**« Backlog » de sprint #000**

Produit : Astéria

Conçu par : EquBolduc

## Équipe :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Initiales** | **Responsabilité** |
| Émile Grégoire | É. G. | Chef d’équipe |
| Jonathan Samson | J. S. | Responsable des livrables |
| Simon-Pierre Deschênes | S.-P. D. | Responsable de la qualité |
| Jérémie Bolduc | J. B. | Responsable des réunions |

## Éch**é**ancier:

Du 11 février au 25 février 2015.

## Légende :

* Vert, indique que ces items sont réalisés.
* Jaune, indique que ces items sont en cours de réalisation.
* Rouge, problème ou questionnement important qui demande une rencontre d’équipe.
* Aucune couleur, indique que ces items ne sont pas encore faits ou commencés, **on peut toujours les enrichir mais il faut le consentement de toute l’équipe**.

## « Backlog » de sprint

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scénario #2** | | | | |
| Acteur | | | Développeurs | |
| Scénario | | | En tant que développeur, je veux pouvoir changer de contrôleur et de vue facilement afin de simplifier les tests sur le modèle physique. | |
| Description | | | Créer un contrôleur principal qui contiendra les vues et les contrôleurs.   * 1. Qui et temps      1. É.G.      2. 30 min.   2. Préconditions   3. Règles d’affaires      1. Créer un contrôleur principal qui contient les vues et les contrôleurs.      2. Rendre ce contrôleur accessible depuis partout (Singleton).      3. Créer les interfaces contrôleurs et vues.   D:\Bureau\Diagramme de classe.png   * 1. Règles d’affaires alternatives      1. S’assurer que les contrôleurs et les vues ne sont pas null.   2. Tests d’acceptation      1. Créer des tests unitaires avec des classes contrôleurs et vues bidons.   3. Post-conditions      1. Le contrôleur principal est accessible partout.   Implémenter les changements entre les contrôleurs.   * 1. Qui et temps      1. É.G.      2. 30 min.   2. Préconditions      1. Le contrôleur principal est accessible partout.   3. Règles d’affaires      1. Charger le contrôleur.      2. Ajouter les écouteurs nécessaires.   4. Règles d’affaires alternatives      1. S’assurer que le contrôleur n’est pas null.   5. Tests d’acceptation      1. Créer des tests unitaires avec une classe contrôleur bidon.   6. Post-conditions   Implémenter le chargement des vues.   * 1. Qui et temps      1. É.G.      2. 30 min.   2. Préconditions      1. Le contrôleur principal est accessible partout.   3. Règles d’affaires      1. Charger le fichier FXML.      2. Charger le fichier CSS.   4. Règles d’affaires alternatives      1. S’assurer que la vue n’est pas null.   5. Tests d’acceptation      1. Créer des tests unitaires avec une vue bidon.   6. Post-conditions | |
| Tests d’acceptation | | | Créer deux contrôleurs de test et de vue pour s’assurer que les changements s’opèrent bien. | |
| Complexité | | | 3 | |
| Effort | | | 2 | |
| Commentaires | | |  | |
| **Scénario #3** | | | | |
| Acteurs | | | Développeurs | |
| Scénario | | | En tant que développeur, je veux pouvoir me servir d’une horloge interne qui me retournera des variations de temps utiles dans les formules de physique. | |
| Description | | | 1. Écrire un thread qui calcule la différence de temps entre un temps x et y.   * 1. Qui et temps      1. J.S      2. 30 min.   2. Préconditions      1. Avoir accès au contrôleur principal.   3. Règles d’affaires      1. Écrire un thread qui calcule le temps courant du système moins la valeur précédente du système.      2. Appeler une méthode qui mettra à jour le moteur physique.   4. Règles d’affaires alternatives      1. S’assurer que le thread s’arrête avec l’application.      2. S’assurer que le setDaemon() est vrai.   5. Tests d’acceptation      1. Avoir un thread simple, propre, facile à comprendre.      2. Vérifier que le thread appelle la méthode update.   6. Post-conditions      1. Nous avons accès aux variables de temps partout.   2. Dans le contrôleur principal, démarrer le thread.   * 1. Qui et temps      1. J.S.      2. 30 min.   2. Préconditions      1. Le thread doit être écrit et doit retourner sa valeur de temps du système.   3. Règles d’affaires      1. Démarrer le thread.      2. Lui donner un sleep de 5 millisecondes pour ne pas le surcharger.   4. Règles d’affaires alternatives   5. Tests d’acceptation      1. Vérifier que le thread est en vie.   6. Post-conditions      1. 3.18.1. Les valeurs de temps sont accessibles. | |
| Tests d’acceptation | | | Demander d’afficher le nombre de secondes ou minutes qui se sont écoulées entre deux appels de la méthode. | |
| Complexité | | | 2 | |
| Effort | | | 1 | |
| Commentaires | | |  | |
| **Scénario #4** | | | | |
| Acteurs | | | Développeurs | |
| Scénario | | | En tant que développeur, je veux utiliser une classe Vecteur afin d’illustrer les mouvements entre les différents corps. | |
| Description | | | Implémenter l’addition, la soustraction et la multiplication par des scalaires de vecteurs.   * 1. Qui et temps      1. S.-P.D.      2. 20 min.   2. Préconditions   3. Règles d’affaires      1. Créer une méthode qui additionne le vecteur attribut à celui passé en paramètre.      2. Créer une méthode qui soustrait le vecteur attribut à celui passé en paramètre.      3. Créer une méthode qui multiplie le vecteur attribut par un scalaire passé en paramètre.   4. Règles d’affaires alternatives      1. S’assurer que les vecteurs passés en paramètre ne sont pas null.   5. Tests d’acceptation      1. Additionner, soustraire et multiplier par des scalaires des vecteurs et vérifier la cohérence des résultats.   6. Post-conditions  1. Implémenter le produit scalaire et le produit vectoriel de vecteurs.    1. Qui et temps       1. S.-P.D.       2. 20 min.    2. Préconditions    3. Règles d’affaires       1. Créer une méthode qui fait un produit scalaire du vecteur attribut avec celui passé en paramètre.       2. Créer une méthode qui fait un produit vectoriel du vecteur attribut avec celui passé en paramètre.    4. Règles d’affaires alternatives       1. S’assurer que les vecteurs passés en paramètre ne sont pas null.    5. Tests d’acceptation       1. Faire des produits scalaires et vectoriels de vecteurs et vérifier la cohérence des résultats.    6. Post-conditions 2. Implémenter la normalisation et le calcul de modules de vecteurs.    1. Qui et temps       1. S.-P.D.       2. 20 min.    2. Préconditions    3. Règles d’affaires       1. Créer une méthode qui crée un vecteur de longueur 1 dans la même direction que le vecteur attribut.       2. Créer une méthode qui calcule la longueur du vecteur attribut.    4. Règles d’affaires alternatives    5. Tests d’acceptation       1. Normaliser et faire le calcul de modules de vecteurs et vérifier la cohérence des résultats.    6. Post-conditions | |
| Tests d’acceptation | | | Effectuer différentes opérations vectorielles et vérifier la cohérence des résultats via des tests unitaires. | |
| Complexité | | | 1 | |
| Effort | | | 1 | |
| Commentaires | | |  | |
| **Scénario #5** | | | | |
| Acteurs | | | Développeurs | |
| Scénario | | | En tant que développeur, je veux pouvoir travailler avec des corps physiques qui seront soit statiques, soit en mouvement. | |
| Description | | | 1. Créer l’interface Corps qui sera utilisée par les vaisseaux et les planètes.    1. Qui et temps       1. J.B.       2. 1 h.    2. Préconditions       1. La classe Vecteur est opérationnelle.    3. Règles d’affaires       1. Créé les méthodes relative à la masse.       2. Créé les méthodes relative à la vitesse.       3. Créé les méthodes relative à la position.       4. Créé les méthodes relative aux forces.    4. Règles d’affaires alternatives       1. S’assurer que la vue n’est pas null.    5. Tests d’acceptation       1. Créer des tests unitaires avec une vue bidon.    6. Post-conditions 2. Créer la classe Planète qui implémente l’interface Corps.    1. Qui et temps       1. J.B.       2. 1 h 30 min.    2. Préconditions       1. L’interface corps est créée.    3. Règles d’affaires       1. Créé les attributs de la classe (property).       2. Compléter les méthodes de l’interface Corps.    4. Règles d’affaires alternatives       1. S’assurer que les attributs ne sont pas null    5. Tests d’acceptation       1. Créer des tests unitaires avec des données bidon.    6. Post-conditions 3. Créer la classe Vaisseau qui implémente l’interface Corps.    1. Qui et temps       1. J.B.       2. 1 h 30 min.    2. Préconditions       1. L’interface corps est créée.    3. Règles d’affaires       1. Créé les attributs de la classe (property).       2. Compléter les méthodes de l’interface Corps.       3. Créé les méthodes pour tourner.    4. Règles d’affaires alternatives       1. S’assurer que les attributs ne sont pas null    5. Tests d’acceptation       1. Créer des tests unitaires avec des données bidon.    6. Post-conditions | |
| Tests d’acceptation | | | Vérifier que les valeurs retournées ont du sens. | |
| Complexité | | | 1 | |
| Effort | | | 2 | |
| Commentaires | | |  | |
| **Scénario #6** | | | | |
| Acteurs | | | Développeurs | |
| Scénario | | | En tant que développeur, je veux que les différents corps puissent s’attirer entre eux afin que le jeu soit réaliste physiquement. | |
| Description | | | 1. Récupérer tous les corps    1. Qui et temps    2. Préconditions       1. Le moteur physique doit être appelé par l’horloge interne.       2. Une liste de tous les corps physiques sont disponibles dans le contrôleur principal.    3. Règles d’affaires       1. À chaque frame, récupérer tous les corps physiques à partir du contrôleur principal.       2. Vérifier qu’ils héritent de l’interface Corps.    4. Règles d’affaires alternatives       1. Si aucun corps n’existe, mettre fin aux calculs.    5. Tests d’acceptation       1. Imprimer dans un syso tous les corps récupérés et vérifier manuellement.    6. Post-conditions       1. Les corps à calculer sont déterminés.   Calculer la force résultante sur chaque corps.   * 1. Qui et temps   2. Préconditions      1. Une liste de corps est disponible.   3. Règles d’affaires      1. Dans une boucle foreach, itérer chaque corps (définit par x).         1. Si le corps n’est pas statique, itérer tous les autre corps (définit par y) dans une autre boucle foreach.         2. Pour chaque corps, calculer la distance séparant le corps x avec le corps y.         3. À l’aide de l’équation de la gravitation de Newton, calculer la force exercée par y sur x.   F = G \frac{m_1 m_2}{r^2}\   * + 1. Additionner toutes les forces calculées pour obtenir la force résultante.     2. À l’aide de la première loi de Newton (F=ma), calculer l’accélération du corps x.     3. En multipliant l’accélération par le temps écoulé, calculer la nouvelle vitesse du corps.   1. Règles d’affaires alternatives      1. Vérifier qu’un corps ne calcule pas la force exercée par lui-même (division par zéro).      2. Vérifier que les corps statiques n’ont pas de vitesse.   2. Tests d’acceptation      1. À l’aide de planètes placées aléatoirement, vérifier par syso que les vitesses des corps sont mises à jour de manières réalistes.   3. Post-conditions      1. Les corps ont leur nouvelle vitesse.   Calculer la nouvelle position des corps.   * 1. Qui et temps   2. Préconditions      1. Les corps ont leur nouvelle vitesse.   3. Règles d’affaires      1. Pour chaque corps, en multipliant le temps écoulé par leur vitesse, on obtient le changement de position.      2. En additionnant la position actuelle avec le changement de position, on obtient la nouvelle position des corps.   4. Règles d’affaires alternatives      1. S’assurer que les corps statiques ne changent pas de position.   5. Tests d’acceptation      1. Vérifier avec des planètes placées aléatoirement que les positions changent de manière réaliste.   6. Post-conditions      1. Les corps changent de position selon la physique. | |
| Tests d’acceptation | | | Mettre plusieurs corps à proximité et vérifier que leur comportement est réaliste grâce à des calculs fait à la main. | |
| Complexité | | | 3 | |
| Effort | | | 3 | |
| Commentaires | | |  | |
| **Scénario #7** | | | | |
| Acteurs | | | Utilisateur | |
| Scénario | | | En tant qu’utilisateur, je veux pouvoir voir les différents corps afin de pouvoir voir ce qui se passe. | |
| Description | | | 1. Charger la vue.   * 1. Qui et temps      1. J.S.      2. 2 hr.   2. Préconditions      1. Avoir rédigé les classes Vue et VueJeu.   3. Règles d’affaires      1. S’assurer que les fichiers FXML sont rédigés et chargeables.   4. Tests d’acceptation      1. Récupérer les FXML, et s’assurer que ce soit les bons.   5. Post-conditions      1. Nous avons accès aux deux vues de notre application.   2. Récupérer les différents nœuds à afficher.   * + 1. J.S.     2. 2hr.   1. Préconditions      1. Avoir accès aux classes métiers qui sont fonctionnelles.   2. Règles d’affaires      1. Charger le contrôleur.      2. Appeler la méthode retrouverCorps().      3. Vérifier que les corps héritent de Dessinable.   3. Règles d’affaires alternatives.   4. Tests d’acceptation   5. Vérifier que les corps sont dans la liste.   3. Afficher les nœuds.   * 1. Qui et temps      1. J.S.      2. 30 min.   2. Préconditions      1. Avoir accès à la liste de nœuds.   3. Règles d’affaires      1. Récupérer la liste de noeuds.      2. Setter le binding sur la position.      3. Ajouter les nœuds dans le gestionnaire de disposition.   4. Règles d’affaires alternatives   5. Tests d’acceptation      1. Vérifier que tout bouge correctement.   6. Post-conditions      1. Nous avons un système graphique à ses débuts. | |
| Tests d’acceptation | | | Démarrer le programme et vérifier que la fenêtre s’affiche. | |
| Complexité | | | 2 | |
| Effort | | | 1 | |
| Commentaires | | |  | |