Document
INFO-F307 - Génie logiciel et gestion de projets

Introduction

Ce document détaille tous les choix de conception et d'utilisation d'outils pour mener à bien le projet demandé par le client. Lors de cette troisième itération, 3 histoires ont été implémentées comme prévu lors de la réunion avec le client. Les 3 histoires sont les suivantes : modes de jeu (6), score et classement (12) et les cartes modulables (21) . Il y est aussi indiqué les quelques designs patterns utilisés pour coder efficacement en orienté objet. De plus, le code suit le pattern d'architecture MVC. Pour finir, les difficultés rencontrées sont également mentionnées

Histoire 6

Difficultés rencontrées

Étant donné la similarité entre les deux modes de jeux (libre et étude), la difficulté fut de ne pas répéter du code et d'essayer de mettre la similarité entre les différentes classes dans une classe abstraite ou une interface.

Choix de conception

Le score pour un paquet a été calculé en divisant la somme des bonnes réponses par le temps pris.

Pour l'interface, il a été décidé de ne garder que les cartes à choix multiples pour le mode de jeu libre. Pour ce faire, nous filtrons les cartes au lancement du mode en n'affichant que les cartes compatibles avec le dit mode. On a donc gardé l'ancienne interface et l'avons modifiée pour qu'elle soit modulable avec le mode sélectionné. Aussi, pour ce mode de jeu, nous piochons une seule fois toutes les cartes pour ensuite afficher le score

Nous avons ajouté une classe parent abstraite CardPicker qui est utilisée pour tirer la prochaine carte à étudier. Elle contient la structure principale que doivent contenir les différents classes ayant la responsabilité de l'algorithme de tirage de carte en fonction du mode d'étude choisi.

Pour le mode d'étude standard (où les cartes les moins maîtrisées reviennent le plus souvent), la classe RandomCardPicker a été codée lors d'une itération précédente. Lors de cette itération, la seule modification effectuée est que celle-ci hérite désormais de CardPicker. De plus, nous avons codé FreeCardPicker (qui hérite également de CardPicker) qui est l'algorithme pour le mode de jeu libre. Ainsi, avec cette structure, il est plus aisé de rajouter des modes de jeux à l'avenir.

Histoire 12

Choix de conception

Pour le ranking nous avons ajouté une nouvelle table qui regroupe : l'utilisateur, le deck et le score que le joueur a fait sur ce deck. Du côté serveur, la même structure que les autres appels a été suivie.

Histoire 21

Difficultés rencontrées

Nous avons eu des difficultés à transcrire une opération sous forme de string a une véritable opération. Cela a été résolu grâce à l'algorithme de transformation Infix à Postfix et de forcer l'utilisateur à mettre des espaces entre chaque élément de l'opération.

Un autre problème a été que l'utilisateur utilise le signe spécial pour faire des opérations dans sa question ou réponse .

Ce problème a été résolu en interdisant d'utiliser ce signe spécial.

Choix de conception

Il a été décidé d'utiliser:

- des variables entre : par exemple, :a:
- il faut instancier la question avec le même nombre de variables que la question
- un exemple complet :
    - titre : "chose"
    - question: "combien font :a: pommes plus :b: pommes"
    - réponse: ":a: + :b:"

L'opération ne peut être utilisée que dans la question.
Les variables seront remplacer par un chiffre entre 0 et 100 dans la question comme dans la réponse, même si elle est utilisée dans une opération.

Amélioration UX UI

Nous avons décidé de subdiviser les fichiers de la vue du deck et d'améliorer l'expérience utilisateur. Avant cela, l’utilisation n’était pas du tout intuitive. Désormais, chaque option est gérée par une nouvelle fenêtre.

Database injections

Les injections ont été patché et nous avons rajouté les exceptions

Pour conclure l’itération 3, nous avons implémenté toutes les fonctionnalités des 3 histoires.