Rapport Arkanoid

Colin Rémi, Ghamnia Karima, Mussard Cassandra, Song Mickael

ENSEEIHT & INSA Toulouse - 4ModIA
2022-2023

1 Partie I : Introduction

Ce projet vise à réaliser un jeu de type casse briques (Arkanoid) en Ocaml. Le but de ce jeu est de détruire le maximum de briques sans que la balle passe derrière la raquette.

Pour lancer le jeu, veuillez bien suivre les instructions du README.md.
Au début du jeu, le joueur doit cliquer sur la souris pour lancer la balle qui ensuite se déplacera seule. Il aura par la suite seulement la main sur la raquette qu'il pourra déplacer afin de faire rebondir la balle pour détruire les briques. Le score augmentera à chaque brique cassée. Lorsque toutes les briques d'un niveau sont cassées, le joueur passe au niveau suivant.
Si les vies du joueur tombent à 0 (le joueur perd une vie lorsqu'il n'arrive pas à rattraper la balle), un menu apparaît avec les options de rejouer et de quitter.

2 Partie II : Répartition des tâches

Nous avons travaillé ensemble à chaque étape du projet car les librairies étaient étroitement liées les unes aux autres. Nous avons commencé à commit notre travail sur le gitlab lorsque le jeu commençait à être bien entamé.

3 Partie III : Architecture du jeu

3.1 Librairie balle

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion de la balle. Une balle a une position (entier*entier), un vitesse (entier) et un paramètre correspondant à si la balle est attachée à la raquette (position initiale au lancé de la partie) ou non (booléen). Cette librairie est composée des fonctions suivantes :

- create_ball : fonction qui crée une balle à partir des coordonnées de la raquette et des paramètres prédéfinis
- draw_ball : fonction qui dessine une balle à l'écran avec les coordonnées spécifiées
- update_ball_position_before_click : fonction qui met à jour la position de la balle avant un clic de souris en utilisant les coordonnées de la raquette et les paramètres prédéfinis
- update_ball_position : fonction qui met à jour la position de la balle en fonction des coordonnées et des vitesses données
- check_collision_racquet : fonction qui vérifie s'il y a une collision entre la balle et la raquette
- collision_sol : fonction qui vérifie si la balle a une collision avec le sol

3.2 Librairie brique

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion d'une brique. Une brique a une position (entier*entier) et un paramètre correspondant à si la brique est détruite ou non (booléen). Cette librairie est composée des fonctions suivantes :

- creation_brique : fonction qui crée une brique avec les coordonnées spécifiées
- dessiner_brique : fonction qui dessine une brique à partir des coordonnées spécifiées
- collision_brique_balle : fonction qui vérifie s'il y a une collision entre une brique et une balle à partir de leurs coordonnées
- update_visibilte_brique : fonction qui met à jour la visibilité d'une brique après une collision
- supprimer_brique : fonction qui supprime une brique en collision avec une balle d'une liste de briques
- detecter_collisions : fonction qui détecte les collisions entre une balle et une liste de briques

— mettre_a_jour_briques : fonction qui met à jour la liste de briques après une collision avec une balle

3.3 Librairie raquette

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion de la raquette. Une raquette a une position (entier*entier), un largeur (entier) et une longueur (entier). Cette librairie est composée des fonctions suivantes :

— create_racquet : fonction qui crée une raquette à partir des coordonnées et dimensions spécifiées

— move_racquet : fonction qui déplace une raquette en fonction de la position de la souris

— draw_racquet : fonction qui affiche une raquette à l'écran avec les paramètres spécifiés

3.4 Librairie joueur

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion du joueur. Un joueur a des vies (entier), un score (entier) et un niveau (entier). Cette librairie est composée des fonctions suivantes :

— create_joueur : fonction qui crée un joueur avec un nombre initial de vies

— decrementer_vie : fonction qui décrémente le nombre de vies du joueur

— joueur_mort : fonction qui vérifie si le joueur est mort (nombre de vies inférieur ou égal à 0)

— incrementer_score : fonction qui incrémente le score du joueur

— draw_vie joueur : fonction qui affiche le score du joueur à l'écran

— mettre_a_jour_briques : fonction qui affiche le nombre de vies du joueur à l'écran

— draw_niveau : fonction qui dessine le niveau du joueur sur l'écran

3.5 Librairie fenetre

Il s'agit d'une libriaire qui permet de créer la fenetre de jeu. Elle est composée des fonctions suivantes :

— init_window : fonction qui initialise la fenêtre graphique en définissant sa taille

— clear_screen : fonction qui efface l'écran de la fenêtre graphique

3.6 Librairie menu du jeu

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion du menu du jeu. Elle est composée des fonctions suivantes :

— draw_text_centered : fonction qui calcule les coordonnées x et y pour centrer le texte

— afficher_menu : fonction qui affiche le menu du jeu Arkanoid avec le titre et les options de jeu

— demarrer_jeu : fonction qui lance le jeu en affichant le menu et en attendant le choix de l'utilisateur

3.7 Librairie Game Over

Il s'agit d'une libriaire proposant les fonctionnalités liées à la gestion de la fin de partie. Elle est composée des fonctions suivantes :

— draw_button : fonction qui dessine un bouton sur l'écran

— draw_game_over : fonction qui affiche l'écran "Game Over" avec les boutons Rejouer et Quitter

3.8 Paramètres

Il s'agit de tous les paramètres initiaux des différents objets de notre jeu.

4 Partie IV : Tests

Nous avons réalisé une fonction test pour chaque fonction de chaque librairies sauf pour le menu du jeu et le menu du Game Over car ils sont directement testables graphiquement sur le jeu.

Les fonctions de tests se trouve dans le répertoire /src/test

5 Partie V : Conclusion

Nous avons donc implémenté le jeu d'Arkanoid pour notre projet de programmation fonctionnelle. Au départ, nous avions codé notre jeu avec des mutables car cela nous semblait beaucoup plus simple de travailler avec. Nous avions ensuite dû tout recoder sans les mutables aux vues des contraintes du projet, ce qui explique entre-outre le temps mis pour notre premier commit.

Il s'agit d'un projet qui nous a globalement satisfait par sa compléxité croissante, avec tous les détails qui fallait rajouter au fur et à mesure. Il existe bien sûr d'autre manière que la nôtre de coder ce jeu et d'autres fonctionnalités à implémenter.