Projet PIP semestre 2 – 2017-2018 – ENSIIE Evry

HONSHU

Votre travail consiste à réaliser, en équipe de 4, une application ludique et le solveur associé.
L'application à programmer est inspirée du jeu de plateau « Honshu ».
En plus de donner la possibilité de jouer, votre application devra disposer d'un solveur qui tâchera de marquer un maximum de points.

Description du problème

Ce jeu se présente sous la forme d'un plateau de taille « n x n » et de « tuiles » de 2x3 cases. Sur chacune de ses 6 cases se trouve un terrain (Plaine P ; Foret F ; Lac L ; Ville V ; Ressource R ; Usine U). Au début de la partie, une tuile se trouve au centre d'un plateau carré de jeu. Le joueur dispose de 12 tuiles dès le début de la partie, et doit les pauser une à la fois en respectant les règles suivantes :

- Une tuile ne peut être totalement recouverte à aucun moment de la partie ;
- Toute nouvelle tuile posée doit recouvrir au moins une case de l'une des tuiles précédemment posée, la précédente case est alors oubliée;
- Aucune case Lac ne peut être recouverte.
- Le joueur peut faire des rotations pour placer les tuiles dans le sens qu'il souhaite.

Si le terrain est de taille n paire, on considérera que son centre est en ([n/2], [n/2]) avec [m] la partie entière inférieure de m

Cependant, le joueur peut faire des rotations pour placer les tuiles dans le sens qu'il souhaite.
Le but du joueur est de marquer le plus de points possible avec ses tuiles.
Les règles pour marquer des points sont :

- 1. On regarde le plus grand village (plus grand bloc de cases Ville contigües). Le joueur marque (nombre de case ville de ce village) points.
- 2. On compte les Forets. Le joueur marque 2x (nombre de cases forêt) points.
- 3. On regarde tout les Lacs. Le joueur marque 3x (nombre de cases Lac du lac -1) points.
- 4. On compte les Ressources. On alloue une Ressource par Usine tant qu'il reste des cases Ressource et des cases Usine. Chaque Ressource alloué à une Usine rapporte 4points. Une Usine ne peut traiter qu'une Ressource et une Ressource ne peut être allouée qu'à une Usine. Une Usine peut traiter une ressource d'une case qui ne lui est pas contigüe.

Contraintes de projet

- Vous devrez effectuer le développement en langage C ;
- Les fonctions du lot A devrons faire l'objet de tests unitaires ;
- Les codes seront compilés avec –Wall –Wextra, la compilation n'entrainera aucun warning ;
- Un makefile sera fourni avec chaque lot ;
- Les codes sources seront gérés avec un dépôt Git (Gitlab) transmis à votre encadrant;
- L'absence de fuite mémoire de vos lots sera vérifiée avec Valgrind ;
- Les codes seront commentés au format Doxygen et une doc pourra être générée à la demande grâce à une cible dans le Makefile ;
- Vos équipes respecteront les consignes de formations de groupe;
- Le suivi de projet sera réalisé grâce à un Trello. Votre encadrant devra être invité à votre projet Trello.

Contraintes de formation de groupe

- Chaque équipe sera constituée de 4 personnes
- Au maximum 1 formation par alternance par groupe
- Au maximum 1 admis sur titre par groupe
- Au maximum 1 étranger natif par groupe

Travail à réaliser

A. Préparation des données.

La première partie de votre travail consiste à tester le code qui vous permettra d'initialiser et manipuler vos structures de données :

- Allocation dynamique d'une grille de jeu carrée de taille variable ;
- Création d'une tuile connaissant les 6 terrains ;
- Libération de l'espace mémoire occupé par une grille ;
- Lecture du terrain sur la case (x, y) ;
- Initialisation de la grille et des tuiles de manière aléatoire ;
- Initialisation de la grille et des tuiles à partir d'un fichier (vous devez être capable de générer lesdits fichiers) ;
- Rotation d'une tuile ;
- Insertion d'une tuile connaissant son orientation et la coordonnée de sa case la plus en haut à gauche ;
- Fonction renvoyant le village associé à une ville ;

- Tests de recouvrement ;
  - "Liste des terrains recouverts sur le plateau"
  - "Liste des terrains recouverts par une tuile si on essaie de la poser sur le plateau"
  - "Test de la règle Une tuile ne peut être totalement recouverte à aucun moment de la partie"
  - "Test de la règle Toute nouvelle tuile posée doit recouvrir au moins une case de l'une des tuiles précédemment posée, la précédente case est alors oubliée"
  - "Test de la règle Aucune case Lac ne peut être recouverte."
- Test de la limitation de la zone de jeu ;
- Fonction permettant le retrait d'une tuile du plateau.

B. Réalisation d'un jeu avec affichage en terminal.

Vous réaliserez une application qui permet d'exécuter une partie de « Honshu ». Elle devra posséder au minimum les fonctionnalités suivantes :
Choix du fichier de chargement ou d'un chargement aléatoire au clavier ;

- Boucle de jeu :
  - Affichage dans le terminal de la grille et des tuiles restantes à placer ;
  - Sélection d'une tuile ;
  - Application possible de rotations à la tuile ;
  - Entrée des coordonnées d'insertion de la tuile dans la grille ;
  - Application des modifications à la grille ;
  - Test de fin de partie (plus de tuiles à placer).
  -

C. Comptage des points et réalisation d'un solveur.

Vous programmerez une fonction qui détermine une solution pour réaliser un maximum de points. Votre première façon de procéder sera par essais successifs. C'est à dire que vous testerez pour une situation donnée toutes les possibilités. Pour chaque situation atteinte vous referez de même.
L'algorithme est de la forme de celui en annexe A. Votre application sera modifiée au minimum de la manière suivante par rapport au lot B :

- Affichage du score atteint par la grille actuelle ;
- Rappel des moyens de gagner des points affiché lors de l'exécution d'une commande ;
- Affichage de la solution trouvée (grille et points) par votre solveur lors de l'exécution d'une commande.

Vous programmerez ensuite un solveur plus efficace. Pour cela, vous pourrez, par exemple, appliquer une stratégie particulière telle que maximiser les points des lacs ou des villes.

D. **Réalisation d'un jeu complet.**

Pour ce lot, vous avez le choix entre deux rendus possibles.

→ Soit vous implémenterez une interface graphique et rendrez votre jeu jouable à la souris
→ Soit vous implémenterez le fait qu'au début de chaque partie, les règles sont modifiés par une variante, choisie aléatoirement ou signifiée dans le fichier descriptif de la partie. Cette variante devra être affichée sur les règles en jeu. Les variantes possibles sont :

- Une case Lac vaut 2 (override règle 3.) ;
- Une plaine de quatre cases vaut 4pts (se rajoute) ;
- Une Usine peut accepter deux Ressources (override règle 4) ;
- Un carré de quatre cases village, s'il est dans la ville comptabilisée, donne 4pts bonus (ajout a la règle 1) ;
- Pour chaque forêt, la première case vaut 1, la seconde 2, la troisième 3, etc. (max5) (override règle 2) ;
- Aucun changement.

Livrables attendus à chaque lot

- Tous les codes source commentés, et documentés au format Doxygen.
- Un rapport de projet (algorithmes, répartition du travail, planning, …).
- Un fichier Makefile permettant de compiler le programme.
- Un README avec les instructions d'installation et utilisation (OBLIGATOIRE).

Dates de rendu des lots

- Date de livraison lot A : 25 mars 2018 (avec les tests unitaires)
- Date de livraison lot B : 8 avril 2018
- Date de livraison lot C : 29 avril 2018
- Date de livraison lot D : 25 mai 2018 (rendu soutenu en séance)

Dates des séances encadrées

- 2 mars 2018 - Démarrage du projet en Amphi
- 16 mars 2018
- 30 mars 2018
- 13 avril 2018
- 04 mai 2018
- 25 mai 2018 – Rendu final en Séance

Barème

- Tout code qui ne compile pas entraînera une note nulle !
- Retard sur un rendu : -1point par jour de retard.
- Absence de warning à la compilation : 1point.
- Absence de fuites mémoire (vérifiés avec Valgrind) : 1point.
- Lot A et rapport associé : 4points.
- Lot B et rapport associé : 3points.
- Lot C et rapport associé : 4points.
- Lot D, soutenance et rapport associé : 5points.
- Respect du Trello et du Git : 2points.
- Tout code mal commenté, illisible, mal indenté et/ou mal organisé se verra pénalisé.

Annexe A :

Fonction Honshu_Opt
- prend en entrée la taille n de la grille, une liste T de tuiles déjà posées (et leur position sur la grille), une liste L de tuiles à poser

- Si L est vide (toutes les tuiles sont posées), renvoyer le score actuel de la grille.
- Sinon
== Best <-- (-1)
== Pour toute tuile t de L, toute rotation r de t, et toute position (x, y) de la grille:
=== Si (x, y) et r sont une manière valide de poser la tuile t sur la grille
==== Poser la tuile sur la grille
==== Calculer récursivement le score Tmp_Score avec Honshu_Opt(n, T U t, L)
==== Best <-- max(Best, Tmp_Score)
==== Retirer la tuile de la grille
== Renvoyer Best

Annexe B : Format des documents fournis

B.1. : Format du fichier des tuiles de jeu

Nb_de_tuiles
id_1
X_1 X_2
X_3 X_4
X_5 X_6
[…]

avec :
- id_N l'id de la n-ième tuile (int)
- X_M le terrain de la M-ieme case (char)

B.2. : Format du fichier de partie

taille_grille taille_main
T1 T2 T3 […]
id_depart

avec :
- taille_grille : la taille de la grille de jeu
- taille_main : nombre de tuiles que possédera le joueur
- T1 T2 T3 […] : ids des tuiles du joueurs séparées par un espace
- id_depart : id de la carte pré-placée sur le terrain de jeu