Projet Programmation 1

Effets en OCaml, Roguelike

12 novembre 2025

Rogue est un jeu sorti en 1980 dans lequel un personnage explore un réseau de souterrains en évitant des ennemis, à la recherche d'un trésor. Ce projet vise à implémenter un petit jeu de ce type en OCaml.

L'implémentation robuste d'un tel jeu dans le paradigme fonctionnel d'OCaml pose des défis uniques. Le flot de contrôle doit passer la main aux différentes entités qui jouent à tour de rôle ; ces entités ont une variété de comportements (complexes) différents.

Pour résoudre ces problèmes, nous allons utiliser les **effets**, disponibles en OCaml 5.3. Nous allons simuler une réserve de *fils* (*thread pool*) coopératifs : chaque entité aura son propre fil d'exécution au sein d'une (ou plusieurs) fonction(s). Les fils signalent par le biais d'un effet lorsqu'ils ont terminé leur tour de jeu, et se passent ainsi la main.

Consignes, attendus

Le projet est réalisé par groupes de trois. Trois jours sont banalisés.

J'attends un retour, au plus tard le **14 novembre 19h** (heure de Cesson-Sévigné). Vous devez rendre uniquement le code source, les scripts de compilation (dune) et un fichier README. Vous pouvez, au choix :

— m'envoyer par mail une archive ;
— m'inviter sur un projet git.

Le fichier `notation.md` fourni contient un barème de notation, et quelques conseils. Prenez connaissance des différents critères d'évaluation et gardez-les à l'esprit pendant le projet.

1 Dépendances

Le projet nécessite OCaml 5.3 (cf TP8), ainsi que le package `Notty` qui permet de réaliser des affichages dynamiques dans le terminal. Vous pouvez installer `Notty` via la commande suivante :

```
opam install notty
```

Le projet est compilé avec l'utilitaire de compilation `dune`.

```
opam install dune
```

Vous pouvez compiler dans le répertoire du projet via `dune build` et lancer le projet via `dune exec rogue`. Vous pouvez ajouter des tests à `test/test_rogue.ml` et les lancer avec `dune runtest`.

2 Base de code fournie

Le fichier `world.ml` contient les types et fonctions nécessaires pour décrire et modifier le monde du jeu. Le monde est un tableau mutable global de cases qui peuvent héberger (au plus) une entité.

Le fichier `ui.ml` contient le nécessaire pour afficher le jeu et interagir avec lui.

Le fichier `engine.ml` contient le moteur principal du jeu, qui gère la piscine de fils. Le moteur est basé sur une file où attendent les fils d'exécution des entités en pause. À chaque nouveau tour, une entité est défilée et la continuation de son exécution est exécutée jusqu'à terminaison, ou jusqu'à ce qu'elle lève l'effet `End_of_turn` (auquel cas elle est réinsérée dans la file).

Le fichier `utils.ml` contient des fonctions qui sont utiles à tous les types d'entités (se déplacer par exemple).

Le fichier `player.ml` contient les fonctions nécessaires pour contrôler le personnage jouable (le chameau), et éteindre le jeu. La fonction `camel` décrit le comportement du chameau : attendre une entrée clavier, l'exécuter, et recommencer via un appel récursif.

Le fichier `main.ml` est en charge d'initialiser l'état du monde au début du jeu et de lancer la boucle de jeu principale.

Vous pouvez compiler le code à l'aide de `dune`. Si tout se passe bien, vous pouvez maintenant faire se déplacer un chameau solitaire dans un petit désert.

3 Questions de base

Objectif 0. Pour le moment le chameau est seul et n'a donc pas besoin de passer la main à une autre entité. Ajoutez l'effet `End_of_turn` au bon endroit dans `player.ml` pour permettre au chameau de passer la main aux autres entités du monde.

Vous pouvez tester votre code en ajoutant temporairement un deuxième chameau dans le monde et en vérifiants qu'ils se déplacent bien à tour de rôle.

On va maintenant ajouter d'autres types d'entités contrôlées par le jeu.

Objectif 1. Créez un fichier `snake.ml` qui contient la logique de contrôle d'une entité qui se déplace aléatoirement à son tour.

Objectif 2. Créez un fichier `elephant.ml` qui contient la logique de contrôle d'une entité au comportement plus complexe :

- tant qu'elle n'a pas vu le chameau, elle vérifie au début de son tour si un chemin rectiligne (horizontal ou vertical) de cases vides la relie au chameau ;
- si ce n'est pas le cas elle fait un déplacement aléatoire et termine son tour ;
- si c'est le cas elle charge et se déplace aveuglément dans la direction où elle a vu le chameau pendant les 10 prochains tours (ou jusqu'à ce qu'elle l'atteigne) ;
- si pendant sa charge elle rencontre un cactus, elle est sonnée et ne fait rien pendant les 20 prochains tours ;
- après sa charge elle oublie qu'elle a vu le chameau et reprend son comportement initial.

Objectif 3. Créez un fichier `spider.ml` et qui contient la logique de contrôle d'entités qui font apparaître d'autres entités :

- les araignées se déplacent aléatoirement. Avec une probabilité de 1%, elles font apparaître un sac d'oeufs sur une case vide adjacente (si possible) ;
- le sacs d'oeufs ne se déplacent pas ; tous les 20 tours ils font apparaître une araignée sur une case vide adjacente (si possible). Après avoir essayé de faire apparaître 3 araignées, le sac d'oeufs disparaît.

Dans la suite du projet vous allez être amenés à étendre cette base de code de façon variée. Si ces extensions vous poussent à modifier de façon significative le code des objectifs de cette partie (par exemple parce que vous modifiez le moteur principal du jeu), pensez à sauvegarder une version du jeu de base propre, commentée et dont les aspects algorithmiques ont été testés. Vous pouvez faire cette sauvegarde dans un dossier spécial ou m'indiquer un commit qui correspond au jeu de base.

4 Extensions

Le jeu n'est pour le moment pas particulièrement intéressant : les entités n'interagissent pas vraiment entre elles, et le monde finit invariablement par être rempli d'araignées (n'hésitez pas à changer le comportement des entités de la Section 3 si vous le souhaitez).

Vous pouvez étendre le jeu dans différentes directions à partir d'ici. J'attends trois extensions dans cette liste. Si vous en avez fait plus, indiquez-moi dans le README les trois sur lesquelles vous voulez que je vous évalue.

Objectif entités malines. Proposez une implémentation de l'algorithme A* et des entités qui l'utilisent pour trouver leur chemin jusqu'au chameau.

Objectif entités coopératives. Implémentez des unités qui communiquent des informations entre elles pour agir de façon plus maline. Exemple de design : les jumeaux qui partagent un espace mutable où ils peuvent écrire la position d'un chameau quand ils le repèrent pour essayer de l'encercler.

Objectif actions à délai. Un système de tour plus expressif que celui actuel pourrait introduire une notion de délai : chaque fois qu'une entité met fin à son tour elle annonce un délai avant son tour suivant. Celà permet par exemple à une entité à délais courts de jouer plus souvent qu'une entité à délais longs. Celà autorise aussi plus de flexibilité dans le design des entités (par exemple les éléphants pourraient charger vite et être sonnés lentement). Modifiez le moteur de jeu pour implémenter un système de ce type.

Objectif mode sandbox. Créez un mode d'exécution alternatif du jeu où passage de chaque tour est contrôlé par un *input* clavier. Ajouter des fonctionnalités utiles au debug comme la possibilité de faire apparaître des entités, pretty-printer l'entité qui vient de jouer et l'état actuel la file d'attente.

Objectif mode script. Créez un mode d'exécution alternatif du jeu où on peut fixer la graine du générateur d'aléatoire ainsi que les mouvements successifs du chameau dans un fichier. Créez des petits niveaux scriptés pour mettre en avant le comportement des différentes entités du jeu.

Objectif système de jeu. Ajoutez un système de score au jeu. Vous pouvez par exemple déduire des points quand une entité tente d'entrer en collision avec le chameau, et au contraire récompenser un chameau qui les évite longtemps ou qui accomplit certaines actions spécifiques (vous pouvez d'ailleurs ajouter des commandes supplémentaires au chameau). On veillera à implémenter élégament ce système de score de sorte à ce qu'il puisse facilement être adapté à d'autres façons de décompter les points.

Objectif visibilité. Créez une fonction qui assigne à une entité son "champ de vision", c'est à dire quelles cases du tableau elle peut voir. Changez l'affichage du jeu pour uniquement montrer à l'écran les cases visibles par le chameau.

Objectif votre idée ici. Vous pouvez me proposer d'autres idées d'extensions jusqu'au 13 novembre à 14h. Je peux évaluer une ou plusieurs extensions personnelles si je les valide au préalable pendant le projet.