TP3 — STRATEGO

Dans ce travail, tu devras apporter des améliorations à une version très élémentaire du jeu de stratégie Stratego.

Il y aura beaucoup de travail à faire et tu peux modifier à peu près tout dans le projet de base qui t'est donné (dossier « **Stratego – version de base** »). Tu dois par contre garder en tête qu'il faut avoir une raison valable pour changer du code qui fonctionne déjà (un principe important à respecter en entreprise) ou la structure de ce code.

RÈGLES DU JEU

Les règles présentées ici sont inspirées principalement de Wikipédia (https://fr.wikipedia.org/wiki/Stratego) et d'un vidéo (https://www.youtube.com/watch?v=Z7eU19TCskI).

- Le jeu se joue sur une grille de **92 cases** (grille de 10 x 10 cases, moins 2 lacs de 2 x 2).
- Chaque joueur contrôle **40 pièces**, qui ont une valeur de force attribuée.

1 maréchal	10
1 général	9
2 colonels	8
3 commandants	7
4 capitaines	6
4 lieutenants	5
4 sergents	4
5 démineurs	3
8 éclaireurs	2
1 espion	1
1 drapeau	-
6 bombes	-

- Au début de la partie, chaque joueur positionne ses pièces comme il le désire sur les **4 premières lignes** de son côté du tableau de jeu.
- Chaque joueur connait ses propres pièces, mais pas celles de son adversaire, qui semblent toutes être identiques de son point de vue.
- Le **premier coup** est toujours fait par le joueur rouge.
- Les pièces se **déplacent** d'une case, soit vers l'avant, l'arrière, la gauche ou la droite.
  Il n'y a pas de déplacement en diagonale.

- La règle sur les **déplacements** comporte quelques **exceptions**.
    - Il est **impossible** de se déplacer sur les cases des lacs.
    - Une pièce ne peut pas se déplacer sur une autre **case occupée** par une **pièce de la même couleur**.
    - Le drapeau et les bombes, une fois positionnés, ne se **déplacent jamais**.
    - L'**éclaireur** n'est pas limité sur le nombre de cases traversées pendant son déplacement, tant que son déplacement ne se fait que dans **une seule direction** (on peut comparer son déplacement à celui d'une tour aux échecs.).
    - L'éclaireur ne peut pas traverser une **case occupée** par une **pièce de la même couleur**.
- Si une pièce termine son déplacement sur une case occupée par une pièce ennemie, c'est une **attaque**.
    - Lors d'une attaque, on **compare la force** des deux pièces impliquées.
        - La pièce la plus forte l'emporte et reste en jeu. L'autre pièce est éliminée.
        - Si les deux pièces sont de force égale, les deux sont éliminées.
    - La règle du **combat** comporte elle aussi quelques **exceptions**.
        - Un combat entre un **espion** et un **maréchal** est gagné par la pièce qui a initié l'attaque.
        - Toute pièce qui attaque une **bombe** est éliminée et la bombe reste en jeu, **sauf** si l'attaquant est un **démineur**, cas dans lequel le démineur est le gagnant.
- Le premier joueur à capturer (attaquer) le drapeau de l'adversaire **gagne la partie**.

TRAVAIL À FAIRE

La **version de base** fournie est une version fonctionnelle, mais **incomplète**, du jeu. Certaines règles sont respectées, d'autres ne le sont pas. Tu ne dois pas assumer que tout est correct, mais il ne faut pas non plus croire que toutes les structures mises en place dans le code sont incorrectes.

Comme lorsqu'on travaille en maintenance en entreprise, tu ne dois **pas changer le code** simplement parce qu'il ne respecte pas tes **préférences**. Les changements apportés doivent donc être liés directement à (et justifiés par) une des modifications demandées, sinon tu pourrais perdre des points!

Note bien que même si les améliorations à apporter sont présentées une par une, ta solution doit fonctionner comme **un tout cohérent**, pas une série de modifications qui s'ignorent les unes et les autres. Autrement dit, garde une vue d'ensemble sur ta solution, car il se peut très bien que plusieurs améliorations puissent se faire ensemble et en harmonie.

Enfin, il y a une section du code qui utilise un **thread**. N'y **touche pas**. Le but est simplement de mettre une pause visuelle pour rendre le déroulement du jeu plus clair pour un être humain.

REFACTORISATIONS

- La classe **Point** est utilisée à plusieurs endroits dans le code. Point utilise le type ***double*** alors que notre application ne nécessite que des variables de type *int*.
    - Tu dois créer une nouvelle classe qui s'appellera **Coordonnee** et qui fera la même chose que Point, mais avec des ***int***.
    - Cette nouvelle classe doit **remplacer Point partout** dans le code (Point ne sera plus du tout utilisé).
- La **couleur** (rouge ou bleu) est représentée par une **chaine de caractères**. Il n'y a pourtant que deux valeurs possibles dans Stratego et ça ne changera jamais.
    - Tu dois créer une **énumération** pour représenter ces couleurs.
    - Le format string ne devra plus être utilisé pour représenter les couleurs dans l'application.
- Pour **déterminer la couleur** de la pièce située sur une case, il faut aller chercher la couleur de la case et la comparer à rouge ou bleu. C'est fait à quelques endroits dans le code présentement et ce n'est pas une approche très OO.
    - Tu dois remplacer cette manière de faire par une méthode nommée **EstDeCouleur(couleurRecherchee)**.
    - La méthode qui servait à obtenir la couleur d'une case devrait disparaitre.
- Les pièces ont présentement toutes un **attribut de force**, qui représente leur puissance relative lors d'un combat. Dans le cas de la bombe et du drapeau, c'est une représentation erronée.
    - Puisque ces deux types de pièces ne peuvent pas se déplacer, tu vas faire en sorte que les pièces qui n'ont pas de mobilité n'ont pas d'attribut de force.

MODIFICATIONS

- Le **déplacement des pièces** est implémenté, mais il est incomplet. Toutes les pièces se déplacent d'une case à la fois, ce qui va à l'encontre de certaines règles.
    - Fais les modifications nécessaires pour que toutes les règles de déplacement soient respectées.
    - Note qu'il faut privilégier l'utilisation des **voisins** pour naviguer dans la grille des cases du jeu et non la **grille 2D** qui doit servir principalement comme point d'entrée (repérer des cases indiquées par les interactions de l'utilisateur).
- La **gestion des attaques** entre les pièces est, tout comme le déplacement des pièces, incomplète.
    - Les cas d'exceptions où la force d'une pièce n'est pas ce qui détermine le résultat doivent tous être gérés.
    - Il faut prendre en considération que certaines pièces n'auront plus de force (tel que demandé dans les refactorisations).

- Quand un drapeau est capturé, la **partie doit se terminer**.
    - Il faut indiquer clairement le fait que la partie est terminée et quel joueur a gagné.
    - On ne doit **plus pouvoir jouer** de coups à partir de ce moment.
- Le jeu s'ouvre avec une partie déjà lancée.
    - La partie ne doit pas être déjà lancée quand on ouvre le jeu. On doit pouvoir **choisir de lancer une partie**.
    - Il doit être possible, **en tout temps**, de lancer une nouvelle partie.
    - Si une nouvelle partie est demandée pendant qu'une partie est en cours, un message de **confirmation** doit être affiché (pour éviter de perdre la partie en cours par accident).
- Lors du lancement d'une nouvelle partie, le joueur doit **choisir la couleur** qu'il veut jouer.
    - Ceci implique qu'il est **possible que l'IA joue du côté rouge**.
    - Quand le **joueur est du côté bleu**, les pièces bleues doivent être positionnées au **bas de l'écran** (on inverse donc les côtés au niveau de l'affichage).
    - Le joueur rouge est **toujours** le premier à faire un coup.
- Afin de pouvoir tester la version présente de l'application, un positionnement pour les deux joueurs est « hardcodé » dans la méthode PositionnerPieces du UserControl.
    - Il faut changer ceci pour que, lors du lancement d'une partie, le joueur puisse choisir la position de ses 40 pièces.
    - C'est à toi de choisir comment la « phase de positionnement » se déroulera. Tu dois viser une approche **facile à comprendre et facile à utiliser**. Idéalement, on veut qu'il soit **impossible de se tromper dans l'attribution des pièces** (pas de position illégale, positionnement du bon nombre de chaque type de pièces et le bon nombre de pièces au total).
    - La seule chose qui t'est imposée pour le fonctionnement de cette phase, c'est que ça ne soit **pas fait dans JeuStrategoControl**. Tu dois créer un nouveau contrôle qui fournira, lorsque les choix sont terminés, l'information nécessaire pour positionner les pièces.
    - L'IA devra lui aussi faire son positionnement (voir la section sur l'IA).
- La **méthode PositionnerPieces** de la classe GrilleJeu peut être améliorée.
    - Les 40 pièces reçues doivent toutes être de la **même couleur**.
    - Parmi ces pièces, on doit retrouver exactement le **bon nombre de pièces** qu'un joueur contrôle en début de partie et ce pour **chaque type** de pièces.
    - Le positionnement devra s'ajuster au fait que le joueur peut choisir sa couleur (voir plus haut).

- L'**affichage des pièces** est, comme bien des facettes du jeu, incomplet.
    - Tu dois trouver des **images** pour chaque type de pièce.
        - Ces images doivent être claires et **faciles à reconnaitre**.
        - Une option simple est de t'inspirer des images qui sont utilisées dans la version physique du jeu.
        - Si tu choisis de t'amuser dans le choix, reste dans le **bon gout** et assure-toi que les images font un certain **sens vis-à-vis du jeu**.
        - Dans tous les cas, la **couleur** de la pièce elle-même doit toujours être **évidente** (rouge ou bleu).
    - Chaque image sera accompagnée du **chiffre** qui représente sa **force**, quand elle en a une.
        - Pour que la gestion de l'affichage soit simple, il est fortement recommandé d'incorporer le chiffre directement dans l'image.
        - Le chiffre doit être suffisamment gros pour être facile à lire.
    - Normalement, on ne doit pas voir les pièces de l'adversaire, juste une **image « face cachée »** de ces pièces.
- Dans le jeu de table, après la résolution d'une attaque, les deux joueurs **connaissent la pièce de l'adversaire** impliquée dans cette attaque.
    - Quand une pièce de l'IA est ainsi **révélée** au joueur, elle devient, et ce pour le reste de la partie, **visible**. On verra donc son image et sa valeur, même si elle est déplacée.
    - Puisque l'IA devra aussi avoir accès à cette information pour prendre ces décisions (voir plus bas), trouve une manière de la **structurer qui pourra être utilisée dans les deux cas** (dans l'affichage des pièces et dans les décisions de l'IA).
- Puisqu'on ne peut pas voir les pièces de l'adversaire, il est très utile de savoir quelles pièces il ne contrôle plus.
    - Tu vas ajouter l'**affichage de des pièces prises** au contrôle utilisé pour le jeu, à droite du tableau de jeu.
    - Seules les **pièces prises à l'IA** nous intéressent.
    - Cet affichage doit être structuré de manière à ce qu'il soit facile à lire et comprendre.
    - L'affichage est, évidemment, ajusté chaque fois qu'une pièce est éliminée.
- Les commentaires ont été « oubliés » dans l'application.
    - Tu dois **ajouter les commentaires manquants** en respectant les exigences indiquées cette session à ce sujet.
    - Il faut bien sûr tenir en compte que ces commentaires devront expliquer ce que fait la version finale du code. Autrement dit, si tu apportes des modifications dans le code, les commentaires doivent refléter ces modifications.
- Porte attention aux petits détails (comme le fait que le nom de la fenêtre est encore « MainWindow ») et corrige-les.

- L'**IA** est présentement plutôt simpliste. Il choisit au hasard un coup parmi les coups possibles.
    - Tu dois **améliorer son fonctionnement**. Aucun algorithme n'est imposé, mais tu dois trouver une logique qui va au-delà de « choisir un coup au hasard parmi les coups légaux ».
    - Soyons clairs. Il n'est pas question de faire une IA compétitive, il suffit de dépasser le niveau présent de complexité. Du moment que ton algorithme est une amélioration et qu'il fonctionne sans erreurs, il sera considéré comme acceptable.
    - De plus, ton algorithme doit éviter d'attaquer une pièce si l'attaque est perdue d'avance dans les cas où cette **pièce a déjà été révélée**.
        - Ceci implique que l'IA connaitre lui aussi les pièces de son adversaire qui ont été révélées.
    - L'IA devra **choisir comment ses pièces sont positionnées** en début de partie.
        - Afin de garder simple cette phase très stratégique, tu peux te contenter de suivre 3 règles simples, sans plus.
        - Les **3 règles** suivies doivent être claires dans l'algorithme (tu peux ajouter un commentaire pour les rendre plus claires).
        - À titre d'exemple de règle, le drapeau est généralement positionné sur la ligne du fond, pour l'exposer le moins possible à l'adversaire.
    - Tu dois fournir des **explications sur le fonctionnement** de cet algorithme dans **le commentaire décrivant la classe IA elle-même**. Les explications doivent me **permettre de comprendre ta logique**. Elles doivent donc être claires et complètes.

BONUS (5%)

Tu peux, si tu le désires, ajouter une fonctionnalité au jeu. Cette fonctionnalité sera évaluée sur une combinaison de sa **complexité**, la **qualité de son implémentation** et son **originalité**. Un seul ajout sera accepté pour ce bonus.

Si tu choisis d'ajouter une telle fonctionnalité, tu dois la nommer et fournir une **explication** à son sujet dans le **commentaire qui décrit la classe de la fenêtre principale**.

PONDÉRATION

Élément	Valeur
Refactorisations	17,5 %
Modifications	62,5 %
Intelligence artificielle	20 %
Bonus	+5 %

Puisque des points sont attribués pour les commentaires, il n'y a pas de pénalité associée à ceux-ci.

Le **français** doit être bien écrit (et est sujet à la pénalité habituelle de 10 %).

REMISE

La remise se fera le **jeudi 10 mai**, à la **fin du cours**.

- Tu dois remettre ton projet au complet dans **LÉA**, dans un fichier .**zip**.
- Le projet doit compiler sur la version de Visual Studio du collège.
  Si tu utilises une version différente à la maison, assure-toi que tout compile et fonctionne correctement au Cégep.