I320

Jeu de balle

Christopher Ristic

Ristic Christopher – FID1

ETML / 1004 LAUSANNERue de Sébeillon 12 1004 Lausanne

Table des matières

[1 Introduction 3](#_Toc189831505)

[1.1 Objectif 3](#_Toc189831506)

[2 Début du raisonnement POO 3](#_Toc189831507)

[2.1 Classes 3](#_Toc189831508)

[2.2 Interfaces 3](#_Toc189831509)

[2.3 Pattern matching 3](#_Toc189831510)

[2.4 Polymorphisme 4](#_Toc189831511)

[3 Game 4](#_Toc189831512)

[4 Game Manager 4](#_Toc189831513)

[5 player 4](#_Toc189831514)

[6 Building 4](#_Toc189831515)

[6.1 Affichage 4](#_Toc189831516)

[6.2 Collision et dégâts 5](#_Toc189831517)

[7 Ball 5](#_Toc189831518)

[7.1 Affichage 5](#_Toc189831519)

[7.2 Simulation 5](#_Toc189831520)

# 

# Introduction

## Objectif

Le projet consiste à créer un jeu avec 2 personnages, 2 bâtiments, un score, des points de vies, et une balle qui est lancée par les 2 joueurs.

Si la balle touche le bâtiment adverse, le bâtiment perds un carré et commence à perdre sa forme rectangulaire.

Si la balle touche le joueur il perd un point de vie, et le joueur qui a tiré gagne un point de vie. Les joueurs peuvent tirer la balle en choisissant un angle et la puissance du tir.

Le jeu se termine lorsqu’un bâtiment est complètement détruit ou qu’un joueur n’a plus de point de vie.

La taille de l’écran de la console sera de 150x40.

Le jeu est codé en langage C# sur Visual Studio 2022, .NET version 4.7.2.

# Début du raisonnement POO

L’idée est de représenter chaque objet du jeu avec une classe. Puis des interfaces pour définir des comportements communs aux classes qui en héritent.

## Classes

J’utilise les classes pour représenter les points suivants :

* Joueur
* Bâtiment
* Balle
* Manager du jeu
* L’ensemble du jeu

## Interfaces

Les interfaces vont me permettre de poser des prototypes de méthode. J’utilise les interfaces pour différents comportements commun des objets :

|  |  |
| --- | --- |
| Fonction | Nom du fichier |
| Entrer en collision | ICollidable.cs |
| Être dommageable | IDamageable.cs |
| Être mis à jour | IUpdatable.cs |

## Pattern matching

Depuis C# 7.0, le mot-clé "is**"** permet non seulement de vérifier le type, mais également d'extraire la valeur convertie dans une variable si la vérification est réussie. Si la condition est vraie, la valeur contenue dans **damageData** est convertie en entier et assignée à la variable **damageAmount**.

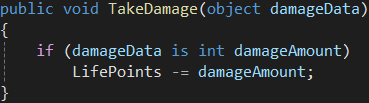


Figure : Pattern matching

On trouve ce mot-clé dans certaines conditions que j'ai appliqué notamment un exemple qui se trouve dans la méthode **TakeDamage** voir Figure 1: Pattern matching.

## Polymorphisme

Comme il y a des classes différentes et que je passe par des interfaces pour définir des comportements communs, je vais utiliser des **listes** de la **bibliothèque** **System.Collection.Generic** et ces listes ont un type polymorphe grâce à l’héritage des classes sur les interfaces.

# Game

La classe game va me permettre de contenir l’ensemble du jeu, c’est-à-dire qu’on va pouvoir coordonner le jeu dans son ensemble. On va pouvoir y créer les objets et ils seront atteignable dans le programme principal. Cette classe va aussi gérer le début et fin du jeu, les tours et la simulation de la balle.

# Game Manager

De manière générale un game manager est utilisé pour gérer la logique du jeu et son état. On peut y retrouver des contrôles de niveaux, la gestion de score, sauvegarde et autres événements.

Pour le projet, on va l’utiliser pour gérer :

* La collision entre la balle et un bâtiment
* Le score
* Le système d’angle de tir

# player

Le joueur sera représenté avec la forme suivante :

Il possède plusieurs paramètres tels que : point de vie, score, une position, la couleur de sa balle, et la forme qui le représente.

Si la balle le touche il perd des points de vie et le joueur adverse en gagne.

# Building

## Affichage

Un bâtiment est affiché et composé de plusieurs et même caractère comme celui-ci : ■

Et sa forme complète :

## Collision et dégâts

La classe hérite des 3 Interfaces car on sait que le bâtiment doit pouvoir être touché et détruit par la balle, et son affichage doit être mis à jour après avoir pris des dégâts.

Pour déterminer quelle case a touché la balle, on a une variable de type vecteur en 2D, qui sera la position de la case haut gauche du bâtiment. Un tableau de Boolean représente si la case est intacte ou non. Ainsi on peut calculer la bonne case du bâtiment touché en faisant des décalages.

# Ball

## Affichage

La balle s’affiche sous cette forme-ci :

La méthode qui permet la mise à jour de la position de la balle est accédé depuis une liste d’objet « updatable » que contient Game.cs. L’affichage ce fait avec la méthode Display.

## Simulation

L’équation est basée sur une forme simplifiée de la méthode Euler : Mouvement rectiligne uniformément accéléré (MRUA).

Pour calculer la position de la balle on utilise la formule suivante :

* Position = Position actuelle + Vitesse \* Temps

Concernant la vitesse de la balle on trouve :

* Vitesse verticale = Vitesse actuelle + Gravite \* Temps

La gravité est définie par une constante à 0,5.