INF4230 – Intelligence artificielle
Hiver 2016 – Examen de mi-session

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Mardi 23 février 2016 – 9h30 à 12h30 (3 heures) – Local SB-M230

Instructions

— Aucune documentation permise.
— Les appareils électroniques, incluant les calculatrices, sont strictement interdits.
— Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
— Aucune question ne sera répondue durant l'examen. Si vous croyez qu'une erreur ou qu'une ambiguïté s'est glissée dans le questionnaire, indiquez clairement la supposition que vous avez retenue pour répondre à la question.
— Une ébauche de la question 2 a été fournie le 18 février. Celle-ci a été modifiée.
— Ne détachez pas les feuilles du questionnaire, à l'exceptions des annexes.

Identification

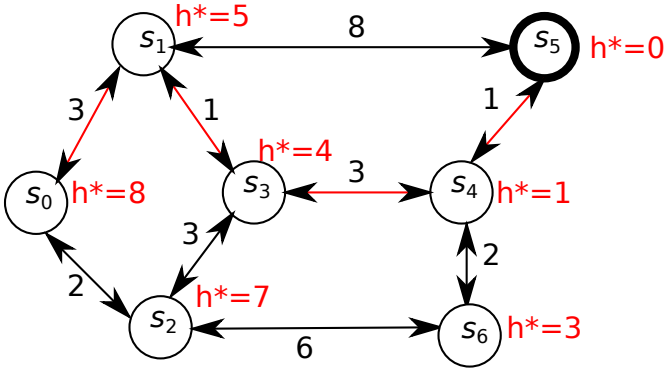
Nom : Solutionnaire

Résultat

Q1 (/5)	Q2 (/4)	Q3 (/5)	Q4 (/4)	Q5 (/4)	TOTAL (/22)

1 Algorithme A* [5 points]

Soit l'espace d'états, les transitions, l'heuristique h, et le but G suivants.



État	$h()$	$h*()$
s_0	7	8
s_1	5	5
s_2	8	**7**
s_3	3	4
s_4	1	1
s_5	0	0
s_6	2	3

$G = \{s_5\}$

(a) Faites la trace de l'algorithme A* en utilisant l'état initial s_0, l'heuristique h et le but $G = \{s_5\}$. [3 points]

Étape	État choisi	Liste *open* $(s, f, g, parent)$ triée par f	Liste *closed* $(s, g, parent)$
#0	–	$(s_0, 7, 0, -)$	
#1	s_0	$(s_1, 8, 3, s_0), (s_2, 10, 8, s_0)$	$(s_0, 0, -)$
#2	s_1	$(s_3, 7, 4, s_1), (s_2, 10, 2, s_0), (s_5, 11, 11, s_1)$	$(s_0, 0, -), (s_1, 3, s_0)$
#3	s_3	$(s_4, 8, 7, s_3), (s_2, 10, 2, s_0), (s_5, 11, 11, s_1)$	$(s_0, 0, -), (s_1, 3, s_0), (s_3, 4, s_1)$
#4	s_4	$(s_5, 8, 8, s_4), (s_2, 10, 2, s_0), (s_6, 11, 9, s_0)$	$(s_0, 0, -), (s_1, 3, s_0), (s_3, 4, s_1), (s_4, 7, s_3)$
#5	s_5	But trouvé !	
#6			
#7			

Solution (chemin) calculée : $\langle s_0, s_1, s_3, s_4, s_5 \rangle$ Coût : 8

(b) L'heuristique h est-elle **admissible** ? Justifiez. [1 point]

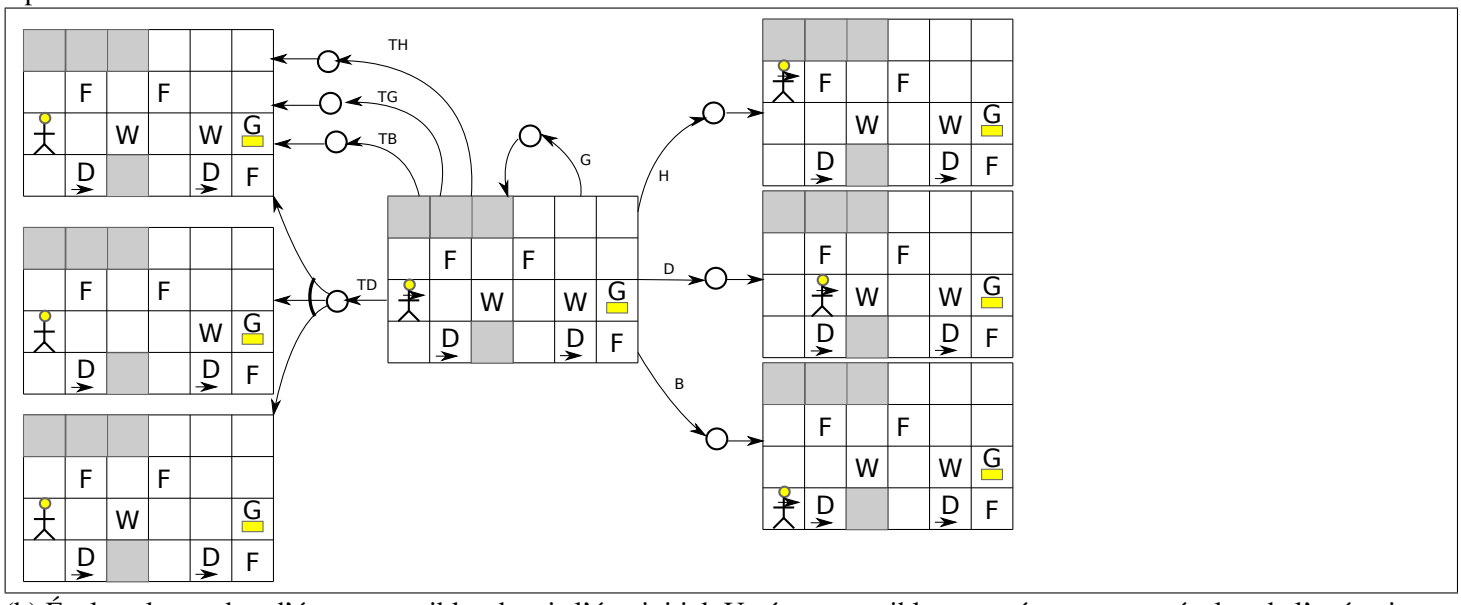
Non. Il y a surestimation à l'état s_2 : $h(s_2) > h*(s_2)$

(c) Proposer des valeurs d'heuristique qui feraient en sorte que l'algorithme A* retourne le plan $\langle s_0, s_2, s_6, s_4, s_5 \rangle$. Si c'est impossible, écrivez simplement «impossible» à droite du tableau ci-bas. [1 point]

$h(s_0)$	$h(s_1)$	$h(s_2)$	$h(s_3)$	$h(s_4)$	$h(s_5)$	$h(s_6)$
0	1000	0	1000	0	0	0

2 Recherche avec actions non déterministes [4 points]

Consultez le monde du Wumpus modifié et décrit à l'Annexe A (page 8).
(a) Dessinez l'arbre ou le graphe ET/OU contenant l'état initial et les états résultant de l'exécution des actions applicables à partir de l'état initial.



(b) Évaluez le nombre d'états accessibles depuis l'état initial. Un état accessible est un état pouvant résulter de l'exécution de zéro, une ou plusieurs actions. Justifiez votre réponse. Vous pouvez exprimer votre réponse à l'aide d'une formule (ex. : $4^2 \times (4+1)$). Si pouvez aussi donner une approximation s'il est difficile d'évaluer le nombre exact.

Note : quand on tombe dans une fosse, on termine dans cette case. Idem pour les monstres.
Estimation (approximation). Nombre cases visitables (non grises) : 20. État de la position du joueur : 20. État des wumpus : 2^2. État de la flèche : 2. Estimation : $20 \times 2^2 \times 2 = 160$.
Calcul exact. En tuant aucun wumpus, les 20 cases sont atteignables. Donc, on peut avoir ou ne pas avoir de flèche dans chaque case, sauf les 2 dépôts de flèches. Donc, 20 positions avec une flèche + 18 positions sans flèche $= 38$. Les 2 wumpus sont tuables. On peut tuer le 2e sans tuer le premier. Calcul : $38 \times 2^2 = 152$.

(c) Dans le contexte d'une recherche dans un arbre/graphe ET/OU, qu'est-ce qu'un **plan contingent** ?

Un plan contingent est une plan qui dit quelle action exécuter pour chaque état.

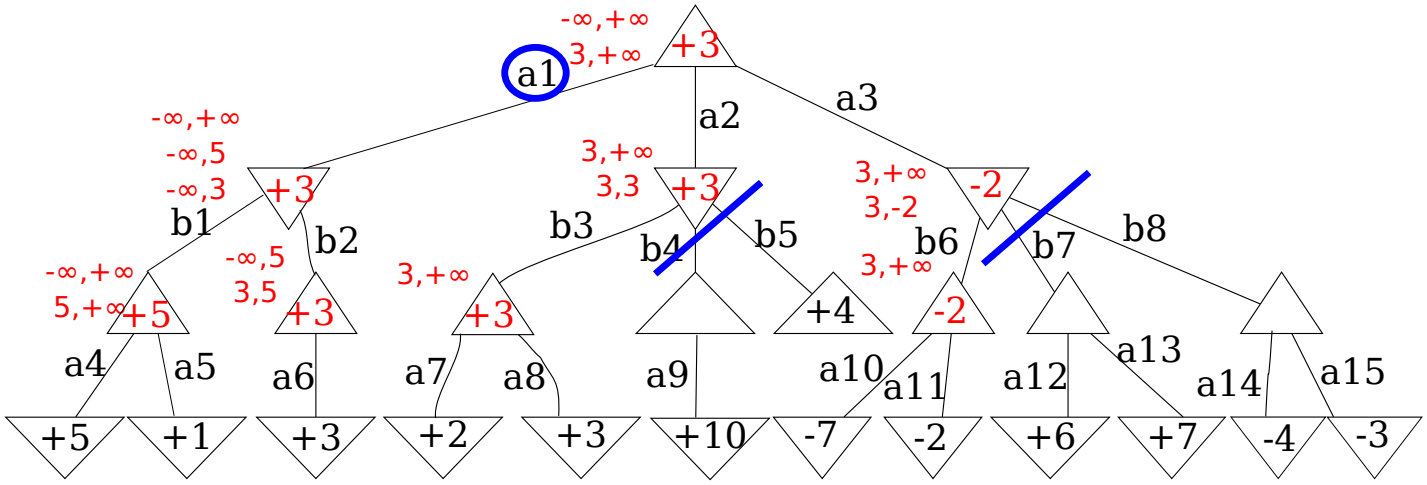
(d) Selon vous, une solution (plan contingent) à un problème du Wumpus modifié peut-elle contenir une action qui essaie de sauter par dessus un puits (fosse) ? Ne considérez pas uniquement le plateau et l'état initial fourni en annexe. **Pourquoi** ?

Non. Dès qu'on exécute une action de sauter par dessus une fausse, il est possible de mourir. Dès que le joueur est mort, il n'y a aucun moyen de satisfaire le but (atteindre l'or). Un plan contingent doit toujours pouvoir mener à état satisfaisant le but. Si c'est impossible, alors il n'y a pas de solution.

3 Algorithme minimax avec élagage alpha-beta pour jeux à somme nulle [5 points]

(a) Donnez la trace de l'algorithme minimax avec élagage alpha-beta sur l'arbre de jeu ci-dessous. [2 points]

1. Indiquez dans le triangle de chaque nœud visité sa valeur minimax calculée.
2. Indiquez la valeur des bornes calculées α et β à gauche de chaque nœud (sauf les feuilles).
3. Rayez au moyen d'une barre les arcs que l'algorithme a élagués.
4. Encerclez l'action retournée par l'algorithme (la décision).



(b) L'algorithme minimax peut-il être généralisé aux jeux contenant du hasard ? Si oui, expliquez comment. Sinon, expliquez pourquoi cela n'est pas possible. [1 point]

Oui. Il suffit de considérer des noeuds «chance» dans l'arbre de recherche. Cela se traduit par une fonction qui évalue la valeur minimax espérée d'un noeud chance. Cette fonction énumère tous les résultats possibles et calcul la moyenne pondérée (par les probabilités) des valeurs minimax des noeuds enfants.

(c) L'algorithme minimax suppose que l'adversaire joue de façon rationnelle. Qu'arrive-t-il si l'adversaire ne joue pas de façon rationnelle ? [1 point]

Quand on suppose que l'adversaire joue de façon rationnelle, on considère le meilleur coup qu'il peut faire (donc le pire pour notre joueur). Si l'adversaire ne joue pas de façon rationnelle, il peut donc se priver de son meilleur coup. Comme on a prévu le pire (pour nous), notre gain sera soit égal ou mieux. Bref, ça ne peut pas être pire.

(d) Expliquez brièvement comment peut-on définir une **fonction d'évaluation** (à ne pas confondre avec fonction d'utilité) pour le jeu des échecs. [1 point]

On peut accorder un score à chaque type de pièce. Exemple, une reine vaut 10 points, une tour 6 points, et un pion 1 seul point. Ensuite, il suffit de compter le nombre de pièces par type et de multiplier ces nombres par la pondération. On obtient ainsi un somme pondérée. La valeur évaluée d'une configuration est égale à la valeur pondérée de notre joueur moins celle de l'adversaire.

4 Problèmes à satisfaction de contraintes (CSP) [4 points]

Lors de votre party de fin de session, on vous mandate d'être le D.J. Votre mission est de sélectionner les cinq pièces musicales à jouer lors de la soirée. Votre répertoire musical est composé d'un ensemble de dix pièces musicales $\{M_1, \cdots, M_{10}\}$. De ce nombre, six sont en anglais $(M_1, \cdots, M_6)$ et quatre en français $(M_7, \cdots, M_{10})$. Les pièces sont classées en styles musicaux : rock=$\{M_1, M_2, M_7\}$, jazz=$\{M_3, M_8\}$, techno=$\{M_4, M_5, M_9\}$ et alternatif=$\{M_6, M_{10}\}$. Le comité organisateur vous impose certaines contraintes que vous devez respecter :

1. vous ne pouvez pas jouer deux pièces consécutives dans la même langue ;
2. vous ne pouvez pas jouer deux pièces consécutives du même style de musique ;
3. vous devez faire jouer au moins une pièce de chaque style ;
4. vous devez placer une demande spéciale du président de votre association qui veut la pièce M_{10}.
5. vous devez terminer la soirée avec une pièce de jazz.

(a) Quelles seraient les variables ? Que représentent-elles ? Quels sont les domaines ? [1 point]

On peut avoir une variable par pièce à sélectionner : S_1, S_2, S_3, S_4, S_5.
Le domaine des variable est l'ensemble es pièces : $\{M_1, \cdots, M10\}$

(b) Comment modéliseriez-vous les contraintes ? Vous pouvez au choix : énumérer toutes les contraintes requises ; ou écrire quelques contraintes de chaque type et indiquer le nombre total de contraintes requises. Vous pouvez utiliser des fonctions. Par exemple, pour un CSP avec deux variables x et y, $f(x) \neq f(y)$, $g(x) > f(y)$ sont des contraintes valides, à condition de définir les fonctions f et g. [1 point]

Contraintes unaires :
$S_5 \in \{M_3, M_8\}$ // Règle #5

Contraintes binaires :
$langue(S_i) \neq langue(S_{i+1}) \forall i \in \{1,2,3,4\}$ // Règle #1
$style(S_i) \neq style(S_{i+1}) \forall i \in \{1,2,3,4\}$ // Règle #2
Ce qui donne : $langue(S_1) \neq langue(S_2)$, $langue(S_2) \neq langue(S_3)$, $langue(S_3) \neq langue(S_4)$, $langue(S_4) \neq langue(S_5)$, $style(S_1) \neq style(S_2)$, $style(S_2) \neq style(S_3)$, $style(S_3) \neq style(S_4)$, $style(S_4) \neq style(S_5)$,

Contraintes globales :
$\cup_i style(S_i) = \{rock, jazz, techno, alternatif\}$ // Règle #3
$(S_1 = M_{10} \vee S_2 = M_{10} \vee S_3 = M_{10} \vee S_4 = M_{10} \vee S_5 = M_{10})$ // Règle #4

(c) Comment implémenteriez un algorithme pour résoudre ce CSP ? Vous pouvez soit écrire un pseudo-code, expliquer son fonctionnement général, ou le simuler. [2 points]
Algorithme *Backtracking search* avec *forward checking* :

1. utiliser les contraintes unaires pour restreindre les domaines des variables ;
2. utiliser les contraintes binaires pour le *forward checking* ;
3. lors d'une assignation complète, tester les contraintes n-aires.

Stratégies (heuristiques) pour choisir la prochaine variable et les valeurs à assigner :

H1. MRV (Most Restrictive VALUE) : commencer par la variable la plus contrainte.

H2. Allouer M10 aussitôt qu'elle apparaît dans le domaine d'une variable.

H3. Dans le choix des valeurs, prioriser les styles qui n'ont pas été encore alloués.

Étape	S_1	S_2	S_3	S_4	S_5
Init.	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$
H5 / S_5	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	$\{M_1,\ldots,M_{10}\}$	M_3
Forward-Checking	$\{M_1,M_2,M_4,M_5,M_6\}$	$\{M_7,\ldots,M_{10}\}$	$\{M_1,M_2,M_4,M_5,M_6\}$	$\{M_7,\ldots,M_{10}$	M_3
H2 / S_2	$\{M_1,M_2,M_4,M_5,M_6\}$	M_{10}	$\{M_1,M_2,M_4,M_5,M_6\}$	$\{M_7,\ldots,M_{10}\}$	M_3
Forward-Checking	$\{M_1,M_2,M_4,M_5,M_6\}$	M_{10}	$\{M_1,M_2,M_4,M_5,M_6\}$	$\{M_7,\ldots,M_9\}$	M_3
H3 / S_4	$\{M_1,M_2,M_4,M_5,M_6\}$	M_{10}	$\{M_1,M_2,M_4,M_5,M_6\}$	M_7	M_3
H3 / S_1	M_4	M_{10}	$\{M_1,M_2,M_4,M_5,M_6\}$	M_7	M_3
Forward-Checking	M_4	M_{10}	$\{M_1,M_2,M_5,M_6\}$	M_7	M_3
S_3	M_4	M_{10}	M_1	M_7	M_3
Tester contraintes	M_4	M_{10}	M_1	M_7	M_3
Solution trouvée	M_4	M_{10}	M_1	M_7	M_3

5 Techniques de recherche locale et Algorithmes génétiques [4 points]

(a) À votre avis, serait-il approprié d'utiliser l'algorithme d'escalade (*hill-climbing*) avec reprises aléatoires pour résoudre le problème du TP1 ? Justifiez votre réponse. [1 point]

Non. La technique de l'escalade ne conserve pas le chemin entre l'état initial et le but.

(b) Sur quelle théorie s'appuient les algorithmes génétiques ? [1 point]

Théorie de l'évolution de Darwin.

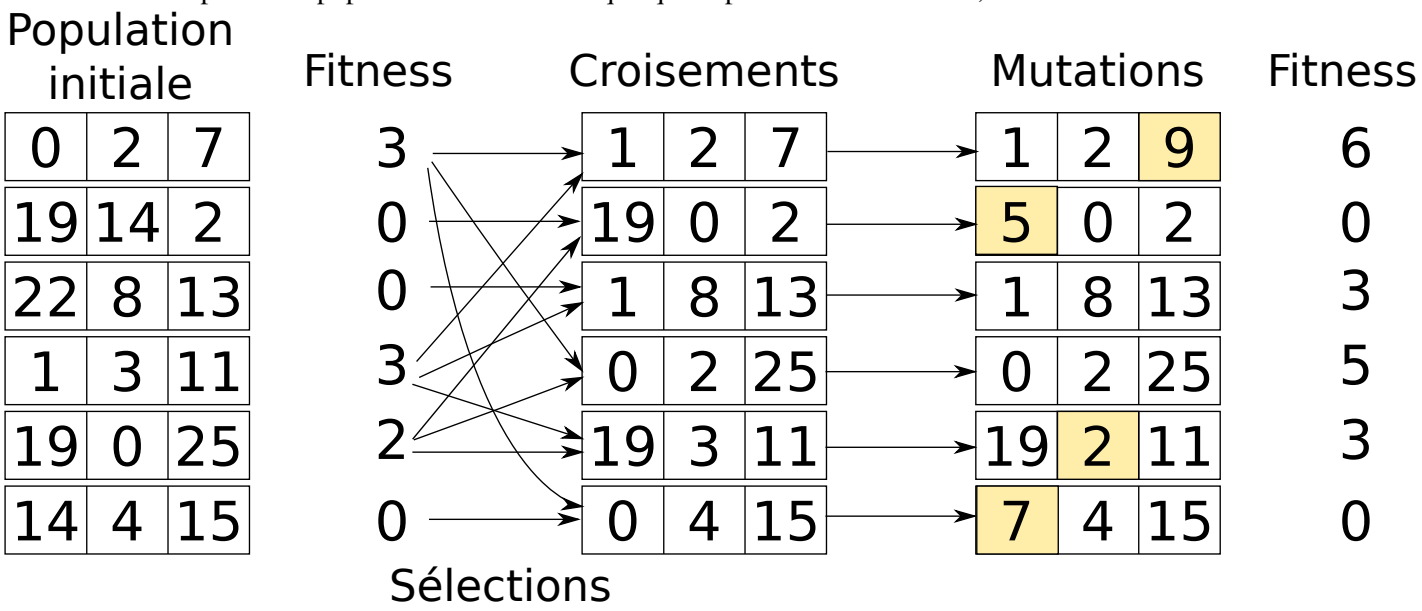
(c) Considérez l'algorithme de cryptographie fourni à l'Annexe B (page 9).
Bien qu'il puisse exister une meilleure technique, on vous demande explicitement d'utiliser un algorithme génétique pour casser (trouver) la clé. Pour y arriver, on vous fournit plusieurs messages chiffrés ainsi qu'un dictionnaire de la langue utilisé dans ces messages. Une clé a été vraisemblablement utilisée lorsque retrouve beaucoup de mots du dictionnaire en décodant des messages chiffrés. [2 points]
Proposez une représentation (chaînes de gènes) :

Vecteur des 3 entiers représentant la clé : $\langle x_1, x_2, x_3 \rangle$ où $0 \leq i \leq 25$.
Exemple : $\langle 2, 3, 17 \rangle$.

Proposez une fonction de *fitness* ;

```
fitness(Messages, Dictionnaire, Clé) :
  f = 0;
  pour tout message m dans Messages:
    m' = dechiffer(m, clé)
    mots = diviser m' en mots (séparer avec espaces, ponctuation, etc.)
    pour chaque mot dans mots :
      si mot est dans Dictionnaire:
        f++
  return f
```

Illustrez un exemple d'une population initiale et de quelques opérations de sélection, de croisement et de mutation.



Annexe A pour la Question 2

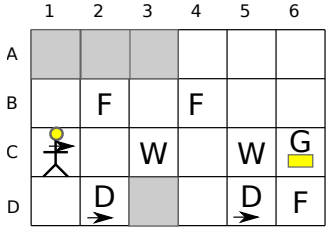
Cette page et les suivantes peuvent être détachées.
Version modifiée du monde des Wumpus.

- Il peut y avoir plusieurs monstres.
- Le joueur peut garder avec lui qu'une seule flèche. Le joueur peut se ravitailler, d'une flèche à la fois, en allant sur un dépôt de flèches (D). Les dépôts ont un nombre illimité de flèches.
- Le monde est totalement observable. Le joueur connaît la position de lui-même, de l'or (G), des dépôts de flèches (D), des puits/fosses (F) et des montres Wumpus (W).
- Les cases grisées sont des obstacles.
- Le monde est non déterministe. Les probabilités ne sont pas connues.
- Le lancement d'une flèche a un résultat incertain. Quand la flèche traverse une case occupée par un Wumpus, ce dernier peut être touché (mourir) ou l'éviter. Si le Wumpus est touché, la flèche arrête. Sinon, la flèche suit son parcours sur les cases suivantes, et ainsi elle peut tuer un autre Wumpus. Une flèche peut tuer au plus un seul Wumpus.
- Lorsque le joueur avance sur un puits/fosse (F), il essaie de sauter par dessus. Il y a 2 résultats possibles : (1) il peut réussir le saut et atterrir sur la case suivante ; (2) il peut échouer le saut, tomber et mourir.
- Il n'y a pas de notion de pointage. On cherche juste à se rendre sur l'or le plus rapidement possible (le moins d'actions possibles).

Le joueur peut exécuter 8 actions :

- 4 actions de déplacement : H, B, G et D pour haut, bas, gauche et droite ;
- 4 actions pour tirer dans les 4 directions : TH, TB, TG et TD.

Il n'y a pas d'action spécifique pour prendre une flèche dans le dépôt de flèches (cela se fait automatiquement) ;
Le jeu se termine dès que le joueur atteint la case contenant l'or, ou quand il meurt (en touchant un Wumpus ou en tombant dans un puits).
Considérez l'état initial suivant (le joueur est à la positon C1 et a une flèche en main).



Annexe B pour la Question 4

Voici un programme C++ de cryptographie simple. La clé est triplet de 3 entiers compris entre 0 et 25 inclusivement. Il y a $26^3 = 17576$ clés possibles.

```
1  #include <iostream>
2  using namespace std;
3
4  void chiffrer(char* chaine, int cle[3]){
5    int k=0;
6    for(int i=0;chaine[i]!=0;i++)
7      if(chaine[i]>='A' && chaine[i]<='Z')
8         chaine[i] = 'A' + (chaine[i]-'A' + cle[k]) % 26;
9      else if(chaine[i]>='a' && chaine[i]<='z')
10        chaine[i] = 'a' + (chaine[i]-'a' + cle[k]) % 26;
11     else
12        k = (k+1) % 3; // changement de mot
13 }
14 void dechiffrer(char* chaine, int cle[3]){
15   int k=0;
16   for(int i=0;chaine[i]!=0;i++)
17     if(chaine[i]>='A' && chaine[i]<='Z')
18        chaine[i] = 'A' + (chaine[i]-'A' - cle[k] + 26) % 26;
19     else if(chaine[i]>='a' && chaine[i]<='z')
20        chaine[i] = 'a' + (chaine[i]-'a' - cle[k] + 26) % 26;
21     else
22        k = (k+1) % 3; // changement de mot
23 }
24
25 // Exemple d'utilisation des fonctions
26 int main(){
27   char test[] = "Bienvenue dans le cours d'intelligence artificielle INF4230!";
28   int cle[] = {1, 2, 25}; // cle de 3 entiers entre 0 et 25 inclusivement
29   chiffrer(test, cle);
30   cout << test << endl;
31   //Cjfowfovf fcpu kd dpvst f'hmsdkkhfdmbd bsujgjdjfmmf KPH4230!
32   dechiffrer(test, cle);
33   cout << test << endl;
34   //Bienvenue dans le cours d'intelligence artificielle INF4230!
35 }
```