La fabrication desportes.
Un jeu 2Den SDL.

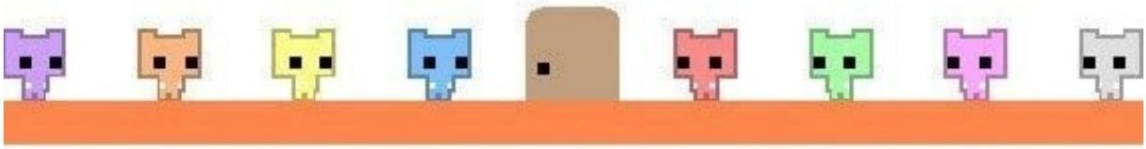
Par OHMAD Abdessamade

Résumé:

- Présentation.
- Développement et vitrine.•
Problèmes rencontrés.

Introduction:

En tant que projet dans notre classe C++, nous avons été chargés de créer un jeu 2D de type puzzle Platformer similaire à PICO PARK, j'ai donc créé ce jeu en utilisant SDL et en suivant le paradigme de la programmation orientée objet.



SDL : Simple DirectMedia Library, est une bibliothèque de développement multiplateforme conçue pour fournir un accès de bas niveau.

Pour créer ce jeu, j'ai dû tout créer à partir de zéro, y compris toutes les classes utilisées par le jeu, mais cela en valait la peine.

Développement & Vitrine :

La première étape que j'ai franchie dans le développement a été de créer un référentiel dans GitHub afin de rendre le flux de travail plus fluide. Après cela, il a initialisé un référentiel git local et l'a connecté à la branche distante.

J'ai fait un Makefile pour faciliter la compilation, et gagner du temps.

Le premier commit était la classe Game, qui lance SDL et la fenêtre, exécute la boucle Game principale, un moteur de rendu statique.
Le rôle du moteur de rendu est de rendre tous les objets du jeu à l'écran, nous l'avons rendu statique pour l'utiliser dans toutes les scènes.

La boucle de jeu s'exécute tous les 60 images par seconde que nous avons plafonnées pour le contrôle, chaque itération de la boucle de jeu est une image, le temps entre chaque image est appelé deltaTime.

Pendant chaque image, nous devons mettre à jour et rendre nos objets de jeu, et gérer les événements définis par le clavier ou la souris.

Scenes :

Afin d'avoir des niveaux et des menus, je devais créer des scènes j'ai donc créé une classe de scène qui s'occupe de l'image de fond, et deux sous classes : MenuScene et LevelScene.

La scène du menu principal :

Chaque scène s'exécute dans sa propre boucle, le tout à l'intérieur de la boucle principale du jeu, donc si nous cliquons sur Démarrer, la scène en cours s'arrête et la scène du menu Niveaux décolle.

La scène du menu Niveaux :

Comme nous pouvons le voir, les deux scènes utilisent des boutons pour contrôler les menus.

Tous les boutons sont de la classe de type MenuItem, qui hérite de la classe GameObject.

Les scènes sélectionnées gèrent les événements séparément. Par exemple, lorsqu'une souris survole le bouton, il change de couleur.

Cela se fait en vérifiant la position de la souris et en la comparant à la position de chaque élément du menu, puis si ce dernier est cliqué, l'événement approprié a lieu.

Les éléments de menu utilisent SDL_ttf.h qui est une bibliothèque SDL pour gère les polices, tandis que SDL_image.h gère toutes les images.

Les scènesde niveaux :

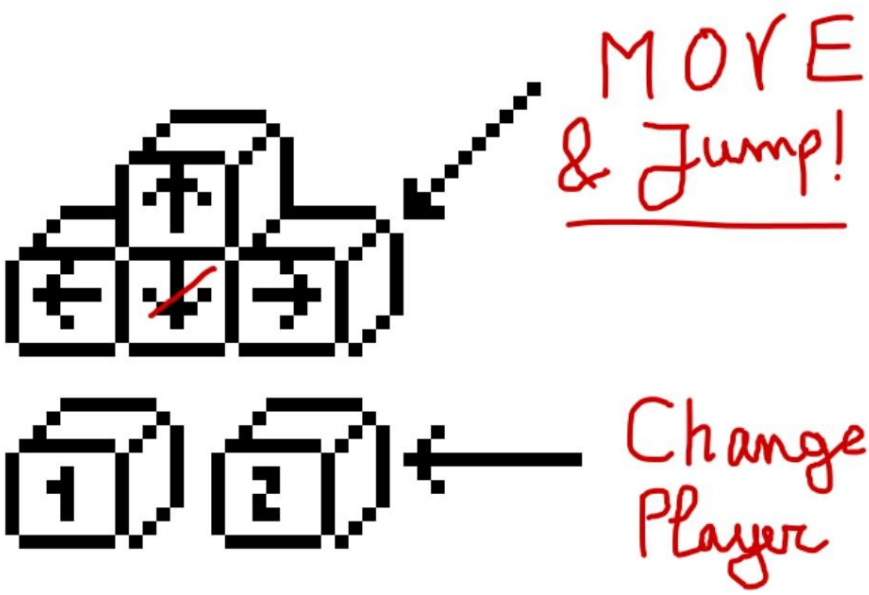
Les scènesde niveauutilisent la classeLevelScene , qui gèretous les différents événements complexesqui ont lieudans chaque niveau (niveau1, 2, 3).

Chaque niveauque nous avonsest uneinstance de la classeLevelScene , nous créons tousles objetsde jeu quenous voulons et nousles passons à la scène, avec des méthodes setter .

L'ensemble du système de classesoffre une bonne couche d'abstraction, ce qui rend le développement plusfluide.

L'idée du jeu est de contrôler tourà tourdeux personnages pourrésoudre l'énigme et passer par lesportails (portes).
Dans chaque niveau,la clé doitêtre récupéréepour pouvoir utiliser le portail.

Les contrôles:



Les

le jeu acceptégalement W,A, D (ouZ, Q, D azerty) pourle mouvement.
Passons au premier niveau :

Niveau 1:

Comme nouspouvons le voir, il y a plusieursobjets dejeu, des platesformes aux boules de feu.Il y a deux boutons, le bouton retour est cliquable, l'autre nous montre quelpersonnage estsélectionné (bleu ou rouge).

Les personnages ont leur propre classe qui hérite également de GameObject, la classe Player a une méthode de mise à jour plus complexe avec la plupart des méthodes, qui remplacent les méthodes dela classe parent.

Dans le développement,la première étapeconsistait à faire bouger les personnages (répondre aux touches fléchées).
Je les ai ensuite fait tester àl'intérieur del'écran dejeu avecla vérification des limites quiest la premièreforme decollision quej'utilise.

Ajout de la collision de base AABB (Axis Aligned Bounding Box), qui est l'algorithme le plus simple et le plus rapide de tous les types de collision.

Ensuite, il fallait ajouter la gravité et les mécanismes de saut.

Aucune de ces tâches n'était facile et chacune prenait beaucoup de temps.

Ensuite, il fallait ajouter la clé, les pointes, les boules de feu et le portail.

Ajout d'une détection de collision pour vérifier si le joueur a obtenu la clé (ou touché une pointe).

J'ai fait entrer en collision les deux personnages (bleu et rouge), afin que le joueur puisse utiliser cette collision pour sauter vers des terrains plus élevés.
→ En fait, c'est la clé pour gagner le premier niveau.

Niveau 2 :

Dans ce niveau, nous ajoutons deux nouveaux objets ; Le bouton et le Fire Wall.
Ce qui, combiné, fait une nouvelle mécanique intéressante.

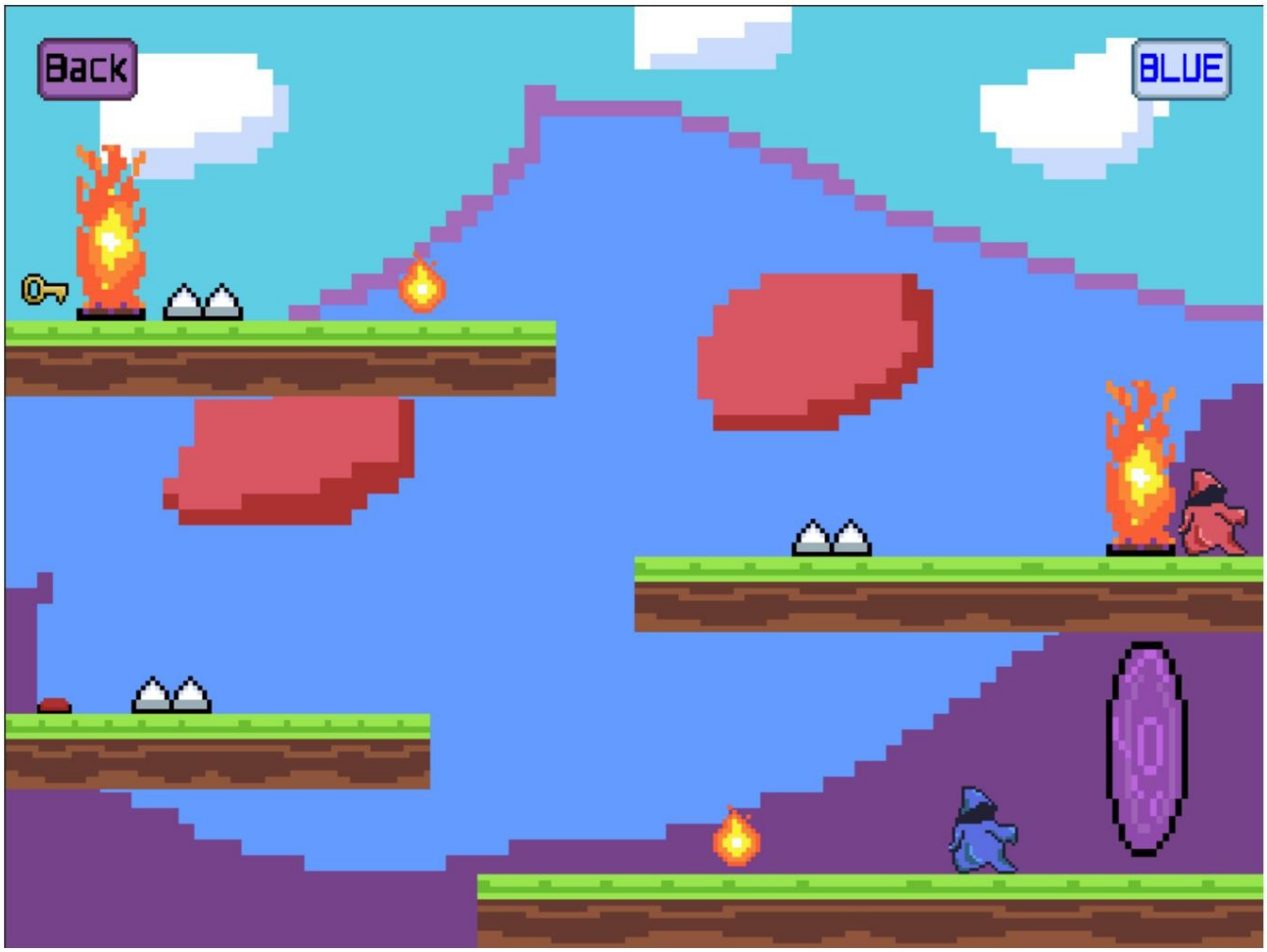
Afin d'obtenir la clé, le bouton doit être appuyé par l'un ou l'autre des joueurs. une fois que cet événement a eu lieu, le parefeu n'est plus rendu.

En utilisant le personnage bleu, le joueur doit maintenir le bouton enfoncé, tandis que le personnage rouge récupère la clé en toute sécurité. Une fois la clé obtenue, nous recevons une notification nous indiquant que nous avons la clé et que nous pouvons donc maintenant passer la porte.

Comme nous pouvons le voir, le joueur bleu désactive le parefeu en appuyant sur le bouton.

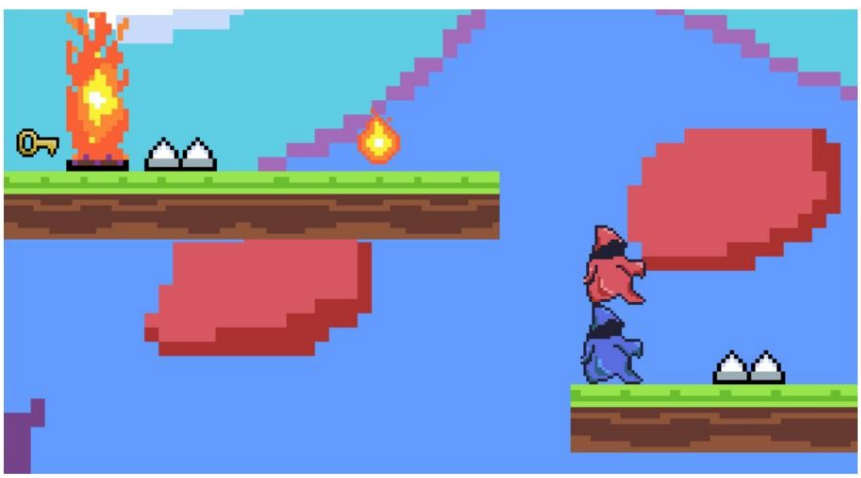
Niveau 3:

Le troisième et dernier niveau est celui qui fusionne à la fois la mécanique d'avoir à utiliser les deux personnages pour sauter et les parefeu.



Le personnage rouge est emprisonné !

Dans ceniveau, le personnagebleu commence seul etdoit sauverrouge. Une fois quele joueur utilise Bleu pourappuyer sur lebouton et libérer Rouge, le saut vers latouche est trop long, lesdeux personnages doivent donc être utilisés.



Après cela, ilfaut appuyerà nouveau sur lebouton pour libérer la clé.

En cas deperte ou de succès, le LevelScene affiche une notification.

Après quelquessecondes (compteur) Lascène estsoit réinitialisée soit à gauche.
En casde défaite, la scène suivante est le niveau actuel, et en cas de victoire, la scène suivante est le menu du niveau.

Le systèmede classe :

Scène > MenuScene, LevelScene.
GameObject > MenuItem, Joueur.

La classe Scene gère les problèmes généraux, tels que le chargement de la feuille de texture (variable statique), l'image d'arrièreplan et le booléen en cours d'exécution.

La classe GameObject a deux variables SDL_FRect qui sont utilisées pourdessiner la texture.

*MenuScene : a un vecteur de MenuItems, gère tous les différents choix.
*LevelScene : vecteur du joueur, vecteurdes élémentsde menu et des différents objets du jeu. (platesformes, pointes, etc.)

*Joueur : la classe de joueurs dispose de diversesméthodes pour gérer les collisions, la gravité et les mouvements.
*Élément de menu : a unenouvelle texture et une surface temporaire pour initier le texte.

Problèmes rencontrés :

J'ai rencontré de nombreux revers lors du développement du jeu, en grande partie lié au fait que je suis nouveau sur SDL et que tout doit être programmé à partir de zéro, donc je vais les ignorer.

Mouvement brutal :

Le premier problème de Face était que le mouvement du joueur était saccadé et rugueux, je pensais que cela avait quelque chose à voir avec le deltaTime.

```
deltaTime =(float)(SDL_GetTicks() - frameStart)/1000 ;
```

Après avoir sorti sa valeur, j'ai remarqué qu'elle était effectivement correcte (0,0166 pour 60 FPS).

Donc, après avoir éliminé deltaTime comme source du problème, j'ai remarqué que le lecteur utilise SDL_Rect de la classe GameObject.

Qui est une structure composée de 4 valeurs entières.

```
SDL_Rect = {int x, int y,int largeur, int hauteur }
```

Je regarde dans le Wiki SDL et j'ai trouvé un SDL_FRect qui remplace les valeurs entières par des valeurs flottantes.

Alors que c'était effectivement la solution, j'avais toujours des mouvements saccadés parce que j'ai géré les événements et mis à jour la nouvelle position dans la même fonction. GérerEvents().
J'ai mis à jour la position du joueur dans une fonction distincte qui s'est assurée qu'elle était mise à jour à chaque image. => solutions.

Collision : Il

y avait de nombreux défis à relever à ce sujet.

J'ai utilisé la collision AABB pour toutes les méthodes, et cela fonctionne.
La collision était difficile à mettre en œuvre car la fonction devait savoir où se dirigeait le joueur. Afin de le remettre.
J'ai aussi eu un problème avec différentes tailles (par exemple vérification de la clé)

j'ai donc ajouté une méthode collisionDetection et une méthode FullCollision.

Le premier est utilisé pour vérifier la collision avec tous les éléments et renvoie vrai lorsque la condition est remplie.

FullCollision est celui qui fait que le personnage entre en collision et ne traverse pas les éléments, de n'importe quel côté.

Saut et Gravité :

Il y avait beaucoup de problèmes rencontrés pour atteindre la gravité et sauter. Le principal était de faire sauter le joueur et de ne pas changer de direction dans les airs.

J'ai ajouté une nouvelle méthode de collision, VerticalCollision. Qui vérifie si le joueur a touché le sol.

J'ai fait en sorte que la vitesse x du joueur soit égale à 0 lorsqu'il est en vol, mais cela ne pouvait pas fonctionner car cela faisait sauter le joueur uniquement verticalement.

La solution consistait à faire bouger le joueur horizontalement uniquement lorsque ce n'est pas Airborne, et pour enregistrer sa vitesse avant de sauter : beforeJumpVelocity, qui est utilisé pour le mouvement dans les airs uniquement.

Le problème suivant était que le joueur planait après avoir touché le sol (puisqu'il n'était pas possible de changer la valeur x), j'ai fait en sorte que le joueur arrête sa vitesse x après l'impact. BeforeJumpVelocity est défini sur zéro.

Une feuille de texture :

Au début, je chargeais différentes images pour chaque texture, mais je les ai toutes déplacées vers une feuille de texture (texture statique dans la classe Scene) pour optimiser les performances et réduire l'utilisation de la mémoire par le GPU.

Ps: tout l'art a été fait par moi en utilisant Aseprite.

Le jeu est monoPlayer, avec deux personnages.