Rapport : Shoot them up

Albert Braimi

Introduction

Le projet "Shoot them up" est un jeu vidéo de type arcade qui s'inscrit dans un genre très proche au jeu "space invaders".
Le concept est simple : le joueur contrôle un vaisseau spatial et doit survivre à des vagues successives d'ennemis tout en cherchant à obtenir le meilleur score possible.
Ce jeu est développé en C#, en utilisant les principes de POO pour garantir une structure de code claire et évolutive.

Objectifs du Produit

L'objectif principal est de concevoir et réaliser un jeu 2D modulaire de tir. Le jeu doit être fonctionnel, rejouable grâce à une génération procédurale d'ennemis, et inclure des mécaniques de jeu complètes telles que les tirs, les collisions, la vie, un score, et des obstacles destructibles.

Objectifs Pédagogiques

Ce projet vise à mettre en pratique les modules de programmation de base et les concepts avancés d'ICT-320. Les objectifs incluent :

- Maîtriser la programmation orientée objet (POO) en C#.
- Réaliser un programme informatique de qualité.
- Utiliser un système de versioning de code (Git) de manière professionnelle.

Planification

La planification initiale du projet a été divisée par fonctionnalités clés sur 8 semaines, conformément au temps alloué de 32 périodes.

Semaines	User stories	Remarque
Semaine 1		
Semaine 2		
Semaine 3	Mouvements	
	Mouvement des ennemis	
Semaine 4	Tir du joueur	
	Interractions	

Semaine 5	Vie
Semaine 6	Obstacles
Semaine 7	Score
Semaine 8	Explosion du boss

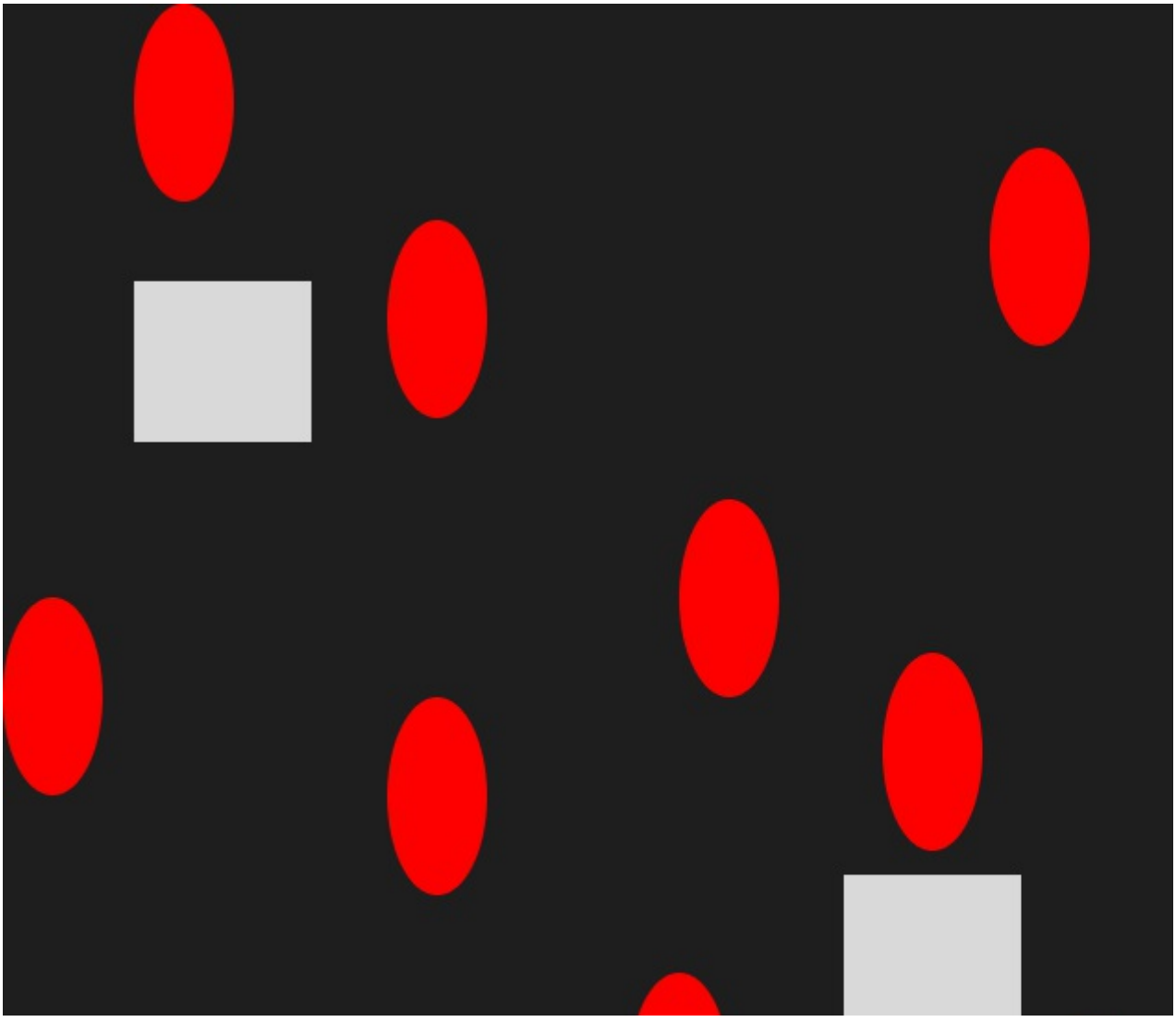
Analyse Fonctionnelle

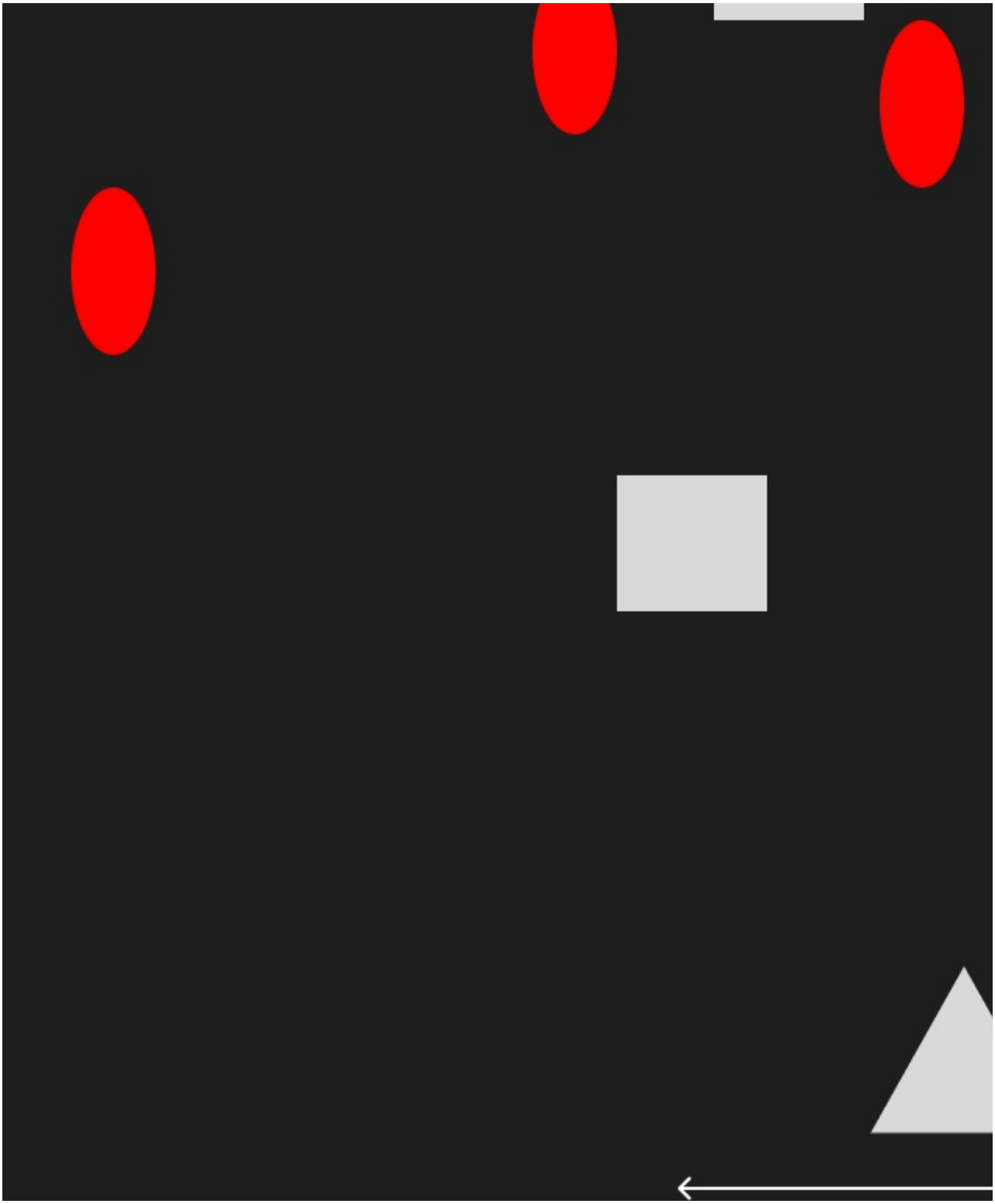
Le but est de créer un jeu vidéo 2D de type "Shoot'em up" (comme Space Invaders) où le joueur affronte des vagues d'ennemis générées procéduralement. Le joueur doit se déplacer, tirer, et éviter les collisions pour survivre et obtenir le meilleur score.

Joueur

En tant que joueur j'aimerai pouvoir contrôler mon vaisseau de droite a gauche pour eviter les projectiles/ennemis

- [x] Bouger latéralement avec `a` et `d`.
- [x] Dès que le joueur passe une des bordures il est téléporté de l'autre coté du plateau de jeu.





Mouvement des ennemis

En tant que joueur je veux que les ennemis bougent de haut en bas.

- [x] Les ennemis bougent de haut en bas.
- [x] Les projectiles ne peuvent que bouger vers le bas.
- [x] La vitesse de mouvement est aléatoire.
- [x] Si le joueur est en colision directe avec un ennemi, le joueur va perdre un point vie 2 fois par

seconde.

Tir du joueur

En tant que joueur je veux pouvoir tirer.

- [x] Les tirs partent en ligne droite depuis le joueur.
- [x] Les tirs de projectiles ne suivent pas les mouvement latéreaux du joueur.

Score

En tant que joueur je veux connaitre mon score à tout instant.

- [x] Le meilleur score est retenu et mit à jour.
- [x] Un affichage de niveau actuel est présent.
- [x] Certain ennemis donnent plus de score que d'autres.
- [x] Il y a un compteur de score.
- [x] Il y a un compteur de kills.

Vie

En tant que joueur je veux connaitre ma vie en tout temps.

- [x] Je veux pouvoir voir ma vie réstante soit de manière graphique
- [x] Je veux voir la vie restante des ennemis de la même manière que je vois la mienne.
- [x] Quand une entité reçoit un prjoectile elle perd un point de vie.

Obstacles

En tant que joueur je veux des obstacles.

- [x] Les obstacles sont statiques.
- [x] Ils auront un nombre de point de vie fixe défini automatiquement au debut de chaque round.
- [x] Ils ne peuvent pas etre franchis par le joueur ou les ennemis
- [x] Quand on leur tire dessus ils perdent des points de vie

---

Conception

L'architecture est basée sur la POO. La classe `Form` est le contrôleur central qui gère les listes et la boucle de jeu. Les autres classes (`Player`, `Enemy`, etc.) sont des modèles qui gèrent leur propre état.

Form

Le cœur du jeu. Cette classe gère la boucle `Update`/`Render`, possède toutes les listes d'objets (joueurs, ennemis, projectiles), contrôle l'état du jeu (score, vagues), et gère les collisions de haut niveau.

Config

Classe `static` qui contient les constantes globales (`WIDTH`, `HEIGHT`) et le générateur aléatoire (`alea`) basé sur la `GameSeed`. Elle garantit que la génération procédurale est reproductible.

Player

Représente le joueur. Gère son propre état (vie, position, direction) et ses actions (`Update` pour le mouvement, `TryShoot` pour le tir, `ApplyContactDamage` pour les dégâts de colision).

Enemy

Représente un ennemi. Son constructeur est autonome : il utilise `Config.alea` pour définir sa propre vie, position et vitesse. Il gère son propre mouvement de descente et son tir.

Obstacle

Un objet de décor passif. Il a une position et une vie, mais pas de mouvement. Il est "infranchissable" (vérifié par `Player` et
`Enemy`) et "destructible" (vérifié par `Form`).

Projectile

Un objet simple représentant un tir. Il gère son propre mouvement (`Update`) et expose ses propriétés de collision pour que la classe `Form` puisse gérer les impacts.

ProjectileType

Une `enumeration` qui distingue les tirs du `Player` de ceux de l'`Enemy`, ce qui est crucial for la logique de collision.

Utilisation de l'IA

L'IA a principalement servi d'outil de débogage et d'aide à la compréhension. Elle a été particulièrement utile pour analyser des erreurs C# spécifiques et pour clarifier des concepts de programmation complexes. Concernant l'algorithmique, l'IA a été utilisée en dernier recours : lorsque je faisais face à un problème complexe, je lui demandais d'expliquer la logique ou de la structurer sous forme de pseudo-

code. Ce pseudo-code a ensuite servi de fondation pour ma propre implémentation, me permettant de comprendre l'architecture de la solution avant de l'écrire. L'IA m'a également servi pour la rédaction du rapport, notamment pour formuler des phrases claires et compréhensibles, comme celle-ci.