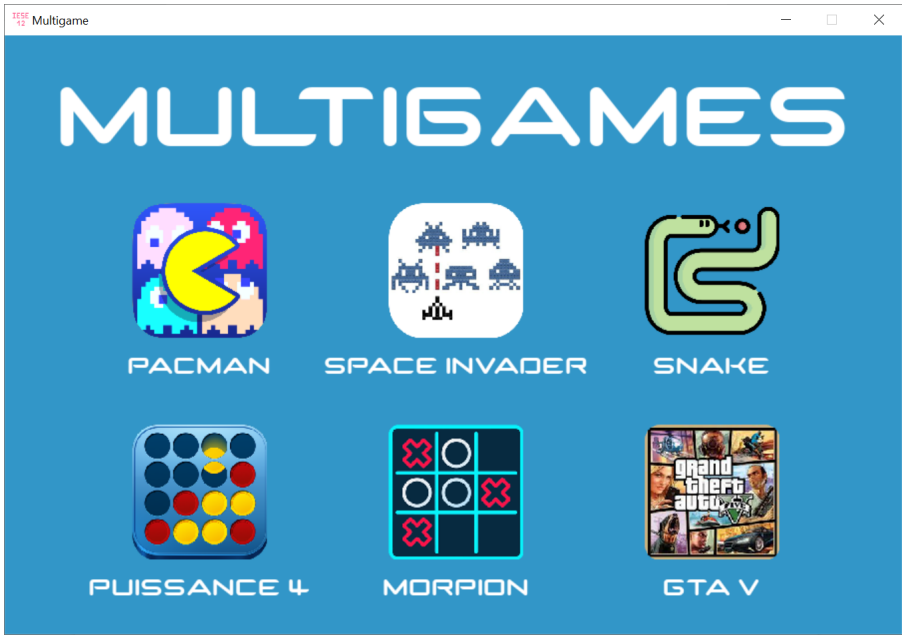
Projet Logiciel

Multigames



Tristan Lailler
Joris Bouvier
Martin Collomb-Clerc
Arthur Abelkalon

Sommaire

I) Présentation générale

L'idée de faire une plateforme multi jeux comme projet logiciel nous est venue assez rapidement. Nous voulions faire un jeu vidéo, la création de ce dernier étant ludique, cependant nous ne savions pas vers quel jeu exactement aller. C'est pour cela que nous avons choisi de faire plusieurs jeux tous rassemblés et accessibles depuis un menu.

Notre projet consiste donc en une plateforme multi jeux se présentant comme un menu permettant d'accéder à différents jeux. Au total, il y a 3 jeux jouables seul et 2 jeux jouables à 2 joueurs. Les trois jeux solo sont Pacman, Space Invader et Snake et les deux jeux multijoueurs sont le puissance 4 et le morpion.

Nous avons ensuite décidé de travailler en utilisant Python avec la librairie Pygame et ce pour plusieurs raisons. La première est que l'utilisation de python est simple et également que la bibliothèque Pygame permet de simplifier énormément la création d'un jeu. La seconde est la grande documentation à laquelle nous avions accès en utilisant ce langage et cette bibliothèque, simplifiant ainsi notre compréhension et la résolution de nos problèmes.

II) Organisation de l’équipe

Outil utilisé

Lors de ce projet et pour pouvoir partager nos fichiers de façon simple nous avons utilisé GitHub avec l’extension GitHub Desktop. On a trouvé cette application très facile pour partager nos fichiers, GitHub détecte automatiquement lorsqu’un changement est détecté et nous propose de push notre document sur GitHub ou de pull un document pour l’avoir en local. Nous avons tout de même trouvé que cette application peut être embêtante lorsque nous travaillons à plusieurs sur un même fichier car la dernière version à avoir été push est conservée et l’autre est écrasée.

Nous avons aussi mis en place un groupe Messenger afin de partager des informations lorsque nous ne sommes pas ensemble, nous avons aussi pu via cette plate-forme nous tenir au courant de l’avancement de chaque projet et nous envoyer de petite démonstration vidéo.

Répartition du travail

Pour pouvoir réaliser notre projet nous nous somme partager les tâches, dans un premier temps nous avons discuté des différents jeux que nous pouvions ajouter à notre plateforme. Une fois cela fait nous nous somme réparti les jeux à coder, Tristan c’est occupé du pacman et du menu, Arthur c’est occupé du Snake, Joris c’est occupé du Puissance 4 et du Morpion et Martin c’est occupé du Space Invader, pour le dernier jeu GTA V nous nous sommes tous mis en collaboration pour pouvoir faire fonctionner ce jeu.

La plupart du temps nous avons travaillé sur le projet depuis chez nous et de notre côté étant donné que chaque personne avait un jeu dédié, mais lors des séances de cours nous nous retrouvions afin de pouvoir apporter une nouvelle vision sur notre code et de pouvoir s’aider lors d’éventuel problème.

III) Structure du projet

Fichiers

La clarté de notre code étant importante, et pour bien délimiter les différentes parties, nous avons utilisé plusieurs fichiers pour les différents jeux voire pour les différentes classes dans un même jeu. De plus, séparer les fichiers sur lesquels les différents membres du groupe travaillaient permettait de sauvegarder dans GitHub sans avoir peur d'écraser le travail de quelqu'un d'autre.

Classes

L'utilisation de classes, fonctionnalité native de python a été très utile et utilisée pour structurer le code. Pour certaines, elles ont hérité de la classe pygame.sprite.Sprite possédant des fonctionnalités et fonctions intéressantes. En effet, elles peuvent aussi être mises en groupe et utilisées ensemble. Par exemple pour afficher les sprites contenus dans un groupe il suffit de faire Groupe.draw(Surface) qui dessine sur la Surface toutes les images, contenues dans l'attributs *image* des Sprite du Group, aux coordonnées contenues dans leur attribut *rect*.

Un main simple et des fonctions explicites

Pour faciliter la compréhension du code par d'autres personnes nous avons essayé de réaliser une fonction main, boucle principale du jeu, simplifiée au maximum dans chaque jeu. Comment ? En utilisant des appels de fonction avec des noms explicites. Cela permet presque de lire une histoire lorsqu'on lit le main.

Une classe modèle

Il nous a manqué de temps, mais une idée de structuration était de créer un classe dont hériterait tous les jeux et qui uniformiserait l'affichage, la gestion des évènements, et d'autres attributs communs.

IV) Problèmes rencontrés, solution apportées et possibilités d'amélioration

Problèmes et solutions

Mener à bien la conception de différents jeux, c'est aussi y apporter des solutions astucieuses. L'ensemble des problèmes rencontrés et les solutions trouvées ont été répertoriés dans le tableau ci-dessous.

Jeu	Description du problème	Date de découverte	Solution(s)	Date de correction
Pacman	Apparition de pacman	17/05	Redimensionnement des images	18/05
Pacman	Déplacement des fantômes différent selon la couleur	17/05	Algorithmes de déplacement indépendants	19/05
Pacman	Collision avec les pastilles	19/05	Masque de collision pygame	24/05
Puissance 4	Échec de la détection de la victoire	18/05	Considération des cas manquants	18/05
Puissance 4	Placement infini des pions lors d'appui prolongé	17/05	– (Solution non trouvée)	**?**
Morpion	Pas de problème : similaire au Puissance 4			
Space Invaders	Descente des vaisseaux après un changement de direction	24/05	Considération d'un vaisseau et non de plusieurs (pour éviter d'avoir une boucle)	24/05
Space Invaders	Écran "game over" non figé	24/05	Affichage hors boucle	24:05
Snake	Déplacement et changement de direction	24/05	Sauvegarde de la direction précédente	24/05
Snake	Dimensionnement	21/05	Échelle du jeu adaptée à l'écran de l'utilisateur	**?**

Améliorations

De manière générale, l'ensemble des jeux peuvent être améliorés en y implémentant une intelligence artificielle (puissance 4, morpion) ou en ajoutant un système de niveaux de difficulté (snake, space invaders, pacman).

De plus, le projet peut être en permanence amélioré : en y ajoutant au fur et à mesure les différents jeux développés par les autres groupes.

V) Annexe

Documentation :

Menu :

Menu

Classe pour gérer le menu d'accueil du multigames.

__init__(self) → None
Initialise l'instance de la classe Menu à sa création.

init_jeux(self) → None
Créé des sprites Jeu et les ajoute dans un groupe de sprite.

resolution_events(self) → None
Gère les events présents dans la file d'attente. Vide la file lorsque tous les events sont traités.

affiche(self) → None
Affiche le menu : affiche les icônes des jeux.

main_menu(self) → None
Boucle principale : lance un jeu si il est sélectionné, attend un choix sinon.

Jeu

Classe héritant de pygame.sprite.Sprite, représente l'icône du jeu dans le menu et fait le lien avec le jeu correspondant.

__init__(self, jeu, titre, x, y, image) → None
Initialise l'instance de la classe Jeu à sa création.

GTA

Classe pour gérer l'icône de GTA V envoyant sur une page internet.*

main(self) → None
redirige vers le lien choisi.

Pacman :

PacmanJeu

Classe pour gérer le jeu

__init__(self) → None
Initialise l'instance de la classe PacmanJeu à sa création.

main(self) → None
Joue une partie de Pacman.

nouveau_niveau(self) → None
Initialise le Pacman, les Fantomes, les Pastilles et les Bonus, pour un nouveau niveau.

suite_manche(self) → None
Réinitialise le Pacman et les Fantômes.

init_murs(self) → None
Créé les sprites Murs et les ajoute dans un groupe de sprite.

init_pastilles(self) → None
Créé des sprites Pastilles et Bonus (Pastilles spéciales) et les ajoute dans un groupe de sprite.

init_fantomes(self) → None
Créé des sprites Fantome et les ajoute dans un groupe de sprite.

affiche(self) → None
Affiche sur un fond noir les murs, les pastilles, les fantomes, pacman, le score et les vies restantes.

affiche_score(self) → None
Affiche le score.

affiche_vies(self) → None
Affiche le nombre de vies en dessinant un pacman par vie restante.

resolution_events(self) → None
Gère les events présents dans la file d'attente. Vide la file lorsque tous les events sont traités.

avance(self) → None
Fait avancer pacman et les fantômes.

update_animations(self) → None
Met à jour l'animation en fonction du temps.

update_score(self,collisions) → None
Met à jour le score en fonction des pastilles mangées.

check_win(self) → None
Vérifie si toutes les pastilles ont été mangées. Si oui, alors la manche est gagnée.

animation_win(self) → None
Affiche les murs et les fait clignoter en bleu/blanc.

game_over(self) → None

Affiche les murs, les pastilles, le score et “game over” au milieu de l’écran.

Mur(pygame.sprite.Sprite)

Classe héritant de pygame.sprite.Sprite, représente les murs du labyrinthe dans lequel pacman évolue.

__init__(self,x,y,jeu) → None

Initialise l’instance de la classe Mur à sa création.
Paramètres :
int x,y : position du Mur dans la grille de la map,
PacmanJeu jeu : instance de PacmanJeu auquel ce Mur appartient.

get_image(self, image_blanche = False) → pygame.Surface

Paramètre :
Bool image_blanche : si à True, retourne une l’image du mur en blanc sinon en bleu
Retourne une pygame.Surface en fonction de la position des murs adjacents.

Pacman(pygame.sprite.Sprite)

Classe héritant de pygame.sprite.Sprite, représente pacman que le joueur contrôle.

__init__(self,jeu) → None

Initialise l’instance de la classe Pacman à sa création.

reset(self) → None

Réinitialise la position du pacman, au milieu bas de la map.

update_rect(self) → None

Met à jour la position de l’image du pacman.

avance(self) → None

Déplace le pacman dans la direction voulue par les flèches du clavier si possible.

colision_prochain(self,direction) → pygame.sprite.Sprite/None

Teste si le déplacement du pacman dans la direction voulue est possible ou s’il rencontre un mur.
Retourne None si le déplacement est possible, sinon retourne le Mur avec lequel pacman collisionnerait.

check_bonus(self,collisions) → None

Teste si pacman passe sur un bonus, l’active si c’est le cas.

update_image(self) → None

Met à jour l’image de pacman en fonction de sa direction et de l’état de son animation.

animation_mort(self) → None

Affiche l’animation de pacman qui meure.

Fantome(pygame.sprite.Sprite)

Classe héritant de pygame.sprite.Sprite, représente un fantôme qui chasse pacman.

__init__(self,jeu,nom) → None

Initialise l'instance de la classe Fantome à sa création.

update_rect(self) → None

Met à jour la position de l'image du fantôme.

avance(self) → None

Déplace le fantôme selon sa direction.

colision_prochain(self,direction) → pygame.sprite.Sprite/None

Teste si le déplacement du fantôme dans la direction voulue est possible ou s'il rencontre un mur.
Retourne None si le déplacement est possible, sinon retourne le Mur avec lequel le fantôme collisionnerait.

on_intersection(self) → Bool

Teste si plusieurs chemins s'offrent au fantôme.

update_direction(self) → None

Met à jour la direction du fantôme en fonction de sa couleur (chaque fantôme se déplace selon un algorithme différent) et de la position de pacman.

directions_possibles(self) → list

Retourne une liste contenant les directions que le fantôme peut prendre. Il ne peut pas faire demi-tour.

objectif(self) → Tuple

Retourne la case de la map qui est l'objectif du fantôme. Il est défini en fonction de sa couleur (chaque fantôme se déplace selon un algorithme différent) et de la position de pacman.

ordre_pref_dir(self,vect_objectif) → Tuple

Retourne un tuple d'int, classant les directions selon la position de l'objectif et celle du fantôme. Le premier élément est la direction la plus souhaitable, la dernière étant la moins souhaitable.

update_image(self) → None

Met à jour l'image de pacman en fonction de sa direction et de l'état de son animation.

check_collision(self) → None

Teste si le fantôme collisionne avec pacman.
Si le bonus est actif, le fantôme est tué et réapparaîtra dans une seconde.
Sinon pacman est tué.

Pastille(pygame.sprite.Sprite)
Classe héritant de pygame.sprite.Sprite, représente une pastille qui doit être mangée par pacman.

__init__(self,x,y,jeu) → None
Initialise l'instance de la classe Pastille à sa création.

Bonus(pygame.sprite.Sprite)
Classe héritant de pygame.sprite.Sprite, représente une pastille activant le bonus. Bonus : pacman devient invincible et peut manger les fantômes.

__init__(self,x,y,jeu) → None
Initialise l'instance de la classe Bonus à sa création.

Snake :

SnakeJeu:
Classe pour gérer le jeu

__init__(self) → None
Initialise l'instance de la classe SnakeJeu à sa création.

main(self) → None
Joue une partie de Snake.

init_game(self) → None
Initialise les paramètres de jeu et crée un snake

set_difficulty(self, a) → None
Modifie le paramètre de difficulté de la partie.
Une difficulté élevée rend le jeu plus dur.
Paramètre:
- <int> a: Difficulté

generate_map(self) → None
Crée un plateau en damier

event_update(self) → None
Garde en mémoire la position précédente de la tête.
Vérifie si des touches ont été enclenchées et détermine si la trajectoire a changé.
Modifie la variable move le cas échéant.

update_move(self) → None
Met à jour le positionnement du Snake.
Vérifie si la partie est perdue. Sinon agrandit le snake s'il est sur le point d'attraper une pomme.

generate_apple(self,a) → None

Définit aléatoirement ou non (selon la valeur de a) la position de la prochaine pomme.

Paramètre:

- <int> a : Génération aléatoire (0:non;1:oui)

display_apple(self) → None

Met à jour l'image de la pomme en fonction de son emplacement.

real_pos_snake(self) → None

Incrémente ou décrémente la position réelle du snake suite à un appel de event_update

grid_coordinates(self,coords) → Tuple

Transpose les coordonnées réelles afin d'avoir un positionnement conforme au plateau.

Paramètre:

- <tuple> coords : tuple de coordonnées réelles

affiche_grille(self) → None

Fonction de debug; affiche l'ensemble des positions du corps du Snake

show_snake(self) → None

Affiche le snake dans son intégralité grâce à ses attributs de positionnement.

dir_to_int(self,s) → int

Fait correspondre la direction littérale par table de correspondance

Paramètre:

- <string> s : direction ('UP','LEFT','DOWN','RIGHT','')

SnakeGenerate(pygame.sprite.Group)

Classe héritant de pygame.sprite.Group, représente l'ensemble des positions du snake, sa taille et ses mouvements.

__init__(self,jeu) → None

Initialise un snake pour la partie

Paramètres :

- <SnakeJeu> jeu : instance de SnakeJeu auquel ce snake appartient.

Puissance 4 :

P4Jeu:

Classe pour gérer le jeu

__init__(self) → None

Initialise l'instance de la classe P4 à sa création.

main(self) → None
Joue une partie de Puissance 4.

init_screen(self) → None
initialise l'écran pour une partie de Puissance 4.

detect_colonne(self) → None
Détecte sur quelle colonne le joueur appuie.

affiche_joueur(self) → None
Affiche le nom du joueur qui doit jouer.

placer_pion(self) → None
Place le pion du joueur dans la colonne ou il a appuyé.

test_suite(self) → None
Test si 4 jetons d'un même joueur sont alignés.

trouver_gagnant(self) → None
Parcourt la grille dans différents sens (horizontal, vertical, diagonal) et appel la fonction test_suite.

afficher_gagnant(self) → None
Affiche le gagnant lorsque la partie est terminée.

fin_partie(self) → None
Affiche un menu avec un bouton pour rejouer et un autre pour revenir au menu.

Morpion :

MorpionJeu:
Classe pour gérer le jeu

__init__(self) → None
Initialise l'instance de la classe Morpion à sa création.

main(self) → None
Joue une partie de Morpion.

init_screen(self) → None
initialise l'écran pour une partie de Morpion.

detect_pos(self) → None
Détecte sur quelle colonne le joueur appuie.

affiche_joueur(self) → None

Affiche le nom du joueur qui doit jouer.

placer_pion(self) → None
Place le pion du joueur dans la colonne ou il a appuyé.

test_suite(self) → None
Test si 3 jetons d'un même joueur sont alignés.

trouver_gagnant(self) → None
Parcourt la grille dans différents sens (horizontal, vertical, diagonal) et appel la fonction test_suite.

afficher_gagnant(self) → None
Affiche le gagnant lorsque la partie est terminée.

fin_partie(self) → None
Affiche un menu avec un bouton pour rejouer et un autre pour revenir au menu.

Space Invader :

Sapce_Invader:
Classe pour gérer le jeu :

__init__(self) → None
initialise l'instance de la classe Space invader et les différentes variables

main(self) → None
Joue une partie de Space Invader

create_obstacle(self, x_start, y_start, offset_x) → None
crée un obstacles dans le jeu
x_start, y_start : position de départ pour un obstacle
offset_x : offset d'un obstacle

create_multiple_obstacle(self, *offset, x_start, y_start) → None
crée l'ensemble des obstacles
*offset: variable pour l'offset de tous les obstacles
x_start, y_start : position de départ de tous les obstacles

aliens_setup(self, rows, cols, x_distance=50, y_distance=50, x_offset=40, y_offset=60) → None
permet la mise en place de tous les aliens
rows, cols : définie le nombre de lignes et de colonnes d'aliens
x_distance,y_distance : définit la distance entre chaque alien dans le bloc
x_offset, y_offset : offsets du bloc d'aliens

alien_position_checker(self) → None

vérifie la position des aliens et gère leurs déplacements latéraux

alien_move_down(self, distance) → None
permet de faire descendre les aliens
distance : donne le nombre de pixel que les aliens descendent

alien_shoot(self) → None
permet aux aliens de tirer

extra_alien_timer(self) → None
gère le timer de l'apparition de l'alien extra

collision_checks(self) → None
vérifie l'ensemble des collisions du jeu

display_lives(self) → None
affiche le compteur de vie

display_score(self) → None
affiche le score

victory_message(self) → None
affiche le message de victoire

death_message(self) → None
affiche le message de défaite

run(self) → None
appelle toute les fonction d'update et de dessin

Player:
Classe pour gérer le joueur :

__init__(self, pos, constraint, speed) → None
initie le joueur, load son image et initie les variable propres au joueur
pos : définit la position de départ du joueur
constraint : définit la contrainte de déplacement du joueur
speed : définit la vitesse du joueur

get_input(self) → None
définit les interactions claviers

recharge(self) → None
définit la cadence de tir du joueur

constraint(self) → None
évite que le joueur de sorte de l'écran

shoot_laser(self) → None
fait tirer les laser du joueur

update(self) → None
permet la mise à jour des précédentes fonction

Block :
Classe qui permet de définir les caractéristiques d'un obstacle

__init__(self, size, color, x, y)→ None
définit les caractéristiques et la forme d'un obstacle

Alien
Classe qui définit les aliens :

__init__(self, color, x, y)→ None
définit les caractéristiques des aliens
color : permet de savoir quel sprite utiliser
x, y : définit la position de l'alien

update(self, direction)→ None
met à jour les aliens
direction : définit dans quel sens va l'alien

Extra
Classe qui définit l'alien extra:

__init__(self, side)→ None
définit les caractéristiques de l'alien extra
side : définit de quel côté apparaît l'alien extra

update(self)→ None
met à jour l'alien extra

Laser
Classe qui définit les caractéristique des lasers

__init__(self,pos,speed,screen_height)→ None
définit les caractéristiques d'un laser
pos : définit la position de départ du laser
speed : définit la vitesse du laser
screen_height : permet d'avoir la hauteur de l'écran

destroy(self)→ None
permet au laser de ne pas sortir de l'écran

update(self)→ None
met à jour les lasers