Année 2023-2024

Clara Rouxel
Léo Thomas
Léa Rissel
Robin Laumonier

Rapport Schotten Totten
Semestre 6

Sommaire

1. Présentation du projet

Au cours de notre première année dans le cursus informatique de Polytech Tours, nous avons réalisé un projet par équipe de 4 en C++. Notre projet était centré sur la création d'un jeu Schotten Totten. Il était constitué de deux parties : une partie réflexion avec modélisation d'un diagramme UML, suivie de la phase de codage.

Le principe global du jeu Schotten Totten Classique est le suivant :

Deux joueurs s'affrontent et posent tour à tour des cartes le long de neuf bornes. Chaque joueur pose une carte de sa main de son côté de la borne de manière à former la meilleure combinaison possible (que ce soit une suite, un brelan…). Celui ayant le plus grand score de son côté de la borne peut revendiquer la borne, c'est-à-dire qu'il gagne la borne. Le jeu se termine lorsqu'un joueur a remporté trois bornes adjacentes ou cinq bornes dispersées, ce qui lui permet de revendiquer la victoire.

Une variante Tactique existe aussi mais nous nous sommes concentrés sur la version Classique, c'est donc sur celle-ci que nous nous focaliserons.

De plus, dans notre version le joueur ne peut pas encore revendiquer une borne sans que de chaque côté, il y ait le nombre de cartes pour revendiquer la borne.

Nous vous présentons ici la façon dont nous avons implémenté notre projet et le bilan que nous en avons tiré.

2. Description de l'architecture

- a Généralité

Afin de permettre le plus petit couplage et plus grande évolutivité possible, nous avons choisi de mettre en place une architecture Modèle-Vue-Contrôleur. En effet, un changement de Vue (l'ajout d'une interface graphique plus développée par exemple) ne change pas le code des Contrôleurs et du modèle.

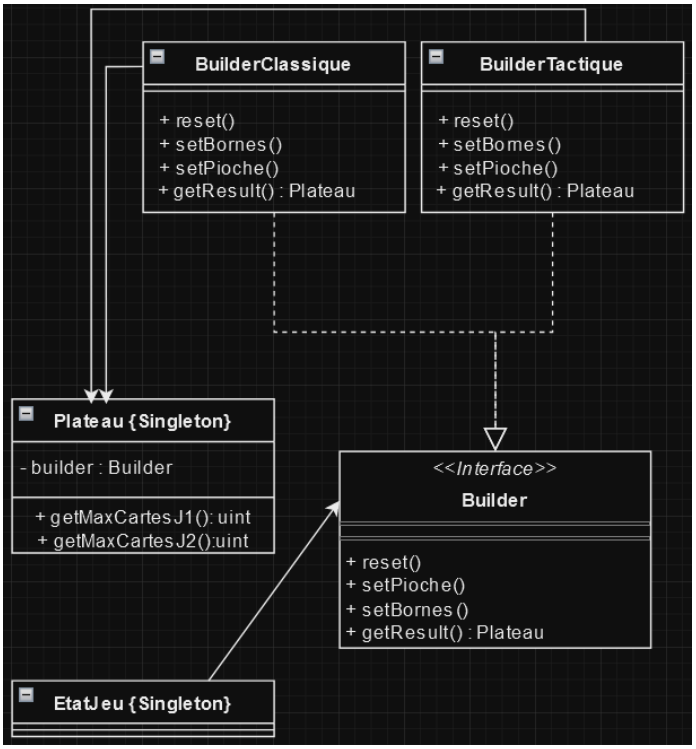
Le modèle stock l'état de la partie comme les bornes, les pioches ou les joueurs. Les vues, quant à elles, permettent l'affichage du jeu et les demandes de choix aux utilisateurs. Chacune d'elles est gérée par un contrôleur qui permet le lien entre le modèle et la vue, ainsi que la logique du jeu.

Nous avons défini trois couples vue-contrôleur : un pour le titre et le choix des variantes, un pour le choix des paramètres de joueurs, et un pour le jeu en lui-même avec deux variantes possibles. L'interaction entre ces couples est permise par la classe Application en changeant le contrôleur actuellement utilisé.

- b Modèle

Le point d'entrée du modèle est le singleton EtatJeu. Il permet d'obtenir les joueurs avec leurs mains et le plateau.

Ensuite nous avons le plateau builder. Il est constitué de bornes avec leurs cartes posées et de pioches. Le builder définit les méthodes nécessaires pour créer l'objet singleton Plateau. Ensuite BuilderClassique et BuilderTactique héritent de Builder et vont permettre de construire le plateau différemment selon le mode choisi (Classique ou Tactique). Dans le cas du modeTactique, la pioche ne sera pas initialisée de la même façon (il y aussi des cartes Tactiques). Utiliser un builder permet de simplifier sa création et d'éviter par exemple d'utiliser de nombreux paramètres utilisables seulement pour BuilderClassique ou BuilderTactique. Il sera ainsi aussi plus facile à l'avenir d'ajouter différentes configurations de plateau sans modifier tout le code.



Nous avons créé deux pioches : une pour les cartes clans et une pour les cartes tactiques. Ce sont eux qui créent le bon nombre et la bonne répartition des cartes.

Les cartes se divisent en deux parties : carte clan et carte tactique. Chaque carte tactique est ensuite répartie suivant sa catégorie (Troupe d'Elite, Ruse ou Mode de Combat) pour finalement retrouver les spécificités de la carte.

A noter que les cartes tactiques Troupe d'Élite forment un héritage multiple avec CarteTactique et CarteClan. En effet, elles se comportent comme telles : elles sont posées sur des bornes mais ne prennent de valeur et/ou de couleur que lors de la revendication de celles-ci.

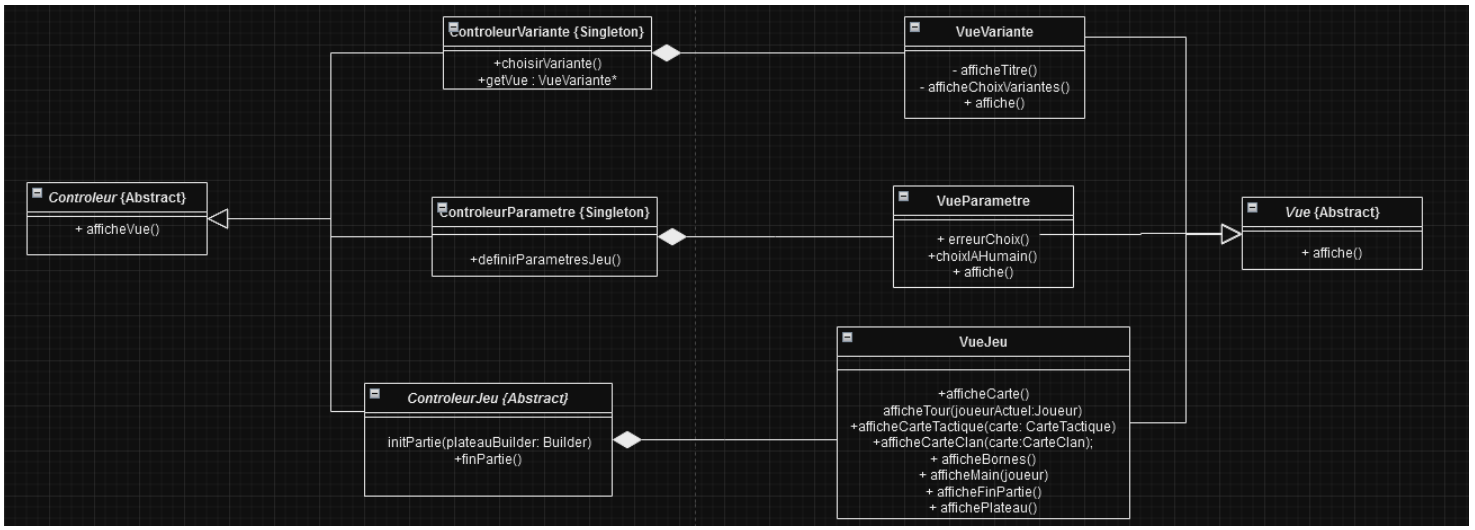
Chaque carte possède une méthode variadic *effet*. Elle est redéfinie dans les classes filles afin d'effectuer l'action voulu. Par exemple, la méthode effet de CarteClan va prendre en paramètre le numéro de borne où il faut poser la carte. Cette méthode est utilisée par les controleur fille de ControleurPersonnage lorsqu'une carte est jouée. Le design pattern Stratégie est utilisé pour appliquer l'effet voulu suivant la carte choisie.

Le même fonctionnement est appliqué pour la méthode *choixEffet* qui permet d'avoir les choix possibles (Sur quel bornes poser la carte) avec sa liste de valeur (de la borne 1 à 9).

- c Fonctionnement de l'interaction entre les contrôleurs

Tout d'abord, chaque contrôleur possède une vue qu'il peut afficher grâce à la méthode *afficheVue*. Par le design pattern Stratégie, la classe Application s'occupe du contrôleur actuel et, par conséquent, de son affichage. Ainsi, les contrôleurs peuvent demander un changement de contrôleur à l'Application.

Une classe ChoixUtilisateur a été créée pour augmenter l'encapsulation des choix utilisateurs c'est-à-dire du choix de la variante et du type de joueurs. Elle permet au ControleurVariante et au ControleurParamètre de stocker les choix au même endroit et au ControleurJeu de les récupérer facilement grâce à ses méthodes.



- d Couple Vue-Contrôleur Variante

La vue affiche le titre du jeu puis demande à l'utilisateur sur quelle variante il veut jouer. Elle doit aussi permettre de fermer l'application. Après le choix et le stockage de la variante choisie, un changement vers les paramètres est demandé à l'application.

- e Couple Vue-Contrôleur Paramètre

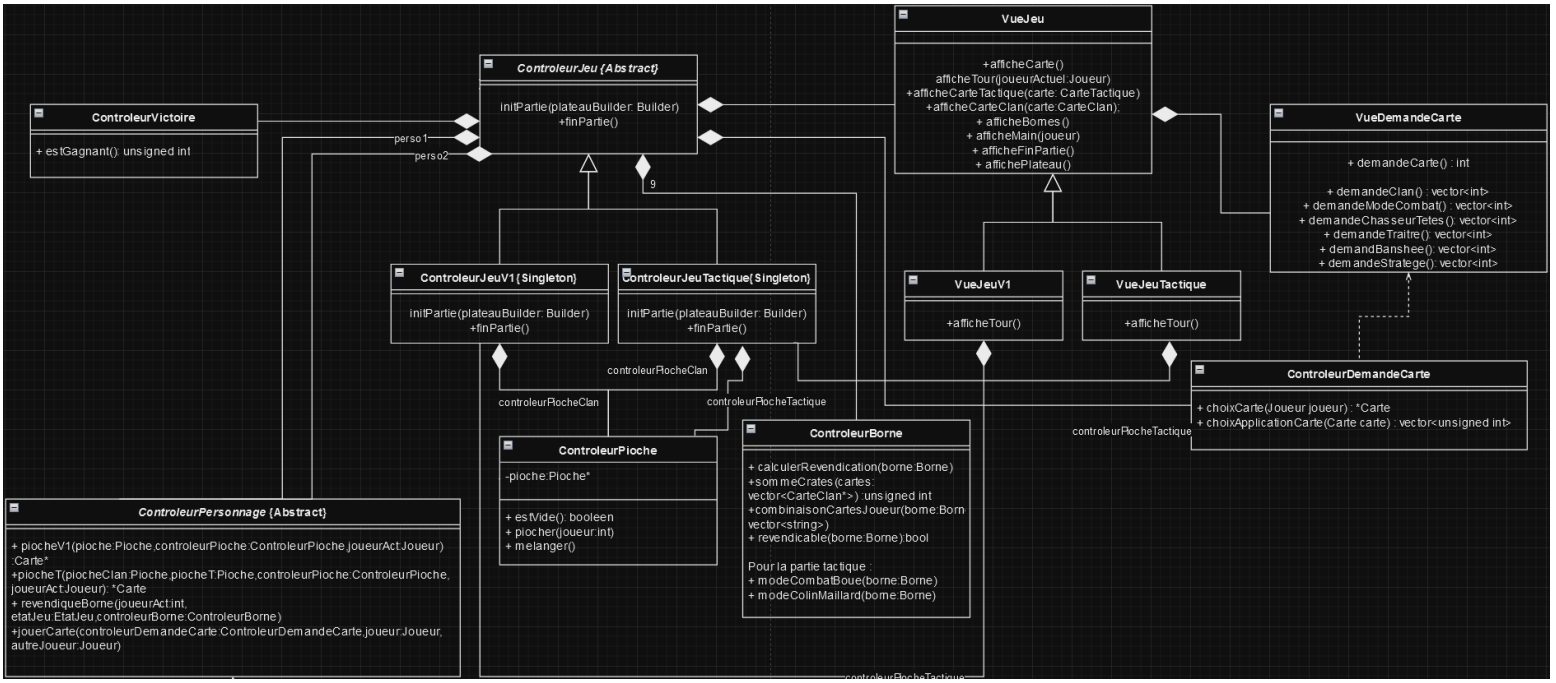
La vue demande si l'utilisateur veut jouer contre un joueur, jouer contre une IA ou s'il veut voir deux IA s'affronter. Le choix est ensuite stocké et l'application change le contrôleur vers la bonne variante de jeu grâce au choix sauvegardé.

- f Affichage du plateau

Le plateau est affiché par ControleurJeu et VueJeu. VueJeu contient toutes les méthodes pour afficher les bornes, les cartes qu'elles soient Clan ou Tactiques mais également le joueur actuel, sa main ainsi que le message de fin de partie.

- g Déroulement de la partie

Une fois la partie lancée, les différentes vues vont demander au joueur l'action à faire lors de son tour puis de choisir parmi les différentes options qui s'offrent à lui pour l'action en question.
Chaque élément du plateau est manipulé par un contrôleur qui est appelé si l'action le requiert.



- h Jouer une carte

Lorsqu'une carte est jouée, la procédure est la suivante : récupérer la carte à jouer, récupérer les choix pour jouer la carte, faire l'effet de la carte avec les choix effectués précédemment.
Cependant, il y a 2 joueurs possibles: l'Humain et l'IA.

L'humain, lui, a besoin d'une interface pour choisir. Il va appeler le ControleurDemandeCarte pour avoir les choix possibles.

L'IA, elle, n'a pas besoin d'interface. Elle va appeler Carte::choixEffet pour récupérer les choix possibles, puis choisir aléatoirement la valeur de ces choix. En effet, elle va choisir la borne sur laquelle elle va jouer de manière aléatoire.

Ensuite, les 2 vont appeler Carte::effet avec la liste de choix pris. Le problème est qu'il est difficile de passer de liste à variadic template. Ainsi, il y a une méthode effet qui permet d'envoyer à la bonne méthode effet
Dans jouerCarte, il relancera la méthode s'il y a une exception d'effet. Par exemple, s'il essaie de poser une carte sur une borne qui est déjà pleine.
Par la suite, la méthode effet va renvoyer une liste des informations à faire remonter. Pour une carte jouée sur une borne, par exemple, il va y avoir la borne sur laquelle la carte a été jouée.

- i Récapitulatif des design pattern utilisés

- Model–view–controller
- Singleton
- Builder
- Stratégie

- j Améliorations possibles

A terme nous pourrions rajouter une interface graphique. Il faudrait notamment adapter les vues. Une à rajouter pour les revendications ou une pour afficher les erreurs venant des effets. Une vue indiquerait par exemple que la borne est déjà remplie et que l'on ne peut pas poser de carte supplémentaire, ou jouer de cartes tactiques.

On pourrait aussi ajouter de nouveaux types de cartes (Il faudrait pour cela rajouter une méthode dans VueDemandeCarte et une condition dans ControleurDemandeCarte), de nouvelles IA (Choix Possible de coups à jouer proposés à l'IA peuvent être facilement automatisables grâce à la méthode choixEffet de Carte).

Nous pourrions aussi ajouter des variantes ou changer les phases de jeu : en ajoutant une fille au ControleurJeu, une entrée dans le choix des variantes, voire un builder pour le plateau ou rajouter Pioche

En outre, il faudrait rajouter la possibilité de revendiquer une borne lorsqu'on est sûre de la gagner.

Enfin, il faudrait bien évidemment finir la phase de débogage sur ce projet. Par manque de temps, notre projet contient assez d'erreurs.

3. Contributions personnelles

Concernant la contribution de chacun des membres, nous avons estimé que tout le monde a participé plus ou moins équitablement ce qui donne la répartition suivante :
25% Clara Rouxel
25% Léo Thomas
25% Léa Rissel
25% Robin Laumonier

4. Bilan du Projet

Nous aurions aimé avoir plus de temps pour avancer sur le projet (stage/travail pour certains membres de l'équipe en plus d'autres projets parallèles). Nous avons réussi à implémenter la base du code et nous aurions aimé pouvoir le rendre entièrement fonctionnel sans bugs et jouable. Malgré le manque de temps, nous avons trouvé le projet stimulant et avons acquis une bonne expérience concernant le C++ mais également le travail en équipe.