FLEISER Victor – VERMEREN Mathis – EISE3 – Architecture des ordinateurs – Jeu du moulin

Rapport – Jeu du moulin

Sujet :

On a du recréer le jeu du moulin en assembleur RiscV. Voici une explication des règles du jeu :

- Le jeu se déroule en 3 phases, et se termine lorsequ'un joueur perds en ayant moins de 3 pions ou aucun mouvement possible.

- Phase 1 : à tour de rôle, chaque joueur pose un pion sur une case vide, la phase 1 se termine une fois que les deux joueurs ont placés leurs 9 pions.

- Phase 2 : à tour de rôle, les joueurs déplacent un pion sur une case vide adjacente (doit être reliée par un chemin), la phase 2 se termine une fois que un joueur n'a plus que 3 pions.

- Phase 3 : Identique à phase 2, mais le(s) joueur(s) ayant 3 pions peu(t/vent) à présent «sauter», lors du déplacement d'un pion, le pion peut être déplacé sur n'importe quelle case vide du terrain.

- A tout moment du jeu, si un déplacement/placement de pion conduit à 3 pions de même couleur alignés, cela s'appelle former un moulin, le joueur ayant formé un moulin capture un pion adverse ne faisant pas partie d'un moulin (la capture consiste juste à enlever du plateau le pion).

- Exception : lors de la capture, si tous les pions adverses font partie d'un moulin, alors le joueur peut capturer le pion adverse de son choix sans restriction.

Fonctionnement du jeu assembleur/C :

- Le terminal affiche toujours deux plateaux : le premier correspond au plateau de jeu, et le deuxième correspond à un plateau d'indices pour voir facilement quel case correspond à quel indice.

- Les symboles – et | sont utilisés pour indiquer les chemins, # indique une case vide, O indique une case du joueur 1, X indique une case du joueur 2, les nombres dans le plateau de droite correspondent aux indices des cases.

- Le terminal explique toujours en quelques lignes quel joueur doit faire quel action, le programme demandera toujours d'entrer un nombre comme input.

- Si l'**utilisateur entre un nombre incorrect** (indice hors du plateau ou un mouvement non accepté) le programme affichera un message et **demandera de recommencer**.
Par contre **le programme n'est pas prévu pour gérer les chaînes de caractères**.

Stratégie pour la réalisation du projet :

La stratégie utilisée pour le projet a été la suivante :
- Compréhension du jeu puis mise en place du résultat visuel attendu (à quoi ressemblera le jeu final en ASCII) (fichier : ressources/model.txt, /!\ certains commentaires créés à l'origine sur ce fichier ne sont plus à jour avec la version finale)

FLEISER Victor – VERMEREN Mathis – EISE3 – Architecture des ordinateurs – Jeu du moulin

- Création du code complet en C :
    - Mise en place de la manière dont seront stockées les données du plateau : 24 structures cases (cf Structure utilisée)
    - Création au fur et à mesure des fonctions nécessaires (cf Fonctions utilisées)
    - Test du bon fonctionnement logique du programme dans toutes les circonstances imaginables

- Traduction du code assembleur :
    - Création du squelette du fichier : toutes les fonctions sont déclarées/commentées et retournent directement. Le main et commencé, et la fonction pour quitter fonctionne.
    - Mise en place des notations pour les étiquettes (cf Notations étiquettes)
    - Traduction depuis le C ainsi que test (lorsque possible) des fonctions.
    - Pour la fonction d'initialisation on a créé un programme C pour automatiser la traduction (cf Ressource crées - Code C d'automatisation)
    - Test du bon fonctionnement logique du programme dans toutes les circonstances imaginables

Ressources crées :

Multiples ressources ont été crées pour faciliter la réalisation du projet, elles sont toutes disponibles dans le fichier ou en ligne sur GitHub :

- Dépôt GitHub (https://github.com/VictorFleiser/Moulin) : nous avons créé le dépôt GitHub pour pouvoir partager les fichiers ainsi que voir historique des changements.

- Code C du jeu du moulin (moulin1.c) : Nous avons d'abord créé le jeu entier en C pour pouvoir créer toute la logique du jeu dans un langage plus haut niveau et seulement ensuite traduire en assembleur une logique qui fonctionne.
Note : les variables utilisées en C ne sont pas globales et sont souvent repassées en paramètre de fonctions, en assembleur ce sont directement des variables globales et ne sont pas repassées en paramètres.

- Tableur (ressources/moulin_struct_guide.ods) : Ce tableur montre la manière dont est stocké notre plateau dans la mémoire ainsi que la manière d'accéder l'adresse de n'importe quel élément du tableau, (24 structures correspondant aux 24 cases du plateau, avec chaque case étant composée de 24 octets correspondant aux champs des structures).
Note : image du tableur aussi disponible ci dessous :

	Tableau[24]				t_case				
case numero	Adresse :		numero elem	Element :	Adresse :				
0	0		0	valeur	0				
1	24		1	symbole	4		-	-	-
2	48		2	vn	8				
3	72		3	ve	12				
4	96		4	vs	16				
5	120		5	vo	20				
6	144								
7	168								
8	192								
9	216			ex:		adresse			
10	240			Tableau[10].ve		10*24+3*4 = 240+12 = 252			
11	264								
12	288								
13	312								
14	336								
15	360								
16	384								
17	408								
18	432								
19	456								
20	480								
21	504								
22	528								
23	552								

FLEISER Victor – VERMEREN Mathis – EISE3 – Architecture des ordinateurs – Jeu du moulin

- Code C d'automatisation (automatisation/automatisation.c et automatisation/voisins.txt) : Nous avons créé un programme C pour automatiser la traduction des 65 lignes C de l'initialisation des voisins de chaque case. Traduire manuellement chaque ligne aurait pris un certain temps à cause des calculs d'adresses de stockage, de plus il y avait un grand risque d'erreur de calcul. Le programme C prend les lignes C copiées dans voisins.txt et affiche dans le terminal les lignes assembleur correspondantes.

Variables Globales :

Plusieurs étiquettes ont été utilisées dans .data pour stocker des variable globales :
- var_J1_nbr_de_pions et var_J2_nbr_de_pions :Nombre de pièces sur le terrain
- var_phase : Phase du jeu (0 = fin du jeu ; 1 = pose des pions ; 2 = mouvement/saut des pions)
- var_tour_j : Tour de quel joueur (1 = tour de Joueur 1 ; 2 = tour de Joueur 2)
- var_tour : Numéro du tour

Il y a également d'autres sections dans le .data que les variables globales :
- var_tab_plateau : Tableau des cases (24 cases * 24 octets = 576 octets)
- Les chaînes de caractères utilisées.

Structure utilisée :

Nous avons eu besoin de une seule structure pour ce projet, la structure des cases t_case. t_case prend 24 octets (dont la représentation graphique est disponible dans Ressource crée).
Dans l'ordre, les champs sont :

.valeur (integer, 4 octet) : correspond au numéro du joueur possédant la case :
0 = vide
1 = case Joueur 1
2 = case Joueur 2

.symbole (caractère ascii, 1 octet aligné sur 4 octets) : correspond au symbole à afficher pour la case
# = vide (ascii = 35)
O = case Joueur 1 (ascii = 79)
X = case Joueur 2 (ascii = 88)

.vn (integer, 4 octet) : correspond à l'indice de la case au Nord de cette case, -1 si il n'y a pas de case voisine au Nord.

.ve (integer, 4 octet) : Idem pour le voisin Est

.vs (integer, 4 octet) : Idem pour le voisin Sud

.vo (integer, 4 octet) : Idem pour le voisin Ouest

Fonctions utilisées :

Toutes les fonctions sont présentes et bien commentées dans le fichier moulin1.c pour une explication plus en détail du fonctionnement.

FLEISER Victor – VERMEREN Mathis – EISE3 – Architecture des ordinateurs – Jeu du moulin

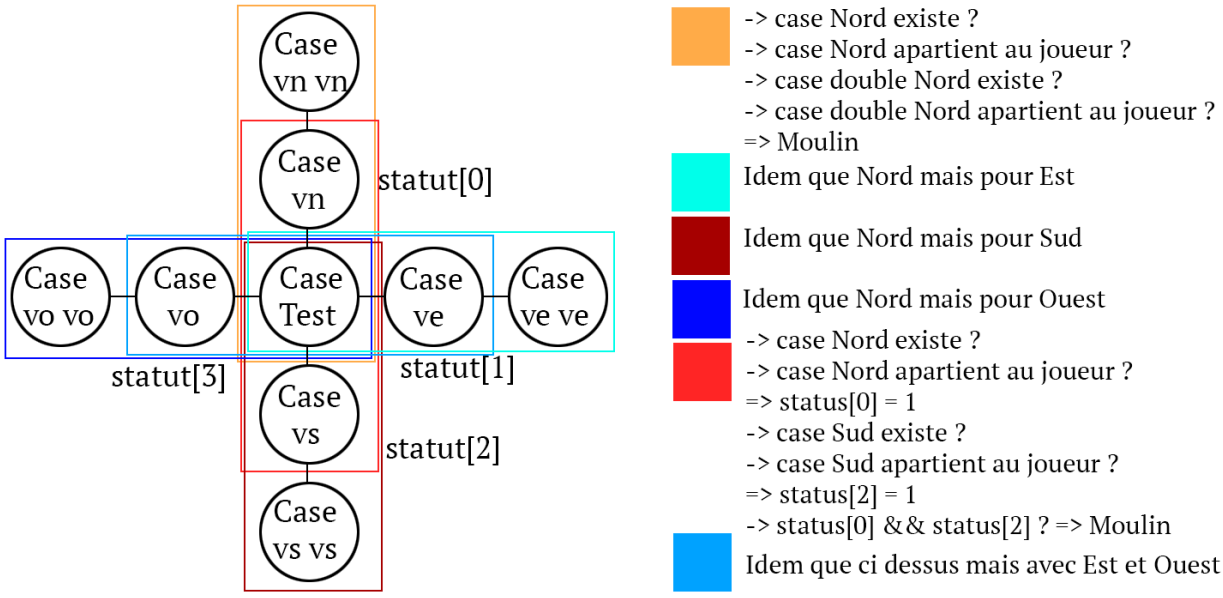
Voici une liste des fonctions utilisées ainsi que une explication plus ou moins détaillée en fonction de la complexité de la fonction et des potentiels problèmes rencontrés.

affiche_plateau : Cette fonction affiche les deux plateaux, elle prend beaucoup de lignes en assembleur car elle change d'appel système à chaque fois qu'elle passe d'un affichage de chaîne de caractère à une variable et vice versa.

initialisation : Cette fonction initialise toutes les valeurs du tableau de t_case correspondant au plateau vide, ceci inclut la mise en place du réseau de voisin dont l'écriture a été automatisée (cf Ressources crées - Code C d'automatisation)

test_moulin : Cette fonction teste si la case correspondant à l'indice en paramètre fait partie d'un moulin. Cette fonction stocke un tableau de statuts (4 registres) (le statut correspond à un 1 si le voisin dans cette direction appartient aussi au joueur), ces statuts seront remplis puis testés à la fin pour savoir si la case est au centre d'un moulin.
Cette fonction regarde si un voisin nord existe, si oui elle regarde si ce voisin nord appartient au même joueur, si oui elle change le statut correspondant est regarde si un voisin double nord existe, si oui elle regarde si il est appartient également au joueur, si oui on sait directement que la case fait partie d'un moulin.
On fait de même pour les autres directions, puis on teste les registres de statuts pour savoir si la case est au centre d'un moulin.
La fonction retourne 1 si la case fait partie d'un moulin, 0 sinon.

Schéma des tests effectués :



capture : cette fonction demande au joueur quelle pion capturer et le capture.
La fonction teste en premier si il existe au moins un pion adverse n'étant pas protégé par un moulin, en fonction de ce test elle affichera différents messages et testera (ou pas) si le pion à capturer fait partie d'un moulin (cf l'exception de capture dans les règles du jeu). Lorsque le nombre entré n'est pas un nombre valide, la fonction demande à l'utilisateur de recommencer.
Enfin la fonction mets à jour le plateau et le nombre de pions du joueur affecté.

FLEISER Victor – VERMEREN Mathis – EISE3 – Architecture des ordinateurs – Jeu du moulin

place_pion : Cette fonction demande à l'utilisateur l'indice de la case sur laquelle placer le pion, elle demandera de recommencer si l'indice est invalide. Elle met à jour le plateau et le nombre de pions du joueur. La fonction retourne l'indice joué.

deplace_pion : Cette fonction demande à l'utilisateur le pion à déplacer puis la destination de ce pion. Si le joueur entre des nombres invalides, la fonction demande de recommencer.
La fonction met à jour le plateau puis renvoie l'indice de la case de destination.

saut_pion : Cette fonction est similaire à deplace_pion, mais ne teste pas si les cases sont voisines.

fin_de_partie : Cette fonction affiche le vainqueur puis quitte correctement le programme.

main : Initialise le jeu puis entre une boucle while qui dure tout le jeu. À chaque tour, main incrémente le tour, modifie le joueur actif, affiche les informations de jeu, et, en fonction de la phase du jeu, effectue les différentes fonctions :

phase de pose des pions :
- → place_pions
- → test_moulin
  - → capture
- → teste si la phase est finie
  - → passe en le jeu en phase de déplacements

phase de déplacements :
- → test si un joueur à 3 pions
  - → saut_pion
  - sinon
  - → test si un coup existe
    - → deplace_pion
    - sinon
    - → fin du jeu
- → test_moulin
  - → capture
- → teste si un joueur a moins de 3 pions
  - → fin du jeu

Notations des étiquettes :

Les notations pour les étiquettes suivent le format suivant :

([précisions])_([if/while/for/else])_[fct/var/str]_[nom_fct/var/str]_[num_élément_dans_fct/data]_([précision_pour_string])

(les [champs] en parenthèses ne sont mis que si applicables à l'étiquette en question)
(fct = fonction, var = variable, str = chaine de caractères)

ex : suite_if_fct_capture_4