Phantom of the opera
AI

version 1.2

Philippe BOUTTEREUX

1 - L’algorithme

Le but de l’algorithme minima est de générer un arbre de décision incluant tous les mouvements possible avec une profondeur donnée. On peut ensuite sélectionner le chemin nous apportant le meilleur gain (ou la plus petite perte). La difficulté de cet algorithme est de simuler les mouvements de l’adversaire pour sélectionner les nôtres en fonction.

L’algorithme minimax est un des plus efficaces pour prendre des décisions dans ce genre de jeux pour plusieurs raisons.
Le fait que les personnages jouables sont sélectionnés aléatoirement au début de chaque tours ainsi que la complexité des pouvoirs de certains personnages rendent l’implémentation pure d’une stratégie très compliquée.
C’est pourquoi l’algorithme minimax n’est pas seulement efficace, il est aussi plus simple d’utilisation. De plus, il est facile d’insérer de la logique dedans pour augmenter encore les performances.

Pourtant, l’algorithme minimax a aussi des défauts:

- Sachant que les personnages jouables sont tirés aléatoirement chaque tour, il est presque impossible de prédire au-delà d’un tour (2 sets, 8 movements).

- La complexité est un facteur important dans un jeu avec autant de possibilités. En effet, voici le nombre moyen de noeuds composant un arbre incluant un set entier (4 movements).

  $$\prod_{n=1}^{4} (6.2 \cdot n) = 24.8 \cdot 18.6 \cdot 6.2 \cdot 12.4 = 35463.20$$

  Dans cette expression, 6.2 est le nombre moyen de mouvements possible pour un personnage (déplacements + pouvoirs). Au début d’un set, on a quatre personnages jouable. A chaque récursion, on en retire un (parce qu’un personnage ne peut pas jouer deux fois en un tour). Générer un arbre si grand demande un certain nombre de ressources et tenter de voir plus loin en demanderait encore beaucoup plus.

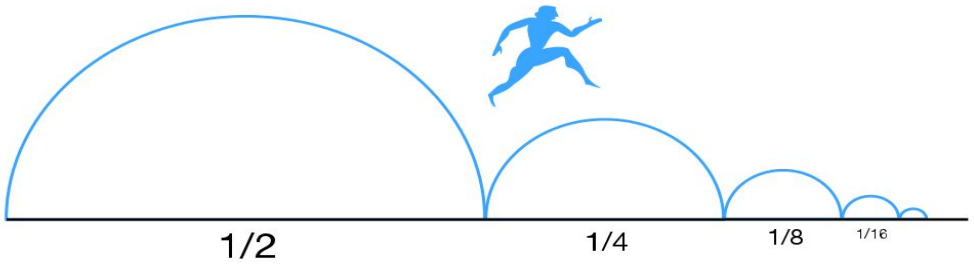
- Comme expliqué précédemment, le but de l’algorithme minimax est de prédire les mouvements de l’adversaire pour choisir les nôtres. Or, lorsque l’on joue l’inspecteur, nous ne disposons pas de la position du fantôme. Cette information est au coeur de la stratégie du fantôme, ce qui rend plus difficile, voire impossible de prédire les mouvements du fantôme.

L’inspecteur et le fantôme utilisent une stratégie similaire (ils partagent une grande partie du code). Les seules différences sont le but et les informations auxquelles ils ont accès.

Pour arriver à leurs fins, nous allons attribuer des poids aux feuilles de notre arbre de décision. Cela nous permettra de suivre le chemin le plus avantageux.
Le calcul du poid dépend du rôle que l’on joue.

1.1 - L'inspecteur

Le but de l'inspecteur est de faire de la dichotomie avec les personnages du jeu.



Cela veut dire qu'il veut disposer les personnages de façon à ce qu'à la fin de chaque set, 4 d'entre eux sont isolés (ou dans la pièce sans lumière) et 4 sont groupés.
Son but, en faisant cela, est de réduire de moitié le nombre de suspects à chaque set. Si il le joue parfaitement, cela réduira au maximum l'avancée de la carlotta et le fantôme sera repéré en 4 sets maximum.
Bien sûr, étant donné le caractère aléatoire et assez complexe du jeu, le fantôme sera certainement repéré plus tôt ou plus tard. Cependant, ce style de jeu est certainement le plus sûr.
Le calcul du gain se fait comme suit:

gain = 8 - abs((isolated) - (grouped))

Le résultat varie entre 0 (très mauvais) et 8 (très bon). Si plus de la moitié des joueurs sont groupés, on ajoute 0.1 pour prioriser cet état en cas d'égalité.

1.2 - Le fantôme

Le but du fantôme est presque opposé à celui de l'inspecteur mais il a accès à la position du fantôme.
Avec cette information, il veut avoir un maximum de personnages soit groupés, soit isolés. Choisir entre ces deux états n'est pas très important, ce qui est capital c'est que le fantôme soit dans le plus grand groupe. Admettons que 6 personnages sont groupés et seulement deux sont isolés. Si le fantôme est l'un des personnages isolés, le gain du set sera catastrophique (6 personnages seront innocentés d'un coup). C'est l'élément à prendre en compte lorsque l'on joue fantôme.
Le calcul du gain est ainsi:

gain = (ghost side) - other side)

Les "sides" représentent les états "groupé" et "isolé".
Le résultat varie entre -6 (catastrophique) et 8 (parfait).
Si le fantôme est isolé, on ajoute 0.1 pour prioriser cet état en cas d'égalité.

1.3 - Atteindre ces buts

Il est important, lorsque l'on met en place la stratégie de garder en vue l'intérêt de l'algorithme minimax: prédire les actions de l'adversaire et minimiser la perte. Cela veut dire que pour un set, en fonction de ce que l'on joue (inspecteur ou fantôme) et de notre position dans le set, on devra utiliser notre calcul de gain ou celui de l'adversaire. Par exemple, si l'on joue inspecteur au premier set, l'arbre se composera comme suit:

Inspector, Ghost, Ghost, Inspector

A propos de l'inspecteur, nous avons expliqué qu'il ne pouvait pas prédire les mouvements du fantôme efficacement sans connaître la position du fantôme. Pour pallier à ce problème, nous allons créer un calcul de gain spécial pour imiter le style de jeu du fantôme:

gain = abs((isolated) - (grouped))

Encore une fois, le résultat varie entre 0 et 8 (8 étant le meilleur gain pour le fantôme) et nous ajoutons 0.1 si la plupart des personnages sont isolés.

2 - Générer l'arbre de décision

Dans notre projet, l'algorithme minimax sera représenté par un arbre de décision. Cet arbre sera composé de noeuds de différent types avec des intérêts différents.
Voici les noeuds de bases de l'arbre:

2.1 - Node

C'est la classe abstraite (basique) pour tous les noeuds. Elle contient les attributs et fonctions qui nous permettent de naviguer dans notre arbre, stocker des informations et les récupérer.
Elle ne devrait pas être utilisé tel quelle.

2.2 - RootNode

Le noeud racine est la base de notre arbre. Il va boucler parmis les personnages jouables et générer les branches. Il ne fait aucun calcul, son intérêt n'est que pour l'architecture.

2.3 - CharacterNode

C'est la classe de base pour tous les personnages (les branches). Son but est de générer les actions du personnage (les feuilles). Comme Node, cette classe ne doit pas être utilisé tel quelle. C'est la classe de base pour l'implémentation des personages

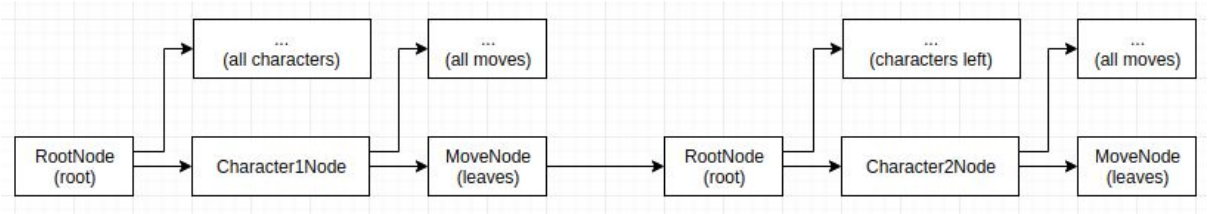
2.4 - DefaultChNode

Cette classe sert à contrôler les personnages qui n'ont pas encore été implémentés. Elle n'utilise jamais le pouvoir du personnage et se contente de déplacer le personnage à une position avantageuse. (bien entendu, le choix du mouvement n'est pas aléatoire).

2.4 - MoveNode

Le noeud de déplacement représente les feuilles de l'arbre. Cependant, ces feuilles appellent à leur tour une RootNode (pour augmenter la profondeur de l'arbre de décision).
Cette RootNode va elle-même appeler une CharacterNode, qui créera des MoveNodes, et ainsi de suite jusqu'au bout du set. Certains pouvoirs des personnages seront des évolutions de MoveNode et seront des feuilles également. Les interactions entre les déplacements et les personnages dépendent des personnage. Elles sont éxpliquées dans la partie 3 de ce document.

Voici un schéma concis de l'aspect de l'arbre (bien sûr, un schéma présentant tout un set serait gigantesque. Nous nous contenterons de quelque noeuds).



3 - Implémenter les pouvoirs

Pour générer l'arbre de décisions, il est nécessaire d'implémenter les comportements des pouvoirs des personnages. Voici, pour chaque personnages, la façon dont il ont été (ou vont être) implémentés. Nous discuterons aussi de la complexité (le nombre de possibilités pour chaque personnage) et, si nécessaire, nous ajouterons un schéma de l'arbre. Certains pouvoir ne sont pas encore implémentés. Une liste à jour est présente dans le README du repository.

Voici les constantes pour le calcul de complexité:

- C: 8 (le nombre de personnages)
- R: 10 (le nombre de pièces)
- P: 2.2 (Le nombre moyen de déplacements possibles)

3.1 Raoul De Chagny

Le pouvoir de Raoul n'a pas besoin de noeud supplémentaire. Il permet de piocher une carte qui va (peut-être) innocenter un suspect. Cette possibilité, quoique aléatoire est prise en compte en ajoutant 0.5 au score finale du chemin. Cela veut dire qu'elle aura la priorité sur un autre chemin n'incluant pas Raoul.
Remarque: le fantôme s'intéresse encore moins aux alibis que l'inspecteur, mais il peut jouer Raoul pour empêcher l'inspecteur de l'obtenir.

Complexité: P

3.2 Meg Giry

Le pouvoir de Meg est le plus simple à implémenter. Elle a accès à plus de chemins que les autres personnages. Elle ne nécessite pas de noeud supplémentaire.

Complexité: 3.3

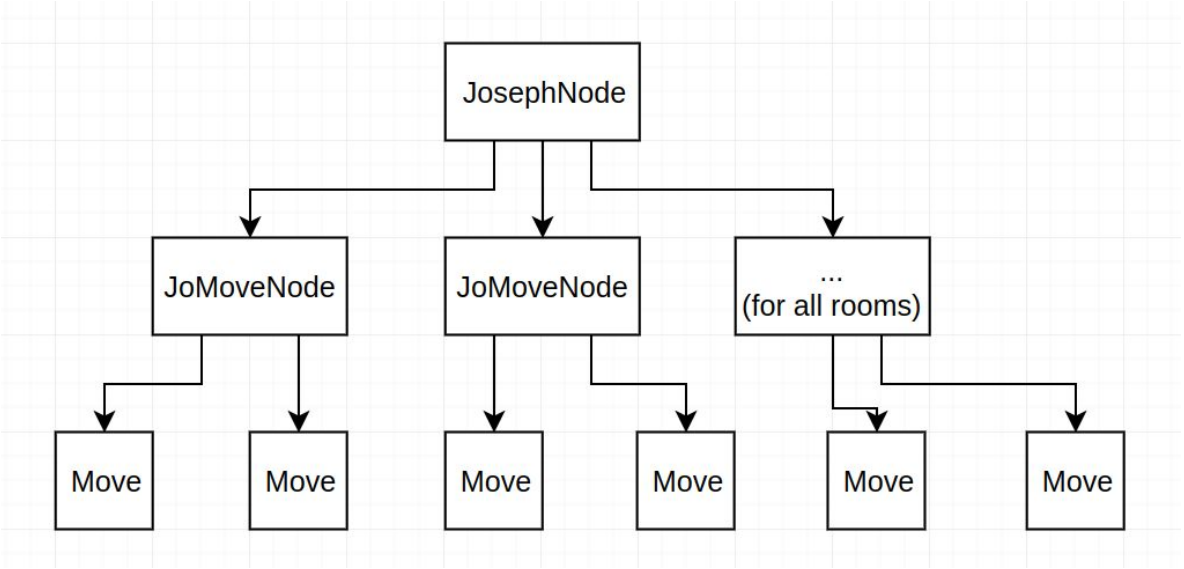
3.3 Ms Giry

Pour Mme Giry, il y a deux façons de gérer la disposition du cadenas. Soit on l'intègre à l'arbre en essayant de le placer dans chaque pièce (ce qui augmentera considérablement la taille de l'arbre), soit on cherche à le placer de façon logique, en fonction des personnages disponibles pendant le set. La deuxième option est un peu moins efficace mais bien plus rapide.

Complexité: P

3.4 Joseph Buquet

Comme pour Mme Giry, nous avons le choix entre tester toutes les possibilités du blackout ou le placer de façon logique. Cependant, ce pouvoir étant certainement le plus efficace, nous allons l'ajouter à l'arbre. Cela va augmenter la complexité mais l'efficacité sera également meilleure.



Complexité: RxP

3.5 Christine Daaé

Pour christine, l'implémentation du pouvoir n'augmente pas beaucoup la complexité. Nous allons donc l'implémenter.

Complexité: 2xP (Tous les déplacements avec ET sans utilisation du pouvoir)

3.6 Mr. Moncharmin

Dans certain cas, ce pouvoir est très coûteux à implémenter, car la possibilité de choisir dans quelle pièce se déplace chaque personnage ouvre la voie à une grande complexité. De plus, cette possibilité n'est pas si intéressante.
C'est pourquoi la meilleur implémentation du pouvoir est un hybride entre minimax est logique. Nous allons tester le pouvoir dans toutes les pièces possibles mais choisir les résultats de ce pouvoir de façon logique, en fonction de quels personnages peuvent encore jouer pendant ce tour.

Complexité: P * 2 (every possible path with AND without using the power).

3.7 Mr. Richard

Mr Richard peut échanger sa place avec n'importe lequel des 7 autres personnages OU se déplacer. Cependant, échanger sa place avec un personnage qui n'agira pas pendant le tour n'a aucun intérêt à court terme. Pour cette raison, on peut réduire son implémentation à "tous les personnages qui vont bouger pendant le set" (max. 3).

Complexité: P + (Nombre de personnages qui peuvent encore bouger)

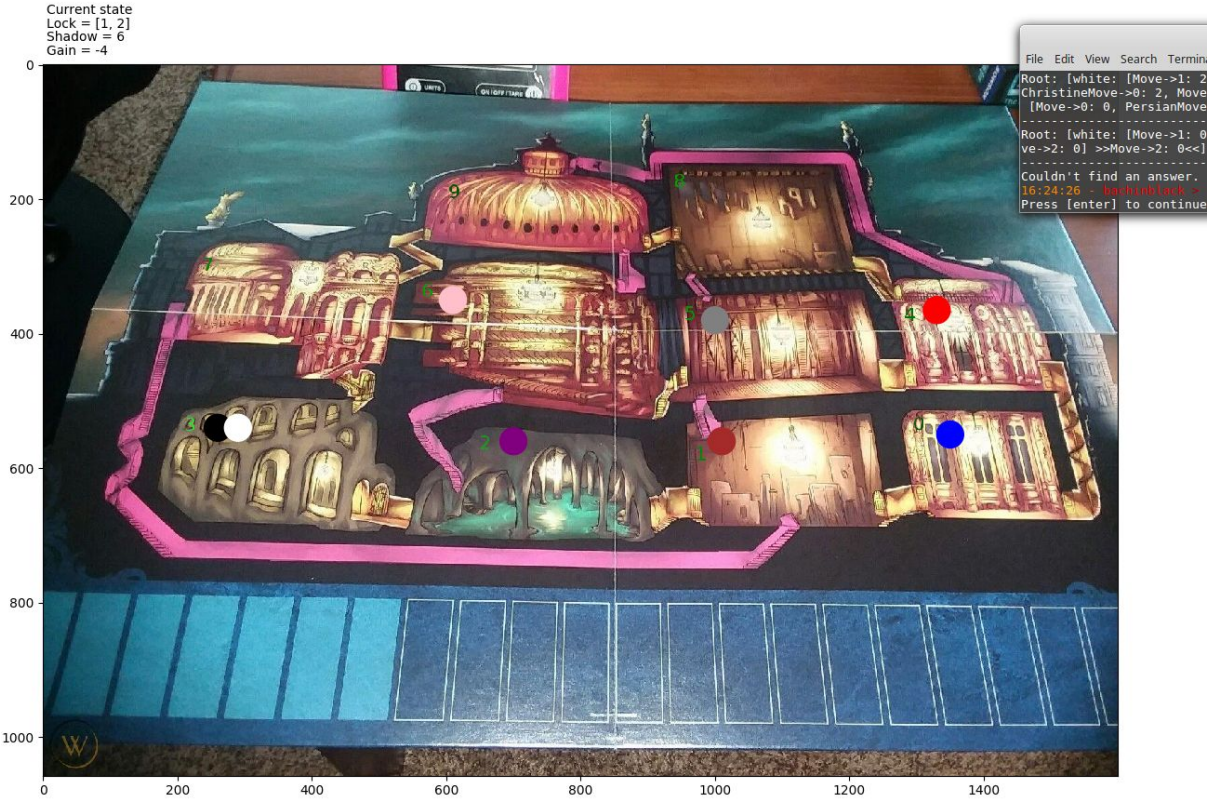
3.8 The Persian

Le pouvoir du Persan n'est pas très coûteux en ressources. Il sera donc implémenté tel quel. Il va essayer de bouger partout ou il peut avec tous les personnages de la pièce.

Complexité: P * (nombre de personnage dans la même pièce)

4 - Visualisation du jeu

Pour débugger le jeu, le plus efficace était de le voir comme en vrai:



Cette interface graphique est réalisée avec matplotlib et nous permet de voir la position de tous les personnages (cercles de couleurs), ainsi que celle du cadenas et du blackout. Sont également visibles, le gain actuel et le dernier mouvement effectué. Les personnages inventés sont représentés par des carrés à la place de cercles.
Quand la fenêtre s'ouvre, il faut retourner sur le terminal et appuyer sur [enter] pour voir l'avancement du jeu.

Pour activer et désactiver cet élément, il faut envoyer True ou False à la fonction `display.init_debug` (player.py, l.19).