UNIVERSITÉ DU QUÉBEC À MONTRÉAL

TRAVAIL PRATIQUE #3

TRAVAIL PRÉSENTÉ À
JACQUES BERGER

DANS LE CADRE DU COURS
GÉNIE LOGICIEL : CONCEPTION
INF5153
GROUPE 30

PAR

ALEXIS VALOIS-ADAMOWICZ
VALA10049105

VAITEA DOPPIA
DOPV30059205

FRANÇOIS PLANET
PLAF17069100

JEUDI 28 AVRIL 2016

Table des matières

Patron GoF #1 : Strategy

Nous avons choisi d'implémenter le patron GoF Strategy pour la gestion de l'algorithme d'attaque à utiliser lors du lancer de torpilles depuis un joueur artificiel. On utilise une interface standard pour effectuer un lancer et la Partie manipule cette interface. L'implémentation de la stratégie d'attaque peut changer en cours d'exécution du logiciel, modifiant ainsi dynamiquement la façon dont les cours du joueur artificiel sont effectués.

Diagramme de classe

Le diagramme montre qu'il existe 2 algorithmes d'attaques différentes dans notre logiciel : au hasard ou bien de façon un peu plus « intelligente » (Minimax). La classe utilisatrice est Partie. Elle n'a pas connaissance de la Stratégie concrète lorsqu'elle simule un lancer.

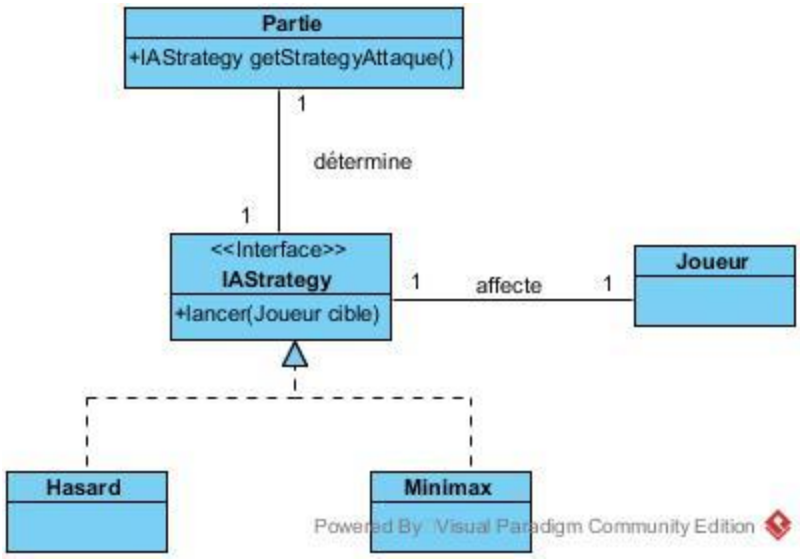
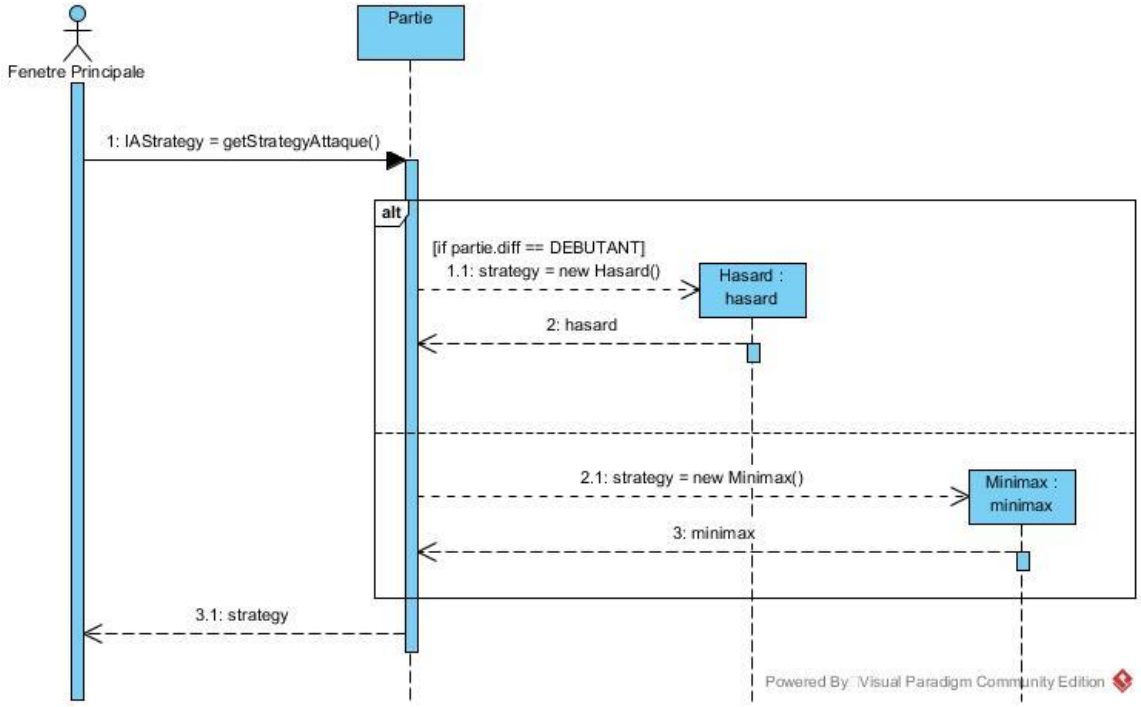


Diagramme de séquence

On voit ici que le cas d'utilisation d'un lancer de torpille depuis un joueur artificiel est déclenché par un événement de la fenêtre principale.



Patron GoF #2 : Facade

Nous avons choisi d'implémenter le patron GoF Facade dans plusieurs cas. Celui que nous détaillerons ici est le cas de la persistance d'un tour. EN effet, pour enregistrer un tour en mémoire, plusieurs manipulations, sur plusieurs classes différentes, doivent avoir lieu. Plustôt que de laisser cette responsabilité à la classe utilisatrice (JoueurHumainController), nous avons décidé de déplacer ce traitement complexe dans un Facade que nous avons nommé « TourService ». Cette dernière est responsable de manipuler les classes nécessaires pour la persistance d'un tour. La classe utilisatrice n'a qu'à utiliser l'interface publique de TourService pour que les opérations complexes s'exécutent.

Diagramme de classe

La classe Utilisatrice « JoueurHumainController » appelle la méthode « ajouterTour » de la façade « TourService ». Cette dernière manipule différentes classes pour parvenir à enregistrer un Tour dans la Partie en cours. Le Tour inclut l'état des grilles des 2 joueurs qui s'affrontent, pour le tour qui vient d'avoir lieu.

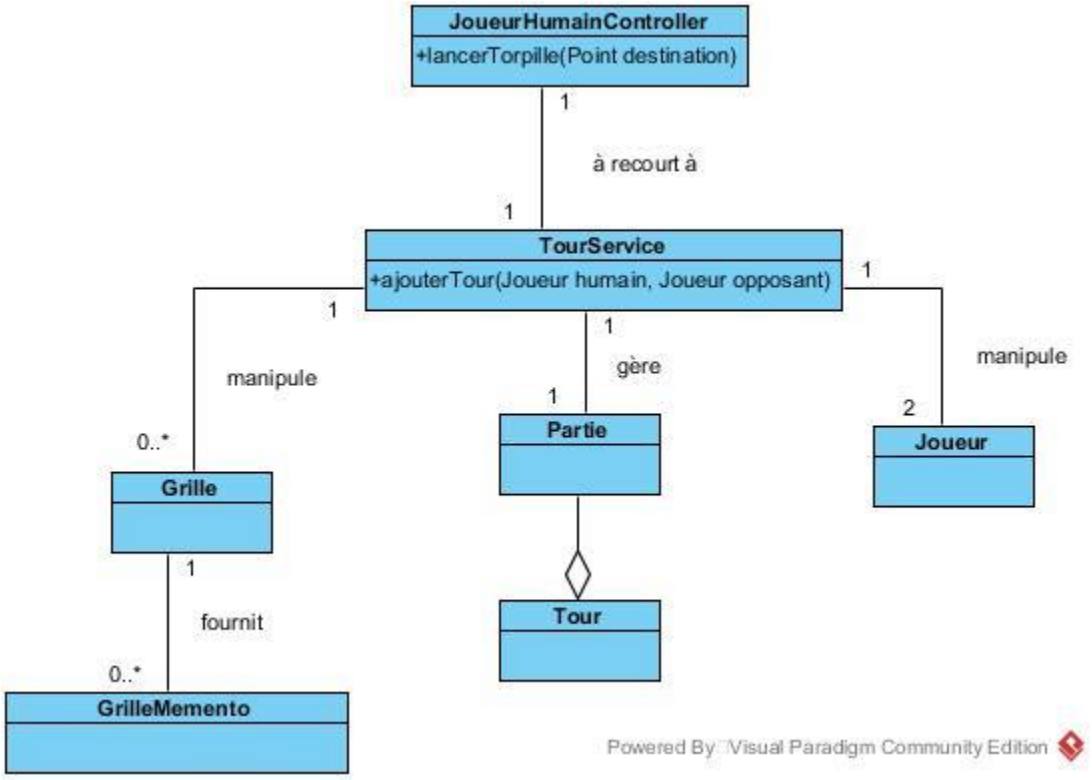
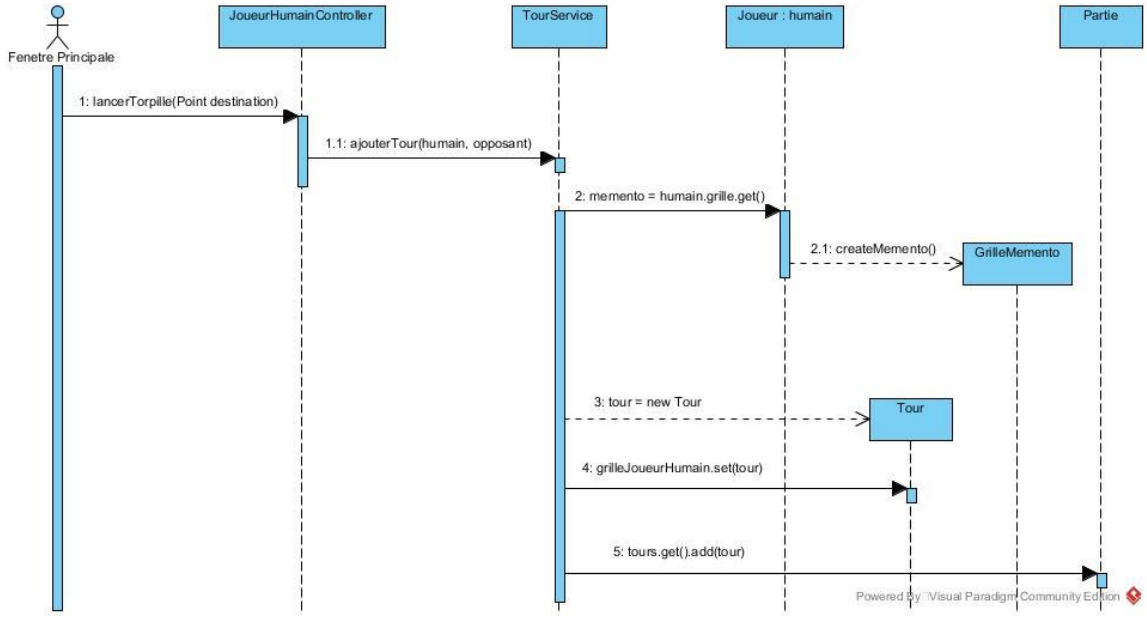


Diagramme de séquence

La Facade « TourService » doit créer un Tour et lui affecter un GrilleMemento (un par joueurs qui s'affrontent). Le TourService doit obtenir les 2 GrilleMemento en les demandant directement aux objets Grille de chaque joueur. Une fois le tour créé et les GrilleMemento correctement obtenus et renseignés, le TourService ajoute le nouveau tour dans la Partie en cours.



Patron GoF #2 : Memento

Nous avons choisi d'implémenter le patron GoF Memento pour la gestion des différentes versions des grilles des joueurs. L'idée est de pouvoir conserver une sauvegarde de l'état des grilles après chaque tour de manière à pouvoir y faire référence par la suite (pour la fonctionnalité de rejouer un tour).

Diagramme de classe

Chaque grille possède un état (GrilleState). C'est à l'intérieur de cet état que l'on retrouve la grille de jeu d'un joueur, c'est-à-dire, une matrice de 10 par 10 cases qui maintient un ensemble de lancée de torpilles. C'est la Grille qui est responsable de fournir son GrilleMemento. Pour ce faire, la Grille demande à son GrilleState de se cloner. Le GrilleState clone est utilisé par le constructeur du GrilleMemento.

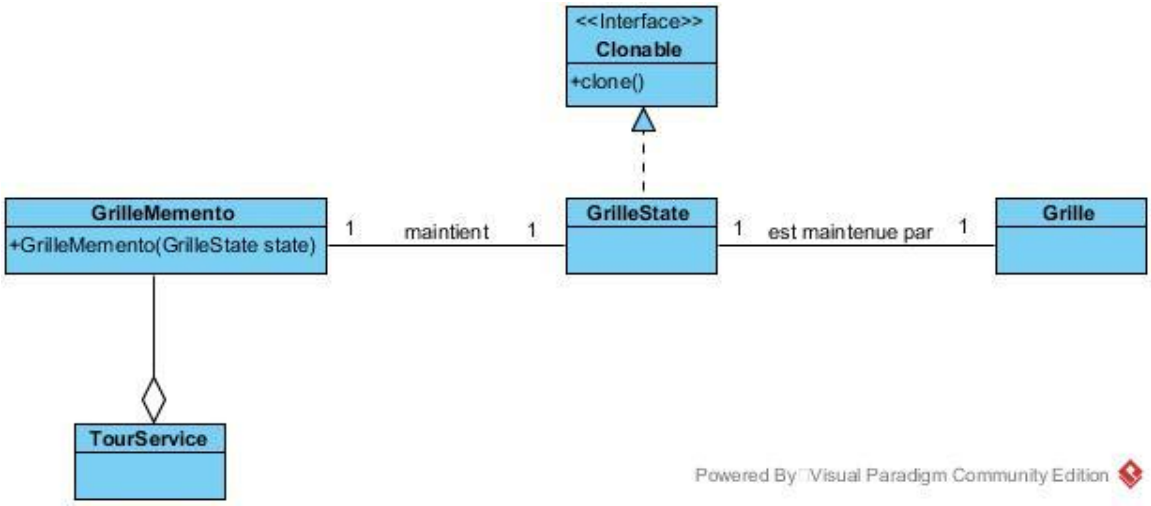
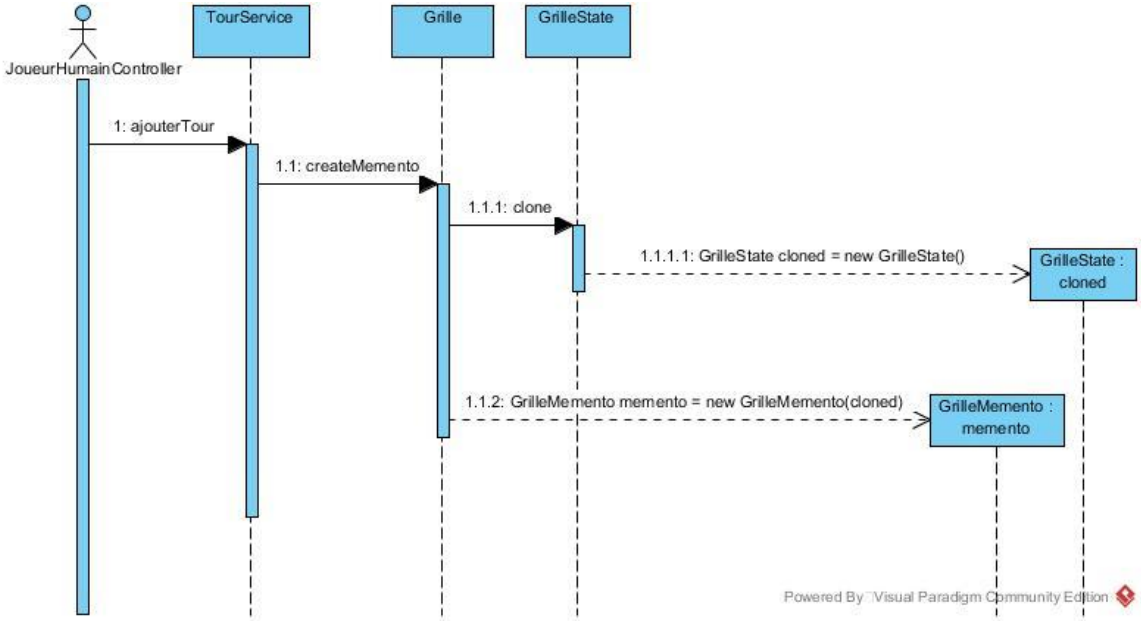


Diagramme de séquence



Modifications apportées à la conception originale

Voici une liste des modifications apportées à la conception originale ainsi que la façon dont nous sommes parvenus à la conclusion qu'un changement devoir avoir lieu :

Classe/Paquetage	Changements apportés	Raisons
ClientApplication	• Ajout de la méthode « start » à l'interface publique de la classe • Ajout de l'extension de la classe « Application » du package JavaFX • Ajouts de certaines méthodes privées pour la configuration de certains éléments propres à JavaFX • La classe à été renommée pour « MainApp »	L'utilisation de JavaFX nous a forcés à apporter ces changements. Lors de la conception initiale, nous n'étions pas au courant de ces nécessités propres à JavaFX.
Paquetage « action » • Classe ApplicationCloseAction • Classe CurrentWindowCloseAction • Classe DialogShowAction • Classe VisualiserPartieAction	• Ajout du paquetage action • création des classes d'actions le composant.	La nécessité d'ajouter des classes d'action provient essentiellement de la façon dont les événements graphiques sont gérés avec JavaFX. Afin de pouvoir gérer efficacement les événements les plus génériques de l'application, nous avons opté créer des classes séparées et rendre leur action générique plutôt que de créer des classes internes dans le contrôleur principal.
Énumération « EtatPartie »	• Création de l'énumération EtatPartie	Nous avons ressenti le besoin de créer une énumération pour l'état d'une partie lorsque nous nous sommes aperçues que nous aurions besoin d'avoir un état « en cours » pour une partie, notamment pour spécifier

		si la partie est prête pour être visualisée ou non.
Énumération JoueursLocation	• Création de l'énumération JoueursLocation	Comme nous avons décidé de placer les 2 opposants dans un tableau traditionnel (au sein d'une Partie), nous avions besoin d'un standard pour déterminer quel indice du tableau correspondait au joueur humain et l'opposant. Pour ne pas avoir à se rappeler de l'indice en tant que tel, nous avons opté utiliser un énumération.
Énumération Reponse	• Renommage pour « StatusDeCase » • Retrait de la valeur « tout_coule » • Ajout des valeurs « PLACEE et VIDE »	Le changement de nom provient du fait que l'énumération ne correspond pas réellement à une réponse, mais davantage à l'état d'une case d'une grille. La valeur « tout_coule » s'est avérée inutile. Par contre, afin de distinguer graphiquement une case placée et vide, nous avons du ajouter ces 2 valeurs à l'énumération.
Paquetage gamecontroller et jfxcontroller	• Renommage du paquetage controller vers gamecontroller • Ajout du paquetage jfxcontroller	La notion de contrôleur est reprise par les standards JavaFX. Afin de distinguer les contrôleurs propres à notre logiciel et ceux qui gèrent une vue FXML, nous avons créé 2 paquetages différents, car les classes qui les composent ont des responsabilités différentes. Les gamecontrollers agissent en tant qu'intermédiaire entre la couche présentation et les couches inférieures du logiciel. Les jfxcontroller ne sont en fait qu'une représentation Java des fenêtres définie à l'aide de fichiers FXML.

IAStrategy	• Ajout du paramètre Joueur à la méthode publique lancer	Nous nous sommes rendu compte que les différentes stratégies d'attaque allaient forcément avoir à affecter la grille du joueur attaqué. Nous avons donc opté pour l'injection de dépendance en ajoutant le paramètre Joueur à la méthode lancer, permettant ainsi à une stratégie d'affecter directement la grille du joueur.
Navire	• Retrait de la méthode handleLance • Retrait de l'attribut int nbCases • Ajout de l'attribut cases (ArrayList) • Ajout méthode abstraite getNbCases	La responsabilité de recevoir un lancer a été déplacée vers la classe Joueur directement. Un Navire n'a pas les informations requises pour déterminer le StatusDeCase d'un lancé (il lui manque la grille). Également, nous n'avons pas besoin d'un entier qui spécifie le nombre de cases à ce niveau. Par contre, un ArrayList qui conserve l'état des cases est plus pertinent. Pour l'information sur le nombre de cases, nous avons opté pour transmettre cette responsabilité aux sous-classes en création la méthode abstraite getNbCases.
Torpilleur / ContreTorpilleur / PorteAvion / Croiseur / SousMatin	• Retrait de l'attribut cases • implémentation de la méthode getNbCases	Voir item précédent
	•	