Nom : _____

Prénom : _____

Identifiant : _____ Groupe : _____ Enseignant : _____

/20

Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

juin 2024
DEV2
Equipe dev2

DEV2 – Développement II

Examen

Examen – dev2

Consignes

1. L'examen dure 1h45 et se fait sur une machine du réseau pédagogique.
2. Vous n'avez pas accès à l'Internet
3. Vous pouvez utiliser vos fichiers disponibles localement et vos notes papier.
4. Avant de commencer, clonez le dépôt (sur Gitesi) qui a été créé pour vous.
5. Tous les fichiers devront être placés dans le clone, créé ci-dessus.
6. Lorsque vous avez fini, créez un commit contenant vos fichiers. Depuis le clone :
   - ▷ `git status`
   - ▷ `git add .`
   - ▷ `git commit -m "remise examen"` (La première commande sert à vérifier que tout est là, et rien d'autre.)
7. Poussez le commit vers le dépôt (sur Gitesi) avec `git push`

Vous allez implémenter le début d'un jeu de bataille navale. Il s'agit d'un jeu de société ou deux joueurs vont placer secrètement des bateaux de dimensions variées sur leur plateau de jeu. Les bateaux peuvent être placés horizontalement ou verticalement. Une fois que les navires sont placés, les joueurs prennent tour à tour des tirs sur le plateau de l'adversaire en essayant de toucher et de couler les navires ennemis. Le jeu continue jusqu'à ce que tous les navires d'un joueur soient coulés, celui-ci perd alors la partie et l'autre joueur la gagne. Lors de cet examen, vous n'implémenterez pas le jeu complet, mais uniquement jusqu'au placement des bateaux sur un plateau de jeux.

Créez un projet nommé Examen2024 dans votre clone, et créez un package nommé `g12345.examen` (où vous remplacez 12345 par votre matricule). Tous les autres packages seront placés dans celui-ci.

Version du document
15 mai 2024

1 Le modèle

1.1 Création des classes

1 Record et énumérations (1 point)

Créez un package nommé `model`, dans celui-ci créez :

- ▷ un `record Position(int x, int y)`, celui-ci représente une position sur un plateau de jeu ;
- ▷ une énumération `Direction` avec comme constantes : `HORIZONTAL` et `VERTICAL` ;
- ▷ une énumération `ShipType` avec comme constantes : `BASE` et `CARRIER` ; cette énumération représente les différents types de bateaux.

2 La classe Ship (1 point)

Dans le package `model` créez une classe `Ship`. Cette classe aura pour arguments :

- ▷ un tableau de positions `Postion[] positions`, représentant les positions occupées par le navire sur un plateau de jeu ;

Générez les accesseurs pour les différents attributs. Les attributs n'ont pas encore de valeur, c'est normal, en pratique, nous ne créerons pas d'objet `Ship`, mais plutôt ses enfants (que vous allez construire juste après). C'est dans ceux-ci que nous initialiserons les attributs.

3 Les classes Carrier et Base (2 points)

Dans le package `model`, créez deux classes : `Base` et `Carrier`. Elles héritent de la classe `Ship`. Pour chacune d'elles, créez un nouveau constructeur qui aura pour rôle d'attribuer des valeurs aux attributs. Faites ceci, en respectant les formes des différents bateaux :

- ▷ la `Base`, est une pièce carrée occupant 4 positions, elle correspond donc à un bateau 2x2. Son constructeur prend un paramètre `Position p`. Le constructeur initialise `positions` comme un tableau de quatre `Position`. Le premier élément de `positions` vaut la `Position p` donnée en paramètre et représente la position en haut à gauche de la `Base`. Les autres éléments de `positions` correspondent aux positions à côté de celle-ci pour former un carré ;
- ▷ le `Carrier`, est une pièce en longueur occupant 4 positions, elle correspond donc à un bateau 1x4. Le constructeur prend un paramètre `Position p` et un paramètre `Direction d`. Le constructeur, initialise `positions` comme un tableau de quatre `Position`. Le premier élément de `positions` vaut la `Position p` donnée en paramètre. Si la `Direction` est `Horizontal`, les autres éléments de `positions` correspondent aux positions horizontales suivante. Si la `Direction` est `Vertical`, les autres éléments de `positions` correspondent aux positions verticales suivantes ;

La dernière question et l'image en fin d'énoncé illustrent la forme des bateaux en fonction des positions passées en paramètre.

4 La classe Board (1 point)

Dans le package `model` créez une classe `Board`, qui représente le plateau de jeu d'un joueur. Cette classe aura pour arguments :

- ▷ un entier `width` représentant la largeur du plateau ;
- ▷ un entier `height`, représentant la hauteur du plateau ;
- ▷ une liste de `Ship` nommée `Ships`. Cet argument représente la liste des bateaux d'un joueur.

Ajoutez un constructeur qui prend `int height` et `int width` en arguments et attribue ces valeurs aux attributs correspondants. Il initialise `Ships` en une liste vide. Ajoutez des accesseurs pour tous les attributs de cette classe.

1.2 Placement des bateaux

5 La méthode validPosition (2 points)

Nous voudrions maintenant ajouter une méthode dans la classe `Board` pour ajouter de nouveaux bateaux à la liste `ships`. Pour cela, il faudra vérifier si les positions auxquelles nous voulons ajouter nos bateaux sont bien valides. C'est-à-dire que les positions sont bien sur le plateau de jeu et qu'aucun autre bateau n'occupe l'une de ces positions. Nous allons créer deux méthodes dans la classe `Ship` pour vérifier cela.

Créez une première méthode `boolean positionInBoard(int width, int height)` dans la classe `Ship`. Les arguments `width` et `height` sont les dimensions du plateau de jeu. Cette méthode vérifie que tous les éléments de l'attribut `Positions` sont bien compris dans le plateau (le plateau commence à la position (0,0)). La méthode retourne `true` si les positions sont bien dans le tableau, `false` sinon.

6 La méthode overlap (2 points)

Créez une méthode `boolean overlaps(Ship s)` dans la classe `Ship`. Cette méthode vérifie qu'aucun élément de `Positions` (du bateau actuel) ne se superpose avec un élément de `Positions` du bateau `Ship s` donné en paramètre. La méthode retourne `false` s'il n'y a pas de superposition, `true` sinon.

7 Les tests (2 points)

Créez deux tests pour la méthode `overlaps`, l'un pour vérifier qu'elle retourne `true` en cas de superposition, l'autre pour vérifier qu'elle retourne `false` quand il n'y a pas de superposition. Nous vous conseillons d'initialiser les tableaux `positions` manuellement dans les tests.

8 La méthode addShip (2 points)

Créez une nouvelle méthode `void addShip(Position p, Direction d, ShipType shipType)` dans la classe `Board`. Cette méthode crée un nouveau bateau de la classe correspondant à `shipType`, en la position `p` et avec la direction `d`. Cette méthode utilise ensuite une des méthodes précédentes pour savoir si le bateau créé à bien des positions valides, dans le cas contraire, il lance une erreur. Elle vérifie ensuite en utilisant une des méthodes précédentes que le bateau nouvellement créé ne se superpose pas avec un autre bateau de l'attribut `ships`. Dans le cas où il y a une telle superposition, la méthode retourne une erreur. Enfin, la méthode ajoute le bateau créé à l'attribut `ships`.

9 La méthode addShip surcharge (1 point)

Effectuez ensuite une surcharge de la méthode `addShip` avec cette fois comme argument uniquement `Position p` et `ShipType shipType`. Cette méthode appelle la méthode `addShip` originelle, en fixant le paramètre `Direction` à `Direction.HORIZONTAL`.

10 Les visibilités (2 points)

Vérifiez que la visibilité de tous vos attributs et de vos méthodes est adéquatement choisie (`public`, `private`, etc).

2 La vue

11 La classe View (2 points)

Créez une package `view`, dans ce package créez une classe `View`. Cette classe contient une méthode `public static void displayFullBoard(Board board)`. Cette méthode affiche le plateau de jeu donné en paramètre avec tous les navires présents sur celui-ci. Les navires sont représentés par des "X", et l'eau par des "." comme illustré sur l'image ci-dessous.

3 Le contrôleur

12 La classe Controller (2 points)

Créez une package `controller`, dans ce package créez une classe `Controller`. Dans cette classe, créez une méthode `main`, et créez un `Board board` à l'intérieur de celui-ci. Ajoutez dans ce `board` :

- une `Base` à la position $(0, 0)$;
- un `Carrier` placé horizontalement à la position $(4, 3)$;
- un second `Carrier` placé verticalement à la position $(6, 6)$.

Affichez ensuite le `board` grâce à la méthode `displayFullBoard` de la classe `View`. Exécutez votre `main`. Vous devriez obtenir un résultat similaire à l'image ci-dessous :