Projet: Programmation du jeu Quoridor

Année 2015-2016

L'objectif du projet est de programmer le jeu Quoridor. Les règles de ce jeu et les différentes fonctionnalités à programmer sont détaillées dans la suite du sujet.

Les consignes générales sont :

- *Le projet est individuel.*
- *Un* `Makefile` *doit permettre de compiler le programme sans erreurs. Le nom de l'exécutable doit être* `quoridor`*.*
- *Il faut rendre un dossier archivé (.tar ou .tar.gz) contenant tous les fichiers nécessaires à la compilation du programme et à son exécution. Le nom de l'archive est le nom de l'étudiant (avec l'extension .tar ou .tar.gz).*
- *L'archive doit aussi contenir un fichier texte* `rapport.txt` *qui contient :*
  - *les règles de fonctionnement du jeu (comment déplacer un pion, comment ajouter un mur, etc...),*
  - *la présentation des structures utilisées, et du découpage du programme en fichiers/fonctions,*
  - *l'explication d'une (ou plusieurs) fonction dans laquelle on a utilisé une astuce de programmation, si il y en a,*
  - *si besoin, des commentaires critiques sur son projet : expliquer les difficultés rencontrées, ce que vous changeriez si vous recommenciez de zéro, etc...*
- *Le projet doit être chargé sur e-campus2 avant le mercredi 16 décembre à 23h59, ou bien envoyé par mail à votre responsable de TD si vous rencontrez des problèmes avec e-campus2.*

Les projets ne respectant pas ces consignes ne seront pas évalués.

Au-delà de la réalisation du jeu, les critères suivants seront pris en compte pour la notation :

- *La modularité du programme.*
- *La lisibilité du programme.*

Ce qui vous est fourni :

- *La librairie graphique utilisée en première année via les fichiers* `graphics.c`*,* `graphics.h`*, la police* `verdana.ttf` *et* `couleur.h`*, ainsi que sa documentation en annexe du sujet qu'il est recommandé de consulter avant de débuter, notamment les fonctions concernant la gestion d'événements clavier et souris.*
- *Un fichier* `Makefile` *de base (qu'il faudra compléter) qui permet de compiler la libraire graphique : celle-ci génère un fichier binaire* `graphics.o` *avec la commande* `make graphics.o`*. Il faut parfois modifier la ligne d'accès à la librairie SDL dans le fichier* `graphics.h` *de* `#include <SDL.h>` *en* `#include <SDL/SDL.h>`

Description du jeu

Certains passages sont extraits de la page `https://fr.wikipedia.org/wiki/Quoridor`

Vous pouvez vous familiariser avec le jeu en y jouant à la page
`http://danielborowski.com/quoridor-ai/v2/display.html`

Position initiale du jeu

- on programmera la version à 2 joueurs
- 1 plateau de taille 9x9 qui n'a pas de murs au début,
- chaque joueur contrôle un pion de couleur différente qui débutent l'un au milieu de la rangée du bas, l'autre au milieu de la rangée du haut.

Objectif L'objectif de chaque joueur est d'amener son pion sur n'importe quel case de la rangée opposée du plateau.

Règles du jeu Le jeu Quoridor comporte vingt murs, chaque joueur en possédant 10. Ce sont des plaquettes de bois longues de deux cases qui peuvent être placées dans l'interstice entre les cases. Les murs restreignent le déplacement de tous les pions, qui doivent les contourner. Une fois placés sur le plateau, ils ne peuvent plus être retirés ni bougés de la partie. Quand vient son tour de jouer, un joueur doit choisir entre déplacer son pion, ou (si il lui en reste) placer un mur.

Les pions peuvent être déplacés d'une case à une case adjacente, mais pas en diagonale. Si un pion est situé sur une case dont une case voisine comporte elle-même un autre pion, alors le premier pion peut sauter au-dessus du second pion. Si un mur empêche d'accéder à cette case, alors le pion sauteur peut être posé sur n'importe quelle autre case adjacente au pion sauté.

Les murs sont plaçables entre deux paires de cases. Ils ne peuvent se chevaucher. Ils ne doivent pas non plus interdire l'accès aux lignes d'arrivées des pions adverses.

Liste des fonctionnalités à programmer

Les fonctionnalités du jeu à implémenter sont les suivantes :

1. Implémenter les règles du jeu pour que 2 humains puissent y jouer ;
2. possibilité de sauvegarder une partie en cours et de la charger au début de l'exécution du programme ;
3. possibilité de revenir plusieurs coups en arrière durant la partie ;

D'autres fonctionnalités peuvent être ajoutées, il faudra le signaler dans le fichier `rapport.txt` joint. Par exemple faire une intelligence artificielle pour qu'un humain puisse jouer contre l'ordinateur.

Conseils

Une des difficultés du projet est de définir des structures de données adaptées qui permettront d'avoir un programme lisible et des fonctions faciles à programmer. Il faut donc y réfléchir avant de se lancer dans l'écriture du programme.

Pour la lisibilité, il bon d'user (et d'abuser) de la possibilité de définir des alias avec les commandes `enum` et `typedef`. Par exemple, pour définir une direction on peut procéder ainsi :

```
enum direction{HAUT, BAS, GAUCHE, DROITE};
typedef enum direction Direction;
...
Direction d;
if(d == GAUCHE)...
```

qui est plus lisible que

```
int d; //0 si Haut, 1 si Bas, 2 si Gauche, 3 si Droite
if(d == 2)...
```

Il faut également construire un ensemble de fonctions simples qui permettent de faire des opérations de base sur le jeu. Par exemple, une fonction qui à partir d'un point retourne la position du plateau correspondante, une fonction qui renvoie la nouvelle position à partir d'une position et d'une direction donnée, etc...

Pour vérifier que l'accès à la rangée objectif n'est pas bloquée en plaçant un mur (dernière règle du jeu), vous pouvez suivre la procédure suivante ; l'algorithme va calculer toutes les positions atteignables par un pion :

- on utilise un tableau à 2 dimensions de la taille du plateau qui est initialisé à 0 sauf la case sur laquelle se trouve le pion qui est initialisée à 1. La valeur 1 signifie que la case est atteignable par le pion.
- On itère ensuite la boucle suivante tant qu'il y a eu une modification du tableau à l'itération précédente (et on le fait au moins une fois au début) :
  - On parcourt l'ensemble des cases qui ont valeur 0.

— Pour chacune de ces cases, on regarde si elle est voisine d'une case de valeur 1. Deux cases sont voisines si elles sont adjacentes et qu'il n'y a pas de mur qui les sépare.
— si tel est le cas, alors on met la valeur de la case à 1 sinon on la laisse à 0.

— On vérifie si il y a une case sur la rangée objectif du pion qui a valeur 1. Si c'est le cas, alors l'accès est possible, sinon il ne l'est pas.

A Annexe : types, variables constantes et fonctions disponibles

A.1 Types, Variables, Constantes

Types

```
typedef struct point int x,y; POINT;
typedef Uint32 COULEUR;
typedef int BOOL;
```

Variables

La largeur et la hauteur de la fenêtre graphique :

```
int WIDTH;
int HEIGHT;
```

Ces deux variables sont initialisées lors de l'appel à `init_graphics()`.

Constantes

Déplacement minimal lorsque l'on utilise les flèches :

```
#define MINDEP 1
```

Les constantes de couleur :

```
#define noir    0x000000
#define gris    0x777777
#define blanc   0xffffff
#define rouge   0xff0000
#define vert    0x00ff00
#define bleu    0x0000ff
#define jaune   0x00ffff
#define cyan    0xffff00
#define magenta 0xff00ff
```

Les constantes booléennes :

```
#define TRUE 1
#define True 1
#define true 1
#define FALSE 0
#define False 0
#define false 0
```

A.2 Affichage

Initialisation de la fenêtre graphique

```
void init_graphics(int W, int H);
```

Affichage automatique ou manuel

Sur les ordinateurs lent, il vaut mieux ne pas afficher les objets un par un, mais les afficher quand c'est nécessaire. c'est aussi utilme pour avoir un affichage plus fluide quand on fait des animations.

On a donc deux modes d'affichage, automatique ou non. Quand l'affichage automatique est activé, chaque dessin d'objet l'affiche automatiquement. Quand il n'est pas automatique c'est l'appel à la fonction `affiche_all();` qui affiche les objets.

Pour basculer de l'affichage automatique au non automatique, il y a deux fonctions :

```
void affiche_auto_on();
void affiche_auto_off();
```
Quand on est en mode non automatique l'affichage se fait lorsque l'on appelle la fonction :
```
void affiche_all();
```
Par défaut on est en mode automatique.

Création de couleur

`COULEUR couleur_RGB(int r, int g, int b);` prend en argument les 3 composantes rouge (`r`), vert (`g`) et bleue (`b`) qui doivent être comprises dans l'intervalle : [0..255].

A.3 Gestion d'événements clavier ou souris

Gestion du clavier

```
POINT get_arrow();
```
Si depuis le dernier appel à `get_arrow()`, il y a eu G appuis sur la flèche gauche, D appuis sur la flèche droite H appuis sur la flèche haut et B appuis sur la flèche bas.

Le point renvoyé vaudra en `x` $D - B$ et en `y` $H - B$.

Cette instruction est non bloquante, c'est à dire que si aucune flèche n'a été appuyée les champs x et y vaudront 0.

Gestion des déplacements la souris

`POINT get_mouse();` renvoie le déplacement de souris avec la même sémantique que `get_arrow()`. Cette instruction est non bloquante : si la souris n'a pas bougé les champs x et y vaudront 0.

Gestion des clics de la souris

`POINT wait_clic();` attend que l'utilisateur clique avec le bouton gauche de la souris et renvoie les coordonnées du point cliqué. Cette instruction est bloquante.

`POINT wait_clic_GMD(char *button);` attend que l'utilisateur clique et renvoie dans `button` le bouton cliqué :

- `*button` vaut 'G' (pour Gauche) après un clic sur le boutton gauche,
- `*button` vaut 'M' (pour milieu) après un clic sur le boutton du milieu,
- `*button` vaut 'D' (pour Droit) après un clic sur le boutton droit.

Cette instruction est bloquante.

Gestion mixte clavier/souris

`POINT wait_key_or_clic(char* isKey);` Attend que l'utilisateur appuie soit sur une touche, soit un clic :

- Retourne un point si une fleche est appuyee : $(-1, 0)$ pour Gauche, $(0, 1)$ pour Haut, $(1, 0)$ pour Droite, $(0, -1)$ pour Bas,
- Retourne un point comme `wait_clic()` si clic.

Le caractère `isKey` vaut

- 1 si il s'agit d'une fleche,
- le caractère alphanumérique clické le cas échéant,
- 0 sinon.

Cette instruction est bloquante.

Fin de programme

`POINT wait_escape();` attend que l'on tape Echap et termine le programme.

A.4 Dessin d'objets

`void fill_screen(COULEUR color);` remplit tout l'écran.
`void pixel(POINT p, COULEUR color);` dessine un pixel.
`void draw_line(POINT p1, POINT p2, COULEUR color);` dessine un segment.
`void draw_rectangle(POINT p1, POINT p2, COULEUR color);` dessine un rectangle non rempli. Les deux points sont deux sommets quelconques non adjacents du rectangle
`void draw_fill_rectangle(POINT p1, POINT p2, COULEUR color);` dessine un rectangle rempli Les deux points sont deux sommets quelconques non adjacents du rectangle
`void draw_circle(POINT centre, int rayon, COULEUR color);` dessine un cercle non rempli.
`void draw_fill_circle(POINT centre, int rayon, COULEUR color);` dessine un cercle rempli.

```
void draw_circle_HD(POINT centre, int rayon, COULEUR color);
void draw_circle_BD(POINT centre, int rayon, COULEUR color);
void draw_circle_HG(POINT centre, int rayon, COULEUR color);
void draw_circle_BG(POINT centre, int rayon, COULEUR color);
```

dessinent des quarts de cercle.

`void draw_fill_ellipse(POINT F1, POINT F2, int r, COULEUR color);` dessine une ellipse remplie.
`void draw_triangle(POINT p1, POINT p2, POINT p3, COULEUR color);` dessine un triangle non rempli.
`void draw_fill_triangle(POINT p1, POINT p2, POINT p3, COULEUR color);` dessine un triangle rempli.

A.5 Écriture de texte

L'affichage de texte n'est pas en standard dans SDL. Il faut donc que la librairie `SDL_ttf` soit installée.

La configuration fournie teste (grâce au `Makefile`) si `SDL_ttf` est installée ou pas. Si elle est installée, l'affichage se fait dans la fenêtre graphique sinon il se fait dans la fenêtre shell.

`void aff_pol(char *a_ecrire, int taille, POINT p, COULEUR C);` affiche du texte avec

- le texte est passé dans l'argument `a_ecrire`
- la police est celle définie par la constante `POLICE_NAME` dans `graphics.c`
- la taille est passée en argument
- l'argument p de type POINT est le point en haut à gauche à partir duquel le texte s'affiche
- la COULEUR C passée en argument est la couleur d'affichage

`void aff_int(int n, int taille, POINT p, COULEUR C);` affiche un entier. Même sémantique que `aff_pol()`.

Les fonctions suivantes affichent dans la fenêtre graphique comme dans une fenêtre shell. Commence en haut et se termine en bas :
`void write_text(char *a_ecrire);` écrit la chaîne de caractère passée en argument.
`void write_int(int n);` écrit l'entier passé en argument.
`void write_bool(BOOL b);` écrit le booléen passé en argument.
`void writeln();` renvoie à la ligne.

A.6 Lecture d'entier

`int lire_entier_clavier();` renvoie l'entier tapé au clavier. Cette fonction est bloquante

A.7 Gestion du temps

Chronomètre élémentaire

Ce chronomètre est précis à la microseconde.
`void chrono_start();` déclenche le chrono. Le remet à zéro s'il était déjà lancé.
`float chrono_val();` renvoie la valeur du chrono et ne l'arrête pas.

Attendre

`void attendre(int millisecondes);` attend le nombre de millisecondes passé en argument.

L'heure

`int heure();` envoie l'heure de l'heure courante.
`int minute();` renvoie le nombre de minutes de l'heure courante
`int seconde();` renvoie le nombre de secondes de l'heure courante.

A.8 Valeur aléatoires

`float alea_float();` renvoie un `float` dans l'intervalle $[0; 1[$.
`int alea_int(int N);` renvoie un `int` dans l'intervalle $[0..N[$ soit N valeurs différentes de 0 à $N-1$.

A.9 Divers

`int distance(POINT P1, POINT P2);` renvoie la distance entre deux points.