

# DEATHROW

## Rapport de Projet

---

### Steampunk Games

Projet de développement de jeu vidéo

**BOYER Etienne • TOPIA Aurélien • MOHAJERI Giv**

Promotion 2025 – 2026

Deathrow — Horde Survival Shooter en Python / Pygame

# Sommaire

---

1. Présentation des membres du groupe .....	5
2. Tâches effectuées (individuelles).....	8
2.1 Graphisme (Responsable) .....	9
A — Map « Grab Der Toten » (première map jouable) .....	9
B — Map « Death On The Beach » (map plage) .....	11
C — Map « Jigoku No Torii » (map Japon).....	12
D — Personnages joueurs.....	14
E — Zombies et Boss.....	15
2.2 Settings du jeu (Responsable).....	16
A — Architecture du menu Settings.....	17
B — Onglet Audio.....	18
C — Onglet Vidéo.....	18
D — Onglet Contrôles.....	19
E — Sauvegarde et persistance des paramètres.....	19
F — Difficultés rencontrées.....	20
Conclusion — Boyer Etienne.....	21
2.3 Musique et effets sonores (Responsable).....	22
A — Musique principale.....	22
B — Sons zombies .....	22
C — Sons d'interaction .....	23
D — Son bonus .....	23
E — Son de manches.....	23
2.4 Graphismes (Suppléant).....	23
A — Bonus .....	24
B — Atouts .....	24
C — Exsanguineur .....	25
D — Boss .....	25
2.5 Site web (Responsable).....	25
A — Hébergement.....	25

B — Contenu .....	25
C — Front-end .....	26
2.6 LAN — Mode réseau multijoueur .....	26
Architecture hôte / client .....	27
Protocole de communication .....	27
Chiffrement des codes de connexion .....	27
Gestion des threads et de l'event loop asynco .....	28
Gestion du pare-feu Windows .....	28
2.7 Map — Gestion map + Pygame (Suppléant) .....	28
Chargement des maps avec pytmx .....	29
Rendu de la tilemap avec pyscroll .....	29
Extraction des objets de jeu depuis le TMX .....	29
Système de navigation pour les zombies .....	30
2.8 Développement du moteur de jeu .....	30
A — Menu principal .....	30
B — Gestion de la carte .....	31
C — Gestion du joueur .....	31
D — Gestion des manches .....	32
E — Gestion de la difficulté .....	32
F — Gestion des armes .....	33
G — Animation des personnages .....	33
H — Déplacement du personnage .....	34
2.9 Intelligence Artificielle (Responsable) .....	35
A — Architecture générale de l'IA .....	35
B — Déplacement vers le joueur le plus proche .....	36
C — Système d'attaque .....	37
D — Gestion multi-cibles en multijoueur .....	37
E — Grille de navigation .....	38
F — Génération et gestion de la horde .....	39
G — Le boss .....	39
H — Difficultés rencontrées .....	40
2.10 Développement des fonctionnalités (Suppléant) .....	40
A — Système de balles .....	40
B — Mort du joueur .....	41

---

C — Atouts : résurrection et freeze .....	42
D — Difficultés rencontrées .....	43
2.11 Programme d'installation (Responsable).....	43
A — Contexte et objectif .....	43
B — PyInstaller : génération de l'exécutable.....	44
C — Inno Setup : création de l'installateur Windows.....	45
D — Difficultés rencontrées .....	46
3. Conclusion générale .....	48
3.1 Bilan par membre .....	48
BOYER Etienne.....	48
TOPIA Aurélien.....	48
MOHAJERI Giv.....	49
3.2 Perspectives et développements futurs .....	49

# 1. Présentation des membres du groupe

L'équipe Steampunk Games est composée de trois membres, chacun titulaire de responsabilités précises et d'un rôle de suppléant permettant d'assurer la continuité du projet en cas d'indisponibilité. La répartition des tâches a été établie en tenant compte des compétences de chaque membre et de leurs souhaits d'explorer de nouveaux domaines techniques.

## BOYER Etienne

Map	Graphisme	Settings du jeu	Site web (suppléant)	Dev. moteur (suppléant)	Dev. fonctionnalités (suppléant)
Map Grab der Toten Map Death on the Beach Map Jigoku no Tori Assemblage de tuiles Création point d'apparition joueur Création point d'apparition zombie Création délimitation	Création Sprite du joueur Création éléments de décor des maps Redimensionnement des sprites non adaptés	Création environnement settings Enregistrement des settings Settings vidéo Settings audio Settings contrôles Environnement graphique	—	—	—

de la map Création collisions					
-------------------------------------	--	--	--	--	--

## TOPIA Aurélien

Musique / FX	Graphisme (suppléant)	Site Web	LAN	Map (suppléant)	Dev. moteur	Dev. fonctionnalités (suppléant)
Musique principale Sons zombies Sons interactions Sons bonus Sons manches	Bonus Atouts Exsangueur Boss	Front-end Back-end Hébergement	Implémentation multijoueur Gestion envoi des données	Design des maps Gestion map + Pygame	Menu Gestion carte Gestion joueur Gestion manches Gestion difficulté Gestion armes	Bonus Atouts Achat en jeu Gestion inventaire Gestion exsangueur

## MOHAJERI Giv

Musique / FX (suppléant)	LAN (suppléant)	Intelligence Artificielle	Dev. moteur	Dev. fonctionnalités	Programme d'installation	Settings (suppléant)
—	—	Pathfinding IA des zombies	Gestion de la mort du joueur	—	Script Inno Setup Commande PyInstaller	—

					Création de l'exécutable	
--	--	--	--	--	--------------------------	--

---

## 2. Tâches effectuées (individuelles)

---



# BOYER Etienne

---

## 2.1 Graphisme (Responsable)

Le graphisme représente l'un des piliers fondamentaux de l'expérience Deathrow. En tant que responsable graphisme, mon rôle a consisté à concevoir et assembler l'ensemble des environnements jouables ainsi que les personnages qui les peuplent. Chaque décision visuelle a été guidée par un objectif unique : renforcer l'atmosphère post-apocalyptique du jeu tout en conservant la lisibilité nécessaire à un bon gameplay.

L'ensemble des maps a été créé avec l'éditeur de niveaux Tiled, un outil open-source spécialisé dans la création de maps au format TMX. Il permet de travailler en couches (layers), de définir des zones de collision via des objets vectoriels, et de placer les points d'apparition des entités directement dans l'éditeur. Les sprites des personnages ont quant à eux été créés ou retouchés dans Piskel, un éditeur de pixel art en ligne adapté aux animations frame par frame.

### A — Map « Grab Der Toten » (première map jouable)

La map Grab Der Toten constitue la première zone jouable du jeu. Son nom, emprunté à l'univers de Call of Duty Zombies, donne le ton : un environnement morbide et claustrophobique au cœur d'un village de la Première Guerre mondiale.

#### *Dimensions et structure*

La map est construite sur une grille de 100 × 150 tuiles avec une taille de tuile de 32 × 32 pixels, ce qui donne une surface de jeu de 3 200 × 4 800 pixels. Cette dimension généreuse offre suffisamment d'espace pour les

déplacements des joueurs et le comportement des vagues de zombies, sans créer de zones mortes difficiles à atteindre.

### ***Couches graphiques***

La map est organisée en neuf couches graphiques et cinq couches d'objets. Chaque couche remplit un rôle précis dans la composition visuelle :

- Background : couche de sol de base formant le fond général de la map.
- Chemin : réseau de chemins reliant les différentes zones de la map.
- Décoration : éléments décoratifs divers posés sur le sol (caisses, objets abandonnés).
- Barbelée\_vertical / horizontal\_barbelée : réseaux de fils barbelés formant les obstacles et délimitations du terrain de combat.
- Zombie\_mort : couche affichant les corps de zombies au sol après leur élimination.
- Arbre\_coupé : arbres abattus servant d'obstacles et d'éléments de décor.
- Mur : murs et structures solides définissant les limites des bâtiments.
- Bâtiment : couche principale des bâtiments et structures architecturales de la map.
- Spawners : couche d'objets définissant les points d'apparition des zombies autour de la zone de combat.
- Player : couche d'objets définissant le point de spawn du joueur et les emplacements des outils interactifs.
- Collision / delimitation : couches d'objets définissant les zones de collision invisibles bloquant les déplacements.
- Buy\_zone : couche d'objets définissant les zones d'achat des armes disponibles sur la map.

## ***Zones de collision et spawners***

Les collisions sont définies via deux couches d'objets : « collision » qui contient les rectangles bloquants autour des bâtiments et des murs, et « delimitation » qui définit les limites extérieures de la zone jouable. Ensemble elles empêchent les joueurs et les zombies de traverser les structures ou de sortir de la map.

Les spawners de zombies sont répartis autour de la zone de combat via la couche « spawners », forçant les joueurs à surveiller plusieurs angles simultanément. Le point d'apparition du joueur est défini dans la couche « player », positionné au centre de la zone de combat.

## ***Identité visuelle***

La palette de couleurs de cette map privilégie les tons bruns, beiges et verts délavés, cohérents avec un décor de ville enneigée abandonnée en temps de guerre. Les bâtiments portent des traces d'usure et de destruction, renforçant l'ambiance post-apocalyptique. L'utilisation de couches d'objets superposées donne une profondeur visuelle malgré la vue du dessus en 2D orthogonale.

## **B — Map « Death On The Beach » (map plage)**

Death On The Beach est la deuxième map conçue pour Deathrow. Elle reprend les mêmes tuiles et la même base technique que Grab Der Toten, car les deux maps partagent le même univers : une plage abandonnée envahie par les zombies. Le nom volontairement évocateur et le design graphique sont pensés pour créer une ambiance différente de la première map, plus ouverte et lumineuse en surface mais tout aussi dangereuse.

Cette map est référencée dans le menu de sélection sous le bouton « Death On The Beach ». Les éléments graphiques sont déjà posés, et la structure de la map est opérationnelle dans Tiled.

## ***Dimensions et organisation***

La map adopte les mêmes dimensions de base que Grab Der Toten (100 × 150 tuiles, 32 × 32 px par tuile). Cette cohérence de taille facilite le développement : les mécaniques de jeu (vitesse de déplacement, portée des armes, comportement des zombies) n'ont pas besoin d'être réétalonnées pour chaque map.

### ***Différenciation graphique***

Death On The Beach propose plus d'espace ouvert, avec une zone de combat centrale plus large et moins de bâtiments, ce qui modifie la dynamique de jeu. Les joueurs ne peuvent plus se réfugier facilement dans des couloirs et doivent gérer des zombies arrivant de plusieurs directions à la fois.

## **C — Map « Jigoku No Torii » (map Japon)**

Jigoku No Torii (littéralement « Le Torii de l'Enfer ») est la troisième map du jeu, constituant un changement radical d'univers visuel. Contrairement aux deux maps précédentes ancrées dans un décor occidental contemporain, cette map transporte le joueur dans un village japonais médiéval fantastique.

### ***Dimensions et tilesets***

La map Japon est construite sur une grille de 100 × 100 tuiles avec des tuiles de 48 × 48 pixels, donnant une surface de jeu de 4 800 × 4 800 pixels. Ce format légèrement plus grand et surtout carré modifie la dynamique de jeu par rapport aux maps rectangulaires précédentes. Le passage à des tuiles de 48 pixels (contre 32 pour les maps plage) permet d'afficher davantage de détails dans les éléments architecturaux japonais sans pixellisation excessive.

Cette map utilise un ensemble de tilesets spécialisés bien différents de ceux de la plage :

- A2\_Master et A2\_Master\_route : tileset principal fournissant le sol, les chemins et les routes en pierre.
- A1\_Cave : tileset de grottes et espaces souterrains pour les zones d'ombre.
- A5\_Master : tileset de nature incluant herbe, eau et végétation.
- A4\_Master : tileset d'architecture intérieure et de structures.
- Happy\_New\_Year / Happy\_New\_Year\_2 / Happy\_New\_Year\_3 : tileset thématique japonais contenant les éléments décoratifs de fête (lanternes, drapeaux, ornements).
- Outside\_E et Outside\_alt : tileset d'extérieur pour les zones ouvertes.

### ***Couches graphiques***

L'architecture en couches de la map Japon est plus complexe que celle de la plage, avec davantage de structures architecturales à gérer :

- Background : sol de base avec alternance de motifs caractéristiques des pavés japonais.
- Route : réseau de chemins en pierre reliant les différentes zones du village.
- Water : zones aquatiques (étangs, ruisseaux) typiques des jardins japonais.
- Roche : rochers et pierres décoratives.
- Temple (×2) : éléments architecturaux des temples shintoïstes ou bouddhistes.
- Champs : parcelles agricoles en terrasses.
- Pont : structures permettant de franchir les zones d'eau.
- Maison / Maison2 : habitations traditionnelles japonaises.
- Escalier : marches reliant les différents niveaux de la map.

- Toit\_temple / Toit\_temple2 : toits des bâtiments en couche haute, afin de passer « sous » les toitures.
- Muraille / Muraille\_vertical : enceintes et murs d'enceinte délimitant les zones.

### ***Points d'apparition et identité visuelle***

La map dispose de neuf spawners de zombies répartis stratégiquement aux quatre coins de la carte et le long des bordures, forçant les joueurs à surveiller l'ensemble du périmètre. Le point de spawn joueur est positionné au centre de la zone principale, à proximité du torii central qui donne son nom à la map.

La palette de couleurs est dominée par les verts profonds des mousses et des bambous, les gris bleutés des pierres, et les rouges typiques des structures de temple japonais. Le contraste entre la sérénité apparente du décor et l'invasion zombie crée une tension visuelle particulièrement efficace.

## **D — Personnages joueurs**

Le personnage joueur de Deathrow est un sprite 2D vue du dessus animé en pixel art. Il a été entièrement réalisé dans Piskel, l'éditeur de pixel art utilisé pour l'ensemble des créations originales du jeu.

### ***Conception dans Piskel***

Piskel est un outil en ligne gratuit permettant de dessiner et d'animer des sprites pixel art frame par frame. Son interface simple mais puissante permet de travailler à l'échelle du pixel tout en prévisualisant l'animation en temps réel. C'est l'outil idéal pour un développement indépendant où la rapidité de prototypage est cruciale.

### ***Structure de la sprite sheet***

Le personnage est représenté par une sprite sheet de  $32 \times 128$  pixels, organisée en quatre bandes horizontales de  $32 \times 32$  pixels correspondant chacune à une direction de déplacement :

- Ligne 0 ( $y=0$ ) : vue de dos (déplacement vers le bas dans la logique de jeu)
- Ligne 1 ( $y=32$ ) : vue de gauche (déplacement vers la gauche)
- Ligne 2 ( $y=64$ ) : vue de droite (déplacement vers la droite)
- Ligne 3 ( $y=96$ ) : vue de face (déplacement vers le haut)

La couleur noire (0, 0, 0) est définie comme transparente via la méthode `set_colorkey` de Pygame, permettant d'afficher le personnage sur n'importe quel fond de map sans fond blanc parasite.

### ***Hitbox et pieds***

Deux rectangles de collision définissent les interactions physiques du joueur. Le premier, `self.rect`, correspond au rectangle englobant l'image complète ( $32 \times 32$  pixels) et sert au rendu visuel. Le second, `self.feet`, est un rectangle réduit centré sur la partie basse du sprite (largeur = 50 % du sprite, hauteur fixe de 12 pixels) simulant les pieds du personnage. C'est ce rectangle réduit qui est utilisé pour les collisions avec les murs, permettant au joueur de passer partiellement « derrière » des éléments de décor pour un rendu plus naturel en vue top-down.

## **E — Zombies et Boss**

Les entités ennemies du jeu ont été conçues pour s'intégrer visuellement dans les différents univers des maps tout en restant immédiatement identifiables par le joueur. Deux types d'entités coexistent : les zombies classiques et le boss.

### ***Zombie standard***

Le zombie standard est représenté par un sprite nommé « `german.png` » dans le répertoire des assets. Ce nom fait référence à l'univers visuel de la map *Grab Der Toten*, inspiré de la Première Guerre mondiale. Le sprite est construit sur la même base technique que le personnage joueur : une sprite sheet  $32 \times 32$  pixels chargée via Piskel, avec la couleur noire définie comme transparente.

Sa conception graphique reprend les codes du zombie classique tout en adoptant une tenue militaire délabrée cohérente avec l'univers de la première map. L'uniforme kaki déchiré, les yeux rouges lumineux et la démarche suggérée par le sprite le rendent immédiatement identifiable comme menace.

### ***Boss***

Le boss est une version améliorée du zombie standard, visuellement distincte pour signaler sa dangerosité accrue. Comme l'indique le code source, son sprite est nommé « `boss.png` » et reprend la silhouette du zombie standard mais avec deux différences visuelles majeures : une tenue beige au lieu du kaki militaire, et des yeux cyan au lieu du rouge classique. Cette modification colorimétrique rapide à réaliser dans Piskel suffit à créer une lecture visuelle immédiate pour le joueur.

## **2.2 Settings du jeu (Responsable)**

Le module Settings de Deathrow constitue l'une de mes responsabilités principales en tant que développeur. Son objectif est de permettre aux joueurs de personnaliser leur expérience de jeu selon leurs préférences matérielles et leurs habitudes, en regroupant dans une interface unique l'ensemble des paramètres audio, vidéo et de contrôle.

L'implémentation complète est réalisée dans le fichier `menu.py` via la classe Settings. L'interface Settings est accessible directement depuis le menu principal via le bouton « Settings ».



## A — Architecture du menu Settings

### *Structure générale*

Le menu Settings est organisé en deux panneaux distincts séparés par un trait de séparation vertical. Le panneau gauche (largeur fixe de 280 pixels) affiche les onglets de navigation et le bouton de retour. Le panneau droit affiche les options de l'onglet sélectionné.

Trois onglets sont disponibles dans la version actuelle :

- AUDIO : paramètres sonores du jeu.
- VIDEO : paramètres d'affichage.
- CONTROLS : configuration des touches clavier.

### *Système de données `SETTINGS_DATA`*

Tous les paramètres sont stockés dans un dictionnaire Python `SETTINGS_DATA`. Lors de l'instanciation de la classe Settings, ce dictionnaire est copié en profondeur via `copy.deepcopy()` pour éviter toute modification des valeurs par défaut lors de la session de jeu.

Quatre types de contrôles sont implémentés :

- Slider : curseur permettant de choisir une valeur entre 0 et 100 (utilisé pour le volume).
- Toggle : bouton ON/OFF à double état (utilisé pour la musique et le plein écran).
- Multi : sélecteur multi-options à cases juxtaposées (utilisé pour la résolution).
- Keybind : touche clavier remappable avec mode d'écoute actif (utilisé pour les contrôles).

### *Charte graphique du menu Settings*

La charte graphique du menu Settings est cohérente avec l'identité visuelle générale du jeu. Le fond est noir (#0A0A0A). Les onglets inactifs sont dans un gris très sombre (#1C1C1C), l'onglet actif est en rouge accentué (#BB0A1E) avec un liseré blanc sur son bord gauche.

## B — Onglet Audio

### *Master Volume*

Le volume général est contrôlé par un slider horizontal. La valeur par défaut est fixée à 25 (sur une échelle de 0 à 100), ce qui correspond à un volume modéré adapté à la plupart des environnements de jeu. Le slider est composé d'une barre de fond grise, d'une portion rouge représentant la valeur actuelle, et d'un cercle blanc servant de poignée de manipulation.

Techniquement, le glissement est géré par l'événement `MOUSEBUTTONDOWN` qui active l'état `self._slider_drag`, puis par la boucle de rendu qui met à jour la valeur en continu tant que le bouton de la souris reste enfoncé.

### *Musique (toggle ON/OFF)*

Le paramètre Musique permet d'activer ou de désactiver la musique de fond indépendamment des effets sonores. Il est représenté par un toggle à deux états : ON et OFF. L'état actif est coloré en rouge accentué, l'état inactif en gris sombre. La valeur par défaut est `True` (musique activée).

## C — Onglet Vidéo

### *Résolution*

Quatre résolutions sont proposées via un sélecteur multi-options : 1280×720 (HD), 1600×900, 1920×1080 (Full HD) et 2560×1440 (2K). La résolution par défaut est 1280×720, qui correspond à la résolution d'initialisation de la fenêtre définie dans `main.py`.

## ***Plein écran (Fullscreen toggle)***

Le toggle Fullscreen active ou désactive le mode plein écran via le flag `pygame.FULLSCREEN` passé à `pygame.display.set_mode()`. La valeur par défaut est `False` (mode fenêtré). Ce paramètre est particulièrement important pour les joueurs souhaitant une expérience immersive sur grand écran.

## **D — Onglet Contrôles**

L'onglet Contrôles offre un système de remappage des touches clavier entièrement personnalisable. Quatre actions de déplacement sont configurables :

- **MOVE UP** : déplacement vers le haut (touche par défaut : Z, conforme au clavier AZERTY)
- **MOVE DOWN** : déplacement vers le bas (touche par défaut : S)
- **MOVE LEFT** : déplacement vers la gauche (touche par défaut : Q, AZERTY)
- **MOVE RIGHT** : déplacement vers la droite (touche par défaut : D)

## ***Mécanisme de remappage***

Le remappage fonctionne via un système d'écoute (listening mode). Lorsqu'un joueur clique sur le bouton d'une action, l'option entre en mode d'écoute : le bouton devient noir avec le texte « APPUIE SUR UNE TOUCHE... ». Toute pression sur une touche (autre qu'Échap qui annule) est alors capturée et enregistrée comme nouvelle touche pour l'action concernée.

## **E — Sauvegarde et persistance des paramètres**

La sauvegarde des paramètres entre les sessions de jeu est identifiée comme une tâche urgente dans le planning du projet. Dans l'état actuel du développement, les paramètres sont stockés en mémoire vive uniquement :

ils sont initialisés à leurs valeurs par défaut au lancement du jeu et perdus à la fermeture.

### ***Architecture prévue***

La solution prévue pour la persistance est l'utilisation du module json de Python, qui permet de sérialiser le dictionnaire SETTINGS\_DATA vers un fichier texte au format JSON, et de le désérialiser au prochain lancement pour restaurer les préférences de l'utilisateur. Le fichier de sauvegarde sera nommé settings.json et placé dans le répertoire de données de l'application.

### ***Implémentation à venir***

Concrètement, deux méthodes seront ajoutées à la classe Settings : save\_settings() sera appelée à la fermeture du menu et écrira le dictionnaire de données dans settings.json ; load\_settings() sera appelée à l'initialisation de la classe et remplacera les valeurs par défaut par les valeurs sauvegardées si le fichier existe. Une gestion d'erreur appropriée permettra de gérer gracieusement les cas où le fichier est absent ou corrompu.

## **F — Difficultés rencontrées**

### ***Adaptation des dimensions des sprites avec Tiled***

L'une des difficultés majeures rencontrées lors de la création des maps a été l'adaptation des dimensions des sprites à l'échelle de Tiled. Les tilesets utilisés définissent des tuiles de tailles variables selon les maps (32×32 px pour la plage, 48×48 px pour le Japon), ce qui impliquait que les objets placés dans l'éditeur ne correspondaient pas exactement à leur rendu final dans Pygame. Ce problème a été résolu en ajustant manuellement les offsets des couches dans Tiled et en calibrant les paramètres de chargement via pytmx pour chaque map.

### ***Gestion des couches entre le joueur et la map***

La gestion de la profondeur visuelle entre le joueur et les éléments de la map a représenté un défi technique important. Dans les premières versions, le joueur apparaissait systématiquement au-dessus de tous les éléments, traversant visuellement les murs et les toitures. La solution a consisté à organiser rigoureusement l'ordre de rendu des couches dans pycscroll.

### ***Difficultés avec les paramètres visuels du menu Settings***

Le module Settings a posé des défis importants liés à l'adaptation de son interface à différentes résolutions d'écran. Lors des premiers tests, les éléments graphiques du menu étaient positionnés avec des coordonnées fixes calculées pour 1280×720, provoquant des décalages visuels et des zones de clic incorrectes sur d'autres résolutions.

## **Conclusion — Boyer Etienne**

La partie graphique et settings de Deathrow représente un travail conséquent qui touche à la fois à la création artistique et au développement technique. La conception des maps dans Tiled, avec leur organisation en couches multiples, leurs systèmes de collision et leurs spawners, a demandé une réflexion approfondie sur l'équilibre entre l'esthétique et la jouabilité.

Chaque map est pensée comme un écosystème visuel cohérent : la map plage avec ses tons chauds et son ambiance côtière post-apocalyptique, et la map Japon avec son architecture traditionnelle contrastant avec l'invasion zombie. L'utilisation de deux outils complémentaires — Tiled pour la composition des niveaux et Piskel pour la création des sprites — a permis de maintenir une pipeline de création efficace malgré les contraintes de temps.

# TOPIA Aurélien

---

## 2.3 Musique et effets sonores (Responsable)

### A — Musique principale

La musique principale du jeu se retrouve dans le menu principal. C'est une musique expérimentale qui a pour but de faire ressentir l'atmosphère que le joueur va ressentir lors d'une partie. La musique commence calmement avec presque aucun son. Rapidement un son strident apparaît, représentant la venue des zombies lors du début de la partie.

Plus la musique avance, plus elle s'intensifie comme les zombies au cours de la partie. À partir d'un moment, une mélodie fait surface, elle s'accorde avec les sons stridents, cela représente les joueurs qui mettent en place des stratégies pour dompter les hordes de zombies qui sont de plus en plus violentes. Le but de cette musique est de représenter de manière sonore ce à quoi une partie ressemble.

La musique en elle-même a été créée grâce à l'outil SUNO. C'est une intelligence artificielle qui permet de générer une musique complète grâce à un prompt. Le choix de l'utilisation de l'Intelligence artificielle a été un moyen de créer la musique souhaitée sans nécessiter des compétences particulières en production musicale.

### B — Sons zombies

En jeu, les zombies produisent des sons afin que le joueur puisse être alerté de leur présence. Il y a au total 21 sons de zombies différents. Ces sons sont tous libres de droit et ont été téléchargés sur un site répertoriant des bibliothèques de sons libres de droit créés par la communauté.

## C — Sons d'interaction

Les sons d'interaction sont principalement l'achat des armes en jeu. Il y a au total 2 sons :

- Achat : lorsque le joueur peut acheter une arme ou des munitions, le son d'achat se lance.
- Erreur : si le joueur n'a pas assez de points, le son d'erreur se lance pour lui faire comprendre qu'il n'est pas en capacité d'acheter l'objet.

Ces sons sont tous libres de droit et ont été téléchargés sur un site répertoriant des bibliothèques de sons libres de droit créés par la communauté.

## D — Son bonus

Le son des bonus est très simple, il a pour but d'informer le joueur de la présence d'un bonus sur la carte. Comme il a été difficile de trouver un son pour représenter un Bonus, nous avons choisi de prendre un son représentant un appareillage électronique défaillant. Ce son est libre de droit.

Difficulté rencontrée : le son des bonus est responsable d'un bug sonore en jeu qui parfois ne se termine pas lorsque le bonus a disparu de la carte. C'est un bug mineur qui n'impacte pas le gameplay.

## E — Son de manches

Le son des manches se déclenche à chaque changement de manche et a pour but de prévenir le joueur du changement de la manche. Nous avons choisi d'utiliser un son unique qui représente une alarme alertant de l'arrivée des zombies.

## 2.4 Graphismes (Suppléant)

## A — Bonus

Les bonus ont été désignés en fonction de leur effet. Par exemple, le bonus « Full Ammo » est représenté par une caisse de munitions car son effet est de recharger toutes les armes du joueur.

Les bonus ont une couleur mélangeant le jaune et le vert afin de mettre un contraste avec les objets réels du jeu, car les bonus ne sont pas des objets physiques mais des orbes. Cela permet de les repérer plus facilement. Les bonus sont des assets libres de droit (src: Pixelrepo) téléchargés et édités afin de leur donner cette couleur si particulière.

## B — Atouts

Le design des atouts se divise en deux parties.

### *Cœur du boss*

Lorsqu'un boss est tué, il drop son cœur qui a des veines de couleur différente. Cette couleur représente l'atout que le joueur pourra extraire de ce cœur. Le cœur est un asset libre de droit téléchargé et édité afin d'ajouter des veines de couleur différente.

### *Fioles d'atouts*

Lorsque le joueur a extrait le sang du cœur d'un boss, il récupère une fiole — plus précisément un tube à essais — contenant un liquide de la même couleur que les veines du cœur. La couleur représente donc l'atout que le joueur a extrait :

- Vert : speed (vitesse accélérée)
- Rouge : résistance (Cuirassé)
- Violet : fourth weapon (quatrième arme)
- Bleu : résurrection automatique



- Orange : double shoot (tir double)
- Bleu clair : freeze (gel des zombies)

## C — Exsanguineur

Le design de l'exsanguineur représente un pistolet à cartouche dont la cartouche a été remplacée par une fiole qui contiendra l'atout. Une tache de sang a été ajoutée à la pointe de l'outil pour représenter son utilisation.

L'utilisation de ce modèle permet au joueur de comprendre facilement que l'objet permet d'extraire les atouts. L'exsanguineur a été téléchargé et édité (src: Pixelrepo) afin de remplacer la cartouche par une fiole et ajouter une tache de sang sur l'embout.

## D — Boss

Le design d'un boss reprend celui des zombies mais avec une tenue différente et une couleur des yeux différente. Le boss a une tenue beige avec des yeux cyan afin de le démarquer des autres zombies, signalant qu'il a subi une modification génétique. Les modifications ont été réalisées à l'aide du logiciel gratuit Piskel.

## 2.5 Site web (Responsable)

### A — Hébergement

Le site internet du projet est hébergé sur le domaine deathrow.topia.fr.

L'hébergeur est Hostinger, un hébergeur performant et peu coûteux pour ce type de projet.

### B — Contenu

Le site internet est développé à l'aide de l'outil WordPress. C'est un outil professionnel et très populaire utilisé par des millions de sites internet. Il

permet de créer facilement des pages et de gérer tous les paramètres du site.

Pour ce projet, nous avons intégré les pages suivantes :

- Page principale : page web sur laquelle on retrouve un compteur indiquant le temps restant avant la sortie du jeu. On y retrouve aussi des détails sur le jeu et un texte expliquant l'histoire.
- Page de téléchargement : page web dans laquelle on retrouve le lien pour télécharger le jeu. Cette page est accessible uniquement pour les membres inscrits sur le site.
- Pages de gestion du compte utilisateur : diverses pages permettant de se connecter, s'inscrire, consulter son profil et modifier ses informations.

## C — Front-end

Le front-end du site a été créé à l'aide du thème KadenceWP. C'est un thème pour débutant très facile à prendre en main et qui permet d'ajouter et de modifier beaucoup d'éléments.

- Choix des couleurs : le site internet utilise principalement les couleurs noir, blanc et rouge. Le noir sert de fond. Le blanc est principalement utilisé pour le texte et les icônes. Le rouge est utilisé sur les icônes et le texte lors du survol avec la souris.
- Choix des polices : la police d'écriture principale est « Rubik Glitch ».

## 2.6 LAN — Mode réseau multijoueur

Le mode multijoueur de Deathrow permet à deux joueurs d'affronter ensemble les vagues de zombies en réseau local. L'architecture réseau repose sur le protocole WebSocket, implémenté en Python via la bibliothèque websockets. L'ensemble de la logique réseau est centralisé dans deux

modules : `encode.py` pour le chiffrement des codes de connexion, et `network.py` pour la gestion des connexions WebSocket.

## Architecture hôte / client

Le modèle réseau adopté est une architecture pair-à-pair asymétrique où l'un des joueurs prend le rôle d'hôte (host) et l'autre de client. L'hôte fait autorité sur l'ensemble de la simulation : il calcule les déplacements des zombies, les dégâts, les collisions et la progression des manches. Il envoie un état complet au client toutes les 3 frames (`SYNC_INTERVAL = 3`), soit environ 20 fois par seconde à 60 FPS.

Cette approche présente l'avantage d'être simple à implémenter et d'éviter les problèmes de désynchronisation : le client ne simule jamais les zombies de son côté, il affiche uniquement des sprites visuels positionnés selon les données reçues de l'hôte.

## Protocole de communication

Les messages échangés entre hôte et client sont sérialisés en JSON. Quatre types de messages sont définis :

- `state` (hôte → client) : état complet de la partie — positions des joueurs, santé, angle des armes, liste des zombies, numéro de manche.
- `input` (client → hôte) : position du joueur client, direction, angle de l'arme.
- `bullet` (client → hôte) : coordonnées monde, angle et dégâts d'une balle tirée par le client.
- `game_over` (hôte → client) : notification de fin de partie avec le nombre de manches atteintes et de zombies éliminés.

## Chiffrement des codes de connexion

Pour rejoindre une partie, le client doit saisir un code de connexion généré par l'hôte. Ce code encode l'adresse IP et le port du serveur WebSocket de l'hôte en un format court et lisible. Le chiffrement est réalisé via la classe `CodecXOR` dans `encode.py` : l'adresse IP (4 octets via `socket.inet_aton`) et le port (2 octets big-endian via `struct.pack`) sont combinés en 6 octets, puis chiffrés par XOR avec la clé fixe `0xDEADB0FFCAFE`.

Le résultat est encodé en base36 et formaté en deux blocs de 5 caractères séparés par un tiret (ex. `XXXXX-XXXXX`), donnant un code court facile à transmettre oralement.

## Gestion des threads et de l'event loop asyncio

La communication WebSocket est entièrement asynchrone et s'exécute dans un thread daemon dédié, séparé du thread principal du jeu Pygame. Cela garantit que les opérations réseau (envoi, réception) ne bloquent jamais la boucle de jeu principale. Les méthodes `send()` et `get_messages()` du thread principal sont thread-safe et non bloquantes, reposant sur une queue.Queue Python.

## Gestion du pare-feu Windows

Lors du démarrage du serveur WebSocket côté hôte, le port TCP sélectionné est automatiquement ouvert dans le pare-feu Windows via la commande `netsh advfirewall firewall add rule`. Cette opération nécessite des droits administrateur : si le jeu n'est pas lancé en tant qu'administrateur, un avertissement est affiché dans les logs.

## 2.7 Map — Gestion map + Pygame (Suppléant)

En tant que suppléant de la partie Map, j'ai participé à l'intégration technique des maps dans le moteur de jeu Pygame. Cette intégration repose sur deux

bibliothèques spécialisées : pytmx pour le chargement des fichiers TMX produits par Tiled, et pyscroll pour le rendu optimisé avec gestion de caméra.

## Chargement des maps avec pytmx

Chaque map est chargée depuis un fichier TMX via la fonction `pytmx.util_pygame.load_pygame()`, qui lit le fichier XML et charge automatiquement les tilesets associés en surfaces Pygame. Le résultat est encapsulé dans un objet `TiledMapData` de `pyscroll`. Le chemin de la map est construit dynamiquement selon l'identifiant de map sélectionné par le joueur (`gdt`, `dotb`, `jnt`).

## Rendu de la tilemap avec pyscroll

Le rendu de la tilemap est assuré par `pyscroll` via la classe `BufferedRenderer`, qui gère le rendu optimisé des couches de tuiles avec buffering pour éviter de recalculer l'intégralité de la carte à chaque frame. Le niveau de zoom est appliqué directement sur le renderer (`map_layer.zoom`), et la caméra est recentrée sur le joueur à chaque frame via `group.center(player.rect.center)`.

## Extraction des objets de jeu depuis le TMX

Les maps TMX contiennent non seulement les couches graphiques mais aussi des couches d'objets définissant les éléments interactifs de la map. Ces objets sont extraits à l'initialisation du moteur de jeu en parcourant `tmx_data.objects` :

- Spawners de zombies : objets de type 'spawners', dont les coordonnées (x, y) sont ajoutées à la liste `self.zombies.spawners`.
- Collisions : objets de type 'collision', convertis en `pygame.Rect` et stockés dans `self.walls`.
- Zones d'achat : objets de type 'buy\_zone', associant un rectangle de proximité et les propriétés de l'arme disponible.

- Outil exsangueux : objets nommés 'tool', avec sélection aléatoire d'un emplacement à chaque partie.
- Position de départ du joueur : objet nommé 'player', utilisé pour positionner le personnage à l'initialisation.

## Système de navigation pour les zombies

Pour permettre aux zombies de naviguer intelligemment dans la map, une grille de navigation est générée à l'initialisation depuis les données de collision. La grille est construite en divisant la surface de la map en cellules de taille  $\max(8, \text{tilewidth})$  pixels, puis en marquant comme infranchissables toutes les cellules intersectant une zone de collision. Cette grille est transmise au module de gestion des zombies via `Zombies_global.set_nav_data()`.

## 2.8 Développement du moteur de jeu

### A — Menu principal

Le menu principal de Deathrow est implémenté dans la classe Menu du fichier `menu.py`. Il constitue le premier écran affiché au lancement du jeu et permet au joueur de naviguer vers toutes les fonctionnalités disponibles.

Le menu affiche cinq boutons de navigation centrés verticalement : Maps, Host, Join, Settings et Quit. La navigation entre les écrans est gérée par le système `next_screen` dans `main.py`.

### *Charte graphique et typographie*

La charte graphique du menu utilise une palette sombre cohérente avec l'univers post-apocalyptique. Le fond est noir (`#0A0A0A`). Les boutons sont transparents et affichent leur texte en blanc, qui passe au rouge (`#DC1E32`) au survol. Le titre DEATHROW est affiché en grand format (100pt). La police utilisée est Rubik Glitch.

## B — Gestion de la carte

La gestion de la carte en mémoire couvre l'ensemble des données de jeu liées à l'environnement : murs de collision, zones d'achat, spawners et outil exsanguineur. Ces données sont extraites de la map TMX à l'initialisation et stockées dans des structures optimisées pour la détection de collision en temps réel.

### *Collisions avec les murs*

Les collisions entre les entités et les éléments statiques de la map sont détectées via la méthode `pygame.Rect.collidelist()`. Pour chaque sprite du groupe `pyscroll`, la hitbox `self.feet` est testée contre la liste `self.walls` à chaque frame. En cas de collision, la méthode `move_back()` est appelée pour restaurer la position précédente.

### *Zones d'achat*

Les zones d'achat sont des rectangles légèrement plus grands que leur zone de collision physique (agrandis de 2 pixels) pour détecter la proximité du joueur avant tout contact. Lorsque le joueur entre dans cette zone élargie, le prompt d'achat est affiché en bas de l'écran avec le nom et le prix de l'arme disponible.

## C — Gestion du joueur

### *Système de santé et mort*

La santé du joueur est représentée par l'attribut `health`, initialisé à 100. Chaque attaque de zombie réduit cet attribut de 25 points. Lorsque la santé atteint 0, la méthode `check_player_death()` du moteur de jeu déclenche la séquence de fin de partie. Avant de déclarer la mort, le moteur vérifie si le joueur possède l'atout de résurrection.

### *Régénération de santé*

Un système de régénération passive est implémenté via la méthode `_regen_health()` : toutes les 2000 millisecondes, si le joueur est en vie et n'est pas à pleine santé, 25 points de vie sont restaurés. Ce système récompense les joueurs qui évitent les dégâts et apporte une dimension stratégique supplémentaire.

### ***Flash de dégâts***

Un retour visuel immédiat sur les dégâts subis est fourni par le système de flash de dégâts. Une surface pré-calculée composée de quatre rectangles rouges sur les bordures de l'écran est affichée avec un alpha élevé (220) dès qu'une réduction de santé est détectée. L'alpha décroît de 7 unités par frame, produisant un fondu en environ 31 frames (0,5 secondes).

## **D — Gestion des manches**

Le système de manches organise le déroulement de la partie en vagues successives d'ennemis de difficulté croissante. La fin d'une manche est détectée lorsque la liste `self.zombies.zombies` est vide, signifiant que tous les zombies ont été éliminés. Un countdown de 3 secondes est déclenché et le son d'alarme de changement de manche est joué.

## **E — Gestion de la difficulté**

La progression de la difficulté est calculée automatiquement par la classe `Round` à chaque changement de manche, selon trois paramètres indépendants : le nombre de zombies, leurs points de vie et leur vitesse de déplacement.

Les points de vie des zombies augmentent de 10 % à chaque manche via la formule  $hp = \text{floor}(hp * 1.10)$ . La vitesse suit une fonction exponentielle convergente :  $speed = 2 - 1.9 * \exp(-0.05 * \text{round})$ , garantissant une augmentation rapide dans les premières manches et une stabilisation autour de 2 unités par frame.



## F — Gestion des armes

### *Rotation et suivi de la souris*

L'arme suit en permanence la position de la souris via un calcul trigonométrique. L'angle est calculé via `math.atan2(-dy, dx)`. L'arme est pivotée de cet angle via `pygame.transform.rotate()` et retournée horizontalement lorsque la souris est à gauche du joueur.

### *Système de tir*

La méthode `try_fire()` de la classe `Weapon` gère deux modes de tir. En mode semi-automatique (`automatic = False`), le tir n'est déclenché que sur `trigger_just_pressed`. En mode automatique (`automatic = True`), le tir se produit à chaque frame où `trigger_held` est vrai, permettant une cadence continue.

### *Rechargement*

Le rechargement est déclenché par la touche R via `weapon.reload()`. Un timer de `RELOAD_TIME = 90` frames (~1,5 secondes) est déclenché. Pendant ce délai, aucun tir n'est possible. Un message visuel 'RECHARGEMENT...' accompagné d'une barre de progression est affiché dans le HUD.

## G — Animation des personnages

### *Animation du joueur*

Le personnage joueur est animé sur 6 frames par direction. La méthode `change_animation()` incrémente un compteur interne et avance d'une frame toutes les 5 itérations, produisant une animation fluide à 12 images par seconde. Deux listes de frames sont maintenues : les frames normales et les frames retournées horizontalement.

### *Animation des zombies et animation de mort*

Les zombies disposent du même système d'animation que le joueur.

Lorsqu'un zombie est éliminé, une animation de mort est déclenchée via la classe `DeathAnimation`, qui découpe une sprite sheet GIF frame par frame et la restitue à l'emplacement de la mort.

### ***Séparation des zombies***

Un système anti-chevauchement est implémenté via la méthode `_separate_zombies()`, qui parcourt toutes les paires de zombies et applique une force de répulsion lorsque deux entités sont à moins de 28 pixels l'une de l'autre. Ce mécanisme maintient une séparation visuelle constante entre les entités ennemies.

## **H — Déplacement du personnage**

### ***Déplacement du joueur***

Le déplacement est géré dans `handle_input()` via `pygame.key.get_pressed()`. Les quatre touches directionnelles (Z, Q, S, D par défaut, remappables via les Settings) déplacent le joueur de `player.speed` pixels par frame. Les contrôles sont lus depuis `settings_manager.get_controls()` à chaque frame.

### ***Détection de collision***

Après chaque déplacement, la hitbox `self.feet` du joueur est testée contre la liste des murs via `feet.collidelist()`. En cas de collision, la position précédente est restaurée via `move_back()`, qui lit `self.old_position` sauvegardée au début de chaque frame par `save_location()`.

### ***Synchronisation réseau***

En mode multijoueur, la position du joueur est incluse dans chaque message réseau envoyé à l'hôte (type input). L'hôte met à jour la position du sprite `RemotePlayer` correspondant via `update_state()`, qui met à jour simultanément la position, la direction et l'angle de l'arme du joueur distant.

# MOHAJERI Giv

---

## 2.9 Intelligence Artificielle (Responsable)

L'intelligence artificielle des zombies constitue le cœur de l'expérience de jeu de Deathrow. En tant que responsable de l'IA, j'ai conçu et implémenté l'ensemble du comportement des entités ennemies : leur déplacement vers les joueurs, leur système d'attaque, leur gestion en horde et leur adaptation au mode multijoueur.

Toute la logique de l'IA est implémentée dans le fichier `zombies.py`, organisé en deux classes distinctes : `Zombie_entity` qui gère le comportement individuel de chaque zombie, et `Zombies_global` qui orchestre l'ensemble de la horde.

### A — Architecture générale de l'IA

#### *Approche choisie*

L'IA des zombies de Deathrow repose sur un modèle comportemental simple et efficace : chaque zombie est un agent autonome qui prend ses décisions individuellement à chaque frame, sans communication avec les autres zombies. Cette approche, dite 'agent-based', est facile à implémenter et à déboguer, elle passe à l'échelle naturellement (ajouter 50 zombies ne change pas la logique), et elle produit des comportements de horde émergents intéressants.

#### *Cycle de vie d'un zombie*

À chaque frame, la méthode `update()` de `Zombie_entity` exécute séquentiellement trois opérations : `save_location()` qui sauvegarde la position

courante pour le système de rollback en cas de collision, `follow_player()` qui déplace le zombie vers sa cible, et `attack_player()` qui vérifie si le zombie est à portée d'attaque et déclenche les dégâts si nécessaire.

## B — Déplacement vers le joueur le plus proche

### *Algorithme de suivi*

Le déplacement de chaque zombie est régi par un algorithme de poursuite directe (direct pursuit). À chaque frame, le zombie calcule le vecteur directeur vers sa cible, normalise ce vecteur pour obtenir une direction unitaire, puis déplace le zombie de `self.speed` pixels dans cette direction :

$dx = (target.x - zombie.x) / distance$ ,  $dy = (target.y - zombie.y) / distance$ , puis  $zombie.x += dx * speed$ ,  $zombie.y += dy * speed$ .

La normalisation du vecteur (division par la distance) est essentielle : sans elle, les zombies proches de leur cible se déplaceraient lentement tandis que les zombies éloignés se déplaceraient rapidement.

### *Sélection de la cible : le joueur le plus proche*

La méthode `_nearest_player()` parcourt la liste `self.players` et calcule la distance euclidienne via `math.hypot()` pour chaque joueur. Le joueur avec la distance minimale est retourné comme cible. Cette sélection est recalculée à chaque frame, permettant aux zombies de changer de cible dynamiquement.

La méthode inclut une gestion robuste des erreurs : si un joueur ne possède pas d'attribut 'position', l'exception est silencieusement ignorée via un bloc `try/except` et ce joueur est simplement exclu du calcul.

### *Relation distance-vitesse*

La vitesse des zombies est un paramètre global géré par la classe `Zombies_global`. La formule de progression ( $speed = 2 - 1.9 * \exp(-0.05 * round)$ ) garantit une montée rapide dans les premières manches et une

stabilisation autour de 2 unités par frame, soit 120 pixels par seconde à 60 FPS.

## C — Système d'attaque

### *Détection de portée*

L'attaque est déclenchée lorsque la distance entre le zombie et sa cible est inférieure à `self.attack_range = 30` pixels. Ce seuil correspond approximativement à la largeur d'un sprite de personnage (32 pixels), garantissant que le zombie doit être visuellement au contact de sa cible pour infliger des dégâts.

### *Cooldown d'attaque*

Sans cooldown, un zombie au contact du joueur infligerait des dégâts à chaque frame, soit 60 fois par seconde. Le système de cooldown résout ce problème : chaque zombie maintient un timestamp `self.last_attack`. Une nouvelle attaque n'est autorisée que si `pygame.time.get_ticks() - last_attack > attack_cooldown`, où `attack_cooldown = 500` millisecondes. Cela limite les attaques à 2 fois par seconde maximum par zombie.

### *Dégâts et effet sonore*

Lorsqu'une attaque est autorisée, deux actions sont déclenchées simultanément : la santé du joueur cible est réduite de `self.damage = 25` points, et le son d'attaque de zombie est joué via `self.sound.play_zombie_attack()`.

## D — Gestion multi-cibles en multijoueur

### *Architecture de la liste de cibles*

L'une des évolutions majeures de l'IA est la gestion du mode multijoueur. La solution implémentée normalise systématiquement le paramètre `players` en liste dans le constructeur de `Zombie_entity` : `if not isinstance(players, list):`

players = [players]. Ainsi, qu'on passe un seul joueur ou une liste de joueurs, le zombie stocke toujours une liste `self.players`.

### *Ajout dynamique d'un joueur*

En mode multijoueur, le client rejoint la partie après que l'hôte a déjà généré les premiers zombies. La méthode `add_player_to_zombies()` de `Zombies_global` résout ce problème en parcourant tous les zombies existants et en ajoutant le nouveau joueur à leur liste de cibles via `z.players.append(player)`.

## **E — Grille de navigation**

### *Problématique*

La poursuite directe présente une limitation fondamentale : les zombies se dirigent en ligne droite vers leur cible, sans tenir compte des obstacles. Dans une map avec des murs et des bâtiments, cela produit des comportements absurdes où les zombies restent bloqués contre les murs au lieu de les contourner.

### *Construction de la grille*

La grille de navigation est construite dans le moteur de jeu lors du chargement de la map. La map est découpée en cellules de taille `_nav_tile = max(8, tmx_data.tilewidth)` pixels. Pour chaque mur de collision, toutes les cellules intersectant ce rectangle (avec un padding d'une cellule autour de chaque côté) sont marquées comme infranchissables.

### *Utilisation par les zombies*

La grille de navigation sert de base à l'amélioration du déplacement des zombies. Ce système constitue la base sur laquelle des algorithmes de pathfinding plus sophistiqués (comme A\*) pourront être intégrés dans des versions futures du jeu, sans nécessiter de refonte de l'architecture.

## F — Génération et gestion de la horde

### *Classe Zombies\_global*

La classe `Zombies_global` orchestre l'ensemble de la population de zombies. Elle maintient la liste `self.zombies` contenant toutes les instances actives, la liste `self.spawners` contenant les coordonnées des points d'apparition, et les paramètres globaux : nombre de zombies (`self.number = 6` au départ), points de vie (`self.hp = 100`) et vitesse (`self.speed = 0.1`).

### *Génération cyclique depuis les spawners*

La méthode `generate_zombies(n, players)` crée `n` instances de `Zombie_entity`, en les répartissant cycliquement entre les spawners disponibles via l'expression `(i % spawners_number)`. Cette répartition cyclique garantit que les zombies apparaissent de façon équilibrée autour de la map.

## G — Le boss

### *Différences avec les zombies standards*

Le boss est généré via la méthode `generate_boss()` de `Zombies_global`, qui crée une instance de `Zombie_entity` avec des paramètres spécifiques. Fonctionnellement, le boss démarre avec seulement 5 points de vie mais une vitesse de déplacement de 0,6 unités par frame, soit 6 fois plus rapide que les premiers zombies. Ce profil 'faible mais rapide' le rend particulièrement dangereux en fin de partie.

### *Mort du boss et drop d'atout*

La mort du boss déclenche une mécanique spéciale dans le moteur de jeu : un atout est généré à l'emplacement du boss via la classe `Perks_utils`. La méthode `generate_perks()` sélectionne aléatoirement un atout parmi les six disponibles. L'atout apparaît sous forme d'un cœur coloré sur la map.

## H — Difficultés rencontrées

### *Zombies bloqués contre les murs*

La principale difficulté rencontrée dans le développement de l'IA a été le comportement des zombies face aux obstacles. Avec la poursuite directe, les zombies se bloquaient systématiquement contre les murs lorsque leur chemin vers le joueur était obstrué : ils continuaient à essayer d'avancer dans la direction de leur cible, mais le système de collision les renvoyait en arrière via `move_back()`, créant un effet de vibration sur place.

### *Synchronisation IA en multijoueur*

La synchronisation de l'IA en multijoueur a posé des défis spécifiques. Côté client, les zombies sont de simples sprites visuels sans logique d'IA. Toute la simulation IA tourne exclusivement sur la machine de l'hôte, ce qui peut créer des décalages perceptibles côté client lors de pics de latence réseau.

### *Calibrage de la difficulté*

Le calibrage des paramètres de l'IA (vitesse, dégâts, cooldown d'attaque, portée) a nécessité de nombreuses sessions de test. Les valeurs finales (speed = 0,1 à 2,0, damage = 25, cooldown = 500ms, range = 30px) ont été validées empiriquement sur plusieurs maps et niveaux de difficulté.

## 2.10 Développement des fonctionnalités (Suppléant)

En tant que suppléant du développement des fonctionnalités, j'ai contribué à l'implémentation de trois fonctionnalités clés : le système de balles, la mort du joueur, et les deux atouts restants (résurrection et freeze).

## A — Système de balles

### *Classe Bullet*



Le système de balles est implémenté via la classe `Bullet` dans `weapons.py`. Contrairement aux entités du jeu comme le joueur ou les zombies, les balles ne sont pas des sprites Pygame : elles sont gérées directement dans une liste `self.bullets` du moteur de jeu et rendues manuellement à chaque frame.

Chaque balle est initialisée avec une position monde (`world_pos`), une vitesse calculée depuis l'angle de tir (`velocity = Vector2(cos(angle), -sin(angle)) * SPEED`), et des dégâts (`damage`). La constante `SPEED = 10` pixels monde par frame correspond à 600 pixels par seconde à 60 FPS. La durée de vie maximale est de `LIFETIME = 150` frames (~2,5 secondes).

### ***Mise à jour et rendu***

À chaque frame, la méthode `update()` de chaque balle ajoute la vitesse à la position monde et décrémente le compteur de durée de vie. La méthode `draw()` convertit les coordonnées monde en coordonnées écran et dessine un cercle jaune de rayon 3 pixels.

### ***Détection de collision balle-zombie***

La détection de collision entre les balles et les zombies est gérée par la méthode `_check_bullet_hits()` du moteur de jeu. Pour chaque balle active, un test de collision est effectué contre tous les zombies actifs. En cas de collision, la balle est désactivée et les points de vie du zombie sont réduits via `lose_hp()`. Si les points de vie tombent à 0 ou en dessous, le zombie est éliminé.

## **B — Mort du joueur**

### ***Détection et déclenchement***

La mort du joueur est détectée dans la méthode `check_player_death()` appelée à chaque frame. La condition de mort est simplement `player.health <= 0`. Avant de déclarer la mort définitive, la méthode `_try_resurrection()` vérifie si le joueur possède l'atout de résurrection.

## ***Écran de Game Over***

L'écran de Game Over est affiché en superposition semi-transparente sur le dernier état du jeu. Il présente le texte 'GAME OVER' en rouge, le numéro de manche atteint et le nombre de zombies éliminés. Un bouton 'Back to menu' ramène au menu principal.

## ***Mort en multijoueur***

En mode multijoueur, la mort d'un seul joueur ne déclenche pas immédiatement le game over. Le joueur mort est figé et attend la fin de la manche. Au début de la manche suivante, `_on_new_round()` réinitialise la santé des deux joueurs à 100. Le game over n'est déclenché que lorsque les deux joueurs sont morts simultanément.

## **C — Atouts : résurrection et freeze**

### ***Résurrection (atout bleu)***

L'atout de résurrection (id = 1) offre au joueur une seconde vie automatique lors de sa prochaine mort. Son implémentation repose sur la méthode `_try_resurrection()` dans le moteur de jeu. Lorsque la mort du joueur est détectée, cette méthode parcourt les slots d'atouts et, si l'atout est trouvé, la santé est portée à 999999 pendant 5 secondes.

Un second atout actif est également retiré aléatoirement de l'inventaire comme 'coût' de la résurrection, équilibrant mécaniquement la puissance de cet atout. Cette période d'invincibilité temporaire donne au joueur le temps de se repositionner.

### ***Freeze (atout bleu clair)***

L'atout freeze (id = 6) a pour vocation de ralentir les zombies environnants lorsqu'il est activé. Son implémentation dans la classe Perks est marquée comme une fonctionnalité à compléter. L'architecture est en place : l'atout est correctement généré, affiché et équipé par le joueur. L'effet de ralentissement

sera implémenté en modifiant temporairement la vitesse des zombies dans un rayon défini autour du joueur.

## D — Difficultés rencontrées

### *Collisions balles et échelle de map*

L'implémentation des collisions balle-zombie a nécessité une adaptation au système de coordonnées monde utilisé par le moteur de jeu. Les positions des balles et des zombies sont en coordonnées monde (non zoomées), mais le rendu à l'écran applique un facteur de zoom. Le rectangle de collision du zombie devait donc être construit en coordonnées monde ( $\text{int}(28 * \text{sprite\_scale})$ ) et non en coordonnées écran.

### *Équilibrage de la mort du joueur*

La calibration des dégâts des zombies (25 points par attaque) et du cooldown (500ms) a été déterminée empiriquement. Avec une santé de 100, un joueur en contact avec un zombie subit 50 dégâts par seconde, le tuant en 2 secondes. Face à deux zombies simultanément, il meurt en 1 seconde. Ces valeurs ont été ajustées après plusieurs sessions de test.

## 2.11 Programme d'installation (Responsable)

En tant que responsable du programme d'installation, j'ai pris en charge la mise en place d'un pipeline de distribution complet pour Deathrow sur Windows. L'objectif était de permettre à n'importe quel utilisateur de télécharger et d'installer le jeu en quelques clics, sans avoir à installer Python ni aucune dépendance.

## A — Contexte et objectif

### *Problématique de distribution d'un jeu Python*

Un jeu développé en Python présente un défi de distribution majeur : le langage Python n'est pas nativement installé sur la plupart des ordinateurs Windows grand public, et les bibliothèques utilisées par le jeu (pygame, pytmx, pycscroll, websockets) doivent également être présentes. La solution standard dans l'écosystème Python est d'utiliser PyInstaller.

### ***Objectif final***

L'objectif final était de produire un fichier installateur Windows (DeathrowSetup.exe) que les membres inscrits sur le site peuvent télécharger et exécuter pour installer le jeu. L'installateur doit créer un raccourci sur le bureau, installer le jeu dans le répertoire Program Files, et permettre la désinstallation propre depuis les paramètres Windows.

## **B — PyInstaller : génération de l'exécutable**

### ***Présentation de PyInstaller***

PyInstaller est un outil open-source qui analyse un script Python et ses dépendances pour créer un exécutable autonome. Il fonctionne en deux étapes : d'abord une phase d'analyse qui identifie tous les modules Python importés et leurs dépendances transitives, puis une phase d'empaquetage qui regroupe l'interpréteur Python, tous les modules et les fichiers de données dans un unique exécutable.

Pour Deathrow, le mode 'onefile' a été choisi : toute l'application est compressée dans un unique fichier .exe. Ce choix simplifie la distribution (un seul fichier à télécharger) et l'intégration dans l'installateur Inno Setup.

### ***Fichier de configuration .spec***

PyInstaller est configuré via un fichier .spec (Deathrow.spec), un script Python décrivant précisément comment construire l'exécutable. Les éléments clés :

- La section `datas` spécifie les fichiers non-Python à inclure : `datas = [('assets', 'assets'), ('data', 'data'), ('Asset_format_map', 'Asset_format_map')]`.
- La section `collect_all('pygame')` force l'inclusion de tous les fichiers de pygame, incluant les composants binaires natifs (SDL).

### ***Paramètres de l'exécutable***

L'exécutable final est configuré avec plusieurs paramètres importants : `name='Deathrow'` (définit `Deathrow.exe`), `console=False` (désactive la fenêtre de console), `icon=['assets\icon.ico']` (définit l'icône), et `upx=True` (active la compression UPX pour réduire la taille du fichier).

## **C — Inno Setup : création de l'installateur Windows**

### ***Présentation d'Inno Setup***

Inno Setup est un outil gratuit et open-source créé par Jordan Russell, utilisé par de nombreux logiciels professionnels pour créer des installateurs Windows. Il génère un fichier `Setup.exe` autonome qui guide l'utilisateur à travers l'installation via une interface graphique professionnelle, installe les fichiers aux bons emplacements et fournit un programme de désinstallation propre.

### ***Configuration du script Inno Setup***

Le script Inno Setup de Deathrow est organisé en plusieurs sections :

- Section `[Setup]` : définit les métadonnées (`AppName=Deathrow`, `AppVersion=1.0`, `AppPublisher=Steampunk Games`, répertoire d'installation `{pf}\Deathrow`).
- Section `[Files]` : spécifie les fichiers à copier lors de l'installation. L'entrée principale copie `Deathrow.exe` avec le flag `ignoreversion`.

- Section [Icons] : crée les raccourcis bureau ({userdesktop}\Deathrow) et menu Démarrer ({group}\Deathrow).

## ***Désinstallation***

Inno Setup génère automatiquement un programme de désinstallation (unins000.exe) lors de l'installation. Ce programme est enregistré dans les paramètres Windows sous 'Applications et fonctionnalités', permettant une désinstallation propre et complète.

## ***Intégration avec le site web***

L'installateur final DeathrowSetup.exe est hébergé sur le site web du projet (deathrow.topia.fr) et accessible uniquement depuis la page de téléchargement, elle-même réservée aux membres inscrits. Cette restriction garantit un contrôle de la distribution.

# **D — Difficultés rencontrées**

## ***Inclusion des assets avec PyInstaller***

La première difficulté majeure rencontrée a été l'inclusion correcte des fichiers de données (assets, data, maps) dans l'exécutable PyInstaller. Dans un projet Python classique, les fichiers sont référencés par des chemins relatifs. Or, PyInstaller extrait les fichiers dans un répertoire temporaire au lancement, et les chemins relatifs ne correspondent plus à la structure d'origine. La solution a été d'utiliser `_HERE = Path(__file__).resolve().parent` dans tous les modules.

## ***Bibliothèques dynamiques de pygame***

pygame dépend de plusieurs bibliothèques dynamiques SDL (SDL2.dll, SDL2\_mixer.dll, SDL2\_image.dll) qui ne sont pas toujours détectées automatiquement par PyInstaller. La solution a été d'utiliser `collect_all('pygame')` dans le fichier .spec, qui force l'inclusion de toutes les bibliothèques pygame, y compris les DLL SDL.

### ***Taille de l'exécutable et compatibilité Windows***

L'exécutable PyInstaller en mode onefile incluant pygame et toutes ses dépendances SDL atteint une taille significative. La compression UPX a permis de réduire la taille finale. Les tests ont été effectués sur plusieurs configurations Windows pour valider la compatibilité, notamment concernant l'ouverture des ports réseau nécessaires au mode multijoueur.

## 3. Conclusion générale

---

Deathrow est un projet de jeu vidéo ambitieux et cohérent, développé intégralement en Python avec la bibliothèque Pygame. Ce rapport de projet illustre la diversité des compétences mobilisées par l'équipe Steampunk Games : création graphique et artistique, développement moteur de jeu, intelligence artificielle, réseau multijoueur, infrastructure web et déploiement logiciel.

Chaque membre a pris en charge plusieurs domaines techniques complémentaires, avec une organisation rigoureuse basée sur des responsables et suppléants pour chaque tâche. Cette structure a permis d'assurer la continuité du développement malgré les contraintes inhérentes à un projet conduit en parallèle de la formation.

### 3.1 Bilan par membre

#### **BOYER Etienne**

La contribution de Boyer Etienne couvre la création graphique (maps pixel art, sprites joueur et ennemis) ainsi que le module Settings avec son interface complète de configuration audio, vidéo et contrôles. Quatre maps ont été conçues dans Tiled, chacune avec une identité visuelle forte et un système de couches optimisé pour la jouabilité.

#### **TOPIA Aurélien**

La contribution de Topia Aurélien couvre l'ensemble de l'expérience sonore (musique générative, sons d'ambiance), la direction artistique des bonus et atouts, le site web communautaire, l'architecture réseau multijoueur en WebSocket, l'intégration technique des maps dans Pygame et le développement du moteur de jeu principal.



## MOHAJERI Giv

La contribution de Mohajeri Giv couvre l'intelligence artificielle des zombies (comportements agents, gestion de horde, grille de navigation), les fonctionnalités de gameplay (balles, mort du joueur, atouts résurrection et freeze), et le pipeline de distribution professionnel PyInstaller + Inno Setup permettant une installation simple sur Windows.

## 3.2 Perspectives et développements futurs

Les prochaines étapes prioritaires identifiées par l'équipe sont :

- Implémentation complète du pathfinding A\* pour permettre aux zombies de contourner les obstacles.
- Intégration du système de points et d'économie en jeu (achats d'armes, zones, outils extracteur).
- Finalisation de la sauvegarde des paramètres via JSON persistant.
- Développement des cartes restantes (Death on the Beach, Jigoku no Torii).
- Amélioration des animations (joueur, zombies, effets de mort).
- Implémentation de l'effet de l'atout Freeze.
- Création d'un HUD graphique complet (barre de vie, slots d'armes visuels, compteur de points).
- Tests de compatibilité approfondis du mode multijoueur sur différentes configurations réseau.

Le projet est bien engagé pour atteindre ses objectifs à la soutenance finale. L'architecture modulaire en Programmation Orientée Objet adoptée dès le début garantit l'extensibilité de l'ensemble et facilitera l'intégration des fonctionnalités restantes dans les délais impartis.

---

*DEATHROW • Steampunk Games • Boyer • Topia • Mohajeri*

---