Activité intégrative 2024

Itération 2

Durée : 10 heures

Cette itération vise à jouer une partie, sans détecter la fin, et à assister les joueurs.

Planification de l'itération 2

1 : Découvrir l'énoncé (20 min)

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

2 : Patcher le projet (10 min)

L'assistance des joueurs passe par une légère modification du code existant.

Modifiez le fichier `hexmo.supervisors.commons.TileType` pour y ajouter la valeur `HIGHLIGHT`. Modifiez ensuite le dictionnaire `TILES` défini dans `hexmo.views.Theme`, ligne 51 comme présenté ci-dessous.

```
        /**
         * Définit la couleur des cases
         * */
        public static final Map<TileType, Color> TILES = Map.of(
                        TileType.UNKNOWN, UNKNOWN,
                        TileType.RED, RED,
                        TileType.BLUE, BLUE,
                        TileType.HIGHLIGHT, SECONDARY_COLOR
        );
```

Avec ces changements, vous pourrez appeler la méthode `PlayGameView.setTileAt(float, float, TileType)` et fournir `TileType.HIGHLIGHT` comme argument.

3 : Concevoir l'itération 2 (60 min et plus)

Comme son nom l'indique, la section sur la conception 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]

3 : implémenter et tester l'application (8 h 30)

Nous te recommandons d'implémenter progressivement l'application, une US après l'autre. **Attention,** ton implémentation doit réussir au moins 10 des 12 tests d'acceptation pour que l'itération soit jugée recevable.

User Stories

Vous devez implémenter trois *US* qui permettront de jouer un tour, de valider l'occupation d'une case et d'assister le joueur. La détection de la fin de la partie sera faite pendant l'itération 3.

Important

Les tests présentés ci-après constituent un échantillon, forcément incomplet, des cas de figure possibles. Nous nous réservons le droit d'adapter les conditions de départ et les coordonnées des cases jouées.

AI-4 : jouer plusieurs tours

En tant que joueur, je souhaite jouer un tour pour passer la main à mon adversaire.

Détails

- Après avoir occupé une première case et effectué le « swapping », les joueurs occupent tour à tour les cases du plateau.
- Le joueur qui a la main peut se déplacer et occuper une case libre.
- Occuper une case, c'est la colorer dans la couleur du joueur actif.
- Si un joueur essaie d'occuper une case déjà prise, l'application affiche un message d'erreur, mais laisse la main au joueur.
- Une fois que le joueur actif occupe une case, il passe la main au joueur suivant.

Tests d'acceptation

- ✓ Soit une partie de rayon quelconque commencée,
  - Quand le joueur rouge occupe la case (q = 0 ; r = 2),
  - Que le joueur bleu refuse le swapping et qu'il occupe la case (-2 ;0),
  - Que le joueur rouge occupe la case (-2 ;2),
  - Alors les cases (0 ;2) et (-2 ;2) sont rouges, la case (-2 ;0) est bleue et les cases restantes sont vides.
- ✓ Soit une partie de rayon quelconque commencée,
  - Quand le joueur rouge occupe la case (-2 ;0),
  - Que le joueur bleu accepte le swapping,
  - Que le nouveau joueur bleu occupe la case (-1 ;0),
  - Que le joueur rouge occupe la case (-1 ;2),
  - Alors les cases (-2 ;0) et (-1 ;2) sont rouges, la case (-1 ;0) est bleue et les cases restantes sont vides.
- ✓ Soit une partie de rayon quelconque commencée,
  - Quand le joueur rouge occupe la case (-2 ;0),
  - Que le joueur bleu refuse le swapping,
  - Que le nouveau joueur bleu essaie d'occuper la case (-2 ;0),
  - Alors un message d'erreur « Case déjà occupée » apparait, la case (-2 ;0) est rouge et les cases restantes sont vides.
- ✓ Soit une partie de rayon quelconque commencée,
  - Quand le joueur rouge occupe la case (0; 1),
  - Que le joueur bleu accepte le swapping,
  - Que le nouveau joueur bleu occupe la case (-2 ;0),
  - Que le joueur rouge occupe la case (-2 ;1),
  - Que le nouveau joueur bleu occupe la case (-2 ;1),

- Alors un message d'erreur « Case déjà occupée » apparait, les cases (0 ;1) et (-2 ;1) sont rouges, la case (-2 ;0) est bleue et les cases restantes sont vides.

AI-5 : valider l'occupation d'une case

En tant que joueur, je souhaite que mon adversaire n'occupe pas certaines cases du bord pour maximiser mes chances de victoire.

Détails

- En plus des cases déjà occupées, un joueur ne peut pas occuper des cases du bord selon sa couleur.
- Pour un plateau de rayon *r*,
  - Les cases du bord sont celles dont la distance par rapport à l'origine vaut *r*.
  - Le joueur rouge peut occuper les cases du bord dont les coordonnées ont un composant égal à *r*.
  - Le joueur bleu peut occuper les cases du bord dont les coordonnées ont un composant égal à *-r*.
- Les cases du bord formant un coin présentent la particularité d'avoir un composant égal à *r* et un autre égal à *-r* : les deux joueurs peuvent l'occuper.
- Les coordonnées qui iraient au-delà des bords du plateau devraient être ignorées. En conséquence, l'application ne devrait pas dessiner une case active située en dehors du plateau.

Tests d'acceptation

- ✓ Soit une partie de rayon 3 en cours où le joueur actif est le joueur bleu. Si j'essaie d'occuper la case (3; -1) vide, ou toute autre case vide située sur un bord « rouge », alors l'application affiche le message d'erreur « La case est incompatible avec votre couleur » et affiche toujours le message « Au tour de P# (bleu) ».
- ✓ Soit une partie de rayon 3 en cours où le joueur actif est le joueur rouge. Si j'essaie d'occuper la case (-3 ;1) vide, ou toute autre case vide située sur un bord « bleu », alors l'application affiche le message d'erreur « La case est incompatible avec votre couleur » et affiche toujours le message « Au tour de P# (rouge) ».
- ✓ Soit une partie de rayon 3 en cours où le joueur actif est le joueur rouge. Si j'essaie d'occuper la case (-3 ; 3) vide, ou toute autre case vide située sur un coin, l'application accepte l'occupation et affiche le message « Au tour de P# (bleu) ».
- ✓ Soit une partie de rayon 3 en cours où le joueur actif est le joueur rouge et la case active est la case (3, -3). Si j'essaie de me déplacer sur la case (-4 ; -3) inexistante, ou toute autre case inexistante, alors l'application ignore mon déplacement.

AI-6 : assister le joueur

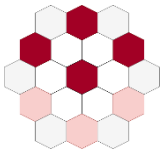
En tant que joueur, je veux connaitre les coups intéressants pour s'assurer la victoire.

Détails

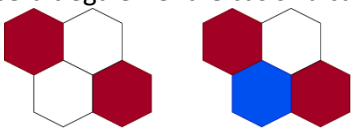
- Parmi les tactiques à appliquer, celui appelé « le pont » consiste à former un motif où deux cases de même couleur partagent **deux cases voisines libres**, comme illustré ci-contre. Ce motif est intéressant, car le joueur pourra connecter ses deux cases à son prochain tour, même si son adversaire occupe une des deux cases.

- Le joueur qui a la main doit visualiser les cases voisines libres, en jaune sur l'image ci-contre.

- Pour une case *C* occupée, les cases candidates sont obtenues à partir des cases diagonales à *C* qui sont dans la même couleur. Sur l'image ci-contre, les diagonales de la case centrale sont présentées en rouge. Celles en rouge pâle sont des diagonales potentielles, mais libres. Les cases grises ne sont pas des diagonales.



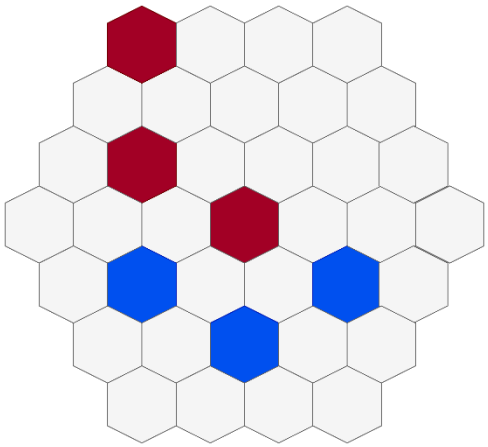
- *C* et une de ses diagonales forment un pont si leurs cases voisines communes sont toutes les deux libres. Sur l'image suivante, seules les quatre cases de gauche forment un pont. Les cases de droite ne forment pas un pont, car une case voisine est occupée par l'adversaire. Ce serait également le cas si la case était rouge.



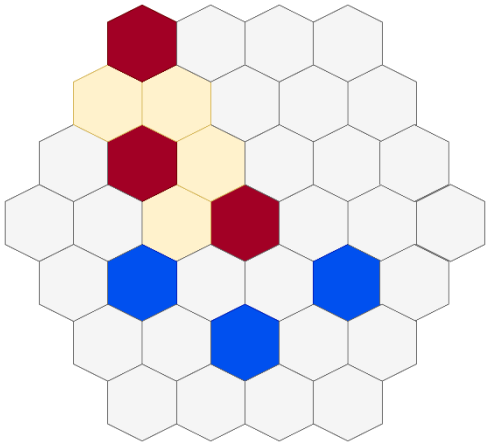
- Vous trouverez une proposition d'algorithme pour détecter les ponts dans la suite de cet énoncé.

Tests d'acceptation

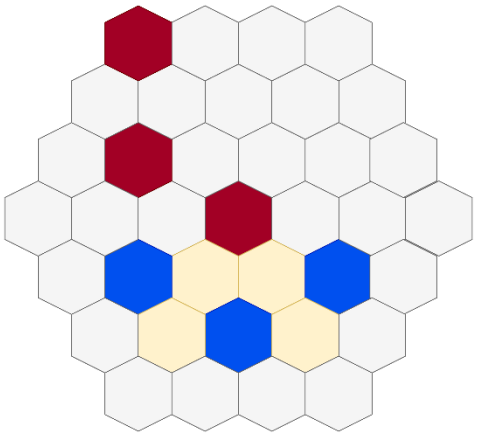
Soit la situation de jeu illustrée par la figure ci-dessous.



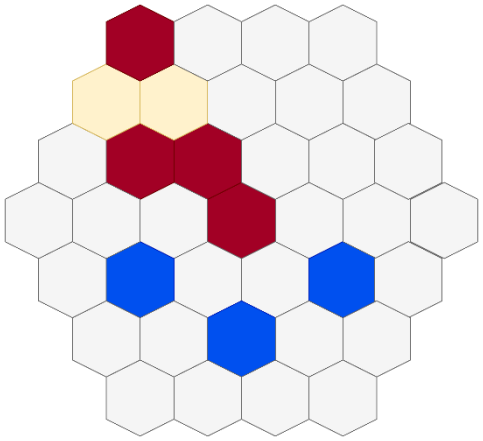
✓ Quand le joueur actif est le joueur rouge, alors l'application met en évidence les cases ci-dessous.



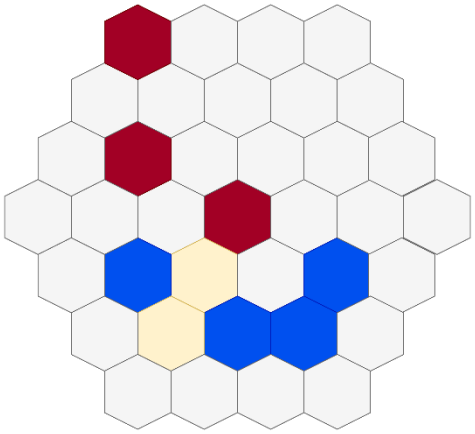
✓ Quand le joueur actif est le joueur bleu, alors l'application doit met en évidence les cases-dessous



- ✓ Si le joueur rouge décide d'occuper la case (0 ; -1), quand il reprend la main, alors l'application met en évidence les cases suivantes.



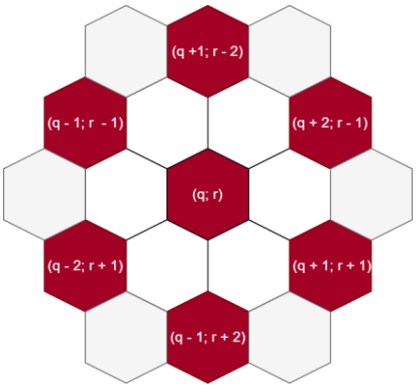
- ✓ Si le joueur bleu décide d'occuper la case (0 ; 2), quand il reprend la main, alors l'application met en évidence les cases suivantes.



Un algorithme pour trouver les cases libres formant un pont

Voici la description de ce que devrait faire votre algorithme pour produire la collection des cases libres formant un pont. L'algorithme reçoit un plateau P et une couleur C. Il retourne une collection de cases ou de coordonnées où chaque case est libre et intervient dans un pont.

Soit la collection des cases du plateau de la couleur souhaitée. Pour chaque case de cette collection, vous devez trouver ses diagonales sur le plateau. Les diagonales d'une coordonnée correspondent aux coordonnées suivantes.

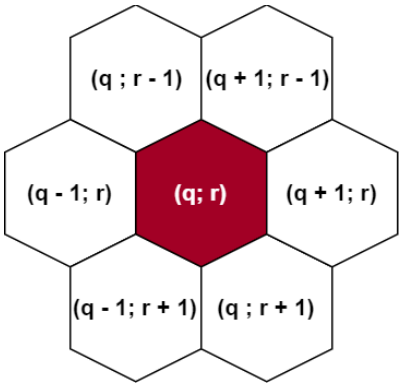


Attention

Éliminez les diagonales inutiles telles que celles qui sont inoccupées et celles qui ne font pas partie du plateau.

Pour chaque diagonale restante, trouvez les voisins communs à la case inspectée. Si tous les voisins communs sont libres, alors vous êtes en présence d'un pont : ajoutez les voisins communs à votre résultat.

Les voisins d'une coordonnée correspondent aux coordonnées suivantes.



Concevoir la solution de l'itération 2

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.

Responsabilités

- Connait son état d'occupation
- Détermine ses diagonales
- Détermine ses voisins
- Filtre les coordonnées qui n'appartiennent pas au plateau
- Filtre les coordonnées sur base d'une couleur
- Décide si une case du bord est occupable par une couleur
- Décide du joueur qui a la main
- Génère une collection de cases à mettre en évidence
- Détecte les cases libres formant un pont pour le joueur actif

Classes candidates

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

Contraintes techniques et algorithmiques

Algorithmique

Condition respectée par les cases formant un pont

Écrivez la condition que doit respecter toutes les cases libres formant un pont. En particulier, que pouvez-vous affirmer des cases voisines d'une telle case ?

It-2-q1 : écrivez votre raisonnement dans la Javadoc de la classe hexmo.domains.HexBoard.

Condition d'une case du bord jouable

Quelle condition une case du bord doit-elle remplir pour qu'un joueur puisse l'occuper ? Soyez suffisamment général pour tenir compte des cases formant un coin.

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

Choix de collection pour mémoriser les cases formant un pont

Étant donné l'algorithme décrit, quelle collection convient le mieux au stockage des case formant un pont ?

Quelle interface de collection (`List, Queue, Deque, (Sorted)Set, (Sorted)Map, etc.`) allez-vous choisir ? Justifiez votre choix en identifiant les principales opérations dont vous aurez besoin au cours de cette itération ? Avez-vous, par exemple, 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 ? Justifiez votre choix en déterminant les CTT des principales opérations que vous utiliserez pour l'itération 1.

Si vous n'utilisez pas de collection spécifique et que vous jugez qu'une collection déjà présente suffit, justifiez votre choix, tant du point de vue de l'interface que de l'implémentation.

It-2-q3 : répondez à cette question dans la Javadoc en-tête de la classe hexmo.domains.HexBoard.

CTT de l'algorithme pour trouver les cases libres formant un pont

Quelle est la complexité temporelle théorique de l'algorithme décrit ?

Pour répondre à cette question, examinez la partie de code concernée, et identifiez les boucles, les imbrications, 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-2-q4 : répondez à cette question dans la Javadoc de hexmo.supervisors.PlayGameSupervisor.

Programmation Orientée Objet

Vous devrez modifier les classes modélisant le domaine **:** la phase de conception vous permettra d'identifier les classes à adapter. Attention, **Les types du paquetage seront indépendants des paquetages `hexmo.supervisors` et `helmo.views`.**

Vous devrez également adapter la méthode dessinant le plateau du `PlayGameSupervisor`. Cette dernière devra mettre en évidence (to highlight) les cases libres formant un pont. Attention, **vous ne pouvez pas changer les signatures des méthodes publiques du superviseur et de sa vue**, 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…

Écrivez des tests de sorte à couvrir 90% des branches du package `hexmo.domains`. Testez en particulier la méthode qui génère les tuiles formant un pont. Pensez notamment aux cas de tests suivants :

- Détecter un pont entre deux cases dont une est située au bord du plateau.
- Détecter l'absence de ponts quand le plateau ne compte qu'une seule case occupée.

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 2.