|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BTS SIO | | | | |
|  |  | | |  |
| Programmation Python | | | | |
|  | | Pygameplay |  | |

TABLE DES MATIÈRES

[1. Lancement 3](#_Toc165294341)

[2. Fonctionnement 4](#_Toc165294342)

[3. Base de donnée 12](#_Toc165294343)

[4. Annexe 13](#_Toc165294344)

# Lancement

Afin de lancer le fichier « maître » de ce projet, rendez vous dans le dossier « dist » du projet Select\_Jeu

Lancez le fichier « app.exe » si vous êtes un utilisateur Windows ou « app.sh » si vous êtes un utilisateur linux

---A l’attention des utilisateurs Linux---  
Veuillez lancer app.sh comme un programme, ou exécutez la commande bash app.sh (ou « ./app.sh » dans un terminal relié au dossier dist)

# Fonctionnement

• Vous avez le choix entre 9 jeux, chaques jeux utilisent des techniques différentes différentes.

ZCasino

Ce code est une implementation d'un jeu de roulette en utilisant la bibliothèque Tkinter pour créer une interface graphique. Voici une explication détaillée du code :

**Classe RouletteGame**

La classe RouletteGame représente le jeu de roulette. Elle a plusieurs attributs et méthodes :

\_\_init\_\_ : c'est le constructeur de la classe, qui est appelé lorsque l'objet est créé. Il initialise les attributs de l'objet, notamment le solde initial du joueur (1000 dollars) et les éléments de l'interface graphique (labels, entrées, bouton).

tourner\_roulette : c'est la méthode qui est appelée lorsque le joueur clique sur le bouton "Tourner la roulette". Elle récupère les valeurs entrées par le joueur (mise et numéro ou couleur misé), vérifie si elles sont valides, puis simule le tirage de la roulette.

get\_numero\_couleur : c'est une méthode qui prend un numéro en entrée et retourne la couleur associée à ce numéro (rouge, noir ou vert).

**Méthode tourner\_roulette**

La méthode tourner\_roulette fait les opérations suivantes :

Elle récupère la mise et le numéro ou la couleur misé entrés par le joueur.

Elle vérifie si la mise est valide (supérieure à zéro) et si le numéro ou la couleur misé est valide (entre 0 et 50 ou "rouge" ou "noir").

Elle simule le tirage de la roulette en générant un numéro aléatoire entre 0 et 50.

Elle détermine la couleur gagnante en fonction du numéro gagnant.

Elle compare le numéro ou la couleur misé avec le numéro gagnant et la couleur gagnante.

Si le joueur a gagné, elle calcule le gain en fonction de la mise et du type de pari (numéro ou couleur).

Elle met à jour le solde du joueur en ajoutant le gain ou la perte.

Elle affiche un message d'information avec le résultat du tirage et le gain ou la perte.

**Interface graphique**

L'interface graphique est créée à l'aide de la bibliothèque Tkinter. Elle comprend :

Un label qui affiche le solde du joueur.

Un label et une entrée pour la mise.

Un label et une entrée pour le numéro ou la couleur misé.

Un bouton "Tourner la roulette" qui appelle la méthode tourner\_roulette lorsque le joueur clique dessus.

**Fonction main**

La fonction main crée une instance de la classe RouletteGame et lance la boucle principale de l'interface graphique avec root.mainloop().

En résumé, ce code crée un jeu de roulette simple avec une interface graphique qui permet au joueur de miser sur un numéro ou une couleur, et de voir le résultat du tirage.

Othello

. Ce code est une implementation du jeu de Reversi (également connu sous le nom d'Othello) en utilisant la bibliothèque Tkinter pour créer une interface graphique. Voici une explication détaillée du code :

**Classe ReversiGame**

La classe ReversiGame représente le jeu de Reversi. Elle a plusieurs attributs et méthodes :

\_\_init\_\_ : c'est le constructeur de la classe, qui est appelé lorsque l'objet est créé. Il initialise les attributs de l'objet, notamment la taille du plateau (8x8), le plateau lui-même (une matrice de zéros), les symboles des joueurs (⚫ et ⚪), et les boutons de l'interface graphique.

update\_board : c'est la méthode qui met à jour l'affichage du plateau en fonction de l'état actuel du jeu.

play\_move : c'est la méthode qui est appelée lorsque le joueur clique sur un bouton pour jouer un coup. Elle vérifie si le coup est valide, met à jour le plateau, et passe le tour au joueur adverse si nécessaire.

make\_move : c'est la méthode qui applique les règles du jeu pour flipper les pièces adverses lorsque le joueur joue un coup.

is\_valid\_move : c'est la méthode qui vérifie si un coup est valide en fonction des règles du jeu.

is\_any\_valid\_move : c'est la méthode qui vérifie si il existe un coup valide pour un joueur donné.

show\_winner : c'est la méthode qui affiche le résultat du jeu lorsque la partie est terminée.

computer\_move : c'est la méthode qui joue un coup aléatoire pour le joueur adverse (l'ordinateur).

**Méthode play\_move**

La méthode play\_move fait les opérations suivantes :

Elle vérifie si le coup est valide en appelant la méthode is\_valid\_move.

Si le coup est valide, elle applique les règles du jeu en appelant la méthode make\_move.

Elle passe le tour au joueur adverse en appelant la méthode computer\_move si nécessaire.

Elle met à jour l'affichage du plateau en appelant la méthode update\_board.

**Méthode make\_move**

La méthode make\_move applique les règles du jeu pour flipper les pièces adverses lorsque le joueur joue un coup. Elle parcourt les directions adjacentes à la pièce jouée et flippe les pièces adverses qui sont entre la pièce jouée et une pièce du même joueur.

**Méthode is\_valid\_move**

La méthode is\_valid\_move vérifie si un coup est valide en fonction des règles du jeu. Elle parcourt les directions adjacentes à la pièce jouée et vérifie si il existe une pièce adverse qui peut être flippée.

**Méthode is\_any\_valid\_move**

La méthode is\_any\_valid\_move vérifie si il existe un coup valide pour un joueur donné. Elle parcourt le plateau et appelle la méthode is\_valid\_move pour chaque case.

**Méthode show\_winner**

La méthode show\_winner affiche le résultat du jeu lorsque la partie est terminée. Elle compte le nombre de pièces de chaque joueur et détermine le vainqueur.

Méthode computer\_move

La méthode computer\_move joue un coup aléatoire pour le joueur adverse (l'ordinateur). Elle sélectionne un coup valide au hasard parmi les coups possibles.

**Fonction main**

La fonction main crée une instance de la classe ReversiGame et lance la boucle principale de l'interface graphique avec root.mainloop().

En résumé, ce code crée un jeu de Reversi avec une interface graphique qui permet au joueur de jouer contre un autre joueur.

Space Invaders

Le jeu Space Invaders est développé en utilisant la bibliothèque Pygame en Python. Le code est structuré en plusieurs modules, chacun ayant une fonction spécifique.

Le fichier "space\_invaders.py" est le point d'entrée du jeu. Il initialise les variables globales, charge les images et les sons, et lance la boucle principale du jeu.

Le module "game" contient la classe "Game" qui gère l'état du jeu. Elle initialise les variables du jeu, gère les événements de la fenêtre Pygame, et met à jour l'état du jeu.

Le module "player" contient la classe "Player" qui gère le vaisseau du joueur. Elle initialise le vaisseau, gère ses mouvements, et gère le tir de ses projectiles.

Le module "invaders" contient la classe "Invaders" qui gère les envahisseurs. Elle initialise les envahisseurs, gère leur mouvement, et gère leur destruction.

Le module "projectiles" contient la classe "Projectile" qui gère les projectiles. Elle initialise les projectiles, gère leur mouvement, et gère leur collision avec les envahisseurs ou le vaisseau du joueur.

Le module "utils" contient des fonctions utilitaires pour gérer les collisions, charger les images et les sons, et afficher du texte à l'écran.

Le code utilise une approche orientée objet pour structurer le jeu, ce qui permet de le rendre modulaire et facile à maintenir. Les classes sont utilisées pour gérer les différents éléments du jeu, tels que le vaisseau du joueur, les envahisseurs, et les projectiles. Les fonctions utilitaires sont regroupées dans un module séparé pour faciliter leur réutilisation.

Snake Eater

**Initialisation**

Le code commence par importer les bibliothèques nécessaires, notamment Pygame et SQLite. Il définit ensuite les variables suivantes :

id : l'identifiant de l'utilisateur, passé en argument lors de l'exécution du programme

puissance : la difficulté du jeu, choisie par l'utilisateur

difficulty : la difficulté du jeu, égale à puissance

frame\_size\_x et frame\_size\_y : la taille de la fenêtre de jeu

**Fonctions**

Les fonctions suivantes sont définies :

game\_over() : affiche l'écran de fin de partie et met à jour le score dans la base de données SQLite

show\_score() : affiche le score du joueur

**Variables de jeu**

Les variables suivantes sont définies :

snake\_pos : la position du serpent

snake\_body : le corps du serpent

food\_pos : la position de la nourriture

food\_spawn : un booléen indiquant si la nourriture doit être spawnée

direction : la direction du serpent

change\_to : la direction que le serpent doit prendre

score : le score du joueur

**Boucle principale**

La boucle principale du jeu est la suivante :

Le code écoute les événements de l'utilisateur, tels que des clics de souris ou des touches de clavier.

Si l'utilisateur appuie sur une touche, le code met à jour la direction du serpent.

Le code déplace le serpent dans la direction choisie.

Si le serpent atteint la nourriture, le score est incrémenté et la nourriture est spawnée à nouveau.

Le code vérifie si le serpent est sorti des limites de la fenêtre ou s'il a touché son propre corps. Si c'est le cas, la fonction game\_over() est appelée.

Le code affiche le serpent, la nourriture et le score sur la fenêtre de jeu.

Le code met à jour la fenêtre de jeu et attend un certain temps avant de répéter la boucle.

**Fin du jeu**

Lorsque le jeu est terminé, la fonction game\_over() est appelée pour afficher l'écran de fin de partie et mettre à jour le score dans la base de données SQLite.

Pendu

Ce code est un jeu de pendu créé avec la bibliothèque Pygame. Voici une explication détaillée de chaque partie du code :

**Initialisation**

Le code commence par importer les bibliothèques nécessaires, notamment Pygame. Il initialise ensuite la fenêtre de jeu avec une taille de 1000x500 pixels et définit les couleurs et les polices utilisées dans le jeu.

**Définition des variables**

Les variables suivantes sont définies :

word : le mot à deviner, initialisé à une chaîne vide

buttons : une liste de boutons pour les lettres, initialisée à une liste vide

guessed : une liste des lettres déjà devinées, initialisée à une liste vide

hangmanPics : une liste d'images pour le pendu, chargées à partir de fichiers

limbs : le nombre de parties du pendu déjà dessinées, initialisé à 0

**Fonctions**

Les fonctions suivantes sont définies :

redraw\_game\_window() : redessine la fenêtre de jeu en affichant les boutons pour les lettres, le mot à deviner et le pendu

randomWord() : choisit un mot aléatoire à partir d'un fichier de mots

hang(guess) : vérifie si la lettre devinée est présente dans le mot

spacedOut(word, guessed) : affiche le mot avec des espaces pour les lettres devinées

buttonHit(x, y) : vérifie si un bouton pour une lettre est cliqué

end(winner) : affiche l'écran de fin de partie, avec un message de victoire ou de défaite

reset() : réinitialise la partie en remettant à zéro les variables et en choisissant un nouveau mot

**Boucle principale**

La boucle principale du jeu est la suivante :

La fonction redraw\_game\_window() est appelée pour redessiner la fenêtre de jeu.

Le code attend les événements de l'utilisateur, tels que des clics de souris ou des touches de clavier.

Si l'utilisateur clique sur un bouton pour une lettre, la fonction buttonHit() est appelée pour vérifier si la lettre est présente dans le mot.

Si la lettre est présente, la fonction hang() est appelée pour vérifier si le pendu doit être dessiné.

Si le pendu doit être dessiné, le nombre de parties du pendu déjà dessinées est incrémenté.

Si le mot est entièrement deviné, la fonction end() est appelée avec un argument True pour afficher l'écran de fin de partie avec un message de victoire.

Si le pendu est complet, la fonction end() est appelée avec un argument False pour afficher l'écran de fin de partie avec un message de défaite.

**Fin du jeu**

Lorsque le jeu est terminé, la fonction pygame.quit() est appelée pour fermer la fenêtre de jeu.

En résumé, ce code crée un jeu de pendu où l'utilisateur doit deviner un mot en cliquant sur des boutons pour les lettres. Le jeu affiche le mot à deviner, les lettres déjà devinées et le pendu, et permet à l'utilisateur de gagner ou de perdre en fonction de ses choix.

Casse brique

**HTML**

Le code HTML définit une page web avec un canvas de taille 480x320. Le canvas est utilisé pour dessiner le jeu.

**CSS**

Le code CSS définit le style de la page web. Le canvas a une couleur de fond grise et est centré sur la page.

**JavaScript**

Le code JavaScript définit les variables et les fonctions du jeu.

ballRadius, x, y, dx, dy, paddleHeight, paddleWidth, paddleX, rightPressed, leftPressed, brickRowCount, brickColumnCount, brickWidth, brickHeight, brickPadding, brickOffsetTop, brickOffsetLeft, score, lives : variables utilisées pour stocker les différentes valeurs du jeu.

bricks : tableau à deux dimensions pour stocker les briques du jeu.

keyDownHandler, keyUpHandler, mouseMoveHandler : fonctions pour gérer les événements clavier et souris.

collisionDetection : fonction pour détecter les collisions entre la balle et les briques.

drawBall, drawPaddle, drawBricks, drawScore, drawLives : fonctions pour dessiner les différents éléments du jeu.

draw : fonction pour dessiner et mettre à jour le jeu.

**Fonctionnement du jeu**

Le jeu commence en appelant la fonction draw(). Cette fonction dessine les briques, la balle, la raquette, le score et le nombre de vies sur le canvas. Elle vérifie également les collisions entre la balle et les briques.

Lorsque la balle touche une brique, la brique est supprimée et le score est incrémenté. Lorsque la balle touche le bas du canvas, une vie est perdue. Lorsque toutes les vies sont perdues, le jeu est terminé.

Le joueur peut déplacer la raquette en utilisant les flèches gauche et droite du clavier. La souris peut également être utilisée pour déplacer la raquette.

Lorsque le score atteint le nombre total de briques, un message de victoire est affiché et le jeu est rechargé.

Tetris

Le jeu Tetris est développé en Python en utilisant la bibliothèque Pygame. Le code est structuré en plusieurs modules, chacun ayant une fonction spécifique.

Le fichier "tetris.py" est le point d'entrée du jeu. Il initialise les variables globales, charge les images et les sons, et lance la boucle principale du jeu.

Le module "game" contient la classe "Game" qui gère l'état du jeu. Elle initialise les variables du jeu, gère les événements de la fenêtre Pygame, et met à jour l'état du jeu.

Le module "board" contient la classe "Board" qui gère le plateau de jeu. Elle initialise le plateau, gère la chute des pièces, et gère la suppression des lignes complètes.

Le module "pieces" contient les classes "Piece" et "Shape" qui gèrent les pièces du jeu. Elles initialisent les pièces, gèrent leur mouvement, et gèrent leur collision avec le plateau de jeu.

Le module "utils" contient des fonctions utilitaires pour gérer les collisions, charger les images et les sons, et afficher du texte à l'écran.

Le code utilise une approche orientée objet pour structurer le jeu, ce qui permet de le rendre modulaire et facile à maintenir. Les classes sont utilisées pour gérer les différents éléments du jeu, tels que le plateau de jeu, les pièces, et les événements de la fenêtre Pygame. Les fonctions utilitaires sont regroupées dans un module séparé pour faciliter leur réutilisation.

Sudoku

Importation des bibliothèques nécessaires : Le code commence par importer les bibliothèques nécessaires telles que pygame, sys, random et time.

Initialisation de Pygame : Le module pygame est initialisé en utilisant la fonction pygame.init().

Définition des constantes : Des constantes telles que la largeur de l'écran, la hauteur de l'écran, la taille de la grille et la taille de la cellule sont définies.

Définition des fonctions : Plusieurs fonctions sont définies pour gérer différents aspects du jeu, telles que le dessin de la grille, la génération d'une grille de Sudoku, la vérification qu'un mouvement est valide et la vérification que le jeu est gagné.

Création de la fenêtre de jeu : Une fenêtre de jeu est créée en utilisant la fonction pygame.display.set\_mode().

Création d'une grille de Sudoku : Une grille de Sudoku est créée en générant un puzzle de Sudoku aléatoire et en masquant certaines des cellules.

Boucle de jeu : La boucle de jeu principale est exécutée en utilisant une boucle while. À l'intérieur de la boucle, les événements sont gérés, l'état du jeu est mis à jour et l'écran est redessiné.

Dessin de la grille : La grille est dessinée sur l'écran en utilisant la fonction draw\_grid().

Gestion des entrées utilisateur : Les entrées utilisateur sont gérées en utilisant la fonction pygame.event.get(). Si l'utilisateur clique sur une cellule, le nombre correspondant est entré ou supprimé.

Vérification d'une victoire : Après chaque mouvement, le jeu est vérifié pour voir s'il est gagné en utilisant la fonction check\_win().

Quitter le jeu : Le jeu est quitté en utilisant la fonction pygame.quit() lorsque l'utilisateur ferme la fenêtre de jeu ou appuie sur le bouton de quitter.

Morpion

Ce code est un jeu de Morpion (Tic-Tac-Toe) créé avec la bibliothèque Tkinter en Python. Le jeu permet à un joueur humain de jouer contre l'ordinateur (IA) ou contre un autre joueur humain.

La classe MorpionGame est le cœur du jeu. Elle initialise la grille de jeu, les boutons radio pour sélectionner le mode de jeu, et les boutons de la grille de jeu.

La méthode AfficherGrille met à jour l'affichage de la grille de jeu en fonction de l'état de la grille.

La méthode jouer\_coup est appelée lorsque le joueur humain clique sur un bouton de la grille de jeu. Elle met à jour la grille de jeu, vérifie si le joueur a gagné, et si le mode de jeu est contre l'IA, elle appelle la méthode jouer\_IA pour que l'IA joue son tour.

La méthode check\_victoire vérifie si le jeu est terminé et qui est le vainqueur. Si le jeu est terminé, elle affiche un message de victoire et ferme la fenêtre du jeu.

La méthode jouer\_IA utilise l'algorithme Minimax pour déterminer le meilleur coup pour l'IA. Elle explore toutes les possibilités de jeu et choisit le coup qui maximise les chances de gagner.

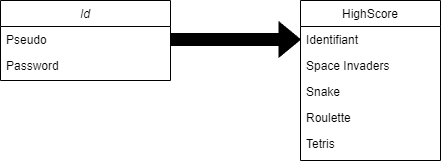
La méthode minimax\_decision est appelée par jouer\_IA pour déterminer le meilleur coup. Elle explore toutes les possibilités de jeu et retourne le coup qui maximise les chances de gagner.

La méthode minimax\_value est appelée par minimax\_decision pour évaluer la valeur d'un coup. Elle explore toutes les possibilités de jeu et retourne la valeur du coup.

La méthode terminal\_test vérifie si le jeu est terminé et qui est le vainqueur.

Enfin, la fonction main crée une instance de la classe MorpionGame et lance la boucle principale du jeu.

# Base de donnée

* La base de donnée est en SQLite, elle se nomme « connect.db », elle se trouve dans le dosser DataBase  
  L’utilité de cette base de donnée est à la fois de gérer les utilisateurs (Création, Connection, Suppression) mais aussi de sauvegarder leurs scores dans certains jeux tel que le Snake, Space Invaders, Tetris et un jeu de roulette de casino (ZCasino)
* Le modèle de la base de donnée ressemble a ceci :

Text

integer

integer

integer

integer

Text

Text

* L’utilisateur peut consulter ses meilleurs scores mais aussi le score des autres utilisateurs de l’application via un menu dédié en renseignant son identifiant ou celui d’un autre utilisateur  
  Il peut aussi supprimer son compte en confirmant son identité via une seconde saisie de son mot de passe dans un menu dédié  
  La fonctionnalité de modifier son mot de passe est aussi possible via un menu dédié en renseignant son ancien et son nouveau mot de passe

# Annexe

* Insérer les schémas