

The Last Stand

Par Jonathan Robinson et Andrzej Wisniowski

Conception

Remis à Monsieur Jean-Christophe Demers

Dans le cadre du cours *420-C61-IN - Projet Synthèse*

Techniques de l'informatique

Cégep du Vieux-Montréal

2 mars 2023

Table des matières

[Interface utilisateur](#)

[Diagrammes](#)

[Cas d'usages détaillés](#)

[Diagrammes de classes détaillés](#)

[Diagramme de données externes](#)

[Éléments de conception](#)

[Structures de données](#)

[Arbre quadtree de points :](#)

[Liste chaînée double :](#)

[Dictionnaire :](#)

[Arbre de comportement :](#)

[Patrons de conception](#)

[Observateur :](#)

[Expression régulière](#)

[Algorithme](#)

[Arbre de comportement :](#)

[A Star :](#)

[Mathématique](#)

[Veille technologique : React](#)

Conception

Interface utilisateur

Les pages suivantes présenteront les maquettes utilisées pour développer l'interface utilisateur de l'application. Ces maquettes ont été construites avec l'aide d'un outil de développement de design appelé Figma, qui permet rapidement de créer des prototypes.

Chaque maquette présentée a été conçue pour représenter une étape clé de l'expérience utilisateur de l'application. Elles permettent de visualiser les différentes fonctionnalités, les interactions possibles, ainsi que l'aspect visuel de l'interface.

En parcourant ces pages, vous pourrez également vous faire une idée de la structure de l'application, de la disposition des éléments sur l'écran et de l'ergonomie générale de l'interface.

Nous espérons que cette présentation des maquettes vous permettra de mieux comprendre la conception de l'interface utilisateur de l'application et de vous donner un aperçu de ce à quoi elle ressemblera une fois développée.

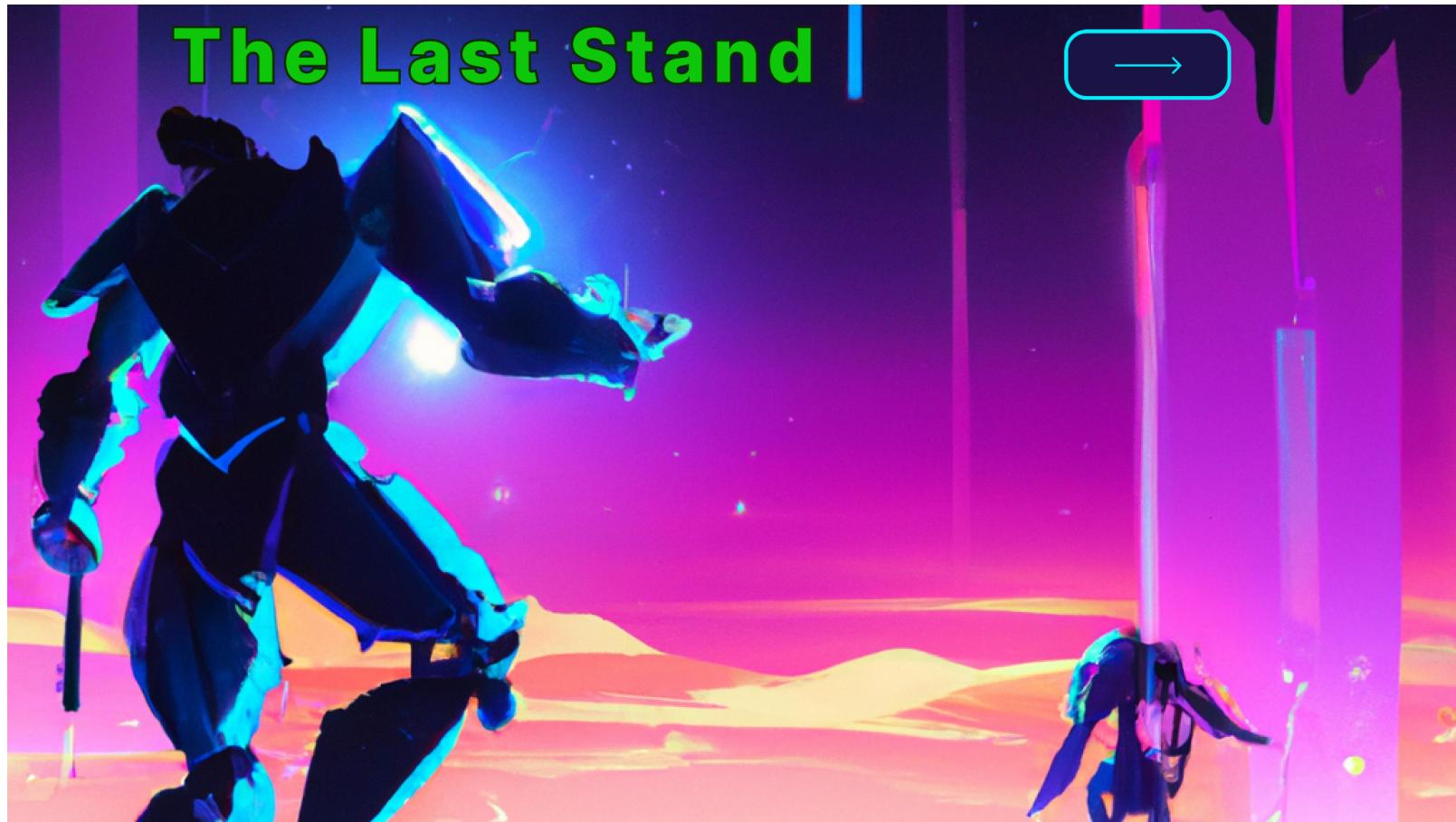


Figure 1 : Page d'accueil.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

Page 2 de 31

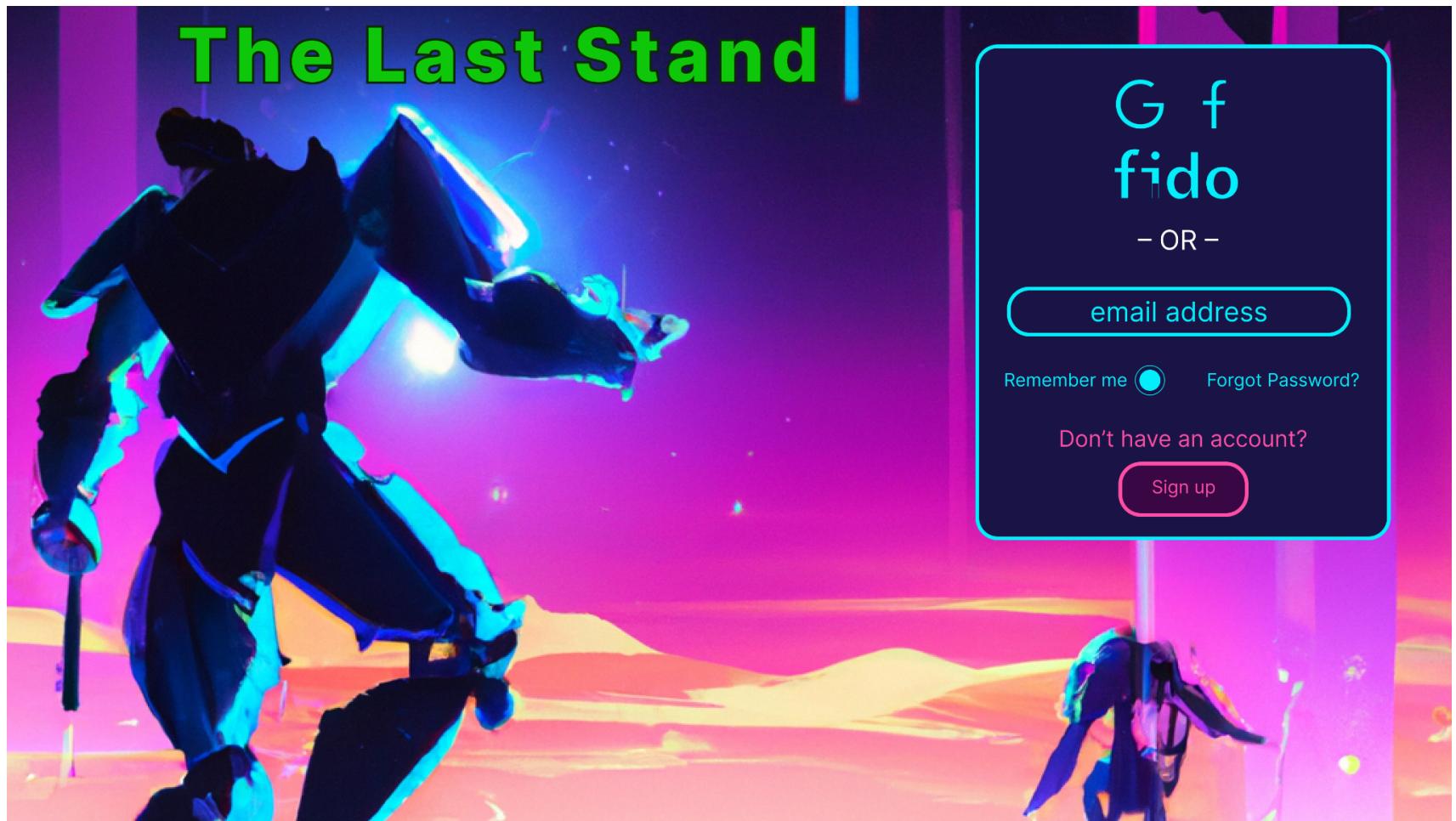


Figure 2 : Page d'accueil avec menu de connexion.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

Page 3 de 31

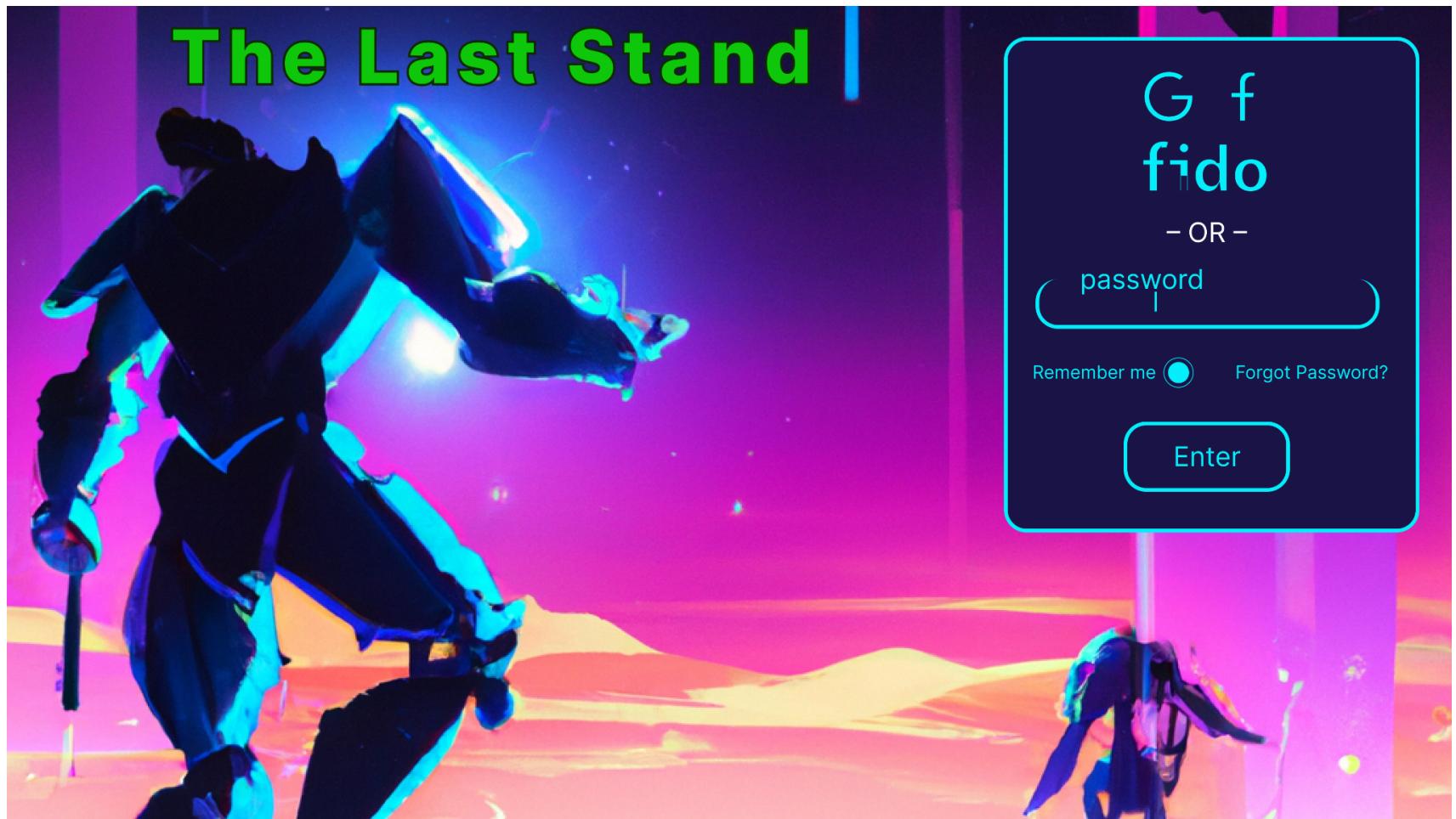


Figure 3 : Page d'accueil avant de confirmer la connexion.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

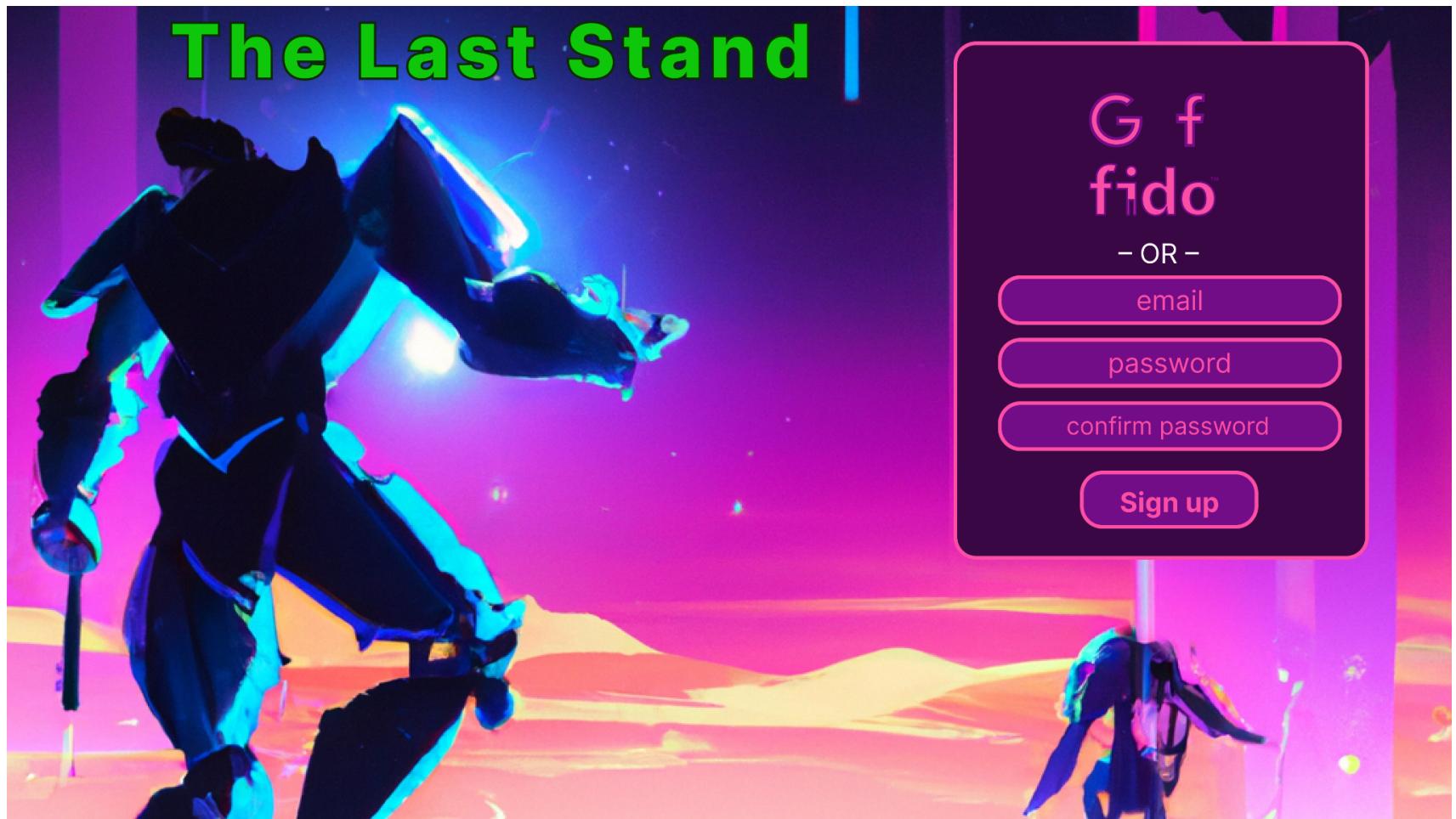


Figure 4 : Page d'accueil avec menu d'inscription.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

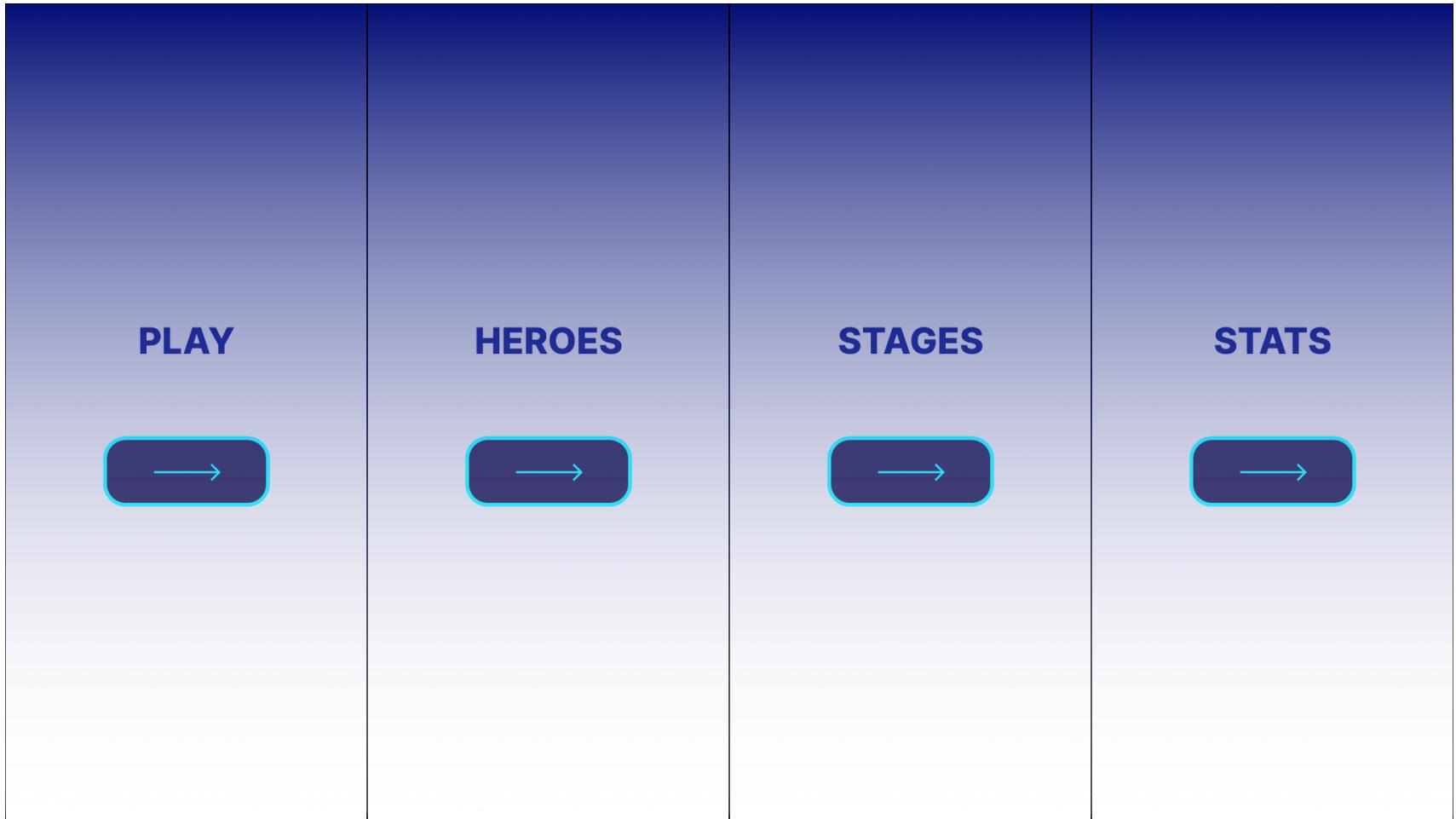


Figure 5 : Salon principal.

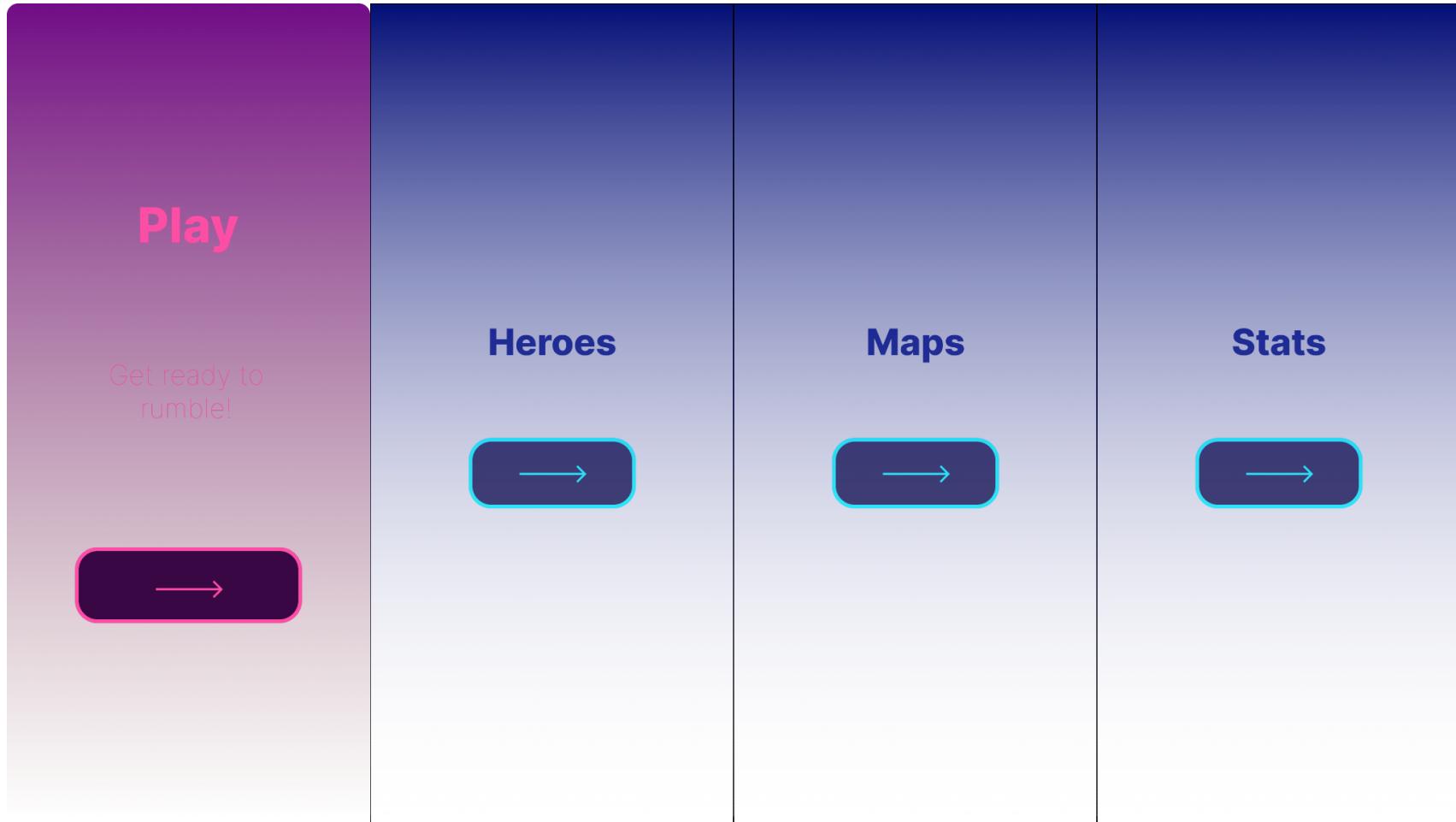


Figure 6 : Salon principal avec la première option 'hovered'.



Figure 7 : Page de jeu modèle. (Rivals of Aether, n.d.)

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception



Figure 8 : Page de jeu modèle. (Rivals of Aether, n.d.)

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

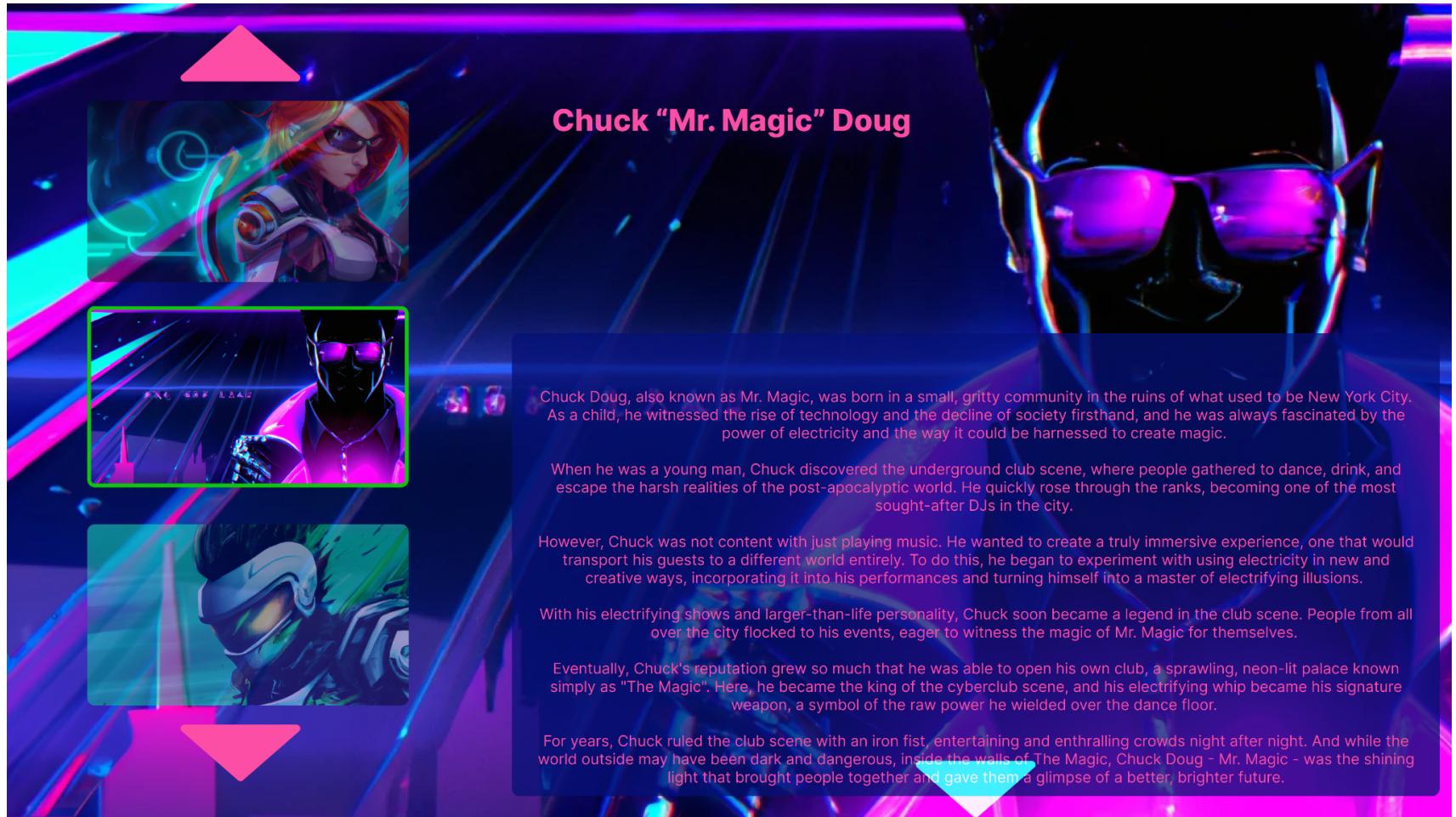


Figure 9 : Page de héros.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception



Figure 10 : Page de stages.

Stats

Most played heroes : Chuck Doug

Win ratio : 0.25

Hit ratio : 0.10



Figure 11 : Page de stats.

Diagrammes

Avant de produire le code nécessaire à la réalisation de la preuve de concept, une planification détaillée de la structure du logiciel à réaliser a été mise de l'avant, afin de faciliter le développement et de s'assurer que la construction sera faite de façon organisée et robuste.

Dans le cadre de cette planification, des diagrammes d'usages, de classes et de données UML ont été utilisés pour décrire les différentes parties du logiciel, leur interrelation, et les actions que les utilisateurs pourront effectuer.

Cas d'usages détaillés

Le diagramme d'usage UML est un outil qui permet de représenter les interactions entre les différents acteurs du système et les cas d'utilisation associés. Il permet de visualiser de manière intuitive le flux d'actions entre les différents éléments du système, et de définir les fonctionnalités du logiciel en termes d'actions à effectuer et de résultats attendus.

Dans le cadre de notre projet, sont présentés ci-dessous les diagrammes d'usage détaillés qui ont été réalisés.

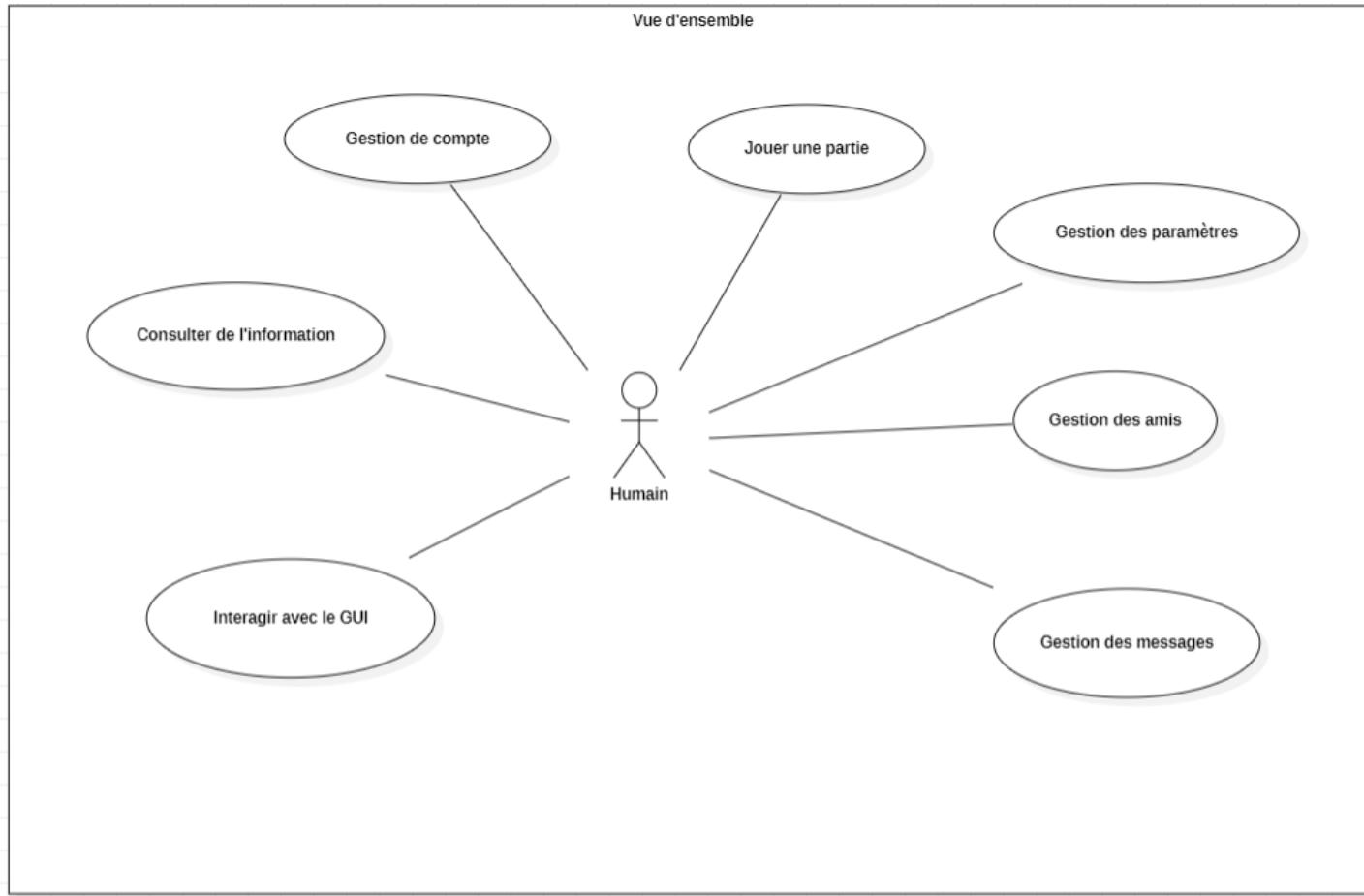


Figure 12 : Survol des cas d'usages.

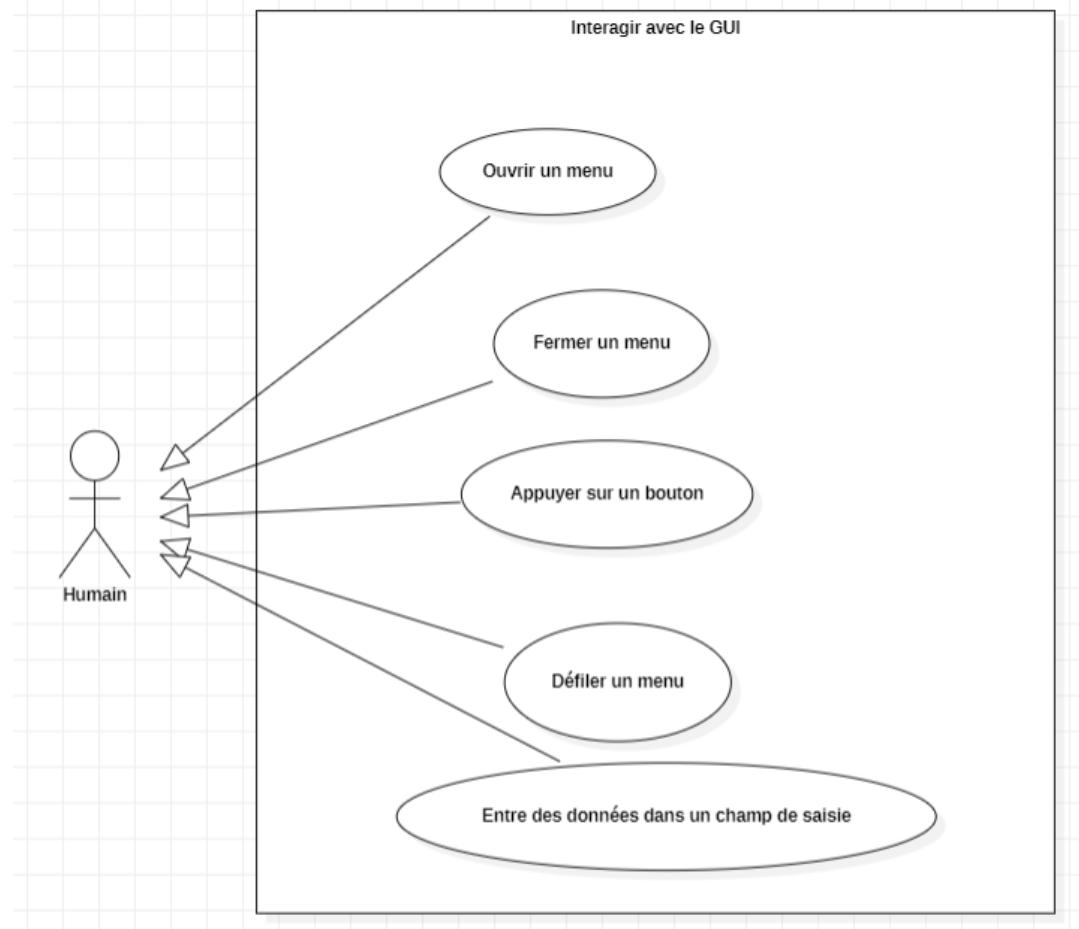


Figure 13 : Cas d'usages - Interagir avec le GUI.

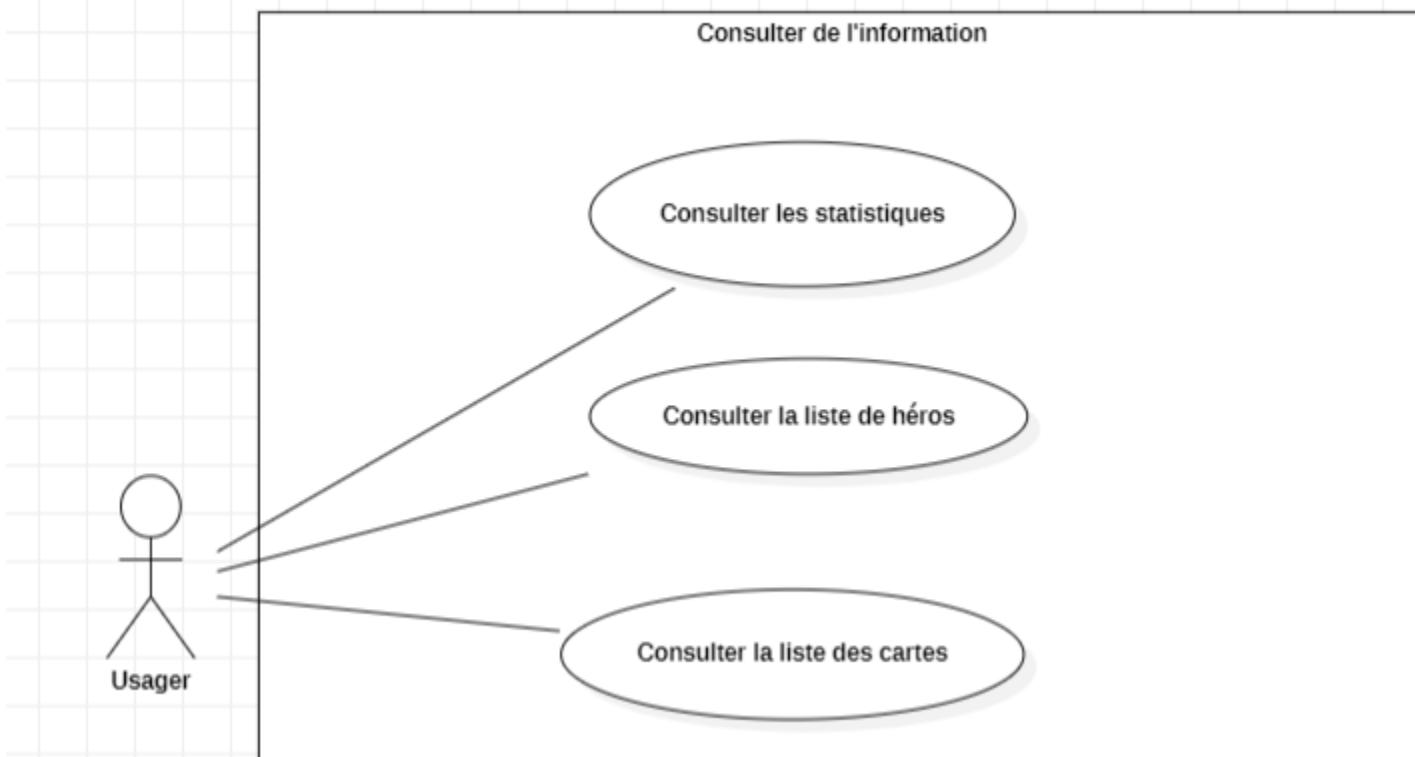


Figure 14 : Cas d'usages - Consulter de l'information.

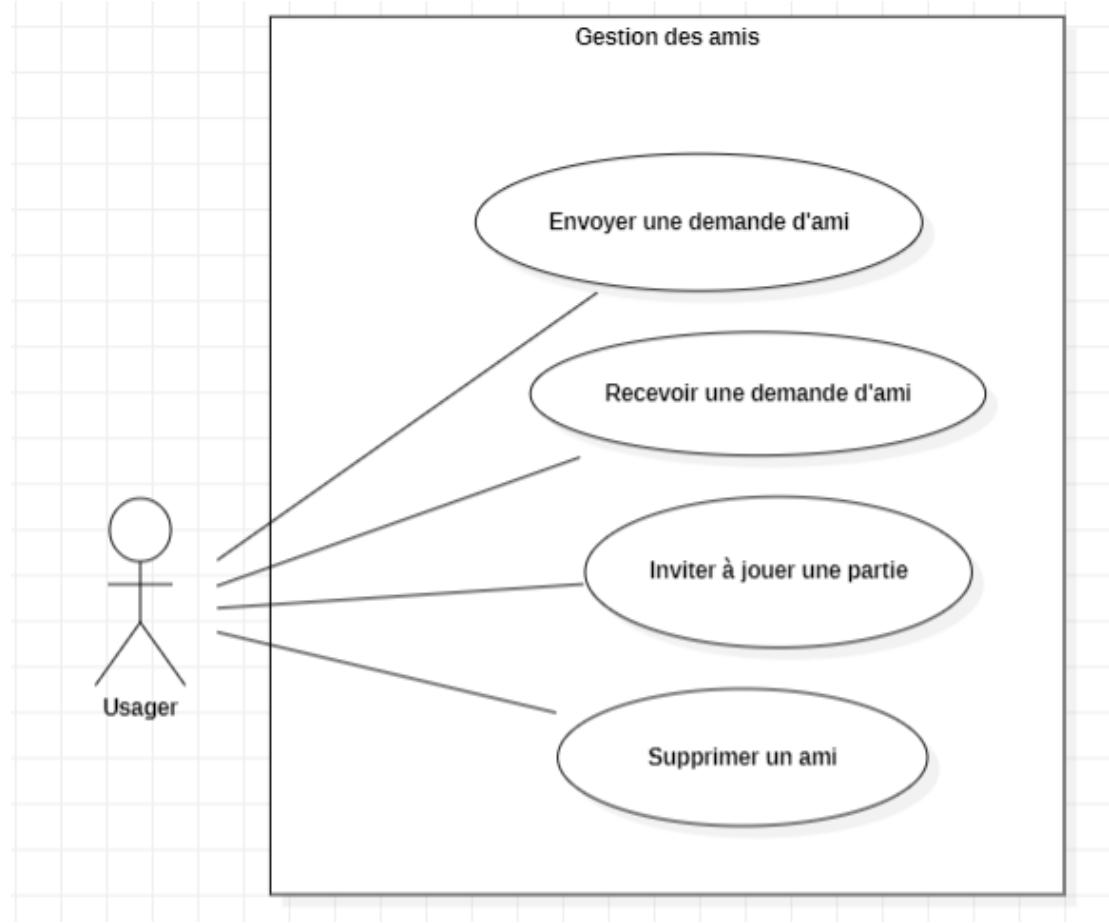


Figure 15 : Cas d'usages - Gestion des amis.

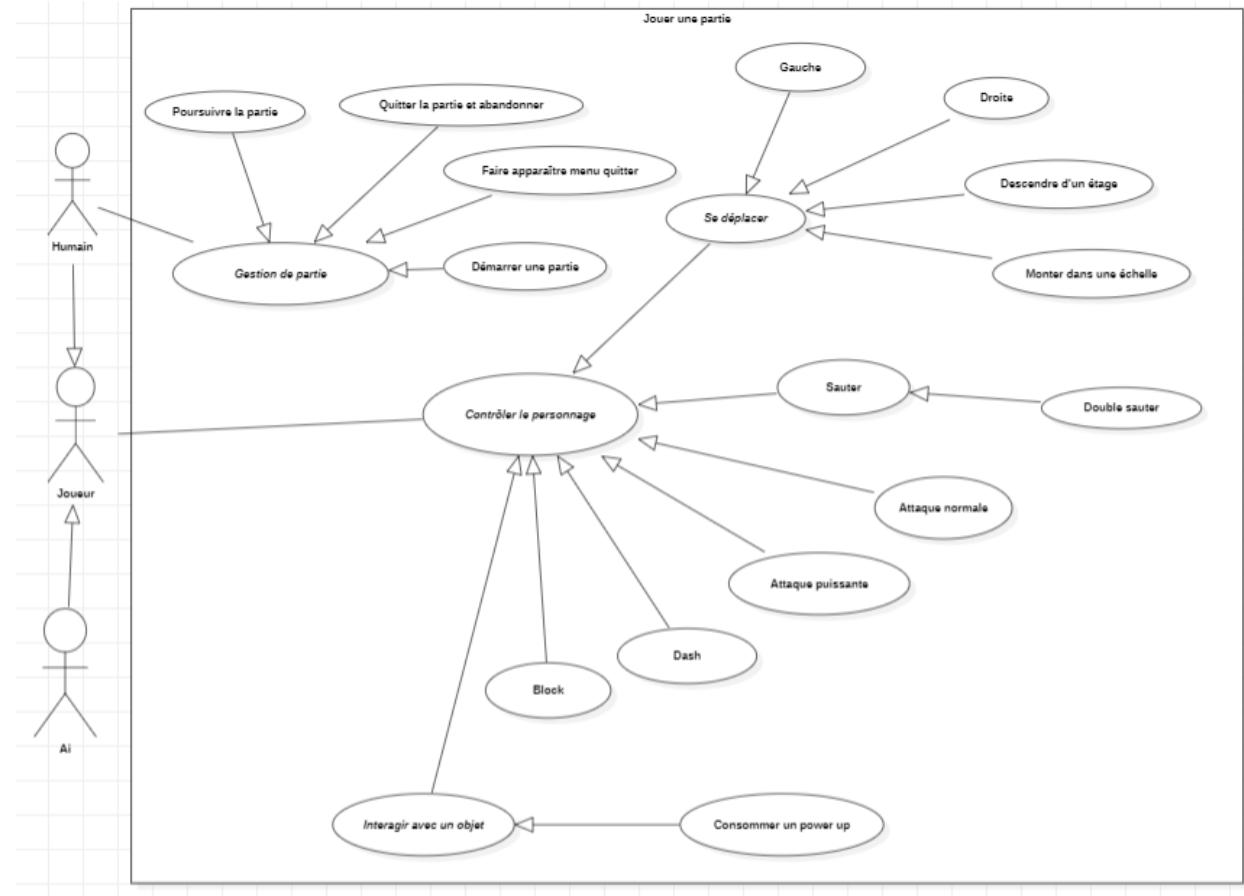


Figure 16 : Cas d'usages- Jouer une partie.

Diagrammes de classes détaillés

Le diagramme de classe UML est un outil qui permet de représenter les différents types d'objets impliqués dans le système, ainsi que leurs attributs et leurs relations. Il permet de définir les différentes classes et leurs méthodes, les relations d'héritage et de composition, et les interactions entre les classes du système.

Dans le cadre de notre projet, sont présentés ci-dessous les diagrammes de classes détaillés qui ont été réalisés.

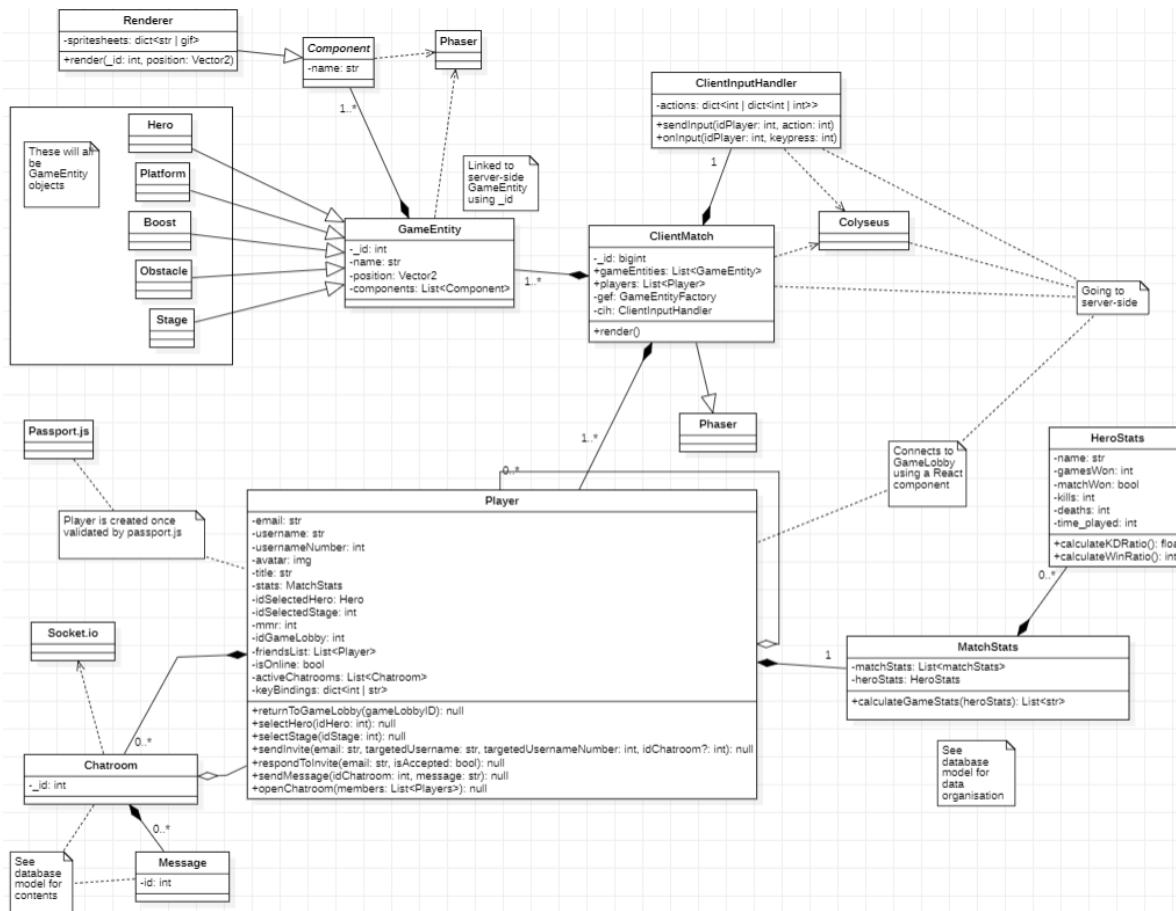


Figure 17 : Logique du côté client.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

Page 20 de 31

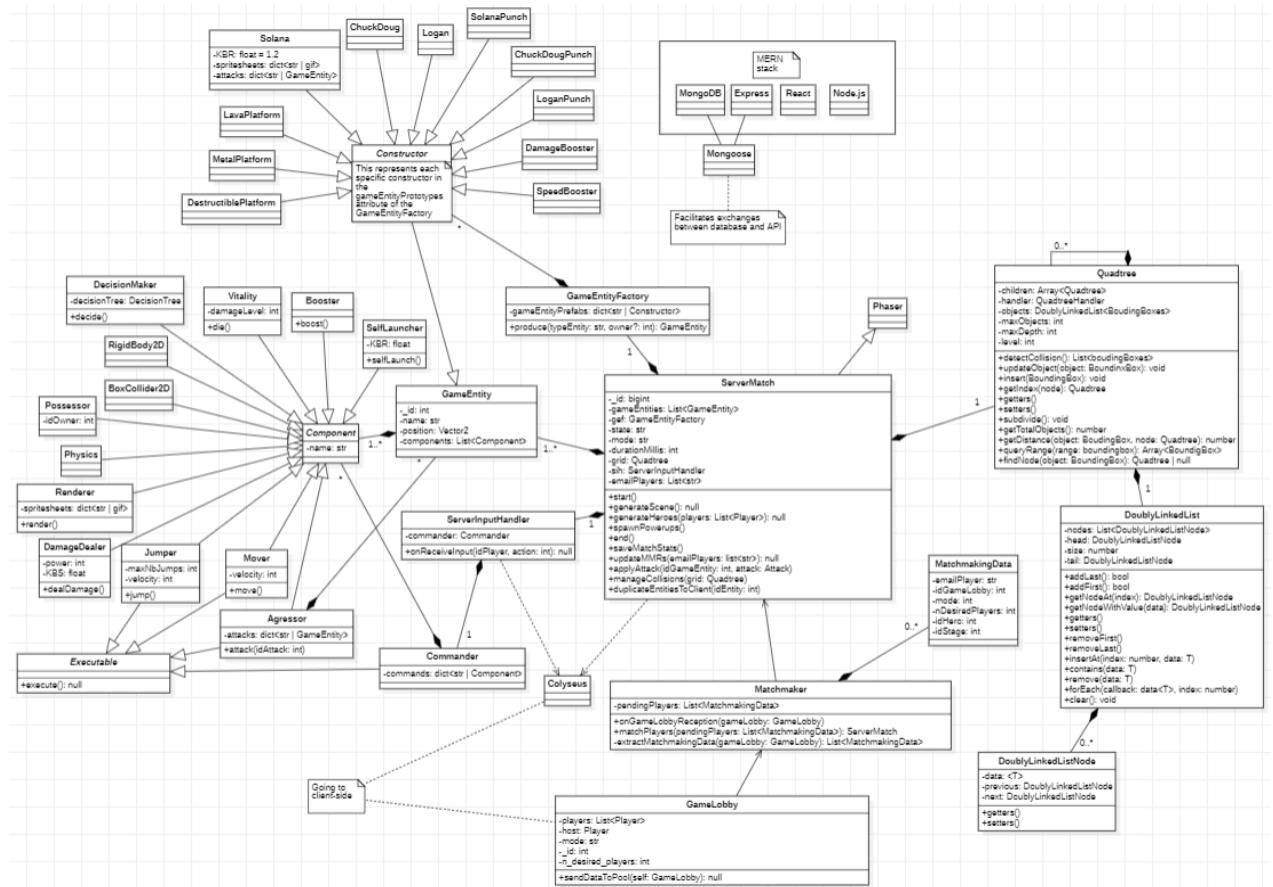


Figure 18 : Logique du côté serveur.

Jeu de combat multijoueur en ligne pour PC, preuve de concept : "The Last Stand"

Conception

Diagramme de données externes

Le diagramme de classe UML est un outil qui permet de représenter les différents types d'objets impliqués dans le système et leurs relations. Dans le contexte de l'utilisation de la base de données MongoDB, le diagramme de classe peut être utilisé pour représenter les modèles de données définis dans l'application.

Ces modèles de données peuvent être représentés sous forme de classes qui définissent les propriétés stockées dans la base de données. Les propriétés d'une classe correspondent aux champs des documents MongoDB et peuvent inclure des types de données tels que des chaînes de caractères, des nombres ou des tableaux.

En outre, les classes peuvent être liées par des associations pour représenter les relations entre les différentes entités du système. Dans le contexte de MongoDB, les références entre documents peuvent être représentées par des associations un-à-plusieurs.

Dans le cadre de notre projet, est présenté ci-dessous le diagramme de données externe répondant à l'évaluation de la structure de la base de données.

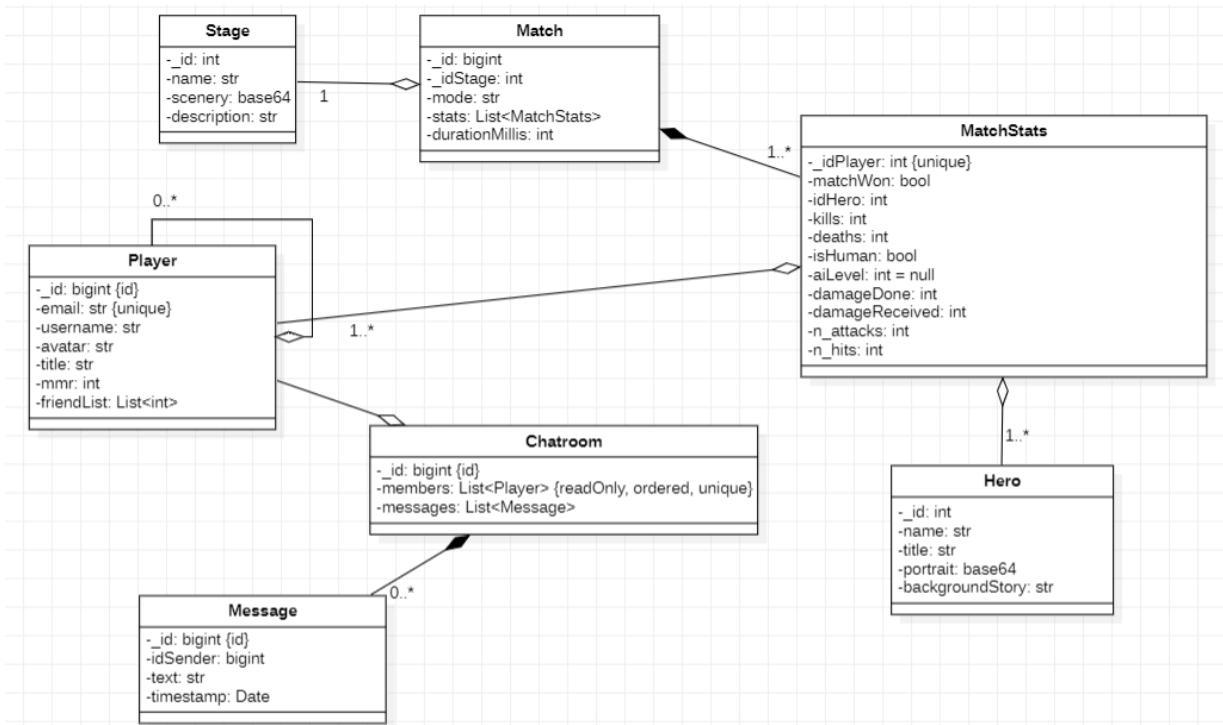


Figure 19 : Base de données MongoDB.

Éléments de conception

Structures de données

Voici les structures de données que nous allons utiliser :

Arbre quadtree de points :

Nous allons utiliser un arbre quadtree de points pour gérer la détection de collision entre les objets. Ce type d'arbre est une structure de données efficace pour la recherche spatiale en raison de sa rapidité de recherche, l'optimisation de la bande passante, la gestion de la densité des objets et la flexibilité. Puisque nous aurons un jeu dynamique où la position des objets changera constamment, il est important d'avoir une structure de données dynamique pour prendre en compte les mouvements des objets dans l'espace. Lorsqu'une entité du jeu se déplacera ou sera créée, il faudra la ranger dans le quadrant approprié. Dans le jeu, la scène entière représente le quadrant principal, qu'on appelle aussi le quadrant racine. Lorsque des objets sont ajoutés au jeu ou se déplacent, il faut dynamiquement les déplacer dans le quadrant approprié. Si un quadrant est rempli d'objets de jeu (la taille maximale est atteinte), l'algorithme du Quadtree subdivise la zone en quatres autres quadrants dans lesquels il distribuera le plus uniformément possible les objets. Dans notre jeu, puisque les joueurs pourront s'attaquer entre eux ou interagir avec des objets dans l'environnement, lorsque le bouton d'attaque ou d'action sera activé, le quadtree devra rechercher si le joueur est entré en "collision" avec un autre joueur ou un "power up". Pour se faire, l'algorithme de détection du quadtree partira de la racine pour voir si le joueur s'y trouve, s'il n'y est pas, il se déplace ensuite vers les noeuds enfants pour continuer la recherche, ainsi de suite récursivement jusqu'à ce que le joueur soit trouvé. Lorsque celui-ci est trouvé, il ne reste plus qu'à évaluer les autres objets dans le quadrant dans lequel il se trouve pour détecter si une collision est survenue et exécuter les actions suivantes.

Liste chaînée double :

Nous allons utiliser la liste chaînée double pour entreposer les entités du jeu, tels que les joueurs, les projectiles et les améliorations, dans notre arbre quadtree de points. Les listes chaînées sont efficaces puisqu'elles facilitent l'insertion et la suppression d'éléments. Puisque nous faisons un jeu dans lequel il y aura beaucoup de mouvement et d'ajouts/suppression de données dynamiquement d'une liste à l'autre, il est important d'avoir une liste qui consomme moins de mémoire, pour ne pas alourdir le jeu. Des listes chaînées feront partis des noeuds des quadtrees et lorsqu'un joueur se déplacera sur la carte de jeu, dépendamment de sa position et du nombre d'objets autour de lui, il sera possiblement déplacé d'un noeud à un autre du quadtree et par le fait même d'une liste chaînée à une autre. Puisque les listes chaînées n'ont pas besoin de déplacer tous les éléments de la liste pour effectuer une opération de suppression ou d'ajout, notre jeu sera plus performant. Une liste chaînée a habituellement une complexité de recherche de l'ordre de $O(n)$

Dictionnaire :

Le dictionnaire est une structure de données performante et pertinente pour un projet de développement de jeu en ligne multijoueur. Dans notre cas, nous utiliserons cette structure de données pour stocker les constructeurs particuliers de chaque objet de type GameEntity qui sera produit pendant une partie. Cette implémentation est simple et efficace, car elle permet d'accéder rapidement aux constructeurs d'une entité en utilisant leur nom en tant que clé, ce qui est une opération en temps constant (complexité $O(1)$). De plus, l'utilisation d'un dictionnaire permet de gérer dynamiquement les composants d'une entité en ajoutant ou en supprimant des paires clé-valeur de manière simple, ce qui est important dans un environnement développement où les entités peuvent changer rapidement. Enfin, cette implémentation est facile à maintenir et à comprendre pour les développeurs, ce qui est important dans un projet de grande envergure impliquant de nombreux collaborateurs.

Arbre de comportement :

Cette structure de données permet de décrire les différentes actions possibles qu'un personnage peut effectuer dans le jeu, telles que le mouvement, l'attaque, la défense, etc., ainsi que les conditions nécessaires pour déclencher ces actions.

La structure hiérarchique de l'arbre de comportement permet de modéliser les relations entre les actions et les conditions de manière modulaire et facilement modifiable. Dans un contexte de conception de jeu multijoueur en ligne, il est important de pouvoir modéliser et gérer le comportement des personnages de manière efficace et flexible. L'arbre de comportement répond à ces besoins en permettant une gestion hiérarchique et modulaire des comportements des personnages.

Patrons de conception

Observateur :

Dans le contexte de notre jeu et de nos interfaces d'utilisateur, le patron de l'observateur sera grandement mis à contribution. Celui-ci consiste à permettre à un objet (le sujet) de notifier un ensemble d'autres objets (les observateurs) lorsqu'un événement se produit. Cela permet de maintenir la cohérence et la synchronisation de l'état du jeu ou de l'interface en temps réel. Au niveau du jeu, un joueur devra notifier le serveur (et ainsi les autres joueurs) lorsqu'il fait une action qui change l'état du jeu, par exemple, si le joueur effectue une attaque, il doit en rendre compte au serveur qui évaluera les conséquences de cette attaque et renverra la réponse à tous les joueurs. Dans cette optique, on peut affirmer que le serveur "observe" le joueur et que le joueur "observe" le serveur. Au niveau des interfaces, les observateurs sont utiles pour notifier les éléments de l'interface lorsqu'un événement se produit. Par exemple, si l'utilisateur appuie sur le bouton "ajouter un ami", l'interface doit réagir de la bonne façon et en temps réel. Puisque nous voulons faire un jeu en temps réel et une interface réactive, le patron de conception de l'observateur sera grandement utilisé et efficace.

Commande :

Le patron de conception commande est utile pour un jeu de combat en temps réel, car il permet d'encapsuler les actions possibles d'un joueur dans un objet de commande, qui peut être ensuite transmis à un gestionnaire de commandes. Ce dernier peut ensuite exécuter la commande, de manière flexible et modulable et effectuer les actions nécessaires comme déplacer le personnage, attaquer, modifier l'état du jeu, envoyer une demande à l'API, etc. Ce patron de conception permet également de mettre en place des commandes complexes, qui peuvent être effectuées en plusieurs étapes. Par exemple, une commande pour effectuer une attaque peut inclure plusieurs étapes telles que la détection de collision avec la cible, le calcul des dégâts et l'affichage des animations. Nous allons donc bénéficier de l'encapsulation des commandes dans des objets pour rendre plus facile la gestion des différentes étapes.

Itérateur :

L'utilisation d'un itérateur est bénéfique dans le contexte d'un jeu en ligne en temps réel, parce que les données peuvent changer fréquemment et il est souvent nécessaire de parcourir de manière itérative une grande quantité de données pour effectuer des actions telles que la mise à jour de l'état du jeu, la détection de collisions, etc. Le patron de conception itérateur permet de parcourir une collection d'objets de manière itérative sans exposer les détails de la structure de la collection. En encapsulant le parcours de ces données dans un objet iterator, le code devient plus modulaire et facile à maintenir. Nous utiliserons un itérateur avec notre liste chaînée double pour simplifier le parcours des éléments à l'intérieur de celle-ci. En effet, contrairement à un tableau, les éléments d'une liste chaînée ne sont pas stockés de manière contiguë en mémoire, ce qui peut rendre leur parcours plus complexe.

Expression régulière

Nous allons utiliser une expression régulière pour nous assurer que l'email utilisé par un client est de forme valide. Voici l'expression en question : `/^\S+@\S+\.\S+$/`

Nous allons également utiliser un regex pour obliger l'utilisateur à utiliser un nombre à quatre chiffres pour son numéro d'utilisateur. Voici un exemple : `/^([0-9]{4})$/`

Algorithme

Arbre de comportement :

Un arbre de comportement est un élément potentiellement crucial dans la programmation d'une intelligence artificielle pour un jeu vidéo, car il permet de simuler le comportement des personnages de manière réaliste. L'arbre de comportement peut être utilisé pour modéliser des comportements simples, comme la marche et la course, mais aussi des comportements plus complexes, comme l'interaction avec des objets ou des personnages non jouables.

L'arbre de comportement est souvent divisé en plusieurs niveaux hiérarchiques, avec des comportements généraux en haut de l'arbre et des comportements plus spécifiques en bas de l'arbre. Les nœuds de l'arbre représentent des comportements spécifiques que le personnage peut adopter, tandis que les branches de l'arbre représentent les conditions qui doivent être remplies pour que le personnage adopte ces comportements.

A Star :

L'algorithme A star est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés.¹ Celui-ci est un choix courant pour la recherche de chemin dans les jeux vidéo, car il est rapide et efficace. Il utilise une estimation heuristique de la distance restante à parcourir pour atteindre la destination, ainsi que la distance parcourue jusqu'à présent pour guider la recherche de chemin. Cela permet à l'algorithme de rechercher

¹[https://fr.wikipedia.org/wiki/Algorithme_A²](https://fr.wikipedia.org/wiki/Algorithme_A%20%22)

efficacement la carte de jeu, en évitant les obstacles et les zones dangereuses, tout en trouvant le chemin le plus court possible.

Dans notre cas, l'algorithme A* est particulièrement utile pour la mise en place de l'intelligence artificielle de notre jeu. L'IA pourra trouver le chemin optimal le plus sûr et le plus court, se rendant par le fait même plus réaliste et plus compétent, ce qui pourrait améliorer l'expérience de jeu pour les joueurs.

En utilisant un arbre quadripartite (quadtree) pour stocker les positions des entités de jeu sur la carte, nous pouvons améliorer les performances du A*. En combinant le A* et le quadtree, les performances peuvent être grandement améliorées puisque le quadtree pourra éliminer rapidement les régions de la carte qui ne sont pas pertinentes pour la recherche de chemin.

Mathématique

Nous utiliserons des mathématiques pour le calcul du MMR (Match Making Rate) de nos joueurs. La formule ne sera pas trop complexe qui ressemble à ceci :

$$\text{Nouveau MMR} = \text{Ancien MMR} + K * (1 + (\text{Score} - 0.5) * (2^{(-\text{Diff}/\text{MMRdiv})}))$$

Où :

- L'ancien MMR est le MMR précédent du joueur.
- K est un facteur de pondération qui détermine l'importance de chaque match (par exemple, pratique vs ranked vs tournoi).
- Score est le résultat du match (1 pour victoire et 0 pour défaite).
- Diff est la différence de MMR entre le joueur et son adversaire.
- MMRdiv est un facteur de division qui permet de contrôler la sensibilité de la formule à la différence de MMR. Plus MMRdiv est petit, plus la formule sera sensible à la différence de MMR.

Veille technologique : React

Nous effectuons une veille technologique sur React.js, une bibliothèque JavaScript largement utilisée pour créer des interfaces utilisateur interactives et réactives pour le web. Bien que nous créions un jeu en ligne, nous désirons également que nos pages web soient interactives, rapides et agréables à utiliser. Nous avons choisi de concevoir toutes nos pages en React, y compris la page de jeu qui sera une composante React affichant notre canevas de jeu. Voici pourquoi nous avons opté pour React pour la création et la maintenance de notre site web :

- Modularité : comme React utilise des composants pour organiser le code, il sera plus facile pour nous de gérer et de réutiliser notre code. Nous visons une application modulaire. Il est donc important d'avoir un code facile à organiser et à maintenir.
- Performance : React est conçu pour être rapide et réactif, ce qui améliorera l'expérience utilisateur sur nos pages. Nous prévoyons d'avoir plusieurs effets "réactifs" activés par les actions du client et les mises à jour automatiques de React rendront le tout plus dynamique et agréable.
- Compatibilité : React est très compatible avec les autres bibliothèques et outils et peut facilement être intégré avec d'autres outils populaires utilisés dans le développement de jeux en ligne, tels que, dans notre cas, Phaser3, Colyseus et Socket.io. Cela nous permettra de travailler plus efficacement avec ces outils pour créer notre jeu.

En somme, utiliser React pour la création et la maintenance de notre site web nous permettra d'avoir un code modulaire, une meilleure performance et une grande compatibilité avec d'autres outils.