

Activité intégrative 2024

Itération 1

Durée : 10 heures

Cette itération vise à créer le projet, lancer une partie, faire les premières actions et, enfin, abandonner.

Planification de l’itération 1

1 : Découvrir l’énoncé (40 min)

En grand groupe, le responsable présente le projet, le document des règles, les modalités d’évaluation, répond aux premières questions, etc.

2 : Créer et configurer le projet (10 min)

Les étudiants créent et configurent leur projet (voir ÉTAPE INITIALE :). Le responsable peut faire cette activité avec les étudiants.

3 : Concevoir l’itération 1 (60 min et plus)

Comme son nom l’indique, la section CONCEVOIR LA SOLUTION DE L’ITERATION 1 liste plusieurs responsabilités et classes candidates. Nous te proposons de réfléchir à la conception de l’application en trois étapes :

1. **Individuellement**, attribue les responsabilités à des candidats. En parallèle, identifie des responsabilités et des candidats manquants. [15 min]
2. **En petits groupes (3 à 5 étudiants)**, compare tes attributions ainsi que tes nouvelles responsabilités et classes candidates. Décline chaque responsabilité et collaborateur en méthodes et attributs. [20 min]
3. **En grand groupe**, deux petits groupes présentent les fruits de leurs réflexions aux autres groupes. Le but est d’évaluer les avantages et inconvénients des solutions proposées. [25 min]

4 : implémenter et tester l’application (8 heures – 480 min)

Nous te recommandons d’implémenter progressivement l’application, une US après l’autre. **Attention**, ton implémentation doit réussir au moins 8 des 10 tests d’acceptation pour que l’itération soit jugée recevable.

Pour cette première itération, nous t’indiquons une marche à suivre plutôt précise pour résoudre les difficultés que tu rencontreras. Les itérations suivantes te donneront plus de libertés.

Étape initiale : créer et configurer le projet

En tant que programmeur, je souhaite exécuter un squelette du programme afin de valider mon environnement de développement.

Architecture du projet et construction du système

Avant d'implémenter les fonctionnalités du programme, tu dois comprendre son architecture. L'architecture d'un programme est un ensemble de règles qui structurent son code source. En dotant ton programme d'une architecture et en veillant à sa cohérence, tu facilites sa prise en main par d'autres programmeurs, notamment ton titulaire de laboratoire 😊.

L'application à développer compte trois écrans responsables des interactions avec le joueur. Nous appellerons ces écrans des Vues. Leurs responsabilités sont limitées à afficher les données, à intercepter des événements utilisateurs de bas niveau comme « J'appuie sur la touche Enter » et à les traduire en événements utilisateurs de haut niveau, plus proches du domaine de l'application. Dans le contexte du menu principal, « appuyer sur la touche Enter » correspond à l'événement de haut niveau « Je veux exécuter l'item sélectionné ». Une vue délègue la gestion des événements de haut niveau à un superviseur.

Un superviseur réagit aux événements de haut niveau. Cela se traduit par la mise à jour des objets du domaine, puis de l'affichage. Note que le superviseur collabore avec sa vue à travers une interface, ce qui permet de substituer une implémentation d'une vue à une autre.

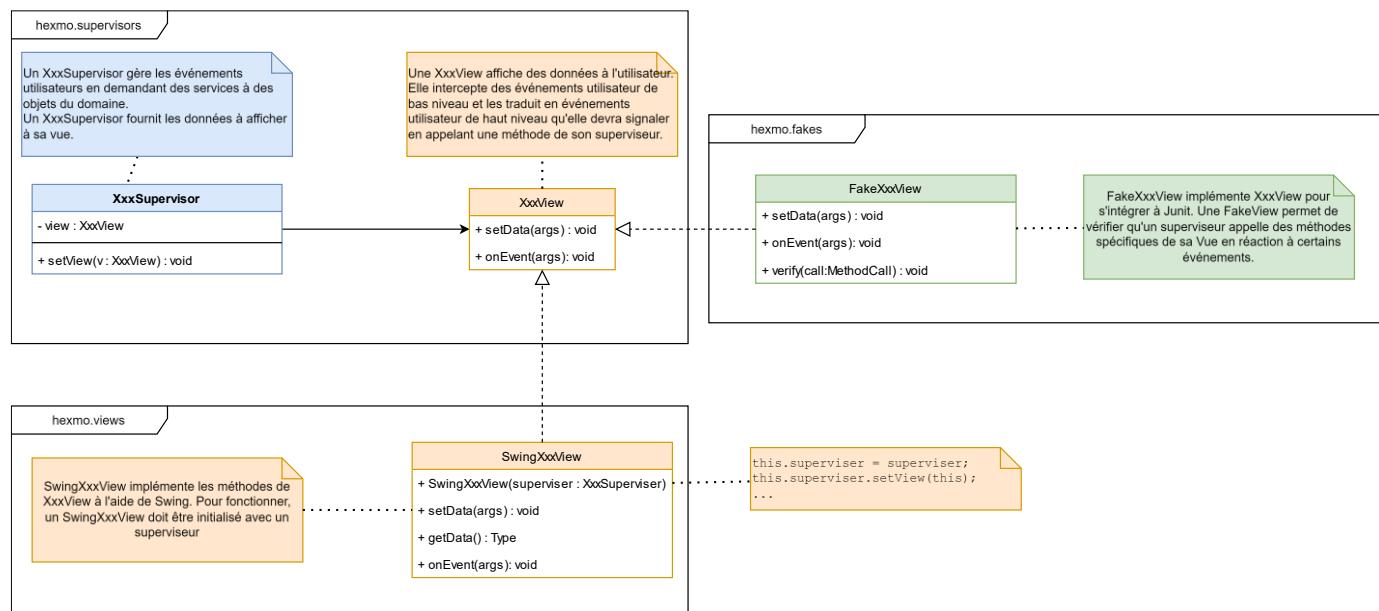


Figure 1 Vue statique présentant l'interface d'une vue, son implémentation et le superviseur qui interagit avec elle

Une Fenêtre principale héberge les vues et permet de naviguer entre elles. À cette fin, chaque vue possède un identifiant. Ces identifiants sont définis dans `ViewId`. Un superviseur demandera à sa vue de naviguer vers une autre vue en appelant la méthode `goTo(ViewId)`. La vue transmettra l'appel à la fenêtre principale qui fera alors la transition en appelant deux méthodes :

- `void onLeave(ViewId toView)` sur la vue qu'elle s'apprête à quitter.
- `void onEnter(ViewId fromView)` sur la vue qu'elle s'apprête à afficher.

Note : tu peux redéfinir ces méthodes dans tes superviseurs.

Chaque itération correspond à un triplet <superviseur, interface de la vue, implémentation en swing de la vue>. Quand tu construiras ton application, les vues des triplets seront à intégrer à une fenêtre principale Swing pour que l'application puisse naviguer vers elle. La fenêtre principale, les superviseurs et les vues concrètes sont définis dans la méthode `Program.main`.

Program
+ main(args):void

```
MainWindow window = new MainWindow("Ai 2024 - HexMo",
    new SwingMainMenuView("MainMenu", new MainMenuSupervisor(...)),
    new SwingPlayGameView("PlayGame", new PlayGameSupervisor(...))
);

window.start("MainMenu");
```

Figure 2 La méthode principale construit le système logiciel et affiche la fenêtre principale.

Les sections suivantes approfondissent les détails de l'architecture. À ce stade, nous souhaitons afficher cette fenêtre principale.

Création et configuration du projet (~10 min)

Télécharge l'archive [ai-2024-stub.zip](#), décomprime-la et importe-la dans Eclipse. Renomme le projet en `hexmo.<ton_nom>.<ton_prenom>`. Ce projet utilise la dernière mise à jour du plugin eclipse-pmd 3.6. Cette mise à jour utilise une nouvelle version majeure de PMD, incompatible avec les versions précédentes. Vous devez mettre à jour le plugin si vous souhaitez voir les alertes PMD.

Tu devrais obtenir une structure finale semblable à celle présentée par la figure suivante.



Exécute la classe `hexmo.Program`. Si tout va bien, elle affichera un écran analogue à la Figure 3.

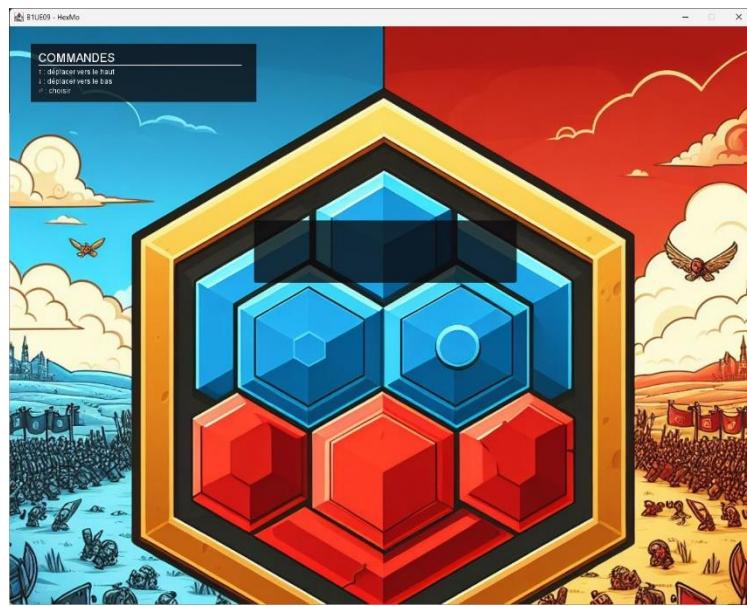


Figure 3 Données présentées par le squelette

À ce stade, nous pouvons présenter les fonctionnalités à implémenter.

User Stories

Vous devez implémenter trois récits utilisateur (*User Story, US*) qui permettront de créer une partie, de faire les premières actions, et de quitter l'application.

AI-1 : lancer une partie

En tant que joueur, je souhaite lancer une nouvelle partie afin de l'afficher à l'écran.

Détails

- L'application commence son exécution en affichant un menu principal composé de quatre items : « Nouvelle partie (r=3) », « Nouvelle partie (r=4) », « Nouvelle partie (r=5) » et « Quitter »
- L'exécution des items 1 à 3 crée une nouvelle partie. La création d'une partie commence par la création du plateau de jeu de rayon 3, 4 ou 5 selon l'item choisi. Les cases du plateau sont toutes vides.
- Une partie concerne deux joueurs. Tout joueur possède un nom, par exemple « P1 » et « P2 ». Chaque joueur se verra attribuer une couleur, qui pourra changer par la suite.
- La création se poursuit par l'attribution d'une couleur aux deux joueurs. Pour faciliter les tests, le premier joueur aura toujours la couleur rouge et le second joueur aura la couleur bleue.
- Une fois la partie créée, l'application navigue vers la vue PLAY_GAME.
- La vue PLAY_GAME présente l'état de la partie. Elle affiche un plateau et un panneau de messages.
 - Le plateau affiché aura un rayon correspondant à l'item choisi dans la vue MAIN_MENU.
 - Le panneau de messages présentera les joueurs et leurs couleurs, le joueur qui a la main et les données de la case active.

Tests d'acceptation

- ✓ Quand je lance l'application, celle-ci me présente un menu principal composé des quatre items.
- ✓ Quand je sélectionne l'item « Quitter », l'application termine son exécution.
- ✓ Soit x , un entier allant de 3 et 5. Quand je sélectionne l'item « Nouvelle partie (r= x) », l'application navigue vers la vue PLAY_GAME, affiche un plateau de rayon x , semblable à l'image ci-dessous, et les messages « P1 (rouge) vs P2 (bleu) », « Au tour de P1 (rouge) », « Case active (0,0,0) Libre ».

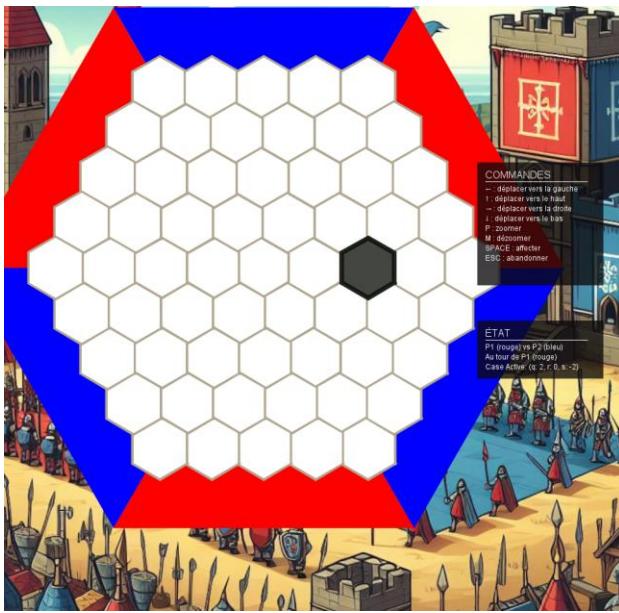


Figure 4 Plateau de jeu attendu

AI-2 : effectuer les premières actions

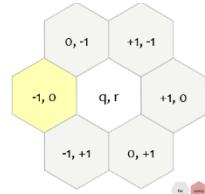
En tant que joueur, je souhaite exécuter les premières actions de jeu, notamment le « swapping », afin de démarrer la partie.

Détails

- Le joueur rouge peut se déplacer librement sur le plateau en appuyant sur les touches fléchées. Il choisit d'occuper la première case de son choix. **Pour cette itération, ignorez les règles relatives aux cases du bord.**
- Appuyer sur « espace » fait que la case active passe du statut « Libre » à « Rouge ».
- L'application détecte qu'il s'agit du premier coup et demande au joueur bleu s'il souhaite changer de couleur.
- En cas de réponse positive, le joueur rouge devient le joueur bleu et inversement. Le nouveau joueur bleu a la main pour choisir la prochaine case.

Tests d'acceptation

- ✓ Soit une partie démarrée, quand j'appuie sur les touches fléchées, la case active devient la case voisine correspondante selon les règles illustrées par la figure ci-contre.
- ✓ Soit une partie démarrée, et la case active de position $\langle q, r \rangle$, quand j'appuie sur « ESPACE », la case correspondante devient rouge et l'application demande au joueur bleu s'il souhaite changer de couleur.
- ✓ Soit une partie démarrée et la question de changement de couleur posée, quand le joueur bleu souhaite changer de couleur, alors l'application actualise le message « P1 (rouge) vs P2 (bleu) » qui devient « P1 (bleu) vs P2 (rouge) » et affiche le message « Au tour de P1 (bleu) ».
- ✓ Soit une partie démarrée et la question de changement de couleur posée, quand le joueur bleu ne souhaite pas changer de couleur, alors l'application affiche le message « P1 (rouge) vs P2 (bleu) » et « Au tour de P2 (bleu) ».



AI-3 : abandonner la partie

En tant que joueur, je peux abandonner une partie et revenir au menu principal, afin d'en relancer une nouvelle.

Détails

- Appuyer sur la touche « Esc » quand une partie est en cours permet de revenir à l'écran principal.
- Demander une nouvelle partie relance la création d'une nouvelle partie. **En particulier, la création d'un nouveau plateau.**

Tests d'acceptation

- ✓ Soit une partie lancée, quand j'appuie sur la touche « Esc », je reviens au menu principal.
- ✓ Soit x , un entier allant de 3 et 5. Soit une partie lancée avec un rayon x et une case occupée, quand je reviens au menu principal et que je relance une nouvelle partie de rayon x , alors l'application affiche un plateau de rayon x vide et les messages décrits dans l'US 1.
- ✓ Soit x et y , deux entiers compris entre 3 et 5, tels que $x \neq y$. Soit une partie lancée avec un rayon x et une case occupée, quand je reviens au menu principal et que je relance une nouvelle partie de rayon y , alors l'application affiche un plateau de rayon y vide et les messages décrits dans l'US 1.

Concevoir la solution de l'itération 1

Nous vous proposons une liste de responsabilités à attribuer à des candidats. Enrichissez ces listes avec de nouvelles responsabilités et de nouveaux candidats. Pour rappel, une responsabilité sera, par la suite, déclinée en méthodes à implémenter.

Attention, certaines responsabilités peuvent être affectées à plusieurs candidats.

Responsabilités

- Connait ses coordonnées
- Possède un nom
- Possède une couleur
- Actualise sa couleur
- Associe une coordonnée à une case
- Actualise son statut
- Permet de parcourir les cases qui la composent
- Crée une nouvelle partie
- Fournit la dernière partie créée
- Associe une composante q à une composante r
- Calcule sa composante s
- Additionne une instance à une autre instance
- Applique les règles du jeu à la carte et au joueur
- Énumère des couleurs

Classes candidates

- AxialCoordinate
- HexColor
- HexTile
- HexBoard
- HexGame
- Player
- HexGameFactory

Problèmes de conception à résoudre

Communication entre les superviseurs

L'architecture choisie conduit à des situations dans lesquelles les actions d'un superviseur ont des conséquences sur les autres. Par exemple, créer une nouvelle partie est une responsabilité du superviseur du menu principal. Cependant, le superviseur de la partie en cours doit obtenir la partie créée pour la contrôler et le superviseur de fin de partie devra la consulter pour calculer les résultats (voir itération 3).

Comment synchroniser ces objets sans introduire de dépendances mutuelles, avec le risque d'obtenir du code spaghetti ?

Pour la problématique de la création du jeu, une solution élégante consiste à déclarer une classe responsable de créer une nouvelle partie et de fournir la dernière partie créée : il s'agit d'une fabrique de jeu. Si les superviseurs partagent la même instance, alors ils peuvent se synchroniser avec elle dans les méthodes `onEnter(ViewId)` et `onLeave(ViewId)`. Idéalement, les superviseurs devraient collaborer avec cette fabrique à travers une interface.

La classe représentant la fabrique est à déclarer dans le paquetage hexmo.domains, comme le reste de vos types propres (voir contraintes techniques). Vous devez l'instancier dans la classe Program et la passer en argument des constructeurs des superviseurs.

Génération des cases du plateau de jeu

Le plateau de jeu possède des cases hexagonales, positionnées pour former un hexagone de rayon r . Rappelons que le rayon du plateau correspond à la distance séparant une case du bord de case centrale, de coordonnées axiales $<0,0>$.

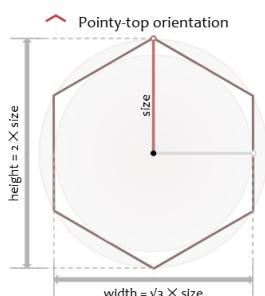
Vous pouvez facilement générer les coordonnées du plateau en faisant varier q et r de -rayon à +rayon à l'aide de deux boucles imbriquées. Toutefois, cette approche produit plus de coordonnées que nécessaires.

Une solution consiste à tirer parti du composant s des coordonnées axiales. En effet, les cases qui forment un plateau hexagonal respectent toutes une condition construite à partir de s et du rayon du plateau. Reste à trouver cette condition...

Conseil : faites un dessin pour la trouver.

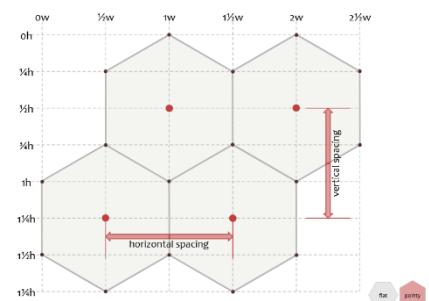
Conversion des coordonnées axiales en coordonnées affichables

Les coordonnées axiales sont commodes pour mesurer la distance séparant deux cases ou pour déterminer les cases voisines d'une autre. Toutefois, elles doivent être converties en coordonnées plus traditionnelles pour être affichées. **Vous devrez prendre en charge la première étape de cette conversion. Cette section explique comment passer de coordonnées axiales $<q,r>$ en coordonnées « pixel » $<x,y>$.**

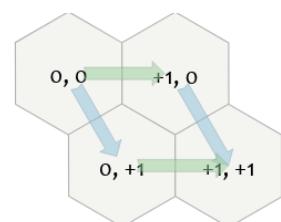


Cependant, la largeur et la hauteur d'un hexagone régulier ne sont pas égales. Elles respectent cependant des proportions bien connues, présentées par la figure ci-contre. Faisons l'hypothèse que $size$ vaut 1. On constate que la largeur vaut $\sqrt{3}$ tandis que la hauteur vaut 2.

Comme la largeur est différente de la hauteur, l'espacement entre deux hexagones varie selon l'axe. Plus précisément, l'espacement horizontal pour $size=1$ sera d'une largeur, donc $\sqrt{3}$ et l'espacement vertical vaudra $\frac{2}{3}$ de hauteur, ce qui se simplifie en $\frac{2}{3}\sqrt{3}$ si $size = 1$. Avec ces valeurs, nous pouvons passer de coordonnées axiales à des coordonnées en pixels.



La figure ci-contre présente les vecteurs de base des coordonnées axiales. Ces vecteurs sont $(1,0)$ et $(0,1)$. Pour exprimer ces vecteurs en coordonnées pixels, il faut substituer les composants correspondants en observant les décalages dans l'espace. $(1,0)$ devient $(\sqrt{3},0)$ et $(0,1)$ devient $(\sqrt{3}/2, 3/2)$



Nous pouvons passer enfin de coordonnées axiales (q,r) en coordonnées pixel (x,y) en calculant les expressions suivantes.

$$x = \sqrt{3} \times q + \frac{\sqrt{3}}{2} \times r$$

$$y = 0 + \frac{3}{2} \times r$$

Pour chaque coordonnée (q,r) du plateau de jeu, appelez la méthode `PlayGameView.setTileAt(float x, float y, TileType)` en fournissant en argument les coordonnées pixel (x,y) correspondantes.

Contraintes techniques et algorithmiques

Algorithmique

Répondez aux questions ci-dessous en commentaires dans votre code source. Pour chaque question, nous indiquons l'endroit où la réponse doit être écrite. Nous ne corrigerais que les réponses présentes à l'endroit demandé.

Invariants sur un plateau de jeu de forme hexagonale

Tous les plateaux hexagonaux de rayon r compteront le même nombre de cases. Ce nombre peut être calculé avec une formule que vous pouvez construire par récurrence.

Combien de cases compte un plateau de rayon 0 ?

Combien de cases compte un plateau de rayon 1 ? Exprimez votre réponse en faisant apparaître la réponse à la question précédente.

Combien de cases compte un plateau de rayon 2 ? Exprimez votre réponse en faisant apparaître la réponse à la question précédente.

...

Combien de cases compte un plateau de rayon r ? Essayez de simplifier votre formule en constatant qu'elle fait intervenir la somme des r premiers éléments d'une suite arithmétique.

It-1-q1 : écrivez votre raisonnement dans la JavaDoc de la classe hexmo.domains.HexBoard.

Par ailleurs, toutes les cases du plateau ont leur composants qui respecte une condition faisant intervenir le rayon. Quelle est cette condition ?

It-1-q2 : écrivez votre réponse et votre raisonnement dans la JavaDoc de hexmo.domains.HexBoard.

Choix de collection pour mémoriser les cases du plateau

Il est fortement conseillé de représenter le plateau par une collection. En effet, un tableau à deux dimensions risque d'avoir plusieurs éléments null, délicats à gérer.

Quelle interface de collection (`List`, `Queue`, `Deque`, `(Sorted)Set`, `(Sorted)Map`, etc.) allez-vous choisir pour représenter le plateau ? Justifiez votre choix en identifiant les principales opérations dont vous aurez besoin au cours de cette itération ? Avez-vous notamment besoin d'accéder à un élément précis ? Si oui, sur base de quelle sorte de clé ?

Quelle implémentation de collection (`ArrayList`, `LinkedList`, `ArrayDeque`, `HashSet`, `TreeMap`, etc.) allez-vous choisir mémoriser les cases ? Justifiez votre choix en déterminant les CTT des principales opérations que vous utiliserez pour l'itération 1.

It-1-q3 : répondez à cette question dans la Javadoc en-tête de la classe hexmo.domains.HexBoard.

CTT du dessin du plateau

Soit un plateau de rayon r composé de n cases. Si on part du principe que la méthode PlayGameView.setTileAt(float x, float y, TileType) fait le rendu en temps constant, quelle sera la CTT de la méthode de dessin du plateau complet ?

Pour répondre à cette question, examinez la partie de code concernée, et identifiez les boucles, les imbriques, mais aussi les collections utilisées et leurs opérations ou les appels de sous-méthodes ou de méthodes d'autres objets. Quand vous répondez, n'oubliez pas d'indiquer à quoi correspondent vos libellés N,M, etc.

It-1-q4 : répondez à cette question dans la Javadoc de hexmo.domains.PlayGameSupervisor.

Programmation Orientée Objet

Vous devrez créer de nouvelles classes modélisant le domaine : la phase de conception est consacrée à cette activité. Placez ces classes dans le paquetage hexmo.domains. Attention, **Les types du paquetage seront indépendants des paquetages hexmo.supervisors et treasurequest.views.**

Vous devrez également implémenter les méthodes du MainMenuSupervisor et les méthodes de la classe PlayGameSupervisor. Attention, **vous ne pouvez pas changer les signatures des méthodes publiques de ces classes**, sous peine de « casser » le programme.

Vous devez respecter les principes et appliquer les pratiques utilisés pendant les leçons de POO : encapsulation correcte (pensez aux copies défensives), classes timides, attributs `private`, une utilisation correcte des attributs et méthodes de classe. Écrivez des méthodes courtes et peu complexes et faites bon usage du polymorphisme, des interfaces, etc. PMD vous alertera de la plupart des infractions, mais il n'est pas parfait...

Testez au minimum la classe HexGame de sorte à couvrir 90% des branches du package hexmo.domains. Prévoyez des cas de test pour les éléments suivants :

- Créer une partie de rayon 3
- Créer une partie de rayon 4
- Créer une partie de rayon 5
- Occuper la première case pour le joueur actif à une position correcte
- Occuper la première case pour le joueur actif à une position qui ne correspond à aucune case du plateau
- Faire le « swapping »
- Tenter plusieurs « swapping ».

N'hésitez pas à ajouter des classes de tests supplémentaires en respectant les conventions de Junit 5. Veillez enfin à ce que tous vos tests réussissent pour l'itération 1.