Clément Lion
Module “Construire un Snake avec Raylib”
Présentation du jeu.

Comment jouer ?

Dans ce jeu, vous incarnez un ou plusieurs Snake, votre but est de manger le plus de pommes possibles. Afin de pouvoir controler tous les serpents, vous allez devoir placer sur leur chemin des **Blocs de direction**. Un snake qui marche sur un bloc de direction va adopter la direction pointée par le bloc.

Le joueur choisit la direction du bloc à placer avec les **flèches directionnelles** et place un bloc en utilisant le **clic gauche de la souris**. Si vous voulez ralentir le jeu afin de réfléchir, appuyez sur la **barre d'espace pour mettre le jeu en pause**. Les snakes ne bougent plus en pause, mais vous pouvez mettre des blocs de direction.

Le jeu ne possède **pas de condition de victoire (c'est un jeu infini)**. Le joueur perd la partie si les snake rentrent en collision avec eux-mêmes ou l'un avec l'autre. Il y a aussi un **timer** de 10s. Si les snake n'ont pas mangé toutes les pommes à l'écran à l'issue du timer, le joueur perd la partie. Le timer se réinitialise à chaque fois que les snake mangent toutes les pommes à l'écran.

Afin de familiariser avec les commandes, il est recommandé de faire le niveau tutoriel avant de se lancer dans le jeu.

Originalité

L’originalité du jeu réside dans **la manière dont Snake se contrôle**. En effet, plutôt que d’agir par réflexe, le joueur est invité à prévoir à l’avance tous les mouvements de snake à la manière d’un jeu tactique. Une autre particularité est aussi le fait que l**e joueur ne contrôle pas seulement un seul snake mais deux**. Il doit faire en sorte de pouvoir les diriger chacun vers les pommes en faisant en sorte qu’ils s’esquivent l’un l’autre.

Le but était d’avoir un mélange entre snake et un jeu de stratégie.

Le code des blocs de Direction est disponible dans le dossier src/Entities/DirectionBlock. Un DirectionBlock contient une position et une direction (celle dans laquelle snake ira en marchant dessus). Le joueur (PlayerHandler disponible dans le dossier src/Services/PlayerHandler) se voit associer une queue possédant tous les blocs de direction qu’il peut placer sur le terrain. A chaque fois que le joueur clique sur une case de terrain, on récupère le prochain élément de la queue des blocs de direction pour le placer sur la grille. Le bloc de direction regagne la queue lorsqu’il rentre en collision avec un autre objet.

Afin de pouvoir gérer la création et la mise à jour de tous les objets, il a été nécessaire de centraliser les mouvements dans un service appelé EntityHandler. Chaque entité se déclare auprès de l’entityHandler au moment de sa création et se voit attribuer un ID. Cet ID lui permet d’indiquer à la grille quelle case

est occupée par l’entité. Il permet aussi à l’EntityHandler de résoudre les collisions en utilisant pour cela un dictionnaire qui permet de retrouver les entités concernées par ID.

Les transitions entre les menus et les niveaux sont gérées au niveau du dossier src/Scenes. Plus de détails sur cette partie là sont disponibles dans la partie structure du code.

Structure du code

Vous trouverez ci-dessous une description des différents dossiers du code :
- src/Entities : Contient toutes les entités du jeu. Elles héritent tous d'une classe abstraite entité. Les entitiés principales sont les suivantes:
  - Snake -> Contient une queue de cellules. Ces cellules permettent d'indiquer les cellules dans la grille occupées par snake.
  - Apple -> Collectible que Snake doit récupérer.
  - DirectionBlock -> Bloc que le joueur peut poser sur le terrain. Il possède une cellule indiquant la direction dans laquelle snake doit tourner quand il marche dessus et la position sur la grille du bloc.
- src/GridElements : Grille et cellules. Les cellules sont les cases de la grille et possèdent leurs propres opérateurs afin de pouvoir gérer les déplacements d'une cellule à une autre.
- src/Input : Ensemble de classes pour gérer les inputs utilisateurs. Pour l'instant, les contrôles utilisent la souris et le clavier. On aimerait bien pouvoir inclure un mode pour que le joueur puisse utiliser une manette.
- src/Scenes : Contient les niveaux et menus. Tous les objets sont des enfants de Scene. Afin de simplifier la construction, nous avons crée un objet abstrait level et un objet abstrait menu qui forme un "niveau générique" et un "menu générique". Les autres menus et niveaux héritent de ces classes abstraites.
- src/Services : Contient les services. Un service est un objet global dont on ne crée qu'une seule instance. Il est possible de trouver le service que l'on cherche grâce à notre Service Locator.
- src/Utils : Contient des objets génériques sont utiles en toutes circonstances (exemple : timer ou encore classe pour désigner des formes géométriques).

Algorithme de queue : les queue sont utilisées à deux reprises dans le code. Une première fois pour gérer les positions du serpent (à chaque déplacement, on enlève le dernier élément de la queue et on en rajoute un correspondant à la nouvelle case occupée par la tête de Snake. On utilise aussi des queue pour gérer la réserves de blocs de direction du joueur (on enlève un élément de la queue quand le joueur place un bloc. Tout bloc détruit regagne la queue des blocs de direction disponibles).

Algorithmes de grille : c’est un tableau à deux dimensions qui stocke la position de toutes les entités sur la grille. La grille s’appuie sur les cellules afin de pouvoir indiquer les positions des éléments et faire des opérations. Lors des mouvements des entités, la grille stocke toutes les nouvelles positions “souhaitées” par les entités et évalue si les mouvements sont possibles. Dans les cas où une entité est déjà dans une case, on appelle l’EntityHandler pour évaluer la collision qui a lieu.

Service Locator : Certains éléments du code sont globaux et il serait compliqué d’essayer d’associer la même instance du service à tous les objets du code. Le service locator permet de régler ce problème. Il stocke un pointeur sur l’instance des services et toutes entités ayant besoin de faire appel à un service peut y accéder grâce au service locator. Par exemple, quand la grille observe qu’une collision va avoir

lieu, elle demande au service locator de lui donner l'addresse de l'EntityHandler qui sait comment gérer les collisions.

Editeur de niveau : Les niveaux et les menus sont tous des enfants des "scenes". Ils possèdent aussi leurs classes abstraites level et Menu. Ces classes abstraites embarquent un certain nombre de méthodes afin de pouvoir simplifier la création de nouveaux niveaux.

Format de données : La différence entre un niveau et un autre provient du nombre de snake à gérer, de la taille de la grille dans laquelle on se déplace ou encore du nombre de pommes à manger. Ces informations se rentrent dans le constructeur du niveau.

Passez une excellente journée.