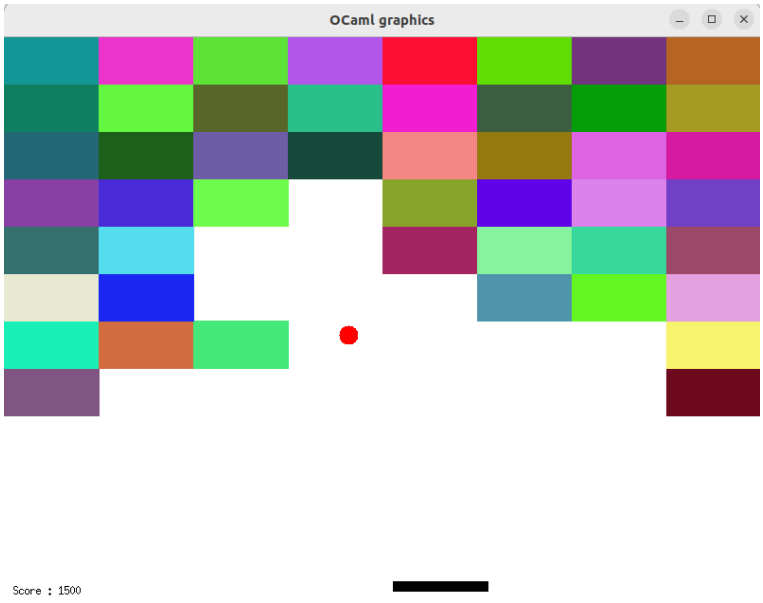
Projet de Programmation Fonctionnelle : NEWTONOID

Canon Ayoub
Barrier Robin
Blot Olivier
Bridoux Antoine



Département Sciences du Numérique - Deuxième année
2023-2024

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est de reproduire en langage OCaml un jeu de type casse-brique en s'inspirant du jeu *Arkanoid*. Dans ce jeu, une balle soumise à la gravité doit rebondir sur une barre contrôlée par le joueur sachant que la barre peut se déplacer uniquement de manière latérale. L'objectif est de faire rebondir la balle sur des briques qui disparaissent après contacte avec la balle. Un score est mis à jour pour chaque brique détruite. Enfin, si le joueur vient à rater la balle, celle-ci tombe et la partie est finie.

2 Contenu du rendu

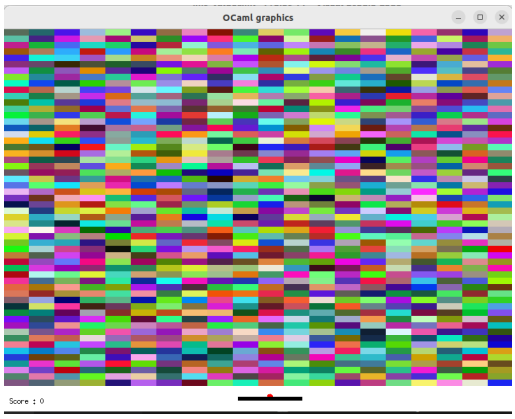
Le rendu comporte, en plus de ce rapport, les fichiers suivants :

— *newtonoid.ml* : fichier principal gérant l'affichage du jeu et utilisé pour l'exécution du jeu
— *game.ml* : fichier gérant l'interaction entre les différents éléments et la mise à jour de l'état du jeu
— *init_values.ml* : fichier définissant les valeurs initiales des éléments du jeu
— *iterator.ml* : fichier fourni implémentant des flux
— *briques.ml* : fichier implémentant la gestion des briques
— *palette.ml* : fichier implémentant la gestion de la barre déplaçable par l'utilisateur
— *quadtree.ml* : fichier implémentant un quadtree permettant la gestion efficace de l'espace de jeu

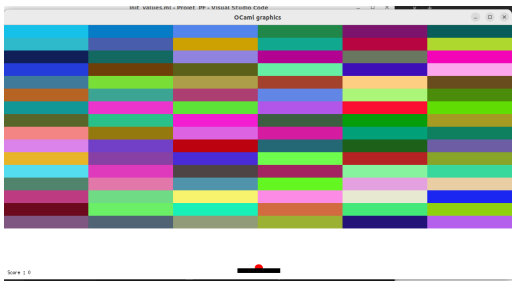
3 Architecture du code

3.1 Modularité

Les figures 1 et 2 illustrent différentes configurations possibles de notre jeu. Ces configurations sont rendues possibles grâce au fichier *init_values.ml* dans lequel les principaux paramètres du jeu sont modifiables. Les paramètres modifiables et les explications de cette implantation sont détaillés plus loin dans la partie 5.5 de ce rapport.

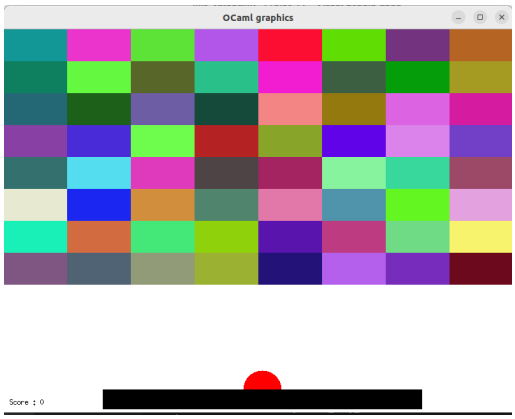


(a) Modification de la hauteur des tuiles

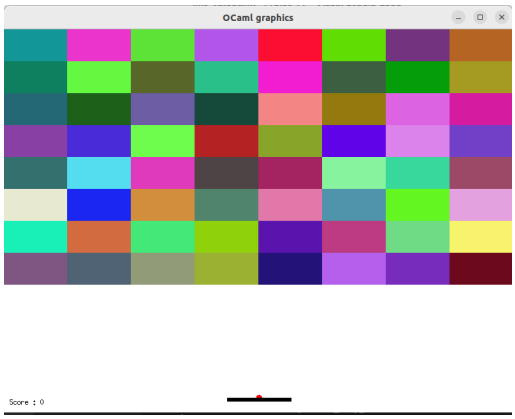


(b) Modification de la largeur des tuiles

FIGURE 1 – Modification possible de la hauteur, de la largeur et du nombre de tuiles



(a) Augmentation taille balle et palette



(b) Diminution taille balle et palette

FIGURE 2 – Modification possible de la taille de la balle et de la palette

4 Validation des critères de conception

4.1 Utilisation des flux

La conception de notre jeu repose sur l'utilisation de flux notamment pour la gestion de l'état du jeu. L'état global du jeu englobant à la fois l'état de la palette, de la balle, du score et celui des briques est représenté par un flux permettant ainsi la mise à jour de ces états et l'affichage à l'écran.

L'avantage de cette façon de faire est que les flux s'évaluent paresseusement. De ce fait, l'appel à la fonction de mise à jour du flux ne se fait pas à chaque frame mais se fait uniquement lorsqu'il y a besoin de recalculer les états, comme par exemple lors d'un contact de la balle avec la palette.

4.2 Détection de collision

L'algorithme que nous avons choisi pour détecter la collision de la balle avec les briques est un algorithme que nous avons trouvé nous-même : nous prenons le vecteur vitesse de la balle, le normalisons, le multiplions ensuite par le rayon de la balle ce qui nous donne après ajout aux coordonnées du centre le point à la surface de la balle le plus susceptible de rencontrer une brique. Ensuite nous récupérons la briques la plus proches et cherchons si ce point-là est à l'intérieur, si oui, il y a collision. Ce n'est pas parfait, mais compte-tenu du peu de calculs nécessaires et des résultats satisfaisants, c'est suffisant pour ce projet.

4.3 Gestion de l'espace de jeu par structure de données efficace

La gestion de l'espace de jeu est gérée par l'utilisation d'un quadtree définit dans le fichier *quadtree.ml*. Une telle structure de données permet de gérer efficacement l'espace de jeu en subdivisant l'espace en quadrants rendant plus rapide la gestion de collisions.

Au final, nous avons dû forcer la position des briques sur une grille de briques de la même taille pour pouvoir les faire marcher avec le quadtree, de ce fait, il aurait peut-être été plus pertinent d'utiliser un array pour avoir un indexage en O(1), mais le quadtree reste très efficace dans ce contexte.

4.4 Code fonctionnel

Le code suit le paradigme fonctionnel presque intégralement, excepté pour les entrées-sorties qui sont des effets de bords inévitables.

4.5 Modularité

Le fichier *init_values.ml* contient tous les paramètres de notre jeu pouvant ainsi être modifiables facilement au sein d'un seul et même fichier. En plus des éléments mentionnés ci-dessous, d'autres paramètres tels que la taille de la fenêtre ou les caractéristiques de la barre sont également modifiables.

4.5.1 Changer les paramètres dynamiques

Le module *PhysicsInit* permet de changer les paramètres dynamiques du jeu tels que la valeur de la gravité, le pas de temps, la vitesse initiale de la balle ou bien encore le facteur d'impulsion ajouté à la vitesse de la balle selon la zone sur laquelle la balle rebondie sur la barre de jeu.

4.5.2 Configuration initiale des briques

Le module *BriquesInit* permet la configuration initiale des briques. La dimension des briques, le score que chaque brique rapporte ainsi que la liste de briques du jeu peuvent être renseignés dans ce module.

4.5.3 Configuration initiale de la balle

Le module *BallInit* permet la configuration initiale de la balle en permettant notamment le choix de sa taille ainsi que de sa couleur.

4.6 Lisibilité du code

L'ensemble des fichiers et des fonctions de notre code possèdent des noms significatifs afin de faciliter la compréhension du code. De plus, chaque fonction possède un contrat explicitant le rôle de celle-ci et les paramètres qu'elle utilise. Le code présente également des commentaires quand cela est utile.

4.7 Ajout d'extensions

L'ajout d'extension est relativement aisé grâce à la modularité de notre code : en effet, ajouter des power-ups pourrait se faire en ajoutant un flux composé de listes d'objets à l'écran et avec les fonctions qui testent le contact avec la palette déjà faites, l'ajout est aisé. Permettre la duplication de balles pourrait se faire en transformant le flux de balle en flux de liste de balles et en itérant les fonctions déjà appliquées sur chaque liste de balles au lieu d'une seule. etc

4.8 Explication de choix effectués et différentes pistes d'amélioration

Notre principal choix a été l'utilisation du quadtree et il a causé un retard dans le développement du jeu à cause de la complexité et de la tentative de le faire fonctionner alors qu'il aurait probablement été plus pertinent de changer de méthode. Nous avons aussi choisi de réinitialiser le niveau à chaque mort, cela permet d'avoir plus de score au risque de perde sa dernière vie par erreur. Nous avons trouvé que cela ajoutait du challenge et permet à ceux qui le veulent de choisir leurs propres défis.

4.9 Conclusion

Ainsi, nous avons créé à partir des sources un jeu de casse brique fonctionnel ! Ce projet a permis vraiment de comprendre l'utilité des flux dans un véritable procontexte plus évolué que celui guidé des TPs et nous a vraiment appris à réfléchir d'une toute autre manière que celle du paradigme impératif.