Atelier Pro 4

Avril-Mai 2021 – Bataille Navale

[Le contexte 2](#_Toc69928441)

[Les règles 2](#_Toc69928442)

[Déroulement du jeu 2](#_Toc69928443)

[Début de la partie 2](#_Toc69928444)

[Mode 1 joueur 2](#_Toc69928445)

[Mode 2 joueurs 2](#_Toc69928446)

[Phase d’attaque 2](#_Toc69928447)

[La fin du jeu 3](#_Toc69928448)

[Instructions 4](#_Toc69928449)

[La classe Grid 4](#_Toc69928450)

[Les attributs 4](#_Toc69928451)

[Les méthodes 4](#_Toc69928452)

[La classe Player 5](#_Toc69928453)

[Les attributs 5](#_Toc69928454)

[Les méthodes 5](#_Toc69928455)

[La classe Game 5](#_Toc69928456)

[Les attributs 5](#_Toc69928457)

[Les méthodes 5](#_Toc69928458)

[La classe Battleship 6](#_Toc69928459)

[Améliorer le jeu 7](#_Toc69928460)

[Coulé 7](#_Toc69928461)

[Intelligence artificielle 7](#_Toc69928462)

# Le contexte

Développeur au sein de la société SIOGames, dans une équipe de 5 personnes, vous participez à la création d’un jeu de bataille navale sur ordinateur.

Pour l’instant en mode texte, le jeu sera plus tard proposé aux joueurs avec une interface graphique. Il sera proposé un mode 2 joueurs, où les joueurs se relaieront devant l’écran et un mode 1 joueur pour lequel l’adversaire sera l’ordinateur. L’ordinateur aura une difficulté réglable (l’algorithme utilisé pour l’attaque sera plus intelligent pour une difficulté plus haute).

# Les règles

La bataille navale est un jeu à deux joueurs où chacun doit placer des navires sur une grille tenue secrète et tenter de toucher les navires adverses en énonçant une coordonnée pour désigner une case de la grille adverse. Si cette case contient un navire, alors on dit « touché », sinon « raté ». Un navire est « coulé » si toutes les cases qui le contiennent ont été touchées.

Deux modes sont disponibles

* Les mode 1 joueur où vous jouerez contre l’ordinateur
* Le mode 2 joueurs où vous jouerez contre une deuxième personne, vous devrez vous relayer devant l’écran.

# Déroulement du jeu

## Début de la partie

Le joueur 1 saisit la **coordonnée de départ** de chacun de ses bateaux avec leur **direction**. Un bateau de taille 4 en **A5** avec une direction **verticale** finira en A9.

Chaque joueur doit placer un total de 7 bateaux.

Voici leurs tailles : 5, 4, 3, 2, 2, 1, 1

Chaque bateau est désigné par une coordonnée de départ (colonne, ligne) et une direction (verticale, horizontale).

### Mode 1 joueur

Le joueur robot a un programme qui génère aléatoirement l’emplacement de ses bateaux.

### Mode 2 joueurs

Le joueur 2 fait de même que le joueur 1.

## Phase d’attaque

Une fois les deux grilles remplies, la phase d’attaque commence.

Le joueur 1 commence et doit saisir une coordonnée.

Si son attaque a réussi, le joueur peut rejouer. Sinon, c’est au tour de l’autre joueur d’attaquer.

La grille des attaques apparaît avant la saisie de la coordonnée afin d’aider le joueur à prendre sa décision.

En mode 1 joueur, la réponse du robot est immédiate.

## La fin du jeu

Le jeu est fini lorsque l’un des deux joueurs a coulé tous les bateaux de l’autre joueur.

Un bateau est coulé lorsque toutes ses cases ont été touchées.

# Instructions

Dans un premier temps, vous développerez le mode 2 joueurs du jeu de Bataille Navale.

## La classe Grid

### Les attributs

Une matrice de 10x10 entiers.

### Les méthodes

*getValue* renvoie la valeur de la matrice à la coordonnée en paramètre.

**public** **int** getValue(**int** column, **int** line)

*initMat* initialise les cases de la matrice à la valeur 0.

**private** **void** initMat()

*randomInit* initialise la grille avec des bateaux aux positions aléatoires. Fait appel à la méthode *isValidShip* avant de placer le bateau avec l’appel à la méthode *placeShip*.

**public** **void** randomInit()

*addShot* met à jour la grille d’attaque avec la valeur 1 si c’est touché (success est vrai) ou avec la valeur 4 si c’est raté (success est faux) à la coordonnée en paramètre.

**public** **void** addShot(**int** column, **int** line, **boolean** success)

*isValidShip* renvoie true si le bateau peut être inséré à cette coordonnée.

**private** **boolean** isValidShip(int c, int l, **int** size, **int** dir)

*addNewShip* vérifie si le bateau peut être inséré avec *isValidShip* puis fait appel à *placeShip* pour l’insérer dans la matrice. « s » est la taille du bateau à insérer.

**public** **boolean** addNewShip(**int** column, **int** line, **int** s, **int** d)

*placeShip* modifie la matrice d’entiers pour mettre la valeur 1 aux cases contenant le nouveau bateau

**private** **void** placeShip (int c, int l, **int** size, **int** dir)

*toString* est une méthode très utilisée pour afficher un objet. La grille affiche sa matrice en colonnes et lignes comme ci-dessous.

A close up of a piece of paper

Description automatically generated

**public** String toString()

## La classe Player

### Les attributs

La classe Player contient le nom du joueur et deux objets Grid (l’un pour enregistrer l’emplacement de ses bateaux, l’autre pour enregistrer ses attaques).

### Les méthodes

*addNewShip* fait appel à la méthode *addNewShip* de la grille d’enregistrement des bateaux du joueur et retourne ce qu’elle renvoie.

**public** **boolean** addNewShip(**int** column, **int** line, **int** s, **int** d)

*recordShot* fait appel à la méthode *getValue* de la grille d’enregistrement des attaques. Si l’attaque n’a pas déjà été faite à cette coordonnée-ci, vérifier que le joueur p a un bateau à cet endroit avec la méthode *hasShip*. Dans les deux cas, faire appel à la méthode *addShot* de la matrice d’attaque. Retourne true si l’attaque est un succès.

**public** **boolean** recordShot(**int** column, **int** line, Player p)

*hasShip* renvoie true si le joueur a un bateau à la coordonnée indiquée, false sinon.

**private** **boolean** hasShip(**int** c, **int** l)

*initGridRandom* initialise la grille du joueur avec des bateaux à des positions aléatoires (pour le mode 1 joueur) en appelant *randomInit* de la grille du joueur.

**public** **void** initGridRandom()

*displayGrid* fait appel à *toString* de la grille du joueur.

**public** **void** displayGrid()

*displayShotGrid* fait appel à *toString* de la grille d’attaque.

**public** **void** displayShotGrid()

*hasWin* renvoie true si le joueur a coulé tous les bateaux de l’autre.

**public** **boolean** hasWin()

*getName* renvoie le nom du joueur.

**public** String getName()

## La classe Game

Cette classe vous est fournie.

### Les attributs

Contient les deux joueurs, le joueur courant et le mode de jeu.

### Les méthodes

*fakeInitPlayerGrid* est une méthode permettant au développeur de ne pas avoir à saisir de nombreuses coordonnées pour pouvoir tester le programme. A utiliser en phase de développement uniquement.

**private** **static** **void** fakeInitPlayerGrid(Player p)

*initPlayerGrid* est la méthode de mise en place d’une grille pour un joueur humain.

**private** **static** **void** initPlayerGrid(Player p)

*launch2players* initialise le jeu au mode 2

**public** **static** **void** launch2players()

launch1player initialise le jeu au mode 1

**public** **static** **void** launch1player()

*initGame* est appelé par launch2players et launch1player pour initialiser le jeu. Cette méthode fait appel à *initPlayerGrid* ou *fakeInitPlayerGrid* pour chaque joueur. Dans le cas du mode 1 joueur, fait appel à *initGridRandom*.

**private** **static** **void** initGame()

*shot* fait jouer un joueur en lui demandant une coordonnée. Retourne true si touché, false si raté.

**public** **static** **boolean** shot()

*changeCurrentPlayer* est appelé par la méthode *nextPlayer* pour changer le joueur en fonction du mode de jeu.

**public** **static** **void** changeCurrentPlayer()

*isOver* renvoie true si un des deux joueurs a coulé tous les bateaux de l’autre. Fait appel à la méthode hasWin de chaque joueur.

**public** **static** **boolean** isOver()

## La classe Battleship

Cette classe vous est fournie.

La classe contenant la méthode main. Fait appel à Game.

# Améliorer le jeu

## Coulé

Dans le jeu présenté à la partie précédente, la notion de bateau coulé n’est pas mise en place. A chaque attaque réussie, vérifiez que le bateau n’a pas été coulé. Il faut ajouter des méthodes et modifier *recordShot*.

## Intelligence artificielle

Robot niveau 0 : attaque aléatoirement le joueur adverse.

Robot niveau 1 : attaque aléatoirement le joueur adverse sur une coordonnée sur laquelle le joueur n'a pas encore attaqué.

Robot niveau 2 : attaque aléatoirement une case proche d'un bateau déjà touché (non coulé) et qui n'attaque pas les cases à proximité d'un bateau coulé.

Robot niveau 3 : améliore la niveau 2 par la détection de la direction d'un bateau touché. La case choisie est une des extrémités du bateau touché.

Robot niveau 4 : à vous de trouver des améliorations pour rendre le robot plus intelligent !