Travaux Pratiques : Jeu de Puissance 4 en Go

Objectif

Développer une version console du jeu Puissance 4 en Go, où les joueurs insèrent alternativement des jetons dans une grille verticale, cherchant à aligner quatre jetons de leur couleur.

Points Totals : 20 Points (+ 2 Bonus)

Instructions Générales

- Utilisez uniquement la librairie standard de Go.
- Suivez les bonnes pratiques de programmation en Go.
- Commentez votre code de manière à expliquer les choix de conception et les fonctionnalités implémentées.

Partie 1 : Logique de Base du Jeu (10 Points)

Objectif :

Créer la grille de jeu et gérer l'alternance des joueurs, en vérifiant les conditions de victoire pour un alignement de quatre jetons.

- Les joueurs insèrent alternativement des jetons dans une colonne de la grille.
- Le jeu s'arrête lorsqu'un joueur a aligné quatre jetons horizontalement, verticalement ou en diagonale, ou lorsque la grille est pleine.
- Le jeu doit afficher l'état actuel de la grille à chaque tour.
- Une grille de puissance 4 standard est de taille 6 lignes x 7 colonnes.

Exemple de squelette de Code :

```
package main

import "fmt"

type Player int

const (
    Empty  Player = iota  // 0
    Player1               // 1
    Player2               // 2
)

type Board [6][7]Player

func main() {
    var board Board
    var currentPlayer Player = Player1

    for {
        fmt.Println("État actuel du plateau :")
        printBoard(board)

        if gameIsOver(board) {
            break
        }
        currentPlayer = switchPlayer(currentPlayer)
    }
}

func printBoard(b Board) {
    // Afficher le plateau ici
}
func switchPlayer(current Player) Player {
    // Changer de joueur
}
func gameIsOver(b Board) bool {
    // Vérifier la condition de victoire
    return false
}
```

Boucle de Jeu

- La boucle principale du jeu doit alterner entre les joueurs, en demandant à chaque joueur de choisir une colonne où insérer son jeton.
- La boucle doit s'arrêter dès qu'un joueur a gagné ou que la grille est pleine.
  - La fonction `gameIsOver` doit retourner `true` si un joueur a gagné ou si la grille est pleine.
  - La fonction `switchPlayer` doit retourner l'autre joueur.
  - La fonction `printBoard` doit afficher la grille de jeu.
- Les joueurs doivent insérer leur jeton dans une colonne de la grille, en vérifiant que la colonne n'est pas déjà pleine.
  - Vous devez donc demander à chaque joueur de choisir une colonne valide, et insérer son jeton dans la première case vide de cette colonne.
  - Vous pouvez utiliser la fonction `fmt.Scan` pour demander à l'utilisateur de saisir un nombre, et vérifier que la colonne est valide.
  - Vous pouvez utiliser une boucle `for` pour demander à l'utilisateur de saisir une colonne tant que la saisie n'est pas valide.
  - Vous pouvez utiliser une boucle `for` pour chercher la première case vide dans une colonne, et insérer le jeton du joueur à cet emplacement.

Fonction `printBoard`

- La fonction `printBoard` doit afficher la grille de jeu, en utilisant des caractères différents pour les jetons des deux joueurs et pour les cases vides.
- Par exemple, vous pouvez utiliser les caractères `X`, `O` et `.` pour représenter les jetons des deux joueurs et les cases vides.
- La grille peut être affichée avec des lignes et des colonnes numérotées pour faciliter la saisie des coups.
- Vous pouvez utiliser la fonction `fmt.Printf` pour afficher la grille de manière plus lisible.

Exemple d'affichage de la grille :

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-------------------------------
0| . | . | . | . | . | . | . |
1| . | . | . | . | . | . | . |
2| . | . | . | . | . | . | . |
3| . | . | . | . | O | X | . |
4| . | . | . | . | X | O | . |
5| . | . | . | X | O | X | . |
```

Fonction `gameIsOver`

- La fonction `gameIsOver` doit vérifier si un joueur a gagné ou si la grille est pleine.
- Vous pouvez utiliser des boucles `for` pour vérifier les lignes, colonnes et diagonales de la grille, et vérifier si quatre jetons de la même couleur sont alignés.
- **Exemple :**

```
func (game *Game) checkWinner() Player {
  for i := range 6 {
    for j := range 7 {
      // Check row
      // Check that we are not too close to the right edge (to avoid out of range error)
      // Check if the current cell is equal to the next three cells
      if j <= *BoardWidth-4 && isEqual(game.Board[i][j], game.Board[i][j+1], game.Board[i][j+2],
game.Board[i][j+3]) {
        // ...
      }
      // ...
    }
  }
  // ...
}
```

- Vous pouvez utiliser des variables booléennes pour vérifier si un joueur a gagné dans une ligne, une colonne ou une diagonale.

Fonction `switchPlayer`

- La fonction `switchPlayer` doit retourner l'autre joueur.
- Vous pouvez utiliser une instruction `if` ou une expression conditionnelle pour retourner le joueur opposé.
- Par exemple, si le joueur actuel est `Player1`, vous devez retourner `Player2`, et vice versa.

Partie 2 : Lecture et Écriture de Fichier (3 Points)

Objectif :

Sauvegarder l'état actuel du jeu dans un fichier et charger une partie existante.

Indications et Squelette de Code :

- Utilisez `os` pour gérer les opérations de fichier.
- Structurez les données de manière lisible et facile à charger.
- Intégrer ces méthodes dans le jeu pour permettre aux joueurs de sauvegarder et de charger une partie.

> *Astuce : Vous pouvez utiliser le format JSON pour sauvegarder et charger l'état du jeu.*

```
func saveGame(b Board) {
    // Écrivez la logique pour sauvegarder l'état du jeu dans un fichier
}

func loadGame() Board {
    // Écrivez la logique pour charger l'état du jeu depuis un fichier
    return Board{}
}
```

Fonction `saveGame`

- La fonction `saveGame` doit écrire l'état actuel de la grille dans un fichier.
- À vous de choisir le format de fichier le plus adapté pour sauvegarder l'état du jeu (Tip : On a vu les formats JSON en cours !).
- Pensez à vous souvenir aussi du joueur actuel pour pouvoir reprendre la partie plus tard.

Fonction `loadGame`

- La fonction `loadGame` doit charger l'état du jeu depuis un fichier.
- Vous devez lire le fichier et reconstruire la grille de jeu à partir des données sauvegardées.
- Vous devez aussi retenir le joueur actuel pour reprendre la partie là où elle s'était arrêtée.

Partie 3 : Interface Utilisateur (2 Points)

Objectif :

Améliorez l'affichage du plateau et assurez-vous que les instructions pour l'utilisateur sont claires.

(Psss... Vous pouvez utiliser des couleurs pour rendre le jeu plus attrayant !)

Partie 4 : Qualité du Code et Commentaires (5 Points)

Objectif :

Assurez-vous que le code est bien structuré, avec des noms de variables et de fonctions significatifs, et bien commenté.

PS : Trop de commentaires tuent le commentaire, ils doivent être utiles et pertinents. Trop de commentaires peut être un signe de mauvaise qualité de code, car cela peut signifier que le code n'est pas assez clair.

Partie 5 : Fonctionnalités Bonus (2 Points)

Objectif :

Ajoutez des fonctionnalités supplémentaires pour gagner des points bonus.

Exemples :

- Option pour rejouer une nouvelle partie sans redémarrer le programme.
- Implémenter un compteur de score ou d'autres variantes de jeu.
- Taille du plateau de jeu personnalisable.
- ...

Critères d'Évaluation

- Fonctionnalité et logique du jeu : **9 points**
- Lecture/écriture de fichier : **3 points**
- Interface utilisateur : **3 points**
- Qualité du code et commentaires : **5 points**
- Fonctionnalités bonus : jusqu'à **2 points**
- **Total** : **20 points** (+ **2 points** bonus)