Rapport de Projet Long 2025
M1 Informatique
Université Paris Cité

Mathaui Yves Abdorohmang, Simon Pariat

March 12, 2025

1 Introduction

1.1 Objectif du projet

L'objectif du projet est de développer un jeu dans un cadre "non trivial", c'est-à-dire ici dans un langage de programmation fonctionnel.

Le joueur sera amené à contrôler un personnage et à franchir des niveaux.

On sera donc sur un jeu de plateforme inspiré de *Shovel Knight*, qui propose un gameplay fluide, avec des mécaniques classiques de plateforme et avec des graphismes en pixel art.

1.2 Présentation du jeu

Le jeu sur lequel nous nous appuierons est Shovel Knight, sorti en 2013. C'est un jeu en 2D indépendant de plateforme où l'on y incarne un chevalier, armé de sa pelle qu'il utilise à la fois pour collecter des ressources et pour se défendre des ennemis.

Figure 1: Le gameplay de Shovel Knight.

1.3 Métriques de succès

Pour décider de si le projet a été abouti, nous nous focaliserons sur les points suivants :

- Fonctionnalités de base implémentées : ennemis basiques, déplacements, sauts, attaques.
- Performance : jeu fluide (60 FPS).
- Gameplay un minimum correct : au moins un niveau jouable comportant ennemis et obstacles sur une map décorée.
- Stabilité du jeu : Les erreurs critiques seront identifiées et corrigées.

2 Implémentation

2.1 Réalisation du projet

Le projet est réalisé en très grande partie avec OCaml. Il est couplé avec Raylib[1], une librairie graphique particulièrement optimisée pour la création d'un tel jeu. C'est une librairie connue qui comporte des bindings dans une multitude de langages.

Les sprites ont été téléchargées ici, même si le site ne contient pas l'intégralité des sprites attendues.

Le versionning est géré par Git et Visual Studio Code est utilisé comme IDE.

2.2 Logiciel déjà codé

Pour le moment, nous disposons d'un jeu simple en terrain plat, contenant:

- **Gestion des niveaux** : nous avons un parser JSON permettant de pouvoir créer un niveau à partir d'un fichier.
- **Modèle physique** : il y a une simulation de la gravité, avec collisions et mouvements.
- **Entités du jeu** : divers entités comme le personnage, ennemis et objets interactifs qui sont présents.
- **Rendu graphique** : Affichage graphique du niveau souhaité.

2.3 Architecture du projet

Le projet est subdivisé en une multitude de modules dans le but d'avoir une aisance avec la maintenabilité ainsi qu'une meilleure organisation. En voici quelques uns:

[1] Pour OCaml Raylib, voir plutôt `https://github.com/tjammer/raylib-ocaml`

- **Types** : Contient tous les types propres au jeu et au gameplay: (map, entités, blocs, etc.)
- **Camera** : Contient les fonctions caractéristiques à la caméra du jeu.
- **Texture** : Gère les sprites, textures du jeu (personnages, blocs par exemple).
- **Object** : pour les objets que le personnage peut récupérer
- **Levelhandler** : pour le chargement d'un niveau depuis un fichier JSON.

2.4 Représentation des données

Dans le jeu, les données sont représentés par des types propres à OCaml.

Les types caractérisés par deux flottants (caméra, position, vitesse) sont représentés par un enregistrement (record) avec comme première composante x et deuxième composante y.

Des types algébriques sont utilisés pour les blocs, ennemis, par exemple, afin d'assurer le polymorphisme.

Une map est caractérisée par une grille de dimensions m*n. Chaque élément est un bloc avec des attributs qui lui sont associés (sprite, position, type du bloc, etc).

L'unité de mesure employé dans le cadre des coordonnées dans le plan est le bloc. En effet, le jeu original est en 25*13 blocs, et on a réussi à avoir un jeu en plein écran avec des dimensions s'en rapprochant. Cela est notamment utile car la majorité des entités (si ce n'est) ont des tailles en nombre de blocs.

2.5 Technologies utilisées

Notre code s'appuie sur les technologies suivantes:

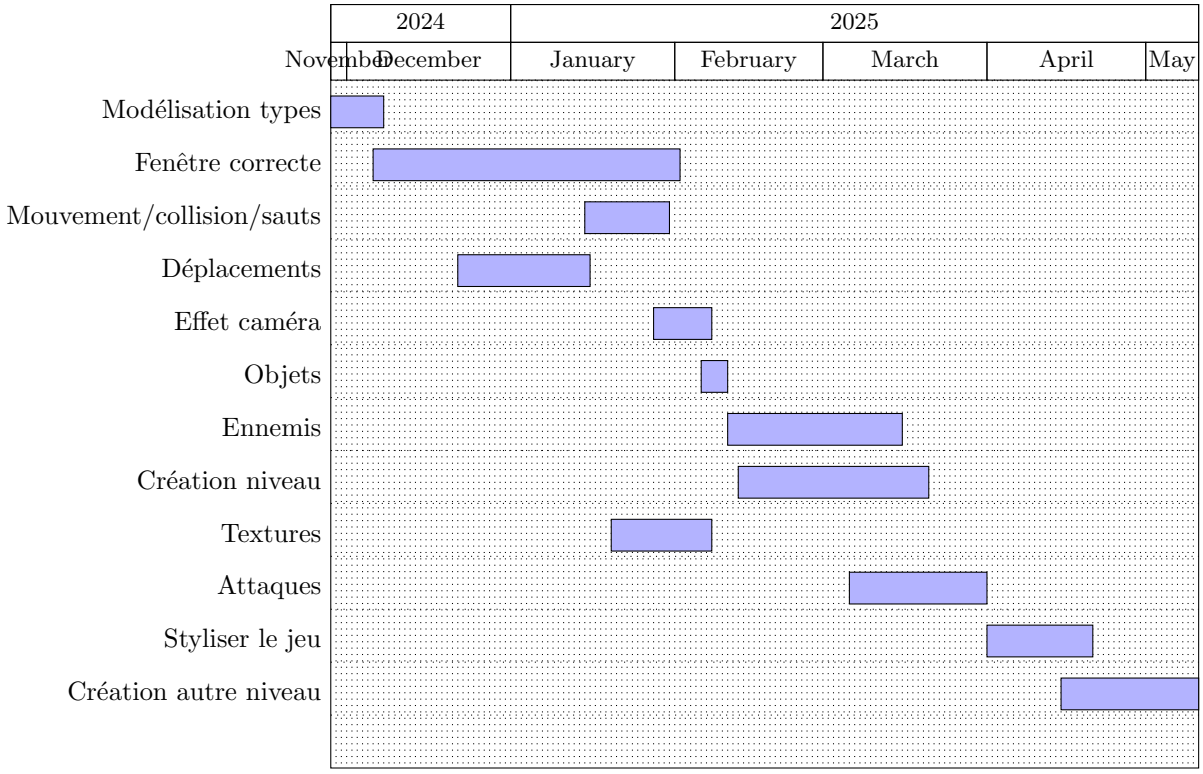
- **OCaml** : (version 5.2.1) Langage de programmation phare durant ce projet.
- **Raylib** : Bibliothèque pour le rendu graphique, événements clavier, son.
- **JSON** : Format pour la configuration des niveaux à partir d'un fichier.
- **Tiled** : Logiciel pour bénéficier d'une interface pour créer les maps.
- **Photopea** : Site pour supprimer les fonds verts des sprites.
- **Git** : Pour le versionning du projet.

3 Jalons

3.1 Tâches à réaliser

- IA des ennemis.
- Création de menus (pause, titre, game over).
- Ajout de nouveaux niveaux.
- Tests unitaires

3.2 Organisation temporelle des tâches



3.3 Tâches terminées

- Moteur de rendu et affichage des sprites.
- Gestions des événements clavier.
- Détection des collisions
- Déplacements des personnages