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](#_Toc192582463)

[1.1 Introduction 3](#_Toc192582464)

[2 Raisonnement 3](#_Toc192582465)

[2.1 Classes 3](#_Toc192582466)

[3 PLAYER 4](#_Toc192582467)

[3.1 Concept 4](#_Toc192582468)

[3.2 Implémentation 4](#_Toc192582469)

[3.3 Défis et solutions 4](#_Toc192582470)

[4 Ball 4](#_Toc192582471)

[4.1 Concept 4](#_Toc192582472)

[4.2 Implémentation 4](#_Toc192582473)

[4.3 Défis et solutions 5](#_Toc192582474)

[5 Tower 5](#_Toc192582475)

[5.1 Concept 5](#_Toc192582476)

[5.2 Implémentation 5](#_Toc192582477)

[5.3 Défis et solutions 6](#_Toc192582478)

[6 Game 6](#_Toc192582479)

[7 class diagram 7](#_Toc192582480)

[8 Conclusion 8](#_Toc192582481)

[8.1 Conclusion technique 8](#_Toc192582482)

[8.2 Conclusion personnelle 8](#_Toc192582483)

[9 Table des illusturations 8](#_Toc192582484)

# 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.

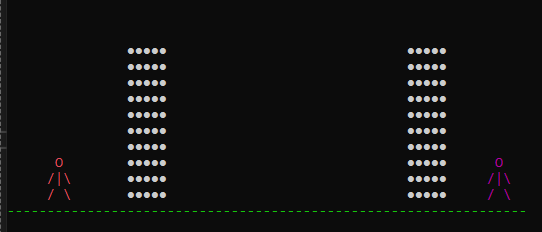


Figure 1 : le jeu dans la console

# Raisonnement

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

## Classes

* Player
* Ball
* Tower
* Game
* Program

# PLAYER

## 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.

## 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** : c’est optionnel, 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

## 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.

## Implémentation

Pour commencer, je savais que la balle avait besoin de plusieurs propriétés

**\_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 le time\_step 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.

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

**IsDestroyed** : l’attribut IsDestroyed est utilisée pour suivre si la balle a été détruite en raison d'une collision avec la tour.

**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.

**UpdateBallPosition()** : Cette méthode est chargée de mettre à jour la position de la balle en fonction de sa vitesse actuelle, d'appliquer la gravité et de prendre en compte un TIME\_STEP pour contrôler la vitesse des mises à jour.

**PressSpaceBar(Player currentPlayer, Tower Tower)** : Cette méthode gère la logique de détection du moment où la barre d'espace est enfoncée, du lancement de la balle et de la vérification des collisions avec la tour. Elle exécute une boucle de jeu où la balle est mise à jour et affichée jusqu'à ce qu'elle soit détruite.

**Destroy ()** : Cette méthode est appelée lorsque la balle doit être détruite. Elle marque la balle comme détruite et efface sa position sur l'écran.

**Display () :** met à jour la position de la balle sur l'écran et s'assure que sa position précédente est effacée avant d'afficher la balle à la nouvelle position.

## 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

## 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.

## Implémentation

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

\_**towerposition** : il s'agit de l’attribut 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 attributs 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.

**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.

**DestroyBlock ()** : La méthode vérifie si les coordonnées fournies (x, y) sont dans les limites de la grille, et si c'est le cas, elle définit l'état de la cellule correspondante sur faux pour indiquer que le bloc a été détruit.

## 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 ou pas.

# Game

La classe Game servira à encapsuler l'ensemble du jeu, permettant ainsi de coordonner toutes ses composantes. Elle permettra la création des objets du jeu, qui seront accessibles

dans le programme principal.

# class diagram

Les structures des classes.

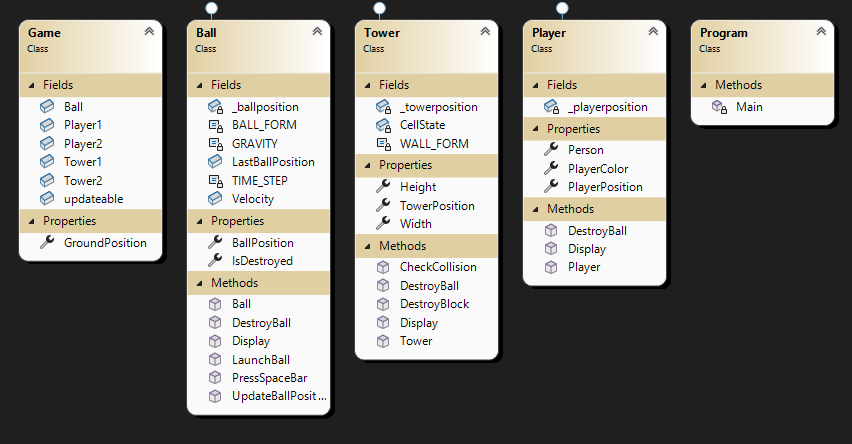


Figure 2 : class diagramme des 5 classes

Les structure des interfaces.

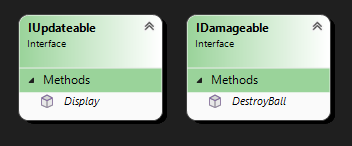


Figure 3 : classe diagramme des 2 interfaces

# Conclusion

## Conclusion technique

Ce projet m'a permis d'appliquer les concepts fondamentaux de la programmation orientée objet (OOP). En développant ce jeu, j'ai eu l'occasion de concevoir et d'implémenter différentes classes et objets, chacun exerçant des responsabilités spécifiques et interagissant les uns avec les autres. Les mécanismes de jeu, tels que l'impression des joueurs dans une position spécifique, les tours de différentes hauteurs et largeurs et leur impression dans différentes positions et l'appui sur la barre d'espace pour lancer la balle ont été implémentés à l'aide de méthodes et d'attributs définis dans des classes spécifiques, ce qui a renforcé ma compréhension des principes de la programmation orientée objet tels que l'encapsulation, l'héritage et le polymorphisme.

En bref, ce projet a été une excellente occasion de mettre en pratique les fondamentaux de la programmation orientée objet.

## Conclusion personnelle

Dans ce projet, j'ai implémenté avec succès des fonctionnalités clés telles que l'affichage des deux joueurs et des tours dans des positions spécifiques et le lancement de la balle lorsque la barre d'espace est enfoncée. De plus, j'ai créé un mécanisme où, si la balle touche la tour du deuxième joueur, le bloc spécifique de la tour est détruit et la balle est retirée du jeu. Cependant, plusieurs fonctionnalités avancées, telles que le système de points, la détermination de l'angle et de la vitesse de la balle et la gestion de la perte ou du gain de points, n'ont pas été finalisées. Ces fonctionnalités se sont avérées plus difficiles que prévu pour moi, et bien que nous n'ayons eu que huit semaines pour terminer le projet, je suis déterminé à continuer à y travailler. A titre personnel, je vais travailler encore sur ce projet pour implémenter ces fonctionnalités restantes et terminer le jeu comme prévu.

# Table des illusturations

[Figure 1 : le jeu dans la console 3](file:///C:\Users\pk65rtq\Documents\GitHub\P_POO%20320\P_POO%20320.docx#_Toc192577504)

[Figure 2 : class diagramme des 5 classes 6](#_Toc192577505)

[Figure 3 : classe diagramme des 2 interfaces 6](#_Toc192577506)