Université du Québec
École de technologie supérieure

Service des enseignements généraux

Auteur : Mathieu Nayrolles, Pierre Bélisle

Révision:

Fred Simard (A2021)

INF111, Travail pratique #3

Programmation Orientée-objet

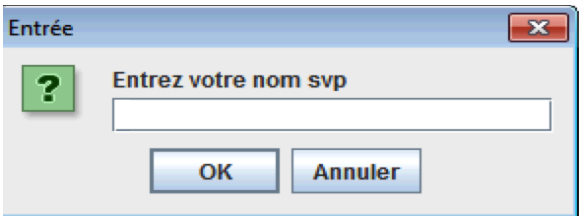
(équipe de 4, max)

1. Description générale du travail

Ce travail recupère le même sujet que pour le travail pratique numéro 2, mais est, en fait, un travail complètement indépendant. Il s'agit de programmer un jeu de bataille navale qui permet à un joueur de jouer contre l'ordinateur. Vous utiliserez des modules existants et devrez produire les classes manquantes.

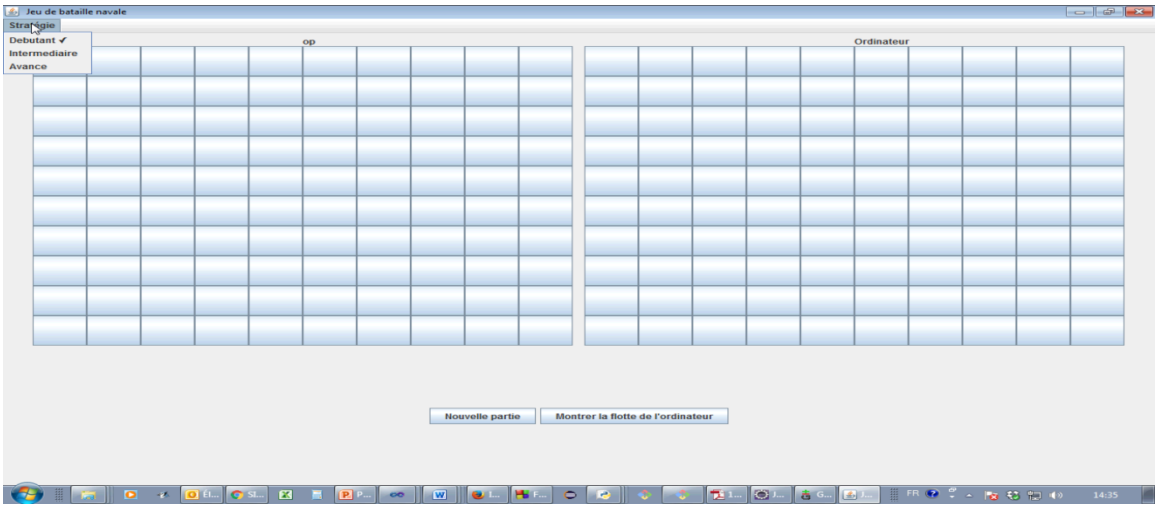
1.1 Déroulement du jeu

Le programme demande d'abord le nom du joueur (JOptionPane).

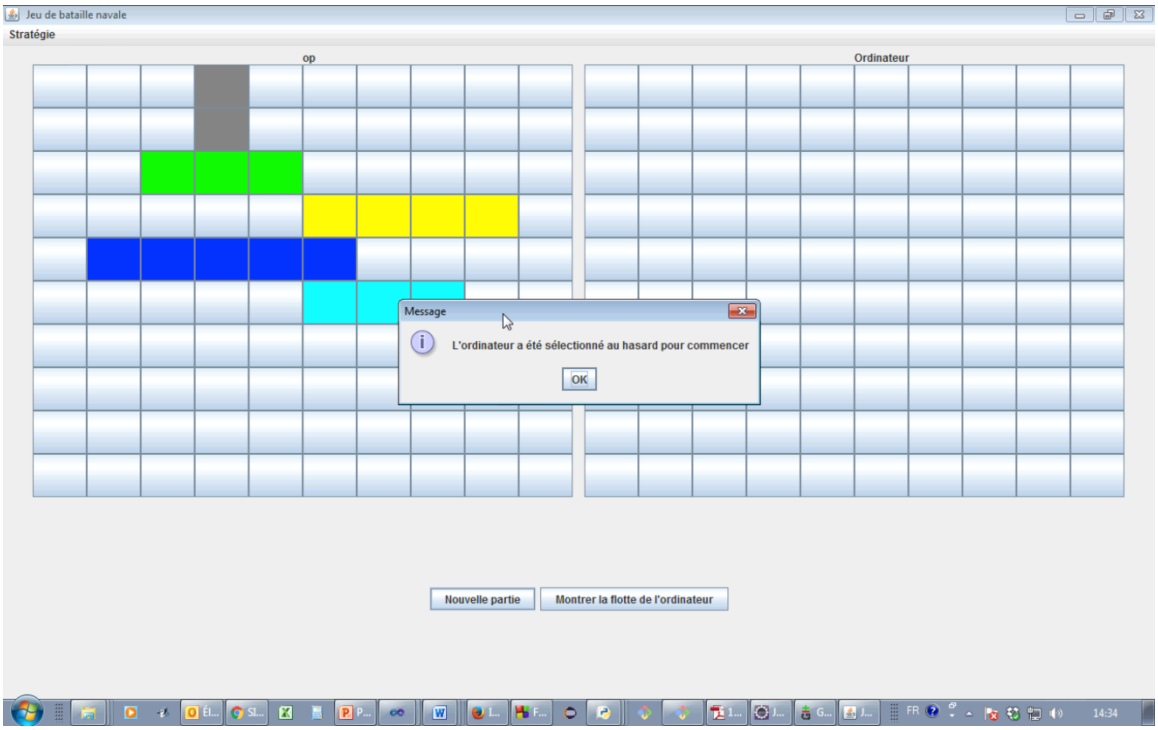


Pour des raisons de temps, le joueur n'aura pas à placer ses navires. Leurs positions seront générées aléatoirement. Il choisit ensuite la stratégie de l'ordinateur[1] et clique sur un bouton pour démarrer la partie. Le choix par défaut est la stratégie: débutant. Si la stratégie change en cours de partie, la partie actuelle est annulée et le joueur doit cliquer sur nouvelle partie pour en commencer une autre.

[1] Seules les stratégies débutant à avancé sont considérées dans ce devoir..


Jeu de bataille navale
Stratégie
Debutant ✓
Intermediaire
Avance
op
Ordinateur
Nouvelle partie
Montrer la flotte de l'ordinateur

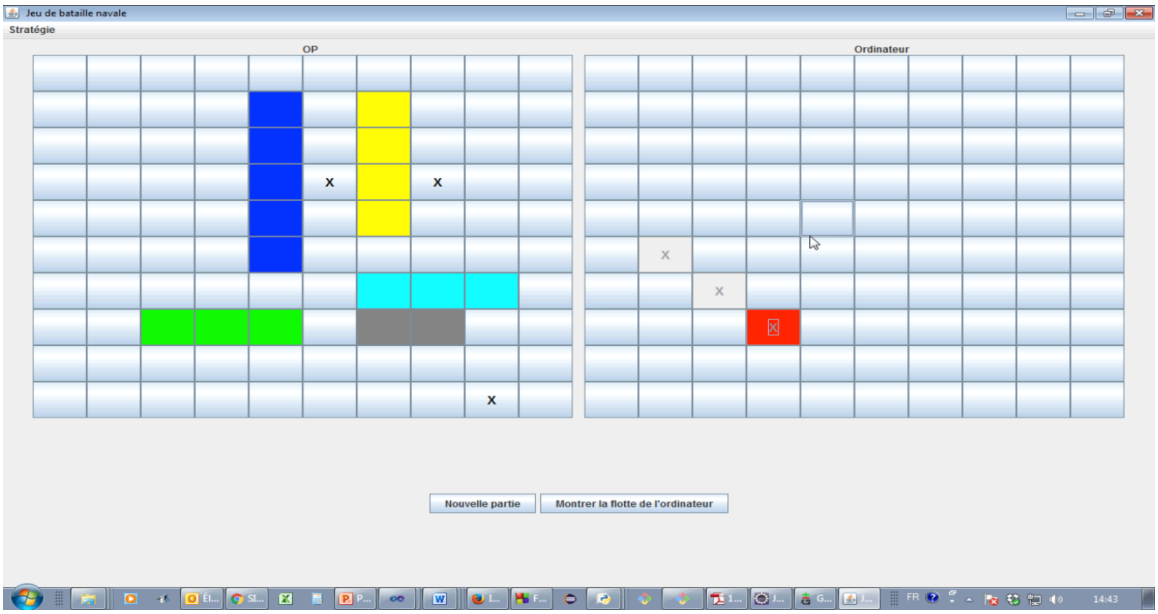
Le premier à jouer est choisi au hasard.



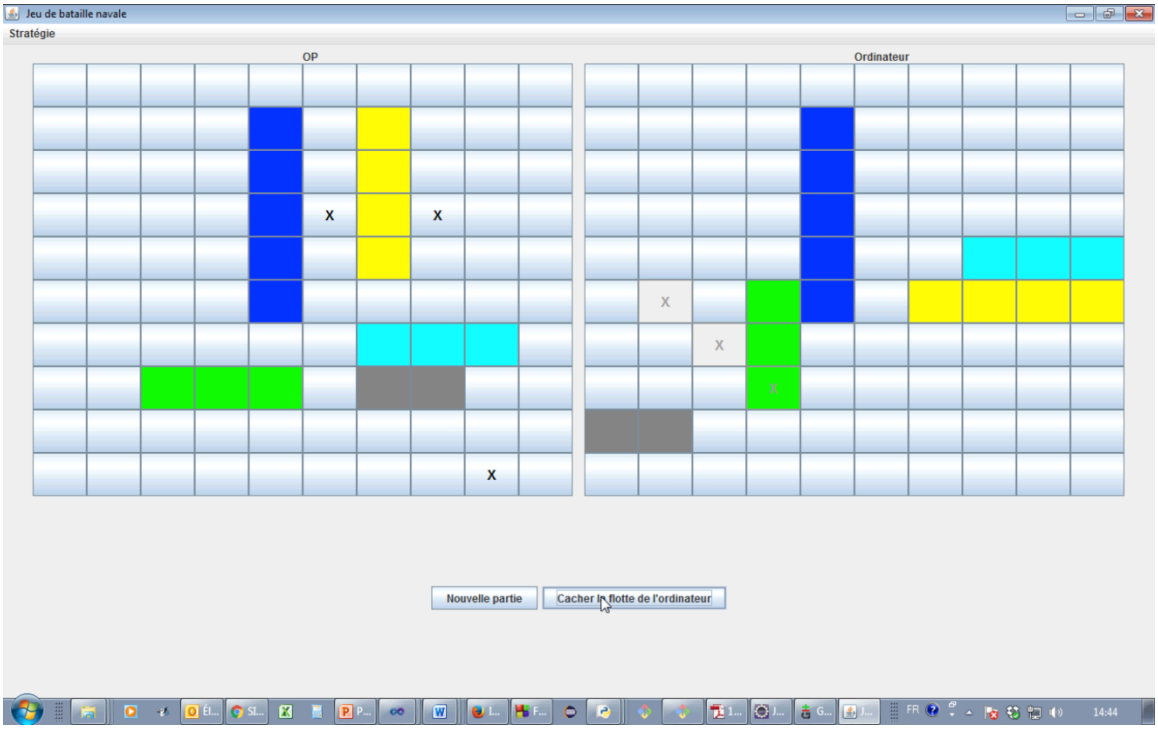
En cliquant sur le bouton 'OK' la partie commence. Si c'est le joueur qui commence, il clique sur une des cases de la flotte de l'ordinateur et l'ordinateur joue son coup. Ensuite, les rôles s'alternent jusqu'à:

- la victoire d'un des deux joueur
- un changement de stratégie
- un clic sur le bouton de nouvelle partie.

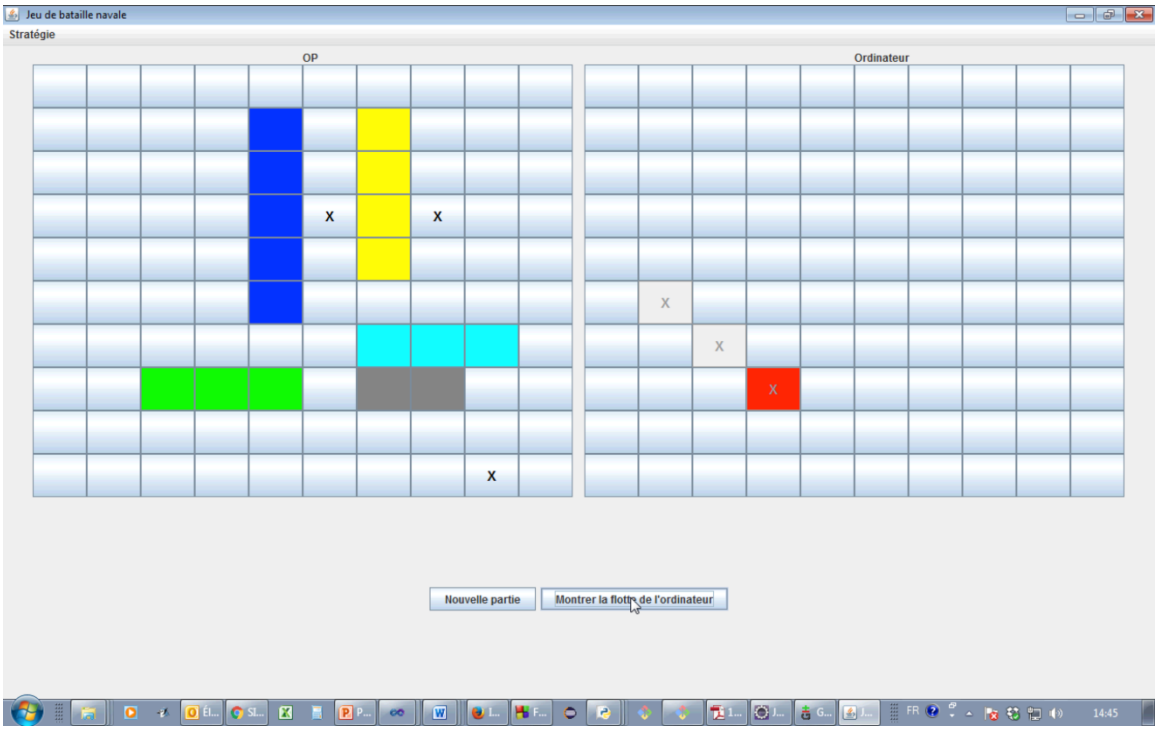
Les cases jouées par le joueur sont désactivées pour l'empêcher de sélectionner la même case deux fois.



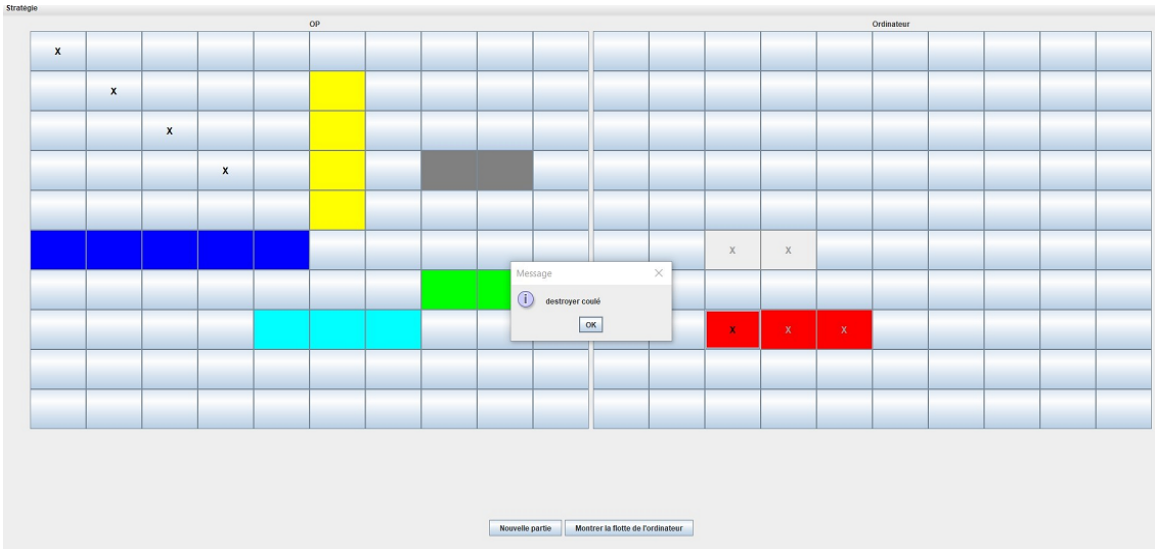
Le second bouton sert essentiellement au débogage. Un clic sur ce bouton entraîne l'affichage de la flotte de l'ordinateur dans la grille de l'ordinateur. Le texte du bouton est alors modifié.



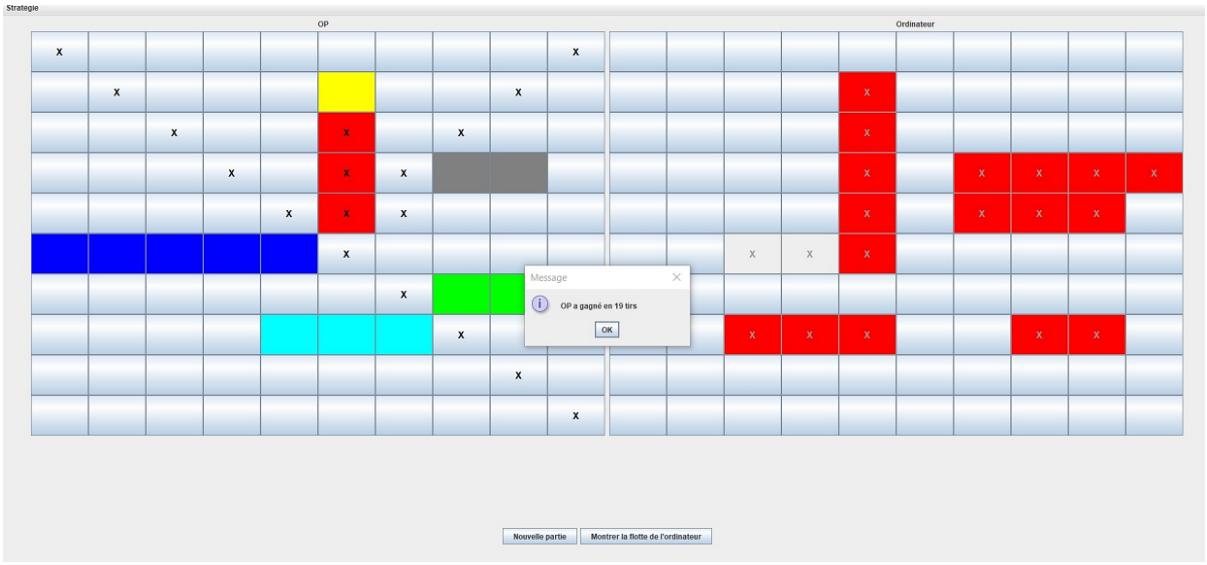
Un second clic cache la flotte et le texte du bouton est ajusté à nouveau..



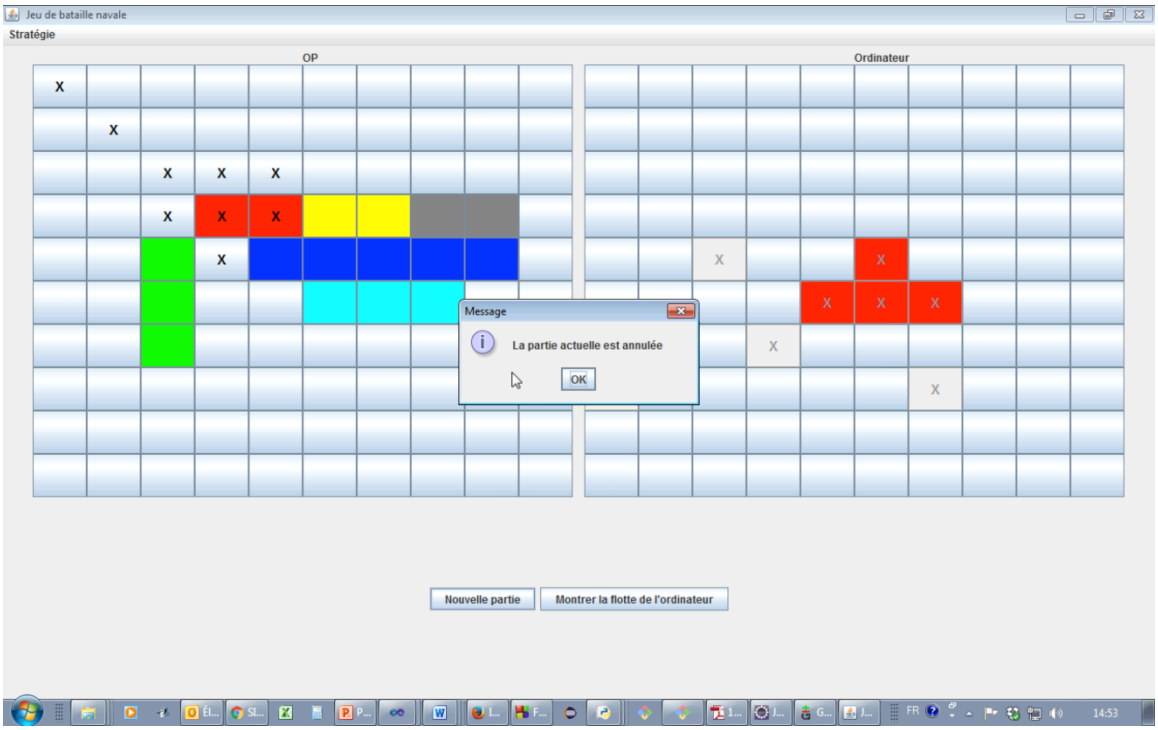
Un message apparaît lorsqu'un navire de la flotte de l'ordinateur ou celle du joueur est coulée. Ce message contient le nom du navire.



Un message apparaît lorsque la partie est terminée en affichant le gagnant et le nombre de tirs. En cliquant sur OK, il est possible de démarrer une nouvelle partie avec le bouton prévu.


Strategie
OP
Ordinateur
Message
OP a gagné en 19 tirs
OK
Nouvelle partie
Montrer la flotte de l'ordinateur

Un message d'avertissement apparaît s'il y a un clic sur le bouton de nouvelle partie en cours de partie. En cliquant sur 'OK', une nouvelle partie débute.

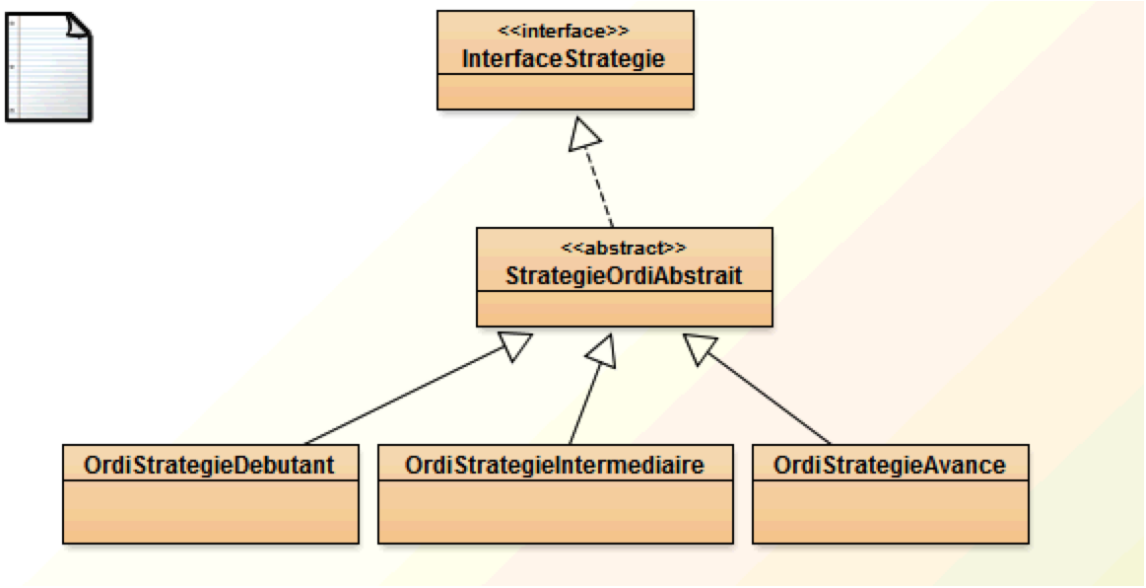


2. Conception imposée

Vous êtes les programmeurs et nous sommes les concepteurs. Nous vous demandons de programmer notre conception que vous découvrirez au fur et à mesure du développement des différentes étapes. Pour le bon déroulement du laboratoire, veuillez respecter les consignes à la lettre.

Il s'agit de créer une hiérarchie de classe pour implémenter les stratégies de l'ordinateur, réduire les sélections de type et la répétition de code. Pour cette version du jeu, nous implémentons seulement des stratégies qui retiennent les tirs qu'ils ont joués.

Nous vous fournissons les trois classes qui comprennent des stratégies fonctionnelles que vous devez modifier pour répondre à la hiérarchie décrite par le diagramme suivant:



2.1 Les interfaces et type abstrait

2.1.1 InterfaceParticipant

Dans le but d'éviter la répétition du code qui s'occupe de la flotte de chacun des participants (le joueur et l'ordi), nous vous avons déclaré, dans une interface, les méthodes que vous devez utiliser. La classe **Participant** implémente cette interface et les classes **Joueur** et **Ordi** héritent de la classe Participant.

2.1.2 InterfaceStrategie

Vous devez écrire l'interface qui déclare les méthodes *getTir* et *aviserTouche* du tp2 (les mêmes). Vous y ajoutez ensuite la déclaration d'une méthode qui vide la collection des tirs joués:

```
public void resetTirsJoues();
```

2.1.3 StrategieOrdiAbstrait

Tout le code qui s'occupe de la collection des tirs joués se retrouve dans cette classe abstraite qui implémente **InterfaceStrategie**. Les trois classes de stratégies concrètes fournies devront en hériter.

Le seul attribut de cette classe est la collection de **Coord**. Voici les méthodes à écrire.

`Coord obtenirTirPasDejaJoue();`

- Délègue cette tâche à l'UtilitaireCollection en lui passant la collection des tirs joués.

`void ajouterTir(Coord) ;`

- Ajoute un tir au début de la collection des tirs joués.

`boolean tirDejaJoue(Coord tir);`

- Délègue cette tâche à l'UtilitaireCollection en lui passant la collection des tirs joués et le tir reçu.

Coord getDernierTir() ;

- Retourne le dernier tir qui a été ajouté dans la collection des tirs joués.

`void resetTirsJoues() ;`

- Respecte une partie du contrat de l'interface et vide la collection des tirs joués.

2.1.4 TesteLesStrategies

Nous vous fournissons un programme pour tester le fonctionnement de votre hiérarchie de stratégies. Au fur et à mesure que les classes concrètes de stratégie sont bien écrites, vous verrez ce programme tirez selon une stratégie choisie au hasard.

Observez : Le polymorphisme dynamique est utilisé au moment d'afficher le tir avec JOptionPane.

2.2 Les classes Participant, Joueur et Ordi

2.2.1 La classe Participant

Elle regroupe le seul attribut commun au joueur et à l'ordinateur c'est la flotte. La classe Participant implémente **InterfaceParticipant** et vous est fournie ainsi que la classe **Flotte**.

2.2.2 La classe Joueur

La classe **Joueur** vous est fournie. Un joueur a un nom et une flotte qu'elle hérite de la classe Participant. Seules les méthodes nécessaires au projet ont été écrites.

2.2.3 La classe Ordi

Vous devez écrire cette classe et elle doit hériter de **InterfaceOrdi**. Au constructeur, la flotte est générée aléatoirement par la classe parent et la stratégie par défaut est celle de débutant.

3. Description générale de la partie graphique

Cette section est la plus imposante puisqu'elle met en place tout ce qu'il faut pour jouer correctement au jeu de bataille navale, dans l'environnement GUI, décrite plus haut.

À la fin de cette section, vous serez en mesure de jouer contre l'ordinateur mais vous ne pourrez pas encore choisir la stratégie par le menu. L'ordinateur jouera donc selon la stratégie qui est instanciée dans son constructeur (rappel: il s'agit de **StrategieOrdiDebutant**), néanmoins vous pouvez changer manuellement la stratégie afin de les tester dans votre nouvel environnement.

Pour la réalisation de ce segment, vous devez utiliser les classes fournies et écrire ou compléter le code manquant.

3.1 Cadre et panneau principal

Nous avons conçu l'application pour utiliser une fenêtre (**JFrame**) graphique démarrée dans un processus séparé de l'EDT pour une bonne gestion par le système d'exploitation (**SwingUtilities** dans le main). Ensuite, nous voulons que le panneau

de contenu (contentPane) de la fenêtre soit remplacé par un objet de la classe **PanneauPrincipal**.
Ce panneau de la famille des **JPanel**, contient quatre attributs. Un panneau en haut pour montrer éventuellement les flottes et un panneau en bas qui contient les boutons pour démarrer une nouvelle partie et montrer la flotte de l'ordinateur. Une instance de la classe **Joueur** et une instance de la classe **Ordi**.
***Écrivez l'implémentation de cette étape AVANT de lire la suite. Vous devez avoir un **JFrame** et trois **JPanel** (**PanneauPrincipal**, **PanneauHaut** et **PanneauBas**) avec leurs attributs et constructeurs.

3.2 PanneauHaut

Le panneau du haut a été conçu sans intelligence. Il ne fait qu'offrir les méthodes pour gérer l'affichage et la saisie du clic du joueur. Le panneau du bas est prévu pour s'occuper du déroulement du jeu. C'est dans ce dernier que la détection des tirs touchés s'effectue et la détection des navires coulés (décrites plus loin).

Nous vous fournissons une classe **PanneauGrilleGui** qui a le même fonctionnement que **GrilleGUI**, des travaux précédents, mais elle hérite (par extension) de **JPanel** et contient quelques méthodes de plus qui vous seront utiles plus loin.

3.2.1 Les attributs

Vous devez composer le panneau du haut avec deux instances de **PanneauGrilleGui** pour montrer les flottes et une troisième instance qui est utilisé pour facilement montrer et cacher la flotte de l'ordinateur (décrit plus loin).

Comme vous pourrez le constater dans la classe **PanneauPrincipal**, le constructeur du panneau du haut reçoit une instance de chaque participant (Joueur et Ordi).
Donc au total cinq attributs dans cette classe.

3.2.2 Le constructeur

Retient simplement les deux paramètres reçus et appelle une méthode pour initialiser les composants.

Initialiser les composants
Il faut calculer la taille des **panneauGrilleGui** en regard à la taille de l'écran que vous obtenez par :

```
Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
```

- Nous voulons les trois instances de largeur identique et la taille du panneau du haut à 60% de la hauteur totale de l'écran.
- Le nom des participants doit apparaître au-dessus de leur panneau respectif.
- N'oubliez pas d'ajouter le panneau de la flotte du joueur et celui de la flotte de l'ordi au panneau du haut (pas la copie).

3.2.3 Accesseurs du joueur et de l'ordi

Le panneau du bas a besoin des références sur les participants. Écrivez les accesseurs getJoueur() et getOrdi().

***Écrivez l'implémentation de 3.1.2 à 3.1.3 AVANT de lire la suite, vous devriez voir les grilles avec les noms au-dessus.

3.2.4 Tests

À partir d'ici, vous pouvez utiliser le panneau du bas pour tester votre panneau du haut. Il est possible de développer les deux classes en même temps. Ici, je vous suggère de poursuivre votre lecture pour comprendre le panneau du bas et son utilisation du panneau du haut. Ensuite, vous pouvez implémenter et tester vos méthodes une à une. Par exemple, vous pouvez commencer par implémenter l'écouteur du bouton qui montre la flotte de l'ordinateur, ensuite tester votre méthode qui cache la flotte qui sont liés avec le bouton du bas.

3.2.5 Les méthodes de saisie du tir du joueur

Comme pour le travail précédent, il faut obtenir la position du clic que le joueur effectue. Comme c'est le panneau du bas qui s'en occupe, il faut que le panneau du haut l'informe. Écrivez getTirJoueur() qui retourne la position du clic dans la panneau du joueur et estClique() qui retourne simplement s'il y a eu un clic dans le panneau du joueur. Cela se passera dans l'écouteur de bouton du bas évidemment.

3.2.6 Les méthodes utiles au déroulement du jeu

Voici la liste des méthodes nécessaires au panneau du bas pour que l'affichage soit possible dans le panneau du haut. On vous demande de les écrire.

- Afficher un tir dans le panneau du joueur (setValeur(Constantes.TOUCHE))
- Afficher un tir dans le panneau de l'ordinateur
    - Tout ce qui modifie le panneau de l'ordinateur doit modifier la copie.
- Montrer qu'une case a été touchée dans le panneau du joueur (fond rouge).
- Montrer qu'une case a été touchée dans le panneau de l'ordinateur (dans la copie aussi).
- Montrer la flotte du joueur (UtilitaireGrilleGui)
- Montrer la flotte de l'ordinateur (UtilitaireGrilleGui)
- Cacher la flotte. Il s'agit de remettre le panneau de l'ordinateur dans le même état
  que la copie qui a été maintenue à jour.
- Rendre indisponible une case du panneau de l'ordinateur (et de la copie).
- Il faut permettre de réinitialiser le panneau d'ordinateur, remettre estClique à faux, et réactiver les cases du panneau de l'ordinateur et de sa copie.

- Une méthode publique doit permettre de remettre le panneau du haut dans l'état d'une nouvelle partie. Les deux flottes sont alors (re)générées aléatoirement. La flotte du joueur est montrée mais pas celle de l'ordinateur.

3.3 PanneauBas

Le panneau du bas a été conçu pour recevoir la référence sur le panneau du haut. Le sous-programme qui gère le déroulement du jeu est implémenté ici. Lorsqu'un des participants effectue son tir, les bonnes méthodes du panneau du haut pour afficher l'état des tirs et des navires touchés sont appelées. La gestion du tour de rôle est également implémentée dans ce sous-programme.

3.3.1 Les attributs

Vous devez composer le panneau du bas avec les deux boutons (nouvelle partie et montrer/cacher la flotte). Vous devez aussi retenir, dans un attribut, quel est le joueur qui doit jouer à ce tour (joueur ou ordinateur). Finalement, un petit attribut booléen qui détermine si la partie est en cours.

3.3.2 Le constructeur

Retient simplement la référence du panneau du haut reçue et appelle une méthode pour initialiser les composants.

Initialiser les composantes

- Nous voulons le panneau du bas à 40% de la hauteur totale de l'écran.
- Les boutons peuvent être disposés selon votre bon vouloir.
- La description des écouteurs de bouton suit dans le texte.

N'oubliez pas d'ajouter les boutons au panneau du bas.

3.3.3 Écouteur du bouton qui montre la flotte

Vous devez associer un écouteur au clic d'un des boutons pour qu'il montre (ou cache) la flotte de l'ordinateur. Le texte du bouton doit être ajusté en conséquence (décrit dans la première étape du travail). Vous pouvez utiliser cet écouteur pour tester vos méthodes du panneau du haut implémentés.

Astuce: Vous pouvez utiliser le texte du bouton pour savoir si la flotte est montrée ou non. Sinon, vous pouvez ajouter un attribut booléen.

3.3.4 Écouteur du bouton qui démarre une partie

C'est ici que tout le déroulement du jeu se produit. Dû au fonctionnement de Java en ce qui concerne les composants graphiques (EDT), nous devons mettre la boucle de jeu dans un processus séparé (Thread). Autrement, les rafraîchissements d'écran entre chaque tir ne se feront pas.
Voici ce qui doit être réalisé dans la méthode actionPerformed :

Début

- Si la partie est en cours, vous avisez qu'elle est annulée (voir énoncé partie 1).

- Initialiser la partie dans le panneau du haut à l'aide de la méthode prévue qui génère les deux flottes.

- Sélectionner quel participant commence la partie en premier. Le joueur ou l'ordi? (un nombre aléatoire avec **UtilitaireFonctions**). Vous avisez l'utilisateur de celui qui a été choisi et vous le retenez pour se souvenir c'est à qui le tour.

- Il faut créer un objet qui implémente l'interface Runnable. Dans la méthode run(), vous écrivez le code qui permet de jouer. Vous envoyez une référence de ce code à une instance de la classe Thread avant de le démarrer avec start (comme dans le main avec le GUI).

  Voici la description en pseudocode de ce qui doit être écrit dans cette méthode et comment le démarrer. Tout l'algorithme est écrit ici mais cela ne vous dispense pas de découper en sous-programmes.

```
Runnable code = new Runnable(){

    Définissez deux entiers pour retenir le nombre de tirs de chaque participant.

    public void run(){

        Obtenir les références sur le joueur et l'ordi du panneau du haut.
        Tant que le jeu n'est pas terminé pour aucun des deux participants
            Si c'est au tour du joueur
                Si le joueur a cliqué sur le panneau du joueur en haut
                    Compter un tir pour le joueur.
                    Obtenir le tir du panneau du haut.

                    Afficher le tir dans la flotte de l'ordinateur.

                   Si le tir a touché la flotte de l'ordinateur

                        Afficher que le tir touché

                        Afficher si le navire est coulé
                    Fin
```

Désactiver la case.

Le tour est à l'ordi.

Fin si le joueur a cliqué

Sinon, (c'est au tour de l'ordinateur)

On obtient le tir selon la stratégie du joueur***

Afficher le tir dans le panneau du joueur

Si le tir touche à la flotte du joueur

Afficher que le tir a touché.

Aviser l'ordinateur que le tir a touché pour qu'il ajuste sa stratégie ***

Un tir de plus pour l'ordinateur.

C'est au tour du joueur à jouer.

Fin si

Fin du premier si

Fin de la boucle de jeu

Afficher qui a gagné et en combien de tirs.

}

}

```
Threat t = new Thread(code);
t.start();
```

Fin de actionPerformed

Vous remarquerez que les algorithmes de jeu entre le joueur et l'ordi sont presqu'identiques. Vous aurez donc 2 sous-programmes très ressemblants. Pas besoin de les fusionner pour en n'avoir qu'un avec paramètre. Nous acceptons cette répétition de code.

Notez : Il faut prendre conscience que la boucle roule en parallèle jusqu'à la fin de la partie mais elle est interrompue de temps à autre pour rafraîchir l'écran. Si nous n'utilisons pas de **Thread**, l'affichage aurait lieu seulement lorsque la boucle est

terminée. Les requêtes de rafraîchissement des composants Swing étant dans la file de L'EDT[2].

Observez : C'est ici que le polymorphisme dynamique mis en place avec la stratégie de l'ordinateur est utilisé. Remarquez qu'il n'y a aucune sélection qui est nécessaire pour obtenir le tir (getTir) et aviser que le tir a touché (aviserTouche) comme il était nécessaire dans le programme principal du tp2.

3.4 Barre de menu

Cette partie est la plus petite du travail mais nécessaire pour permettre de changer la stratégie de l'ordinateur et utiliser la conception que nous avons mis en place dans les pages précédentes. Nous voulons que vous expérimentiez l'héritage par extension de la classe **JMenuBar** dans sa propre classe (pas une classe interne). Vous devez modifier la barre de menu (setJMenuBar) de la classe **CadreBatailleNavale** et la remplacer par une instance de la vôtre.

Pour permettre de modifier la stratégie du panneau principal dans votre classe, il faut que vous lui envoyiez sa référence. Vous pouvez écrire votre constructeur qui reçoit une instance de la classe **PanneauPrincipal** et qui la conserve pour pouvoir modifier la stratégie (setStrategie), à partir du ou des écouteurs, associé(s) au(x) items de menu. Un écouteur doit simplement créer la bonne instance de stratégie et la passer au panneau principal selon la stratégie choisie au menu.

Résumé des étapes :

- Créez une classe qui hérite de JMenuBar et qui reçoit le panneauPrincipal.
- Créer et ajouter les trois items au menu.
- Écrire le(s) écouteur(s) pour les items de menu[3]
- Ajouter le menu à la barre.
- Remplacer la barre de menu du cadre par le vôtre dans **CadreBatailleNavale**.

3.5 PanneauPrincipal

Dans setStrategie, si ce n'est pas déjà fait, vous devez modifier la stratégie de l'ordinateur et recréer le panneau (initComposants).

4. Critères de notations

Statégies (20%) :

- Implémentation d'interface et héritage par extension
- Conception des stratégies (le test doit fonctionner pour les trois stratégies).

[2] http://gfx.developpez.com/tutoriel/java/swing/swing-threading/

[3] Nous espérons que notre enseignement vous sert maintenant à ne pas écrire les trois écouteurs d'un coup et à les tester après. Un à la fois svp.

- La classe Ordi.
- Votre programme teste la collection de tirs joués.

GUI : (75%) :

- Introduction à la programmation avec Swing
- PanneauPrincipal
- CadreBatailleNavale
- PanneauHaut
- PanneauBas
- Gestion du jeu

Menu : (5%) :

- Introduction à la gestion de menu
- Menu de stratégies (affichage)
- Changement de stratégie (écouteurs de menu)

60% sont sur la qualité du code (bonnes pratiques, code optimisé (temps, mémoire et code)) et 40% sur l'exécution et le respect des consignes.

Bon travail !