Wumpus IA

Ulysee Brehon, Luis Enrique González Hilario

May 2020

Contents

1 Introduction

La première phase du projet vise à cartographier l'environnement de jeu Wumpus en appliquant les éléments de la logique propositionnelle et les règles du jeu. Ceci afin de ne pas révéler exhaustivement la grille. Notre objectif? 1 seul, dépensez le moins d'or intelligemment (logiquement, bien sûr)

2 Environnement

Notre programme est entièrement dans le fichier: **cartographie.py**

La variable **gophersat_exec** contient le chemin gophersat pour linux. En cas d'exécution sur Windows, remplacez-le simplement par: **./lib/gophersat**

Pour l'executer, simplement run **cartographie.py**

3 Caractéristiques du programme

- **Vocabulaire** Il y a 5 fonctions pour créer des listes de tous les symboles (la quantité dépend du taille de la grille): generate_wumpus_voca (W), generate_stench_voca (S), generate_gold_voca (G), generate_brise_voca (B) et

generate_trou_voca (T)

- **Règles** Nous avons 7 règles pour créer notre "cerveau déductif" dans le jeu de Wumpus:
  - **insert_only_one_wumpus_regle:** W$_{i=a,j=b} \Rightarrow \wedge(\neg W_{i\neq a,j\neq b})$

- **insert_safety_regle:** $\neg W_{0,0} \wedge \neg T_{0,0}$

- **insert_trou_regle:** $(\neg T_{i,j} \vee B_{i-1,j}) \wedge (\neg T_{i,j} \vee B_{i+1,j}) \wedge (\neg T_{i,j} \vee B_{i,j-1}) \wedge (\neg T_{i,j} \vee B_{i.j+1})$

- **insert_brise_regle:** $\neg B_{i,j} \vee T_{i-1,j} \vee Ti+1, j \vee T_{i,j-1} \vee T_{i,j+1}$

- **insert_wumpus_stench_regle:** $(\neg W_{i,j} \vee S_{i-1,j}) \wedge (\neg W_{i,j} \vee S_{i+1,j}) \wedge (\neg W_{i,j} \vee S_{i,j-1}) \wedge (\neg W_{i,j} \vee S_{i.j+1})$

- **insert_stench_regle:** $\neg S_{i,j} \vee W_{i-1,j} \vee W_{i+1,j} \vee Wi, j-1 \vee Wi, j+1$

- **insert_une_menace_par_case_regle:** $(\neg W_{i,j} \vee \neg T_{i,j}) \wedge (\neg T_{i,j} \vee \neg W_{i,j})$

4 Fonctions pour aider à la prise de décision

- **is_wumpus_possible**: Tester la satisfiabilité si on ajoute un Wumpus dans la position donnée sous la forme (i,j)
- **is_trou_possible**: Tester la satisfiabilité si on ajoute un trou dans la position donnée sous la forme (i,j)
- **get_implicit_negative_facts**: Si B$_{i,j} estVrai.Donc \wedge(\neg S, \neg W, ...dans(i,j))$

5 Résultats importantes

- **Satisfiabilité:** Variable Boolean
- **Modèle:** Modèle retourné par le solveur SAT
- **Cout en Or:** Nôtre chiffre: 366