420-5W5 – Sprint 3

Projet

Super Cartes Infinies (aka Super HeartStone Infini)

Date de remise : Semaine 15

# L’application

## **Objectifs** :

Ajouter des fonctionnalités au jeu de cartes développé lors des Sprints 1 et 2.

## Note importante :

Si vous n’avez pas réussi à compléter une partie du TP2, regardez avec votre enseignant pour savoir comment faire le TP3 sans devoir travailler en double.

## Fonctionnement du 3e Sprint :

Pour le 3e Sprint, il n’y aura que 4 tâches à réaliser, chaque tâche devra être à attribuer à un coéquipier et il n’y aura pas d’autre tâche d’équipe supplémentaire.

Le détail des tâches individuelles

## Achat de paquets de cartes dans le magasin

* Ajouter une rareté aux cartes. Il doit y avoir 4 niveaux:
  + Commune (Gris)
  + Rare (Vert)
  + Épique (Mauve)
  + Légendaire (Orange)
* Ajouter la possibilité de voir et changer la rareté d’une carte en MVC.
* Sur le client, il faut afficher un code de couleur sur les cartes pour pouvoir voir leur rareté. Une option simple c’est de modifier la couleur de fond du titre.
* Ajouter 3 paquets de cartes que les joueurs peuvent acheter. Les cartes sont obtenues au hasard, mais en suivant les règles suivantes.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Rareté par défaut | Nb Cartes | Règles d’obtention des cartes (sinon c’est la rareté par défaut) |
| Basic | Commune | 3 | 30% de rare |
| Normal | Commune | 4 | 1 carte rare. Pour le reste : 30% rare, 10% épique, 2% légendaire |
| Super | Rare | 5 | 1 carte épique. Pour le reste 25% épique, 10% légendaire. (Aucune commune) |

* Les paquets ont également un nom, une url d’image et un prix.
* Sur le client, il faut afficher les différents paquets que le joueur peut acheter.
* Lorsque le joueur en achète un, il faut lui afficher les cartes qu’il vient de recevoir dans un Dialog. Il faut également mettre à jour son argent et ses cartes.
* Le joueur ne peut pas acheter un paquet s’il n’a pas assez d’argent.
* La configuration des paquets et des probabilités doit être fait dans un seed, ce n’est pas nécessaire de pouvoir les modifier en MVC.
* Voici le pseudo code pour obtenir une liste de raretés pour notre paquet de carte

// Une Probability possède : une value (entre 0 et 1), une rarity et une baseQty

// Faire une liste de rareté de carte à obtenir

List<Rarity> GenerateRarities(int nbCards, int defaultRarity, List<Probability> probabilities)

rarities = new List<Rarity>

// Ajouter la quantité de base pour chaque probability à la liste

foreach(probability of probabilities)

for probability.baseQty

add probability.rarity to rarities

// Continuer de remplir la liste jusqu'à atteindre la quantité voulue

while(rarities.Count < nbCards)

rarity = GetRandomRarity(probabilites)

if(rarity == null)

add defaultRarity to rarities

else

add rarity to rarities

return rarities

// Cette méthode permet d'obtenir une rareté au hasard

Rarity? GetRandomRarity(List<Probability> probabilities)

X = Random Number Between 0 and 1

for each rarity of probabilities:

if probability.value < X:

return probability.rarity

else:

X -= probability.value

return null

* Une fois que l’on a une liste de rareté, on peut prendre une carte au hasard avec chacune des raretés pour faire notre paquet. Les doublons sont permis.

## Statistiques des joueurs

* Un joueur aura la possibilité de voir des statistiques à propos de ses decks ou de l’ensemble de ses cartes
  + Il pourra voir le **nombre de victoire** et **défaites** avec ce deck (ou général)
  + La distribution des cartes (En utilisant des graphs similaires):
    - Coût
    - Rareté
    - Attaque et défense
  + Vous pouvez utiliser la technologie de graph que vous préférez, mais on propose la suivante : <https://canvasjs.com/angular-charts/pie-chart-index-data-label/>

*Lorsqu’on affiche l’ensemble des cartes*

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

*Lorsque l’on sélectionne un deck*

Une image contenant texte, capture d’écran, diagramme, Logiciel multimédia

Description générée automatiquement

## Habilités supplémentaires

* Il y aura l’ajout d’habilités pour les cartes qui vont modifier un état.
  + **Poison X**, qui ajoute une valeur de poison à la carte attaquée. Le poison diminue ensuite la vie d’une carte de la valeur du poison à la fin de son activation. Si une carte a déjà une valeur de poison et qu’elle est à nouveau attaquée, la valeur de poison est augmentée.
  + **Stunned X**, qui empêche une carte d’agir pendant son activation durant X tours. (Mais elle reçoit quand même les dégâts de poison!)
* Il y aura également l’ajout de cartes qui auront un effet immédiat et qui se déplaceront directement dans le « graveyard » par la suite (On n’attend pas la fin du tour).
  + **Lightning Strike X**, fait X dégâts au joueur adverse. (Attention, c’est possible que la partie se termine)
  + **Earthquake X**, fait X dégâts à TOUTES les cartes en jeu.
* Il faut ajouter au moins un test complet pour chacun des pouvoirs. C’est une bonne idée de travailler en Test Driven Development et d’écrire les tests d’abord. Pour simplifier ce TP, le même programmeur va écrire les tests et la logique.

## SignalR

* Il faudra remplacer toutes les requêtes http du « gameplay » par l’utilisation de **SignalR**.
  + Tout doit être remplacé, du polling réalisé pour JoinMatch, jusqu’à la fin de la partie. Le **Hub** doit donc fournir une méthode pour faire **JoinMatch**, **Cancel**, **PlayCard**, **EndTurn** et **Surrender**.
  + L’action **StartMatch**, n’est plus nécessaire et peut être exécuté directement dans la méthode de Hub qui gère le **JoinMatch**.
* **SignalR** devra utiliser **Identity** et **[Authorize]** dans le **Hub** pour restreindre l’accès aux utilisateurs connectés.