Ecole Publique d'Ingénieurs en 3 ans

Rapport

VIS MA VIE D'ENSICAENNAIS

le 16 octobre 2023,

Calliste RAVIX,
calliste.ravix@ecole.ensicaen.fr

Blaise HECQUET,
blaise.hecquet@ecole.ensicaen.fr

Thibaut LETOURNEUR
thibaut.letourneur@ecole.ensicaen.fr

Maxime MICHEL
maxime.michel@ecole.ensicaen.fr

Clément JANTET,
clement.jantet@ecole.ensicaen.fr

William SAMPAIO-FERREIRA
william.sampaio-ferreira@ecole.ensicaen.fr

Eliott DIDELOT
eliott.didelot@ecole.ensicaen.fr

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

www.ensicaen.fr

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

TABLE DES MATIERES

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

1. INTRODUCTION

Ce projet consiste en la réalisation d'un jeu de l'oie adapté à la vie d'un étudiant à l'ENSICAEN. L'objectif fondamental de ce projet n'est pas de réaliser un jeu complet mais plutôt d'avoir la bonne méthode de travail en utilisant une méthode de développement agile.

Ce jeu reprend donc les codes classiques d'un jeu de l'oie que nous avons légèrement personnalisé pour l'adapter à la vie d'un étudiant à l'ENSICAEN.

2. DESCRIPTION DU JEU

Comme expliqué dans l'introduction, le jeu que nous avons développé est un jeu de l'oie qui se base sur les étapes principales de la vie d'un étudiant à l'ENSICAEN. Ce jeu se veut caricatural et non parfaitement réaliste.

Le plateau de jeu est donc composé de 65 cases inédites avec des effets particuliers. Les joueurs évoluent sur le plateau en lançant un dé de 6 faces. Le plateau est séparé en 3 parties représentant les 3 années nécessaires à l'obtention du diplôme d'ingénieur. Chacune de ces parties se termine par un examen. Tout au long de l'année en cours, le joueur va voir son niveau de compétence être influencé en fonction des cases sur lesquelles il tombe. A chaque fin d'année, le joueur s'arrête nécessairement sur la case examen. Sur cette case, le niveau de compétence du joueur est comparé au niveau de compétence nécessaire pour passer l'année. Si le niveau de compétence du joueur est supérieur au niveau requis, le joueur se rend sur la case d'après (case rentrée de l'année suivante). Sinon, il retourne sur la case rentrée de l'année

qu'il a échouée. Si un joueur tombe sur une case déjà occupée par un autre joueur, leur position sont inversées (le joueur situé sur la case d'arrivée se téléporte sur la case initiale du joueur qui joue). Le jeu se termine lorsque tous les joueurs ont réussi l'examen de troisième année.

Au début du jeu, chaque joueur peut personnaliser son pion en choisissant un pseudo ainsi qu'une couleur. Il va également pouvoir choisir sa filière (Informatique, Electronique ou Matériaux-Chimie) et son cursus pré-ENSICAEN (Classe Préparatoire ou AST). Ces deux choix vont avoir une influence sur la suite du jeu et l'effet des cases. Au début de la partie, le joueur reçoit également une compétence personnelle attribuée aléatoirement (Dilettante, Assidu ou Brillant). En fonction de sa compétence personnelle, le résultat du dé lancé par le joueur va être multiplié par un facteur (0.5 si Dilettante, 1 si Assidu, 2 si Brillant).

Chaque case du plateau possède un effet qui lui est propre :

- Les cases de cours. Ces cases permettent à l'étudiant d'augmenter son niveau de compétence. Certains cours communs aux 3 filières (DDRS, Anglais, LV2…) augmentent le niveau de compétence du joueur peu importe sa filière. D'autres cours (Cryptographie, Capteurs, Chimie organique…) n'augmentent le niveau du joueur que si le joueur appartient à la bonne filière. Enfin, certains cours ont un effet différent en fonction du type d'étude précédent de l'étudiant (les cases mathématiques font perdre 1 point de compétence aux étudiants AST alors que les cases C++, TP de Chimie et TP de carte à puce font perdre 1 point de compétence aux étudiants venant de classe préparatoire).
- Les cases *Weekend d'intégration*. Lorsqu'un joueur tombe sur cette case, une nouvelle compétence personnelle lui est attribuée aléatoirement, remplaçant celle qu'il a actuellement.

- Les cases *révision*. Lorsqu'un joueur tombe sur une case révision, son niveau de compétence augmente de 2.

- Les cases *soirée*. Lorsqu'un joueur tombe sur une case soirée, son niveau de compétence baisse de 1.

- La case *burnout*. Lorsqu'un joueur tombe sur la case burnout, il retourne au début du plateau.

- Les cases *rentrée* et *examen*.

- Les cases *vacances*. Ces cases sont sans effets.

3. TRAVAIL EN EQUIPE

Dans un projet où autant de membres sont impliqués, l'un des éléments le plus important est le travail en équipe. En effet, afin d'être efficace et de rendre un code propre et fonctionnel, il faut que chaque membre ait un rôle pour qu'il sache quoi faire et quand. Ainsi, plusieurs rôles ont été attribués aux membres. Maxime était le chef d'équipe : son rôle était donc d'organiser le travail mis en place et de faire le lien entre le client et l'équipe pour répondre au mieux à ses attentes. Eliott était lui le responsable de version. En effet, nous avons utilisé pour réaliser ce travail le logiciel Gitlab afin de pouvoir coder tous en même temps et faciliter le partage des lignes de code. Ce logiciel permet à chacun de travailler sur une « branche » à l'aide d'une copie du projet à un instant donné. Une fois le travail sur une branche terminé, on peut la fusionner (*merge*) avec la branche principale (*master*) représentant le véritable projet, cette fusion étant gérée par le logiciel. Eliott s'occupait donc de vérifier que chacun travaillait

correctement sur des branches différentes et était le seul à pouvoir réaliser la fusion de n'importe quelle branche secondaire avec la branche principale. Les développeurs avaient néanmoins le droit de demander une fusion (*merge request)* que le responsable de version se donne le droit d'accepter ou non dans le but de conserver un projet fonctionnel sur le *master*. Thibaut était l'architecte de ce projet. Comme son nom l'indique, son objectif était de gérer l'architecture du projet pour que celui-ci soit SOLID et le plus propre possible. De plus, il mettait en place différents diagrammes montrant l'architecture du projet pour contribuer au bon développement de celui-ci. Clément, Calliste, Blaise et William étaient eux les développeurs. Bien que tout le monde participait à l'élaboration du code, ce sont eux qui avaient pour rôle principal l'implémentation des fonctionnalités.

Le travail en équipe ne s'arrête cependant pas à la mise en place de rôle pour chaque membre. Pour réaliser un bon travail en équipe, nous avons dû faire preuve de beaucoup de communication et d'écoute.

4. ETUDE D'AVANT CODE

Lors de la première séance de travail, nous ne nous sommes pas directement lancés dans le code mais nous avons effectué un travail préliminaire pour essayer de prévoir le travail.

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

Nous avons par exemple effectué une analyse des risques du projet. Dans celle-ci, nous avons essayé de lister tous les problèmes qui pouvaient survenir lors de la réalisation de ce projet. Nous avons attribué à chaque risque une valeur de fréquence et une valeur de criticité notées sur 4. Plus la valeur de fréquence est élevée, plus le risque a de chance de survenir, et plus la criticité est élevée, plus le risque aura un impact négatif sur la réalisation du projet s'il survient. Pour chacun des risques trouvés, nous avons également essayé de trouver une solution pour les contrer et éviter au maximum qu'ils arrivent.

Enoncé des risques	Gravité (/4)	Fréquence (/4)	Criticité (/16)	Prévention
Manque de connaissance sur le langage	3,5	3	10,5	Bien se renseigner sur le langage et tirer avantages de ceux qui connaissent le mieux le langage.
Manque de communication au sein du groupe	3	1,5	4,5	Suivre le rôle de chacun, faire des petites réunions avec les avancées du projet.
Mauvaise utilisation du GitLab	2,5	2	5	Bien se renseigner sur le Git (TP de début d'année), bien vérifier ce que l'ont fait.
Le projet ne convient pas au client	4	1,5	6	Rendre des prototypes (MVP) régulièrement au client et écouter ses demandes et ses remarques.
Projet trop ambitieux pour le temps imparti	2	2	4	Bien connaître nos compétences et nos limites.
Projet trop commun qui n'apporte pas grand-chose par rapport à ce qui existe déjà	1	3	3	Etre original, ajouter des petits trucs en plus.
Absentéisme trop prononcé	3,5	0,5	1,75	Doubler les rôles pour prévenir une absence.
Problème de versions, configuration de l'environnement	2	1	2	Bien regarder les versions et bien paramétrer les logiciels avant de commencer à coder.
Problème au niveau des serveurs de l'école	3	2	6	Avoir des sauvegardes du projet sur plusieurs machines et sur le Git

Si on prend par exemple le risque « Absentéisme trop prononcé », la solution que nous avons trouvée est de doubler les rôles afin de prévenir l'absentéisme. Ainsi, Calliste était vice-responsable de gestion, William était vice-architecte et Clément vice-chef de projet.

Avant de commencer à coder, nous avons élaboré un diagramme de classe et un diagramme des cas d'utilisation qui représentaient l'idée que nous avions du projet. Au fur et à mesure de l'avancement du code et du fait de l'utilisation d'une méthode agile, les diagrammes finaux s'avèrent être différents des diagrammes préventifs.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

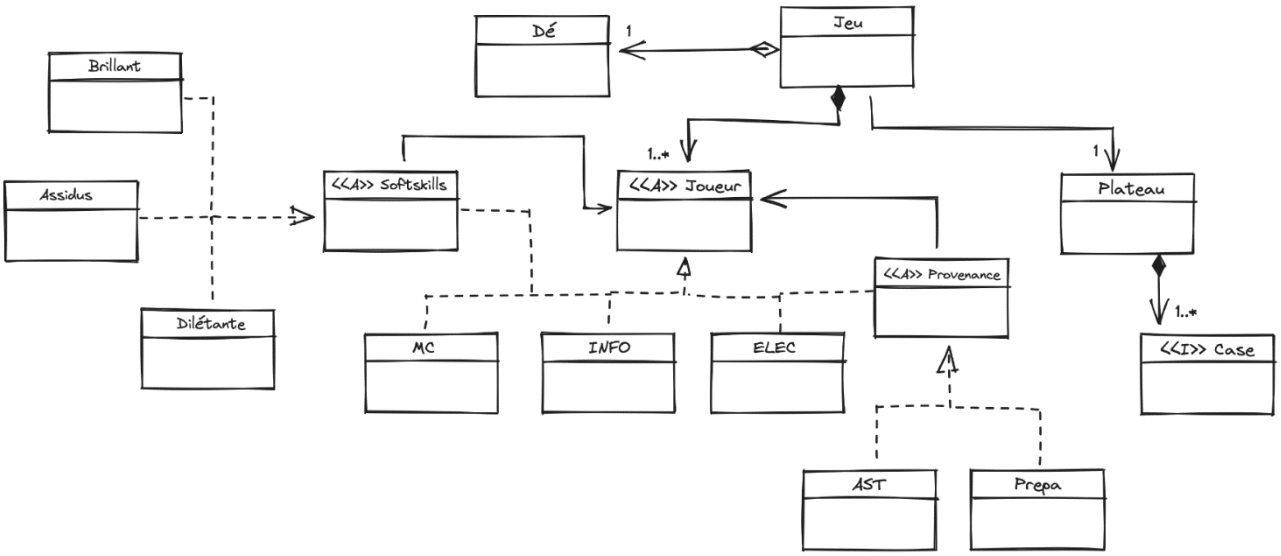

Figure 1: diagramme de classes préventif

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

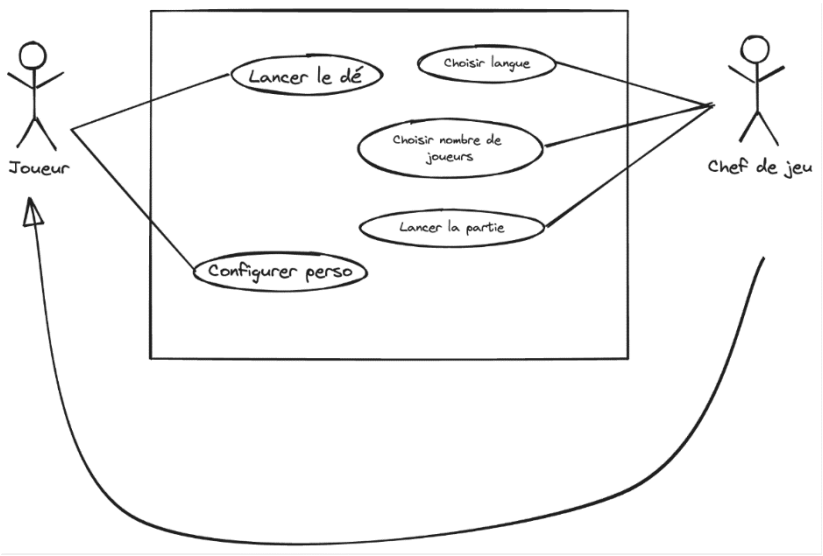

Figure 2: diagramme de cas d'utilisation préventif

5. DEVELOPPEMENT DU CODE

Lorsque les diagrammes nous indiquant la marche à suivre ont été terminés, nous pouvions commencer à coder. En discutant, nous nous sommes reparti les différentes classes à coder. Chacun codait sur sa propre branche, et lorsqu'il avait fini d'implémenter sa classe, il faisait une *merge request* au gestionnaire de versions, en ayant au préalable résolu les éventuels conflits qu'entrainerait une fusion avec la branche principale. Certains ont voulu avancer le projet en dehors des séances prévues à cet effet, auquel cas ils travaillaient de la même manière

(sur une branche attitrée et en demandant des merge). Nous avons ensuite appliqué la méthode agile. Nous avions un premier objectif pour la troisième séance (après six heures de travail) : avoir un MVP (*Minimum Viable Project)*. Ici, on cherchait à avoir une version simple, fonctionnelle et sans bug. Nous devions donc avoir un plateau, au moins un pion dont la couleur pouvait être choisit au début de la partie ainsi qu'une manière de déplacer les pions à l'aide d'un dé lancé en appuyant sur un bouton. Cet objectif a été atteint. Ensuite, on s'est approché de la version finale avec l'implémentation des effets des différentes cases (notamment celle des cases examens avec le système de niveau de compétence), la contrainte de superposition des pions qui engendre l'inversion des positions des deux joueurs, le fait de « rebondir » sur la case d'arrivée ainsi que la prise en compte de la filière et des précédentes études.

Le choix de répartition des tâches s'est initialement fait en rassemblant l'équipe au début des séances. Le chef d'équipe faisait un résumé de ce qu'il restait à faire, puis c'est en parlant les uns avec les autres que les choix étaient faits. Ce n'est qu'à partir de la quatrième séance de travail (après la sixième heure) que nous avons un outil sur le logiciel Gitlab : un gestionnaire des tâches, permettant de trier les tâches à réaliser, celles en cours et celles terminées dans un tableau que chacun peut consulter et utiliser en s'attribuant une tâche. Le chef d'équipe a pu rempli ce tableau ce qui nous a fait gagner du temps et a permis de simplifier l'attribution des tâches à faire.

En avançant dans le développement, certaines parties du diagramme de classes ont été modifiées afin de rendre le code plus simple. Cependant, certaines erreurs ont été faites, et lors d'une séance, à la demande de l'architecte, nous nous sommes tous rassemblés. Il avait

certaines requêtes quant à l'organisation du code, ayant pour but de rendre le code SOLID et plus propre.

Grâce à une très bonne communication au sein de l'équipe, il n'y a eu que très peu de conflits lors des tentatives de *merge*. Cependant, à la fin du projet, il y eût un problème majeur. Un développeur a travaillé trop longtemps sur sa branche et a demandé de *merge* avec la branche principale. Du fait de sa relative inexpérience, le gestionnaire de version donna son accord à une action qui fit perdre certaines parties importantes du code. Il dût alors s'occuper de récupérer les parties du code perdues en récupérant une version antérieure du projet et d'y incorporer les fonctionnalités que le développeur avait codé.

6.ARCHITECTURE DU CODE

Lors de la première itération de la méthode AGILE, nous avons défini un diagramme de classe de départ, comme expliqué précédemment. Au fur et à mesure que le projet avançait, nous changions plus ou moins l'architecture de celui-ci. Car nous nous sommes rendu compte qu'il était impossible de prévoir à l'avance une structure fixe. En effet, un projet est voué à grandir, surtout en phase de développement. Prévoir un modèle à l'avance ne permet pas de tenir compte des fonctionnalités futures. C'est pourquoi nous nous sommes concentrés sur les fonctionnalités à implémenter. Ce n'est qu'après cela que nous changions l'architecture du code. Cela permet d'éliminer le couplage entre les classes. C'est donc pour cela que notre diagramme de classe final, ci-après, est différent du premier.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

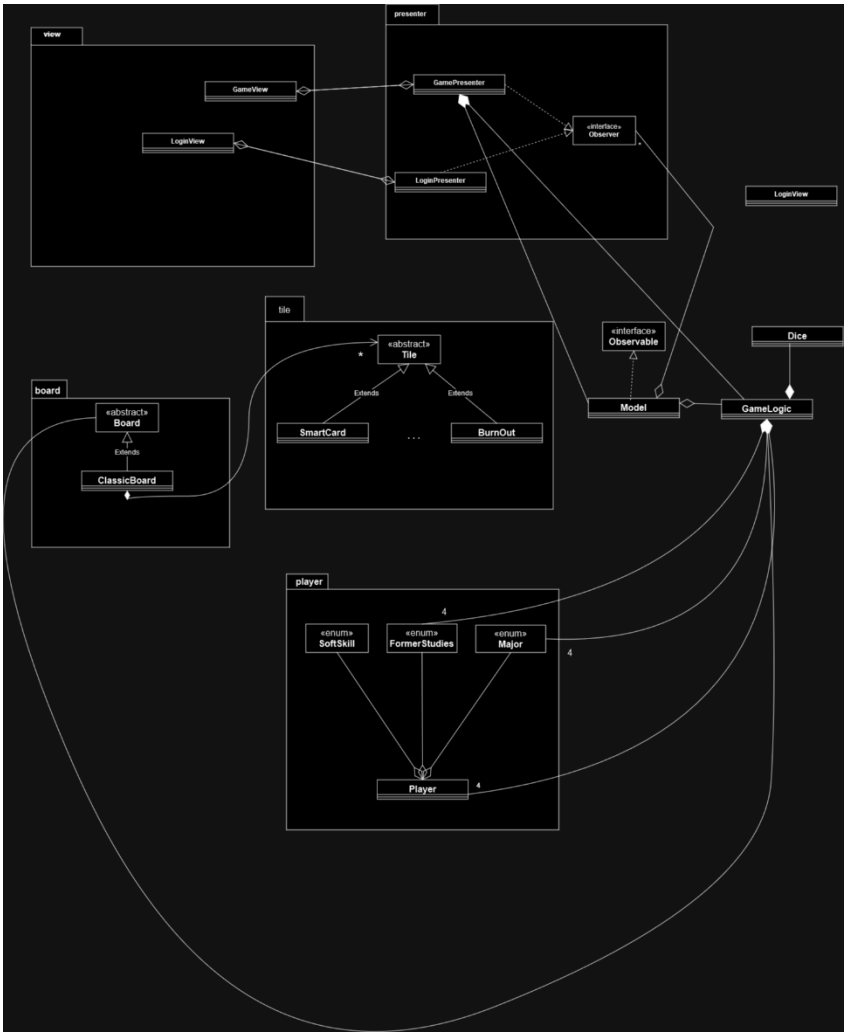

Figure 3: diagramme de classe final

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

Ensuite, notre code respecte le patron d'architecture MVP (modèle – vue – présentateur). Le modèle ignore tout de la vue et la vue n'a aucune idée de l'implémentation du modèle. Le présentateur faisant office d'intermédiaire entre les deux classes. Il n'y a aucun import du modèle dans la vue, pas plus qu'il n'y a d'import d'éléments graphiques dans le modèle. Afin de mettre en place l'architecture MVP, nous avons utilisé le patron de conception Observer. Nous avons créé une interface Observable que le modèle implémente et une autre interface Observer que le présentateur utilise pour se tenir informé des changements du modèle. Cette façon de coder, en utilisant le patron Observer nous a paru tout à fait adéquat au projet.

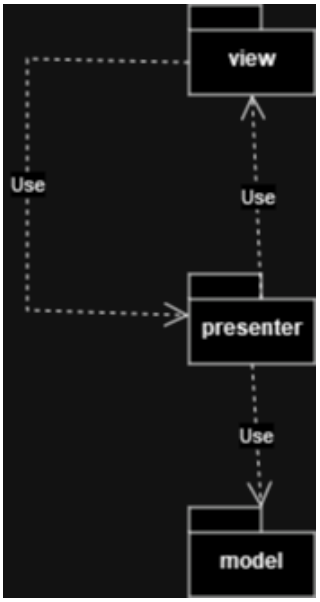

Figure 4: diagramme de paquet

En plus de cela nous avons des parties de code SOLID comme le plateau. En effet, l'utilisateur peut ajouter d'autres plateaux à sa convenance sans modifier le reste du code. Il en est de même pour les cases. En conclusion, nous ne nous sommes pas forcés à utiliser des patrons de conception à tout va. Le plus important reste que le code soit solide, maintenable, et testable.

7. PERSPECTIVES D'AMELIORATION

Dans le cadre de ce projet, nous n'avons pas pu terminer entièrement la programmation du jeu de l'oie et de toutes ses fonctionnalités. Ainsi, nous avons de nombreuses perspectives d'amélioration pour notre code et de nouvelles implémentations possibles.

Nous n'avons par exemple pas implémenté la possibilité d'avoir plusieurs plateaux ni la mise en place d'un fichier JSON pour lire le plateau.

Nous n'avons pas non plus mis en place la détermination du salaire de sortie lorsqu'un joueur passe la ligne d'arrivée et la base de données des meilleurs scores qui en découle.

En prenant du recul sur le travail que nous avons réalisé, certaines améliorations aurait pu être effectuées.

Il y a par exemple l'utilisation du git que nous n'avons pas toujours su bien gérer et qui a été la source de la perte du travail de certains membres. C'est aussi le cas de notre méthode de programmation. En effet, nous n'avons pas toujours codé initialement en méthode agile avec un code SOLID. Cela nous a donc obligé plusieurs fois à revenir sur des portions de code que nous avions déjà écrites pour les modifier afin de rendre le code plus SOLID.

8. CONCLUSION

Enfin, la réalisation de ce projet était un exercice très intéressant. Elle nous a permis de comprendre concrètement ce qu'est la méthode agile et les difficultés d'écrire un code SOLID. Au travers de ce projet, nous avons également pu nous rendre compte que dans un travail en équipe, la communication et la répartition claire des rôles est primordiale : elle permet un gain de temps conséquent lors de l'écriture du code.

Bien que le jeu de l'oie ne soit pas entièrement fini, ce projet fut également gratifiant car très concret. En effet, il était très satisfaisant de voir l'évolution du comportement des pions, des cases et autres fonctionnalités qui étaient apportées par ce que nous codions.

Enfin, par suite de ce projet, nous nous sommes également rendu compte de l'importance du nombre de personne dans un groupe de travail. En effet, être 7 sur un travail de ce type est

peut-être un peu trop et, en parlant avec des personnes n'étant que 5 dans leur groupe, nous constations parfois qu'ils avançaient autant voire plus vite que nous.

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04