Groupe 1-b

Université de Strasbourg

carcaspace

AMEGAH Godwin
DISTEL Louis
EL-CHAMAA Omar
FROEHLY Martin
GANZITTI Deborah
GUNES Anthony
HABAS Achraf
HAOUD Anass
HENRY Nathan
KACHOUH Marc
KILANGALANGA Michel
KOCHER Florentin
LEFEVRE Matthieu
LEGLAND Guillem

2021/2022

Sommaire

Introduction

Ce projet a été fait dans le cadre du cours “Projet Intégrateur” à l’Université de Strasbourg dans l’année scolaire 2021/2022, dernier semestre de notre licence d’informatique. Nous sommes 14 étudiants à avoir travaillé ensemble à sa réalisation. Ce projet a pour but de recréer entièrement un jeu de société en mode plateau sur une application téléchargeable sur ordinateur ou via un accès internet. Nous avons choisi le jeu Carcassonne qui peut se jouer de deux à cinq joueurs. Ayant la possibilité de changer le thème graphique du jeu, nous avons opté pour un univers spatial au détriment de l’univers de base qui est médiéval. Cela nous a semblé davantage attrayant pour nos futurs consommateurs. Ainsi, nous avons implémenté tous les aspects de ce jeu en partant de l’interface utilisateur, en passant par le réseau, le désign et bien sûr tous les algorithmes sous-jacents au bon fonctionnement du jeu. Tous ces aspects seront expliqués, détaillés et accompagnés parfois de schémas dans la suite de ce rapport.
Tout d’abord, ce rapport commence avec une brève présentation du jeu pour se familiariser avec notre projet. Ensuite, une description technique de notre travail est présentée en commençant par l’architecture générale puis, une justification technique des outils que nous avons utilisés. A la suite de cela, nous entrerons dans le vif du sujet en détaillant tous les points techniques et algorithmiques de notre projet. Ensuite, nous passerons à une description complète de notre fonctionnement en mode projet. Et pour finir, nous avons chacun écrit un court descriptif de notre implication et de notre ressenti au sein de ce projet.

Présentation du jeu

Carcassonne est un jeu de plateau pouvant se jouer de deux à cinq joueurs. Il a été conçu par Klaus-Jürgen Wrede (2000, éditeur Hans im Glück, Allemagne). C'est un jeu de construction de paysage médiéval. Depuis sa sortie, le jeu se réinvente avec des versions alternatives proposant une expérience différente et des règles similaires. Nous nous sommes intéressés au jeu de base.

Ce jeu dispose d'un plateau et de différentes tuiles qui sont des petits cartons carrés agrémentés de dessin pour pouvoir les distinguer. Aussi, il est constitué de pions de couleurs qui sont attribués à chaque joueur au début de la partie.

Au cours d'une partie, les joueurs posent des tuiles piochées aléatoirement tour à tour. Ces tuiles forment des zones avec des routes, des villes et des abbayes. Le joueur peut poser son pion pour marquer des points. Ces points sont comptés pendant la partie quand une zone est close, ou à la fin de la partie si les dernières zones ne sont pas complétées. Une zone close se traduit par la fermeture d'un chemin ou lorsqu'une ville est encerclée de remparts.

Le but du jeu est de finir la partie avec plus de points que les autres joueurs pour ainsi la gagner.

Image du jeu traditionnel:

https://www.philibertnet.com/fr/zman-games/10544-carcassonne-vf-8435407619968.html

Architecture Générale

Le projet a été codé majoritairement grâce au moteur de jeu Unity et au langage de programmation C#. Ce langage de programmation est le langage par défaut de Unity. Grâce à une API, on peut interagir avec les différents objets et fonctionnalités de Unity qui nous permettent, par exemple, de créer un cube, de le transformer (position, rotation, taille, etc), de changer de scène ou encore de modifier un texte dans une scène ; tout cela géré par un script C# (et donc pas “à la main”).

Lors de l’ouverture de notre projet, nous arrivons sur une première scène Unity que nous allons appeler “MainMenu”. C’est une page d'accueil qui affiche simplement le logo de notre programme. En cliquant sur ce logo, une nouvelle scène Unity apparaît et permet de se connecter à notre compte ou de s’inscrire. Également, si nous n’avons pas de compte, alors le bouton pour en créer un va nous ramener directement à une scène contenant des entrées pour récupérer les données d’un nouveau compte. Ces données sont enregistrées dans une base de données distante sur le serveur. Ainsi, lors de la connexion ou de l’inscription, le client envoie une requête au serveur et récupère les informations utiles pour la connexion ou l'inscription.

Une fois l’identification finie, l’utilisateur est amené sur la scène “Join“. La scène contient un bouton qui l’emmène vers la scène “Profile”, où il pourra voir ses statistiques (parties gagnées et jouées). Un autre bouton sur cette même scène lui permet de revenir vers la scène précédente. Le bouton “Join” permet de se connecter au *build* sur la VM0, et grâce à *Mirror*, une fois le client connecté, il sera amené sur la scène “Lobby”.

Cette scène représente une salle d’attente pour tous les joueurs ayant lancé le programme. Dès qu’un joueur arrive sur cette scène, une carte représentant le joueur s’affiche parmi d’autres dans une liste. Cette dernière permet d’identifier les autres joueurs ayant lancé le programme et se retrouvant au même endroit. Sur ces cartes figure le nom du joueur ainsi que son statut pour jouer. Chaque carte présente un bouton “Prêt” cliquable uniquement par le joueur assigné à la carte et qui permet d'indiquer si le joueur est prêt à rejoindre la partie ou non. La partie se lance uniquement lorsque tous les joueurs présents dans *lobby* ont indiqué être prêts à jouer.

Vient ensuite la scène principale du projet, la plus importante, appelée “Game Interface”. Cette scène est le cœur du jeu. Elle permet de faire les interactions du jeu de société numériquement et dans un espace 3D. Sans rien toucher, on peut déjà voir apparaître sur cette scène une interface montrant une petite carte en haut à gauche avec en bas une barre de temps qui décrémente et change de couleur au fil du temps. Du côté droit, on aperçoit un carré qui représente l’arrière d’une tuile. Il s’agit d’un bouton qui actionne le tirage au sort

d’une tuile du jeu. Ensuite, le joueur pourra placer sa tuile sur le plateau en respectant les règles du jeu et son tour sera fini.

Deux caméras sont attachées à cette scène. Cela permet de voir le plateau de jeu de deux manières différentes. La première caméra est légèrement inclinée donnant une perspective aux tuiles et au pions et donc une véritable sensation de 3D. La seconde caméra est fixe et apporte une vue du dessus. C’est d’ailleurs la caméra qui se situe en haut à gauche de l’interface ; en cliquant dessus, elle prend la place de la première caméra.

Pour faire respecter les règles du jeu, nous avons implémenté une multitude d’algorithmes et de contraintes. Tout d’abord, le comptage des points s’effectue à chaque tour, après la pose d’une tuile par un joueur. Trois algorithmes différents se lanceront : lorsqu’une ville se ferme, lorsqu’un chemin est clos et lorsqu’une tuile contenant une abbaye est entourée de 8 autres tuiles.

Lorsque la tuile posée par le joueur contient une ville, un algorithme itératif se lance qui parcourt toutes les villes connectées directement avec elle. Lorsque l’algorithme détecte que la tuile en cours de parcours possède une composante “Ville” sur l’un de ses côtés et qu’aucune tuile n’y est encore connectée, la ville dans son ensemble est alors considérée comme non fermée. Dans le cas contraire, une ville est complète et les points sont distribués.

L’algorithme des chemins fonctionne différemment : il crée des structures “Roads” qui forment un ensemble de tuiles connectées par des composantes “chemins”. Ces structures servent à détecter si des chemins sont à fusionner ou clos. Aussi, elles servent à compter plus facilement les points lors de fermeture des chemins.

L’algorithme des abbayes est le plus simple. Lorsqu’une tuile “abbaye” est posée, celle-ci est ajoutée à une liste d’abbayes. Par conséquent, à chaque pose d’une tuile, un autre algorithme se lance et vérifie si chaque tuile abbaye possède un total de 8 voisins autour d’elle.

L’annexe 1 est un *wireframe* avant de débuter une partie. **L’annexe 2** est un *wireframe* de la partie.

Justification des Choix Techniques

Unity et Mirror

1) Unity

D'après les conseils de notre tuteur, nous avons fait le développement de notre jeu de société sur Unity. Il s'agit d'un moteur de jeu très répandu sur le marché des jeux vidéo. Par exemple, le célèbre jeu Genshin Impact a été développé sur ce moteur-là. Nous voulions faire un jeu en 3D pour permettre un design plus adapté au jeu Carcassonne que de la simple 2D. Unity nous a permis de le faire.

Unity a surtout comme avantage d'être facile d'accès. Ce qui signifie qu'énormément de documentations, de tutoriels et de forums sont disponibles sur internet. Ce moteur de jeu nous a permis, grâce à plusieurs *packages*, de relier une BDD ou de mettre un *build* de notre jeu sur un serveur et de le faire *loger* dessus. Toutes ces fonctionnalités étaient obligatoires pour la réalisation de notre projet et étaient présentes sur ce moteur de jeu.

Unity est aussi supporté par presque toutes les plateformes, dont le web et le mobile, ce qui était aussi requis pour le projet. Une autre fonctionnalité très importante pour un jeu vidéo est justement la capacité à jouer en mode multijoueur. De ce fait, Unity offre cette souplesse grâce à ces *packages* pouvant assurer les communications entre les clients (joueurs) et un serveur.

Unity permet aussi la création d'interfaces ainsi que la création de scènes et la disposition de celles-ci par des transitions. Ceci nous a permis donc de créer une page d'authentification, un lobby de jeu avant qu'une partie ne commence ainsi que l'affichage final de notre jeu (affichage du plateau, interaction avec les tuiles, affichage des points de chaque joueur). Tout cela pour garantir une interface ergonomique et intuitive pour satisfaire au mieux les utilisateurs.

D'autre part, Unity nous a facilité le développement car il est possible d'y lier un IDE de type Vscode utilisé dans la majorité des projets de développement logiciels.

2) Mirror

Dans le cahier des charges, nous avons mentionné que nous allions utiliser *MLAPI* ; une solution de *high level networking* proposée par Unity. Le problème que nous avons rapidement rencontré a été que cette API est encore expérimentale et en cours de développement. De plus, nous n'avions pas énormément de ressources qui pouvaient nous aider dans notre développement. En cherchant davantage, nous avons trouvé que *Mirror* était recommandé par beaucoup de développeurs comme introduction au *networking* multijoueur, ce

qui était parfait pour nous. En effet, *Mirror* contient beaucoup d'exemples concrets intégrés dans le *package* qui nous ont beaucoup aidés.

Mirror nous a permis d'ajouter par exemple, des "SyncVar", variables qui seront synchronisées chez tous les joueurs, des "Network Identities", composant qui s'ajoutent sur un objet *In Game* pour le *Spawn* chez tous les joueurs, ou même encore des "Networkmanager", qui permettent de choisir le protocole de transport (que l'on veut), le port d'écoute et l'adresse IP sur laquelle il faudra se connecter.

Tous ces outils et fonctionnalités nous ont permis d'avoir des communications clients-serveur solides et efficaces.

Authentification

Lors de la mise en place de l'authentification, nous avons choisi de créer un serveur web sous Node.js étant donné les facilités de mise en place et le niveau d'abstraction qu'il offre dans la facilité d'apprentissage du langage. Aussi, les bibliothèques associées et la quantité de documentations et de ressources concernant cette technologie a rendu bien plus aisé la mise en place du serveur web. Express.js a beaucoup de documentations en ligne et permet de créer une application web ainsi que d'y associer des routes de manière triviale.

De plus, nous avons choisi de faire une base de données NoSQL avec MongoDB car l'intégration est facilitée par Node.js grâce à la bibliothèque "Mongoose".

Ainsi, l'utilisation de bibliothèques tierces telles que "argon2-ffi" permet une mise en place simple de hachage de mot de passe avec "crypto". Cela permet aussi de créer des hash de 32 bits de longueur avec un *sel* associé non-prédéfini à l'avance. "argon2" a l'utilité d'inclure le *sel* dans le hachage du mot de passe afin de le réutiliser lors de la vérification de compte, et donc d'éviter des valeurs par défaut nous prédisposant à des attaques telles que du brute force.

TCP

Nous avons fait le choix de TCP comme protocole de transport de nos informations car elle est l'une des plus classiques et simple à implémenter, tout en étant tout de même sécurisée. Nous ne voulions pas UDP pour d'évidentes raisons de sécurisation des paquets. L'autre choix qui s'offrait à nous était KCP implémenté par Unity pour son fonctionnement avec Mirror mais nous avons fait le choix de ne pas le prendre pour éviter des problèmes de compatibilité.

En d'autres termes, nous avions plus de gains que de pertes à utiliser TCP que KCP, car nous ne connaissions pas non plus son comportement avec les machines virtuelles et les retransmissions par le Bastion.

Description des Modules

Algorithmes et Fonctionnalités

1) Villes

La résolution des villes se fait de façon itérative, l'idée générale derrière l'algorithme est de voir la ville comme un labyrinthe, s'il existe une sortie, la ville n'est donc pas fermée, dans le cas où il est impossible de sortir de la ville, la ville est fermée.

L'algorithme vérifie les éléments présents sur la tuile : haut, bas, gauche et droite. S'il existe une zone ville sur la tuile il doit y avoir dans la même direction un voisin possédant lui-même une zone tuile pointant vers la tuile. A cette étape, il faut relancer le programme avec ledit voisin. Logiquement, si la zone tuile pointe vers une zone ou aucune tuile n'est placée, la ville n'est donc pas fermée. Il suffit de répéter l'opération jusqu'à avoir parcouru toutes les tuiles. Si nous avons parcouru toutes les tuiles sans qu'aucune tuile ne pointe vers une zone vide alors la ville est fermée.

Une fonction pour vérifier si la tuile posée n'est pas une tuile spéciale est nécessaire. En effet, les tuiles, dites spéciales, sont les tuiles "10" et "15" (voir figure 1) car elles possèdent deux éléments de villes non connectées. Or, ces deux éléments peuvent faire partie de deux villes différentes, il faut donc lancer l'algorithme de vérification deux fois : une fois par morceau de ville. Pour vérifier si une tuile "ville" est spéciale, nous vérifions si elle possède un "10" ou un "15" dans son nom puis, nous testons tous les cas possibles en fonction de comment elle a été placée. Enfin, nous lançons l'algorithme sur les deux voisins connectés par les villes de ladite tuile.

2) Chemins

L'algorithme traitant la clôture des chemins est séparé en deux algorithmes. Un premier créant et modifiant ce qu'on appelle des *Roads* (voir dernier paragraphe de l'architecture général), qui représentent un ensemble de tuiles ayant des composantes "chemins" connectées les une aux autres. Un second algorithme parcourt l'ensemble des *Roads* actuellement sur le plateau et distribue les points aux joueurs ayant un pion posé sur la *Road*. Suite à cela, la *Road* est supprimée car elle ne sera plus modifiée.

Le premier algorithme implémenté via la méthode "roadIsClosed_Struct()" utilise une approche basée sur un stockage d'informations pour les différents chemins parcourus. L'idée est simple ; à chaque fois qu'un joueur pose une tuile, l'état du plateau est observé afin de récupérer des informations telles que les

chemins en cours de construction et les chemins fermés. Pour celà, *Roads* conserve l'ensemble des tuiles connectées les unes aux autres (il s'agit d'un chemin). Cet ensemble est représenté sous forme d'une liste de *GameObject* (les tuiles) contenu dans chaque structure. La liste est donc construite en ajoutant successivement les tuiles ayant une composante "chemin" et qui relient d'autres tuiles ayant la même composante.

Lorsqu'une tuile posée par un joueur est isolée de tout chemin, elle forme alors une structure (*Road*) dont la liste est formée d'un seul élément (elle-même). Cependant, lorsqu'une tuile "A" est déjà posée et que l'on veut la lier avec une tuile "B", l'algorithme détecte que la tuile "A" appartient déjà à une *Road*. Plutôt que d'en créer une nouvelle, la tuile "B" est ajoutée à la *Road* de la tuile "A". Un autre cas de figure possible est l'existence de deux *Roads* indépendantes en cours de construction. Lorsqu'un joueur les relie, l'algorithme va alors détecter les deux voisins autour de la tuile courante, puis fusionner les deux *Roads* en une seule.

Pour détecter efficacement qu'un chemin est clos et donc attribuer des points à un joueur, nous avons besoin de connaître l'ensemble des chemins actuellement sur le plateau ainsi que leur état (clos, en cours de construction etc...). La liste de *Roads* "listeOfStructRoad" permet de garder une trace des différents chemins sur le plateau. Le deuxième algorithme vérifie à chaque tour si une des *Roads* sur le plateau est close. Une tuile est considérée comme étant fermante si elle débute ou finit un chemin. De plus, les carrefours sont considérés eux-aussi comme étant fermants. Par conséquent, lorsqu'une intersection est posée, un nombre de structures égal au nombre de chemins que l'intersection va former, est créé. Une fois une *Road* close, la liste complète des tuiles qui la forme est récupérée. Puis, les pions posés sur les tuiles sont récupérés et les points sont calculés et attribués aux joueurs correspondants. Suite à cela,les *Roads* fermées sont supprimées puisque celles-ci ne seront plus accessibles, donc inutiles durant le reste de la partie.

3) Abbayes

L'algorithme pour la clôture des abbayes n'a pas été difficile à réaliser. Il a surtout fallu choisir le meilleur moyen pour vérifier les 8 tuiles autour de la tuile abbaye. Nous avons donc opté pour l'utilisation des coordonnées de la tuile abbaye.

Nous avons d'abord créé une liste pour pouvoir utiliser les tuiles plus facilement. A chaque fois qu'un joueur pioche une abbaye et la pose, elle sera donc ajoutée à cette liste. La fonction principale "abbeyLaid" va ainsi compter le nombre de tuiles autour de la tuile posée en parcourant tout le plateau. A l'initialisation, il y a un compteur et si le compteur est à 8, alors la zone abbaye est fermée. Cette fonction est appelée dans "abbeyIsClosed" pour tester toutes les abbayes du plateau déjà posées.

A chaque tour, on vérifie si la zone d'une abbaye de la liste a pu être complétée. Si c'est le cas, alors le joueur pourra récupérer son pion ainsi que les points associés.

4) Gestion des Pions

Lorsqu'un joueur pose la tuile piochée, des étoiles apparaissent pour lui indiquer les emplacements où il a le droit de poser son pion, c'est-à-dire soit sur une ville, un chemin ou une abbaye. Au moment où le pion est posé, les autres étoiles disparaissent pour empêcher la pose d'un autre pion. Ces pions seront utilisés pour compter les points du joueur pendant et à la fin de la partie.

Pour identifier quel pion est à quel joueur, nous avons créé une classe "Player" avec comme variable son identifiant (id), son nombre de points ainsi qu'un compteur du nombre de pions qu'il lui reste en jeu.

Durant la partie, les joueurs ne pourront plus toucher les pions posés sauf lorsqu'une zone se ferme (ville ou abbaye complétée ou chemin fermé) ; l'algorithme pour compter les points sera alors appelé. Le ou les pions sur la zone seront donc supprimés du plateau avec la fonction "rmMeeple". Le compteur de chaque joueur avec les pions présents sur cette zone sera incrémenté du nombre de pions posés.

5) Représentation des tuiles

Pour la représentation des tuiles, nous avons pris une image sur internet de toutes les tuiles pour avoir un ordre définitif afin de pouvoir faire correspondre le design des tuiles avec leur script plus facilement. Cela nous a aussi beaucoup aidé, lors de nos réunions, pour identifier plus facilement de quelle tuile nous parlions.

Figure 1 : illustration des tuiles du jeu traditionnel

A partir de la figure 1, nous avons créé une classe abstraite “type_tile” dans un script du même nom. Dans ses classes filles, nous avons défini les 24 types de tuile ainsi qu’une tuile “type_tile_0” qui est la première tuile piochée et qui a les mêmes caractéristiques que la 20.

Au début, nous avons simplement défini une énumération "Type_land" afin de définir “Ville”, “Plaine”, “Chemin”, “Abbaye” et “Continue” comme étant des valeurs. “Continue” est utilisé pour plus de facilité pour les scripts lorsque plusieurs côtés d'une tuile sont du même type et qu'ils sont connectés. Par exemple, la tuile 8 à son milieu continue car les côtés haut et gauche ont le même type.

Nous avons par la suite déclaré 5 variables de cette énumération qui sont haut, gauche, bas, droite et milieu. Cela permet de décrire les tuiles. Il est possible de les faire pivoter en cliquant sur la flèche gauche ou droite du clavier. Pour ce faire, on s’est contenté de modifier ces valeurs en les échangeant.

Par la suite, nous avons ajouté 3 variables de type booléen :

- La variable “blason” qui permet de savoir si une tuile de “type_land” ville a un blason. Cette variable est essentiellement utilisée pour compter les points,
- La variable “carrefour” pour représenter les croisements comme sur la tuile 6, 7, 17, 21 et 22. Cette variable est utilisée pour l’algorithme de fermeture de chemins,
- La variable “estFermante” pour la tuile qui ouvre ou ferme un chemin. Cette variable est aussi utilisée pour l’algorithme de fermeture de chemins.

6) Comptage des points

L'algorithme de comptage des points a été codé en utilisant l'algorithme “townIsClosed” de Martin. Dans cet algorithme, comme dans “townsIsClosed”, nous commençons par récupérer les positions de la tuile actuelle (celle qui vient d’être posée) et une liste des voisins de la tuile est créée (nord, sud, est, ouest) dont les informations sont récupérées par la liste “plateau”.

Ensuite, nous faisons les mêmes conditions que dans “townIsClosed” c'est-à-dire que l’on vérifie pour chaque côté de la tuile s’il y a une ville. Pour chaque côté, nous vérifions si le voisin directement concerné est une des tuiles spéciales. Si c’est le cas, alors nous incrémentons la variable *p* qui permet de compter une tuile spéciale sans faire le parcours de ses voisins. Sinon, nous vérifions si le voisin n'est pas *null* et n'est pas visité. Dans ce cas, nous donnons au voisin le statut de “visité” et nous lançons une récursion sur le voisin en additionnant le résultat avec la variable *p*. Dans tous les autres cas, nous ne faisons rien et nous passons à la condition sur le côté suivant de la tuile.

A la fin de cet algorithme, nous retournons *p+1* car *p* est le score calculé et celui de la première tuile. Nous avons modifié aussi l'algorithme “drawshit” pour lancer directement le comptage des points. Pour chaque condition sur les tuiles spéciales, nous vérifions si le voisin du côté concerné complète une ville. Dans ce

cas, nous lançons l'algorithme “score” sur le “voisin + 1” pour la tuile actuelle. Si la tuile n'est pas spéciale alors nous vérifions si la tuile actuelle complète une ville : si c’est le cas, l'algorithme “score” est lancé sur ladite tuile.

Nous avons aussi modifié “abbeyIsClosed”. Nous vérifions si chaque abbaye de la liste des abbayes est complète. Si c’est le cas, nous comptons les 9 points puis l’abbaye est supprimée de la liste. Aussi, nous avons modifié légèrement l’algorithme “abbeyLaid” ; nous renvoyons *true* si l'abbaye est complète, *false* sinon.

7) Contraintes

Le scripts “constraints” a pour but de vérifier s’il est autorisé, d'après les règles du jeu, de poser une tuile à l'endroit où le joueur clique. La fonction principale pour la réalisation de cette tâche est “verif”. Elle est appelée au moment de la pose par le script “move”. Cette méthode prend en argument les valeurs de la tuile et retourne un booléen. Si la pose n’est pas possible, alors une animation annonce au joueur qu’il ne peut pas poser la tuile à cet emplacement, sinon, elle est placée sur la grille.

La fonction “verif” prend en argument les caractéristiques de la tuile à poser. Pour se faire il y a plusieurs conditions:

- La tuile doit être adjacente à une autre tuile. Il y a cependant une exception pour la première tuile piochée qui doit pouvoir être posée sans respecter cette condition.
- La tuile doit être posée sur une case existante. Nous avons des objets construits selon un *prefab* qui crée une grille. Ces objets ont pour nom “x/y” (avec x et y des nombres entiers correspondant aux coordonnées de la case).
- La case sur laquelle on souhaite poser la tuile doit être vide.
- Il doit y avoir des correspondances de valeurs de direction. Par exemple, si une tuile a “chemin” en valeur gauche alors, à sa gauche se trouvera une tuile ayant pour valeur de droite “chemin”. Cependant, il est nécessaire d'accéder aux voisins de la tuile. Pour ce faire, nous accédons aux valeurs “haut”, “bas”, “gauche” et “droite” non pas de la tuile voisine, difficile à trouver, mais de la case de la grille correspondante. Il est facile de retrouver les cases voisines puisque leurs noms correspondent à leurs coordonnées. Intrinsèquement, il est nécessaire de stocker les valeurs de la tuile dans la case une fois une tuile posée.

Intéractions Hommes-Machines

1) 3D in Game

La mécanique du jeu se déroule essentiellement dans un univers 3D. Cet univers 3D est très utile pour représenter des tuiles avec de la perspective. Ainsi, les tuiles sont représentées par des objets 3D sous forme de cube natif à Unity avec un matériau sur chaque face de ce cube. Ce matériau permet d'afficher une image sur une des faces ou de modifier la couleur, la brillance et les ombres de cette face.

De ce fait, nous avons créé un nombre prédéfini de cubes 3D dans Unity et le bon cube apparaît selon l'algorithme de tirage au sort du groupe Back. Pour piocher au hasard, l'utilisateur clique sur un bouton en haut à gauche qui va instancier la tuile tirée au sort et la retourner sur la face dessinée. Une fois retournée et dévoilée, la tuile se “colle” directement à la souris pour que l'utilisateur puisse indiquer où il veut la poser. Pour la poser, l'utilisateur doit simplement cliquer sur une des cases de la grille et la tuile va automatiquement se “coller” à la case de la grille la plus proche (comme on ne pointe jamais des coordonnées précises, ainsi la tuile sait à peu près où l'utilisateur veut la poser et s'aligne sur la zone la plus proche). En recliquant, la tuile sort de la case et peut être posée à un nouvel endroit.

Enfin, l'utilisateur peut également retourner la tuile pour lui faire changer de sens et donc changer son angle (±90°). Toutes les actions menées par l'utilisateur (comme la pose, la rotation, etc.) sont animées, beaucoup d'entre elles, avec une fonction "Lerp" qui permet de faire la transition d'un état A à un état B tout en respectant une limite de temps qui module le changement entre les deux états. Ainsi, nous pouvons définir à quelle vitesse et pendant combien de temps nous transitons entre deux états définis.

2) Bouton - Interface - Caméra

L'interface dans le jeu est très minimaliste et simple d'utilisation. Elle comporte en tout 3 boutons principaux :

- Le premier nommé "Pick a Tile". C’est un bouton de même couleur que l’arrière d’une tuile déclenchant le tirage au sort d'une tuile avec des algorithmes développés par le groupe Back. Une fois le bouton actionné, la tuile se crée juste derrière le bouton et ce dernier devient invisible. Cela fait croire à l'utilisateur que le bouton n'en est pas vraiment un. Ce bouton n'apparaît que lorsque c'est le tour du bon joueur.
- Le deuxième bouton est un bouton de validation. Il apparaît une fois que la tuile est posée et permet à l'utilisateur de confirmer son choix et de terminer son tour. Ainsi, il disparaît une fois que l'utilisateur clique dessus.

- Le dernier bouton fait apparaître un menu déroulant à gauche de l'écran qui permet d'accéder aux paramètres et de quitter le jeu. Il s'agit simplement d'un canevas qu'on déplace au clic.

Le reste de l'interface est composé d'une *minimap* en haut à gauche de l'écran. Il s'agit d'une *rawimage*, c'est-à-dire un objet UI dynamique qui permet de modifier son contenu en temps réel et par exemple, de charger des textures ou des vidéos : ici ce sera une texture. Cette texture provient d'une caméra (autre que la principale) qui est assignée à l'élément UI. Cette caméra apporte une vue du plateau par le dessus. Pour finir avec l'interface, nous avons mis en place une barre qui représente un compte à rebours pour informer le joueur du temps qui lui reste pour finir son tour. Cette barre se réduit et change de couleur : il y a une transition du vert jusqu'au rouge. Il s'agit simplement d'un compteur dans un script qui incrémente la *frame* avec une valeur "Time.time" dans l'API Unity et qui nous permet de connaître le temps passé depuis la première incrémentation.

Pour la gestion de la caméra, nous avons décidé de faire les déplacements horizontal et vertical via la souris et les touches du clavier. La souris est plus intuitive que les touches mais plus compliquée à gérer lorsqu'on a une pièce en mains et que l'on veut la tourner ou la placer. Pour compléter le déplacement à la souris, nous avons donc décidé de rajouter une option pour déplacer la caméra avec les touches du clavier. Nous avons décidé de prendre les touches "zqsd" et non la croix directionnelle pour une question d'ergonomie. En effet, il n'est pas agréable d'utiliser la souris et en même temps d'avoir la main gauche du côté droit du clavier. Nous avons opté pour mettre les touches vers le côté gauche, ce qui est bien plus agréable lors des sessions de jeu.

Il est possible de zoomer et dé-zoomer la caméra via la molette de la souris ce qui permet, lors d'une partie, de prendre de la hauteur sur le plateau et donc d'avoir la possibilité de mieux analyser ce dernier.

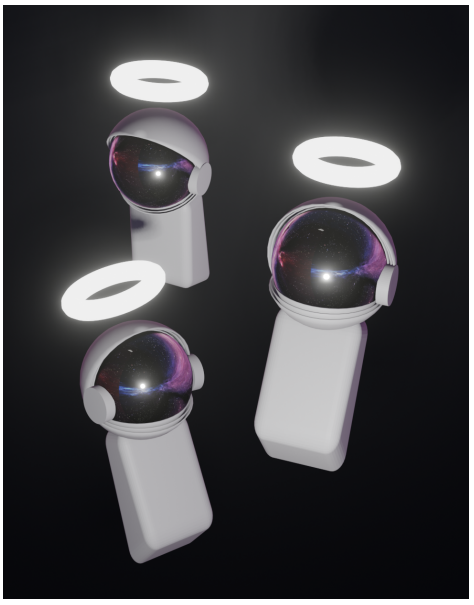
Enfin une *minimap* qui est une vue du dessus du plateau est disponible pour aider le joueur à voir les différents éléments du plateau. Si le joueur le souhaite, il peut cliquer sur la *minimap* pour que la caméra principale prenne l'angle de la *minimap*, c'est-à-dire une vue de dessus.

3) Pions

Les pions peuvent être posés une fois que la tuile à été posée. Cependant, un pion ne peut pas être posé n'importe où. Nous avons ainsi mis en place un moyen de notifier le joueur des endroits où il peut les placer. Nous avons décidé de représenter ces endroits par des étoiles qui clignotent. Ces étoiles apparaissent sur les tuiles pour indiquer au joueur les emplacements possibles de pose d'un pion. Pour les faire apparaître aux endroits possibles, il s'agit de faire passer un tableau de booléens statique de 5 valeurs en paramètre d'une fonction qui affecte une étoile sur la tuile précisée en paramètre. Ce tableau est organisé comme ceci : [haut, bas, gauche, droite, centre].

Lors du clic sur une de ces étoiles, un pion est créé à l'endroit où l'étoile a été cliquée. Finalement, un pion déjà présent dans l'interface mais invisible, est dupliqué pour être déplacé à l'endroit souhaité. Ces pions sont modélisés grâce à une succession d'objets de modèles 3D ; à savoir un anneau, une sphère, et des quadrilatères. Une illustration est disponible en **figure 2**.

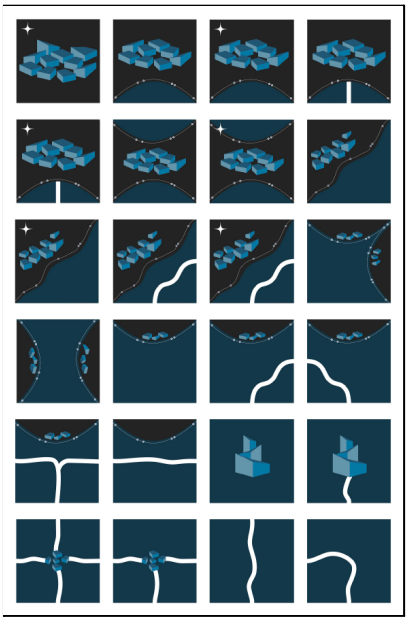
Figure 2 : illustration de nos pions



4) Tuiles

Nous avons donc dû changer le design des tuiles car, rappelons-le, nous faisons le jeu Carcassonne sur le thème de l'espace. Nos différentes tuiles sont représentés sur la **figure 3**:

Figure 3 : illustration de nos tuiles



Réseau

1) Multijoueur

Chaque joueur en multijoueur est synchronisé en temps réel avec les autres joueurs. Une fois la partie lancée, il est impossible de la rejoindre à nouveau. Ainsi, la partie peut être lancée entre tous les joueurs d'un lobby qui a une taille limitée à 5 joueurs. Cette synchronisation et la communication entre tous les joueurs se font par le biais de *Mirror*. Je vous invite à vous référer à cette partie pour plus d'informations à ce sujet.

Les communications en multijoueur sont faites au travers du port 11 000 du Bastion et sur le port 10 000 de la machine virtuelle ; nous n'en avions guère besoin de plus. La machine virtuelle 1-0 est donc celle qui accueille le serveur de jeu et le serveur web, et les informations pour les bases de données transitent vers la machine virtuelle 1-1. Cette architecture simple permet d'avoir la main sur tous les échanges et n'est pas un frein ou un problème lors des différentes étapes du projet.

De plus, Unity permet de créer une version serveur uniquement du jeu relativement facilement (plus que de faire nous-même un serveur asynchrone). Pour ce faire, nous pouvons faire un script ajoutant notre fonctionnalité de *build* serveur à l'interface Unity, ou alors utiliser l'autre méthode qui *build* depuis les options du projet. Nous avons choisi un script de *build* de serveur. Ceci nous permettait de choisir exactement ce que nous mettions dans le serveur et s'est avéré utile ayant beaucoup de parties uniquement graphiques. Ne pas éviter les parties graphiques aurait généré des erreurs inutiles sur un serveur ne pouvant les utiliser. Mais ce dernier n'apporte finalement aucun gain assez conséquent par rapport à son homologue, à savoir le *build* de serveur sans script. Nous avons alors fait le choix de *build* depuis les options du projet une version serveur uniquement, sécurisant par la même occasion le jeu ; ayant jugé le *peer to peer* comme une mauvaise solution.

Mis à part la difficulté de confection d'un script de *build* de serveur, nos choix rendent ce projet adaptable aux futures modifications car, comme précédemment mentionné, nous avons fait le choix d'utiliser des fonctionnalités existantes et modulables plutôt que les recréer nous même.

Voir **annexe 3** pour une illustration de ce mécanisme.

2) Lobby et Chat

Au début, nous voulions avoir la possibilité d'avoir plusieurs parties en même temps pour exploiter toute la capacité de nos VMS et de *Mirror*. Après de nombreuses tentatives et heures passées à *debugger*, nous étions très proche d'avoir une solution stable. Mais pour une raison que l'on ignore, nous n'avons pas

réussi à implémenter cette option donc nous avons un lobby qui ne supporte qu'une partie à la fois.

La solution proposée par la bibliothèque *Mirror* est le "NetworkRoomManager" qui nous permet de paramétrer le port et l'adresse sur lesquels nous voulons communiquer et combien de clients au maximum nous pouvons connecter au serveur simultanément. On initialise aussi un "player prefab" qui sera *spawn* au lancer d'une partie, une scène *offline* du joueur (la scène *join*), une scène *lobby* et enfin une scène de *gameplay*.

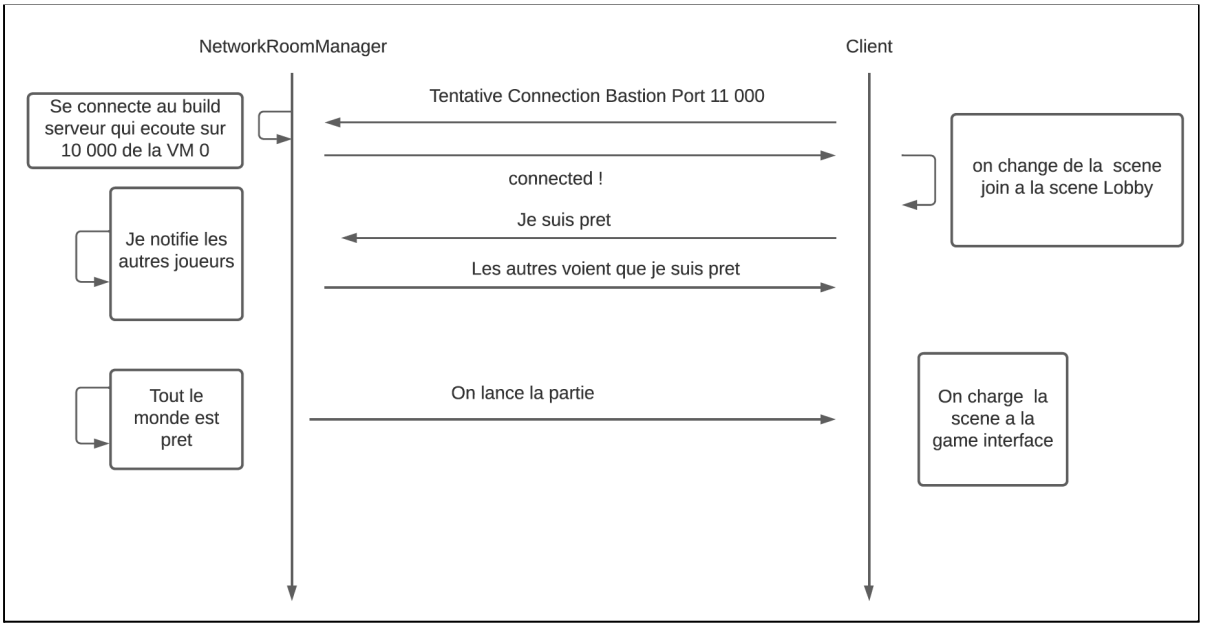
De plus, on spécifie le protocole de transport de données, qui sera dans notre cas, TCP. Une fois le client connecté au serveur, il sera alors amené sur la scène *lobby* où il sera en attente de tous les autres joueurs. Une fois tous les joueurs prêts, la partie est lancée et les joueurs sont emmenés vers la scène de *gameplay*. Le nombre minimum de joueurs pour lancer une partie est de deux et le maximum cinq (d'après les règles de Carcassonne).

Pour avoir un affichage des joueurs présent dans le lobby, la première chose qui a été créé était une grille composée de cinq divisions verticales afin d'accepter les cinq joueurs qui souhaitent intégrer la partie de jeu.

Le *prefab* de la carte est prédéfinie et à chaque nouvelle connexion, on *spawn* une carte en faisant attention à changer le *string* qui consiste à définir le nom du joueur. Chaque carte contient le nom de l'utilisateur et un bouton afin que l'utilisateur puisse définir son statut (prêt à jouer ou non).

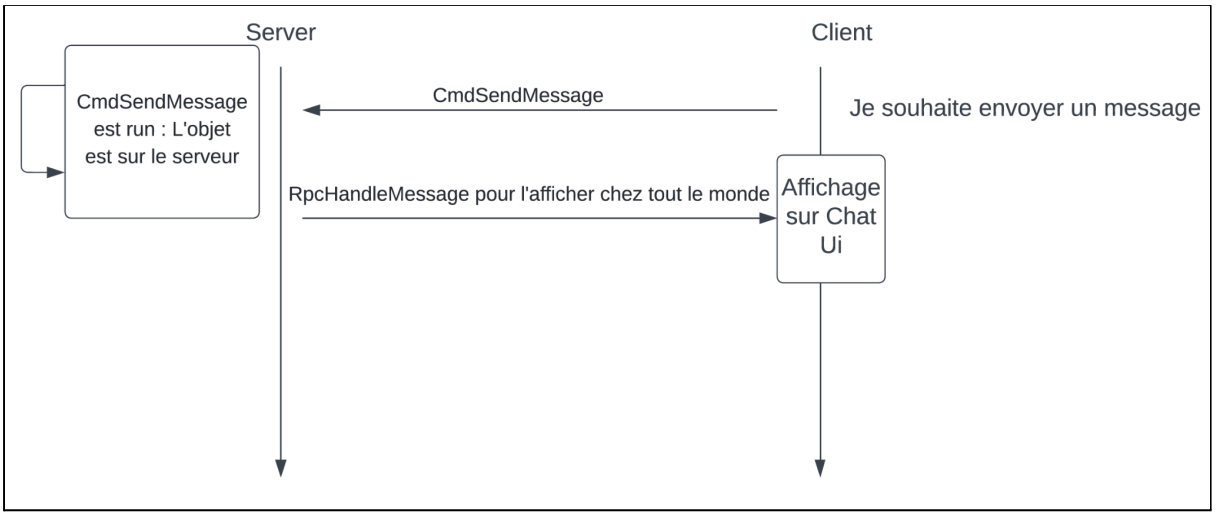
On détecte l'entrée d'un joueur grâce à la méthode "OnRoomClientEnter" du "NetworkRoomManager". Un schéma explicatif est mis à disposition ci-dessous :

Figure 4 : création d'une partie avec un lobby



Afin d'implémenter un *chat*, la première étape a été de créer son interface graphique. La seconde a été de créer le script “ChatUI.cs” qui récupère *l’input* de l’utilisateur et grâce à l’attribut *[Command]* de la fonction “CmdSendMessage”, le serveur récupère *l’input*, lui ajoute l'identité du *sender* et avec l’attribut *[ClientRpc]* de la fonction “‘RpcHandleMessage”, l’affiche chez tous les clients. Un schéma explicatif est mis à disposition ci-dessous :

Figure 5 : fonctionnement du chat



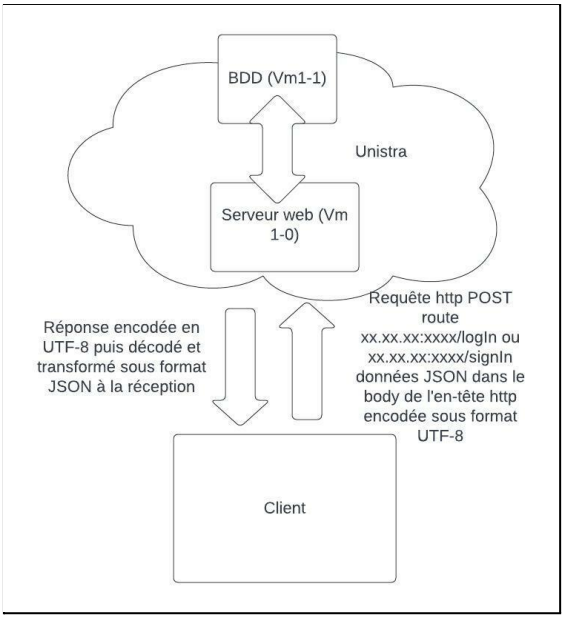
3) Authentification

L’authentification se scinde en 2 parties. Premièrement, la préparation des données à envoyer au préalable sous forme de formulaire depuis le client Unity qui effectuera des requêtes « http post ». En effet, la mise en place du formulaire sous Unity a été quelque peu laborieuse. Pendant toute la mise en place de l’authentification, il y avait soit une incompatibilité des objets mis en place sur l’interface homme-machine avec le type de formulaire mis en place, soit le type d’objet du formulaire n'effectuait pas correctement la connexion entre le client et le serveur. Unity propose différents types d’objets capables de communiquer avec un serveur à distance. Ayant initialement travaillé avec le type d’objet “WWW” qui est une méthode dépréciée, les quelques exemples disponibles sur internet nous ont permis de comprendre la logique mise en place par Unity. Ensuite, passant sur un type “WWW Form”, certains types d'objets mis en place avec ce type de formulaire ne pouvaient pas communiquer correctement, c’est-à-dire que certaines entrées n’étaient pas envoyées du tout ou correctement au serveur distant. C’est alors qu’en écumant la documentation officielle d’Unity, il existe l’objet *UnityWebRequest* ainsi que le type d’objet TMP pour l’interface homme-machine permettant de créer des types d’entrées. Cela nous a permis effectivement de lire correctement les entrées et de pouvoir les envoyer avec *UnityWebRequest*.

Secondement, le serveur web est quant à lui développé en Node.js avec Express.js respectant le modèle *vue-controlleur* qui nous permet facilement de créer une application web et ainsi de pouvoir communiquer avec le client sur la machine “vmProjetIntegrateurgrp1-0”. L’application contient 2 routes dans lesquelles le client Unity va tenter d’envoyer ses informations. La première information concerne l’inscription de l’utilisateur et la seconde la vérification et la connexion de l’utilisateur en interrogeant la base de données. La base de données étant faite avec MongoDB, cela permet de créer nos propres structures de données avec l’application web et de les lui transmettre par le biais de bibliothèques tiers Mongoose (MongoDB + Node.js). En effet, la base de données étant hébergée directement sur la “vmProjetIntegrateurgrp1-1”, le serveur web peut communiquer sous forme de requêtes grâce à Mongoose ; la base de données n’étant pas sous format SQL. De plus, la configuration et l’installation des modules pré-requis lors de la mise en place des serveurs a été elle aussi laborieuse.

Le serveur web ainsi que le Client Unity communiquent sous format de message *JSON* : nous avons opté pour un format spécifique. C’est-à-dire si la requête a fonctionné sans problème, le serveur enverra un message *JSON* ne comportant qu’un seul champ “success : true”. Si la requête n’a pas fonctionné, il est possible, soit qu’une erreur réseau soit intervenue pendant le processus et alors un autre champ sera rempli en plus de *success*, soit un champ “error” comportant les attributs “status” et “message” notifiant le type d’erreur réseau impliqué. Enfin, si aucune erreur réseau n’est intervenu et que la requête n’a pas pu aboutir, le serveur renverra un message comportant le champ “success : false” ainsi qu’un autre champ “message” comportant l’erreur spécifié : par exemple “Compte non existant”, “Duplicata de création de compte” sans forcément donner trop d’indication pour un utilisateur malveillant. Un schéma explicatif est mis à disposition ci-dessous :

Figure 6 : schéma communication authentification



Base de Données

L'équipe chargée de la base de données a initialement choisi d'utiliser MYSQL comme système de gestion mais, finalement, MongoDB et NodeJs ont été choisis car ils sont plus récents et simples à utiliser avec Unity. Notre base de données se compose d'une table qui contient les informations des utilisateurs inscrits au jeu. Elle contient le pseudonyme, l'adresse électronique, le mot de passe, et la date de création du compte pour chaque joueur inscrit.

Notre base de données est sollicitée par les scripts d'authentification et d'inscription. En ce qui concerne l'inscription, lorsque l'utilisateur remplit le formulaire (pseudonyme, mail, mot de passe), ses informations sont récupérées grâce au script "SignIn.cs". Ensuite, elles sont envoyées sous format JSON au serveur qui vérifie qu'il n'y a pas de redondance et les stocke dans notre base de données. L'authentification se fait de la même manière mais avec un formulaire différent (pseudonyme, mot de passe). Les informations sont envoyées sous format JSON au serveur qui vérifie qu'elles sont présentes sur notre base de données pour permettre au joueur de se connecter.

Sécurité

La sécurité étant un aspect non négligeable de tout projet, nous avons fait le choix de séparer la base de données du serveur de jeu et de disposer l'algorithme de jeu sur le serveur et non sur le client. Mis à part ces mesures, nous avons une authentification pour accéder au jeu et une base de données conservant les informations sensibles encodées.

De plus, les communications entre le client et nos machines virtuelles sont encodées par le certificat du Bastion que nous utilisons. Cela permet aux informations de ne pas transiter "en clair" et de ne pas se faire usurper par un *sniffer* (logiciel tel *wireshark* sur le réseau).

Avec ceci, nous voulions implémenter un JSon Web Token qui aurait permis une authentification presque sans failles pour authentifier chaque paquet reçu. Il fonctionne de la manière suivante : lors de l'authentification, les informations de connexion sont envoyées au serveur JWT associé. Ce dernier répond au client ayant fait la requête avec ses informations encodées en base64 mais surtout avec une signature faite à partir d'une clé que seul possède le serveur. Toute tentative d'usurpation est donc vaine.

Nous n'avons pas jugé nécessaire d'ajouter à ceci des mesures supplémentaires de sécurité.

Portage Web

Notre projet est disponible en application bureau et depuis un navigateur. Unity permet de générer aisément un exécutable pour la version bureau et Apache permet de le rendre accessible depuis un navigateur. Pour que ce dernier soit accessible depuis le navigateur, nous avons converti le projet en WebGL, et, par le biais d'Apache, cela permet d'avoir le site du jeu à l'adresse IP du Bastion et sur le port 11001.

Ouvertures

Evolution du système des Lobbys (5 à 6 heures de travail supplémentaires):

Nous pourrions améliorer le système de Lobby pour faire en sorte d'avoir plusieurs parties simultanées sur le serveur en implémentant un système où il y aurait un code propre à chaque partie. Pour cela, il aurait fallu implémenter un script "MatchMaker", qui, à la demande de création d'une partie de la part d'un joueur, aurait demandé au serveur de créer une *room* (ou lobby), et aurait généré un code pour le joueur. Ce dernier pourrait alors partager ce code avec ses amis, qui eux, à leur tour, pourraient le rejoindre. Le *matchmaker* est au courant de toutes les parties et lobby en cours. Pour la synchronisation de ces parties, Unity utilise un GUID (Globally Unique Identifier), qui sert à synchroniser les *assets* qui partagent un même identifiant.

Musique et effets sonores (4 heures de travail supplémentaires):

Nous pourrions ajouter du relief et de la profondeur à notre application en y incorporant des musiques et différents effets sonores. Pour la musique, il suffirait d'associer à nos pages une musique que nous téléchargeons et installons sur Unity. Comme thème principal, une musique douce rappelant l'espace aurait été choisie car CarcaSpace est un jeu de stratégie donc nous ne voulions pas une musique trop présente ni pesante. Nous aurions donc privilégié une musique calme qui ne déconcentre pas le joueur. Pour les effets sonores, il faudrait rajouter quelques lignes dans les scripts qui gèrent les boutons pour qu'un bruitage soit exécuté lorsque le joueur appuie. Nous aurions choisi des bruitages simples et peu fort disponibles sur Youtube par exemple. Ces bruitages seraient présents uniquement pour ajouter un effet de conséquence aux actions.

Rivières (5 à 10 heures de travail supplémentaires):

Une autre perspective d'amélioration serait d'ajouter l'extension de la rivière. Cette dernière respecterait les mêmes règles que le jeu de base par rapport à la pioche, la pose de tuiles, de pions et de points.

Cette extension est composée de 12 tuiles, voir figure 7. Ces tuiles sont toujours composées de villes, de chemins ou même d'abbayes combinées à la rivière, avec une tuile "source" et une tuile "lac". Aucune de ces 12 tuiles n'est mise dans la pioche principale : elles ont leur propre pioche, une pioche rivière. Ainsi, le jeu comporte deux types de pioches : les joueurs piochent d'abord dans la pioche rivière et seulement lorsqu'elle est épuisée, commencent à piocher dans la pioche normale.

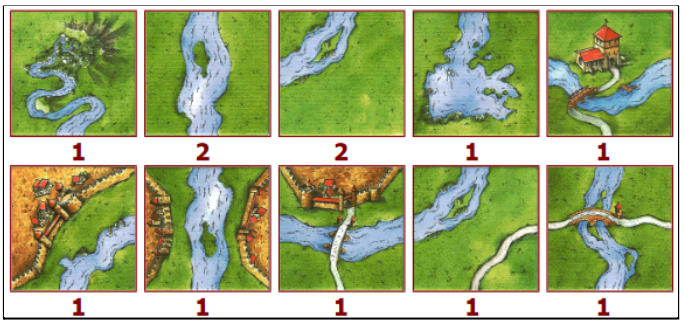
La "source" serait initialement affichée sur le plateau au début de la partie et la tuile "lac" serait mise à la fin de la pioche rivière. Lorsque le joueur piochera la

tuile “lac”, il la posera et pourra instantanément repochier une tuile normale. De plus, comme une vraie rivière, les joueurs ne peuvent pas faire revenir la rivière vers la source.

Pour l'implémentation de l'extension:

- Nous aurions d'abord créer les tuiles de la même manière que le jeu basique.
- Ensuite, dès le début de la partie, nous aurions ajouté sur le plateau la tuile "source".
- Pour la tuile "lac", nous aurions créé un compteur qui se décrémente à chaque tuile piochée. Lorsque le joueur piochera la dernière tuile, le compteur sera alors à 0 et la pose de cette tuile sera automatique : à la fin de la rivière pour la clôturer. Ce n'est qu'après cela que le joueur suivant pourra jouer en piochant la première tuile du jeu de base.
- Pour finir, nous aurions dû implémenter une fonction pour vérifier qu'un joueur n'essaie pas de ramener la rivière vers la source.

Figure 7 : tuiles rivières



Mot de passe oublié (10 à 15 heures de travail supplémentaires):

Nous avons aussi implémenté une fonctionnalité pour pouvoir récupérer le mot de passe en cas d'oubli. Lorsque le client clique sur le bouton "mot de passe oublié", il est amené vers une scène où il doit entrer l'adresse mail utilisée lors de son inscription. Si le mail est valide, un code secret lui est envoyé par mail et il passera vers une autre scène où il doit entrer le code secret lui permettant d'entrer un nouveau mot de passe.

Changer le fond d’écran (2 à 4 heures de travail supplémentaires):

Nous aurions voulu implémenter la possibilité de changer l'arrière-plan. C’est-à-dire de présenter à l’utilisateur trois fonds d’écran de base : un avec les couleurs de notre logo, un thème plus sombre et un thème plus clair. Il était aussi prévu d'implémenter une option pour télécharger une image au bon format et de l'avoir comme fond d’écran. Chaque utilisateur aurait eu ses propres fonds d’écran pendant une partie s’il le souhaitait.

Description Projet

Les équipes

Au tout début de ce projet, nous avons formé des équipes en fonction des préférences de chacun. Cependant, au fur et à mesure de l'avancée des travaux, nous avons dû changer certains groupes pour répondre aux jalons et apporter du soutien dans certains sous-groupes. Ainsi, au démarrage du projet, nous avons scindé notre groupe en 4 sous-groupes :

- **Back** : Godwin, Deborah, Michel, Florentin, Guillem, Anass et Achraf
- **Front** : Matthieu, Martin et Marc
- **Réseau** : Louis, Omar et Anthony
- **BDD** : Anass et Achraf

Comme explicité ci-dessus, ces groupes ont rapidement changé pour différentes raisons. Par exemple, l'implémentation de la BDD a été assez rapide donc Anass et Achraf ont rapidement intégré le groupe Back car le travail y était conséquent. De plus, au début de ce projet, le groupe Front a réalisé un travail considérable, ils étaient donc en avance sur les prévisions du Gantt donc nous avons décidé de le scinder et de muter certaines personnes pour aider à nouveau le groupe Back qui est le pilier de notre projet.

Ainsi, la répartition des groupes à été mainte fois modifiée pour être en adéquation avec nos échéances mais, à terme, ces groupes sont globalement représentatifs des préférences de chacun et de leur implication dans ces domaines.

Organisation et Modes de Travail

Au début de ce projet, nous savions qu'il y aurait des réunions hebdomadaires de 4 heures qui serait notre rendez-vous de la semaine pour mettre en commun nos avancées et réfléchir tous ensemble à la direction du projet. Cependant, nous avons, dès la première séance, mis en place d'autres dispositifs de communication et de partage pour que chaque personne puisse connaître l'avancée des autres et qu'il soit simple de communiquer entre nous. De ce fait, nous avons mis en place un serveur Discord avec différents *channels* pour chacun des groupes. De plus, nous avons bien évidemment créé un GIT ; indispensable à tout projet informatique. Nous avons aussi mis en place un

Google Drive collaboratif pour, par exemple, créer un *timesheet* recensant toutes les heures de travail de chacun. Ce fût aussi un endroit où chaque membre de ce projet pouvait consulter les différents rendus hebdomadaires et compte rendus de réunions que je (le chef de projet) produisais chaque semaine.

Grâce aux outils évoqués précédemment, nous avons pu organiser nos travaux et tâches au sein du groupe. En effet, grâce à l'utilisation de "Tickets" sur GIT, chaque membre pouvait consulter à tout moment les tâches qui lui étaient attribuées. De plus, grâce au compte rendus hebdomadaires, chacun pouvait s'informer de l'avancée globale du projet mais aussi de l'avancée de son groupe. Additionnellement, nous avons organisé plusieurs réunions Discord tout au long de ce trajet pour davantage travailler ensemble et avoir une meilleure cohésion.

Difficultés et Solutions

Globalement, nous n'avons pas rencontré de difficultés majeures au niveau de notre fonctionnement d'équipe. Cependant, comme vous avez pu le constater dans les descriptifs individuels, chaque membre a vécu une expérience différente.

En effet, certains membres de notre groupe manquaient de communication. C'est-à-dire que nous avons rencontré le problème que deux codes similaires soient produits car une sous-tâche à été mal attribuée ou justement, il y a eu un manque de communication au sein d'un sous-groupe. Ceci n'est heureusement arrivé qu'une seule fois. Cependant, la communication a parfois été difficile au sein d'un groupe rendant l'expérience de travail moins agréable mais, c'était la première fois pour chacun que nous travaillions avec autant de personnes. Mais, ce n'est pas une excuse ! C'est pour cela que nous avons mis des dispositifs en place pour l'empêcher au maximum comme par exemple les "Tickets" sur GIT pour que chaque membre se voit attribuer des tâches différentes. Aussi, la création de sous-groupes a été salvatrice, cela a facilité l'entraide et la communication. Nous avons donc rapidement mis en place une forme de hiérarchie pour optimiser au maximum nos travaux. C'est-à-dire que nous avons instauré des sous-chefs dans chaque groupe pour y améliorer l'organisation.

Additionnellement, chaque membre avait ses domaines de prédilections avec ses propres compétences mais il faut noter que certaines personnes avaient davantage de lacunes que d'autres. De ce fait, nous avons mis en place des sous-groupes stratégiques pour que les membres avec beaucoup d'expériences et de connaissances puissent aider ceux qui en avaient moins. Il a fallu donc, tout au long de ce projet, avoir une organisation méticuleuse pour que chaque tâche soit faite dans les temps et que personne ne soit "largué", mis à l'écart ou encore moins rejeté. C'était donc un travail quotidien pour attribuer les bonnes tâches aux bonnes personnes pour qu'*in fine*, tout le monde ait la possibilité de coder et donc d'apporter sa pierre à l'édifice.

Rôle du chef de projet

Ma vision d'un chef de projet a très peu changé depuis le début de notre projet. Cependant, ayant pratiqué ce rôle pendant plus de 15 semaines, j'ai appris de mes erreurs et j'ai pu mieux comprendre ce que ce rôle impliquait.

Au début, je pensais que le chef de projet aurait moins de travail que les développeurs car je n'arrivais pas à quantifier la charge de travail qui m'incombait. Je pensais naïvement qu'elle se résumait à suivre de manière hebdomadaire mes camarades pour faire régner une certaine organisation et rédiger nos documents : je me suis trompé ! En effet, ce rôle est primordial. Il nécessite un suivi presque quotidien de l'avancée de chacun et un accompagnement de chaque tâche pour que le projet progresse au mieux et se bâtisse petit à petit sans déséquilibre ni querelles. Il m'a donc fallu ajuster mes actes et ma vision d'un chef de projet au cours de cette aventure. Comme je l'avais dit dans notre premier rapport, c'est la première fois que j'endosse une telle tâche. J'ai donc été davantage présent au fur et à mesure que le projet avançait pour organiser méticuleusement le travail du groupe car l'intensification du travail se faisait ressentir dans toutes les matières de notre licence. Nous devions donc restés soudés et efficaces tout au long du projet. De plus, j'ai réellement compris l'utilité des différents outils de collaboration et de partage du travail car ils ont été décisifs.

Avec un peu de recul, j'ai certains regrets. J'aurai du comprendre plus rapidement ce que ce rôle impliquait car nous avons eu des dysfonctionnements dans certaines équipes qui auraient pu être évités avec une meilleure gestion notamment humaine. Cependant, grâce à l'infaillible motivation de mes camarades, cela n'a pas trop affecté notre travail. De plus, j'aurais dû mieux gérer le logiciel de gestion de versions que nous avons utilisé. Outre le fait que j'ai créé des « Tickets » pour organiser notre travail, j'aurais dû être davantage présent pour suivre chaque tâche et demander plus tôt les fusions des différentes branches pour éviter les multitudes de difficultés que nous avons eu à ce niveau-là.

En somme, ce fut une expérience incroyable et enrichissante. Elle m'a apporté une vision pour mon futur et aidé à le construire. En effet, j'ai constaté que cet exercice me plaisait et que je voulais l'explorer davantage. Globalement, nous avons été chacun présents les uns pour les autres et j'ai aimé travailler avec cette équipe. Je suis satisfait du résultat et ce projet nous ressemble. Je suis content qu'*in fine*, chaque personne ait apporté sa pierre à l'édifice pour construire notre premier projet le plus complet et complexe à ce jour.

Descriptifs Individuels

Godwin AMEGAH

Le projet Carcassonne fut une expérience formidable tant du côté technique que du côté humain. J'ai été affecté à trois équipes : une équipe Réseaux qui se chargeait globalement de la mise en place des fonctionnalités multijoueur, une équipe BDD chargée de la conception de la base de données, et enfin l'équipe SDA chargé de développer les algorithmes de jeu.

Dans un premier temps, j'ai rejoint la team Réseau où j'ai, avec la collaboration de mes collègues, contribué à la configuration du serveur web et du serveur de base de données et des installations nécessaires. Une fois fait, nous avons commencé à mener des recherches sur les diverses technologies qui vont nous permettre de mettre en place le multi-joueur. Nous avons finalement choisi la bibliothèque *Mirror*. Dès lors, nous avons commencé par nous former à la l'utilisation de *Mirror*. Une fois quelques fonctionnalités développées par la team Front, nous avons commencé par appliquer le multi-joueur tout en synchronisant les actions du jeu en mode Client-Serveur. Nous avons mis en place premièrement une approche *peer-to-peer* en local (sans notre vrai serveur) dans laquelle un client avait aussi le rôle du serveur et les autres clients (joueurs) pouvaient se connecter à ce dernier avant le début du jeu. Les actions synchronisées entre le client et le serveur concernés étaient le déplacement, la rotation, la pose et un timer des tuiles. Une fois ces éléments synchronisés, nous avons fini par intégrer les autres fonctionnalités. Dès lors, j'ai rejoint le groupe Back pour les soutenir.

Dans ce groupe, certaines fonctionnalités étaient en retard par rapport au cahier des charges. Il s'agissait de l'algorithme pour détecter la fermeture des chemins et des villes. J'ai dû me mettre à jour sur les fonctionnalités déjà faites. Avec la collaboration de mes collègues, pour plus d'efficacité, nous avions émis deux approches pour l'algorithme de fermeture de chemin : une approche itérative avec les structures et une autre récursive. Je me suis penché personnellement sur la version avec les structures.

Il est important de noter que pour assurer une bonne communication et se mettre à jour sur l'avancée du projet dans sa globalité (tâches terminées, bugs rencontrés au courant de la semaine, etc), nous organisions plusieurs réunions hebdomadaires présidées par notre chef de projet auquel je participais activement. Celà nous permettait de faire le point mais aussi de s'entraider mutuellement. Malheureusement, certaines tâches peuvent causer un retard mais nous avons pu les rattraper grâce à notre volonté quotidienne même à des heures tardives. A la fin, nous en sortons riches et mûris d'expériences. Merci à toute l'équipe, à notre tuteur et son assistance sans égale et à notre chef de projet qui a fait un travail remarquable pour mener à bien le projet.

Louis DISTEL

Ce projet a été très enrichissant pour ma part. Étant habitué à faire des projets seul, cela m'a demandé d'apprendre à travailler en équipe. Godwin, Omar et moi avons été attribués dans le groupe réseau qui était en fait au départ plus du système pour les VMs que du réseau. Par la suite, nous avons eu une tâche très importante qui était la parallélisation. Mais suite à un problème de coordination, nous étions plusieurs sur la tâche et Omar ayant terminé avant, nous avons récupéré son travail. Je n'ai donc pas mon empreinte sur la synchronisation *Mirror* entre les joueurs.

La plus grande partie de mon travail consistait à mettre le jeu accessible depuis un navigateur, aider ceux qui avaient des questions par rapport aux machines virtuelles et les paramétrer avec Anthony pour faire les correctes redirections et héberger le serveur.

Ensuite, je me suis alors attelé à faire le serveur de jeu et une fois fonctionnel, j'ai aidé ceux qui avaient des difficultés sur leurs tâches comme Anthony et l'authentification ou Omar pour la création du lobby. Au-delà de la partie technique qui m'a permis de bien me familiariser avec C# et Unity et le lien avec la partie système, j'ai trouvé l'organisation de l'équipe hétérogène, très agréable et même si notre communication n'a pas toujours été parfaite, j'ai trouvé que nous nous sommes bien démenés pour un premier projet d'une telle envergure.

Omar EL-CHAMAA

J’ai surtout travaillé sur la partie multijoueur du jeu, en commençant par la configuration du serveur et l’installation de logiciels nécessaires. Puis, sur la génération de la synchronisation des déplacements de tuiles entre les clients, celle des pions, celle des stars et enfin la création de système de Lobby.

J’ai été très déçu du manque d’implication et de l’incompétence de certains membres du groupe et le manque de communication d’autres (quelques exemples : Unity toujours pas installé une semaine avant le rendu pour une membre de l’équipe du back, 3 commits total depuis le début du projet…). A cause de cela, d’autres membres et moi nous retrouvons en sous-effectif constant, et à travailler sans cesse pour essayer de rattraper le retard causé par les membres mentionnés auparavant. Cela s’est traduit par une expérience assez stressante, alors que j’étais très optimiste au départ.

Les points positifs de ce projet sont que j’ai appris à utiliser beaucoup de nouvelles technologies, et à mieux maîtriser certaines que je connaissais déjà, comme setup et backup un serveur, C#, GIT…

J’ai aussi beaucoup apprécié le travail avec les membres de l'équipe qui étaient impliqués et avec lesquels on a pu créer des solutions algorithmiques et résoudre les problèmes inévitables lors du développement.

Martin FROEHLY

Mon travail au sein du projet consistait, dans un premier temps, à travailler sur le Front du jeu. J'ai notamment fait la gestion de la caméra, c'est-à-dire, ses mouvements via la souris ou via les touches "zqsd" et la possibilité de zoomer et dézoomer via la molette de la souris. De plus, j'ai assisté Matthieu au changement de caméra, entre une vue de trois quarts ou une vue du dessus.

Puis, je suis rapidement passé dans le groupe Back où j'ai travaillé sur les algorithmes de fermeture de zones. Plus précisément, les algorithmes qui régissent les règles pour qu'une ville, un chemin ou une abbaye soient clos. En parallèle à cela, j'ai aidé à la création des tuiles car leur encodage est nécessaire pour pouvoir faire les algorithmes de fermeture.

A l'origine, je ne devais pas travailler dans le Back ou beaucoup moins que prévu, mais suite à un retard, moi ainsi que d'autres avons été obligés d'intégrer ce groupe. Florentin, Godwin et moi étions obligés de rattraper le retard accumulé du précédent groupe.

Quand j'ai dû intégrer le groupe Back, j'ai supervisé le travail. Je proposais régulièrement des réunions et des sessions de travail afin de savoir ce qui avait été fait et ce qui nous manquait, mais aussi apporter la possibilité de travailler à plusieurs sur certains problèmes. J'ai fait des rapports réguliers à Nathan, le chef de projet, afin que lui aussi soit informé de nos avancées et de nos problèmes.

Déborah GANZITTI

Durant ce projet, au niveau du travail fourni, j'ai essentiellement travaillé sur les scripts "tile_type_x". J'ai dû les rectifier à plusieurs reprises car au fur et à mesure de l'avancement des autres scripts, il fallait changer des éléments ou en ajouter. J'ai travaillé avec Florentin sur l'algorithme de fermeture de zones pour les abbayes que nous avons terminées. J'ai aussi travaillé sur les algorithmes de fermetures de villes et de chemins. Cependant, comme je n'ai pas pu participer à certaines réunions, j'ai arrêté de travailler sur ces algorithmes. Sur la fin du projet, j'ai travaillé avec Guillem sur l'algorithme concernant les pions, pour qu'ils soient retournés à son joueur lorsqu'une zone est clôturée.

En dehors du code, j'ai expliqué en détails comment sont calculés les points d'une zone à Michel pour qu'il puisse faire l'algorithme des points. J'ai aussi écrit la documentation utilisateur.

Au niveau de l'équipe, je trouve que l'ambiance était plutôt bonne. Il manquait cependant un peu de communication à certains moments. Au début du semestre, avec les membres du groupe Back, nous n'avions pas vraiment d'idées sur quoi travailler, du coup nous avons perdu un temps précieux qui s'est répercuté sur une intensification du travail vers la fin. Sinon tous les membres étaient à l'écoute et disponibles si un membre avait besoin d'aide.

Anthony GUNES

Ayant travaillé initialement sur l'Interface Homme-Machine (IHM) lors du développement des scènes graphiques, la lisibilité et l'ergonomie ont été mises en avant grâce par exemple à la taille des boutons qui est proportionnelle à leur utilité. De plus, ayant travaillé avec Marc sûr le choix des maquettes, cela a permis à l'équipe d'avoir davantage de choix quant à la mise en place des maquettes de design. Initialement, le choix du nombre de scènes à été fixé au nombre de 7, certaines scènes de transitions ont été retirées et se dénombrent à un total de 4 scènes actuellement.

Ainsi, la prise en main d'Unity concernant l'interface graphique et les bibliothèques à utiliser, ont été plus ou moins longues : 7 semaines au total, notamment lors de l'implémentation de la scène de l'authentification. Le travail de groupe à été agréable jusque là pour l'équipe IHM.

De la réflexion, des choix de design à leur implémentation, une communication claire, toutes ces contributions m'ont semblé répondre aux problématiques nous concernant pour l'équipe. Notamment pour la nécessité de mettre en place mes connaissances en programmation web afin de développer un serveur web sur les serveurs qui nous ont été fournis et ainsi les configurer en adéquation avec les besoins du projet.

Achraf HABAS

Mon rôle principal dans ce projet était de spécifier et développer la BDD et le Back-end du jeu. J'ai d'abord commencé par la création des modèles MCD et MPD puis, avec Anass, nous avons réalisé les scripts de création des tables et les contraintes statiques et dynamiques qui y sont liées. Ensuite, j'ai commencé à me documenter sur comment faire la liaison entre Unity et notre base de données grâce à MongoDB et NodeJs.

Ensuite, vu que le deadline pour délivrer l'Alpha se resserrait, nous avons dû réorienter nos efforts sur la partie Back-end/Reseau. J'ai proposé l'utilisation du *framework Mirror* a l'équipe et quand on a décidé de l'adopter, j'ai configuré le *framework* et les outils nécessaires pour les tests dans la branche "multijoueur". J'ai ensuite travaillé avec Anass pour trouver une solution au problème de déconnexion.

Puis Nathan m'a assigné la tâche de gérer le tour-par-tour ; j'ai donc dû approfondir *Mirror*. J'ai créé un *GameManager* qui sert à contrôler le *flow* de la partie et j'ai modifié le "PlayerManager" pour avoir un tour-par-tour fonctionnel. J'ai aussi proposé à Marc de faire un *lobby* pour que les joueurs puissent accéder en même temps à la scène de jeu pour éviter tous les bugs liés à la connexion.

Sinon, la prise en main de C# et de Unity c'est fait assez naturellement à force de regarder des tutos et de coder. J'ai approfondi mes connaissances en réseaux même si de base ce n'est pas trop mon point fort et que j'aurai préféré travailler sur le front-end mais ce groupe était saturé. Je trouve que le travail dans un grand projet est une expérience enrichissante même si j'ai eu quelques soucis avec l'organisation du travail au sein de l'équipe. J'ai senti qu'il y avait une certaine concurrence entre les membres de l'équipe pour prendre les tâches, ce qui faisait que parfois je devais insister auprès du chef de projet pour en avoir une. J'ai aussi senti une certaine négligence par rapport à l'aspect de cohésion d'un groupe de la part des autres membres.

Anass HAOUD

Au début du projet, il m'a été attribué de travailler sur la base de données et le backend. J'ai commencé avec Achraf à faire tout ce qui est relié à cette dernière comme la réalisation de modèle entité association, MCD, MPD et l'écriture de scripts de création des tables en MYSQL. Les premières semaines, j'ai suivi des tutoriels sur Unity 3D et C# et je suis arrivé à réaliser une interface graphique basique d'inscription et de connexion sur Unity et de la lier avec notre base de données pour la tester. J'ai fait des recherches pour voir ce qui marchait le mieux avec Unity au niveau de la base de données et j'ai pensé à utiliser Node.js et Mongodb comme base de données. Donc, j'ai suivi des tutoriels sur ces derniers et je suis arrivé à créer une première authentification avec Node.js et Mongodb.

Quand j'ai fini mes tâches sur la base de données, j'ai essayé d'intégrer le groupe Back. J'ai demandé une tâche mais j'ai été ignorée et aucune tâche ne m'a été attribuée donc j'ai travaillé sur des tâches d'autres collègues de mon groupe. J'ai réalisé la tâche de génération aléatoire des tuiles et j'ai demandé la permission du chef de projet pour faire un push de mon travail. Après cela, j'ai continué à travailler sur les contraintes et la correction de quelques bugs sur la branche des contraintes.

Ensuite, j'ai intégré l'équipe Réseaux pour travailler sur le multijoueur. J'ai commencé par suivre des tutoriels en *Mirror* et d'essayer de comprendre le code qu'ils ont écrit avant mon arrivée. Il y avait un problème de synchronisation que j'ai résolu et j'ai ensuite commencé à faire la synchronisation directement sur la branche "develop". Cependant, ma version n'a pas été prise par le groupe.

Ensuite, j'ai travaillé avec Achraf sur le tour par tour. On a réussi à trouver une solution pour différencier chaque joueur par des id et on a réussi à faire une version correcte du tour par tour dans la branche "tourpartour".

A la fin, j'ai travaillé sur l'authentification. J'ai trouvé une méthode pour crypter le mot de passe et le stocker en utilisant 'salt' comme autre attribut dans notre base de données. Au niveau des relations entre les équipes, au début c'était normal mais petit à petit j'ai commencé à sentir que je suis exclu surtout à la fin où aucune tâche m'a été attribuée tandis que tout le monde avait une tâche à faire.

Nathan HENRY

Étant chef de projet, je n'ai malheureusement pas pu apporter un savoir technique à ce projet mais j'ai tout du long, organiser et planifier le travail de chacun pour qu'il y ait moins d'inégalités et que tout le monde puisse travailler dans leur domaine de compétence favori. Mais aussi, j'ai été le médiateur lors de tensions entre certains membres et j'ai essayé d'emmener tout le monde dans ce projet pour qu'il y est une bonne cohésion et que nous puissions produire une belle application. Cependant, j'ai certains regrets. Avec du recul, j'aurais dû être davantage exigeant et être "derrière" chaque membre pour les pousser à travailler car j'ai eu le défaut d'avoir trop confiance en mon équipe et d'être ainsi, peut être, trop laxiste. Malgré un suivi très régulier, certains membres étant moins motivés ont ralenti notre travail. Cela s'est répercuté sur d'autres membres et j'en suis sincèrement désolé. À l'avenir, je serai plus sévère pour pouvoir faire respecter au mieux une égalité de travail et de dévouement pour que chacun s'approprie au mieux le projet.

Secondement, j'ai constitué des rapports hebdomadaires de nos avancés et des comptes rendus de réunion pour pouvoir, d'un côté, suivre personnellement l'avancée de chaque groupe et donc du projet dans sa globalité. De l'autre côté, j'ai fait en sorte que chaque membre puisse savoir quoi faire, avec qui et dans quel délai. De plus, au tout début de ce projet j'ai constitué le cahier des charges avec l'aide de mes collègues pour planifier nos travaux et pour avoir des supports pour mieux nous organiser (comme le diagramme de Gantt). De ce fait, j'ai créé une multitude de "Tickets" sur GIT tout au long de ce projet pour qu'individuellement, chaque membre soit assigné à une tâche en particulier et qu'il n'ait jamais rien à faire.

Marc KACHOUH

Tout d’abord, l’idée d'un projet en informatique avec un grand groupe me faisait un peu peur car étant une personne perfectionniste, j’aime bien faire mes projets individuellement. Ayant des points forts en Design, j'ai voulu tout d’abord m’investir dans tous les aspects graphiques du projet. En effet, dans les premières semaines du projet, j'ai apporté des idées pour la personnalisation des tuiles, des pions, etc. Et après le choix du groupe, j’ai développé les autres aspects graphiques comme le *Wireframe*, les différentes pages et menus et les objets 3D, pour que les autres membres du groupe puissent s’y baser.

Puis, j’ai rejoint le groupe Front et j’étais le sous-chef. Nous avons eu quelques petits problèmes au début du projet avec des membres, mais cela a été rapidement réglé. J’ai eu la chance de coder quelques aspects graphiques au début du projet et j’avais énormément aimé le monde de Unity.

Ensuite, après plusieurs discussions avec notre tuteur, j’ai été obligé de me concentrer sur d’autres aspects du projet comme le Back ou le Réseau. Alors, j’ai rejoint le groupe Back et j’ai commencé à faire des pseudo-codes pour le *main* et les pions. Finalement, j’ai fini par rejoindre aussi le Réseau pour la création du *lobby* avec Omar.

En conclusion, après cette expérience, je me suis sentie plus proche du monde du travail. J’ai remarqué qu’avec de grandes équipes, tout ce qui compte c'est la bonne communication et les compromis. De plus, je trouve que notre équipe, même si on avait tous des avis différents, arrivait toujours par être d’accord. De plus, notre esprit d’équipe s'est développé en même temps que nos affinités les uns envers les autres, et travailler avec des personnes qu’on apprécie nous motive davantage car cela devient plaisant et agréable.

Michel KILANGALANGA

Dans ce projet j'ai globalement travaillé sur l'algorithme des comptages de points (chemins, villes, abbayes). Dû à mon aménagement d'étude dans la restauration, j'ai travaillé environ 75 heures sur ce projet. Malgré cela, j'ai quand même réussi à participer tant bien que mal à quelques discussions sur certains algorithmes. Entre autres, je me suis penché sur celui des pions et de la fermeture des villes malgré un manque de communication. J'ai eu beaucoup de mal sur l'apprentissage du langage C# et sur l'utilisation de Unity mais aussi pour travailler en groupe. À cause de mon aménagement d'étude et le manque de communication au milieu du projet, je me sentais un peu décalé sur l'évolution du projet, mais vers la fin je trouve que la communication s'est améliorée.

Florentin KOCHER

Étant davantage tenté par la programmation technique, j'ai rejoint le groupe du Back afin de me concentrer sur la réalisation concrète du jeu Carcassonne.

Durant les premières semaines, j'ai proposé ma vision des choses notamment sur la façon de représenter les tuiles, les interactions qui étaient possible de faire entre elles dont plusieurs ont été retenues. De plus, on m'a assigné la tâche supplémentaire de chef du groupe Back, un rôle qui m'a tenu à cœur. Il m'a fallu séparer un maximum les tâches à réaliser dans les premières semaines de développement afin de gagner un maximum de temps.

Après séparation des tâches, j'ai rejoint deux personnes du réseau durant plusieurs semaines afin de me concentrer sur l'aspect multijoueur de notre jeu avec *Mirror* (changement du code en mode "network", spawn des tuiles, création de la pioche et synchronisation des informations chez tous les clients).

Suite à cela, il a été demandé à deux autres personnes et moi de rejoindre à nouveau le Back car plusieurs fonctions n'avaient pas encore été faites. Nous nous sommes retrouvés uniquement nous trois à devoir faire la totalité du travail, en devant en plus respecter les délais de la Bêta. Nous avons donc fait l'algorithme de la fermeture des chemins et celui des villes. Je me suis penché beaucoup plus sur celui des chemins et en deux semaines au lieu de six, nous l'avons terminé sans réelle aide de nos collègues qui devaient initialement s'en charger. Je pense m'être très bien investi dans ce projet, aussi bien dans la recherche de solutions pour l'implémentation des éléments nécessaires dans les règles du jeu, que dans leur réalisation sur Unity.

J'ai aussi été au maximum disponible lorsque quelqu'un avait une question ou besoin d'aide. J'ai été particulièrement déçu par le manque d'intérêt et de travail de certaines personnes dans mon groupe. De ce fait, un retard considérable à été pris dans mes différentes tâches mais aussi dans la totalité du groupe. Cela nous a obligé à rattraper ce qui n'a pas été fait dans un laps de temps beaucoup plus court que celui prévu initialement.

Matthieu LEFEVRE

Ma partie de développement a été essentiellement sur l'architecture initiale du projet à la fois sur GIT et dans Unity. Tout d'abord, j'ai fait en sorte que les fichiers de scripts et leurs utilisations dans Unity soient arrangés pour la "version control", c'est-à-dire pour du travail collaboratif notamment grâce à un gestionnaire de versions comme GIT. Également, j'ai fait en sorte que les membres évitent de mettre sur le git les configurations personnelles, les fichiers cachés des IDE et encore des fichiers superflus que crée Unity et qui ne sont pas nécessaires pour ouvrir le projet.

Une fois la gestion de version terminée, j'ai développé la mécanique visuelle, le côté graphique, la manière dont les utilisateurs vont pouvoir piocher et poser leurs tuiles. Ainsi grâce à un premier script de Marc qui montre l'interaction de saisie et de poses de modèles 3D, j'ai trouvé une manière de représenter les tuiles dans le jeu et de créer une logique de "grille" qui permet d'aligner des tuiles sans les superposer ou encore de récupérer les coordonnées sur lesquelles l'utilisateur veut poser la tuile. Ensuite, la plus grosse partie de mon travail a été les animations, c'est-à-dire de faire de jolies transitions entre deux états sans qu'une animation en coupe une autre (exemple : pioche d'une tuile, rotation d'une tuile à gauche/droite...).

Enfin, pour finir, j'ai beaucoup aidé le groupe back pour qu'il comprenne comment obtenir les coordonnées des tuiles, ou alors comment afficher un message d'erreur lorsqu'on ne peut pas poser une tuile. J'ai fini le projet en faisant une UI sur la scène Unity dans le jeu qui permet d'afficher une mini carte à gauche ou encore de modifier les vitesses d'animation, etc.

Guillem LEGLAND

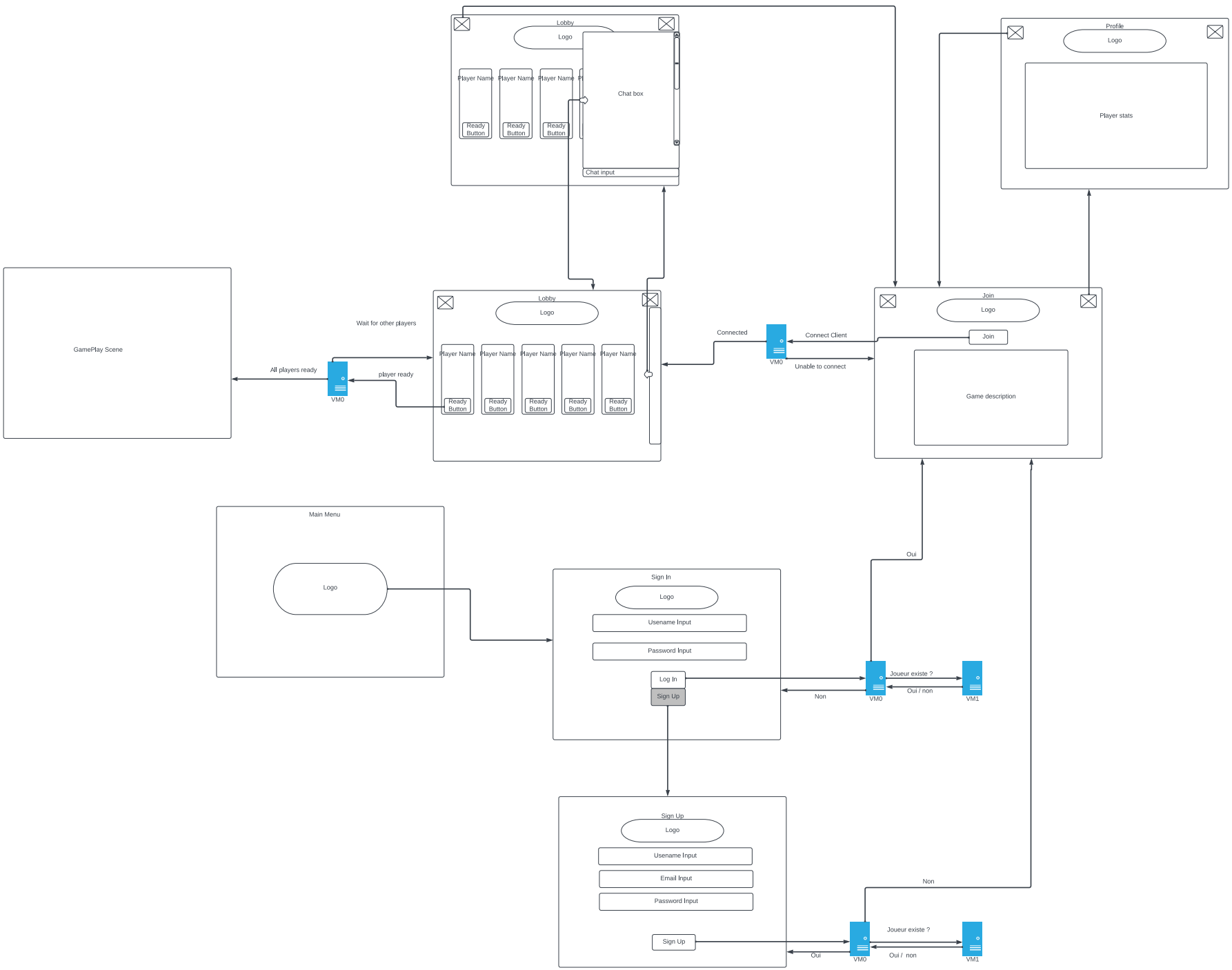
Je fais partie de ceux qui connaissaient le jeu avant de commencer à travailler dessus. J'ai donc pu facilement participer à la conception des classes au début du projet. J'ai fait partie de l'équipe chargée du backend. Je me suis en grande partie occupé de l'algorithme de vérification de pose de tuile de constraint.

Ensuite, j'ai passé un très long moment sur algorithme de vérification des chemins, des villes et des abbayes et c'est là que les problèmes sont apparus. En effet, nous ne maîtrisions pas bien unity et c# et avons passé beaucoup de temps et avons recommencé en changeant de méthode. Finalement, il y a eu une dispute avec un autre membre du groupe, puis nous avons été rejoints par deux autres personnes qui ont presque entièrement fait l'algorithme des chemins. Tout cela nous a fait perdre beaucoup de temps.

A la fin du projet, j'ai participé à la création de algorithme de comptage et à au développement de gestion des pions. Mon temps de travail est en dessous de ce qui est demandé, cela s'explique par le fait que au départ nous étions dans les temps par rapport au gant, je n'ai donc pas travaillé plus que nécessaire, puis après cela, suite a de nouveau engagement que j'ai pris, mon temps libre a diminué. J'aurais dû profiter de mon avance plutôt que de m'en satisfaire.

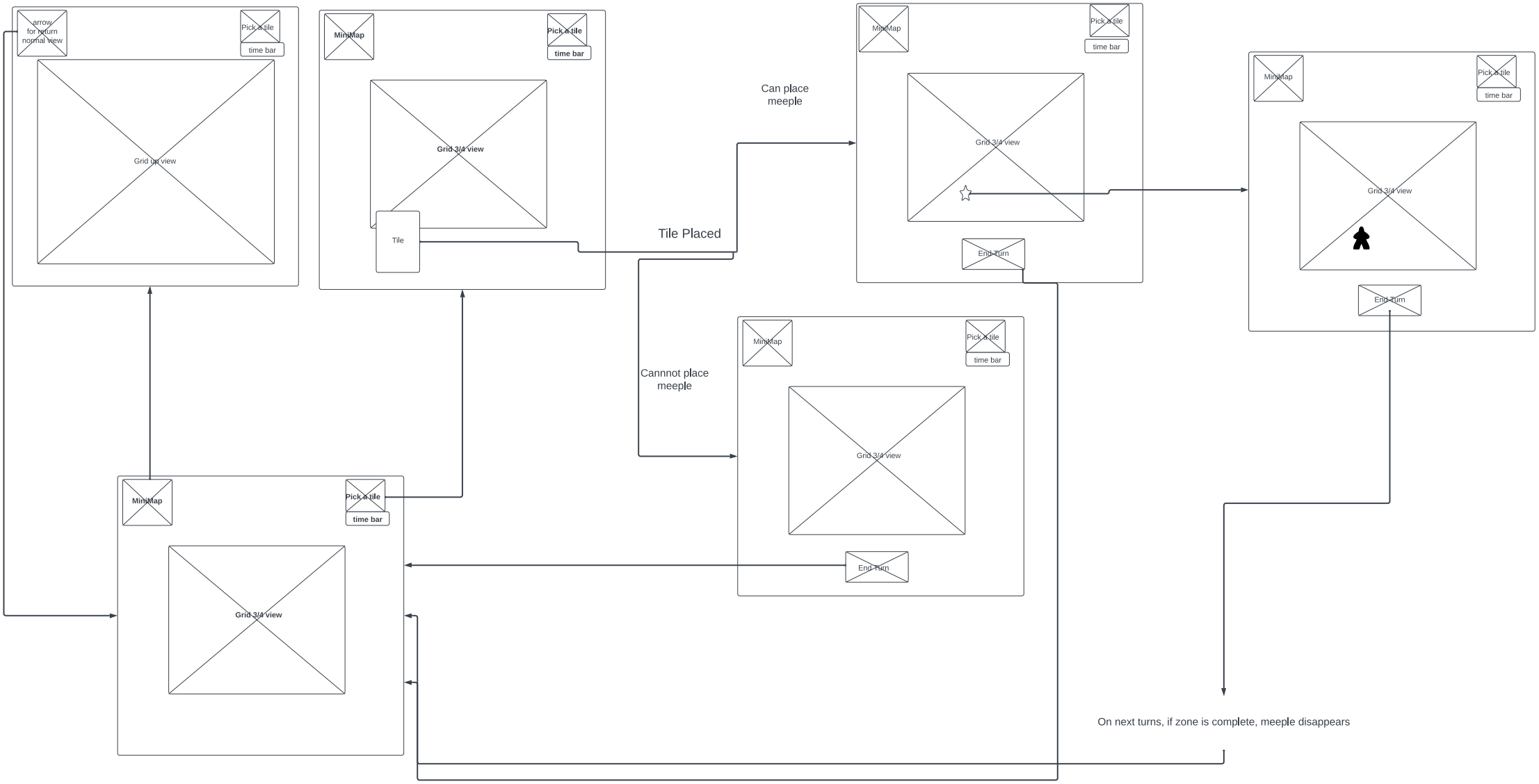
ANNEXE 1

Wireframe : Avant une partie

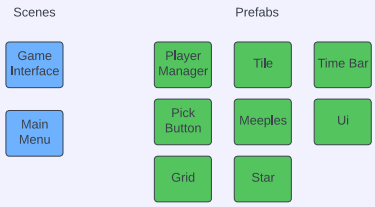


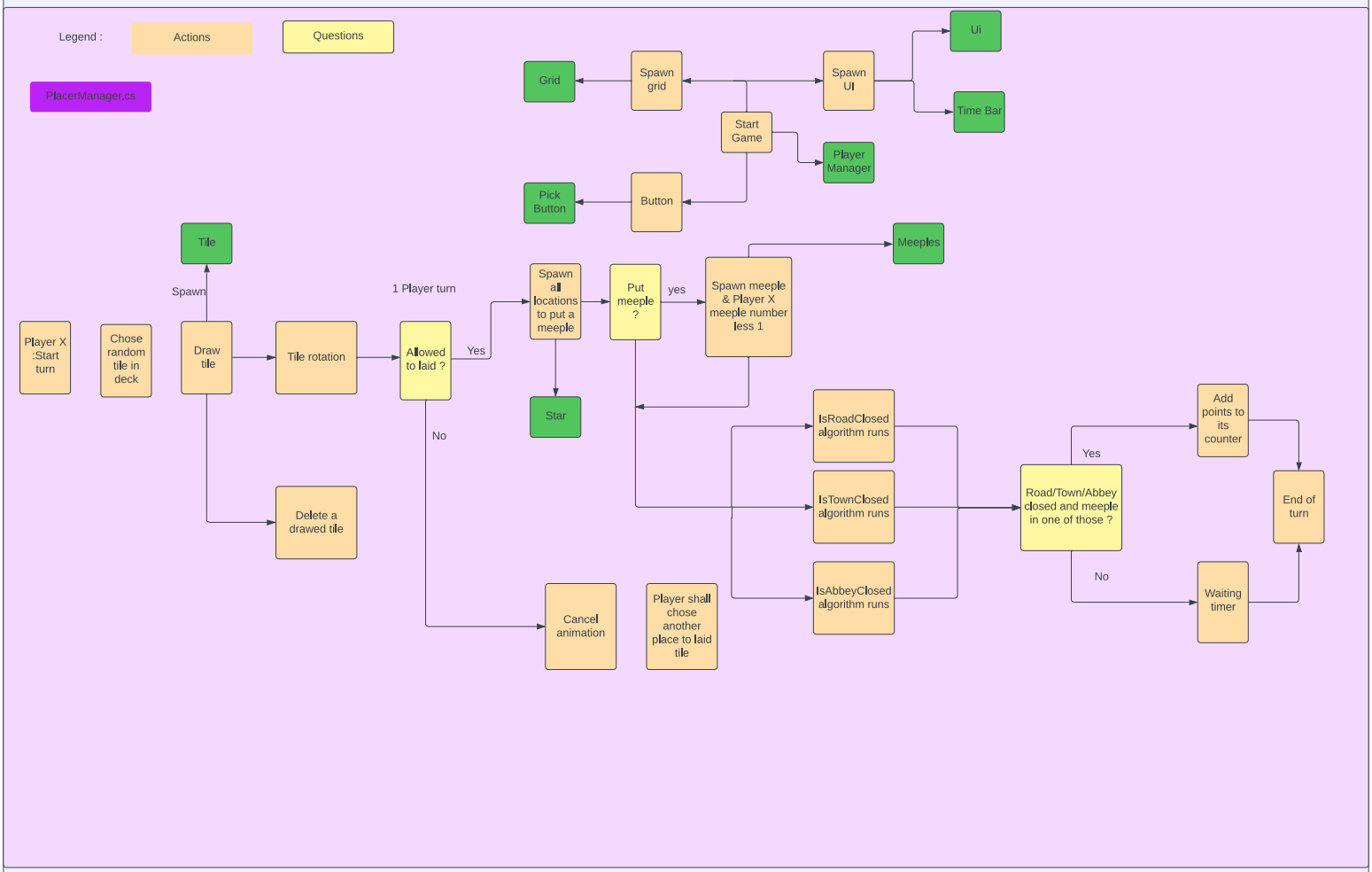
ANNEXE 2

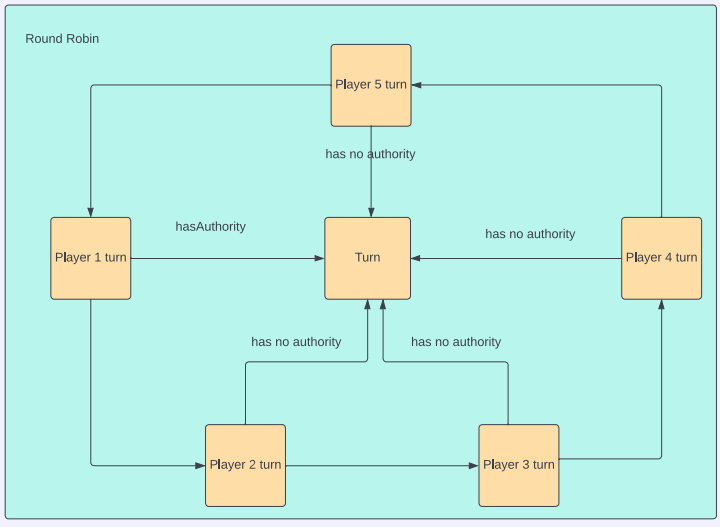
Wireframe : Gameplay



ANNEXE 3


Scenes
Prefabs
Game Interface
Main Menu
Player Manager
Pick Button
Grid
Tile
Meeples
Star
Time Bar
Ui


Legend :
Actions
Questions
PlacerManager.cs
Grid
Spawn grid
Spawn UI
Ui
Time Bar
Start Game
Player Manager
Pick Button
Button
Tile
Spawn
Meeples
1 Player turn
Spawn all locations to put a meeple
Put meeple ?
yes
Spawn meeple & Player X meeple number less 1
Player X :Start turn
Chose random tile in deck
Draw tile
Tile rotation
Allowed to laid ?
Yes
No
Star
IsRoadClosed algorithm runs
IsTownClosed algorithm runs
IsAbbeyClosed algorithm runs
Road/Town/Abbey closed and meeple in one of those ?
Yes
No
Add points to its counter
End of turn
Waiting timer
Delete a drawed tile
Cancel animation
Player shall chose another place to laid tile


Round Robin
Player 5 turn
has no authority
Player 1 turn
hasAuthority
Turn
has no authority
Player 4 turn
has no authority
has no authority
Player 2 turn
Player 3 turn