ALVES FABIO

AUBRY PIERRE

Projet PFA
L3 Informatique

RAPPORT - PROJET PFA

ALVES FABIO

AUBRY PIERRE

Description rapide du jeu

Forest Dash est un Geometry-Dash like, développé en OCaml en utilisant le modèle ECS (Entity Component System). Ce projet met en scène un esprit qui cherche à s'échapper d'une forêt mystérieuse remplie de pièges et d'obstacles. Dans Forest Dash, le joueur contrôle l'esprit à travers différents niveaux, en esquivant les pièges tels que les piques, les trous et les sauts mal calculés. Le jeu mise sur l'adresse et la réflexion du joueur pour réussir à traverser la forêt en toute sécurité. Le joueur peut appuyer sur la touche 'C' pour faire sauter le personnage. Nous avons implémenté un écran d'accueil avec un menu de sélection des niveaux, permettant au joueur de naviguer facilement entre les différentes étapes du jeu. Les animations du personnage principal et des décors renforcent l'immersion dans l'univers de Forest Dash. De plus, le jeu intègre des musiques et des effets sonores (sauts, atterrissage, musique d'ambiance). Le jeu comporte 2 niveaux, avec chacun 2 décors différents.

Résumé rapide des fonctionnalités :

Effets visuels

- Quand le personnage saute, le sprite fait une rotation
- Il y a des particules derrière le personnage
- La partie gauche de l'écran disparait progressivement.
- Décor parallax (Composé de 4 images, 2 paquets de 4 images par niveau)
- Le personnage a une animation de mouvement (Sprite qui change)

Ecran d'accueil

Un "faux niveau". Utilisant les fonts avec des textes qui ont des contours avec une animation sinusoïdale.

Autre

- Types de blocs
- Une barre de progression du niveau, qui prends le ratio (Fin de niveau X / position joueur X).
- Système d'audio utilisant une div html
- Optimisation du système d'affichage avec du clipping.
- Système de niveau avec son propre format de fichier.
- Les sprites ne sont pas étirés mais répétés si un bloc est plus long sur un axe que l'autre.
- Resource queue, implémenté comme en cours.

Répartition du travail

On se retrouvait chaque lundi matin entre 10h30 et 12h15 pour travailler ensemble sur le projet durant une heure de creux. L'après-midi, on continuait sur le créneau dédié. Si on n'atteignait pas nos objectifs, on finissait les tâches restantes chez nous. Même si on codait globalement ensemble en parallèle sur les mêmes fonctionnalités, on se répartissait parfois les tâches.

Pierre s'est occupé de l'implémentation des certains blocs. Il a également travaillé sur la création des niveaux et le système de rendu graphique (System_draw).

De son côté, Fabio s'est concentré sur le développement du code en général, en s'assurant que toutes les parties du projet fonctionnaient correctement et en apportant des améliorations au fur et à mesure.

Cette répartition des tâches nous a permis de travailler efficacement sur le projet et de progresser régulièrement dans le développement du jeu.

Difficultés rencontrées

Le système audio.

Il a fallu se renseigner sur une manière de le faire. L'idée de base était de regarder dans la lib jsoo et les librairies de la communauté. Il n'y en avait pas, donc on a dû passer par la création d'un nouvel élément html. Le souci était après, car en effet, il est impossible dans les navigateurs actuels de lancer un son quand on veut. Certaines conditions sont à respecter, consultables ici : https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay_guide#autoplay_availability.
Comme vous pouvez le voir, il est impossible de lancer un son sans que l'utilisateur ait interagi avec la page. L'idée était donc de lancer le son à la première interaction. Néanmoins c'était de mauvaise qualité d'attendre que le joueur fasse une entrée clavier. Donc il a fallu modifier la librairie Gfx pour ajouter les inputs de souris, et créer des éléments audios.

Système physique de base

En effet, lors de la collision avec un bloc, le joueur passait sous le sol à certains moments. Aussi, il pouvait avoir une grande impulsion vers le haut. Néanmoins, cela a été fixé d'une manière inattendue. En effet, dans le vrai jeu, le joueur meurt quand il touche frontalement le décor. En implémentant cette fonctionnalité, cela a fixé le problème qui arrivait quand justement on touchait frontalement un bloc. Cela arrivait avec le système de collision donné sur ecampus. Dû au problème de physique, des blocs ont été retirés des niveaux, mais sont toujours dans le code (Double saut, gravité inversée).

Le système de niveau

Il y a une fin de niveau, caractérisé par un bloc avec ce type spécifique. Le problème est que cela causait une dépendance cyclique. En effet :

- La collision appelle le système de niveau
- Le système de niveau appelle la création d'un bloc dans le fichier bloc.ml
- Fichier bloc.ml qui est nécessaire au système de collision car la collision a besoin des types de blocs pour fonctionner

La solution fut de mettre la fonction de set level dans le game.ml

Description du code à haut niveau

- Entities:

    - Fonction globale comportant la fonction pour créer les blocs, et une fonction pour retirer un bloc (Pour changer de niveau)
- Systems:
    - Audio : Système audio, permettant de lancer une musique, de lancer la création des éléments html audio.
    - Collision : Même que celui du cours, mais à une fonction on_collision en plus pour permettre des interactions différentes en fonction du type de bloc que le joueur à toucher. Aussi, une fonction get_side pour savoir de quel côté le joueur touche le bloc.
    - Draw: Toutes les textures, l'affichage de l'hud, affichage de l'écran d'accueil, affichage des sprites, affichage du background. Comporte un système de clipping.
    - Forces : Pareil
    - Move : Pousse le bloc vers la droite
    - Input : Permet le saut avec la touche C. Permet de changer de niveau avec les touches 0, 1, 2. Permet d'initialiser la musique après le premier clic.
- Utils:
    - Levels:
        - Système de niveau, stocke les blocs actuels du niveau.
    - Block_type: Type des blocs
    - Rect: pareil
    - Resource_queue: Permet de vérifier que toutes les resources d'une file soient chargés. Ex : Charger toutes les textures
    - Vector: Pareil
- Components_defs: Pareil, juste des types en plus tel que flying, first_collide, inverted_gravity, etc.
- Game_state: pareil, avec en plus l'état du jeu qui est soit Menu, soit Joue.
- Game.ml : Pareil, mais comporte en plus la fonction set_level, qui charge les niveaux.
- Les blocs ont différents types (Piques, Fin de niveau, Double saut, Gravité inversée, Lave, etc.). Accompagné d'une fonction lors de la collision de deux blocs, pour donner un effet personnalisé en fonction des types des blocs impliqués dans la collision.
- System_defs: Pareil.