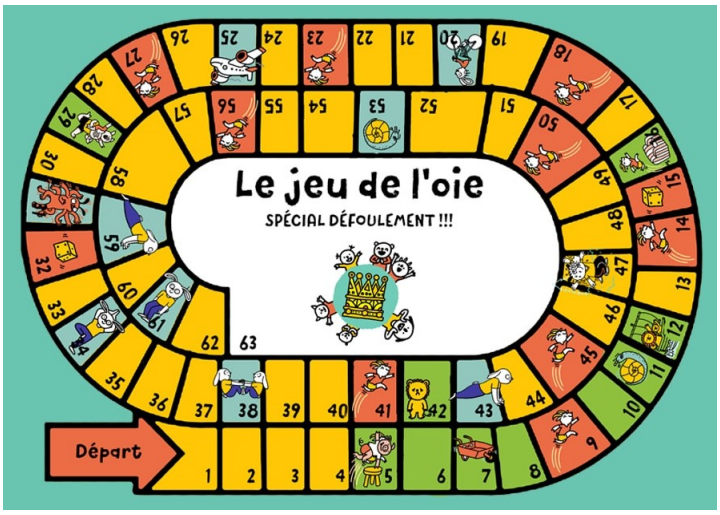
DM de Programmation C 2023-2024

Résumé

Lors de ce projet, il vous sera demandé d'implémenter une variante du jeu de l'oie. L'interface graphique se fera via le terminal, l'utilisation de la libMLV n'est pas demandée.



1 Introduction

Le jeu de l'oie se joue entre 2 et 4 joueurs, sur un plateau ayant traditionnellement 64 cases, numérotées de 0 à 63. Ici, nous implémenteront une variante sur un plateau de 100 cases, numérotées de 0 à 99. Les joueurs commencent tous en case 0, et jouent chacun leur tour. Lorsque c'est le tour d'un joueur, il lance 2 dés à 6 faces et avance son pion d'un nombre de cases égal à la somme des deux dés. Si le joueur atterrit sur la case 99, il a alors gagné. Si le total obtenu aux dés devrait amener le pion au delà de la case 99, le pion recule du nombre de case restant à avancer. Par exemple, si le joueur est en case 93 et fait 6 et 4 (pour un total de 10), il avance d'abord de 6 cases jusqu'en case 99, puis recule des 4 cases restantes, terminant son mouvement en case 95. Une fois la case finale déterminée, on regarde sur quel type de case le joueur s'est arrêté :

- Des case neutres sans propriétés spéciales. Ces cases sont les plus fréquentes sur le plateau de jeu.
- Des cases "Oie", qui permettent au joueur de se re-déplacer du même nombre de cases que son lancé de dés. Les oies sont en général disposées toutes les 9 cases (sans compter certaines d'entres elles disposées aléatoirement). A noter que dans notre version, nous imposerons qu'il n'y ait pas de case Oie après la case 92 (on vous laisse réfléchir à la raison).

— Une case "Trou", située en case 53. Si un joueur tombe dans le trou, ce dernier doit attendre qu'un autre joueur prenne sa place (en tombant *exactement* sur la même case) pour se déplacer à nouveau. Ce joueur devra à son tour, pour sortir du trou, attendre qu'un autre joueur y tombe à sa place.
— Une case "Raccourci", située en case 8 qui permet de se rendre directement en case 16.
— Une case "Prison" située en case 74. Le joueur qui tombe sur la prison, il se retrouve bloqué derrière les barreaux si personne ne s'y trouve déjà. Si un autre joueur est présent, les deux joueurs parviennent alors à s'évader et pourront jouer au prochain tour. Les joueurs hors de prison ne peuvent pas aider un joueur en prison à s'en évader.
— Une case "Hotel", située en case 31. Le joueur se repose et passe son tour deux fois. C'est-à-dire, chaque autre joueur jouera 2 fois avant que le joueur à l'hôtel ne puisse rejouer.
— Une case "Labyrinthe", située en case 65. Le joueur qui tombe sur un labyrinthe subit un malus et recule de 13 cases, se retrouvant alors en case 52, juste avant le trou.
— Une case "Tête de mort", située en case 94. Le joueur retourne à la case 0 (et a trop le seum !)
— À part pour les cases Prison et Trou, si le joueur (joueur 1) tombe sur la case d'un autre joueur (joueur 2), joueur 2 se retrouve déplacé sur la case sur laquelle joueur 1 a débuté son tour et **l'effet de la case s'applique à joueur 2**. Notez que si joueur 1 atterrit sur la case où se trouve joueur 2 en passant par des cases spéciales, c'est tout de même la case sur laquelle joueur 1 a démarré son tour où sera envoyé joueur 2. De même si joueur 1 est revenu en arrière en "dépassant" la case 99.

On compte de plus deux cas spéciaux en tout début de partie :
— Si un joueur fait 3 et 6 pour son tout premier lancé de dés de la partie, il se rend directement en case 40.
— De même, si un joueur fait 4 et 5 pour son tout premier lancé de dés de la partie, il se rend directement en case 89.

Notez que ces cas ne s'appliquent pas lorsqu'un joueur est revenu au début du plateau après s'être arrêté sur la tête de mort. Le joueur aura alors la possibilité de tomber sur plusieurs cases "Oie" à la suite s'il fait 9.

2 Structures de données

Le tableau `Plateau` contient 100 caractères :
— `' '` si c'est une case neutre
— `'O'` si c'est une oie
— `'T'` si c'est un trou
— `'R'` si c'est un raccourci
— `'P'` si c'est une prison
— `'H'` si c'est un hôtel
— `'L'` si c'est un labyrinthe
— `'X'` si c'est une tête de mort

Le tableau `Positions` contient un entier pour chaque joueur indiquant sa position sur le plateau. En début de partie, tous les joueurs sont à la position 0.

Le tableau `Attente` contient un entier pour chaque joueur :
— 0 si le joueur peut jouer son tour,
— 1 ou 2 si le joueur est sur une case hôtel en train d'attendre,
— -1 si le joueur est bloqué en prison ou dans un trou

3 Fonctions

La fonction `int avancerJoueur(char plateau[], int positions[], int attente[], int joueur_courant, int nb_joueurs), int des[2], int premier_tour`) effectue le tour de jeu d'un joueur. Elle renvoie le numéro du joueur s'il gagne la partie et -1 si la partie n'est pas finie. Elle constitue la principale fonction du programme et fonctionne comme suit :
— On regarde dans le tableau `Attente` si le joueur peut jouer,
— Si le joueur peut jouer, on lance 2 dés à 6 faces (fournis par le paramètre `des[2]`) et on calcule la nouvelle position attendue (attention au retour en arrière si le joueur dépasse la dernière case du tableau)
— Dans le cas où c'est le premier tour de jeu (indiqué par le paramètre `premier_tour`), on gère les cas spéciaux, à savoir quand un jour fait 3 et 6, ou 4 et 5.
— En fonction de la case sur laquelle le joueur est censé s'arrêter, on teste si l'on se trouve sur une case spéciale ou si l'on entre en collision avec un autre joueur et résout de manière appropriée.
— On met à jour les tableaux `Attente` et `Positions`.

Important : l'ordre dans lequel les tests de cas spéciaux sont effectués est très important. Notamment, on doit tester si la nouvelle position dépasse 99 en premier, et résoudre les éventuelles collisions en dernier.

La fonction `void collision(char plateau[], int positions[], int attente[], int nb_joueurs, int joueur_courant, int nouvelle_position)` est appelée par la fonction `avancerJoueur` et gère les cas de collision, c'est-à-dire qu'elle teste si le joueur courant termine son mouvement sur la case d'un autre joueur. Elle prend en entier le tableau des positions des joueurs (`int positions[]`) et la nouvelle position du joueur courant (aussi passé en paramètre) à la fin de son mouvement. Elle modifie le tableau `positions` (et éventuellement `attente`) de manière appropriée en cas de collision.

4 Affichage en spirale

La fonction `void conversion(int pos, int* x, int* y)` prend en arguments la position d'un joueur en coordonnées linéaires (entre 0 et 99), et les adresses de deux entiers `x` et `y`. Elle modifie les valeurs `x` et `y` pour représenter les coordonnées dans un tableau à 2 dimensions de taille 10×10 correspondant à la case `pos`. Par exemple, la case 53 du tableau a pour coordonnées x = 8 et y =5 (comme d'habitude, les lignes et colonnes sont numérotées de 0 à 9).

La fonction `void afficherPlateau(int plateau[], int positions[], int nb_joueurs)` prend en argument un tableau de 100 entiers représentant la position des cases spéciales sur le plateau de jeu, un tableau positions contant les positions des différents joueurs sur le plateau

(en coordonnées linéaires, c'est-à-dire entre 0 et 99), ainsi que le nombre de joueur. Elle affiche l'état du plateau chaque fois qu'un joueur débute son tour. Le plateau est représenté sous la forme d'une grille où les joueurs se déplacent en spirale.

Important : Dans le cas où plusieurs joueurs occupent la même case (notamment au début du jeu quand tous les joueurs sont sur la case 0), le joueur courant est affiché en priorité. Si plusieurs joueurs sont sur une même case autre que celle où se trouve le joueur courant, c'est le joueur qui minimise $(joueur - joueur\_courant)\%nb\_joueurs$ qui est est affiché sur le plateau. De plus, afin de garder une trace de la position des différents joueurs, on affichera en texte sous le plateau la liste des positions. Cet affichage sera de la forme "Joueur 0 : case positions[0], Joueur 1 : case positions[1], Joueur 2 : case positions[2], etc.".

Voir la figure ci-dessous pour une représentation du tableau avec coordonnées en spirale.

0	1	2	3	4	5	6	7	8	9
35	36	37	38	39	40	41	42	43	10
34	63	64	65	66	67	68	69	44	11
33	62	83	84	85	86	87	70	45	12
32	61	82	95	96	97	88	71	46	13
31	60	81	94	**99**	98	89	72	47	14
30	59	80	93	92	91	90	73	48	15
29	58	79	78	77	76	75	74	49	16
28	57	56	55	54	53	52	51	50	17
27	26	25	24	23	22	21	20	19	18

La figure ci-dessus indique les coordonnées du plateau de jeu avec déplacement en spirale. Les joueurs démarrent tous en case 0 (bleue). Un joueur gagne s'il termine son tour exactement en case 99 (rouge).

5 Programme principal (`main`)

5.1 Base

Le programme demande un nombre de joueurs (entre 2 et 4) puis à chaque tour, on propose au joueur courant d'indiquer les valeurs des dés ou d'arrêter la partie. L'état du plateau est affiché à la fin de chaque tour.

On s'assurera que les valeurs entrées par l'utilisateur sont correctes (nombres de joueurs, et valeurs des dés).

5.2 Fichier de sauvegarde

Les parties de jeu de l'oie pouvant être longues, on se donne la possibilité de sauvegarder une partie pour la reprendre ensuite.

Une sauvegarde de notre jeu de l'oie est un fichier texte contenant au moins deux lignes. La première ligne contient seulement les caractères `JO` (pour Jeu de l'Oie). La deuxième comprend

Exemple d'exécution (sans l'affichage du plateau)

```
$ ./jeudeloie
Combien de joueur ? 3
Pour chaque tour, indiquer les valeurs des deux dés ou taper 'q' pour quitter
Joueur 1: 1 5
<AFFICHAGE DU PLATEAU>
Joueur 2: 2 3
<AFFICHAGE DU PLATEAU>
Joueur 3: q
Arrêt
```

le nombre de joueurs (entre 2 et 4). Chacune des lignes suivantes contient deux valeurs entre 1 et 6 correspondant aux valeurs de dés des tours sauvegardés.

```
1  JO                  JO <- permet de reconnaître notre format
2  3                   <nombre de joueurs>
3  1 5                 <dé 1> <dé 2>
4  2 4                 <dé 1> <dé 2>
5  3 1                 ...
```

FIGURE 1 – Format du fichier de sauvegarde et exemple

Pour charger et/ou sauvegarder une partie, le nom du fichier est donné en argument au programme. Si le fichier n'existe pas, il est créé au début de la partie (une fois le nombre de joueurs connu) et mis à jour à chaque tour de jeu en ajoutant les valeurs des dés sur une nouvelle ligne. Si le fichier existe mais ne respecte pas le format le programme signale une erreur et s'arrête. Si le fichier existe et est dans le bon format, les lancés dés sont simulés. Puis, si la partie n'est pas terminée, le programme redevient interactif (tout en mettant à jour le fichier de sauvegarde à chaque tour).

Exemple avec chargement d'une partie valide

```
$ ./jeudeloie ma_sauvegarde.jo
Chargement de partie : 4 joueurs, 12 tours simulés, partie non terminée
Pour chaque tour, indiquer les valeurs des deux dés ou taper 'q' pour quitter
Joueur 3: 1 5
<AFFICHAGE DU PLATEAU>
Joueur 4: 2 3
<AFFICHAGE DU PLATEAU>
Joueur 1: q
Arrêt, partie sauvegardée dans ma_sauvegarde.jo
```

6 Temps de vol

L'idée est d'explorer combien il faut en moyenne de tours pour terminer une partie et si il arrive que la partie ne termine pas. Il faut donc être capable de simuler une partie. Pendant

la simulation nous allons réaliser les déplacements (en utilisant des valeurs aléatoires pour les dés) des joueurs, compter le nombre de tours de jeu, jusqu'à ce que l'un des joueurs ait gagné, ou bien que tous les joueurs soient bloqués.

Pour lancer une simulation il faut utiliser l'option `-W` suivie de deux valeurs *(nombre de joueur),(nombre de simulations)*, par exemple : `-W 4,200` pour 200 simulations d'un jeu à 4 joueurs. Une deuxième option `-S` graine permet de spécifier la **graine** entière pour la génération aléatoire.

Exemple d'exécution des temps de vol

```
$ ./jeudeloie -W 1,1000 -S 33
Calcul des temps de vol : 0001
...
Nombre de simulation : 1000 avec : 1 joeurs et une seed: 33
Nombre de tours moyen: ___
Nombre de tours minimum: ___
Nombre de tours Maximum: ___
Nombre de fois ou le jeu a bloqué: ___
```