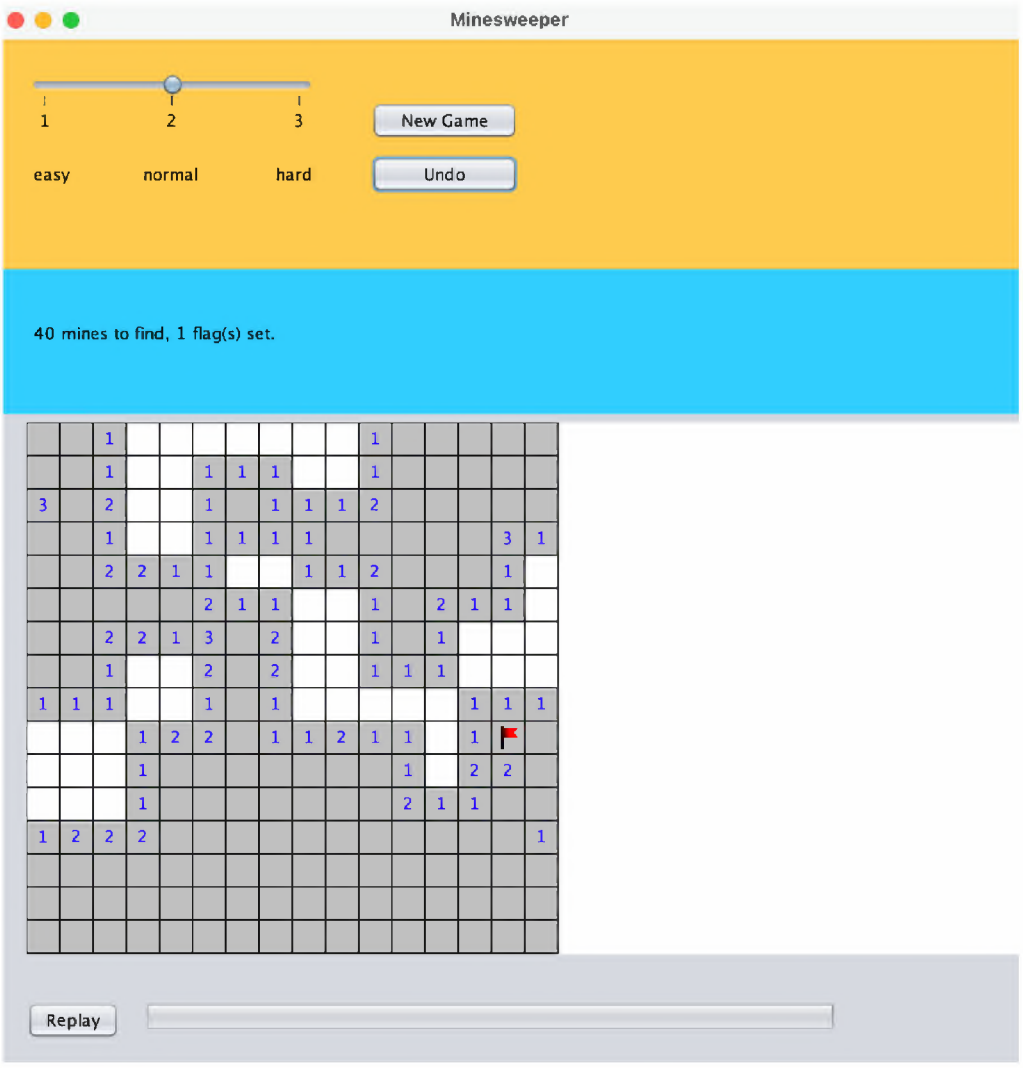
EXAMEN DE FIN D'ÉTUDES SECONDAIRES GÉNÉRALES **Sessions 2023 – QUESTIONNAIRE ÉCRIT**					
Date :	17.05.23	*Durée :*	08:15 - 12:15	*Numéro candidat :*	
Discipline :	Programmation	*Section(s) :*	GIN		

Indication importante :

- Renommez d'abord le dossier EEEE_GIN_x-xx en remplaçant EEEE par le code de votre établissement et x-xx par votre numéro d'examen !
- Un modèle de démonstration du programme fini **Minesweeper.jar** se trouve dans le sous-dossier **dist** de ce dossier.
- *Attention : après avoir ouvert le projet en NetBeans et avant d'ouvrir **MainFrame**, effectuez un « Clean and Build » pour éviter des difficultés avec l'intégration du **DrawPanel**.*

Question 1 : « Minesweeper » [60p.]

Le projet **Minesweeper** est à compléter qui permet de jouer au jeu classique « démineur ». La copie d'écran ci-dessous montre un exemple d'exécution.



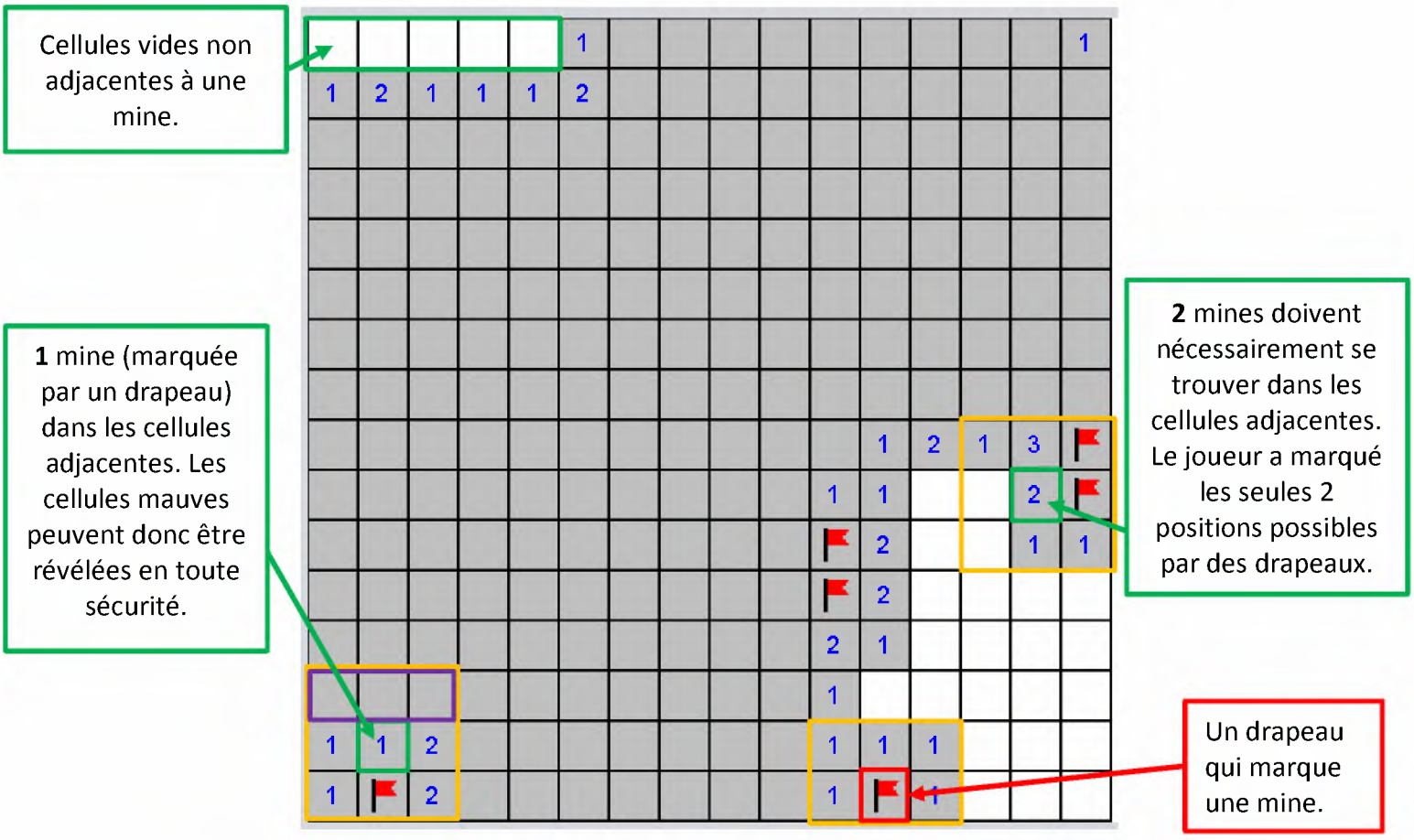
Minesweeper (« démineur ») est un jeu de réflexion qui se joue seul. Le but est de localiser des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seule indication le nombre de mines dans les zones adjacentes[1] (Source : Wikipedia). Avec stratégie et un peu de chance, le joueur doit essayer dans un délai limité de révéler toutes les mines sans en faire exploser.

Au début, le contenu de toutes les cellules est caché, donc le champ de mines est représenté par une grille composée de cellules grises. En cliquant du bouton gauche sur une cellule, la cellule est révélée et rend visible son contenu. Ce contenu est l'un des suivants :

- Une cellule vide en blanc  , ce qui indique l'absence de mines dans les cellules adjacentes.
- Un nombre entre 1 et 8 de couleur bleue  , indique le nombre de cellules adjacentes minées.
- Une mine symbolisée par ce dessin 
- Lorsque le joueur pense qu'une mine se trouve dans une cellule, il peut marquer cette cellule à l'aide d'un clic droit de la souris. Un drapeau  sera alors planté sur cette cellule.

Le jeu est gagné si toutes les cellules ne contenant pas de mines sont révélées sans être tombé sur une mine auparavant. Un joueur peut annuler un mouvement en appuyant sur le bouton « Undo ». De plus, un bouton « Replay » permet de revoir tous les mouvements à la fin de la partie.

Exemple d'une partie en cours qui montre la stratégie du jeu à suivre :



[1] adjacent: angrenzend, benachbart

1ère partie : Liste des mouvements du joueur *[10,5p.]*

Ouvrez le project « **Minesweeper** » et faites tout d'abord un **CLEAN AND BUILD**. Certaines classes sont déjà complétement définies, dont notamment **Move** et **DrawPanel**.

La classe **Move** représente un mouvement du joueur et possède les attributs suivants :

String playersMove	L'exécution du mouvement du joueur il y a 4 possibilités : 1) "**revealCell**" : Le joueur a révélé une cellule. 2) "**revealAllCells**" : Le joueur a touché une mine ou a gagné la partie et toutes les cellules doivent donc être révélées. 3) "**putFlag**" : Le joueur a placé un drapeau sur une cellule. 4) "**removeFlag**" : Le joueur a retiré un drapeau d'une cellule ().
int col	La colonne de la cellule affectée.
int row	La rangée de la cellule affectée.

La classe LinkedList *[10,5p.]*

Dans le paquet **list** développez la classe **LinkedList** qui sert à enregistrer les mouvements du joueur dans une liste chaînée et la structure **Node** associée.

Travail à réaliser :

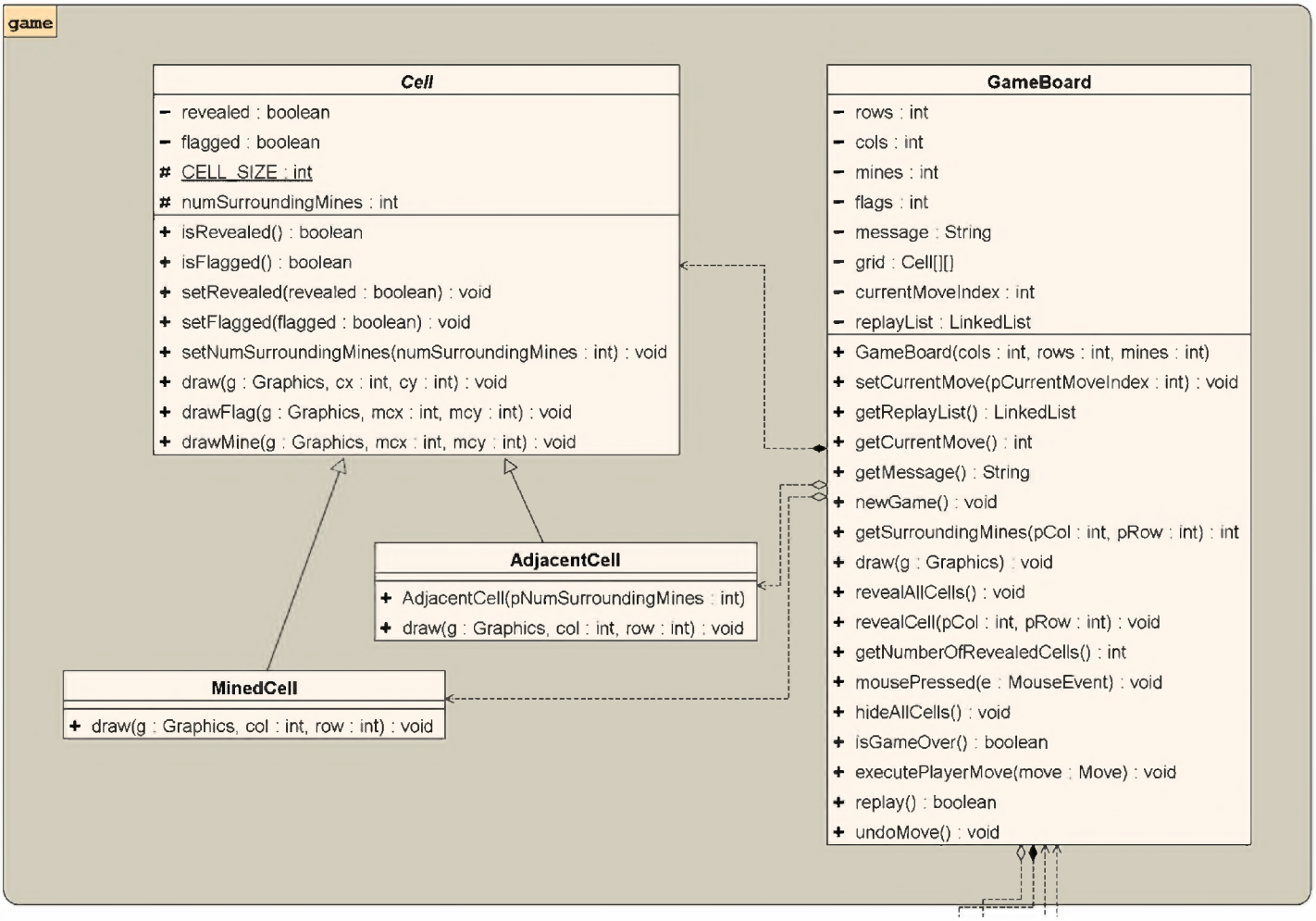
Développez la classe **Node** avec les attributs et le constructeur nécessaire. [1p.]

Implémentez les méthodes suivantes dans la classe **LinkedList**.

1. Implémentez la méthode ***public void add(Move move)*** qui ajoute un mouvement du joueur à la fin de la liste. [2p.]
2. Implémentez la méthode ***public Move get(int n)*** qui retourne le mouvement qui se trouve à la position ***n*** de la liste. Au cas où cette position est invalide, la méthode retourne ***null***. [2,5p.]
3. Implémentez la méthode ***public int size()*** qui retourne le nombre d'éléments de la liste. [1,5p.]
4. Implémentez la méthode ***public boolean removeLast()*** qui supprime le dernier élément et retourne ***true*** si l'élément a été supprimé, sinon ***false***. [3,5p.]

2ème partie : Le terrain de jeu *[45,5p.]*

Voici le diagramme UML du paquet **game** qui contient les classes représentant le terrain de jeu et son contenu.



La classe Cell *[2p.]*

La classe <u>abstraite</u> **Cell** représente une cellule sur la grille du jeu et possède les attributs suivants :

boolean revealed	Indique si la cellule est révélée ou non.
boolean flagged	Indique si la cellule est marquée par un drapeau ou non.
int CELL_SIZE	La taille de la cellule en pixels.
int numSurroundingMines	Le nombre de cellules adjacentes contenant une mine.

Les accesseurs ainsi que les modificateurs standards des attributs sont déjà implémentés.
Les méthodes ***drawMine(…)*** et ***drawFlag(…)*** sont prédéfinies dans **Cell** et peuvent être utilisées pour dessiner une mine respectivement un drapeau. Les paramètres **mcx** et **mcy** représentent les <u>coordonnées</u> du milieu de la cellule.

Travail à réaliser :

Implémentez dans **Cell** la méthode ***public void draw(Graphics g, int cx, int cy)*** qui dessine la cellule avec les coordonnées **cx** et **cy** sur le canevas sous forme de carré de côté **CELL_SIZE**.

- Le bord de la cellule est dessiné en noir.
- Son intérieur est de couleur gris clair (LIGHT_GRAY).
- Si la cellule est non révélée mais a été marquée (bouton droit). Elle est dessinée avec un drapeau à l'intérieur. 

Développez les classes **AdjacentCell** et **MinedCell** qui héritent de la classe **Cell**.

La classe AdjacentCell *[3p.]*

La classe **AdjacentCell** représente une cellule révélée sans mine. Le constructeur initialise l'attribut **numSurroundingMines** avec le nombre de cellules adjacentes minées.

- La cellule est dessinée avec le nombre de cellules adjacentes minées en bleue 

. Toute inscription est décalée de 4 px vers le bas et la gauche par rapport au milieu de la cellule.
- S'il n'y en a pas alors la cellule est dessinée avec un fond blanc vide. 

La classe MinedCell *[1p.]*

La classe **MinedCell** représente une cellule révélée contenant une mine. Elle est dessinée avec une mine à l'intérieur  .

Profitez au mieux du héritage pour dessiner les cellules.

La classe GameBoard *[39,5p.]*

La classe **GameBoard** représente le gestionnaire du jeu. Elle contient la grille avec les mines et enregistre les différents mouvements du joueur. Elle va aussi générer des messages à afficher. Elle possède les attributs suivants :

int cols, rows	Le nombre de colonnes et de lignes du plateau de jeu.
int mines, flags	Le nombre de mines et de drapeaux mis en place.
String message	Le message à afficher.
Cell [][] grid	Un tableau (EN : array) à deux dimensions.
int currenMoveIndex	L'indice du mouvement actuel du joueur dans la **replayList**.
LinkedList replayList	La liste qui sert à sauvegarder tous les mouvements du joueur.

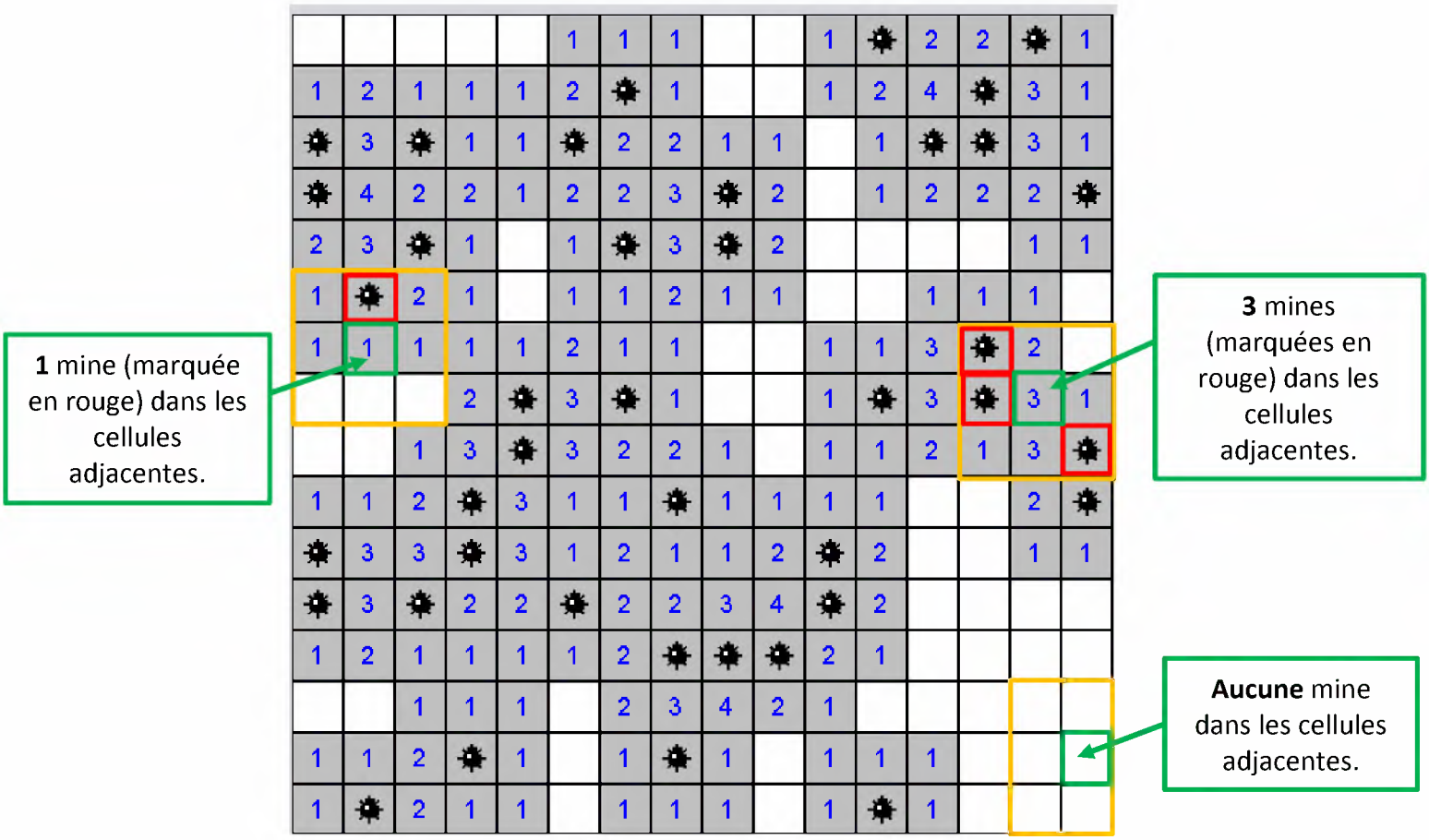
Travail à réaliser :

1. Implémentez dans **GameBoard** la méthode **public void newGame()** qui effectue les opérations suivantes :

a. Création et initialisation de la grille **grid** avec les nouvelles cellules.
b. Placement aléatoire des mines dans la grille. Placer le nombre exigé de mines (attribut **mines**) en faisant attention à ne pas placer deux fois une mine dans la même cellule.
c. Pour chaque cellule non-minée de la grille, il faut déterminer le nombre de cellules adjacentes qui sont minées afin de mémoriser ce nombre (voir point 2) ci-dessous, (méthode ***getSurroundingMines()***) .
d. Initialisation du message à afficher par « X mines to find. » en remplaçant X par le nombre de mines. [6p.]

2. Implémentez la méthode ***public int getSurroundingMines(int pRow, int pCol)*** qui retourne un nombre de 0 à 8 de cellules adjacentes contenant une mine. Une cellule est définie comme adjacente si elle est en contact direct dans une des 8 directions. Evitez de dépasser les limites de la grille. Consultez le programme de démonstration pour trouver des exemples. [6p.]

Voici un jeu terminé avec toutes les cellules révélées :



3. Implémentez la méthode ***public void draw(Graphics g)*** qui dessine toutes les cellules de la grille. [1p.]
4. Implémentez la méthode ***public void revealAllCells()*** qui révèle toutes les cellules du jeu. [0,5p.]

Pour tester le fonctionnement de votre programme, vous pouvez maintenant afficher la grille. Pour ceci, il est utile de révéler toutes les cellules après l'initialisation. (Après ce test n'oubliez pas de le désactiver de nouveau.)

5. Implémentez la méthode ***public void revealCell(int pRow, int pCol)*** qui révèle la cellule indiquée par les paramètres. Si cette cellule n'est pas entourée de mine(s) alors les 8 cellules adjacentes sont révélées de manière récursive. [7,5p.]
6. Implémentez la méthode ***public int getNumberOfRevealedCells()*** qui retourne le nombre total de cellules déjà révélées du jeu. [1,5p.]
7. Implémentez la méthode ***public void mousePressed (MouseEvent e)*** prend comme paramètre l'événement du clic de la souris. Si les coordonnées du clic de la souris correspondent à une cellule et si cette cellule n'était pas encore révélée, la méthode effectue les actions suivantes :
    a. Pour un clic du bouton gauche : (MouseEvent.BUTTON1)
        i. La cellule est démasquée et un nouveau mouvement du joueur est créé et ajouté à la **replayList** (mouvement « revealCell »).
        ii. Si la cellule contient une mine, alors le jeu est perdu, toutes les cellules sont démasquées et le message « You hit a mine! Game over! » est affiché. Un nouveau mouvement du joueur est créé et ajouté à la **replayList** (mouvement « revealAllCells »).
        iii. Si toutes les cellules non-piégées ont été démasquées, alors le jeu est gagné, toutes les cellules sont démasquées et le message « You won! Congratulations! » est affiché. Un nouveau mouvement du joueur est créé et ajouté à la **replayList** (mouvement « revealAllCell »).
    b. Pour un click du bouton droit (MouseEvent.BUTTON3)
        i. Si la cellule est marquée, supprimez le marquage avec un drapeau, sinon plantez un marquage. De plus un nouveau mouvement de joueur est créé et ajouté à la **replayList**. (mouvement « removeFlag » ou « putFlag » le cas échéant).
        ii. Le message du jeu est mis à « X mines to find, Y flag(s) set ! », X dénotant le nombre total de mines du jeu et Y le nombre total de drapeaux posés.

    [8p.]
8. Implémentez la méthode ***public void hideAllCells()*** qui masque toutes les cellules. [1,5p.]
9. Implémentez la méthode ***public boolean isGameOver()*** qui retourne si le jeu est terminé ou non. Testez pour cela simplement le contenu de l'attribut **message**. [1p.]
10. Implémentez la méthode ***public void executePlayerMove(Move move)*** qui exécute le mouvement donné si celui-ci n'est pas ***null***. Veuillez revoir la classe **Move** pour différencier les cas possibles. Faites attention de mettre à jour le nombre de drapeaux. [2,5p.]
11. Implémentez la méthode ***public boolean replay()*** qui exécute le mouvement de la **replayList** à l'indice **currentMoveIndex** (Profitez de la méthode ***executePLayerMove()***). Elle renvoie ***true*** si la liste a été entièrement traitée sinon ***false***. Veillez à incrémenter l'indice si un mouvement a été traité. [2p.]
12. Implémentez la méthode ***public void undoMove()*** qui annule le dernier mouvement du joueur si possible. Utilisez la méthode ***hideAllCells()*** pour masquer à nouveau toutes les cellules (=état après le lancement du jeu), enlevez le dernier mouvement de la liste et exécuter ensuite tous les mouvements du joueur. Veillez à mettre à jour le message à la fin. [2p.]

3ème partie : L'interface graphique *[4p.]*

Ouvrez dans le paquet **gui** la classe **MainFrame** qui représente la vue et le contrôleur du MVC (Model-view-controller).

La classe MainFrame *[4p.]*

Travail à réaliser :

Complétez l'action du bouton « **Replay** » de la classe **MainFrame** qui permet de revoir les mouvements effectués par le joueur dans le même ordre chronologique, mais uniquement si la partie est terminée. Pour cela, utilisez un chronomètre (« **replayTimer** ») qui exécute chaque seconde un mouvement du joueur. Veillez à ce que le **replayTimer** est arrêté s'il est déjà en cours d'exécution (au cas où le joueur appuie une deuxième fois sur le bouton alors que le replay n'est pas encore terminé).