



ENSEEIH

PROGRAMMATION FONCTIONNELLE
RAPPORT

Projet Newtonoid

Élèves :

Romain HAESSLER
Benjamin PASCAL
Leïlie CANILLAC
Vianney HERVY

Enseignant :

Guillaume DUPONT

23 janvier 2025

Table des matières

1	Introduction	1
2	Objets	1
3	Niveaux de jeu	1
4	Paramètres	2
5	Contrôles	2
5.1	Démarrer le jeu	2
5.2	Mettre le jeu en pause	2
6	Fin du jeu	2
7	Améliorations	2

1 Introduction

2 Objets

Chaque objet du jeu est modélisé par un type enregistrement afin de permettre un accès efficace aux différents champs (par opposition aux tuples qui nécessitent du filtrage par motif). Chaque type objet est appelé **t** et est défini dans son propre module qui regroupe les opérations associées. Parmi ces opérations, on note surtout **make**, **draw** et **draw_shadow** et qui permettent de respectivement créer un objet et de l’afficher à l’écran et de dessiner son ombre.

3 Niveaux de jeu

Initialement, un niveau de jeu était défini en OCaml et chaque brique ajoutée séparément. C'était à la fois verbeux et peu pratique. Nous avons donc "développé" une syntaxe très simple pour définir un niveau textuellement. Chaque type de brique est représenté par un caractère différent et l'air est représenté par un espace. Les bordures sont fixées avec les caractères tiret et barre verticale. Deux exemples de niveaux sont donnés à la figure 1.

-----		-----
#####		@+ + + + + @
=====		@ + + + + + +@
+++++++		@+ + + + + @
		@ + + + + + +@
		@@@@@@@@@@@@@@@
-----		-----

FIGURE 1 – Exemples de niveau

4 Paramètres

Nous avons vite compris que la possibilité de changer efficacement les paramètres était cruciale pour le développement et pratique pour le joueur. C'est pourquoi nous avons créé un module `Params` qui contient tous les paramètres du jeu. L'idée était de paramétrer les autres modules en fonction de `Params`. Ainsi, chaque module est en partie défini par les paramètres qui lui sont donnés.

Cette paramétrisation des modules permet de diminuer le nombre d'arguments des fonctions ainsi que la présence de valeurs "magiques" dans le code.

Pour faciliter la gestion des paramètres, nous avons développé un "parsing" primitif de fichier ".conf" qui récupère les paramètres du jeu et crée un module `Params` correspondant. Cette fonctionnalité est implémentée via le foncteur (au sens d'OCaml) `Params.Make` qui prend en argument un simple module contenant le nom du fichier de configuration et renvoie un module `Params`.

5 Contrôles

5.1 Démarrer le jeu

Initialement, le jeu est en pause, la balle collée sur la raquette. Lors du clic du joueur, la balle est propulsée vers le haut et les mises-à-jour commencent à s'effectuer. Le clic change l'état de jeu de `Init` à `Playing`.

5.2 Mettre le jeu en pause

Le joueur a la possibilité de mettre le jeu en pause en cliquant sur le bouton de sa souris. La fonction `State.update` change le `game_status` de `Playing` à `Paused` (et inversement) lorsqu'un clic est détecté. Étant donné que la boucle de mise-à-jour continue de tourner mais sans modifier l'état de jeu, il est impératif de rajouter un `sleep` lors d'un clic, pour éviter de le comptabiliser sur plusieurs frames.

6 Fin du jeu

La fin du jeu intervient dans deux cas : le joueur n'a plus de point de vie (état `GameOver`), ou le niveau ne contient plus que des briques incassables (état `Victory`). Dans les deux cas, la mise-à-jour s'arrête et un message de fin s'affiche à l'écran.

7 Améliorations

- **Taille du texte** : Les messages de fin de jeu affichés à l'écran sont écrit en petit. Le module `Graphics` d'OCaml ne possède pas d'implémentation de la fonction `set_text_size` pourtant spécifiée dans l'interface. Il est donc difficile de changer la taille d'écriture.