Synthèse 👍

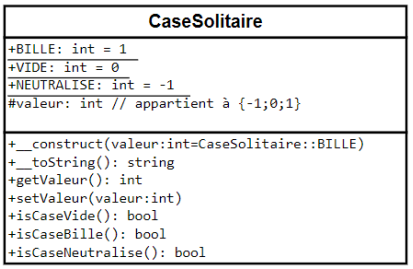
Rappel du projet :

Le solitaire est un jeu qui, comme l'indique son nom, se pratique seul. Le joueur déplace des pions (généralement des billes ou des fiches) sur un plateau dans le but de n'en avoir plus qu'un seul à la fin du jeu.
L'objectif de ce projet est de programmer une version fonctionnel avec une interface utilisateur de ce jeu, et cela en utilisant le langage PHP qui est un langage de programmation puissant qui permet la manipulation de données et la création de pages Web dynamiques, ce qui en fait un choix idéal pour la programmation de ce jeu.

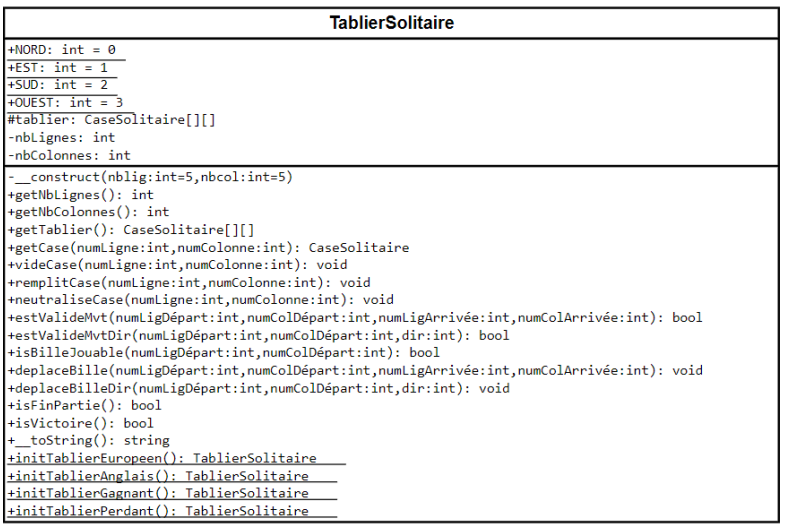
Etape 01:

Dans cette étape étape on a implanté deux classes :

- Classe **CaseSolitaire**: qui est une case du tablier, on peut lui associer 3 états (vide, neutralisée, bille) .



Classe **TablierSolitaire :**
Cette classe représente le tablier du jeu qui est constitué de plusieurs Cases. Cette classe nous permet de récupérer le contenu des cases , d'effectuer certaines modifications dans les cases , et vérifier les mouvements des billes vers une destination ou dans une direction, comme elle permet aussi de



vérifier l'état du tablier si aucune bille ne peut être jouée alors c'est la fin du jeu, et si il reste une seule bille à la fin qui est une victoire.
Enfin cette classe nous permet de construire des tabliers à base des modèles standards.

Etape 02:

Lors ce cette étape on s'intéresse à la réalisation d'une interface graphique pour le jeu de Solitaire dont on a réalisé les classes métier dans le TP précédent.

Sélection Bille (phase 1) : Lors de l'affichage du tablier dans index, en appuyant sur une bille ça nous renvoie dans la page action avec le formulaire qui contient les coordonnées de la bille dans $_POST. On renvoie ces coordonnées dans index ce qui permet de sélectionner la bille et désactiver toutes les autres, et ça permet d'afficher les destinations disponibles et de désactiver tout le reste.

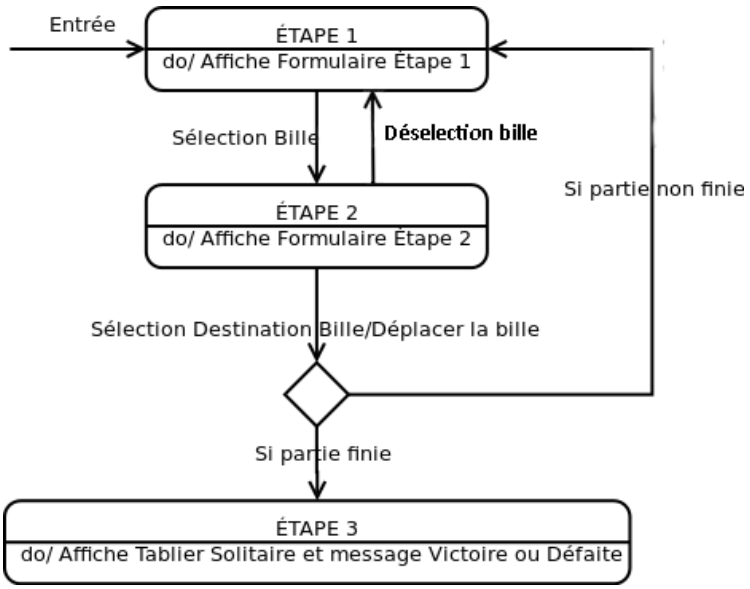
Sélection Destination (phase 2) : Lors de la sélection de la destination en envoie vers action les coordonnées du départ par GET ainsi que les coordonnées de la destination par POST, ce qui permet de vérifier à action si une bille à été déjà sélectionnée(car j'ai des coordonnées de départ dans GET) et j'ai une destination (Dans POST). on distingue 2 cas :

- Destination est case Vide : dans ce cas on appelle une fonction qui modifie le tablier du jeu en effectuant un mouvement de la bille sélectionnée vers la destination en vidant la case au milieu. ensuite faire une redirection vers index sans envoi d'informations.
- Destination est Origine : dans ce cas on est censés désélectionner la bille et cela en faisant une redirection directe vers index sans envoi d'informations ;

▨ Après chaque jeu avant affichage on vérifie l'état du jeu dans index : en vérifiant si la partie est finie en appelant par notre tablier la fonction isFinPartie() de la classe TablierSolitaire .

- **Cas 1** : Partie non finie : On affiche le Tablier et on continue le jeu;
- **Cas 2** : Partie Finie : On vérifie si la partie est gagnée avec la fonction isVictoire() de la classe TablierSolitaire, qui est une défaite sinon. ensuite on gère l'affichage en fonction. (Sans oublier de fermer la session)

ce qui nous donne **après modification** l'automate suivant :



Etape 03 :

Dans cette étape, le principe est de créer un bouton au début du jeu qui permet de configurer le tablier avant de commencer la partie. ce bouton nous envoie vers action pour saisir le nombre de lignes et de colonnes souhaité pour initialiser le tablier. avec **initTablierDefaut(lignes, colonnes)** dans la classe **TablierSolitaire** qui permet de créer un tablier avec un nombre de lignes et de colonnes donné.

TablierSolitaire

```
+NORD: int = 0
+EST: int = 1
+SUD: int = 2
+OUEST: int = 3
#tablier: CaseSolitaire[][]
-nbLignes: int
-nbColonnes: int
-__construct(nblig:int=5,nbcol:int=5)
+getNbLignes(): int
+getNbColonnes(): int
+getTablier(): CaseSolitaire[][]
+getCase(numLigne:int,numColonne:int): CaseSolitaire
+videCase(numLigne:int,numColonne:int): void
+remplitCase(numLigne:int,numColonne:int): void
+neutraliseCase(numLigne:int,numColonne:int): void
+estValideMvt(numLigDépart:int,numColDépart:int,numLigArrivée:int,numColArrivée:int): bool
+estValideMvtDir(numLigDépart:int,numColDépart:int,dir:int): bool
+isBilleJouable(numLigDépart:int,numColDépart:int): bool
+deplaceBille(numLigDépart:int,numColDépart:int,numLigArrivée:int,numColArrivée:int): void
+deplaceBilleDir(numLigDépart:int,numColDépart:int,dir:int): void
+isFinPartie(): bool
+isVictoire(): bool
+__toString(): string
+initTablierEuropeen(): TablierSolitaire
+initTablierAnglais(): TablierSolitaire
+initTablierGagnant(): TablierSolitaire
+initTablierPerdant(): TablierSolitaire
+initTablierPerdant(): initTablierDefaut()
```

ensuite on revient vers index qui nous renvoie à action pour configurer ce tablier .

Principe:

Après avoir choisi la configuration manuelle du tablier, on initialise une valeur dans get qu'on appelle config qui a au début la valeur 0.

Remarque : en appuyant sur le bouton NEXT on envoie le formulaire ainsi que la valeur suivante de config dans POST["next"].

TablierSolitaireUI

```
-ts: TablierSolitaire
+__construct(ts:TablierSolitaire=null)
+getFormulaireOrigine(): string
+getFormulaireDestination(coord_depart:string): string
+getPlateauFinal(): string
-getBoutonCaseSolitaire(classe:string,ligne:int,colonne:int,disabled:bool): string
+setTablier()
```

- **Étape 1** : définir le nombre de lignes et le nombre de colonnes : quand config est à 0, dans index on appelle la fonction setTablier de TablierSolitaireUI qui affiche un formulaire qui demande d'entrer le nombre de lignes et de colonnes souhaité entre 3 et 8. ensuite le formulaire sera envoyé vers action ou on crée ce tablier, en initialisant notre tablier dans SESSION. et la valeur de config sera incrémentée à 1;

- **Etape 2** : définir les cases neutralisées : Quand la valeur de config est à 1 ça nous permet de définir les cases à neutraliser: on affiche le tablier avec la fonction setTablier() de TablierSolitaireUI en l'appelant dans index, on sélectionnant une bille on envoie par POST les coordonnées de la case à neutraliser vers action ainsi que la valeur de config. Alors si la valeur de config est à 1 on neutralise la case dans le tablier dans la session, on incrémente la valeur de config en faisant une redirection vers index avec la valeur de config incrémentée dans un GET (`"Location: index.php?config=".$_POST["next"]`) où POST[“next”]est la valeure suivante de config.

- **Etape 3 :** définir les cases vides: Quand la valeur de config est à 2 ça nous permet de définir les cases à neutraliser: on affiche le tablier avec la fonction setTablier() de TablierSolitaireUI en l'appelant dans index, on sélectionnant une bille on envoie par POST les coordonnées de la case à neutraliser vers action ainsi que la valeur de config. Alors si la valeur de config est à 2 on vide la case dans le tablier dans la session on incrémente la valeur de config en faisant une redirection vers index avec la valeur de config incrémentée dans un GET (`"Location: index.php?config=".$_POST["next"]`) où POST[“next”]est la valeur suivante de config.

- **Étape 4** : Quand la valeur de config sera de 3 on vérifie si un mouvement est possible, sinon cela signifie qu'aucune case n'est vide alors on revient à l'étape 3 avec la valeur de **config à 2** pour pouvoir vider une case. Si un mouvement est possible on fait une redirection vers index pour commencer la partie.

Cela nous produit un **automate final** comme suivant :