Centrale
CASABLANCA

SNAKE GAME

ANNEE UNIVERSITAIRE : 2022/2023

Réalisé par :

MOHAMMED EL BARHICHI

HIBA NOKRA

ABDERRAZZAK OUTZOULA

MOHAMED OUTAABOUT

Contents

1

Introduction et Motivation

Ce projet constitue la phase finale de notre formation **PMOO** dans ce semestre. Il est une opprotinuté pour nous d'éxploiter différents techniques acquises en séances de **TDs** et les **CMs** .

Bien qu'un jeu video, généralement, soit de nature complexe englobant plusieurs parties plus ou moins semblables, la programmation orienté d'objet permet de simplifier sa construction.

Le jeu, objet de notre projet, est le fameux jeu classique snake. Son but est de contrôler une petite ligne qui se déplace sur l'écran et de ramasser des objets (bonus,nourriture) tout en évitant les murs et les malus.

2

Guide de jeu

Le jeu consiste à diriger le snake pour manger des nourritures qui apparaissent aléatoirement dans la fenêtre. Chaque fois que le snake mange, sa queue s'allonge et son score augmente. Le but est de manger le plus possible sans heurter les murs ou sa propre queue. Le jeu se déroule dans une fenêtre de résolution 960 × 560 (qHD), qui peuvent être modifiés dans le code.

On clique sur ESPACE pour commencer notre jeu :

Figure 2.1: L'interface

En haut droite, votre score initié par 0 s'affiche à droite, et qui s'augmente à chaque seconde tant que le snake est vivant, et votre nombre de vie restant s'affiche en bas à gauche.

Il est initié par 3 et peut-être modifié dans le code.

Le snake apparaît en position initiale au milieu de la fenêtre qui est dirigé vers le haut. Vous utilisez les flèches dans votre clavier pour pouvoir diriger le snake au sein de la fenêtre qui est une suite de carrés de la longueur 20 pixels.

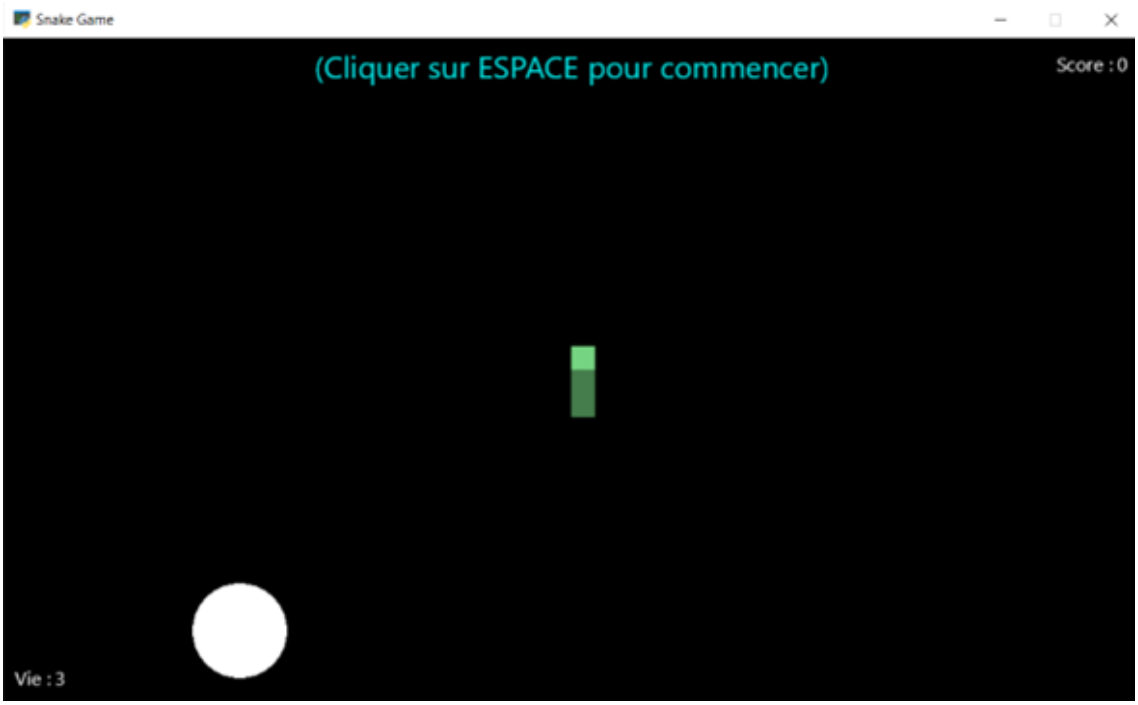

Figure 2.2: L'interface

Si le Snake entre en collision avec les limites de la fenêtre, la partie s'arrête, tout comme s'il passe par sa propre queue. Toutes les 5 secondes, une boule blanche de rayon 40 pixels apparaîtra de manière aléatoire dans la fenêtre de telle sorte qu'elle ne tombe pas sur le corps du Snake . Chaque fois que vous passez par ce disque appelé nourriture, la taille de votre Snake augmentera de 5 carrés et votre score augmentera également.

Toutes les 17 secondes, une boule de bonus de taille de rayon 20 pixels apparaît de manière aléatoire dans la fenêtre, reste pendant 5 secondes et disparaît (boule jeune). Après avoir passé par cette boule, la taille du serpent diminue de 3 carrés.

Toutes les 12 secondes, une boule de bonus de score de rayon 20 pixels apparaît de manière aléatoire dans la fenêtre, reste pendant 5 secondes et disparaît (boule violette). Après avoir passé par

cette boule, votre score augmente de 20 points.

Toutes les 8 secondes, une boule rouge représentant un malus apparaît de manière aléatoire dans la fenêtre.

Si le Snake passe par cette boule, le jeu s'arrête. Si des vies restent encore, vous pouvez cliquer sur ESPACE pour recommencer un autre match. Le score du dernier match est ajouté à un score total qui est affiché à la fin de l'ensemble des vies.

En cliquant sur ESPACE tout au long du jeu, le jeu se met en pause et en cliquant une autre fois, le jeu continue.

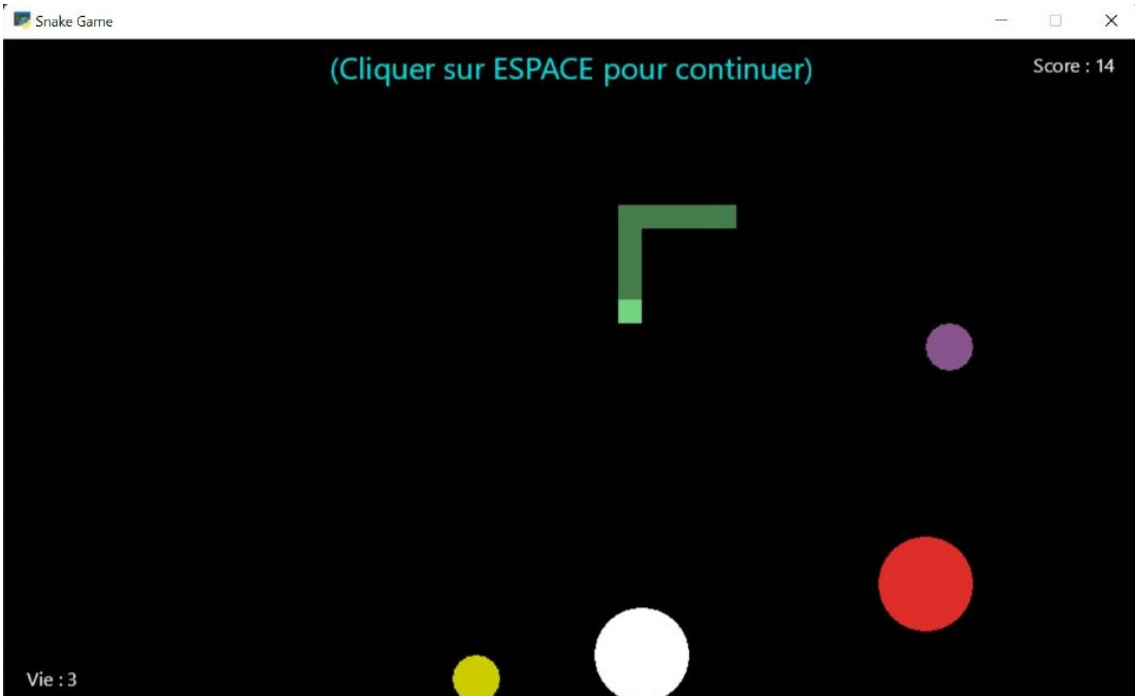

Figure 2.3: L'interface

3

Les étapes de développement du code

Initialisation

- Création de la classe **Fenetre** qui gère l'interface graphique.
- Création d'une fenêtre vide de dimensions 960x560 pixels.
- Ajout du compteur du score et de vies disponibles.
- Création de la méthode on_draw() pour tracer sur la fenêtre le score et les vies.
- Discrétisation de l'espace d'évolution en des cellules carrés de longueur taille _de_cellule=20 pixels.

Création du snake

- Création de la classe Snake qui gère les différents paramètres et les mouvements du snake
- Initialisation de la position de la tête du snake au milieu de la fenêtre:

$$x = longuer//2 et y = largeur//20$$

, ainsi de s'assurer que ces carrés tombe sur des cellules.

- Création de la queue: une liste qui contient les coordonnées de la queue qui se changent à chaque mouvement du Snake. Initialement, cette liste contient 2 tuples qui représentent les coordonnées des deux carrés de départ.

- Création du test de la position initiale du snake.

Gestion du mouvement du snake

- Création de la variable direction dans la classe **Fenetre** , qui identifie la direction du mouvement du snake : un tuple de deux coordonnées (x,y) ; +1 dans une coordonnée identifie que le snake doit bouger dans le sens positif du coordonnée. -1 signifie le sens négatif.
- Création de la méthode **on_key_press** dans la classe **Fenetre** qui gère la modification de la direction.
- Création de la méthode bouge dans la classe du snake, qui gère le mouvement du snake en se basant sur la direction . le principe du mouvement est le suivant :
  - On ajoute les coordonnées de la tête à la liste de la queue.
  - On donne la nouvelle position de la tête en se basant sur la direction.
  - On supprime les coordonnées du carrée qui représente l'extrémité de la queue
  - Si le Snake passe par la nourriture, la variable **queues_a_** ajoute devient 5, cette variable diminue de 1 à chaque 0.1 secondes. Tant que cette variable est supérieure à 0, on ne supprime pas les coordonnées du carrée extrême.
- Définition de la méthode **update1** dans la classe **Fenetre** ;

  à chaque 0.1 seconde la fonction est appelée pour assurer le mouvement du Snake. Cela est assuré par l'utilisation de la méthode **schedule_interval**.
- Assurer que le snake ne se déplace pas dans le sens interdit : direction inverse du déplacement.
- Création du test des différents cas du mouvement.

Initialisation de la classe Nourriture

- Utilisation de la fonction **Randint** pour obtenir une position aléatoire : on obtient un entier qui est entre le rayon de la nourriture et (la longueur de la fenêtre - le rayon).

- Pour s'assurer que que la nourriture ne tombe pas sur des positions non acceptables, il faut que les coordonnées des positions soient des multiples de la taille des cellules. On retient donc le quotient du nombre obtenue par Randint, et on le multiple par la taille de la cellule.
- Création de la méthode **update3** qui assure l'apparition des nourritures chaque 5 secondes
- Création du test de la position initiale de la nourriture : elle doit être dans la fenêtre et ne doit pas tomber initialement sur le Snake (sa tête et sa queue)

Détection de la collision entre le snake et la nourriture

- Création de la méthode collision dans la classe Snake qui retourne True si le Snake passe pas une nourriture, et False sinon.
- Pour assurer que les points obtenues en passant par la nourriture (5 points) ne s'ajoutent qu'une seule fois, et pour supprimer la nourriture de la fenêtre au moment de la collision, on crée l'attribut nourriture_encore, qui est un Booléen égal à True tant que le Snake n'a pas fait une collision avec la Nourriture. Et à chaque nouvelle apparition de nourritures (update3), cette attribut est revient à True.
- Création de la méthode **update3** qui assure l'apparition des nourritures chaque 5 secondes
- On vérifie donc la condition **nourriture_encore** avant de tracer les nourritures.

Gestion du score

- À chaque 1 seconde, le score s'augmente tant que le Snake est vivant, cela est assuré par le méthode update2 qui est rappelée chaque 1 seconde par la methode **schedule _interval**.
- Au moment de la collision entre le Snake et une nourriture, le score s'augmente de 5 points.

Gestion de la fonction 'recommencer'

- Création de la fonction recommencer qui a pour but de réinitialiser les différentes paramètres
- Ajout de la possibilité de continuer une autre partie avec la même taille du Snake après avoir perdu si le nombre de vies restantes est supérieure à 1: cela est assuré par l'attribut jeu_termine

qui et initié par **False**, et qui devient **True** si le jeu est terminé. En cliquant sur ESPACE tant que **jeu_termine** est **True** et tant que le score est supérieure à 1, le jeu une nouvelle partie recommence.

Gestion des bonus et des malus

- Création des deux classes**BonusTaille** ,**BonusScore** , et **Malus** qui gèrent les bonus de taille, ceux du score, et les malus; ces classes héritent les mêmes méthodes que dans la classe **Nourriture** .
- Pour rendre le jeu un peu plus difficile, on a choisi le rayon des bonus égale à 20 pixels, et celui des malus égale à 40 pixels.

Gestion du BonusTaille

- Assurer que ces bonus (de couleur jaune) apparaissent chaque 17 secondes dans une position aléatoire dans la fenêtre et disparaissent après 5 secondes de leurs apparition, cela est assuré par la méthode définie dans la classe **fenetre**: cinq_seconde_passe_score. (le temps d'apparition est modifiable dans le fichier **fenetre.py**).
- Creation de la méthode **update5** qui assure l'apparition de ces bonus chaque 17 secondes.
- Diminution de la taille du snake à chaque fois qu'il fait une collision avec un BonusTaille de 3 carrés **queue=queue[3:]** (le nombre de carrés à supprimer est modifiable dans le code).
- Création du test qui s'assurer que le BonusTaille ne tombe ni sur le snake ni sur les autres disques (Nourriture et autres bonus/malus qu'on va expliquer ultérieurement).

Gestion du BonusScore

- Assurer que ces bonus (de couleur violet) apparaissent chaque 12 secondes dans une position aléatoire dans la fenêtre et disparaissent après 5 secondes de leurs apparition cela est assuré par la méthode définie dans la classe **Fenetre**: **cinq_seconde_passe_score**. (le temps d'apparition est modifiable dans le fichier **fenetre.py**).

- Creation de la méthode **update5** qui assure l'apparition de ces bonus chaque 12 secondes.
- À chaque fois que le Snake fait une collision avec un BonusScore, le score augmente de 20 points (le nombre de points à ajouté est modifiable dans la classe **Fenetre**)
- Création du test qui s'assurer que les BonusScore ne tombent ni sur le snake ni sur les autres disques (Nourriture et autres bonus/malus).

Gestion des Malus

- Assurer que ces malus (de couleur Rouge) apparaissent chaque 8 secondes dans une position aléatoire dans la fenêtre (le temps d'apparition est modifiable dans le fichier **fenetre.py**)
- Création de la méthode update6 qui assure l'apparition de ces malus chaque 8 secondes
- Au moment de la collision entre le Snake et un malus, l'attribut **jeu_stop** devient **True**.
- Création du test qui s'assurer que les Malus ne tombent ni sur le snake ni sur les autres disques.

Gestion de l'écran du GameOver et de Pause

- Création de l'attribut **jeu_termine** qui est initié par False
- Avant de faire une update, on vérifie que cette attribut est égale à False
- Création de la méthode **jeu_termine_condition** qui s'assure que le Snake ne sort pas de l'écran et ne passe pas pas son queue.
- À chaque 0.1 seconde, on s'assure que les conditions d'arrêt du jeu ne sont pas vérifiées (dans **update1**)
- L'attribut **jeu_pause** est une booléen qui gère le fait de mettre le jeu en pause en cliquant sur ESPACE. Si on clique une autre fois sur ESPACE, le jeu continue.

4

Le problème rencontré et sa solution

Quand on change rapidement la direction du Snake deux fois avant que 0,1 secondes passe, il s'avère que le snake peut bouger dans une direction qui est interdite (l'opposé de la direction actuelle).

Nous avons donc pensé à ajouter l'attribut direction_choisi qui assure qu'on ne choisit que 1 nouvelle direction chaque 0.1 seconde.

A

Les axes d'amélioration

- Améliorer l'interface graphique en ajoutant des images et des formes.
- Ajouter des effets sonores.
- Ajouter d'autres type bonus: bonus qui ajoute une vie supplémentaires.
- Ajouter d'autres type de malus: malus qui augmente la vitesse de déplacement du snake; malus qui diminue le temps d'apparition des bonus.
- Jouer sur le nombre de malus qui peuvent être affichés en même temps (former des obstacles)
- Ajout des murs indestructibles
- Ajouter des portails de téléportation

Grâce à ce projet, nous avons appris comment manipuler la bibliothèque pyglet tout en exploitant différentes connaissances. Cela nous a permis de nous initier dans le monde de construction et de développement des jeux video. En effet, bien que le jeu snake soit de nature simple, le cheminement vers différentes solutions de construction a contribué au développement de notre reflexion.