Apolline Gruaux

Benjamin Lanuza

The Smoke Snake

EISE 4

Projet en autonomie réalisé dans le cadre du cours de Langage Orientée Objet

Année 2019-2020

Table des matières

I- Règles du jeu

Le projet que nous avons développé reprend les codes d'un célèbre jeu sur mobile : le snake. Nous avons adapté ce jeu afin d'offrir différents modes de jeu et de convenir au sujet (SMOKE).

Lors du démarrage, plusieurs choix de jeux sont présentés :

I-1. Le jeu simple

Ce jeu vous offre un terrain de jeu où votre serpent peut évoluer librement. 3 pastilles sont disponibles sur ce terrain. La première est une pastille faisant grandir le serpent. Plus le serpent est grand, plus le nombre de points augmentent. La seconde pastille est une pastille vortex. Lorsque le serpent passe dessus, il est téléporté à un endroit aléatoire du terrain de jeu. La dernière pastille est la partie liée au sujet "SMOKE". En effet, lorsque le serpent passe sur cette pastille, les murs bordant le terrain deviennent de la fumée et donc traversables par le serpent. Ces deux dernières pastilles sont très utiles afin d'avancer le plus loin dans le jeu, surtout quand le serpent est grand et où il est difficile de ne pas perdre.

Lorsque le serpent se mord une partie du corps, la partie est terminée.

I-2. Le jeu complexe

Ce jeu a exactement le même principe que le jeu simple, à l'exception que sur le terrain de jeu des murs sont présents, rendant le jeu plus compliqué.

Au démarrage du jeu, ces murs ont le même aspect et le serpent ne peut que les contourner. Ce mode de jeu comprend les mêmes pastilles que le jeu simple. Cependant, lorsque le serpent passe sur une pastille vortex, certains murs du terrain changent d'apparence. Ainsi, le serpent peut traverser un de ces murs et être ainsi téléporté à un endroit aléatoire du jeu. Lorsque la pastille traversée est une pastille SMOKE, d'autres murs changent d'apparence et deviennent plus transparent et donc de la fumée. Ainsi, il est possible pour le serpent de traverser ces murs.

Ce mode de jeu dispose de 3 niveaux : easy, medium, hard. Ces niveaux définissent la probabilité croissante d'obtenir dans le jeu des murs. Cependant dans ces murs, les niveaux définissent la probabilité décroissante d'avoir dans la partie des murs de type SMOKE et VORTEX (téléportation).

Comme pour le jeu simple, la partie s'achève quand le serpent mord une partie de son corps.

II- Développement du jeu

II-1. Les Éléments

Comme l'introduction laisse à penser, le jeu est constitué de différents éléments. Chacun de ces éléments doit avoir 3 propriétés, *une position horizontale, une position verticale, ainsi qu'un indicateur de capacité à être traversé ou non.*
Nous avons donc besoin d'une classe abstraite Element dont héritera tous les éléments du jeu.

II-2. Les Chemins & Le Snake

Dans le jeu nous avons tout d'abord besoin de chemins, où le serpent pourra se déplacer. Ainsi, nous avons développé une classe Chemin, fille de la classe Element qui est traversable. Ensuite, nous avons le serpent. Le serpent est une entité qui est amenée à grandir. Ainsi, une partie du corps du serpent correspond à la classe Body, héritant aussi de la classe Element, mais qui ne peut être traversable (une partie du serpent ne peut pas traverser une autre partie de lui-même). Le serpent doit avoir une tête qui est une partie du corps. Ainsi, nous avons implémenté une classe Tete héritant de la classe Body.

II-3. Les Murs

Les murs du jeu sont aussi des éléments. Ainsi, la classe Mur hérite de la classe Element et n'est pas traversable (le serpent ne peut traverser un mur). Comme présenté dans l'introduction, il existe différents types de murs : les murs normaux (classe Mur présentée précédemment), les murs SMOKE ayant la faculté dans certains cas de devenir de la fumée et traversable par le serpent, et les murs VORTEX ayant les même spécificités qu'un mur SMOKE à la différence qu'en plus d'être traversable, ils téléportent le serpent sur un chemin aléatoire du terrain. Ainsi en suivant la description précédente, on remarque que la classe représentant les murs VORTEX hérite de celle représentant les murs SMOKE qui hérite elle-même de la classe Mur. Les 2 dernières classes présentées disposent chacune d'une fonction supplémentaire permettant de devenir traversable ou non (en fonction du déroulement du jeu et des pastilles sur lesquelles le serpent passe).

II-4. Les Pastilles

Le dernier élément à aborder sont les pastilles.Celles-ci sont un élément cruciales du jeu. Elles permettent d'évoluer dans le jeu.

Il existe 3 pastilles différentes (héritant donc de la classe abstraite Pastille qui hérite de la classe Element). La première pastille EtablePastille permet au serpent de grandir (ajout d'un Body au serpent) lorsqu'il passe dessus. La deuxième permet d'activer le comportement SMOKE du jeu et la dernière permet d'activer la fonction VORTEX du jeu. Comme introduit précédemment, ces 2 dernières pastilles ont un rôle différent suivant le mode de jeu choisi (simple ou complexe). En effet, pour le mode complexe, lorsque le serpent passe sur une de ces 2 pastilles, une fonction propre à chacune des 2 classes est appelées. Cette fonction, la fonction activate, active tous les murs du jeu associé à la pastille traversée (SMOKE ou VORTEX). Changeant ainsi l'affichage des murs activés, ainsi que leur capacité à être traversés.

Lorsque le serpent atteint un de ces murs activé, la fonction disable propre aux pastilles sont appelées et désactive tous les murs associés à la pastille en question.

III- Gestion du jeu

III-1. Partie

Le jeu est constitué d'une partie. Une partie est représentée comme un terrain carré de taille définie (par défaut 15 Élément x 15 Élément).

Ainsi, une partie est composé de vector pour chacun des éléments dont elle est constituée. Ce sont ces vector qui sont parcourus pour l'affichage du jeu, ainsi que pour les différentes actions réalisées lorsque le serpent atteint les pastilles. Pour diminuer les calculs, ainsi que le parcours des différents vector lors du déroulement du jeu, un tableau à 2 dimensions de pointeur sur Element. Lors de la création du jeu, chaque case du tableau pointe vers l'élément du jeu (présent dans les différents vector) dont la position est égale à la position du pointeur dans le tableau. Par exemple au démarrage du jeu le vector snake (qui contient le snake et son évolution future) est composé de la tête du snake qui a pour position (0,0). Ainsi la case [0][0] du tableau pointe vers la tete du serpent présente dans le vector snake.

Ceci permet 2 choses, lorsque l'on déplace le serpent, nous pouvons voir si le déplacement est possible en vérifiant si l'élément pointé du tableau à la position correspondant au déplacement est traversable ou non (pas besoin de parcourir tous les vector du jeu pour trouver l'élément présent à la position du déplacement voulu et ensuite de vérifier si cet élément est traversable ou non).

Dans un second temps, lors du déplacement du serpent dans le jeu, il faut déplacer tous les Body constituants le serpent et positionner le chemin, qui était à la position où le serpent veut se déplacer, à l'ancienne position de dernier Body du serpent.

Pour effectuer ceci, nous avons juste à agir sur les éléments pointés par le tableau aux indices correspondantes aux positions (sans parcourir le vector représentant le serpent et le vector représentant les chemins pour agir sur les bons éléments). En revanche lorsque le serpent parcourt une EatablePastille, il grandit. Ainsi, le seul moyen de mettre en place ceci (en plus d'ajouter un élément Body à la fin du vector correspondant au serpent) est de parcourir le vector correspondant aux chemins et de supprimer l'élément présent à la place du Body ajouté au serpent (la méthode erase des vector ne pouvant prendre en paramètre seulement des iterator et non des pointeurs).

III-2. Partie Simple & Partie Complexe

Ensuite, nous avons différents types de partie. Ainsi, de cette classe Partie héritent 2 classes filles Partiesimple et Partiecomplexe.

Dans Partiesimple nous retrouvons les mêmes attributs que dans la classe mère Partie. Nous retrouverons seulement 2 types de vector : vector<Chemin> et vector <Body> modélisant les chemins du jeu et le serpent, ainsi qu'une instance pour chacune des 3 pastilles régissant le jeu.

Partiecomplexe quant à elle, est composée de vector supplémentaires : vector<Mur> représentant les murs normaux dont le comportement n'évolue pas au cours du jeu, vector<SmokedMur> et vector<VortexMur>.

Nous avons choisi d'utiliser des vector pour regrouper les différents éléments du jeu car ceci nous permet d'unifier certaines méthodes du jeu sous forme de méthode template de vector.

Dans les paramètres du jeu, les murs bordant le terrain de jeu ne sont pas créés. En effet, ces derniers n'ont aucune réelle utilité à être instancié, ce serait créer des instances redondantes de la classe Mur. C'est pourquoi, ils sont juste présents à l'affichage, mais pas dans le tableau à 2 dimensions de pointeurs sur Élément.

III-3. Snake - gestion du jeu

Afin d'ajouter d'autres fonctionnalités au jeu et de gérer le début et la fin du jeu, nous avons créé une classe Snake. Cette classe est chargée de proposer au joueur de reprendre la partie précédemment jouée ou de choisir un nouveau jeu (en sélectionnant le mode et le niveau de jeu souhaité), de lancer le jeu, de sauvegarder l'état du jeu lorsqu'il se finit.

Pour enregistrer l'état du jeu, nous avons défini des surcharges d'opérateurs présentes dans les classes clefs du jeu (Element, Partie, Partiesimple, Partiecomplexe) permettant d'utiliser plus simplement l'ajout de données dans un flux (ici un fichier). L'état du jeu est stocké dans un fichier texte, de la façon suivante :

1. nombre de points
2. type de partie (ps -> Partiesimple ; pc -> Partiecomplexe)
3. chemins
4. positions des différents éléments présents dans le vector chemin du jeu
5. snake
6. positions des différentes parties du serpent
7. pastilleSmoke
8. position de la pastille Smoke
9.

..... et ainsi de suite pour tous les éléments du jeu en fonction du type de Partie en cours.

Ainsi, lorsque l'on veut relancer la partie précédente, on peut recréer le bon type de partie dans sa totalité en lisant les données enregistrées.

IV- Affichage graphique

IV-1.Choix de la librairie

Nous avons décidé de réaliser l'entièreté du jeu pour qu'il soit fonctionnel et ainsi aviser selon le temps qu'ils nous restaient pour se pencher sur l'interface graphique du jeu.

Nous avons décidé de faire un affichage statique , c'est à dire que l'utilisateur lance le jeu depuis le terminal, les règles du jeu s'affichent sur le terminal, puis le choix de la partie. Au moment où le type de partie est choisi la fenêtre s'ouvre. L'affichage sera rafraichît après chaque action du joueur, mais ces actions (les bords , les collisions etc.) sont gérés par le code en C++ sans utiliser les fonctionnalités de la bibliothèque que nous avons choisie : **SFML**. Enfin la fenête se ferme seule quand on quitte le jeu (ECHAP + ENTREE).

Comme son nom l'indique elle simple et efficace, elle est destinée à l'implémentation de jeu vidéo et elle est écrite en C++. De plus elle s'accompagne de modules permettant de gérer la 2D, l'audio , les évènements et les fenêtres.

IV-2. Mise en place de l'affichage

Pour implémenter l'affichage graphique , nous avons ajouté en attributs une texture et un sprite à la classe Element, ainsi chaque type d'élément est créer en même temps que son affichage graphique dans le constructeur. Nous avons une texture différente pour chaque type de mur, pastille, un fond vert, les contours du terrain , la tête et un morceau de corps du snake.

Nous avons choisi de prendre une fenêtre de taille 1071 x 1071 px , elle peut donc contenir 17 par 17 éléments qui chacun sont des carrés de taille "ronde" de 63 x 63 px.
En enlevant les bordures nous avons un terrain jouable de 15 par 15 cases de 63 x 63 px.
Nous indiquons ainsi cette taille pour les sprites des éléments.

La fenêtre est rafraîchit grâce à MatrixGame qui contient tous les éléments du terrain. Elle est d'abord nettoyer entièrement, puis on redessine le fond et les bordures grâce à la fonction drawFond(fenêtre).On parcourt ensuite cette matrice et on affiche (fonction drawSprite(fenêtre) dans la fenêtre les éléments un par un , grâce à leur sprite en attribut, et enfin on display la fenêtre.

// ajouter pour la fermeture des bails