Projet Ricosheep

NAUD Mathieu
MAM ALICE
TP10

Tables des matières

I) Fonctionnement de l'équipe et organisation du travail

Pour faire ce projet, nous avons travaillé via discord en partage d'écran. Nous avons travaillé ensemble jusqu'a la partie 3 (Mathieu a fait les parties une et deux et Alice a fait la partie trois). Cependant, Alice ayant décidé d'arreter la fac, Mathieu a aussi fait la quatrieme partie ainsi que le fait de sauvegarder les parties.

II) Guide d'utilisation du programme

Au début, le joueur devra choisir un plateau parmis ceux qui lui sont proposé en écrivant le nom dans la console. Ensuite, un fenêtre apparaitra avec un plateau de jeu, le joueur devra faire bouger les moutons en appuyant sur les touches directionnelles. Il aura aussi la possibilité de sauvegarder sa partie en appuyant sur la touche "s". Pour gagner, il devra placer un mouton sur chacune des herbes du plateau. Le joueur aura la possibilité d'abandonner s'il le souhaite, pour cela, il devra fermer la fenêtre.

III) Avancement du projet

-Avancement de la tâche 1:

Nous avons plusieurs variables importantes, "Plateau" qui est une liste de liste dont chacune des listes represente une ligne du plateau de jeu. Si la valeur vaut 0, la case est vide, si elle vaut "g", il y à de l'herbe, si elle vaut "b", il y a un buisson. La variable "Positions_Moutons" est une liste de couple (y, x) qui indique ou sont les moutons dans le plateau.

Dans un premier temps, nous avons réalisé les fonctions qui permette de savoir si le jeu est fini ou non. Pour cela, nous avons fait deux fonctions, "determine_pos_herbe" qui renvoie la liste des positions des herbes dans le plateau. Ensuite, la fonction "fin_jeu" determine si toutes les herbes sont occupées par des moutons. Pour cela, maintenant que l'on peut connaitre la liste des coordonnées des positions des herbes, on regarde si toutes les coordonnées des herbes sont occupées par des moutons.

Après avoir fait ses fonctions, nous devons nous occuper des déplacements. Pour cela, on doit déjà determiner dans quelle ordre les moutons se deplacent. Effectivement, il faut qu'ils se deplacent dans un ordre particulier pour pas qu'ils se bloquent entre eux. On a alors fait quatre fonction, une pour chaque direction. Par exemple, "tri_haut" tri les moutons du plus haut au plus bas. Les fonctions "tri_bas", "tri_gauche" et "tri_droite" marche de la même manière. Ensuite, on va regarder si le déplacement est possible. Nous avons à nouveau fait une fonction pour chaque direction. Les fonctions "est_possible_haut", "est_possible_bas" , "est_possible_gauche" et "est_possible_droite" fonctionne de la même facon. Par exemple, avec "est_possible_haut", on regarde si le mouton que l'on veut déplacer n'est pas déjà tout en haut, ensuite s'il y a pas de buisson au dessus de lui ni de mouton. Si il y a rien de tout ca, il peut alors se déplacer et la fonction renverra True. Après cela, les fonction "deplace_mouton_bas", "deplace_mouton_haut", "deplace_mouton_gauche" et "deplace_mouton_droite" permettent au groupe de moutons de se deplacer de une case. Pour ca, on tri les moutons en fonction de la direction que l'on souhaite appliquer. Ensuite, pour chaque mouton, on regarde s'il peut se deplacer, dans le cas ou il peut, on ajoutera dans une nouvelle liste les coordornées du moutons après son déplacement.
Maintenant que les moutons se déplace correctement de une case, nous avons fait une fonction "deplacement" qui va s'occuper de toutes les directions. Pour cela, on mettra en argument la

direction souhaitée ("Haut", "Bas", "Gauche" ou "Droite") et en fonction de la directions, on utilisera une fonction différentes, par exemple, si la direction est "Bas", on utilisera la fonction "deplace_mouton_bas". Cependant, on faisant ca, on deplacera le mouton que de une case comme dit précedement. On va donc appeler la fonction autant de fois que l'on peut. Par exemple, si un plateau fait huit case de haut, pour un deplacement vers le haut, on appelera la fonction huit fois.

Avec toutes ses fonctions, on peut faire un jeu fonctionnel.

-Avancement de la tâche 2:

La tâche 2 consiste à faire en sorte que l'on puisse charger une grille via un fichier .txt. Pour cela, nous avons fait une fonction "creer_plateau" qui prend une chaine de caractère et la converti en liste de liste qui sera alors le plateau du jeu. Pour cela, si le caractère vaut "_", on sait qu'il s'agit d'une case vide occupé par aucun mouton, "G" qu'il y a de l'herbe, "B" qu'il y a un buisson et "S" qu'il y a un mouton sur une case vide, si un caractère ne correspond à rien, la fonction renverra alors None. La fonction renverra après le plateau et la liste des positions des moutons.

Ensuite, la fonction "charger" prend en argument un nom de fichier et l'ouvre avec open(,"r"). on recupere alors les lignes du fichier avec .readlines() puis on ferme le fichier avec .close(). On a alors une chaine de caractère qui était dans le fichier .txt, on utilise alors la fonction "creer_plateau" pour obtenir le plateau et la position des moutons.

-Avancement de la tâche 3:

Pour faire la partie graphique du jeu, nous avons utilisé fltk. Nous avons créer plusieurs fonctions, chacune se chargeant d'afficher un élément différent. La fonction "affiche_grille" affiche dans une fenêtre les cases du plateau, pour cela, on utilisera "fltk.ligne()". De plus, cette fonction trace un nombre de ligne et colonne correspondant a la taille du plateau.

La fonction "affiche_buisson_herbe" affiche les buissons et herbe du jeu. Pour cela, elle fait un parcours de la liste Plateau et dès qu'il y a "b" ou "g", la fonction affichera l'image du buisson ou herbe aux coordonnées correspondant avec fltk.image().

Les fonctions "affiche_buisson_herbe" et "affiche_grille" ne seront appelé que une fois, donc avant la boucle principale car les grilles, buissons et herbes sont fixes.

La fonction qui s'occupe des moutons, "affiche_mouton", sera par contre appelé a chaque fois dans la boucle principale. Comme pour "affiche_buisson_herbe", on parcours une liste, cette fois Positions_moutons pour connaitre les coordonées des moutons et affichera l'image des moutons aux coordonnées correspondant. Cependant, les moutons ne sont pas fixe au cours de la partie, on met donc un tag aux images de mouton. Nous effacerons alors à chaque fois les moutons avec "fltk.efface("moutons")" puis on les affichera à nouveau. Cela sert a faire en sorte que les images de moutons ne se multiplie par sur la fenêtre.

Pour le jeu en lui même, on affichera dans un premier temps une fenêtre de 1000*1000 puis on appele les foncions "affiche_grille" et "affiche_buisson_herbe". Ensuite, dans la boucle principale, on appele "affiche_moutons". On verifira aussi si l'utilisateur appuie ou non sur une touche avec "fltk.donne_ev". Si c'est le cas et qu'il appuie sur une flèche directionnelle, on appelera deplacement avec comme direction la flèche qu'on a pressé. On detectera aussi si l'utilisateur ferme ou non le programme, si c'est le cas, on considère que l'utilisateur à perdu. Si l'utilisateur perd ou gagne, un

écran s'affiche avec un écran ou il est indiqué s'il a gagné ou perdu.

-Avancement de la tâche 4:

Pour le solveur, nous avons pas reussi à faire mieux qu'un solveur qui se base sur l'aléatoire. La fonction "solveur" va donc faire des coups au hasard jusqu'a ce que les conditions de victoire soient remplis. Si elle trouve une facon de gagner, elle le renverra.

-Amélioration ajouté:

On à créé deux fonctions qui ont pour but de sauvegarder une partie. Pour cela, elle fonctionne dans le sens inverse des fonctions de la tâche deux. La fonction "Plateau_vers_texte" va prendre le Plateau et Positions_Moutons en arguments. Comme pour "creer_grille", si dans le plateau il y a un buisson, alors dans la chaine de caractère, il y aura "b", idem pour les buisson, case vide et moutons. La fonction "sauvegarder" va ouvrir un fichier avec open(, "w"). Puis la fonction va ajouter les lignes du plateau dans le fichier avec .write(), après chaque ligne, on mettra "\n" pour un retour à la ligne.

-Les problèmes ou possibles améliorations:

Pour les possibles améliorations, nous aurions pu faire une interface graphique pour choisir le plateau que l'on veut en début de jeu. Pour cela, il aurait fallut creer une autre fenêtre avant la fenêtre de jeu, y mettre des rectangles avec "fltk.rectangle" et determiner ou on clique avec "fltk.abscisse" et "fltk.ordonnee".

Nous aurions aussi pu améliorer la fonction "deplacement". En effet, on fait plus d'appel de fonction que necessaire.

Nous aurions aussi pu faire un solveur plus "intelligent" qui ne se base pas sur l'aléatoire.

Ensuite, pour les problèmes, nous en avons trouvé deux.
Le premier, est le fait qu'on puisse pas afficher de grille autre que rectangulaire ou carré et les grilles qui ont une longueur ou largeur supérieur a 10 case. Dans les deux cas, on peut techniquement y jouer car marche si on joue sans la partie graphique, donc uniquement avec la console. Le problème vient du fait que l'on savait pas afficher une grille triangulaire et aussi que la fenêtre fait 1000*1000, les cases faisant 100 de longueur, au bout de 10 cases, elles s'afficheront en dehors de la fenêtre de jeu.
Le deuxième problème est plus grave, en effet, au bout d'un certains nombre d'action, le jeu peut planter avec le message "fail to allocate bitmap". Nous avons vu des méthodes pour éviter ce problème mais pas avec le module fltk.