Jouer à tirs croisés [je13] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 22 mai 2018

Table des matières

C++ - Jouer à tirs croisés (TP)

Mots-Clés Jeu avec stratégie ■
Requis Axiomatique impérative (sauf Fichiers) ■
Difficulté ●●○ (6 h) ■

Objectif
Cet exercice réalise le jeu dénommé *Tirs Croisés*, paru dans [Jeux et Stratégies :n7].

1 Présentation du jeu

Le jeu « Tirs Croisés »

C'est un jeu tactique qui se joue sur un échiquier comprenant $n \times n$ cases (par exemple 5×5). On prépare le jeu en disposant des entiers pseudo-aléatoires compris entre 1 et 9 sur la grille. Pratiquement, on peut matérialiser les entiers par des cartes à jouer de valeurs correspondantes.

A tour de rôle, chaque joueur prend une carte. Le premier joueur choisit celle qu'il désire n'importe où, mais le second doit nécessairement prélever une carte dans la ligne ou dans la colonne où se trouvait la carte prise par son adversaire. Dans la version programmée, c'est la machine qui tire au hasard la position de départ et le joueur qui commence.

Le jeu se poursuit en respectant cette règle jusqu'à épuisement des cartes, ou jusqu'à ce que l'on ne puisse plus prendre (plus de carte ni dans la ligne, ni dans la colonne). Le vainqueur est alors celui dont le total des points prélevés est maximum.

Exemple

Voici le début d'une partie de Tirs-Croisés. Une position `(x,y)` désigne la case (colonne `x`, ligne `y`). Le curseur est identifié par un `#`.

- La machine propose le tableau suivant. Le curseur est en (2,1) et c'est à l'humain de commencer.

y↓ x→	1	2	3	4	5
1	7	#	4	5	6
2	1	5	8	5	7
3	4	5	6	9	1
4	4	3	2	6	2
5	9	9	8	7	3

- L'humain doit jouer dans la colonne 2 ou ligne 1. Il prend le 9 en (2,5). Le score actuel de l'humain : 9.

y↓ x→	1	2	3	4	5
1	7	.	4	5	6
2	1	5	8	5	7
3	4	5	6	9	1
4	4	3	2	6	2
5	9	#	8	7	3

- La machine doit jouer dans la colonne 2 ou ligne 5. Elle prend le 9 en (1,5). Le score actuel de la machine : 9. Le tableau est alors :

y↓ x→	1	2	3	4	5
1	7	.	4	5	6
2	1	5	8	5	7
3	4	5	6	9	1
4	4	3	2	6	2
5	#	.	8	7	3

- L'humain prend le 7 en (1,1). Le score actualisé est : 9+7=16.

y↓ x→	1	2	3	4	5
1	#	.	4	5	6
2	1	5	8	5	7
3	4	5	6	9	1
4	4	3	2	6	2
5	.	.	8	7	3

- La machine prend le 6 en (5,1). Son score actualisé est : 9+6=15. Le tableau actualisé est :

y↓ x→	1	2	3	4	5
1	.	.	4	5	#
2	1	5	8	5	7
3	4	5	6	9	1
4	4	3	2	6	2
5	.	.	8	7	3

...(suite page suivante)...

2 Modélisation et Découpage

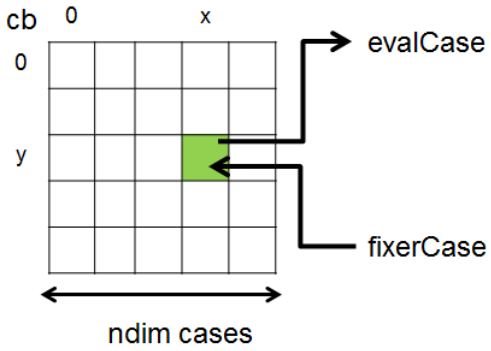
2.1 Modélisation de l'échiquier

Définissez les valeurs constantes de la case `VIDE=0`, la case `CURSEUR=-1` et la case du `BORD=-2`. Rappel : les entiers 1, 2, etc. identifient la valeur des objets.

Type Echiquier
Le plateau (tableau bidimensionnel) sera représenté par un type.

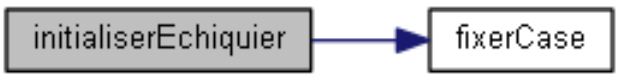
Définissez les constantes `TMIN=4` (taille minimale) et `TMAX=8` (taille maximale) du plateau d'un échiquier puis le type `Echiquier` modélisant le plateau dont la dimension effective sera un entier compris entre `TMIN` et `TMAX` (inclus).

L'accès à une case en `(x,y)` (colonne,ligne) d'un `Echiquier` se fera via une opération `evalCase(...)` et la modification par une opération `fixerCase(...)`. Écrivez ces deux opérations puis justifiez leur intérêt.



Définissez les constantes de la case `VIDE=???` (<- à vous de voir), la case `DETRUITE=???` et la case du `BORD=???`. Rappel : les entiers 1, 2, etc. identifient les (pions des) joueurs.

Écrivez une opération `initialiserEchiquier(...)` qui initialise un `Echiquier`. Le tableau ne doit contenir que des cases `VIDE`s et des cases de `BORD`.



Téléchargez le fichier suivant et mettez-le dans votre dossier.
C++ @[UtilsConsole.cpp]

Copiez/collez ensuite les lignes suivantes :

C++ **Au début** de votre programme :

```
#include "UtilsConsole.cpp"
using namespace UtilsConsole;
```

Soient les procédures d'entrée/sortie en mode texte de l'écran :

- Procédure `clrscr()` : Efface l'écran
- Procédure `gotoxy(x,y)` : Positionne le curseur en (colonne `x`, ligne `y`)

C++ @[UtilsConsole.cpp]

Définissez les constantes `OFFSET_X=2` et `OFFSET_Y=2`, offsets X et Y d'affichage d'une case de l'échiquier, de sorte qu'une case sera toujours affiché au même endroit.

Écrivez une opération `afficherCase(...)` qui affiche la case en `(x,y)` d'un `Echiquier`.

Enfin écrivez une opération `afficherEchiquier(...)` qui affiche un `Echiquier` sous la forme suivante (cas d'un échiquier de dimension 5 avec 3 joueurs après initialisation) :

```
x012345
y******
1*..2..
2*.3...
3*.....
4*....1
5*.....
```

Type Position

Une case est caractérisée par ses coordonnées : indice de ligne et indice de colonne.

Définissez le type `Position` modélisant la position d'une case.

Écrivez une opération `caseDansEchiquier(...)` qui teste et renvoie `Vrai` si une case en une `Position` donnée est dans le plateau d'un `Echiquier`, `Faux` sinon.

De même, écrivez une opération `caseVide(...)` qui teste et renvoie `Vrai` si une case en une `Position` donnée (de coordonnées valides) d'un `Echiquier` est `VIDE`, `Faux` sinon.

2.2 Modélisation des joueurs

Définissez la constante `NJOUEURS=10` (nombre maximal de joueurs), puis le type `Joueurs` comme étant un tableau d'entiers mémorisant les scores.

Écrivez une procédure `initialiserJoueurs(...)` qui initialise le tableau des points à zéro. Pour des raisons de cohérence, on pourra considérer que les joueurs sont numérotés à partir de 1.

Écrivez une procédure `actualiserPoints(...)` qui incrémente les points du joueur `k` de `npoints` dans un `Joueurs`.

2.3 Découpage du problème

Donnez en français, en quoi consiste le **déroulement d'une partie**.

En quoi consiste le **tour d'un joueur** ?

Quelles sont les **conditions d'arrêt** ?

2.4 Modélisation du Jeu

Définissez le type `TirsCroises` modélisant le jeu.

Écrivez une procédure `afficherJeu(...)` qui affiche un `TirsCroises` en affichant son plateau, le numéro du joueur actif, son nombre de points ainsi que la position du curseur.

Écrivez une procédure `donnerMainSuivant(...)` qui donne la main au joueur suivant.

...(suite page suivante)...

3 Version Joueur(s)/Joueur

3.1 Initialisation du jeu

Écrivez une procédure `initialiserJeu(...)` qui initialise un `JeuTirs jeu`.

Écrivez une procédure `test_initialisationJJ` qui déclare un `JeuTirs`, l'initialise puis l'affiche.

Outil C++
L'initialisation du générateur de nombres pseudo-aléatoires (ici avec l'horloge système) s'effectuera dans le **programme principal** avec l'instruction :

```
srand(time(NULL));
```

La procédure `srand` est définie dans la bibliothèque `<random>` et la fonction `time` dans la bibliothèque `<ctime>`.

Testez.

3.2 Destruction d'une case

Écrivez une fonction `caseValide(...)` qui teste et renvoie `Vrai` si une case en une `Position` donnée d'un `JeuTirs` est valide, `Faux` sinon.

Déduisez une fonction `demanderCaseADetruire(...)` qui demande au joueur actif la `Position` de la case à prendre **jusqu'à** ce qu'elle soit valide puis renvoie cette position.

Écrivez une procédure `detruireCase(...)` qui détruit la case en une `Position` donnée (coordonnées valides) d'un `JeuTirs`.

Aide simple
La case du curseur devient une case vide et la case à détruire devient la nouvelle position du curseur.

Enfin écrivez une procédure `detruireCaseSurJeuJJ(...)` qui demande au joueur actif la case à détruire, actualise son nombre de points, puis détruit la case sur un `JeuTirs`.

3.3 Test de partie terminée

Écrivez une fonction `partieTerminee(...)` qui teste et renvoie `Vrai` si la partie est terminée, `Faux` sinon.

Aide simple
Traitez la négative : il suffit d'une case ayant une valeur positive pour que la partie ne soit pas terminée.

3.4 Affichage des gagnants

Écrivez une procédure `afficherGagnant(...)` qui affiche les scores des `Joueurs` ainsi que le(s) joueur(s) gagnant(s).

3.5 Mise en place du tout

Écrivez une procédure `jeuTirsCroisesJJ` qui effectue une partie Joueur(s) contre Joueur. La procédure devra :

- Déclarer un `JeuTirs` et l'initialiser.
- Itérez `TantQue` la partie n'est pas terminée :
  - Afficher le jeu.
  - Afficher le joueur actif et son nombre de points.
  - Afficher la position du curseur (pour mémo).
  - Demander la case à détruire au joueur actif.
  - Donner la main au joueur suivant.
- Afficher les scores et le(s) gagnant(s).

...(suite page suivante)...

4 Version Joueur(s)/Machine

4.1 Stratégies de la machine

Dans la version solitaire, la machine joue l'adversaire.
Vous devrez définir les deux stratégies suivantes :

Écrivez une fonction `rechCaseADetruireJMnaif(...)` qui recherche et renvoie la `Position| de la case à détruire dans un \lstinline`JeuTirs@. Dans cette version, la machine recherche une case de valeur maximale dans la colonne ou ligne du curseur : stratégie très naïve.

Écrivez une fonction `rechCaseADetruireJMniv1(...)` qui recherche et renvoie la `Position` de la case à détruire dans un `JeuTirs`. Dans cette version, la machine se positionne sur chacune des positions jouables et recherche la valeur maximale que peut jouer l'adversaire en supposant qu'il joue naïvement la plus grande des valeurs issue de cette position. Ensuite, il calcule la (une) position qui offre le plus grand score, c.-à-d. une position où la différence entre la valeur jouable et celle de l'adversaire est la plus grande : stratégie de profondeur 1.

Aide simple
Il est utile de disposer d'une structure de données `ListePos` qui mémorise la liste des positions jouables issues d'une position (colonne, ligne).

4.2 Destruction d'une case

Faites un copier/coller de la procédure `detruireCaseSurJeuJJ` en la procédure `detruireCaseSurJeuJM(...)` qui demande au joueur actif la `Position` de la case à détruire puis la détruit sur un `JeuTirs` comprenant des joueurs machine.

Aide simple
Modifiez la procédure de sorte que :

- Si le joueur actif est un humain : faites-le jouer comme dans la version joueur(s)/joueur.
- Sinon dans le cas de la machine : déterminez la case à détruire en utilisant une des stratégies de la machine.

4.3 Procédure jeuTirsCroisesJM

Écrivez une procédure `jeuTirsCroisesJM` qui effectue une partie Joueur(s) contre Machine.

4.4 Stratégie de profondeur 2

On souhaite réaliser une stratégie de profondeur 2 pour la machine.

Écrivez une fonction `rechCaseADetruireJM(...)` recherche et renvoie la `Position| de la case à détruire dans un \lstinline`JeuTirs@. Dans cette version, la machine se positionne sur chacune des positions jouables et recherche la valeur maximale que peut jouer l'adversaire en supposant qu'il joue naïvement la plus grande des valeurs issue de cette position. Puis pour ces positions, il itère l'opération de recherche de la position jouable pour la machine de plus grand score. Enfin il effectue le cumul des différences et restitue la (une) position qui offre le plus grand score pour la machine.

Que pensez-vous de cette stratégie ?
Comment pouvez-vous l'améliorer ?

Aide simple

En jouant un peu, on constate rapidement que la tactique de l'humain n'est pas de jouer naïvement la case qui offre le plus grand gain.