Bordeaux INP
ENSC

RE: SHOWER

un remake en C# du jeu Rush Hour

Auteurs
Guillaume Creusot
Louis Hache
Alexandra Pometko

Bordeaux INP
ENSC

Table des matières

I - Introduction

L'objectif de ce projet était de réaliser une application console en langage C# pour recréer le fameux jeux Rush Hour.

C'était pour nous l'occasion de mettre en pratique nos connaissances en Programmation Orientée Objet. De plus, afin de gagner en connaissances, nous avons décidé de suivre le pattern "Modèle Vue Contrôleur" lors du développement.

II - Le design pattern MVC

Le “Modèle Vue Contrôleur” est un motif d’architecture logicielle destiné aux applications avec une interface graphique. Il est décomposé en 3 modules ayant chacun un rôle particulier :

- Le **Modèle** contient les données à afficher .
- La **Vue** contient la présentation de l'interface graphique.
- Le **Contrôleur** contient la logique concernant les actions effectuées par l'utilisateur (saisie utilisateur.

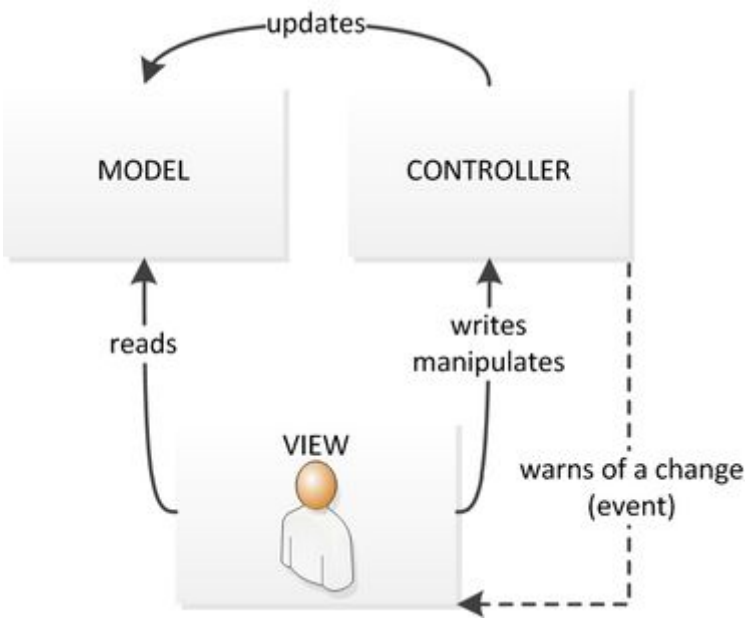


Fig 1 : Le motif MVC

Les avantages de ce motif d’architecture sont les suivants :

- **La maintenance et la mise à jour** de l’application est simplifiée : s’il y a un détail à changer dans la vue par exemple, cela n’affecte en rien la manière dont sont fait les calculs ou le modèle global de l’objet.
- **La répartition des tâches** entre les différents développeurs est plus aisée, car ils peuvent coder sur le même objet simultanément. Par exemple pour un véhicule, un des développeurs pourra s’occuper de l’affichage et l’autre des déplacements.

Au cours de notre développement, nous avons constaté que :

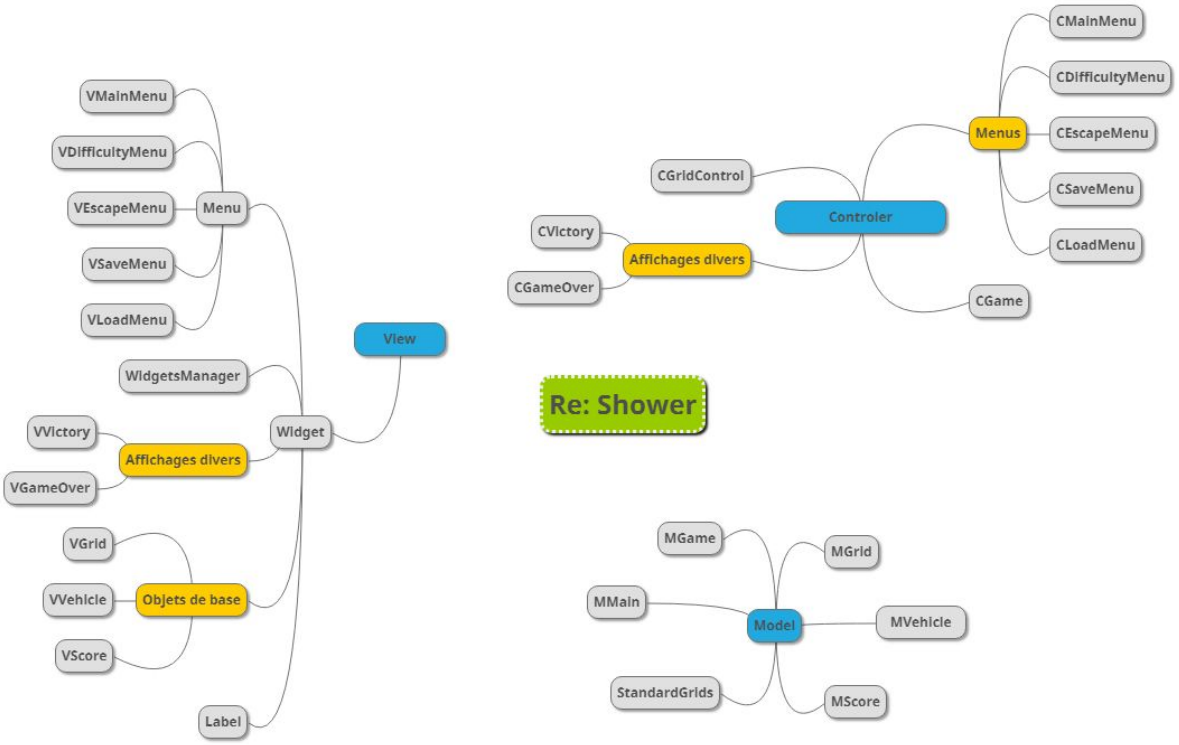
- Ce motif nécessite parfois une manière de réfléchir particulière et inhabituelle pour parvenir à séparer le modèle de la vue ou du contrôleur.
- Ce motif multiplie le nombre de classes nécessaires.

Sources :

1. *https://openclassrooms.com/courses/apprendre-asp-net-mvc/le-pattern-mvc*
2. *https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur*

III - Notre implémentation

III.1 - Architecture globale



Les éléments en gris représentent une classe.

Fig 2 : architecture générale du projet Re: Shower

III.1.a - Le modèle

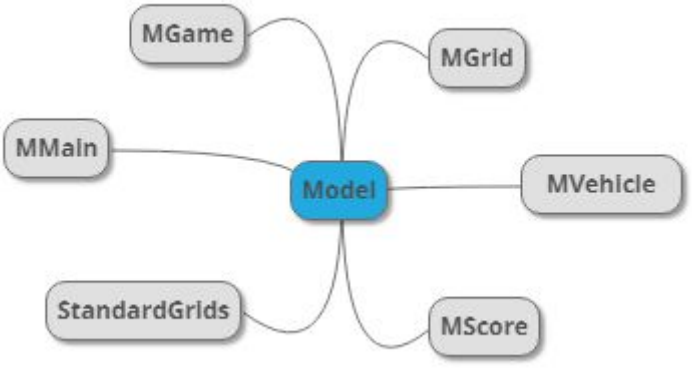


Fig 3 : Organisation du modèle

Le modèle contient les informations essentielles pour chaque objet clé du jeu. Par exemple :

- La classe MMain contient les types énumérables Direction et Difficulty qui seront utilisés dans l'ensemble du code.
- Pour le véhicule, la longueur, la direction et le fait qu'il soit un joueur ou non, sont des données primordiales.
- MGrid contient la liste de tous les véhicules présents sur la grille.

Ce sont ces données qui vont participer à construire la vue.

III.1.b - La vue

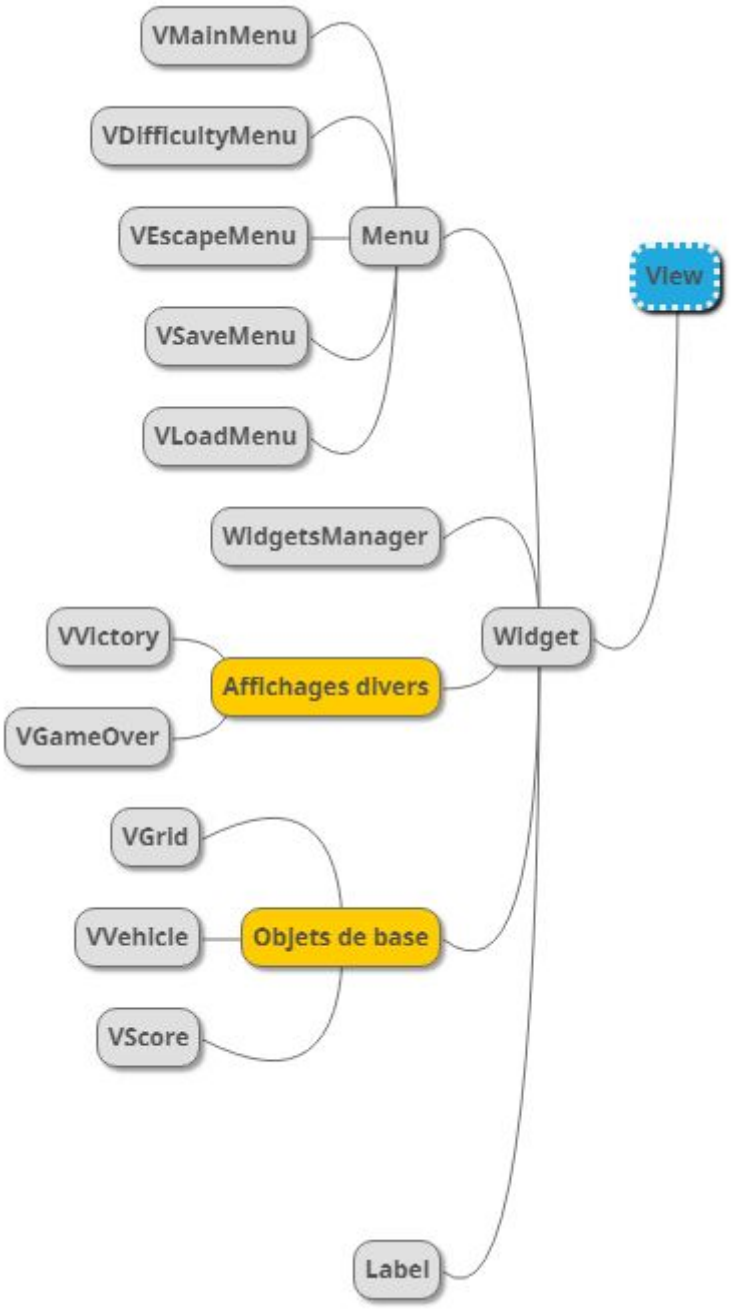


Fig 4 : Organisation de la vue

Pour la Vue, nous nous sommes inspirés des bibliothèques graphiques comme tkinter (en Python) pour développer des outils qui nous permettent de gérer l'affichage de manière plus simple.

Ainsi, nous avons conçu une classe Widget qui est l'objet de base pour concevoir l'ensemble des éléments graphiques. La classe Widget possède pour principal champ le "content", une chaîne de caractère représentant la manière dont le widget doit être affiché dans la console.

Pour afficher ces "Widgets", nous avons réalisé une classe WidgetManager qui gère l'affichage des widgets, il est cependant également un widget. Avec ce WidgetManager, on peut mettre un ou plusieurs widget sans altérer les autres. De plus on peut les placer comme on le souhaite.

Ainsi, on peut concevoir et tester chaque widget indépendamment des autres, et on peut également utiliser le même widget sur plusieurs vues (sans qu'il soit nécessairement situé au même endroit sur l'écran).

Fig 5 : Écran de jeu avec l'affichage des différents widget et widget manager

On peut voir sur la figure 5 tous les widgets et WidgetManagers utilisés pour notre écran de jeu. La majorité du texte (titre, légende, …) affiché ici utilise un type de Widget nommé "Label" qui permet d'afficher du texte et de le mettre à jour. Les WidgetManagers sont utilisés pour l'affichage du score (widget manager avec trois label à l'intérieur) et pour l'affichage de la grille de jeu (widget manager contenant les widgets des véhicules).

Cet outil est loin d'avoir toutes les fonctionnalités de librairies comme tkinter, mais nous a permis d'avancer rapidement (une fois conçu) dans la création des vues et pourrait nous servir sur d'autres projets sur console.

III.1.c - Le contrôleur

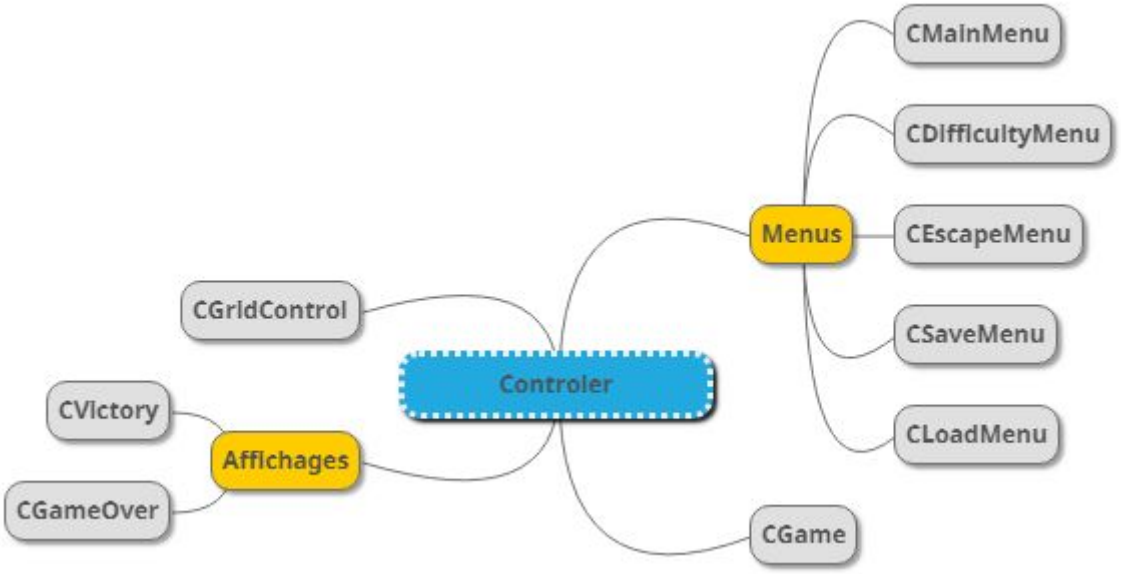


Fig 6 : Organisation du contrôleur

Le contrôleur principal s'appelle CGame : il lie tous les contrôleurs. Le déroulement du jeu est géré par CGridControl qui s'occupe de la sélection et du déplacement des véhicules. Les contrôleurs relatifs au menu s'occupent de la navigation dans les menus.

Ces classes permettent de gérer les saisies utilisateur lors de la navigation dans les menus et des déplacements des voitures avec les touches directionnelles.

III.2 - Fonctionnalités intéressantes

III.2.a - Types énumérés

Nous avons inclu 2 types de valeurs énumérées afin de faciliter la lisibilité de nos opérations:

- **Direction**, qui définit l'orientation du véhicule : North, West, East, South.
- **Difficulty**, qui définit le niveau de difficulté de la partie : Easy, Medium, Hard.

III.2.b - Ergonomie et navigation

Nous avons accordé une attention particulière à l'ergonomie de notre application console.

En effet, pour plus d'intuitivité et une meilleure expérience utilisateur,

- une légende indique à l'utilisateur les instructions et la signification des différents objets.
- les véhicules sont affichés en couleurs différentes.
- le véhicule sélectionné se distingue des autres par son apparence.
- le véhicule sélectionné ET en déplacement se distingue du véhicule simplement sélectionné.
- l'utilisateur sélectionne le véhicule qu'il souhaite déplacer grâce aux flèches directionnelles et en appuyant sur la touche entrée.
- la navigation dans les menus se fait grâce aux touches *haut, bas* et *entrée*.
- le menu Pause est accessible à tout moment, même quand un véhicule est en cours de déplacement.
- une mauvaise saisie utilisateur (caractères spéciaux lors de la sauvegarde) ne fait pas planter l'application.
- les affichages sont adaptés à la taille de l'écran de l'utilisateur.
- les affichages sont grands et lisibles.

IV - Pistes d'évolution

Re: Shower a encore quelques possibilités d'évolution. En effet, les points suivants pourraient être mis en place :

- la génération automatique de grilles de difficultés variables.
- un affichage de conseils sur le meilleur chemin à suivre, calculé grâce à une IA qui pourrait compter le nombre de coups minimum, le coup à faire pour débloquer la situation...
- une interface graphique plus poussée, qui pourrait éventuellement être réalisée avec Winforms ou WPF, ce qui pourrait permettre de contrôler les véhicules grâce à la souris pour un gain d'intuitivité.
- la possibilité de modifier l'apparence des véhicules, avec des skins contenus dans des lootbox à remporter (en respectant bien évidemment la législation sur les lootbox).
- Notre application étant en partie responsive, il est théoriquement possible d'ajouter dans les options la taille des cases de la grille.
- Il serait également possible de créer des grilles avec un nombre de cases plus important, ainsi que des véhicules à la géométrie plus complexe.

V - Gestion de projet

V.1 - La méthode Scrum

Nous avons appliqué une méthode Agile pour ce projet : la méthode Scrum.

V.1.a - Gestion des tâches

Ainsi, nous avons utilisé la fonctionnalité “Projects” de GitHub afin de mettre en place notre outil de gestion des tâches.

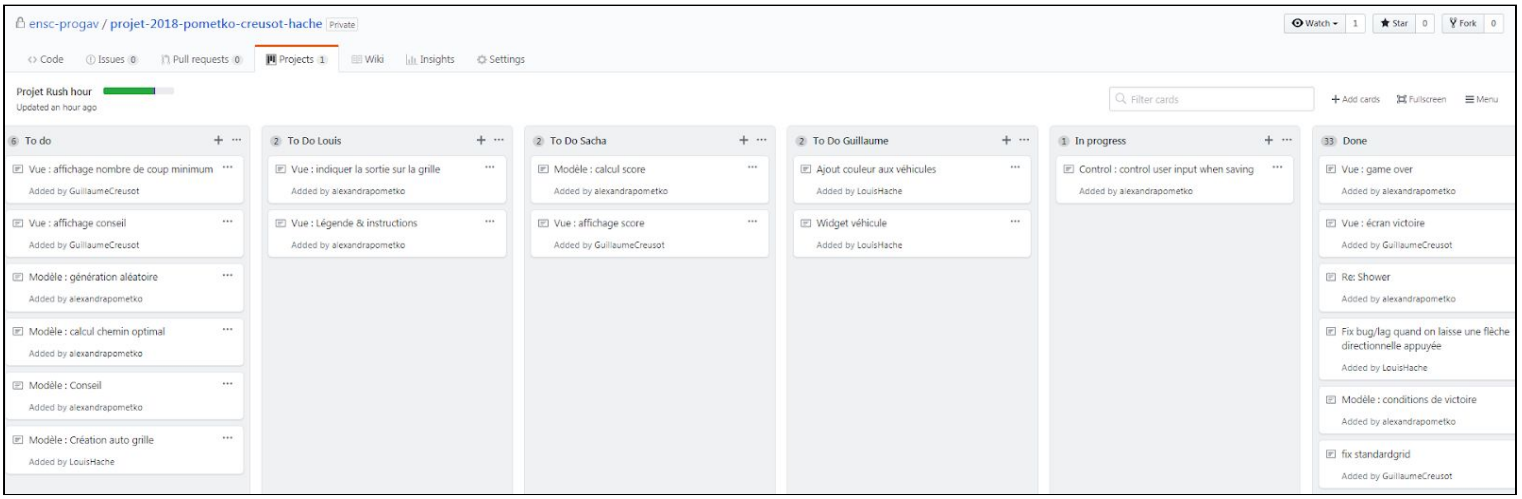


Fig 7 : Tableaux de gestion des tâches

Nous avons défini les 6 colonnes suivantes afin de répartir et déplacer nos cartes de tâches:

- To Do: qui représentait le **Backlog du projet**
- To Do Alexandra : qui permettait d’affecter les tâches que devait réaliser Alexandra pour le sprint en cours : le **Sprint Backlog d’Alexandra**
- To Do Louis : qui permettait d’affecter les tâches que devait réaliser Louis pour le sprint en cours : le **Sprint Backlog de Louis**
- To Do Guillaume: qui permettait d’affecter les tâches que devait réaliser Guillaume pour le sprint en cours : le **Sprint Backlog de Guillaume**
- In Progress : qui contenait les tâches en cours de réalisation par l’ensemble de l’équipe
- Done : qui contenait les tâches achevées

Au cours du projet, nous nous sommes rendus compte que certaines tâches supplémentaires étaient nécessaires, elles ont donc été ajoutées au Backlog à posteriori, démontrant ainsi la **flexibilité** de cette méthode Agile.

V.1.b - Sprints et revue de sprint

Nous avons mis en place trois sprints d'une longueur d'une semaine :

- Un **sprint initial** afin de mettre en place de manière fonctionnelle et robuste les objets essentiels : grilles, véhicules, Widgets pour l'affichage.
- Un **sprint intermédiaire** pour aboutir à une version basique fonctionnelle du jeu (déplacements, scores, affichage des menus…).
- Un **sprint final** pour optimiser l'affichage, implémenter des fonctionnalités supplémentaires telles que la sauvegarde.

A la fin de chaque sprint, une revue a été réalisée afin de vérifier que les objectifs du sprint ont été atteints et que le code était fonctionnel et conforme au modèle MVC.

V.1.c - Communication

Une communication brève (environ 10 minutes par jour) mais quotidienne a été mise en place pour faire le point sur les avancées de la veille et les problèmes rencontrés.

Ceci a garanti une avancée régulière et constante sur le projet (fig 8) et un investissement de la part des trois développeurs (fig 9), comme en témoigne la partie Insights de GitHub sur l'ensemble de la durée de projet.

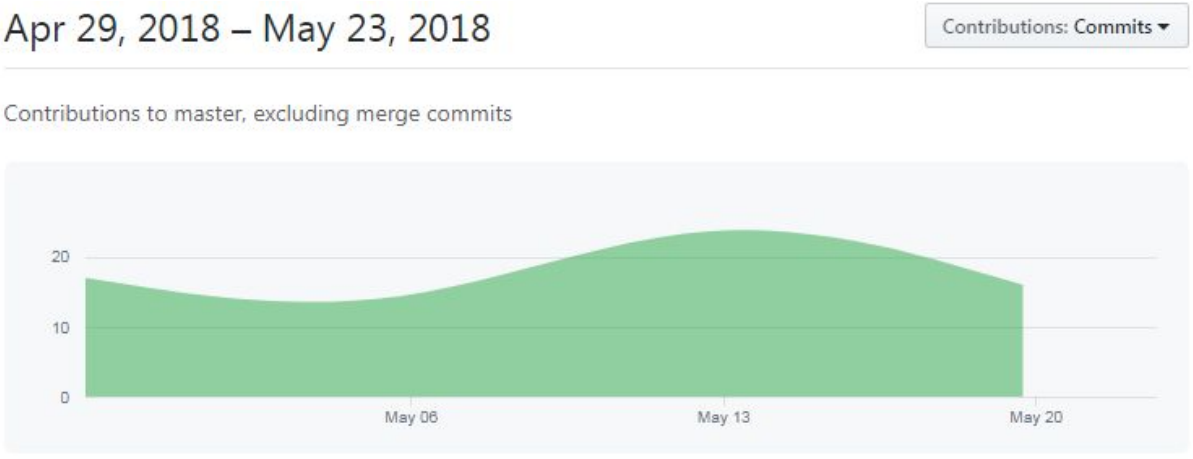


Fig 8 : Contributions au projet sur l'ensemble de la durée du projet, illustrées par le nombre de commits

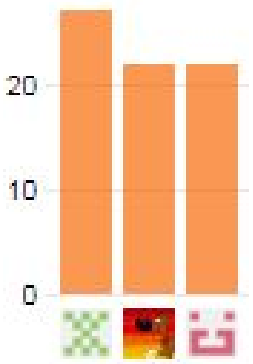


Fig 9 : Nombre de commits par développeur

V.2 - Tests

Tout au long du projet, chaque nouvelle classe a été testée de façon manuelle afin de vérifier le bon fonctionnement de ses méthodes.

De plus, des **tests intégrateurs** à chaque étape clé ont été réalisés pour vérifier que les différents objets interagissent bien entre eux et fonctionnaient correctement.

Enfin, des **tests de robustesse** ont été réalisés pour mettre à l'épreuve le jeu et le rendre plus solide. Ainsi, nous avons cherché à déclencher des erreurs et des bugs en effectuant des saisies incorrectes ou bien des manipulations peu ordinaires (charger une partie déjà existante, la sauvegarder à nouveau, lancer une nouvelle partie directement…).

VI - Conclusion

Ce projet fut riche en gain d'expérience. En effet, nous avons pu mettre en pratique nos connaissances théoriques sur la gestion de projet Scrum, et nous avons découvert le développement suivant le pattern MVC.

Ceci nous a permis de nous rendre compte que ce pattern de design n'était peut-être pas adapté pour une application simple comme le Rush Hour car elle a nécessité la création d'un grand nombre de classes. De plus, son avantage d'avoir des opérations de maintenance et de mises à jour n'allait pas être exploité.
Cependant, il a permis une grande flexibilité dans la répartition des tâches et nous a fait comprendre les grands principes d'un développement en 3 modules.

Des transferts de connaissances et de savoir-faire ont eu lieu entre les différents membres, notamment au niveau de la mise en place des widgets et des techniques d'affichage.