Rapport du projet long de Technologies Objet : Casse-Briques Multijoueur

Groupe 44
Matthias Roger, Jean-Luc Hervy, Jocelyn Bourduge, Mohammed Amine Tourari, Amine Er-roundi, Robin Xambili

Table des matières

1 Introduction

Nous donnons ici le rapport de notre travail mené pour la réalisation de notre projet de développement d'un jeu de Casse-Briques multijoueur. Il s'est agit de mener ce projet en groupe de six, un effectif conséquent par rapport aux autres projets que nous avons eu cette année, et s'étalant sur plusieurs mois. Cet exercice a donc été particulier tant dans la gestion du travail de groupe que pour l'organisation sur la durée.
Nous détaillons ici, après avoir rappelé les objectifs du projets et leur évolution au cours du développement, l'architecture de notre code accompagnée d'explications détaillées et suivies d'un bilan de notre méthode de travail en groupe et dans le temps avec le modèle des "méthodes agiles".

2 Objectif Général du projet et évolutions

Notre projet consiste à réaliser un jeu de casse-briques permettant de joueur à plusieurs joueurs sur un même ordinateur. Le casse-briques est un jeu d'arcade dont le principe général est de détruire, au moyen d'une ou plusieurs balles, un ensemble de briques se trouvant dans un niveau. Pour renvoyer la balle, le joueur controle une raquette pouvant être déplacée le long du bord de l'écran de jeu.

2.1 Principe de base

Chaque joueur contrôle une barre sur un des bords de la carte de jeu. Cette barre permet de renvoyer les balles dans une direction dépendant de l'endroit du rebond sur la barre, permettant au joueur de viser. Si une balle traverse le bord d'un joueur, celui-ci perd une vie. Un joueur perd quand il n'a plus de vie. Le but du jeu est d'être le dernier joueur en vie, ou de casser toutes les briques, dans ce cas le joueur avec le score le plus élevé gagne.

2.2 Mode multijoueur

Le mode multijoueur permet de jouer à deux, trois ou quatre joueurs controllant chacun une raquette sur un coté de l'écran de jeu.
Les balles peuvent être jouées indifféremment par tous les joueurs lorsqu'elles arrivent de leur coté, un coté n'ayant pas ou plus de joueur se comporte comme un mur sur lequel la balle rebondit.
Le nombre de joueurs, leurs positions et les touches de contrôle pour chacun d'eux sont déterminés ou modifiés par l'utilisateur avant de commencer la partie.
Il était question à l'origine de pouvoir jouer sur plusieurs supports différents et de permettre de jouer en réseau local ou par internet en multijoueur. Au profit du développement du jeu, le jeu final se joue sur ordinateur et et le multijoueur est permis uniquement sur une même machine.

2.3 Briques et niveaux

En plus des briques classiques disparaissant au contact de la balle, plusieurs briques spéciales sont intégrées au jeu :
-La brique incassable, se comporte comme un rebord du jeu et ne se brise jamais, elle rend le jeu plus difficile en obstruant le chemin des balles ;

-La brique explosive, fait disparaitre les briques qui l'entourent au moment ou elle est touchée ;
-La brique faisant apparaitres de nouvelles balles en disparaissant lorsqu'elle est touchée, balles que les joueurs doivent ensuite gérer, augmentant ainsi la difficulté en cours de partie.
Plus de briques spéciales étaient prévues à l'origine, nous n'avons pas jugé leur ajout prioritaire sur d'autres parties du jeu à développer.

Il est possible de jouer sur différentes "cartes" ou "niveaux" changeant le nombre, la position des briques ainsi que les proportions de briques spéciales.
La variété des types de cartes initialement imaginées a été réduite : les cartes triangulaires avec briques triangulaires pour les parties à trois joueurs demandaient de revoir le code des rebonds, des rebords, de l'affichage, or il nous fallait avant tout assurer un casse-brique multijoueur jouable sur des cartes carrées classiques.
De plus il était au départ prévu d'intégrer un éditeur de niveaux, l'interface n'a pas été implémentée mais on peut ajouter des fichiers.dat "dessinant" une nouvelle carte dans un éditeur de texte de manière intuitive.

2.4 Calcul du score

A chaque joueur en cours de partie est associé un score : lorsque la balle détruit une brique, le score du dernier joueur ayant touché cette balle augmente. Si un joueur manque une balle, il perd une vie et une partie de son score.
Ce score permet de départager les joueurs encore en jeu lorsque toutes les briques ont été détruites.

3 Architecture générale

3.1 Diagramme de classe complet

Le diagramme de classes suivant, en deux parties, décrit l'architecture complète du casse-briques multijoueur. Les classes qu'il présente ainsi que certaines des méthodes sont expliquées dans la section suivante.

FIGURE 1 – Diagramme de classe général partie 1

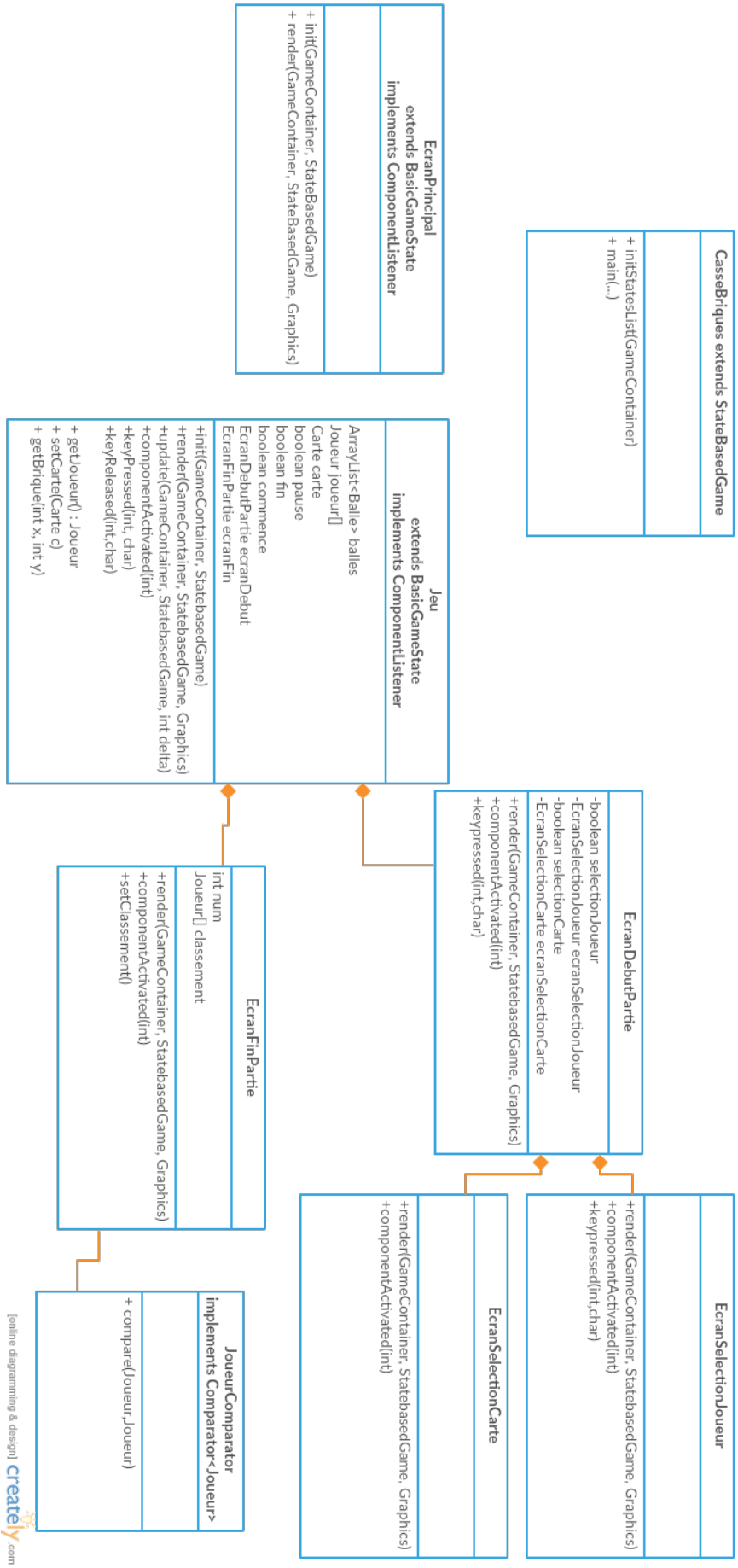
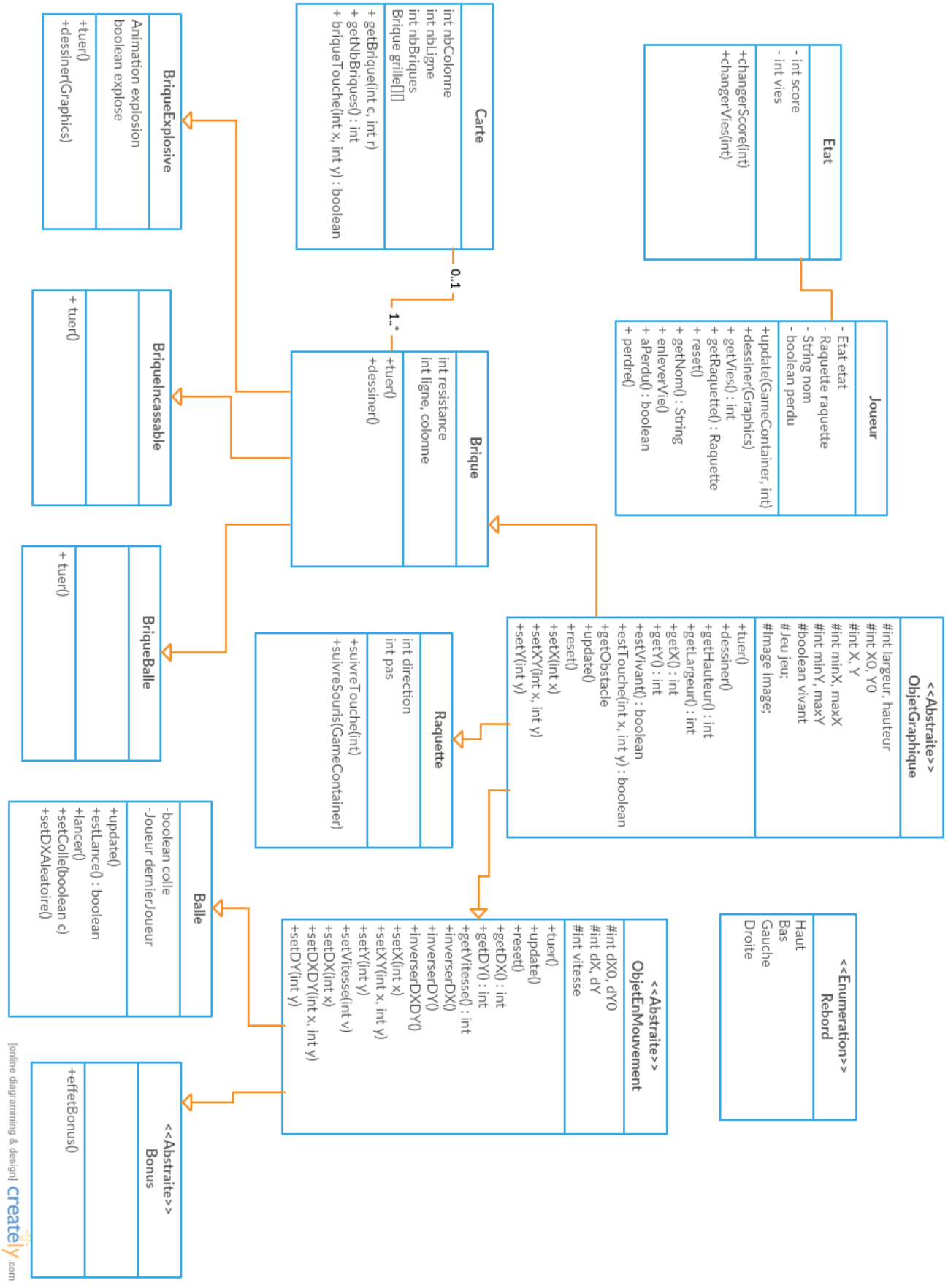


FIGURE 2 – Diagramme de classe général partie 2



3.2 Découpage en sous-systèmes pour le partage de a réalisation

Ce diagramme décrit le package "cassebriques" comprenant l'intégralité du jeu. A partir de ce diagramme général, le découpage s'est naturellement fait par classes ou groupes de classes présentant un lien direct entre elles. Nous avons ainsi séparé en différentes parties de travail les classes ou ensemble de classes détaillés dans la section suivante :
-Carte, Brique, BriqueExplosive, BriqueIncassable, BriqueBalle ;
-Raquette ;
-Rebord ;
-ObjetEnMouvement, Balle ;
-EcranDebutPartie, EcranFinPartie, EcranPrincipal ;
-Jeu ;
-CasseBriques ;

4 Fonctionnalités et réalisation

4.1 Les Objets graphiques

Pour une bonne gestion des objets du jeu, nos choix de conceptions se sont orientés vers la classe abstraites « Objet Graphique » qui factorisent les comportements communs des objets (balles, briques, raquette). Ceci nous a permis de définir un comportement standard des objets concernant notamment :
-Tuer un objet graphique.
-Récupérer les coordonnées de l'objet.
-Savoir si un objet est encore en vie.
-Savoir si un objet est touché.
-Dessiner l'objet sur l'interface graphique s'il est toujours en vie.
Les « Getters » et les « setters » standards et l'opérateur de réinitialisation de l'objet.
Ceci est donc un cadre global qui peut être complété par les sous classes, apportant ainsi plus de spécialisation selon la nature de l'objet et ses caractéristiques particulières. Notons bien qu'un objet en mouvement est l'objet qui bouge tout seul (La raquette n'est pas un objet en mouvement).

4.2 Première sous classe : « Objet en mouvement »

La classe abstraite « Objet en mouvement » qui héritent donc les caractéristiques d'un objet graphique, et on la complète par les méthodes et attributs qui décrivent le mouvement. Ainsi, les modifications apportées sont :
-De nouveaux attributs de vitesse et de déplacement.
-De nouveaux « Getters » et « setters » de vitesse
La méthode tuer doit annuler la vitesse et le déplacement avant d'appeler le « tuer » de la superclasse. Idem pour la réinitialisation.
-Des inverseurs de déplacement, utile pour le contrôle des rebords.
Dans le jeu, la seule classe qui hérite de « Objet en mouvement » est la classe « Balle ». Or, avec une telle conception, on laisse la porte ouverte pour toute possibilité de développement qui

propose de nouveaux objets dynamiques.

4.3 La classe Balle

La classe Balle hérite de Objet en mouvement. Elle possède des attributs propres qui sont :
-Un attribut pour savoir si la balle est collée à une raquette
-Un attribut pour connaître le dernier joueur qui a touché la balle
-Un attribut de temps pour lancer la balle automatiquement au bout d'un certain temps.
La classe possède notamment une méthode Balle.update() qui est appelée dans la méthode Jeu.update() de la classe Jeu. La méthode update de la classe jeu est la méthode qui est appelée régulièrement pour mettre le jeu à jour, et est inhérente à la classe BasicGameState dont Jeu hérite.
La méthode Balle.update() sert à mettre à jour les balles. Elle fait ceci :
- Lancer une balle collée à une raquette si un joueur appuie sur la touche action, ou si la balle est collée depuis trop longtemps.
-Modifier les coordonnées des balles en fonction de leur vitesse et de leur direction (par défaut toutes les balles ont la même vitesse).
-Détecter une collision avec Jeu.getCollision(), et faire appel à Balle.rebondir() dans ce cas.

La méthode rebondir() distingue deux cas : L'objet touché est une raquette ou pas.
S'il s'agit d'une brique ou d'un bord de l'écran, on inverse alors dX lorsque le rebord touché est horizontal, dY sinon, qui représentent en quelque sorte le vecteur vitesse. Dans le cas de la briques, on tue la brique et on augmente le score du dernier joueur qui a touché la balle.

Si c'est une raquette, rebondir() fait appel à la méthode rebondRaquette(Raquette raquette) qui calcule la nouvelle trajectoire de la manière suivante : tout se passe comme si la raquette était une portion de cercle et la balle arrivait avec une incidence normale à celui-ci. L'angle correspondant à la nouvelle direction est calculé à partir du cosinus, position de la balle relative au centre et la taille de la raquette, et les nouveau dX et dY calculés par projection sur les deux axes de la vitesse de la balle selon cette direction.

4.4 Deuxième sous classe : « Raquette »

La classe « Raquette » hérite de « Objet Graphique », et définit plus précisément le comportement de la raquette relatif aux commandes du joueurs. Ce n'est donc pas un objet en mouvement. Au début du jeu, une balle est collée à la raquette. La raquette bouge dans les deux directions selon les commandes du joueur (souris ou touches). Les méthodes pour suivre les commandes sont donc nécessaires, et c'est le jeu qui les utilise selon le choix des joueurs. Les attributs de fonctionnement ajoutés sont donc : Balle Collée à la raquette qui doit être lancée. Direction du mouvement. Le rebord de la raquette : Celui-ci est unique et change selon la direction.

4.5 Troisième sous classe : « Brique »

La brique est l'objet visé par les balles pour être détruit. Il est caractérisé dans une carte par ses indices de ligne et de colonne.
Pour avoir la liberté de définir la durabilité d'une brique, on définit un attribut résistance,

qui nous permettre de générer avec souplesse des briques de résistances différentes. Ainsi, définir une brique « Semi-incassable » revient à doubler la résistance par rapport aux briques standards.
La méthode « tuer » est donc redéfinit, et décrémente juste la résistance, et n'appelle le tuer graphique que si la résistance est nulle. La classe « Brique » se contente de ces spécialisations qui définissent ce qui est communs aux briques, et laisse à ses différentes sous-classes le soin de mieux spécifier leur comportement.
Brique incassable :
Elle ne fait que redéfinir le « tuer » avec un traitement vide.
Brique explosive :
Avant de détruire la brique, elle récupère son entourage de brique, par le biais d'une méthode de « Carte » et les tuent aussi.
Elle apporte un enrichissement de « dessiner », en ajoutant des animations d'explosions.
Brique à balles :
Elle libère un nombre aléatoire de balles pour le dernier joueur.

4.6 Utilisation de Slick2D

Nous avons choisi d'utiliser la librairie Slick2D pour gérer l'affichage et la logique du jeu dans le but de simplifier l'implémentation de l'interface graphique. En effet, cette librairie est prévue pour la création de jeu 2D, elle était donc adapter à notre projet.

Les BasicGameState de Slick2D :

La classe BasicGameState correspond à un état de jeu. Elle contient les méthode init, enter, render, update, leave.
La méthode init est appelée au lancement du jeu et permet de charger les éléments graphiques du jeu (images, ...).
La méthode enter est appelée lorsque l'on entre dans cet état de jeu et permet d'initialisée l'état.
Les méthodes render et update sont appelées en boucle lorsque le jeu est dans cet état. La méthode render gère l'affichage et la méthode update gère la logique du jeu.

4.7 Classe Principale CasseBriques

La classe principale est la classe CasseBriques qui hérite de la classe StateBasedGame de la librairie slick2D. Cette classe contient donc la méthode main qui lance le jeu. Cette classe ajoute deux BasicGameState : EcranPrincipal et Jeu.

4.8 Classe EcranPrincipal

C'est l'état initial du jeu. Elle permet d'afficher un menu permettant d'accéder aux autres états du jeu ou de quitter le jeu.

4.9 Classe Jeu

Cette classe gère le début de partie, la partie en cours et la fin de partie. Différents booléens indique si la partie a commence ou est fini.
Ainsi, la méthode render affiche les éléments de la partie si elle est en cours, appelle la méthode render de EcranDebutPartie si c'est le début de partie, la méthode render de EcranFinPartie si la partie est fini.

De même, les différents objets graphiques de la partie sont mis a jour dans la méthode update si la partie est en cours.

4.10 Classe EcranDebutPartie

Cette classe gère le début de partie. Elle permet de créer les différents objets de la partie notamment de sélectionner les joueurs et la carte de jeu. La sélection des joueurs et de la carte sont gérées dans les classes EcranSelectionJoueur et EcranSelectionCarte afin qu'elles puissent être réutilisées à l'avenir. Différents booléens permettent de tester si la sélection de joueur ou de la carte est en cours.
La méthode render permet donc d'afficher les différentes partie du début de partie grâce aux méthodes render de EcranSelectionJoueur et EcranSelectionCarte.

4.11 Classe EcranSelectionJoueur

Cette classe permet de choisir les différents attributs d'un joueur (nom, commandes) puis de le créer. Le joueur est ensuite ajouter au tableau de joueur de la classe Jeu lorsque l'utilisateur clique sur le bouton « ajouter ».

4.12 Classe EcranSelectionCarte

Cette classe permet de choisir une carte de jeu parmi celle prédéfinies. Il est aussi possible de créer une carte aléatoire. La carte est affiché pour avoir une prévisualisation. La carte sélectionné est affecté à l'attribut correspond de Jeu (via la méthode setCarte(Carte c)) lorsque l'utilisateur appuie sur «jouer».

4.13 Classe EcranFinPartie

Cette classe gère la fin de partie. Elle crée donc le classement des joueurs. Pour cela, la méthode setClassement() utilise la methode de classe sort de la classe Array et un comparateur specifique définit dans la classe JoueurComparateur qui réalise l'interface Comparator.

4.14 Classe Joueur

La classe Joueur possède comme attributs : une raquette, un état, des commandes et un nom. Ses méthodes sont notamment des getters et des setters ainsi que les méthodes render et update.
La methode render affiche la raquette et l'état du joueur. La méthode update met a jour la raquette en appellant suivreTouche() ou suivreSouris().

4.15 Classe Etat

Cette classe gère le nombre de vies et le score d'un joueur. Ses méthodes sont les setters et getters des attributs.

4.16 Classe Carte

Cette classe gère la disposition des briques dans le jeu. Elle possède comme attributs une grille de brique, un nombre de ligne et un nombre de colonne. La méthode dessiner de Carte appelle la méthode dessiner sur chaque brique de la grille.

Cette classe possède deux constructeurs : l'un permet de créer une carte aléatoirement, l'autre prend un fichier en argument qui permet de créer la carte. Ainsi, il est possible de créer une carte dans un fichier. Le nom, le nombre de lignes, le nombre de colonnes et la disposition des briques est disponible dans le fichier. Dans un fichier de carte, certains caractères correspondent à certaines briques (par exemple : i → brique incassable).

Un fichier de carte a la forme suivante :

FIGURE 3 – carte.dat

```
nom de la carte/14/14/$
i221  22  122i
2x1   11   1x2
21          12
1   i1221i   1
   ib1111bi
   112  211
21 21    12 12
21 21    12 12
   112  211
   ib1111bi
1   i1221i   1
21          12
2x1   11   1x2
i221  22  122i
```

5 Les méthodes agiles

Le projet a été réalisé en faisant appel aux techniques des méthodes agiles. C'était une expérience très différente des autres travaux en groupe, surtout sur le projet de tout un semestre. Ceci nous a permis de mieux évaluer l'impact des méthodes agiles sur le bon rendement du travail.

On a commencé par nominer un responsable du groupe agile : XAMBILI Robin. Le groupe se réunissait régulièrement pour donner le travail effectué et prévu pour chaque membre. La tâche la plus intéressante au tout début du travail était se mettre d'accord sur l'architecture et les bons choix de conception.
On a défini notre Product Backlog, contenant toutes les classes qu'on doit réaliser, puis on a défini un cadre général des itérations :

5.1 Première itération

Première itération : Etre capable de produire la version la plus basique du jeu (Casse-briques mono joueur). Ceci nécessitait les classes standards (Carte, Brique, Balle, Raquette) et la gestion de l'affichage. Par manque d'expérience, le temps de l'itération s'était écoulé sans avoir réalisé la balle.

	Brique	Balle	Raquette	Affichage	Carte
Achevée	X		X	X	X
Inachevée		X			

On avait mis du temps pour choisir Slick au lieu de Swing pour l'affichage. Juste après on a assisté au TD SCRUM2, où on a été vivement encouragé d'évaluer la difficulté de chaque tâche avec une échelle donnée pour les autres itérations.

5.2 Deuxième itération

Deuxième itération : En se référant aux difficultés de la première itération, on a défini les tâches de cette seconde itération, en considérant les classes non achevées de la première. On avait prévu le mode multi joueurs, la gestion des vies et des rebords et la gestion des scores. A la fin de l'itération on avait réussi le tout sauf les scores.

	Balle	Multijoueurs	Rebords	Vies	Score
Achevée	X	X	X	X	
Inachevée					X

5.3 Dernière itération

Dernière itération : On a géré le restant des tâches : Carte prédéfinie et le classement, ainsi que les scores qui étaient encore inachevés.

Nous n'avons pas réussi à satisfaire la story d'importation du jeu en ligne ni la création de carte, mais on a pu coder les différentes briques, ce qu'on craignait ne pas avoir le temps

pour le faire aussi. La création de carte consiste juste à définir un fichier et l'importer. On peut toujours le faire implicitement mais on a pas conçu d'interfaces H/M pour le faire.

	Scores & Classement	Accès paramètres	Carte prédéfinie	Bonus	Jeu en ligne	Création carte
Achevée	X		X	X		
Inachevée		X			X	X

6 Conclusion

La réalisation de ce projet a été enrichissante du fait d'un travail de groupe sur une durée relativement longue. Se sont posés des problèmes d'organisation, de répartition du travail, de planification, de communication qui nous on forcé à revoir nos objectifs au cours du développement. Pour autant, nous avons réussi à faire un jeu fini, présentant au moins les fonctionnalités essentielles du projet initial, permettant bien de jouer à plusieurs à un jeu de casse-briques fonctionnel.
Du fait d'un calcul souvent aléatoire des directions, et d'une partie graphique importante, les tests ont été fait en lançant l'application et observant son comportemant dans le plus de situations possibles tout au long du développement.
Ce travail nous a aussi permis de mettre en pratique les notions étudiées en cours de Technologies Objet, ainsi que d'approfondire notre connaissance du langage java dans lequel le jeu a été codé.