Le jeu de la balle



Matias Denis – FID1

ETML - Vennes

32 période

Professeur Mr. Patrick Chenaux

Table des matières

Table des matières

[1. Introduction 3](#_Toc192582190)

[2. Description générale et Objectifs du jeu : 3](#_Toc192582191)

[3. Contraintes et fonctionnalités spécifiques : 3](#_Toc192582192)

[4. RAISONNEMENT 3](#_Toc192582193)

[4.1 Classes 3](#_Toc192582194)

[4.3 Utilisation de structures conditionnelles et vérifications 6](#_Toc192582195)

[4.4 Polymorphisme simple via instanciation 6](#_Toc192582196)

[5. INTERFACE GRAPHIQUE 7](#_Toc192582197)

[6. DIAGRAMME DE CLASSES 8](#_Toc192582198)

[7. AMÉLIORATIONS, CORRECTIONS ET NOUVEAUTÉS (IA) 9](#_Toc192582199)

[a) Correction des collisions murales précises : 9](#_Toc192582200)

[b) Gestion des collisions avec son propre mur : 9](#_Toc192582201)

[c) Correction des positions de départ : 9](#_Toc192582202)

[d) Ajout de sons différenciés : 9](#_Toc192582203)

[8. RÉCAPITULATIF DES MÉTHODES CORRIGÉES OU AJOUTÉES GRÂCE À L'IA 10](#_Toc192582204)

[Classe Ball : 10](#_Toc192582205)

[Classe Game : 10](#_Toc192582206)

[Classe SoundManager : 10](#_Toc192582207)

[9. COMPARAISON ENTRE LES CLASSES ET MÉTHODES PRÉVUES INITIALEMENT ET CELLES UTILISÉES 10](#_Toc192582208)

[Les classes prévues au départ : 10](#_Toc192582209)

[Classes effectivement utilisées : 11](#_Toc192582210)

[Méthodes initialement prévues mais modifiées ou supprimées : 11](#_Toc192582211)

[10. CONCLUSION ET APPORT DE L'IA 11](#_Toc192582212)

#### 1. Introduction

Ce projet consiste à développer un jeu de balle interactif en C#, où deux joueurs s'affrontent à tour de rôle en tentant de détruire les murs adverses ou d'atteindre directement leur adversaire à l'aide d'une balle lancée selon un angle et une puissance définis. Le jeu se déroule dans la console, avec une interface ASCII, un système de gestion des scores et des vies, et une intégration complète de sons pour rendre l'expérience plus immersive.  
  
Le projet a été enrichi et corrigé grâce à l'aide de l'IA pour résoudre plusieurs problèmes techniques, notamment la gestion des collisions et l'ajout de sons contextuels.

#### 2. Description générale et Objectifs du jeu :

- Deux joueurs sont séparés par des murs constitués chacun de 6 cases.  
- À chaque tour, un joueur sélectionne un angle de tir via un système de points clignotants, puis une puissance via une barre de progression.  
- La balle suit une trajectoire réaliste, influencée par la gravité.  
- Objectifs pour marquer :  
 \* Toucher le mur adverse : supprimer une case et gagner un point.  
 \* Toucher l'adversaire : enlever une vie à l'adversaire et gagner un point.  
- Attention : si la balle touche son propre mur, le joueur perd une vie et l'adversaire gagne un point.  
- Le premier joueur à atteindre 7 points ou faire perdre toutes les vies de l'adversaire gagne.

#### 3. Contraintes et fonctionnalités spécifiques :

- Interface console avec affichage des scores, vies, murs, et joueurs.  
- Sélection d'angle et de puissance via interactions clavier.  
- Trajectoire réaliste influencée par la gravité.  
- Gestion précise des collisions avec mur, propre mur, et adversaire.  
- Sons distincts pour tir, impact mur, impact joueur.  
- Fin de jeu avec annonce du gagnant.

#### 4. RAISONNEMENT

L'idée principale de ce projet a été d'utiliser la Programmation Orientée Objet (POO) afin de structurer de manière claire et efficace tous les éléments du jeu. Chaque composant du jeu (joueurs, murs, balle, sons, gestion de la partie) a été représenté par une classe spécifique. Cette approche permet une meilleure organisation du code, une maintenance facilitée et une meilleure compréhension globale du programme.

### 4.1 Classes

J’ai utilisé les classes suivantes pour représenter les différents objets du jeu :

* **Game** : Gère le déroulement complet du jeu, les tours de chaque joueur, l'affichage et la fin de la partie. Contient toutes les méthodes nécessaires pour :
  + Afficher l'interface de jeu avec les scores, les vies, les murs et les joueurs (méthode DisplayInterface).
  + Sélectionner l'angle et la puissance de tir via une barre et des points clignotants (méthodes ChooseAngle et ChoosePower).
  + Exécuter le tour complet d'un joueur, en appelant la trajectoire de la balle et en gérant les collisions (méthode PlayTurn).
  + Changer de joueur après chaque tour (méthode SwitchPlayer).
  + Déterminer la fin de la partie et afficher le gagnant (méthode EndGame).
  + Gérer le dessin visuel des joueurs et murs (méthodes DrawPlayer et DrawWall).

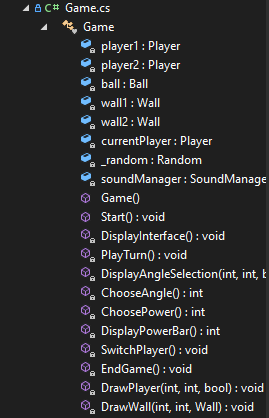


Figure 1, structure classe Game

* **Player** : Contient les informations propres aux joueurs (nom, score, vies). Cette classe permet de créer les deux joueurs du jeu avec un nom défini, un nombre de vies initialisées à 5, et un score qui commence à 0. Elle permet aussi de mettre à jour ces informations tout au long de la partie selon les actions (perte de vie, gain de point).

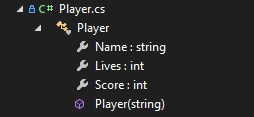


Figure 2, structure classe Player

* **Ball** : Gère la trajectoire, les collisions, et les effets de la balle avec les murs, les joueurs, et son propre mur. Cette classe contient notamment la méthode CalculateTrajectory qui permet de simuler le mouvement de la balle en prenant en compte la gravité, et les méthodes CheckWallCollision et CheckOwnWallCollision pour détecter et gérer les impacts avec les murs, les joueurs, et les conséquences sur les scores et les vies.

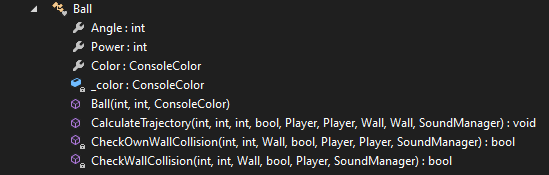


Figure 3, structure classe Ball

* **Wall** : Représente un mur constitué de cellules. Chaque mur est un tableau de 6 lignes et 3 colonnes, représentant les cases du mur. La méthode Hit permet de gérer l'impact de la balle sur une cellule : lorsqu'elle est touchée, la cellule devient invisible (disparaît du mur), ce qui permet de simuler la destruction partielle du mur.

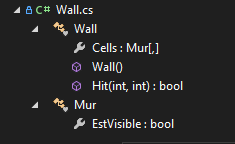


Figure 4, structure classe Wall

* **SoundManager** : Gère tous les sons du jeu (tir, impact mur, impact joueur). Elle permet de jouer différents sons en fonction des actions réalisées (lancement de balle, impact sur un mur, impact sur un joueur) pour rendre le jeu plus immersif.

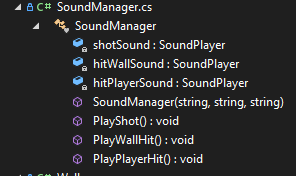


Figure 5, structure classe SoundManager

4.2 Organisation des comportements communs

Pour gérer certains comportements spécifiques, j’ai utilisé des méthodes directement intégrées aux classes (ex. : collisions). Il n’a pas été nécessaire d'utiliser des interfaces, car les interactions sont gérées par des méthodes précises selon les objets. Ainsi, les collisions sont directement prises en charge dans la classe Ball, avec des méthodes dédiées telles que CheckWallCollision et CheckOwnWallCollision.

### 4.3 Utilisation de structures conditionnelles et vérifications

Des vérifications précises ont été implémentées pour gérer les collisions entre la balle et les murs ou les joueurs, notamment grâce à des calculs de positions (X, Y) adaptés à la console. Les coordonnées sont ajustées et contrôlées pour s’assurer que la balle ne sorte pas des limites du terrain. La gestion des collisions se fait notamment grâce à des méthodes conditionnelles, sans utiliser le pattern matching, car les types sont connus et fixes.

### 4.4 Polymorphisme simple via instanciation

Le polymorphisme a été utilisé de manière simple : chaque objet (joueur, mur, balle, son) est une instance d'une classe spécifique, et ces instances interagissent via leurs méthodes publiques. Par exemple, la classe Game crée des objets Ball et Player, et utilise leurs méthodes pour exécuter les actions du jeu. Bien que je n’aie pas utilisé d’héritage complexe ou d’interfaces polymorphes, la structure du code permet tout de même une bonne séparation des responsabilités.

#### 5. INTERFACE GRAPHIQUE

Le jeu se déroule entièrement dans une interface graphique basée sur la console Windows, utilisant des caractères ASCII pour représenter les différents éléments du jeu. L'affichage est composé de plusieurs parties bien distinctes : en haut de l'écran, les scores et vies de chaque joueur sont affichés, permettant de suivre en temps réel l'évolution de la partie. Au centre, on retrouve les deux joueurs positionnés chacun à une extrémité du terrain, représentés par des bonhommes stylisés en ASCII. Entre les deux joueurs, les murs sont composés de blocs visibles, formant une ligne de défense que l'adversaire doit détruire pour marquer des points. La balle, quant à elle, est représentée par un petit caractère ('o') qui se déplace de façon réaliste à travers l'écran selon l'angle et la puissance choisis, créant une trajectoire influencée par la gravité. En bas de l'écran, une ligne continue ("---------") symbolise le sol du terrain.

L'interface permet également des interactions dynamiques : la sélection de l'angle de tir est réalisée à l'aide de points clignotants (O et \*), tandis que la puissance est choisie via une barre de progression horizontale qui augmente rapidement. Ces éléments interactifs offrent une expérience visuelle complète et immersive malgré la simplicité d'une console texte.

Pour illustrer cette interface, des captures d'écran du jeu en cours de partie, montrant la position des joueurs, des murs, et la trajectoire de la balle, seront insérées ci-dessous afin de mieux visualiser l'agencement graphique du jeu.



Figure 6, scores et vies des joueurs



Figure 7, joueur 1 en bleu

Figure 8, joueur 2 en rouge



Figure 10, balle en caractère 'o'

Figure 9, mur en ASCII █



Figure 11, ligne verte pour le sol

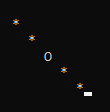


Figure 12, sélection d'angle clignotante



Figure 13, barre pour la puissance

#### 6. DIAGRAMME DE CLASSES

Figure 14, structure des classes

#### 7. AMÉLIORATIONS, CORRECTIONS ET NOUVEAUTÉS (IA)

Le projet a fortement évolué grâce à l’aide continue de l’IA, qui m’a accompagné dans la conception, la correction et la réorganisation du code. De nombreuses fonctionnalités prévues au départ ont été simplifiées, améliorées, ou corrigées pour fonctionner correctement. Voici les principales améliorations :

### a) Correction des collisions murales précises :

* **Recalcul précis des coordonnées des murs :** pour que la balle puisse correctement toucher les bonnes cases des murs.
* **Correction des formules de collision :** réécriture des calculs de trajectoire et d'impact avec l'IA.
* **Ajout de vérifications pour éviter les erreurs de débordement de tableau.**

**Attention : malgré ces corrections, les collisions entre la balle et les murs ne sont pas toujours détectées correctement. Il arrive que la balle traverse le mur sans supprimer les cases touchées.**

### b) Gestion des collisions avec son propre mur :

* **Création d'une méthode spécifique CheckOwnWallCollision()** grâce à l’IA.
* **Ajout d'une pénalité** : si un joueur touche son propre mur, il perd une vie et l’adversaire gagne un point. Cette fonctionnalité n’était pas prévue initialement et a été suggérée par l’IA pour équilibrer le gameplay.

### c) Correction des positions de départ :

* **Recalage précis des positions des joueurs et murs** pour éviter que le tir touche directement son propre mur.
* **Recalcul dynamique de la sélection d’angle** selon la position du joueur, corrigé avec l’aide de l’IA.

### d) Ajout de sons différenciés :

* L’IA a permis d'intégrer une gestion claire et fonctionnelle des sons via la classe SoundManager, avec trois sons distincts :
  + **Tir** : laser-shot-ingame-230500.wav.
  + **Impact mur** : hit-by-a-wood-230542.wav.
  + **Impact joueur** : metal-hit-15-193280.wav.

Ces sons renforcent l’immersion du jeu et permettent de signaler les actions importantes.

#### 8. RÉCAPITULATIF DES MÉTHODES CORRIGÉES OU AJOUTÉES GRÂCE À L'IA

Voici la liste des principales méthodes corrigées, repensées ou créées grâce à l’aide de l’IA :

### Classe Ball :

* **CalculateTrajectory()** : entièrement réécrite pour intégrer la gravité, les collisions, et l'affichage dynamique.
* **CheckWallCollision()** : corrigée pour prendre en compte les coordonnées exactes. **Cependant, cette méthode nécessite encore des ajustements pour une fiabilité complète.**
* **CheckOwnWallCollision()** : ajoutée pour gérer les tirs sur son propre mur.

### Classe Game :

* **PlayTurn()** : corrigée pour bien gérer le déroulement d’un tour, la création de la balle, les sons et les collisions.
* **ChooseAngle()** : corrigée et améliorée pour afficher une sélection d’angle fluide et précise.

### Classe SoundManager :

* **PlayShot()**, **PlayWallHit()**, **PlayPlayerHit()** : ajoutées pour gérer les sons spécifiques liés aux différentes actions du jeu.

**Remarque** : la classe UserInterface, initialement prévue pour gérer l’affichage des scores et de la barre de puissance, a été intégrée directement dans la classe Game pour simplifier la gestion globale du jeu. Cette simplification a été proposée par l’IA pour éviter la multiplication inutile des classes.

#### 9. COMPARAISON ENTRE LES CLASSES ET MÉTHODES PRÉVUES INITIALEMENT ET CELLES UTILISÉES

### Les classes prévues au départ :

* **Game** : Gérer la logique globale (début, fin, tours).
* **Player** : Gérer vies, score, position.
* **Ball** : Gérer trajectoire et collisions.
* **Wall** : Gérer les murs et impacts.
* **SoundManager** : Gérer les sons.
* **UserInterface** : Gérer l'affichage (scores, barre de puissance, angles).

### Classes effectivement utilisées :

* **Game** : Regroupe à la fois la logique de jeu et les éléments d'affichage (interface).
* **Player** : Gestion des vies et scores (conforme au plan initial).
* **Ball** : Gestion de la trajectoire et des collisions (intégrant gravité, murs, joueurs, propre mur).
* **Wall** : Gestion des murs sous forme de grille (avec cellules individuelles "Mur").
* **Mur** : Nouvelle classe pour gérer chaque case du mur (visible ou non).
* **SoundManager** : Simplifié avec seulement trois sons clés (tir, impact mur, impact joueur).

### Méthodes initialement prévues mais modifiées ou supprimées :

* **StartGame()**, **DisplayStatuts()**, **IsGameOver()** → Fusionnées dans **Start()**.
* **SetInitialPosition()**, **Move()** et **(Ball)** → Fusionnées dans **CalculateTrajectory()**.
* **RemoveBlock()**, **(Wall)** → Remplacée par **Hit()**.
* Plusieurs méthodes de la classe UserInterface (**DisplayScores**, **DisplayProgressBar**, **DisplayBlinkingPoints)** → Intégrées directement dans Game.

Ainsi, **l’IA a permis de simplifier la structure en regroupant certaines fonctionnalités** tout en gardant un jeu fonctionnel et lisible.

#### 10. CONCLUSION ET APPORT DE L'IA

L'IA m’a beaucoup aidé pour la réalisation de ce projet. Grâce à ses conseils et corrections, j’ai pu :

* Structurer le projet selon les principes de la POO.
* Corriger des erreurs complexes dans la gestion des trajectoires et collisions.
* Ajouter des fonctionnalités nouvelles (comme la pénalité pour tir sur son propre mur, les sons d’impact).
* Simplifier et rendre plus fluide la gestion du jeu (réduction des classes inutiles, simplification des méthodes).

Globalement, l’IA m’a permis de passer d’une version basique et imparfaite du jeu à un projet complet, fonctionnel, avec une interface et des sons, offrant une expérience de jeu immersive et structurée. Ce projet m’a appris à organiser un code complexe et à résoudre des problèmes avancés.

Je remercie **Monsieur Patrick Chenaux** pour ce projet qui m’a permis d'apprendre la programmation orientée objet de manière concrète et ludique. Grâce à ce travail, j’ai beaucoup progressé, même si certaines fonctionnalités restent à améliorer pour une version parfaite du jeu (comme les collisions balle-mur).