9 avril 2021

RAPPORT PROJET PROLOG

[Intelligence Artificielle Symbolique]

GADEAU Juliette - GUERIN Élisa - PERRIER ALBAN - THOMAS Énora - THOMASSON Lucie

BORDEAUX INP

ENSC 2A – Groupe 1

SOMMAIRE

INTRODUCTION

Ce rapport a pour but de fournir un compte-rendu du projet de programmation en IA symbolique. L'objectif de ce projet était de mettre en application nos connaissances acquises dans le cadre du module enseigné au semestre 8 à l'ENSC. Nous avons créé un jeu du Quoridor. Les règles de ce jeu seront rappelées dans la suite de ce rapport.
Codé en Prolog, le but principal était d'énoncer des prédicats pour représenter les règles du jeu afin de déplacer (ou non) les pions des joueurs.

MODE D'EMPLOI

En bref

Quoridor est un jeu de stratégie dont le but est d'avancer votre pion d'une extrémité à l'extrémité opposée du plateau. À votre tour, vous pouvez déplacer votre pion ou placer une barrière. Vous pouvez gêner les mouvements de votre adversaire lors du placement de la barrière, sans toutefois le bloquer complètement. En attendant, il essaie de faire la même chose pour vous. Le premier pion à atteindre l'extrémité adverse du plateau gagne.

Lancement du jeu

Pour lancer le jeu, il faut d'abord télécharger le dossier, disponible sur ce lien : https://github.com/aperrier004/PROLOG, puis ouvrir le fichier *projetProlog.pl* dans SWI-Prolog. Cela devrait ouvrir une fenêtre de jeu et vous pourrez commencer à jouer !
Une fois la partie terminée, pour recommencer, il faudra fermer toutes les fenêtres (incluant SWI-Prolog) et relancer *projetProlog.pl*.

EXEMPLE D'UTILISATION

Au lancement du programme, une interface graphique s'affiche avec le plateau de jeu (le quadrillage où s'affichent les pions des joueurs et les barrières), ainsi que les différentes variables permettant de déplacer son pion ou placer une barrière.
À chaque tour, une fenêtre s'ouvre pour vous dire qui doit jouer et combien de barrières le joueur peut placer sur le plateau.

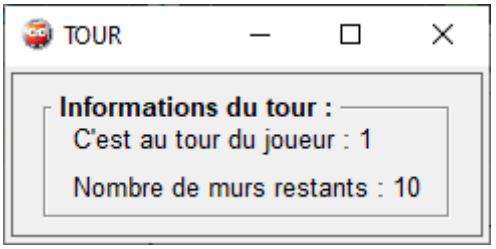

***Figure 1** : Fenêtre des informations du joueur*

Lors d'un tour, vous pouvez choisir entre deux actions :

1. Déplacer votre pion : pour cela, vous devez entrer les coordonnées sur lesquelles vous souhaitez bouger le pion. Il vous faudra spécifier le numéro de ligne et de colonne de la case d'arrivée.
   A savoir qu'il est uniquement possible de déplacer son pion d'une case à l'horizontale ou à la verticale. Il est impossible de le déplacer en diagonale et de passer par dessus une barrière.
   Dans le cas où vous voudriez déplacer votre pion sur une case adjacente, mais que celle-ci est occupée par le pion adverse, il est possible de lui "sauter" par-dessus pour placer votre pion derrière. Cela revient à avancer de deux cases.

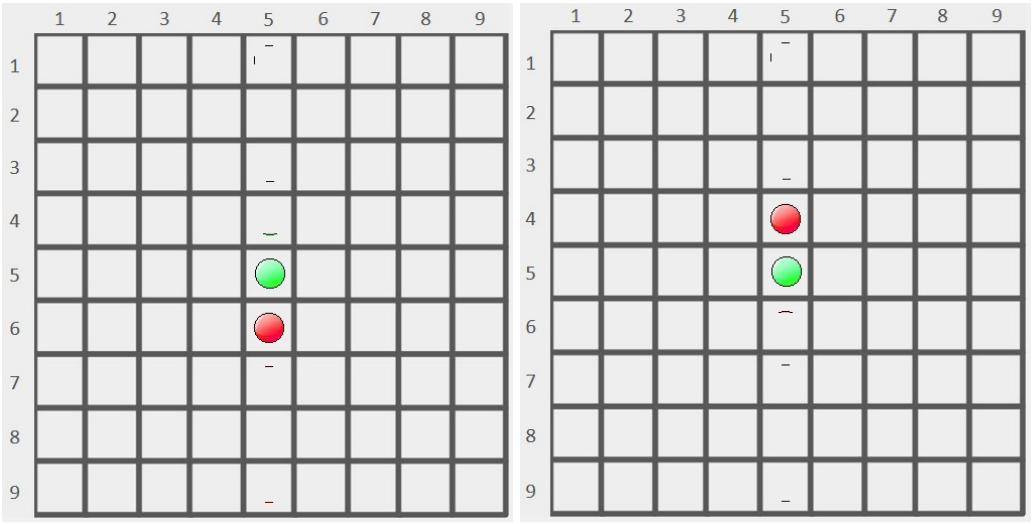

***Figure 2** : Exemple de situation "saute-mouton' par le pion rouge*

2. Placer une barrière: vous devez choisir entre une barrière horizontale ou verticale, puis saisir les coordonnées de l'endroit où vous souhaitez placer la barrière. La barrière occupera deux cases de long. Un exemple est proposé juste dessous.
   A savoir qu'il n'est pas possible de superposer les barrières. Celles-ci faisant 2 cases de long, vous ne pouvez pas les faire se croiser.
   De plus, il n'est pas possible de placer une barrière qui empêcherait complètement l'adversaire d'atteindre son objectif. Les barrières placées doivent toujours laisser à l'adversaire la possibilité d'atteindre l'extrémité adverse du plateau (cette fonctionnalité n'a pas pu être implémentée, nous comptons sur le fair play des joueurs pour respecter cette règle).

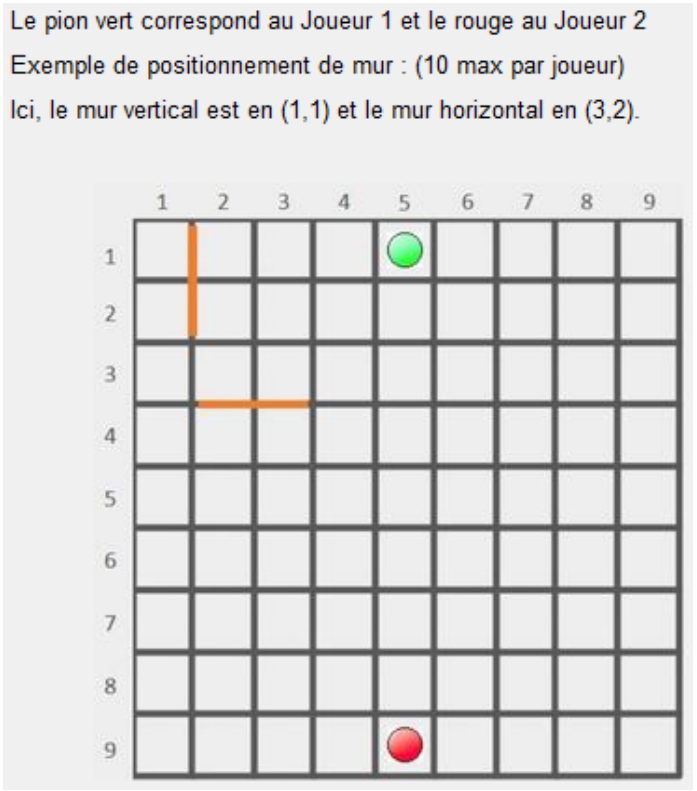

Figure 3 *: Plateau de jeu avec placement de barrières*

Dans le cas où l'une des actions demandée par le joueur au moment de cliquer sur Valider n'est pas valable, une pop-up s'ouvrira pour lui dire de corriger sa saisie.

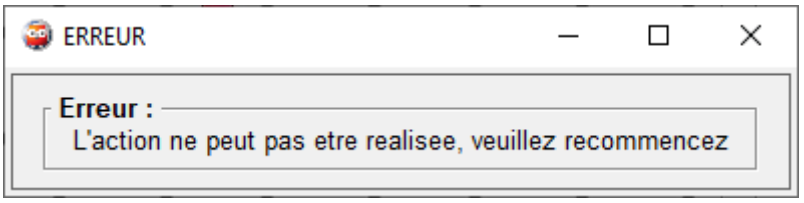

Figure 4 *: Fenêtre d'erreur si le joueur veut réaliser une action impossible*

CHOIX TECHNIQUES

RÈGLES DU JEU

Choix et justifications techniques

Pour coder les règles du jeu, nous sommes partis du principe que l'état du jeu était mémorisé au niveau de l'interface. Ainsi, nous récupérons cet état et renvoyons à l'interface un booléen leur indiquant si ce que le joueur veut faire respecte les règles ou non, ou encore si un des joueurs a gagné et si la partie est terminée.

Nous avions donc besoin de récupérer :

- Les positions des 2 joueurs sous forme de liste de 2 éléments pour chaque adversaire (la coordonnée X et la coordonnée Y de chaque pion)
- Les listes contenant les positions des barrières horizontales et verticales
- Le type d'action que le joueur veut effectuer (déplacer un pion, placer une barrière verticale ou horizontale)
- La position voulue pour l'élément manipulé selon l'action choisie.

Choix des prédicats

Pour mener à bien cette mission, nous avons décidé de découper le code en plusieurs parties selon les actions possibles. Dans chaque partie, nous avons écrit plusieurs prédicats pour rendre le fonctionnement de notre code plus lisible.

Validation d'un coup

Nom du prédicat	Paramètres	Description
tour /6	Position1, Position2, BarrieresH, BarrieresV, TypeAction, NewPosition	Renvoie true si l'action demandée est réalisable grâce à l'appel aux prédicats *accessiblePion*, *accessibleBarrieresH*, et *accessibleBarrieresV* en fonction de la valeur de TypeAction
accessiblePion /5	[X1,Y1], [X2,Y2], BarrieresH, BarrieresV, [Xadv,Yadv]	Renvoie true si le déplacement du pion de (X1,Y1) à (X2,Y2) est possible en appelant d'autres prédicats décrits plus bas en fonction du déplacement souhaité
accessibleBarrieresH /3	PositionVoulue, BarrieresH, BarrieresV	Renvoie true si on peut placer une barrière horizontale à la PositionVoulue grâce à l'appel à d'autres prédicats décrits plus bas
accessibleBarrieresV	PositionVoulue, BarrieresH, BarrieresV	Renvoie true si on peut placer une barrière verticale à la PositionVoulue grâce à l'appel à d'autres prédicats décrits plus bas

Déplacement normal d'un pion

Nous allons ici prendre l'exemple d'un déplacement vertical normal du pion.

Nom du prédicat	Paramètres	Description
memePosition /2	[X1, Y1], [X2, Y2]	Renvoie true si la position (X1,Y1) est confondue avec (X2,Y2)
validePositionPion /2	[X, Y], [Xadv, Yadv]	Renvoie true si la position (X,Y) demandée par le joueur est sur le plateau et si elle n'est pas confondue avec celle (Xadv,Yadv) de l'adversaire (on fait appel à *memePosition* pour déterminer cela)
deplacementVertical /2	[X1,Y1], [X2,Y2]	Renvoie true si le déplacement de (X1,Y1) à (X2,Y2) correspond à un déplacement vertical sur une case adjacente
alignementHorizontalBarriere Horizontale /2	BY, Y	Renvoie true si la barrière horizontale est alignée horizontalement avec le pion pour lui bloquer le passage (BY étant le numéro de la colonne de la barrière et Y celui du pion)
alignementVerticalBarriereHo rizontale /3	BX, X1, X2	Renvoie true si la barrière horizontale est alignée verticalement avec le pion pour lui bloquer le passage (BX étant le numéro de la ligne de la barrière, X1 celui du pion, X2 celui de la case d'arrivée)
accessiblePionBarrieresH /4	PositionActuelle, PositionVoulue, BarrieresH, nbBarrièresBloquant	Renvoie true si la position voulue est accessible et qu'aucune barrière ne bloque le passage c'est une fonction récursive qui va parcourir la liste des barrières et appeler *alignementHorizontalBarriereHorizontale* et *alignementVerticalBarriereHorizontale*

Ici, nous nous intéressons uniquement aux barrières horizontales. C'est simplement car lors d'un déplacement vertical, les barrières verticales ne nous posent pas de problème.

Ainsi pour valider le déplacement, le prédicat *accessiblePion* décrit plus haut fera appel à *validePositionPion*, *deplacementVertical*, et *accessiblePionBarrieresH*.

Pour un déplacement horizontal, nous avons créé des prédicats similaires, mais dont les axes sont inversés.

Déplacement “saute-mouton” d’un pion

Nous allons ici prendre l’exemple d’un déplacement vertical par-dessus le pion de l’adversaire.

Nom du prédicat	Paramètres	Description
deplacementMoutonVertical /3	[X1,Y1], [X2,Y2], [Xadv,Yadv]	Renvoie true si le déplacement voulu correspond bien à un “saute-mouton” vertical
accessibleMoutonVertical /4	Position1, Position2, BarrieresH, PositionAdv	Renvoie true si aucune barrière ne bloque le passage pour ce déplacement grâce à l’appel à *accessiblePionBarrieresH* décrit dans la partie précédente

Pour valider ce déplacement, le prédicat *accessiblePion* décrit plus haut fera appel à *validePositionPion*, et *accessibleMoutonVertical*.

Pour un déplacement du même type mais horizontal, nous avons écrit des prédicats similaires que nous ne redétaillerons pas.

Placement d'une barrière

Nous allons ici prendre l'exemple d'un placement d'une barrière horizontale.

Nom du prédicat	Paramètres	Description
validePositionBarriere /1	[X,Y]	Renvoie true si la position (X,Y) est une position valide sur le plateau pour poser une barrière
alignementBHbloqueBH /2	[BX,BY], [X,Y]	Renvoie true si la position (BX,BY) d'une barrière horizontale existante empêche le placement d'un nouvelle barrière horizontale à la position (X,Y)
barriereHbloquantBarriereH /2	PositionVoulue, BarrieresH, Acc	Renvoie true s'il existe une barrière horizontale qui bloque le placement d'une nouvelle barrière horizontale à la PositionVoulue ce prédicat fonctionne récursivement en parcourant la liste des barrières existantes et en appelant *alignementBHbloqueBH* (Acc est un accumulateur qui compte le nombre de barrières qui posent problème)
alignementBVbloqueBH /2	[BX,BY], [X,Y]	A la même fonction que *alignementBHbloqueBH*, mais pour vérifier si une barrière verticale bloque le placement de la nouvelle barrière horizontale
barriereVbloquantBarriereH /3	PositionVoulue, BarrieresV, Acc	A la même fonction que *barriereHbloquantBarriereH*, mais pour vérifier s'il existe une barrière verticale qui empêche le placement de la nouvelle barrière horizontale.

Pour valider le placement de la barrière, le prédicat *accessibleBarrieresH* décrit plus haut fera appel à *validePositionBarriere*, *barriereHbloquantBarriereH,* et *barriereVbloquantBarriereH*.

De la même façon que précédemment, nous avons écrit des prédicats similaires pour le placement d'une barrière verticale.

Fin de jeu

Finalement pour que la partie interface sache quand le jeu est terminé, nous avons écrit un prédicat qui renvoie true si c'est le cas.

Nom du prédicat	Paramètres
jeuTermine /2	PositionJ1, PositionJ2

INTERFACE GRAPHIQUE

Choix et justifications techniques

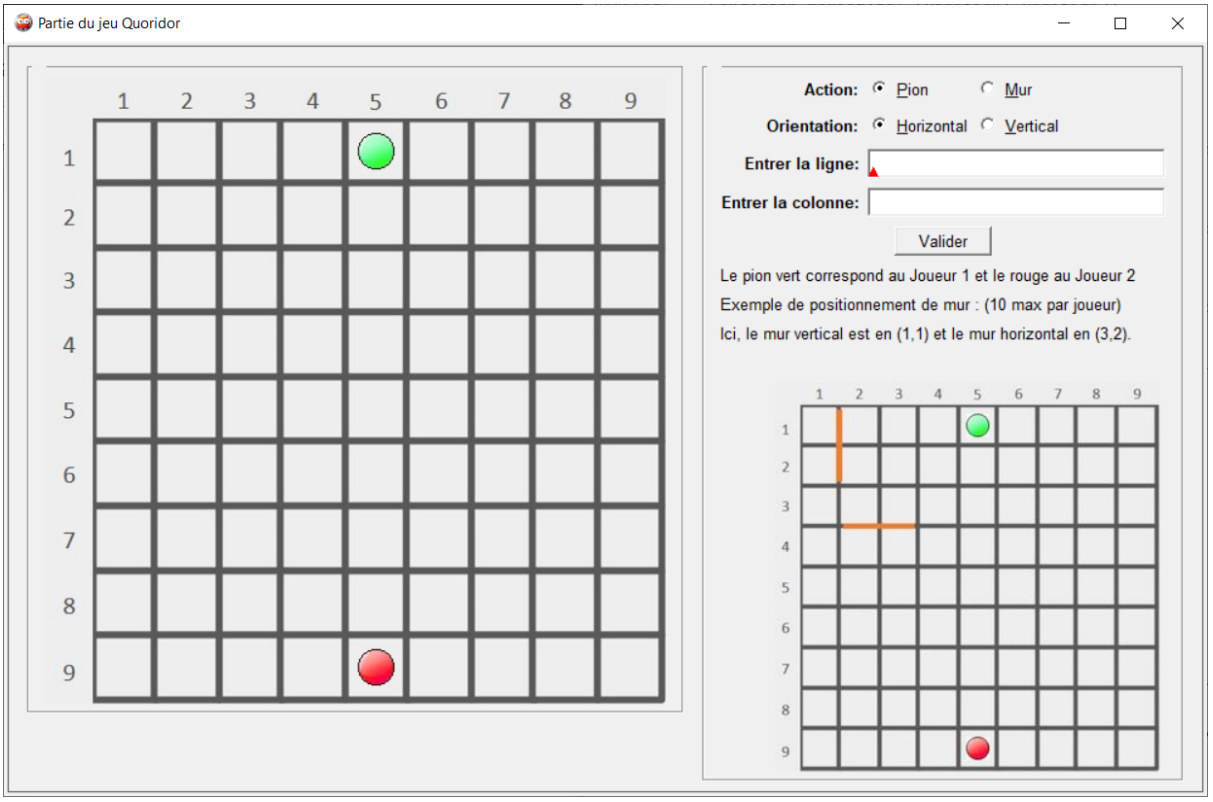

***Figure 5** : Aperçu de l'interface graphique*

Concernant l'interface graphique, nous avons choisi d'utiliser la **bibliothèque XPCE**. Dans une nouvelle fenêtre, il nous est alors possible d'**afficher** la grille, les pions, les barrières ainsi que de **récupérer les informations** entrées par l'utilisateur pour jouer.

Pour la partie **affichage**, nous avons décidé d'envoyer dans la fenêtre des **images** correspondant à la grille, aux pions et aux barrières. En effet, si nous avions décidé de créer chaque case de la grille à l'aide de fonctions de dessin de XPCE, il nous aurait fallu réaliser des prédicats de dessins pour optimiser le nombre de lignes de code, contre un seul pour afficher une image de grille.

Connaissant la taille de l'image de la grille et des cases, il est facile de placer les pions selon les coordonnées choisies par un simple calcul :

marge de la grille + (coordonnée -1) * (demi taille de la case)

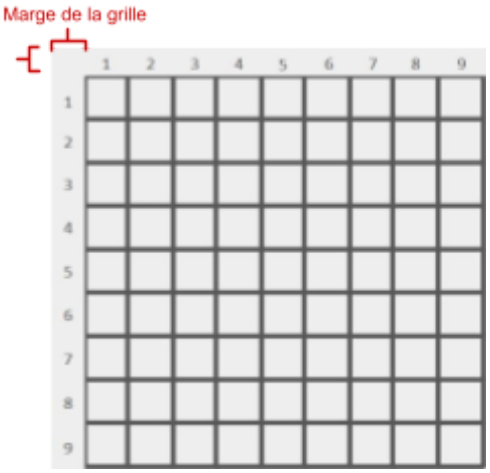

Figure 6 *: La grille vide*

Pour la **récupération** des actions que le joueur veut effectuer, nous utilisons des radio boutons pour récupérer si le joueur souhaite placer un pion ou une barrière, ainsi que pour l'orientation de la barrière. Le joueur entre les coordonnées en X et Y à la main pour placer son pion ou sa barrière. Dans le cas d'une barrière, les coordonnées entrées correspondent à la première case où sera placée la barrière (une barrière ayant la taille de deux cases).

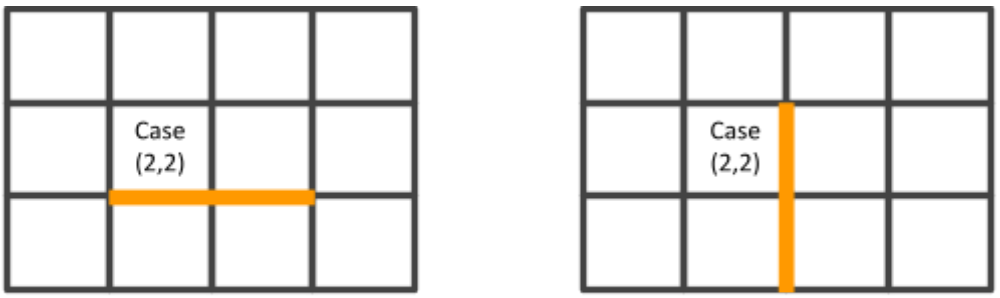

Figure 7 : Exemple de placement de barrières

Choix des prédicats

Prédicats dynamiques

Nom du prédicat	Description
image/4	Envoie une image dans une fenêtre à partir de son chemin et sa position
affichageAction/6	Permet l'affichage de l'action (pion ou barrière) en utilisant les prédicats correspondant (deplacerPion, afficherBarriereHorizontal ou afficherBarriereVertical)
affichageJoueur/1	Affiche l'action du joueur grâce aux prédicats deplacerPion, afficherBarriereHorizontal ou afficherBarriereVertical
deplacerPion/4	Place le pion selon les coordonnées entrées par le joueur et efface le pion précédent
joueurActuel/1	Renvoie True si le numéro du joueur actuel est celui entré en paramètre
nbBarriere/2	Garde en mémoire le nombre de barrières/barrières qu'un joueur a déjà posé
coordonnees/2	Garde en mémoire les coordonnées actuelles des pions des joueurs, ainsi que de toutes les barrières horizontales et verticales existantes

Prédicats non dynamiques

Nom du prédicat	Description
init/0	Initialise la partie, ouvre la fenêtre d'interface avec ses différents éléments constitutifs
libérer/0	Permet de libérer les ressources
afficherFinDeTour/0	Permet d'afficher une petite fenêtre pour indiquer quel joueur doit jouer, ainsi que son nombre de barrières restants à poser
affichageErreur/0	Permet l'affichage d'un message d'erreur si le joueur veut réaliser une action impossible (par exemple lorsqu'il entre des coordonnées qui correspondent à un déplacement impossible)
afficherBarriereHorizontale/4	Envoie une barrière horizontale dans la fenêtre selon les coordonnées entrées par le joueur
afficherBarriereVerticae/4	Envoie une barrière verticale dans la fenêtre selon les coordonnées entrées par le joueur
submit/6	Prédicat appelé à chaque fois que l'on appuie sur le bouton valider, il permet de vérifier si la partie est terminée, vérifie si les coordonnées permettent un déplacement possible, met à jour les différents coordonnées des pions et murs et affiche l'action si elle est possible ainsi que la fin de tour et de partie
afficherFinDePartie/0	Affiche une petite fenêtre pour signaler que la partie est terminée

BILAN DES FONCTIONNALITÉS

Les fonctionnalités que **nous avons pu développer** sont les suivantes :

- Déplacer un pion sur le plateau
  - → Uniquement à l'horizontale ou à la verticale
  - → Uniquement dans la zone du plateau
  - → Possibilité de "sauter" par-dessus un pion situé sur une case adjacente théoriquement atteignable
  - → Impossibilité de déplacer un pion sur une case adjacente séparée par une barrière

- Placer une barrière sur le plateau
  - → La barrière peut être verticale ou horizontale
  - → Partout sauf sur la périphérie du plateau
  - → Impossibilité de superposer tout ou partie d'une barrière sur une autre
  - → Impossibilité de croiser une barrière verticale et une barrière horizontale l'un sur l'autre

- Décompte du nombre de barrières pouvant être placées en fonction du nombre de barrières déjà placés sur le plateau

- Déterminer quand le jeu est fini, c'est-à-dire quand un des pions est arrivé à l'extrémité adverse du plateau

Les fonctionnalités que **nous n'avons pas pu développer** sont les suivantes :

- Empêcher un joueur de poser une barrière qui bloquera complètement le passage de l'adversaire, mais nous avions quelques pistes :
    - Créer des prédicats pour savoir sur une barrière en touche une autre ou une extrémité du plateau
    - Un prédicat qui déterminerait de façon récursive s'il existe un "chemin" de barrières d'une extrémité à une autre car cela impliquerait qu'un espace clos à été créé sur le plateau

Cependant, cela n'a pas abouti, car d'une part le prédicat récursif écrit avait une complexité temporelle trop élevée, et d'autre part cela ne prenais pas en compte tous les cas possibles (un pion pourrait être entouré de barrières sans qu'aucune d'entre elle touche un mur)

- Pouvoir faire jouer un ordinateur

GESTION DE PROJET

Notre équipe étant composée de cinq étudiants, nous avons fait le choix de créer **deux équipes** et de nous répartir en fonction des tâches à effectuer : une équipe de deux s'est consacrée à l'**interface du jeu** et une équipe de trois s'est consacrée aux **règles du jeu**.

Date	Equipe	Tâches
22/02 (TP)	Alban & Juliette	Recherche sur XPCE et premiers affichages
	Élisa, Énora, Lucie	Réflexion sur l'approche à utiliser
08/03 (TP)	Alban & Juliette	Maquette de l'interface, détermination des prédicats à écrire et affichage de la grille du plateau
	Élisa, Énora, Lucie	Détermination des prédicats à écrire
15/03 (TP)	Alban & Juliette	Utilisation de DialogGroup, Text Input, Label et Radio bouton pour récupérer les informations du joueur
	Élisa, Énora, Lucie	Définition des fonctions restantes pour déplacer les pions et les barrières (+ vérifier si ce déplacement est possible), débuggage de tests d'égalité
17/03 (2H)	Alban & Juliette	Amélioration de la grille et essais infructueux pour des affichages dynamiques sur l'interface (au clic d'un radio button par exemple)
	Élisa, Énora, Lucie	Réécriture des règles avec la correction de M.Favier
31/03 (TP)	Alban & Juliette	Déplacement et affichage d'un pion et de barrières
	Élisa, Énora, Lucie	Placement des barrières horizontales et verticales Vérification qu'aucune barrière ne gène un pion, et si aucun pion ne gène un autre pion ou une barrière, et faisabilité d'un déplacement Saut d'un pion par dessus le pion adverse
03/04 (4H)	Alban & Énora	Fusion du code des règles et de l'interface graphique et amélioration de l'expérience utilisateur

BILAN PERSONNEL

Ce fut un projet assez exotique, parce qu'il nous a forcé à prendre un contre pied de la logique de programmation que l'on utilise d'habitude. La mise en route a été laborieuse et la première moitié du projet frustrante, parce qu'on a dû pour commencer, appréhender cette nouvelle façon de réfléchir, et ensuite parce que les documentations sur la librairie XPCE et sur Prolog en général, sont terribles, et souvent insuffisantes. Pour compenser, on pourrait se dire que le travail de la communauté de développeurs pourrait suffir, mais elle est si faible que les réponses sur StackOverflow dont on peut se servir sont assez limitées. Toutefois, nous avons pu nous appuyer sur quelques projets de jeu en prolog pour développer l'interface graphique. Il était donc en somme frustrant de ne pas réussir à obtenir des résultats rapidement tel qu'on l'imagine quand, sur un autre langage de programmation, nous aurions pu réaliser la même chose en moins de temps et d'efforts.
Heureusement, nous avons divisé le groupe en deux équipes au fonctionnement interne et inter-groupe très bon. Nous avons donc réussi à avancer régulièrement durant toute la durée du projet. La gestion de projet et la division des tâches ont été très satisfaisantes, nous n'avons pas perdu du temps et lorsque nous avons dû réunir nos deux parties, cela a bien fonctionné sans perdre de temps, puisque nous nous étions coordonnés sur les paramètres à donner à nos prédicats (listes de coordonnées etc…).
En somme, les trois séances de TP nous ont permis de bien travailler sur le projet, bien qu'afin d'atteindre complètement nos objectifs, nous avons seulement dû travailler 6 heures en plus, notamment pour obtenir une interface de jeu plus agréable.
Dans tous les cas, ce projet est original dans son contenu, et vraiment différent des projets informatiques que nous avons déjà eu l'occasion de faire.