Shooter Spatial

Projet de Paradigmes de Programmation

Licence Informatique

Paradigmes : Orienté Objet, Procédural, Événementiel, Concurrent

Langage : Python 3.7+

Technologies : tkinter, threading, pygame, JSON, HTML/CSS

Réalisé par

Valentin Lesnes

Table des matières

1 Introduction **3**
1.1 Contexte du projet . . . 3
1.2 Choix du jeu : Shooter Spatial . . . 3
1.3 Objectifs pédagogiques . . . 3

2 Description du Gameplay **3**
2.1 Règles du jeu . . . 3
2.2 Conditions de victoire et défaite . . . 3

3 Conception Logicielle **4**
3.1 Architecture générale . . . 4
3.2 Diagramme de classes . . . 4
3.3 Classes principales . . . 4
3.3.1 ObjetVolant (classe abstraite) . . . 4
3.3.2 Vaisseau . . . 5
3.3.3 Ennemi . . . 5
3.3.4 Projectile . . . 5
3.3.5 Bonus . . . 5
3.3.6 GameEngine . . . 5
3.3.7 ScoreManager . . . 6

4 Implémentation des Paradigmes **6**
4.1 Programmation Orientée Objet (POO) . . . 6
4.2 Système de Bonus . . . 6
4.3 Programmation Procédurale . . . 7
4.4 Programmation Événementielle . . . 8
4.5 Programmation Concurrente (Threads) . . . 8

5 Gestion des Scores **10**
5.1 Format de sauvegarde JSON . . . 10
5.2 Classe ScoreManager . . . 10

6 Interface Graphique **11**
6.1 Choix technologique : tkinter . . . 11
6.2 Architecture de l'interface . . . 11
6.3 Gestion des événements . . . 12
6.4 Boucle de rendu . . . 13

7 Site Web de Scores **14**
7.1 Architecture du site . . . 14
7.2 Serveur HTTP Local . . . 14
7.3 Chargement des scores . . . 14
7.4 Fonctionnalités . . . 15
7.5 Export HTML automatique . . . 15

8 Tests et Exemples d'Utilisation **16**
8.1 Scénario de test 1 : Partie rapide . . . 16
8.2 Scénario de test 2 : Défaite . . . 16

9 Conclusion **16**
9.1 Bilan du projet 16
9.2 Fonctionnalités implémentées 17
9.3 Perspectives d'amélioration 17
9.4 Compétences acquises 18
9.5 Réflexion sur les paradigmes 18

1 Introduction

1.1 Contexte du projet

Ce projet s'inscrit dans le cadre du cours de Paradigmes de Programmation en Licence Informatique. L'objectif est de développer un mini-jeu complet intégrant plusieurs paradigmes de programmation : procédural, orienté objet, événementiel et concurrent. Tout en incluant une documentation complète.

1.2 Choix du jeu : Shooter Spatial

Le jeu choisi est un **shooter spatial en 2D** de type arcade. Le joueur contrôle un vaisseau spatial qui doit détruire des ennemis (astéroïdes) descendant vers lui. Ce type de jeu, inspiré des classiques comme Space Invaders, permet d'illustrer efficacement tous les paradigmes requis.

1.3 Objectifs pédagogiques

- Maîtriser la programmation orientée objet (POO) avec héritage et polymorphisme
- Implémenter une logique procédurale pour la boucle de jeu
- Gérer les événements utilisateur (clavier, souris) dans une interface graphique
- Utiliser la programmation concurrente avec des threads
- Sauvegarder et charger des données persistantes (scores)
- Créer une interface web pour visualiser les scores

2 Description du Gameplay

2.1 Règles du jeu

Objectif principal Le joueur doit détruire un maximum d'ennemis pour accumuler des points. La partie se termine quand le joueur perd tous ces points de vie (ennemi atteint le bas de l'écran = - 1 vie).

Mécaniques de jeu

- **Déplacement** : Le vaisseau peut se déplacer dans les 4 directions (gauche/droite/haut/bas)
- **Tir** : Le joueur peut tirer des projectiles vers le haut avec cooldown
- **Ennemis** : Des ennemis apparaissent périodiquement en haut et descendent
- **Collision** : Un projectile qui touche un ennemi les détruit tous les deux
- **Points** : Chaque ennemi détruit rapporte 10 points
- **Vies** : Le joueur dispose de 3 vies (extensible à 5 avec bonus)
- **Bonus** : 5 types de bonus tombent aléatoirement (vie, vitesse, tirs multiples)
- **Difficulté progressive** : Vitesse et fréquence d'apparition augmentent
- **Invincibilité** : Courte période d'invincibilité après avoir perdu une vie

2.2 Conditions de victoire et défaite

Objectif Le jeu est un survival shooter : l'objectif est de survivre le plus longtemps possible et d'accumuler le maximum de points.

Défaite
- Toutes les vies sont perdues (3 vies de départ)
- Un ennemi touche le vaisseau : -1 vie
- Un ennemi atteint le bas de l'écran : -1 vie

Système de vies et bonus
- **Vies de départ** : 3 vies
- **Maximum** : 5 vies (avec bonus)
- **Invincibilité** : Après perte d'une vie, 20 frames d'invincibilité
- **Bonus vie** : Collectables pour gagner une vie supplémentaire

3 Conception Logicielle

3.1 Architecture générale

Le projet est structuré en plusieurs modules Python pour respecter le principe de séparation des responsabilités :
- `game_classes.py` : Classes du modèle de jeu (POO)
  - ObjetVolant (classe abstraite)
  - Vaisseau, Ennemi, Projectile, Bonus
  - GameEngine (moteur de jeu)
- `score_manager.py` : Gestion des scores avec historique et statistiques
- `shooter_console.py` : Version console avec support plein écran
- `shooter_gui.py` : Version graphique avec menu et interface responsive
- `serveur_web.py` : Serveur HTTP pour le leaderboard web
- `index.html` : Page web du leaderboard (générée par score_manager)

3.2 Diagramme de classes

Le fichier **Shooter_Spatial_Classes_converted.pdf** présente le diagramme de classes UML du projet. On observe une hiérarchie d'héritage claire avec la classe abstraite `ObjetVolant` dont héritent `Vaisseau`, `Ennemi` et `Projectile`.

3.3 Classes principales

3.3.1 ObjetVolant (classe abstraite)

Classe de base pour tous les objets du jeu possédant une position et pouvant entrer en collision.

```
class ObjetVolant:
    def __init__(self, x: int, y: int, largeur: int, hauteur: int):
        self.x = x
        self.y = y
        self.largeur = largeur
        self.hauteur = hauteur
        self.actif = True

    def deplacer(self, dx: int, dy: int):
        self.x += dx
```

```
        self.y += dy

    def collision_avec(self, autre: 'ObjetVolant') -> bool:
        return (self.x < autre.x + autre.largeur and
                self.x + self.largeur > autre.x and
                self.y < autre.y + autre.hauteur and
                self.y + self.hauteur > autre.y)
```

Listing 1 – Classe ObjetVolant

3.3.2 Vaisseau

Représente le vaisseau du joueur avec système de vies, bonus et tirs multiples.
Attributs principaux :
— `vies` : Nombre de vies (3 à 5)
— `invincible_jusqu_a` : Frame jusqu'à laquelle le vaisseau est invincible
— `vitesse_base` : Vitesse de déplacement (adaptée à la taille d'écran)
— `vitesse_bonus` : Multiplicateur de vitesse (1.0 ou 1.5)
— `tir_double`, `tir_triple` : Modes de tir spéciaux
— `bonus_actif_jusqu_a` : Dict des bonus actifs avec leur durée

3.3.3 Ennemi

Représente un ennemi avec vitesse variable et déplacement progressif.
Particularité : Utilise `deplacement_fractionnaire` pour un mouvement fluide même à faible vitesse.

3.3.4 Projectile

Représente un tir du joueur. Monte automatiquement à vitesse fixe.

3.3.5 Bonus

Représente un bonus collectable avec 5 types différents :
— **Vie (+)** : Ajoute une vie (max 5)
— **Vitesse (»)** : Augmente vitesse de 50% pendant 10 secondes
— **Tir double (=)** : Tire 2 projectiles simultanément
— **Tir triple ()** : Tire 3 projectiles simultanément
— **Tir rapide (!!!)** : Réduit le cooldown de moitié
Chaque bonus a un poids de probabilité d'apparition différent.

3.3.6 GameEngine

Moteur de jeu principal qui coordonne toute la logique avec système de frames.
Responsabilités :
— Gestion des listes d'objets (ennemis, projectiles, bonus)
— Détection de collisions (projectile-ennemi, vaisseau-ennemi, vaisseau-bonus)
— Mise à jour du score et état du jeu
— Gestion de l'invincibilité temporaire
— Vérification si un bonus peut être ramassé (pas de doublon)

— Système de frames pour le timing des événements

3.3.7 ScoreManager

Gère la persistance des scores avec historique et statistiques complètes.
Fonctionnalités :
— Sauvegarde du meilleur score par joueur
— Historique des 10 dernières parties
— Statistiques : parties jouées, score moyen, score total
— Classement des meilleurs joueurs
— Export HTML du leaderboard pour affichage web

4 Implémentation des Paradigmes

4.1 Programmation Orientée Objet (POO)

Héritage La hiérarchie de classes illustre l'héritage : `Vaisseau`, `Ennemi`, `Projectile` et `Bonus` héritent tous d'`ObjetVolant`, factorisant ainsi les attributs et méthodes communs (position, déplacement, collision).

Encapsulation Chaque classe encapsule ses données et expose des méthodes publiques claires. Par exemple, `GameEngine` cache sa logique interne de vérification de collisions dans des méthodes privées (`_verifier_collisions`, `_peut_ramasser_bonus`).

Polymorphisme La méthode `collision_avec()` peut être appelée sur n'importe quel `ObjetVolant`, démontrant le polymorphisme. Tous les objets répondent à la même interface commune.

4.2 Système de Bonus

Le système de bonus illustre la POO avec une classe dédiée et un système de types :

```
class Bonus(ObjetVolant):
    TYPES = {
        "vie": {"nom": "Vie +1", "couleur": "#ff00ff",
                "icone": "+", "poids": 15},
        "vitesse": {"nom": "Vitesse", "couleur": "#00ffff",
                    "icone": ">>", "poids": 25},
        "tir_double": {"nom": "Tir Double", "couleur": "#ffff00",
                       "icone": "=", "poids": 25},
        "tir_triple": {"nom": "Tir Triple", "couleur": "#ff8800",
                       "icone": "   ", "poids": 15},
        "tir_rapide": {"nom": "Tir Rapide", "couleur": "#ff0000",
                       "icone": "!!!", "poids": 20},
    }

    def __init__(self, x, y):
        super().__init__(x, y, largeur=1, hauteur=1)
        # Selection aleatoire ponderee par les poids
        types_disponibles = list(self.TYPES.keys())
```

```
        poids = [self.TYPES[t]["poids"] for t in types_disponibles]
        self.type = random.choices(types_disponibles,
                                   weights=poids, k=1)[0]
```

Listing 2 – Système de bonus avec probabilités

Gestion des bonus actifs Le vaisseau utilise un dictionnaire pour gérer les bonus temporaires :

```
def activer_bonus(self, type_bonus, frame_actuelle, duree=300):
    if type_bonus == "vitesse":
        self.vitesse_bonus = 1.5
        self.bonus_actif_jusqu_a["vitesse"] = frame_actuelle +
           duree
    # ... autres bonus ...

def mettre_a_jour_bonus(self, frame_actuelle):
    bonus_a_retirer = []
    for type_bonus, frame_fin in self.bonus_actif_jusqu_a.items():
        if frame_actuelle >= frame_fin:
            bonus_a_retirer.append(type_bonus)
            # Desactiver le bonus
    for type_bonus in bonus_a_retirer:
        del self.bonus_actif_jusqu_a[type_bonus]
```

Listing 3 – Activation et expiration des bonus

4.3 Programmation Procédurale

La version console utilise une approche procédurale dans sa boucle principale :

```
def boucle_jeu_procedurale(game_engine):
    while not game_engine.jeu_termine:
        # 1. Afficher l' tat
        afficher_grille(game_engine)

        # 2. Lire commande
        cmd = lire_commande()

        # 3. Traiter commande
        if cmd == 'q':
            game_engine.vaisseau.deplacer_gauche()
        elif cmd == 'd':
            game_engine.vaisseau.deplacer_droite()
        elif cmd == ' ':
            game_engine.tirer()

        # 4. Mettre  jour
        game_engine.mettre_a_jour()

        # 5. V rifier victoire
        game_engine.verifier_victoire()
```

Listing 4 – Boucle de jeu procédurale

Cette approche séquentielle et linéaire est caractéristique du paradigme procédural.

4.4 Programmation Événementielle

La version graphique avec `tkinter` utilise le paradigme événementiel :

```
class ShooterGUI:
    def __init__(self, root):
        # ...
        # Liaison des    vnements
        self.root.bind('<Left>', self.deplacer_gauche)
        self.root.bind('<Right>', self.deplacer_droite)
        self.root.bind('<space>', self.tirer)

    def deplacer_gauche(self, event):
        """Callback d clench  par l'  vnement   Left"""
        if self.jeu_en_cours:
            self.game_engine.vaisseau.deplacer_gauche()

    def tirer(self, event):
        """Callback d clench  par l'  vnement   Space"""
        if self.jeu_en_cours:
            self.game_engine.tirer()
```

Listing 5 – Gestion des événements clavier

Les **timers** sont également des mécanismes événementiels :

```
def boucle_mise_a_jour(self):
    if not self.jeu_en_cours:
        return

    # Logique de mise    jour
    self.game_engine.mettre_a_jour()
    self.dessiner()

    # Relancer le timer (   vnement    r current)
    self.timer_jeu = self.root.after(50, self.boucle_mise_a_jour)
```

Listing 6 – Timer pour mise à jour périodique

4.5 Programmation Concurrente (Threads)

Trois threads s'exécutent en parallèle de la boucle principale pour améliorer l'expérience de jeu :

Thread Musique Joue la musique de fond sans bloquer le jeu, avec support de pause/reprise :

```python
class MusiqueThread(threading.Thread):
    def __init__(self, fichier="musique.mp3"):
        super().__init__(daemon=True)
        self.actif = True
        self.en_pause = False
        pygame.mixer.init()
        pygame.mixer.music.load(fichier)
        pygame.mixer.music.set_volume(0.3)

    def run(self):
        pygame.mixer.music.play(-1)  # Boucle infinie
        while self.actif:
            time.sleep(0.5)

    def pause(self):
        self.en_pause = True
        pygame.mixer.music.pause()

    def reprendre(self):
        self.en_pause = False
        pygame.mixer.music.unpause()
```

Listing 7 – Thread de musique avec pygame

Thread Spawner Ennemis Fait apparaître des ennemis périodiquement avec difficulté progressive :

```python
class SpawnerThread(threading.Thread):
    def __init__(self, game_engine):
        super().__init__(daemon=True)
        self.game_engine = game_engine
        self.intervalle = 2.0  # secondes
        self.vitesse = 0.3

    def run(self):
        while not self.game_engine.jeu_termine:
            time.sleep(self.intervalle)
            self.game_engine.ajouter_ennemi(self.vitesse)

    def ajuster_difficulte(self, ennemis_detruits):
        # Augmenter vitesse et frequence
        niveau = 1 + (ennemis_detruits // 5)
        self.vitesse = min(2.0, 0.3 * 1.08 ** niveau)
        self.intervalle = max(0.8, 2.0 - niveau * 0.3)
```

Listing 8 – Spawner avec difficulte adaptative

Thread Spawner Bonus Fait apparaître des bonus aléatoirement avec probabilité :

```python
class BonusSpawnerThread(threading.Thread):
    def run(self):
```

```
        while not self.game_engine.jeu_termine:
            # Intervalle aleatoire 8-15 secondes
            temps = random.uniform(8.0, 15.0)
            time.sleep(temps)

            # 30% de chance d'apparition
            if random.random() < 0.3:
                self.game_engine.ajouter_bonus()
```

Listing 9 – Spawner de bonus aleatoire

Ces threads démontrent la **programmation parallèle** : plusieurs tâches s'exécutent simultanément (musique, spawn ennemis, spawn bonus, boucle principale), améliorant la fluidité et la réactivité du jeu.

5 Gestion des Scores

5.1 Format de sauvegarde JSON

Les scores sont sauvegardés avec un historique complet et des statistiques détaillées :

```
{
  "Alice": {
    "meilleur_score": 230,
    "parties_jouees": 12,
    "score_total": 1850,
    "historique": [
      {"score": 180, "date": "2026-02-08 14:30:25"},
      {"score": 230, "date": "2026-02-08 15:12:10"},
      ...
    ]
  },
  "Bob": {
    "meilleur_score": 180,
    "parties_jouees": 5,
    "score_total": 750,
    "historique": [...]
  }
}
```

Listing 10 – Format JSON des scores avec historique

5.2 Classe ScoreManager

La classe `ScoreManager` gère toutes les opérations sur les scores avec fonctionnalités avancées :

- **Chargement** : Lecture du fichier JSON au démarrage avec gestion d'erreurs
- **Sauvegarde** : Écriture automatique après chaque partie
- **Enregistrement** : Mise à jour du meilleur score et de l'historique
- **Historique** : Conservation des 10 dernières parties
- **Statistiques** : Calcul du score moyen, total, nombre de parties

— **Classement** : Tri des scores par ordre décroissant
— **Export HTML** : Génération d'une page web interactive

```python
def enregistrer_score(self, joueur: str, score: int) -> bool:
    ancien_record = self.obtenir_meilleur_score(joueur)
    nouveau_record = score > ancien_record

    if joueur not in self.scores:
        self.scores[joueur] = {
            "meilleur_score": 0,
            "parties_jouees": 0,
            "score_total": 0,
            "historique": []
        }

    # Mettre a jour les statistiques
    self.scores[joueur]["parties_jouees"] += 1
    self.scores[joueur]["score_total"] += score

    if score > self.scores[joueur]["meilleur_score"]:
        self.scores[joueur]["meilleur_score"] = score

    # Ajouter a l'historique avec timestamp
    self.scores[joueur]["historique"].append({
        "score": score,
        "date": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    })

    # Garder seulement les 10 dernieres parties
    if len(self.scores[joueur]["historique"]) > 10:
        self.scores[joueur]["historique"] = \
            self.scores[joueur]["historique"][-10:]

    self._sauvegarder_scores()
    return nouveau_record
```

Listing 11 – Méthode d'enregistrement avec historique

6 Interface Graphique

6.1 Choix technologique : tkinter

La bibliothèque `tkinter` a été choisie car :
— Elle est incluse dans Python (pas de dépendance externe)
— Elle permet la gestion d'événements (clavier, souris, timers)
— Elle offre un canvas pour le dessin 2D
— Elle supporte le redimensionnement dynamique

6.2 Architecture de l'interface

L'interface graphique est organisée en plusieurs écrans :

Menu Principal (MenuPrincipal)
- Fond animé avec étoiles défilantes
- Boutons : Jouer, Instructions, Scores, Quitter
- Titre avec effet de glow animé
- Responsive : adaptation automatique à la taille d'écran

Écran Instructions
- Explications des contrôles et mécaniques
- Liste des bonus avec leurs effets
- Scrollable pour petit écran

Écran Scores
- Affichage du classement des 10 meilleurs joueurs
- Médailles pour le top 3
- Bouton pour ouvrir le leaderboard web
- Couleurs différenciées par rang

Écran de Jeu
- Canvas principal avec fond spatial animé
- Affichage en temps réel : score, vies, temps, niveau
- Indicateurs de bonus actifs avec couleurs
- Support du redimensionnement en plein écran
- Contrôles musique (pause, volume)

6.3 Gestion des événements

L'interface graphique réagit à plusieurs types d'événements :
- **Événements clavier** : Flèches directionnelles (← → ↑ ↓), ESPACE, P (pause musique), ESC (quitter)
- **Événements bouton** : Clics sur les boutons du menu
- **Événements timer** : Mise à jour périodique (30-50ms), spawn d'ennemis, spawn de bonus
- **Événements redimensionnement** : Adaptation dynamique des dimensions du jeu

```
def sur_redimensionnement(self, event):
    if event.widget != self.root:
        return

    nouvelle_largeur = event.width
    nouvelle_hauteur = event.height

    # Mise a jour des dimensions
    self.LARGEUR_PIXELS = nouvelle_largeur
    self.HAUTEUR_PIXELS = nouvelle_hauteur

    # Ajuster les dimensions du moteur de jeu
    nouvelle_largeur_jeu = self.LARGEUR_PIXELS // self.TAILLE_CASE
    nouvelle_hauteur_jeu = self.HAUTEUR_JEU // self.TAILLE_CASE
```

```
15
16      # Ajuster position du vaisseau proportionnellement
17      ratio_x = nouvelle_largeur_jeu / self.game_engine.largeur
18      self.game_engine.vaisseau.x *= ratio_x
19
20      # Mettre a jour les dimensions
21      self.game_engine.largeur = nouvelle_largeur_jeu
22      self.game_engine.hauteur = nouvelle_hauteur_jeu
```

Listing 12 – Gestion du redimensionnement dynamique

6.4 Boucle de rendu

Le jeu utilise une boucle de rendu asynchrone via `root.after()` :

```
1  def boucle_mise_a_jour(self):
2      if not self.jeu_en_cours:
3          return
4
5      # Mise a jour logique
6      self.game_engine.mettre_a_jour()
7
8      # Rendu graphique
9      self.dessiner_fond_anime()
10     self.dessiner_objets()
11     self.dessiner_interface()
12
13     # Relancer dans 30-50ms selon la performance
14     self.root.after(30, self.boucle_mise_a_jour)
15
16 def dessiner_objets(self):
17     # Dessiner bonus
18     for bonus in self.game_engine.bonus:
19         couleur = bonus.info["couleur"]
20         self.canvas.create_oval(...)
21
22     # Dessiner ennemis
23     for ennemi in self.game_engine.ennemis:
24         self.canvas.create_oval(..., fill='red')
25
26     # Dessiner projectiles
27     for proj in self.game_engine.projectiles:
28         self.canvas.create_line(..., fill='yellow')
29
30     # Dessiner vaisseau avec effet invincibilite
31     if self.game_engine.vaisseau.invincible:
32         # Clignotement
33         couleur = 'yellow' if frame % 4 < 2 else 'cyan'
34     else:
35         couleur = self.couleur_vaisseau_selon_bonus()
36     self.canvas.create_polygon(..., fill=couleur)
```

Listing 13 – Boucle de rendu graphique optimisee

7 Site Web de Scores

7.1 Architecture du site

Le site web est une application **Single Page Application (SPA)** composée de :
— `index.html` : Structure HTML, styles CSS et JavaScript embarqué
— `scores.json` : Données chargées dynamiquement
— `serveur_web.py` : Serveur HTTP local optionnel

7.2 Serveur HTTP Local

Un serveur HTTP simple permet de visualiser le leaderboard dans un navigateur :

```python
def demarrer_serveur():
    PORT = 8000
    handler = http.server.SimpleHTTPRequestHandler

    with socketserver.TCPServer(("", PORT), handler) as httpd:
        url = f"http://localhost:{PORT}/index.html"
        print(f"Serveur sur {url}")

        # Ouvrir automatiquement le navigateur
        webbrowser.open(url)

        # Demarrer
        httpd.serve_forever()
```

Listing 14 – Serveur HTTP avec auto-ouverture

7.3 Chargement des scores

Le site utilise l'API `fetch()` pour charger le fichier JSON :

```javascript
async function chargerScores() {
    const response = await fetch('scores.json');
    const data = await response.json();

    // Creer le tableau des scores
    let scoresArray = [];
    for (const [joueur, info] of Object.entries(data)) {
        scoresArray.push({
            joueur: joueur,
            score: info.meilleur_score,
            parties: info.parties_jouees,
            moyenne: (info.score_total / info.parties_jouees).
               toFixed(1)
        });
```

```
    }

    // Trier par score decroissant
    scoresArray.sort((a, b) => b.score - a.score);

    // Afficher dans le tableau HTML
    afficherTableau(scoresArray);
}
```

Listing 15 – Chargement JavaScript des scores

7.4 Fonctionnalités

- **Classement en temps réel** : Affichage des 20 meilleurs scores
- **Médailles** : Symboles spéciaux pour les 3 premier
- **Design moderne** : Gradient de fond, animations, effets hover
- **Actualisation automatique** : Actualisation toutes les 30 secondes
- **Bouton refresh** : Actualisation manuelle
- **Responsive** : Adaptation mobile/tablette/desktop
- **Statistiques** : Score total, moyenne, dernière mise à jour

7.5 Export HTML automatique

La méthode `exporter_html()` du ScoreManager génère automatiquement la page :

```
def exporter_html(self, fichier_sortie="index.html"):
    classement = self.obtenir_classement(20)

    html = """<!DOCTYPE html>
<html>
<head>
    <title>Shooter Spatial - Leaderboard</title>
    <style>
        /* Styles modernes avec gradients */
    </style>
</head>
<body>
    <h1> SHOOTER SPATIAL </h1>
"""

    for i, (joueur, score) in enumerate(classement, 1):
        medaille = "      " if i == 1 else "      " if i == 2 else
            "      " if i == 3 else f"{i}."
        html += f'<div class="score-entry top{i if i <= 3 else
            ""}">'
        html += f'  <span>{medaille}</span>'
        html += f'  <span>{joueur}</span>'
        html += f'  <span>{score:,} pts</span>'
        html += '</div>'

    html += """
```

```
    <button onclick="location.reload()"> Actualiser</button>
</body>
</html>
"""

    with open(fichier_sortie, 'w', encoding='utf-8') as f:
        f.write(html)
```

Listing 16 – Generation HTML du leaderboard

8 Tests et Exemples d'Utilisation

8.1 Scénario de test 1 : Partie rapide

1. Lancement du jeu (version GUI)
2. Saisie du nom : "Alice"
3. Destruction de 10 ennemis = 100 points
4. Nouveau record enregistré

8.2 Scénario de test 2 : Défaite

1. Lancement du jeu
2. Le joueur ne tire pas assez vite
3. Un ennemi atteint le bas de l'écran
4. Défaite avec score de 30 points
5. Score non enregistré (inférieur au record)

9 Conclusion

9.1 Bilan du projet

Ce projet a permis de mettre en pratique quatre paradigmes de programmation étudiés :

— **POO** : Architecture claire avec héritage (5 classes héritant d'ObjetVolant), encapsulation des données et méthodes, polymorphisme sur les collisions
— **Procédural** : Boucle de jeu linéaire en version console avec séquence d'instructions
— **Événementiel** : Interface graphique réactive avec tkinter (clavier, boutons, timers), gestion d'événements asynchrones
— **Concurrent** : Threads pour musique, spawn d'ennemis et spawn de bonus s'exécutant en parallèle

Le système de scores avec persistance JSON, historique détaillé et le site web de leaderboard complètent le projet en ajoutant des fonctionnalités modernes et professionnelles.

9.2 Fonctionnalités implémentées

Le jeu inclut de nombreuses fonctionnalités avancées :

1. **Système de bonus** : 5 types de bonus avec effets temporaires et pondération de probabilité
2. **Système de vies** : 3 à 5 vies avec invincibilité temporaire après perte
3. **Difficulté progressive** : Vitesse et fréquence d'apparition augmentent avec le score
4. **Interface moderne** : Menu animé, écrans multiples, responsive design
5. **Support multi-plateformes** : Version console (Windows/Linux/Mac) et GUI
6. **Musique** : Support de musique de fond avec pause et contrôle du volume
7. **Statistiques complètes** : Historique des 10 dernières parties, score moyen
8. **Leaderboard web** : Page HTML générée automatiquement avec design moderne
9. **Plein écran adaptatif** : Redimensionnement dynamique pour maximiser l'expérience
10. **Adaptation de la difficulté** : Vitesse du vaisseau adaptée à la taille d'écran

9.3 Perspectives d'amélioration

Plusieurs améliorations pourraient être apportées dans le futur :

1. **Nouveaux types d'ennemis**
   — Ennemis avec patterns de déplacement différents (zigzag, cercles)
   — Ennemis qui tirent des projectiles
   — Boss de fin de niveau avec points de vie multiples
2. **Power-ups avancés**
   — Bouclier temporaire absorbant les dégâts
   — Ralentissement du temps
   — Bombe détruisant tous les ennemis à l'écran
   — Aimant attirant tous les bonus
3. **Effets visuels et audio**
   — Animations d'explosion avec particules
   — Effets sonores pour tirs, collisions, bonus
   — Musiques thématiques changeant selon le niveau
   — Effet de secousse d'écran lors des impacts
4. **Modes de jeu additionnels**
   — Mode survie : durée illimitée, difficulté croissante extrême
   — Mode campagne : série de niveaux avec objectifs
   — Mode contre-la-montre : atteindre un score en temps limité
   — Mode zen : pas de mort, pour l'entraînement
5. **Multijoueur**
   — Mode coopératif local (2 joueurs, clavier partagé)
   — Mode compétitif (qui survit le plus longtemps)
   — Leaderboard en ligne avec API REST
   — Synchronisation des scores entre machines
6. **Personnalisation**

— Choix du vaisseau (apparence, caractéristiques)
— Skins et thèmes visuels (espace, océan, cyberpunk)
— Système de progression et déblocage de contenus
— Achievements et trophées

7. **Optimisations techniques**
— Pool d'objets pour éviter l'allocation mémoire excessive
— Quadtree ou spatial hashing pour détection de collisions optimisée
— Compilation avec Cython pour améliorer les performances
— Support de shaders avec OpenGL via PyOpenGL

9.4 Compétences acquises

Ce projet a renforcé :
— La maîtrise de Python et de ses bibliothèques standard (tkinter, threading, json, pathlib)
— La conception orientée objet et les diagrammes UML (classes, séquence, activités)
— La gestion d'événements dans les interfaces graphiques
— La programmation concurrente avec threads et synchronisation
— La manipulation de fichiers JSON pour la persistance
— Le développement web frontend (HTML/CSS/JavaScript)
— La documentation technique avec LaTeX et PlantUML
— L'architecture logicielle et la séparation des responsabilités
— Le debugging et la gestion d'erreurs
— L'optimisation de performances (framerate, affichage)

9.5 Réflexion sur les paradigmes

Ce projet démontre qu'aucun paradigme n'est supérieur aux autres : chacun a ses forces et son domaine d'application optimal. La clé d'un bon logiciel est de savoir combiner judicieusement les paradigmes selon les besoins :
— **POO** pour structurer le code et favoriser la réutilisation
— **Procédural** pour les algorithmes séquentiels et la logique de jeu
— **Événementiel** pour les interfaces interactives et réactives
— **Concurrent** pour les tâches parallèles et l'amélioration des performances

Cette approche multi-paradigme est représentative du développement logiciel moderne, où la flexibilité et l'adaptation aux problèmes sont essentielles.

Annexes

A. Structure des fichiers

```
shooter_spatial/

 game/                        # Module principal du jeu
    game_classes.py          # Classes POO (ObjetVolant, Vaisseau,
                               #   Ennemi, Projectile, Bonus, GameEngine)
    score_manager.py         # Gestion scores avec historique
    shooter_console.py       # Version console plein écran
```

```
    shooter_gui.py          # Version GUI avec menu
    serveur_web.py          # Serveur HTTP pour leaderboard
    index.html              # Leaderboard web (généré)
    scores.json             # Données des scores

 shooter_console.bat         # Lanceur Windows (console)
 shooter_gui.bat             # Lanceur Windows (GUI)
 installer_dependencies.bat  # Installation automatique

 diagramme_classes.pdf       # Diagrammes UML exporter en PDF (PlantUML)
 diagramme_classes.puml      # Diagrammes UML (PlantUML)
 rapport.pdf                 # Ce rapport exporter en PDF (LaTeX)
 README.md                   # Documentation utilisateur
```

B. Commandes d'exécution

Installation des dépendances

```
# Windows (recommandé) :
installer_dependencies.bat

# Ou manuellement :
pip install pygame
```

Version console

```
python game/shooter_console.py
# ou double-clic sur shooter_console.bat (Windows)
```

Version graphique (recommandé)

```
python game/shooter_gui.py
# ou double-clic sur shooter_gui.bat (Windows)
```

Serveur web pour leaderboard

```
python game/serveur_web.py
# Ouvre automatiquement http://localhost:8000/index.html
```

C. Configuration et personnalisation

Difficulté (dans shooter_console.py et shooter_gui.py)

```
class ConfigDifficulte:
    VITESSE_INITIALE = 0.3    # Vitesse des ennemis
    SPAWN_INITIAL = 2.0        # Intervalle spawn (secondes)
    CHANCE_BONUS = 0.3         # Probabilité d'apparition bonus
    VIES_DEPART = 3            # Nombre de vies initial
```

Volume de la musique

```
# Dans le constructeur de MusiqueThread
pygame.mixer.music.set_volume(0.3)  # 0.0 à 1.0
```

Dimensions de la fenêtre GUI

```
# Adaptation automatique à l'écran
# Peut être modifié dans shooter_gui.py ligne ~25
largeur = 600  # Minimum
hauteur = 800  # Minimum
```