ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

# Rapport Projet Génie logiciel 2A-Informatique 2024 SolidArc

**Membres de l'équipe :** Amode-Mall Samir, Kervran Valentin, FOUAD Rania, Lepoultier Benoit, Boccara Benjamin, Pakzad Yam, Zygmuntowicz Patrick, Yagapin Achille, El-Grably Jeremy

# TABLE DES MATIÈRES


**1 INTRODUCTION** **4**

**2 Présentation du projet** **5**
2.1 Rappel du sujet . . . . . 5
2.2 Contraintes technologiques . . . . . 5
2.3 Contraintes Fonctionnelles . . . . . 5

**3 ORGANISATION DU PROJET** **6**
3.1 Rôle Des Membres De L'équipe . . . . . 6
3.2 Structure Du Projet . . . . . 6

**4 ANALYSE DES RISQUES** **8**
4.1 Identification et évaluation des risques . . . . . 8
4.2 Mesures de prévention et plans d'action . . . . . 8

**5 ARCHITECTURE** **9**
5.1 Cas d'utilisation : . . . . . 9
5.2 Classes Du Domaine : . . . . . 9
Relations entre les classes . . . . . 10
5.3 MVP (Model, View, Presenter) : . . . . . 10
5.3.1 Modèle . . . . . 11
5.3.2 Présentateur . . . . . 11
5.3.3 Vue . . . . . 11
5.4 Conception En UML : . . . . . 12
5.4.1 Patrons De Conception Utilisés : . . . . . 12
- Composite : . . . . . 12
- Décorateur : . . . . . 12
- Stratégie : . . . . . 12
- Agrégation : . . . . . 12
5.4.2 Diagramme de paquet : . . . . . 12
5.5 Choix d'interface . . . . . 14

**6 IMPLÉMENTATION** **16**
6.1 MODEL : . . . . . 16
6.1.1 Player : . . . . . 16
6.1.2 Target : . . . . . 16
6.1.3 Classic : . . . . . 16
6.1.4 Baloon : . . . . . 16
6.1.5 Bow : . . . . . 16

6.1.6 Arrow : . . . 16
6.1.7 Wind : . . . 16
6.1.8 Score : . . . 17
6.2 PRESENTER : . . . 17
6.2.1 LoginPresenter : . . . 17
6.2.2 GameClassicPresenter : . . . 17
6.2.3 GameBalloonPresenter : . . . 17
6.3 VIEW : . . . 17
6.3.1 LoginView : . . . 17
6.3.2 GameClassicView : . . . 17
6.3.3 GameBalloonView : . . . 17

**7 Gestion De Projet** **18**
7.1 MVP 1 . . . 18
7.2 MVP 2 . . . 18

**8 ASPECTS TECHNIQUE (git)** **20**

**9 Retour sur expérience** **22**
9.1 Remarques . . . 22
9.2 Possible MVP 3 : . . . 22

**10 CONCLUSION** **24**

**11 ANNEXES** **25**
11.1 Diagramme de classe (Dossier solidarc) . . . 25
11.2 Diagramme de séquance (Dossier solidarc) . . . 25
11.3 Etude des forces de la flèche (Dossier solidarc) . . . 25

# TABLE DES FIGURES

5.1 Use Case Diagram . . . . . 9
5.2 Domain Class Diagram . . . . . 10
5.3 Schéma de l'architecture MVP . . . . . 11
5.4 Package Digram . . . . . 13
5.5 Écran titre . . . . . 14
5.6 Menu de sélection . . . . . 14
5.7 Maquette avec des cibles de type ballon . . . . . 14
5.8 Maquette avec une cible classique . . . . . 14

# CHAPITRE 1

# INTRODUCTION

Bienvenue dans l'univers palpitant du tir à l'arc. Notre rapport vous emmène dans les coulisses de notre projet de jeu de tir à l'arc, où précision, maîtrise et compétition se mêlent pour offrir une expérience inoubliable.
Notre projet consiste en un jeu de tir à l'arc composé de challenges successifs où le joueur doit atteindre des cibles de types différents, à des positions et distances différentes.

# CHAPITRE 2

# PRÉSENTATION DU PROJET

## 2.1 Rappel du sujet

Le projet consiste à développer un jeu vidéo de tir à l'arc pour promouvoir une entreprise d'accessoires d'archerie, incarnée par Monsieur Régis Clouard. Ce jeu, qui met l'accent sur la précision et la concentration, est conçu pour un joueur unique. Il propose plusieurs challenges successifs où l'archer gagne des points en fonction de sa performance, lesquels peuvent être utilisés pour acheter des accessoires améliorant la précision de son tir. Le passage à un nouveau challenge nécessite un minimum de points, conformément aux règles définies.

## 2.2 Contraintes technologiques

Le langage de développement choisi pour ce projet est le Java, en utilisant la bibliothèque JavaFX pour l'interface graphique. Ce choix de langage, orienté objet, nous a été imposé par Monsieur Régis Clouard. De plus, il a exprimé le souhait que nous réalisions des tests unitaires avec les bibliothèques JUnit et Mockito.

La seule contrainte architecturale qui nous a été imposée concerne l'utilisation du modèle MVP (Modèle-Vue-Présentateur), sur lequel nous avons basé l'intégralité de notre conception afin de répondre aux attentes du client.

En ce qui concerne la gestion du projet, notre client a exigé l'utilisation de Git pour la gestion de version, en passant par la plateforme GitLab de l'ENSICAEN. Enfin, nous avons opté pour le moteur de production Gradle, qui nous a permis d'exécuter facilement les tests unitaires.

## 2.3 Contraintes Fonctionnelles

D'un point de vue fonctionnel, trois contraintes ont été posées :
Tout d'abord, le logiciel se doit d'être multilingue, disponible obligatoirement en Français et en Anglais et ouvert à de nouvelles langues dans le futur.
Ensuite, le système doit permettre la mise à jour du tableau de score et de la cagnotte dès qu'une flèche atteint (ou manque) une cible, tout en restant découplé de la cible elle-même.
Enfin, la visée doit intégrer des mouvements erratiques, simulant les petites variations naturelles du tir à l'arc.

# CHAPITRE 3

# ORGANISATION DU PROJET

## 3.1 Rôle Des Membres De L'équipe

- Chef de projet :
  - YAGAPIN Achille
- Responsables technique :
  - ZYGMUNTOWICZ Patrick
  - EL-GRABLY Jeremy
- Architectes :
  - PAKZAD Yam
  - FOUAD Rania
  - LEPOULTIER Benoit
- Développeurs :
  - AMODE-MALL Samir
  - KERVRAN Valentin
  - FOUAD Rania
  - LEPOULTIER Benoit
  - BOCCARA Benjamin
  - PAKZAD Yam
  - ZYGMUNTOWICZ Patrick
  - EL-GRABLY Jeremy
  - YAGAPIN Achille

## 3.2 Structure Du Projet

```
.gitignore
.gitlab-ci.yml
LICENSE
README.md
build.gradle
gradle.properties.ci
gradlew
gradlew.bat
settings.gradle
gradle
```

```
                wrapper
                    gradle-wrapper.jar
                    gradle-wrapper.properties
        src
                main
                      java
                            fr/ensicaen/ecole/genielogiciel
                                model
                                      Arrow.java
                                      Balloon.java
                                      Bow.java
                                      Classic.java
                                      Player.java
                                      Score.java
                                      Target.java
                                presenter
                                      GameBalloonPresenter.java
                                      GameClassicPresenter.java
                                      LoginPresenter.java
                                view
                                    GameBalloonView.java
                                    GameClassicView.java
                                    LoginView.java
                      resources
                          fr/ensicaen/ecole/genielogiciel
                              view
                                    BoardBalloon.css
                                    BoardBalloon.fxml
                                    BoardClassic.css
                                    BoardClassic.fxml
                                    LoginDialog.css
                                    LoginDialog.fxml
                                    Title_screen.png
                                    arrow_shot.mp3
                                    balloon_background.png
                                    classic_background.png
                                    exploding_balloon.mp3
                                    music.mp3
                                    point.png
                                    red-balloon.png
                              MessageBundle.properties
                              MessageBundle_en_US.properties
        test
            java
                fr/ensicaen/ecole/genielogiciel
                      model
                            ArrowTest.java
                            BalloonTest.java
                            BowTest.java
                            ClassicTest.java
                            PlayerTest.java
                            WindTest.java
                      presenter
                          LoginPresenterTest.java
```

# CHAPITRE 4

# ANALYSE DES RISQUES

L'analyse des risques est une phase cruciale pour anticiper les défis potentiels et mettre en place des solutions proactives afin de garantir la continuité du projet sans perturbations majeures.

## 4.1 Identification et évaluation des risques

Dans cette section, nous détaillons les principaux risques identifiés dans le cadre du projet, en les classant selon leur gravité, fréquence, et criticité.

| **Description** | **Gravité (1-4)** | **Fréquence (1-4)** | **Criticité** | **Responsable** | **Prévention** | **Réparation / Plan B** |
|---|---|---|---|---|---|---|
| Manque de connaissance en JAVAFX et GIT | 3 | 2 | 6 | Chef de projet | Mise en commun des connaissances, bonne répartition des rôles | Travail en équipe |
| Problème matériel | 2 | 2 | 4 | Toute l'équipe | Utilisation de Git, Sauvegarde régulière | On aura tout l'historique sur Git |
| Perte de connexion | 2 | 1 | 2 | Architecte | Prévoir une alternative en cas de perte de connexion | Au lieu de récupérer les informations météo sur internet, on les génère automatiquement |
| Ne pas réussir à finaliser le projet en 6 séances | 4 | 4 | 16 | Chef de projet | Suivi régulier et adaptation de la quantité de travail selon les délais | Augmenter le temps de travail |
| Mauvaise affectation des responsabilités sur les tâches | 2 | 1 | 2 | Chef de projet | Adaptation rapide et recul du chef de projet | Suivi régulier pour surveiller le travail de tout le monde |
| Absence de certaines options dans le jeu final | 4 | 2 | 8 | Toute l'équipe | Bonne répartition du travail | Travailler à plusieurs sur les options les plus compliquées |
| Absentéisme des membres de l'équipe | 4 | 1 | 4 | Toute l'équipe | Travail en pair programming | Le binôme de l'absent pourra continuer le travail |

Table 4.1 – Tableau récapitulatif des risques identifiés.

## 4.2 Mesures de prévention et plans d'action

Chaque risque identifié est accompagné de mesures préventives et de solutions correctives afin de réduire son impact ou sa probabilité d'occurrence. Les plans d'action sont décrits de manière à pouvoir réagir rapidement en cas de survenue de ces risques.

# CHAPITRE 5

# ARCHITECTURE

## 5.1 Cas d'utilisation :

Dans ce jeu, le joueur est l'utilisateur principal. Il a la possibilité de contrôler l'arc, une action essentielle qui lui permet de manipuler l'interface du jeu pour ajuster la trajectoire et la force de son tir. C'est une interaction directe entre le joueur et le système, où il doit viser et tirer en fonction de ses compétences.
Le jeu garde également une trace des scores du joueur. À chaque session de tir, les performances du joueur sont enregistrées, et un score est calculé en conséquence. Cette fonctionnalité permet au joueur de suivre ses progrès et de se mesurer à ses performances précédentes.
Un aspect unique de ce jeu est l'intégration des conditions météorologiques réelles, notamment le vent, qui influence la trajectoire des flèches. Pour cela, le jeu se connecte à un service externe de prévisions météorologiques "prevision-meteo.com". Ce service fournit des données en temps réel sur la force et la direction du vent, qui sont ensuite utilisées pour rendre le tir plus réaliste. Le joueur doit donc adapter ses actions en fonction des conditions de vent actuelles pour améliorer ses performances.

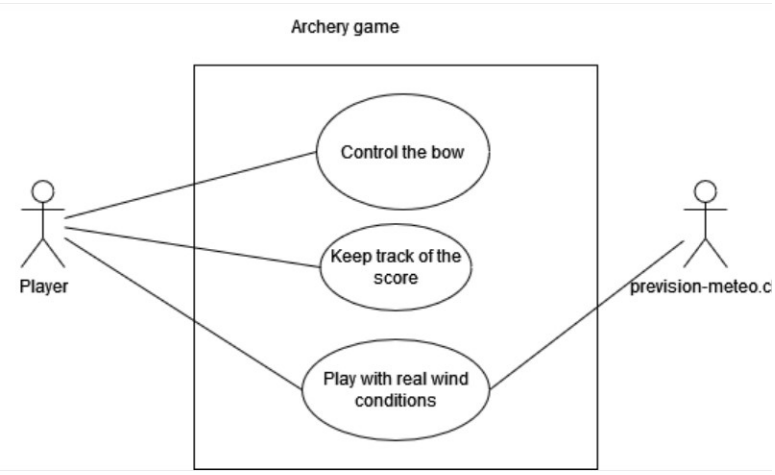


FIGURE 5.1 – Use Case Diagram

## 5.2 Classes Du Domaine :

Le diagramme de classes présenté modélise la structure du jeu de tir à l'arc, illustrant les relations et interactions entre les différents composants du système. Ce modèle montre

comment les éléments essentiels du jeu sont interconnectés pour offrir une expérience de jeu fluide .

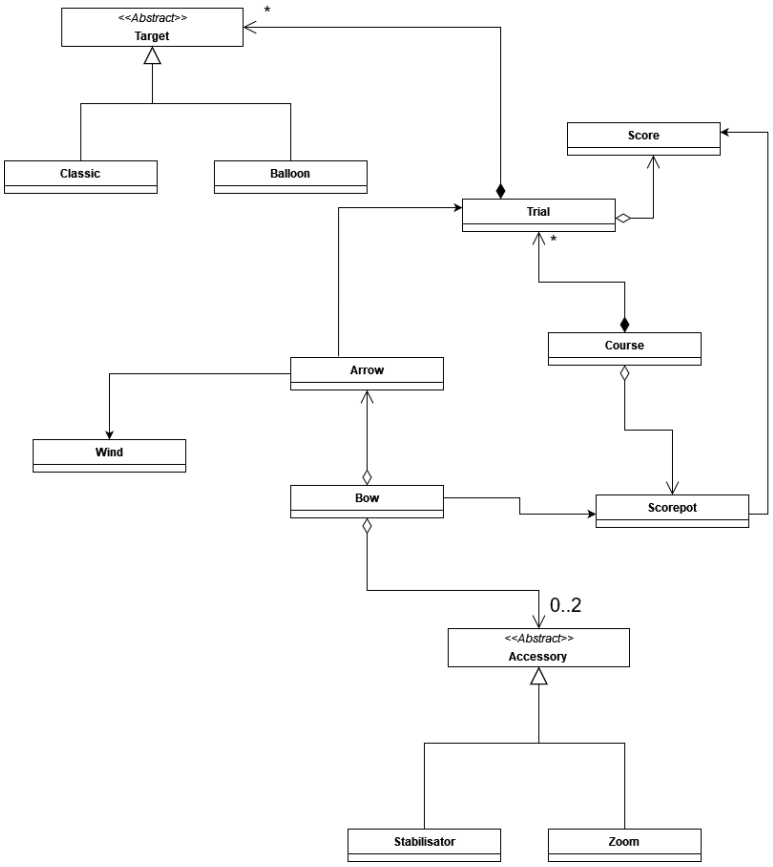


FIGURE 5.2 – Domain Class Diagram

**Relations entre les classes**

— Héritage :
Les classes Classic et Balloon héritent de la classe abstraite Target. Cela permet d'implémenter des comportements spécifiques pour chaque type de cible tout en partageant des caractéristiques communes.

— Composition/Agrégation :
La classe Bow possède une agrégation avec les accessoires (Accessory), et une composition avec les flèches (Arrow), indiquant qu'un arc est constitué de flèches, mais qu'il peut être personnalisé avec différents accessoires. De même, la classe Arrow dépend de l'arc et est influencée par le vent.

— Multiplicité :
Un Trial (épreuve) est associé à une Target, et un Course (parcours) est composé de plusieurs essais. Ces relations reflètent la structure des sessions de tir dans le jeu.

Ce diagramme permet de gérer différents types de cibles, accessoires, et conditions environnementales comme le vent, tout en suivant les scores des joueurs dans différents épreuves et parcours. Ce type de conception rend le système flexible et facilement extensible pour ajouter de nouvelles fonctionnalités ou types d'objets dans le futur.

## 5.3 MVP (Model, View, Presenter) :

Le modèle d'architecture utilisé pour ce projet est le MVP (Modèle, Vue, Présentateur). Ce modèle est particulièrement adapté à la création de logiciels avec une interface graphique, car

il permet de dissocier l'interface utilisateur (Vue) de la logique métier (Modèle) du programme.

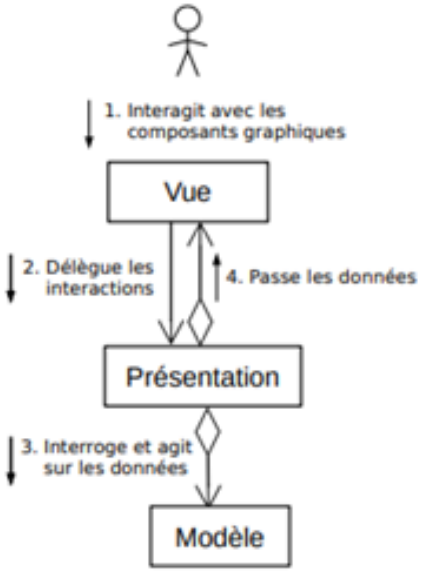


FIGURE 5.3 – Schéma de l'architecture MVP

### 5.3.1 Modèle

La partie Modèle se concentre sur la logique métier et englobe le modèle du domaine ainsi que les données. Dans le cadre de ce projet, elle comprend les classes relatives à la flèche (Arrow), au vent (Wind) et au joueur (Player). Ces éléments sont essentiels pour gérer la dynamique du jeu, y compris les mouvements et les interactions avec l'environnement.

### 5.3.2 Présentateur

La partie Présentateur a pour rôle de faire le lien entre la partie Modèle et la partie Vue. Elle contient la logique de présentation. Les événements sur la Vue y sont traités, ce qui permet de modifier les données du Modèle et d'actualiser la Vue en conséquence. Dans notre projet, la partie Présentateur est constituée des classes Ballon (Baloon), Arc (Bow), Score et Cible (Target).

### 5.3.3 Vue

Enfin, la partie Vue se limite à la gestion de l'aspect graphique et ne contient aucune logique métier. Elle est composée des classes liées à l'espace de jeu (Game) ainsi qu'aux fonctionnalités de connexion (Login).

## 5.4 Conception En UML :

Pour développer un logiciel de qualité, l'équipe a dû intégrer les patrons de conception que nous estimions essentiels pour garantir la robustesse du logiciel tout en respectant les principes SOLID. Cette section met en lumière les différents patrons adoptés ainsi que leurs utilités.

### 5.4.1 Patrons De Conception Utilisés :

**- Composite :**

Ce patron est utilisé pour modéliser une hiérarchie d'objets. Dans le diagramme de classe, la classe Target est abstraite et possède deux sous-classes (Classic et Balloon). Cela permet de traiter les cibles de manière uniforme tout en ayant des comportements spécifiques pour chaque type de cible. Le principe du composite permet de gérer de manière transparente des objets individuels et des compositions d'objets.

**- Décorateur :**

Le lien entre la classe Bow et la classe abstraite Accessory (avec ses sous-classes Stabilisator et Zoom) suit le patron décorateur. Ce patron permet d'ajouter dynamiquement des fonctionnalités supplémentaires à l'arc de manière flexible, sans modifier sa structure de base.

**- Stratégie :**

Le facteur Wind dans le diagramme représente une stratégie qui influence la trajectoire des flèches. Le patron stratégie permet de définir différentes stratégies pour influencer le comportement d'un objet. Ici, la stratégie de gestion du vent pourrait être interchangeable avec d'autres conditions environnementales, offrant ainsi des variations dans le gameplay.

**- Agrégation :**

L'agrégation entre les classes Bow, Arrow, et Accessory montre une relation où l'arc et ses composants ne sont pas liés de manière rigide, mais peuvent fonctionner ensemble de façon indépendante. Ce concept, bien qu'il ne soit pas un patron à part entière, fait référence à une bonne gestion de la modularité des composants du jeu.

### 5.4.2 Diagramme de paquet :

Pour mieux illustrer l'architecture de notre projet, nous avons réalisé un diagramme de paquets. Cela nous a permis de structurer efficacement notre organisation en définissant clairement les responsabilités de chacun, ce qui a contribué à éviter les conflits lors de la répartition des tâches.

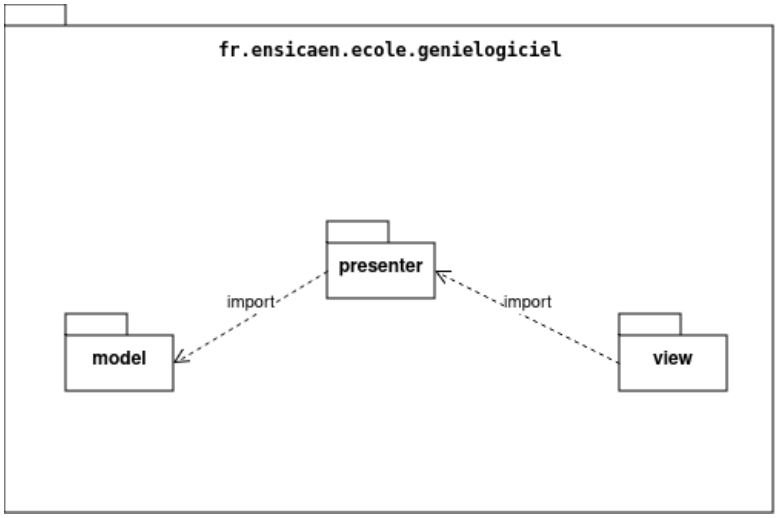


FIGURE 5.4 – Package Digram

## 5.5 Choix d'interface

FIGURE 5.5 – Écran titre

FIGURE 5.6 – Menu de sélection

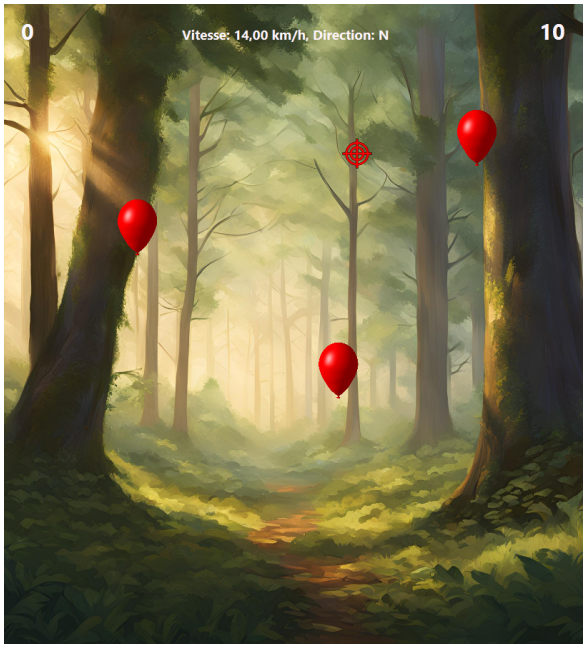


FIGURE 5.7 – Maquette avec des cibles de type ballon

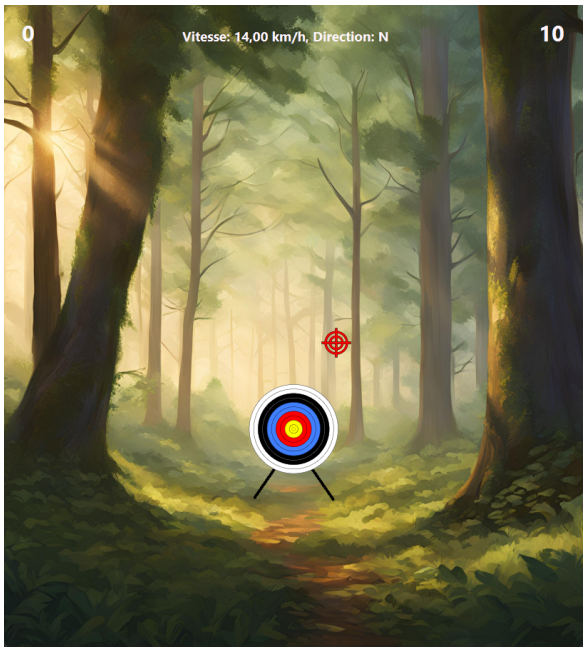


FIGURE 5.8 – Maquette avec une cible classique

Pour que notre interface utilisateur soit confortable, nous avons réparti les éléments dans différentes fenêtres comme suit :

— **Menu de sélection** : La fenêtre présente une barre permettant de saisir son pseudo, suivie de deux boutons pour sélectionner le mode de jeu souhaité. En appuyant sur l'un des boutons, l'écran titre s'affiche. Il suffit ensuite de cliquer sur la barre d'espace pour entrer dans le jeu.

— **Jeu** : Le nombre de flèches restantes, le score, ainsi que les caractéristiques du vent sont affichés à l'écran. Un viseur rouge aide à visualiser la position de l'arc, et lorsqu'une flèche est décochée, un point noir indique l'endroit où elle va atterrir.

— **Différences entre les deux modes** : Dans le mode « cible classique », une cible est présente au centre. En revanche, pour le mode avec des ballons, trois ballons statiques sont affichés, avec des positions préétablies.

Durant la création de la maquette,nous avons du renoncer à certains détails faute de temps et d'organisation.
Concernant le vent, l'idée initiale était d'afficher la direction à l'aide d'une boussole. Nous avions également prévu de faire bouger les ballons. De plus, une barre pour évaluer la force appliquée lors d'un tir aurait dû être présente en bas de la fenêtre, mais cela n'a pas été réalisé.

# CHAPITRE 6

# IMPLÉMENTATION

## 6.1 MODEL :

### 6.1.1 Player :

La classe **Player** permet de stocker le pseudo du joueur à partir du LoginView avant de sélectionner son mode de jeu.

### 6.1.2 Target :

La classe **Target** est une classe abstraite qui permet l'héritage des classes *Classic* et *Balloon*.

### 6.1.3 Classic :

La classe **Classic** définit les caractéristiques de la cible classique et permet de calculer le score entre la flèche et la cible.

### 6.1.4 Baloon :

La classe **Balloon** définit les caractéristiques de la cible ballon et permet de calculer le score entre la flèche et la cible.

### 6.1.5 Bow :

La classe permet de tirer une flèche

### 6.1.6 Arrow :

La classe arrow permet de calculer le point d'arrivé de la flèche dans le plan de la cible

### 6.1.7 Wind :

La classe Wind modélise les conditions de vent, incluant la vitesse et la direction. Elle est conçue pour récupérer ces informations à partir d'une API météo ou, en cas d'échec de la connexion, pour générer des conditions aléatoires.

La classe Wind est incluse dans le package Model car elle représente un élément clé du modèle de l'application dans le cadre du pattern MVP (Model-View-Presenter). Elle gère directement les données relatives aux conditions météorologiques, et encapsule la logique de récupération et de calcul du vent. Elle est indépendante de la vue et du présentateur, respectant ainsi le principe de séparation des préoccupations inhérent au pattern MVP.

### 6.1.8 Score :

La classe **Score** permet d'ajouter des points au score et permet d'être obtenu pour l'affichage du jeu.

## 6.2 PRESENTER :

### 6.2.1 LoginPresenter :

La classe **LoginPresenter** permet de gérer la création des classes (GameClassicPresenter + GameClassicView) ou (GameBalloonPresenter + GameBalloonView).

### 6.2.2 GameClassicPresenter :

La classe **GameClassicPresenter** connait Player et GameClassicView.

### 6.2.3 GameBalloonPresenter :

La classe **GameBalloonPresenter** connait Player et GameBalloonView.

## 6.3 VIEW :

### 6.3.1 LoginView :

La classe **LoginView** gère l'interface graphique utilisateur.

### 6.3.2 GameClassicView :

La classe **GameClassicView** gère l'interface graphique utilisateur pour le mode Classic.

### 6.3.3 GameBalloonView :

La classe **GameBalloonView** gère l'interface graphique utilisateur pour le mode Balloon.

# CHAPITRE 7

# GESTION DE PROJET

Ce projet a constitué une opportunité pour nous de travailler en équipe de 9 personnes en utilisant la méthode agile. Nous avons ainsi établi des objectifs à court terme en élaborant deux MVP (produits minimaux viables) .

## 7.1 MVP 1

Le premier MVP consistait à poser les bases du projet en développant trois classes principales : une classe pour la cible, une pour l'arc, et une pour la flèche. L'objectif de cette phase était de permettre au joueur de viser et de tirer en cliquant gauche sur la cible à l'aide de la souris.
Pour organiser efficacement le travail, nous avons adopté une approche de pair programming, qui consiste à répartir les tâches de manière collaborative. Nous avons donc formé trois binômes, chacun étant responsable de l'implémentation d'une des classes.
Pendant que les binômes travaillaient sur leurs parties respectives, les responsables techniques étaient chargés de gérer les aspects liés à GitLab, notamment la gestion des versions du code et le partage des fichiers. Le chef de projet, quant à lui, s'est occupé de superviser l'ensemble du processus tout en prenant en charge la conception de l'interface graphique.

## 7.2 MVP 2

Le deuxième MVP a pris appui sur les bases posées par le premier, tout en ajoutant un niveau de complexité supplémentaire. L'objectif principal de cette phase était d'intégrer la prise en compte des forces externes, telles que le vent, qui influencent la trajectoire de la flèche. En outre, un nouveau mode de jeu a été introduit, permettant de choisir entre une cible classique ou des ballons. Nous avons également travaillé à la gestion du système de score.
Ce MVP s'est avéré plus difficile à réaliser, car il a nécessité une restructuration importante du projet afin de rendre le code plus modulaire et adaptable. L'absence du chef de projet durant cette période a considérablement déséquilibré l'équipe, compliquant la coordination des tâches et le suivi des objectifs.
Par ailleurs, nous avons porté une attention particulière à l'ajout de tests unitaires pour garantir la stabilité du projet. Alors que pour le MVP1, seule une classe avait fait l'objet de tests, nous avons cherché, pour le MVP2, à couvrir davantage de fonctionnalités, notamment celles relatives aux nouvelles mécaniques de jeu et à l'interaction avec les forces externes.
Le MVP2 s'est ainsi articulé autour de deux axes principaux : l'amélioration des fonctionnalités existantes afin de les rendre pleinement opérationnelles et l'ajout de nouvelles fonctionnalités.

Plus précisément, nous avons intégré les éléments suivants :

- L'ajout du nombre de flèches affichées à l'écran.
- L'affichage de la direction et de la vitesse du vent pour une meilleure expérience utilisateur.
- L'ajout d'une musique de fond et d'un son pour la flèche afin d'enrichir l'immersion.
- L'ajout du vent pour rendre le tout plus réaliste.

Grâce à ces améliorations, le jeu est devenu plus complet et immersif, tout en proposant des défis supplémentaires au joueur.

# CHAPITRE 8

# ASPECTS TECHNIQUE (GIT)

Concernant l'aspect technique de notre projet, nous avons utilisé un logiciel de gestion des versions qui permet de centralisé les versions de chaque personne indépendamment des autres du nom de GitLab.

Ce logiciel fonctionne de la manière suivante, on crée un espace virtuel auquel chaque membre de l'équipe peut accéder avec des droits différents en fonction de ses besoins (développeur s'il n'ajoute que du code, mainteneur s'il gère l'addition des versions du projet, etc...). Ce fut la première contrainte, répartir les droits comme il se doit entre les différents membres car il a fallut rajouter des personnes pour maintenir la gestion des versions dans le cas où les autres seraient absentes ou indisponible car entrain de gérer les versions du projet.

Il y a ensuite toute la partie locale de Git où chaque membre a sa propre version locale du projet qu'il peut ajouter sur la version distante pour la mettre à jour et que tout le monde ait accès à son travail. Pour faire cela il a fallu d'abord transmettre toutes les connaissances aux autres membres pour qu'il puisse se connecter et ajouter sur le Git distant leur travail sans eronner le travail des autres. Il y avait pour cela un système de branche où chacun avait la sienne et pouvait l'ajouter sans écraser le travail commun.

La partie la plus compliquée était la gestion des version à proprement parler car il fallait gérer les différents conflits entre les versions locales et la version distante. Il était très facile de se perdre et surtout de perdre des données lorsqu'un membre poussait sa version locale sur la version distante. Il y a eu plusieurs fois des données qui n'étaient pas les bonnes ou alors simplement que l'on ne retrouvait pas sur le Git ce qui était aussi un défi car il fallait d'abord identifié la source du problème, pas toujours si évidente, ensuite le régler, ce qui était délicat car il ne fallait pas écraser toutes les données importantes du projet.

Enfin nous avions une contrainte sur la structure du projet qu'il fallait gérer car cela amenait à une restructuration qu'il fallait d'abord appliquer à la version distante du git (pendant ce temps personne ne pouvait modifier la version distante du git au risque de perdre des informations essentielles) et ensuite transposé ces modifications structurelles à chaque version locale de chaque membre où certain ont eu d'autres problèmes qu'il fallait gérer de manière locale.

Pour finir, GitLab fut un excellent outil car il permettait à tout membre d'avoir la version la plus récente du projet. Même s'il y avait un certain nombre de conflits à gérer cet outil est assez ergonomique et permet de les gérer quelle que soit leur ampleur. Ce fut un outil intéressant et cela nous a montré à quel point la gestion de version d'un projet est essentiel

lorsque plusieurs équipes travailles sur des parties différentes d'un même projet.

# CHAPITRE 9

# RETOUR SUR EXPÉRIENCE

## 9.1 Remarques

Dans ce projet, l'architecture logicielle, bien que fondée sur des principes établis, a constitué un défi majeur pour garantir la flexibilité, la robustesse et la maintenabilité du jeu.

On peut aussi observer que l'implémentation du projet a été réalisée en respectant les bonnes pratiques de programmation, avec un souci constant de modularité, de réutilisabilité du code, et d'efficacité.

Il convient de souligner que nous avons été confrontées à des défis liés à la gestion des différentes propositions des membres de l'équipe dans l'implémentation des classes, car étant une grande équipe, il a parfois été difficile de coordonner les idées et d'assurer une cohérence dans les approches techniques adoptées. Cela a nécessité un effort supplémentaire pour harmoniser les contributions et garantir que les décisions prises soient alignées avec les objectifs du projet.

Il est important de mentionner que l'utilisation de Git a été un atout majeur pour la gestion du code source, permettant un suivi rigoureux des modifications, une collaboration efficace et une maîtrise des versions.

Il est également à noter que nous sommes passées par plusieurs conceptions différentes avant de parvenir à une version fonctionnelle. Sans une entente et une coordination solide, cela n'aurait pas pu être possible.

## 9.2 Possible MVP 3 :

Objectifs Principaux :

- **Ameliorer l'impact du vent dans le jeu** : Le vent a bien été implémenté, mais des erreurs de calculs affectaient la trajectoire des flèches. Nous avons donc décidé de le retirer du calcul de la trajectoire.
- **Implémenter une mécanique d'ajustement de la force de tir** : Le joueur doit cliquer et maintenir le bouton de la souris pour ajuster la force du tir. En fonction du temps utilisé pour maintenir le bouton de la souris, la force du tir varie. Une barre de puissance visuelle est affichée pour indiquer la force en temps réel.
- **Mettre en place une boutique et une Monnaie de Jeu** : Les points obtenus sont

convertis, à la fin d'une épreuve, en pommes. Ces pommes s'accumulent dans une cagnotte utilisable pour acheter des accessoires :

  - **Stabilisateur** : Cet accessoire offre une meilleure stabilité lors du tir.
  - **Zoom** : Il permet de zoomer sur la cible, améliorant ainsi la précision.

- **Assurer que le jeu soit bilingue** : Toutes les instructions, menus et descriptions dans le jeu sont disponibles en français et en anglais. Le joueur peut choisir la langue dans les paramètres.
- **Ajouter davantage de tests unitaires** : L'ajout de tests unitaires permet d'assurer la stabilité et la fiabilité du code, améliorant ainsi sa qualité. Ces tests vérifient automatiquement que chaque fonctionnalité du jeu fonctionne comme prévu

# CHAPITRE 10

# CONCLUSION

En conclusion, ce projet nous a permis de découvrir l'importance de travailler en grande équipe, où chacun joue un rôle essentiel dans la réussite collective. Nous avons appris à respecter les responsabilités de chacun, à collaborer efficacement et à valoriser la complémentarité des compétences. De plus, nous avons rigoureusement suivi les principes et les bonnes pratiques du génie logiciel, garantissant ainsi la qualité, la robustesse et la maintenabilité de notre travail. Ces apprentissages se sont révélés indispensables pour mener à bien un projet d'une telle envergure.

# CHAPITRE 11

# ANNEXES

## 11.1 Diagramme de classe (Dossier solidarc)

## 11.2 Diagramme de séquance (Dossier solidarc)

## 11.3 Etude des forces de la flèche (Dossier solidarc)