**Université de Sherbrooke**
**Faculté de génie**
**Département de génie électrique et de génie informatique**

# JOUEURS INTELLIGENTS POUR JEUX VIDÉO

# GUIDE DE L'ÉTUDIANT

**Session: S8**
**Module: Intelligence artificielle**
**Unité: APP 1**
**Semaines 2, 3 et 4**

**AUTOMNE 2016**

Auteur: Charles-Antoine Brunet
Version: 194 (2 septembre 2016 à 15:17:49)

Ce document est réalisé avec l'aide de LaTeX et de la classe de document `gegi-app-guide`.

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# 1 ACTIVITÉS PÉDAGOGIQUES ET ÉLÉMENTS DE COMPÉTENCES

## GEI790 - Intelligence artificielle formalisable

### Compétences

1. Concevoir et mettre en oeuvre des techniques de l'intelligence artificielle formalisables appropriées à partir de spécifications descriptives.
2. Mettre en oeuvre un système intelligent basé sur des techniques formalisables.

### Contenu

- Logique propositionnelle et logique du premier ordre
- Systèmes experts
- Méthodes de recherche
- Planification

Description officielle : http://www.usherbrooke.ca/fiches-cours/gei790

# 2 QUALITÉS DE L'INGÉNIEUR

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP.

| Q01 | Q02 | Q03 | Q04 | Q05 | Q06 | Q07 | Q08 | Q09 | Q10 | Q11 | Q12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | ✓ | | ✓ | ✓ | | ✓ | | | | | |

Les qualités de l'ingénieur sont les suivantes. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant :
http://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/.

| Qualité | Titre |
|---|---|
| Q01 | Connaissances en génie |
| Q02 | Analyse de problèmes |
| Q03 | Investigation |
| Q04 | Conception |
| Q05 | Utilisation d'outils d'ingénierie |
| Q06 | Travail individuel et en équipe |
| Q07 | Communication |
| Q08 | Professionnalisme |
| Q09 | Impact du génie sur la société et l'environnement |
| Q10 | Déontologie et équité |
| Q11 | Économie et gestion de projets |
| Q12 | Apprentissage continu |

# 3 ÉNONCÉ DE LA PROBLÉMATIQUE

La compagnie *UdeSoft* fabrique des jeux vidéos de dernière génération. Afin de garder son avantage concurrentiel, *UdeSoft* mène souvent des investigations sur de nouvelles technologies afin de rendre ses jeux plus attrayants, plus réalistes et aussi afin de réduire ses couts. Le département de recherche et développement est donc très actif et bien financé.

Un des problèmes auquel fait face *UdeSoft* est l'intelligence insuffisante des joueurs contrôlés par la console de jeu, les joueurs informatisés (JI), et qui évoluent dans le même environnement que les joueurs humains. Afin de rendre les JI plus intelligents et capables de s'adapter au grand nombre de situations différentes et imprévisibles, il serait avantageux que les JI soient dotés de capacités leur permettant de planifier de manière efficace et autonome leurs actions, sans plans préétablis. Les JI pourraient alors démontrer des comportements délibératifs, logiques et utiles.

Ainsi, *UdeSoft* planifie d'intégrer des algorithmes de planification aux JI dans leurs jeux de dernière génération. En première version, les capacités de planification visent les actions typiques de plusieurs jeux et qui sont donc représentatives : se déplacer, prendre ou laisser tomber un objet, et attaquer un adversaire dans le but d'obtenir un objet. En planifiant dynamiquement, les JI s'adapteraient aux situations inconnues de manière autonome. La planification est préférable à une programmation au préalable de plans préétablis pour chacune des situations spécifiques au jeu, car les plans préétablis requièrent non seulement du temps de programmation, mais en plus ils sont difficilement réutilisables dans d'autres situations ou d'autres jeux. Ce type de programmation est couteux pour *UdeSoft*. De plus, avec de la planification, les JI paraitraient plus intelligents et adaptatifs. Ils rendraient le jeu plus intéressant pour les joueurs humains, ce qui pourrait augmenter les ventes.

Une investigation technologique réalisée par l'ingénieur en chef du groupe de recherche et de développement de *UdeSoft* indique que certaines méthodes issues de l'intelligence artificielle pourraient être intéressantes pour cette problématique de planification qui est une forme de recherche de solution. Les techniques de recherche non informée et de recherche informée avec heuristique ont été identifiées et retenues. Des méthodes de planification ont aussi été retenues, comme la planification par recherche d'espace d'états et les graphes de planification. Un choix de technique devra être fait en accord avec la complexité de la tâche de planification de jeu.

D'autres techniques plus sophistiquées de planification ont été identifiées, semblables à celles utilisées par les humains, comme la planification conditionnelle, la replanification en ligne, la

planification hiérarchique et la planification multiagents. Par manque de temps, elles n'ont pas été investiguées, mais la question de leur pertinence dans un jeu vidéo est encore sans réponse.

L'investigation indique aussi que le langage avec lequel sont implémentés les jeux chez *UdeSoft*, le C++, n'est pas bien adapté pour le raisonnement logique et la représentation des connaissances, avec des faits et des règles, comme il est nécessaire pour cette problématique. Le langage Prolog a été sélectionné pour faire le prototypage rapide et les tests d'algorithmes de planification. Afin de raisonner sur des connaissances l'usage de la logique propositionnelle, de la logique du premier ordre, de mécanismes d'inférences, et de chainage avant et arrière sont très utiles. Le langage Prolog est bien adapté pour l'usage de ces techniques.

*UdeSoft* juge qu'il est maintenant temps de passer à l'étape suivante : trouver quelle méthode de planification est la plus appropriée pour leurs jeux et pour quelles raisons. Le grand patron de la compagnie a des doutes quant à ces techniques d'intelligence artificielle, car il a entendu dire que la planification fait souvent face à une explosion des combinaisons possibles lors de la recherche de solution. Il demande des preuves à l'appui qu'une de ces techniques est vraiment efficace et jusqu'à quel point elle est utilisable. Les algorithmes doivent donc être testés dans des situations réalistes et idéalement dans un jeu simplifié. L'annexe A donne plus de détails sur le jeu simplifié afin de faire des tests : le jeu *BlockBlitz*. L'importance n'est pas la performance du JI dans *BlockBlitz*, mais plutôt la validité des recommandations faites.

C'est à vous qu'est confié ce dossier recherche et développement. Vous devez remettre votre rapport de recommandations à l'ingénieur sénior du groupe qui mène ce dossier. Il veut toutes les données en main afin de convaincre le grand patron qu'il faut y aller de l'avant avec les techniques d'intelligence artificielle dans les jeux vidéos. Selon l'ingénieur sénior, elles représentent la clé du succès de la compagnie et du futur des jeux vidéos.

# 4 CONNAISSANCES NOUVELLES

## Connaissances déclaratives : QUOI

### Mathématiques

- Recherche informée
- Recherche non informée
- Logique propositionnelle
- Logique du premier ordre
- Inférence et unification en logique du premier ordre
- Complexité des algorithmes de recherche

### Sciences de l'ingénierie

- Systèmes à base de connaissances
- Chainage avant et arrière
- Planification par recherche de l'espace d'états
- Graphes de planification
- Planification : ressources et temps, hiérarchique, conditionnelle, replanification en ligne, et multiagents

## Connaissances procédurales : COMMENT

- Utilisation d'un langage de programmation logique (Prolog)
- Établir la procédure de conception et de développement de systèmes logiciels basés sur la logique
- Mise en oeuvre d'un système intelligent basé sur des techniques formalisables

## Connaissances conditionnelles : QUAND

- Identifier lorsque les techniques formalisables d'intelligence artificielle sont appropriées pour un problème spécifique
- Choisir les techniques et algorithmes appropriés d'intelligence artificielle formalisable pour un problème spécifique

# 5 DOCUMENTATION

**Livre de référence [2]**

Les chapitres 1 et 2 sont intéressants pour votre culture personnelle sur l'intelligence artificielle et pour vous donner un peu de perspective sur le sujet. La matière contenue dans les chapitres 1 et 2 n'est pas sujette à évaluation. La lecture de ces deux chapitres est donc uniquement suggérée.

La matière officielle de l'APP et qui est sujette à évaluation est la suivante :

- Chapitre 3
- Chapitre 4 (s'attarder surtout, mais pas uniquement, aux concepts)
- Chapitre 7, pages 234-253.
- Chapitre 8
- Chapitre 9, jusqu'à la section 9.4 inclusivement
- Chapitre 10 au complet (sections 10.3 et 10.4, s'attarder surtout, mais pas uniquement, aux concepts)
- Chapitre 11 (s'attarder surtout, mais pas uniquement, aux concepts)

Les activités de la 1re pratique procédurale et du premier laboratoire se concentrent sur les chapitres 3, 4, 7 et 8 ainsi que sur le langage Prolog. Les activités de la 2e pratique procédurale se concentrent sur les chapitres 9, 10 et 11.

**Lectures facultatives sur la logique**

- Dans [1] : sections 1.1. à 1.3, pages 1-30. Ce livre a été acheté en S2 pour ceux qui sont en génie informatique.

**Tutoriel Prolog**

Plusieurs tutoriels pour le langage Prolog existent sur internet et ils changent continuellement. Faites une recherche et vous en trouverez facilement un qui vous plait.

**Structures de données Prolog**

Le site web de l'unité vous donne de structures de données en Prolog qui sont utiles pour les algorithmes de planification et de recherche. Les structures de données offertes sont la pile, la queue, la queue avec priorité, et le *set*. La queue avec priorité peut vous amener à redéfinir le prédicat de comparaison de 2 objets qui se nomme `precedes` afin qu'il convienne à vos besoins d'ordonnancement.

# 6 LOGICIELS ET MATÉRIEL

SWI-Prolog est utilisé dans le cadre de cette activité pédagogique et de son évaluation (autant le rapport que les examens). Il est installé dans les laboratoires du Département. Il est disponible sur le site web de SWI-Prolog, consultez le site web de l'unité pour plus de détails. Les environnements de développement Prolog ont tous leurs particularités et surtout en ce qui concerne le langage Prolog, bien qu'il soit standardisé. Consultez toujours la documentation de l'environnement que vous utilisez afin de vous assurer de faire les choses correctement.

# 7 SOMMAIRE DES ACTIVITÉS

## Semaine 1

- 1re rencontre de tutorat
- Séminaire : le langage Prolog
- Formation à la pratique procédurale I
- Formation à la pratique au laboratoire

## Semaine 2

- Formation à la pratique procédurale II
- Rencontre de collaboration à la solution de la problématique
- Validation pratique de la solution

## Semaine 3

- 2e rencontre de tutorat
- Évaluation formative
- Remise des livrables en lien avec l'unité
- Consultation
- Évaluation sommative

# 8 PRODUCTIONS À REMETTRE

- Les productions se font en équipe de 2.
- La même note sera attribuée à tous les membres de l'équipe.
- L'identification des membres des équipes doit être faite sur la page web de l'unité avant 16h30, le lendemain de votre 1er tutorat.
- La date limite pour le dépôt électronique des livrables est 09h00 (pas 21h00!), le jour du 2e tutorat. Tout retard entraine une pénalité immédiate de 20% et ensuite de 20% par jour supplémentaire.
- Seules les collaborations intraéquipe sont permises. Cependant, vous devez résoudre la problématique de façon individuelle pour être en mesure de réussir les validations et les examens.
- Les productions soumises à l'évaluation doivent être originales pour chaque équipe, sinon l'évaluation sera pénalisée en cas de non-respect de cette consigne.
- Suivez les directives données plus bas pour la remises de vos productions. **Aucune remise, complète ou partielle, ne sera acceptée ou traitée par courriel ; elle sera ignorée.**

## Schéma de concept

Individuellement, faites un schéma de concepts qui illustre le rapport entre les méthodes de recherche (informée et non informée), la logique du premier ordre, l'inférence en logique du premier ordre, la planification et le langage Prolog.

Pour valider votre schéma de concepts, apportez-le lors des rencontres de formation à la pratique procédurale et de formation à la pratique en laboratoire. Les schémas sont à faire lors de l'étude personnelle et ils sont remis à la fin de la 2e rencontre de tutorat.

## Rapport d'unité

La partie principale du rapport ne devrait pas dépasser 15 pages. La partie principale exclut la page couverture et les autres pages préliminaires, comme la table des matières. La partie principale du rapport devrait contenir au minimum les items suivants :

- Une introduction qui décrit le problème dans vos propres mots et qui présente le contenu de votre rapport.
- La recommandation d'une technique de planification avec sa justification incluant la recommandation d'une technique de recherche sous-jacente avec sa justification. Donnez

aussi votre appréciation sur la technique de planification et sur la technique de recherche que vous avez retenues et jusqu'à quel point elles sont vraiment efficaces et à quelles conditions elles sont utilisables dans le jeu proposé. Il est très pertinent d'avoir une discussion la longueur et la qualité des plans générés, la taille maximale ou raisonnable de l'espace de jeu, et autres considérations semblables. Vous pouvez aussi discuter sur les temps d'exécution et l'espace mémoire requis.

- Les résultats de tests avec explications qui appuient votre recommandation.
- La représentation PDDL des actions possibles dans le jeu.
- Une description du vocabulaire et une explication des connaissances représentant les parties principales de votre solution (le JI). Cette partie doit être exprimée en logique du premier ordre et non pas en Prolog.
- Une description des parties principales de votre solution (le JI) en Prolog. Discutez aussi du lien entre ces parties en Prolog et les descriptions en logique du premier ordre du point précédent.
- Une recommandation de la pertinence d'utiliser dans le jeu des algorithmes de planification évolués, comme la planification conditionnelle, la replanification en ligne et la planification multiagents.
- Une conclusion sur ce qui a été fait et observé, et jusqu'à quel point la technique de planification et la technique de recherche résolvent le problème en question.

## Dépôt électronique

Vous devez faire le dépôt électronique de votre implémentation d'un JI ainsi que de votre rapport d'unité. Votre dépôt est un fichier d'archive (zip) nommé selon les CIP des membres de l'équipe séparés par un tiret. Il y a donc un seul dépôt par équipe. Votre archive doit contenir ce qui suit.

- Un seul fichier Prolog (**un seul !**) contenant le code Prolog de votre JI, voir l'annexe A.4 à ce sujet.
- Une version électronique en format PDF de votre rapport d'unité.
- Un seul sous-répertoire nommé `Tests` qui contient les tests principaux que vous avez faits pour valider votre JI ainsi que leurs résultats.

# 9 ÉVALUATIONS

## 9.1 Sommaire

L'examen sommatif et l'examen final portent sur tous les éléments de compétences de l'unité. L'examen sommatif est un examen écrit sans documentation et l'examen final comporte une partie théorique sur papier et une partie pratique sur ordinateur dans l'environnement SWI-Prolog. Le sommaire de l'évaluation est donné au tableau 9.1.

| | GEI790-1 (390) | GEI790-2 (210) | Total (600) |
|---|---|---|---|
| Rapport d'unité (équipe de 2) | 90 | 90 | 180 |
| Examen sommatif (individuel) | 180 | 30 | 210 |
| Examen final (individuel) | 120 | 90 | 210 |

Tableau 9.1 : Sommaire de l'évaluation de l'unité

## 9.2 Rapport et livrables associés

L'évaluation du rapport porte sur les compétences figurant dans la description des activités pédagogiques. Ces compétences ainsi que la pondération de chacune d'entre elles dans l'évaluation de ce rapport sont indiquées au tableau 9.2. Quant à la qualité de la communication technique, elle n'est pas évaluée de façon sommative, mais si votre rapport est fautif sur le plan de la qualité de la communication et de la présentation, il vous sera retourné et vous devrez le reprendre pour être noté.

| | GEI790-1 (90) | GEI790-2 (90) |
|---|---|---|
| Introduction | 6 | |
| Recommandations et justifications | 24 | |
| Résultats et explications | | 30 |
| Représentation PDDL, description et explication du vocabulaire des parties principales | 18 | |
| Description des parties principales de votre code Prolog et lien avec le point précédent | 18 | |
| Recommandation des techniques évoluées de planification | 18 | |
| Conclusion | 6 | |
| Fonctionnement du prototype | | 30 |
| Implémentation Prolog | | 30 |

Tableau 9.2 : Sommaire de l'évaluation du rapport

# 10 PRATIQUE PROCÉDURALE 1

### But de l'activité

- Comprendre les algorithmes de recherche non informée
- Comprendre les algorithmes de recherche informée
- Comprendre la logique du premier ordre

Afin de bien réussir cette activité, il vous est suggéré de la préparer. Vous pouvez la préparer en lisant la documentation qui est suggérée à la section 5pour ces sujets.

## 10.1 Exercices

### E.1 Recherche non informée

1. Qu'est-ce que la recherche non informée ?
2. Quels problèmes peut-on rencontrer avec ce type de recherche ?
3. Est-ce que ce sont des algorithmes de recherche optimaux et complets ?

### E.2 Recherche informée

1. Qu'est-ce que la recherche informée ?
2. Qu'est-ce que la recherche avare ? Est-elle optimale et complète ? Quels problèmes peut-elle amener ?
3. Qu'est-ce que la recherche A* ? Est-elle optimale et complète ? Quels problèmes peut-elle amener ?

### E.3 Recherche locale

1. Qu'est-ce que les algorithmes de recherche locale ?

### E.4 Manque d'information ?

1. Si les actions sont non déterministes, qu'est-ce qui arrive à la recherche ?
2. Si l'environnement est partiellement observable, qu'est-ce qui arrive à la recherche ?
3. Si l'environnement est inconnu, qu'est-ce qui arrive à la recherche ?

## E.5 Exemples

Trouver le chemin utilisé par les techniques suivantes pour aller d'Arad à Bucarest en utilisant les figures 3.2 (page 68) et 3.22 (page 93) de [2].

1. Recherche profondeur d'abord
2. Recherche avare (*greedy best-first search*)
3. Recherche $A^*$

## E.6 Logique du premier ordre

1. Qu'est-ce que la logique du premier ordre
2. À quoi sert-elle ?
3. Est-ce qu'il y a un moyen rapide de discuter d'ensembles et de listes en logique du premier ordre ?

## E.7 Exemples

Traduisez les énoncés suivant en logique du premier ordre ou en texte.

1. Tous les chats sont des mammifères
2. Tous dans la classe sont futés
3. Spot a une soeur qui est un chat
4. Quelqu'un dans la classe est futé
5. $\exists x \, \forall y \; Aime(x, y)$ Note : c'est $x$ qui aime $y$
6. $\forall y \, \exists x \; Aime(x, y)$ Note : C'est $x$ qui aime $y$
7. Spot a au moins 2 soeurs

# 11 PRATIQUE EN LABORATOIRE

### But de l'activité

Le but de l'activité est de se familiariser avec le langage Prolog.

## 11.1 Exercices

### E.1 Assistant de choix de menu

Réalisez en Prolog un aide aux clients à la sélection de repas dans un restaurant. Le programme doit suggérer aux clients des repas ou des repas légers. Un repas est composé d'un hors-d'oeuvre, d'un plat et d'un dessert. Un plat est une assiette de viande ou de poisson. Un repas est léger si le total des points pour le repas est de moins de 10 points. Les choix au menu aujourd'hui sont les suivants :

| Type | Contenu | Points |
|---|---|---|
| hors d'oeuvre | salade | 1 |
| hors d'oeuvre | pâté | 6 |
| poisson | sole | 2 |
| poisson | thon | 4 |
| viande | porc | 7 |
| viande | boeuf | 3 |
| dessert | glace | 5 |
| dessert | fruit | 1 |

L'interrogation de l'assistant pour un repas ou un repas léger doit se faire avec des questions du type : `repas(H,P,D)` ou encore `repasLeger(H,P,D)` avec `H` un hors-d'oeuvre, `P` un plat et `D` un dessert. Quelles sont les significations et les réponses des questions suivantes ?

```
repas(pate,porc,X). repasLeger(pate,porc,X). repasLeger(X,Y,glace).
```

### E.2 Un sur deux

Sans utiliser les fonctionnalités de la librairie de liste de l'environnement Prolog, réalisez un prédicat qui permet d'imprimer 1 valeur sur 2 d'une liste, donc les éléments 2, 4, 6, etc. Le prédicat est appelé `un_sur_deux` et a un seul paramètre, la liste à imprimer. Le prédicat devrait être capable d'imprimer toutes les listes et il devrait donc toujours être évalué à vrai (**`true`**). Testez au moins avec la liste vide et les listes avec 1, 2, 3, 4, 5 et 10 éléments.

## E.3 Gestion de liste

Sans utiliser les fonctionnalités de la librairie de liste de l'environnement Prolog, réalisez le prédicat `gerer_liste` qui modifie une liste de valeurs à partir d'une liste de commandes. Un exemple d'appel est le suivant :

```
gerer_liste([1,2,2,5,3,6,4,3], [2,6,3,4,5,6], R).
```

La 1re liste est la liste de commandes, la 2e est la liste à gérer et la 3e est la liste résultat. Le format d'une commande est une paire $c\,v$ où $c$ est la commande et $v$ la valeur. Une liste de commandes est une suite de 0 ou plusieurs paires $c\,v$, comme dans l'exemple d'appel. Les valeurs possibles de $c$ sont : 1- ajouter à la fin, 2- retirer la 1re occurrence, 3- retirer toutes les occurrences et 4- dupliquer la ne valeur.

La commande s'applique avec le paramètre $v$. Une commande est exécutée sur la liste modifiée par les commandes précédentes. La liste de commandes de l'exemple correspond donc à ajouter la valeur 2 à la fin, retirer la 1re occurrence de la valeur 5, retirer toutes les occurrences de la valeur 6 et dupliquer la 3e valeur. Le résultat de l'exemple est donc `R=[2,3,4,4,2]`. Assumez que la liste de commandes contient uniquement des commandes valides, donc des valeurs de $c$ valides. Retirer des valeurs qui ne sont pas dans la liste ou d'une liste vide n'est pas une erreur, c'est une commande qui ne fait rien. Dupliquer une valeur lorsque la liste est vide où lorsque la ne n'existe pas est une commande qui ne fait rien. Tout appel au prédicat devrait donc réussir et être évalué à vrai (**`true`**).

## E.4 Trouver les actions possibles

Cet exercice vous fait faire un 1er pas vers la compréhension de la problématique. Il s'agit de trouver la liste des actions qui peuvent être posées pour un état donné d'un environnement, comme celui de la figure 11.1. L'utilisation de la structure de données `set` (distribuée sur le site web de l'unité) et le prédicat **`findall`** de Prolog sont très utiles.

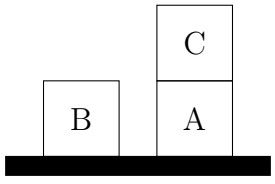



Figure 11.1 : Le monde des blocs

L'état de l'environnement est donné sous forme de liste et l'ordre des éléments n'a pas d'importance. Pour la figure 11.1, l'état serait :

```
[on(b, table), on(a, table), on(c, a), clear(b),
               clear(c), block(a), block(b), block(c)]
```

- `on(X,Y)` signifie que le bloc `X` est sur `Y`, `Y` pouvant être un autre bloc ou la table.
- `clear(X)` signifie que le bloc `X` est libre de recevoir un autre bloc ou d'être déplacé.
- `block(X)` signifie que `X` est un bloc.

Il y a deux actions possibles dans le monde des blocs :

- `move(B,X,Y)` : Signifie de déplacer le bloc `B`, de l'endroit X à l'endroit `Y`. Pour que cette action soit valable, il faut que `B` et `Y` soient libres (`clear`), que `B` soit sur `X` (`on`) et que `B` et `Y` soient des blocs (`block`). Afin de ne pas considérer des actions inutiles, `B`, `X` et `Y` doivent être tous différents.
- `moveToTable(B,X)` : Signifie de déplacer le bloc `B` de l'endroit `X` et de le mettre sur la table. Pour que cette action soit valable, il faut que `B` soit libre (`clear`), que `B` soit sur `X` (`on`) et que `B` et `X` soient des blocs (`block`). Évidemment, `B` et `X` doivent être différents. Une action dédiée pour un déplacement sur la table (`moveToTable`) est réalisée, car moins de vérifications sont nécessaires, par exemple, la table est toujours libre.

Vous devez réaliser le prédicat nommé `actionsPossibles` qui permet de trouver, sous forme de liste, toutes les actions possibles pour un état donné. Un exemple d'appel serait le suivant :

```
actionsPossibles([on(b, table), on(a, table), on(c,a),
        clear(b), clear(c), block(a), block(b), block(c)], R).
```

Le résultat donné dans R serait alors :

```
R = [move(b, table, c), move(c, a, b), moveToTable(c, a)]
```

Validez et testez votre prédicat dans différentes situations, et non pas uniquement celle de la figure 11.1. Le prédicat `actionsPossibles` est toujours évalué à vrai (**`true`**), car dans le pire des cas, il n'y aurait aucune action possible et le résultat serait alors la liste vide.

## E.5 Trouver un état successeur

Cet exercice vous fait faire un 2e pas vers la compréhension de la problématique. Vous devez réaliser le prédicat *etatSuccesseur* qui trouve l'état successeur à partir d'un état courant et d'une action. L'environnement est le même que la question précédente. Un exemple d'appel serait le suivant :

```
etatSuccesseur([on(b,table),on(a,table),on(c,a),clear(b),
  clear(c),block(a),block(b),block(c)],moveToTable(c,a),R).
```

Le premier paramètre est l'état courant de l'environnement, le deuxième est l'action à poser et le troisième est le résultat, c'est-à-dire l'état courant modifié en accord avec l'action. Assumez que l'action est valide pour l'état donné. Le résultat donné dans R serait alors le suivant :

```
R = [on(a,table),on(b,table),on(c,table),clear(a),clear(b),
     clear(c),block(a),block(b),block(c)]
```

## E.6 Se rendre à Bucarest

Afin de réaliser votre solution de cet exercice (ou de la problématique), basez vous sur les algorithmes vus à la pratique procédurale : le *tree search* et le *graph search* qui sont à la page 77 de [2]. Ces algorithmes utilisent des structures de données (pile, queue, queue avec priorité et *set*). Ces structures de données vous sont distribuées sur le site web de l'unité.

Avec les exercices précédents, vous maitrisez les notions de base du fonctionnement de Prolog. Vous maitrisez aussi certains des éléments importants de la problématique (ou de se rendre à Bucarest) : trouver les actions possibles et trouver des états successeurs. Cet exercice se rapproche encore plus de la problématique, car il requiert l'utilisation des algorithmes de *tree search* ou de *graph search*. Cet exercice est optionnel et certains vont plutôt préférer démarrer la résolution de la problématique. Pour ceux qui décident de faire cet exercice, ne vous attendez pas à le finir, en plus des précédents, dans les 3 heures.

Pour cet exercice, il s'agit de réaliser en Prolog le code nécessaire afin de trouver un chemin de Arad à Bucarest en utilisant une des techniques suivantes (ou toutes !) : profondeur d'abord, largeur d'abord, recherche avare et recherche $A^*$. Une fois le chemin trouvé, le programme doit afficher son cout total et l'ordre des villes pour arriver à destination. Afin de réaliser cet exercice, utilisez la carte de la page 68 et, si nécessaire, les informations du tableau de la page 93 de [2]. Les données de base de cet exercice vous sont données sur le site web de l'unité.

# 12 PRATIQUE PROCÉDURALE 2

### But de l'activité

- Utiliser la logique du premier ordre pour représenter des algorithmes
- Comprendre l'inférence en logique du premier ordre
- Se familiariser avec la planification
- Se familiariser avec les problèmes de la planification
- Se familiariser avec la planification par recherche de l'espace d'états
- Se familiariser avec les problèmes de la planification dans un monde réel

## 12.1 Exercices

### E.1 Logique du premier ordre et pseudocode

Exprimez cet algorithme en logique du premier ordre.

**PROCÉDURE** BonZoo
    Trouvé := faux
    Animal := premier animal de la liste des animaux du zoo
    **TANT QUE** trouvé est faux et animal est valide
        **SI** animal est un éléphant **ALORS**
            Trouvé := vrai
        **SINON**
            Animal := l'animal suivant de la liste des animaux du zoo
    **SI** trouvé est vrai **ALORS**
        BonZoo := vrai
    **SINON**
        BonZoo := faux

### E.2 Logique du premier ordre et Prolog

Exprimez ce code Prolog en logique du premier ordre. Ce code vérifie, s'il y a un passage valide d'une case vers une autre.

```
passage(X1, Y1, X2, Y1) :- X2 is X1+1, libre(X2,Y1).
passage(X1, Y1, X2, Y1) :- X2 is X1-1, libre(X2,Y1).
passage(X1, Y1, X1, Y2) :- Y2 is Y1+1, libre(X1,Y2).
passage(X1, Y1, X1, Y2) :- Y2 is Y1-1, libre(X1,Y2).
passage(X1, Y1, X2, Y2) :- X2 is X1+1, Y2 is Y1+1, libre(X2,Y2).
passage(X1, Y1, X2, Y2) :- X2 is X1+1, Y2 is Y1-1, libre(X2,Y2).
passage(X1, Y1, X2, Y2) :- X2 is X1-1, Y2 is Y1+1, libre(X2,Y2).
passage(X1, Y1, X2, Y2) :- X2 is X1-1, Y2 is Y1-1, libre(X2,Y2).
```

## E.3 Inférence

1. Qu'est-ce que l'inférence
2. À quoi sert-elle ?
3. Quelles techniques connaissez-vous pour faire de l'inférence ?

## E.4 Inférence en logique du premier ordre

1. Bibi est un bison et Coco est un cochon. Les bisons sont plus rapides que les cochons. Est-ce que Bibi est plus rapide que Coco ? Utilisez le chainage avant pour faire votre démonstration.
2. Démontrez en utilisant l'algorithme de chainage avant que Julie est la représentante de Jean.

   (1) $\forall x\ SalairePlusQue(x, 100000) \Rightarrow Client(x, Or)$
   (2) $\forall x\ Homme(x) \wedge SalairePlusQue(x, 100000) \Rightarrow EnvoyerPublicite(x)$
   (3) $\forall x\ Homme(x) \wedge Client(x, Or) \Rightarrow Representant(Julie, x)$
   (4) $SalairePlusQue(Jean, 100000)$
   (5) $Homme(Jean)$

3. Refaites le problème précédent, mais en utilisant le chainage arrière.

## E.5 Planification

1. Qu'est-ce que la planification ?
2. À quoi sert-elle ?
3. Quels sont les problèmes liés à la planification ?
4. Qu'est-ce que la planification par recherche de l'espace d'états ?
5. Qu'est-ce que la planification avec des graphes de planification ?

## E.6 Planification et le monde réel

1. Comment tenir compte du temps, d'une cédule et des ressources dans la planification ?
2. Qu'est-ce que la planification hiérarchique et quand est-elle utile ?
3. Comment peut se faire la planification si l'environnement est inconnu, partiellement observable ou non déterministe ?
4. Comment planifie-t-on s'il y a plusieurs intervenants (agents) ?

# 13 VALIDATION AU LABORATOIRE

Le but de cette activité est d'offrir du support technique dans les étapes finales de la réalisation de la problématique. Des personnes ressources seront sur place.

# A SPÉCIFICATION DU JEU BlockBlitz

## A.1 Description

*BlockBlitz* est un jeu simple. La surface de jeu, une grille, est peuplée de N joueurs intelligents (JI) et de M blocs, comme illustrés à la figure A.1. Graphiquement, les joueurs, tous des JI, sont représentés par des cercles bleus et les blocs par des carrés verts. Chaque joueur est identifié par un nombre de 1 à N et chaque bloc est identifié par un nombre de 1 à M. Le nombre qui identifie un bloc représente aussi sa valeur : plus le nombre est grand plus la valeur du bloc est grande. Initialement, les joueurs et les blocs sont disposés aléatoirement sur la surface de jeu. Les rangées et les colonnes sont numérotées à partir de 0 et la case (0,0) est en haut à gauche. Par exemple, le bloc 2 est à la case (0,1).

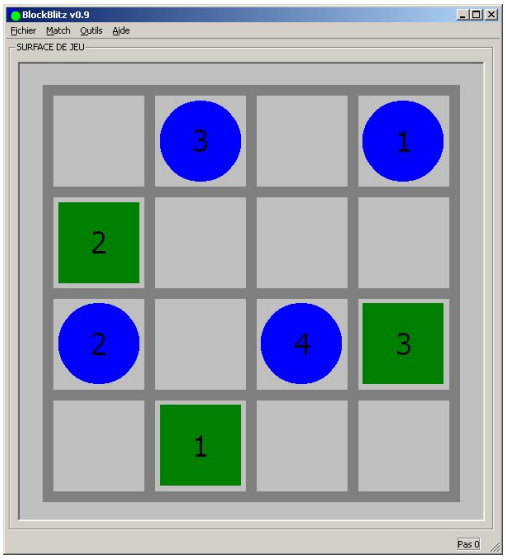



Figure A.1 : Le jeu *BlockBlitz*

Le but d'un joueur est de posséder un bloc ayant la plus grande valeur possible. Une partie se termine après un nombre fixé d'actions par chaque joueur, le mode chacun son tour, ou encore après qu'un temps déterminé se soit écoulé, le mode chrono. Peu importe le mode, un joueur n'est pas au courant de l'état d'avancement de la partie (près ou pas de la fin) ou du mode de partie (chacun son tour ou chrono).

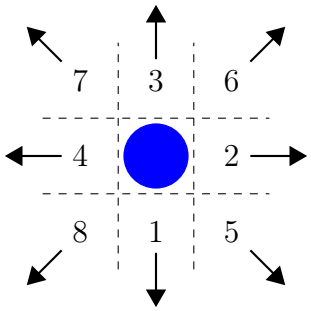
En mode chacun son tour, les joueurs posent une action chacun à leur tour. Une fois que tous les joueurs ont posé leur action, un nouveau tour de table est fait. La partie se termine lorsque le nombre de tours fixé est atteint. À chaque tour de table, l'ordre de jeu des joueurs est décidé aléatoirement. À son tour, le joueur reçoit comme information l'état courant de la surface de jeu.

En mode chrono, il n'y a plus d'ordre de jeu comme dans le mode chacun son tour. Les joueurs sont démarrés dans un ordre aléatoire et les actions des joueurs sont exécutées dans

l'ordre reçu. La partie est terminée lorsque le temps fixé est écoulé. Dans ce mode, s'il réfléchit assez rapidement, un joueur peut avoir le temps d'exécuter plusieurs actions pendant qu'un autre n'en aura exécuté aucune. Cela a comme implication que par le temps qu'un joueur décide de poser une action, il est possible que l'état de la surface de jeu ait changé et que l'action posée par le joueur ne soit plus valide : c'est un environnement dynamique. Comme dans le mode chacun son tour, le joueur reçoit l'état de la surface de jeu lorsqu'il est sollicité pour sa prochaine action.

## A.2 Actions possibles

*BlockBlitz* permet à un joueur de poser 5 actions différentes : se déplacer d'une case à une autre, prendre un bloc libre, déposer le bloc qu'il possède, attaquer un autre joueur pour obtenir le bloc de son adversaire, et évidemment, ne rien faire. Les actions respectent un format simple : `type(direction)`. Le type est l'un de ceux indiqués plus bas dans les explications de chacune des actions. Les directions sont les suivantes :




### Se déplacer

Un joueur peut se déplacer dans une des 8 cases adjacentes à condition qu'elle soit libre et dans les limites de la surface de jeu. Une case n'est pas considérée comme libre si elle contient un bloc ou un joueur. Le type de l'action est `move` et un exemple d'action est `move(2)` qui signifie que le joueur veut se déplacer d'une case vers l'est (ou la droite).

### Prendre un bloc

Un joueur peut prendre un bloc libre dans une case adjacente. Un joueur ne peut posséder qu'un seul bloc à la fois. Prendre un bloc alors qu'un joueur en possède déjà un cause un échange de bloc. Prendre d'une case hors des limites de la surface de jeu, d'une case vide ou d'une case déjà occupée par un joueur est une action invalide. Le type de l'action est `take` et un exemple est `take(3)` qui signifie de prendre le bloc dans la case vers le nord (ou en haut).

### Déposer un bloc

Un joueur peut déposer son bloc dans une case adjacente. Cette case doit être libre et dans les limites de la surface de jeu. Déposer dans une case hors des limites de la surface de jeu ou déjà occupée par un joueur ou un autre bloc est invalide. Le type de l'action est `drop` et un exemple est `drop(8)` qui signifie de déposer le bloc dans la case située au sud-ouest du joueur.

### Attaquer

Un joueur (l'attaquant) peut en attaquer un autre (l'attaqué) dans une case adjacente. Le but de l'attaquant est d'obtenir le bloc de l'attaqué. Diriger une attaque vers une case vide ou hors de la surface de jeu, vers une case contenant un bloc ou un autre joueur qui ne possède pas de bloc est une action invalide. Si l'attaquant gagne le combat, il s'approprie le bloc de l'attaqué. Si l'attaquant possède un bloc et qu'il gagne le combat, les blocs des 2 joueurs sont échangés. Dans le cas où l'attaquant perd la bataille, rien n'est changé. Le type de l'action est `attack` et un exemple est `attack(1)` qui indique d'attaquer dans la case située au sud (ou en bas) de l'attaquant.

Les chances de succès d'une attaque dépendent de la situation. Si l'attaquant ne possède pas de bloc, les chances de succès sont de 25%. Si les deux joueurs possèdent un bloc alors les chances de succès sont dictées par les valeurs respectives des blocs. Si $v_1$ est la valeur du bloc de l'attaquant et $v_2$ la valeur du bloc de l'attaqué, alors les chances de succès de l'attaquant sont de $\frac{v_1}{v_1+v_2}$.

### Ne rien faire

Un joueur peut décider de ne pas poser d'action. Le type de l'action est `none` et n'a aucun paramètre de direction et est donc sans parenthèses.

## A.3 Réalisation d'un JI

La réalisation d'un joueur informatisé (JI) consiste à programmer en Prolog les prédicats qui sont invoqués par le moteur de jeu. Ces prédicats permettent au moteur de jeu d'obtenir des informations sur le JI (nom et auteurs), de faire une remise à zéro du JI, de connaitre son plan restant et d'obtenir la prochaine action qu'il désire faire. **Ces prédicats ne doivent jamais échouer (`false`) ; ils doivent donc toujours être vrai (`true`).**

Aussi, un préfixe est ajouté au nom des prédicats pour les distinguer d'un JI à l'autre. Le préfixe est une valeur qui vous sera indiquée une fois que les équipes seront formées. Le site web de l'unité donne un exemple simplifié de JI afin de montrer des exemples de ces prédicats. **Basez-vous sur cet exemple pour démarrer votre propre JI.** Les prédicats peuvent être testés avec l'application *JITest* qui est décrite à l'annexe B.

Les descriptions qui suivent utilisent la convention de Prolog pour indiquer les paramètres en entrée (`+`) et les paramètres en sortie (`-`) d'un prédicat. Ces descriptions utilisent comme préfixe un `X`.

### Prédicat de nom

Ce prédicat, `X_nom(-Nom)`, est appelé par le moteur de jeu afin de connaitre le nom du JI et de pouvoir l'afficher à titre informatif sur l'interface graphique du jeu. Il est appelé une seule fois lors de l'initialisation du jeu. Ce prédicat doit toujours être vrai. Un exemple de résultat de ce prédicat est `Nom = Big Ben`.

### Prédicat d'auteurs

Ce prédicat, `X_auteurs(-Auteurs)`, est appelé par le moteur de jeu afin de connaitre les auteurs d'un JI et de pouvoir l'afficher à titre informatif sur l'interface graphique du jeu. Il est appelé une seule fois lors de l'initialisation du jeu. Ce prédicat doit toujours être vrai. Un exemple de résultat de ce prédicat est `Auteurs = Jim et Bob`.

### Prédicat de remise à zéro

Ce prédicat, `X_reset`, est appelé par le moteur de jeu afin de faire une remise à zéro du JI, par exemple, au début d'une nouvelle partie. Ce prédicat peut être appelé à tout moment et doit toujours être vrai. Ce que réalise ce prédicat dépend de l'implémentation du JI. Il peut permettre, par exemple, de remettre des valeurs par défaut dans certaines variables d'états. Dans sa forme la plus simple, ce prédicat ne fait rien.

### Prédicat de plan

Ce prédicat, `X_plan(-Plan)`, est appelé par le moteur de jeu afin de connaitre le *plan restant* d'un JI et de pouvoir l'afficher à titre informatif sur l'interface graphique du jeu. Il est appelé régulièrement lors d'une partie afin de montrer le *plan restant* du JI et donc les *actions restantes* d'un plan. Ce prédicat doit toujours être vrai.

Un plan est simplement une liste d'action. **Un plan n'est jamais vide**, car même un JI qui décide ne rien faire a un plan qui contient l'action `none`. Un exemple de résultat de ce prédicat est `Plan=[move(4),move(1),take(8)]`.

Le prédicat de plan peut être appelé en parallèle au prédicat d'action, le prochain à être décrit, et donc pendant que ce dernier est peut être en train de modifier le plan.

### Prédicat d'action

Ce prédicat, `X_action(+Etat, -Action)`, est l'intelligence du JI. Ce prédicat reçoit en entrée l'état de la surface de jeu (`Etat`) et donne en retour l'action que le JI désire poser (`Action`). Ce prédicat doit toujours être vrai. Un JI qui décide de ne pas poser d'action doit donc donner `none` comme action et le prédicat ne doit pas échouer. Ce que ce prédicat fait dépend des choix des concepteurs du JI. Par exemple, ce prédicat peut être amené à recalculer un nouveau plan complet et donc de changer complètement de plan. Par contre, si le plan prévu se déroule comme prévu, ce prédicat peut être amené à simplement exécuter la prochaine action du plan et de le mettre à jour avec une action de moins étant donné que cette action sera exécutée.

Le prédicat de plan et d'action sont donc liés. Par exemple, si le prédicat d'action est appelé et qu'un nouveau plan composé de 4 actions est trouvé, alors la première action du plan est donnée en sortie du prédicat d'action (l'action que le JI désire exécuter), et les 3 actions restantes du plan seront le plan restant donné par un prochain appel du prédicat de plan.

Ce prédicat peut être appelé en parallèle à au prédicat de plan et donc pendant que le prédicat de plan est en train de lire le plan courant. Un exemple de résultat de ce prédicat est `Action = move(4)` et un exemple d'état (celui de la figure A.1) est :

```
[4,3,4,4,
 [[2,'Brutus',0,2,0],[3,'Zouf',1,0,0],[1,'Ares',3,0,0],
  [4,'Buddy',2,2,0]],[[1,1,3],[3,3,2],[2,0,1]]]
```

Cet exemple montre que l'état de la surface de jeu est spécifié par une liste de 6 items : `[n,m,c,r,p,b]` avec

- `n` : le nombre de joueurs, un entier
- `m` : le nombre de blocs, un entier
- `c` : le nombre de colonnes de la surface de jeu, un entier
- `r` : le nombre de rangées de la surface de jeu, un entier
- `p` : les états des joueurs, une liste
- `b` : les états des blocs libres, une liste

La liste `p` comporte `n` items et chacun d'eux indique l'état d'un joueur. L'état d'un joueur est une liste de la forme `[id,nom,x,y,b]` : `id` est le numéro du joueur (un entier), `nom` est le nom du joueur (celui fourni par le prédicat de nom), `x` est l'index de la colonne de sa position (un entier), `y` est l'index de la rangée de sa position (un entier) et `b` est le numéro du bloc qu'il possède (un entier). Un bloc de numéro 0 indique que le joueur ne possède pas de bloc. Notez qu'il n'y a pas de lien entre le numéro d'un joueur (`id`) et le préfixe `X` des prédicats. La liste `b` comporte `m` items ou moins et chaque item indique l'état d'un bloc libre. L'état d'un bloc libre est une liste de la forme `[id,x,y]` : `id` est le numéro et la valeur du bloc (un entier), `x` est l'index de la colonne de sa position (un entier), `y` est l'index de la rangée de sa position (un entier).

## A.4 Remise finale d'un JI

Il est normal qu'un JI soit développé dans plusieurs fichiers Prolog et il est encouragé de le faire. Par contre, à cause de considérations dues au fait qu'il y a plusieurs JI dans une séance de jeu de *BlockBlitz*, **le code du JI remis lors de votre dépôt doit être dans un seul fichier**.

Si un JI est réalisé dans plusieurs fichiers, il suffit de fusionner les différents fichiers en 1 seul pour la remise. Entre autres, assurez-vous que les structures de données que vous utilisez (`stack`, `queue` et `set`) sont bien présentes dans le fichier. Si vous utilisiez des prédicats de fichiers tels que `ensure_loaded`, **`consult`** et `load_files`, assurez-vous de les retirer, puisqu'ils ne sont plus nécessaires. Tout doit être dans 1 seul fichier.

Il est aussi important de retirer toute forme d'affichage sur la console, par exemple avec les prédicats **`write`**, **`print`**, **`listing`**, **`tab`** et **`nl`**.

# B AIDE AU TESTS : JITest

## B.1 Description

*JITest* sert à valider votre joueur informatisé (JI). Il permet de charger votre JI, codé en un ou plusieurs fichiers Prolog, et de valider les prédicats d'interface (PI) avec *BlockBlitz*. Si votre JI est réalisé en plusieurs fichiers, les prédicats de fichiers, tels `ensure_loaded`, **`consult`** et `load_files`, sont utiles. *JITest* peut aussi être utilisé pour valider votre fichier Prolog de dépôt électronique afin de vous assurer qu'il ne manque rien. Lisez la section A.4 à ce sujet.

Vous pouvez créer la surface de jeu que vous voulez et voir quels seraient les résultats des appels aux PI. Tout comme dans *BlockBlitz*, un joueur est représenté graphiquement par un cercle bleu et un bloc par un carré vert, comme le montre la figure B.1.

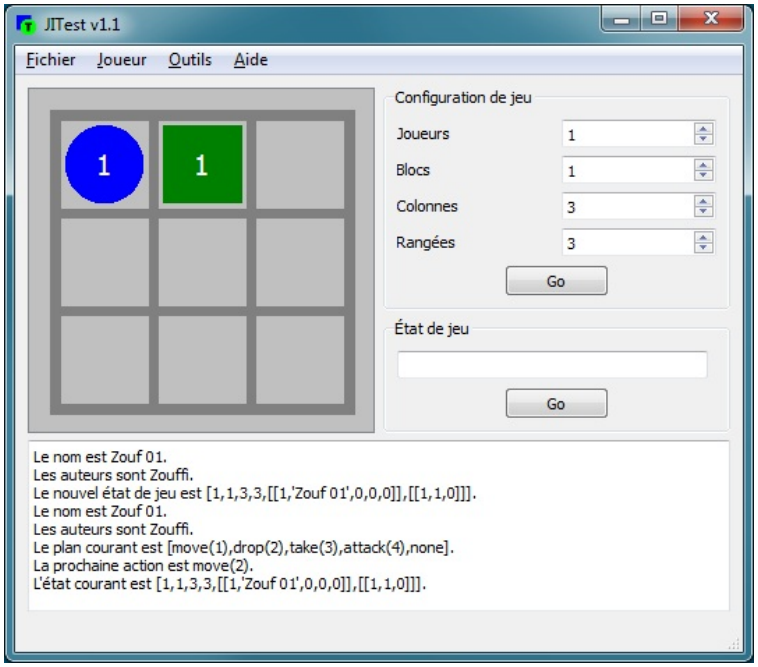



Figure B.1 : L'outil de test de JI : *JITest*

## B.2 Usage

L'utilisation de *JITest* est simple. Tout de même, voilà quelques indications.

1. Chargez votre JI avec le menu `Fichier`Charger joueur|. Le numéro de votre JI est décidé aléatoirement.
2. Ajustez la configuration de jeu dans la partie de droite et pesez sur le bouton `Ok`. Une surface de jeu apparait alors dans la partie de gauche. Vous pouvez changer la configuration de jeu autant de fois que vous le voulez.

3. En manipulant la surface de jeu avec la souris, vous pouvez obtenir l'état que vous désirez. Consultez la section suivante pour plus de détails sur la manipulation de la surface de jeu.
4. Utilisez le menu `Joueur` pour invoquer les PI de votre joueur : nom, auteurs, plan et action. Le prédicat qui permet d'obtenir la prochaine action du JI sera invoqué avec l'état de l'environnement de jeu qui est affiché. Les résultats des appels sont affichés dans la console au bas de la fenêtre.

## B.3 Manipulation de la surface de jeu

**Déplacement :** Pour déplacer un joueur ou un bloc, cliquez-le avec le bouton de gauche de la souris, déplacez-le au-dessus d'une cellule vide, puis relâchez le bouton.

**Donner un bloc à un joueur :** Il suffit de déplacer un bloc sur la case d'un joueur qui n'a pas de bloc.

**Retirer le bloc d'un joueur :** Il suffit de tenir la touche `shift` enfoncée et de faire comme si vous vouliez déplacer le joueur qui a un bloc. En fait, vous allez extraire le bloc et vous pourrez alors le déplacer et le déposer où vous le désirez.

## B.4 Autres fonctions utiles

**Recharger un fichier de joueur :** Utiliser le menu `Fichier|Recharger joueur`.

**Effacer la console :** Utiliser le menu `Outils|Effacer console`.

**Afficher l'état courant du jeu :** Utiliser le menu `Outils|Afficher etat de jeu`.

# LISTE DES RÉFÉRENCES

[1] Rosen, K. H. (2002). *Mathématiques discrètes, édition révisée*. Chenelière–McGraw-Hill, 798 p.

[2] Russell, S. J. et Norvig, P. (2010). *Artificial Intelligence - A Modern Approach*, 3e édition. Pearson Education, Upper Saddle River, NJ, USA, 1132 p.