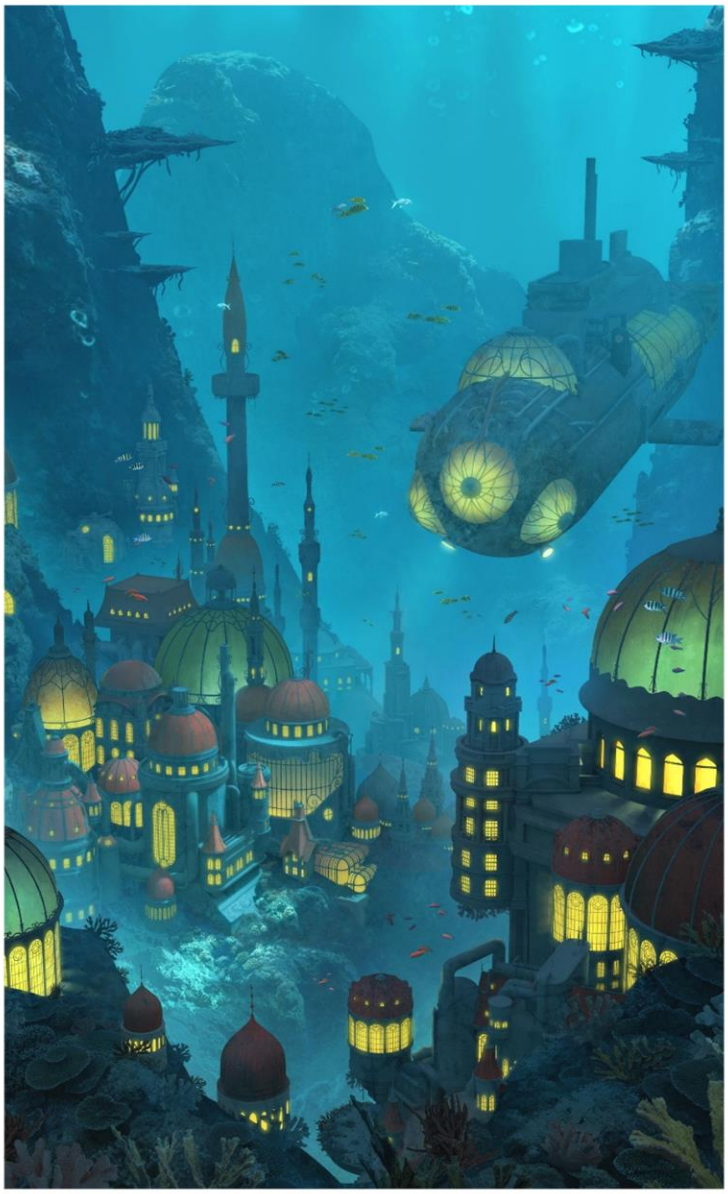


Rapport de Projet Intégrateur Carcassheim

Membres de l’équipe

Chef de projet :

Quentin Gerling

Membres de l’équipe :

Issam Eizouki

Adham Elbahrawy

Aya El-Mesaoudi

Sarah Emery

Justin Filipozzi

Matthieu Freitag

Axel Grimmer

Cedric Geissert

Victor Hahnschutz

Florian Halm

Gauthier Heimerdinger

Fahd Mahraz

Caleb Mukaya-Gakali

Table des Matières

Introduction

Dans le cadre de notre projet intégrateur, nous avons décidé de réaliser une adaptation du jeu de plateau Carcassonne, que nous avons appelé Carcassheim. Notre jeu, contrairement au jeu de base, se déroule dans les fonds sous-marins. Le but était en définitive de pouvoir jouer au jeu sur une version Windows, mais également sur une application Android. Ce projet étant une grande nouveauté pour nous tous, nous présenterons les parties techniques de celui-ci, mais porterons également un regard important sur les différents aspects humains rencontrés. Dans un premier temps, nous nous concentrerons sur le rapport du chef de projet, avant d'étudier plus en détail le travail fait par chaque équipe. Pour finir, nous nous concentrerons sur le point de vue individuel de chaque membre de l'équipe concernant le projet, où sera présenté le travail de chacun, mais surtout le ressenti de chacun sur le projet.

Partie du chef de projet

Notre groupe est constitué de quatorze personnes, réparties de la manière suivante :

- Système de jeu (trois personnes) qui s'occupent de la mise en forme des règles du jeu et de la création du serveur. Ils travaillent donc sur le back du côté client et du côté serveur.
- Interface in Game (trois personnes) qui s'occupent de l'IHM une fois dans une partie.
- Base de données (trois personnes)
- Menu (deux personnes) qui s'occupent de l'IHM du démarrage du jeu jusqu'au lancement d'une partie et reprend une fois la partie terminée.
- Réseau (deux personnes) qui s'occupent des transferts de données entre les clients et le serveur.

Concernant l'organisation du travail, la majeure partie se faisait en distanciel via Discord et des chats textuels ou vocaux. Une fois affecté à une équipe, chaque membre avait l'opportunité de choisir une tâche présente dans la catégorie "Issues" de git. Dans le cas où la description de cette dernière n'était pas suffisante, le membre pouvait consulter l'équipe liée à cette tâche ou bien contacter directement le chef de projet. Chaque membre pouvait accéder au gantt, reliant chaque "Issue" à une durée estimée et étant attribuée à un livrable spécifique.

Pour ce qui est du déroulement du projet, nous nous sommes réunis chaque mardi pour une réunion hebdomadaire et un check-up avec le professeur référent. Au cours de cette réunion, le chef de projet navigue entre les équipes pour prendre connaissance de l'avancement des tâches mais aussi afin d'observer les relations entre les membres.

Chaque groupe avançait au rythme qui leur convenait. Nous avions d'une part l'équipe "Menu" qui avait finalisé la majeure partie de ses tâches au cours des premières semaines, et d'autre part, une autre équipe qui s'est sentie plus inspirée par un développement hâtif en fin de projet. A titre d'exemple, les membres de ce dernier étaient constamment en retard par rapport au gantt d'environ cinq à huit jours. Sinon, de manière générale, les autres groupes ont avancé en se fiant au gantt.

Malheureusement, tout ne s'est pas aussi bien passé que nous l'espérions. En effet, deux membres nous ont manqué durant le projet à des périodes différentes, plus particulièrement le deuxième qui ne nous a pas transmis d'informations pendant plusieurs semaines. Ceci a naturellement entraîné un retard et par conséquent, il nous a fallu compenser. En sachant que nous étions alors à deux semaines de l'échéance, nous étions tous obligés de mettre les bouchés doubles afin d'espérer rendre le projet avec succès et dans les temps. Au-delà de ce problème, le projet s'est bien déroulé.

Dans notre cas, le chef de projet s'est occupé d'une multitude de tâches, allant des relations et communications entre les membres au développement et en passant par l'organisation. Dans un premier temps, ce dernier a mis en place les différents outils à utiliser pour l'organisation du groupe tel que le tableau des heures. Dans un second temps, il a renseigné toutes les "Issues" sur le git et a aussi amélioré l'organisation du serveur Discord qui nous a permis d'interagir et de retrouver certains documents très facilement. Pour finir, il a également créé le gantt qui a servi de ligne directrice à toutes les équipes en listant notamment toutes les tâches ainsi que les livrables.

Du côté des relations et communications, le chef de projet était le porte-parole du groupe lors des interactions avec le professeur pour décrire l'avancement du projet et transmettre les questions de certains membres. De plus, il a servi d'intermédiaire avec les autres groupes, auprès desquels il s'est renseigné afin d'en apprendre plus concernant leur progression. Finalement, il a aussi joué le rôle de médiateur lorsque les débats devenaient animés.

Ce dernier a également participé au développement, mais uniquement en fin de projet afin que nous puissions atteindre les objectifs que nous nous étions fixés.

En effet, il a participé à la phase d'intégration finale qui avait pris du retard en développant des fonctionnalités manquantes car trop compliquées à réaliser pour certains ou en corrigeant des erreurs qui n'avaient pas pu être réglées par d'autres.

Parties par équipes

Une fois le chef de projet choisi et les préférences de chacun prises en compte, les rôles ont pu être répartis aux différents membres, de sorte à avoir deux ou trois personnes par équipe. Les équipes ont finalement été les suivantes :

Chef de projet : Quentin Gerling

Reseau : Matthieu Freitag & Victor Hahnschutz

Bases de donnees : Adham Elbahrawy & Fahd Mahraz & Aya El-Mesaoudi

Design : Sarah Emery & Aya El-Mesaoudi

Systeme de jeu : Issam Eizouki & Justin Filipozzi & Axel Grimmer

Menu : Cedric Geissert & Florian Halm

Affichage InGame : Gauthier Heimerdinger & Caleb Mukaya-Gakali & Sarah Emery

Machines virtuelles

Etant donné que personne ne s'est porté volontaire pour le poste d'administrateur des machines virtuelles, le chef de projet a attribué ce rôle à Axel Grimmer (responsable "Système de jeu") et Matthieu Freitag (responsable "Réseau") car ils étaient les plus à mêmes d'en avoir l'utilité. Notons que le chef de projet s'est également attribué ce rôle.

Afin de renforcer la sécurité des VMs, nous avons appliqué un mot de passe au super-utilisateur "ubuntu" avant de le retirer de la liste des super-utilisateurs. Par la suite, nous avons créé un nouveau super-utilisateur "pro" auquel nous avons attribué le même mot de passe qu'à l'utilisateur "root". En effet, ce mot de passe est uniquement connu des administrateurs (Axel, Quentin, Matthieu) et permet notamment aux autres membres d'accéder aux machines, sans pour autant pouvoir y faire n'importe quoi, puisqu'en effet il est nécessaire d'ajouter un sudo devant chaque commande.

Nous avions deux machines virtuelles à notre disposition. Ainsi, la première sera celle qui permettra à notre programme du serveur de fonctionner et la deuxième permettra d'y placer une base de données.

Dans l'objectif de faciliter notre intégration, nous avons bien évidemment mis git en place sur nos deux machines virtuelles. Pour pouvoir pull, il est nécessaire de renseigner ses informations git.unistra personnelles, c'est-à-dire son nom d'utilisateur et son mot de passe, ainsi que de connaître le mot de passe administrateur (via sudo donc).

Comme indiqué précédemment, la première machine virtuelle est celle qui contient le programme du serveur. Ce dernier ayant été réalisé en C#, nous y avons donc téléchargé "dotnet". Nous avons notamment éprouvé des difficultés à installer une bonne version, compatible avec le code déjà rédigé par nos équipes.

Ensuite, afin de respecter le rôle d'un serveur, nous tenions à ce que notre programme soit robuste. Pour ce faire, nous avons fait en sorte que, via systemd, notre programme soit toujours en état de fonctionner. Si notre

programme s'arrête pour une raison X ou une raison Y, il redémarre automatiquement. De plus, si la machine virtuelle elle-même redémarre, le programme démarrera automatiquement une fois le redémarrage de cette dernière terminé. Nous avons également appliqué un pare-feu bloquant tous les ports excepté le 22 (SSH) et les ports 10000 à 10007 (pour les clients, voir la partie "Système de jeu").

La deuxième machine virtuelle comporte donc notre partie base de données. A ce titre, il était initialement prévu qu'elle soit réalisée en SQLite, mais dû à un besoin qui avait évolué et des contraintes auxquelles nous ne nous attendions pas, nous nous sommes plutôt tournés vers une base de données PostgreSQL. Nous avons également appliqué un pare-feu sur cette machine, bloquant tous les ports excepté le 22 (SSH) et le 5432 (PostgreSQL).

Equipe Réseau

Cette équipe est composée de deux personnes :

- Matthieu Freitag (responsable)
- Victor Hahnschutz

Notre branche par défaut sur git est la branche "reseau", qui n'est plus totalement à jour car nous avons notamment dû répartir notre travail sur différentes branches, mais elle n'en reste pas moins pertinente car elle liste une grande partie de notre contribution au projet.

Notre code et nos contributions se trouvent maintenant sur les branches "serveur" et "alpha". A noter que ces branches viennent compléter notre travail et ne sont pas de simples duplicatas de notre code. De plus, il est possible que d'ici la présentation prévue le 18/05, notre code se retrouve sur des branches encore différentes. Celui-ci est documenté en anglais, selon les normes C# via le format XML.

Notre équipe a débuté par la rédaction de la spécification. Cette tâche nous paraissait simple au départ, mais nous nous sommes très vite aperçus, au fur et à mesure de notre avancée, que nous nous étions trompés lors de la rédaction de cette dernière. En effet, la base restait correcte, mais certains détails laissaient à désirer et n'avaient pas d'utilité concrète. Nous avons donc maintenu un lien très fort avec l'équipe "Système de jeu" tout le long de la réalisation de ce projet, afin d'adapter notre code à leurs besoins et inversement.

Une fois la spécification établie, nous nous sommes alors tournés vers le code. Nous avions décidé, d'un commun accord entre les différentes équipes, de développer avec le langage C#, tant du côté client que du côté serveur. De ce fait, nous avons donc effectué des recherches concernant les différentes possibilités qu'offre le C# en termes de communication réseau. Il existe plusieurs moyens de communiquer en C#, comme des "Socket", des "WebSockets" ou encore "TCPClient". Etant donné que nous sommes des novices, nous avons préféré rester sur une base que nous maîtrisons un minimum et nous nous sommes alors tournés vers les sockets on ne peut

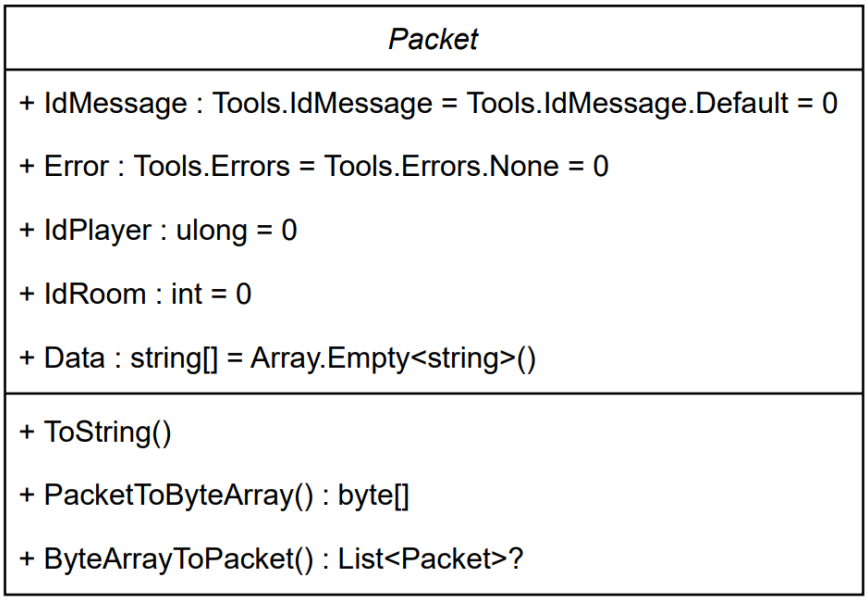
plus classiques de la classe System.Net.Sockets qu'offre C#. A ce titre, la documentation qui accompagne cette classe est extrêmement complète et comporte même un code qui nous a servi de base.

Nous avons donc essentiellement développé sur la branche "reseau", qui nous permettait alors de tester nos programmes en local, car il nous a semblé être plus intéressant d'abord d'établir une base fonctionnelle avant de déployer notre travail sur les machines virtuelles. De plus, afin de faciliter notre développement, nous avons mis en place des tests unitaires dans l'objectif de vérifier le bon fonctionnement de certaines de nos méthodes. Enfin, pour automatiser tout ce processus, nous avons mis un CI/CD. Ce dernier permet de vérifier que le nouveau programme mis à jour compile, que le style de code est respecté, que les tests unitaires sont validés et affiche également le résultat des tests ainsi que la valeur du coverage dans le README.

Comme indiqué précédemment, nous avons simplement suivi la documentation sur les sockets et avons obtenu la base de code suivante : un serveur asynchrone et un client synchrone. En effet, selon nous, le serveur se devait d'être asynchrone pour qu'il puisse recevoir plusieurs connexions de clients en simultané. De plus, à ce moment-là, nous pensions encore qu'avoir un client synchrone serait suffisant.

La base de ces deux programmes a été récupérée depuis la documentation Microsoft qui les accompagne et nous avons bien évidemment vérifié au préalable que cette première version était fonctionnelle avant de poursuivre. Lorsque le serveur détecte une demande de connexion d'un client, il va créer un thread et gérer ce client de manière asynchrone. Pour en savoir plus sur le fonctionnement à l'intérieur d'une partie, nous vous redirigeons vers le rapport de l'équipe "Système de jeu".

Malheureusement, cette base permettait uniquement de transférer une chaîne de caractères, ce qui n'est pas très permissif dans le cadre d'un jeu vidéo. C'est pourquoi, nous avons mis en place une classe "Packet" nous permettant de renseigner différentes informations que nous souhaitons transmettre depuis les clients vers le serveur ou inversement. Cette classe a été soumise à de multiples modifications au fur et à mesure du projet afin de s'adapter aux besoins des différentes équipes.



Cependant, pour que cette classe puisse être transmise, il nous a fallu implémenter plusieurs méthodes. Etant donné que les méthodes de transfert de la classe “Socket” requièrent impérativement des tableaux de bytes, nous avons donc réalisé des méthodes de sérialisation/désérialisation, qui sont utilisées à chaque envoi/réception de données entre sockets. Ces méthodes ont été particulièrement compliquées à mettre en place, notamment à cause des problèmes de compatibilités entre Unity et les applications consoles C#.

Cette classe possède une taille maximale en nombre de bytes. En effet, nous souhaitions alors avoir des tailles maximales pour faciliter et régulariser le transfert des données, bien que TCP possède déjà une taille maximale. Pour ce faire, nous avons implémenté deux méthodes se basant sur la taille maximale définie. Elles permettent de transformer un unique paquet en une liste de paquets, ou inversement, de transformer une liste de paquets en un unique paquet. Ces méthodes étaient en relation avec l’attribut booléen Final de la classe “Packet”. Lorsque ce dernier était à false, les données étaient placées dans un buffer puis on poursuivait l’écoute jusqu’à recevoir un paquet avec la valeur à true, qui signifiait alors la fin d’une transmission avant de poursuivre vers une toute nouvelle écoute.
Malheureusement, dû à une modification majeure que nous détaillerons plus bas, ces fonctionnalités n’ont pas été conservées car trop compliquées à maintenir en place. De plus, le risque qu’un paquet possède une taille supérieure à celle que nous avons fixée existe toujours, mais il est minime. D’après nos observations, un paquet possède en moyenne une taille de 100 bytes, or, à la base, nous avions fixé une taille maximale de 512 avant de la doubler à la suite de notre modification par mesure de sécurité.

Nous avons ensuite tenu à mettre à jour notre programme client sur la branche “alpha”. Le fait que ce programme soit synchrone nous empêchait notamment de recevoir des informations indépendamment de ce que le client envoyait au serveur. Cette tâche a été attribuée à Victor, qui malheureusement a rencontré des difficultés. Par conséquent, nous avons demandé de l’aide au chef de projet qui travaillait déjà avec Unity car nous étions en pleine phase d’intégration entre les équipes.

Ainsi, le client possède deux sockets. Le premier correspond aux interactions liées au menu et le deuxième correspond aux interactions à l’intérieur d’une partie. Seul un des deux sockets est utilisé à la fois. Pour mieux comprendre les changements de ports, se référer au rapport de l’équipe “Système de jeu”.

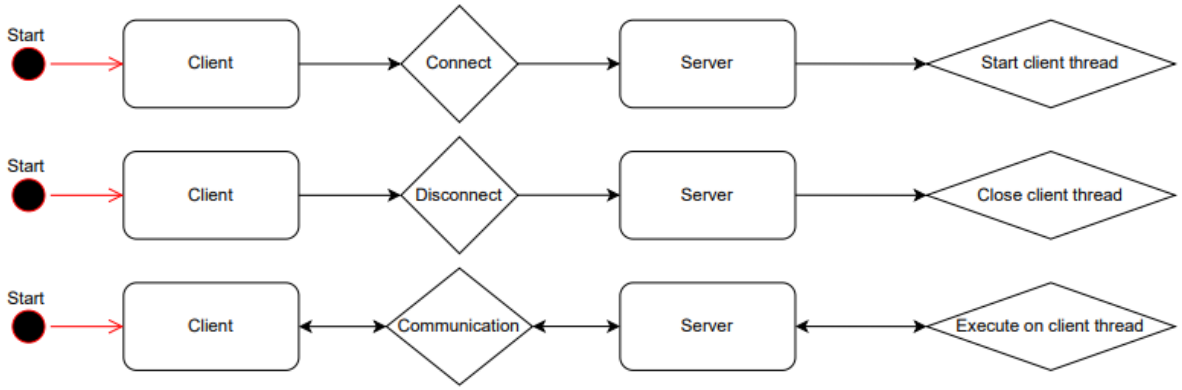
Des timers ont été placés sur les sockets lors des écoutes afin d’éviter des attentes infinies. Nous avons implémenté la récupération des informations locales/distantes pour le serveur/client depuis des fichiers de configuration au format JSON. Un système de gestion des erreurs avec une énumération et des try/catchs est également en place.

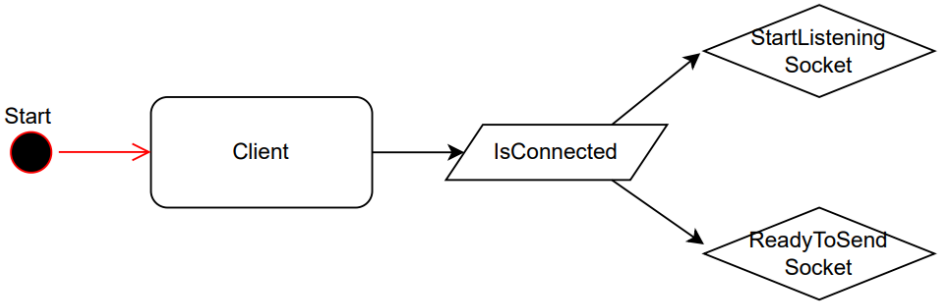
Suite à ces implémentations, nous avons alors estimé que notre code était assez stable pour être utilisable et l’avons donc mis en place sur les machines virtuelles. Pour ce faire, nous avons alors procédé à une mise en commun avec l’équipe “Système de jeu” sur la branche “serveur”. Grâce à cela, nous nous sommes alors aperçus d’un problème majeur de notre code (que nous avons mentionné auparavant).

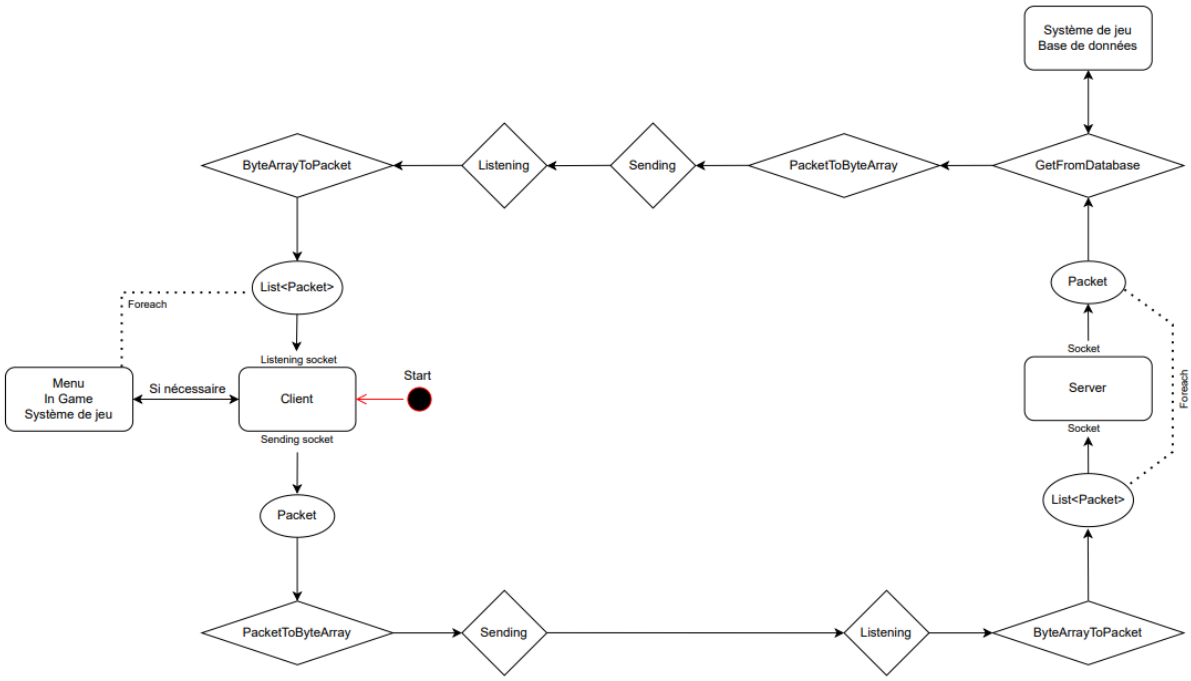
En effet, lorsqu’un socket débute une réception, il s’attend idéalement à recevoir un unique paquet. Pourtant, dans de rares cas, il est possible qu’entre le moment où il reçoit ce dit paquet et celui où il stoppe sa phase de réception, d’autres paquets arrivent. Ainsi, on se retrouve avec plusieurs paquets (encore sous forme de bytes) qui se suivent et donc, la désérialisation n’est plus fonctionnelle et il devient impossible de lire ces paquets. Nous avons remédié à ce problème en s’attendant plutôt à recevoir une liste de bytes arrays, mais cette nouvelle

version est incompatible avec notre implémentation précédente, du moins compliquée à maintenir, ce qui explique pourquoi nous l'avons retirée pour le moment.

Parallèlement à la résolution de ce problème, nous avons mis en place différentes méthodes permettant de réagir de manière adéquate à la requête d'un client. Ces méthodes sont donc reliées au travail des équipes "Système de jeu" et "Base de données".







Dans l'objectif d'une amélioration future, il serait intéressant de remettre en place les méthodes de séparation ou de concaténation de paquets lorsque sa taille dépasse la taille maximale. De plus, nous pourrions également remplacer le tableau de string dans la classe "Packet" par un tableau d'objet, ce qui serait plus permissif. Par ailleurs, nous devons encore mettre en place, d'ici la soutenance, un moyen de chiffrer les données transmises, mais aussi et surtout arrêter d'envoyer le mot de passe en clair (que nous avions modifié pour nous aider lors du debug).

Equipe Système de jeu

Cette équipe est composée de trois personnes, et a reçu l'aide de Gauthier Heimerdinger, Quentin Gerling et Matthieu Freitag :

- Axel Grimmer
- Justin Filipozzi
- Issam Eizouki

Structure de données

Aspect général

Pour stocker les informations lors du déroulement d'une partie, nous utilisons un objet plateau sur lequel sont posées les tuiles. Celles-ci sont composées de différents slots sur lesquels les joueurs pourront poser leurs meeples, et chacun de ces slots est défini par un type de terrain (ville, champs, chemin, abbaye, etc).

Les slots sont reliés aux positions internes de la tuiles (voir « modèle de tuile »), cela facilite les algorithmes lorsqu'il est nécessaire de parcourir le plateau. Les modèles de tuiles sont quant à eux stockés dans un dictionnaire, dont la clef est l'id de la tuile.

Quand une tuile est positionnée sur le plateau, un nouvel objet tuile est instancié, dont tous les attributs sont des copies de ceux de la tuile modèle. De plus, des attributs pour définir sa position lui sont attribués : des coordonnées et une valeur de rotation (comprise entre 0 et 3). Il est ensuite possible d'accéder à une tuile du plateau grâce à ses coordonnées.

A cela s'ajoutent enfin de nombreuses méthodes permettant de vérifier la validité de la position d'une tuile par rapport à ses voisines. Il en est de même pour la pose d'un meeple. D'autres vérifient si une zone a été fermée et, si c'est le cas, comptent le nombre de points tirés de cette zone pour chaque joueur, etc.

Création des tuiles

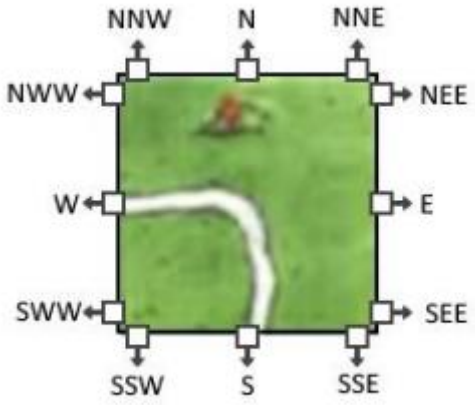
Après plusieurs discussions avec l'équipe de BDD, il fut décidé que les données des tuiles allaient être stockées dans un fichier XML. La BDD quant à elle, ne servira plus qu'à organiser les tuiles par extension et connaître le taux de présence de chacune.

Gauthier a, par ailleurs, codé en python un programme permettant de facilement créer n'importe quelle tuile et l'exporter au format XML, ce qui nous permis de générer l'ensemble des différentes tuiles du jeu sans difficulté

particulière. Programme qui saura être d'une grande aide pour de futurs ajouts au jeu, dans le cas de la sortie d'une extension par exemple.

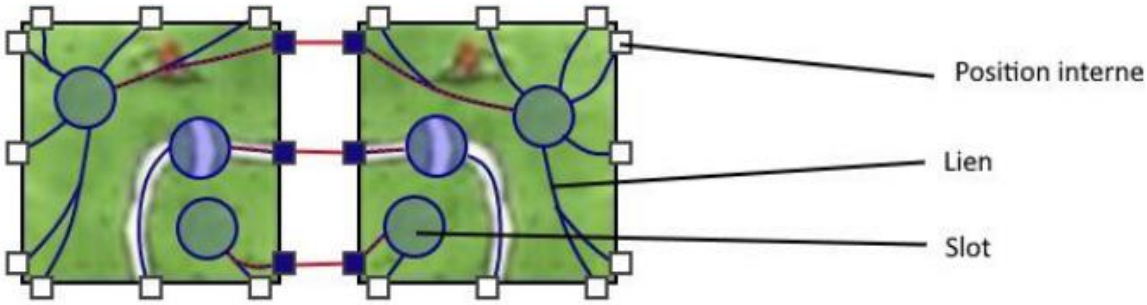
Au lancement d'une partie, ce fichier est lu et les données des tuiles sont stockées dans la RAM, autant du côté client que du serveur. Ainsi, si triche il y a du côté client dans le but de modifier les données d'une tuile, nous pourrons aisément la détecter depuis le serveur.

Modèle de tuile



Chaque tuile possède douze positions internes : Nord Nord-Ouest, Nord, Nord Nord-Est, etc. Une fois posée sur le plateau, elle est identifiable par le couple idTuile/position, sa position étant représentée par une position en X, Y et une rotation.

Second point important concernant les tuiles : la notion de slot. Carcassonne est un jeu particulier dans le sens où la pose de tuile se fait toujours de manière à « continuer l'illustration », interdisant les positions qui rendraient incohérent le dessin ainsi formé. Gauthier, de l'équipe de l'affichage in-game est donc venu nous proposer une structure adaptée afin de gérer cette particularité le plus simplement.



Afin de modéliser au mieux les différentes « parties » d'une carte, nous avons choisi de modéliser ces zones par des slots, sous les conseils avisés de notre camarade. Chaque modèle de tuile se voit donc attribuer un nombre de slots relié au nombre de « zones » visibles sur la carte. Puis, chaque slot est relié par un lien à plusieurs

positions afin de le localiser au sein de la tuile. Ces slots possèdent un type de terrain, représentant le genre de zone dessiné sur la tuile : pré, route, ville, etc.

De cette manière, nous pouvons facilement décider de la validité d'une pose de tuile. De même, il nous est aisé de parcourir l'entièreté d'une zone composée de plusieurs cartes juxtaposées, en suivant les slots qui se rejoignent par des positions internes et possédant le même type de terrain.

Tirage de tuile

Une spécificité de notre système de tirage de tuile se trouve dans le fait que, plutôt que de tirer uniquement une tuile, nous en tirons trois. En effet, dans Carcassonne il est possible que la tuile piochée ne puisse pas être posée sur le terrain. Ce triplet permet de réduire le nombre de communications, puisqu'il y aura drastiquement moins de chance qu'aucune des trois tuiles ne puisse être posée.

Spécificités serveur

Antitriche (pioche)

Le serveur possède un système d'antitriche particulier, se basant sur les réponses des autres joueurs pour ne pas avoir à parcourir lui-même les plateaux de jeu et ainsi gagner en performance.

Il fonctionne de la manière suivante :

(Phase de pioche)
• Le serveur envoie les 3 tuiles piochées à tous les joueurs.

(Phase de vérification/choix)
• Tous les clients envoient au serveur la première tuile qui peut être posée selon eux.

(Automatiquement, sans intervention joueur)
• Le serveur analyse les réponses des autres joueurs dont ce n'est pas le tour de jouer :

- Si aucun d'entre eux n'affirme qu'une tuile peut être posée, le serveur renvoie trois nouvelles tuiles.
- Si au moins un d'eux affirme qu'une des tuiles peut être posée, et que la position s'avère valide, alors le serveur informe le joueur actif de sa tentative de triche et lui renvoie les trois mêmes tuiles.
- Si la tuile renvoyée par le joueur actif correspond à celle renvoyée par les autres clients, alors cette tuile est piochée et le tour débute.

Après deux tentatives de triche, le joueur concerné est renvoyé de la partie.

Organisation du code

Le code du serveur de jeu possède une organisation codifiée : toutes communications entrantes passent par le fichier Methods.cs, dans lequel un traitement particulier lui sera accordé. Suivant l'idMessage de la communication (sa nature), le traitement et donc la méthode appelée seront différents.

Cette méthode, au sein de Methods.cs, trie et s'assure que la communication soit correcte (taille correcte, destinataire correct, etc.). Si la communication ne nécessite pas d'échange avec un thread de communication, la méthode s'en charge elle-même en appelant généralement la base de données, puisqu'il s'agit principalement

des communications de connexion, création de compte, etc.
En revanche, s'il s'agit d'une communication avec une partie de jeu, elle appellera un singleton nommé GestionnaireThreadCom qui s'occupe de la gestion des threads de communication.

Au sein du gestionnaire, la méthode liée à la nature de la communication va parcourir les threads de communication pour trouver celui qui s'occupe de la partie en question. Alors, il appelle une méthode de ce thread de communication.

Enfin, au sein de cette méthode-ci, on parcourt toutes les parties gérées par le thread de communication pour trouver la bonne puis on appelle la méthode du Thread_serveur_jeu liée, afin d'appliquer ou récupérer les informations de la communication.

« Thread » de serveur de jeu

Le terme de « thread de serveur de jeu » est en réalité un reliquat de la première version de l'architecture système. Il ne s'agit en vérité pas d'un thread, mais simplement d'un objet représentant un serveur de jeu.

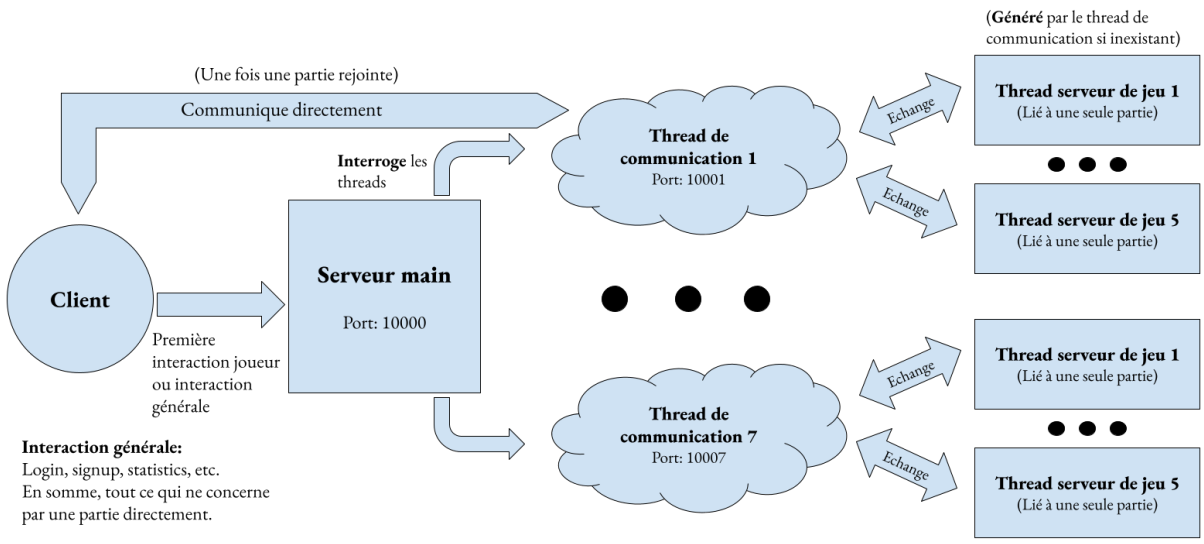
En effet, il n'est pas nécessaire d'en faire des threads puisque le serveur de jeu ne possède pas de méthode qui tourne en boucle : ce n'est qu'un agrégat de méthodes appelées suivant les communications entrantes.

En cela, puisque nous avons une écoute asynchrone de la part du serveur, chaque communication entrante est automatiquement traitée par un thread et, par conséquent, l'appel aux différentes méthodes jusqu'à arriver sur les méthodes du Thread_serveur_jeu est exécuté en parallèle, par ledit thread.

Communication client/serveur

Modèle de communication

Le modèle de communication utilisé est représenté par le schéma suivant :



Tour de jeu type

Un tour de jeu type débute par la demande au serveur d'une pioche de tuile de la part du joueur actif. Ce dernier y répond donc et partage cette pioche à tous les joueurs, mettant en œuvre le système d'antitriche expliqué plus haut.

La tuile ainsi choisie, le joueur peut la poser aux emplacements indiqués par l'affichage in-game. Ce placement est transmis au serveur et ce dernier en vérifie la validité, le transmet à tous les joueurs, avant de le stocker à part pour pouvoir, au besoin, se souvenir du dernier coup joué.

Après cela, il peut changer la rotation de la tuile en cliquant une nouvelle fois dessus, choisir de placer la tuile autre part en cliquant sur un autre emplacement ou bien poser un pion. Dans tous les cas, cela génèrera une communication vers le serveur pour que ce dernier vérifie le coup et le broadcast aux autres joueurs pour leur permettre de l'afficher.

S'il vient de poser un pion, il peut là aussi cliquer à un autre endroit pour choisir de placer son pion autre part, générant là aussi une communication pour vérifier et transmettre l'information.

Après avoir obligatoirement posé une tuile, et potentiellement un pion, il peut choisir de terminer son tour avec la touche entrée. Une communication de fin de tour est envoyée avec un résumé du coup du joueur, permettant au serveur d'en vérifier la validité s'il diverge du dernier coup stocké. Enfin, le serveur broadcast le coup finalisé à tout le monde, met à jour sa structure de données et n'a plus qu'à attendre une nouvelle demande de pioche de tuile, provenant du prochain joueur.

À tout moment de l'échange, si le client envoie un placement illégal au serveur, ce dernier lui répond qu'une tentative de triche a été repérée et prise en compte. En effet, le code du client n'est pas censé proposer de placements illégaux. Par conséquent, si un tel placement est envoyé, il y a fort à parier que le joueur a modifié son client et cherche donc à tricher.

Timer de tour

Le serveur dispose d'une fonctionnalité de timer, pour le tour des joueurs et la partie en cours. Ils sont tous deux initialisés au début de la partie, mais celui du joueur est relancé à chaque fin de tour pour le prochain joueur. Dans le cas où un joueur voit son timer expirer pendant son tour de jeu, son dernier coup est malgré tout pris en compte par le serveur grâce à la sauvegarde de chaque placement de tuile et de pion. Si d'aventure il n'a posé aucun pion ni tuile, son tour se termine sans effectuer de coup et la tuile piochée au début de son tour est remise dans la liste des tuiles de la partie.

En conclusion, nous sommes parvenus à réaliser ce que nous avions prévu de faire, dans la majorité, bien qu'un manque d'anticipation dans la spécification nous ait mené à rencontrer plus d'erreurs que nous aurions dû. Quelques fonctionnalités ne sont pas tout à fait au point en jeu multijoueur, notamment les différents modes de jeu, mais ce ne sont pas d'importantes modifications qu'il manque afin de les rendre fonctionnelles. La plus grande difficulté rencontrée, qui fut à l'origine de notre manque de spécification, se trouve être la perception de l'envergure du projet. Il nous a été très difficile de nous rendre compte de la complexité d'un système de jeu. Et ce, non pas d'un point de vue de la logique pure du jeu -dont l'implémentation a été facilitée par la structure de donnée choisie-, mais plutôt du nombre d'échanges, de paramètres et de précisions à prendre en compte.

Equipe BDD

Cette équipe est composée de 3 personnes, que Quentin Gerling a dû rejoindre par la suite :

- Adham Elbahrawy
- Aya Elmesaoudi
- Fahd Mahraz

Nous allons présenter ici l'implémentation de la base de données, qui s'est faite en deux temps, et nous verrons ensuite les modifications apportées au modèle de la base de données. Dans un premier temps, les trois membres de l'équipe BDD initiaux l'ont écrite en SQLite, puis, pour les raisons expliquées dans la partie de l'équipe « intégration », Quentin Gerling a dû réécrire cette base de données en PostGreSQL.

SQLite

Initialement et comme indiqué dans notre cahier des charges, nous avions implémenté la base de données de notre jeu en utilisant SQLite, étant donné que nous étions familiers avec ce langage, qu'il était « light weight » et qu'il permettait d'exploiter une base de données sans avoir à passer par un serveur. Malheureusement, une fois l'implémentation terminée, nous nous sommes rendu compte que la façon dont nous avions codé la base de données n'était pas conforme aux attentes de notre encadrant, c'est pourquoi nous avons finalement opté pour un autre langage.

PostgreSQL

Notre chef de projet a trouvé que le langage PostgreSQL était plus adapté pour une multitude de raison. D'abord, il nous permettait essentiellement de mettre notre base de données sur une VM différente pour, conformément aux exigences de notre encadrant, avoir une base de données du type serveur (plutôt que local). En plus, un autre avantage de PostgreSQL par rapport à SQLite est qu'il nous permet d'exécuter les requêtes de manière asynchrone, ce qui optimise les accès à la base de données. Quentin a pu apporter d'autres modifications au code initial, parmi lesquelles l'utilisation de requêtes préparées pour mieux protéger la base de données et éviter d'éventuels comportements malveillants.

Modifications du modèle

Le modèle de la base de données a un peu changé, une grande partie des éléments a été supprimée ou est pour l'instant inutilisée. Le stockage de photos de profil n'est par exemple plus implémenté, car nous l'avons jugée trop cosmétique, et donc pas prioritaire. Cette option pourrait, dans de futures versions du jeu, être réintroduite, en l'adaptant à la base de données actuelle.

Finalement, même s'il était prévu que la base de données soit rapidement implémentée, celle-ci a subi un certain nombre de modifications tout au long du projet pour subvenir aux besoins des autres pôles de développement.

Equipe Menu

Cette équipe est composée de deux personnes :

- Cédric Geissert
- Florian Halm

Notre menu est composé de boutons amenant à différentes parties du menu tout en restant dans la même scène, mais aussi de sliders, Inputfields et toggle. Nous travaillons sur une seule scène en activant/désactivant les différents menus, pour optimiser et économiser de la puissance de calcul du GPU et ainsi être plus fluide et rapide. Nous allons donc présenter les différents points qui caractérisent ce menu.

Navigation

La navigation dans le menu est possible avec la souris et le clavier. Pour le clavier, on a décidé d'utiliser le new input system de Unity. Celui-ci nous facilite énormément la gestion du clavier : en effet, il permet de connaître à l'avance l'ordre possible de navigation des boutons avec le clavier sans n'avoir rien à coder, il est même possible de modifier nous-mêmes cet ordre de déplacement. Avant de découvrir ce new input system, la gestion du clavier était un vrai casse-tête. Pour la gestion de la souris, nous avons décidé d'utiliser le Raycast des boutons pour gérer les détections de survol et les clics. Le Raycast permet de déterminer si le curseur se trouve sur un élément graphique dans la scène (hover et clic). Il accepte les données d'événements du pointeur de la souris en tant que paramètre. C'est d'ailleurs la raison pour laquelle nous l'avons uniquement activé pour les inputfield, toggle et boutons. Ainsi, nous facilitons l'accès aux objets, sans superflus résiduels des autres objets qui ne nous intéressent pas. En effet, préfiltrer la récupération des objets survolés ou cliqués nous permet de réduire drastiquement le tri dans le code pour sélectionner le bon élément survolé parmi tous les autres (sans compter la hiérarchie, c'est-à-dire les parents, enfants, etc. de l'objet en lui-même).

Fonctionnement de l'UI

Lors du survol d'un bouton avec la souris ou le clavier, celui-ci est mis en surbrillance et son texte est légèrement agrandi. Lorsqu'on change de bouton, le bouton préalablement sélectionné reprend son apparence normale. De plus, un trident s'affiche de part et d'autre des boutons larges lors d'un survol (pas sur les petits boutons pour ne pas surencombrer l'interface). Lors du survol d'un bouton d'engrenage (le bouton qui contient le menu pop-up options), son image s'agrandit, et lors d'un clic sur ce bouton, une animation de rotation se joue sur l'image d'engrenage, tant que le menu est ouvert. Entre chaque changement de menu, le dernier bouton sélectionné avant le changement de menu sera sélectionné.

Options

Le jeu possède des options. Il y a deux façons d'y accéder :

- Soit on appuie sur le bouton « Options » se trouvant sur le menu principal : on y trouvera toutes les options du jeu comme : la gestion du son (son et musique) avec un réglage précis du volume qui se fait à l'aide d'un slider, les langues (non implémenté, et donc caché pour l'instant) et un accès au menu des Crédits.

- Soit on appuie sur le bouton d'engrenage présent dans la majorité des menus : celui-ci permet d'accéder aux réglages dits « rapides » du jeu : un pop-up options (sous-menu) s'ouvrira et celui-ci permettra de couper ou lancer la musique et le son du jeu.

Ces deux accès aux options agissent l'un sur l'autre : couper le son dans le menu principal de celles-ci coupera aussi le son dans le menu pop-up, et vice-versa.

Création de compte

Il est possible de créer un compte, permettant par la suite au joueur de se connecter. Celui-ci devra renseigner un pseudo, une adresse mail respectant la casse, une date de naissance (dont l'âge minimum sera géré par un refus côté "back" (serveur et BDD) si l'âge légal n'est pas respecté) et d'un mot de passe. Il devra aussi cocher le toggle confirmant qu'il accepte les CGU. Une fois le compte créé, le joueur sera directement connecté.

Connexion

Le joueur pourra se connecter s'il possède déjà un compte. Celui-ci devra renseigner son pseudo ou son adresse mail, et son mot de passe. Une fois le joueur connecté, son statut changera sur le menu principal : il passera de « Déconnecté » à « Connecté ».

Les boutons « Multijoueur » et « Statistiques » seront accessibles uniquement quand le joueur sera connecté. Le bouton « Jeu Local » est quant à lui accessible sans connexion.

Sélection de room

Le joueur pourra sélectionner une room pour jouer : quand il aura appuyé sur le bouton « Multijoueur », une liste de toutes les room publiques possibles sera affichée et le joueur pourra faire son choix de room selon différents critères comme : la condition de fin de partie, qui est l'hôte, s'il reste encore de la place, etc.

Chaque room possède un ID. Le joueur peut soit double cliquer sur la room choisie, ou alors cliquer une fois sur la room choisie et cliquer sur le bouton « Rejoindre » pour rejoindre la room.

Pour accéder à une room privée, le joueur devra appuyer sur le bouton « Rejoindre par ID » et entrer l'ID de la room qu'il souhaite rejoindre.

Création de room

La création de room est possible : son créateur (qui sera désigné comme l'hôte) pourra choisir si la room sera privée ou publique et le nombre de joueurs maximum possible.

Une fois la room créée, un menu indiquant le bon déroulement de la création de la room sera affiché.

L'hôte (et seulement lui) pourra modifier les paramètres de la room qu'il a créée. Il aura le choix de la condition de fin de partie, le temps par round et les extensions de jeu activées.

Attente du lancement du jeu

Dans le menu de la room créée, l'hôte pourra lancer la partie uniquement quand tous les membres qui l'ont rejoint seront prêts. Un statut est indiqué pour chaque joueur déterminant s'il est prêt ou non à jouer.

La partie du menu (IHM) fut délicate à débuter, car nous n'avions aucune expérience dans le domaine. L'apprentissage de celle-ci était relativement long et fastidieux, mais notre anticipation était la clé de la réussite. En effet, nous avions commencé le projet avant même que la spécification ne soit terminée. Plus précisément,

nous avons cherché l'information, et appris de manière autodidacte le C# ainsi que Unity PC et mobile. Par la suite, nous avons effectué des tests et une pré-alpha, ajoutée ensuite dans l'alpha grâce à une base solide. Nous avons aussi mis en place les différentes structures pour effectuer une intégration simple et efficace, sans alourdir le code post-intégration. Finalement, nous avons débugué et mis en forme les dernières modifications, pour que le "look & feel" de l'utilisateur soit le plus agréable possible.

Equipe Affichage In Game

Cette équipe est composée de trois personnes :

- Gauthier Heimerdinger
- Caleb Mukaya Gakali
- Sarah Emery

Dans cette partie, nous présenterons toutes les décisions que nous avons prises concernant l'affichage du jeu dans une partie, et nous verrons également comment cette partie a été implémentée.

Nos choix

Nous avons choisi d'utiliser le moteur déjà existant Unity. Ce choix nous a permis d'économiser le temps qu'il aurait fallu si nous avions codé notre propre moteur de jeu. Un des avantages était que Unity permet de réaliser un jeu utilisant la 3D et 2D performante. Il permet d'unifier le développement sur différentes plateformes en une seule, l'isolation des assets vitaux de l'utilisateur et l'utilisation de puissants outils, comme les coroutines. Autre avantage pratique, Unity est un moteur répandu, vraiment très bien documenté avec une communauté active, des éléments importants pour des néophytes dans le développement de jeu.

Nous utilisons le même langage pour l'ensemble de notre code, le C#. Le choix d'un langage commun nous permet une meilleure flexibilité dans l'équipe, ainsi qu'une prise en main adaptée pour des individus ayant déjà pratiqué des langages orientés objets. Même si le C# nous est quasiment imposé par Unity, ce choix présente des atouts. Côté technique, le csharp propose tous les avantages d'un langage orienté objet, pattern design, facilité de test, ségrégation des fonctionnalités. Le C# offre comme particularité la possibilité de faire des listeners facilement, une maintenance réduite et une gestion automatique de la mémoire.

Côté affichage

L'écran du jeu affiche toutes les informations essentielles aux déroulements d'une partie. Comme information sur le jeu : les conditions de victoires, le timer, la liste des joueurs ainsi que leurs scores et leur nombre de meeple restants.

Pour le déroulement du jeu en lui-même, le joueur actif est signalé en étant simultanément en haut de la liste et ayant sur son client la bougie allumée. Tous les joueurs ont la vue sur la main du joueur élu. Pour les actions du tour, les positions possibles où poser une tuile sont indiquéess avec la couleur du joueur élu, le joueur élu peut

en sélectionner une pour poser la tuile. Des clics consécutifs sur la tuile posée permettent alors de la faire tourner.

Lors de la pose du pion, le joueur élu peut retourner à la pose de tuiles. De manière similaire aux positions possibles, les slots où un meeple peut être posé sont affichés, et le joueur élu peut les sélectionner. Les scores des joueurs sont mis à jour à chaque changement et les meeples sur la zone sont enlevés du plateau. A la fin, le score final du joueur ainsi que son rang sont affichés.

Il était important d'implémenter le déplacement de la caméra afin de suivre l'avancement du jeu, un écran de score à la fin du jeu indiquant le rang du joueur élu et son score final. Il fallait aussi implémenter un menu de pause permettant au joueur de quitter une partie ainsi qu'une zone de notifications.

Pour le déplacement de la caméra, les déplacements sont possibles avec les touches du clavier ou avec un "click and drag" pour un jeu sur ordinateur, et avec l'écran tactile pour le jeu sur mobile. Il y a une possibilité de zoom avec les touches du clavier ou avec deux doigts sur mobile.

Un écran de score s'affiche à la fin de la partie indiquant au joueur élu son score, son rang dans la partie jouée. Il s'agit d'une fenêtre en avant plan laissant voir le plateau de jeu ainsi que l'évolution de la carte.

Le menu de pause a été implémenté pour permettre à un joueur de quitter une partie ou modifier les réglages du jeu (son, langue, etc.) en cours de partie. Dès qu'il quitte une partie, il revient sur le menu principal du jeu et une notification est envoyée à tous les autres joueurs dans la partie qu'un joueur s'est déconnecté.

Enfin, des boutons permettant d'annuler un placement ou de revenir en arrière, de valider une position, ou de passer au meeple ou à la tuile de droite et de gauche ont été implémentés. Ces boutons sont surtout très utiles pour la version Android, mais ils sont aussi utilisables sur la version Windows.

Fonctionnalités changées ou non implémentées

En général, les photos de profils des joueurs n'ont pas été réalisées.

Comme changement au niveau du design par rapport à la spécification, la pile de tuile n'est plus de taille variable et est devenue un modèle fixe. Les extensions choisies ne sont pas affichées sur la page de jeu. Un meeple/une tuile sont toujours sélectionnés.

Certaines fonctionnalités n'ont pas pu être implémentées. Le plus gênant pour l'utilisateur est l'affichage d'information sur une zone et la mise en valeur des zones à différents moments du jeu. Mettre en valeur une zone est déjà possible dans notre code, mais le lien n'est pas fait avec le fonctionnement classique du jeu. Pour l'affichage d'information, aucune partie n'est prête à part la gestion de l'état d'affichage dans la table. Mettre ces fonctionnalités en place prendrait un minimum de 5h, en moyenne 10h, et au maximum 20h.

L'autre fonctionnalité non implémentée sont les animations. Certaines parties du code sont prêtes à les accueillir, mais le corps en entier serait à faire, sachant qu'elles perturberaient le rythme d'exécution de la partie. Il faudrait compter dans les eaux de 30h pour les implémenter.

Nous souhaitions également implémenter le déplacement automatique dès la pose de tuile d'un joueur adverse, mais cela nécessiterait la réception des coordonnées de la tuile posée sur le plateau et un ajustement de la caméra en fonction de ces coordonnées.

De plus, la seule notification reçue est celle de déconnection d'un joueur. Il aurait fallu en implémenter d'autres telles que le changement du meilleur joueur dans le classement.

La mise en place de ces fonctionnalités nécessiterait une communication entre le back-end et le front-end pour synchroniser les informations chez tous les joueurs d'une partie ; il faudrait compter en moyenne 10h pour les implémenter.

Equipe design

Cette équipe est composée de deux personnes, et a été aidée par Gauthier Heimerdinger pour la réalisation de certains éléments :

- Aya Elmesaoudi
- Sarah Emery

L'équipe de design était chargée de créer les différents designs nécessaires pour le jeu. Elle a été surtout nécessitée en début de projet, lorsqu'il fallait créer les logos, et le sera à nouveau plus en fin de projet, durant les deux semaines avant la soutenance, dans le but d'améliorer les graphismes actuels, notamment ceux des tuiles. Le travail effectué par cette équipe se divise en trois sous parties, qui sont les dessins, les modélisations ainsi que les vidéos.

Dessins

Les dessins de l'équipe de design ont été faits, pour la plupart, sur tablette graphique, à l'aide d'un croquis préalablement réalisé sur papier.

Aya s'est donc occupée de la création du logo du jeu et des différents dessins nécessaires aux publications sur les réseaux sociaux.

Sarah a commencé, au début du projet à réaliser le design des tuiles à l'aide du logiciel Krita, qui sera terminé pour la soutenance du 18 mai. En attendant que ces tuiles soient prêtes, Gauthier a créé une version temporaire de celles-ci ne comportant pas le thème des fonds marins afin de pouvoir tester le jeu dans sa globalité sans être restreint par l'absence de design.

Les dessins réalisés ont ensuite été récupérés par l'équipe de communication, qui a pu s'en servir dans ses publications sur les réseaux sociaux.

Modélisation

Il a été nécessaire de créer des modèles pour les meeples. Gauthier s'en est chargé, à l'aide du logiciel Blender.

Vidéo

L'équipe de design, en lien avec l'équipe de communication, a réalisé plusieurs vidéos, dont la grosse vidéo de publicité rendue en même temps que ce rapport. Aya s'est en grande partie chargée de cette vidéo et des designs qu'elle contient.

Finalement, la grosse difficulté rencontrée dans cette partie a été de mesurer le temps nécessaire à la création d'un élément de design. En effet, pour créer quelque chose d'esthétique, il fallait souvent passer de nombreuses heures à sa réalisation.

Equipe communication

Cette équipe est composée de deux personnes, et a été rejointe par Caleb Mukaya Gakali pour la finalisation de la vidéo :

- Aya Elmesaoudi
- Adham Elbahrawy

Dans cette partie, nous décrirons les différents procédés de communication employés pour faire connaître notre jeu.

Réseaux que l'on a créés (Instagram + Twitter)

Nous avons créé un courriel sur proton pour effectuer la création de compte professionnel Instagram et Twitter. Notre principal objectif a été de partager des photos et vidéos de notre projet, pour mettre en image de manière immersive les coulisses de notre avancée. Ainsi, cela a permis de mettre en avant notre projet auprès de nos abonnés. Nous avons privilégié les interactions dans nos publications et ainsi que le storytelling (dans les descriptions de nos publications, etc.), cela a rendu notre compte plus attrayant. Nous avons utilisé l'outil Business Manager pour nous aider à cibler nos "publicités"/"stories", afin de développer rapidement une communauté avec précision, en utilisant des ciblages en affinité avec notre public. Nos publications sont concises et favorisent les retweets, pour plus de visibilité. De plus nous avons publié les mêmes images que sur notre compte Instagram.

Gain d'abonnés

Nous sommes passés par des pages de jeu afin que nos abonnés soient intéressés par le concept que nous étions en train de publier. Par ailleurs, nous avons également effectué des publicités via nos comptes personnel.

Organisation de la bio

Dans la bio, nous avions mis un lien Linktree (contenant les liens de Twitter, Instagram, et du courriel) pour que nos abonnés aient accès rapidement à nos contacts et nos autres réseaux sociaux. En plus de cela nous avons imaginé la phrase d'accroche « every battle is better with a buddy ».

Story & publications

Au tout début, nous avons effectué des publications et stories temporaires, en attendant d'avoir une base graphique du jeu sur laquelle s'appuyer. Pour cette raison nous ne pouvions pas être systématiquement actif.

Nous avons également mis une story « à la une » qui présente rapidement notre projet, cela permet aux abonnés d'avoir plus d'informations sur nous, tout en leur donnant les informations sur la date du lancement.

Dans nos publications, nous avons essayé de mettre des descriptions captivantes ainsi que des hashtags qui permettront de référencer et lier nos publications à d'autres dans le même thème.

Thème

Nous avons essayé de coordonner les couleurs de nos publications pour respecter notre thème et ainsi attirer le plus d'abonnés possible (Stratégie visuelle). Les choix des filtres et de la bonne luminosité étaient essentiels pour réaliser ceci.

Création du contenu

Les publications présentes sur notre page sont des œuvres qui ont été désignées par notre équipe (logo dessiné et numérisé ainsi que photo prise IRL et montée par nous-même avec réalisation d'une diapo brève de présentation).

Concernant les horaires de publications, nous étions plus actifs durant les créneaux 12-14h et 18-20h afin d'avoir un public disponible (pause de midi et après le travail ou les cours).

À venir

Dans cette partie seront présentés les stratégies de communications qui seront mises en place avant la soutenance du 18 mai.

Flyer

Le flyer nous a servi à mettre en avant notre projet. Ils seront distribués dans la rue, pour effectuer notre "street marketing", et seront également placés dans divers endroits stratégiques pour que les passants les voient et les prennent (au RU ou à la fac, etc.).

Démo du jeu (Twitch etc.)

Une démonstration en direct (live) est bientôt envisageable avec une interaction orale de nos streamers, et des réponses "live" écrites des viewers. Cette démonstration sera enregistrée directement sur Twitch et partagée avec un lien sur Instagram/Twitter.

Une vidéo courte sera diffusée sur nos réseaux sociaux qui annoncera le lancement de notre jeu. Cette vidéo montrera quelques extraits du menu ainsi qu'un extrait d'une partie du jeu pour créer le suspens chez nos abonnés et les pousser à essayer notre jeu.

Finalement dans cette partie, nous avons essayé d'être créatifs dans notre contenu et dans les choix des supports de communication utilisés. Cette partie du projet nous a surtout permis de faire connaître notre jeu et

d'essayer de le vendre au mieux auprès de notre cible (+13ans) en passant principalement par les réseaux sociaux les plus utilisés par cette catégorie de personnes.

Equipe d'intégration

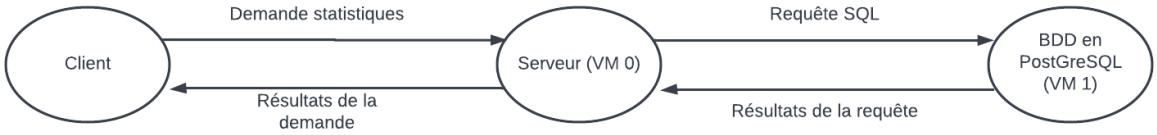
Cette équipe est composée de deux personnes :

- Matthieu FREITAG (responsable)
- Victor Hahnschutz

Dans cette partie, nous allons aborder le travail effectué par l'équipe d'intégration, constituée d'Axel, Quentin, Gauthier et Matthieu. Cette phase d'intégration fut la plus délicate à mettre en œuvre, car c'est durant celle-ci que la majeure partie des problèmes a fait surface. Cette mise en commun a consisté en trois grandes étapes que nous présenterons par la suite : l'intégration entre la base de données et le serveur, puis entre le réseau, le client et le serveur, et enfin entre les différentes parties du système de jeu et le serveur.

Intégration entre la base de données et le serveur

Dès le début, durant la phase d'intégration entre la base de données et le serveur, est apparu un premier problème. En effet, le serveur fonctionnait de manière asynchrone, tandis que la base de données, écrite en SQLite, fonctionnait en synchrone, ce qui signifie que plusieurs appels pouvaient avoir lieu en même temps. Ces deux modes n'étant pas compatibles, il a finalement été décidé, après plusieurs discussions, de réécrire entièrement la base de données en PostgreSQL, en se basant, pour certains points, sur la première BDD écrite. Quentin s'est chargé de cette tâche, dont la durée, entre l'installation de PostGreSQL et la rédaction C# de la BDD, a été d'environ 12 heures. La base de données a ensuite été installée sur la machine virtuelle lui étant attribuée, la deuxième étant réservée au serveur. Le processus de récupération d'éléments du client sur la BDD a ensuite été le suivant, en prenant l'exemple de la récupération des statistiques :

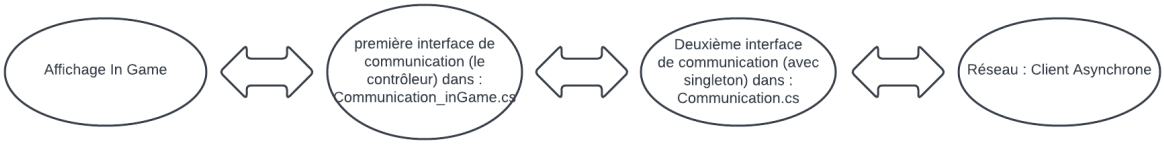


Une fois tous les scripts C# modifiés, les premiers tests de récupération d'informations ont fonctionné directement.

Intégration entre le réseau, le client et le serveur

La deuxième phase d'intégration, celle entre le réseau, le client et le serveur, a également apporté son lot de surprises. Dans un premier temps, malgré la grande quantité de travail de qualité produite par Matthieu dans la partie réseau, il manquait les parties asynchrones, qu'il a fallu ajouter. Le deuxième membre de l'équipe réseau ne réussissant pas à réaliser cette tâche, c'est encore une fois Quentin qui s'en est chargé, tandis que les autres membres de l'équipe d'intégration travaillaient sur d'autres aspects de l'intégration. Quentin a donc

implémenté l'entièreté du code du client asynchrone. Pour se faire, il a créé un singleton, classe qui ne possède qu'une seule instance dans l'application entière. Ainsi, cela a permis d'accéder à cette même instance sans se soucier de devoir être à un endroit précis dans le jeu pour le faire. Afin de transmettre des informations de l'Affichage InGame jusqu'au réseau et inversement, un certain cheminement est effectué, passant par les deux interfaces présentes dans « communication_inGame.cs » (le contrôleur) et « Communication.cs » (le singleton). Le schéma suivant décrit ce cheminement :



Une fois les méthodes du réseau fonctionnelles, il était possible de communiquer en local uniquement, mais pas avec le serveur directement, car la partie permettant de le faire n'était alors pas écrite. En effet, la mise en place du réseau a nécessité de créer un lien avec le serveur ainsi qu'avec la base de données. La plus grosse difficulté rencontrée pour l'écriture de ces méthodes était la nécessité pour le serveur de recevoir des éléments concrets et d'une certaine forme, alors qu'il y avait un grand nombre de choses différentes à prendre en compte et à transmettre, il fallait donc réfléchir à une uniformisation.

Lorsque les méthodes de communication furent mises en place des deux côtés et après quelques tests d'envois successifs rapides, un problème de communication fut détecté. Au moment d'accepter les données, il était possible d'en recevoir plusieurs d'un seul coup si les envois avaient été faits trop rapidement, et les paquets se concaténaient alors. Ce cas a donc dû être géré.

Une autre complication est apparue au niveau de la communication via les threads. Lorsqu'une communication entrait du côté client et qu'il fallait réceptionner les données du paquet, ces dernières ne pouvaient pas être utilisées pour modifier un objet Unity, car ceux-ci ne peuvent l'être que par des threads Unity. Pour corriger cela, il a fallu diviser en deux parties le traitement des données, à l'aide d'une liste placée dans la fonction Update de Unity (qui est appelée à chaque frame du jeu). Cependant ce bug est régulièrement réapparu, même s'il était néanmoins plus facile de le corriger les fois suivantes car ce n'était plus un problème inconnu. Il nécessitait malgré tout de rester vigilant.

Tout au long de cette phase d'intégration, il a été compliqué de gérer la partie réseau. Celle-ci devait très régulièrement être mise à jour et retouchée et représentait une portion de code sensible à la modification des autres branches. La partie système de jeu, qui avait auparavant été codée de façon très théorique, pouvait enfin être testée de façon pratique, ce qui faisait apparaître certains bugs, et demandait de réaliser de fréquentes modifications dans le code du réseau.

Intégration entre les différentes parties du système de jeu et le serveur

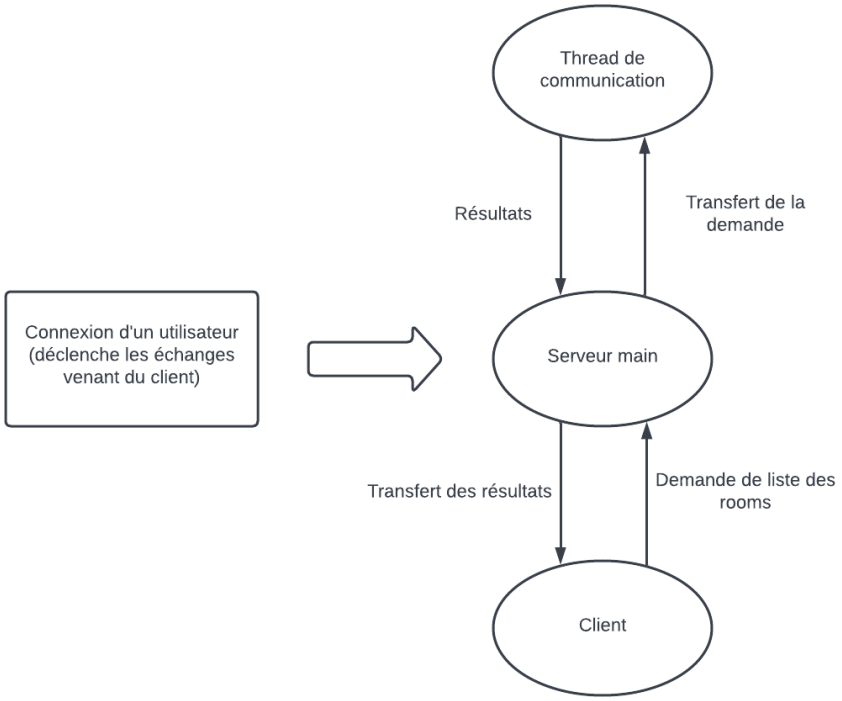
Au début de cette phase, il n'y avait, dans le système de jeu, que la logique du jeu pure (poser des tuiles, meeples, etc.), qui avait été écrite par Justin et Issam. Il fallait donc réaliser un certain nombre de changements, dans le but d'obtenir une communication et une réaction afin de discuter avec le client. Ces changements ont été faits en quatre étapes, qui seront présentées ci-dessous.

Première étape

Dans un premier temps, il fallait se mettre d'accord avec l'équipe de réseau pour effectuer des premières communications très basiques (par exemple la connexion ou la création d'un compte). Heureusement la structure réseau avait été préalablement bien écrite par Matthieu, ce qui a rendu cette étape plus rapide.

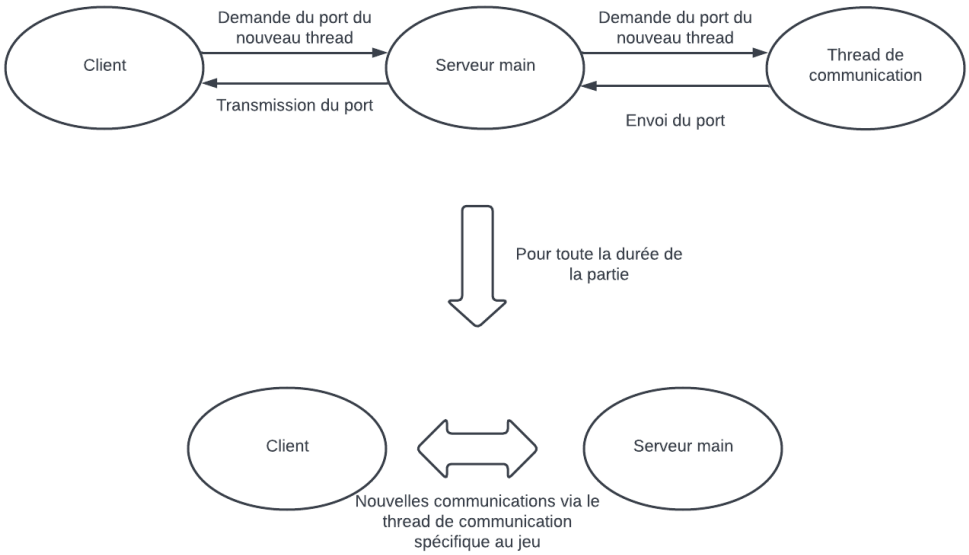
Deuxième étape

Cette communication terminée, il a été possible de commencer l'implémentation en dur du jeu, avec la création de rooms, le lobby, la préparation pré-game et enfin la mise à jour des paramètres. L'architecture de communication de cette partie a été la suivante, de la connexion d'un utilisateur à son accès à la liste des rooms :



Troisième étape :

La troisième étape a consisté en l'implémentation de toutes les communications en jeu (comme la pose des tuiles) et toutes les autres communications nécessaires. Pour passer de la communication pré-game avec le menu à la communication en jeu, nous avons dû changer de thread de communication pour le client, ce qui a été fait de la manière suivante :



Lorsque le client a reçu le port du nouveau thread avec lequel il va communiquer (celui de communication en jeu) il ne va plus communiquer qu'avec lui vers le serveur de jeu jusqu'à la fin de la partie.

Quatrième étape

La dernière étape de cette phase, et donc de toute l'intégration, a été de compléter, peaufiner les détails et débugger. Les différents modes de fin de jeu, la rivière, et d'autres détails de ce type ont également été revus.

Finalement, cette phase d'intégration fut très chronophage car le système de jeu n'avait été créé que sur la théorie, et n'avait pas pu être testé en pratique avant l'intégration. Ces tests ont généré beaucoup d'erreurs et ont mis au jour des parties de code non implémentées pourtant nécessaires. Il a donc fallu faire preuve de beaucoup de rigueur et de communication entre les différents pôles de développement, et donc entre les membres de l'équipe d'intégration, dont chacun représentait un pôle (Gauthier l'affichage In Game, Matthieu le réseau et Axel et Quentin le système de jeu).

Tout au long de cette intégration, l'affichage In Game a été adapté par Gauthier, pour l'adapter aux besoins des autres parties. Il a également participé à la mise en place de la communication entre l'affichage et le système côté client. Cela prend en compte la gestion de l'avancement du jeu sur les différentes structures de données ainsi que la coordination sur l'envoi et réception de paquets.

Plus globalement, l'intégration du jeu fut une étape très éprouvante pour l'équipe qui en était chargée, en raison du très grand nombre de problèmes rencontrés et du peu de temps qu'il restait pour la réaliser. Elle aura malgré tout pu être terminée à temps, au prix de certains autres examens dans d'autres matières. Une intégration avec plus de membres n'aurait pas forcément aidé, mais il aurait été possible de gagner beaucoup de temps avec du code plus complet respectant les spécifications, qui elles-mêmes auraient pu être mieux écrites.

Parties individuelles

Issam Eizouki

Dans un premier temps, nous avons réalisé le cahier des charges, dont chaque équipe a rédigé une partie. Par la suite, d'après ce premier document, nous avons pu travailler sur la spécification. Pour la rédaction de celle-ci, je me suis occupé des aspects suivants : serveur principal (avec Axel), Threads de communication (avec Axel), Thread_serveur_du_jeu et Thread_client.

Cette partie a été très importante, car c'est elle qui nous a permis de concevoir en détail la logique du jeu, et ce qui était attendu du code que nous allions par la suite écrire. Ainsi, notre spécification associait une logique de jeu écrite en français à des morceaux de pseudo-code dont le but était de définir et préciser celle-ci. La mise en commun de ce travail de spécification a ensuite été réalisée par Axel.

Une fois la spécification terminée, nous avons démarré l'implémentation. Je me suis occupé de plusieurs tâches. J'ai commencé par définir les différentes classes (joueur, Thread_serveur_du_jeu, …), puis j'ai écrit la fonction LireXml2, dont le rôle est de lire les données (les tuiles, leurs slots ainsi que les terrains) depuis notre fichier XML. J'ai par la suite codé la méthode RandomSortTuiles qui génère aléatoirement la liste des tuiles de la partie, et dont l'algorithme m'a été proposé par Quentin. Enfin, j'ai crée la classe Thread_serveur_du_jeu, avant de réaliser des tests unitaires de mes méthodes sur Unity grâce aux Unity-tests.

Durant ce projet, j'ai pu découvrir beaucoup de nouvelles choses, mais j'ai également rencontré certains obstacles et difficultés, que je vais aborder. Concernant l'implémentation, j'ai rencontré une première difficulté, qui a été que je n'avais presque jamais travaillé sur Unity et que je n'avais pas de bonnes connaissances en C#. Le démarrage de l'implémentation n'était donc pas facile, mais au bout de quelques heures, je me suis familiarisé à ce nouvel environnement de travail. Un autre contretemps que j'ai eu pour l'implémentation a été que la phase d'intégration a débuté tardivement par rapport à ce qui était prévu initialement, raison pour laquelle je me retrouvais parfois face à des difficultés. Ce retard m'a par exemple contraint à implémenter la méthode Random_sort_tuiles(), dont le fonctionnement nécessite une méthode de la BDD, et son test unitaire associé, sans que celle-ci ne soit encore faite (méthode qui récupère les ID des tuiles ainsi que leurs probabilités d'apparition).

Pour finir, je vais présenter mon ressenti sur le projet et sur mon équipe. Durant les premières semaines, nous avons passé beaucoup de temps sur la spécification, ce qui m'a, au premier abord, un peu stressé. Finalement, j'ai réalisé pendant l'implémentation qu'une telle spécification détaillée facilite effectivement la réalisation des tâches et l'intégration par la suite. Aussi, au départ, le fait de me retrouver dans une équipe de 14 personnes avec une multitude d'avis sur chaque détail du projet était un peu étrange et nouveau. Je ne voyais pas comment réussir ce projet. Mais tout compte fait, à l'issue de plusieurs réunions, nous avons réussi à nous mettre d'accord sur l'organisation du projet, et sur la prise de décisions, dont les principales ont été votées par l'ensemble de notre équipe. J'ai cependant trouvé cela fatiguant, car les membres de l'équipe ne sont pas tous compétents dans les mêmes domaines, et il était peu judicieux selon moi que certains prennent des décisions sur des sujets qu'ils n'avaient pas à implémenter.

Heureusement, après la division de notre équipe en sous-équipes, les décisions sont devenues plus fluides et correspondaient mieux à nos attentes. Elles ont ainsi été écrites dans les rapports de décisions hebdomadaires.

Adham Elbahrawy

Étant le responsable de l'équipe en charge de la base de données, j'étais surtout impliqué dans la rédaction et la mise en forme de la spécification de la BDD du jeu. Mon rôle était aussi de répartir les tâches entre Fahd, Aya et moi-même pour l'implémentation en SQLite et l'intégration avec Unity de cette BDD. Malheureusement, le code que nous avons fourni n'a pas été conforme aux mesures de sécurité et d'optimisation exigées par notre encadrant. Par conséquent, Quentin, notre chef de projet, s'en est inspiré pour implémenter et peaufiner la nouvelle BDD en PostgreSQL.

Au niveau du code, j'étais chargé de la création des fonctions qui auraient interrogé la base de données et exploité les données fournies afin de les mettre à jour. J'avais également comme tâche l'implémentation des contraintes dynamiques sous forme de fonctions, afin d'éviter l'utilisation de triggers. Par ailleurs, j'ai aussi travaillé avec Fahd pour déboguer le fichier .db de la BDD qu'il avait créé.

Une fois notre travail sur la BDD terminé, j'ai été assigné avec Aya à l'équipe de communication, où j'ai travaillé sur l'amélioration des comptes de notre jeu sur les réseaux sociaux, notamment en modifiant les biographies et en créant notre Linktree. De plus, nous avons travaillé sur la création de posts et de stories Instagram pour annoncer les progrès de notre groupe et communiquer les différentes dates clé. Nous avons aussi revisité les stories « à la une », surtout pour rajouter une story « à propos de nous » majoritairement faite par Aya. La communication devant accompagner le projet jusqu'à sa sortie, mon travail sur cette partie est censé continuer jusqu'à la soutenance du projet. Enfin, j'ai créé le manuel/document utilisateur de notre jeu qui servira d'accompagnement aux joueurs qui ne connaissent pas du tout Carcassonne, le jeu de base duquel est inspiré le nôtre. Le manuel est assez concis car le jeu contrôle ce que peut faire un joueur, de sorte qu'il ne puisse réaliser que les actions qui lui sont proposées. En outre, j'ai évité de reprendre certains graphismes de la version du jeu que l'on possède actuellement, car il est possible que le design soit amélioré avant la soutenance.

Enfin, en ce qui concerne mes sentiments personnels, j'ai trouvé le fait de travailler sur un projet avec tellement de personnes peut être un peu difficile à priori. En effet, il n'a pas été facile pour moi d'être en communication constante avec l'entièreté du groupe pour les mettre à jour sur mon avancement et vice versa. En outre, de mini débats surgissait de temps en temps concernant notamment mes choix d'implémentation. Cependant, de tels problèmes n'ont pas causé un dysfonctionnement sur l'intégralité du travail et finissaient souvent par se résoudre tous seuls. D'ailleurs, malgré ces petites difficultés, nous avons pu travailler et rendre un bon résultat dont je suis fier en fin de compte.

Aya Elmesaoudi

Dans le cadre de ce projet, j'ai intégré dans un premier temps l'équipe s'occupant de la base de données, et lorsque cette dernière fut terminée, j'ai rejoint l'équipe de communication. En parallèle, j'ai également participé au design du jeu, en travaillant principalement sur celui du menu.

Ainsi, pour la base de données, je me suis occupée de la rédaction de la spécification fonctionnelle et technique avec Adham. Nous avons ensuite effectué les changements nécessaires, à la demande des autres membres. Lorsque cette tâche fut terminée, nous avons pu commencer l'implémentation de notre base de données, où mon rôle a été de créer les tables, de réaliser une partie des fonctions d'insertion, de suppression, ainsi que la création de quelques fonctions demandées par le système du jeu et qui avaient pour but principal de récupérer des champs. Par la suite, notre équipe BDD et le chef de projet avons réadapté le code dans un autre langage (de sqLite à PostgreSQL), en adéquation avec les besoins serveur.

Je vais maintenant décrire mon implication dans l'équipe de communication. Avant la réalisation des spécifications et l'implémentation du jeu, j'avais consacré du temps à créer le design de notre logo, qui nous sert maintenant pour les publications sur les réseaux et que l'on utilise pour représenter notre jeu. J'ai participé à la création des comptes sur les réseaux sociaux (Instagram principalement) et ai donné des idées de contenu (par exemple, quelles images choisir, les descriptions sur les publications, dans quel ordre poster les différents éléments, etc.). J'ai également réalisé une story « à la une », qui présente notre projet rapidement. Nos publications, descriptions, biographies et stories ont été essentiellement rédigées afin de convenir à un public cible de treize ans et plus, et cela se remarque dans le langage utilisé et les outils employés dans nos contenus (phrases d'accroche, filtres, hashtags, etc.). Nous avons de plus rappelé dans chacune de nos publications le thème du projet (fond sous-marin). Dans les descriptions que j'avais écrites et dans la story de présentation, j'ai essayé d'être brève pour rester captivante et veiller à ce que les autres ne s'ennuient pas en visitant notre profil. Je me suis aussi occupée de la vidéo finale de communication, qui montre brièvement l'interface de notre jeu et qui invite les autres à l'essayer.

Pour le refactoring du menu, Cédric et moi-même nous sommes mis d'accord et avons travaillé ensemble pour mettre en place un design avec des sprites, une musique de fond, un son de clic ainsi qu'un background animé. Pour finir, nous avons débogué certaines parties du code du menu ensemble, et avons ajouté quelques fonctionnalités pour chaque page du menu car cette implémentation a demandé une restructuration complète.

Tout au long du projet il y a eu une bonne entraide et cohésion dans le groupe. En effet, les différents membres ont toujours été là pour aider les autres, quand il le fallait. Ainsi, ce projet m'a permis de découvrir le travail en groupe, ce qui me servira d'expérience dans le futur. Il m'a également appris à travailler en continu, à savoir respecter les directives du chef de projet et les deadlines imposées par le Gantt. Enfin, dans le cadre de ce jeu, j'ai pu m'exercer à apprendre par moi-même, sans cadre universitaire pour le faire (Unity, C#, sqLite).

Sarah Emery

Dès le début du projet, j'ai été beaucoup impliqué dans les choix de départ, la rédaction du cahier des charges et la réalisation de notre spécification dans l'équipe d'affichage In Game. J'ai également pu m'impliquer dans la réalisation des premiers designs.

Aux débuts de la phase d'implémentation, j'ai pu commencer à comprendre le fonctionnement de Unity, et à démarrer les prefab des tuiles, mais rapidement après le début de cette phase a démarré pour moi une session de trois semaines d'examens intenses de mi-semestre dans ma seconde licence, en Physique. J'ai donc malheureusement dû mettre le projet intégrateur temporairement de côté afin de me concentrer sur ces examens. Je suis finalement tombée malade la semaine suivante, m'empêchant de venir à la réunion hebdomadaire de l'équipe, et diminuant mes capacités de travail.

C’est seulement à ce moment-là que j’ai pu lentement commencer à rattraper mon retard, d’autant plus difficile à rattraper que mes camarades avaient bien avancé sur leurs parties. J’ai donc bénéficié de l’aide de Gauthier, dont les explications m’ont permis d’accélérer le processus d’apprentissage et de compréhension de certains éléments clés de notre jeu (Collider, Transform, etc.). En vue du retard pris, j’ai cédé la tâche de création des tuiles à Gauthier, et me suis occupée de la réalisation du plateau, avec ses conseils.

Pendant que la phase d’intégration était en cours, j’ai travaillé de mon côté après m’être renseignée auprès de Cédric concernant la création de boutons, sur l’élaboration de quatre boutons, permettant d’annuler, de valider ou de se déplacer sur les possibilités de tuiles et meeples à l’aide de la souris. L’usage de certaines touches du clavier permettant de réaliser les mêmes actions.

Finalement, comme indiqué au départ, je me suis chargée de la rédaction des parties communes du rapport, et de la correction et relecture de toutes les parties individuelles et par pôles du rapport. J’ai en outre aidé à l’organisation des autres documents à rendre et ai également participé à leur relecture. Cette partie fut très fatigante en raison de la monotonie qu’elle représentait, mais du temps pourtant nécessaire pour la réaliser.

Durant les deux semaines précédant la soutenance, je m’occuperai de réaliser un design de tuiles plus élaboré et plus adapté à notre thème sous-marin initial.

En définitive, ce projet aura été pour moi un vrai défi en raison de mon double cursus universitaire, m’empêchant d’avoir une présence régulière à certains moments cruciaux. J’ai travaillé de manière très intense et éprouvante en début et surtout en fin de projet, lorsque cela était possible pour moi, afin de compenser les périodes d’absence. J’avais par ailleurs quelques appréhensions en démarrant un projet d’une telle envergure tant du côté humain, avec une équipe de quatorze personnes, que du côté matériel, avec les nouvelles technologies à apprendre seule et sans encadrement de la faculté. J’ai finalement été vraiment surprise par les qualités de persévérance et de résignation de certains de mes camarades, notamment vers la fin du projet, lorsqu’il a fallu rattraper notre retard au moment de l’intégration. Je retiens de ce projet beaucoup de stress et de frustration, de ne pas avoir pu faire plus que ce que je produisais déjà. J’ai malgré tout fait de belles rencontres, et ai apprécié découvrir le C# et le fonctionnement de Unity. Cette expérience riche en émotions et en leçons me permet de très bien clôturer la licence d’Informatique.

Justin Filipozzi

Durant la réalisation de ce projet, j'étais chargé du développement du système de jeu, en équipe avec Axel Grimmer et Issam Eizouki. Nous devions nous assurer du bon fonctionnement du back-end, à savoir de la gestion du plateau et des tuiles. C'est-à-dire le respect des règles, le comptage des points, etc. Lors de la conception de la structure de données, mon équipe et moi-même nous fîmes grandement aider par Gauthier Heimerdinger, qui nous a fait part de ses idées pour conceptualiser une structure de données efficace. J'étais heureusement déjà familier avec le langage C#, ce qui me permis de gagner beaucoup de temps et de pouvoir directement coder. Je ne connaissais Unity que de nom, mais cela ne m'a finalement pas posé problème car ma partie était centrée sur la manipulation de structures de données, ce qui n'avait donc aucun rapport avec l'interaction homme-machine ou la partie graphique du jeu. Lorsque l'équipe commença à écrire le code, je me suis rapidement mis au travail et j'ai pu, sans problème majeur, finir les méthodes fondamentales dont le jeu a

besoin pour son bon fonctionnement. Ensuite, je pris du retard lorsqu'il a fallu écrire les tests unitaires. Ceci fut dû à la complexité de la structure de données des tuiles qu'il fallait (à l'époque) simuler pour tester les méthodes que j'avais réalisées.

Vint le moment de l'intégration et de la mise en commun du code de toutes les équipes du projet. En cette période délicate, je n'avais au début pas de mission claire, je tentais d'aider là où je le pouvais. Devant la difficulté de cette tâche, il fut décidé que Gauthier et moi devions réaliser une version du jeu ''locale'', choix qui fut encouragé par l'arrêt des machines virtuelles pendant quelques jours. Nous avons donc assemblé le frontend et le backend, ce qui donna lieu à une version jouable.

Finalement, durant ce gros projet, il y eu des périodes où je me sentais à ma place, et d'autres durant lesquelles je n'avais pas l'envie de m'investir. J'ai beaucoup aimé développer le jeu, mais je me suis cependant ennuyé pendant les choix de design. Je n'ai jamais aimé travailler en équipe, et ce projet n'a pas fait exception à la règle, les tâches étant pourtant bien réparties et me permettant de travailler seul sur mes fichiers. Je suis néanmoins reconnaissant de l'aide que Gauthier m'a apportée lors de la conception de la structure de données et en fin de projet, quand il a falluassembler le front et le back.

Matthieu Freitag

En tant que responsable "Réseau", j'ai bien évidemment participé à l'élaboration du cahier des charges de cette même partie et j'ai également maintenu un lien très fort avec le groupe "Système de jeu" dans l'objectif de nous coordonner. La branche par défaut du groupe "Réseau" étant la branche "reseau", j'y ai rédigé un README.md et intégré un CI/CD. Ce dernier permet de vérifier que le nouveau programme compile, que le style de code est respecté, que les tests unitaires sont validés et affiche également le résultat des tests unitaires ainsi que la valeur du coverage dans le README.

Avec mon collège Victor, nous avons développé en commun la classe "Packet" et avons développé différentes méthodes ou parties possédant des fonctionnalités triviales.

Plus individuellement, j'ai développé des méthodes afin de sérialiser/désérialiser une instance de la classe "Packet", de découper un paquet en une liste de paquets et de concaténer une liste de paquets en un unique paquet. J'ai également mis en place la récupération des informations locales/distantes pour le serveur/client depuis des fichiers de configuration au format JSON, des timers sur les sockets afin d'éviter les attentes infinies et un système de gestion des erreurs avec une énumération et des try/catch. J'ai également nettoyé, reformaté et documenté notre code en anglais au format XML.

Étant l'un des administrateurs des VMs, j'ai également participé à l'organisation de celles-ci. Nous avons appliqué un mot de passe à l'utilisateur "root". Nous avons retiré l'utilisateur "ubuntu" de la liste des super-utilisateurs après avoir créé un nouveau super-utilisateur "pro", possédant un mot de passe différent de "ubuntu" auquel uniquement les admins ont accès. Ensuite, j'ai mis en place un moyen d'avoir le programme du serveur qui fonctionne de manière continue, même si celui-ci s'arrête ou que la VM redémarre. Finalement, je me suis régulièrement chargé de récupérer les modifications apportées au programme depuis git afin d'effectuer des tests avant de valider l'intégration.

J'ai également participé à l'intégration finale, c'est-à-dire à la mise en commun des différentes parties. À ce titre, j'ai dû intervenir dans différentes parties du code, bien que certaines ne concernent pas le réseau, afin d'éditer, de corriger, d'améliorer ou de développer des parties nécessaires mais malheureusement absentes. J'ai alors

principalement développé sur la branche “serveur” et j’ai, par exemple, développé des méthodes permettant de lier un paquet reçu au code du groupe “Système de jeu”. Outre le développement, c’était une étape requérant énormément de réflexions et discussions.

A titre personnel, j’ai trouvé cette expérience très enrichissante, tant d’un point de vue individuel que d’un point de vue collectif. Tout d’abord, j’ai pu découvrir un tout nouveau langage. Ensuite, j’ai pu approfondir mes compétences en communication client/serveur, mais aussi en DevOps et en machines virtuelles. Concernant le collectif, j’en garde un très mauvais souvenir. Bien que j’aie pu développer des compétences en gestion d’effectifs (en tant que responsable “Réseau”), je me suis très vite rendu compte de la difficulté de la gestion d’équipe.

En effet, il est parfois compliqué de composer avec les lacunes techniques de certains, mais il l’est d’autant plus lorsque ces derniers ne font preuve d’aucune motivation. Je retiendrai principalement les sacrifices personnels nécessaires afin de mener à bien ce projet.

Cédric Geissert

J’ai commencé très tôt le projet, avec une pré-alpha du menu. En effet, lors de la mise en place de la spécification, j’ai profité d’être en début de semestre et d’avoir plus de temps libre pour apprendre le C# et Unity en parallèle, tout en mettant en place une grande base pour le menu, qui a évolué durant du semestre. Le début était fastidieux et chronophage, notamment pour la recherche d’informations et l’héritage de classe. J’ai cherché pendant plusieurs jours une manière efficace de faire, ce qui m’a permis par la suite d’avoir une bonne base dans la hiérarchie de mon code pour l’héritage, que nous avons utilisée tout au long du projet jusqu’à la fin de celui-ci.

Par la suite, pour essayer de commencer l’intégration au plus vite, je me suis appuyé sur la spécification effectuée auparavant, ainsi que sur le premier jet du menu créé par Caleb via Figma à l’aide de prototypes de menus imaginés préalablement en réunion. J’ai donc commencé par effectuer le menu “principal”, “connexion” et "création de compte”. Ensuite, j’ai ajouté le menu “options” avec ses fonctionnalités. Une fois cela achevé, j’ai inséré un background fixe puis animé, avec de la musique et des bruits de clics, à l’aide d'event system. J’ai malencontreusement rencontré quelques difficultés pour synchroniser le hover entre inputfield et boutons, notamment lors de la réinitialisation de la couleur de base en cas de hover. Initialement j’avais mis une couleur, que j’ai pu remplacer une fois plus à l’aise, par d’autres éléments. Ainsi, j’ai mis en place des boutons comportant des graphismes (sprites) avec l’aide d’Aya (membre de l’équipe de design). De plus, nous avons entièrement “refactoré” le design et la totalité des fonctionnalités/apparences de tous les menus, pour les rendre esthétiques, et surtout user friendly.

J’ai ensuite voulu synchroniser la gestion clavier et souris pour optimiser la navigation du menu (c’est-à-dire hover un bouton puis cliquer sur une flèche directionnelle pour sélectionner un autre bouton, etc.). En effet, une fois que j’avais mis en place une gestion de souris avec un hover fonctionnel, j’ai voulu essayer de coder la gestion clavier, mais ce n’était pas très optimal. Mon binôme, Florian, a finalement trouvé une alternative, directement avec le nouvel “input system” de Unity, ce qui nous a permis d’allier clavier et souris de manière optimale, sans lourdeur dans le code.

Une fois les menus essentiels mis en place, je me suis concentré sur les menus restants, ainsi que diverses optimisations et correctifs pour faciliter l’intégration continue. En effet, un mois après avoir achevé ces tâches,

l'intégration continue a commencé, et s'est déroulée avec aise, car mon code était subdivisé de manière claire et concise. En effet, il fallait remplacer mon code "dummy" par le networking liant le serveur et la BDD pour pouvoir envoyer les données. Cette partie fut effectuée de manière collaborative.

Post-intégration, il a fallu rectifier quelques bugs, pour avoir une bonne homogénéité, ainsi qu'une certaine fluidité permettant un bon ressenti "in game" dans l'objectif de faire rêver le joueur, afin qu'il puisse s'amuser ! Une dernière problématique fut la mise en place de la version Android. Pour cela, il fallut installer Unity Remote 5, et lier l'ordinateur et le téléphone pour réaliser les tests. Nous pensions que le raycasting ne fonctionnait pas sur téléphone, mais le problème venait finalement de l'application Unity Remote 5 qui n'est pas à jour avec le nouvel "input system" de Unity sur PC. La solution fût de simplement téléverser l'APK sur le téléphone sans utiliser leur application PC-Android.

J'ai également effectué une partie communication, notamment la création de comptes professionnels sur Twitter et Instagram, ainsi qu'un courriel proton. J'ai alors réalisé les premières publications (in game, etc.) et ai encouragé un apport de followers.

Globalement, ce projet était très intéressant et instructif, j'ai passé de très nombreuses heures pour réussir au mieux à finaliser ce jeu, et également atteindre une satisfaction personnelle. En effet, j'aime quand un projet est bien fait, même si je dois y passer plus d'heures que nécessaire. De plus, l'apport en connaissances que cela me procure est non négligeable. J'ai aussi appris de nouvelles méthodes de travail et ai expérimenté la mise en place et la cohésion du travail de groupe à travers différents aléas.

Quentin Gerling

Au cours de ce projet, j'ai essayé de réaliser au minimum cinq heures par semaine (en dehors d'une semaine de maladie et une autre semaine liée à des projets dans d'autres matières). J'ai principalement appliqué mon rôle de chef de projet dès le départ afin d'avoir l'organisation la plus optimale possible pour que les autres membres ne se retrouvent pas coincés au bout de quelques semaines ou mois.

Etant donné que personne ne paraissait motivé pour s'occuper des machines virtuelles, j'ai pris en main ces dernières avec deux autres membres qui me paraissaient susceptibles d'en avoir besoin : Matthieu, qui s'occupe du réseau et Axel, qui s'occupe du fonctionnement du serveur du jeu. Avec Matthieu, nous avons mis en place un mot de passe sur l'utilisateur "root", et dans la même optique, nous avons créé un nouvel utilisateur "pro" qui a récupéré tous les droits de l'utilisateur "ubuntu" et qui a aussi hérité d'un mot de passe. L'utilisateur "ubuntu", lui, a perdu tous les droits pour augmenter la sécurité sur le serveur. Ensuite, j'y ai installé différents outils pour que les autres membres puissent travailler dans les meilleures conditions. Notre projet étant développé en C#, la première machine virtuelle possède un compilateur pour ce langage et la deuxième possède PostgreSQL pour accueillir les communications asynchrones avec la base de données. Finalement, j'ai configuré le pare-feu pour protéger les ports des serveurs.

Lors des dernières semaines, je n'ai pas eu d'autres choix que de participer au développement. J'ai modifié les scripts C# de la base de données pour qu'elle réponde mieux à nos besoins qui avaient alors évolué. Je me suis occupé de l'implémentation de la communication asynchrone du côté client car le membre chargé de celle-ci était bloqué et n'arrivait pas à en saisir le fonctionnement, pendant que son partenaire était occupé par d'autres

tâches tout autant importantes. J'ai aussi soulagé l'un des membres de l'équipe de système de jeu qui avait des fonctionnalités à implémenter plus complexes que prévu initialement.

Enfin, j'ai participé à l'intégration finale, où j'ai développé des fonctionnalités manquantes et mis en lien les implémentations des différents groupes.

Malgré tous les problèmes que j'ai rencontrés durant ces quinze semaines, je garde un ressenti positif sur cette expérience. En effet, les difficultés d'ordre humain ont été nombreuses, entre un membre qui a malheureusement eu des problèmes familiaux, ceux qui ne veulent pas s'impliquer dans le projet et finalement ceux qui rencontrent des difficultés lors de la compréhension ou exécution des fonctionnalités à développer. Toute cette gestion humaine m'a posé un grand nombre de problèmes pour une première expérience en tant que chef de projet.

Cependant, cela m'a permis de revêtir la casquette de développeur pendant les deux dernières semaines, j'ai donc pu élargir mes capacités en C# et surtout apprendre l'outil principal du projet, Unity. Pour finir, j'aimerais remercier les camarades qui, comme moi, ont fait un trop grand nombre de sacrifices pour avoir une version respectable du projet et du rapport durant ces dernières semaines.

Axel Grimmer

Ma partie consistait à concevoir et implémenter l'architecture interne du serveur et sa communication avec le client. L'idée principale était de fragmenter le travail du serveur en différents threads, afin de minimiser le temps de réponse et de calcul. J'ai donc travaillé avec l'équipe du système de jeu et celle chargée de la partie réseau, puis rédigé en collaboration avec Issam la spécification serveur. Ceci étant fait, j'ai pu commencer à implémenter l'architecture et le découpage en différents threads permettant de gérer les parties.

Je me suis ensuite attelé à développer l'ensemble des échanges au sein du serveur lorsque le code de la logique du jeu et l'architecture réseau (pour échanger des données) furent terminés. Ces échanges comprennent tous les différents types de communication que doit recevoir et envoyer le serveur, la mise à jour des paramètres des parties qu'il gère et l'intégration du code de la logique du jeu produit par mes camarades. J'ai pu disposer de l'aide de Matthieu pour la rédaction de quelques méthodes ainsi que pour l'organisation des méthodes appelées par le réseau.

Après cela je me suis tourné vers l'intégration, aidé par quelques membres du groupe réseau, du groupe affichage ingame et de notre chef de projet. Nous nous sommes concentrés uniquement sur la mise en commun et l'intégration des parties entre elles : tester toutes les communications entre le serveur et le client, régler les différents bugs, tester les fonctionnalités, etc. En parallèle de cela, j'ai continué l'implémentation du reste des méthodes serveur afin de rajouter et compléter les dernières fonctionnalités et paramètres.

Pour terminer, je dirai que ce projet de grande envergure, qui a nécessité le travail de plusieurs personnes, m'a permis de beaucoup apprendre, notamment concernant la répartition des tâches et l'anticipation. Je pense avoir grandement manqué de clairvoyance dans la prévision des méthodes et des implémentations nécessaires pour le système de jeu, ce qui s'est bien ressenti lors du sprint d'intégration des dernières semaines. Pour nuancer mon propos, j'avoue tout de même garder un souvenir un peu mitigé. Il m'a permis de "grandir" en

ce qui concerne les méthodes de travail et manières d'interagir au sein d'une équipe. Mais les dernières semaines furent particulièrement éprouvantes de par la charge de travail. Heureusement, j'ai pu être épaulé par mes camarades d'intégration, mais il m'est impossible d'ignorer un sentiment amer, provenant d'une quantité de travail trop inégale entre les différents acteurs du projet.

Victor Hahnschutz

Durant ce projet j'ai fait partie de l'équipe chargée du réseau, et me suis principalement occupé du développement du code en local. Après avoir participé à la rédaction de notre spécification réseau avec l'aide des équipes de système de jeu et in game, j'ai débuté mon travail avec la prise en main du code sélectionné comme base afin de pouvoir le modifier plus facilement par la suite.

J'ai également participé au développement du client synchrone / serveur asynchrone et je me suis beaucoup axé sur les méthodes. J'ai créé le squelette des méthodes utilisées après la réception d'un paquet. Pour cela, j'ai récupéré les informations du paquet entrant et les ai transmises aux méthodes concernées. Celles-ci retournant ensuite le paquet sortant adéquat, selon nos besoins du moment. Ce squelette a ensuite été modifié, mis à jour et complété par mes collègues afin de répondre aux besoins de notre jeu, qui évoluaient constamment.

Dans la partie réseau, j'ai travaillé sur le champ data du paquet, que j'ai implémenté en un tableau de chaînes de caractères. J'ai dû pour cela modifier la fonction split qui était utilisée pour qu'elle fonctionne avec. J'ai travaillé sur plusieurs autres petites modifications mais certaines ont été supprimées à terme puisque des mises à jour plus importantes les rendaient obsolètes. Lors du passage du client en asynchrone, j'ai eu plusieurs problèmes (en cause, le comportement du client et des paquets) que je n'ai pas réussi à résoudre seul.

Au début du semestre je voulais travailler sur la partie système de jeu qui me plaisait beaucoup. Cependant, un certain nombre de membres du groupe s'y intéressaient aussi, donc je me suis redirigé vers le réseau. Malgré mes faibles connaissances dans ce domaine, j'ai beaucoup apprécié travailler sur ce projet, et cela m'a permis de renforcer mes compétences en réseau. Nous avons rencontré de nombreux problèmes au cours du développement, ce qui est normal pour un tel projet. Heureusement, nous avons beaucoup travaillé pour trouver des solutions à chaque problème.

Florian Halm

Tout d'abord, j'ai rendu possible un affichage adaptatif du jeu pour toutes les tailles d'écrans d'ordinateurs et ai amélioré la structure des fichiers (héritage entre les classes). J'ai implémenté diverses fonctionnalités, en commençant par un menu déroulant pour les options, qui est accessible dans chaque menu. J'ai ensuite trouvé un système pour afficher/cacher les mots de passe, et un autre système qui permet la bonne sélection d'un bouton lors d'un changement de menu. De la même façon, j'ai pu effectuer l'affichage des salles de jeu (room) disponibles. J'ai aussi créé la base et la structure des menus crédits, création de room et paramètres de room. Lors de l'intégration de notre menu avec les autres sous-équipes (système de jeu, réseau), j'ai fait en sorte qu'il soit possible de récupérer les valeurs choisies par le joueur.

J'ai aussi beaucoup travaillé sur les événements nécessaires à la navigation du menu, et plus particulièrement sur les manipulations avec clavier. De plus, j'ai aidé à la résolution de plusieurs bugs. Tout d'abord, sur les inputfields, avec la récupération de la chaîne de caractères écrite, et la correction de certains bugs de survol. Mais j'ai surtout aidé à résoudre un problème survenu lors de la mise en place du build Android, où nous avons utilisé l'application Unity Remote 5, une application qui permet de tester le jeu sur son téléphone, tout en ayant encore l'inspecteur de Unity visible sur le PC. Ce problème était que les événements de toucher avec le doigt ne fonctionnaient pas comme ils auraient dû (les événements de toucher sont considérés comme des clics de souris par Unity). Nous avions alors pensé devoir recoder tous nos événements pour faire fonctionner notre menu sur mobile. En réalité, le problème ne venait pas de notre code mais de l'application : en effet, nous utilisons le New Input System de Unity pour gérer nos événements, mais celui-ci n'est pas encore pris en charge par l'application. J'ai alors trouvé un patch pour corriger cela. Ce patch nous a permis d'économiser des dizaines d'heures de travail.

En conclusion, ce projet m'a beaucoup plu sur l'aspect collaboratif avec la découverte de la programmation en groupe et l'utilisation avancée de Git, mais aussi sur l'aspect technique : le moteur de jeu Unity et le langage C# étaient pour moi des technologies complètement nouvelles, le fait de devoir se renseigner là-dessus et d'apprendre sur le tas a plutôt été une bonne expérience. De plus, étant novice dans la gestion de groupe et la gestion des aléas, ce projet m'a apporté une bonne expérience du travail collaboratif et cela me sera sûrement utile dans le futur.

Gauthier Heimerdinger

Mon rôle principal concernait l'affichage de la partie en elle-même, mais j'ai dû participer à de nombreux aspects du projet.

Au tout début, avant même la mise en place de la spécification et la rédaction du cahier des charges, j'ai commencé à réfléchir au modèle de données et algorithmes du jeu Carcassonne. En collaboration avec Justin, nous avons donc produit un document visant à spécifier la structure et ces algorithmes. Celui-ci m'a permis de créer un outil pour générer facilement les tuiles du jeu et leurs besoins complexes, autant pour le back que pour le front.

Par la suite, la division des tâches de l'affichage m'a amené à me pencher sur la question de l'interface des informations et la sélection des tuiles. Afin d'être synchronisé avec mes camarades, j'ai réalisé un diagramme UML de l'architecture globale de l'affichage. Le nom des méthodes et leur intersection furent beaucoup modifié à cause du paradigme imposé par unity pour la gestion des gameobjects. Mais au cœur, l'architecture n'a pas changé.

Comme je ne maîtrisais pas Unity, il m'a fallu un moment d'apprentissage et de prise en main, ce qui a ralenti en partie mon travail. En effet, notre architecture d'interface mêle beaucoup de systèmes de ce moteur de jeu. Ainsi, j'ai dû maîtriser les layers, les transform et tag des différents éléments pour séparer la table du plateau et la rendre interactive. Pour les différentes informations sur les joueurs et la partie, j'ai également dû apprendre le système de canvas d'Unity, ce qui prend quasiment autant de temps que la partie 3D du projet à maîtriser. Chaque composant était également testé manuellement à travers des scripts. L'apprentissage des méthodes de tests automatiques sur Unity aurait été un investissement temporel trop important par rapport à la quantité de fonctionnalités à faire.

Suite à ce travail, je me suis attaqué au cœur du système d'affichage, le display system. Son rôle est de manipuler un ensemble d'états gérant simultanément les inputs, les communications avec le jeu et l'avancement de la partie.

Pour la partie communication, j'ai préparé avant l'intégration une classe représentant le back, avec des méthodes recevant les informations lorsque le système en a besoin (lorsqu'il entre dans certains états). Le schéma de communication associé est organisé de la façon suivante. L'entrée dans un état se fait automatiquement ou par demande extérieure. Des informations sont reçues, desquelles il est possible de déduire, en fonction de l'état actuel, les objets modifiés. Un traitement est alors exécuté sur la ou les représentations des objets dans l'affichage. Par exemple, lors d'un changement de score, nous sommes prévenus de celui-ci à travers un état. On récupère ensuite les joueurs concernés ainsi que les modifications à effectuer, avant de mettre à jour le score pour chacun d'entre eux et sur la bannière si nécessaire. Enfin, lorsqu'une action a lieu, elle est transmise avec une mise en forme basique. Grâce à cette manière de communiquer, lors des phases d'attente, le système a juste à prévenir l'affichage pour que celui-ci passe à la suite.

J'ai testé en deux grands temps le display system. Premièrement avec des données aléatoires et sans les parties de plateau et de tuile. Une fois ces parties implémentées, je l'ai testé avec des scénarios lus depuis un fichier xml contenant des informations que le back enverrait.

Avant la mise en place du scénario du back, j'ai d'abord dû aider mes camarades à cause du retard pris. Je me suis donc focalisé sur les parties les plus vitales, à savoir les tuiles, dont je me suis occupé, et le plateau, où j'ai assisté Sarah. Grâce à mon expérience sur Unity, trois jours intenses ont suffi à rattraper cela.

Une partie de mon temps fut aussi dédiée à la création des différentes icônes et modèles du projet. J'ai dû apprendre à utiliser le logiciel Blender pour ces modèles, mais j'ai aussi utilisé paint.net (qui est un outil assez complet pour les tâches simples et que je maîtrise bien).

J'ai finalement rejoint l'équipe d'intégration, pour aider à relier tous les différents composants. J'ai participé à la mise en place de la communication entre l'affichage et le système côté client. J'ai également réalisé quelques adaptations de l'affichage aux besoins des autres parties. Pour finir, j'ai remanié des bouts de code (notamment dans le menu du jeu), et ai ajouté des fonctionnalités aux interfaces qui étaient incomplètes. Ces modifications n'auraient pas dû être nécessaires, d'autant plus qu'elles ont été chronophages et stressantes.

En conclusion, j'ai été au début du projet un défenseur du jeu Carcassonne auquel on aurait ajouté quelques extensions, car initialement, les membres du groupe montraient une grande motivation et une certaine confiance dans leurs capacités. Finalement, nombreux m'ont déçu par leur éthique de travail. J'estime que nous nous sommes retrouvés à faire un projet adapté à 14 avec une équipe active ayant seulement la moitié des membres, et cela malgré les nombreux appels à l'aide. Cela a rendu la fin du projet particulièrement éprouvante et stressante. Ce projet reste malgré tout, dans sa globalité, une expérience forte et enrichissante, où j'ai pu apprendre un beau mélange de savoir technique et où j'ai surtout pu travailler avec des personnes intéressantes.

Fahd Mahraz

Mon rôle principal dans ce projet était de spécifier et développer la BDD. J'ai d'abord commencé par la création des modèles E/A et MCD avec les autres membres de l'équipe BDD en nous mettant d'accord sur les

tables et les attributs nécessaires. Nous avons ensuite créé les tables en question avec SQLite, ainsi que les contraintes statiques et dynamiques associées.

Par la suite, j'ai commencé à me documenter afin de comprendre comment faire la liaison entre Unity et notre base de données en C#, étant donné que c'était la première fois que je travaillais sur ce moteur de jeu et que je n'avais pas de bonnes bases sur ce langage de programmation. Quelques temps après et la deadline pour délivrer l'alpha se resserrant, nous avons dû nous mettre d'accord avec l'équipe de système de jeu sur les fonctions de base, afin qu'ils puissent tester notre base de données. J'ai proposé de m'occuper de l'implémentation des fonctions d'insertion des données des utilisateurs dans la base de données et de vérifier les identifications. À la fin de notre travail et comme indiqué sur le rapport de la BDD, nous avons constaté avec notre chef de projet que SQLite n'était pas le meilleur choix pour pouvoir communiquer d'une part, entre les deux VM et d'autre part, de manière asynchrone. Cela a obligé notre chef à modifier les scripts et à les adapter au mieux possible au projet.

Finalement, la prise en main du C# et Unity s'est faite assez naturellement à force de regarder des tutoriels et de coder. J'ai aussi approfondi mes connaissances en C#, même si de base ce n'est pas trop mon point fort et que j'aurais préféré travailler avec l'équipe du système de jeu.

J'ai malheureusement eu des empêchements ne me permettant pas de le faire, donc je suis en charge maintenant de la rédaction du rapport final avec Sarah et de la préparation de la présentation pour la soutenance. Je trouve que travailler dans ce grand projet était une expérience enrichissante, où j'ai pu consolider mes connaissances générales et apprendre à travailler en équipe.

Caleb Mukaya Gakali

Au début du projet, j'ai été chargé d'accompagner le chef de projet à mettre en place un bon environnement pour l'organisation du travail sur Gitlab, définir les branches de chaque partie, créer des issues, etc. Je suis ensuite passé à la numérisation d'une première maquette des menus avec l'outil Figma, en fonction des contraintes de la spécification, et dont le jeu s'est inspiré. En parallèle de ces tâches, j'ai dû apprendre le fonctionnement de Unity et le C# durant mon temps libre.

Ma participation dans le développement du jeu a principalement été la gestion du déplacement de la caméra, nécessaire pour obtenir un suivi de la partie en cours. L'objectif était d'une part, de pouvoir afficher la partie de l'espace sur lequel le jeu se déroule, et d'autre part, d'élargir le champ de vision de la caméra en fonction de l'espace bi-dimensionnel occupé par le plateau. Il a fallu implémenter les mécanismes nécessaires pour déplacer la caméra avec le clavier et la souris. J'ai commencé par mettre en place des déplacements simples sur les axes de l'espace tridimensionnel permettant les déplacements haut/bas - gauche/droit - zoom avant/arrière, puis à l'ajout des limites du champ de vue.

Je suis ensuite passé à la création des composants servant à la réception d'une notification provenant du back-end, à son affichage sur l'écran de jeu, ainsi qu'à la création de l'écran des scores affiché à la fin du jeu. Enfin, j'ai travaillé sur la création du menu de pause du jeu avec les interactions des boutons pour quitter ou revenir à celui-ci. Une fois toutes ces tâches faites, je suis passé à l'exportation de l'interface du jeu sur Android afin de vérifier que les fonctionnalités et composants ajoutés étaient fonctionnels.

Le plus dur dans la réalisation de ces tâches était de veiller à une bonne intégration de toutes les fonctionnalités réalisées par mes collègues. La scène de jeu était difficile à débugger parce qu'on ne comprenait pas d'emblée les éléments contenus dans ce fichier et on se retrouvait très souvent à avoir des conflits sur celui-ci.

Dans l'ensemble, ce projet était intéressant et instructif car il m'a apporté des connaissances sur de nouvelles technologies et m'a permis d'en approfondir d'autres, mais il m'a aussi aidé à me développer sur la méthode de travail collectif, qui nécessite une bonne cohésion et communication entre les membres des équipes. Je n'avais cependant pas le même rythme de travail que mes collègues, en raison de mon activité professionnelle tous les weekends et de mon absence pendant trois semaines suite à un problème familial, éléments qui ont pesé sur ma productivité.

Conclusion

En définitive, ce projet intégrateur Carcassheim aura permis à la plupart d'entre nous d'apprendre de nouvelles technologies comme Unity, mais également le C# dans sa globalité. Nous avons aussi pu découvrir le travail dans un grand groupe, avec les points positifs et négatifs qui accompagnent ces grands projets. En effet, c'était très intéressant pour chacun de rencontrer de nouvelles personnes et de travailler avec elles. Cependant, il est indéniable que la quantité de travail fournie, et plus globalement l'implication dans le projet, n'a pas été la même pour tout le monde. Certains se sont trop reposés sur les autres, quand ceux-là même faisaient de trop grands sacrifices pour arriver à terminer le jeu. Cette différence était peut-être dû à une confiance trop importante donnée à tous les membres, ou à des attentes personnelles qui ne sont pas les mêmes pour tout le monde. Nous retenons donc qu'il aurait fallu être plus ferme, afin que les tâches ne prennent pas autant de retard. De plus, malgré un travail important sur les spécifications, celles-ci n'ont pas été suffisantes, ou pas assez bien respectées, ce qui a créé de nombreux problèmes lors de l'intégration. Pour certains, nous sortons de ce projet marqués par la quantité de travail qu'il a exigé. Nous retenons tout de même que, au-delà tous les problèmes rencontrés, les détails que nous n'avons pas pu implémenter, Carcassheim fonctionne sur ordinateur ainsi qu'en version mobile sur Android, ce qui est la plus belle des récompenses.

Lexique

Meeple : Nom des pions de Carcassonne (le jeu original)

Tuile : brique du jeu

Joueur élu : Un des joueurs séléctionné de manière unique pour le tour courant

Joueur : non élu Tous les joueurs non élus

Joueur : Un joueur quelconque de l'ensemble

Rotation : Une direction parmi Rot=(nord, est, ouest, sud)

Coordonnées : Triplet de coordonnées (x, y, r) de NxNxRot; soit une position et une rotation

Slot : Endroit d'une tuile pouvant être connecté à d'autres slots de type compatible dans les tuiles voisines

Zone : Ensemble de slots inter-connectés; elle appartient au joueur ayant le plus de meeples dans la zone

Plateau : Espace où les tuiles sont posées

Ecran : Ensemble des fonctionnalités du jeu visibles à l'écran

Table : Zone de l'IHM où sont affichées les pièces (meeple ou tuile) possibles à jouer

Case joueur : Case contenant les informations principales des joueurs de la partie (pseudonyme, nombre de meeples restant, score actuel)

Meeple actif : Signifie que le meeple est actuellement présent sur le plateau

Bibliographie

ABD EL AZIM, Wonder Developper. 2019. *Youtube.* [En ligne] 13 04 2019. (Android , Windows Phone , Windows , IOS, WINRT). https://www.youtube.com/watch?v=dezAuScV9ZY.

DIGESTIBLE. 2020. *Youtube.* [En ligne] 08 09 2020. https://www.youtube.com/watch?v=8bpYHCKdZno&t=750s.

Exploration, Nazjatar Cinematic. 2019. *Extrait background animé menu.* 12 05 2019.

Liang, Niels Swimberghe & Ying. 2022. *swimburger.* [En ligne] 2022. https://swimburger.net/blog/dotnet/how-to-run-a-dotnet-core-console-app-as-a-service-using-systemd-on-linux.

LIBERTY, Pluralsight. 2013. *Youtube.* [En ligne] 05 07 2013. Create a Dictionary Using Collections in C#. https://www.youtube.com/watch?v=Dke0dlCgFSk&t=176s.

MICROSOFT. 2022. *microsoft.* [En ligne] 2022. https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite/?tabs=netcore-cli.

—. 2022. *Mcrosoft.* [En ligne] 2022. https://docs.microsoft.com/fr-fr/dotnet/framework/network-programming/synchronous-client-socket-example.

—. 2022. *Microsoft.* [En ligne] 2022. https://docs.microsoft.com/fr-fr/dotnet/framework/network-programming/asynchronous-server-socket-example.

Music, Nazjatar. 2019. *Extrait musique de fond menu.* 25 05 2019. Rise of Azshara Music.

SWALHA, amr swalha. 2016. *Youtube.* [En ligne] 30 06 2016. https://www.youtube.com/watch?v=Ii5a6-4IUWI&t=10s.

Technologies, Unity. 2022. *unity3d.* [En ligne] 30 04 2022. https://docs.unity3d.com/ScriptReference/.

—. *unity assetstore.* [En ligne] https://assetstore.unity.com/?gclid=Cj0KCQjwpcOTBhCZARIsAEAYLuWpA6K0v63LYO2ycrBd9ifSQHPEXHzvZEmETFQiTesMhfzRgCSKZvUaAk3NEALw_wcB&gclsrc=aw.ds.