Rapport Tournoi Escampe

# Partie 1 : Conception théorique

1. Pour la représentation du plateau et des pions, nous allons utiliser des mots binaires.

Pour le plateau, chaque case sera un mot de 2 bits avec *tt* où

t => type de case (01 simple, 10 double, 11 triple, 00 non initialisée)

Et pour les pions nous utiliserons pour chaque case un mot de 3 bits *cpo*

c => 1 si blanc, 0 si noir

p => 1 si licorne, 0 si paladin

o => 1 si la case est occupée 0 sinon

Cette représentation permet d’utiliser des opérations binaires peu coûteuses et rapides.

On peut également faire des modifications du plateau de jeu rapidement et globales.

Le principal défaut, c’est la complexité de mise en œuvre et la partie débogage qui devient plus compliquée qu’un format « human-readable ».

1. Le jeu se finit lorsque la licorne de l’adversaire est capturée, nous pouvons donc stopper le jeu à ce moment.   
   Pour une partie « déjà en cours », on peut simplement regarder si un joueur n’a plus sa licorne, auquel cas il a perdu la partie. (Cette dernière vérification est extrêmement simple à regarder avec notre représentation binaire, il suffit d’appliquer un masque binaire sur le plateau).
2. Les principales sources de difficultés sont :
   1. Les règles imposent un lien direct entre les actions de l’adversaire et ses coups possibles.
   2. La position initiale des pièces n’est pas connue à l’avance.
   3. Les déplacements liés au liseré peuvent compliquer l’analyse des plateaux.

Le facteur de branchement maximal est dans le cas où tous les pions du joueur sont placés sur des cases de liseré triple. Dans ce cas nous avons donc 6 pions (5 paladins et une licorne), qui peuvent se déplacer chacun sur 21 cases théoriques ou ne rien faire. Cela fait un total de 126 coups possibles, cependant une règle va venir réduire ce chiffre. Une grosse proportion de coups seront interdits car les cases accessibles seront déjà occupées par un pion allié ou adverse.

1. La complexité des règles et le fait de pouvoir impacter le jeu de l’adversaire pendant son tour rendent quasiment impossible de trouver une série de coups imparables. Néanmoins, il doit exister des séries permettant d’atteindre des positions avantageuses, mais seulement si les mouvements de l’adversaire suivent ce qu’on a prévu.
2. Les 3 critères principaux que nous envisageons d’inclure dans nos heuristiques sont :
   1. La mobilité des pions : avoir plus de possibilités de déplacements permet de s’adapter plus facilement aux situations complexes dans lesquelles l’adversaire peut nous diriger.
   2. La sécurité des licornes : si une solution expose la licorne, cela signifie qu’on se met en danger.
   3. Le contrôle des parties importantes du plateau : ces parties peuvent être le centre de plateau, qui offre une meilleure mobilité, ou encore certaines cases stratégiques.

De manière générale, contrôler le plateau et prendre l’ascendant de possibilité permet de s’assurer des stratégies efficaces. Néanmoins, beaucoup d’autres heuristiques peuvent être pris en compte, tels que la densité des troupes ou encore la distance avec l’adversaire.

1. Globalement la stratégie de jeu dépend énormément de la stratégie adverse. Néanmoins, un rythme général peut en ressortir incitant à d’abord prendre des positions tactiques en menaçant la licorne adverse puis commencer l’offensive jusqu’à la capture de la licorne adverse.
2. Nous comptons faire une architecture utilisant « l’Iterative deepening » afin de contrôler le temps de réflexion, car la recherche va pouvoir aller aussi loin que possible dans le temps limité.   
     
   Nous aimerions ajouter également ajouter un maximum de possibilités de coupe. Pour cela, une « aspiration window » avec des limites dynamiques permettrait d’adapter le ratio entre précision et profondeur d’arbre parcouru. Ça sera particulièrement utile en fin de partie afin de pouvoir aller loin dans l’arbre et trouver le moyen de capturer la licorne adverse. De manière plus générale, si nous pouvons couper les branches plus vite, nous aurons la possibilité d’aller voir beaucoup plus loin et donc d’avoir de meilleures décisions.  
     
   Nous souhaiterions ajouter une « refutation table », qui permet de ne pas recalculer les embranchements similaires à ce qu’on a déjà calculé. Si deux suites de mouvements amènent à la même situation, alors nous pouvons faire le calcul qu’une seule fois.

Afin d’optimiser les performances, il pourra être intéressant de calculer les heuristiques en combinant du cache des précédents calculs et du multithreading.