TRAVAIL PRATIQUE 1 (15 % de la note finale)
IFT 2003 Intelligence artificielle I

Ce travail collaboratif vise à concevoir et implémenter une intelligence artificielle capable de jouer de manière autonome à un jeu choisi, en utilisant des techniques de recherche heuristique.

Voici les principaux éléments du projet :

Objectif principal :
Développer un programme qui joue automatiquement à un jeu sélectionné, en utilisant une technique de recherche heuristique pour déterminer les meilleurs coups à effectuer.

Organisation et délais :

- Équipes : Groupes de 3 à 4 membres
- Date limite de formation des équipes : 15 février 2025, 23h59. Tout travail remis en retard ne sera pas évalué et recevra la note 0, à condition qu'il y ait eu une entente préalable (entente prise avant la date de remise).
- Remise : avant le 1e mars 2025 (9h00) via le Portail des cours (fichier PDF pour le rapport, fichier PL pour le programme)

Sélection du jeu :

- Jeux exclus : taquin, tic-tac-toe, nim (jeux utilisés dans les notes de cours)
- Préférence pour des jeux aux règles simples (ou simplifiables)
- Possibilité de jeux en duel ou individuels

Développement du projet

1. Création d'un joueur IA utilisant la recherche heuristique
2. Application stricte de la méthodologie de résolution par recherche heuristique dans un espace d'états, telle que présentée dans le cours

Points clés

- La simplification des règles du jeu est autorisée si nécessaire
- Seule l'approche de résolution par recherche heuristique sera considérée
- L'accent est mis sur l'implémentation efficace de l'algorithme de recherche

Ce projet est une opportunité unique d'appliquer les concepts théoriques de l'intelligence artificielle à un cas pratique, tout en développant des compétences en programmation et en travail d'équipe.

ÉTAPES DE TRAVAIL

Le travail à réaliser se décompose en 5 étapes principales, chacune contribuant de manière spécifique à la note finale du projet. Pour chaque étape, un pourcentage de la note totale est attribué, reflétant son importance dans l'évaluation globale. De plus, un nombre de pages recommandé est indiqué comme guide sur l'étendue attendue de chaque section. Il est important de noter que le nombre de pages suggéré pour chaque étape sert de référence et peut être ajusté si nécessaire, notamment pour inclure des figures, des diagrammes ou des tableaux pertinents. La qualité et la pertinence du contenu sont privilégiées par rapport à la quantité stricte de pages.

1. Introduction (10 %, 1/2 page).

- Choisir un jeu quelconque, sauf les jeux suivants : taquin, tic-tac-toe, nim (jeux utilisés dans les notes de cours). Il est préférable de choisir un jeu avec des règles simples ou alors il est possible de modifier les règles afin qu'elles soient plus simples. Le jeu peut se jouer à 1 ou deux joueurs.
- Décrire le jeu choisi et expliquer les règles de ce jeu. Cette étape est importante pour faciliter la compréhension de l'équipe de correction.

2. Représentation du jeu (30%, 2 pages).

- Représenter le jeu par un espace d'états, et expliquer en quoi consiste un état, l'état initial, le ou les états finaux, les opérations possibles.
- Choisir une technique de recherche heuristique parmi celles vues dans le cours. Indiquer la technique choisie et décrire la ou les fonctions heuristiques qui seront utilisées par la technique de recherche.
- Illustrer avec des exemples.

3. Programmation (30%, fichier PL).

- Programmer la solution proposée en langage Prolog. L'interface utilisateur n'est pas un aspect primordial pour ce travail. L'accent est mis sur la technique de recherche heuristique. Aussi, aucune interface graphique n'est demandée, la saisie par l'interpréteur est tout à fait acceptable, et au besoin adapter l'interface de jeu.
- Documenter le programme : les prédicats doivent avoir une description surtout pour les plus importants et la description peut être plus succincte pour ceux moins importants.
- Produire un guide d'utilisation en entête du programme.

4. Discussion (20 %, 3 pages).

- Reprendre la partie de code qui correspond à la technique de recherche heuristique. Expliquer cette partie de code en détail en faisant le lien avec la description en (2).
- Donner les tests effectués pour valider le programme et les résultats obtenus.
- Analyser les résultats obtenus : les buts fixés sont-ils atteints ? La ou les fonctions heuristiques ont donné de bons résultats ? Y a-t-il des limites ? Comment améliorer ?

5. Rédaction et relecture (10%).

- Intégrer les parties 1, 2 et 4 dans un même rapport. Ajouter une page couverture (titre, auteurs, date, etc.), une conclusion (résumé du travail accompli et ce que vous auriez pu ou aimé faire de plus), une table des matières et, s'il y a lieu, une bibliographie en précisant comment vous avez utilisez les sources citées.
- Relire le rapport et l'évaluer en fonction de la grille d'autoévaluation donnée en annexe. Ajuster le rapport au besoin.

REMARQUES

1) Un exemple de programme Prolog qui implémente un joueur pour le jeu du taquin est donné en annexe. Dans ce programme, la technique de recherche choisit le meilleur coup à jouer parmi les enfants de l'état courant.

2) Si un outil d'intelligence artificielle générative est utilisé dans le cadre de ce travail, ajouter une page supplémentaire contenant les informations suivantes :

Justification de l'utilisation

- Expliquer brièvement pourquoi vous avez choisi d'utiliser un outil d'IA générative pour cette tâche.
- Préciser le nom et la version de l'outil utilisé (par exemple, ChatGPT, DALL-E, Midjourney).

Description de l'utilisation

- Détailler la ou les questions posées à l'outil d'IA.
- Si pertinent, mentionner les paramètres spécifiques utilisés (par exemple, le niveau de créativité).

Bénéfices obtenus

- Décrire les avantages concrets apportés par l'utilisation de l'outil.
- Expliquer en quoi cela a amélioré ou facilité votre travail.
- Quantifier, si possible, le gain de temps ou d'efficacité réalisé.

Vérification de la véracité

- Détailler la méthodologie utilisée pour vérifier l'exactitude des informations fournies.
- Citer les sources primaires consultées pour corroborer les réponses.
- Mentionner les éventuelles corrections ou ajustements apportés aux réponses.

Cette page supplémentaire doit être rédigée de manière claire et concise. Elle vise à assurer la transparence sur l'utilisation de l'IA et à démontrer une approche réfléchie et responsable dans son application.

Annexe 1 – Grille d'autoévaluation

Description du jeu choisi (10%)				
Présentation du jeu	Le but du jeu est présenté ainsi que des exemples.	Le but du jeu est présenté ou des exemples sont donnés.	Les explications et/ou les illustrations données sont incorrectes ou incomplètes.	Le but du jeu n'est pas présenté et aucun exemple n'est donné.
Description des règles	Les règles du jeu sont expliquées et illustrées.	Les règles du jeu sont expliquées ou illustrées.	Les explications et/ou les illustrations données sont incorrectes ou incomplètes.	Les règles du jeu ne sont ni expliquées, ni illustrées.
Modélisation du problème et de la solution (30%)				
Description du problème	Tous les éléments du problème sont expliqués et illustrés en utilisant l'approche par espace d'états.	Les éléments du problème sont expliqués ou illustrés en utilisant l'approche par espace d'états.	Les explications et/ou les illustrations données sont incorrectes ou incomplètes.	Les éléments du problème ne sont ni expliqués, ni illustrés.
Description de la solution	Tous les éléments de la solution sont expliqués et illustrés.	Les éléments de la solution sont expliqués ou illustrés.	Les explications et/ou les illustrations données sont incorrectes ou incomplètes.	Les éléments du problème ne sont ni expliqués, ni illustrés.
Implantation (30%)				
Fonctionnement	Le programme compile sans erreur et joue correctement.	Le programme compile sans erreur et joue correctement au moins 50 % du temps.	Le programme compile sans erreur mais joue correctement moins de 50 % du temps.	Le programme ne compile pas sans erreur.
Documentation et guide d'utilisation	Les prédicats sont documentés et un guide d'utilisation est présent.	Les prédicats ne sont pas documentés, mais un guide d'utilisation est présent.	Les prédicats sont documentés mais le guide d'utilisation est absent.	Pas de documentation et pas de guide d'utilisation.
Résultats et discussion (20%)				
Explication du code	Le code de la technique utilisé est dans le rapport et bien détaillé en lien avec la représentation.	Le code de la technique utilisé est dans le rapport mais l'explication manque de détails.	Seul le code de la technique utilisé est dans le rapport, sans explication.	Aucun code présenté.
Jeux d'essais	Plusieurs jeux d'essais sont présentés avec les résultats.	Plusieurs jeux d'essais sont présentés mais pas les résultats.	Un seul jeu d'essai est présenté.	Pas de jeu d'essai ou de résultat.
Analyse des résultats	Les tests sont analysées par rapport aux attentes.	Les tests sont analysées indépendamment des attentes.	L'analyse ne prend pas en compte les tests relatés.	Aucune analyse.
Appréciation globale (10%)				
Expression écrite	Le rapport ne contient aucune faute (vocabulaire, grammaire, syntaxe, etc.).	Le rapport ne contient pas plus d'une dizaine de fautes (vocabulaire, grammaire, syntaxe, etc.).	Le rapport ne contient pas plus de 5 fautes par page (vocabulaire, grammaire, syntaxe, etc.).	Le rapport contient plus de 5 fautes par page (vocabulaire, grammaire, syntaxe, etc.).
Présentation du rapport	Le format proposé est respecté. Le rapport est paginé.	Le rapport n'est pas paginé ou il manque un élément du format.	Le rapport n'est pas paginé. Il manque 2 éléments du format.	Il y a plus de 2 éléments du format qui ont été oubliés.

Annexe 2 – Programme Jeu du taquin

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% resolveur du taquin (la technique de recherche implantée choisit le
% meilleur noeud seulement parmi les noeuds enfants et non parmi tous
% les noeuds ouverts)
%
% La question a poser est : ?- resolveur( Deplacements ).
% Deplacements donne tous les deplacements a faire pour atteindre la situation finale
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

resolveur(Deplacements) :-
	configuration_initiale( CI),
	configuration_finale(CF),
	resoudre(CI, CF, [], Deplacements).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Technique de recherche
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
resoudre( Configuration, Configuration, _, [] ).
resoudre( Configuration, ConfigurationFinale, Deja_generees, [Deplacement | Deplacements] ) :-
	trouver_deplacements_legaux( Depls_Possibles, Configuration ),
	generer_configurations( Depls_Possibles, Configuration, ListeConfigurations ),
	valider( ListeConfigurations, Depls_Possibles, Deja_generees, ListeConfsValides, DeplsValides ),
	valeur_VH_tous( ListeConfsValides, ConfigurationFinale, ListeValeurs ),
	faire_liste( ListeValeurs, DeplsValides, ListeConfsValides, ListeConfsValuees ),
	trier( ListeConfsValuees, ListeConfsTriees ),
	member( [ V, Deplacement, NouvelleConf ], ListeConfsTriees ),
	resoudre( NouvelleConf, ConfigurationFinale, [NouvelleConf | Deja_generees], Deplacements ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats necessaires au jeu du taquin
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Situation de depart et situation a atteindre
configuration_initiale([1,*,4,8,3,5,7,2,6]).
configuration_finale([1,4,5,8,3,*,7,2,6]).

% Les deplacements possibles de l'asteristique
deplacer( deplacer_vers_gauche, De, A ) :- A is De - 1.
deplacer( deplacer_vers_haut, De, A ) :- A is De - 3.
deplacer( deplacer_vers_droite, De, A ) :- A is De + 1.
deplacer( deplacer_vers_bas, De, A ) :- A is De + 3.

% Les contraintes du jeu (ne pas sortir des 9 positions possibles
contraintes( deplacer_vers_gauche, [1, 4, 7] ).
```

```
contraintes( deplacer_vers_haut, [1, 2, 3] ).
contraintes( deplacer_vers_droite, [3, 6, 9] ).
contraintes( deplacer_vers_bas, [7, 8, 9] ).

% Determine le premier deplacement possible (gauche, haut, droit, bas)
trouver_deplacement_legal( Deplacement, Configuration ) :-
	position_element( Configuration, '*', P),
	contraintes( Deplacement, C),
	not( member( P, C ) ).

% Determine tous les deplacements legaux qui s'appliquent a une configuration
trouver_deplacements_legaux( Deplacements, Configuration ) :-
	findall( X, trouver_deplacement_legal( X, Configuration ), Deplacements ).

% Genere une nouvelle configuration (C2) a partir d'une autre (C1) en appliquant Deplacement
generer( Deplacement, C1, C2 ) :-
	position_element( C1, '*', P ),
	deplacer( Deplacement, P, P1 ),
	element_position( C1, Element, P1 ),
	remplacer( P1, '*', C1, C3 ),
	remplacer( P, Element, C3, C2 ).

% Determine toutes les configurations obtenues a partir d'une configuration en appliquant la liste
des deplacements
generer_configurations( [], Configuration, [] ).
generer_configurations( [ Deplacement | Deplacements ], Configuration, [ Config | Configs ] ) :-
	generer( Deplacement, Configuration, Config ),
	generer_configurations( Deplacements, Configuration, Configs ).

% Si une configuration a deja ete generee, on la rejette ainsi que le deplacement associe
valider( [], _, _, [], [] ).
valider( [ C | Configurations ], [ D | Deplacements ], Deja_generees, [C | Configs], [D | Depls] ) :-
	not( member( C, Deja_generees ) ),!,
	valider( Configurations, Deplacements, Deja_generees, Configs, Depls ).
valider( [ C | Configurations ], [ D | Deplacements ], Deja_generees, Configs, Depls ) :-
	valider( Configurations, Deplacements, Deja_generees, Configs, Depls ).

% Calcul de la valeur de la fonction heuristique
valeur_VH_tous( [], _, [] ).
valeur_VH_tous( [C|Configs], CF, [V|Valeurs] ) :-
	valeur_VH( C, CF, V ),
	valeur_VH_tous( Configs, CF, Valeurs ).

valeur_VH( [], [], 0 ) :- !.
valeur_VH( [X | Xs], [X | Ys], V_HC ) :-
	valeur_VH( Xs, Ys, V_HC ), !.
valeur_VH( [X | Xs], [Y | Ys], V_HC ) :-
	valeur_VH( Xs, Ys, V_HC1 ), V_HC is V_HC1 + 1.


% faire une seule liste de triplet a partir de trois listes
```

```
faire_liste( [], [], [], [] ).
faire_liste( [X | Xs], [Y | Ys], [Z | Zs], [[X,Y,Z] | Reste] ) :-
        faire_liste( Xs, Ys, Zs, Reste ).

% trier (algorithme de tri QuickSort)
trier( [], [] ).
trier( [X|Reste], ListeTriee ) :-
        partitionner( Reste, X, Les_Petits, Les_Grands ),
        trier( Les_Petits, Ps ),
        trier( Les_Grands, Gs ),
        append( Ps, [X | Gs], ListeTriee ).

partitionner( [], _, [], [] ).
partitionner( [ [X1,X2,X3] | Xs], [Y1,Y2,Y3], [ [X1,X2,X3] | Ps], Gs ) :-
        X1 < Y1, !,
        partitionner( Xs, [Y1,Y2,Y3], Ps, Gs).
partitionner( [ [X1,X2,X3] | Xs], [Y1,Y2,Y3], Ps, [ [X1,X2,X3] | Gs] ) :-
        partitionner( Xs, [Y1,Y2,Y3], Ps, Gs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats de service (on peut aussi utiliser les prédicats prédéfinis)
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% position_element/3 : recherche la position d'un element dans une liste
position_element( [ Element | _ ], Element, 1 ) :- !.
position_element( [ _ | Reste ], Element, P ) :-
                        position_element( Reste, Element, NP ), P is NP + 1.

% element_position/3 : recherche l'element a une position donnee dans une liste
element_position( [ Element | _ ], Element, 1 ) :- !.
element_position( [ _ | Reste ], Element, P ) :-
                         NP is P - 1, element_position( Reste, Element, NP).

% remplacer/3 : remplace un enieme element d'une liste par un autre
remplacer( 1, Nouveau, [ Element | Reste ], [ Nouveau | Reste ] ) :- !.
remplacer( N, Nouveau, [ Element | Reste ], [ Element | Final ] ) :-
                        NP is N - 1, remplacer( NP, Nouveau, Reste, Final ).
```