Rapport du Projet

PontuXL

Loic Franck KAMGAIN MEUPIAP
Tresor KENFAK SOKENG
Aubry MBIENDOU

Groupe 8 – AA-24-25, INFO B317
Intelligence Artificielle et programmation symbolique
(2024 - 2025)
Faculty of Computer Science, University of Namur, Belgium

Table des matières

1 Introduction

Le projet PontuXL est une implémentation numérique du jeu de société Pontu, développé dans le cadre du cours INFO B317 - IA et programmation symbolique. Cette version du jeu a été adaptée pour quatre joueurs, avec une particularité majeure : deux des joueurs sont contrôlés par une intelligence artificielle avancée implémentée en Prolog.

2 Objectifs du Projet

- Implémenter une version fonctionnelle du jeu Pontu pour quatre joueurs
- Développer une intelligence artificielle en Prolog utilisant l'algorithme Max^n avec élagage superficiel
- Créer une interface graphique intuitive et réactive
- Intégrer un assistant conversationnel (bot) pour aider les joueurs à comprendre les règles
- Assurer une communication fluide entre JavaScript (frontend) et Prolog (IA)

3 Architecture du projet

Le projet est organisé selon l'architecture suivante :

```
Pontu-Game/
├── index.html            # Page principale du jeu
├── game.js               # Logique principale du jeu en JavaScript
├── styles.css            # Styles CSS pour l'interface
├── pontu_ai.pl           # Intelligence artificielle en Prolog
├── prolog_interface.js   # Interface entre JavaScript et Prolog
├── pbot-elm.pl           # Bot conversationnel en Prolog
├── bot.js                # Interface JavaScript pour le bot
├── README.md             # Documentation du projet
└── favicon.ico           # Icône du site
```

4 Technologies Utilisées

HTML5/CSS3 : Structure et mise en forme de l'interface utilisateur

JavaScript : Logique du jeu et interactions utilisateur

Prolog : Implémentation de l'intelligence artificielle et du bot conversationnel

Tau Prolog : Bibliothèque permettant d'exécuter du code Prolog dans le navigateur

5 Composants Principaux

5.1 Interface Utilisateur (`index.html`, `styles.css`)

L'interface utilisateur est conçue pour être intuitive et visuellement attrayante. Elle comprend :

- Un plateau de jeu 6x6 avec des cases et des ponts
- Des lutins de quatre couleurs différentes (vert, bleu, jaune, rouge)
- Un panneau d'information indiquant le joueur actuel et l'état du jeu
- Un système de chat pour interagir avec le bot explicateur
- Des boutons de contrôle (nouvelle partie, aide)

5.2 Logique du Jeu (`game.js`)

Le fichier `game.js` contient toute la logique du jeu :

- Initialisation du plateau et placement des lutins
- Gestion des tours de jeu
- Validation des mouvements
- Détection des conditions de victoire/défaite
- Animation des actions (déplacement des lutins, suppression des ponts)

Des effets visuels spécifiques ont été implémentés pour améliorer l'expérience utilisateur, notamment :

- Animation de pulsation en rouge pour les ponts avant leur suppression
- Message textuel "Pont supprimé" apparaissant au-dessus du pont retiré
- Bordure rouge autour du pont pour le rendre plus visible
- Mise à jour garantie de l'interface après chaque action

5.3 Intelligence Artificielle (`pontu_ai.pl`)

5.3.1 Logique d'IA

L'intelligence artificielle (IA) du jeu est implémentée en `Prolog` et repose sur l'algorithme Max^n avec *élagage superficiel* (*shallow pruning*), spécialement adapté aux jeux à plus de deux joueurs. L'objectif est de sélectionner le coup le plus pertinent en équilibrant stratégie défensive et offensive, tout en optimisant les performances de calcul.

Étapes générales de prise de décision

1. **Génération des coups possibles**
   L'IA analyse l'état actuel du plateau afin de déterminer l'ensemble des actions légales disponibles pour le joueur courant.
2. **Évaluation des coups**
   Chaque coup est évalué à l'aide des deux heuristiques définies :

— **H1 : Mobilité propre** — maximise la liberté de mouvement du joueur.
— **H2 : Isolation adverse** — réduit la mobilité des adversaires.

Les scores sont éventuellement *pondérés* et *normalisés* pour un équilibre stratégique.

3. **Sélection du meilleur coup**
   L'IA choisit le coup présentant la valeur heuristique globale la plus élevée.
4. **Optimisation par élagage**
   Des techniques d'optimisation telles que l'*élagage superficiel* (*shallow pruning*) sont appliquées pour réduire l'espace de recherche et accélérer la prise de décision.

Algorithme Max^n
Contrairement à l'algorithme Minimax, adapté aux jeux à deux joueurs (un maximise son score tandis que l'autre le minimise), l'algorithme Max^n est conçu pour les jeux multi-joueurs. Chaque joueur tente de maximiser son propre score indépendamment des autres, ce qui permet une modélisation plus réaliste des interactions entre plusieurs adversaires.

1. À chaque état du jeu, un n-uplet de scores est calculé (un score par joueur).
2. Chaque joueur sélectionne le coup qui maximise son propre score.
3. L'*élagage superficiel* est appliqué pour éviter d'explorer des branches inutiles.

Implémentation en Prolog
Les principales fonctions développées sont :

— `trouver_meilleur_coup/2` : point d'entrée pour déterminer le coup optimal.
— `maxn_shallow/4` : implémentation de l'algorithme Max^n avec élagage superficiel.
— `evaluer_etat_tous_joueurs/2` : évalue l'état du jeu pour l'ensemble des joueurs.
— `obtenir_coup_ia_maxn/5` : interface principale pour obtenir le coup de l'IA.
— `obtenir_coup_ia/5` : version standard utilisée comme solution de secours.

5.3.2 Shallow Prunning : explication et contexte

Partie A : Explication

Le shallow pruning (ou élagage superficiel) est une technique utilisée en intelligence artificielle et plus particulièrement dans les algorithmes de recherche (comme la recherche d'arbre en IA, par exemple dans le jeu d'échecs ou de go).

L'idée est simple :

— On coupe (prune) certaines branches de l'arbre de recherche avant de les explorer complètement.

— Contrairement au deep pruning qui coupe des branches entières en se basant sur des informations provenant de niveaux plus profonds, le shallow pruning se fait plus tôt, avec moins d'informations, souvent juste après avoir exploré quelques coups ou quelques niveaux.

En pratique :

— **But** : gagner du temps en réduisant le nombre de nœuds explorés.

— **Inconvénient** :comme la décision de couper est prise tôt, il y a un risque plus élevé de passer à côté d'un meilleur chemin, car on ne regarde pas assez profondément.

— **Exemple** : dans un algorithme minimax avec alpha-beta pruning, le shallow pruning pourrait se produire si, dès le début d'un nœud, on juge qu'il est peu prometteur et on n'explore pas ses descendants — même si, plus loin, il aurait pu donner un meilleur résultat.

En résumé : Le shallow pruning est un **élagage anticipé** de l'arbre de recherche, qui accélère le calcul mais peut réduire la précision des résultats car il coupe avec peu d'informations.

Bon à savoir : Le deep pruning est une technique d'optimisation dans les algorithmes de recherche où l'on ignore certaines branches de l'arbre, mais seulement après avoir exploré plusieurs niveaux en profondeur, lorsqu'on sait qu'elles ne pourront pas améliorer le résultat final.

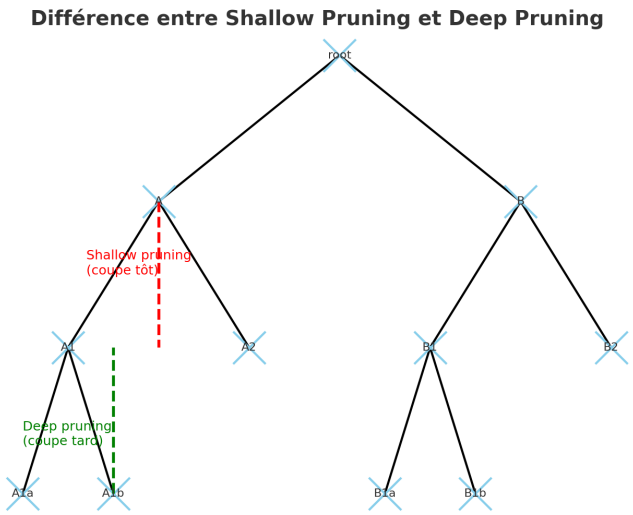

FIGURE 1 – Différence Entre Shallow Pruning Et Deep Pruning

Partie B : Contexte dans PontuXl

Le shallow pruning (ou **élagage superficiel**) est une optimisation utilisée dans les algorithmes de recherche comme **Minimax** ou **Alpha-Beta**.

Le principe consiste à **arrêter l'exploration de certaines branches de l'arbre de recherche très tôt**, sur la base de critères simples, **avant** d'avoir atteint la profondeur maximale fixée.

Dans PontuXL, cela signifie qu'on peut décider de **ne pas explorer un coup** si :

- Il conduit immédiatement à une position extrêmement défavorable (par exemple, un lutin isolé sans pont disponible).
- Il mène à une configuration identifiée comme "perdante rapide" grâce à une heuristique.
- Il ne modifie pas significativement l'évaluation actuelle de l'état du jeu (coup inutile ou équivalent à d'autres déjà explorés).

Avantages :

- Réduction importante du nombre de nœuds explorés.
- Accélération du temps de réponse de l'IA.

Inconvénient :

- Risque de rater des coups gagnants masqués par un mauvais score à court terme.

5.3.3 Heuristiques utilisées

Heuristique 1 – Mobilité des lutins Principe
Cette heuristique évalue **la liberté de mouvement** du joueur. Elle considère que plus un joueur peut déplacer ses lutins vers différentes cases, plus il a de flexibilité stratégique, plus il est difficile à enfermer, et plus il a de chances de survivre et d'attaquer.

En clair :

> "Plus mes pions peuvent bouger, mieux je me porte."

Formule
Soit P_J l'ensemble des pions du joueur J et $mov(p)$ le nombre de déplacements légaux possibles pour le pion p :

$$H_1(J) = \sum_{p \in P_J} mov(p)$$

Dans le code :

```
mobilite_lutins(Plateau, Ponts, Joueur, Score) :-
    trouver_lutins(Plateau, Joueur, Lutins),
    calculer_mobilite_totale(Lutins, Plateau, Ponts, 0, Score).
```

Avantages :

- **Simple et rapide** à calculer (pas besoin de simuler de nombreux coups).
- Reflète **directement** la capacité à jouer plusieurs coups futurs.
- Favorise les positions **flexibles** et **non bloquées**.
- Utile dans des jeux de blocage comme PontuXL où l'isolement est létal.

Inconvénients :

- **Myopie stratégique** : ne prend pas en compte la qualité des déplacements.

— **Pas de vision offensive** : ignore la situation adverse.

— Peut **gonfler artificiellement** si un pion se déplace dans un espace vide.

— Non normalisée : dépend du nombre de pions restants.

Heuristique 2 – Isolation des adversaires Principe
Cette heuristique mesure **à quel point les pions adverses sont bloqués**. Elle considère que plus l'adversaire a peu de mouvements, plus il est vulnérable à l'élimination.

En clair :

"Moins mon adversaire peut bouger, mieux je me porte."

Formule
Soit A_J l'ensemble des adversaires du joueur J, P_a l'ensemble des pions de l'adversaire a et $m_{\max}$ la mobilité maximale théorique d'un pion (ici 4 dans un plateau ouvert) :

$$H_2(J) = \sum_{a \in A_J} \sum_{p \in P_a} (m_{\max} - mov(p))$$

Dans le code :

```
isolation_adversaires(Plateau, Ponts, Joueur, Score) :-
    couleurs_joueurs(Couleurs),
    delete(Couleurs, Joueur, Adversaires),
    calculer_isolation_totale(Adversaires, Plateau, Ponts, 0, Score).
```

Avantages :

— **Oriente l'IA vers l'attaque** : cherche à bloquer et enfermer l'adversaire.

— Complète bien H1 en combinant confort et gêne adverse.

— Stratégie proactive qui augmente les chances d'élimination rapide.

— Cohérent avec la règle d'élimination (pion sans mouvement = joueur en danger).

Inconvénients :

— **Non normalisée** : le score augmente mécaniquement avec plus d'adversaires.

— Ne distingue pas les adversaires prioritaires.

— Suppose que $m_{\max} = 4$ est constant, ce qui peut être faux en plateau réduit.

— Risque de surévaluer l'attaque au détriment de la sécurité propre.

Résumé comparatif

Aspect	H1 : Mobilité propre	H2 : Isolation adverse
Objectif	Maximiser ses options	Réduire celles des autres
Formule	$\sum mov(p)$	$\sum(4 - mov(p))$ sur pions adverses
Avantage principal	Défensif : évite l'isolement	Offensif : force l'isolement adverse
Inconvénient principal	Ignore l'état adverse	Ignore sa propre vulnérabilité
Rôle	Flexibilité, survie	Pression, élimination

5.3.4 Comparaison des algorithmes et heuristiques

Contexte du test

- **Algos testés** :
  - *Max^n + shallow pruning* (spécial multi-joueurs)
  - *Minimax + α–β* (classique 2 joueurs, adapté ici « par défaut » au 4 joueurs)
- **Heuristiques** :
  - **H1** = mobilité des lutins (plus de coups possibles = mieux)
  - **H2** = isolation des adversaires (moins de coups pour eux = mieux)
- **Bench** : coups limités au nœud racine (cap = 30) + timeout 3 s/cas.

Résultats observés

```
Maxn+Shallow   H1   0.11 s
coup(deplacement(pos(4,0),pos(3,0)), retirer_pont(horizontal,3,0))

Maxn+Shallow   H2   0.20 s
coup(deplacement(pos(4,0),pos(3,0)), retirer_pont(horizontal,3,0))

Minimax a-b    H1   timeout

Minimax a-b    H2   timeout
```

Lecture rapide

- Max^n rend un coup dans le délai, identique pour H1 et H2.
- H2 $\approx 2\times$ plus lente que H1 (plus d'évaluations adverses).
- Minimax time-out dans ces conditions.

Comparaison des algorithmes

Max^n + shallow pruning Conçu pour le multi-joueurs, renvoie un vecteur d'évaluations, passe bien à l'échelle avec un pruning léger. Rapide (0.1–0.2 s) avec H1/H2.

Minimax + α–β Très efficace en 2 joueurs, mais l'élagage perd son efficacité en 4 joueurs $\Rightarrow$ timeouts.

Comparaison des heuristiques

Critère	**Heuristique 1 : Mobilité propre**	**Heuristique 2 : Isolation adverse**
Objectif principal	Maximiser la liberté de mouvement du joueur courant.	Réduire la liberté de mouvement des adversaires.
Orientation stratégique	**Défensive** : éviter d'être bloqué, maintenir plusieurs options.	**Offensive** : bloquer et piéger les adversaires pour provoquer leur élimination.
Formule	$H_1(J) = \sum_{p \in P_J} mov(p)$	$H_2(J) = \sum_{a \in A_J} \sum_{p \in P_a} (4 - mov(p))$
Comportement induit	Encourage l'IA à se déplacer vers des zones ouvertes et connectées.	Encourage l'IA à retirer ou faire pivoter des ponts pour réduire les déplacements adverses.
Avantages	— Simple et rapide à calculer. — Garantit une position flexible et adaptable. — Réduit le risque d'élimination par isolement.	— Favorise une pression constante sur l'adversaire. — Augmente les chances d'élimination rapide. — Complète bien la mobilité propre en ajoutant une dimension agressive.
Inconvénients	— N'attaque pas directement l'adversaire. — Peut mener à un jeu trop passif si utilisée seule. — Ne différencie pas la qualité des déplacements.	— Non normalisée (plus d'adversaires = score plus grand). — Peut pousser l'IA à attaquer même en danger. — Ne cible pas forcément l'adversaire le plus menaçant.
Adaptation au multi-joueurs	Fonctionne bien pour conserver ses chances face à plusieurs joueurs.	Doit être normalisée ou pondérée pour éviter un biais lié au nombre d'adversaires restants.
Impact sur l'IA	Produit un style de jeu prudent et positionnel.	Produit un style de jeu agressif et orienté blocage.
Risque	Jeu trop défensif → manque d'initiative.	Jeu trop agressif → risque de mauvaise posture.

Analyse comparative

— **Complémentarité** : H1 protège la survie, H2 affaiblit les adversaires ⇒ synergie.

— **Choix selon situation** : début ou danger → H1 ; adversaire fragilisé → H2 ; pondérer selon état de partie.

— En multi-joueurs : H1 stable ; H2 à normaliser (moyenne par adversaire).

Recommandations

1. Par défaut (4 joueurs) : Max^n + shallow avec H1.
2. Mixer : $Score = w_1 H_1 + w_2 H_2$ avec $w_1 > w_2$ en early game, équilibrer ensuite.
3. Si Minimax : profondeur 1–2, cap 20, timeout $\geq 5\,\text{s}$.

Conclusion Max^n + shallow surpasse Minimax ici. H1 est rapide, H2 plus tactique mais plus lente. La combinaison pondérée H1+H2 donne une évaluation plus complète.

5.4 Interface JavaScript-Prolog (`prolog_interface.js`)

Ce fichier assure la communication entre le code JavaScript du jeu et l'IA implémentée en Prolog. Il utilise la bibliothèque Tau Prolog pour exécuter le code Prolog directement dans le navigateur.

Le processus d'intégration fonctionne comme suit :

1. Le code Prolog est chargé dans une session Tau Prolog.
2. L'état du jeu JavaScript est converti en format Prolog.
3. Une requête Prolog est exécutée pour trouver le meilleur coup.
4. Le résultat est converti en format JavaScript et utilisé par le jeu.

Bot Conversationnel (`pbot-elm.pl`, `bot.js`). Le bot conversationnel, nommé PBot, est implémenté en Prolog et permet aux joueurs de poser des questions sur les règles du jeu, les stratégies, etc. Il utilise un système de reconnaissance de mots-clés et de motifs pour comprendre les questions des utilisateurs.

Le fichier `bot.js` sert d'interface entre le chat HTML et le code Prolog du bot, permettant une interaction fluide avec l'utilisateur.

6 Processus de Développement

Le développement du projet s'est déroulé en plusieurs phases :

Phase de conception : Définition des règles du jeu, de l'architecture du projet et des technologies à utiliser

Implémentation de base : Création de l'interface utilisateur et de la logique de jeu en JavaScript

Développement de l'IA : Implémentation de l'algorithme Max^n en Prolog

Intégration : Création de l'interface JavaScript-Prolog pour permettre la communication entre les deux langages

Développement du bot : Création du bot conversationnel en Prolog

Tests et optimisations : Correction des bugs, amélioration des performances et de l'expérience utilisateur

Finalisation : Documentation et préparation du livrable final

7 Défis Rencontrés et Solutions

1. **Intégration JavaScript-Prolog.**
   Défi : Faire communiquer efficacement le code JavaScript du jeu avec l'IA implémentée en Prolog.
   Solution : Utilisation de la bibliothèque Tau Prolog qui permet d'exécuter du code Prolog directement dans le navigateur. Création d'une interface dédiée (`prolog_interface.js`) pour convertir les données entre les deux formats.
2. **Algorithme Max^n pour Jeux Multi-joueurs.**
   Défi : Adapter l'algorithme Minimax traditionnel pour un jeu à quatre joueurs.
   Solution : Implémentation de l'algorithme Max^n avec élagage superficiel, qui évalue un n-uplet de scores (un pour chaque joueur) et permet à chaque joueur de maximiser son propre score.
3. **Feedback Visuel pour les Actions de l'IA.**
   Défi : Les joueurs avaient du mal à suivre les actions de l'IA, notamment lors de la suppression des ponts.
   Solution : Implémentation d'effets visuels spécifiques (animation de pulsation, message textuel, bordure rouge) pour rendre les actions de l'IA plus visibles et compréhensibles.
4. **Performance de l'IA.**
   Défi : L'algorithme Max^n peut être très coûteux en termes de calcul, surtout avec une profondeur de recherche élevée.
   Solution : Implémentation de l'élagage superficiel (Shallow Pruning) pour réduire l'espace de recherche et optimisation des fonctions d'évaluation pour améliorer la prise de décision.

8 Fonctionnalités Principales

Jeu à Quatre Joueurs : Deux joueurs humains (vert et jaune) et deux IA (bleu et rouge)

IA Avancée : Utilisation de l'algorithme Max^n avec élagage superficiel

Interface Intuitive : Plateau de jeu interactif avec animations et feedback visuel

Bot Conversationnel : Assistant pour aider les joueurs à comprendre les règles et stratégies

Effets Visuels : Animations pour rendre les actions plus compréhensibles

9 Comment Exécuter le Projet

1. Cloner le dépôt ou télécharger les fichiers du projet.
2. Ouvrir un terminal dans le dossier du projet.
3. Lancer un serveur web local : `python3 -m http.server 9000`
4. Ouvrir un navigateur et accéder à `http://localhost:9000`.

10 Effets Visuels et Feedback Utilisateur

Un aspect particulier du projet a été l'attention portée aux effets visuels pour améliorer l'expérience utilisateur, notamment lors des actions de l'IA. Ces effets comprennent :

Animation de pulsation en rouge : Les ponts qui vont être retirés pulsent en rouge avant de disparaître, attirant l'attention du joueur.

Message textuel "Pont supprimé" : Un message apparaît au-dessus du pont pour indiquer clairement l'action en cours.

Bordure rouge : Une bordure rouge est ajoutée autour du pont pour le rendre plus visible.

Mise à jour garantie de l'interface : Après chaque action, l'interface est mise à jour pour refléter l'état actuel du jeu.

Ces améliorations visuelles ont résolu un problème important où l'IA indiquait qu'elle supprimait un pont à un endroit mais en supprimait un autre. Grâce à ces effets, les actions de l'IA sont maintenant parfaitement claires pour les joueurs humains.

11 Conclusion

Le projet PontuXL constitue une implémentation réussie et enrichie du jeu de société Pontu, intégrant une intelligence artificielle avancée en `Prolog`. L'utilisation de l'algorithme Max^n avec élagage superficiel (*shallow pruning*) s'avère particulièrement pertinente pour la nature multi-joueurs du jeu, permettant une prise de décision stratégique équilibrée. Les deux heuristiques développées offrent une complémentarité entre approche défensive (mobilité propre) et offensive (isolation adverse), renforçant la capacité d'adaptation de l'IA.

L'intégration entre JavaScript et Prolog, rendue possible grâce à la bibliothèque `Tau Prolog`, a permis l'exécution directe du code Prolog dans le navigateur, malgré la complexité technique de ce couplage. Des effets visuels soignés et un retour utilisateur clair contribuent à une expérience de jeu plus immersive, tandis que l'ajout d'un bot conversationnel apporte une dimension pédagogique en guidant les joueurs dans la compréhension des règles et stratégies.

Enfin, l'approche méthodique adoptée, incluant des tests de performance, a permis de valider empiriquement les choix algorithmiques et d'optimiser le comportement de l'IA selon les contraintes du jeu. Ce projet illustre non seulement la

pertinence de `Prolog` pour l'implémentation d'IA complexes, mais aussi la réussite de son intégration avec des technologies web modernes.

12 Perspectives d'Amélioration

Optimisation de l'IA : Augmenter la profondeur de recherche et améliorer les fonctions d'évaluation

Mode Multijoueur en Ligne : Permettre à plusieurs joueurs de jouer ensemble via Internet

IA Adaptative : Développer une IA qui apprend des parties précédentes

Personnalisation : Ajouter des options de personnalisation (thèmes, niveaux de difficulté)

13 Équipe de Développement

Cette version du projet a été développé dans le cadre du cours INFO B317 - IA et programmation symbolique par les 04 étudiants mentionnés plus haut.