2017 / 2018
Licence 3 - Maths Informatique
Interface Homme Machine

CONSTANTINO Victor
HALLER Mathieu
NIEZGODA David
MEYER Adrien
SCHOTT Lucas

Rapport de Projet

Sous la direction de Régis Witz

Dans le cadre de l'unité d'enseignement Interface Homme Machine (IHM) dispensée au cours de la licence Maths-Informatique de l'Université de Strasbourg, nous avons été amenés à réaliser un projet consistant à implémenter un jeu de cartes. En privilégiant l'aspect libre du sujet, nous avons donc choisi de réaliser une application permettant de jouer au jeu de la série Yu-Gi-Oh © : Duel Monsters. Aussi, le présent rapport se propose de détailler les différentes étapes et choix de la conception, pour ensuite se focaliser dans un second temps sur les aspects touchant davantage à l'interface. Il conclut alors sur une discussion, expliquant les problèmes et solutions rencontrés, ainsi que les possibilités d'évolution envisagées.

Conception

Description

Afin de répondre au cahier des charges, l'objectif principal était d'implémenter une version de base du jeu, où il est possible de pouvoir affronter des adversaires autant réels que virtuels, en solitaire ou via le réseau. De même, nous voulions que le joueur puisse créer ses propres decks, ses propres cartes, ainsi que personnaliser certaines règles de jeu comme le nombre de points de vie de départ.

Concernant les duels, nous pouvons les résumer de la façon suivante. Les deux joueurs s'affrontant disposent de points de vies. A chaque tour, le joueur ayant la main pioche. Il peut alors poser des cartes et attaquer les monstres de l'adversaire avec ses propres monstres, s'il les a placé en mode attaque. Si un monstre défait n'est pas en mode défense, son propriétaire déduit la différence de points d'attaque des monstres de ses points de vie, jusqu'à perdre la

partie si ceux-ci atteignent 0 ou qu'il ne peut plus piocher. Un lien vers les règles complètes figure en annexe 1.

Pour pouvoir implémenter ce système de jeu, il nous a alors fallu mettre en place de nombreuses fonctionnalités dont voici une rapide liste non exhaustive, et qui concerne uniquement le duel en lui-même :

- Piocher des cartes.
- Placer des monstres sur le terrain.
- Placer des magies/pièges sur le terrain.
- Choisir la position des cartes placées (attaque/défense pour les monstres, visible/masquées sinon).
- Choisir quelle carte sacrifier pour placer un monstre de niveau plus élevé.
- Attaquer, détruire éventuellement les monstres et déduire les points de vies.
- Determiner la fin du combat.

Implémentation

Pour développer l'application, nous avons fait le choix de nous tourner vers la programmation événementielle, principalement pour son caractère modulaire nous permettant une bonne répartition des tâches. Au vu de la complexité du projet, une machine à états ou des fichiers de description de type QML nous paraissaient non adaptés. De même, modifier du code issu d'un constructeur en interface graphique pour ajuster son comportement nous est apparu plus complexe que réaliser la totalité en programmation événementielle.

Aussi, nous avons opté pour l'utilisation du framework Qt 5 pour développer l'application, en C++. Celui-ci nous semblait être un bon compromis entre les librairies graphiques de Java et une réalisation en HTML/Javascript. En effet, concernant java, Swing nous a paru quelque peu désuet, et FX n'était pas disponible sur de nombreuses plateformes. De même, étant habitués à la programmation Web, le choix de Qt5 nous permettait de découvrir à la fois un nouveau langage de programmation et un nouveau framework, mettant ainsi plus de challenge ou d'enjeux dans la réalisation du projet.

Tout au long du projet, nous avons utilisé la plateforme Github pour synchroniser nos dossiers. De même, nous étions en contact permanent au moyen d'un salon de chat Discord, nous permettant de nous contacter à tout moment. Aussi, au moindre problème ou pour effectuer une requête, nous pouvions nous concerter pour obtenir une réponse rapide de la part de l'ensemble du groupe.

Le projet a alors été décomposé en 5 principales composantes : IHM, Noyau du jeu, Créateur de deck/cartes, Intelligence Artificielle (IA) et Réseau. Ce découpage nous a alors permis de

travailler à la fois individuellement et en équipe. En effet, ces composantes étaient fortement en interaction, que ce soit par exemple pour la conception du modèle de cartes utilisé dans le noyau et le créateur, l'ergonomie liée à la partie réseau, ou encore les échanges entre l'IA et le noyau. Une vue de l'évolution générale est disponible en annexe 2, et une approximation de la répartition du travail de chacun est disponible en annexe 3 et 4. Un graphique UML de l'ensemble de l'application est également disponible en annexe 5. On peut noter en particulier que l'ensemble des élements concernant l'IHM ont été distinguées des autres élements, dans une logique de développement en Modèle Vue Controller (MVC). On peut y voir notamment l'imbrication des classes entre elles, les classes fortement connectées aux autres et, à l'inverse, les classes qui sont d'avantages des sortes d'utilitaires.

De manière générale, Costantino s'est chargé du noyau et des combats en réseau, Niezgoda s'est chargé de l'IHM, Meyer s'est occupé du créateur de Deck et de Cartes, Schott a créé l'IA et Haller s'est chargé de parties diverses incluant la traduction, le support clavier et le portage Windows.

En terme d'architecture, vous pouvez voir sur la figure 1 la façon dont s'articule les principales structures dans lesquelles l'utilisateur peut naviguer. Après un écran de chargement, le joueur arrive dans le menu principal. Différents choix s'offrent à lui dont notamment celui de lancer une partie. S'il le fait, il est amené à choisir le type de partie (local, réseau, …) souhaité, avant d'arriver sur l'écran de jeu en lui-même. Depuis le menu principal, il a également accès à différentes pages lui permettant de construire son deck, de changer les règles ou encore d'accéder à l'aide du jeu. On peut également noter qu'à chaque étapes, l'utilisateur a la possibilité de revenir en arrière, à l'écran précédent.

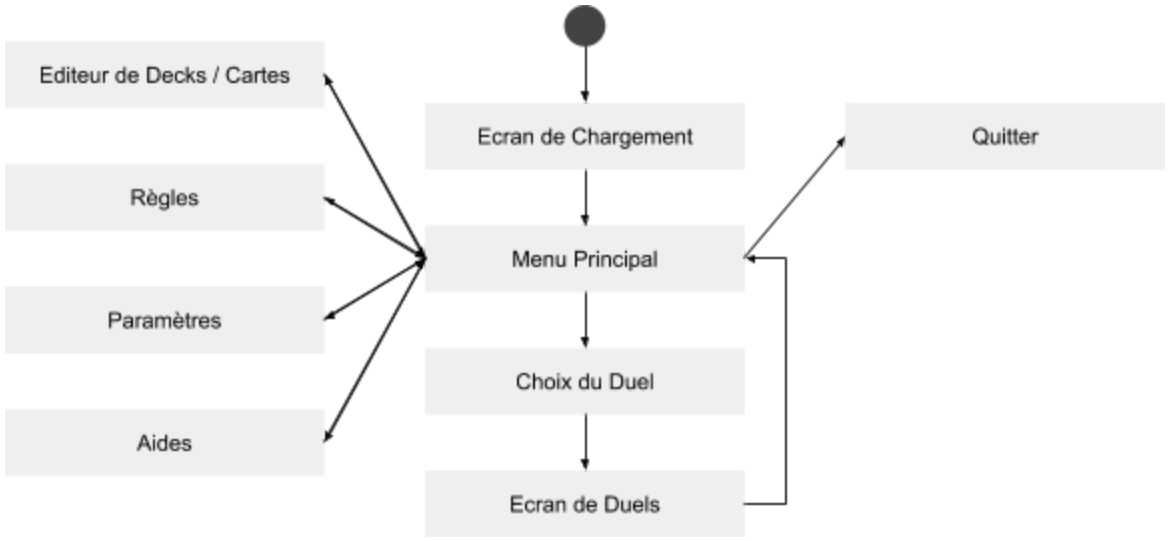


Figure 1. Schéma des principales structures de l'application.

Editeur de Deck

Yu-Gi-Oh © étant un Trading Card Game (jeu de cartes à échanger et à collectionner), les chances de victoires sont en partie déjà déterminées avant même que la partie commence en fonction de la composition des decks de cartes. C'est pour cela que nous avons choisi de permettre aux joueurs de créer leur propre deck à l'aide d'un éditeur simple et concis.

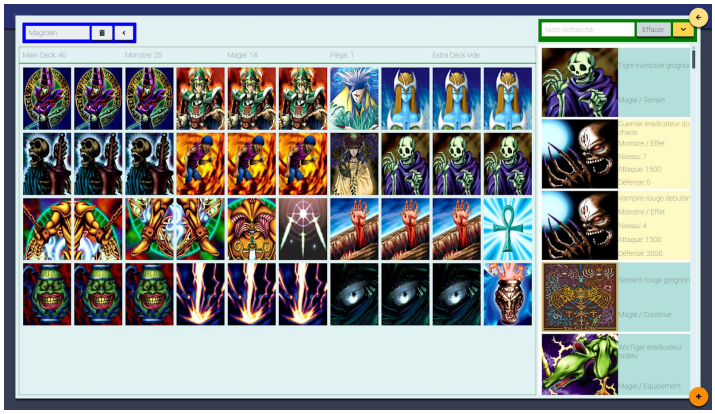


Illustration 1 : Aperçu de l'éditeur de deck

Conceptuellement, on peut le simplifier pour le percevoir en deux grandes zones.

Une zone d'édition du deck (en rouge), où l'utilisateur peut voir le nombre de cartes dans son deck, combien de monstres, magie, piège… ainsi qu'une vu visuel de son deck, sur laquelle il peut interagir pour ajouter ou enlever des cartes.

Une zone explorateur de Cartes (en bleu), où l'utilisateur peut y rechercher des cartes en ce basant sur certains critère et les y ajouter au deck.

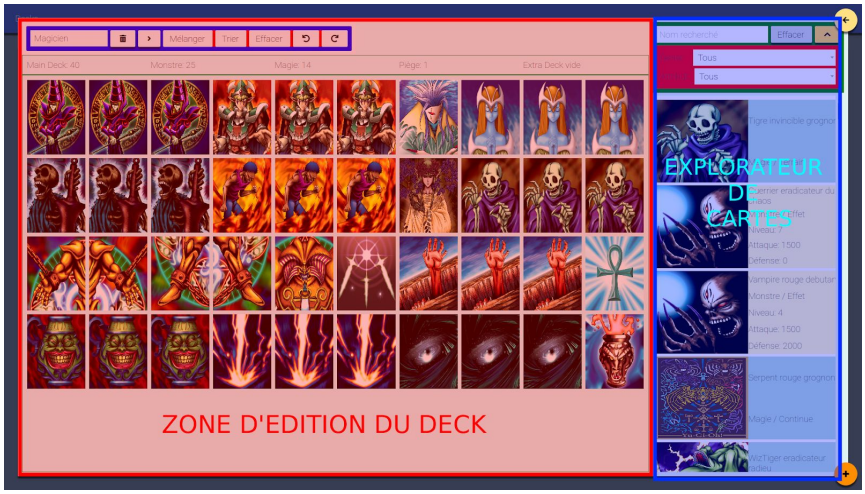


Illustration 2 : Découpage en deux zones de l'éditeur de decks

Nous somme passés par plusieurs design avec plus ou moins de fonctionnalitées, mais celle-ci prenaient rapidement de la place et n'étaient pas forcément indispensable. Nous avons finalement opté pour limiter le nombre de bouton afin d'éviter aux utilisateurs un sentiment d'agression ou de confusion. Afin de pousser cet aspect à l'extrême, nous avons fait le choix de pouvoir masquer les fonctionnalités un temps soit peu avancées.

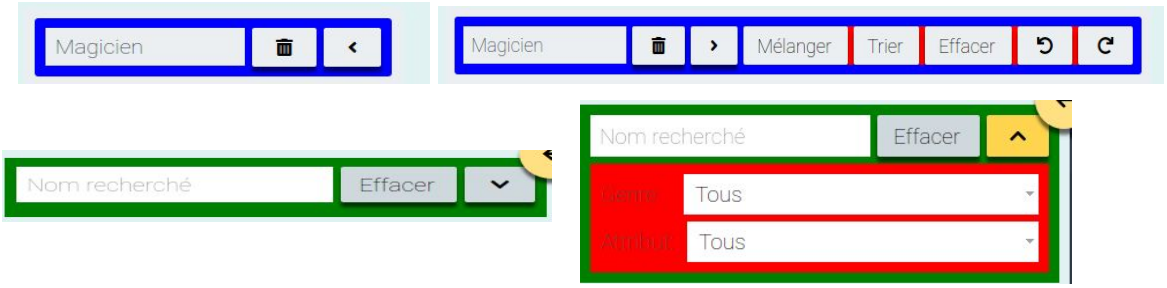


Illustration 3 : Ouverture et Fermeture des fonctionnalités

De la même manière, nous utilisions auparavant des boutons "enregistrer" et "trier". C'est dans un soucis de confort pour l'utilisateur et d'homogénéité avec le reste de l'application que nous avons choisi de rendre ces actions automatiques et transparentes.

On observe néanmoins une limite dans ces deux résolutions :

- Quand considère-t-on que le nom du deck a été modifié ? On ne peut pas le faire à chaque insertion de lettre, cela modifierait de trop nombreuse fois les fichiers… Nous avons choisi de n'actualiser le nom du deck que quand la touche "entré" est pressé ou après n'importe quelle autre action. (ajout d'une carte…)

- Le filtrage dynamique lors de la recherche de nom. Etant donné que nous actualisons la liste des cartes à chaque insertion de caractère dans la barre de recherche, si l'utilisateur entre un mot de x lettres, le système filtre en réalité x fois… Cela peut commencer à se ressentir si la liste des cartes est longue.

Editeur de Carte

Nous souhaitions que les utilisateurs puissent eux aussi créer et ajouter leurs propres cartes, c'est pour cela que nous avons implémenté un éditeur de carte. Il permet de modifier toutes les composantes d'une carte de Yu-Gi-Oh ©.

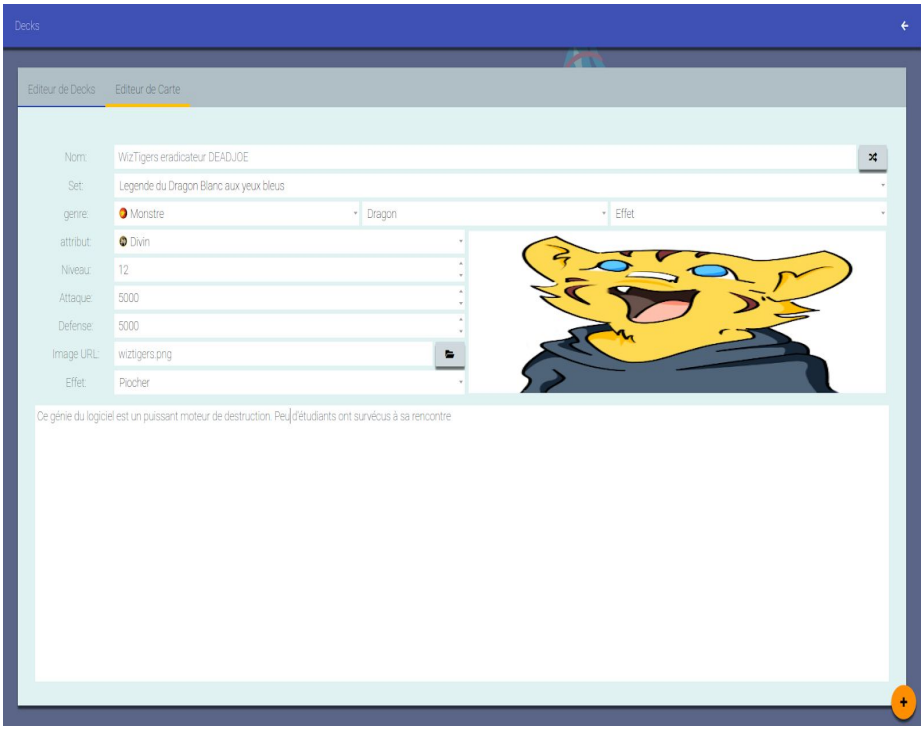


Illustration 4 : Editeur de Carte

Les éditeurs de cartes et de decks utilisent un système de fichier .set, qui comporte les cartes d'une extension, et .deck, qui caractérise un deck. Vous pouvez retrouver leur description exhaustive dans le fichier README de notre dépot github.

C'est ainsi une manière d'aborder la question des extensions plus en profondeur. Il nous est ainsi très simple de diffuser de nouvelles cartes ou deck et on peut envisager un système où il suffirait par exemple de simplement les télécharger sur notre futur site internet, que ce soit directement ou par l'intermédiaire d'une nouvelle option dans l'application.

Noyau

Selon le modèle MVC, le noyau est séparé de la partie réseau et de la partie IHM. Il se charge de l'ensemble du déroulement des parties, et gère notamment les attaques, les invocations de monstres ou encore la vie des deux joueurs. Il connait donc toutes les informations relatives à la partie en cours comme le contenu des decks ou le numéro du tour courant.

Lorsque cela est nécessaire les actions que le joueur effectue sont transmise au réseau ou à l'intelligence artificielle, au moyen de différents signaux. A l'inverse, à chaque action d'un des joueurs, selon le paradigme de la programmation événementielle, le noyau réagit à des signaux envoyés par l'IHM qui elle, est totalement aveugle en terme de contenu. Celle-ci se contente en effet de transmettre des numéro d'emplacements que le noyau convertit, ou de recevoir des images à afficher. Toutes les cartes sont donc stockées avec différentes coordonnées comme le lieu où elles se trouvent (main, terrain, …) ainsi que leur emplacement dans cet ensemble.

Réseau

Le réseau fonction sur le principe d'un joueur client et d'un joueur serveur, avant le début de la partie. Le fonctionnement en cours de partie s'apparente ensuite à une connexion en Peer-to-Peer. L'ensemble des fonctionnalités fonctionne en réseau local mais, suite aux limitations du réseau de l'université et de certains opérateurs, il n'a pas été possible de faire fonctionner les combats en réseau de manière globale. Après un rapprochement et une discussion avec les professeurs de l'option Réseau, il s'avère que la raison concerne un filtrage IPV6 ainsi que le blocage de l'ensemble des ports non conventionnels.

Serveur de MatchMaking

Nous avons également pensé à ceux qui n'ont pas de connaissances jouant au jeu. Pour eux, nous avons mis en place un serveur de matchmaking qui permet donc à 2 joueurs ne se connaissant pas de se trouver et de jouer ensemble.

Le serveur reçoit et envoie des messages avec le protocol UDP. Il ne sert qu'à mettre en relation 2 joueurs, le reste de l'algorithme suivant ensuite le même fonctionnement que si les 2 joueurs se connaissaient. Il stocke donc dans une pile les adresses des joueurs, tout en les plaçant en attente. Lorsqu'un second joueur se connecte, les deux reçoivent les adresses de leurs adversaires respectifs et la partie commence.

Intelligence Artificielle (IA)

Le jeux de Yu-Gi-Oh © étant relativement complexe, il peut y avoir de très nombreuses tactiques à adopter en fonction des situations toutes très différentes les unes des autre. Un algorithme simple du type: "je pose une carte en position d'attaque puis j'attaque" n'aurait pas suffit pour avoir une expérience de jeux optimal et immersive. On remarquerait tout de suite la différence de style de jeux entre un adversaire réel et cette IA.

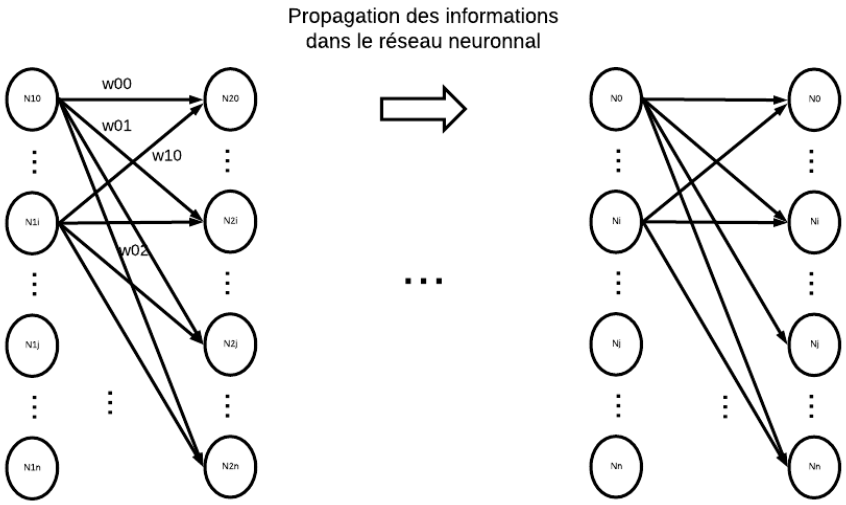
Nous avons décidé de réaliser une IA utilisant le deep learning. Ce domaine est très intéressant, riche, et permet de faire énormément de choses différentes, et d'appréhender des situations très complexes, sans que l'on ait à prévoir ces situations dans un algorithme. Les algorithmes de deep learning permettent en effet de choisir un action parmis des actions prédéfinis en fonction des données que l'on entre dans le réseau neuronal et de l'objectif à atteindre.

L'IA a été programmée sans framework de deep learning, la plupart des framework étant incompatible avec les ordinateurs des salles de TP. L'IA a donc été conçue entièrement en C++ en utilisant la librairie de calcul matriciel Eigen. Nous nous sommes néanmoins inspiré du code source de plusieurs framework de deep learning open source disponibles sur Github, ainsi que suivi un cours sur le deep learning sur le site Udemy, pour bien comprendre le fonctionnement des réseaux neuronaux et réussir à en implémenter un.

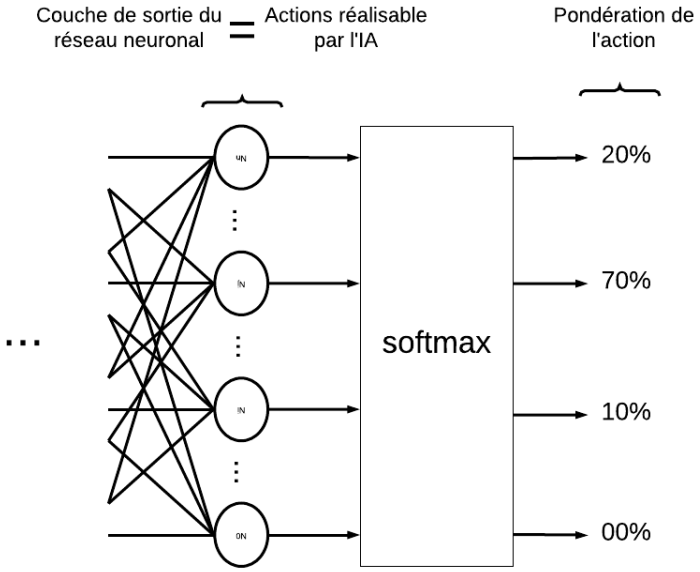
Fonctionnement de l'Intelligence Artificielle

1. Quand c'est au tour de l'IA de jouer, elle va lire l'état du jeu (les cartes sur le terrain, dans le deck, etc) , c'est à dire, qu'elle va remplir un tableau avec les données des cartes, leurs points d'attaque, de défense, leur position sur le terrain etc... et cela pour chaque carte sur le terrain, chaque carte de sa main, et chaque carte dans son propre deck.

2. Une fois que l'IA a toutes les informations du terrain, elle va passer ce vecteur d'entrée dans son réseau neuronal, chacune des données entrant dans un neurone, puis l'information va se propager de neurone en neurone, couche par couche, de manière à ce que la dernière couche de neurone, la couche de sortie, compte exactement autant de neurone que d'actions possible à effectuer. Chacun de ces neurones de sortie aura alors une valeur numérique.



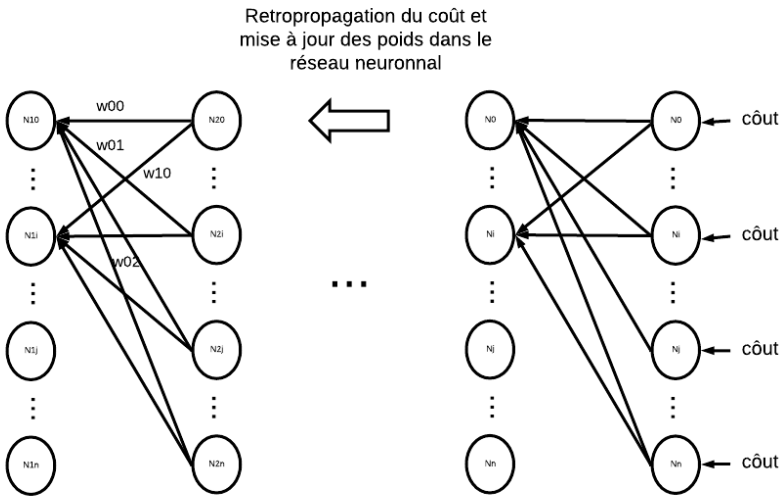
3. On applique ensuite l'algorithme du softmax sur ce vecteur de sortie, pour obtenir une pondération des actions possible en fonction du choix du réseau neuronal. Ce qui signifie que l'action que le réseau neuronal pense être la meilleur a effectuer aura la plus haute pondération, et la pire action à



effectuer aura la pondération la plus faible

4. L'IA va ensuite simuler chacune des actions possibles (également celles qui ont une pondération de zéro) et regarder à chaque fois l'état dans lequel le jeu se trouverait si cette action était effectué, elle va alors noter les états du jeu simulé et elle va comparer ces notes avec les pondérations trouvées par le réseau neuronal. Plus les valeurs sont proches (c'est à dire qu'un état avec une bonne note est associé à une action à forte pondération ou inversement qu'un état peu favorable avec une note faible est associé à une action à pondération faible) plus le coût du neurone en question sera faible, alors que si les notes et la pondération sont diamétralement opposés, le coût sera très élevé.

5. Une fois ces coûts établis, le réseau de neurones va mettre à jour les poids des synapses des neurones pour réduire le coût de chaque neurone. Ensuite ce coût par neurone est rétro-propagé dans le réseau neuronal, pour que chaque synapses soit rétro-propagée dans le but d'améliorer les performances de l'IA.



6. Après plusieurs milliers de parties jouées, l'IA aura accumulé assez d'expérience pour que la sortie du réseau neuronal, c'est à dire les pondérations de chaque actions possible à effectuer, soit le plus proche possible de la notation de l'état du jeu obtenu. A terme, l'IA calcul donc à chaque fois la meilleur action possible pour chaque coup à jouer.

Les phases de jeux étant bien distinctes, avec : la phase principale où l'on place ses cartes sur le terrain, et la phase de combat où l'on attaque l'adversaire avec les cartes déjà présentes sur le terrain. Nous avons donc deux moteurs d'IA, un qui s'occupe de choisir l'action à effectuer parmis les actions effectuables pendant la phase principale, et l'autre qui va choisir parmis les

actions effectuables pendant la phase de combat. Les deux IA ont donc chacun un réseau neuronal qui leur est propre.

Avec ce type d'IA il est assez difficile de trouver un moyen de faire en sorte d'avoir plusieurs difficultés d'IA, nous avons pensé au début à utiliser des IA plus ou moins entraînées pour avoir des IA plus et moins efficaces que les autre. Mais le défaut majeur de cette technique c'est qu'une IA peu entraîné aurait tendance à faire n'importe quoi plutôt que jouer de façon peu efficace. Le jeu reposant beaucoup sur la qualité du deck de carte du joueur, nous avons choisi d'utiliser la même IA pour toutes les difficultés, mais avec différents Deck de carte en fonction de la difficulté choisie.

La partie IA a été très difficile et longue à réaliser, et nous a posé pas mal de soucis. Le deep learning et notamment la rétropropagation du coût dans le réseau neuronal sont des choses très complexes et il a été très difficile de faire converger le réseau neuronal, surtout avec une application qui comporte un si grand nombre de données en jeux et un si grand nombre d'action effectuables. Le système d'apprentissage de l'IA n'est donc pas tout à fait au point et l'IA a tendance à effectuer tous le temps les mêmes actions. Mais cela est suffisant pour pouvoir réaliser des parti solo.

Interface Homme - Machine

Utilisateurs et Profil

De manière générale, l'application s'adresse aux joueurs de Yu-Gi-Oh ©, qu'ils soient novices ou experts. Aussi, nous avons fournis pour les premiers différents guides et aides pour leur permettre de se former au jeu. Pour les experts, nous avons notamment laissé la possibilité de personnaliser certaines règles du jeu afin de donner de nouvelles dimensions aux parties.

Les amateurs de jeux de cartes s'adonnant souvent à plusieurs jeux différents, nous avons cherché à nous inspirer des interfaces et possibilités offertes par d'autres applications de jeu, qu'elles soient liées à des jeux de cartes (HearthStone ©, Plants vs Zombies Heroes ©, ...) ou non (Call of Duty - Black Ops 3 ©, Sims 4 ©, …). Nous voulions ainsi obtenir un rendu ergonomique, cohérent et proche de ce que les utilisateurs ont l'habitude de rencontrer qui, en plus de leur donner un sentiment de familiarité, diminuerait leur temps d'apprentissage. Aussi, et pour augmenter la convivialité, nous avons adoptés une approche anthropocentriste en nous posant à chaque instant la question du point de vue de l'utilisateur ; de manière globale par rapport à l'application et centrée sur les éléments, leur comportement propre ou par rapport aux autres.

Bien que la tâche principale des utilisateur restait de jouer virtuellement aux cartes, nous avons définit que d'autres s'y ajoutaient de manière plus ou moins lié : consulter les règles, modifier son deck, changer certains paramètres, discuter avec l'adversaire. Différents scénarios ont donc été envisagés, comme par exemple :

- Le joueur veut jouer contre un ami.
- Le joueur ne connaît pas d'adversaire disponible.
- Après une défaite, le joueur veut modifier son deck.
- Le joueur veut juste consulter les règles.

Etant nous-même joueurs et futurs utilisateurs, nous avons pu travailler à leur cotés d'une certaine façon. Néanmoins, même en posant des questions relatives à certains éléments à quelqu'un travaillant sur d'autres, nous avons gardé conscience du caractère partial et biaisé de cet avis. Aussi, nous avons régulièrement fait appel à nos proches, joueurs ou non, pour nous renseigner sur leurs impressions. Ce recueil s'est fait de manière directive et non-directive, et lorsque cela était possible, avec la méthode du think-aloud. On peut citer en exemple de consigne directive :

- *(Depuis l'accueil)* → Change la taille du texte.
- *(Depuis le menu des paramètres)* → Lance une partie.

Enfin, en terme de matériel, le profil envisagé fut un ordinateur personnel, équipé d'une souris et/ou d'un clavier. En effet, que ce soit avec un clavier ou avec une souris, l'ensemble de l'application est accessible et utilisable, y compris lors des parties. A cette fin, des contrôles claviers ont d'ailleurs été ajoutés pour permettre un déplacement facilité sur le terrain, tout en conservant au mieux le comportement natif de touches comme la Tabulation.

La taille de l'application fut pensée de manière responsive, allant d'un petit écran de la taille d'un terminal standard (80 x 24 caractères de police 11) à une application en plein écran, sur un écran de type télévision grand-format. Ceci permettait alors autant à l'utilisateur de jouer sur une fenêtre réduite en faisant autre chose, que de profiter totalement du confort visuel d'un grand écran. On peut ici noter au passage que les tailles de la fenêtre sont conservés à chaque fois que l'utilisateur relance l'application, pour lui conférer un sentiment de stabilité par rapport à ses choix de redimensionnement.

Le système envisagé est Linux, mais un portage Windows a pu être entamé. De la même façon, l'ensemble de l'application fut pensé pour permettre la compatibilité avec le système Android, ou même un éventuel futur portage Web. C'est par exemple pour cette raison que, notamment dans les différents menus d'option, conformément à ce qui se fait généralement sur le web (ex : sites internet, menu du navigateur Chrome) ou dans les applications mobiles, la sauvegarde est automatique après chaque modification. De même, contrairement à certains jeux, il n'est pas nécessaire de relancer l'application pour prendre en compte les changements de paramètres, comme le passage en pleine écran.

Style et Ergonomie

De manière générale, nous avons opté pour un style conversationnel, notamment dans les menus d'options ou de création de decks/cartes, qui adoptent alors la structure des formulaires. Néanmoins, le style de la page des duels s'apparente davantage à un style par manipulation d'objets. Ainsi, même si cette option ne verra le jour que dans les versions ultérieures, cette interface a été pensée pour permettre d'implémenter une gestion des cartes par glisser-déposer (Drag-and-Drop). De plus, dans cette zone complexe qui reste celle où l'utilisateur est censé passer le plus de temps, deux styles cohabitent puisque le chat s'apparente à un style conversationnel. Le soin de ces deux parties l'une par rapport à l'autre et leur ergonomie mutuelle fut donc un point crucial dans le développement.

En terme de charte graphique, nous nous sommes imposé la contrainte d'obtenir une interface de type Flat Design, en couplant un style graphique adaptant les codes visuels de la franchise Yu-Gi-Oh © aux principes du Material Design. Ceci permet en effet d'obtenir une application utilisant un style que l'on retrouve régulièrement, de nos jours, dans les interfaces web et mobiles et que les utilisateurs trouveront donc familier. Le fait d'implémenter notre propre style

nous permet également d'obtenir un rendu homogène entre les plateformes. Une illustration des évolutions du style de l'interface est disponible en annexe 5.

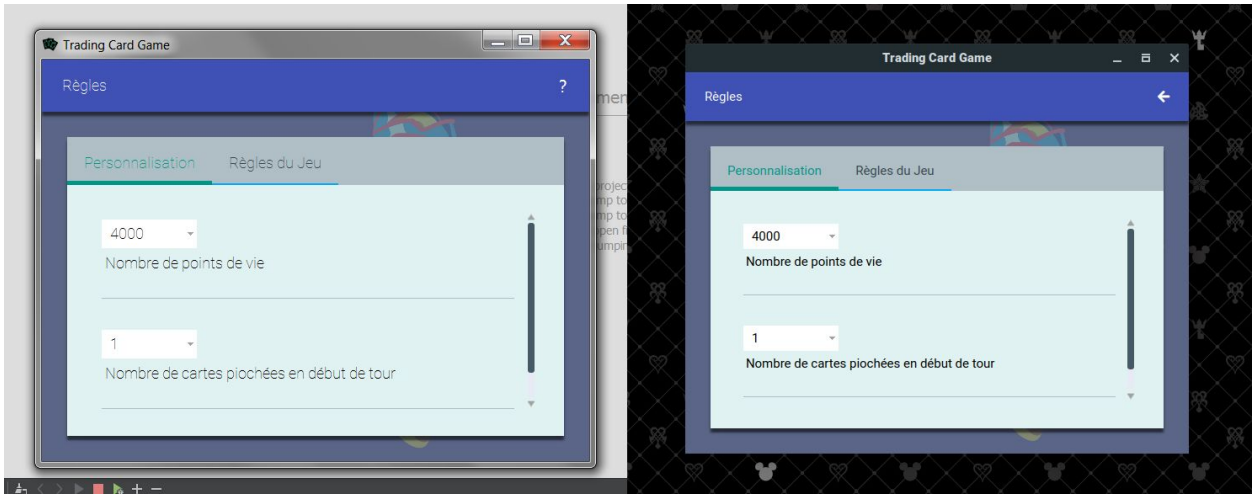


Illustration 5. Homogénéité du Style entre plateforme Windows (à gauche) et Linux (à droite).

Pour ce faire, nous avons notamment sélectionné une palette de couleurs la plus restreinte possible, en plus d'avoir réimplémenté la plupart des QWidgets existant. Ceci nous a alors autant permit de personnaliser leur style que d'ajuster leurs comportements. Néanmoins, nous avons gardé à l'esprit de conserver les conventions de la plateforme, que ce soit par exemple pour Windows ou pour Linux.

Nous avons également implémenté deux styles de polices différentes. Roboto d'une part, qui s'avère être la police préconisée par les guides relatifs au Material Design. Elle offre une écriture composée de lettres permettant une lecture simple et intuitive, notamment par ses courbes et l'absence de serifs et se prête bien à une utilisation dans un jeu. D'autre part, l'utilisation de Font Awesome nous a permit, pour l'équivalent en poid d'une image, de charger un grand ensemble d'icônes pour l'application. Ceci a alors accéléré le temps de chargement, donné une plus grande cohérence à l'application et une apparence davantage flat design ; malgré une plus grande difficulté d'implémentation au départ.

Nous avons également tâché de rendre l'environnement familier aux joueurs de Yu-Gi-Oh ©. Ainsi le plateau de jeu de l'application reprend les couleurs du plateau officiel, comme le montre l'illustration 2. De la même façon, nous comptions réutiliser un système de template pour afficher l'ensemble des informations de manière lisible. Néanmoins, cette façon de faire s'est avérée non-responsive. Aussi, nous avons recréé un affichage complet à base de QWidget pour simuler la prévisualisation d'une carte (illustration 3).

Le guidage des utilisateurs se fait alors de manière la plus implicite possible. Néanmoins, pour chaque action, des tooltips peuvent renseigner l'utilisateur sur les conséquences de son action,

notamment en vue d'augmenter sa capacité de contrôle. Le fait d'avoir inclus une sauvegarde automatique à presque chaque action permet également à l'utilisateur de connaître l'état du système dans lequel il se trouve. De même, en dehors des duels, les actions sont rétro-actives et permettent à l'utilisateur d'aussitôt revenir sur un de ses choix. Lorsque ce n'est pas le cas, comme dans le menu de création de cartes, un QDialog permet par exemple d'indiquer qu'une carte a été enregistrée.

Nous avons également pris en charge les erreurs, par exemple dans les cas où les decks n'existent pas, ou quand un adversaire se déconnecte. Les autres cas d'erreurs qui résultent sont difficiles à prendre en charge, et relèvent davantage du système de l'utilisateur que de l'application. Dans le cas où un comportement inattendu survient néanmoins, notamment lors des parties, un fichier .log est initialisé et permet à l'utilisateur de transmettre les informations adéquates aux développeurs.

Accessibilité

En terme d'accessibilité, nous avons inclus des spécifications à la fois implicites et explicites. En effet, de manière explicite à l'utilisateur, nous avons inclus des options dédiées dans le menu des paramètres. Ainsi, le joueur peut au choix sélectionner une police de grande taille, une police adaptée à la dyslexie, OpenDyslexic, ou encore des affichages au contraste élevé ou adaptés à la dyschromatopsie. Néanmoins, pour cette dernière, nous regrettons de ne pas avoir eut à nos côté un expert de cette pathologie pour nous aider à implémenter des options dédiées à la majorité ou à la totalité de ces types d'affections visuelles.

De manière implicite, nous avons également voulu rendre accessible l'application aux personnes en situation de handicap, autant en terme visuel qu'au niveau du comportement de l'interface. Cela passe, par exemple, par la possibilité d'augmenter la taille des cartes du terrain en redimensionnant la fenêtre. De même, il n'était initialement possible d'afficher la description complète d'une carte du terrain qu'au survol de celle-ci. Nous avons donc d'une part, implémenté ce comportement de prévisualisation par rapport à une navigation au clavier. D'autre part, pour des personnes ayant des difficultés à maintenir la souris, par exemple dans le cas de pathologies telles que le Trouble de Déficit de l'Attention/Hyperactivité (TDAH) ou la Maladie de Parkinson, nous avons implémenté la possibilité de verrouiller la prévisualisation au moyen du clique-droit. Un autre clique-droit sur une carte du terrain ou un clique sur la prévisualisation permet de retirer ce verrouillage.

Comme évoqué précédemment, l'application est accessible autant à la souris uniquement, au clavier uniquement, ou les deux. Elle a également été testée, avec succès, avec un pointeur de type stylet. En théorie, elle est également accessible pour utilisateurs disposant d'un joystick. Nous avons tenté de rendre l'application la plus agréable possible à parcourir au clavier, quitte à

changer le comportement initial de nombreuses touches. Une attention particulière a été portée sur le comportement des flèches directionnelles, puisque ces touches attirent instinctivement l'utilisateur quand il découvre l'application. Or, par défaut, c'est Tab qui est utilisé pour naviguer entre les Widget dans une application QT.

En ce qui concerne les menus, nous avons remplacé les fonctionnement des flèches par ceux de "Tab" ou de "Maj+Tab". Ces comportements rendent la navigation dans les menus bien plus agréable mais ont également apporté certaines difficultés : nos menus comportent en effet certains Widget de type Input, utilisant les flèches directionnelles pour beaucoup de choses, nous avons donc dû garder le fonctionnement initial des flèches dans ces Widgets. Au final, pour sortir d'un QLineEdit, un utilisateur doit utiliser la touche "Tab" (sauf si le QLineEdit est vide, auquel cas les flèches permettent encore d'en sortir), et pour sortir d'un QTextEdit un utilisateur doit utiliser "Maj+Tab" (sauf si le QTextEdit est vide).

En ce qui concerne les contrôles clavier durant un duel, tout est jouable au clavier, avec certains contrôles plus particulier ajoutés et spécifiés dans le menu d'aide de l'appli. Nous nous sommes arrangés (notamment avec les focus de chaque Widget) pour que l'utilisateur ne se perde pas quoi qu'il arrive. Il est donc par défaut "coincé" sur le terrain et peut accéder au menu d'abandon avec "ESC" et au Chat avec la touche "F1".

Nous avons également, dans une logique d'internationalisation, décidé de traiter trois langages dans notre application: le français, l'anglais et l'arabe. La langue choisie dans l'application est par défaut celle du système. L'utilisateur peut néanmoins choisir entre les trois via une comboBox dans le menu options. Pour ce faire, nous avons utilisé les outils fournis par QT, à savoir : QTranslator, lupdate et lrelease, ainsi que QLinguist. La langue choisie par l'utilisateur dans le menu options est alors sauvegardée dans une QSettings.

En utilisant uniquement ces outils, nous nous sommes aperçu que le menu de sélection de la langue ne s'actualisait qu'après un redémarrage. Nous avons ensuite décidé de rendre le changement de langue dynamique, sans redémarrage nécessaire, pour correspondre au comportement du reste de l'application. Il a donc été nécessaire de recharger le QTranslator de l'application à chaque changement de langue et de réimplémenter certaines fonctions pour qu'elles actualisent également les textes du menu options lors d'un changement de langue.

Le traitement de la langue arabe n'étant pas une simple traduction, il impliquait également de passer en mode "right to left", norme régionale correspondante qui impose un effet miroir par rapport à l'implémentation européenne. Pour ce faire, malgré les outils fournis par QT comme la fonction setLayoutDirection(Qt::RightToLeft), il a encore été nécessaire de faire quelques ajustements, notamment au niveau des contrôles clavier sur le terrain.

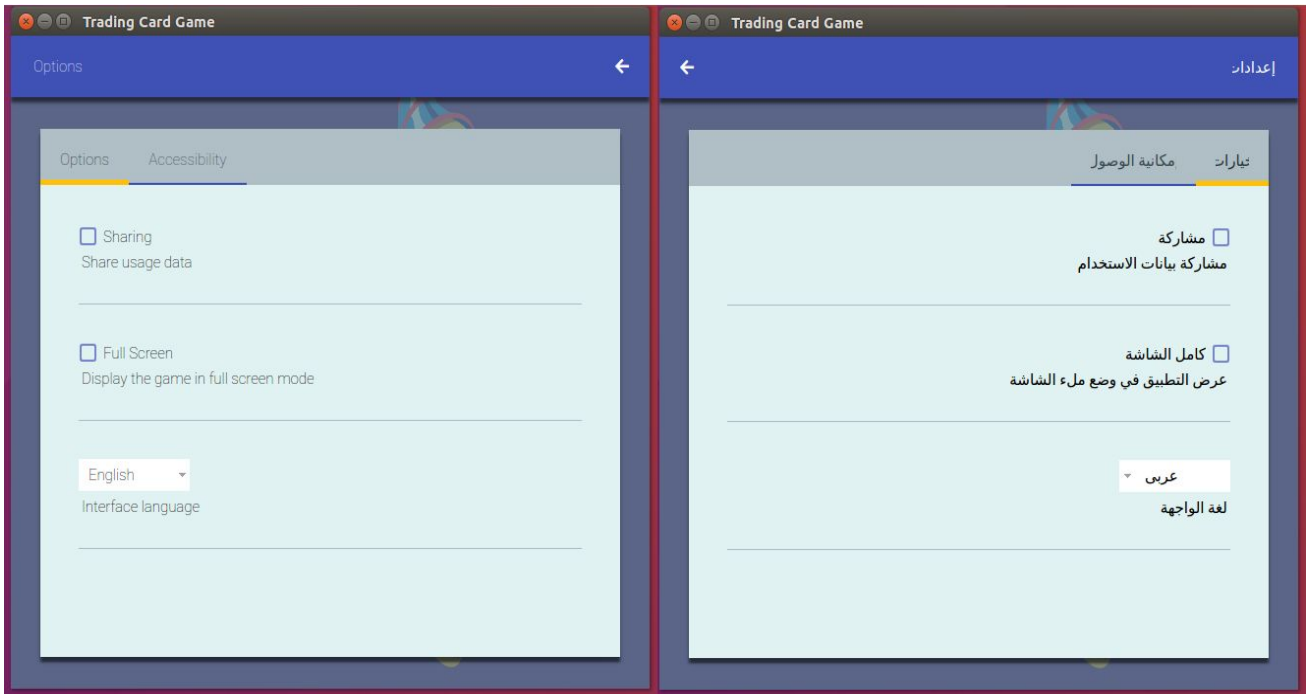


Illustration 6. Le même menu en left-to-right (à gauche) puis en right-to-left (à droite)

En raison de la possibilité offerte à l'utilisateur de créer des cartes personnalisées, il nous est apparu impossible de traduire l'intégralité des cartes envoyées d'un utilisateur à un autre lors d'une partie en ligne, dans leur langues respectives. Aussi, l'équipe s'est accordée pour, dans le cas d'un affrontement avec un joueur utilisant une interface dans une autre langue, lui transmettre les cartes dans la langue d'origine. Dans le cadre de versions futures, on peut envisager un certain verrouillage des parties onlines, pour ne laisser par exemple s'affronter que des joueurs de même langue. Une autre solution envisageable serait d'interdire les cartes personnalisées en dehors des parties en solo ou entre amis.

Enfin, nous avons été confronté à la problématique de la régionalisation de l'application. En effet, de nombreux graphiques liés au jeu Yu-Gi-Oh © peuvent s'avérer choquant pour certaines cultures. Après discussion et réflexion, il a alors été convenu de ne pas filtrer ces contenus pour ne pas altérer l'esprit de la licence. Une version future comprenant différents graphiques en fonction de la région du jeu peut néanmoins être envisagé.

Discussion

En terme d'évolutions et d'un point de vue général, nous avons été bloqué par les limites de la plateforme Turing, notamment coté serveur. Une manière de pallier à ce problème serait de souscrire un abonnement chez Amazon (AWS) ou OVH, pour nous permettre de placer le serveur sur un site ouvert et accessible.

En lien avec le réseau, une évolution envisagée fut la possibilité d'inclure un système de pseudonymes pour le joueur. Celui-ci pourrait alors directement contacter un autre joueur par ce biais, sans passer par le systèmes d'IPV6 qui peut être jugé très ou trop technocentré. L'interface deviendrait également plus convivial et personnelle, en affichant directement les pseudonymes plutôt que des pronoms comme "il" ou "vous" dans les messages. Enfin, ceci pourrait constituer la base d'un nouveau mode de jeu : le mode tournoi.

Toujours d'un point de vue général, un plus pour notre application serait de la transformer en un outil réellement multi-plateforme. Comme évoqué, nous avons déjà commencé le portage sur Windows. Grâce au choix dès le départ du style Material Design (larges boutons, interface espacée, …), il nous apparaît également plus facile de passer à une plateforme Web ou Android, sans modifier nos composants. Néanmoins, nos premiers tests avec les outils mis à disposition par QT s'avèrent peu concluant pour un portage Android, et il pourrait s'avérer nécessaire de passer par la plateforme de développement Android Studio pour réaliser cette tâche.

Concernant un éventuel portage Web, il imposerait de réimplémenter totalement l'interface sur des bases de type HTML/Javascript côté client, et de disposer d'un serveur soit global, soit propre à chaque joueur (ex: node.js) afin de conserver la possibilité des parties entre amis. Néanmoins, cette inconvénient peut être compensé par une interface plus dynamique et plus riche, les technologies web possédant un point de maturité plus avancé dans ce domaine. De même, si les versions ultérieures de l'application intègrent un rendu plus riche à base d'effets OpenGl, par exemple lors de l'activation d'une carte magique, il sera possible de les réimplémenter en Web au moyens des outils de type Canvas ou WebGl.

De manière plus spécifique, le choix d'implémenter un environnement sonore dans des versions ultérieurs s'est vite imposé à nous. En effet, en plus de fournir un caractère plaisant à l'utilisateur, il permet de donner davantage de caractère et d'empreinte à l'application, en particulier lorsque le joueur reconnaît des thèmes ou sons qu'il a déjà entendu ; par exemple dans notre cas, dans la série animée Yu-Gi-Oh ©. Néanmoins, ce travail peut nécessiter une version de l'application à lui tout seul, entre le choix des sons, la façon de les implémenter et la possibilité de désactiver cet environnement dès que souhaité, notamment lorsque l'application est en plein écran.

Par rapport au rendu actuel, nous aurions également voulu rajouter différentes options ou capacités pour le joueur. Parmis celles qui ont retenus notre attention, on peut citer les fonctionnalités de glisser-déposer les cartes du terrain, en plus du comportement actuel par rapport aux cartes.

Concernant la limite de temps entre chacune des phases, il a été difficile de la concevoir. D'une part, elle est nécessaire pour que même si un joueur ayant la main ne fait rien, le joueur adverse puisse activer des pièges à ce moment là. Elle permet également de vérifier que l'utilisateur est toujours actif. Néanmoins, elle est parfois jugée trop courte quand beaucoup de cartes sont à jouer, et trop longue quand aucun des deux joueurs ne réalise d'action. Aussi, nous avons pensé à concevoir pour de futures version un système de *handshake*, où après un clique sur un bouton de la part des deux joueurs, la phase de terminerai. Ceci pourrait alors permettre d'allonger la phase, tout en permettant de la raccourcir à l'envie des deux joueurs.

Concernant le chat, faire parler l'IA, avec un set de phrases choisies au préalable en fonction des situations aurait également été agréable pour l'utilisateur ; autant pour lui permettre de comprendre l'état du système ou de la partie que pour agrémentées celles-ci. Toujours concernant le chat, un système de jeu par commande a été envisagé, pour permettre aux joueurs de jouer selon un style conversationnel de type console. L'utilisateur aurait alors pu entrer des commandes de type *#Invoquer le dragon blanc aux yeux bleus en mode attaque* pour initialiser cette action.

Ce projet nous aura ainsi permis d'une part de nous initier à la programmation en C++, que ce soit de manière général avec l'implémentation de classes ou de manière plus avancée en manipulant des threads ou en communiquant via le réseau, et de découvrir d'autre part un nouveau framework de développement d'application : QT. Nous avons en effet pu retrouver les concepts découvert en Java comme l'encapsulation ou la surcharge, par exemple en multipliant les différents types de QPushButton pour qu'ils s'adaptent à l'ensemble de nos besoins. Ceci nous a néanmoins également permit de voir ce qui nous apparaît comme étant une limite par rapport à la création d'interface en technologies web : quand, pour schématiser, dans un cas une classe ou un identifiant suffisent à multiplier les comportements, un grand nombre de lignes de codes sont nécessaires pour obtenir le même rendu dans l'autre cas. Malgré tout, ceci nous permet de constater les tâches que peuvent effectuer le navigateur ou un moteur comme node.js pour permettre cette simplification. De même, contrairement aux technologies web, la programmation d'interfaces via des outils plus bas niveau comme C++ apparaît bien moins limitée en termes de fonctionnalités, à condition d'y passer le temps nécessaire.

Plus généralement, les membres de l'équipe ayant tous réalisé les projets antérieurs de manière individuels, ce projet nous a permit de découvrir une nouvelle manière de travailler, en équipe, en prenant en compte les besoins et motivations de chacun, en trouvant des solutions collectives plutôt qu'individuelles, et en travaillant main dans la main.

Nous remercions chaleureusement Régis Witz pour nous avoir permit de choisir et de travailler sur ce projet en toute liberté, pour les conseils et connaissances qu'il nous a apporté et pour le soutien qu'il nous a fournit.

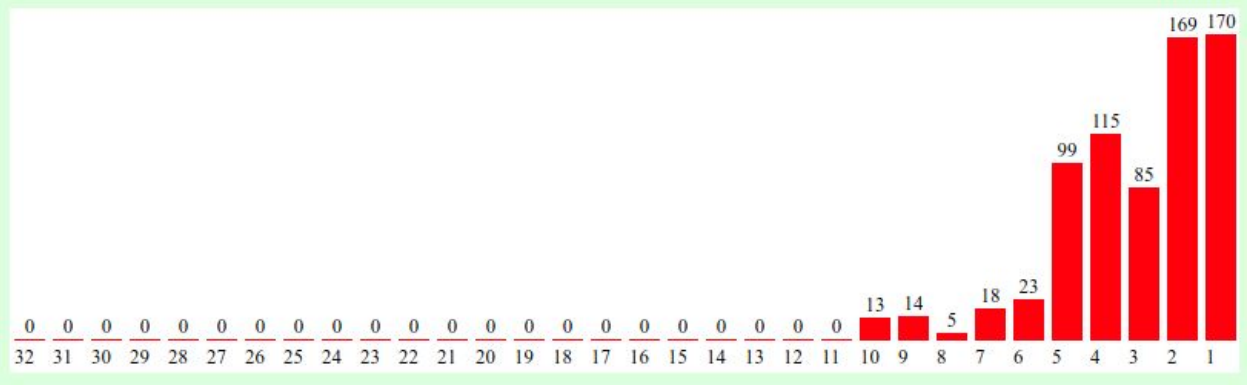
Annexe 1: livret de règles du jeu

http://www.yugioh-card.com/ygo_cms/ygo/all/uploads/Rulebook_v9_fr.pdf

Egalement disponible sur Github :

https://github.com/4rkiel/yugioh/blob/master/Rulebook_v9_fr.pdf

Annexe 2 : Evolution globale du projet

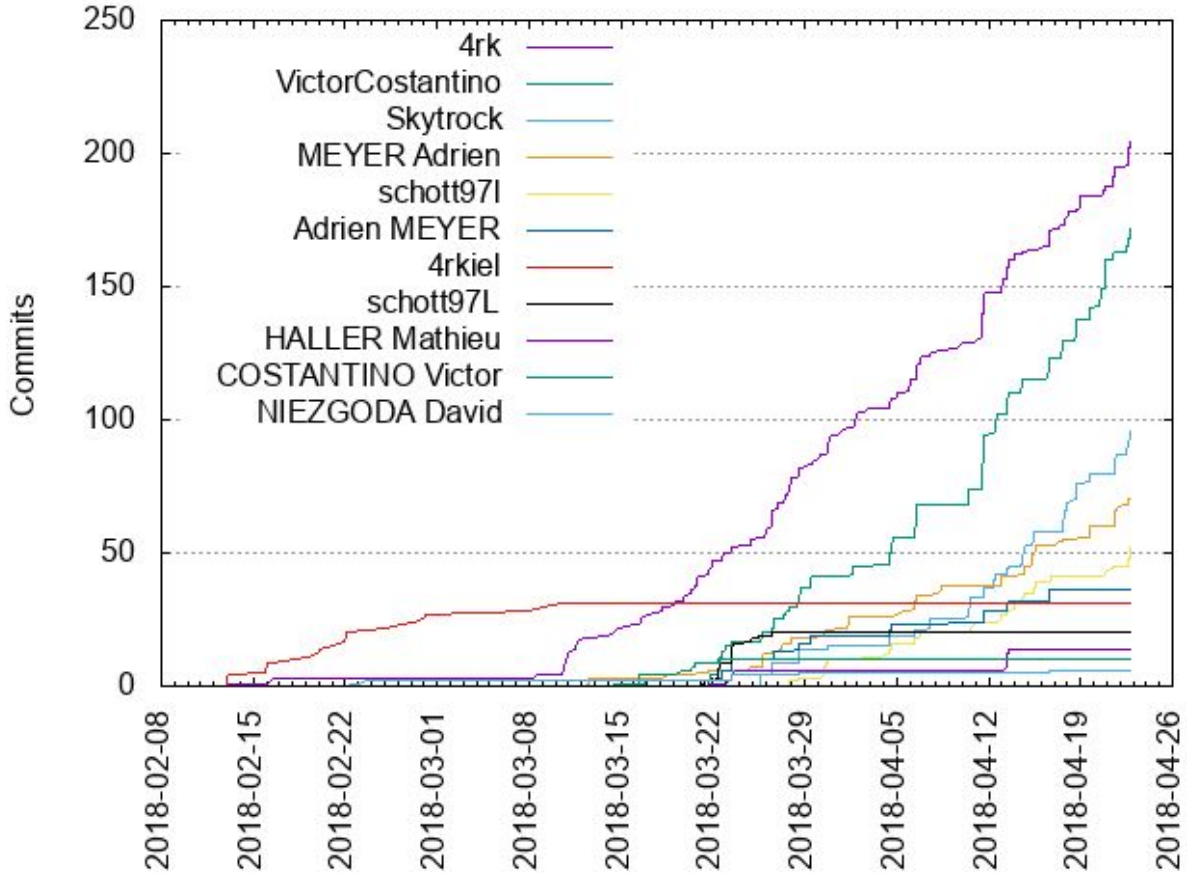


Annexe 3 : Tableau de la répartition des commits

Niezgoda David : 4rk, 4rkiel, Niezgoda David
Victor Costantino : VictorCostantino, COSTANTINO Victor
Haller Mathieu : Skytrock, HALLER Mathieu
Meyer Adrien : MEYER Adrien, Adrien MEYER, Chop1, Wargamme
Schott Lucas: schott97l, schott97L

Author	Commits (%)	+ lines	- lines	First commit	Last commit	Age	Active days	# by commits
4rk	204 (28.69%)	51272	31404	2018-02-12	2018-04-22	68 days, 21:12:59	44	1
VictorCostantino	172 (24.19%)	31820	12871	2018-03-14	2018-04-22	39 days, 5:45:43	26	2
Skytrock	96 (13.50%)	17155	19335	2018-03-25	2018-04-22	28 days, 4:59:06	18	3
MEYER Adrien	70 (9.85%)	5457	6371	2018-03-12	2018-04-22	41 days, 3:51:42	26	4
schott97l	52 (7.31%)	25784	364250	2018-03-27	2018-04-22	26 days, 0:00:07	15	5
Adrien MEYER	36 (5.06%)	2525	1860	2018-03-21	2018-04-16	26 days, 1:23:34	11	6
4rkiel	31 (4.36%)	10707	10356	2018-02-12	2018-03-10	25 days, 14:31:07	16	7
schott97L	20 (2.81%)	346123	1202	2018-03-21	2018-03-26	5 days, 5:13:14	6	8
HALLER Mathieu	14 (1.97%)	8479	8713	2018-03-21	2018-04-13	22 days, 21:54:54	3	9
COSTANTINO Victor	10 (1.41%)	2824	1388	2018-03-16	2018-03-22	6 days, 0:46:13	4	10
NIEZGODA David	6 (0.84%)	5046	406	2018-02-22	2018-04-16	52 days, 23:40:04	5	11

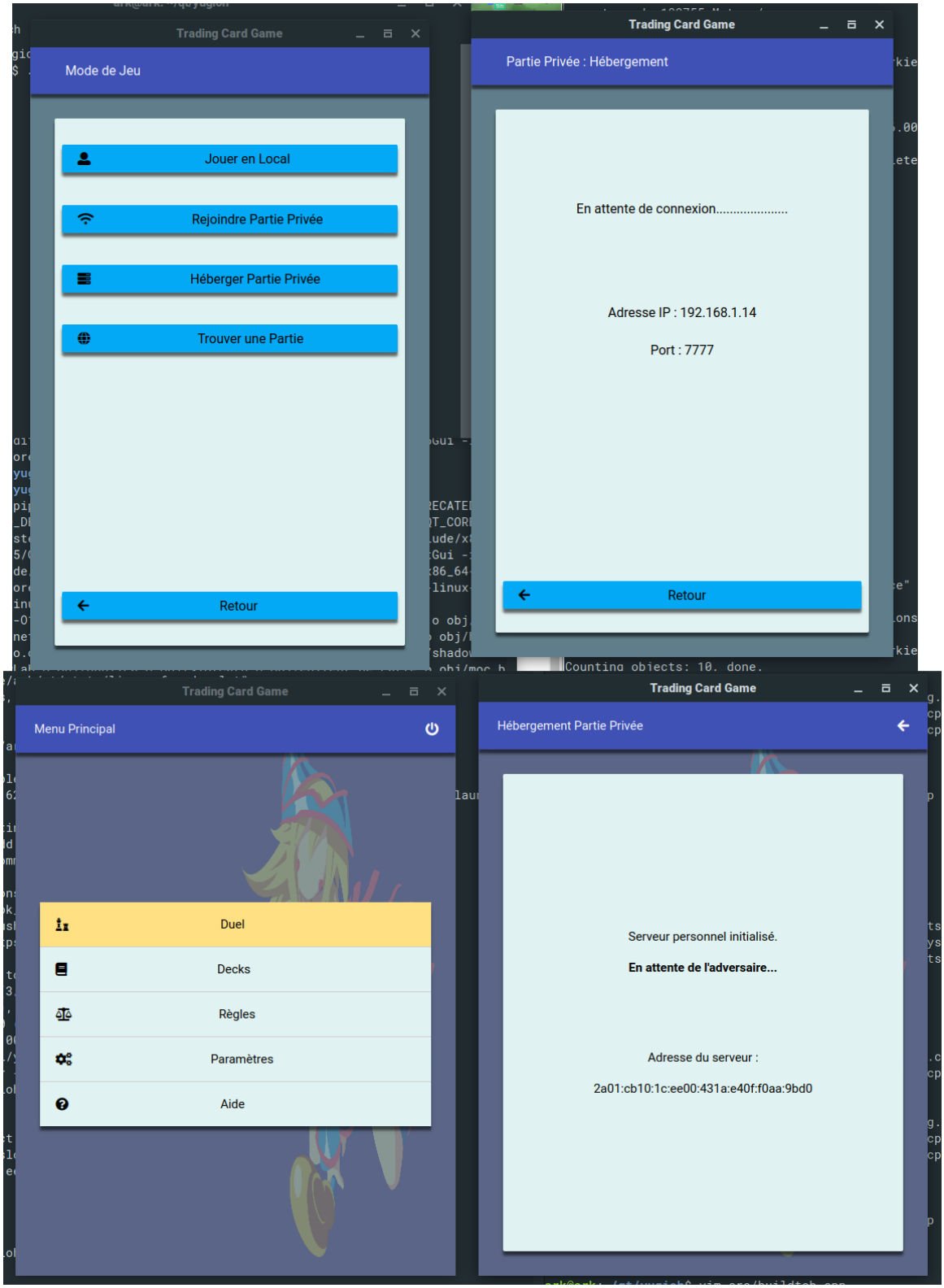
Annexe 4 : Graphique de la répartition des commits



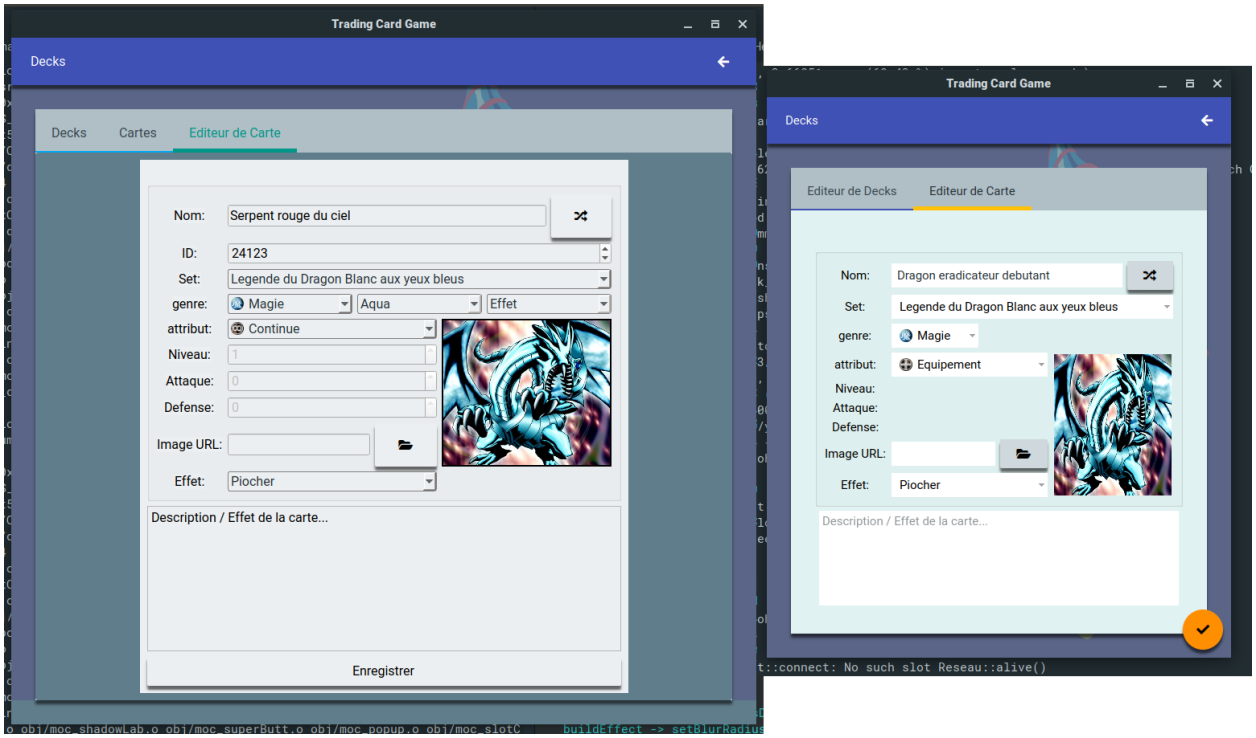
Annexe 5 : graphique UML de l'application

Voir Y-UML.png, disponible à la racine du projet :
https://github.com/4rkiel/yugioh/blob/master/Y-UML.png

Annexe 5 : Evolution du style graphique



En haut, avant. En bas, après.



A gauche, avant. A droite, après.