Projet d'Algorithmique 2
`Bomberman Survivor`

François Goasdoué
ENSSAT – Université de Rennes 1

Le jeu `Bomberman Survivor`

`Bomberman` est un célèbre jeu de labyrinthe où le héro, Bomberman, cherche à explorer un niveau à la recherche de sa sortie cachée. Dans un niveau, certains murs sont cassables à l'aide de bombes, et, une fois cassés, peuvent révéler des bonus ou l'unique sortie du niveau. Les bonus peuvent aider Bomberman à trouver la sortie plus rapidement, soit en augmentant le nombre de bombes utilisables simultanément, soit en augmentant leur portée. Ces bombes servent également à se défendre car Bomberman n'est pas toujours seul dans un niveau. Il peut y avoir des ennemis « Flame » ou « Ghost » qui le tuent s'ils le touchent. Les « Flame » se baladent aléatoirement sur les chemins d'un le niveau, alors que les « Ghost » peuvent en plus traverser les murs cassables. Le jeu prend fin si la sortie est révélée et atteinte, si Bomberman se fait toucher par un ennemi ou par le souffle d'une de ses bombes, ou si Bomberman fait un mouvement invalide. De plus, Bomberman gagne des points à chaque fois que ses bombes détruisent des murs ou des ennemis (attention les bombes détruisent aussi les bonus). Vous pouvez vous rafraîchir la mémoire sur ce jeu ici : `https://fr.wikipedia.org/wiki/Bomberman`.

Vous devez concevoir et programmer l'Intelligence Artificielle de Bomberman, c'est-à-dire sa stratégie d'exploration de labyrinthe pour (espérer) pouvoir terminer un niveau, et si possible maximiser le score.

Un niveau de jeu est codé de la façon suivante en ASCII :

```
***************
*@..=.**..=...*
*.*.*=*.*=*=*.*
*..====&....=.*
*.*.*.*.*.*.*.*
*..E.....==...*
*.*.*.*.*.*.*F*
*.=.......=..=*
*.*=*.*.*=*.*.*
*=.&=......=.%*
*=*=*=*.*.*.*=*
*.=..B....==..*
***************
```

où

- '*' est un mur incassable
- '=' est un mur cassable
- '.' est un chemin
- '@' est Bomberman
- '&' est un ennemi « Flame »
- '%' est un ennemi « ghost »
- 'E' est la sortie (une fois révélée)
- 'B' est un bonus « bombe supplémentaire » (une fois révélé)
- 'F' est un bonus « augmentation de la portée des bombes » (une fois révélé)

À chaque tour de jeu, Bomberman doit prendre la décision de poser une bomber ou d'aller soit au nord, soit à l'est, soit au sud, soit à l'ouest (on ne peut pas aller en diagonale). Pour cela, le contrôleur de jeu vous demande uniquement la prochaine action à effectuer grâce à la fonction `bomberman` que vous devez développer :

```
action bomberman(
 tree map, // 4-ary tree of char modeling the subset of the game map Bomberman sees
 action last_action, // last action made, -1 in the be beginning
 int remaining_bombs, // number of bombs
 int explosion_range // explosion range for the bombs
 ) {...}
```

où l'action demandée est codée par

```
/*
  Enumeration for the different actions that the player may choose
 */
enum actions {BOMBING, NORTH, EAST, SOUTH, WEST};
typedef enum actions action; // define action as a shorthand for enum actions
```

et où l'arbre de ce que voit Bomberman est codé par

```
/*
  Tree of char modeling a subset of the game map
 */
struct node_s {char c; struct node_s * n; struct node_s * e; struct node_s * s; struct node_s * w;};
typedef struct node_s node;
typedef node * tree;
```

En sortie de fonction, vous devez retourner l'action choisie parmi les cinq possibilités : BOMBING, NORTH, EAST, SOUTH, WEST. Attention, vous n'avez pas le droit de traverser un mur (cassable ou incassable) ou une bombe que vous auriez déposée, sinon vous perdez immédiatement ! La différence essentielle avec le projet d'Algorithmique 1 est que vous n'avez q'une vision partielle du niveau de jeu modélisé par l'arbre map (la partie en surbrillance lors de l'affichage de la carte). Bomberman est à la racine l'arbre map et ce qu'il voit dans les quatre directions correspond aux quatre sous-arbres n, e, s et w comme le montre le mode déboguage du jeu :

```
***************
*..@=.**..=...*
*.*.*=*.*=*=*.*
*..====.....=.*
*.*.*.*.*.*.*.*
*..=.....==&..*
*.*.*.*.*.*.*=*
*.=.......=..=*
*.*=*.*.*=*.*.*
*=.&=......=.&*
*=*=*=*.*.*.*=*
*.=..=....==..*
***************
North view tree:
<*,0,0,0,0>
East view tree:
<=,0,0,0,0>
South view tree:
<.,0,<*,0,0,0,0>,<=,0,0,0,0>,<*,0,0,0,0>>
West view tree:
<.,<*,0,0,0,0>,0,<*,0,0,0,0>,<.,<*,0,0,0,0>,0,<.,0,<*,0,0,0,0>,<.,0,<.,<*,0,0,0,0>,<=,0,0,0,0>,
<*,0,0,0,0>,0>,<.,0,<*,0,0,0,0>,<.,0,<.,0,0,0,0>,<.,0,0,0,0>,<*,0,0,0,0>>,<*,0,0,0,0>>,
<*,0,0,0,0>>,<*,0,0,0,0>>,<*,0,0,0,0>>>
```

Votre objectif est de concevoir l'Intelligence Artificielle de Bomberman afin qu'il finisse (ou aille assez loin dans) l'exploration d'un niveau, tout en maximisant son score. Notez toutefois que l'objectif premier est de terminer le niveau (=gagner) ! Enfin et surtout, vous ne devez pas perdre de vue qu'il s'agit d'une évaluation d'Algorithmique, et donc que ça ne sera pas nécessairement la meilleure stratégie de jeu qui aura la meilleure note. Un bon projet sera celui avec une stratégie raisonnablement intelligente et, surtout, développée en utilisant au mieux les outils algorithmiques vus en cours, TD et TP.

Travail à réaliser en monôme

Récupérer les fichiers de travail sur le moodle, en bas de la page du cours d'*Algorithmique 2* dans la section Projet :

1. `bomberman.h` : le fichier d'entête définissant ce qui est nécessaire pour coder votre IA dans `player.c`. CE FICHIER NE DOIT PAS ÊTRE MODIFIÉ. Le fichier *original* sera utilisé pour compiler votre fichier `player.c`.
2. `bomberman.o` : le moteur précompilé du jeu bomberman.
3. `player.c` : le joueur aléatoire que vous devez modifier pour faire votre propre joueur. Pour cela, vous devez modifier la fonction `bomberman`. Celle-ci pourra faire appel à autant de sous-modules que nécessaires, ceux-ci devant être développés UNIQUEMENT dans le *seul et unique fichier* `player.c`. Chaque module développé devra être **impérativement** commenté de la façon suivante : (i) avant chaque module, la stratégie de résolution du problème dévolu au module doit être décrite (càd. l'idée générale de ce que fait le module et comment il le fait), puis, (ii) dans le code du module, la mise en oeuvre de la stratégie de résolution doit être expliquée (càd. les détails du codage de la stratégie utilisée sont expliqués au fur et à mesure du code). ATTENTION ! ! ! Votre code ne doit utiliser AUCUNE variable globale, fichier sur disque ou autre moyen visant à stocker de l'information entre deux appels à la fonction `bomberman` : c'est interdit !
4. `level0.map` : le premier niveau du jeu Bomberman, mais sans ennemi pour commencer à tester votre IA dans un milieu non hostile.
5. `level1.map` : le premier niveau du jeu Bomberman, avec trois ennemis « Flame ».
6. `level2.map` : le premier niveau du jeu Bomberman, avec deux ennemis « Flame » et un « Ghost ».

Les commandes de compilation pour fabriquer le jeu bomberman à partir des fichiers bomberman.h, bomberman.o et player.c sont données sur la page du moodle, en fonction de la version d'Ubuntu utilisée (12.04 à l'ENSSAT ou 18.04/20.04 sur vos machines personnelles).

Commencez par compiler le programme avec le joueur « Random » fourni dans le fichier player.c et lancez `./bomberman` sans niveau de jeu. Ainsi, vous découvrirez les options du programme et pourrez vous familiariser avec : testez les !

Le travail à rendre consiste en un rapport et le fichier `player.c`.

Le rapport décrira (en français) la stratégie générale de votre IA, c'est-à-dire les choix caractéristiques de réaction de Bomberman selon les situations de jeu. Votre rapport donnera aussi pour 3 modules centraux de votre stratégie : sa description en termes de fonctionnalités et entrées/sorties, la justification des choix algorithmiques (fonction/procédure, boucle, récursivité, etc) et son pseudo-code commenté. Il expliquera aussi les limites du programme actuel, et comment celui-ci pourrait encore être étendu.

Le fichier `player.c` devra être parfaitement indenté, commenté, etc. Il devra aussi compiler sans erreur ni warning, et jouer avec une stratégie raisonnable.

Enfin, attention, le code de `player.c` ne doit pas utiliser de code qui ne serait pas de vous (forums, bibliothèques, etc).

Modalités de remise du travail

Votre projet devra être rendu **avant** le dimanche 24 janvier à 23h59 via le moodle, dans le dépôt prévu pour votre groupe sur la page du cours d'*Algorithmique 2*.

Vous remettrez votre rapport au format **PDF** (seul format accepté), ainsi que votre fichier `player.c`. Ces deux fichiers seront déposés sous la forme d'une archive **ZIP** (seul format accepté) : `NomPrenom.zip` où `Nom` et `Prenom` sont vos nom et prénom. Cette archive comprendra UNIQUEMENT un répertoire `NomPrenom` contenant votre rapport et votre fichier `player.c`. N'oubliez pas non plus de mettre vos nom et prénom dans la constante globale `binome` du fichier `player.c` !

Enfin, votre code sera passé au détecteur de plagiat afin de le comparer à ceux de la promotion. Les codes plagieurs **et** plagiés auront 0/20.

Bon travail !