BACH Jérémy | BLESCH Eve | CERDAN Baptiste |
GIRARD Gaëlle | REILAND Valentine | STEINMETZ Thomas
L3 CMI UNIVERSITE DE STRASBOURG
S5 Automne 2019

Rapport 3 : Projet Intéraction Homme Machine

Timebomb

Organisation du développement

Répartition du travail

Lors de la réflexion de notre projet, nous avons défini les différents objectifs. Nous avons donc décidé de diviser le groupe en 3 binômes pour chaque partie, à savoir interface, noyau et le réseau. Gaëlle et Jérémy se sont occupés du réseau, Thomas et Baptiste du noyau (puis du graphique) et Eve et Valentine du graphique. Comme nous avons rapidement avancé dans la partie du noyau, nous avons changé les binômes pour que 2 binômes travaillent sur la partie interface qui demande un travail plus conséquent. Quand nous sommes arrivés dans les derniers jours du rendu du projet, nous avons tous travaillé ensemble pour lier les différentes parties pour que le jeu soit cohérent dans son ensemble.

Plannings et jalons

Les dates de rendu de rapport nous ont permis de planifier les tâches à faire. La mise en place du concept du jeu ainsi que la réalisation des wireframes ont été nos premiers objectifs et ont été fait rapidement, étant donnée qu'ils étaient nécessaires pour le premier rendu. Ensuite, nous avons composé les équipes de travail en binôme afin de mieux se répartir les taches. Chaque binôme travaillait de son côté sur leur partie jusqu'a avoir quelque chose de concret. Le noyau logiciel a été le premier module à être terminé. En effet, il représente la base du jeu, base sur laquelle les autres modules se reposent. La partie graphique s'est ensuite basée sur le noyau pour avoir un enchaînement de wireframes qui soit fonctionnel ainsi qu'un jeu jouable sur une version offline, c'est à dire sur un seul ordinateur. Ainsi, la combinaison noyau/graphique permettait d'avoir une première version. La partie réseau plus complexe a ensuite été rajoutée afin de faire une version online, chaque joueur pouvant joueur sur son propre ordinateur et communiquer avec les autres joueurs au moyen d'un chat.

Partie noyau

La partie du noyau a été la partie la plus rapidement terminée parce que nos fonctions devaient être utilisées principalement dans l'interface. Le réseau devait se baser sur les structures pour définir le format des différents paquets à échanger. Pour débuter notre implémentation, nous avons listé les différentes fonctions qui devaient être codées pour que le jeu puisse fonctionner. Nous avons effectué des tests au fur et à mesure de nos fonctions en simulant le jeu sur le terminal de commandes. Une fois les tests terminés, nous avions fini la partie du noyau. Par la suite, nous avons dû faire quelques ajouts et correctifs pour une meilleure adaptation. A la fin de la partie du noyau, nous nous sommes lancés dans la partie interface graphique pour pouvoir lier les parties plus rapidement.

Partie interface graphique

La partie interface graphique était assez conséquente. Avant que la partie noyau soit finie, nous avons commencé par créer les différents wireframes du jeu comme présentés sur le premier rapport et les lier les uns aux autres même si le contenu n'était pas encore toujours présent. Cette version n'était pas encore jouable mais la base de l'IHM était là. Quand le noyau nous a été transmis, le binôme du noyau nous a rejoint. Au sein de notre équipe graphique, certains se sont concentrés sur l'aspect graphique de l'IHM en s'occupant du CSS, du responsive, des options et tout ce qui touche au visuel pendant que d'autres se sont concentrés sur le fonctionnement du jeu en liant le noyau. Nous avons alors implémenté une version offline du jeu (qui n'était pas présentée dans le premier rapport). Cette version nous permettait d'avancer sur le projet en attendant la partie réseau mais aussi de compléter notre jeu, permettant d'y jouer même si un seul ordinateur n'est disponible. Nous avons ensuite préparé le plateau de jeu pour la version online même si il n'était pas encore fonctionnel. Lorsque la partie réseau nous a été transmis, nous avons tous travaillé ensemble afin de lier le réseau à notre interface et pouvoir finaliser le gros du projet. Il fallait ensuite travailler sur les derniers petits détails pour rendre l'IHM plus agréable.

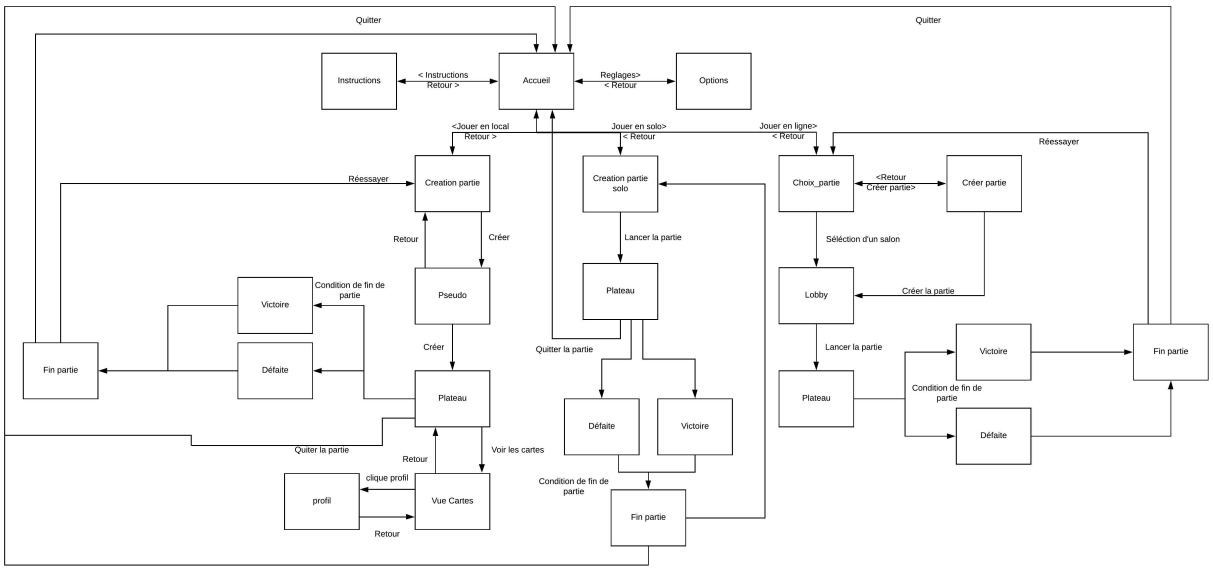
Partie réseau

La première questions que nous nous sommes posée était celle de l'architecture du réseau. Pour en trouver une qui correspondrait le plus au logiciel que nous devions développer, nous sommes allés voir du côté des jeux en réseau du style MMORPG et des jeux multijoueurs en temps réel trouvables sur internet. Des deux architectures les plus récurrentes, nous avons choisi le système serveur-client (l'autre modèle possible étant le pair à pair). Cette architecture nous paraissaît plus simple à gérer car elle permettait de séparer complètement la gestion du jeu de l'interface. En effet, toutes les actions de manipulation de cartes dues au jeu (couper un câble, mélanger le paquet, redistribuer les cartes ...) sont effectuées au niveau du serveur, et le client se contente de recevoir des informations du serveur, de les traiter et d'offrir à l'utilisateur une vue du jeu en fonction.

Après avoir défini le modèle de notre réseau, nous avons défini quelles informations devaient s'échanger le client et le serveur, et à quelle fréquence, pour pouvoir condenser au maximum l'information. Nous avons plusieurs types de messages : ceux qui renferment des données à traiter à l'attention du serveur ou du client, ceux qui constituent des requêtes d'information (généralement venant du client à l'intention du serveur, par exemple une demande de renvoi de la liste des salons de jeu disponibles), ceux qui sont des sortes de "je peux ?" et "vas-y" (par exemple pour rejoindre une partie en ligne qui n'a pas encore commencé).

Nous avons ajouté une notion de propriétaire de partie : en effet, le créateur d'une partie à accès au bouton "lancer la partie" dès qu'il y a plus de 4 joueurs qui l'ont rejointe. Et lorsqu'une partie est complète (nombre de joueurs égal au nombre maximal défini lors de la création de la partie), elle ne s'affichera plus dans la liste des salons rejoignables.

Schéma de conception



(Le schéma sera également disponible en annexe pour une meilleure visualisation).

Travail d’analyse

Analyse conceptuelle

Nous avons d’abord mûrement réfléchi quant à la conception des différents modes de jeu. Nous nous sommes d’abord penchés sur la version multijoueurs en local. Le problème principal du mode multijoueurs sur un seul écran est qu’il est difficile d’afficher les cartes aux joueurs sans que tout le monde ne les voient. Nous avons donc opté pour un onglet “Voir les cartes” sur le plateau de jeu permettant à chaque joueur de voir chacun son tour son jeu de cartes et son rôle. Ainsi, chaque joueur clique sur son pseudo ce qui va afficher ses cartes, puis clique sur “Cacher les

cartes” pour les effacer. Cette version demande aux autres joueurs d’être honnêtes et de ne pas tricher. C’est la variante que nous avons trouvé qui soit la plus pratique et qui permette de voir ses cartes n’importe quand dans la partie.

Ensuite, nous avons fait un mode solo permettant de jouer seul. Pour cela, nous avons pensé à faire une sorte d’IA. Le concept repose sur un joueur entouré d’un nombre de “robots” (nombre choisi par le joueur compris entre 3 et 7). Au début de la partie, l’utilisateur commence par couper une carte et ensuite c’est le robot en question qui décide quelle carte couper. Le choix de la carte à couper et du joueur choisi par le robot est aléatoire. L’utilisateur reprendra la main lorsqu’un robot décide de couper dans son paquet. Pour rendre chaque étape visuelle (chaque carte coupée) nous pouvons voir au centre les cartes coupées par les robots ainsi que le changement de tours et de manches à droite du plateau. Lorsque c’est au tour du joueur, un affichage “A votre tour” apparaît à gauche des cartes du joueur, et les cartes au milieu disparaissent.

Enfin, nous avons en même temps réfléchi à faire un mode multijoueurs en ligne [voir ***implémentation du réseau***].

Analyse graphique

Notre jeu propose plusieurs thèmes différents. Pour rendre le style facilement modifiable entre plusieurs thèmes nous avons utilisé des feuilles de style séparées que nous chargeons le moment voulu (avec la comboBox dans option).

Le thème classique
Pour le choix de notre thème principal, nous avons utilisé la charte graphique officielle du jeu (images fournies, livret des règles, etc…) Elle retranscrit bien la période dans laquelle le jeu est sensé se dérouler et l'ambiance de celui-ci. Les graphismes créent un style texturé dans le genre du skeuomorphisme car ceux-ci tournent principalement autour de la technique du digital painting. Le style "peint" cherche à inspirer une ambiance, il est très utilisé dans des jeux se voulant "fantasy" ou "steampunk" comme Hearthstone, un jeu de carte très en vogue ou League of legends. Nous nous sommes servis des médias officiels, cependant nous avons aussi crée nos propres média (voir mode Jacky).

Le thème daltonien
Celui-ci change radicalement du thème classique et ne retranscrit pas spécialement d'ambiance. Pour le créer nous avons cherché des palettes de couleurs adaptées sur Internet et avons opté pour un graphisme simple et basé sur le contraste. Les textures sont uniquement utilisées dans les endroits importants, là où elles doivent attirer l'attention.

Le thème Jacky
C'est un Easter egg. Nous avons créé nos propres média, montages etc… avec des photographies et le logiciel photoshop pour adapter les visuels à notre interface.

Implémentation du réseau

Pour implémenter notre architecture réseau, nous avons utilisé les librairies QTcpSocket et QTcpServer. L'emploi du protocole TCP étant une évidence, car il est beaucoup plus fiable que le protocole UDP, et la différence de performance est négligeable pour notre cas de figure.

Le programme serveur contient toutes les composantes nécéssaires pour faire tourner une partie, ainsi que des fonctions standard de réseau : gestion des connexions et déconnexions d'utilisateurs, maintenance d'une liste des clients connectés, décodage des messages envoyés, envoi de messages.

Les différents messages employés pour la communication entre client et serveur sont implémentés au moyen de chaînes de caractères (QString). En effet, nous nous sommes rendus compte que générer des chaînes de caractères d'un côté, de les envoyer à travers le réseau et de les parser à l'autre extrémité de la connexion est beaucoup plus facile que de gérer des flux d'octets et/ou des structures. Le seul endroit où nous avons employé des structures de données est du côté client, pour agréger des données envoyées par le serveur avant de les employer pour l'affichage.
Ces messages sous formes de QString ont des formes assez diverses, mais suivent quelques conventions : pour faciliter le parsing des données à l'autre extrémité de la connexion, des séparateurs sont utilisés. Le séparateur de premier niveau est l'esperluette `&`, et celui de deuxième niveau (parfois nécéssaire pour parser des objets) est le hash `#`.
Ces séparateurs permettent un parsing assez rapide, car l'objet QString possède une méthode split qui permet de diviser le message principal à partir du caractère séparateur.

Pour pouvoir gérer plusieurs parties en parallèle, ainsi que des créations de parties, des connexions, des déconnexions, et tous les évènements dus à la maintenance d'un réseau, nous avons choisi d'employer des threads (librairie QThread). Le thread principal se charge de gérer les connexions, déconnexions, etc.
Lorsqu'un message annonçant la création d'une nouvelle partie arrive, le thread principal va créer un nouveau thread ainsi qu'un objet Worker (qui contient tout le nécéssaire pour faire tourner une partie en réseau), va déplacer l'objet Worker dans le thread nouvellement créé, et laisser le Worker gérer la partie. Lorsqu'une partie est terminée, le thread qui la gérait se termine.

La chose la plus simple a gérer a été le chat. Lors de l'envoi d'un message de chat, le client n'affiche pas directement le message tapé ; il l'envoie directement au serveur, qui lui se charge de retransmettre le message à tous les clients connectés au salon d'où le message provient.

La gestion des déconnexions en cours de partie étant une problématique, nous l'avons résolue avec un système de placeholder qui est détaillé plus bas.

Éléments implémentés

Pour rendre notre interface la plus complète possible, nous avons implémenté plusieurs fonctionnalités.

Plusieurs **modes de jeu** :

- Le mode solo (Fonctionnel)
- Le mode multijoueurs en local (Fonctionnel)
- Le mode multijoueurs en réseau (Non fonctionnel, se référer à la partie "difficultés d'implémentation" plus bas).

Accessibilité : L'IHM est accessible aux malvoyants. Pour cela, il faut simplement cocher la case à coté de "Mode malvoyant" dans les réglages (il n'est à nos jours pas possible de sortir de cet affichage, donc à éviter sauf si vous en avez besoin). La police est alors augmentée.

Localisation: Nous avons adapté notre IHM à deux langues : l'anglais et le français. Il suffit de la changer sur le menu principal.

Autres éléments Graphiques (aller dans réglages) :

- Il est possible de mettre le jeu en plein écran .
- Plusieurs thèmes sont disponibles (classique, Jacky, Daltonien)

Autres éléments textuels :

- Description des règles
- Manuel d'instruction détaillé
- Possibilité de voir la description de son rôle pendant la partie

Éléments sonores : Paramètres de son et de musiques présents mais non implémentés

Communication réseau :

- Un programme serveur
- Un programme client
- Un chat

Solutions envisagées mais non retenues

Versions simples
Nous voulions au départ faire un jeu en réseau avec des téléphones mobiles. Le principe était d'avoir un écran d'ordinateur servant de plateau de jeu, et chaque joueur aurait sur son téléphone ses cartes, similaire à la plateforme Kahoot. Cependant, nous voulions d'abord avoir un jeu fonctionnel avant de s'attaquer au développement d'applications mobiles, domaine dans lequel

nous n'avons pas assez de connaissances et pas forcément de professeurs pour nous aider si besoin. Nous nous sommes rapidement rendus compte qu'une version comme ça n'était pas vraiment envisageable .

Nous avons ensuite pensé à faire une version "Split screen" comme version locale pour que plusieurs joueurs puissent jouer sur le même écran. Cependant, ce n'est pas du tout adapté pour le timebomb comme il faut que les cartes de chaque joueur soient cachées. Nous avons alors opté pour la version courante de notre jeu.

Ensuite, nous avons dû réfléchir aux solutions techniques envisagées pour réaliser ce projet. L'implémentation du jeu en web a été une solution envisagée, cette solution n'as pas été retenue car le challenge de développer une application en C++ était une pratique nouvelle et nettement plus intéressante que le web dans lequel nous étions déjà plutôt à l'aise.

Versions en réseau

Lorsque nous avons étudié les différentes architectures réseau utilisées dans les jeux multijoueurs en temps réel existants, nous avons envisagé pendant un temps un système de pair à pair (peer-to-peer). L'avantage de ce système nous aurait permis de nous affranchir du serveur de jeu nécessaire pour le jeu en ligne : en effet, dans ce modèle, toutes les composantes employées par le jeu en ligne sont présentes sur la version "utilisateur" du programme. Cette solution a été abandonnée assez rapidement, pour plusieurs raisons.

La première étant la consommation de ressources de l'ordinateur qu'elle impliquait. La version client-serveur que nous avons développée est déjà relativement gourmande pour des ordinateurs portables avec peu de ressources, alors si en plus il était demandé à l'ordinateur d'héberger une partie du jeu en ligne, ce serait devenu pratiquement ingérable pour des ordinateurs portables peu puissants. Or le but est quand même de faire une application qu'un maximum de monde peut utiliser, et de manière portable.

Une autre raison qui nous a poussé à abandonner cette piste est la gestion des ports réseau. On a à peu près tous eu un jour ce problème des ports bloqués par le pare-feu de windows ou de son FAI via la box internet.
Avec un seul port, c'est gérable : les systèmes récents facilitent la gestion des ports ouverts, notamment grâce à des pop-ups annonçant directement que tel processus tente d'utiliser tel port, et nous demandant si nous autorisons le trafic via le port en question. Pour une architecture peer-to-peer, il aurait fallu ouvrir autant de ports que de joueurs dans la partie en réseau, ce qui est autant une faille de sécurité (donc un risque accru d'attaque de l'ordinateur) qu'une corvée car besoin d'autoriser le trafic sur chaque port (si une fois c'est déjà quelques clics, alors s'il faut le faire 8 fois ...). L'alternative à l'utilisation d'un port différent par client était le multiplexage, qui a également été abandonné très rapidement car trop compliqué à implémenter.

Difficultés d'implémentation

Difficultés pour le graphique
Ce projet ayant été par la même occasion nos premiers pas avec Qt, il y a des détails que nous ne maîtrisons pas et des problématiques auxquelles nous n'avons pas toujours trouvé de solutions.

Le style : Qt prend en charge les feuilles de style, cependant certaines propriétés existant en css3 ne sont pas prises en compte par les différents widgets de Qt (par exemple : transition, box-shadows, background-size) il a donc fallu trouver des alternatives. Cependant nous n'avons pas trouvé d'alternative pour faire des transitions (qui permettent de lisser un effet entre un stade de style et un autre stade).

Les formats animés : Pour les animations nous avons utilisé des formats gif, un bon compromis avec la vidéo car celui-ci est plus rapide à charger mais néanmoins limité (255 niveaux de couleurs et 2 de couche alpha). Néanmoins, il faut une ligne de code pour animer le gif ce qui fait que ce n'était pas possible d'animer un gif qui était un background-image.

Difficultés pour le réseau
La principale difficulté que nous avons rencontré à l'implémentation a été le debug d'un programme asynchrone. En effet, librairie orientée évènementiel et réseau oblige, nous avons dû nous plonger dans le paradigme évènementiel, ce qui est loin d'être inné pour des étudiants qui ont passé le plus clair de leur cursus à programmer de manière procédurale.

Le debug de ces programmes asynchrones a été une réelle difficulté, qui a malheureusement eu raison de notre capacité à faire fonctionner le jeu en ligne. Au moment du rendu, la gestion de l'affichage de la liste des salons disponible est fonctionnelle (mais peu stable, des erreurs de segmentation peuvent arriver au niveau du serveur ou du client), ainsi que la possibilité de rejoindre une partie et de chatter en attendant le début de la partie. Jouer une partie en ligne est à cette date pas encore possible, d'ailleurs un message d'alerte le signale si le créateur de la partie tente de lancer le jeu.

Dans un autre registre, une des difficultés que nous avons eu à gérer est venue de là où on n'en attendait pas. En effet, il est impossible d'écrire et de lire sur un socket créé depuis un autre thread que celui où on essaie de lire/écrire. Cela a posé problème, surtout au niveau des threads gestionnaires de salon qui ont besoin d'envoyer des données aux clients. Nous avons résolu ce problème en implémentant des systèmes de forwarding de messages : si un message à destination d'un thread salon arrive sur le thread principal du serveur, ce dernier sera redirigé vers le thread via l'appel d'une méthode spécifique à l'objet s'exécutant dans le thread (objet Worker). Inversement, lorsqu'un thread doit envoyer un message à un client, ses fonctions "sendSpecific" et "sendAll" ne font en réalité qu'envoyer un signal porteur du message au thread principal du serveur qui, à la réception de ces signaux, écrira le message sur le/les sockets des clients concernés.

La dernière difficulté que nous avons rencontré, c'est la question de la déconnexion en cours de

partie.
Pour régler ce problème, nous avons créé des classes filles de User et QTcpSocket, dont certaines méthodes sont redéfinies (pour simuler une coupe de carte aléatoire, ou pour simuler des écritures/lectures socket), ce qui permet de n'avoir à faire aucune modification dans le code qui gère une partie. On s' "adresse" à un placeholder comme on le ferait à un User réel. L'état de l'avancement du debug étant ce qu'il est, nous n'avons pas encore eu la possibilité de tester le bon fonctionnement de ce mécanisme. Mais l'idée y est.

Différences avec les wireframes de départ

Nous avons relativement bien respecté les wireframes exposés lors de notre premier rapport. Nous avions bien réfléchi au projet ce qui nous a permis de ne pas faire de changement majeur. Nous avons juste rajouté une version offline et une version solo pour rendre l'IHM plus complète.

Pour aller plus loin

- Un problème qui se pose est que notre jeu, dans l'éventualité que nous le souhaitons, n'est pas diffusable comme les images utilisées ne nous appartiennent pas. Pour rendre notre jeu encore mieux il aurait fallu créer nos propres cartes.
- Tous les éléments présentés dans les réglages ne sont pas tous fonctionnels. Si nous avions plus de temps nous les aurions mis en place.

Apports du Projet

Réussites

Dans un premier temps, le projet nous a permis d'en apprendre plus sur la gestion de projet en groupe. En effet, nous étions habitués à travailler par groupe de 2. Le fait de faire un projet subdivisé en partie (2 personnes par partie) n'a pas été une tâche facile, mais ce fut très instructif. Nous avons su utiliser les atouts de chaque personne pour que chaque personne soit épanouie dans sa tâche. Ce fut également une réussite de mener à bien ce projet ensemble.

Difficultés

La plus grosse difficulté a été la communication au sein du groupe. Les tâches étaient séparées et nous devions tenir informés les autres membres du groupe de nos avancés. Cela aurait sans doute plus facile si nous avions nommé un responsable de projet. En effet, chaque binôme se gérait de manière autonome pendant la première majorité du projet. Dans la partie du code, la mise en commun de toutes les parties a été une difficulté puisqu'il a fallu que chaque partie soit cohérente ainsi que régler les conflits des différentes modifications que chaque personne a faite.

Leçons apprises

En rapport aux difficultés énoncées précédemment, les problèmes rencontrés lors de ce projet nous sont bénéfiques et nous permettent de ne pas refaire les mêmes erreurs. Pour le prochain projet de groupe, nous élirons un chef de projet qui vérifiera l'avancement de chaque groupe ainsi que fera l'intermédiaire entre les parties. De plus, il faudra instaurer des deadlines afin qu'aucun groupe ne soit ralenti par un autre car il n'avance pas assez rapidement. Enfin, il faudra faire des récapitulatifs chaque semaine afin de discuter des problèmes et interrogations de certains groupes, le manque de communication étant un frein majeur à l'avancement de notre projet.

Expérience acquise

Nous avons appris sur les problèmes qui se posent dans le travail en groupe. Nous avons su prendre en main le langage c++ et l'interface de Qt.