Documentation technique

1. Introduction

- **Présentation du projet** : Le jeu choisi est Pong. L'objectif est d'empêcher la balle de passer derrière votre raquette tout en essayant de faire passer la balle derrière celle de votre adversaire. Chaque fois qu'un joueur échoue, l'autre marque un point. La partie continue jusqu'à ce qu'un score limite soit atteint ou qu'un joueur se retire.

- **Objectif du projet** : Développé dans le cadre de la matière *Qualité de Développement*, ce projet permet de pratiquer la programmation tout en respectant les bonnes pratiques. Il aide à comprendre les bases des mécaniques de jeu, comme la gestion des mouvements et des collisions, tout en écrivant un code lisible et maintenable.

2. Architecture générale

- **Langage et environnement de développement** : Pour ce projet nous avons décidé d'utiliser Python en tant que langage de programmation tout en utilisant la bibliothèque Pygame pour l'affichage.

- **Structure du code** : Le code est séparé entre 4 fichiers principaux.

  Tout d'abord nous avons le fichier **main.py** qui permet le lancement du jeu, ce fichier récupère des informations de 3 fichiers différents.

  Le premier est nommé **interface.py** : il contient la classe Interface qui permet d'afficher les raquettes, la balle de jeu et le score.

  Le second est nommé **menu.py** : il contient la classe nommée Menu. Celle-ci permet l'affichage du menu d'accueil du jeu avec le choix "Jouer" ou "Quitter" ainsi que les commandes à utiliser pour pouvoir se déplacer dans ce menu et valider son choix.

  Le dernier fichier est nommé **jeu.py** : Il contient la classe nommée Jeu. Cette classe permet de gérer la logique de jeu et ses paramètres. Nous y initialisons la taille des raquettes ainsi que leur position d'origine, définissons aussi la vitesse de la balle lors d'une partie ainsi que la taille de la fenêtre de jeu. De plus cette classe permet de gérer la collision des raquettes avec les bords de l'écran, la collision de la balle avec les raquettes, ainsi que la collision avec le bord de l'écran pour pouvoir mettre à jour le score.

3. Fonctionnalités principales

- **Contrôles des joueurs** : Les joueurs contrôlent les raquettes à l'aide des touches Z et S pour le premier, et flèche haute et basse pour le deuxième.

- **Mouvement de la balle** : La balle se déplace en ligne droite, suivant un angle défini par un vecteur de direction (avec des composantes horizontales et verticales). Lorsqu'elle rebondit contre le plafond/sol ou les raquettes, elle rebondit.

- **Score et conditions de victoire** : A chaque fois que la balle touche le mur derrière une raquette, le joueur en face gagne un point. Un joueur gagne quand l'autre décide d'arrêter la partie.

- **Menu et interface utilisateur** : L'utilisateur commence sur un menu principal, et peut naviguer parmi les options disponibles (Jouer ou quitter) puis appuyer sur Entrée pour valider son choix. S'il clique sur Echap, il revient au menu principal et termine la partie.

4. Détails techniques

Les différents classes et leur méthodes:

- Classe **Jeu**
  - *__init__ :* Initialise les positions et vitesses de la balle, les positions des raquettes, les scores, et les dimensions de l'écran.
  - reset : Réinitialise les variables à leur état par défaut.
  - *update :* Met à jour la position de la balle, gère les collisions avec les bords de l'écran et les raquettes, et met à jour les scores.
  - *reset_balle :* Réinitialise la position et la vitesse de la balle au centre de l'écran.

- Classe **Interface**
  - __init__ : Initialise l'écran de jeu.
  - render : Affiche les raquettes, la balle, et les scores sur l'écran.

- Classe **Menu**
  - __init__ : Initialise l'écran du menu, la police de caractères, les options du menu, et l'option sélectionnée.
  - render : Affiche les options du menu sur l'écran.
  - handle_event : Gère les événements de clavier pour naviguer dans le menu et sélectionner une option.

Animation et rendu graphique:

- Rendu :
  - Interface : La méthode render de la classe Interface dessine les raquettes, la balle, et les scores sur l'écran.

    - Menu : La méthode render de la classe Menu dessine les options du menu sur l'écran.

- Mise à jour de l'écran : L'écran est rempli avec une couleur de fond (noir) avant de dessiner les éléments pour éviter les artefacts visuels.

5. Optimisation et gestion des performances

- **Gestion de la boucle de jeu** : La boucle principale fait plusieurs actions : capturer et gérer les interactions utilisateur, alterne entre le menu et le jeu, mettre à jour l'état du jeu et les positions des objets, et enfin redessiner l'écran et limite la fréquence à 60 images par seconde pour une fluidité optimale.

- **Problèmes de performance et solutions** : Afin d'éviter les problèmes de rafraichissement, les images par secondes sont limitées à 60.

6. Déploiement et configuration

Pré-requis techniques

1. Langage de programmation : Python 3.6 ou supérieur.
2. Dépendances : pygame, une bibliothèque utilisée pour la création de jeux en 2D.

Instructions d'installation et d'exécution

1. Installer Python : Téléchargez et installez Python depuis python.org.
2. Installer les dépendances : Ouvrez un terminal ou une invite de commandes puis exécutez la commande suivante pour installer pygame : **pip install pygame**
3. Télécharger le code source depuis GitHub : Ouvrez un terminal puis clonez le dépot GitHub à l'aide de la commande suivante : **git clone https://github.com/MatheoStv/Pong-STEVANCE-Matheo-FAISCA-Noah-Hallot-Florent**
4. Lancer le jeu : Lancer un terminal dans le répertoire contenant le jeu puis exécuter la commande suivante : **python main.py**

Plan de tests

Tests de la logique de jeu

Récapitulatif des tests :

1. **Initialisation du jeu** : Vérification que les attributs initiaux du jeu sont correctement définis.
2. **Mise à jour de la position de la balle** : Vérification que la position de la balle est mise à jour selon sa vitesse.

3. **Collision haut/bas** : Vérification que la balle rebondit correctement sur les bords supérieur et inférieur.
4. **Collision raquette J1** : Vérification que la balle rebondit en touchant la raquette du joueur 1.
5. **Collision raquette J2** : Vérification que la balle rebondit en touchant la raquette du joueur 2.
6. **Mise à jour du score J2** : Vérification que le score du joueur 2 s'incrémente si la balle dépasse la raquette de J1.
7. **Mise à jour du score J1** : Vérification que le score du joueur 1 s'incrémente si la balle dépasse la raquette de J2.
8. **Réinitialisation de la balle** : Vérification que la balle se réinitialise correctement après un point.

1. Test d'initialisation

- **Objectif** : Vérifier que les attributs du jeu sont correctement initialisés.
- **Scénario** : Initialiser un objet Jeu et vérifier les valeurs initiales des attributs.
- **Données de test** : N/A
- **Critères d'acceptation** :
    - position_balle est [400, 300]
    - vitesse_balle est [4, 4]
    - position_j1 et position_j2 sont 250
    - score1 et score2 sont 0
- **Validation du test** : Les valeurs initiales sont celles attendues.

2. Mise à jour de la position de la balle

- **Objectif** : Vérifier que la position de la balle est mise à jour correctement.
- **Scénario** : Appeler la méthode update et vérifier la nouvelle position de la balle.
- **Données de test** : Appel de self.jeu.update()
- **Critères d'acceptation** : La nouvelle position de la balle doit être [404, 304].
- **Validation du test** : La balle s'est déplacée de [4, 4].

3. Collision de la balle avec le haut et le bas de l'écran

- **Objectif** : Vérifier que la balle rebondit lorsqu'elle touche le bord supérieur ou inférieur.
- **Scénario** : Placer la balle à [400, 0] et [400, 600] et vérifier la direction de la vitesse verticale après update.
- **Données de test** :
    - position_balle = [400, 0]
    - position_balle = [400, 600]
- **Critères d'acceptation** :
    - Si position_balle est [400, 0], vitesse_balle[1] doit être 4.
    - Si position_balle est [400, 600], vitesse_balle[1] doit être -4.
- **Validation du test** : La vitesse verticale de la balle est inversée en cas de collision.

4. Collision de la balle avec la raquette du joueur 1

- **Objectif** : Vérifier que la balle rebondit lorsqu'elle touche la raquette de J1.
- **Scénario** : Placer la balle en collision avec la raquette de J1 et appeler update.
- **Données de test** : position_balle = [50 + largeur_raquette, position_j1 + 50]
- **Critères d'acceptation** : vitesse_balle[0] doit être 4.
- **Validation du test** : La balle rebondit correctement sur la raquette du joueur 1.

5. Collision de la balle avec la raquette du joueur 2

- **Objectif** : Vérifier que la balle rebondit lorsqu'elle touche la raquette de J2.
- **Scénario** : Placer la balle en collision avec la raquette de J2 et appeler update.
- **Données de test** : position_balle = [largeur_ecran - 50 - largeur_raquette, position_j2 + 50]
- **Critères d'acceptation** : vitesse_balle[0] doit être -4.
- **Validation du test** : La balle rebondit correctement sur la raquette du joueur 2.

6. Mise à jour du score du joueur 2

- **Objectif** : Vérifier que le score du joueur 2 est mis à jour lorsque la balle dépasse la raquette de J1.
- **Scénario** : Placer la balle au-delà de la raquette de J1 et appeler update.
- **Données de test** : position_balle = [50, position_j1 + hauteur_raquette + 10]
- **Critères d'acceptation** :
    - score2 doit être incrémenté de 1.
    - position_balle doit être réinitialisé à [400, 300].
- **Validation du test** : La balle est réinitialisée et le score de J2 est mis à jour.

7. Mise à jour du score du joueur 1

- **Objectif** : Vérifier que le score du joueur 1 est mis à jour lorsque la balle dépasse la raquette de J2.
- **Scénario** : Placer la balle au-delà de la raquette de J2 et appeler update.
- **Données de test** : position_balle = [largeur_ecran - 50, position_j2 + hauteur_raquette + 10]
- **Critères d'acceptation** :
    - score1 doit être incrémenté de 1.
    - position_balle doit être réinitialisé à [400, 300].
- **Validation du test** : La balle est réinitialisée et le score de J1 est mis à jour.

8. Réinitialisation de la balle

- **Objectif** : Vérifier que la balle est correctement réinitialisée après qu'un point est marqué.
- **Scénario** : Appeler reset_balle et vérifier les valeurs de position_balle et vitesse_balle.
- **Données de test** : Appel de reset_balle()

- **Critères d'acceptation** :
    - position_balle doit être [400, 300].
    - vitesse_balle doit être [4, 4].
- **Validation du test** : La position et la vitesse de la balle sont correctement réinitialisées.

Tests de l’interface de jeu

Récapitulatif des tests :

1. **Initialisation de l'interface** : Vérification de la création correcte de l'écran avec les bonnes dimensions.
2. **Rendu des raquettes** : Vérification que les raquettes sont dessinées à leurs positions respectives avec la couleur correcte.
3. **Rendu de la balle** : Vérification que la balle est rendue à la position correcte.
4. **Rendu du score** : Vérification que le score est correctement affiché à l'écran.

1. Test d'initialisation de l'interface

- **Objectif** : Vérifier que l'interface est correctement initialisée.
- **Scénario** : Initialiser un objet Interface, vérifier que l'écran est correctement créé et que ses dimensions sont correctes.
- **Données de test** : N/A
- **Critères d'acceptation** :
    - L'attribut ecran de l'interface ne doit pas être None.
    - La largeur de l'écran doit être 800 pixels.
    - La hauteur de l'écran doit être 600 pixels.
- **Validation du test** : L'écran a bien été créé avec les bonnes dimensions.

2. Test du rendu des raquettes

- **Objectif** : Vérifier que les raquettes sont dessinées aux bonnes positions sur l'écran.
- **Scénario** : Définir les positions des raquettes (joueur 1 et joueur 2), appeler la méthode render et vérifier les pixels aux positions attendues.
- **Données de test** :
    - position_j1 = 100
    - position_j2 = 200
    - Appel de self.interface.render(self.jeu)
- **Critères d'acceptation** :
    - La raquette du joueur 1 doit être dessinée à la position [50, 100].
    - La raquette du joueur 2 doit être dessinée à la position [largeur_ecran - 50 - largeur_raquette, 200].
    - Les pixels correspondants aux positions doivent être blancs (255, 255, 255, 255).
- **Validation du test** : Les raquettes sont bien dessinées aux bonnes positions avec la couleur correcte.

3. Test du rendu de la balle

- **Objectif** : Vérifier que la balle est dessinée à la bonne position sur l'écran.
- **Scénario** : Définir la position de la balle, appeler la méthode render et vérifier les pixels autour de la position de la balle.
- **Données de test** :
  - position_balle = [400, 300]
  - Appel de self.interface.render(self.jeu)
- **Critères d'acceptation** :
  - La balle doit être dessinée à la position [400, 300].
  - Les pixels autour de cette position doivent être blancs [255, 255, 255].
- **Validation du test** : Les pixels de la balle à la position donnée sont blancs, indiquant qu'elle est bien rendue à la bonne position.

4. Test du rendu du score

- **Objectif** : Vérifier que le score des joueurs est affiché correctement.
- **Scénario** : Définir les scores des joueurs, appeler la méthode render et vérifier les pixels autour de la position du score sur l'écran.
- **Données de test** :
  - score1 = 3
  - score2 = 5
  - Appel de self.interface.render(self.jeu)
- **Critères d'acceptation** :
  - Le score doit être visible quelque part sur l'écran, typiquement dans la région où il est attendu (par exemple, autour de [30:70, 300:500] pour le test).
  - Les pixels dans cette région doivent être blancs [255, 255, 255].
- **Validation du test** : Le score des joueurs est affiché à l'écran et les pixels sont correctement rendus.