Rapport DEVLO-PROJ :

Bénétreau Arthur
de Sallier Dupin Arthur

Introduction :

1) Principe du projet :

Notre projet s'inscrit dans le développement d'un jeu vidéo multijoueur en temps réel, intitulé **Arena**. Inspiré du célèbre jeu *League of Legends*, **Arena** reprend les grands principes d'un jeu de type *MOBA* (*Multiplayer Online Battle Arena*), tout en proposant une architecture technique simplifiée.

Dans **Arena**, deux équipes s'affrontent sur un terrain symétrique dans le but de défendre leur territoire tout en cherchant à détruire les structures adverses. Chaque joueur incarne un **champion**, une entité vivante dotée de caractéristiques spécifiques (points de vie, attaques, portée, etc.). Le jeu intègre également des **tours de défense**, une logique de **ticks** synchronisant les actions en temps réel, un système de **priorisation des messages réseau**, et une **synchronisation continue de l'état du jeu** entre les clients et le serveur.

Ce projet a été l'occasion de concevoir un **moteur de jeu serveur** entièrement en Java, basé sur une architecture orientée messages, avec une gestion fine des entités, des actions et des réponses, dans un univers multijoueur réactif. L'objectif principal n'était pas de reproduire la complexité de *League of Legends*, mais d'en reproduire une version simplifiée.

2) Outil utilisé :

Pour la partie cliente et l'interface graphique du jeu, nous avons utilisé Unity. Ce moteur de développement a été choisi pour sa compatibilité avec le multijoueur en temps réel, sa richesse en outils de modélisation 3D, d'animation et d'interface utilisateur, ainsi que pour sa capacité à interagir facilement avec un serveur externe via des sockets ou des API réseau. Unity permet également de développer rapidement des prototypes visuels et de gérer efficacement les scènes, les entrées utilisateur et les effets visuels, ce qui en faisait un choix pertinent pour accompagner notre moteur serveur Java.

Le développement du moteur serveur a été réalisé principalement sous IntelliJ IDEA.

JetBrains Rider a été utilisé pour le développement sous Unity, notamment pour sa prise en charge de C# et son intégration avec le moteur.

Enfin, la gestion du versionnement et du travail collaboratif a été assurée via GitKraken, qui fournit une interface graphique efficace pour suivre les modifications, gérer les branches et faciliter les fusions.

3) Principales étapes du projet :

Le développement du projet s'est déroulé en plusieurs étapes :

1. **Installation des outils nécessaires** : Mise en place de l'environnement de travail, incluant Unity, JetBrains Rider et GitKraken, afin de préparer le terrain pour le développement collaboratif et multiplateforme.

2. **Prise en main des outils** : Familiarisation avec GitKraken pour la gestion du versionnement, et surtout avec Unity pour le développement de l'interface graphique et la construction des éléments du jeu. Cette étape inclut également le choix de la carte de jeu, élément central du gameplay.

3. **Conception de l'architecture du projet** : Réflexion sur l'organisation générale du code, la répartition des responsabilités entre le serveur et les clients, la définition des classes et des méthodes principales, ainsi que la mise en place des protocoles de communication nécessaires au bon fonctionnement du jeu en temps réel.

4. **Développement et intégration** : Implémentation progressive du moteur de jeu côté serveur, création et assemblage des composants visuels dans Unity, et réalisation de nombreux tests pour valider le bon fonctionnement de l'ensemble et corriger les éventuels dysfonctionnements.

Le diagramme UML:

Étant donné la taille du diagrammes UML il sera livré en annexe avec le script de plantUML

Cahier de test :

voire Annexe

Explication des choix technique :

Le serveur joue un rôle central dans la gestion de l'état de la partie. Il est responsable de l'envoi régulier des mises à jour aux clients toutes les 10 millisecondes. Ces derniers adaptent leur environnement en fonction des entités présentes dans la réponse "Game State" du serveur, garantissant ainsi une synchronisation fluide et cohérente des données.

Chaque joueur se voit assigner une entité spécifique. Des scripts sont à l'écoute de cette entité et récupèrent des informations telles que sa position et ses mouvements, permettant ainsi une interaction dynamique et réactive.

Structure des classes :

- **Classe Serveur** : Gère l'ensemble des parties en cours et les joueurs associés.
- **Classe Game** : Contient les entités actives, un identifiant unique de partie (**game ID**) et les joueurs.
- **Core en Thread** : Cadence l'exécution des actions de la partie, assurant une fluidité et une réactivité optimale.

Cette architecture garantit une bonne gestion des interactions en temps réel et une mise à jour efficace des entités du jeu. Dis-moi si tu veux affiner ou développer un point !

Conclusion :

1) Qui a fait quoi :

Arthur de Sallier Dupin :

- **Développement** : Code Java et tests sur l'ensemble du projet.
- **Code Rider** : Implémentation et ajustements sur tous les aspects du jeu.
- **Documentation** : Élaboration du support de présentation et du rapport.

Arthur Bénétreau :

- **Développement** : Code Java et tests.
- **Code Rider** : Implémentation de la barre de vie et des quatre cooldowns.
- **Modélisation** : Création des diagrammes UML.
- **Documentation** : Participation à la rédaction du support de présentation et du rapport.

2) Analyse des résultats :

Réussites :

- Le héros est entièrement contrôlable, avec une barre de vie fonctionnelle et quatre cooldowns correctement implémentés pour représenter les attaques.
- L'attaque directionnelle fonctionne comme prévu, accompagnée d'animations adaptées.
- La gestion des entités principales (Tour, Nexus, Inhibiteur) est en place, chacune disposant d'une barre de vie et d'une animation lors de leur destruction.
- La carte est opérationnelle, avec des textures et des collisions correctement implémentées.

Pistes d'amélioration :

- Absence d'autres entités, notamment des sbires, pour combattre aux côtés des joueurs.
- Possibilité de n'incarner qu'un seul héros, limitant le choix des joueurs.

3) Utilisation de l'IA :

Dans le cadre de notre projet, nous avons utilisé l'intelligence artificielle ChatGPT et Copilot pour identifier et corriger les erreurs présentes dans notre code et durant la modélisation. ChatGPT nous a également apporté des conseils sur l'utilisation de Unity et de GitKraken, facilitant ainsi notre travail.