Rapport IHM

Victorine Cassé, Margot Pellegrin, Manon Poulain

Contrôles et fonctionnalités disponibles

Contrôles

- **Déplacements gauche/droite** : joystick gauche, gamepad, flèches, QD
- **Saut / double saut** : A ou Espace
- **Sprint** : LR ou Left Control
- **Dash** : X
- **Grab** : RR ou Right Control
- **Pause** : bouton Start ou Echap

Fonctionnalités

- Inertie horizontale du joueur, la vitesse du joueur augmente jusqu'à atteindre un vitesse max et diminue jusqu'à atteindre 0, l'inertie est très faible.
- Le joueur ne peut ni sprinter ni dasher verticalement.
- Le saut est proportionnel au temps d'appui et le joueur peut réaliser un double saut.
- Le joueur peut effectuer des wall jumps en sautant sur un mur. Il peut enchaîner les jumps le long du même mur.
- Le joueur tombe moins vite lorsqu'il est en contact avec un mur.
- Le joueur utilise de l'énergie lorsqu'il sprint et dash. Le sprint utilise de l'énergie proportionnellement au temps du sprint, alors que le dash utilise une portion d'énergie fixe. L'énergie se régénère au fil du temps. Si le joueur épuise son énergie, elle se régénère plus lentement et le joueur doit attendre une régénération totale pour recommencer à utiliser de l'énergie.
- Le joueur peut mettre le jeu en pause pour revoir les contrôles, modifier les paramètres de son, recommencer le niveau, rejoindre le menu principal ou quitter le jeu.
- Le joueur meurt lorsqu'il tombe trop bas.

Environnement

- **Plateforme simple** : le joueur peut s'arrêter sur les plateformes classiques, mais ne peut pas passer au-travers.
- **Plateforme passable** (par le dessous) : le joueur peut traverser cette plateforme s'il arrive par en-dessous.
- **Plateforme tueuse** : si le joueur entre en contact avec cette plateforme, il meurt et donc il respawn au départ du niveau
- **Trampoline** : le joueur rebondit sur cette plateforme, toucher la plateforme déclenche un saut automatique. Ne prend pas en compte la vitesse du joueur et son orientation pour rebondir.
- **Liane** : le joueur peut s'accrocher à une liane et se déplacer selon la physique d'un pendule. Lorsque le joueur s'accroche à une liane, toutes les autres lianes se

réinitialisent : elles retrouvent leur état immobile. La liane reçoit une impulsion horizontale à l'appuie des déplacements horizontaux et si l'on reste appuyé, ajoute légèrement de la vitesse par frame. La vitesse perçue dépend de l'angle actuel de la liane, projection sur la tangente au mouvement de la liane.
- **Ascenseur** : l'ascenseur a un mouvement périodique de montée et descente. Il reste 2 secondes en bas et en haut de sa trajectoire. C'est au joueur de monter sur l'ascenseur au bon moment, le temps de pause entre chaque état permettant au joueur de monter ou descendre plus aisément.
- **Pente** : l'orientation du joueur suit l'inclinaison des plateformes en pente
- **Portail** : lorsque le joueur entre en contact avec un portail, la scène est déchargée et le niveau suivant est chargé. Le portail du Level 2 (dernier niveau) renvoie vers le menu principal.
- **Trous** : présents sur le tutoriel 2, ils correspondent à des endroits dans le niveau sans plateforme horizontale. Si le joueur tombe, il tombe à l'infini (ou du moins, jusqu'à ce qu'il atteigne le y de la variable "*DistanceBeforeDeath*", ce qui permet d'enclencher sa mort et de le faire respawn au début du niveau)

Feedback choisis

Feedback visuels
- Des lignes horizontales quand on sprint
- Un effet flash quand on dash

Feedback sonores
- Lorsqu'on saute
- Lorsqu'on sprinte
- Lorsqu'on dash
- Lorsqu'on meurt
- Lorsque l'on finit le jeu / le niveau
- Lorsqu'on clique sur un bouton

Valeurs exactes des réglages

Dans le script *PlayerMoves* :
- **Speed** : 4 → Vitesse de base du joueur
- **Sprint_Factor** : 3 → Facteur multipliant la vitesse pendant le sprint
- **Dash_Factor** : 4 → Pareil mais pour le dash
- **Dash_Timer** : 0.5 → Temps pendant lequel le dash s'exécute
- **Energy** : 20 → Energie du joueur
- **Energy_loss** : 0.1 → Energie que le joueur perd pendant un sprint (à chaque fixed update)
- **DistanceBeforeDeath** : -8 (tuto et tuto 2), -15 (level 1) et -30 (level 2) → Distance selon l'axe y avant que le joueur soit considéré comme "hors de l'écran" et donc mort (ce qui lui permet de re-pop au début du niveau)

Dans le script *Jump* :
- **Gravity** : 9.81 → Gravité de monde, utilisée lors de la chute

- **JumpSpeed** : 6 → Vitesse ajoutée vers le haut au moment du saut (plus elle est grande, plus le joueur saute haut)
- **TimeOfJump** : 0.5 → Durée pendant laquelle on peut appuyer de nouveau pour faire un double saut.

Dans le script *Pendule* :
- **Speed_Climb** : 2 → Vitesse à laquelle le joueur peut grimper/descendre sur la liane flèches de directions verticales.
- **Speed_Balance** : 3 → Vitesse d'impulsion horizontale maximale (dépend de l'angle) lorsqu'on appuie sur les flèches de directions horizontales.

Dans le script *Plateforme* :
- **Tetamax** : 25 (tuto 2) et 80 (level 2) → Uniquement pour la Liane ! Angle max qu'elle peut atteindre

Playtests

Nous avons réalisé 4 playtests, sur des personnes joueuses avancées et non joueuses.
Nous avons réalisé les playtests sur la version précédente à notre build final, au sein de Unity.
Deux testeurs ont dit ne pas avoir compris la différence des plateformes jaunes, et auraient aimé savoir ce qu'étaient les plateformes rouges avant de mourir dessus. Nous avons donc choisi d'ajouter les plateformes jaunes au 1er tutoriel pour que les joueurs voient son intérêt, et nous avons décidé de laisser l'effet de surprise pour les plateformes tueuses.

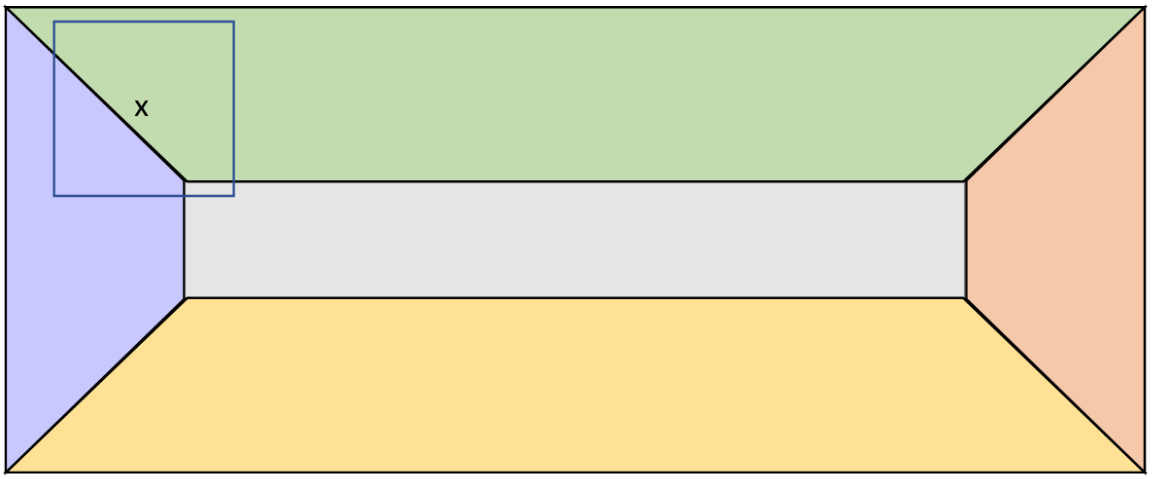
Un des testeurs n'a pas compris tout de suite ce qu'était le portail de fin de niveau (un asset en forme de cercle blanc), nous avons donc décidé de changer l'aspect du portail pour que ce soit plus clair pour les joueurs.

De nombreux bugs ont été remonté lors des tests, comme par exemple le dash qui ne fonctionnait pas si le joueur n'était pas en train de sauter, ou le jeu qui restait en pause lorsqu'on recommençait le niveau depuis le menu de Pause. Ces bugs ont ensuite été résolus, afin d'offrir une meilleure expérience aux joueurs.

Choix techniques :

Pour gérer les collisions, nous avons choisi de créer un empty object des mêmes dimensions du player qui a une frame d'avance sur le player, possible avec un découpage discret du temps. La détection de collision se fait donc sur cet empty object, et s'il est en contact avec une plateforme, le player se retrouve à la limite de la plateforme. Toutes les plateformes ont des types, et la gestion de ce que l'on fait après détection des collisions se fait en fonction du type de plateforme.
Nous avons rencontré des difficultées sur les collisions multiples, un rafraîchissement des détections avec un OnCollisionStay2D a réglé une bonne partie du problème. Ensuite une des principales difficultés, qui a beaucoup été mise à jour, c'est comment détecter si la collision se faisait par dessus, par dessous, par la droite, par la gauche. Après de

nombreuses solutions qui n'étaient pas assez précises du fait que l'empty pouvait rentrer dans la plateforme, nous avons opté pour cette solution :

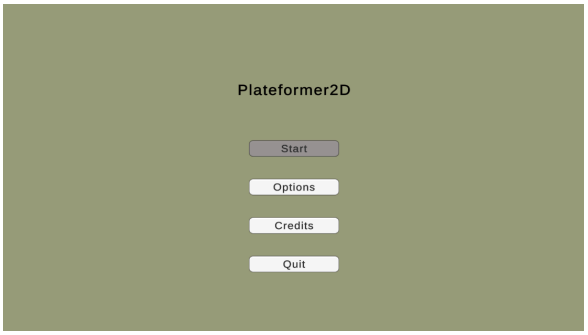


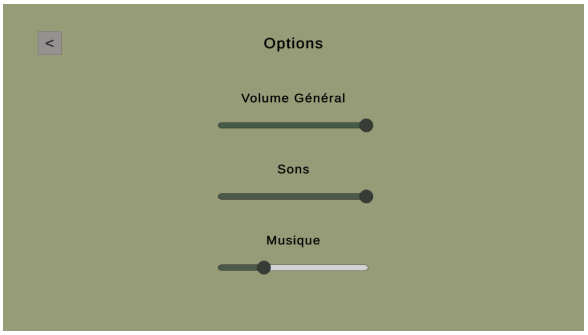
Si la position du empty object se trouve dans la zone verte, alors on est au-dessus, orange, à droite et ainsi de suite. L'implémenter n'a pas été simple mais s'est avéré efficace.
Les problèmes que nous avons rencontré avec les collisions, sont les cas de "sandwich" entre plateformes, mais ce cas là n'est pas censé arriver. Ensuite lorsque les plateformes ont des intersections, et que chacune d'entre elles dit que le player doit se trouver dans des sens opposés par rapport à elles, cela freeze le déplacement horizontal, la placement des plateformes dans les levels était important.
Un autre problème est arrivé avec les ascenseurs. En effet, les plateformes en mouvements agissent sur les mouvements du player, avec le rafraichissement de la collision, on arrive à faire en sorte que le player reste sur la plateforme, mais on voit qu'il rentre dans la plateforme. L'empty object ici ne peut pas anticiper. Nous voyons deux solutions pour régler ce problème :

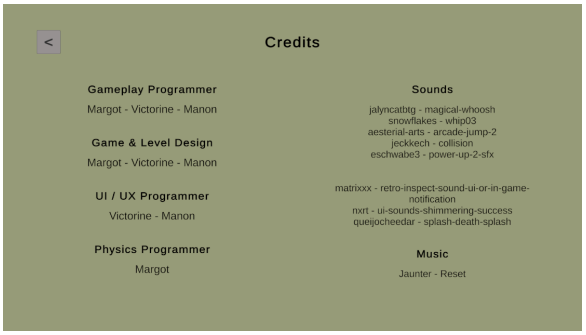
- faire le même principe que le player avec l'empty object sur l'ascenseur, et l'ascenseur empty en détectant le player, le repousserait.
- avoir accès à la vitesse de l'ascenseur et ajouter cette vitesse au player s'il est en collision.

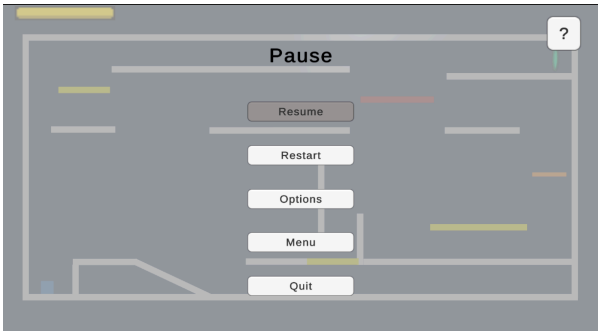
Un autre problème que nous avons rencontré est que nous pouvons passer à travers les murs avec le dash. Celui-ci augmente tellement la vitesse qu'entre 2 frames, la position de l'empty n'a pas pu détecter la collision du mur. cela pourrait être réglé avec un raycast dans la direction du dash au moment de l'appel et détecter ainsi si on doit avoir une collision. Sinon on peut le considérer comme une feature.

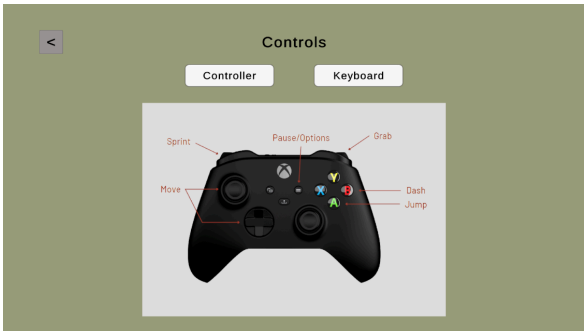
Captures d’écran

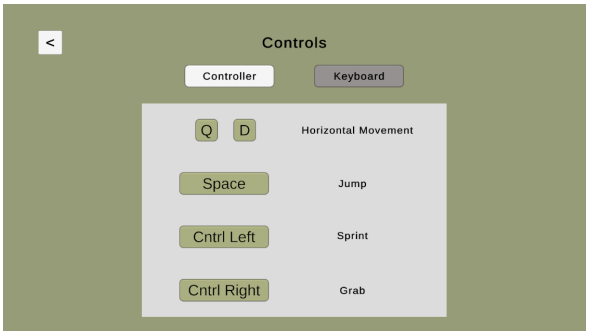


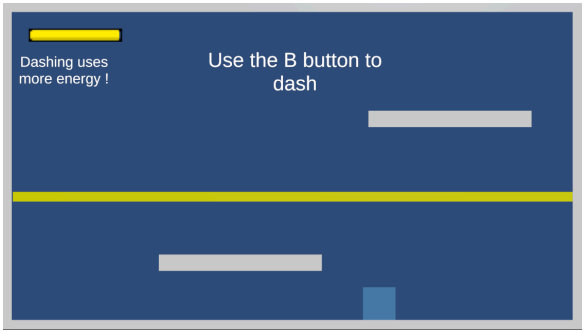


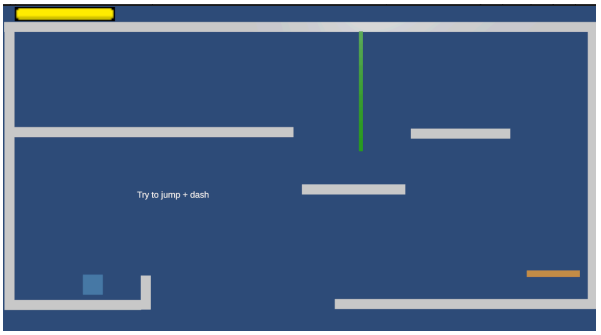




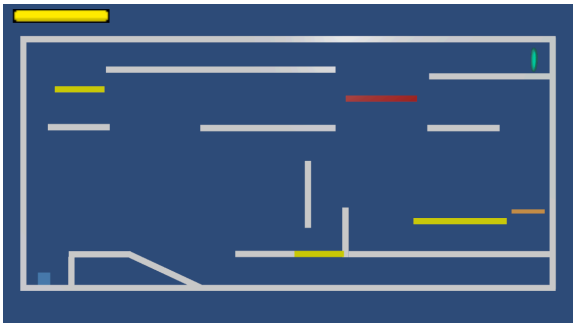


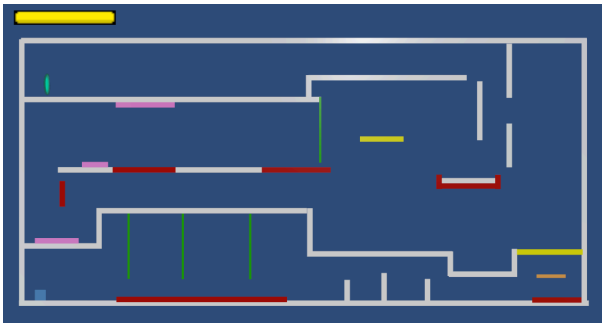






Tutoriel 1 Tutoriel 2





Level 1 Level 2