**BOW MAN**

Documentation technique

Auteur : CUREAU Melvin

Date : 22/07/2024

Ce document comporte X pages :

* Page 1 : Page de garde ; Sommaire
* Page 2 : Introduction ; Architecture ;
* Page 3 : Choix techniques ; Structure de données
* Page 4 : Structure de données
* Page 5 : Références ; Conclusion

Introduction

Installation

Architecture du Projet

Description des classes principales

GameScene

Archer

AIArcher

Arrow

Obstacle

PauseScene

OptionsScene

EndGameScene

Gestion des événements

Gestion des collisions

Système de tir

Fonctionnement de l’ia

Conclusion

Annexes

Ressources

Références

# Introduction

**Bow Man** est un jeu de tir à l’arc développé en ***Python*** utilisant la bibliothèque ***Pygame***. Ce document vise à fournir une description complète de l’architecture du projet, des classes principales, de la gestion des événements, du système de tir, et de l’intelligence artificielle (IA) intégrée dans le jeu. Ce jeu se catégorise comme un jeu stratégique pouvant se jouer à maximum 2 joueurs. Le but du jeu est de toucher l’adversaire en tirant une flèche par l’intermédiaire de son arc sans voir son adversaire. Chaque joueur peut tirer une flèche par tour. Une fois le tir effectué le joueur aura un aperçu de la précision de sa flèche et pourra prendre en compte son résultat dans le but d’être plus précis au tour suivi.

Ce document explique les détails techniques de l’implémentation et de la réalisation du projet en ***Python*** et utilisant la librairie ***PyGame***.

# Installation

Pour installer et exécuter ce jeu, suivez les étapes ci-dessous :

1. Cloner le dépôt :
2. Accéder au répertoire du projet
3. Installer les dépendances
   1. Pip install -r requirements.txt
4. Lancer le jeu

# Architecture du projet

La structure des dossiers du projet est organisé comme suit :

bow-man/

├── assets/

│ ├── backgrounds/

│ ├── characters/

│ └── sounds/

├── gameLogic/

│ ├── \_\_init\_\_.py

│ ├── archer.py

│ ├── arrow.py

│ ├── obstacle.py

│ └── AIArcher.py

├── scenes/

│ ├── \_\_init\_\_.py

│ ├── gameScene.py

│ ├── pauseScene.py

│ ├── optionsScene.py

│ └── endGameScene.py

├── mainScene.py

├── requirements.txt

└── README.md

# Description des classes principales

## GameScene.py

***GameScene*** est la classe principale qui gère la logique de jeu, l'affichage des éléments graphiques, et la gestion des événements.

### Attributs :

* **screen** : L'écran de jeu.
* **width**, **height** : Dimensions de l'écran.
* **clock** : Horloge pour gérer les FPS.
* **background\_img** : Image de fond du jeu.
* **archer\_left, archer\_right** : Instances des archers (joueur et IA).
* **camera\_x** : Position de la caméra.
* **paused** : État du jeu (en pause ou non).
* **turn** : Tour actuel (gauche ou droite).
* **arrows** : Liste des flèches en jeu.
* **obstacle** : Instance de l'obstacle.

### Méthodes :

* **\_\_init\_\_(self, screen, settings)** : Initialisation de la scène de jeu.
* **handle\_events(self)** : Gestion des événements utilisateur.
* **player\_shoot(self)** : Gestion du tir du joueur.
* **ai\_shoot(self)** : Gestion du tir de l'IA.
* **check\_collisions(self)** : Vérification des collisions entre les flèches et les archers.
* **update(self)** : Mise à jour de l'état du jeu.
* **draw(self)** : Affichage des éléments graphiques.
* **run(self)** : Boucle principale du jeu.

## Archer.py

***Archer*** représente un archer dans le jeu.

### Attributs :

* **image** : Image de l'archer.
* **rect** : Rectangle de collision de l'archer.
* **screen** : Écran de jeu.
* **health** : Santé de l'archer.

### Méthodes :

* **\_\_init\_\_(self, image, x, y, screen)** : Initialisation de l'archer.
* **draw(self, camera\_x)** : Affichage de l'archer.

## AIArcher.py

***AIArcher*** représente un archer contrôlé par l'IA.

### Attributs :

* **image** : Image de l'archer.
* **rect** : Rectangle de collision de l'archer.
* **screen** : Écran de jeu.

shoot\_power, shoot\_angle, previous\_shots : Similaires à ceux de Archer.

### Méthodes :

* **\_\_init\_\_(self, image, x, y, screen)** : Initialisation de l'IA.
* **update(self)** : Mise à jour de l'IA (décision de tir).
* **shoot(self)** : Gestion du tir de l'IA.
* **draw(self, camera\_x)** : Affichage de l'IA.

## Arrow.py

***Arrow*** représente une flèche dans le jeu.

## Attributs :

* **screen, x, y, x\_velocity, y\_velocity** : Caractéristiques de la flèche.
* **rect** : Rectangle de collision de la flèche.

## Méthodes :

* **\_\_init\_\_(self, screen, x, y, x\_velocity, y\_velocity)** : Initialisation de la flèche.
* **update(self)** : Mise à jour de la position de la flèche.
* **draw(self, camera\_x)** : Affichage de la flèche.

## Obstacle.py

***Obstacle*** représente un obstacle dans le jeu.

## Attributs :

* **screen, x, y, width, height** : Caractéristiques de l'obstacle.
* **rect** : Rectangle de collision de l'obstacle.

## Méthodes :

* **\_\_init\_\_(self, screen, x, y, width, height)** : Initialisation de l'obstacle.
* **draw(self, camera\_x)** : Affichage de l'obstacle.

## PauseScene.py

***PauseScene*** gère l'affichage et la logique du menu de pause.

## Méthodes :

* **\_\_init\_\_(self, screen)** : Initialisation de la scène de pause.
* **draw(self)** : Affichage du menu de pause.
* **handle\_events(self)** : Gestion des événements du menu de pause.

## OptionsScene.py

***OptionsScene*** gère l'affichage et la logique du menu des options.

### Méthodes :

* **\_\_init\_\_(self, screen)** : Initialisation de la scène des options.
* **draw(self)** : Affichage du menu des options.
* **handle\_events(self)** : Gestion des événements du menu des options.
* **run(self)** : Boucle principale du menu des options.

## EndGameScene.py

***EndGameScene*** gère l'affichage et la logique de la scène de fin de jeu.

### Méthodes :

* **\_\_init\_\_(self, screen, winner)** : Initialisation de la scène de fin de jeu.
* **draw(self)** : Affichage de la scène de fin de jeu.
* **handle\_events(self)** : Gestion des événements de la scène de fin de jeu.
* **run(self)** : Boucle principale de la scène de fin de jeu.

# Gestion des événements

La gestion des événements est essentielle pour interagir avec le jeu. Dans gameScene.py, la méthode handle\_events(self) capture et traite les événements comme les pressions de touches, les mouvements de la souris, et les clics.

## Extrait du code : handle\_events

def handle\_events(self):

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            sys.exit()

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_ESCAPE:

                self.paused = not self.paused

            elif event.key == pygame.K\_a and not self.paused:

                self.player\_shoot()

            elif event.key == pygame.K\_o:

                self.obstacle = Obstacle(self.screen, self.scene\_width // 2 - 100, self.scene\_height - 150, 200, 800)

            elif event.key == pygame.K\_UP:

                self.shoot\_angle += 1

                if self.shoot\_angle > 90:

                    self.shoot\_angle = 90

            elif event.key == pygame.K\_DOWN:

                self.shoot\_angle -= 1

                if self.shoot\_angle < 0:

                    self.shoot\_angle = 0

            elif event.key == pygame.K\_RIGHT:

                self.shoot\_power += 1

                if self.shoot\_power > 100:

                    self.shoot\_power = 100

            elif event.key == pygame.K\_LEFT:

                self.shoot\_power -= 1

                if self.shoot\_power < 1:

                    self.shoot\_power = 1

        elif event.type == pygame.MOUSEBUTTONDOWN:

            if event.button == 1 and not self.paused:

                self.move\_camera\_right = not self.move\_camera\_right

            elif self.paused:

                action = self.pause\_menu.handle\_events()

                if action == "continue":

                    self.paused = False

                elif action == "options":

                    options\_menu = OptionsScene(self.screen)

                    options\_menu.run()

                elif action == "main\_menu":

                    from mainScene import MainMenu

                    main\_menu = MainMenu(self.screen)

                    main\_menu.run()

                    pygame.quit()

                    sys.exit()

                elif action == "quit":

                    pygame.quit()

                    sys.exit()

## Explications:

* Quitter le Jeu : Lorsque l'événement pygame.QUIT est capturé (fermeture de la fenêtre), le jeu se termine proprement.
* Pause : K\_ESCAPE permet de mettre en pause et de reprendre le jeu.
* Tirer une Flèche : K\_a tire une flèche si le jeu n'est pas en pause.
* Créer un Obstacle : K\_o crée un nouvel obstacle.
* Ajuster l'Angle et la Puissance du Tir : Les flèches directionnelles modifient self.shoot\_angle et self.shoot\_power.
* Gestion des Clics de Souris : Permet de bouger la caméra ou de naviguer dans le menu de pause.

# Gestion des collisions

La méthode check\_collisions(self) vérifie les collisions entre les flèches et les archers. Cette méthode est cruciale pour déterminer les impacts des flèches et réduire la santé des archers en conséquence.

## Extrait du code : check\_collisions

def check\_collisions(self):

    for arrow in self.arrows:

        if arrow.x < self.archer\_left.rect.right and arrow.x > self.archer\_left.rect.left and arrow.y < self.archer\_left.rect.bottom and arrow.y > self.archer\_left.rect.top:

            self.archer\_left.health -= arrow.damage

            self.arrows.remove(arrow)

            if self.archer\_left.health <= 0:

                end\_game\_menu = EndGameScene(self.screen, "Archer Gauche")

                end\_game\_menu.run()

        elif arrow.x < self.archer\_right.rect.right and arrow.x > self.archer\_right.rect.left and arrow.y < self.archer\_right.rect.bottom and arrow.y > self.archer\_right.rect.top:

            self.archer\_right.health -= arrow.damage

            self.arrows.remove(arrow)

            if self.archer\_right.health <= 0:

                end\_game\_menu = EndGameScene(self.screen, "Archer Droit")

                end\_game\_menu.run()

## Explications:

* Vérification des Coordonnées : Pour chaque flèche, on vérifie si ses coordonnées se trouvent à l'intérieur des coordonnées de l'archer.
* Réduction de la Santé : Si une flèche touche un archer, sa santé est réduite du montant des dégâts de la flèche.
* Suppression de la Flèche : Une fois la flèche ayant touché un archer, elle est supprimée de la liste self.arrows.
* Fin du Jeu : Si la santé de l'archer atteint 0, la scène de fin de jeu est lancée.

# Système de tir

Le système de tir est géré par les méthodes player\_shoot(self) et ai\_shoot(self). Le tir du joueur est déclenché par la touche ‘a’, tandis que le tir de l’IA est déclenché automatiquement selon une logique définie dans le fichier AIArcher.py.

## Extrait du code player\_shoot

def player\_shoot(self):

    angle\_radians = math.radians(self.shoot\_angle)

    x\_velocity = self.shoot\_power \* math.cos(angle\_radians)

    y\_velocity = -self.shoot\_power \* math.sin(angle\_radians)

    if self.turn == 'left':

        new\_arrow = Arrow(self.screen, self.archer\_left.rect.right, self.archer\_left.rect.centery, x\_velocity, y\_velocity)

        self.arrows.append(new\_arrow)

        self.turn = 'right'

    elif self.turn == 'right':

        new\_arrow = Arrow(self.screen, self.archer\_right.rect.left, self.archer\_right.rect.centery, -x\_velocity, y\_velocity)

        self.arrows.append(new\_arrow)

        self.turn = 'left'

## Explications:

* Calcul des Composantes de Vitesse : La vitesse en x et y est calculée en utilisant la puissance de tir et l'angle de tir.
* Création de la Flèche : Une nouvelle flèche est créée avec les paramètres calculés.
* Changement de Tour : Le tour change après chaque tir.

## Extrait de code ai\_shoot :

def ai\_shoot(self):

    if random.random() < 0.05:  # Taux de tir aléatoire

        self.shoot()

## Explications:

* Probabilité de Tir : L'IA tire avec une certaine probabilité à chaque frame.
* Tir de l'IA : La méthode shoot est appelée pour gérer le tir réel.

# Fonctionnement de l’IA

L’IA utilise la classe AIArcher pour tirer automatiquement des flèches lorsque c’est son tour. La méthode update(self) de AIArcher décide quand tirer en fonction d’un taux de probabilité défini.

## Extrait de code AIArcher :

import random

import math

from gameLogic.arrow import Arrow

class AIArcher:

    def \_\_init\_\_(self, image, x, y, screen):

        self.image = image

        self.rect = self.image.get\_rect(topleft=(x, y))

        self.screen = screen

        self.shoot\_power = 20

        self.shoot\_angle = 45

        self.previous\_shots = []

    def update(self):

        # Tirer un coup aléatoire ou basé sur la logique

        if random.random() < 0.05:  # Taux de tir aléatoire

            self.shoot()

    def shoot(self):

        angle = self.shoot\_angle

        power = self.shoot\_power

        # Tirer un coup

        x\_velocity = power \* math.cos(math.radians(angle))

        y\_velocity = -power \* math.sin(math.radians(angle))

        # Créer une flèche avec ces paramètres

        new\_arrow = Arrow(self.screen, self.rect.right, self.rect.centery, x\_velocity, y\_velocity)

        self.screen.arrows.append(new\_arrow)  # Ajouter la flèche à la liste des flèches du jeu

        # Simuler la vérification si le tir a été trop court ou trop long

        if self.previous\_shots:

            last\_shot = self.previous\_shots[-1]

            if last\_shot['distance'] > 0:  # Juste un exemple, la vraie logique dépendra des collisions

                # Ajuster la précision en fonction des résultats précédents

                if last\_shot['distance'] > 0:

                    self.shoot\_power -= 1

                elif last\_shot['distance'] < 0:

                    self.shoot\_power += 1

        # Conserver les informations du tir

        self.previous\_shots.append({'distance': random.randint(-10, 10)})  # Exemple de distance aléatoire

    def draw(self, camera\_x):

        self.screen.blit(self.image, (self.rect.x - camera\_x, self.rect.y))

## Explications:

* Probabilité de Tir : L'IA décide de tirer avec une certaine probabilité à chaque frame.
* Calcul des Composantes de Vitesse : La vitesse en x et y est calculée de manière similaire à celle du joueur.
* Ajustement Basé sur les Tirs Précédents : L'IA ajuste la puissance de tir en fonction des résultats des tirs précédents pour améliorer la précision.
* Stockage des Résultats des Tirs : Les résultats des tirs précédents sont stockés pour référence future.

# Conclusion

Pour résumer, la réalisation de ce projet a été une expérience enrichissante et stimulante. Ce projet m’a permis de mettre en pratique les compétences en programmation que j’ai acquises, tout en découvrant de nouveaux outils de gestion de fichier en Python.

En ce qui concerne l’architecture du projet, j’ai opté pour une structure de code modulaire qui favorise une meilleure organisation et facilite la maintenance, contribuant ainsi à rendre le code plus lisible et simple.

En ce qui concerne les perspectives d’amélioration, il serait intéressant d’explorer la possibilité d’introduire une fonctionnalité de reprise des parties en sauvegardant l’état du jeu dans un fichier .txt, permettant ainsi aux joueurs de reprendre là où ils s’étaient arrêtés.

En fin de compte, je suis satisfait du travail accompli, et je suis convaincu que le jeu Bow Man offre une expérience de jeu à la fois divertissante et stimulante pour les joueurs de jeux de société. Cette documentation technique fournit une vue d’ensemble complète de la structure et des fonctionnalités du jeu ‘Bow Man’. Pour des détails supplémentaires, veuillez-vous référez au code source et aux commentaires incus dans chaque fichier.

# Annexes

## Ressources

* Pygame documentation : <https://www.pygame.org/docs/>
* Python Documentation : <https://docs.python.org/3/>

## Références

* X