Projet Metroidvania

Combeau Alexandre
Estevan Benjamin

Manuel Utilisateur

Principe du jeu : Dans ce jeu, vous incarnez un chasseur de trésor du futur tombé dans au fond d'une faille alors qu'il cherchait un trésor d'une valeur sans pareille.Vous êtes bloqué en bas sans moyen de remonter mais votre instinct de chasseur, notamment par la présence de pièges, vous fait penser que cette zone caverneuse à été creusé par la civilisation et qu'elle contient sûrement le trésor que vous recherchez.

Comment jouer : Notre chasseur peut se déplacer de gauche à droite quand le joueur appuie sur les touches directionnelles correspondantes et sauter quand le joueur appuie sur la touche "w".
Son instinct de survie incomparable lui permet de tenter plusieurs scénarios dans lesquels il essaie d'échapper à une série de pièges et de ne garder que celui où il réussit son action.
En appuyant sur la touche directionnelle haut le joueur peut interagir avec le décors (cela correspond ici à activer un téléporteur)
Sa nature de kleptomane lui permet de récupérer les objets qui traînent par terre en toute discrétion juste en passant dessus.
Au fur et à mesures de vos péripéties, vous ferez l'acquisition de quelque accessoires qui vous faciliteront la vie:

- Le planeur qui vous permettra de réduire considérablement votre vitesse de chute et de saut, il peut être activé de façon prolongée en restant appuyée sur la touche "s"
- Le boulèminator qui vous permettra de réduire votre taille en vous comprimant dans une boule qui roule. Il peut être activé de façon prolongée en restant appuyée sur la touche directionnelle du bas. Cependant de part sa nature transformative, les autres gadgets ne peuvent pas êtres activés tant que le boulèminator est actif
- Le propulseur qui vous permettra d'avancer en ligne droite à pleine vitesse sur une courte distance. Il peut être activé en appuyant sur la touche "c" mais a besoin que notre personnage s'appuie sur une plateforme pour être rechargé.

- Le kit d'escalade du futur fera de vous un grimpeur invétéré. Le posséder vous permettra de vous appuyer contre un mur pour réaliser un saut supplémentaire dans la direction opposée au mur. Pour cela il suffit juste que le joueur appuie sur la touche de saut après s'être collé à un mur et la touche directionnelle correspondant à la position du mur par rapport à lui-même.

Ordre des niveaux : Depuis le hub central, le joueur aura accès à 5 niveaux différents qui seront pour certains eux mêmes découpés en plusieurs niveaux. Le joueur est libre de tenter les niveaux auxquels il arrive à accéder dans l'ordre qu'il veut. Cependant, si un niveau semble trop dur à finir pour lui, il pourra retourner au hub et tenter un autre niveau afin de récupérer le gadget qu'il renferme et d'avancer dans sa quête. Le jeu se termine dès lors que notre héros arrive à mettre la main sur le magot qu'il recherche

Développement :

Répartition des tâches :

- Alexandre Combeau : mouvements, level design et graphismes.
- Benjamin Estevan : architecture complète (chargements des objets), gestion des objets et leurs comportements.

Model originel,
Nous avons repris les briques de base données avec le tp 7. Ce faisant, les dossiers lib et prog étant primordial au bon fonctionnement du modèle et nous n'y avons donc pas touché.

Ressources, contient tous les graphismes du jeu ainsi que les niveaux.
Ceux-ci sont constitués d'un fichier txt, que l'on va parser, pour chaque type d'objet différents. C'est-à-dire, les portes, les pièges, les murs, la taille …
Chacun de ses fichier contient une ligne par objet de ce type,

par exemple :
un bout du fichier resources/level1/wall.txt
0.0,0.0,8000,50,large_floor_3
0.0,550.0,8000,50,large_floor_3
0.0,0.0,50,275,large_wall_3
7950.0,0.0,50,550,large_wall_3

On a les coordonnées x y puis largeur et hauteur et enfin l'image qui va lui être appliqué. Nous appliquons ce procédé pour coder nos niveaux dans des fichiers txt. Chaque niveau ayant son propre répertoire.

Les images/textures quant à elles sont triés, un dossier pour le personnage et l'autre pour le monde (murs, plateformes, pièges etc).

Src, dossier contenant l'intégralité des composants actifs du jeu, a la racine.

Component, nous avons ajouté des composants permettant de contrôler les mouvements du joueurs, ce sont des sortes de verrou bloquant des actions nécessitant un objet spécifique. D'autres servent simplement à catégoriser des principes essentiels, comme la destination d'une porte ou la vie du joueur.

Io, tout comme lib et prog, ce dossier gérant les input/output du joueur n'a pas été touché.

System, c'est draw_system et surtout collision_system qui a été grandement complété dans notre cas. Le contrôle des connexions entre portes/téléporteurs est effectué dans collision_system, chaque porte contient sa sortie associée, nous devons juste tester la collision d'un joueur et d'une porte pour lui mettre à jour sa position (et charger le niveau correspondant).

Un fichier pour chaque entité différente de notre jeu, player, trap, item, door etc
Chacune de ces entités sont enregistrées dans un ensemble de systèmes qui varie selon leur comportement souhaité.

Un module important de notre jeu est level.ml, il nous permet de charger et décharger des niveaux, respectivement en parsant les fichiers du niveau correspondants et vidant ces objets de toutes les listes du modèle ECS afin de faire de la place pour le prochain niveau. Nous ne chargeons donc pas l'intégralité des niveaux en même temps, mais seulement le niveau actuel.
Il est également intéressant de noter que dans l'état, le module Gfx ne nous permet pas d'afficher d'images sur du vide, pour afficher un écran de fin ainsi que les background de nos niveaux nous ajoutons une boîte de la taille de notre écran ou niveau et appliquons une texture sur elle. L'ordre affichage dans la boucle principale ainsi que notre boucle chargeant les niveaux sont primordiales, car le premier élément dessiné sera toujours au premier plan. Le background devra alors être affiché en dernier et le joueur en premier (sauf pour les objets transparents).

Difficultés rencontrées :

Nous voulions originellement faire un metroidvania avec des ennemies et un vaste monde, mais le temps nous a manqué pour arriver à rendre intéressant les ennemis. Il en est de même pour des caisses déplaçables et autres objets ayant différentes propriétés. Nous avions des prototypes fonctionnels mais nous n'avons pas réussi à les rendre amusants, ils n'étaient pas facile de les déplacer avec précisions n'y de d'en utiliser plusieurs à la suite rapidement. C'était une mécanique lente (comme les ennemies) et seulement interactive avec le joueur, ce qui n'allait pas dans le même sens que notre vision pour le jeu (que nous voulions rapide et réactif). Les ennemies nécessitait également le développement d'une micro intelligence, ce mouvement automatique s'est avéré trop complexe et nécessitant un changement de notre architecture. En effet pour avoir accès aux objets ennemis et ainsi pouvoir modifier leur vélocité par exemple nous devions parcourir la liste d'objets pour chaque ennemi, ce qui avec notre modèle allait augmenter grandement la complexité.

Dès le commencement du développement du chargement de niveau et l'ajout de nouveaux déplacements pour le joueur nous avons fait face à plusieurs problèmes. Le premier étant que la majorité de notre code se passait dans collision_system, cette solution n'étant pas maintenable, nous avons tenté de déplacer notre code vers un module pour le chargement de niveau et dans le joueur. Mais cette solution temporaire nous a bloqué un certain temps car une erreur de dépendance cyclique est apparue. Nos fichiers appelaient

collision_system et vice versa. Pour régler ce soucis nous avons créer des component pour chacuns des comportements dont nous avions besoin, collision_system les modifiaient et nos modules peuvent ensuite vérifier leur états. Nos modules et collision_system ne s'appelaient plus l'un l'autre, ce qui en plus de résoudre la dépendance cyclique, reduit considerablement le code présent, et sa complexité dans collision_system.

Le chargement des textures nous a également posé de nombreux soucis, il est fréquent que le personne (avec texture) ne spawn pas. En effet nous n'avons pas réussi à faire fonctionner le parallax scrolling et plus simplement des méthodes permettant d'économiser de la place sur l'image de fond et ne pas avoir à charger une image de taille importante mais également pour avoir un fond de taille fixe se répétant et non distordu.

Le chargement de texture fonctionne correctement mais il pose de nouveaux problèmes, la hitbox du personnage n'est pas en complet accord avec sa texture. Cette différence n'est pas problématique lors des premiers niveaux. Cependant lors des derniers niveaux, une connaissance précise de la hitbox de notre personnage est nécessaire pour une jouabilité correcte. Comme dit précédemment les textures du background ainsi que murs, animation des pièges, joueurs, téléporteurs créent des ralentissement qui vont casser le moteur physique, avoir des fps stables est primordial pour une expérience agréable surtout avec un jeu rapide et précis comme celui-ci. Or le temps de chargement des images était tellement long que le joueur n'apparaissait pas au bon endroit ou pas du tout.

Pour construire intelligemment les niveaux, nous nous sommes inspirés de salles présentes dans le jeu "Super Metroid" et avons ainsi tenté de faire un agencement similaire de certaines plateformes en calquant au brouillon pour que les dimensions correspondent avec celles de notre personnage (voir exemple ci-dessous)

```
level_2 : 1600 600
**wall**
0.0, 340.0, 100, 30,large_floor_3
280.0, 390.0, 40, 20,large_floor_3
440.0, 350, 20, 20,wall_1
580.0, 430, 20, 20,wall_1
740.0, 300, 20, 20,wall_1
1010.0, 430, 20, 20,wall_1
1145.0, 300, 40, 20, wall_1
1330.0, 320, 20, 20,wall_1
1500.0, 390, 100, 30,large_floor_3
0.0, 0.0, 1600, 40,mur
0.0, 0.0, 70, 170,mur
1530.0, 0.0, 70, 170,mur
**portes**
0.0, 170.0, 40, 160,level_1
1580.0, 170.0, 40, 160,salle_planneur

**pieges**
0.0, 450.0, 1600, 150,lave
```