L021 - Projet

Rapport 1

Schotten Totten

P23

Grzadziel Bartlomiej, Piteux Nathan, Scalabre Mattéo, Leprêtre Thomas, Hermoso Guillaume

Sommaire:

I. Introduction :

Durant ce semestre et dans le cadre de l'UV LO21, nous allons travailler sur un projet de groupe. Le but de ce dernier est de nous familiariser avec la programmation orientée objet en produisant une version numérique du jeu Schotten-Totten. Nous mettrons en application dans ce projet les différentes notions vues en cours de LO21.

Dans ce premier rapport, nous présenterons les premières analyses que nous avons faîtes sur le sujet. Nous les expliciterons à l'aide d'un premier schéma UML. Ce schéma sera amené à être modifié durant nos différentes réunions jusqu'à atteindre une version suffisamment proche de la réalité.

Nous allons dans un premier temps expliquer brièvement le fonctionnement d'une partie de jeu. Ensuite, nous expliquerons notre compréhension du sujet vis-à-vis des modules, des classes et des concepts à utiliser. Enfin, nous présenterons notre première version du schéma UML ainsi que la répartition des tâches.

II. Explication d'une partie classique :

a. De quoi est composé le jeu ?

Le plateau de jeu :

Le plateau de jeu est constitué de neuf bornes indépendantes côte à côte situées entre les deux joueurs. On retrouve la pioche contenant les cartes clan. De chaque côté des bornes vont apparaître des cartes disposées par les joueurs au fur et à mesure de la partie.

Les cartes :

Le jeu Schotten Totten contient un deck de 54 cartes clan, constitué de cartes numérotées de 1 à 9 dans six couleurs différentes. Chaque carte est présente en un unique exemplaire. Les cartes sont utilisées pour former des combinaisons de trois cartes sur chaque borne, afin de gagner des points. Elles peuvent se trouver dans la pioche, dans la main d'un joueur ou sur une borne.

Les bornes :

Les bornes sont au nombre de neuf. Chaque borne peut accueillir 3 cartes de part et d'autre. Les joueurs posent les cartes de chaque côté pour effectuer des combinaisons.

b. Quel est le but du jeu ?

Dans Schotten-Totten, le but est de revendiquer des bornes. Chaque joueur va devoir user de sa meilleure stratégie pour poser habilement ses cartes et rafler les meilleures bornes en faisant les meilleures combinaisons possibles. De la meilleure à la moins forte, les combinaisons sont :

- La suite couleur : 3 cartes qui se suivent dans la même couleur
- Le Brelan : 3 cartes de même valeur
- La couleur : 3 cartes de la même couleur
- La suite : 3 cartes qui se suivent peu importe la couleur
- La somme : Somme des valeurs des cartes

c. Déroulé d'une partie :

On installe les 9 bornes entre les 2 joueurs. On mélange le deck de cartes et on distribue 6 cartes à chaque joueur. On désigne le premier joueur qui peut commencer à jouer.

A son tour, un joueur choisit une des cartes de sa main et la pose de son côté devant une borne qui ne contient pas déjà 3 cartes afin de commencer ou de compléter ses combinaisons. Puis à la fin de son tour il pioche une carte pour toujours posséder 6 cartes en main.

Avant de piocher, il peut revendiquer une borne :

- Si la borne possède 3 cartes de part et d'autre, elle est complète. On peut alors procéder à l'évaluation en comparant les combinaisons. Celui qui possède la plus forte gagne la borne. Si les 2 combinaisons sont de même puissance, on fait la somme des valeurs des cartes de chaque côté et on regarde la plus élevée. Si l'égalité persiste, c'est le joueur à avoir posé sa troisième carte en premier qui remporte la borne

- Si le joueur dont c'est le tour a posé sa troisième carte et complété sa combinaison, mais que de l'autre côté le joueur adverse a une combinaison incomplète, il peut revendiquer la borne en prouvant que l'adversaire ne pourra pas la contester peu importe la prochaine carte qu'il jouera. Pour prouver sa supériorité, le joueur est obligé de s'appuyer uniquement sur les cartes présentes sur le plateau.

Le premier joueur à posséder 5 bornes gagne la partie, ou avant cela si un joueur possède 3 bornes adjacentes alors il gagne également.

III. Variante Tactique :

La variante tactique apporte de la profondeur au jeu. Il est plus facile de retourner la situation à notre avantage grâce aux nouvelles cartes ajoutées au jeu. La variante apporte 9 cartes inédites différentes divisées en 3 sous-catégories :

- **Les Troupes d'élites** (se jouent comme une carte clan) :

  Les cartes tactiques Troupes d'élites sont des cartes qui vont se jouer devant les bornes, à la manière des cartes clans. Elles permettent de choisir leur valeur et couleur au moment de la revendication de la borne, sous certaines restrictions selon la carte posée.

- **Les Modes de combat** (se jouent sur une tuile borne) :

  Ces cartes se posent directement sur les bornes. Elles permettent de modifier le comportement de la borne au niveau de la revendication. Cela peut facilement retourner des situations en notre faveur au cours de la partie.

- **Les Ruses** (se jouent face visible à côté de la pioche) :

  Ce sont des cartes à usage unique qui permettent de modifier sa main ou le plateau. Tant qu'une borne n'est pas revendiquée, les jeux ne sont pas encore faits dans la variante tactique, les cartes Ruses en sont justement la preuve. Elles permettent notamment de déplacer, voler ou supprimer une carte du plateau.

Au début de la partie, chaque joueur reçoit 7 cartes clans au lieu de 6 et on dispose une nouvelle pioche de carte tactique sur le plateau à côté de la pioche de base. A la fin de son tour, au lieu de piocher une carte clan, un joueur peut décider de piocher une carte tactique. Une carte tactique se joue pendant son tour à la manière d'une carte clan et on suit l'effet selon la carte posée. Un joueur peut piocher autant de cartes tactiques qu'il souhaite mais il ne peut en jouer qu'une de plus que son adversaire, et il ne peut jouer qu'un seul joker.

Au moment de revendiquer une borne, il faut faire attention à bien appliquer les effets des cartes tactiques en jeu, notamment les effets des modes de combats. Mais dans le processus pour prouver sa supériorité lorsque l'on souhaite revendiquer une borne en avance, les cartes tactiques non jouées n'entrent pas en jeu.

Le reste de la partie se déroule de la même manière que le jeu de base.

IV. Classes :

a) Etude des classes utilisées dans le sujet

Lors d'une première réunion de projet, nous avons déterminé les classes suivantes, mais celles-ci sont amenées à être modifiées ou supprimées, et de nouvelles classes à être rajoutées.

Nous avons pour l'instant déterminé les classes suivantes :

- **Classe Carte :** Classe permettant de regrouper toutes les cartes du jeu sous une même classe. Elle ne présente aucun attribut ou méthode spécifique à une carte en particulier.
- **Classe CarteClan :** Hérite de la classe carte, et représente toutes les cartes clans. Elle possède deux attributs : la couleur de type énuméré et la puissance de type int.
- **Classe CarteTactique :** Hérite de la classe carte et représente toutes les cartes tactiques. Elle possède un unique attribut nom de type string, désignant le type de la carte.
- **Classe CarteTroupeElite :** Cette classe hérite des deux classes **CarteTactique** et **CarteClan** (pour pouvoir se retrouver dans les tableaux des deux types de cartes). Elle représente les cartes joker, espion et porte-bouclier. Elle ne possède pas d'attribut supplémentaire, mais comporte une méthode pour chacune de ces cartes.
- **Classe CarteModeCombat :** Hérite de la classe Carte tactique et représente les cartes Colin-Maillart et Combat de Boue. Elle ne possède pas d'attribut supplémentaire mais deux méthodes, pour les deux cartes.
- **Classe CarteRuse :** Cette classe hérite de la classe **CarteTactique**. Elle représente les cartes Chasseur de tête, Stratège, Traître et Banshee. Elle ne possède pas d'attribut supplémentaire, mais comporte une méthode permettant la résolution de l'effet pour chacune de ces cartes.
- **Classe Borne :** Cette classe représente les bornes du jeu. Elle possède différents attributs : son numéro pour pouvoir identifier les bornes adjacentes lors de la vérification de fin de partie , les cartes jouées sur cette borne par les joueurs représentés par 2 tableaux de **Cartes**, 2 attributs booléens pour savoir si les cartes Colin-Maillard et Combat de boue ont été jouées sur cette borne, un attribut de type int pour connaître le joueur ayant posé sa troisième carte en premier et enfin un attribut pour le joueur ayant revendiqué la borne. Elle possède également les méthodes pour obtenir ses différents attributs, les modifier, et pour ajouter et retirer des cartes sur cette bornes. On dispose également de la méthode d'évaluation pour revendiquer une borne.
- **Classe Partie :** Cette classe représente le plateau de jeu : elle contient un tableau contenant les 9 bornes, les différentes pioches (clans et tactiques) et les mains des 2 joueurs, ainsi que les informations sur la partie en cours (variantes, joueur actif, gagnant). Elle possède les méthodes permettant le déroulement de la partie (piocher une carte, revendiquer une borne…)

- **Classe Joueur :** Cette classe représente un joueur, et contient sa main et son “type” (humain ou IA).
- **Classe Pioche :** Cette classe représente une pioche. Elle a pour attribut le nombre de cartes restantes dans la pioche, et un vecteur de cartes, ainsi que les méthodes pour obtenir ses attributs et piocher une carte.
- **Classe Main :** Cette classe représente la main d’un joueur, et a pour attribut le nombre de cartes, les différentes cartes composants la main, et deux attributs pour savoir le nombre de cartes tactiques jouées et si un joker a déjà été joué.

Après le td4 et le cours du 30/03, nous réfléchissons aux modifications suivantes :

- Séparer la classe **Partie** en deux, une classe **Plateau** qui gèrera tous les éléments du jeu (les bornes, pioches et mains des joueurs dans une partie…)et une classe **Controleur** qui s’occupera du déroulement d’une partie et qui suivra normalement le modèle Singleton.
- Ajouter une ou deux classes **Jeu**, aussi un Singleton, qui génèrera les différentes cartes en séparant potentiellement les cartes normales des cartes tactiques. Les classes pioches seront ensuite associées à un jeu.
- Ajouter des compositions et des agrégation.
- Ajouter des méthodes d’affichages (en console dans un premier temps) pour les différentes cartes, et certains tableaux de cartes.
- Ajouter une classe **Combinaison**, remplaçant les tableaux de cartes clan dans les différentes bornes, avec différentes méthodes retournant le total des points, et si cette **Combinaison** est une suite/couleur…
- Réfléchir aux classes nécessitant un modèle **Iterator**.

b) Analyse des concepts

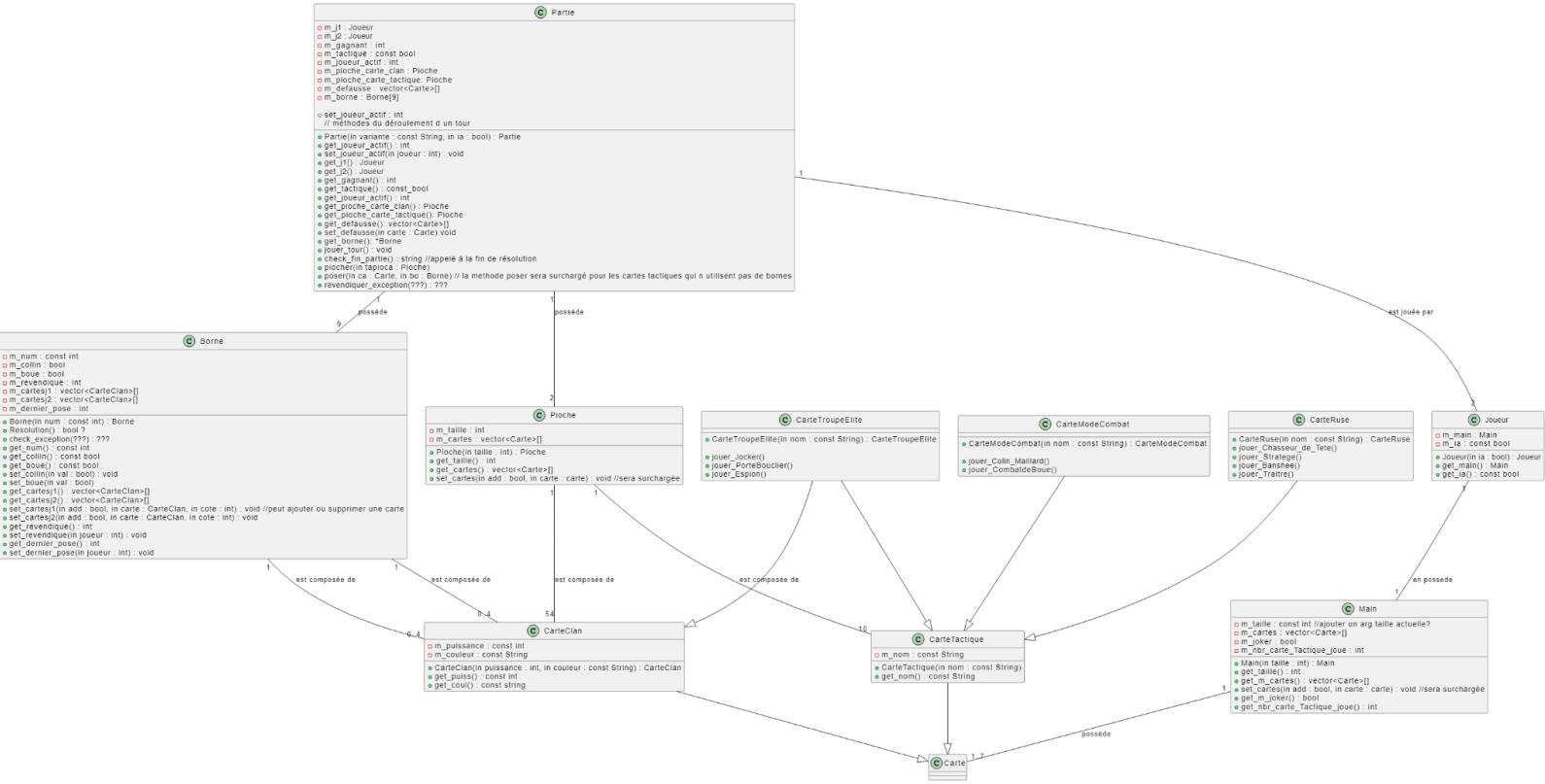
i) Classe et objet

Ces concepts sont à la base de notre projet. Ils viendront, par leur nature, diviser les différentes notions liées au projet afin d’en expliciter leur existence. Dans notre cas, les objets physiques cartes (par exemple) seront modélisés par des objets d’une classe que l’on appellera carte. Ainsi, une carte devient un objet manipulable, associable, destructible, assignable (nous aborderons ces thématiques dans un prochain rapport quand nous aurons une idée plus claire des structures de nos classes). Chaque objet ou concept du jeu (une partie n’est pas physiquement manipulable, mais sera un objet dans notre jeu) sera modélisé sous forme de classe. Les liaisons que nous établirons d’abord en UML puis implémenterons en C++ formeront notre jeu. C’est pourquoi le concept d’objet et de classe est fondamental dans notre sujet.

ii) Héritage et héritage multiple

L'héritage sera fondamental dans notre projet. En effet, par la structure du jeu, il existe différents types de cartes (clans et tactiques). Autant de cartes qui nécessiteront probablement leur propre classe, pour permettre de définir leur propre effet. C'est ici que le concept d'héritage nous aidera à la tâche, car les attributs et la structure de chaque classe mère n'auront pas à être redéfinie dans les classes filles, ces dernières ne contiendront que les ajouts par rapport à leur classe mère. De plus, l'héritage présente un avantage pour l'extensibilité de l'application.

V. UML :



NB : la méthode check_exception de la classe Borne qui fait référence à la règle permettant de revendiquer une borne si on peut prouver qu'elle est forcément gagnée n'a pas encore d'attribut car nous n'avons pas réfléchi à comment l'implémenter

VI. Répartition des tâches :

Tâches terminées :

- Découverte du jeu ; Tous ; 4h
- Analyse des mécaniques du jeu ; Tous ; 2h
- Création de l'UML ; Tous ; 10h (confondues)
- Rédaction du rapport ; Tous ; 7h

Tâches à faire :

- Modification de notre architecture Uml ; indispensable
- Programmation des premières classes en C++ ; importante (celles à la racine de nos héritages)
- Programmer le jeu en Version console ; à faire
- Programmer le jeu en version graphique avec QT ; à faire
- Rédiger le 2ème rapport ; à faire

VII. Conclusion :

Ces trois premières semaines nous ont permis de nous familiariser avec le jeu Schotten-Totten. Nous avons alors pu éclaircir les notions et concepts qui nous seront utiles pour mener à bien le projet. Même si toutefois encore beaucoup d'interrogations persistent autour de l'architecture de notre projet.

Ce projet sera pour nous l'occasion de mobiliser les connaissances théoriques acquises lors des cours magistraux et travaux dirigés en les appliquant à un cas concret. La réalisation du planning nous permettra de fixer des dates butoirs afin de réaliser le projet avec les contraintes temporelles imposées.