Travail Pratique #2 – Plan de test

**Introduction** – Décrivez vos objectifs principaux et vos critères de succès. Par exemple, allez-vous concentrer vos efforts de test sur certaines parties de votre système ou tenter de tout tester de façon uniforme? Pourquoi? Allez-vous tester seulement les besoins fonctionnels, ou avez-vous des besoins non-fonctionnels à tester également? Comment allez-vous déterminer à quel point vous aurez suffisamment testé votre système?

Le présent document a pour objectif de décrire les stratégies de tests qui seront employés pour le jeu de cartes collectionnables en ligne Mysticat.

Pour ce faire les tests seront regroupés en trois catégories distinctes soit les tests unitaires, les tests d’intégration et les tests d’application et seront répartis de telle manière que l’équipe testera à la fois l’interface du logiciel (application) et son fonctionnement back-end (test unitaires).

Les tests mettront généralement l’emphase sur les diverses fonctionnalités du programme mais ne se limiteront toutefois pas à celles-ci.

En effet, l’équipe prévoit également tester les aspects techniques du logiciel tel que la performance et la gestion du programme selon un nombre défini d’usagers connectés en même temps, la réactivité des éléments de l’interface vis-à-vis plusieurs formats d’écran, l’ergonomie pour l’utilisateur, etc.

De plus une attention particulière sera portée sur l’authentification, les parties de cartes en cours et les files d’attente car ils représentent les fonctionnalités clé du logiciel et constituent le cœur du produit minimum viable.

Des revues de code ainsi que des critères de succès ont également été préalablement établis par l’équipe afin d’établir la pertinence de chacun des tâches de développement.

Parmi ces derniers

* Pour l’interface : afficher un message de chargement si le service n’est pas disponible, une zone main de cartes pour les joueurs s’affiche durant une partie en cours et doit pouvoir contenir jusqu’à 10 cartes représentées, les cartes de l’adversaire s’affichent seulement de dos, etc.
* Pour le fonctionnement back-end : le programme indique un problème si les informations utilisateurs ne correspondent à aucun utilisateur durant l’authentification, les cartes sont jouées une à une selon leur vitesse, etc.

Le système ne sera complètement testé que quand toutes les méthodes testables du code back-end autre que les mutateurs et les accesseurs auront leur test unitaire correspondant avec tous les cas envisagés possible, que l’interface aura été éprouvé par tous les cas usagers possible, que la gestion et les performances du logiciels répondront adéquatement aux standards fixé par l’équipe (soit qu’il puisse gérer une trentaine de joueurs à la fois ainsi qu’un administrateur) et qu’aucune erreur ne sera trouvé par aucun membre de l’équipe.

Pour les tests, nous allons faire un système de test qui va couvrir le code testable. Les tests vont passer principalement le back-end et les critère de succès relier au back-end. On pourra tester la partie du code qui gère les élements important et les calcul pour le fonctionnement des fonctionnalités.

Voilà nos critères de succès :

* Si le service n'est pas disponible, on affiche un message chargement en tentant de rejoindre le service.
* Un grand rectangle s'affiche pour englober le plateau de jeu.
* Une zone par joueur pour la main s'affiche et est suffisament grande pour contenir 10 cartes..
* Les cartes s'affichent dans la main du joueur courrant (self) et peut contenir jusqu'a 10 cartes.
* Des card backs s'affichent pour representer les cartes dans la main du joueur adverse et peut contenir jusqu'a 10 cartes.
* Une carte dans la main du joueur courrant affiche tous les details de la carte (Speed, Health, Power, Name).
* Une carte dans la main du joueur adverse s'affiche comme un card back.
* Une pile de carte (card back) s'affiche pour les graveyards et les decks des 2 joueurs.
* Un compteur affiche le nombre de cartes restantes dans les graveyards et les decks des 2 joueurs.
* Les champs de battaille s'affichent au milieu du board, avec suffisamment de place pour 7 minions de chaque cote.
* Les cartes, minions, deck, graveyards s'affichent du cote du joueur auquel ils appartiennent.
* Un minion s'affiche sur le champ de battaille et affiche ses informations (Speed, Health, Power, Name).
* Le nom de chaque joueur s'affiche de son côté du plateau de jeu.

**Détails de tests** – Décrivez chaque groupe de tests défini, ainsi que ses objectifs et ses conditions de succès. Pour les tests unitaires, il est suffisant de décrire la stratégie générale des tests pour chaque classe (et non pas de décrire précisément chacun des tests). En particulier, n’incluez pas de code dans votre remise. Par exemple, pour une classe responsable de transformer du texte, vous pouvez mentionner que vous utiliserez une série d’entrées et de sorties générées manuellement, et de complexité variable. Mentionnez aussi comment vous prévoyez tester les cas d’exception (par exemple, quels types d’exceptions vous désirez tester), si applicable.

**Les tests unitaires**

Tests effectués avec l’implémentation des blocs @Before et @After pour s’assurer de rester DRY.

DeckEditingService

Tester le service avec des paquets de cartes au caractéristiques variables (nombre de cartes, nombre de caractères dans le nom, etc.) conformes et non-conformes. Faire des assertions d’exceptions personnalisées lancées lorsque le paquet de cartes est non conforme.

ConnectionUtils

Tester les méthodes utilitaires avec des informations utilisateurs variables (mots de passes, nom d’usager, etc.)

GameService

Tester le service avec plusieurs rapports spécifiques entre des actions pour vérifier le déroulement du programme selon le type d’actions, les caractéristiques des minions, etc.

LoginService

Tester l’authentification des informations, l’ajout d’un utilisateur et la suppression d’un utilisateur avec un utilisateur valide, des informations utilisateurs variables et un mock du repository d’utilisateurs.

Tester la construction et l’attribution d’un deck par défaut avec un mock du repository de cartes ainsi qu’un paquet de cartes dummy.

Isoler chacun des cas valides et invalides des méthodes avec des assertions et utiliser assertJ pour faire des assertions d’exceptions personnalisées lancées.

Enfin, vérifier que les mocks se sont bien fait appelés un certain nombre de fois pour savoir si le code concernant leur appel a été couvert.

JsonUtils

Tester les méthodes utilitaires avec des Json de validités variables.

**Plan d’intégration** – Décrivez votre plan d’intégration en prenant bien soin de justifier vos choix. Présentez ici comment vous prévoyez effectuer les tests d’intégration au cours du développement. Indiquez par exemple dans quel ordre les modules seront assemblés pour former un système complet. Les détails des tests d’intégration peuvent être présentés ici ou dans la section suivante, à votre choix.

**Responsabilités des membres de l’équipe** – Décrivez ici la responsabilité de chacun des membres de l’équipe. Chaque membre doit obligatoirement participer aux activités de tests. Si votre équipe divise le travail par composant, chaque membre de l’équipe devrait au minimum être responsable de son (ses) composant(s).

**Calendrier** – Vous devez établir un calendrier réaliste des tests présentés dans votre document.