|  |
| --- |
| Shoot me up ! |

Pour projet Shoot Me Up avec XCL

Table des matières

[1 Analyse préliminaire 3](#_Toc181130216)

[*1.1* Introduction 3](#_Toc181130217)

[*1.2* Objectifs 3](#_Toc181130218)

[1.3 Gestion de projet 3](#_Toc181130219)

[2 Gameplay 4](#_Toc181130220)

[2.1 Généralités 4](#_Toc181130221)

[2.2 Le joueur 4](#_Toc181130222)

[2.3 Les ennemis 4](#_Toc181130223)

[2.4 Les obstacles 4](#_Toc181130224)

[2.5 Les objets ramassables 4](#_Toc181130225)

[2.6 2Les niveaux 4](#_Toc181130226)

[2.7 Concept 5](#_Toc181130227)

[2.7.1 Diagramme de classe 5](#_Toc181130228)

[2.7.2 Diagramme d’état 6](#_Toc181130229)

[*2.8* Analyse fonctionnelle 6](#_Toc181130230)

[*2.9* Stratégie de test 6](#_Toc181130231)

[3 Réalisation 6](#_Toc181130232)

[3.1 Points de design spécifiques 6](#_Toc181130233)

[3.1.1 Game.cs 6](#_Toc181130234)

[3.1.2 GameElement.cs 6](#_Toc181130235)

[3.1.3 Les méthodes de déplacement 7](#_Toc181130236)

[3.1.4 Spawn.cs 7](#_Toc181130237)

[*3.2* Déroulement 8](#_Toc181130238)

[*3.3* Mise en place de l’environnement de travail 8](#_Toc181130239)

[*3.4* Description des tests effectués 8](#_Toc181130240)

[*3.5* Erreurs restantes 8](#_Toc181130241)

[4 Conclusions 9](#_Toc181130242)

[4.1 Objectifs atteints 9](#_Toc181130243)

[4.2 Objectifs non-atteints 9](#_Toc181130244)

[4.3 Difficultés particulières 10](#_Toc181130245)

[5 Annexes 11](#_Toc181130246)

[*5.1* Manuel de référence 11](#_Toc181130247)

[*5.2* Journal de travail 11](#_Toc181130248)

# Analyse préliminaire

## Introduction

Dans le cadre de ma formation d’informaticienne à L’ETML, chaque étudiant est amené à réaliser un jeu de tir en C# sur Windows. Ce travail s’étale sur un total de 80 périodes, elles-mêmes réparties sur le premier trimestre.

## Objectifs

L’objectif de ce projet est de mettre en pratique nos acquis du module “ICT-320”à travers la création d’un jeu vidéo. Pour ce faire, nous devions programmer un jeu de tir du genre “Shoot’em up” et fournir un rapport détaillant les attentes et le résultat final.

Le code produit doit respecter les points suivants :

* Respect des normes imposée par l’établissement
* Un code clair et précis
* Un code optimisé grâce à une bonne utilisation des structures
* Des tests unitaires
* Un code commenté là où c’est nécessaire

Le résultat final du jeu doit comprendre les fonctionnalités suivantes :

* Un système de niveaux de jeu
* Le joueur
* Des ennemis
* Des obstacles

## Gestion de projet

Pour la gestion de ce projet, j’ai opté pour la méthode “agile scrum”. Cette technique consiste à planifier des « sprints », des éléments du projet complets, qui peuvent être d’une durée de quelques heures à plusieurs jours. Les sprints représentent une partie du programme à réaliser et ont l’avantage d’être très flexibles et de permettre, à tout moment, d’ajouter ou de retirer certaines fonctionnalités du projet. Pour ce projet, il a été décider de n’utiliser qu’un seul sprint pour la durée totale du projet. Ainsi la méthode agile est facile à mettre en place et offre une grande liberté dans le choix de l’avancement du projet.

Ainsi, je pouvais visualiser l’ensemble des tâches effectuées et des tâches à réaliser, ce qui m’a permis d’estimer le temps de travail nécessaire à chaque tâche afin de m’organiser.

Pour appliquer au mieux la méthode agile et l’utilisation des sprints, j’ai eu recours au site IceScrum

# Gameplay

## Généralités

Les différents aspects du gameplay du jeu ont été imposés par le client. Concrètement, le jeu demandé est un « Shoot’em up ». C’est un jeu de tir à la troisième personne en 2D, similaire aux classiques du genres que sont « Space Invader » ou encore « Asteroids ».

## Le joueur

Le joueur possède un sprite unique en jeu. Il peut se déplacer dans quatre directions ; haut, gauche, bas et droite. Les touches utilisées sont respectivement W, A, S et D.

Au début d’une partie, le joueur possède 5 points de vie qu’il perd en entrant en collision avec un ennemi ou avec un projectile ennemi. Une fois les 5 points de vies perdus, le joueur meurt. Il peut mourir 3 fois avant que la partie ne soit terminée.

Le joueur peut tirer des projectiles dans les mêmes directions que les déplacements à l’aide des flèches directionnelles. Le joueur peut aussi ramasser une arme alternative qui modifie la manière de tirer du joueur. Contrairement à l’arme de base qui a des munitions infinies, cette dernière possède des munitions limitées, vingt. Ces dernières peuvent, elles aussi, être ramassées au sol.

## Les ennemis

Il existe deux types d’ennemis ayant chacun leur propre sprite. Chaque ennemi possède 10 points de vie et disparait en mourant.

Le premier type d’ennemi peut se déplacer en poursuivant directement le joueur. Le deuxième type d’ennemi reste dans un coin du jeu pour tirer sur le joueur.

## Les obstacles

Les deux types d’obstacles définis sont les rochers et les barricades. Tous deux peuvent bloquer les tirs ~~(~~du joueur ou des ennemis~~)~~. Ils sont aussi impossible~~s~~ à traverser, autant pour le joueur que pour les ennemis. Les rochers sont indestructibles par le joueur tandis que les barricades sont détruites au troisième projectile intercepté.

## Les objets ramassables

Lors d’une partie, le joueur peut ramasser des objets qui apparaissent aléatoirement sur le jeu :

• Le soin : redonne la totalité de ses points de vie au joueur

• L'arme alternative : apparaît avec des munitions limitées

• Les paques de munitions pour l’arme alternative

## Les niveaux

Deux niveaux sont actuellement attendu pour le jeu.

Le premier niveau doit posséder plusieurs obstacles pour aider le jeu à se déplacer loin des ennemis. Il y aura davantage de rochers que de barricades. Les ennemis apparaissent par vague et chaque vague d’ennemi apparaît aléatoire sur le jeu au début du niveau et après que le joueur ait tué tous les monstres.

Le deuxième niveau les monstres apparaissent toujours par vague du début à la fin. La première différence est que les monstres apparaissant toutes les 20 secondes ou si tous les monstres de la vague sont morts. La deuxième différence vient du nombre d’obstacle moins nombreux que sur le premier niveau et il n’y a que des barricades.

## Concept

### Diagramme de classe

Une image contenant texte, diagramme, Parallèle, Plan

Description générée automatiquement

### Diagramme d’état

CEST QUOI ?

## Analyse fonctionnelle

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

## Stratégie de test

***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

### Game.cs

La classe “Game” est la classe chargée de gérer tout le fonctionnement du jeu. Il gère l'affichage de tous les éléments du jeu, leurs déplacements mais aussi leurs collisions et leurs suppressions de l’écran. C’est aussi dans cette classe je gère les évènements au moment où le joueur appuie sur une touche pour se déplacer ou pour tirer dans une direction.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

### GameElement.cs

La classe “GameElement” est la classe qui sert de représentation pour tous les objets qui peuvent apparaître sur le jeu. Dans le langage de programmation, «GameElement » est la classe parent de la sorcière, des projectiles, des slimes et des rochers.

Chaque objet est géré par un objet Rectangle pour l’affichage, la position et ses déplacements. Chaque objet possède aussi des points de vies pour gérer son affichage ou non.

Cette classe me permet aussi de gérer la collision et la fin de vie d’un objet avec les fonctions “IsTouching”, qui prend en paramètre un autre Rectangle, et “IsDead”.

### Les méthodes de déplacement

Les classes Witch, Slime et Projectil ont leur propre fonction “Move”. En effet, bien que mon code puisse être similaire dans mes classes “Witch” et “Projectil” par exemple, les deux fonctions n’ont pas les mêmes paramètres. Il était préférable de ne pas avoir une fonction de déplacement en héritage car les objets qui ne bougent pas auraient hérité pour rien de cette fonction. C’est le cas de l’objet spawn qui ne contient et n’as pas besoin de mobilité. Mais utiliser une interface “IMovable” par exemple aurait pu être une solution.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Figure 1Fonction Move Witch.cs

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Figure 2 Function Move Projectil.cs

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 3 Function Move Slime.cs

### Spawn.cs

Cet élément a été ajouté pour me faciliter la gestion de l’apparition des ennemis. N’arrivant pas à utiliser l’entièreté de la zone de jeu comme zone d’apparition, j’ai créé quatre zones plus petites. Ces zones ont une taille et une position fixe et apparaissent au début du jeu. Ainsi lors de la création des slimes, ces derniers sont initialisé à la position des spawn.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 4 Apparition des slimes dans une zone

## Déroulement

Lors du déplacement du joueur, il y a eu une difficulté pour délimiter les bordures de la map.

***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

Le projet peut être retrouvé sur un [dépôt GitHub](https://github.com/AbigaelP/ShootAbby).

Le code est développé en C# avec .NET 8. C’est la librairie graphique WinForm qui est utilisé pour l’affichage du jeu.

Le développement est effectué à l’aide de Visual Studio 2022 et peut être exécuté sur Windows 10 et 11.

* ***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/#:~:text=La%20dette%20technique%20survient%20quand,de%20plus%20en%20plus%20fr%C3%A9quents.) ***connue. S’appuyer sur la pratique des // TODO***

# Conclusions

*Développez en tous cas les points suivants :*

* *Objectifs atteints / non-atteints*
* *Points positifs / négatifs*
* *Difficultés particulières*
* *Suites possibles pour le projet (évolutions & améliorations)*

## Objectifs atteints

J’ai pu effectuer les premières étapes du jeu qui permette de découvrir le but du jeu : tirer et tuer des ennemis.

Concernant le joueur, ce dernier peut se déplacer dans sa zone de jeu, ne pas sortir de cette zone de jeu et être bloqué par le décor. De plus, le joueur peut aussi tirer des projectiles qui s’arrêtent sur des ennemis, en dehors de la zone de jeu ou dans le décor. Ces projectiles sont supprimés de l’affichage une fois la cible atteins.

En ce qui concerne les ennemis, les slimes apparaissent uniquement dans leur zone d’apparition définie et poursuivent bien le joueur. Des dégâts sont cumulés sur le joueur lorsqu’il est touché par les slimes. De surcroît, des slimes réapparent en jeux quand il n’y en a plus sur la zone de jeu.

Pour terminer, chaque ennemi tué rapporte des points qui sont cumulés dans un score. De plus, la partie se freeze une fois que le joueur perd toute sa vie au contact des monstres.

## Objectifs non-atteints

En premier lieu, le joueur peut tirer en continue sans s’arrêter et sans limite de tir ou de munition. Non seulement on peut dire qu’il manque là une gestion du temps entre chaque tir du joueur mais de plus la limitation des munitions n’a pas été implémenté. Etant donné qu’il manque là une gestion du temps entre chaque tir du joueur et chaque dégât pris par les ennemis, le joueur prend des dégâts en continue durant les collisions.

En deuxième lieu le slime, qui poursuit le joueur, n’a pas de collision avec les rochers. D’un autre coté, l’ennemi des coins de la zone de jeu n’ont pas été implémenté dans le jeu actuel. Par conséquence, aucun projectile n’est tiré en direction du joueur.

De même que l’implémentation des barricades, des objets bonus ; soin, arme alternative et munitions n’ont pas été sur la zone de jeu pour aider le joueur dans sa partie.

Pour conclure, aucun système de niveau n’est présent dans le jeu et aucune difficulté apparait en fonction du temps de jeu. Si bien que le jeu n’apporte ni difficulté ni variation dans son fonctionnement.

## Difficultés particulières

Pour commencer, je suis partie dans le développement en créant directement mes classes les uns après les autres sans réfléchir aux éléments communs. De plus, je me suis concentré sur le fonctionnement individuel des classes. Par conséquent, j’ai éprouvé de la difficulté à regroupé grâce à l’héritage les éléments communs.

C’est pourquoi la conception le projet à prit du retard si bien que du code similaire a été implémenter plusieurs fois rendant la lecture des fonctionnalités plus difficiles. Ainsi j’ai eu du mal à préparer un diagramme de classe dès le départ et a organisé mes idées.

En deuxième lieu, j’ai remarqué que mes bases en programmation n’étaient pas solides. C’est-à-dire que je me suis retrouvée en difficulté à devoir utiliser les boucles foreach. Dans le projet l’utilisation des foreach est cruciale pour les collisions et la mise a jour des positions des objets. Par conséquent j’ai dû recommencer et utiliser plus de temps que prévus pour réussir à implémenter les fonctions dans Game.cs.

## Suites possibles pour le projet

Afin de pouvoir continuer le projet, un diagramme des classes contenant les informations sur les éléments manquants permettrait une plus grande facilité à organisé et implémenter le code. De plus, la gestion du visuel et des sprites.

# Annexes

## Manuel de référence

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

## Journal de travail