P\_POO 320

Meron Essayas

Classe – FID1

Lieu - Lausanne

Durée - 32 périodes/8 semaines

Nom du chef de projet – Patrick Chenaux

Table des matières

[1 dESCRIPTION DE L’APPLICATION 3](#_Toc192511419)

[1.1 Introduction 3](#_Toc192511420)

[2 Raisonnement 3](#_Toc192511421)

[2.1 Classes: 3](#_Toc192511422)

[3 PLAYER 3](#_Toc192511423)

[4 Ball 4](#_Toc192511424)

[5 Tower 5](#_Toc192511425)

# dESCRIPTION DE L’APPLICATION

## Introduction

Le projet consiste à développer un jeu comportant deux personnages, deux bâtiments, un système de score, des points de vie et une balle que les deux joueurs peuvent lancer.

Si la balle touche le bâtiment de l'adversaire, ce dernier perd un carré, réduisant ainsi sa forme rectangulaire.

Lorsqu'un joueur est touché par la balle, il perd un point de vie, tandis que le joueur ayant effectué le tir gagne un point de vie. Les joueurs ont la possibilité de régler l'angle et la puissance de leur tir.

Le jeu se termine lorsqu'un bâtiment est complètement détruit ou lorsqu'un joueur n'a plus de points de vie.

L'écran du jeu aura une résolution de 150x40.

Le jeu est programmé en C# sur Visual Studio 2022, avec la version .NET 4.7.2.

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*

O \*\*\*\*\* \*\*\*\*\* O

/|\ \*\*\*\*\* \*\*\*\*\* /|\

/ \ \*\*\*\*\* \*\*\*\*\* / \

----------------------------------------------------------------------------------------------------------------------------------

# Raisonnement

L’idée est de représenter chaque objet du jeu avec une classe.

## Classes:

* Player
* Ball
* Tower
* Game
* Program

# PLAYER

**3.1 Concept**

La classe Player a été conçue pour représenter les joueurs individuels du jeu. Chaque joueur a une position, une couleur et une représentation visuelle spécifiques sur l'écran. L'objectif de la classe Player est d'encapsuler les propriétés et les comportements du joueur liés au jeu, tels que sa position sur l'écran, la façon dont il s'affiche et la façon dont il interagit avec d'autres éléments comme la balle et la tour.

**3.2 Implémentation**

Pour commencer, je savais que chaque joueur avait besoin de plusieurs propriétés :

\_playerposition: la position du joueur sur l'écran devait être suivie afin que le joueur puisse être dessiné correctement pendant la boucle de jeu. La position serait représentée à l'aide d'une structure Vector2, qui contient à la fois les coordonnées X et Y.

PlayerColor: chaque joueur aurait une couleur spécifique pour les distinguer visuellement sur l'écran. C'est là que j'ai utilisé l'énumération ConsoleColor pour définir une couleur pour chaque joueur.

string []Person: je devais définir à quoi ressemblerait chaque joueur sur l'écran. Comme le jeu utilise la console, j'ai décidé de représenter le joueur avec un simple « bonhomme allumette » dessiné à l'aide de caractères ASCII.

Display (): j'ai implémenté une méthode appelée Display pour gérer le dessin du joueur à l'écran. Cette méthode utilise la position actuelle du joueur pour définir la position du curseur et afficher la forme du joueur. Elle garantit également que le joueur est dessiné en utilisant la couleur spécifiée pour la différenciation.

**Défis et solutions**

**Affichage du joueur** : l'un des défis était de dessiner le joueur sur l'écran de la console avec sa position exacte et dans la bonne couleur. Pour résoudre ce problème, j'ai utilisé **Console.SetCursorPosition** pour placer le joueur aux coordonnées correctes avant d'imprimer l'illustration ASCII.

# Ball

**4.1 Concept**

La classe Ball a été conçue pour représenter la balle dans le jeu, qui est un élément clé du gameplay. La balle doit pouvoir se déplacer, entrer en collision avec des objets (comme des tours et des joueurs) et être lancée avec une force et un angle réglable. La classe Ball suit également la position et la vitesse de la balle et gère la logique de ses interactions dans le monde du jeu.

**4.2 Implémentation**

L'idée de la classe Ball est venue du besoin d'avoir un objet qui se déplace sur l'écran, réagit aux entrées de l'utilisateur et interagit avec l'environnement du jeu (joueurs et tours). Voici les principaux composants que j'ai pris en compte lors de l'implémentation de la classe Ball :

\_ballposition **et** Velocity**:**

La position de la balle doit être suivie sur l'écran afin qu'elle puisse être dessinée au bon endroit.

La balle a également une vitesse, qui dictera la distance à laquelle elle se déplace à chaque image et dans quelle direction.

GRAVITY **et** TIME\_STEP **:**

La gravité et la friction sont introduites pour que la balle se comporte de manière plus réaliste. La gravité tire la balle vers le bas, tandis que la friction la ralentit progressivement au fil du temps.

LaunchBall(float angle, float force, Vector2 startPosition)**:**

La balle doit être lancée avec un angle et une force spécifique. L'angle est converti en radians à des fins de calcul et la force détermine la vitesse initiale de la balle.

**Détection de collision :**

La balle doit interagir avec les tours et les joueurs. Lorsque la balle touche une tour, des parties de la tour sont détruites. Si la balle touche un joueur, ce dernier perd un point de vie.

Display()**:**

La balle doit être attirée vers la console à sa position actuelle. La position de la balle sera mise à jour en continu en fonction de sa vitesse et des forces qui lui sont appliquées.

**Gestion de l'état :**

La balle peut être détruite lorsqu'elle touche certains objets, la classe doit donc savoir si la balle est détruite ou non.

**Défis et solutions**

**Physique du mouvement** : l'implémentation du comportement de la vitesse et de la gravité a nécessité une certaine réflexion, notamment sur la manière de simuler un mouvement réaliste dans les limites de la console. Les constantes de friction et de GRAVITÉ permettent un comportement plus fluide.

**Détection des collisions** : la détection des collisions entre la balle, les joueurs et les tours était essentielle au gameplay, et la structure du code permet de vérifier facilement la position de la balle par rapport aux autres objets du jeu.

**Affichage visuel** : la balle est représentée par un simple caractère (« ● ») qui s'affiche à la position appropriée. La balle est effacée de l'emplacement précédent à l'aide de LastBallPosition.

**TIME\_STEP** : Si le time\_step est plus grand que 0.1 alors il y aura des problèmes des suppression des bonnes cases.

# Tower

**5.1 Concept**

La classe Tower représente les tours du jeu. Chaque tour possède une position spécifique, des dimensions (hauteur et largeur) et une représentation à l'écran. Le but de la classe est d'encapsuler les propriétés et les comportements de la tour, notamment sa position, sa représentation graphique et ses interactions avec d'autres éléments du jeu tels que la balle.

**5.2 Implémentation**

Pour commencer, je savais que chaque joueur avait besoin de plusieurs propriétés :

\_**towerposition** : il s'agit du champ privé pour stocker la position de la tour. Il est représenté par un Vector2, qui stocke les coordonnées X et Y.

**TowerPosition** : cette propriété publique permet d'obtenir ou de définir la position de la tour.

**Height** et **Width** : ces propriétés représentent les dimensions de la tour.

WALL\_FORM : un caractère constant ('●') est utilisé pour représenter visuellement la tour à l'écran.

**CellState** : un tableau 2D de valeurs booléennes représente l'état de chaque cellule de la tour. true signifie que la cellule est intacte et false signifie qu'elle est détruite.

Le constructeur initialise la hauteur, la largeur et la position de la tour et définit l'état initial de toutes les cellules de la tour sur intact (true).

Affichage de la tour La méthode Display() gère le dessin de la tour sur la console. Il utilise **Console.SetCursorPosition** pour déplacer le curseur vers la position de départ de la tour, puis parcourt chaque cellule de la tour, en imprimant le caractère WALL\_FORM si la cellule est intacte ou un espace si elle est détruite.

**Défis et solutions**

**Dessin de la tour** : l'un des défis lors de l'affichage de la tour était de s'assurer que chaque partie de la tour était correctement placée sur l'écran. Ce problème a été résolu en utilisant **Console.SetCursorPosition** pour positionner le curseur et imprimer les caractères correspondants.

**Détection de collision** : le défi consistait à vérifier si la balle touchait une partie de la tour. Ce problème a été résolu en vérifiant chaque cellule de la tour individuellement pour déterminer si elle était intacte et si la position de la balle se chevauchait.