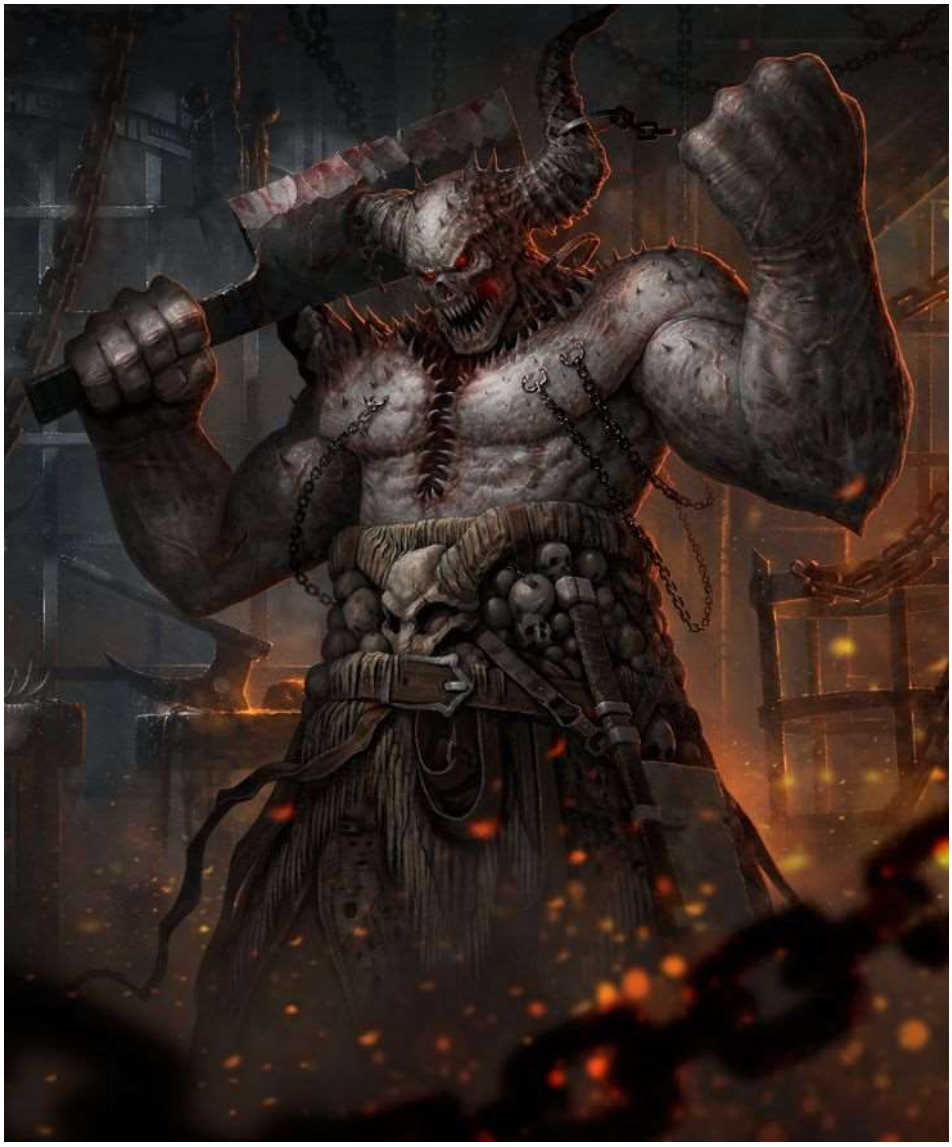


Demon Butcher

Projet P_ShootEmUp

Eliott Scherrer – MIN2A
ETML, Vennes – 2024

Table des matières

1 Analyse préliminaire

1.1 Introduction

Ce projet consiste à créer un jeu en 2D de type *Shoot 'Em Up* réalisé dans le cadre de différents modules (C106 : Base de données, 320 : Programmation orienté objet et 322 : Expérience utilisateur) réunis en un gros projet. Le choix de rassembler ces trois sujets en un projet ludique permet aux élèves d'en apprendre plus sur chacun de ces domaines et d'acquérir de l'expérience en étant immergés dans un projet concret.

1.2 Objectifs

UX : Réaliser les wireframes du menu principal, écran de jeu, éditeur de niveau et high scores. Approfondir l'editeur de niveau avec une maquette de type Mockup (High fidelity).

POO : Réaliser un jeu de type *Shoot 'Em Up* en 2D avec certaines fonctionnalités requises telles que :
Un joueur avec des capacités de déplacement et de tir, ainsi que des vies limitées. Des ennemis avec des caractéristiques variées (nombre de vies, minutage d'apparition, capacité de tir). Des obstacles avec des positions, tailles et comportements définis (en cas de tir ou de collision).

1.3 Gestion de projet

Pour gérer ce projet, nous utilisons IceScrum pour organiser les tâches, tenir le journal de travail, et créer des user stories ainsi que des tests d'acceptation. D'un autre côté, GitHub est utilisé pour le versionnement du code et le suivi des modifications au fil du temps.

2 Analyse / Conception

2.1 Gameplay

2.1.1 Le joueur

Le joueur pourra se déplacer librement en utilisant les touches directionnelles, lui permettant d'éviter les projectiles ennemis et de se positionner stratégiquement dans le niveau. En appuyant sur le clic gauche de la souris, le joueur déclenchera un tir dans la direction du curseur, avec une trajectoire linéaire partant de sa position actuelle. Un délai sera appliqué entre chaque tir pour introduire un cooldown, obligeant le joueur à gérer le timing de ses attaques.

Le joueur aura également la possibilité de placer une protection en appuyant sur le clic droit de la souris, lui offrant un moyen de défense temporaire pour bloquer les projectiles ou ralentir les ennemis.

2.1.2 Les ennemis

Les ennemis apparaissent dans le niveau à partir de spawners, introduisant progressivement des vagues d'ennemis pour maintenir la pression sur le joueur. Ils se déplacent en direction du joueur en cherchant constamment à le rapprocher et le forcer à esquiver ou à réagir rapidement.

Pour l'attaque, les ennemis lancent des projectiles dans la direction du joueur. Chaque ennemi dispose également d'un nombre de points de vie, et peut être détruit après avoir subi un certain nombre de dégâts de la part du joueur.

Le rythme des tirs ennemis peut varier, permettant de créer des défis en fonction de leur niveau de dangerosité et rendant les affrontements plus dynamiques.

2.1.3 Les déplacements

Le joueur peut se déplacer librement dans toutes les directions grâce aux touches de déplacement. Les touches W, A, S, D ou les flèches directionnelles permettent au joueur de contrôler ses mouvements horizontalement et verticalement.

2.1.4 Le tir

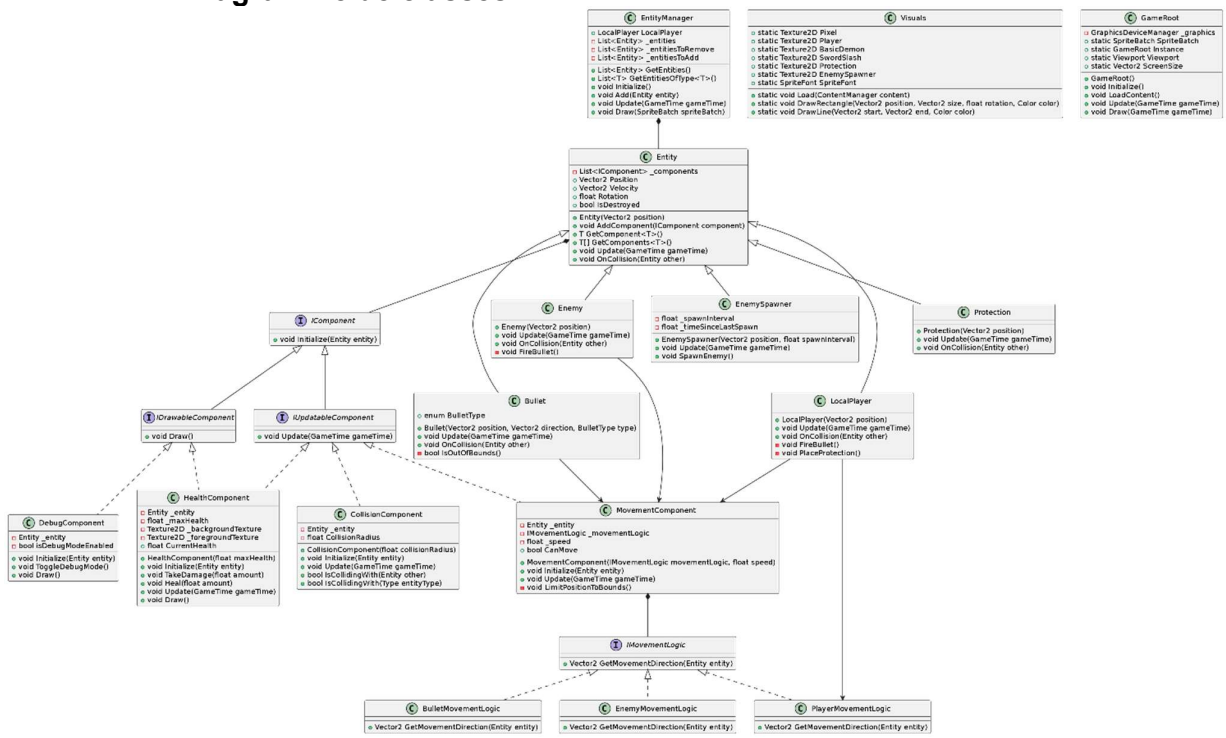
Le joueur peut tirer en appuyant sur le clic gauche de la souris. Lorsqu'il tire, un projectile est lancé en direction du curseur, suivant une trajectoire droite depuis sa position actuelle. Un délai de 200 millisecondes est appliqué entre chaque tir, ce qui impose un rythme aux attaques et demande au joueur de temporiser ses tirs pour optimiser ses chances contre les ennemis.

2.1.5 La gestion des vies

Le joueur dispose d'un certain nombre de points de vie qui diminuent chaque fois qu'il est touché par un projectile ennemi. Lorsque ses points de vie atteignent zéro, la partie se termine. Ces points de vie permettent au joueur de résister à plusieurs attaques, mais chaque impact réduit ses chances de survie.

2.2 Concept

2.2.1 Diagramme de classes



(Pour une meilleur visibilité,
consulter « POO/docs/D-P_ShootMeUp-ESR-DiagrammeUML »)

2.3 Analyse fonctionnelle

2.3.1 Déplacement du joueur

(Auteur: Eliott Scherrer)

En tant que joueur Je veux pouvoir me déplacer dans toutes les directions (X, Y) Afin de pouvoir esquiver les obstacles et les missiles ennemies.	
Tests d'acceptance:	
Touches de déplacement	Pendant une partie, si j'appuie sur un des boutons de déplacement (WASD par défaut), le personnage bouge en fonction.
Arrêt ralentis	Pendant une partie, quand je passe d'un état de déplacement à un état d'arrêt de mouvement, mon personnage s'arrête petit à petit et non pas d'un coup sec.
Out of bounds	Pendant la partie, quand j'essaie de sortir des limites du terrain de jeu, mon personnage s'arrête et reste dans les limites.

2.3.2 Mécaniques d'attaque du joueur

(Auteur: Eliott Scherrer)

En tant que joueur, je souhaite pouvoir attaquer avec une arme afin de pouvoir tuer les ennemies.	
	Tests d'acceptance:
Lancement d'attaque	Pendant une partie, quand je clique sur le bouton d'attaque, mon personnage lance une attaque dans la direction de mon curseur.
Cooldown	Pendant une partie, quand j'essaie de lancer une attaque mais que je suis dans une période de cooldown, l'attaque ne se lance pas.
Vitesse	Pendant une partie, quand une attaque sort du terrain de jeu, elle est détruite.

2.3.3 Mécaniques de défense du joueur

(Auteur: Eliott Scherrer)

En tant que joueur, je souhaite être capable de me protéger afin de pouvoir survivre plus longtemps	
	Tests d'acceptance:
Bouclier	Pendant une partie, quand j'appuie sur le bouton de bouclier en dehors d'une période de cooldown, un bouclier de taille variable qui pare les attaques ennemies apparait devant moi.
Bouclier cooldown	Pendant une partie, quand j'appuie sur le bouton de bouclier mais que je suis dans une période de cooldown, le bouclier ne s'ouvre pas.
Diminution de la surface du bouclier	Pendant une partie, quand je sors le bouclier, sa taille diminue au fil du temps jusqu'à être nulle puis active la période de cooldown.

2.3.4 Progression dans le jeu (niveaux)

(Auteur: Eliott Scherrer)

En tant que joueur, je veux pouvoir jouer à plusieurs niveaux, afin d'avoir une expérience qui n'est pas monotone et linéaire.	
	Tests d'acceptance:
Un seul niveau	Pendant une partie, quand je progresse dans le jeu, je remarque qu'il n'y a qu'un seul "niveau" global, qui se divise en différents paliers avec des ennemis et des environnements variés. Lorsque je meurs, je recommence toujours ce même "niveau" depuis le début, tout en traversant à nouveau les mêmes paliers.
Difficulté	Pendant une partie, à chaque changement de palier, l'environnement change et les ennemis également (stats et apparence).
Boutique entre pallier	Entre chaque pallier, j'apparais dans une boutique.
Achat dans la boutique	Entre chaque pallier, dans la boutique, je peux dépenser la monnaie collectée pendant le jeu pour acheter des améliorations (ex: plus de dégats, abilités spéciales, etc.).
Après achat dans la boutique	Dans la boutique, après avoir acheté quelque(s) chose(s), je continue vers le prochain palier avec l(es) amélioration(s) active(s).
Pas d'achat dans la boutique	Dans la boutique, si je ne fais pas d'achat, je peux quand même avancer au prochain palier.
Un seul niveau	Pendant une partie, quand je progresse dans le jeu, je remarque qu'il n'y a qu'un seul "niveau" global, qui se divise en différents paliers avec des ennemis et des environnements variés. Lorsque je meurs, je recommence

	toujours ce même "niveau" depuis le début, tout en traversant à nouveau les mêmes paliers.
Stockage des palliers	Dans la base de donnée, chaque niveau est stockés avec toutes ses informations (nombre d'ennemis, leur statistiques etc.) sous le format JSON.
Choix des améliorations	Dans la boutique, le choix des trois améliorations disponibles est aléatoire.

2.3.5 Système de santé

(Auteur: Eliott Scherrer)

En tant que joueur, je souhaite avoir un nombre de points de vie, afin de rajouter du challenge et de la rejouabilité	
Tests d'acceptance:	
Barre de PV	Pendant une partie, je vois que les entités ont une barre de points de vie.
Récupération de PV	Entre chaque pallier, dans la boutique, je regagne tous mes points de vie.
Amélioration de PV	Entre chaque pallier, dans la boutique, je peux acheter des améliorations afin d'avoir plus de points de vie.
Dégâts	Pendant une partie, quand une entité se prend des dégâts, sa vie diminue.

2.3.6 Déplacement des ennemis

(Auteur: Eliott Scherrer)

En tant que joueur, je veux que les ennemis puissent se déplacer afin de rendre les combats plus dynamiques et difficiles, et me forcer à ajuster ma stratégie et mes mouvements pour les éviter ou les attaquer efficacement.	
Tests d'acceptance:	
Direction	Pendant la partie, les ennemis se déplacent en direction du joueur.
Collisions	Pendant la partie, quand un ennemi entre en collision avec un mur, il le contourne afin de ne pas rester bloquer.

2.3.7 Mécaniques d'attaque des ennemis

(Auteur: Eliott Scherrer)

En tant que joueur je veux que les ennemis puissent attaquer, afin de créer des situations de danger où je dois esquiver ou me défendre, ce qui rend le jeu plus engageant et amusant.	
Tests d'acceptance:	
Cooldown	Pendant une partie, les ennemis tirent chaque N nombre de temps, avec N choisi aléatoirement avec +- 500ms.
Lancement d'attaque	Pendant la partie, quand l'ennemi tire, un projectile s'envoie en direction du joueur (la trajectoire ne change pas pendant le tir).
Out of bounds	Pendant la partie, quand un projectile quitte le terrain de jeu, il disparait.
Proche du joueur	Pendant la partie, quand l'ennemi entre dans la zone de collision du joueur, l'ennemi s'arrête et commence à mettre des dégâts de mêlée

2.3.8 Visuels

(Auteur: Eliott Scherrer)

En tant que joueur, Je veux que les objets/personnages du jeu aient une texture Afin de pouvoir jouer à un jeu plaisant pour les yeux.
Tests d'acceptance:

Joueur	Pendant la partie, je vois que le joueur est représenté par un sprite.
Ennemis	Pendant la partie, je vois que chaque ennemi est représenté par un sprite.
Attaques	Pendant la partie, je vois que chaque attaque est représentée par un sprite.
Protection	Pendant la partie, je vois que chaque objet de protection est représenté par un sprite.
Spawners	Pendant la partie, je vois que chaque point de spawn des ennemis est représenté par un sprite.

3 Analyse UX

3.1 Conception centrée utilisateur

Dans la création de jeux vidéo, l'approche de conception centrée utilisateur met l'accent sur les besoins et préférences des joueurs, ce qui favorise une expérience de jeu optimale et engageante.
Pour *Demon Butcher*, nous avons identifié deux profils d'utilisateurs types, ou « personas ». Ces personas représentent les motivations et attentes principales de groupes de joueurs distincts, ce qui permet alors d'orienter les choix de conception vers des expériences axés sur les utilisateurs.

1) Diego Garcia

Figure 1 Document de présentation du premier persona Diego Garcia

Ce persona représente un joueur jeune et passionné par les défis et la compétition. Il recherche une expérience intense où il peut repousser ses limites et comparer ses scores à ceux des autres joueurs. Il apprécie particulièrement les jeux de tir et d'action, où chaque partie est l'occasion de se surpasser. La diversité des défis, le rythme rapide, et une fluidité dans le gameplay sont essentiels pour lui.

2) Stella Yomi

Figure 2 Document de présentation du deuxième persona Stella Yomi

Ce persona incarne une joueuse plus détendue, qui se tourne vers les jeux pour se divertir sans pression. Cette joueuse apprécie les environnements immersifs, avec une préférence pour des jeux qui permettent d'explorer et de prendre le temps de découvrir des éléments cachés ou uniques. Les joueurs « casual » cherchent avant tout un jeu accessible, intuitif et plaisant, où le gameplay peut être aussi simple que relaxant.

3.2 Choix de la palette graphique

Pour Demon Butcher, le choix des couleurs est important pour créer l'ambiance du jeu. La palette de couleur est divisée en deux parties : des tons neutres pour les éléments de fond et des couleurs d'accent pour attirer l'attention sur certains détails.

Figure 3 Présentation de la palette de couleur

1) Tons Neutres :

Les gris clairs au noir profond servent de fond au jeu. Ces couleurs neutres posent les bases visuelles sans être trop présentes, ce qui laisse les éléments principaux ressortir.

2) Couleurs d'Accent

Les couleurs d'accent vont des tons rose clair jusqu'aux rouges très foncés. Ces couleurs sont parfaites pour les éléments clés, comme les ennemis, les effets d'attaque ou les éléments d'UI importants. Elles rappellent les thèmes « démon » et « sanguinaire » du jeu, ce qui aide à renforcer l'immersion dans l'univers du jeu.

3.3 Eco-conception

L'éco-conception dans Demon Butcher vise à réduire l'empreinte environnementale du jeu en optimisant les ressources utilisées et en minimisant l'impact des éléments graphiques et techniques. Pour accomplir cela, plusieurs principes ont été appliqués dans la conception de l'interface et des menus :

1) Optimisation des images

Les éléments visuels (les icônes, les arrière-plans etc.) sont conçus pour être légers. En limitant la taille et la complexité des images dans les menus, le jeu devient non seulement moins lourd à télécharger mais utilise aussi moins de mémoire et de puissance de traitement, ce qui vise à réduire la consommation d'énergie.

2) Palette de couleurs limitée

En restreignant la palette de couleurs aux tons neutres et à quelques accents, on réduit le nombre d'éléments de design uniques. Cela simplifie les fichiers de style et diminue l'utilisation des ressources matérielles, ce qui rend l'interface plus efficace. L'utilisation de nuances de gris et de rouge reste cohérente avec l'ambiance tout en gardant le design efficace et minimaliste.

3) Interface épurée et intuitive

Quand on adopte un design épuré, on réduit le nombre d'éléments à afficher, ce qui diminue la demande en ressources graphiques. Les menus sont clairs et bien organisés, avec des éléments essentiels bien mis en évidence, ce qui rend la navigation rapide et intuitive tout en optimisant les ressources.

3.4 Accessibilité

L'accessibilité vise à rendre le jeu et ses menus accessibles à un maximum de joueurs, quels que soient leurs besoins, limitations ou handicaps. Plusieurs éléments ont été mis en place pour faciliter la navigation et l'interaction avec le jeu afin d'avoir une expérience inclusive.

1) Contrastes de couleurs élevés

La palette de couleurs utilise des contrastes forts entre les éléments interactifs (rouge pour la plupart) et les arrière-plans neutres (généralement blanc). Ces contrastes permettent de rendre le texte et les boutons facilement visibles, même pour les joueurs ayant des problèmes de vue ou des difficultés à distinguer certaines couleurs comme le daltonisme.

2) Textes lisibles

Les menus et les informations de jeu utilisent une taille de police suffisamment grande pour être lisible sans effort. Des polices claires et sans empattement ont été choisies pour renforcer la lisibilité, en particulier pour les titres et les CTA.

3) Navigation simplifiée

L'interface est conçue de manière à éviter les surcharges d'information, avec une organisation claire et des options bien identifiées. Les écrans sont épurés et faciles à naviguer, minimisant les distractions pour les joueurs qui peuvent rencontrer des difficultés de concentration ou des troubles de l'attention.

4 Conception UX

4.1 Définition des wireframes

Les wireframes sont des maquettes simples qui structurent les principaux écrans et menus du jeu. Ils servent à planifier l'organisation de chaque élément de l'interface pour assurer une navigation intuitive et cohérente. Ces maquettes présentent la disposition des menus, des options de personnalisation et des écrans de gameplay, offrant une vue d'ensemble claire avant la mise en forme finale.

1) Menu principal

Figure 4 Wireframe du menu principal

2) Ecran de gameplay

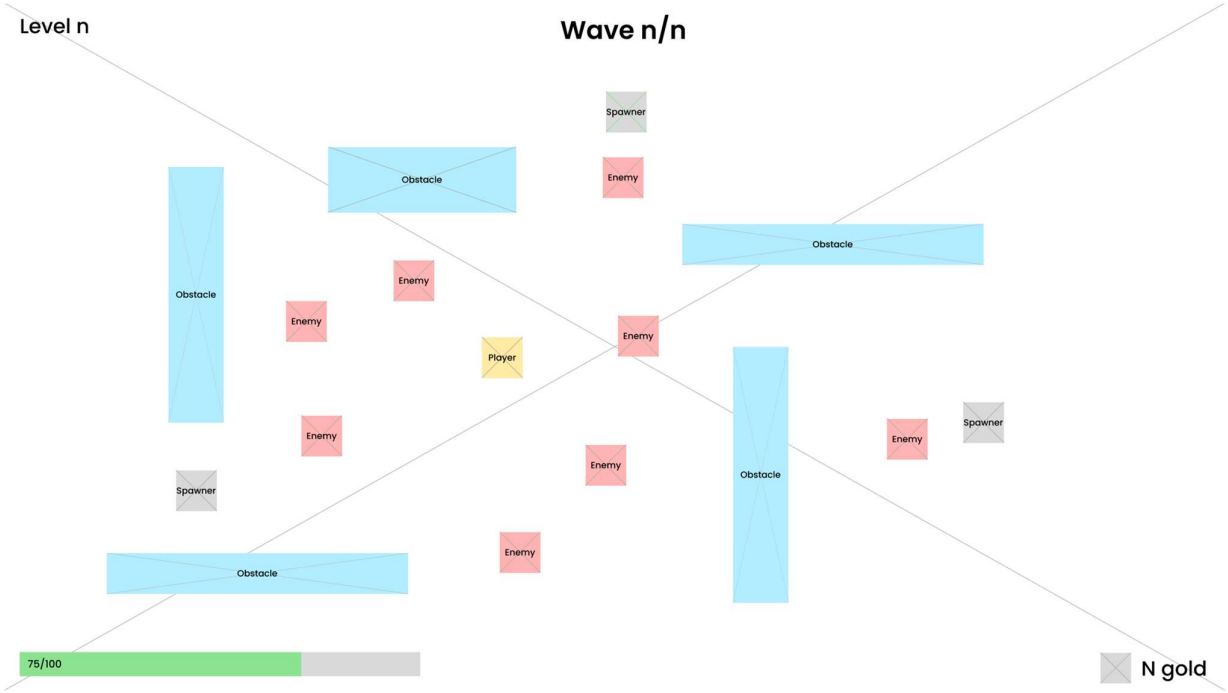


Figure 5 Wireframe de l'écran de jeu

3) Editeur de niveau

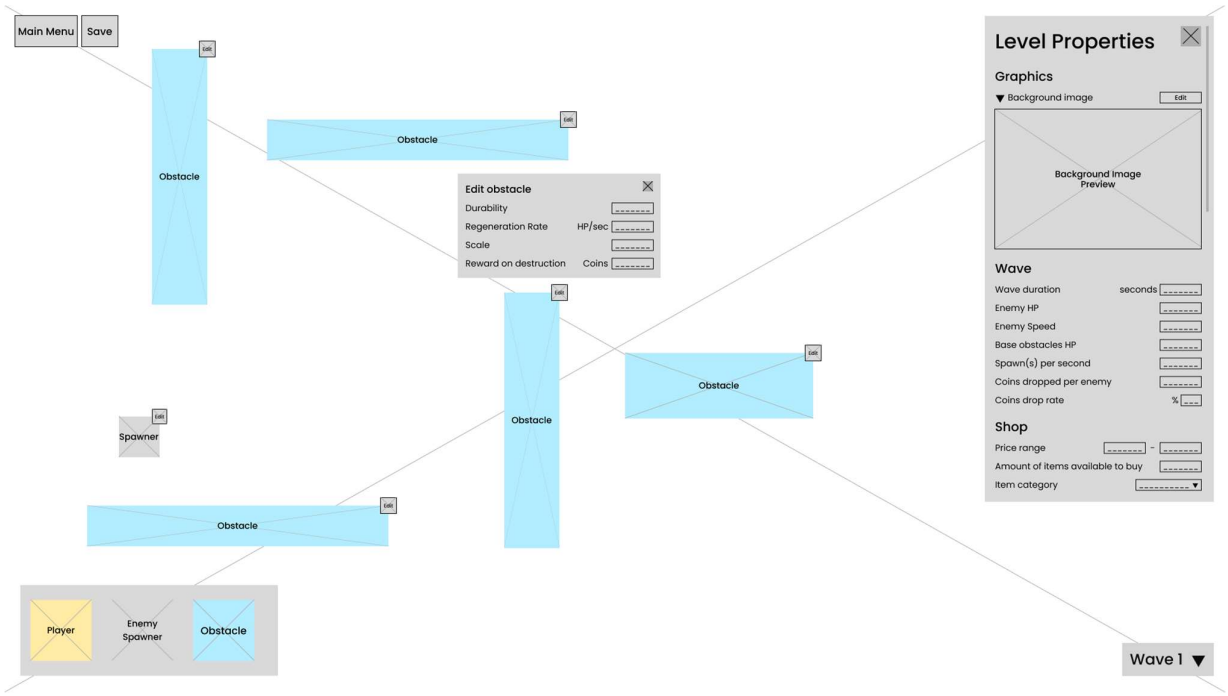


Figure 6 Wireframe de l'éditeur de niveau

4) Ecran des high scores

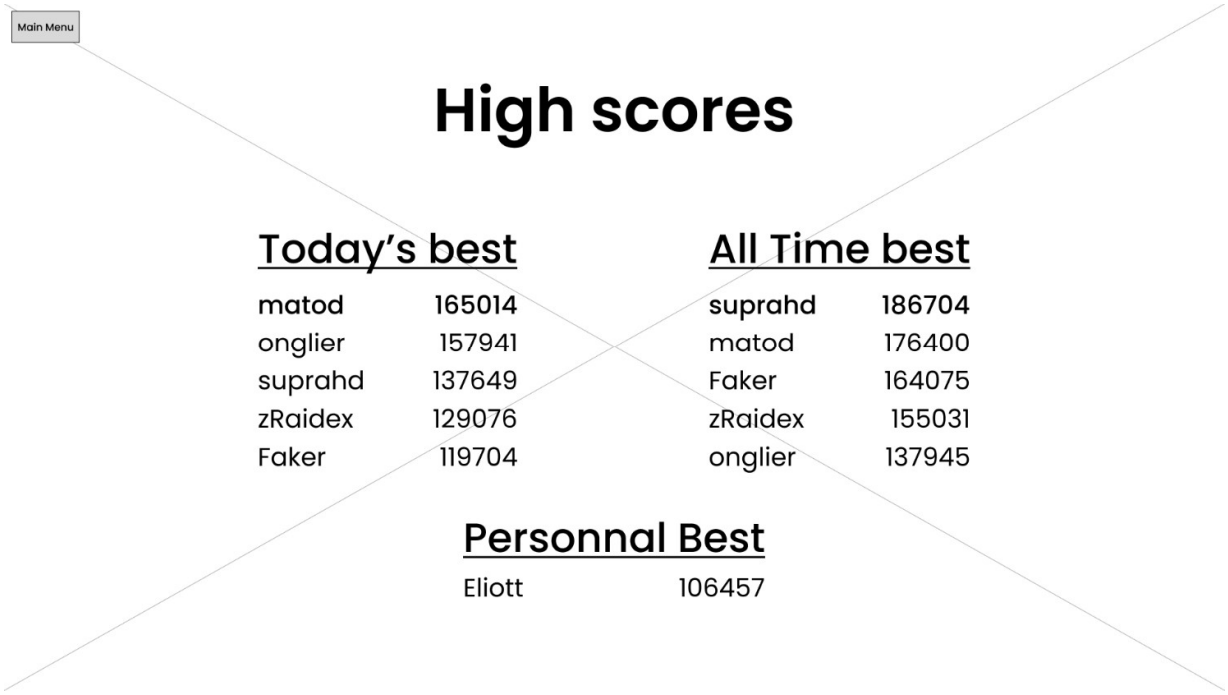


Figure 7 Wireframe de l'écran des high scores

5) Editeur de niveau (High fidelity Mockup)

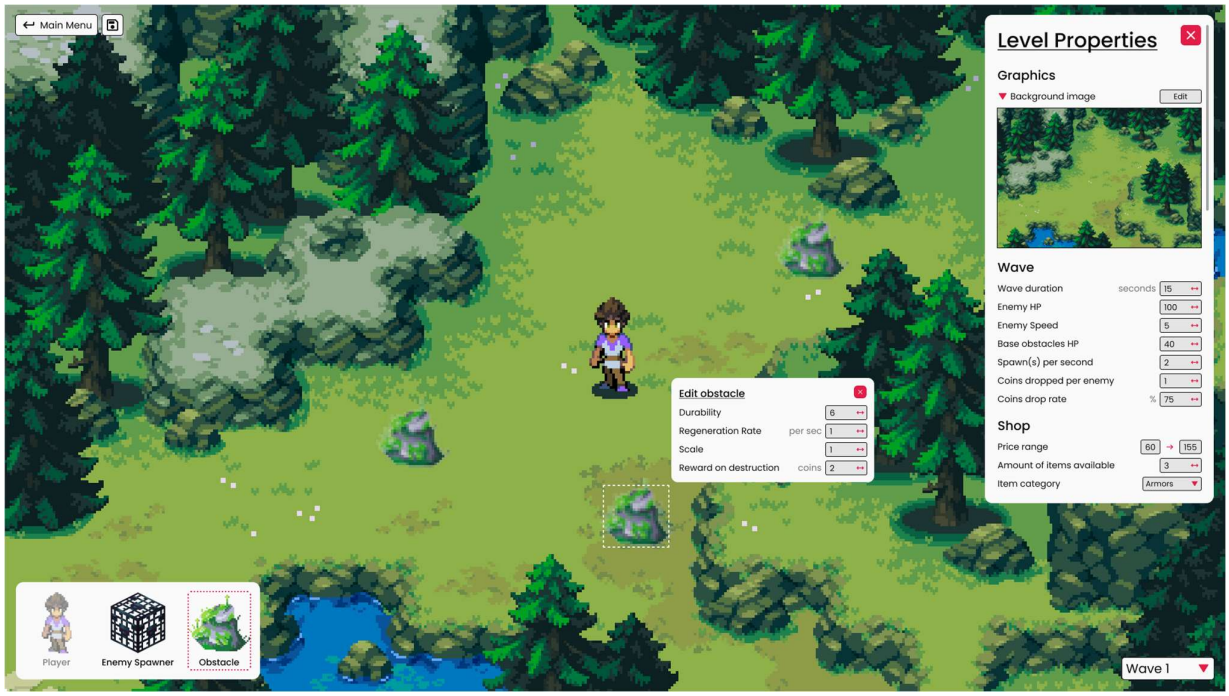


Figure 8 Mockup high fidelity de l'éditeur de niveau

4.2 Choix effectués

Dans la conception UX, plusieurs choix ont été faits pour améliorer l'expérience utilisateur et garder une interface fluide et intuitive.

1) Menus flottants pour économiser de l'espace

L'éditeur de niveau utilise des menus flottants pour minimiser l'encombrement de l'écran. En plaçant les options de création et de personnalisation dans des panneaux discrets ou en overlay sur les côtés, les joueurs ont d'une vue dégagée sur le niveau, ce qui facilite la création. Cette configuration aide à focaliser l'attention sur le contenu du niveau mais offre tout de même un accès rapide aux options essentiels.

2) Uniformité de la résolution entre l'éditeur et le jeu

L'éditeur conserve la même résolution que celle utilisée en jeu, ce qui permet aux joueurs de voir le niveau exactement tel qu'il apparaîtra lorsqu'ils joueront. Ce choix renforce la cohérence visuelle et réduit les ajustements nécessaires entre l'édition de niveau et le gameplay.

3) Pop-ups de paramétrage pour les objets et entités

Lorsqu'un joueur sélectionne un objet ou une entité dans l'éditeur, une pop-up s'ouvre pour permettre la modification de ses paramètres (comme la taille, la position, les caractéristiques, etc.). Cela améliore l'interactivité en donnant un accès rapide aux options de personnalisation sans naviguer dans des menus compliqués.

4) **Accessibilité rapide aux outils essentiels**

Le menu de l'éditeur va à l'essentiel, regroupant les outils fréquemment utilisés dans des barres de navigation bien placées. Cette organisation complète sans être complexe permet aux joueurs d'effectuer leurs actions rapidement et efficacement. Les options sont claires et facilement accessibles, facilitant les ajustements en temps réel et optimisant le flux de création.

5) Indicateur de placement unique pour le joueur

Une fois le joueur placé depuis la barre d'objets dans le niveau, il apparaît en gris dans la barre pour indiquer qu'il ne peut plus être ajouté à nouveau. Ce choix garantit qu'un seul joueur est présent sur le terrain de jeu avec un indicateur intuitif.

5 Réalisation

5.1 Points de design spécifiques

5.1.1 Système Basé sur les Composants (CBS)

Dans Demon Butcher, la création d'un système basé sur les composants (Component-Based System, CBS) a été une solution clé pour concevoir un jeu flexible et modulaire. Ce système permet de définir les entités du jeu (comme le joueur, les ennemis, et les projectiles) en leur associant différents composants, chacun responsable d'une fonctionnalité distincte. Voici une analyse de ce design technique et de son fonctionnement.

5.1.1.1 Principe du CBS

Le CBS repose sur le découpage des fonctionnalités des entités en composants individuels. Plutôt que d'attribuer des comportements directement aux entités elles-mêmes, chaque fonctionnalité, comme le mouvement, l'affichage, ou la gestion de la santé, est encapsulée dans un composant. Cela permet d'ajouter, de retirer, ou de modifier les capacités d'une entité sans toucher à sa structure de base.

5.1.1.2 Interfaces de Composants

Pour garantir que chaque composant peut être utilisé de manière cohérente, trois interfaces principales sont définies :

1) IComponent : Interface de base, que tous les composants implémentent, assurant une méthode d'initialisation Initialize(Entity entity).

2) IUpdatableComponent : Interface héritée pour les composants nécessitant des mises à jour régulières (Update).

3) IDrawableComponent : Interface héritée pour les composants nécessitant un rendu graphique (Draw).

Ce découpage permet de déterminer précisément les rôles de chaque composant tout en les rendant interchangeables et facilement extensibles.

5.1.1.3 Exemple de Composants : Mouvement, Affichage, et Santé

Mouvement (MovementComponent)

Le composant MovementComponent est responsable de gérer les déplacements d'une entité. Il utilise une logique de mouvement (IMovementLogic) et une vitesse définie pour calculer la direction et la vitesse des déplacements.

Affichage (RenderComponent)

Le composant RenderComponent gère l'affichage à l'écran d'une entité. En utilisant une texture, une échelle, une rotation, etc., il dessine l'entité à l'écran et ajuste automatiquement sa taille.

Santé (HealthComponent)

Le composant HealthComponent suit et gère les points de vie d'une entité. Il réduit les points de vie lorsqu'une entité subit des dégâts et peut déclencher une destruction automatique de l'entité lorsque la santé atteint zéro. Ce composant inclut des méthodes pour la guérison, et des ajouts futurs (comme la régénération ou l'invulnérabilité temporaire) peuvent y être facilement intégrés.

5.1.1.4 Avantages du CBS dans ce Projet

Le CBS rend le jeu plus extensible et modulaire. Par exemple, ajouter un nouveau type d'ennemi ou une capacité spécifique pour le joueur ne nécessite que l'ajout ou la modification de composants, sans toucher au cœur du code des entités. Ce modèle simplifie aussi la maintenance, car chaque composant est indépendant et peut être testé, mis à jour, ou remplacé sans perturber les autres fonctionnalités.

5.1.2 Gestion de la Destruction des Balles Hors de l'Écran

Pour optimiser les performances et maintenir un environnement de jeu propre, une gestion spécifique est mise en place pour détruire les projectiles quand ils quittent les limites de l'écran. Ce mécanisme empêche l'accumulation de balles inutiles en mémoire, ce qui rend le jeu plus fluide.

5.1.2.1 Principe de Destruction des Balles

Chaque balle suit une trajectoire définie après son lancement, alors quand elle sort de la zone visible de l'écran, elle devient inutile. Pour gérer cela, la méthode Update de la balle vérifie systématiquement si sa position est en dehors des limites de l'écran à chaque cycle de jeu. Lorsque la balle dépasse ces limites, elle est automatiquement marquée pour destruction.

5.1.2.2 Implémentation de la Détection Hors Limites

```
2 references
public override void Update(GameTime gameTime)
{
    // Destroy when out of bounds to free memory
    if (IsOutOfBounds())
        IsDestroyed = true;
}

1 reference
private bool IsOutOfBounds() => !Position.X.IsInRange(0, GameRoot.ScreenSize.X) ||
                                !Position.Y.IsInRange(0, GameRoot.ScreenSize.Y);
```

La vérification se fait grâce à la méthode IsOutOfBounds(), qui utilise la position actuelle de la balle pour déterminer si elle est dans les limites de l'écran :

- Les Position.X et Position.Y de l'entité sont comparés aux dimensions de l'écran (GameRoot.ScreenSize.X et GameRoot.ScreenSize.Y).

- Si la balle est en dehors de cet intervalle, la méthode retourne true, et l'attribut IsDestroyed de la balle est mis à true, déclenchant ainsi sa suppression lors de la mise à jour.

5.1.3 Logique de Mouvement des Ennemis

La classe EnemyMovementLogic est un composant qui définit la logique de déplacement des ennemis. Elle permet aux ennemis de se diriger vers le joueur tout en s'adaptant aux obstacles présents, comme les protections. Cette logique de mouvement rend les ennemis plus intelligents, introduisant plus de défis pour le joueur.

5.1.3.1 Principe de la EnemyMovementLogic

Le composant de logique de mouvement des ennemis suit une série de conditions pour adapter la trajectoire des ennemis en fonction de leur environnement :

- **Arrêt en cas de collision avec le joueur**

  Si un ennemi entre en collision avec le joueur, sa logique de mouvement l'amène à s'arrêter immédiatement. Cette condition permet d'éviter que les ennemis ne traversent le joueur et les permet de lancer des attaques au corps à corps.

- **Évitement des protections**

  Lorsqu'un ennemi rencontre un objet de protection placé par le joueur, il ajuste sa direction pour éviter cet obstacle. L'ennemi calcule un vecteur de mouvement perpendiculaire aux protections proches, afin de s'éloigner des positions de protection.

  - La méthode identifie les objets de protection à proximité et cumule des directions opposées à chacun de ces obstacles.

  - Le vecteur final, normalisé, représente la direction d'évitement qui éloigne l'ennemi des protections environnantes, rendant les déplacements plus intelligents et réalistes.

- **Déplacement vers le joueur en absence d'obstacles**

  En l'absence de protection ou de collision, l'ennemi se dirige directement vers le joueur. Ce mouvement est calculé en prenant la direction du joueur par rapport à la position de l'ennemi.

5.2 Déroulement

Ce chapitre présente un résumé de la réalisation de chaque user story du projet **Demon Butcher**, incluant les difficultés rencontrées, les alternatives envisagées, et les surprises durant le développement.

5.2.1 Déplacement du joueur

Objectif : Permettre au joueur de se déplacer dans toutes les directions pour éviter les obstacles et missiles ennemis.

- **Déroulement** : L'implémentation initiale du déplacement a bien respecté la story, avec des contrôles fluides via les touches WASD et les flèches directionnelles.
- **Difficultés** : La gestion des collisions avec les limites de l'écran a été un défi pour faire en sorte que le joueur ne sorte pas du terrain de jeu.

5.2.2 Mécaniques d'attaque du joueur

Objectif : Donner au joueur la capacité d'attaquer en tirant dans la direction du curseur.

- **Déroulement** : Le tir s'est facilement intégré avec une trajectoire linéaire basée sur la position du curseur. Un cooldown de 200 ms entre chaque tir a été ajouté pour éviter le spam.
- **Difficultés** : La gestion précise du cooldown afin d'assurer une réactivité sans autoriser des tirs trop fréquents.
- **Alternative** : Un cooldown plus long a été envisagé pour augmenter la difficulté, mais finalement, la durée actuelle s'est avérée satisfaisante pour le gameplay.

5.2.3 Mécaniques de défense du joueur

Objectif : Permettre au joueur d'utiliser un bouclier pour se protéger des attaques ennemies.

- **Déroulement** : L'implémentation de la défense a consisté en un bouclier activable en appuyant sur un bouton avec un cooldown.
- **Difficultés** : Ajuster le nombre de points de vie et la taille du bouclier pour trouver un bon équilibre entre défense et vulnérabilité.

5.2.4 Progression dans le jeu (niveaux)

Objectif : Créer un unique niveau global, divisé en plusieurs paliers avec une boutique entre chaque étape.

- **Déroulement** : Cette partie n'a pas été effectuée par manque de temps.

5.2.5 Système de santé

Objectif : Intégrer un système de points de vie visible avec une récupération et des améliorations disponibles dans la boutique.

- **Déroulement** : La barre de vie dynamique est bien visible en jeu et se met à jour lors des dégâts reçus.
- **Difficultés** : Trouver un emplacement optimal pour la barre de vie sans distraire le joueur.
- **Alternative** : Afficher les points de vie en nombre uniquement, mais cela aurait réduit la lisibilité en temps réel.

5.2.6 Déplacement des ennemis

Objectif : Donner aux ennemis la capacité de se déplacer vers le joueur et de contourner les obstacles.

- **Déroulement** : Les ennemis se dirigent automatiquement vers le joueur et contournent les protections.
- **Difficultés** : Gérer les collisions avec les protections et murs tout en maintenant un déplacement naturel.

5.2.7 Mécaniques d'attaque des ennemis

Objectif : Permettre aux ennemis de lancer des projectiles à intervalles réguliers, dirigés vers le joueur.

- **Déroulement** : Les ennemis tirent en direction du joueur avec un intervalle de cooldown aléatoire pour varier le rythme.
- **Difficultés** : Calculer les intervalles de tir pour éviter une surcharge de projectiles à l'écran.
- **Alternative** : Fixer un intervalle unique pour tous les ennemis, mais le modèle aléatoire rend le jeu plus imprévisible.

5.3 Mise en place de l'environnement de travail

Pour développer et tester Demon Butcher, il est nécessaire de mettre en place un environnement de travail compatible avec le framework Monogame utilisé dans ce projet.

5.3.1 Accéder au code source

Le code source du projet est disponible sur GitHub et peut être consulté en suivant ce lien : P_ShootMeUp-Eliott_GitHub_Repository. Les développeurs peuvent cloner le projet en utilisant Git ou le télécharger directement en tant que fichier ZIP. Voici les étapes pour configurer l'environnement :

1) Cloner le dépôt GitHub

Ouvrez un terminal ou utilisez un client Git pour cloner le dépôt :

```
git clone https://github.com/eliottscherrer/P_ShootMeUp-Eliott.git
```

2) Installation de l'extension Monogame

Demon Butcher est développé avec le framework Monogame. Pour l'exécuter et le modifier dans Visual Studio, il est indispensable d'installer l'extension Monogame :

1. Ouvrez Visual Studio.
2. Accédez à l'onglet Extensions > Manage Extensions.
3. Recherchez Monogame et installez l'extension Monogame pour Visual Studio.
4. Redémarrez Visual Studio pour activer l'extension.

3) Ouverture du projet dans Visual Studio

1. Une fois le dépôt cloné, ouvrez Visual Studio.
2. Allez dans File > Open > Project/Solution et sélectionnez le fichier .sln du projet P_ShootMeUp-Eliott.
3. Assurez-vous que le framework Monogame est bien installé pour éviter les erreurs de compatibilité lors de la compilation.

5.3.2 Versions des systèmes d'exploitation et des outils logiciels

Pour le développement de Demon Butcher, les versions suivantes des systèmes et des logiciels ont été utilisées :

- Système d'exploitation : Windows 10, version 22H2 (Build 19045.4894)
- Environnement de développement : Visual Studio, version 17.9.4
- Framework graphique : Monogame, extension pour Visual Studio

Le projet a été développé sur un PC de bureautique disposant du matériel suivant :

- Processeur : Intel® Core™ i7-11700 2.50 GHz
- Mémoire (RAM): 32 Go

- Stockage: 512 Go SSD
- Carte graphique intégrée : Intel UHD Graphics 750

5.3.3 Liste de tous les fichiers et une rapide description de leur contenu

- GameRoot.cs : Fichier de démarrage principal, initialisant le jeu et configurant la boucle de jeu.

- Program.cs : Point d'entrée de l'application, lançant le jeu via GameRoot.

- Configs.cs : Fichier de configuration centralisant les constantes et paramètres de jeu (vitesse, taille, textures, etc.) pour un ajustement global.

- Entity.cs : Classe de base pour toutes les entités du jeu (joueur, ennemis, projectiles), gérant les composants et leurs interactions.

- EntityManager.cs : Responsable de la gestion de toutes les entités dans le jeu, ajoutant, supprimant et mettant à jour chaque entité.

- BulletMovementLogic.cs : Implémente la logique de déplacement des projectiles (bullets), définissant leur trajectoire.

- CollisionComponent.cs : Composant gérant les collisions des entités entre elles et avec l'environnement.

- DebugComponent.cs : Composant ajoutant des fonctionnalités de debug pour visualiser et tester les entités.

- EnemyMovementLogic.cs : Définit la logique de déplacement des ennemis, y compris leur direction vers le joueur et l'évitement des obstacles.

- HealthComponent.cs : Composant gérant la santé des entités, avec des méthodes pour infliger et récupérer des points de vie.

- IComponent.cs : Interface de base pour tous les composants, assurant une structure commune.

- IDrawableComponent.cs : Interface pour les composants nécessitant un rendu graphique, avec la méthode Draw.

- IMovementLogic.cs : Interface pour les composants de logique de mouvement, implémentée par différents systèmes de mouvement.

- IUpdatableComponent.cs : Interface pour les composants nécessitant des mises à jour à chaque frame.

- MovementComponent.cs : Composant gérant le déplacement des entités, utilisant la logique de mouvement associée.
- PlayerMovementLogic.cs : Logique de déplacement du joueur, définissant comment le joueur se déplace en réponse aux entrées.
- RenderComponent.cs : Composant responsable de l'affichage des entités sur l'écran, avec gestion des textures et échelle.
- Bullet.cs : Classe représentant les projectiles du joueur ou des ennemis, avec leur comportement spécifique.
- Enemy.cs : Classe de l'ennemi, définissant son comportement et ses caractéristiques de base.
- EnemySpawner.cs : Classe de gestion des points de spawn, permettant de faire apparaître les ennemis à intervalles réguliers.
- LocalPlayer.cs : Classe représentant le joueur, intégrant ses actions comme le tir et l'activation de protections.
- Protection.cs : Classe des objets de protection que le joueur peut placer, servant de boucliers temporaires.
- GlobalHelpers.cs : Classe utilitaire avec des méthodes globales utilisées dans différentes parties du code.
- InputManager.cs : Gère les entrées clavier et souris du joueur, servant d'interface pour les contrôles.
- MathHelpers.cs : Classe utilitaire pour les calculs mathématiques et géométriques, utilisée notamment pour le mouvement et les collisions.
- Visuals.cs : Fournit des méthodes utilitaires pour le rendu visuel et les effets graphiques.

5.4 Description des tests effectués

5.4.1 Sprint 1

5.4.1.1 Mécaniques de défense du joueur

Bouclier	Pendant une partie, quand j'appuie sur le bouton de bouclier en dehors d'une période de cooldown, un bouclier de taille variable qui pare les attaques ennemies apparait devant moi.	OK 2 Nov
Bouclier cooldown	Pendant une partie, quand j'appuie sur le bouton de bouclier mais que je suis dans une période de cooldown, le bouclier ne s'ouvre pas.	OK 2 Nov
Diminution de la surface du bouclier	Pendant une partie, quand je sors le bouclier, sa taille diminue au fil du temps jusqu'à être nulle puis active la période de cooldown.	ko 2 Nov

5.4.1.2 Système de santé

Barre de PV	Pendant une partie, je vois que les entités ont une barre de points de vie.	OK 2 Nov
Récupération de PV	Entre chaque pallier, dans la boutique, je regagne tous mes points de vie.	ko 2 Nov
Amélioration de PV	Entre chaque pallier, dans la boutique, je peux acheter des améliorations afin d'avoir plus de points de vie.	ko 2 Nov
Dégâts	Pendant une partie, quand une entité se prend des dégâts, sa vie diminue.	OK 2 Nov

5.4.1.3 Mécaniques d'attaque des ennemis

Cooldown	Pendant une partie, les ennemis tirent chaque N nombre de temps, avec N choisi aléatoirement avec +- 500ms.	OK 2 Nov
Lancement d'attaque	Pendant la partie, quand l'ennemi tire, un projectile s'envoie en direction du joueur (la trajectoire ne change pas pendant le tir).	OK 2 Nov
Out of bounds	Pendant la partie, quand un projectile quitte le terrain de jeu, il disparait.	OK 2 Nov
Proche du joueur	Pendant la partie, quand l'ennemi entre dans la zone de collision du joueur, l'ennemi s'arrête et commence à mettre des dégâts de mêlée	OK 2 Nov

5.4.1.4 Mécaniques d'attaque du joueur

Lancement d'attaque	Pendant une partie, quand je clique sur le bouton d'attaque, mon personnage lance une attaque dans la direction de mon curseur.	OK 2 Nov
Cooldown	Pendant une partie, quand j'essaie de lancer une attaque mais que je suis dans une période de cooldown, l'attaque ne se lance pas.	OK 2 Nov
Vitesse	Pendant une partie, quand une attaque sort du terrain de jeu, elle est détruite.	OK 2 Nov

5.4.1.5 Déplacement des ennemis

Direction	Pendant la partie, les ennemis se déplacent en direction du joueur.	OK 2 Nov
Collisions	Pendant la partie, quand un ennemi entre en collision avec un mur, il le contourne afin de ne pas rester bloquer.	OK 2 Nov

5.4.1.6 Visuels

Joueur	Pendant la partie, je vois que le joueur est représenté par un sprite.	OK 2 Nov
Ennemis	Pendant la partie, je vois que chaque ennemi est représenté par un sprite.	OK

		2 Nov
Attaques	Pendant la partie, je vois que chaque attaque est représentée par un sprite.	OK 2 Nov
Protection	Pendant la partie, je vois que chaque objet de protection est représenté par un sprite.	OK 2 Nov
Spawners	Pendant la partie, je vois que chaque point de spawn des ennemis est représenté par un sprite.	OK 2 Nov

5.4.1.7 Déplacement du joueur

Touches de déplacement	Pendant une partie, si j'appuie sur un des boutons de déplacement (WASD par défaut), le personnage bouge en fonction.	OK 2 Nov
Arrêt ralentis	Pendant une partie, quand je passe d'un état de déplacement à un état d'arrêt de mouvement, mon personnage s'arrête petit à petit et non pas d'un coup sec.	ko 2 Nov
Out of bounds	Pendant la partie, quand j'essaie de sortir des limites du terrain de jeu, mon personnage s'arrête et reste dans les limites.	OK 2 Nov

5.5 Erreurs restantes

Plusieurs tests d'acceptance sont marqués comme non réussis (KO), indiquant des fonctionnalités incomplètes ou des comportements inattendus dans certaines parties du jeu. Voici un résumé des erreurs restantes et des fonctionnalités manquantes :

1. Diminution de la surface du bouclier
   - Problème : Le bouclier activé par le joueur ne diminue pas progressivement jusqu'à être nul avant de déclencher le cooldown.
   - Impact : Ce problème limite l'aspect stratégique de la défense du joueur, car la gestion de la durée du bouclier n'est pas entièrement fonctionnelle.
2. Récupération de PV entre les paliers
   - Problème : Dans la boutique entre les paliers, la fonctionnalité de régénération des points de vie n'est pas active, empêchant le joueur de récupérer sa santé entre les niveaux.
   - Impact : Cela rend la progression plus difficile, car le joueur ne peut pas regagner de vie et commence chaque palier avec la même quantité de PV qu'au palier précédent.

3. Amélioration de PV dans la boutique
    - Problème : L'option d'acheter des améliorations de points de vie dans la boutique n'est pas fonctionnelle, privant le joueur d'une possibilité d'augmentation de santé maximale.

    - Impact : La rejouabilité est affectée, car le joueur ne peut pas développer son personnage en augmentant sa capacité de vie au fil de la progression.

4. Arrêt progressif du joueur
    - Problème : Lorsque le joueur passe de l'état de déplacement à un état d'arrêt, il devrait ralentir progressivement, mais ce comportement n'est pas implémenté ; le joueur s'arrête immédiatement.

    - Impact : Ce manque de fluidité dans le mouvement réduit le réalisme du jeu et peut rendre les déplacements moins naturels.

6 L'utilisation de l'IA dans ce projet

Dans le cadre du développement du jeu, l'IA a été utilisée pour assister dans plusieurs aspects du projet, afin d'optimiser la productivité et d'améliorer la qualité de la documentation et du code.

1. **Optimisation du code**
   L'IA a été très utile pour identifier des moyens d'optimiser certaines parties du code. Elle a suggéré des améliorations sur la lisibilité, pour réduire la complexité et augmenter l'efficacité de certaines fonctions, en particulier dans les systèmes de gestion des entités et les composants de mouvement. Ces suggestions ont permis de rendre le code plus fluide et maintenable.

2. **Aide à la documentation**
   L'IA a également été utilisée pour m'aider à rédiger la documentation du projet, y compris la description des fonctionnalités, la formulation de certains tests d'acceptance, etc. En fournissant des formulations claires et organisées, l'IA a contribué à créer une documentation plus compréhensible afin de faciliter la compréhension du projet pour les lecteurs de la documentation.

3. **Création du diagramme de classe avec PlantUML**
   Pour la modélisation de l'architecture logicielle, l'IA a aidé à convertir le code en un diagramme de classe en utilisant PlantUML. En fournissant des indications sur la structure des relations entre les classes et les composants, l'IA a permis de générer un diagramme détaillé qui reflète la conception orientée objet du projet. Cela a non seulement simplifié la visualisation de l'architecture mais a aussi permis d'identifier et de corriger certaines incohérences potentielles dans la hiérarchie des classes.

7 Conclusions

Ce chapitre présente un bilan du projet *Demon Butcher*, en examinant les objectifs atteints, les aspects positifs et négatifs, les difficultés rencontrées, et les éventuelles pistes d'évolution pour le futur.

7.1 Objectifs atteints / non-atteints

- **Objectifs atteints** : La majorité des objectifs fixés au départ ont été atteints. Le jeu est opérationnel avec des fonctionnalités de base comme les déplacements, les attaques et les défenses. Les ennemis possèdent des comportements distincts, et toutes les entités essentielles du jeu sont représentées par des sprites.

- **Objectifs non-atteints** : Quelques fonctionnalités secondaires, comme des améliorations avancées en magasin ou des événements spécifiques déclenchés par la santé, n'ont pas été totalement intégrées en raison de contraintes de temps.

7.2 Points positifs / négatifs

- **Points positifs** : Le projet a permis de mettre en place une architecture modulaire grâce au système basé sur les composants (CBS), rendant le code plus flexible et maintenable. L'utilisation de Monogame pour le rendu graphique a permis d'obtenir des visuels fluides, et les tests d'acceptance ont confirmé que chaque entité était bien représentée et fonctionnelle.

- **Points négatifs** : Quelques limitations graphiques liées à l'utilisation de Monogame ont restreint la qualité visuelle et les effets avancés du jeu. De plus, le temps consacré à certains aspects techniques a réduit la possibilité d'ajouter des fonctionnalités supplémentaires.

7.3 Difficultés particulières

- **Gestion des collisions et des déplacements** : La mise en place d'un système de gestion de collisions efficace a nécessité des ajustements pour s'assurer que les entités ne traversent pas les limites du terrain de jeu, et les déplacements adaptatifs des ennemis ont présenté des défis techniques pour éviter les obstacles sans trop ralentir les performances.

7.4 Suites possibles pour le projet (évolutions & améliorations)

- **Ajout de nouveaux types d'ennemis** : Des ennemis avec des comportements spécifiques ou des capacités uniques (par exemple, des tirs de zone ou des attaques de mêlée renforcées) pourraient être ajoutés pour diversifier les défis du jeu.

- **Améliorations du système de santé** : Introduire des capacités de régénération, des objets de soin temporaires, et des boosts de santé dans la boutique entre les niveaux pourrait enrichir les options de survie pour le joueur.

- **Optimisation graphique et visuelle** : Intégrer des effets graphiques plus avancés, comme des explosions ou des animations de destruction, rendrait le jeu plus immersif et visuellement attractif.

- **Évolution vers un mode multijoueur** : Un mode coopératif où deux joueurs peuvent combattre ensemble pourrait être envisagé.

En résumé, le projet a atteint la plupart de ses objectifs principaux et offre une base solide pour de futures améliorations. Les aspects techniques et les fonctionnalités de gameplay de base sont bien en place, et les évolutions envisagées pourraient apporter une nouvelle profondeur à l'expérience de jeu.

8 Annexes

8.1 Journal de travail

Le journal de travail se trouve dans « POO\docs\T-P_ShootMeUp-ESR-FinalJdT.pdf ».