|  |
| --- |
| TankBattle |

Pour projet Shoot Me Up avec XCL

Table des matières

1 Analyse préliminaire 3

1.1 Introduction 3

1.2 Objectifs 3

1.3 Gestion de projet 4

2 Analyse / Conception 4

2.1 Gameplay 4

2.2 Analyse de l’UX 5

2.3 Conception de l’UX 6

2.4 Concept 6

2.5 Analyse fonctionnelle 6

2.6 Stratégie de test 6

3 Réalisation 7

3.1 Points de design spécifiques 7

*3.1.1* *…* 7

*3.1.2* *…* 8

*3.1.3* *…* 8

3.2 Déroulement 8

3.3 Mise en place de l’environnement de travail 8

3.4 Description des tests effectués 8

3.5 Erreurs restantes 8

4 Conclusions 8

5 Annexes 9

5.1 Manuel de référence 9

5.2 Journal de travail 9

*NOTE L’INTENTION DES UTILISATEURS DE CE CANEVAS :*  
*Toutes les parties en italiques sont là pour aider à comprendre ce qu’il faut mettre dans cette partie du document. Elles n’ont donc aucune raison d’être dans le document final.*

*De plus, en fonction du type de projet, il est tout à fait possible que certains chapitres ou paragraphes n’aient aucun sens. Dans ce cas il est recommandé de les retirer du document pour éviter de l’alourdir inutilement.*

**Analyse préliminaire**

**Introduction**

TankBattle est un jeu de combat type Shoot em up. Celui-ci est réalisé dans le cadre de plusieurs projets à l’ETML sur les modules suivants : C106 (base de données), 320 (Programmation orientées objets), 322 (Expérience utilisateur). Ce projet rallie différents modules et permet de réunir ces différentes compétences en un seul projet et permet de m’apporter une meilleure expérience technique sur ces modules.

*Ce chapitre décrit brièvement le projet, le cadre dans lequel il est réalisé, les raisons de ce choix et ce qu'il peut apporter à l'élève ou à l'école. Il n'est pas nécessaire de rentrer dans les détails (ceux-ci seront abordés plus loin) mais cela doit être aussi clair et complet que possible (idées de solutions). Ce chapitre contient également l'inventaire et la description des travaux qui auraient déjà été effectués pour ce projet.*

*Ces éléments peuvent être repris des spécifications de départ.*

**Objectifs**

L’objectif de ce projet est de rallier mes différentes connaissances en un seul jeu. Pour cela, le jeu devra contenir différents critères selon le module, les voici :

UX : Maquettes du menu principal, de l’écran de jeu, et du meilleur score.

Programmation : Différents niveaux contenant un joueur et des ennemis avec un certain nombre de points de vie qui peuvent se déplacer et tirer. Il devra également y avoir des obstacles permettant au joueur d’esquiver les balles ennemies.

Base de données : Gestion des scores.

Ce projet sera réalisé selon un thème orienté militaire choisi au préalable.

*Ce chapitre énumère les objectifs du projet. L'atteinte ou non de ceux-ci devra pouvoir être contrôlée à la fin du projet. Les objectifs pourront éventuellement être revus après l'analyse.*

*Ces éléments peuvent être repris des spécifications de départ.*

**Gestion de projet**

Pour la gestion de ce projet, nous avons recours à différents outils afin de pouvoir s’organiser convenablement. En premier temps nous utilisons GitHub qui nous permet de sauvegarder notre travail au fur et à mesure et d’avoir un suivi de tout le projet. Nous utilisons également IceScrum qui nous permet de planifier des tâches et qui sert également à avoir un journal de travail. Concernant la documentation, tout se trouvera dans ce rapport.

***Ce chapitre décrit la méthode de gestion de projet utilisée, ainsi que les éventuelles particularités requises par le contexte et/ou le chef de projet***

**Analyse / Conception**

**Gameplay**

Le joueur :

Le joueur aura la capacité de se déplacer sur l’axe horizontal en utilisant les touches ‘A’ et ‘D’. Ceci lui permettra d’esquiver les différentes balles ennemies. De plus le joueur pourra tirer en appuyant sur le clic gauche de la souris. Une munition partira de son arme sur l’axe vertical avec une trajectoire linéaire. Cependant, le joueur devra attendre un délai de 500 millisecondes avant de pouvoir tirer à nouveau.

Les ennemis :

Les ennemis apparaîtront automatiquement au début de la partie, ils se situeront dans la partie supérieure de l’écran. Les ennemis seront sous forme de tank et leur objectif sera d’éliminer le joueur. Pour se faire, le tank pourra envoyer des missiles. A la différence du joueur, le tank ne pourra pas se déplacer, cependant il aura la capacité de nous viser et donc de tirer en diagonal.

Les protections

Sur la carte se trouveront des protections permettant au joueur de se cacher des missiles ennemis. Ceux-ci seront automatiquement placés sur la carte au lancement de la partie. Ils auront une taille fixe, ainsi que des points de vie. Ils pourront donc être détruits par le joueur ou un ennemi.

Les points de vie

Chaque entité aura des points de vie, le joueur ainsi que les ennemis auront 3 points de vie, à chaque fois qu’ils recevront un missile, ils en perdront un. Les protections auront elles 5 points de vies, rendant donc les structures plus solides. Lorsqu’un ennemi atteint 0 point de vie, il meurt ou disparaît si c’est une protection. Si c’est le joueur, la partie se termine.

Les niveaux

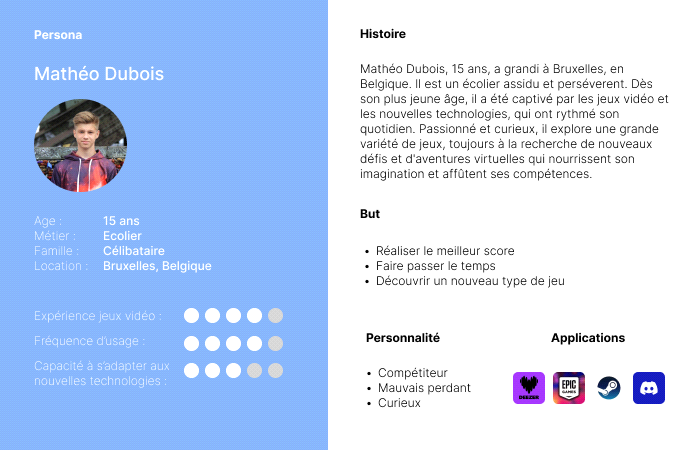
Le jeu comportera différents niveaux, pour que le joueur atteigne le second niveau, il devra réussir le premier, et ainsi de suite.

* *Le joueur*
* *Les ennemis*
* *Les déplacements*
* *Les niveaux*
* *Le tir*

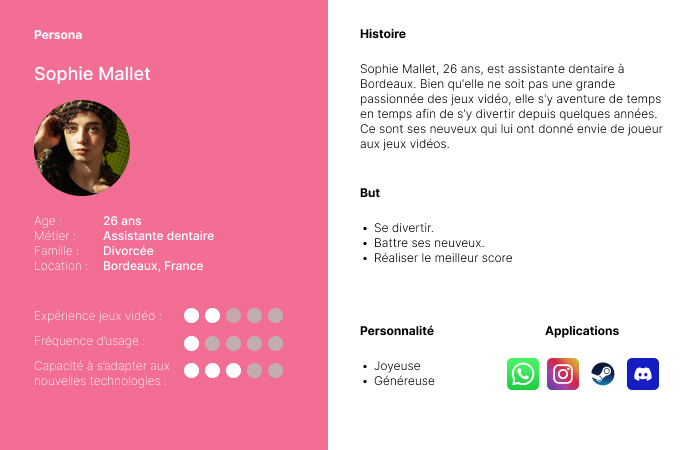
***Analyse de l’UX***

**Personas**

Ce chapitre concerne l’analyse de l’expérience utilisateur. En premier temps, il a fallu concevoir deux Personas. Ceux-ci sont des personnages fictifs créés afin de représenter différents types d'utilisateurs qui pourraient utiliser notre produit. Ils aident à mieux comprendre les besoins, les comportements et les frustrations des utilisateurs cibles. Les voici :



Ce premier persona est orienté en tant qu’un joueur aguerri, il a de l’expérience dans le domaine des jeux-vidéos et y joue régulièrement. Ses objectifs sont précis, devenir le meilleur.

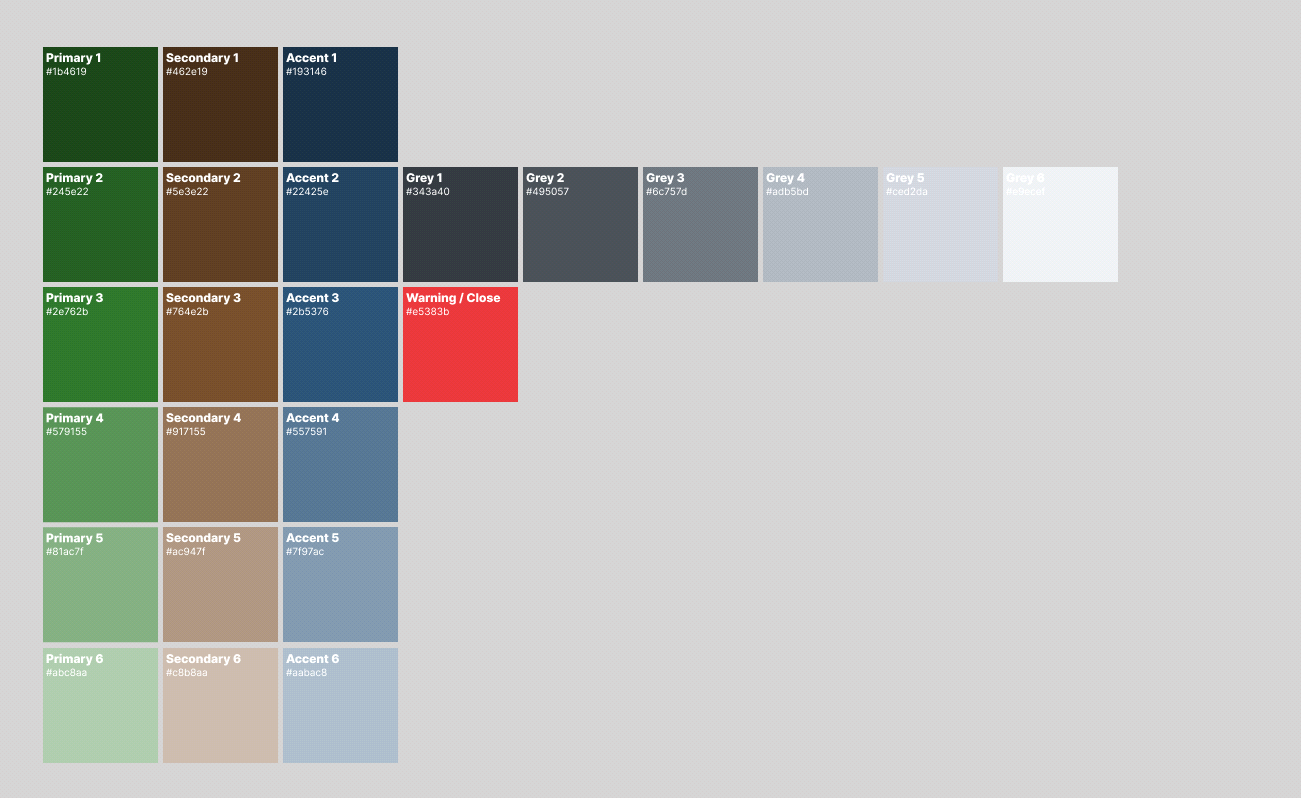


Ce second persona est bien différent du premier, Sophie est une amatrice des jeux vidéo et n’y joue que de temps à autres. Ses objectifs sont tout à fait différents car elle souhaite uniquement se divertir, et non pas de jouer en compétition.

Ces différents critères permettent donc d’adapter le jeu en fonction de leurs objectifs et personnalités.

**Palette graphique**

Pour la conception graphique de ce jeu, j’ai dû choisir une palette de couleur qui représentait correctement le thème de TankBattle. Il est également important de choisir des couleurs ayant bien ensemble. Voici la palette de couleur que j’aie utilisée pour la création des diverses maquettes et du logiciel.



On y retrouve différentes variétés de couleurs. La principale est la couleur verte, elle est la couleur qui sera le plus souvent utilisée. Elle représente totalement le thème militaire du jeu. La couleur secondaire est utilisée moins fréquemment, principalement pour les couleurs de fonds. Elle se marie bien avec le vert. Et finalement la couleur d’accent est bleue. Elle permet donc d’accentuer divers éléments sur l’interface et est utilisée avec parcimonie.

On y retrouve ensuite une variété de gris, celle-ci se trouve par défaut sur les palettes et est, pour mon cas, principalement utilisées pour la couleur des textes. Et en dernier, on trouve la couleur rouge qui est unique, elle est simplement utilisée pour les boutons « Fermer » ou les messages d’alertes.

**Eco-conception et accessibilité**

L’éco-conception est le fait de rendre son interface la plus économe possible, pour ce faire, une des pratiques courantes est d’utiliser des couleurs sombres afin de réduire la luminosité et donc de réduire la consommation d’énergie. J’ai également créé une interface sobre et épurée en utilisant la typographie de base et en limitant les éléments inutiles.

L’accessibilité, elle, est le fait d’avoir une interface simple, compréhensible et utilisable par tous types de personnes (avec ou sans handicap). Pour ce faire, j’ai différencié chaque objet avec un contraste afin de permettre à l’utilisateur de correctement différencier chaque objet.

J’ai donc mélangé l’éco-conception et l’accessibilité afin de produire une interface économe tout en restant accessible par n’importe qui.

***Conception de l’UX***

Ce chapitre résume la conception des maquettes des différents écrans du jeu.

Il se résume en 3 chapitres, les Wireframes, l’éditeur de niveau en haute-fidelité et les choix effectués.

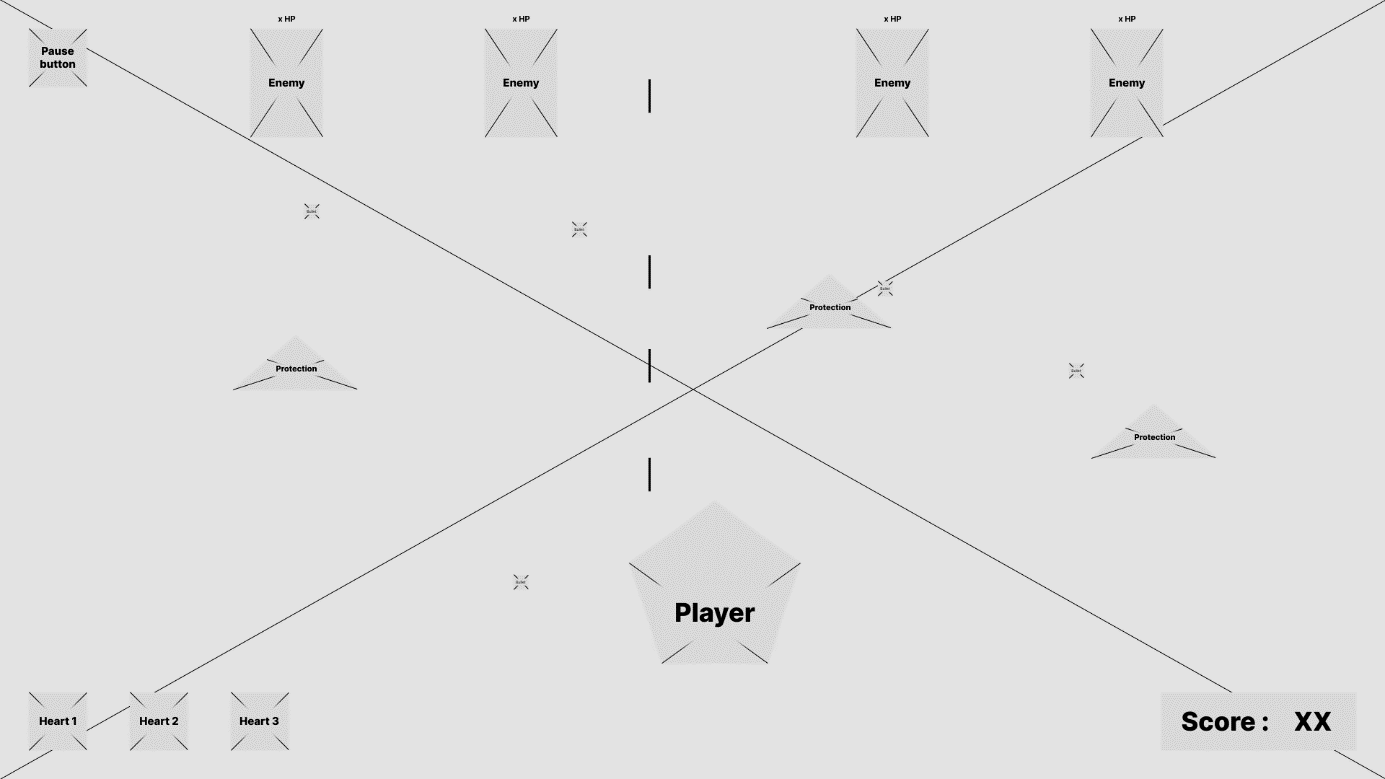
**Définitions des Wireframes**

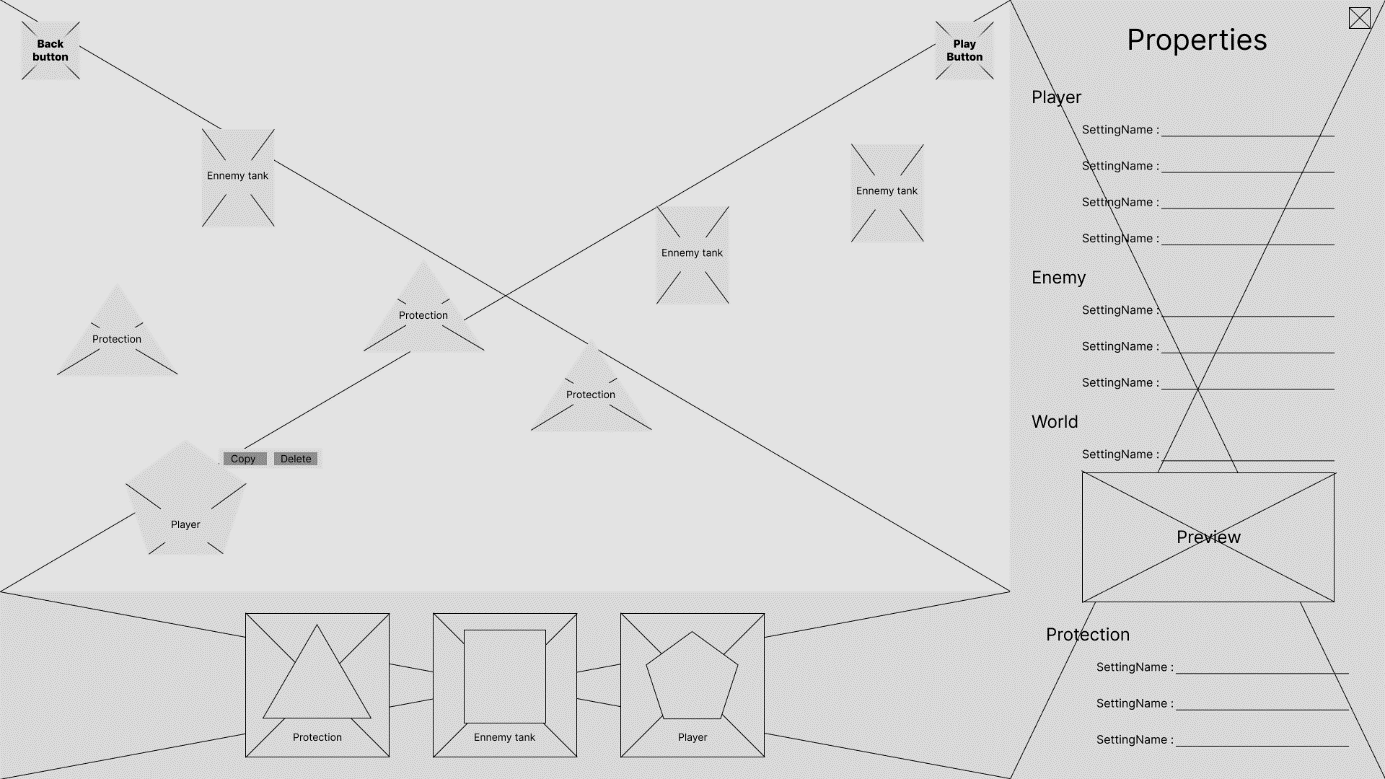
Un wireframe est une représentation simplifiée d'une interface en utilisant uniquement des lignes pour décrire la structure de l’interface. Il ne comporte aucune couleur et aucune « données ». Il sert souvent à visualiser la disposition d'un design. Ci-dessous se trouvera donc tous les Wireframes demandés :

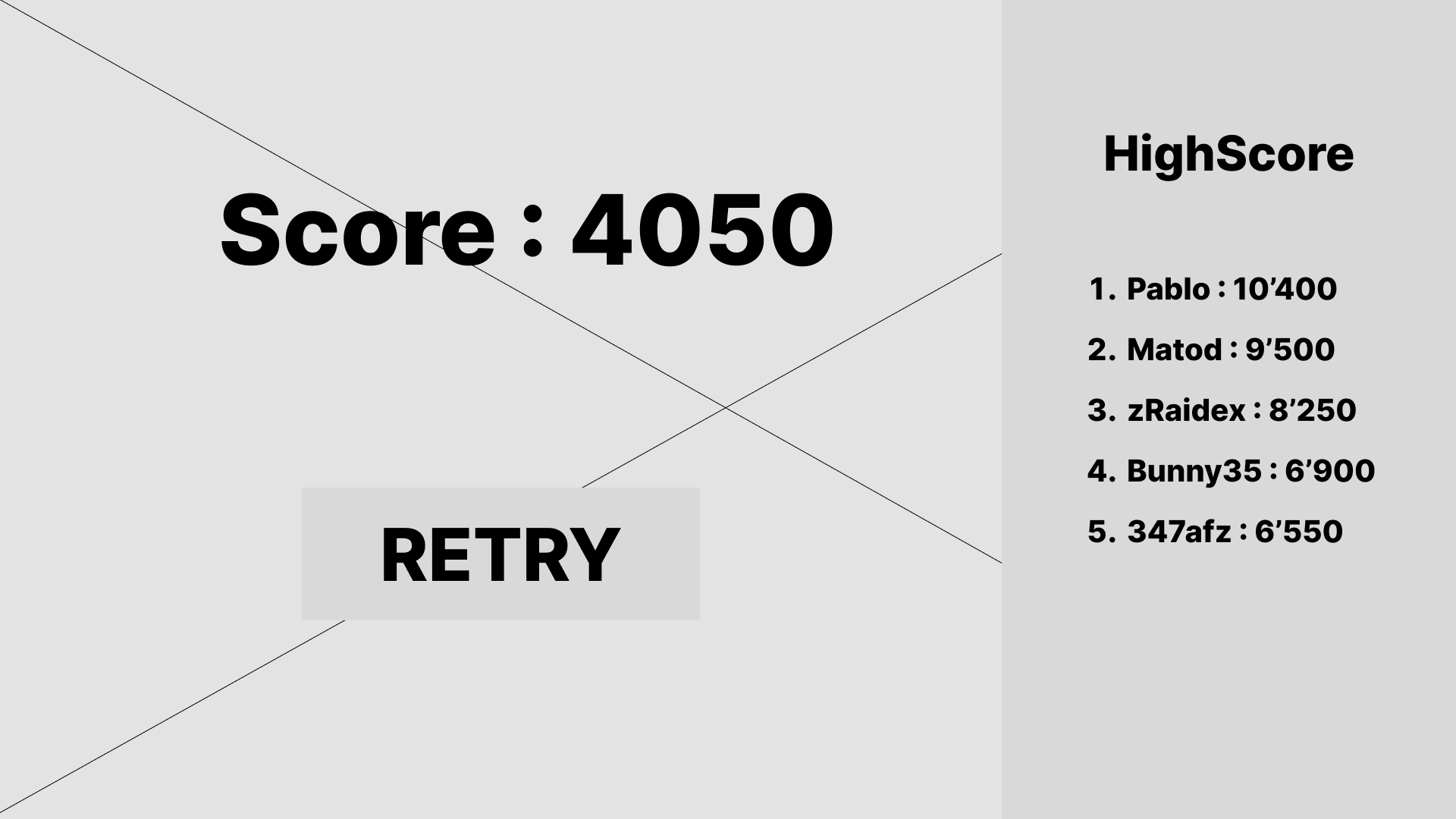
* Menu principal
* Ecran de jeu
* Editeur de niveau
* High scores

Les voici :









* **Définition de l’High fidelity**

Une maquette hautefidélité est une maquette avancée du Wireframe, il comporte des couleurs, des polices et ressemble au rendu final de l’interface. Cependant l’interface n’est pas cliquable et/ou utilisable. Cependant, pour cette interface, il nous était demandé uniquement l’interface de l’éditeur de niveau.

Pour le choix de la map, l’utilisateur peut dérouler un drop down qui affiche une liste de map prédéfinies. L’utilisateur peut en choisir une, la map actuelle est écrite sur le drop down, et un aperçu est affiché juste en dessous.



* **Choix effectués**

Pour ces différentes maquettes, j'ai donc décidé de partir sur un design sobre, simple et efficace. Tout en rendant l'interface lisible et compréhensible pour tout type de personne en mélangeant correctement les couleurs préalablement choisies sur la palette.

**Diagramme UML**

Un diagramme UML est un schéma visuel permettant de visualiser la structure et le comportement du logiciel. Mon diagramme UML, étant trop grand, ne rentrait pas dans le rapport. Il peut être visualisé en cliquant [ici](../POO/docs/diagrammeUML.pdf)ou dans le dossier P\_ShootMeUp/POO/docs/diagrammeUML.pdf!

* *Diagramme de classe*
* *Diagramme(s) d’état*
* **Analyse fonctionnelle**

### Sprites

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur, je souhaite avoir différents sprites, afin d'embellir le jeu. |
| Tests d'acceptance:   |  |  | | --- | --- | | Joueur | Durant la partie, quand je regarde mon joueur, c'est un militaire avec une arme. | | Background | Durant la partie, quand je regarde le fond, c'est une map (rue) | | Protections | Depuis la partie, quand je regarde les protections ce sont des sacs de sable avec des barbelés. | | Points de vie du joueur | Depuis la partie, quand je regarde les points du vie du joueur, ce sont des casques militaires pour chaque HP. | | Point de vie du tank | Depuis la partie, quand je regarde le tank je vois au dessus, qu'il a un sprite qui montre ses vies sous forme de LifeBar | | Chargeur | Depuis le jeu, quand je regarde les munitions restantes, il y a une icone de munitions à coté du nombre de balles restantes. | |

### Niveaux

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur je souhaite pouvoir jouer a différents niveaux, afin de pouvoir éviter une répetition. |
| Tests d'acceptance:   |  |  | | --- | --- | | Niveau suivant | Depuis la partie, quand je fini le niveau actuel, je passe automatiquement au niveau suivants | | Affichage | Depuis la partie, quand je regarde en haut a gauche, je vois le niveau actuel | | Difficulté | Dans le jeu, le premier niveau contient 2 tanks, puis a chaque niveau + 2 tanks. Des qu'il y a 8 tanks, le jeu passe en mode infini avec un tank qui réparait a chaque mort | |

### Obstacles

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur je souhaite pouvoir avoir des obstacles, afin de me protéger des missiles ennemis. |
| Tests d'acceptance:   |  |  | | --- | --- | | Position | Pendant la partie, quand je clic droit je peux poser des protections ou je clique dans une limite ou les tanks ni le joueur ne peuvent entrer. (ex: impossible de poser une protection derrière le joueur) | | Vie | Depuis la partie, quand je regarde les obstacles ils peuvent recevoir 5 balles avant de disparaître | | Protection | Depuis la partie, quand un obstacle reçoit une munition, la munition se supprime et l'obstacle perd un point de vie | | Limite | Dans la partie, je peux poser deux protections, puis une toutes les 20 secondes. | |

### Ennemi

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur je souhaite combattre contre des ennemis afin d'ajouter de la difficulté au jeu |
| Tests d'acceptance:   |  |  | | --- | --- | | Lancement | Au lancement du jeu, un nombre X d'ennemi apparaissent en dehors de l'écran et avance jusqu'à une position fixe à une certaine position X, Y. | | Déplacement | Durant la partie, quand je regarde les ennemis, ils ne peuvent pas se déplacer une fois leur position finale atteinte. | | Tire | Depuis le jeu, quand je regarde les ennemis ils peuvent me tirer dessus en visant ma position. | | Délai tire | Depuis le jeu quand je regarde les ennemis, ils doivent attendre un délai de 2.2s avant de pouvoir retirer | | Sprites | Durant la partie, quand je regarde les ennemis, ce sont des tanks. | | Points de vie | Dans le jeu, quand je regarde au-dessus des ennemis, ils ont une barre de HP (ils résistent à 2 tirs). | | Dégats | Dans le jeu, quand je touche un ennemi avec une balle, il perd des points de vie. | | Mort | Dans le jeu quand un ennemi n'a plus de point de vie, il meurt et disparait | |

### Joueur

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur, je souhaite avoir un personnage a déplacer, pour interagir avec le jeu |
| Tests d'acceptance:   |  |  | | --- | --- | | Déplacement | Depuis le jeu, en appuyant sur certaines touches, mon joueur se déplace sur l'axe X. | | Touches de déplacement | Dans le jeu, quand je clique sur A,D mon personnage se déplace sur un côté. | | Limite | Depuis le jeu quand je me déplace je ne peux pas sortir de la bordure de l'écran avec une marge de 50. | | Tir | Depuis le jeu, quand je clique sur le bouton gauche de ma souris, le joueur tire une munition qui part tout droit (verticalement) | | Délai | Dans le jeu, après avoir tiré, je ne peux plus tirer pendant 200 millisecondes. | | Munitions | Dans le jeu, quand je regarde en bas, j'ai une limite de 15 munitions. | | Chargement de l'arme | Depuis le jeu, quand le joueur appuie sur 'R' l'arme se recharge durant 1.5 seconde et le joueur ne peut plus tirer. | | Calcul de trajectoire de la munition | Depuis le jeu quand une balle sort de l'écran elle se supprime. | | Points de vie | Depuis la partie mon joueur a 3 de points de vie et ceux-ci peuvent être vérifié en regardant en bas à gauche de l'écran | | Dégâts personnels | Dans le jeu, quand je me fais toucher par une balle, je perd des points de vie. | | Mort | Dans le jeu quand je n'ai plus de points de vie je meurt et la partie se finit | |

### Lancement

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur je souhaite pouvoir lancer le jeu afin de pouvoir y jouer |
| Tests d'acceptance:   |  |  | | --- | --- | | Menu | Depuis le bureau, quand je lance le jeu j'arrive sur un écran de menu avec un Bouton start et exit | | Bouton | Depuis le menu quand je clique sur le bouton start le niveau se lance avec le joueur, ennemi, et la possibilité de poser des protections | | Tank | Dans le jeu quand les ennemis apparaissent ils ne se chevauchent pas. | | Relancer | A la fin de la partie, quand jappuie sur restart, je peux rejouer une partie | | Interface | Au lancement du jeu, il y a une interface fonctionnel avec les boutons jouer, et quitter | | Pause | Dans la partie, quand le joueur appuie sur ESC, un menu de pause s'ouvre | |

### Collisions

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur je souhaite avoir des collisions afin de pouvoir gagner ou perdre la partie |
| Tests d'acceptance:   |  |  | | --- | --- | | Tank | Dans le jeu quand une munitions rentre dans un tank le tank perd une vie | | Protections | Dans le jeu, quand une munitions allié ou adverse rentre dans une protection elle perd une vie. | | Munitions | Dans le jeu quand une munitions rentre en contact avec un tank, un joueur ou une protection, elle se supprime | | Joueur | Dans le jeu, quand le joueur se fait toucher par un missile ennemi, il perd une vie | | Protections placement | Dans le jeu, quand le joueur essaye de poser une protection par dessus une protection déja existante, elle ne se place pas | |

### Score

(Auteur: Mathis Olaya)

|  |
| --- |
| En tant que joueur, je souhaite avoir des scores, pour voir mon niveau |
| Tests d'acceptance:   |  |  | | --- | --- | | Position | Dans le jeu, quand je regarde en bas à droite je vois le score | | Calcul | Dans le jeu quand je vois le score, je remarque qu'il s'incrémente uniquement de 5 pts lorsque je tue un tank. | |

***Reprendre le contenu des User Stories d’IceScrum : Story + tests d’acceptance (avec IceTools)***

* **Stratégie de test**

Pour vérifier le bon fonctionnement de mon jeu vidéo, je vais effectuer diverses séries de test qui me permettront d’évaluer les améliorations a effectuer. En premier temps je vais vérifier les tests d’acceptances IceScrum. Je vais les reprendre un par un et vérifier que le travail effectuer respectent bien les tests créer au préalable.

Ensuite, je vais créer une solution de test (MStest) sur mon projet, et y créer différents tests sur des parties spécifique du jeu. Les tests s’exécuterons par la suite automatiquement.

Et finalement, je vais faire tester mon jeu à plusieurs camarades, pour avoir leur ressenti, et vérifier qu’il n’y a pas de bugs.

Toute cette stratégie de test me permet de vérifier différents aspects du jeu, le contenu est respecté, le jeu est fonctionnel, etc… Cependant il est tout de même très compliqué d’avoir une couverture de test complète pour quelques raisons. Déjà, effectuer des tests pour chaque partie du code prendrait beaucoup de temps. De plus certaines parties sont très compliquées à tester. Exemple :

* Imaginons que nous souhaitons tester le fait que quand le joueur appuie sur A, le joueur se déplace sur la gauche. Et bien, cela est en faite compliquer (dans mon cas), car je serais obliger de simuler l’appuie de la touche ‘A’, ce qui est plus ou moins compliqué.

***Décrire quels sont les MOYENS utilisés pour faire les tests, ne pas décrire les tests à effectuer !!!***

***Décrire l’environnement dans lequel se fait la sprint review***

*Décrire la stratégie globale de test :*

* *types de des tests et ordre dans lequel ils seront effectués.*
* *les moyens à mettre en œuvre.*
* *couverture des tests (tests exhaustifs ou non, si non, pourquoi ?).*
* *données de test à prévoir (données réelles ?)* ***et comment elles seront mises en place****.*
* *les testeurs extérieurs éventuels.*

**Réalisation**

***Points de design spécifiques***

1. Placement de protections spécifique.

Lorsque le joueur essaye de placer une protection, il y a une multitude de conditions qui sont vérifiées :

* + La protection se trouve dans la zone de placement ?
  + Le joueur dispose de protections ?
  + La protection ne superpose pas une autre déjà existante ?

Pour ce faire voici l’ordre de l’exécution du code.

1. Créer une hitbox pour le rectangle dont la position est centrée sur le curseur de la souris, avec une hauteur et largeur définie par la texture. Pour calculer la position, il faut utiliser : Position X du curseur - (Largeur de la protection / 2 \* échelle de la texture), et faire de même pour Y.
2. Vérifier que le joueur appuie sur le clic droit tout en vérifiant qu’il ait relâché le clic précèdent. Cela permet d’éviter qu’en un clic, il pose plusieurs protections. Et il faut également vérifier que la protection se trouve dans les limites (entre les tanks et le joueur). Pour calculer cette zone, voici le calcul :

Créer un rectangle, sa cordonnée X est 0 car il peut poser de gauche à droite. Puis calculer sa position Y de départ. Nous savons que la limite doit se situer plus loin que le tank. Donc, on prend la position (sa position se trouve au centre de la texture) limite du tank, et on y ajoute la moitié de la hauteur de la texture du tank. Grâce à cela, la limite commencera au bout du canon du tank ! Ensuite, la largeur est tout l’écran (donc 1920). Et finalement la hauteur est un calcul complexe. Nous savons que le rectangle de la limite acceptée se finit devant le bout du fusil du joueur, donc la hauteur de l'écran (1080) - la zone supprimée en haut (car sinon le rectangle est décalé étant donné qu'on l'a abaissé.) - la hauteur du joueur. Tout cela nous donne une zone délimitant au pixel près la limite où les protections peuvent être placées.

1. Après cela, il faut vérifier le mode de constructions. Au lancement le joueur peut poser 2 protections sans devoir attendre, puis après cela, une toutes les 20 secondes. Donc le programme vérifie s’il a déjà posé 2 constructions (la limite), si non, le programme place une protection (la logique de placement est expliquée plus loin). Puis incrémente d’un le nombre de protections posées. S’il à poser 2 protections, on passe le mode construction toutes les 20 secondes.
2. La partie précédente concernait donc le mode placement de deux protections. Mais dans le cas inverse, si le joueur a déjà posé ces deux protections, on vérifie si le joueur a attendu assez de temps pour en placer une nouvelle (20 secondes). Si oui, alors on place une construction. A noter que dans la méthode update, on vérifie si le mode est en mode placement unique, et si oui, on augmente le timer.
3. Pour placer les protections, une autre logique est mise en place, il faut vérifier que la nouvelle protection ne se superpose pas avec les autres protections. Le programme détient une liste de toutes les protections actuellement en vie. En premier temps on vérifie si dans toute la liste, notre protection intersecte avec une autre protection existante. Si oui, alors on arrête le tout. Une fois que toute la liste a été vérifiée, et que celle-ci n’a pas été interrompue, alors cela signifie que la protection ne se superposera pas avec une existante. Nous plaçons donc la protection.
4. Faire supprimer les munitions lorsqu’elles sortent de l’écran

Lorsque les munitions sont tirées, il se peut qu’elles sortent de l’écran. Il est primordial de les supprimer afin d’éviter que le programme les calculs sans cesses. Cela peut causer des problèmes techniques s’il y a beaucoup de munitions. Pour chaque munition, la méthode Update() s’effectue à chaque frames. Le programme vérifie donc à chaque tic s’il la balle est en dehors de l’écran.

1. La méthode CheckLimitPosition() s’effectue
2. Cette méthode contient :

If(isOutOfBounds)

EntityManager.Remove(this) ;

1. isOutOfBounds est une propriété de type bool qui retourne true si Position.Y < 0 || > Position.Y > LaHauteurDelEcran. Le ‘||’ est là car les munitions vont dans les sens (haut et bas), car il y a les munitions alliées et ennemies.
2. Si le résultat est true, alors la méthode Remove s’effectue ayant pour object la munition. Elle est ensuite retirée de la liste.

***Ce chapitre est constitué de plusieurs sous-chapitre.***

***Chaque sous-chapitre explique un point de design technique particulier, quelque chose que vous avez dû inventer pour répondre au besoin et qui ne peut pas s’expliquer par de simples commentaires dans le code.***

***Il s’agit d’explications techniques sur le fonctionnement du système. Les explications sont appuyées par des diagrammes, ou de très brefs éléments de code.***

***NE PAS mettre ici des pratiques usuelles que tout professionnel de la branche connaît déjà. Par exemple, n’EXPLIQUEZ PAS ICI CE QU’EST LE PATTERN MVC.***

***Exemple (simplifié à l’extrême) : Protection contre des formulaires mal intentionnés ou modifiés***

* ***Au moment de générer le formulaire, le script php :***
* ***Concatène les noms de tous les champs contenus dans le formulaire***
* ***Calcule un hash SHA256 de la chaîne obtenue***
* ***Ajoute un input nommé « CSRF » de type hidden dans le form***
* ***A la réception du POST du fromulaire***
* ***Concatène les noms des indices de $\_POST***
* ***Calcule un hash SHA256 de la chaîne obtenue***
* ***Vérifie que la valeur du champ CSRF correspond***
* ***…***
* ***…***
* ***…***
* **Déroulement**

Le déroulement des story s’est globalement bien passé, cependant j’ai eu quelques problèmes, notamment dans la réalisation de l’apparition des tanks.

Le problème était que certains tanks se superposaient lors de l’apparition. Au départ, j’ai essayé de comparer la position de tous les tanks entre-deux, et d’en générer un ne croisant pas avec les autres tanks déjà existants. Mais cela posait un problème, car le programme essayait de générer une position aléatoire valable au minimum 100 fois avant de comprendre qu’aucune position n’étaient disponibles. Donc à partir d’un certain nombre de tank, cela faisait crasher le jeu et/ou pouvait provoquer des longs délais d’attente. Pour régler ce problème, j’ai créé un dictionnaire contenant comme clés des positions, et comme valeurs des booléens disant si la position est occupée ou non. Au lancement, le programme regarde chaque position disponible grâce au booléen, et choisi aléatoirement une position parmi celle-ci.

Sinon, mise à part celle-ci, toutes les autres story se sont déroulées sans accrocs.

***Résumer comment s’est passé la réalisation de chaque story, ses difficultés, les alternatives envisagées mais rejetées, ses surprises, …***

* **Mise en place de l’environnement de travail**

**Installation des logiciels :**

1. Visual Studio

Se rendre sur le site de [Visual Studio](https://visualstudio.microsoft.com/fr/), et installer la version Community, qui ne demande pas de licence.

1. Monogame

Il est important d’installer le moteur de jeu, pour cela lancer Visual Studio, dans l’onglet en haut de l’écran, sélectionner extensions, puis gérer les extensions. Et dans la barre de recherche, taper « Monogame ». Installez le et redémarrez VS.

1. GitHub Desktop

GitHub desktop s’avère être très utile, et permet de récupérer le travail à jour en quelque clic. Pour cela, il faut se rendre sur le site de Github, et d’y installer la version Desktop. Connectez vous à votre compte, et cloner le repos contenant la solution.

**Environnement :**

Pour accéder au code source du projet, il suffit de se rendre sur [GitHub](https://github.com/MathisOlaya/P_ShootMeUp), de cloner le projet, puis d’ouvrir la solution à : « POO/TankBattleV2/TankBattleV2.sln ». A partir de là vous aurez accès à toutes les classes et donc tout le code.

La version de mon système d’exploitation est Windows 10 22H2 19045.4894. J’ai également utilisé Visual Studio, dont la version est 17.9.4.

Pour le matériel, un simple PC de bureautique avec un Intel® Core™ i7-11700 de 2.50Ghz, 32 GO de RAM, 512 GO de stockage, et un intel® UHD Graphics 750 a été utilisé.

**Liste des fichiers :**

Ci-dessous se trouve la liste complète des fichiers du jeu. En premier temps se trouve les classes, puis les autres fichiers.

Classe :

Bullet.cs : Configuration et calcul des tirs ennemis et alliés.

Player.cs : Logique de déplacement, placement de protection, tir, du joueur.

Protection.cs : Configuration et dessin de la protection

Tank.cs : Logique de tir automatique, déplacement automatique

Entity.cs : Modèle pour toutes les entités avec des propriétés définies (Tout objet ayant une position, point de vie etc..)

IEntity.cs : Défini toutes les méthodes qu’une entité doit inclure

IMovable.cs : Défini toutes les propriétés qu’une entité ayant la capacité de bouger doit inclure

IShootable.cs : Défini toutes les propriétés qu’une entité ayant la capacité de tirer doit inclure

EntityManager.cs : S’occupe de gérer toutes les entités (Update, Draw, Suppression, Collision, etc…)

Level.cs : S’occupe de lancer la partie, et d’initier les entités. Avec un nombre de tank prédéfinis.

Config.cs : Gère la configuration de l’écran et les paramètres de la partie.

EntityConfig.cs : Défini toutes les caractéristiques de chaque entité (ex : vitesse, cooldown de tir, etc…)

GlobalHelpers.cs : Gère les entrées de l’utilisateur et la création de nombre aléatoire.

Visuals.cs : S’occupe de charger tout les aspects visuels du jeu au lancement.

Buttons.cs : Permet la création de bouton.

Menu.cs : Permet la création d’une interface en utilisant des boutons.

Program.cs : S’occupe de créer GameRoot.

GameRoot.cs : Fichier par défaut, gérant le jeu.

Autres :

\*dossier Content : Stock tous les sprites du jeu.

app.manifest : Fichier de configuration de l’application.

TankBattleV2.sln : Solution du projet.

* ***Comment accéder au code source***
* *la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
* *les versions des systèmes d'exploitation et des outils logiciels*
* *la description exacte du matériel*

***Ce chapitre décrit précisément comment un employé qualifié peut recréer l’environnement dans lequel vous avez effectué ce travail***

* **Description des tests effectués**

***Reprendre les tests d’acceptance d’IceScrum au moyen de la feuille ad hoc d’IceTools***

* **Erreurs restantes**

*S'il reste encore des erreurs :*

* *Description détaillée*
* *Conséquences sur l'utilisation du produit*
* *Actions envisagées ou possibles*

***Reporter la*** [***dette technique***](https://www.premaccess.com/qu-est-ce-que-dette-technique-comment-la-maitriser/) ***connue. S’appuyer sur la pratique des // TODO***

* **Conclusions**

Pour conclure, je vais tout résumer en 4 points pertinents :

1. Les objectifs ayant été atteints ou non
2. Les points positifs et négatifs du projet
3. Les différentes difficultés du projet
4. Les améliorations possibles sur le projet
5. Objectifs

Je suis satisfais de mon travail, car j’ai réussi à créer un jeu fonctionnel, avec des déplacements, tirs, protections. Les objectifs principaux que je m’étais fixés ont été réalisés avec succès, et tout est fonctionnel. Cependant, avec plus de temps, j’aurai pu ajouter certaines fonctionnalités au jeu, tel qu’un boss final.

1. Positifs / négatifs

Ce projet est selon moi positif car il introduit correctement l’utilisation des classes et des interfaces. Et de plus, la réalisation d’un jeu rend le projet plus amusant.

Je n’ai pas particulièrement apprécié l’utilisation d’IceScrum, car créer une structure efficace y prend trop de temps, un temps qui pourrait être mieux investi dans le développement du code. Cela est particulièrement vrai lorsqu’on utilise une méthode agile en étant seul sur le projet.

1. Difficulté du projet

Le projet a été complexe au départ, car je ne connaissais pas le moteur de jeu MonoGame. J'ai donc dû apprendre à l'utiliser et l'adapter à mon projet.

1. Améliorations possibles

Si je devais améliorer mon jeu, j’ajouterais certaines fonctionnalités, tels qu’un boss final, ou encore des habilités que le joueur pourrait utilisé.

* **Annexes**
* **Manuel de référence**

***Issu de la génération automatique à partir des commentaires***

* **Journal de travail**