INSTITUT DES SCIENCES,
DES TECHNOLOGIES
ET DES ÉTUDES AVANCÉES D'HAÏTI

L'Université de la nouvelle Haïti

INF2300 Infographie

ÉTÉ 2024

Travail No. [2]

Groupe

[isteah.josephsamuel@gmail.com] – [JOSEPH Samuel Jonathan]

[isteah.jpierrelouis03@gmail.com] – [JONATHAN Pierre Louis]

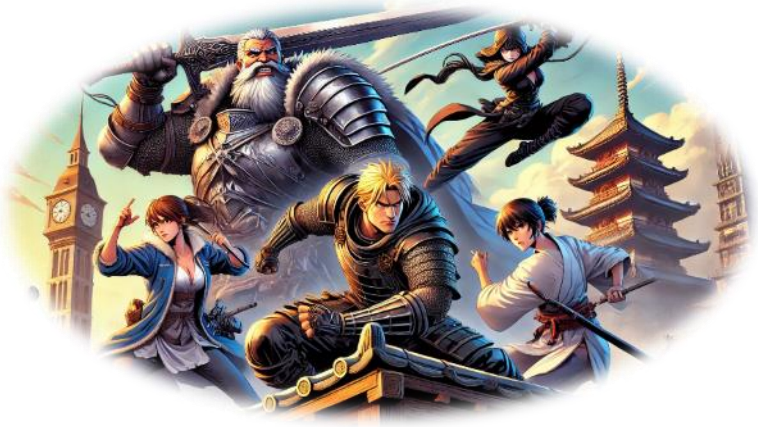
https://github.com/josephsamijona/INF2300_TP1_JOSEPH_Samuel_JONATHAN_Pierre.git

Soumis au : Dre. Franjieh El Khoury

[7/1/2024]

Sommaire

1. Introduction



« Blades of Honor: Clash of Cultures » est un jeu de combat captivant qui plonge les joueurs dans un duel intense entre guerriers et sorciers. Ce jeu se distingue par ses combats dynamiques, ses animations fluides et son gameplay stratégique. Les joueurs peuvent s'affronter en mode joueur contre joueur (1v1) ou observer des batailles entre intelligences artificielles (AI vs AI).

Le principal objectif du jeu est de réduire la santé de l'adversaire à zéro en utilisant une combinaison de mouvements, sauts et attaques. Les joueurs doivent maîtriser les commandes de leur combattant pour attaquer, esquiver et bloquer les attaques ennemies. Chaque combattant possède des capacités uniques qui peuvent être utilisées stratégiquement pour gagner un avantage sur l'adversaire.

Modes de jeu disponibles :

- **Mode 1v1** : Deux joueurs humains s'affrontent dans un combat en temps réel.
- **Mode AI vs AI** : Deux intelligences artificielles s'affrontent, permettant aux joueurs d'observer et d'analyser les stratégies des combattants contrôlés par l'IA.
- **Mode Animation** : Ce mode inclut trois types d'animations différentes, permettant d'explorer des simulations graphiques variées :
    - **Simulation de la vie** : Un environnement dynamique où les éléments interagissent de manière biologique.
    - **Pendule Double** : Une animation de pendule double, montrant les mouvements complexes de ce système physique.
    - **Cube 3D en Mouvement** : Un cube en 3D qui se déplace dans l'espace, illustrant les transformations et projections graphiques en 3D.

Contexte et inspirations

Ce projet s'inspire des jeux de combat classiques et intègre des éléments de jeux emblématiques comme Doom. Doom, avec son approche révolutionnaire du combat en 2.5D, a influencé la

conception graphique et le style d'animation de "Blades of Honor". L'objectif était de créer un jeu de combat avec une esthétique rétro tout en incorporant des mécanismes modernes de gameplay et d'intelligence artificielle.

Il a été développé dans un but éducatif pour appliquer et illustrer plusieurs concepts de programmation graphique et de développement de jeux vidéo.

Objectifs pédagogiques :

- **Programmation graphique - OpenGL en Python** : Utilisation de PyOpenGL pour créer et manipuler des objets graphiques.
- **Pipeline de transformation et pipeline graphique programmable** : Mise en œuvre des étapes du pipeline de transformation, y compris la modélisation, la visualisation et la projection des objets graphiques.
- **Opérations sur les fragments, illumination et textures** : Application des textures et techniques d'illumination pour améliorer le rendu visuel du jeu.
- **Développement d'IA pour les combattants** : Création de comportements d'intelligence artificielle pour simuler des combats réalistes entre combattants contrôlés par l'ordinateur.

L'objectif technique est de démontrer comment ces concepts peuvent être intégrés dans un projet de jeu vidéo complet, en offrant une expérience de jeu engageante tout en servant de ressource éducative pour les développeurs intéressés par la programmation graphique et le développement de jeux.

2. Installation

2.1. Prérequis

Liste des logiciels et bibliothèques nécessaires :

- **Python :** Version 3.7 ou supérieure.
- **Pygame :** Bibliothèque pour les jeux en 2D.
- **Numba :** Accélération de code Python avec LLVM.
- **PyOpenGL :** Bindings Python pour OpenGL.
- **pip :** Gestionnaire de paquets pour Python.

2.2. Instructions d'installation

Guide étape par étape pour installer les dépendances :

1. **Installer Python :** Téléchargez et installez Python depuis python.org. Assurez-vous d'ajouter Python à votre PATH pendant l'installation.
2. **Installer pip :** pip est généralement inclus avec les installations récentes de Python. Vous pouvez vérifier si pip est installé en exécutant :

```
bash
Copy code
pip --version
```

3. **Installer les dépendances :** Ouvrez un terminal ou une invite de commandes et exécutez les commandes suivantes pour installer les bibliothèques nécessaires :

```
bash
Copy code
pip install pygame
pip install numba
pip install pyopengl
```

Configuration de l'environnement de développement :

1. **Cloner le dépôt du projet :**

```
bash
Copy code
git clone
https://github.com/josephsamijona/INF2300_TP1_JOSEPH_Samuel_JONATHAN_Pi
erre.git
cd INF2300_TP2_JOSEPH_Samuel_JONATHAN_Pierre
```

2. **Configurer l'environnement virtuel (optionnel mais recommandé) :** Créez et activez un environnement virtuel pour isoler les dépendances du projet :

```
python -m venv venv
source venv/bin/activate    # Sur Windows: venv\Scripts\activate
```

3. **Installer les dépendances à partir du fichier requirements.txt :**

```
bash
Copy code
pip install -r requirements.txt
```

2.3. Dépendances

Fichier `requirements.txt` :

Le fichier `requirements.txt` doit contenir toutes les dépendances nécessaires pour exécuter le projet. Voici un exemple de ce fichier :

```
makefile
Copy code
pygame==2.1.0
numba==0.54.0
PyOpenGL==3.1.5
```

Pour installer les dépendances listées dans ce fichier, exécutez simplement :

```
pip install -r requirements.txt
```

4. Utilisation

3.1. Lancer le jeu

Pour démarrer le jeu "Blades of Honor: Clash of Cultures" depuis la ligne de commande, suivez les étapes ci-dessous :

1. **Ouvrez un terminal** :
   - Sur Windows, utilisez `cmd`, PowerShell ou Git Bash.
   - Sur macOS/Linux, utilisez le terminal intégré.
2. **Naviguez jusqu'au répertoire racine du projet** :
   - Utilisez la commande `cd` pour changer de répertoire. Par exemple :

     ```
     sh
     Copy code
     cd path/to/project_root
     ```

3. **Exécutez le script principal** :
   - Assurez-vous que tous les prérequis et les dépendances sont installés (voir section Installation).
   - Exécutez le script `main.py` avec Python :

     ```
     sh
     Copy code
     python main.py
     ```

4. **Lancez le jeu** :
   - Le jeu devrait démarrer et afficher l'écran de menu principal. À partir de là, vous pouvez naviguer dans les différents modes de jeu et paramètres.

3.2. Commandes du jeu

Voici la liste des commandes clavier pour les différents modes de jeu :

- **Mode 1v1 (Player vs Player)**
  - Joueur 1 :
    - `A` : Aller à gauche
    - `D` : Aller à droite
    - `W` : Sauter
    - `S` : Attaquer
  - Joueur 2 :
    - `Flèche gauche` : Aller à gauche
    - `Flèche droite` : Aller à droite

        - `Flèche haut` : Sauter
        - `Flèche bas` : Attaquer
- **Mode AI vs AI**
    - Aucun contrôle direct, les combattants sont contrôlés par l'IA.
- **Mode 1vAI (Player vs Computer)**
    - Joueur :
        - `A` : Aller à gauche
        - `D` : Aller à droite
        - `W` : Sauter
        - `S` : Attaquer
    - IA :
        - Contrôle automatique basé sur les algorithmes d'intelligence artificielle intégrés.
- **Mode 3D Adventure**
    - `W` : Avancer
    - `S` : Reculer
    - `A` : Déplacer à gauche
    - `D` : Déplacer à droite
    - `Espace` : Sauter
    - `Clique gauche` : Attaquer

3.3. Interface utilisateur

L'interface utilisateur de "Blades of Honor: Clash of Cultures" est conçue pour être intuitive et informer le joueur de l'état actuel du jeu. Voici les principaux éléments de l'interface :

- **Barres de santé** :
    - Situées en haut de l'écran pour chaque joueur.
    - Affichent la santé actuelle sous forme de barre colorée.
    - Rouge pour la barre de fond, jaune pour la santé restante.
- **Compte à rebours** :
    - Affiché au début de chaque round.
    - Indique le temps restant avant le début du combat.
    - S'affiche au centre de l'écran.
- **Score** :
    - Affiché en haut à gauche pour le joueur 1 et en haut à droite pour le joueur 2.
    - Indique le nombre de rounds gagnés par chaque joueur.
- **Messages de victoire/défaite** :
    - Affichés au centre de l'écran à la fin de chaque round.
    - Indiquent quel joueur a gagné le round ou si c'est une égalité.
- **Chronomètre du round** :
    - Affiché en haut au centre de l'écran.
    - Indique le temps écoulé depuis le début du round.

4. Détails Techniques

4.1. Structure du projet

La structure du projet est organisée de manière à séparer les ressources du jeu, la logique du jeu, et les scripts principaux. Voici un aperçu de la structure des répertoires et des fichiers principaux :project_root/

```
│
├── game/
│   ├── assets/
│   │   ├── audio/
│   │   │   ├── sword.wav
│   │   │   ├── magic.wav
│   │   │   └── ...
│   │   ├── backgrounds/
│   │   │   ├── moon.png
│   │   │   └── ...
│   │   ├── fonts/
│   │   │   └── Seagram tfb.ttf
│   │   ├── icons/
│   │   │   └── victory.png
│   │   ├── music/
│   │   │   └── LEMMiNO - Cipher (BGM).mp3
│   │   ├── sprites/
│   │   │   ├── warrior.png
│   │   │   ├── wizard.png
│   │   │   └── ...
│   │   └── textures/
│   │       ├── floor.jpg
│   │       ├── skybox2.jpg
│   │       └── wall.jpg
│   └── sounds/
│       ├── playerstep.mp3
│       ├── enemystep.mp3
│       ├── sword.mp3
│       ├── damage.mp3
│       ├── deadmonster.mp3
│       ├── hitmonster.mp3
│       ├── hitmonster2.mp3
│       ├── healthup.wav
│       ├── died.wav
│       ├── won.wav
│       └── battlemusic0.mp3
│
├── src/
│   ├── game_logic/
│   │   ├── aivai/
│   │   │   └── mode_ai_vs_ai.py
│   │   ├── trid/
│   │   │   └── mode_3d_adventure.py
│   │   ├── unvdeux/
│   │   │   └── mode_1_vs_2.py
│   │   └── vcomputer/
│   │       └── mode_1_vs_computer.py
```

```
│   ├── credits.py
│   ├── game_modes.py
│   ├── menu.py
│   └── settings.py
│
├── main.py
└── README.md
```

- **game/** : Contient les ressources du jeu, telles que les fichiers audio, les images de fond, les polices, les icônes, la musique, les sprites et les textures.
- **src/** : Contient la logique du jeu répartie en différents modules pour chaque mode de jeu ainsi que les scripts pour les paramètres, le menu, et les crédits.
- **main.py** : Le script principal pour lancer le jeu.
- **README.md** : Documentation du projet.

4.2. Architecture du code

L'architecture du code est modulaire, avec une séparation claire des responsabilités entre les différents modules et classes. Voici un aperçu des classes et modules principaux :

- **Classes principales** :
  - `Gameaivai` : Gère le mode de jeu AI vs AI.
  - `Fighter` : Représente un combattant dans le jeu, avec ses actions et animations.
- **Modules** :
  - `aivai/mode_ai_vs_ai.py` : Contient la logique du mode de jeu AI vs AI.
  - `trid/mode_3d_adventure.py` : Contient la logique du mode de jeu 3D Adventure.
  - `unvdeux/mode_1_vs_2.py` : Contient la logique du mode de jeu 1 vs 2.
  - `vcomputer/mode_1_vs_computer.py` : Contient la logique du mode de jeu 1 vs Computer.
  - `credits.py` : Affiche les crédits du jeu.
  - `game_modes.py` : Gère les différents modes de jeu.
  - `menu.py` : Gère le menu principal du jeu.
  - `settings.py` : Gère les paramètres du jeu.

5. Programmation Graphique

5.1. Concepts de base

Dans ce projet, nous utilisons principalement Pygame pour la gestion des graphismes et des événements. Pygame est une bibliothèque Python conçue pour écrire des jeux vidéo en utilisant des images et des sons. OpenGL pourrait être utilisé pour des rendus 3D avancés, mais pour ce projet, nous nous concentrons sur la 2D avec des éléments de simulation 3D.

- **Pygame** : Utilisé pour charger et afficher des images, lire des sons, gérer les événements clavier et souris, et créer une boucle de jeu.
- **NumPy** : Utilisé pour des calculs numériques efficaces, en particulier pour les transformations et les manipulations d'images.

5.2. Pipeline de transformation

Le pipeline de transformation est une série d'étapes qui prennent les coordonnées d'objets dans un espace de jeu et les transforment en coordonnées d'écran afin qu'ils puissent être rendus. Dans ce projet, les transformations incluent la translation, la rotation et l'échelle.

- **Translation** : Déplacement des objets dans l'espace de jeu.
- **Rotation** : Orientation des objets dans l'espace de jeu.
- **Échelle** : Ajustement de la taille des objets pour simuler la profondeur.

Voici un exemple de pipeline de transformation pour un personnage dans le jeu :

```
python
Copy code
def move(self, screen_width, screen_height, surface, target, round_over):
    SPEED = 10
    GRAVITY = 2
    dx = 0
    dy = 0
    self.running = False
    self.attack_type = 0

    # Appliquer les transformations basées sur l'input ou l'IA
    if self.attacking == False and self.alive == True and round_over ==
False:
        # Logique de mouvement et d'attaque
        if self.rect.centerx < target.rect.centerx:
            dx = SPEED
            self.running = True
        elif self.rect.centerx > target.rect.centerx:
            dx = -SPEED
            self.running = True
        if abs(self.rect.centerx - target.rect.centerx) < 100:
            self.attack(surface, target)
            self.attack_type = 1
        if self.jump == False and random.randint(0, 100) < 5:
            self.vel_y = -30
            self.jump = True

    # Appliquer la gravité
    self.vel_y += GRAVITY
    dy += self.vel_y

    # Mettre à jour la position en appliquant les transformations
    self.rect.x += dx
    self.rect.y += dy

    # Mettre à jour l'image du combattant
```

```
    self.update()
```

5.3. Pipeline graphique programmable

Dans les jeux modernes, le pipeline graphique programmable permet d'utiliser des shaders pour des effets visuels avancés. Les shaders sont des programmes exécutés sur le GPU pour manipuler les graphismes.

- **Vertex Shader** : Transforme les positions des sommets dans l'espace de jeu en positions d'écran.
- **Fragment Shader** : Calcule la couleur des pixels, en appliquant des effets d'éclairage et des textures.

Dans ce projet, nous utilisons des techniques de base pour simuler certains de ces effets. Par exemple, la manipulation des textures et l'application de l'éclairage pour donner l'illusion de profondeur.

5.4. Opérations sur les fragments, illumination et textures

Les opérations sur les fragments incluent l'application des textures, l'éclairage et les effets visuels pour améliorer l'apparence des objets dans le jeu.

- **Textures** : Les textures sont des images appliquées aux surfaces des objets pour leur donner une apparence détaillée. Dans ce projet, nous utilisons des textures pour les personnages, les arrière-plans et d'autres éléments visuels.

```
python
Copy code
def load_images(self, sprite_sheet, animation_steps):
    animation_list = []
    for y, animation in enumerate(animation_steps):
        temp_img_list = []
        for x in range(animation):
            temp_img = sprite_sheet.subsurface(x * self.size, y * self.size,
self.size, self.size)
            temp_img_list.append(pygame.transform.scale(temp_img, (self.size
* self.image_scale, self.size * self.image_scale)))
        animation_list.append(temp_img_list)
    return animation_list
```

- **Éclairage** : Simule l'effet de la lumière sur les surfaces des objets. Dans ce projet, nous utilisons des techniques simples pour simuler l'éclairage en ajustant la luminosité des textures en fonction de leur position et de l'angle de la lumière.
- **Effets visuels** : Incluent des effets comme les ombres, les reflets et les particules pour améliorer l'apparence visuelle du jeu. Par exemple, nous utilisons des images de fond et des éléments animés pour donner de la vie à la scène de jeu.

5.5. Effet Parallax

L'effet parallax est utilisé pour créer une illusion de profondeur dans le background du mode combat. Cet effet est obtenu en faisant défiler plusieurs couches de background à des vitesses différentes.

- **Principe** : Les objets au premier plan se déplacent plus rapidement que ceux à l'arrière-plan, créant ainsi une impression de profondeur.
- **Implémentation** : Utilisation de plusieurs couches d'images de background qui se déplacent à des vitesses différentes pour simuler la profondeur.

Dans le cadre de ce projet, chaque aspect de la programmation graphique mentionné revêt une pertinence directe pour les trois modes de jeu spécifiques : la Simulation de la vie, le Pendule Double et le Cube 3D en Mouvement.

- **Programmation graphique - OpenGL en Python :** OpenGL est une bibliothèque graphique puissante qui permet de créer des rendus 2D et 3D complexes en utilisant Python. Pour les trois modes de jeu mentionnés :

  - **Simulation de la vie :** OpenGL en Python est utilisé pour créer des simulations biologiques dynamiques en 2D. Il permet de gérer l'affichage des entités biologiques simulées à l'écran, en utilisant des techniques de rendu adaptées aux simulations basées sur des automates cellulaires ou des algorithmes de simulation biologique.
  - **Pendule Double :** Bien que ce mode soit principalement en 2D, OpenGL en Python peut être utilisé pour optimiser les performances d'affichage des animations du pendule double. Il permet de gérer efficacement le rendu des lignes représentant les pendules et d'appliquer des techniques de rendu 2D avancées pour améliorer la précision visuelle.
  - **Cube 3D en Mouvement :** OpenGL est essentiel pour la modélisation et le rendu 3D du cube. Il permet de créer un environnement 3D réaliste où le cube peut se déplacer et interagir avec son environnement. OpenGL en Python facilite l'utilisation de shaders pour appliquer des effets d'éclairage, de textures et d'autres effets visuels avancés qui enrichissent l'expérience du joueur.

- **Pipeline de transformation et pipeline graphique programmable :** Le pipeline de transformation est crucial pour tous les modes de jeu car il définit les étapes par lesquelles les objets virtuels sont transformés de leur position et forme dans l'espace de jeu vers leur rendu à l'écran :

  - **Simulation de la vie :** Le pipeline de transformation est utilisé pour positionner et animer les entités biologiques à l'écran, en utilisant des transformations comme la translation et la rotation pour simuler leur mouvement et leur interaction.
  - **Pendule Double :** Ce mode utilise le pipeline de transformation pour calculer les positions des pendules en fonction de leurs angles et de leurs vitesses angulaires. Les transformations graphiques comme la translation sont utilisées pour déplacer les pendules à chaque itération temporelle.
  - **Cube 3D en Mouvement :** Pour ce mode, le pipeline de transformation gère la projection perspective du cube dans l'espace 3D virtuel et sa conversion en coordonnées 2D pour

l'affichage à l'écran. Les transformations comme la rotation et l'échelle sont appliquées pour simuler le mouvement du cube dans l'espace.

- **Opérations sur les fragments, illumination et textures :** Ces concepts sont appliqués pour améliorer la qualité visuelle et le réalisme des rendus dans chaque mode de jeu :

  - **Simulation de la vie :** Les textures peuvent être utilisées pour représenter visuellement les entités biologiques simulées, tandis que l'illumination simule l'effet de la lumière sur leur apparence. Les opérations sur les fragments permettent de gérer les interactions lumineuses et les effets visuels des entités biologiques.
  - **Pendule Double :** Bien que principalement en 2D, l'application de techniques d'illumination et de textures peut améliorer la représentation visuelle des pendules et de leur environnement, rendant les animations plus réalistes et engageantes.
  - **Cube 3D en Mouvement :** OpenGL permet d'appliquer des textures détaillées sur le cube, de simuler des effets d'ombre et de lumière grâce aux opérations sur les fragments, et d'utiliser des shaders pour des effets visuels avancés comme l'éclairage dynamique et les réflexions. Ces techniques enrichissent l'apparence du cube et de son environnement 3D.

6. Intelligence Artificielle

6.1. IA des combattants

L'intelligence artificielle (IA) dans "Blades of Honor: Clash of Cultures" est conçue pour rendre les combats dynamiques et intéressants. Les combattants contrôlés par l'IA utilisent divers algorithmes et logiques pour simuler des comportements humains tels que le mouvement, l'attaque, et la défense.

Algorithmes et logiques utilisés pour l'IA des combattants

- **Détection de la position du joueur** : Les combattants IA déterminent la position de leur adversaire pour ajuster leur mouvement.
- **Gestion des états** : L'IA des combattants utilise une machine à états pour gérer différentes actions comme courir, sauter, attaquer et défendre.
- **Logique de décision** : Les combattants IA prennent des décisions basées sur des règles prédéfinies et des probabilités pour rendre leurs actions moins prévisibles.

Voici un extrait de code illustrant la logique de mouvement et d'attaque de l'IA :

```
python
Copy code
def move(self, screen_width, screen_height, surface, target, round_over):
    SPEED = 10
    GRAVITY = 2
    dx = 0
    dy = 0
    self.running = False
    self.attack_type = 0
```

```
    # Ne peut pas bouger pendant l'attaque
    if self.attacking == False and self.alive == True and round_over ==
False:
        # Logique de l'IA pour les deux combattants
        if self.rect.centerx < target.rect.centerx:
            dx = SPEED
            self.running = True
        elif self.rect.centerx > target.rect.centerx:
            dx = -SPEED
            self.running = True

        # Déterminer l'attaque de l'IA
        if abs(self.rect.centerx - target.rect.centerx) < 100:
            self.attack(surface, target)
            self.attack_type = 1

        # Faire sauter l'IA de temps en temps
        if self.jump == False and random.randint(0, 100) < 5:
            self.vel_y = -30
            self.jump = True

    # Appliquer la gravité
    self.vel_y += GRAVITY
    dy += self.vel_y

    # S'assurer que le combattant reste dans les limites de l'écran
    if self.rect.left + dx < 0:
        dx = -self.rect.left
    if self.rect.right + dx > screen_width:
        dx = screen_width - self.rect.right
    if self.rect.bottom + dy > screen_height - 110:
        self.vel_y = 0
        self.jump = False
        dy = screen_height - 110 - self.rect.bottom

    # Mettre à jour la position du rectangle
    self.rect.x += dx
    self.rect.y += dy

    # Mettre à jour l'image du combattant
    self.update()
```

6.2. Comportements de l'IA

Les combattants IA dans le jeu affichent une variété de comportements pour rendre les combats plus intéressants et réalistes. Voici une description de ces comportements :

Mouvement

Les combattants IA ajustent constamment leur position en fonction de celle de leur adversaire. Ils peuvent se déplacer vers l'avant, vers l'arrière, et sauter pour éviter les attaques.

Attaque

Lorsqu'un combattant IA est à portée d'attaque, il lance une attaque contre son adversaire. L'IA décide du type d'attaque en fonction de la distance et des conditions du combat.

Défense

Bien que le code fourni ne montre pas de comportements défensifs avancés, un combattant IA pourrait être programmé pour bloquer ou esquiver les attaques en fonction des actions de l'adversaire.

Aléatoire et imprévisibilité

Pour rendre les combats moins prévisibles, des éléments de hasard sont introduits. Par exemple, la décision de sauter ou d'attaquer à un moment donné peut être influencée par un générateur de nombres aléatoires.

Voici un extrait de code illustrant la mise à jour des actions et l'animation en fonction de l'état du combattant :

```
python
Copy code
def update(self):
    # Vérifier l'action en cours
    if self.health <= 0:
        self.health = 0
        self.alive = False
        self.update_action(5)  # Mort
    elif self.attacking == True:
        self.update_action(3)  # Attaque
    elif self.jump == True:
        self.update_action(2)  # Saut
    elif self.running == True:
        self.update_action(1)  # Courir
    else:
        self.update_action(0)  # Inactif

    animation_cooldown = 50
    # Mettre à jour l'image
    self.image = self.animation_list[self.action][self.frame_index]
    # Vérifier si assez de temps s'est écoulé depuis la dernière mise à jour
    if pygame.time.get_ticks() - self.update_time > animation_cooldown:
        self.update_time = pygame.time.get_ticks()
        self.frame_index += 1
    # Vérifier si l'animation est terminée
    if self.frame_index >= len(self.animation_list[self.action]):
        # Si le combattant est mort, terminer l'animation
        if self.alive == False:
            self.frame_index = len(self.animation_list[self.action]) - 1
        else:
            self.frame_index = 0
            # Vérifier si une attaque a été exécutée
            if self.action == 3:
                self.attacking = False
                self.attack_cooldown = 20
```

Ces comportements permettent de créer des combats dynamiques et engageants, où les joueurs doivent constamment ajuster leurs stratégies pour vaincre leurs adversaires contrôlés par l'IA.

7. Assets et Ressources

7.1. Gestion des ressources

La gestion des ressources dans "Blades of Honor: Clash of Cultures" est cruciale pour garantir une performance optimale et une expérience utilisateur fluide. Les ressources incluent les images, sons, musiques, et autres fichiers nécessaires au bon fonctionnement du jeu.

Explication de la gestion et du chargement des ressources

1. **Organisation des fichiers** : Les ressources sont organisées dans des répertoires spécifiques sous le dossier `game/assets`. Cette structure facilite la gestion et le chargement des ressources nécessaires.
    - **audio/** : Contient les effets sonores.
    - **backgrounds/** : Contient les images de fond.
    - **fonts/** : Contient les polices de caractères.
    - **icons/** : Contient les icônes de victoire et de défaite.
    - **music/** : Contient les musiques de fond.
    - **sprites/** : Contient les sprites des personnages.
    - **textures/** : Contient les textures pour le sol, le mur et le ciel.
2. **Chargement des ressources** : Le chargement des ressources se fait au démarrage du jeu pour s'assurer que toutes les ressources nécessaires sont disponibles en mémoire avant le début du jeu. Voici un exemple de chargement des ressources :

```
python
Copy code
# Charger les sons
self.sword_fx = pygame.mixer.Sound(os.path.join(self.root_dir, "..", "game",
"assets", "audio", "sword.wav"))
self.sword_fx.set_volume(0.5)
self.magic_fx = pygame.mixer.Sound(os.path.join(self.root_dir, "..", "game",
"assets", "audio", "magic.wav"))
self.magic_fx.set_volume(0.75)

# Charger l'image de fond
self.bg_image = pygame.image.load(os.path.join(self.root_dir, "..", "game",
"assets", "backgrounds", "moon.png")).convert_alpha()

# Charger les feuilles de sprites
self.warrior_sheet = pygame.image.load(os.path.join(self.root_dir, "..",
"game", "assets", "characters", "warrior", "Sprites",
"warrior.png")).convert_alpha()
self.wizard_sheet = pygame.image.load(os.path.join(self.root_dir, "..",
"game", "assets", "characters", "wizard", "Sprites",
"wizard.png")).convert_alpha()
```

3. **Gestion des chemins d'accès** : Pour simplifier le chargement des ressources, des fonctions utilitaires sont utilisées pour construire les chemins d'accès relatifs :

```
python
Copy code
def get_asset_path(self, relative_path):
    return os.path.join(self.root_dir, relative_path)
```

7.2. Création et intégration des assets

La création et l'intégration des assets graphiques et sonores sont des étapes essentielles pour le développement d'un jeu immersif et engageant.

Processus de création des sprites, textures et autres assets graphiques

1. **Création des assets** : Les assets graphiques, comme les sprites et les textures, sont généralement créés à l'aide de logiciels de graphisme tels que Photoshop, GIMP, ou des outils de dessin vectoriel comme Illustrator. Les assets sonores sont créés ou modifiés à l'aide de logiciels d'édition audio comme Audacity.
2. **Intégration des assets** : Une fois créés, les assets doivent être intégrés dans le jeu. Cela inclut le découpage des feuilles de sprites en images individuelles, la conversion des fichiers dans des formats compatibles avec Pygame (comme PNG pour les images et WAV pour les sons), et l'ajustement des dimensions pour s'adapter aux besoins du jeu.

Voici un exemple de la manière dont les sprites sont chargés et transformés pour une utilisation dans le jeu :

```
python
Copy code
def load_images(self, sprite_sheet, animation_steps):
    animation_list = []
    for y, animation in enumerate(animation_steps):
        temp_img_list = []
        for x in range(animation):
            temp_img = sprite_sheet.subsurface(x * self.size, y * self.size,
self.size, self.size)
            temp_img_list.append(pygame.transform.scale(temp_img, (self.size
* self.image_scale, self.size * self.image_scale)))
        animation_list.append(temp_img_list)
    return animation_list
```

3. **Gestion des animations** : Les animations des personnages sont gérées en chargeant une série d'images (sprites) et en les affichant successivement pour donner l'impression de mouvement. Chaque action (comme courir, sauter, attaquer) a son propre ensemble d'images d'animation.

Voici un extrait de code qui gère l'animation d'un combattant :

```
python
Copy code
```

```
def update(self):
    # Vérifier l'action en cours
    if self.health <= 0:
        self.health = 0
        self.alive = False
        self.update_action(5)  # Mort
    elif self.attacking == True:
        self.update_action(3)  # Attaque
    elif self.jump == True:
        self.update_action(2)  # Saut
    elif self.running == True:
        self.update_action(1)  # Courir
    else:
        self.update_action(0)  # Inactif

    animation_cooldown = 50
    # Mettre à jour l'image
    self.image = self.animation_list[self.action][self.frame_index]
    # Vérifier si assez de temps s'est écoulé depuis la dernière mise à jour
    if pygame.time.get_ticks() - self.update_time > animation_cooldown:
        self.update_time = pygame.time.get_ticks()
        self.frame_index += 1
    # Vérifier si l'animation est terminée
    if self.frame_index >= len(self.animation_list[self.action]):
        # Si le combattant est mort, terminer l'animation
        if self.alive == False:
            self.frame_index = len(self.animation_list[self.action]) - 1
        else:
            self.frame_index = 0
            # Vérifier si une attaque a été exécutée
            if self.action == 3:
                self.attacking = False
                self.attack_cooldown = 20
```

8. Développement Futur

8.1. Fonctionnalités à ajouter

Pour les futures versions de "Blades of Honor: Clash of Cultures", plusieurs fonctionnalités sont envisagées afin d'améliorer l'expérience de jeu et d'offrir plus de diversité aux joueurs.

Liste des fonctionnalités prévues pour les futures versions

1. **Mode Multijoueur en Ligne** :
    - Implémenter un mode multijoueur en ligne permettant à des joueurs du monde entier de s'affronter en temps réel.
2. **Nouveaux Personnages Jouables** :
    - Ajouter de nouveaux personnages avec des compétences et des styles de combat uniques pour diversifier le gameplay.
3. **Système de Compétences et d'Améliorations** :

    - Introduire un système de progression où les joueurs peuvent améliorer leurs personnages et débloquer de nouvelles compétences en gagnant des combats.
4. **Modes de Jeu Supplémentaires** :
    - Développer des modes de jeu supplémentaires comme le mode "Survival" où les joueurs affrontent des vagues d'ennemis de plus en plus difficiles.
5. **Environnements et Arènes Diversifiés** :
    - Ajouter de nouveaux environnements et arènes de combat avec des caractéristiques uniques influençant la stratégie des combats.
6. **Scénarios et Quêtes** :
    - Introduire des scénarios et des quêtes pour offrir un mode histoire, permettant aux joueurs de s'immerger davantage dans l'univers du jeu.
7. **Compatibilité avec les Manettes de Jeu** :
    - Ajouter le support des manettes de jeu pour une expérience de jeu plus fluide et immersive.

8.2. Améliorations potentielles

En plus des nouvelles fonctionnalités, plusieurs améliorations peuvent être apportées au jeu et à son code pour en optimiser les performances et la jouabilité.

Suggestions pour améliorer le jeu et son code

1. **Optimisation des Performances** :
    - Réduire l'utilisation des ressources en optimisant les boucles de rendu et en utilisant des techniques de culling pour ne rendre que ce qui est visible à l'écran.
2. **Amélioration de l'IA** :
    - Développer des algorithmes d'IA plus avancés pour rendre les ennemis plus intelligents et offrir des défis plus variés et imprévisibles.
3. **Refactorisation du Code** :
    - Effectuer une refactorisation régulière du code pour améliorer sa lisibilité et sa maintenabilité, en suivant les meilleures pratiques de programmation.
4. **Tests Automatisés** :
    - Mettre en place des suites de tests automatisés pour couvrir plus de cas d'utilisation et détecter les régressions plus rapidement.
5. **Amélioration de l'Interface Utilisateur** :
    - Retravailler l'interface utilisateur pour la rendre plus intuitive et agréable à utiliser, en ajoutant des animations et des effets visuels.
6. **Support Multilingue** :
    - Ajouter des options de langues multiples pour rendre le jeu accessible à un public international plus large.
7. **Documentation Complète** :
    - Étoffer la documentation technique et utilisateur pour faciliter la prise en main du projet par de nouveaux développeurs et utilisateurs.

En intégrant ces fonctionnalités et améliorations, "Blades of Honor: Clash of Cultures" pourra offrir une expérience de jeu encore plus riche et captivante, tout en facilitant la gestion et l'extension du code.

9. Exploration et Expansion Futures

9.1. Nouveaux Horizons

Dans le cadre de notre ambition à long terme, nous envisageons d'explorer de nouveaux horizons pour "Blades of Honor: Clash of Cultures". Nous aimerions continuer à travailler sur ce jeu pour le porter à un niveau supérieur et éventuellement le publier officiellement sur diverses plateformes. Voici les étapes envisagées :

1. **Publication sur les Consoles de Jeu** :
    - Adapter le jeu pour qu'il soit compatible avec les principales consoles de jeu comme PlayStation, Xbox et Nintendo Switch.
2. **Portage sur Téléphone Mobile** :
    - Optimiser le jeu pour les plateformes mobiles, y compris iOS et Android, afin d'atteindre un public plus large.
3. **Version pour Ordinateurs** :
    - Finaliser et publier une version stable et optimisée pour les ordinateurs personnels sous Windows, macOS et Linux.
4. **Marketplace et Distribution** :
    - Explorer les options de distribution via des marketplaces en ligne comme Steam, Google Play Store, Apple App Store et les boutiques en ligne des consoles de jeu.
5. **Marketing et Promotion** :
    - Développer une stratégie de marketing et de promotion pour accroître la visibilité du jeu et atteindre une audience mondiale.

En réalisant ces objectifs, nous espérons offrir "Blades of Honor: Clash of Cultures" à un public plus vaste et diversifié, tout en continuant à améliorer et à enrichir l'expérience de jeu.

9. Annexes

Liste des Documentations Utilisées

Pour la réalisation de "Blades of Honor: Clash of Cultures", plusieurs ressources ont été consultées pour s'inspirer des concepts de jeu, des techniques de programmation et des bonnes pratiques en matière de développement de jeux vidéo. Voici une liste des documentations et sources d'inspiration utilisées :

1. Brawler - Jeu de Combat 2 Joueurs

- **Titre :** Brawler: A two player fighting game based on Street Fighter.
- **Description :** Ce jeu est basé sur les anciens jeux de combat arcade où deux joueurs s'affrontent. Il inclut des contrôles indépendants pour chaque joueur, chacun ayant ses propres attaques. Il y a deux types d'attaques que chaque joueur peut exécuter, avec

différentes animations d'attaque. Ce jeu peut être étendu pour ajouter plus de personnages ainsi que plus de types d'attaques.
- **Tutoriel Vidéo :** Brawler Tutorial
- **Source Code :** GitHub - Brawler
- **Contrôles :**
  - **Joueur 1 :** A et D pour se déplacer, W pour sauter, Q et E pour attaquer.
  - **Joueur 2 :** Flèches directionnelles pour se déplacer et sauter, Numpad 1 & 2 pour attaquer.

2. Ray Casting in Python - Mode 3D

- **Titre :** Ray casting in Python
- **Description :** Cette vidéo montre comment convertir un script de "Dead And" en un jeu de tir à la première personne (First Person Shooter - FPS) en utilisant le dernier script disponible sur GitHub comme base. Le résultat final nécessite encore des ajustements au niveau du gameplay.
- **Tutoriel Vidéo :** Ray Casting Tutorial
- **Executable :** Dead And - Itch.io
- **Source Code :** GitHub - Ray Casting

3. Documentation des Frameworks et Librairies Utilisés

- **Pygame**
  - **Description :** Pygame est une bibliothèque en Python qui permet de créer des jeux vidéo. Elle fournit des fonctionnalités telles que le contrôle des images, des sons et des événements, ainsi que la gestion des collisions et des animations.
  - **Documentation Officielle :** Pygame Documentation
  - **Tutoriels :** Pygame Tutorials
- **Numba**
  - **Description :** Numba est un compilateur JIT (Just-In-Time) pour Python qui permet de rendre les fonctions Python très rapides en utilisant LLVM. Il est utilisé pour accélérer les calculs numériques et est particulièrement utile pour les applications de traitement de données et de calcul scientifique.
  - **Documentation Officielle :** Numba Documentation
  - **Tutoriels :** Numba Tutorials

4. Autres Ressources de Documentation

- **Python**
  - **Documentation Officielle :** Python Documentation
  - **Tutoriels :** Python Tutorials
- **GitHub**
  - **Description :** GitHub est une plateforme de développement collaboratif qui permet d'héberger des projets, de collaborer avec d'autres développeurs et de suivre les modifications de code.
  - **Documentation Officielle :** GitHub Documentation
  - **Tutoriels :** GitHub Guides

5. Livres et Articles

- **Game Programming Patterns** par Robert Nystrom
  - **Description :** Ce livre explore des modèles de conception (design patterns) spécifiques aux jeux vidéo, offrant des solutions aux problèmes courants rencontrés lors du développement de jeux.
  - **Lien :** Game Programming Patterns
- **Real-Time Rendering** par Tomas Akenine-Möller, Eric Haines, et Naty Hoffman
  - **Description :** Ce livre est une référence complète sur le rendu en temps réel, couvrant des sujets tels que l'éclairage, les ombres, les shaders, et les techniques de rendu avancées.
  - **Lien :** Real-Time Rendering

Ces ressources ont été cruciales pour comprendre les différentes techniques de développement de jeux vidéo, de la gestion des sprites et animations dans un jeu de combat à l'implémentation du ray casting pour créer un environnement 3D immersif. Elles ont permis d'accélérer le processus de création du jeu, en se basant sur des techniques éprouvées et en les adaptant pour créer un produit final unique et captivant. Ces documentations resteront une référence précieuse pour les futures itérations et améliorations du jeu.

Glossaire

1. Animation

- **Définition :** Technique consistant à créer l'illusion de mouvement en affichant une séquence d'images statiques de manière rapide et successive.
- **Application :** Utilisée pour animer les personnages et les objets du jeu.

2. API (Application Programming Interface)

- **Définition :** Ensemble de fonctions et de protocoles permettant à des applications logicielles de communiquer entre elles.
- **Application :** Pygame fournit une API pour la création et la gestion des jeux vidéo.

3. Assets

- **Définition :** Tous les éléments visuels, audio et autres éléments de contenu utilisés dans un jeu.
- **Application :** Inclut les sprites, textures, sons, et musiques utilisés dans "Blades of Honor: Clash of Cultures".

4. Bibliothèque

- **Définition :** Ensemble de fonctions et de routines regroupées en un seul package pour être utilisées par des programmes.
- **Application :** Pygame et Numba sont des bibliothèques utilisées dans le projet.

5. Classe

- **Définition :** Modèle définissant des objets en encapsulant des données et des fonctions dans une seule unité.
- **Application :** La classe `Fighter` définit les combattants dans le jeu.

6. Collision Detection

- **Définition :** Processus de détection du moment où deux ou plusieurs objets entrent en contact dans un jeu.
- **Application :** Utilisée pour déterminer quand un combattant frappe un autre dans le jeu.

7. Compilation JIT (Just-In-Time)

- **Définition :** Technique de compilation qui traduit le code en bytecode ou en code machine au moment de l'exécution plutôt qu'à l'avance.
- **Application :** Numba utilise la compilation JIT pour accélérer les calculs numériques.

8. Frame Rate (Fréquence d'Images)

- **Définition :** Nombre d'images affichées par seconde dans une animation ou un jeu vidéo.
- **Application :** Le jeu est configuré pour fonctionner à une fréquence de 60 FPS (Frames Per Second).

9. GitHub

- **Définition :** Plateforme de développement collaboratif hébergeant des projets de programmation et permettant le contrôle de version.
- **Application :** Utilisée pour héberger le code source du projet et collaborer avec d'autres développeurs.

10. IA (Intelligence Artificielle)

- **Définition :** Simulation de processus d'intelligence humaine par des machines, notamment par des systèmes informatiques.
- **Application :** Utilisée pour contrôler les comportements des combattants dans les modes AI vs AI.

11. Module

- **Définition :** Unité de code source en Python, qui peut contenir des fonctions, des classes, et des variables.

- **Application :** Le code du projet est organisé en différents modules, comme `mode_ai_vs_ai.py`, `mode_3d_adventure.py`, etc.

12. Numba

- **Définition :** Bibliothèque Python pour la compilation JIT qui accélère les calculs numériques.
- **Application :** Utilisée pour améliorer les performances des fonctions critiques du jeu.

13. Pygame

- **Définition :** Bibliothèque Python permettant de créer des jeux vidéo et des applications multimédia.
- **Application :** Utilisée pour la gestion des graphismes, des sons et des événements dans le jeu.

14. Pipeline de Transformation

- **Définition :** Série d'étapes de transformation des coordonnées des objets pour les afficher correctement sur l'écran.
- **Application :** Implémenté dans le jeu pour gérer les animations et les mouvements des personnages.

15. Pipeline Graphique Programmable

- **Définition :** Utilisation de shaders et d'autres techniques avancées pour le rendu graphique.
- **Application :** Techniques avancées de rendu utilisées dans les modes de jeu 3D.

16. Ray Casting

- **Définition :** Technique de rendu utilisée pour détecter les objets en projetant des "rayons" depuis un point de vue.
- **Application :** Utilisée pour créer l'effet de perspective en 3D dans le mode aventure.

17. Sprite

- **Définition :** Image 2D ou animation intégrée dans une scène plus grande.
- **Application :** Utilisés pour représenter les combattants, les objets et autres éléments du jeu.

18. Shader

- **Définition :** Programme exécuté sur le GPU pour déterminer l'apparence des pixels ou des sommets.

- **Application :** Utilisés dans le pipeline graphique programmable pour les effets visuels avancés.

19. Texture

- **Définition :** Image appliquée à une surface d'un objet 3D pour lui donner une apparence détaillée.
- **Application :** Utilisées pour les murs, le sol, et autres surfaces dans le jeu.

20. Transformation

- **Définition :** Opération mathématique appliquée aux coordonnées des objets pour les déplacer, les faire pivoter ou les redimensionner.
- **Application :** Utilisée pour animer les mouvements des combattants et des objets dans le jeu.