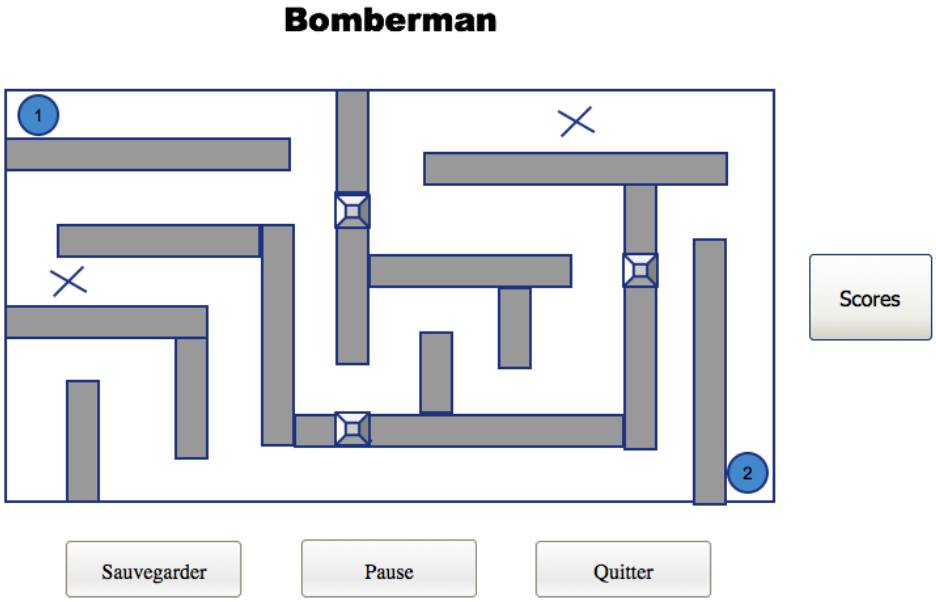

FIGURE 1 – Maquette de l'écran principal du jeu

2.2.1 Version 1

La première version du jeu comportera 6 écrans.

- écran 1 : écran d'accueil du jeu, on pourra consulter les scores ou lancer une partie.
- écran 2 : écran qui permet de consulter la liste des meilleurs scores.
- écran 3 : écran qui permet de choisir de rejoindre une partie ou d'en créer une nouvelle.
- écran 4 : écran qui permet de choisir le nom de sa partie.
- écran 5 : écran qui permet de rejoindre une partie existante en cliquant sur le bouton correspondant.
- écran 6 : écran qui permettra de jouer.

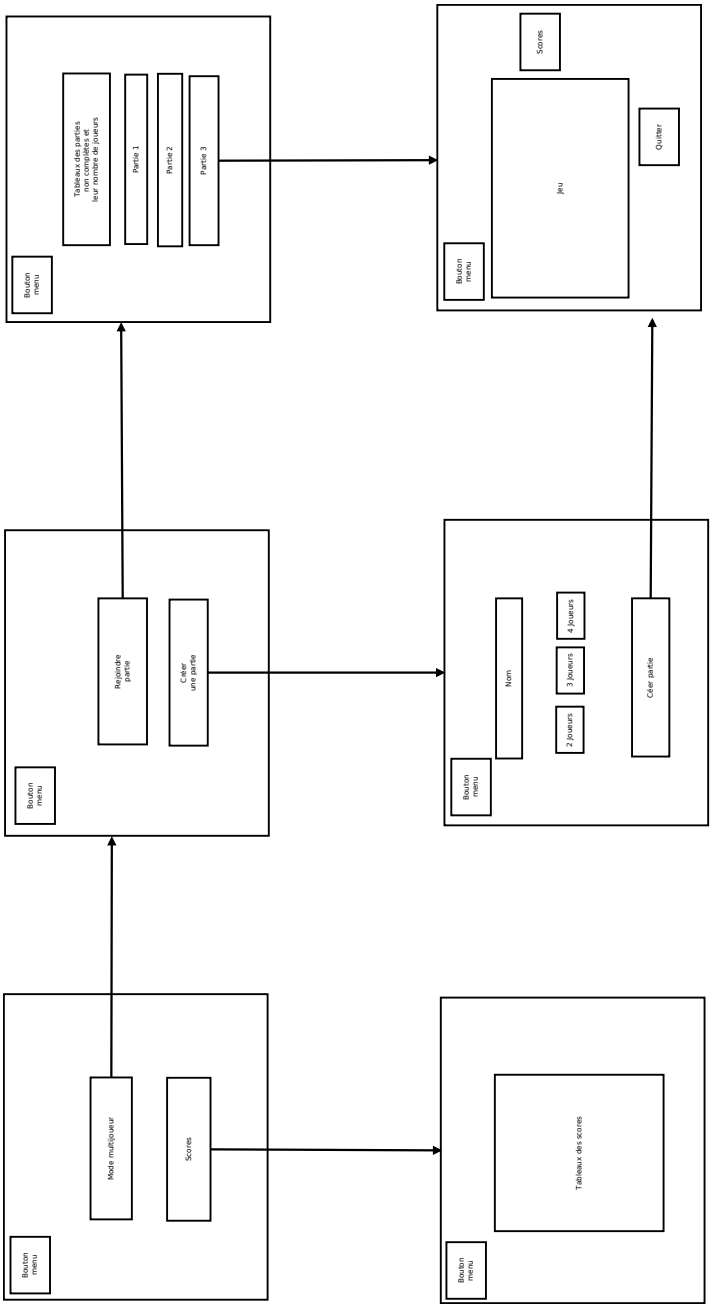

FIGURE 2 – Diagramme de navigation de la version 1

2.2.2 Version 2 (si le temps le permet)

La seconde version du jeu comportera des améliorations par rapport à la première. C'est pour cela qu'on trouvera de nouveaux écrans :

— écran 6 : écran qui permettra de jouer. Il permet aussi de consulter les scores, mettre sur pause et quitter le jeu.
— écran 7 : écran qui permettra de choisir de rejouer les actions d'une sauvegarde.
— écran 8 : écran qui permettra de choisir son avatar.

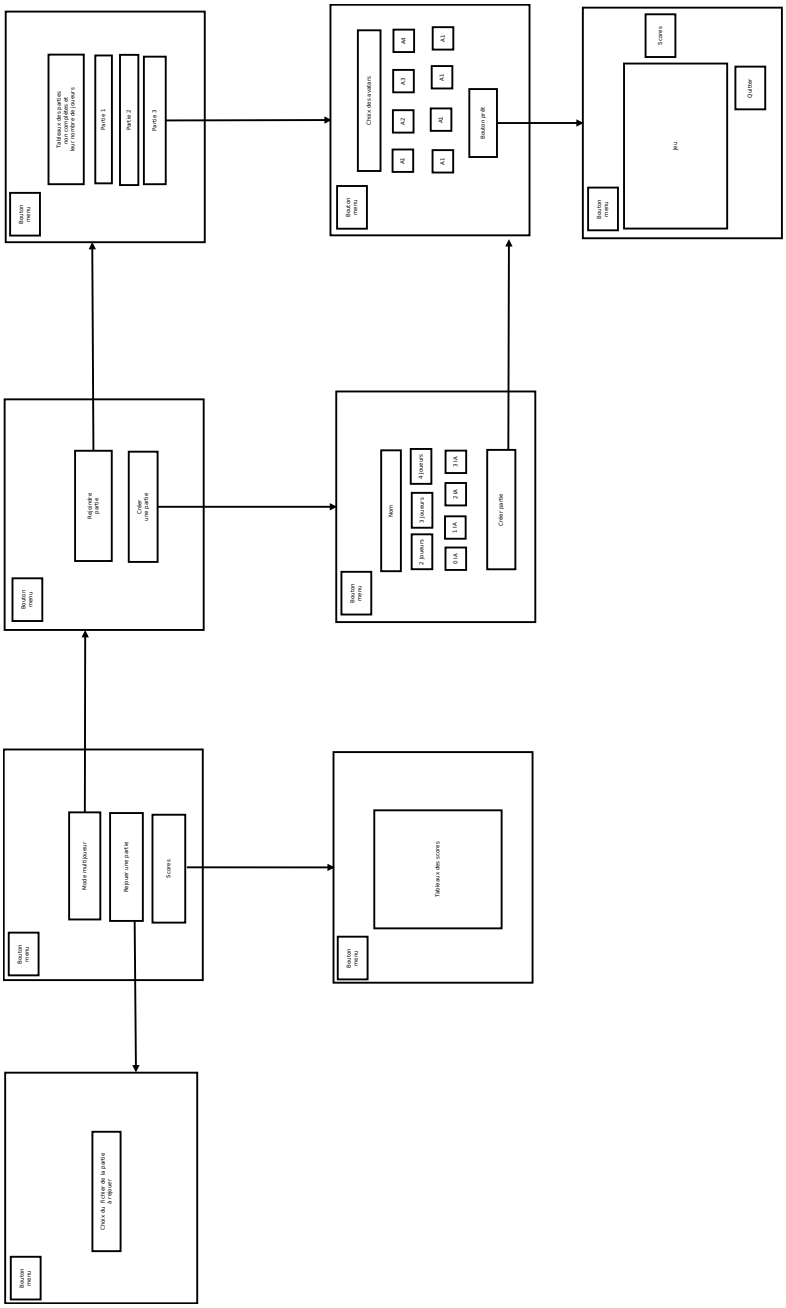

FIGURE 3 – Diagramme de navigation de la version 2

2.3 Spécifications opérationnelles

Pour chaque étape du projet, de nouvelles spécifications s'ajoutent aux précédentes. Nous avons donc séparé les spécifications opérationnelles par versions.

2.3.1 Version 1

— Le serveur pourra gérer un minimum de 10 clients simultanément : 4 pour la partie et un message d'information pour les autres ;
— Une seule partie peut être en cours sur le serveur ;
— Le serveur peut héberger une partie de 4 joueurs, s'il n'y a pas assez de joueurs, le serveur remplacera les joueurs manquant par une IA ;
— Aucune donnée à caractère personnelle ne devra être stockée en dur sur le serveur afin d'assurer la protection des données ;
— Le produit doit fonctionner au minimum sous l'environnement Linux ;
— Puisqu'il s'agit d'un jeu, la sécurité n'est pas prioritaire et ne sera pas traitée dans la première version du jeu ;
— Le produit n'aura pas à être installé, il fonctionnera en lançant l'exécutable associé ;
— Les clients ou tout autre personne extérieure à la maintenance du système ne doit pouvoir y accéder ;
— Avant d'effectuer une modification du système il est nécessaire d'obtenir l'accord de toutes les personnes responsables du système ;
— Toute modification du système devra pouvoir être traçable, ce qui implique qu'il faudra garder une trace de la personne ayant effectué la modification, ainsi que toutes les différences par rapport à la version précédente des l'ordre chronologique.

2.3.2 Version 2 (si le temps le permet)

— Le serveur sera en capacité de stocker les pseudonymes et scores des 5 parties précédentes, ainsi que les 3 meilleures parties de l'histoire du jeu ;
— Le serveur peut héberger jusqu'à deux parties simultanément et peut traiter les requêtes de 20 joueurs : jusqu'à 8 joueurs dans une partie en cours et un message pour les autres joueurs cherchant à se connecter ;
— Aucune donnée personnelle ne sera stockée sur le serveur : il n'est nécessaire de stocker que les pseudonymes et les scores ;
— Aucune donnée à caractère personnelle ne devra être stockée en dur sur le serveur afin d'assurer la protection des données.

3 Conception préliminaire

3.1 Modèle du domaine

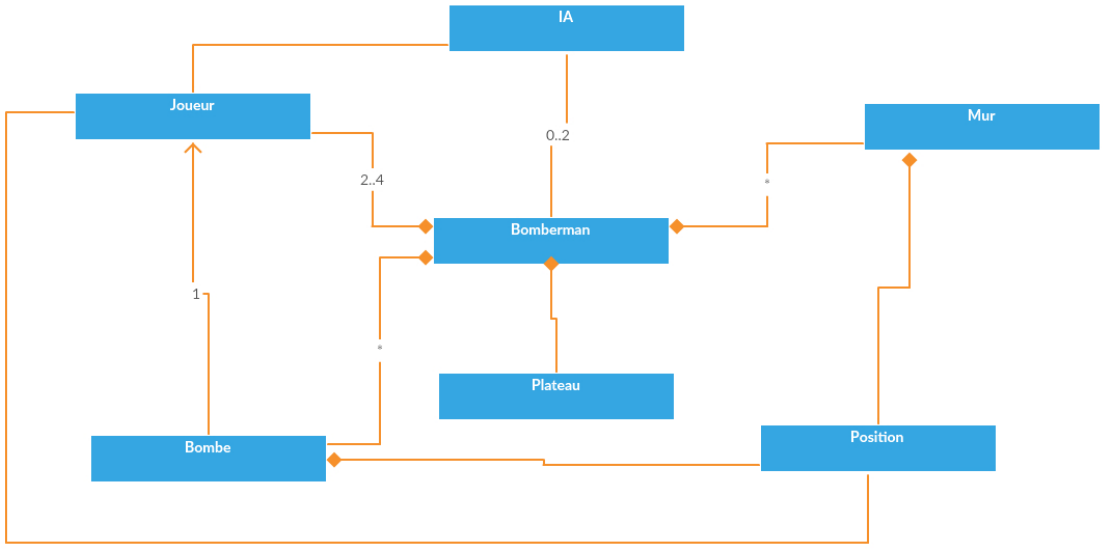

FIGURE 4 – Modèle du domaine

3.2 Diagrammes de séquence système

3.2.1 Connexion

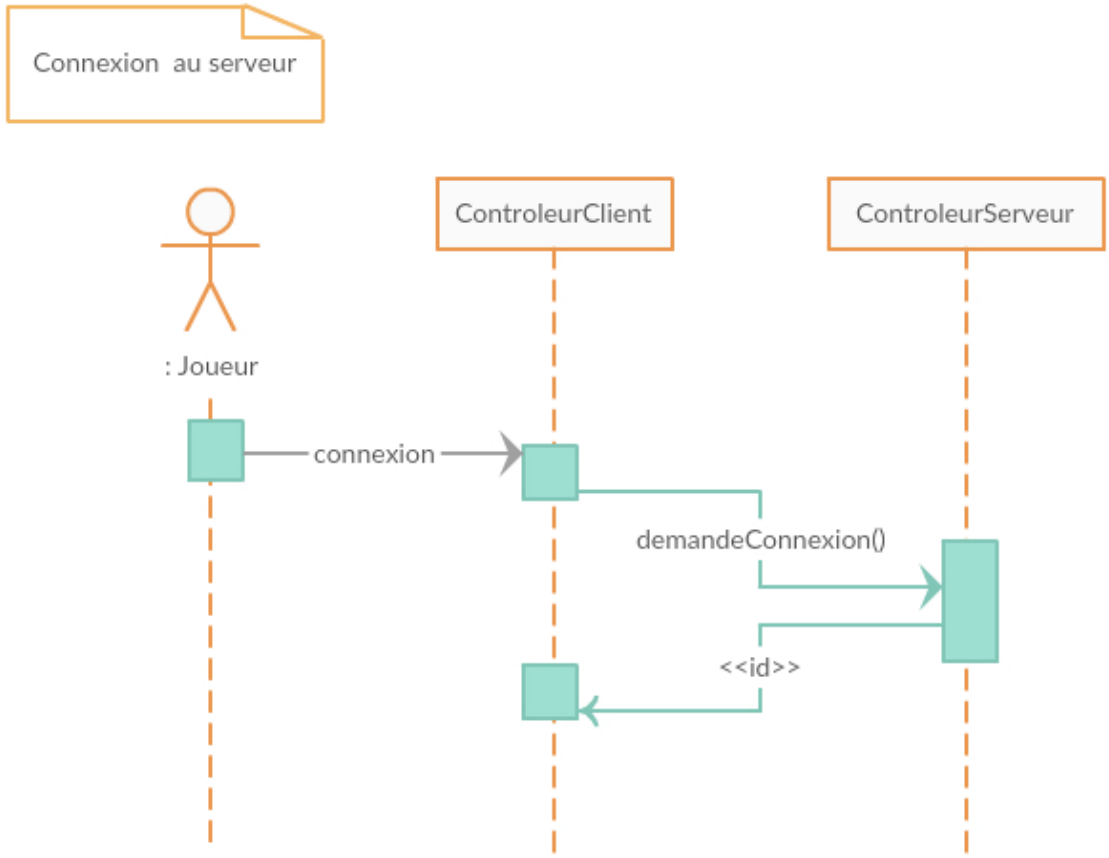

FIGURE 5 – Diagramme de séquence système du cas d'utilisation "Se connecter"

3.2.2 Créer partie

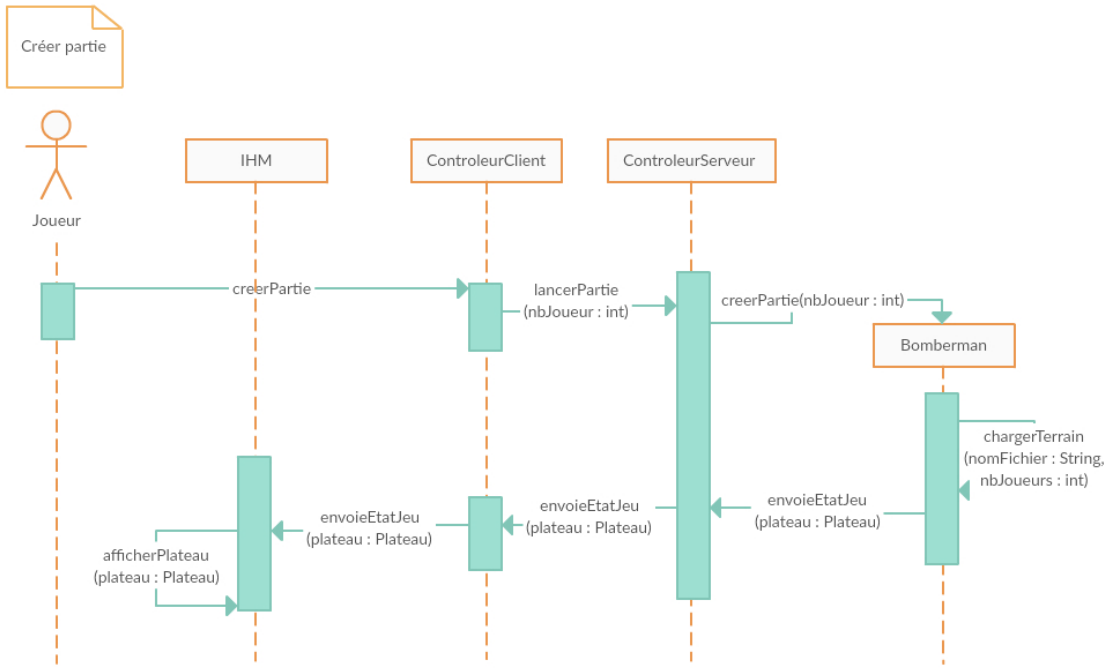

FIGURE 6 – Diagramme de séquence système du cas d'utilisation "Créer partie"

3.2.3 Jouer

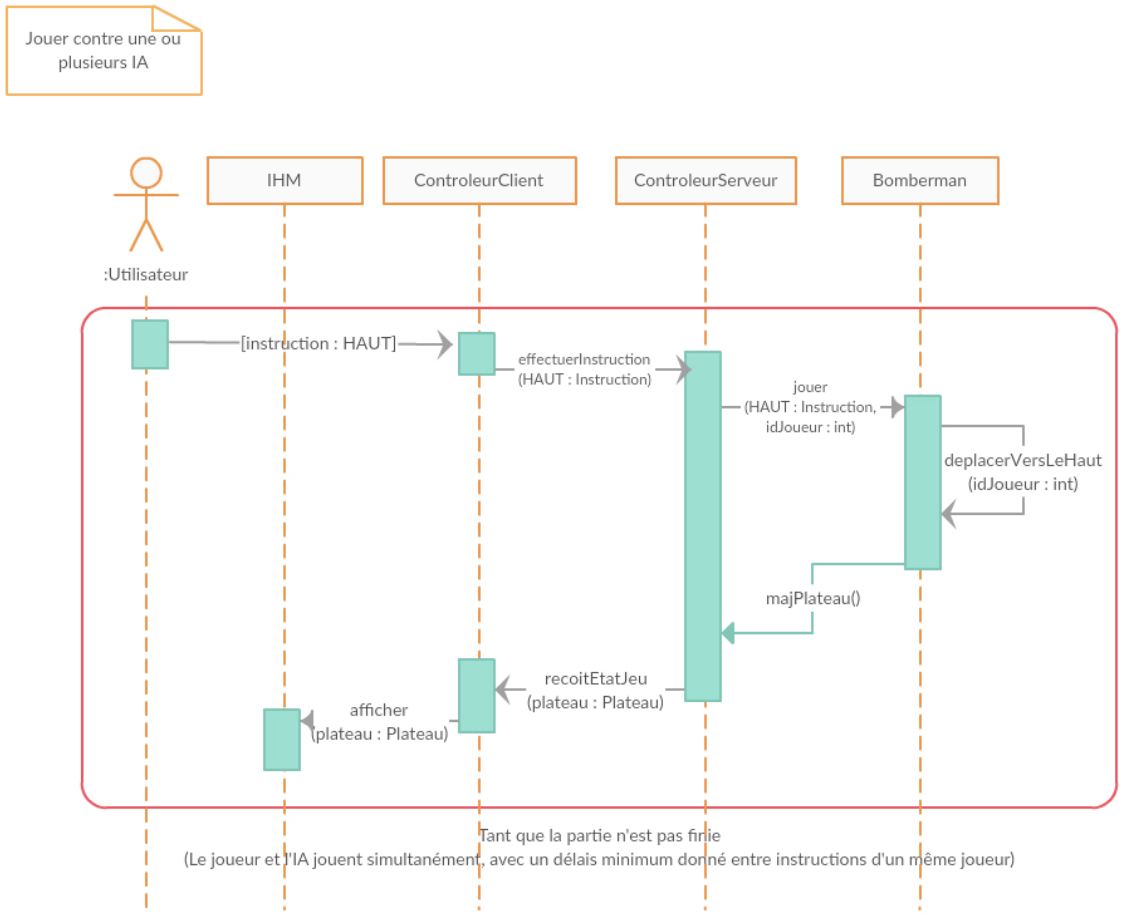

FIGURE 7 – Diagramme de séquence système du cas d'utilisation "Jouer"

3.2.4 Bonus

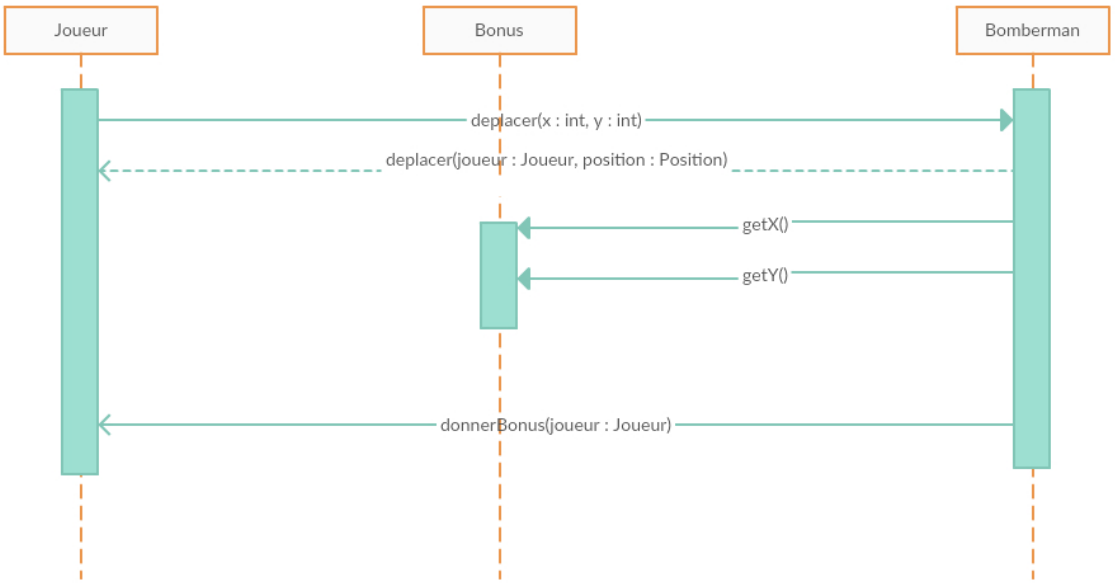

FIGURE 8 – Diagramme de séquence système du cas d'utilisation "Bonus"

3.3 Diagrammes de navigation

Diagramme de navigation du Bomberman à la fin de la version 1.

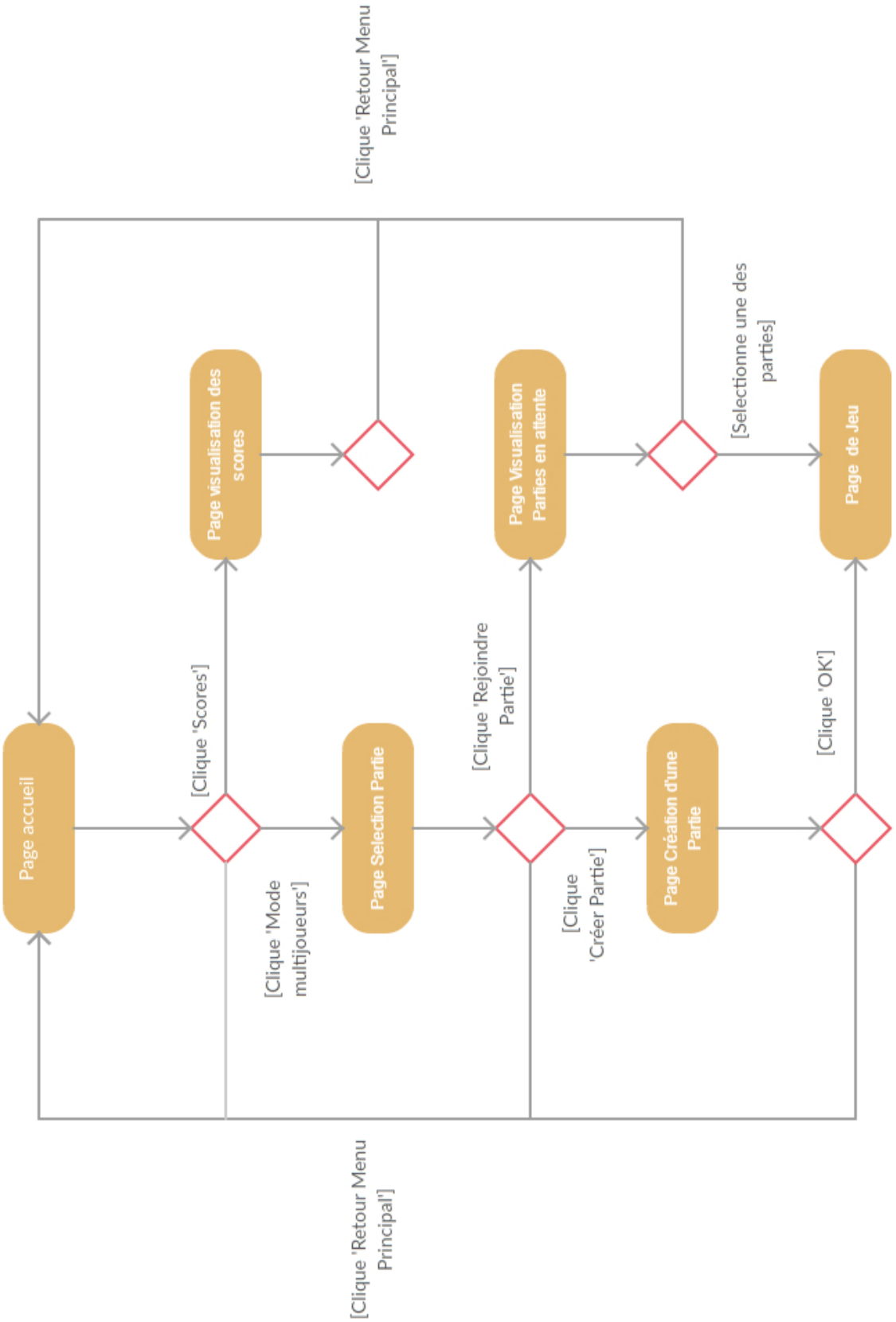

FIGURE 9 – Diagramme de navigation v1

Diagramme de navigation du Bomberman à la fin de la version 2.

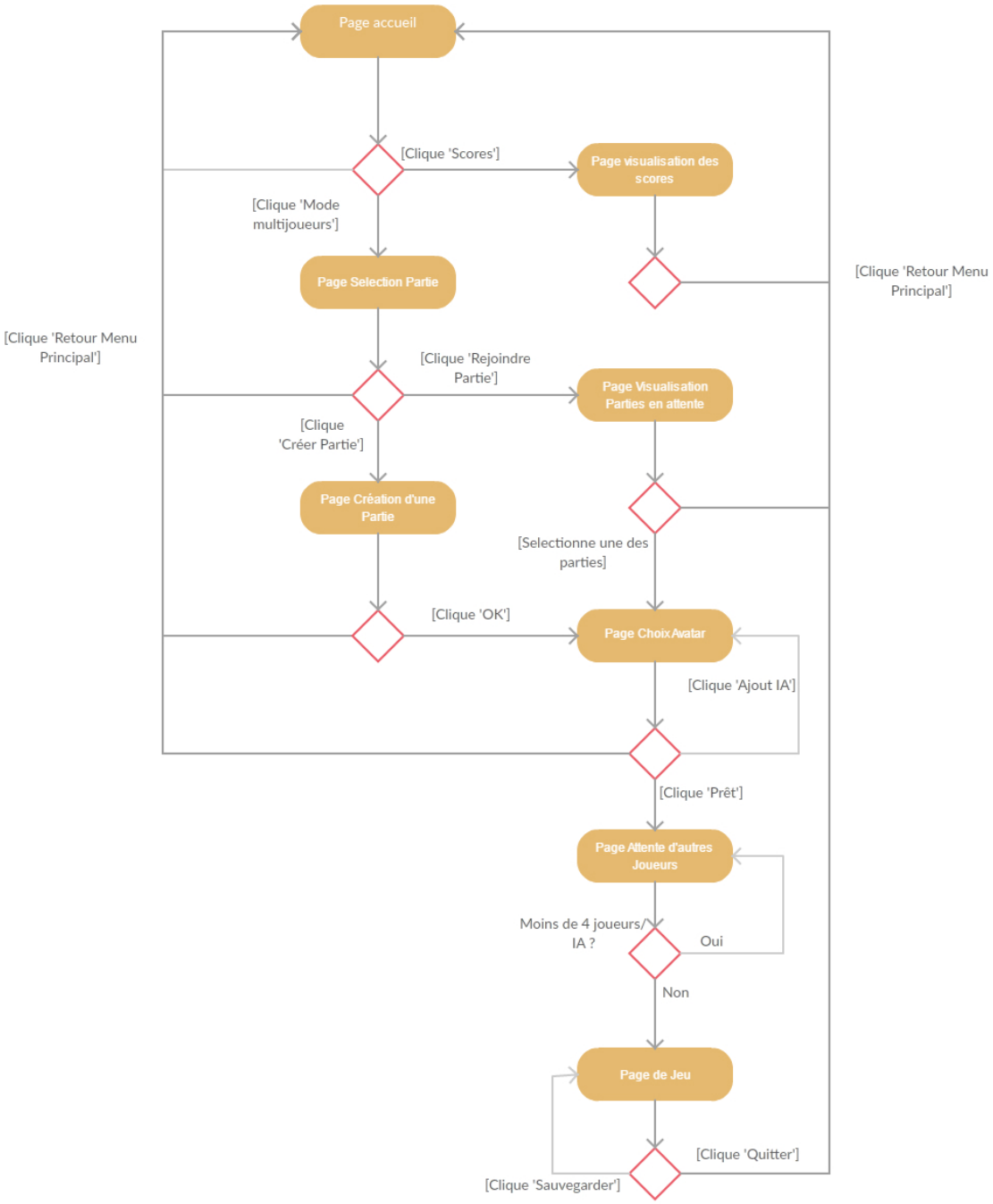

FIGURE 10 – Diagramme de navigation v2

3.4 Diagramme de classes de conception préliminaire

Pour ce diagramme, les accesseurs,modificateurs, equals, hashcode, ajouts dans listes et toString n'ont pas été précisés.

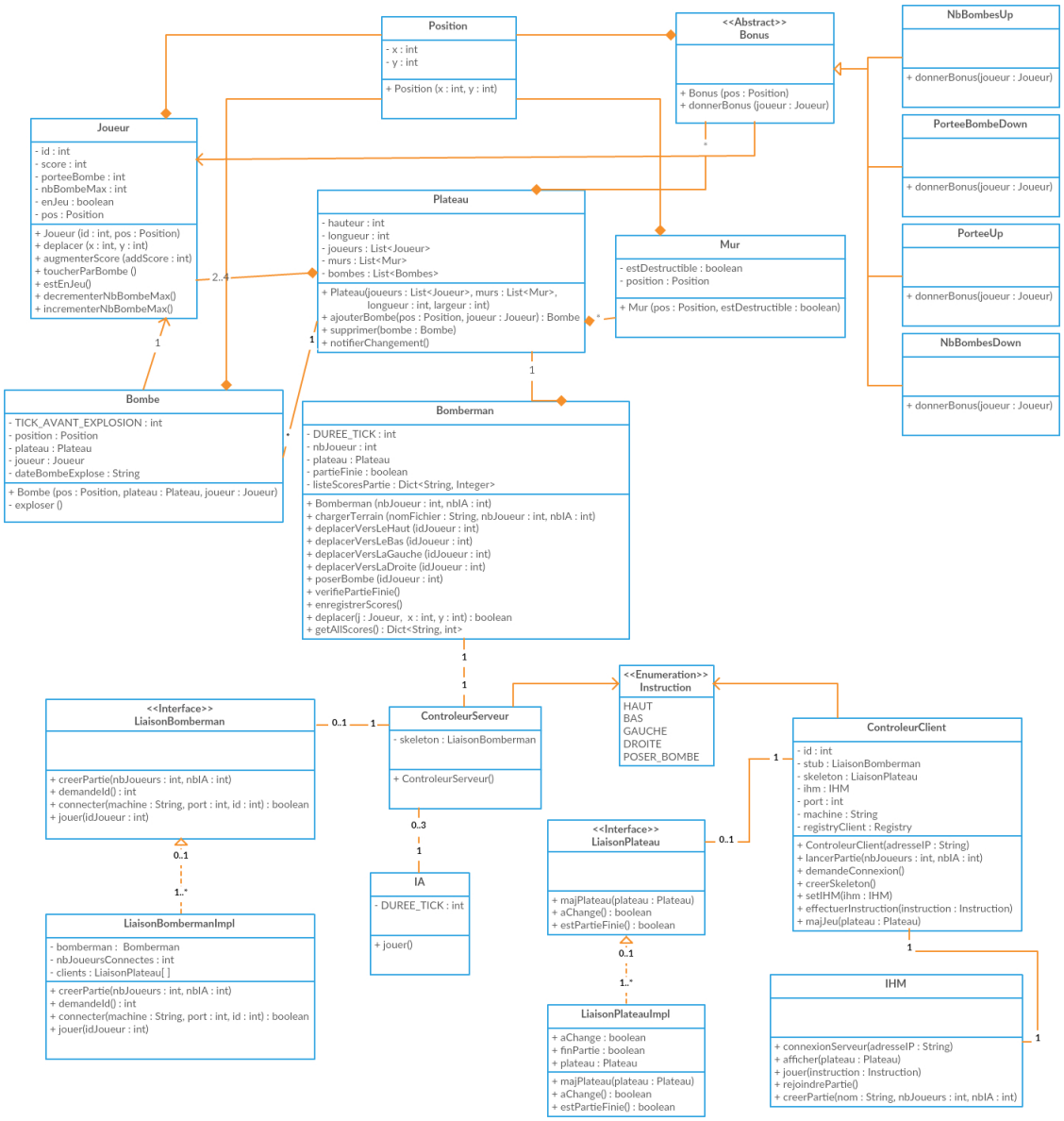

FIGURE 11 – Diagramme de classes

3.5 Division en packages

Ce diagramme permet de visualiser les différents packages dans lesquels seront organisées les classes.

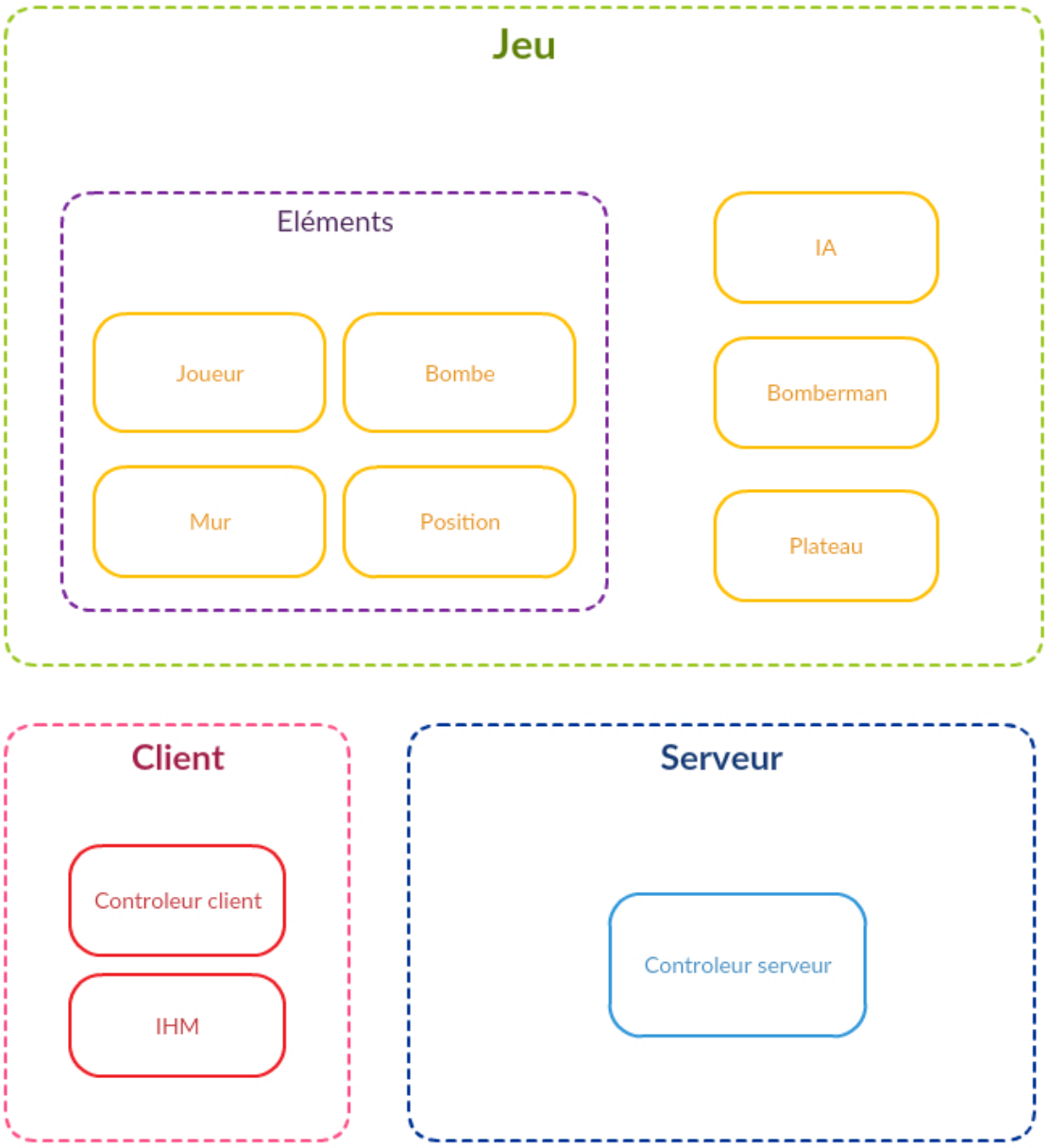

FIGURE 12 – Division en packages

4 Conception détaillée

4.1 Pseudo-code de la méthode deplacer()

Cette méthode se trouve dans la classe Bomberman et a donc accès aux Listes de murs, de joueurs, de bombes et de bonus présents sur le plateau de jeu.

```
procédure deplacer (E j : Joueur, p : Position S deplacementPossible : Booleen)
    Déclaration   bonusUtilise : null ,
                  deplacementPossible : Booleen
debut
    deplacementPossible ← vrai

    pour chaque mur de murs
        si mur.getPosition.getX() == p.getX() ET mur.getPosition.getY() == p.getY()  alors
            deplacementPossible ← faux
        finsi
    finpour


    pour chaque joueur de joueurs
        si joueur.getPosition.getX() == p.getX() ET joueur.getPosition.getY() == p.getY()  alors
            deplacementPossible ← faux
        finsi
    finpour


    pour chaque bombe de bombes
        si bombe.getPosition.getX() == p.getX() ET bombe.getPosition.getY() == p.getY()  alors
            deplacementPossible ← faux
        finsi
    finpour


    pour chaque bonus de listeBonus
        si bonus.getPosition.getX() == p.getX() ET bonus.getPosition.getY() == p.getY()  alors
            bonusUtilise ← bonus
            bonus.donnerBonus(j)
        finsi
    finpour


    listeBonus.supprimer(bonusUtilise)
    retourner deplacementPossible
fin
```

4.2 Pseudo-code de la méthode exploser() dans Bombe

```
procédure exploser ()
    Déclaration  positionsExplosion : Liste<Position>
                 mursExploses, listeTmpMurs : Liste<Mur>
                 i : Entier
                 gaucheStop, droiteStop, hautStop, basStop : Booleen
                 posG, posD, posH, posB : Position
debut
    gaucheStop ← faux
    droiteStop ← faux
    hautStop ← faux
    basStop ← faux
    positionsExplosion ← Liste<Position>()
    mursExploses ← Liste<Mur>()

    // Suppression de la Bombe de la liste de Bombes
    this.plateau().setBombes(this.plateau().getBombes().supprimer(this))

    // On récupère les Positions atteintes par l'explosion sachant que
    // l'explosion est stoppée par un mur
    positionsExplosion.ajouter(this.getPosition())
    pour i ←1 à this.joueur.getPortee() faire
        posG ← Position(this.position.getX()-i, this.position.getY())
        posD ← Position(this.position.getX()+i, this.position.getY())
        posH ← Position(this.position.getX(), this.position.getY()-i)
        posB ← Position(this.position.getX(), this.position.getY()+i)
        pour chaque mur de this.plateau().getMurs()
            si NON gaucheStop alors
                gaucheStop ← posG == mur.getPosition()
                si gaucheStop ET mur.getEstDestructible() alors
                    mursExploses.ajouter(mur)
                finsi
            finsi
            si NON droiteStop alors
                droiteStop ← posD == mur.getPosition()
                si droiteStop ET mur.getEstDestructible() alors
                    mursExploses.ajouter(mur)
                finsi
            finsi
            si NON hautStop alors
                hautStop ← posH == mur.getPosition()
                si hautStop ET mur.getEstDestructible() alors
                    mursExploses.ajouter(mur)
                finsi
            finsi
            si NON basStop alors
                basStop ← posB == mur.getPosition()
                si basStop ET mur.getEstDestructible() alors
                    mursExploses.ajouter(mur)
                finsi
            finsi
        finpour
        si NON gaucheStop alors
            positionsExplosion.ajouter(posG)
        finsi
        si NON droiteStop alors
            positionsExplosion.ajouter(posD)
        finsi
        si NON hautStop alors
            positionsExplosion.ajouter(posH)
        finsi
        si NON basStop alors
            positionsExplosion.ajouter(posB)
```

```
        finsi
    finpour

    // Parcours des Positions de l'explosion pour éliminer les Joueurs touchés et
    // faire exploser les bombes touchées
    pour chaque pos de positionsExplosion
        pour chaque joueur de this.plateau().getJoueurs()
            si joueur.estEnJeu() ET joueur.getPosition() == pos alors
                joueur.toucherParBombe()
            finsi
        finpour
        pour chaque bombe de this.plateau().getBombes()
            si bombe.getPosition() == pos alors
                bombe.exploser()
            finsi
        finpour
    finpour

    // Suppression des murs
    listeTmpMurs ← this.plateau().getMurs()
    pour chaque mur de mursExploses
        listeTmpMurs.effacer(mur)
    finpour
    this.plateau().setBombes(listeTmpMurs)
fin
```

5 Implémentation et tests

5.1 Langage et bibliothèques utilisées pour implémenter le projet

Nous avons tout d'abord choisi la technologie à utiliser pour nous permettre d'implémenter notre projet en informatique répartie. Dans la mesure où il s'agit d'un jeu en temps réel, il faut que le temps de réponse du serveur soit très court. Les Sockets et le RMI correspondent à ce critère.
Nous avons choisi d'implémenter le projet en langage Java puisqu'il possède une API permettant de faire du RMI, de plus Java est un langage orienté objet donc adapté à l'implémentation d'un jeu et tous les membres de l'équipe sont formés à ce langage. Pour cela, nous avons utilisé les bibliothèques suivantes :
- java.util.Random, package pour implémenter le comportement aléatoire de l'IA ;
- java.rmi.registry et java.rmi.server, packages contenant un ensemble de classes pour implémenter du RMI en Java ;
- java.io, pour pouvoir implémenter les exceptions et pouvoir lire les fichiers ;
- javax.swing, javax.imageio et java.awt, pour implémenter l'IHM ;
- java.io.Serialisable pour spécifier les javabean permettant de représenter les différents éléments du jeu ;
- java.junit et le package org pour importer les variables Is et IsEqual pour pouvoir implémenter les tests des éléments de l'application.

5.2 Guide d'utilisation

Pour pouvoir faire une partie de Bomberman, il est tout d'abord nécessaire de lancer le serveur. Une fois le serveur lancé, il est possible de s'y connecter en mode local (le client est lancé sur la même machine dans un autre terminal) ou à distance, soit sur une autre machine en rentrant l'adresse IP retournée par le serveur. Une fois le client connecté, il est possible de créer une partie ou de rejoindre une partie existante. Si aucune partie est en cours, il faudra créer une nouvelle partie et pour cela, choisir le nombre de joueurs que l'on veut dans la partie dont le nombre d'IA. Un autre joueur peut rejoindre à tout moment la partie en cours en lançant un autre client.

Les commandes pour pouvoir jouer sont les flèches du clavier afin de pouvoir se déplacer ainsi que la touche espace pour poser une bombe.

5.3 Tests de validation

Afin de vérifier que chacune des classes de notre projet était bien fonctionnelle, nous avons réalisé des tests unitaires sur chacune d'entre elles. Toutes les classes ont été testées par une personne ne l'ayant pas codé pour rendre les tests le plus objectifs possibles, à l'exception de toutes les classes composant l'IHM. En effet, nous avons jugé que pour vérifier la fonctionnalité de l'IHM se ferait de façon visuelle.

5.4 Partie Métier

La partie métier a été développée pour être indépendante de l'IHM. En effet, comme dit précédemment, des tests unitaires ont été créés et ceux-ci ont été réalisés sans que l'IHM ne soit terminée et chaque méthode remplissait donc correctement son rôle.
Les classes côté métier étant en général assez basiques, seule la classe Bomberman reliant le serveur à la partie métier a été un peu plus longue a développer que prévu. Elle n'a cependant pas posé de problème particulier et nos efforts ont donc pu être concentrés sur le développement Client-Serveur.

6 Conclusion et perspectives

Au cours de ce semestre, nous avons réalisé dans le cadre de l'Informatique Répartie un Bomberman distribué. L'objectif était que jusqu'à 4 personnes puissent jouer simultanément sur un même serveur mais sur des machines différentes. Pour établir ce projet, nous avons tout d'abord réalisé un Document de Spécification qui nous a permis d'établir l'ensemble des fonctionnalités que devra comporter ce projet. Ensuite, nous avons réalisé le Document de Conception avant de passer au développement du projet.
Malgré que notre équipe ait réussi à implémenter la plupart des fonctionnalités prévues dans le Document certaines ont été mises de côté car elles n'étaient pas prioritaires compte tenu du temps dont nous disposions. Parmi ces fonctionnalités on retrouve le choix de l'avatar par le joueur ainsi que toutes les fonctionnalités relatives aux bonus. Pour qu'un joueur puisse choisir l'avatar de son choix, il aurait fallu développer plus l'IHM mais ce n'était pas une priorité puisqu'il s'agit d'un projet d'informatique répartie. De même, pour inclure les bonus dans le jeu, il aurait fallu créer une classe pour gérer les éventuels bonus d'un joueur et modifier l'implémentation d'un joueur pour qu'elle les prenne en compte.
En ce qui concerne les fonctionnalités qui auraient pu être ajoutées non précisées dans les spécification, une IA plus performante aurait pu être envisagée. En effet, le fait d'avoir une IA qui ne se déplace pas aléatoirement aurait constitué un challenge supplémentaire pour le joueur. De plus, afin d'améliorer le serveur, nous aurions pu envisager que celui-ci supporte plusieurs parties en simultanées, soit créer une partie pour chaque groupe de 4 joueurs.
Enfin, ce projet d'informatique répartie nous a permis d'appliquer les notions vue en cours dans le cadre d'un projet plus concret et de remettre en œuvre des notions vues en UML notamment au travers des différents diagrammes pour la conception.