Programmation fonctionelle

NEWTONOID

par

Wissal ABOUMEJD
Khadija AKKAR
Safae BELAHRACH
Louiza CHEKRAOUI

Groupe PR-L12-1

Département Science
du Numérique

TOULOUSE-INP ENSEEIHT

Date : 26 janvier 2024

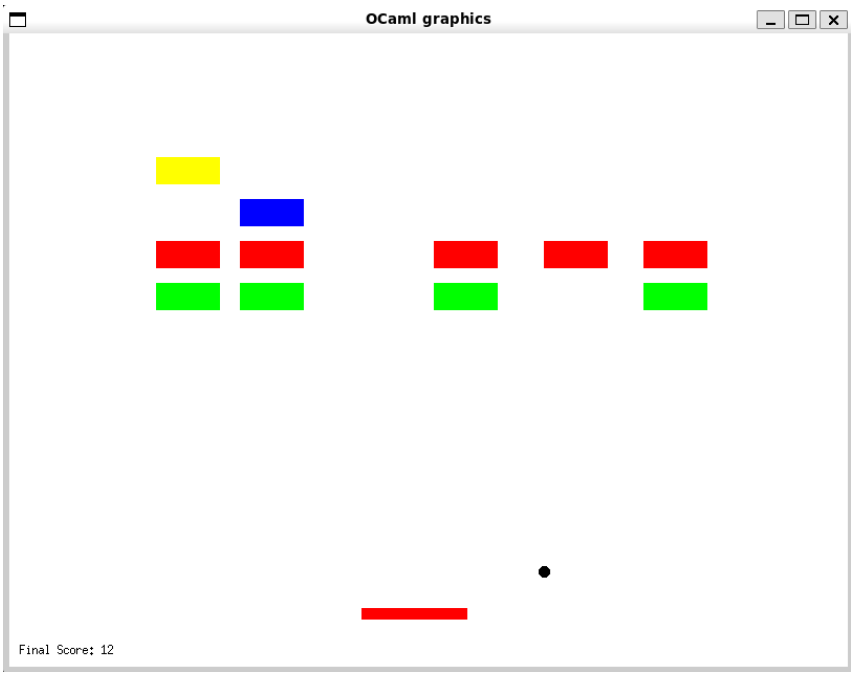
Table des matières

Table des figures

1. INTRODUCTION

Le but de ce projet est de reproduire de manière assez similaire le comportement du jeu auquel nous avons tous joué durant notre enfance : **Arkanoid**. Ce qui a été le plus challengeant est le fait de le coder en langage **Ocaml** en utilisant les notions de flux pour représenter l'évolution de l'état de jeu, le **circle detection collison** pour gérer les collisions et enfin la notion de **Quadtree** pour gérer l'espace de jeu sur le **Frame**.

Ce présent rapport s'organise selon trois axes principaux qui sont les suivants : dans un premier, nous allons mettre en avant nos choix de conception, puis dans un second temps nous vous présenterons d'un point de vue objectif un avis critique sur le rendu final et enfin, nous finirons par l'analyse des divers résultats du test.



2. CHOIX DE CONCEPTION

Au départ, nous avons récupéré le module **Frame** réalisé durant la séance de TP7 qui nous permet de créer des instances des modules que nous avons définis par la suite comme Bouncing, FreeFall, Paddle et Drawing.

2.1 Abstraction et Modularité

Nous avons choisi de diviser notre projet en plusieurs fichiers (paddle, ball, quadtree, mainBQB, iterator, Brick et game) pour avoir un code structuré et surtout pour pouvoir travailler en groupe et collaborer en parallèle. Au sein de ses fichiers, on a défini plusieurs modules. On a opté pour une représentation modulaire, car elle est pratique, et offre une certaine flexibilité, c'est-à-dire que l'appel des fonctions à l'extérieur du module est possible. On retrouve aussi des fonctions concernant le contact et le rebond comme pour la balle avec le **Frame** de notre jeu.
Chacun des modules que nous avons définis expose un type abstrait (interface du module) et le module est une implémentation sous forme de structure. Voici une brève présentation des modules importants :

2.1.1 Module FreeFall

Nous avons récupéré le module du TP tel qui l'ait. En particulier le type qui modélise l'état de la balle dans le systeme.

```
type etat = (float * float) * (float * float)
```

2.1.2 Module Paddle

Dans ce module, on définit le type t qui est un enregistrement des caractéristiques principales du Paddle (largeur, longueur, position selon x et y). On définit aussi différentes fonctions comme la création du paddle, mouvement du paddle selon le mouvement de la souris en positionnant la souris exactement au milieu du paddle afin de manipuler et rendre le jeu interactif.

```
type t={mutable x: float; mutable y: float; width: float; height: float;}
```

2.1.3 Module Bouncing

C'est le module qui se trouve dans le fichier principal **game** et responsable du rebond et de la collision entre la balle, la raquette et les briques en faisant appel aux modules **FreeFall**, **Paddle** et **Frame** . Pour que la balle rebondisse sur la raquette ou les briques, il faut définir des conditions de collisions selon x et y. Lorsque ces dernières sont vérifiées, le signe de la vitesse selon l'axe sur lequel la balle arrive sera inversé.

2.1.4 Module Drawing

C'est le module de représentation graphique de balle-raquette en 2D et la simulation s'obtient en appliquant draw à un flux de l'état de la balle combiné avec l'ensemble des briques.

2.1.5 Module Brick

Le module Brick implémente la signature définie dans l'interface **BrickInterf**, on y a défini un type t enregistrement qu'on va expliquer dans la partie suivante **(Les choix de types)** ainsi que deux fonctions qui permettent de créer et dessiner les briques dans le **Frame**.

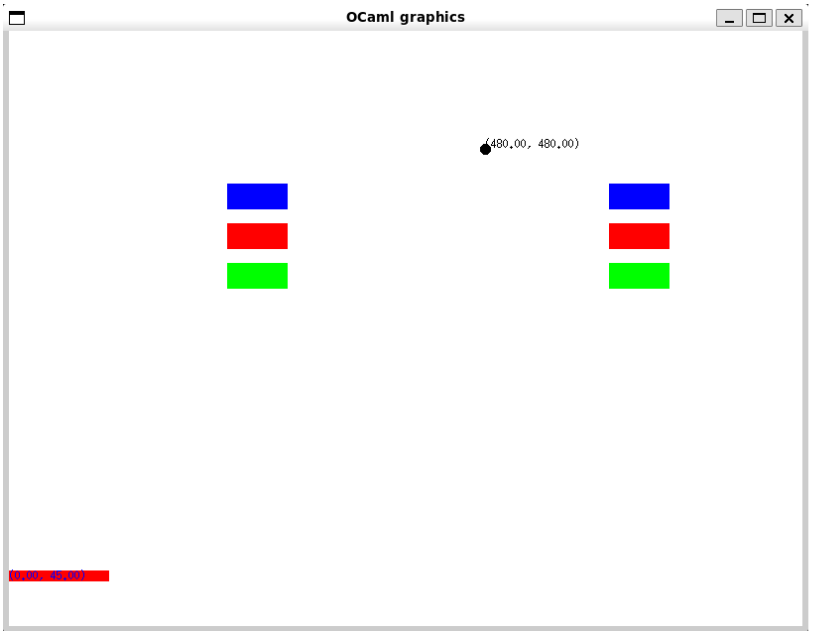


Figure 2.1 – Jeu Initial

2.2 Les choix de types

Dans cette section, nous allons nous intéresser aux choix des types de données qui nous ont permis une manipulation fluide des paramètres dans notre code. Pour tous les types définis, nous avons opté pour le type enregistrement, car il nous permet d'englober plusieurs caractéristiques de différents types pour un même objet.

2.2.1 Type Brick

Le type t dans le **module Brick** défini comme un enregistrement des coordonnées de position (x, y) d'une brique, son niveau (level), sa couleur, son hauteur/largeur et enfin le booléen qui nous permet de dire si la brique est visible ou pas.

```
type t={x:float; y:float; width:float; height:float; level:int;
   mutable visible:bool}
```

2.2.2 Quadtree

L'implémentation du Quadtree dans notre jeu était nécessaire pour avoir une gestion efficace de l'espace de jeu. Le Quadtree est un arbre qui permet de stocker efficacement des points

bidimensionnels. Dans l'arbre, chaque nœud possède au plus quatre enfants, (nw ; ne ; sw ; se).

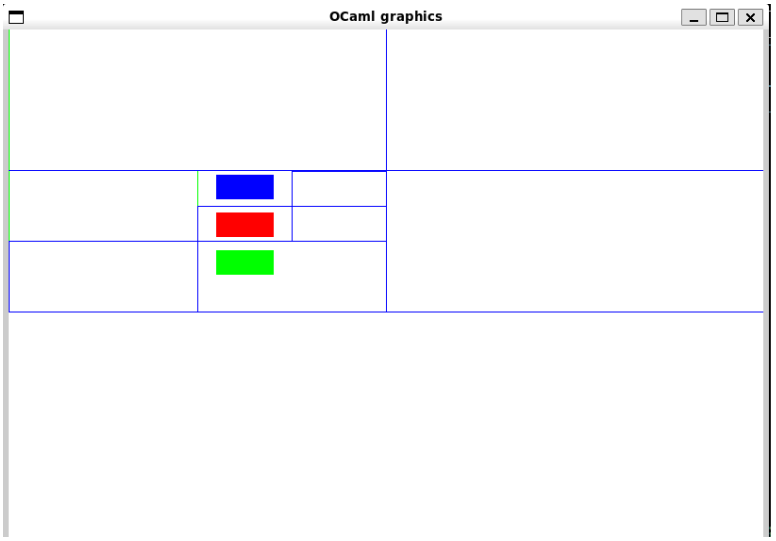


Figure 2.2 – Version initiale du QuadTree

Choix important : Nous avons décidé d'implémenter le Quadtree uniquement dans la moitié supérieure du Frame, car pour la deuxième moitié, nous avons pu gérer les collisions avec des conditions simples, donc il n'était pas primordial de définir le Quadtree dans la moitié inférieure.

— **Type région :** C'est un rectangle qui est délimité et défini par deux points, UL et LR, chacun étant un vecteur de coordonnées x et y.

```
type rect = {
  ul : vec;
  lr : vec;
}
```

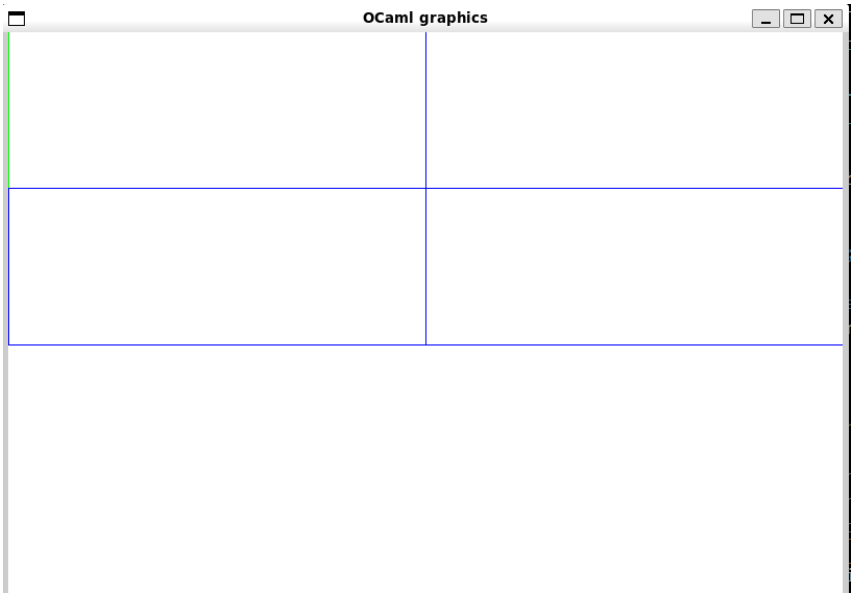


Figure 2.3 – Les régions du QuadTree

— **Type quadtree :** Le type qui définit l'arbre de division de notre fenêtre de travail.

```
type 'a quadtree =
  | Leaf of 'a list * rect \
  | Node of 'a quadnode
```

— **Type quanode :** Les nœuds de notre type quadtree.

```
type 'a quadnode = {
  centre : vec;
  region : rect; (
  nw : 'a quadtree;
  ne : 'a quadtree;
  sw : 'a quadtree;
  se : 'a quadtree;
}
```

2.3 Organisation des fonctions importantes

Dans cette section, nous allons nous intéresser à l'organisation des fonctions importantes, des choix algorithmiques qui nous ont permet d'avancer d'une maniére fluide tout au lon de notre projet et d'avoir un code structuré.

2.3.1 Collisions

Nous avons utilisé une fonction contact qui détecte s'il y a contact entre la balle et la boîte du jeu ainsi que le contact entre elle et la raquette en utilisant la position de la souris, tout en utilisant des fonctions de contact du fichier Ball.

En ce qui concerne le contact entre la balle et les briques, nous avons utilisé les fonctions contactBriquesX et contactBriquesY. La fonction contactBriquesY par exemple évalue si la balle entre en contact avec une brique en termes de coordonnées y. Elle utilise les bornes de la brique ((infx, supx), (infy, supy)), la position de la balle (x, y), la direction de la balle (dx, dy). La première partie vérifie si la balle se déplace vers le haut (dy >= 0.) et si elle se trouve dans la plage verticale de la brique. Elle vérifie également si la balle se trouve dans la plage horizontale de la brique élargie par 20 unités à gauche et à droite.

La deuxième partie de la condition traite du cas où la balle se déplace vers le bas (dy < 0.). Elle vérifie si la balle se trouve dans la plage verticale de la brique et de même, elle vérifie si la balle est dans la plage horizontale élargie de la brique.

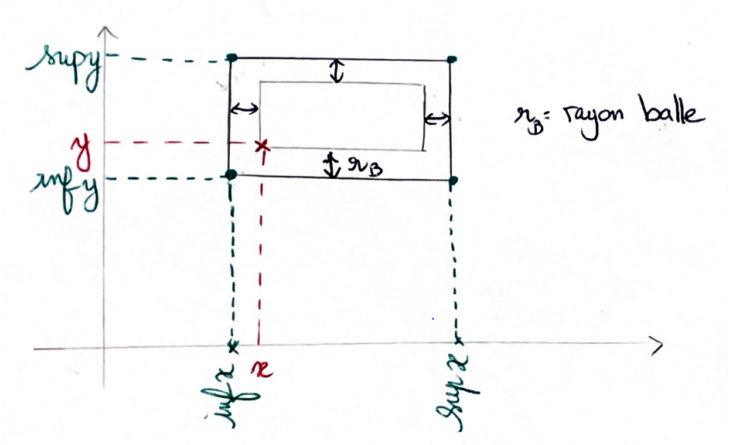


FIGURE 2.4 – Bornes Brique

2.3.2 Rebonds

— **Fonction rebondBox :**
- Gère le rebond de la balle sur la boîte de jeu ainsi que le rebond entre la balle et la raquette.
- La direction en x est ajustée par la fonction rebondx.
- La direction en y est inversée si la balle touche le bas ou le haut de la boîte ou le haut de la raquette.
- Si la balle atteint le bas de l'écran, elle est replacée à la position (300, 300).

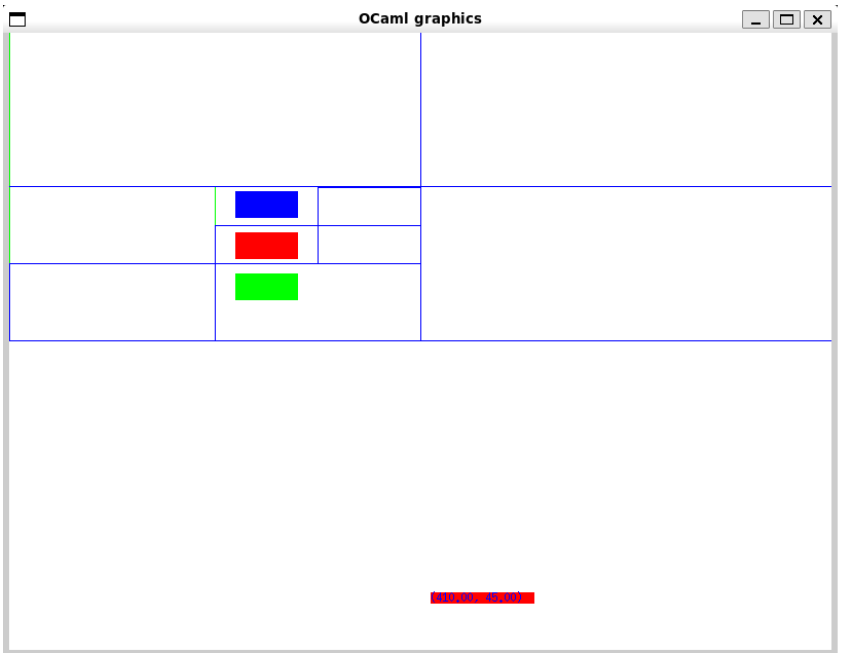


FIGURE 2.5 – Affichage de raquette - briques

— **Fonction rebondBrique :**
- Gère le rebond de la balle lorsqu'elle touche une brique.
- Les collisions avec les briques sont vérifiées en utilisant les bornes élargies.
- Les nouvelles composantes de la vitesse sont ajustées en fonction des collisions détectées en x et en y avec les briques.
- Si la balle atteint le haut de l'écran, elle est replacée à la position (300, 300).

— **Fonction until :**
La fonction until utilise un flux pour générer une séquence de valeurs jusqu'à ce qu'une condition soit satisfaite. La condition est déterminée par les fonctions contactBox et contactBriques, qui vérifient si la balle entre en contact respectivement avec la boîte de jeu ou la raquette et les briques. La fonction prend également deux autres fonctions (f1 et f2) qui génèrent des valeurs à chaque étape. La séquence générée s'arrête dès qu'une des conditions de contact est vérifiée, et la fonction retourne la séquence jusqu'à ce point.

— **Fonction synchronisation raquette - souris :**
Pour rendre le déplacement à la souris opérationnel, il est indispensable de mettre à jour les méthodes de position xg, xc et xd en recalculant la position à partir de Graphics.mouse pos.

3. REGARD CRITIQUE SUR NOTRE RÉALISATION

3.1 Lacunes et pistes d'amélioration

Bien que plusieurs aspects du jeu fonctionnent, quelques défis persistent et des améliorations restent à faire.
Par exemple, le Quadtree rencontre quelques problèmes, la division de certaines régions (pas toutes) ne se déroule pas comme prévu, c'est-à-dire que ces zones ne peuvent pas se diviser en quatre enfants. Ce problème induit un autre problème au niveau de l'introduction des briques dans le Quadtree. Plus précisément, dans ces zones, l'introduction d'une seconde brique n'engendre pas la division de la zone.

On pense que pour remédier à ce problème, on pense qu'on devrait ajouter une fonction de superposition qui tient en compte le fait que deux régions peuvent se superposer et par la suite construire un arbre plus cohérent.
La seconde lacune, c'est que nous n'avons pas eu le temps d'implémenter la partie de code qui nous permet de définir un nombre de vies au joueur (limité à trois tentations).
Nous aurions pu ajouter une fonction qui décrémente le nombre de vies quand la balle détecte un contact avec la borne inférieur de la Box et aussi ajouter l'affichage. Nous ce que nous avons fait, c'est que quand la balle détecte un contact avec la borne inférieure de la Box il revient au point d'origine (0,0).

3.2 Problèmes rencontrés et solutions apportées

Tout au long de la réalisation du projet, nous avons dû faire face à une panoplie de problèmes et d'obstacles qui nous ont demandé un long temps de réflexion.

3.2.1 Problèmes du contact

On a rencontré des problèmes de contact entre la balle et les briques, sur la face supérieure et inférieure des briques. Lors de la combinaison des deux conditions, une des deux n'était pas prise en compte. Cela a nécessité beaucoup de temps pour vérifier à chaque fois les conditions que nous avons définies.
Nous avons pu trouver les conditions exactes qui permettent le rebondissement de notre balle en cas de contact avec la brique verticalement. Ceci s'est fait après plusieurs vérifications et modifications des règles de contact que nous avions construites.

3.2.2 Problèmes d'accélération

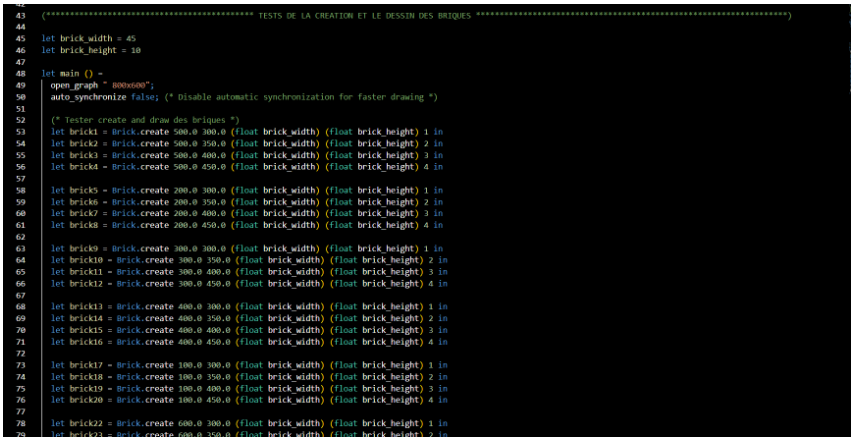
Au bout d'un certain moment, l'accélération de la balle augmente tellement qu'elle n'arrive plus à effectuer de contact avec les briques.

3.2.3 Exécution du programme en binaire

Par manque de temps, nous n'avons pas réalisé cette feature qui permet de lancer le jeu pas seulement avec utop mais aussi avec un appel à dune exec bin/newtonoid.exe.

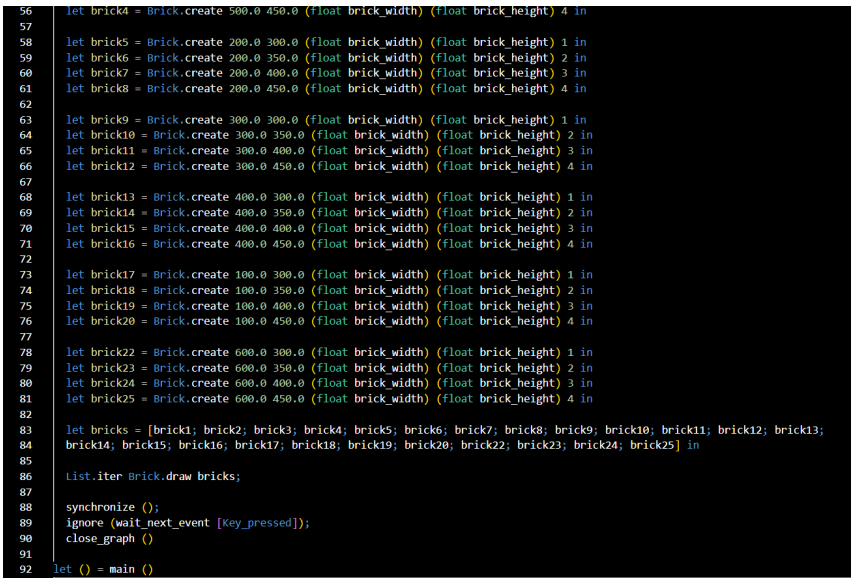
4. TESTS

Pour évaluer la fonctionnalité de création et de dessin des briques dans notre jeu Arkanoid, nous avons effectué un ensemble de tests au moyen d'une fonction main. Cette fonction crée une série de briques à des positions spécifiques sur l'écran, chacune étant associée à un niveau distinct, et nous avons vérifié visuellement si les positions et les niveaux étaient corrects en utilisant la fonction draw.



```
(********************************************** TESTS DE LA CREATION ET LE DESSIN DES BRIQUES **********************************************)

let brick_width = 45
let brick_height = 10

let main () =
  open_graph " 800x600";
  auto_synchronize false; (* Disable automatic synchronization for faster drawing *)

  (* Tester create and draw des briques *)
  let brick1 = Brick.create 500.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick2 = Brick.create 500.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick3 = Brick.create 500.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick4 = Brick.create 500.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick5 = Brick.create 200.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick6 = Brick.create 200.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick7 = Brick.create 200.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick8 = Brick.create 200.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick9 = Brick.create 300.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick10 = Brick.create 300.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick11 = Brick.create 300.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick12 = Brick.create 300.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick13 = Brick.create 400.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick14 = Brick.create 400.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick15 = Brick.create 400.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick16 = Brick.create 400.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick17 = Brick.create 100.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick18 = Brick.create 100.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick19 = Brick.create 100.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick20 = Brick.create 100.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick22 = Brick.create 600.0 300.0 (float brick_width) (float brick_height) 1 in
```

FIGURE 4.1 – Test de création des briques



```
  let brick4 = Brick.create 500.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick5 = Brick.create 200.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick6 = Brick.create 200.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick7 = Brick.create 200.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick8 = Brick.create 200.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick9 = Brick.create 300.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick10 = Brick.create 300.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick11 = Brick.create 300.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick12 = Brick.create 300.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick13 = Brick.create 400.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick14 = Brick.create 400.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick15 = Brick.create 400.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick16 = Brick.create 400.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick17 = Brick.create 100.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick18 = Brick.create 100.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick19 = Brick.create 100.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick20 = Brick.create 100.0 450.0 (float brick_width) (float brick_height) 4 in

  let brick22 = Brick.create 600.0 300.0 (float brick_width) (float brick_height) 1 in
  let brick23 = Brick.create 600.0 350.0 (float brick_width) (float brick_height) 2 in
  let brick24 = Brick.create 600.0 400.0 (float brick_width) (float brick_height) 3 in
  let brick25 = Brick.create 600.0 450.0 (float brick_width) (float brick_height) 4 in

  let bricks = [brick1; brick2; brick3; brick4; brick5; brick6; brick7; brick8; brick9; brick10; brick11; brick12; brick13;
  brick14; brick15; brick16; brick17; brick18; brick19; brick20; brick22; brick23; brick24; brick25] in

  List.iter Brick.draw bricks;

  synchronize ();
  ignore (wait_next_event [Key_pressed]);
  close_graph ()

let () = main ()
```

FIGURE 4.2 – Test du dessin des briques

Lors de l'évolution du projet, nous avons bien sûr essayé à chaque fois de tester nos différentes fonctions en implémentant des tests unitaires simples.

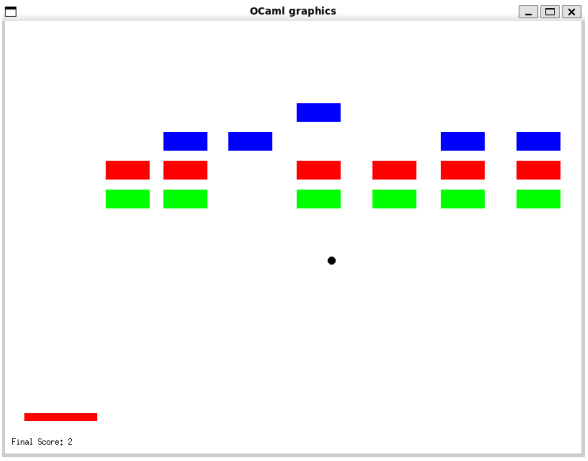


Figure 4.3 – Début du jeu

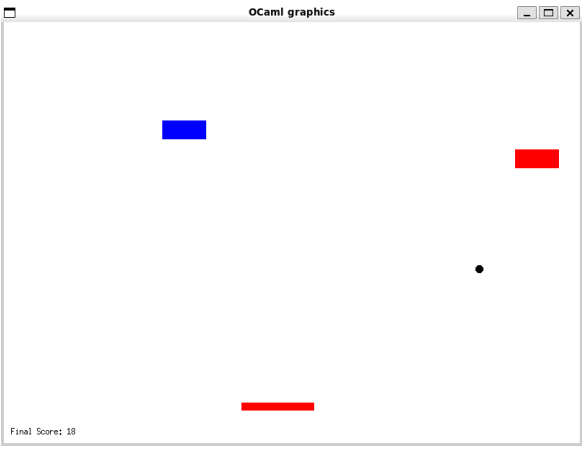


Figure 4.4 – Fin du jeu

5. MANUEL D'UTILISATION

5.1 Manuel d'utilisation

Pour lancer notre jeu, placez-vous dans le répertoire source/lib et exécutez le jeu à travers Utop en utilisant les commandes dans l'ordre suivant :

- `se placer dans lib`
- `dune utop`
- `open Libnewtonoid;;`
- `Game.main();;`
- `Start playing, Enjoy your game...`

6 Méthodologies

Le développement de notre projet Arkanoid a été organisé en sprints successifs, suivant les principes des méthodes agiles. Chaque sprint a été conçu pour atteindre des objectifs spécifiques et itérer sur le code existant. Voici un aperçu des sprints réalisés :

6.1 Sprint 1 - Établissement des Fondations

Objectif: Mise en place de l'environnement de développement OCaml.
Tâches:
-Configuration de l'environnement de développement.
-Création des modules de base tels que Frame, Bouncing, Paddle, et Drawing.

6.2 Sprint 2 - Abstraction et Modularité

Objectif: Définir des modules pour les différents aspects du jeu.
Tâches:
-Création des modules FreeFall, Brick, et les fonctions associées.
Abstraction des caractéristiques principales du Paddle.

6.3 Sprint 3 - Gestion des Collisions

Objectif: Implémentation de la détection des collisions.
Tâches:
-Développement des fonctions de collisions entre la balle, la raquette, et les briques.
-Intégration des modules de collision dans le module principal du jeu.

6.4 Sprint 4 - Intégration du Quadtree

Objectif: Implémentation du Quadtree pour une gestion efficace de l'espace de jeu.
Tâches:
-Définition du type Quadtree et de ses composants.
-Intégration du Quadtree dans le code principal pour optimiser la détection des collisions.

6.5 Sprint 5 - Tests

Objectif: Intégration des tests et ajustements en fonction des retours.
Tâches:

-Révision du code en fonction des retours obtenus lors des réunions de sprint.

6.6 Sprint 6 - Finalisation et Rédaction du Rapport

Objectif: Finalisation du jeu et préparation du rapport.
Tâches:
-Correction des derniers problèmes identifiés.
-Rédaction du rapport final décrivant les choix de conception, les défis rencontrés, et les solutions apportées.

6.7 Outils utilisés

Nous avons choisi **GitHub** comme plateforme principale pour la publication et la gestion de notre code source, garantissant ainsi une traçabilité et une collaboration optimale entre les membres de l'équipe.

Parallèlement, l'utilisation de **Trello** a été essentielle pour planifier et suivre nos tâches de manière visuelle, offrant une vue d'ensemble cohérente de l'avancement du projet.

Les réunions hebdomadaires, et même une fréquence accrue pendant les périodes de vacances, ont été organisées de manière efficiente grâce à des outils de visioconférence tels que **Zoom** et **Discord**. Ces initiatives ont contribué de manière significative à la cohésion de l'équipe et à la réussite de notre projet.

7 Conclusion générale

Pour conclure, ce projet a été très enrichissant en termes de notions à comprendre, analyser et à maîtriser. En effet, il constitue ainsi une nouvelle expérience enrichissante. Sur le plan pratique, ce projet nous a permis de mûrir nos compétences en programmation Ocaml et de consolider les notions vues lors des TPs et les utiliser dans un contexte plus complexe.Du point de vue travail en groupe, la prise de décision sur chaque étape du projet était une prise de décision collective, ce qui a demandé une bonne communication. Nous avons tous fait de notre mieux pour contribuer de manière significative à l'avancement du projet,pour cela nous avons remedier à plusieurs outils de communications tel que Trello et de nombreuses réunions Zoom.
Nous avons réussi à tous contribuer au projet grâce à notre détermination, chacun avec son point de vue et ses idées. Nous sommes convaincus que les compétences acquises au cours de ce travail collaboratif, seront précieuses dans nos futures collaborations professionnelles.

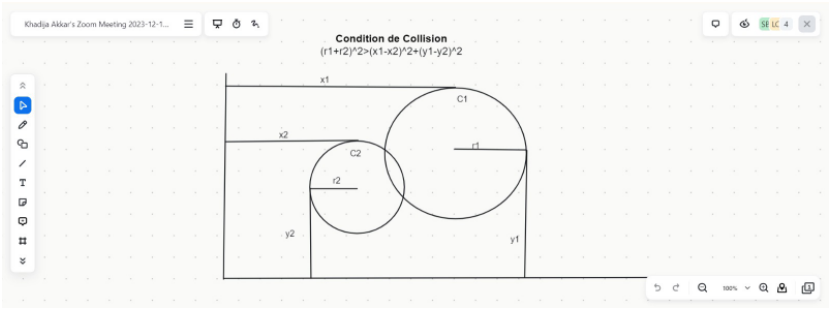


Figure 1: Image de l'une de nos réunions sur zoom

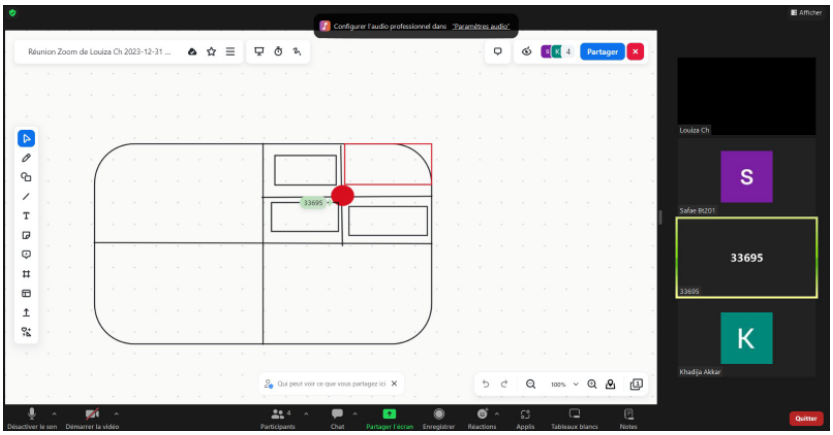


Figure 2: Image de l'une de nos réunions sur zoom

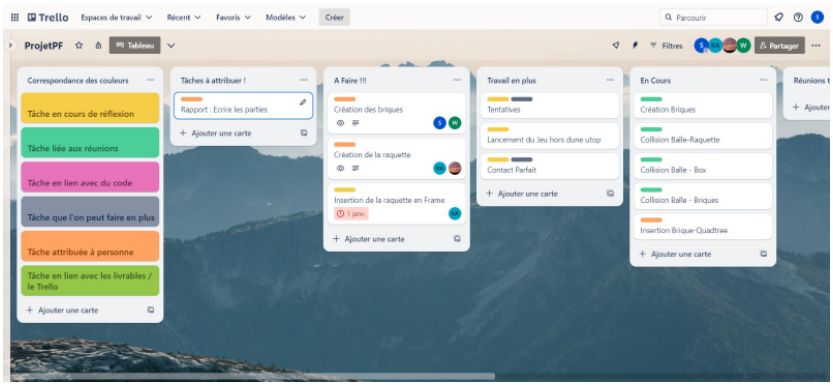


Figure 3: Extrait de notre contribution sur trello