Projet semestriel

Jeu de Puissance 4 en C

Dossier d'Analyse

Aix-Marseille Université

MAZZOLA Alexandre et MUCHA Pierre

Sous la supervision de Valérie Campillo

May 8, 2017

Table des Matières

1 Structure générale

1.1 Schéma de fichiers

Pour ce projet, c'est une structure de fichiers modulaire à base d'en-têtes et de fichiers sources qui a été choisie, notamment dans une optique de propreté et de lisibilité. Cette structure est détaillée dans le schéma ci-dessous. Le processus de compilation est facilité par un script Makefile, largement reconnu et utilisé pour la programmation en C / C++ / C#.

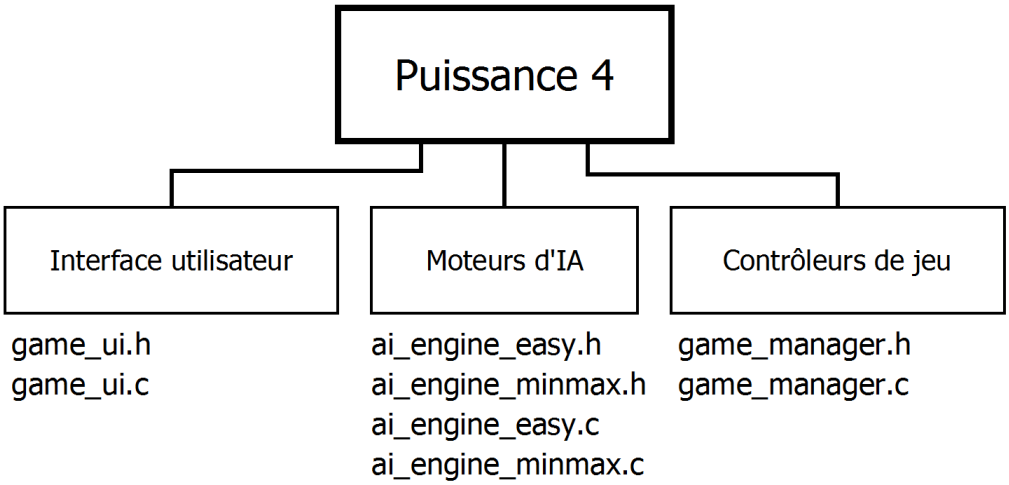

Figure 1: Structure de fichiers générale

1.2 Routine principale

La routine principale, aussi appelée main(), sert ici de "point d'entrée" et de "hall" pour l'utilisateur dans le programme : c'est à partir d'elle qu'est invoqué le menu principal et que le choix est fait du contrôleur de jeu, en fonction du choix du ou des joueurs. C'est le niveau le plus haut du programme, étant donné sa modularité.

1.3 Interface utilisateur

L'interface utilisateur (UI) regroupe toutes les fonctions relatives à la prise en charge de l'affichage. Dans le code, pour faciliter la lecture, elles sont séparées en quatre grandes familles. D'abord les fonctions génériques, permettant l'initalisation et la purge de l'affichage. Ensuite viennent les fonctions relatives au menus, et celles relatives aux boîtes de dialogue qui permettent de faire passer des informations à l'utilisateur (popups). Vient ensuite un groupe qui est utilisé par les deux derniers, qui permet de dessiner les boîtes contentant à la fois les menus et les popups. Non mentionné dans les schéma viennent aussi les fonctions qui dessinent le curseur désignant la colonne à jouer, et la fonction affichant l'entièreté du plateau de jeu.

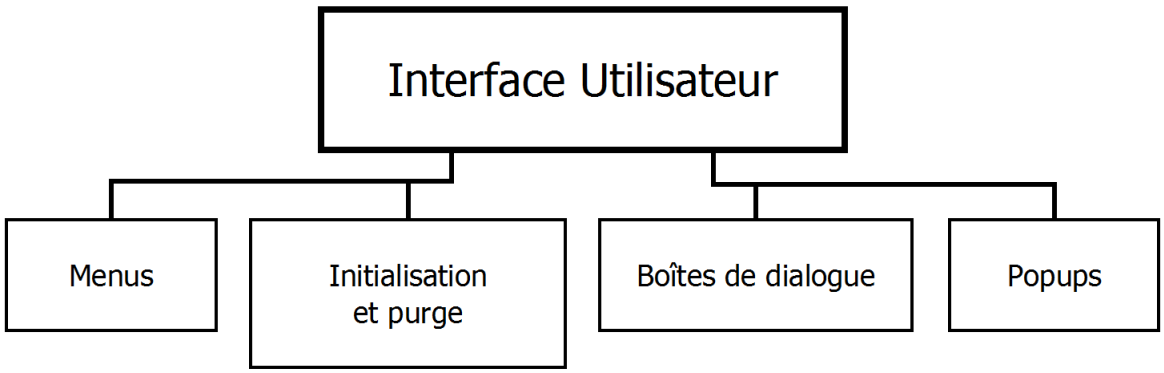

Figure 2: Interface utilisateur

1.4 Gestion de partie

La gestion de partie est segmentée en 3 grands éléments :

- les deux gestionnaires de partie relatifs aux deux IA disponibles
- le gestionnaire qui opère les parties humain contre humain
- les fonctions diverses, qui touchent plus à la mécanique du jeu (notamment liées à la maintenance de la représentation en mémoire du plateau de jeu)

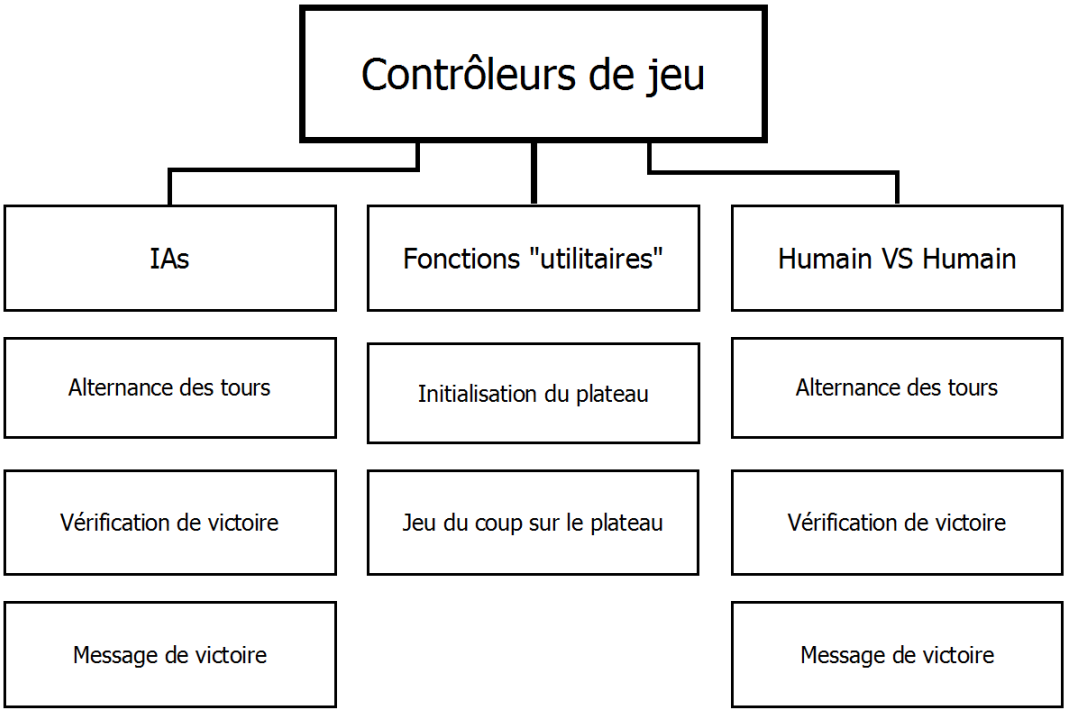

Figure 3: Gestion de partie

2 Structure et différentes couches événementielles

Le coeur d'un jeu comme celui-ci est une ou plusieurs boucles événementielles, qui à chaque itération vérifie si certaines actions ont été effectuées par l'utilisateur, de manière à répondre à ces actions de manière adéquate. Cette boucle possède une condition, qui constitue bien souvent la condition sous laquelle l'utilisateur veut quitter le jeu, ou le bloc événementiel actuel, pour retourner du menu au jeu après une erreur exemple.

Le schéma ci-dessous représente les différents blocs événementiels du programme, et leur hiérachisation. On remonte dans la hiérarchie par des codes de retour qui sont considérés comme des événements par le bloc parent et donnent lieu à une ou plusieurs actions appropriées.

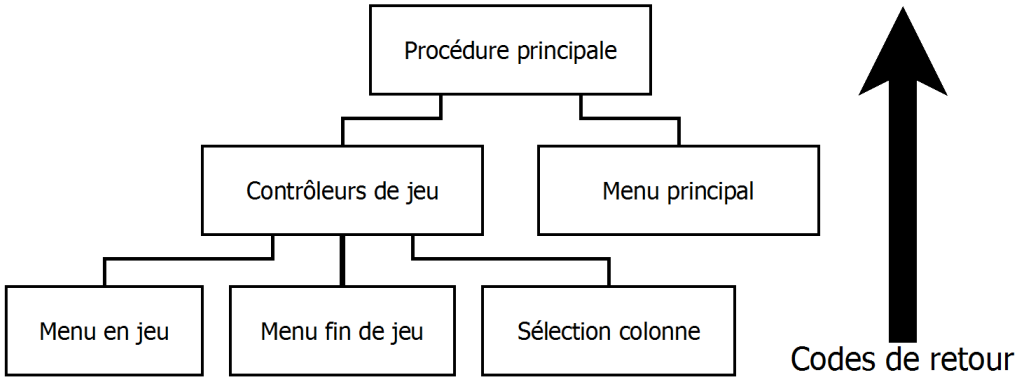

Figure 4: Structure événementielle

3 Intelligence Artificielle (MinMax et heuristique)

Cette partie du jeu, nommée "IA 2" dans le menu principal n'est pas achevée ; aussi l'IA n'est-elle pas opérationnelle. Le code produit a cependant été laissé à l'appréciation de l'examinateur

Le choix de l'algorithme pour l'IA non suggérée par l'énoncé s'est porté sur la méthode MinMax. Le principe de cet algorithme est d'évaluer l'arbre de possibilités, et de favoriser, tour à tour, les solutions les plus avantageuses quand c'est notre tour, et les solutions les moins avantageuses quand c'est le tour de l'adversaire : en effet, l'adversaire devrait choisir les solutions qui nous avantagent le moins. Il faut donc voir l'arbre comme une succession de couches "min" et de couches "max".

Une solution d'optimisation pour cet algorithme est appelée l'élagage Alpha-Beta : mettons que l'on cherche la valeur maximale pour les enfants d'un noeud. Ces enfants eux-mêmes cherchent donc la valeur minimale de leurs propres enfants. Si un des enfants dont on cherche la valeur maximale a une valeur supérieure à un autre enfant qui n'a été que partiellement évalué, puisque ce dernier ne cherchera que les valeurs minimales, on sait que sa valeur ne grandira pas, et donc que le premier enfant sera de toutes façons supérieur au second. Ce n'est donc pas la peine de continuer à évaluer le second.

En clair, il s'agit d'éviter d'évaluer des parties de l'arbre dont on sait qu'elles sont inutiles au calcul de l'algorithme minmax.

Une simplification supplémentaire de l'algorithme est nommée "NegaMax" : il s'agit d'utiliser une propriété simple : max(a,b) = - min(-a, -b). On peut ainsi simplifier l'algorithme en une seule fonction "negamax" au lieu d'avoir une fonction min et une fonction max.

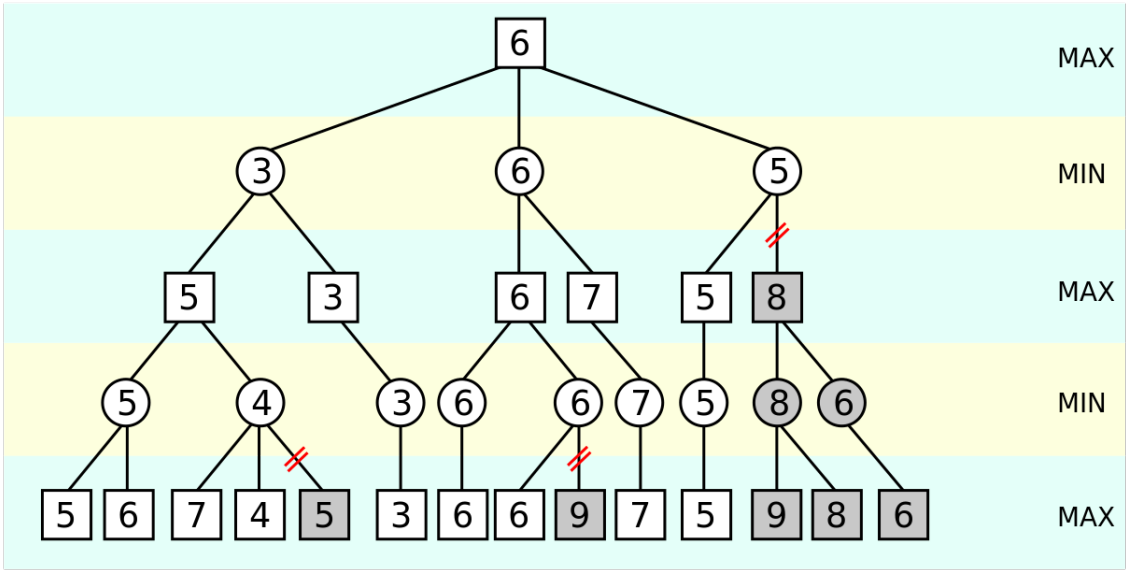

Figure 5: Exemple d'un élagage Alpha-Beta sur un algorithme MinMax

L'évaluation d'une situation du jeu fait appel à une heuristique : une méthode d'évaluation basée sur des critères arbitraires, dont la qualité fait l'efficacité l'IA. Ici, l'heuristique choisie est relativement simple, puisqu'il s'agit de faire la différence entre le nombre de groupes de 4 pions formables par l'IA, peu importe le nombre de tours nécessaires, et ce nombre de groupes formables par le joueur.

On aurait également pu se pencher, puisque le jeu a été résolu mathématiquement, sur la méthode développée à cette fin, qui permet notamment d'opposer un jeu parfait, c'est à dire de gagner à tous les coups si on est premier à commencer, et de ne jamais perdre (égalité si l'adversaire a lui aussi un jeu parfait) si on commence second.