BowMasters



(Une image originale représentant le projet)

Maël Naudet – FID1

Lausanne

40 périodes

Patrick Chenaux

Table des matières

[1 Spécifications 3](#_Toc189218480)

[1.1 Titre 3](#_Toc189218481)

[1.2 Description 3](#_Toc189218482)

[1.3 Matériel et logiciels à disposition 3](#_Toc189218483)

[1.4 Prérequis 3](#_Toc189218484)

[1.5 Cahier des charges 3](#_Toc189218485)

[1.5.1 Objectifs et portée du projet (objectifs SMART) 3](#_Toc189218486)

[1.5.2 Caractéristiques des utilisateurs et impacts 3](#_Toc189218487)

[1.5.3 Fonctionnalités requises (du point de vue de l’utilisateur) 3](#_Toc189218488)

[1.5.4 Contraintes 4](#_Toc189218489)

[1.5.5 Travail à réaliser par l'apprenti 4](#_Toc189218490)

[1.5.6 Si le temps le permet … 4](#_Toc189218491)

[1.5.7 Méthodes de validation des solutions 4](#_Toc189218492)

[1.6 Les points suivants seront évalués 4](#_Toc189218493)

[1.7 Validation et conditions de réussite 4](#_Toc189218494)

[2 Planification Initiale 4](#_Toc189218495)

[3 Analyse 4](#_Toc189218496)

[3.1 Opportunités 4](#_Toc189218497)

[3.2 Document d’analyse et conception 4](#_Toc189218498)

[3.3 Conception des tests 5](#_Toc189218499)

[3.4 Planification détaillée 5](#_Toc189218500)

[4 Réalisation 5](#_Toc189218501)

[4.1 Dossier de Réalisation 5](#_Toc189218502)

[4.1.1 Fenêtre de jeu 5](#_Toc189218503)

[4.1.2 Création des joueurs 5](#_Toc189218504)

[4.1.3 Création des tours 5](#_Toc189218505)

[4.1.4 Création de la balle 5](#_Toc189218506)

[4.2 Modifications 6](#_Toc189218507)

[5 Tests 6](#_Toc189218508)

[5.1 Dossier des tests 6](#_Toc189218509)

[6 Conclusion 6](#_Toc189218510)

[6.1 Bilan des fonctionnalités demandées 6](#_Toc189218511)

[6.2 Bilan de la planification 6](#_Toc189218512)

[6.3 Bilan personnel 6](#_Toc189218513)

[7 Divers 6](#_Toc189218514)

[7.1 Journal de travail 6](#_Toc189218515)

[7.2 Bibliographie 6](#_Toc189218516)

[7.3 Webographie 6](#_Toc189218517)

[8 Annexes 6](#_Toc189218518)

# Spécifications

## Titre

BowMasters – Projet C# pour le module 320

## Description

Dans une application console, 2 joueurs, séparés par 2 grandes tours s’affrontent en se lançant des balles chacun à leur tour. Pour gagner la partie, il suffit de toucher 3 fois le joueur adverse.

## Matériel et logiciels à disposition

* Accès à Internet
* Visual Studio

## Prérequis

* Avoir suivi le module I319
* Suivre le module I320

## Cahier des charges

### Objectifs et portée du projet (objectifs SMART)

A compléter. Il s’agit d’ébaucher des réponses aux questions de l’acronyme CQQCOQP (Combien, Quoi, Qui, Comment, Où, Quand, Pourquoi)

### Caractéristiques des utilisateurs et impacts

A compléter… Il s’agit d’identifier le(s) profil(s) de(s) utilisateur-trice(s) type, et les conséquences que cela va avoir sur la conception (couleurs, ergonomie, utilisation, etc.)

### Fonctionnalités requises (du point de vue de l’utilisateur)

A compléter par une espèce de mode d’emploi du produit. S’il s’agissait d’une montre, décrire qu’à part l’heure, il y aura la possibilité d’utiliser un chronomètre, un réveil, …

### Contraintes

Sécurité, backups, disponibilité, système utilisé, interfaces avec autres logiciels, etc.

### Travail à réaliser par l'apprenti

Décrire à quoi doit ressembler le travail produit, ce qu’il faudra rendre …

### Si le temps le permet …

Objectifs complémentaires au cas où le projet n’est pas assez ambitieux dans le temps imparti…

### Méthodes de validation des solutions

Comment les tests vont être entrepris, quels tests doivent être entrepris, etc.…

## Les points suivants seront évalués

* Le rapport
* Le code et les commentaires
* Les documentations de mise en œuvre et d’utilisation

## Validation et conditions de réussite

* Compréhension du travail
* Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
* Etat de fonctionnement du produit livré

# Planification Initiale

Ce paragraphe présente le planning d'origine (date de début, date de fin, vacances et congés, liste hiérarchique des tâches ou GANTT, jalons, durée totale)

Toutes les mises à jour subies par le planning sont à reporter (avec date de mise à jour) et peuvent déboucher sur plusieurs versions de plannings.

# Analyse

## Opportunités

Ce paragraphe énumère la liste des difficultés potentielles de tout ordre :

Liste des compétences à acquérir ou approfondir

Liste du matériel à exploiter

Recherche d’informations particulières

Gestion du travail en équipe & collaboration

Ainsi que les solutions possibles

Si les spécifications de départ ne laissent pas de doutes sur la manière de réaliser un projet, ce chapitre ne fera que renvoyer le lecteur aux spécifications.

## Document d’analyse et conception

Ce paragraphe décrit le fonctionnement de manière détaillée.

Autant que possible de manière graphique, imagée, tableaux, etc.

Tous les cas particuliers devraient y être spécifiés…

Il s’agit d’y présenter les fonctionnalités à développer :

Découpage en étapes, en modules, en fonctionnalités, etc.

Formulaires, interfaces graphiques, pages web, etc.

Schémas de navigation, schémas événementiels, structogramme, pseudocode, etc.

Si le projet inclut une base de données :

Dictionnaire des données

Modèle conceptuel des données, modèles logique des données.

## Conception des tests

Ce paragraphe permet de spécifier la stratégie de test qui sera menée au point 5.1

Qui, quand, avec quelles données, dans quel ordre, etc.

## Planification détaillée

A ce stade, après l’analyse complète du projet, un planning détaillé et complet (avec tâches, sous-tâches, dépendances, durée, …) peut être finalisé.

Le planning détaillé doit s’inscrire dans le planning initial. Il faut que l’on puisse situer cette planification détaillée par rapport à la planification initiale.

# Réalisation

## Dossier de Réalisation

### Configuration de la fenêtre du jeu

Le projet a démarré avec la création de la fenêtre de jeu et sa modification. Pour cela, j’ai créé une classe Config avec les paramètres de grandeurs de la hauteur et de la largeur de la fenêtre et j’ai créé une méthode statique qui permet de changer de la taille de la fenêtre ainsi que la zone de mémoire tampon d’écran (pour éviter de pouvoir scroller dans la console).

### Gestion des positions

Pour gérer les positions des différents objets que je vais utiliser, j’ai créé une classe *PositionByte* qui a comme attributs la position X et Y de type byte dans l’écran directement.

### Gestion des Hitboxes

Comme certains objets ont besoin d’une Hitbox (zone de collision), j’ai créé une classe « *Hitbox* » qui contient une liste de positions qui forment un rectangle composant la Hitbox de l’objet.

L’explication sur comment j’ai utilisé ces différents attributs sera donnée plus tard.

### Création des joueurs

Les joueurs de la classe *Player* ont plusieurs propriétés comme la gestion de leur vie, leur position dans la grille de jeu, leur modèle et leur hitbox.

Ils peuvent être affichés avec la méthode Display(), peuvent prendre des dégâts avec la méthode TakeDamage(byte damage) et peuvent afficher leurs informations directement au-dessus d’eux avec leur nombre de point de vie, leur score et la barre de progression d’un lancer de balle.

### Création des tours

Pour faire la tour, j’ai d’abord créé les pièces avec la classe *TowerPiece* qui la composent en leur donnant un modèle et une position. J’ai permis de les afficher et de les enlever également. Ensuite, j’ai créé une classe *Tower* qui est en fait une liste de TowerPiece qui peut s’afficher avec une Hitbox et une Position.

### Création de la balle

Pour lancer la balle, il fallait d’abord faire une classe statique *Balistic* qui calcule le mouvement balistique d’un objet sur l’axe x et y en fonction du temps, de la vélocité initiale, de la position initiale de l’objet et de l’angle.

Ensuite, j’ai implémenté la balle en elle-même. La balle possède tous les attributs nécessaires à sa manipulation dans l’espace grâce aux méthodes de la classe *Balistic* donc une position et une vitesse initiale et un angle ainsi qu’une couleur aléatoire définie dès sa création grâce à une méthode dans une classe statique *Custom* qui retourne aléatoirement une couleur affichable dans une console windows.

Pour cela, plusieurs méthodes ont été implémentées comme celle qui permet de mettre à jour la position de la balle en fonction du temps, celle pour afficher la balle et celle pour l’effacer. Tout ça dans des try catch afin de faire en sorte que la balle ne se retrouve pas en dehors des limites de la fenêtre.

### Gérer l’angle de la balle

Pour pouvoir gérer l’angle de la balle, plusieurs points s’affichent et le joueur n’a cas appuyer sur espace pour bloquer un angle. Pour faire cela, j’ai dû parcourir différents angles, 0 à 90 ou 90 à 180 en fonction de la direction du tir, faire en sorte d’afficher un point au bon endroit et dès que l’utilisateur appuie sur espace, le dernier point affiché est bloqué et le dernier angle parcouru est affiché.

### Gérer la vitesse de la balle

Pour gérer la vitesse de la balle, le joueur doit appuyer sur espace pour faire progresser une barre et relâcher pour bloquer une vitesse. La vitesse retournée est dans tous les cas une valeur entre 0 et 50, qu’importe le temps maximum donné en paramètre. Au début, la barre de progression se remplissait toute seule et il suffisait d’appuyer sur espace pour l’arrêter mais ce n’est pas ce qui était demandé ; il faut maintenir la barre espace pour augmenter la vitesse. Pour ce faire, j’ai dû utiliser la méthode GetAsyncKeyState sur la barre espace qui permet de savoir en temps réel si une touche est appuyée afin de ne pas saturer le buffer. Et tant que cette touche est appuyée, le temps maintenu augmente avec un seuil maximum. Une fois la touche relâchée, la vitesse est retournée sous la forme d’un pourcentage entre le temps pressé et le temps maximum multiplié par 50 (comme ça je peux changer le temps maximum et toujours avoir des valeurs entre 0 et 50).

### Gérer les sons

Pour pouvoir faire les sons j’ai utilisé la classe implémentée *SoundPlayer* qui permet de lancer un son en mode console. Au début, j’ai simplement lancé un son quand je voulais qu’il se lance mais un temps de latence posait un problème. Alors, j’ai créé une classe *SoundEffect* qui possède un dictionnaire avec les différents fichiers sons ainsi que le nom qui les définit, une méthode pour pouvoir précharger les différents sons et les stocker dans le dictionnaire et une méthode pour lire le son. Maintenant, il suffit de précharger le son et de le lire afin d’avoir beaucoup moins de temps de latence qu’avant.

### Logique du jeu

La logique de mon jeu est gérée par la classe *BowGame.* Cette classe initialise le jeu, fais tourner la boucle du jeu et affiche le message de fin. Pour créer le jeu, il suffit de lui donner 2 joueurs et 2 tours, les deux gérés par un struct (*Players* et *Towers*) qui permet de ne pas pouvoir lancer un jeu sans le nombre requis d’objets.

#### Initialisation

L’initialisation du jeu affiche les joueurs, les tour et précharge les sons utilisés dans le jeu.

#### Boucle du jeu

La boucle du jeu suit un schéma simple. Le joueur 1 doit jouer alors il donne un angle et une vitesse grâce aux outils expliqués précédemment à sa balle et une méthode qui permet de le faire tout seul implémentée dans cette classe. La balle est lancée, un son de lancer est joué, et traverse le terrain tant que la balle ne touche ni une tour, ni le joueur en face et bien évidemment si elle sort du terrain (la balle peut tout de même passer au-dessus de la hauteur maximum de la fenêtre, elle retombera simplement un peu plus tard).

Quand la balle touche une tour ou un joueur, un son spécifique à l’objet touché est également joué. Et une fois que la balle a terminé sa course, c’est au tour du joueur 2.

Pour pouvoir gérer la collision de la balle sur un objet, des méthodes qui checkent en permanence si la balle touche la tour alliée, la tour ennemie ou encore le joueur ennemi ont été implémentées en utilisant notamment la hitBox des joueurs.

Si la balle touche un joueur, l’un de ses points de vie est retiré est si un des joueurs ne possède plus de point de vie, un son pour signifier qu’un joueur est mort est lancé et la boucle s’arrête.

#### Fin du jeu

Pour le message de fin de jeu, une petite musique est lancée avec un message qui félicite simplement le joueur encore en vie tandis que le perdant est dessiné, à terre.

### Quatrième semaine

Lors de cette semaine, j’ai surtout amélioré la classe Game avec une meilleure gestion des collisions avec notamment une nouvelle classe HitBox qui est implémentée dans les joueurs et les tours. La puissance est maintenant indépendante du temps maximum donné et sera toujours entre 0 et 50. La logique du jeu est complète avec notamment une fin si un des joueurs n’a plus de vie. J’ai également ajouté des sons pour respectivement le tir, quand une balle touche une tour et quand une balle touche un joueur et quand on maintient la touche espace, on charge la barre au lieu que la barre se charge tout seul avec le fait d’appuyer sur espace pour stocker la valeur.

## Modifications

Historique des modifications demandées (ou nécessaires) aux spécifications détaillées.

Date, raison, description, etc.

# Tests

## Dossier des tests

On dresse le bilan des tests effectués (qui, quand, avec quelles données…) sous forme de procédure. Lorsque cela est possible, fournir un tableau des tests effectués avec les résultats obtenus et les actions à entreprendre en conséquence (et une estimation de leur durée).

Si des tests prévus dans la stratégie n'ont pas pu être effectués :

raison, décisions, etc.

Liste des bugs répertoriés avec la date de découverte et leur état:

Corrigé, date de correction, corrigé par, etc.

# Conclusion

## Bilan des fonctionnalités demandées

Il s’agit de reprendre point par point les fonctionnalités décrites dans les spécifications de départ et de définir si elles sont atteintes ou pas, et pourquoi.

Si ce n’est pas le cas, estimer en « % » ou en « temps supplémentaire » le travail qu’il reste à accomplir pour terminer le tout.

## Bilan de la planification

Distinguer et expliquer les tâches qui ont généré des retards ou de l'avance dans la gestion du projet. Indiquer les différence entre les planifications initiales et détaillées avec le journal de travail.

## Bilan personnel

Si c’était à refaire:

Qu’est-ce qu’il faudrait garder ? Les plus et les moins ?

Qu’est-ce qu’il faudrait gérer, réaliser ou traiter différemment ?

Qu’est que ce projet m’a appris ?

Suite à donner, améliorations souhaitables, …

Remerciements, signature, etc.

# Divers

## Journal de travail

Date, activité (description qui permet de reproduire le cheminement du projet), durée, liens et références sur des documents externes. Lorsqu’une activité de recherches a été entreprise, il convient d’énumérer ce qui a été trouvé, avec les références.

## Bibliographie

Références des livres, revues et publications utilisés durant le projet.

## Webographie

Références des sites Internet consultés durant le projet.

# Annexes

Listing du code source (partiel ou, plus rarement complet)

Guide(s) d’utilisation et/ou guide de l’administrateur

Etat ou « dump » de la configuration des équipements (routeur, switch, robot, etc.).

Extraits de catalogue, documentation de fabricant, etc.