Projet d'étude :
Programmation du jeu Awalé.

Okrou Poda
Souleiman Iman Choukri

L2 science pour l'ingénieur
2014/2015

Encadrant : Barrault Loïc

I. Introduction

L'Awale est un jeu de société combinatoire abstrait crée en Afrique. C'est le plus répandu des jeux africains de type « compter et capturer » dans lesquels on distribue des cailloux, graines ou coquillages dans des coupelles ou des trous, parfois creusés à même le sol .Étant en deuxième année de licence science pour ingénieur, dans le module projet, nous avons choisi comme projet le jeu Awalé pour réaliser notre projet d'Étude .

Cependant comment un jeu si simple à jouer peut être programmer ? Quel langage utiliserons nous ?Y-a t-il des règles à respecter pour ce jeu ? Quel est le but de ce jeu ? Pour répondre à bien à ces différentes assertions dans notre analyse nous détaillerons comment se jeu à été mis en place à travers les différents cours reçu en programmation et tous les différents éléments qui nous ont permis à réaliser ce jeu .

II. Organisation du travail

Concernant l'organisation du travail, on commence par diviser les tâches en partant des tâches les plus prioritaire jusqu'à ceux les moins importantes (interfaces graphique...). Cependant on se pose trois tâches différentes : la construction de la plateforme et du jeu, l'aide et la sauvegarde et finalement les amélioration possible. A partir de chaque étape on forme des sous étapes et on se partage les fonctions à écrire.

Grâce à l'outil de version on a accès aux dernières modifications de notre collègue et l'on travaille ensemble sur les fonctions compliquées. Avec les séances réservées au projet, on évalue le progrès de chaque personne et l'on continue sur les étapes qui suivent.

III. Règles du jeu

Le jeu Awalé est composé de plusieurs règles comme la majorité des jeux. Ces différentes règles se résument en deux parties : les conditions de pour gagner une partie du jeu et les conditions de la fin d'une partie.

Le plateau est divisé en deux territoires. Le territoire du sud appartient au joueur 1 et le territoire à l'opposé à l'adversaire. Chaque territoire est composé de six cases dont chacune contient quatre graines. Le jeu consiste à ramasser les graines de l'une des cases de son territoire puis de les distribuer.

Les occasions où manger des graines est possible sont lorsque dans le territoire adverse il existe une case contenant une ou deux graines et que la dernière graine distribuée par le joueur en question tombe dans les cases citées. Cependant il existe aussi la capture multiple qui permet d'élargir la capture des graines des cases qui précèdent la case où la dernière graine est posé si ces dernières contenaient deux ou une graines au avant la distribution.

Un joueur remporte la parte si ce dernier obtient un score minimum de 25 graines capturées ou si son adversaire est affamé et que le joueur en cause ne peut pas le nourrir.

Cependant cette notion d'adversaire affamé signifie que le territoire de l'adversaire est affamé et qu'on doit le nourrir c'est à dire distribuer des graines à son territoire. Cependant cette notion d'affamé ne s'arrête pas ici, un joueur ne doit pas capturer toutes les graines restantes de son adversaire si cela va l'affamé.

En conclusion, on a pas le droit de jouer une case qui va affamé l'adversaire ou une case qui ne nourrit pas l'adversaire en cas de famine. Si le joueur ne peut pas le nourrir donc ce dernier gagne la partie et les graines restantes dans son territoire sont ajoutées à son score.

Il existe un deuxième cas qui pourrait finaliser une partie : si au bout de plusieurs les coups gagnants sont difficiles à jouer donc dans ce cas chaque joueur récupère les graines de son camp et ces dernières sont ajoutées à son score.

IV. Analyse du problème et conception du jeu

Comptant les différents points pour la programmation du jeu, on divise la conception du jeu en deux parties :

- la construction du jeu c'est à dire la plate-forme du jeu ainsi que la gestion des joueurs, scores, déplacement dans la grille, capture de graines,
- la sauvegarde du jeu c'est-à-dire le chargement d'une partie sauvegardée,
- l'affichage des meilleurs scores et des instructions du jeu.

1- Construction de la grille et du jeu

Dans cette partie nous allons aborder les différents parties pour mettre en place le jeu. Ces dernières sont les structure de donnée utilisée pour définir le jeu et les différentes fonctions.

a) Structure de données

Concernant le plateau de jeu on a choisit une matrice 2x6, sachant que le territoire de chaque joueur est composé de 6 cases. On représente les graines de chaque case par des numéros représentant le nombre de graines dans chaque case. Le territoire du joueur 1 est la ligne 1 de la matrice, celui de l'adversaire la ligne 0.

Pour déplacer une case le joueur saisit le numéro de la case à récupérer or pour ne pas gêner l'utilisateur, les cases sont numérotées de 1 à 6 mais pendant les traitements, ces case sont numérotées de 0 à 5.

b) Fonctionnalités du programme

Pour la construction du plateau de jeu on dispose de deux fonctions qui permettent d'initialiser la matrice et l'afficher :

- init_matrice : permet d'initialiser la matrice avec quatre graines dans chaque case et prend en paramètre la matrice à initialiser.
- affiche_matrice : permet d'afficher le plateau de jeu et prend en paramètre la matrice à initialiser.

Ensuite vient le tour de jouer une partie. Cependant cette partie intègre deux types de jeu différents, un jeu qui associe deux joueurs et un autre qui associe l'ordinateur et un autre joueur. Ces deux catégorie différents sont joués séparément mais les fonctions utilisées sont pareils. Donc en général on établit les fonctions nous permettant de distribuer les graines et de les capturer tout en prenant compte des différents conditions du jeu.

On introduit les fonctions suivantes, qui elles font appel à une autre fonction dans la majorité pour remplir toutes les conditions de ramassage des graines :

- dpl_arriere : permet de se déplacer en arrière sur la matrice en modifiant les coordonnée reçu en paramètre. Elle prend en paramètre des pointeurs sur les coordonnées x et y.
- dpl_avant : permet de se déplacer an avant sur la matrice en modifiant les coordonnée reçu en paramètre. Elle prend en paramètre des pointeurs sur les coordonnées x et y.
- manger_graines : fonction permettant de distribuer les graines, à l'aide des fonctions de déplacements (dpl_avant et dpl_arriere) tout en incrémentant le score du joueur en cause si le tour joué s'agit d'un tour gagnant. Cette fonction prend en paramètre le nombre de graines dans la case en question à ramasser, la matrice représentant le plateau de jeu, le coordonnée x et y de la case dans la matrice et un pointeur sur le score du joueur. Elle incrémente le score du joueur au nombre de graine ramassée quand un coup gagnant est détecté. Cependant le ramassage de graines est possible que dans certaines conditions telles que :
  - les graines à ramasser se trouvent dans le clan adverse ;
  - la case où la dernière graine est déposée contient une ou deux graines ;
  - le ramassage des graines n'affame pas l'adverse.

La fonction non_affame cité ci dessus permet de vérifier si le ramassage n'affamera pas l'adversaire.

- non_affame : cette fonction simule la deuxième partie de la fonction manger_graine c'est-à-dire la partie où les coups gagnants sont évaluées et exécutés. Or la fonction non_affame joue cette partie en avance sans pour autant modifier le plateau de jeu (une matrice temporaire est utilisée) puis examine la plateau de jeu temporaire grâce à la fonction plateau_vide, qui indique si le territoire d'un joueur est vide ou pas. Ainsi on peut évaluer si le coup joué va affamer l'adversaire ou pas.
- plateau_vide : cette fonction permet d'indiquer si le territoire d'un joueur est vide ou pas. Elle prend en paramètre le plateau de jeu et le joueur en question.

Ainsi grâce à ces fonctions on construit le premier bloc permettant le jeu entre deux joueurs puis à chaque tour joué, le score du joueur est également affiché donc on a :

- afficher_score : cette fonction affiche le score du joueur et son pseudonyme, elle prend en paramètre le score et le pseudonyme du joueur.

Pour le jeu entre l'ordinateur et un autre joueur, on crée une fonction qui permet de simuler le jeu de l'ordinateur.

- jeu_ordi : cette fonction renvoie la case que l'ordinateur va jouer. Cependant certaine conditions sont posées telles que :
  - la case choisit ne soit pas une case vide ;
  - la case est choisit selon le gain qu'elle rapporte ;
  - la case choisit nourrisse l'adversaire en cas de famine.

Avec la fonction ci-dessus, une partie entre entre l'ordinateur et un joueur est faisable sachant que le jeu du joueur adverse à l'ordinateur est identique au jeu d'une partie entre deux joueurs mais le jeu n'est pourtant pas complet, les conditions d'arrêts de jeu reste à être évaluer..

Concernant ces dernières, on dispose de certaines conditions qui permettent de définir si la partie est finie ou pas telles que :

- le score d'un des joueurs est égale ou supérieure à 25 ;
- le territoire d'un des joueurs est vide mais son adversaire est incapable de le nourrir ;
- les coups gagnants sont difficiles à trouver au bout d'un certains nombres de tours joués.

On fixe le nombre de tours à cinq pour la dernière condition même si que sur un exemple de jeu d'Awalé ce dernier est de dix donc au bout de cinq tour si les deux joueurs n'ont pas réalisé de coup gagnant (score inchangé pendant les cinq tours) et que ces derniers sont incapable de nourrir l'autre pendant les cinq tours donc la partie s'arrête et chacun récolte les graines disposées dans son territoire. Pour exécuter ces différentes tâches on construit les fonctions suivantes :

- partie_finie : permet de vérifier si l'un des joueurs a un score supérieure ou égale à 25 et prend en paramètre les scores des deux joueurs;
- nourrir : cette fonction permet de vérifier si un joueur peut nourrir son adversaire ou pas, elle prend en paramètre la matrice et la ligne du joueur en cause. Elle permet de vérifier pour chaque case du territoire du joueur si cette dernière est capable de nourrir l'adversaire ou pas ;
- jeu_possible : cette fonction permet d'évaluer si la partie est jouable pour un joueur en examinant la situation du joueur c'est-à-dire si le joueur est affamé ou pas puis si oui son adversaire peut il le nourrir ou pas et si son adversaire est affamé, le joueur peut il le nourrir ou pas. Si ces derniers cas ne sont pas validés donc la partie continu. Elle prend en paramètre la matrice (plateau de jeu) et la ligne du joueur en question ;
- gain_difficile : cette fonction modélise la troisième condition d'arrêt de jeu cité ci-dessus en vérifiant si les dispositions des graines du plateau répondent à ces critères. Elle prend en paramètre la matrice et un pointeur sur le nombre de tour qui lui est incrémenté à l'intérieur

de la fonction à chaque fois que la situation en question est rencontrée.
Finalement on a une dernière fonction :

- victoire : permettant d'afficher le joueur gagnant et en enregistrant ce dernier dans le fichier contenant les records des joueurs gagnants. Elle prend en paramètre un fichier.

c) Récupération et gestion des cas d'erreur

Après avoir établit les bases de fonctionnement du jeu, on traite les différents cas d'erreur envisageable. Ces derniers sont liés à la saisie de l'utilisateur or on compte trois cas possibles :

- la saisie d'un indice d'une case qui n'est pas dans l'intervalle prévu c'est-à-dire de un à six ;
- la saisie d'un indice d'une case vide ;
- la saisie d'un indice d'une case qui ne nourrit pas l'adversaire si ce dernier est affamé.

Cependant pour traiter ces cas d'erreurs on met en place quelques fonctions qui traitent l'une des cas d'erreur et un bloc composé d'un condition vérifiant si la case est vide ou si l'indice saisie est incorrect. Ce genre de traitement est directement introduit dans la fonction ordinateur car aucune valeur est récupérée. Les fonctions disposées à la gestion de ces cas sont donc :

- nourrir_case: cette fonction permet de vérifier si la case sélectionner pour un joueur dont son adversaire est affamé peut nourrir ce dernier. Elle prend en paramètre la matrice (plateau de jeu), la ligne du joueur et le coordonnée x de de la case.
- coup_possible : cette fonction permet de déterminer si le coup joué par un joueur est possible dans le cas où l'adversaire est affamé. Elle prend en paramètre la matrice (plateau de jeu), la ligne du joueur et le coordonnée x de de la case et fait appel à la fonction nourrir_case si l'adversaire est affamé.

Ainsi si c'est différents cas sont rencontrés et que l'utilisateur ne saisit pas une valeur correcte donc on affiche un message d'erreur et on redemande la saisit jusqu'à une bonne saisie.

2- Sauvegarde du jeu et aide

Dans cette partie, on progresse vers la mise en place d'une fonction aide et la construction de la sauvegarde du jeu qui se présente sous deux parties c'est-à-dire la sauvegarde puis le chargement de la partie.

a) Structure de données

Concernant les structures de données on dispose d'un fichier texte qui permet de stocker les graines de chaque case de la matrice.

b) Fonctionnalités du programme

Pour la sauvegarde et le chargement de la partie on développe deux fonctions différentes :

- sauvegarder : elle permet d'écrire le nombre de graines de chaque case dans un fichier ainsi que le pseudonyme du joueur 2, ce qui nous permet d'identifier si la partie à charger est une partie avec l'ordinateur ou une partie entre deux joueurs. Elle prend en paramètre le fichier de sauvegarde.
- charger_partie : elle permet de lire les données d'un fichier puis d'initialiser la matrice avec les données du fichier reçu en paramètre ainsi que le pseudonyme du joueur 2 .

Concernant l'aide, on établit une fonction aide qui permet de calculer la case qui permet de recueillir le plus de graines. On décide de limiter le nombre de l'aide à trois joker.

c) Récupération et gestion des cas d'erreur

On remarque que dans le cas d'un fichier vide les processus fonctionne mais sans avertissement ce qui peut être désagréable à l'utilisateur donc avant de charger une partie, on vérifie si le fichier ne serait pas vide ou pas. Puis si le cas est oui donc on affiche un message précisant qu'aucune sauvegarde est présente pour le moment.

3- Affichage des records et des instructions

Dans cette partie, on construit une étape plutôt lié au cadre des informations supplémentaires : la liste des records et l'affichage des instructions du jeu.

a) Structure de données

Concernant les structures de données on dispose d'un fichier texte qui permet de stocker les scores et pseudonyme des joueurs gagnants ainsi qu'un tableau de taille M égale à 100, de type struct : t_joueur, définissant les informations d'un joueur (score et pseudonyme).

b) Fonctionnalités du programme

On a adopter une règle à l'affichage des scores c'est-à-dire, seuls les dix premiers joueurs sont affichés. Pour manipuler les différents scores on développe la fonction suivante :

- classer_records : cette fonction permet de stocker tous les pseudonymes et leurs scores respectifs dans un tableau de type t_joueur puis trie ce tableau de façon qu'il soit ordonné du plus grand au plus petit puis affiche seulement les dix premiers du tableau. Elle reçoit en paramètre le fichier contenant les records, un pointeur sur un un tableau de type t_joueur et un pointeur sur le nombre de score initialisé à zéro dans le main.
- regles : fonction permettant d'afficher les différentes instructions du jeu.

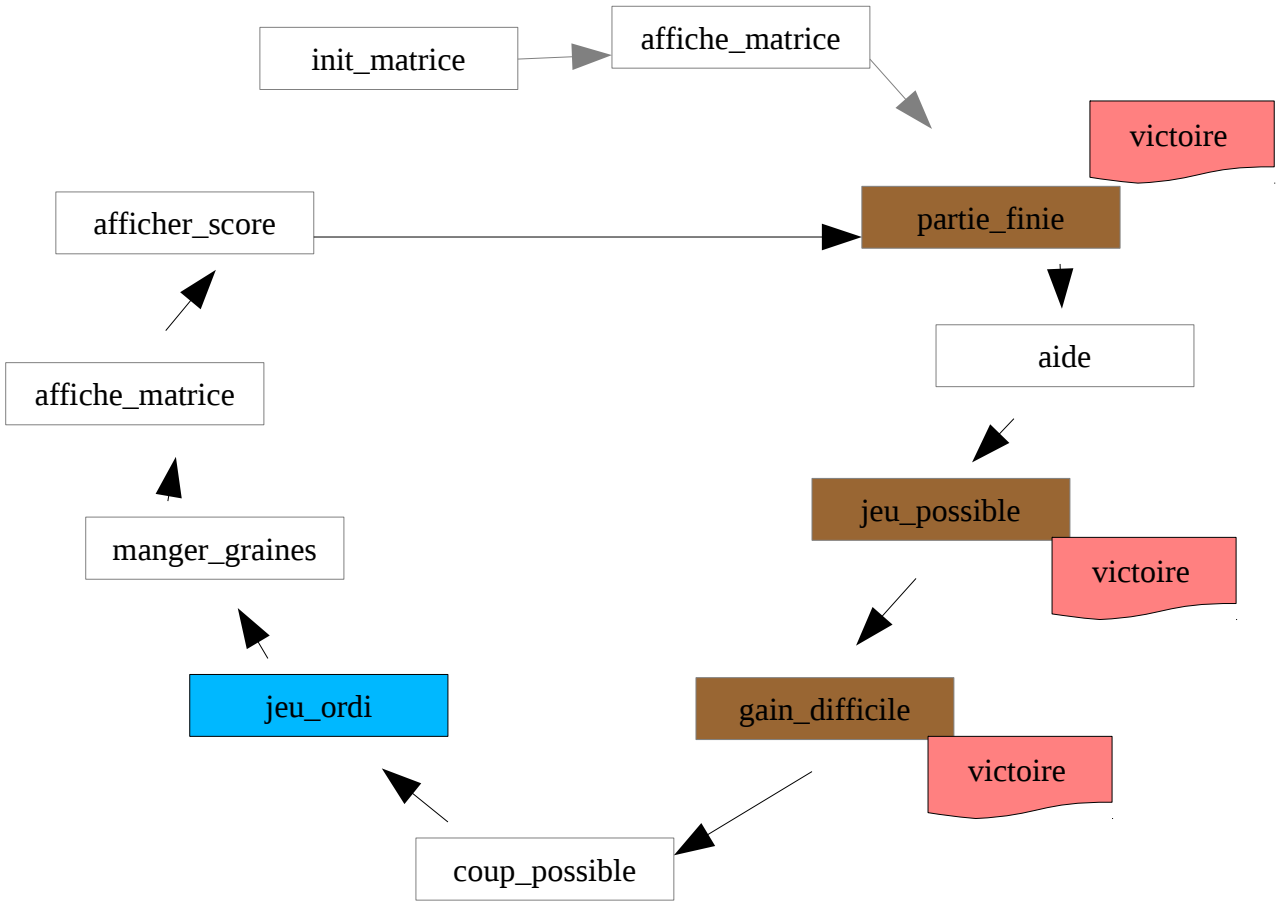
c) Récupération et gestion des cas d'erreur

On remarque que dans le cas d'un fichier vide les processus fonctionne mais sans avertissement ce qui peut être désagréable à l'utilisateur donc avant de charger une partie, on vérifie si le fichier ne serait pas vide ou pas. Puis si le cas est oui donc on affiche un message précisant qu'aucun score est enregistré pour le moment.

V. Codage et méthode de programmation

En première phase, on construit tout le programme dans le programme principal, le main, et les différentes fonctions sont toutes dans ce même fichier. Grâce la commande switch on réalise un menu qui demande le choix de jeu de l'utilisateur, c'est-à-dire si une partie entre deux joueurs ou une partie avec l'ordinateur est désirée, ainsi que d'autres(records, continuer une partie, instruction, nouvelle partie). Puis selon le choix de l'utilisateur, la partie commence.

On enchaîne les différentes fonctions selon le schéma de la figure 1.



Légendes :

Étapes représentant la boucle while du fonctionnement d'une partie.

Début d'une partie.

Segment pouvant arrêter la partie si les conditions de la fonction sont validées.

Segment introduit si la partie est avec l'ordinateur.

Fonctions exécutées si les conditions sont validées et la partie terminée.

Figure 1 : Schéma présentant le déroulement d'une partie.

Concernant la partie rechargée, on réutilise le schéma de la figure 1 mais on vérifie le type de la partie sauvegardée, soit une partie avec l'ordinateur soit une partie entre deux joueurs.

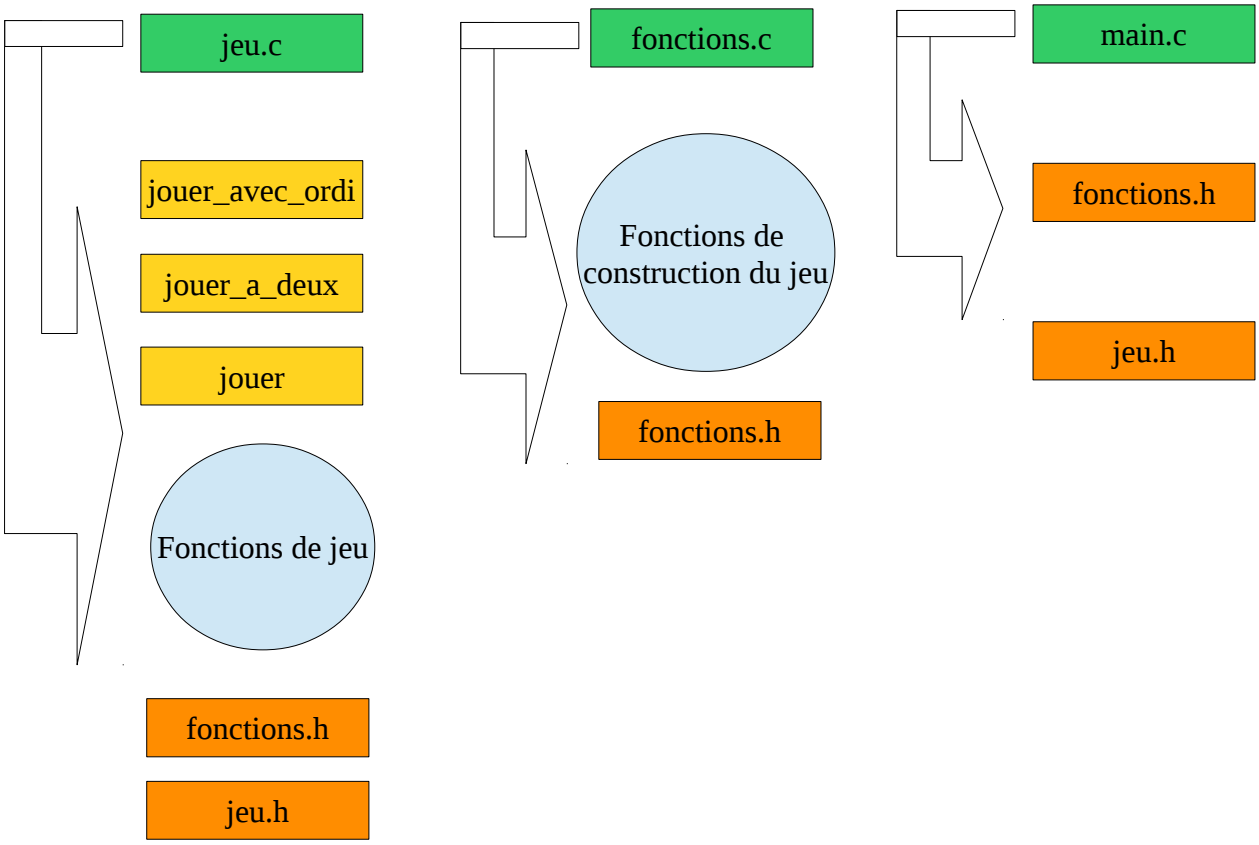
Cependant pour établir cette vérification on pose une condition qui compare le pseudonyme du joueur 2 enregistrée avec le mot « Ordinateur ». Ainsi selon le résultat on exécute les différents étapes de la figure 1.

Ensuite on découpe le programme en plusieurs fichiers, on crée les fichiers jeu.c et fonctions.c qui contiennent respectivement les fonctions ordonnant le déroulement d'une partie et les fonctions nécessaire à la construction du jeu (affiche_matrice, afficher_score...).

On remarque que le système de la figure 1 se répète pour les options « nouvelle partie » et « continuer une partie ». Cependant on décide de développer deux fonctions jouer_a_deux et jouer_avec_ordi permettant respectivement de reprendre le déroulement du schéma de la figure 1 tout en respectant pour chaque partie ses propres conditions. On ajoute une dernière fonction, jouer, qui permet de reprendre les deux dernières foncions citées ci-dessous pour jouer à une partie sauvegarder. Ces dernières fonctions prennent en paramètre le fichier de sauvegarde et

le fichier

des records des scores. On présente l'ensemble dans la figure 2 ci dessous.



Légendes :

Fichiers composant les sources du programme.

Fonction créée lors du découpage du programme .

Fonctions propres à chaque fichier.

Fichier header liant les différentes fonctions au fichier en question.

Figure 2 : Schéma présentant la liaison des différents fichiers et différentes fonctions après le découpage du programme.

Pour éviter les conflits causés par les donnée déjà définit on utilise les conditions #ifndef qui nous permettent d'éviter les erreurs à la compilation.

Ensuite on ajoute d'autre fonctionnalités au programme tels que l'affichage des meilleurs score et l'affichage des instructions en ajoutant les fonctions classer_records et regles_jeu dans le fichier jeu.c.

Concernant la fonction jeu_ordi on décide de limiter le choix de la case à la case qui rapporte le plus de graines. Puis si l'adversaire est affamé, on adapte cette dernière pour que l'ordinateur nourrit automatiquement l'adversaire en choisissant la case appropriée. Dans le cas où les gains possibles sont tous égaux, la case choisie est la case la plus proche comptant de 0.

VI. Résultats

On commence à jouer pour tester les différentes conditions du jeu et on a rencontré quelque bug or grâce au GDB on suit les différentes fonctions pour régler le problème. La partie jouée avec l'ordinateur dispose de la même logiue que celle d'une partie à deux donc on obtient les résultats des figures suivants.

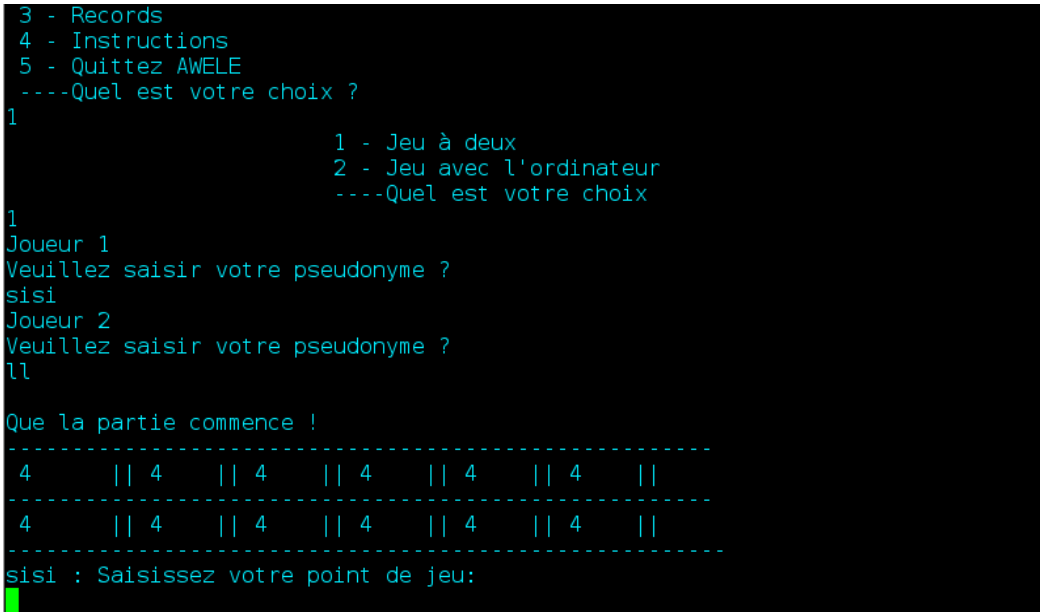


Figure 3 : Image présentant le début du jeu

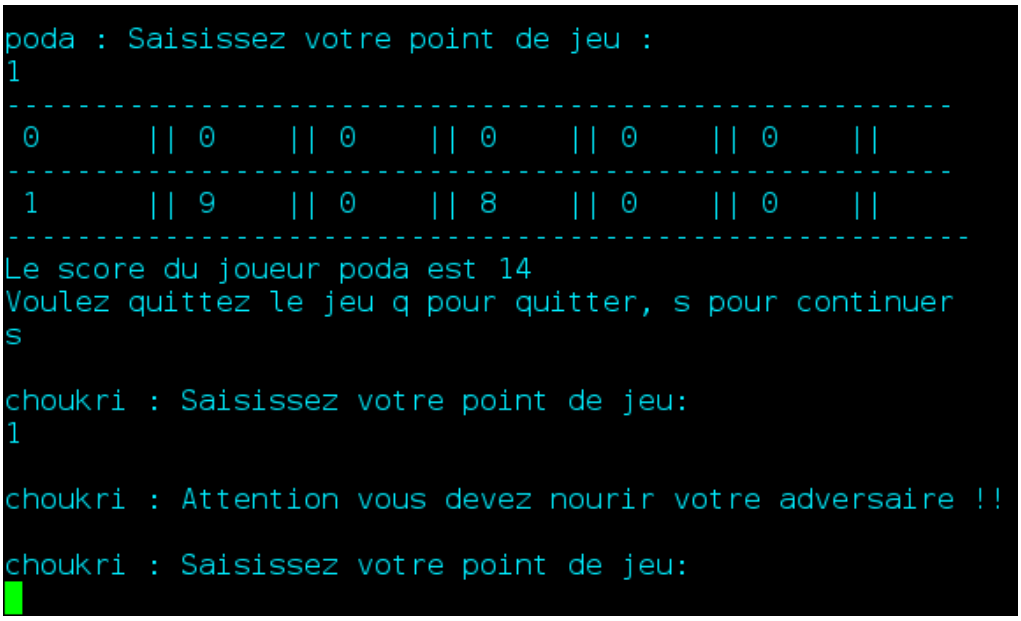


Figure 4 : .Image présentant le cas où l'indice d'une case vide est saisit.

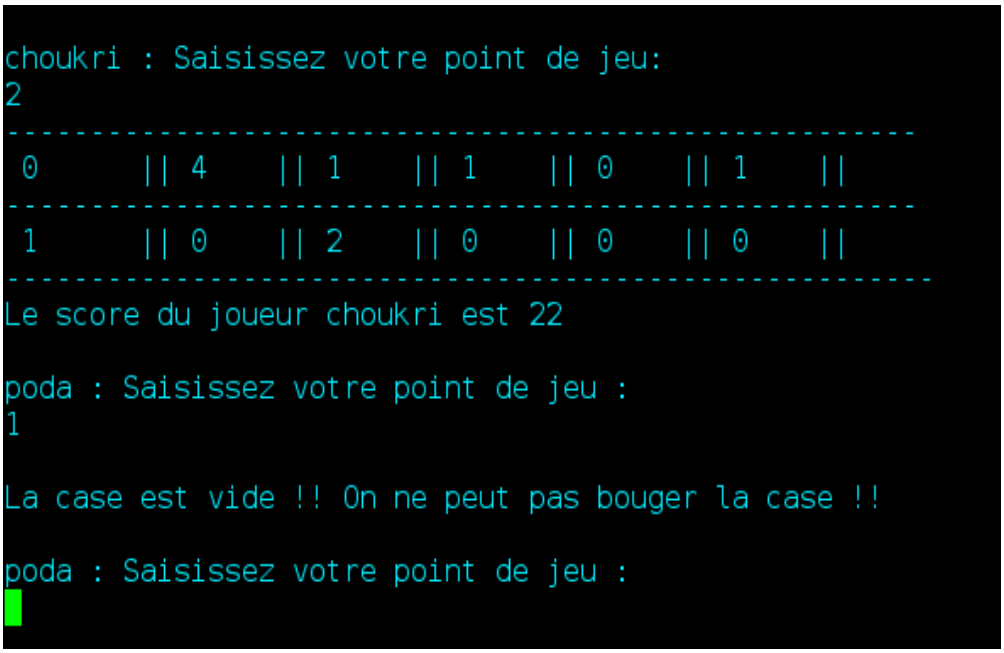


Figure 4 : Image présentant le cas où l'un des joueurs est affamé et que le choix de l'adversaire est évalué.

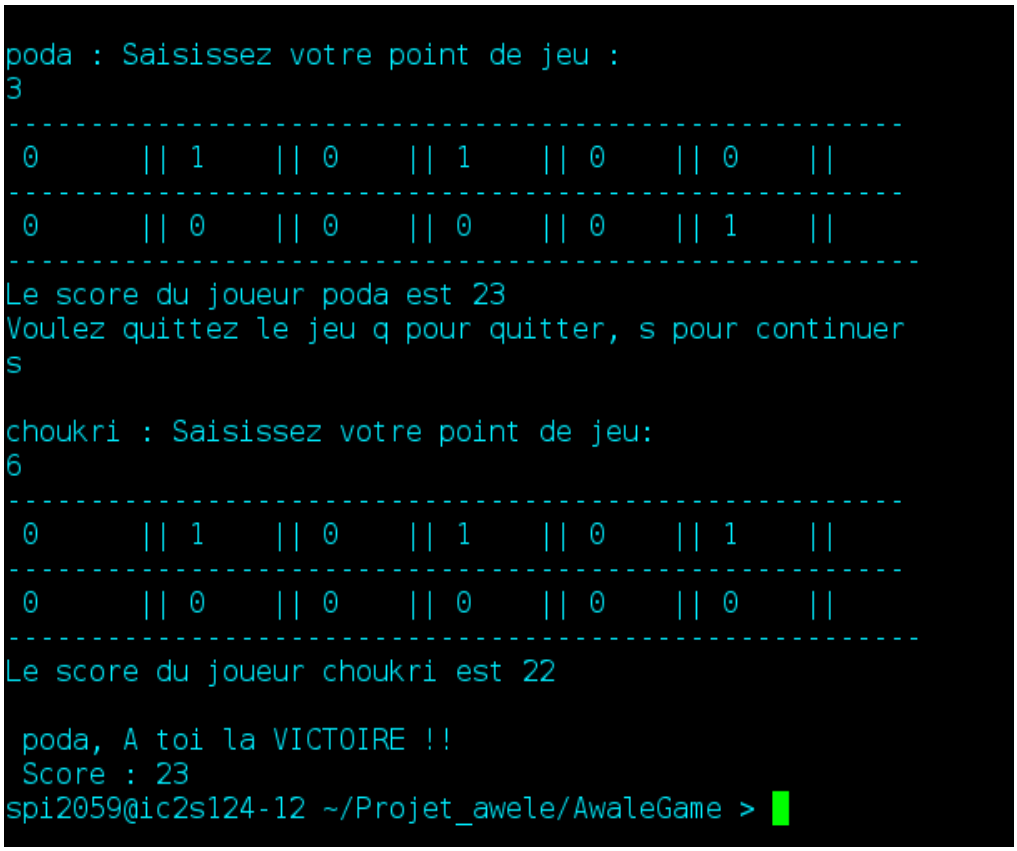


Figure 5 : Image présentant la victoire d'un joueur lorsque ce dernier ne peut plus nourrir son adversaire.

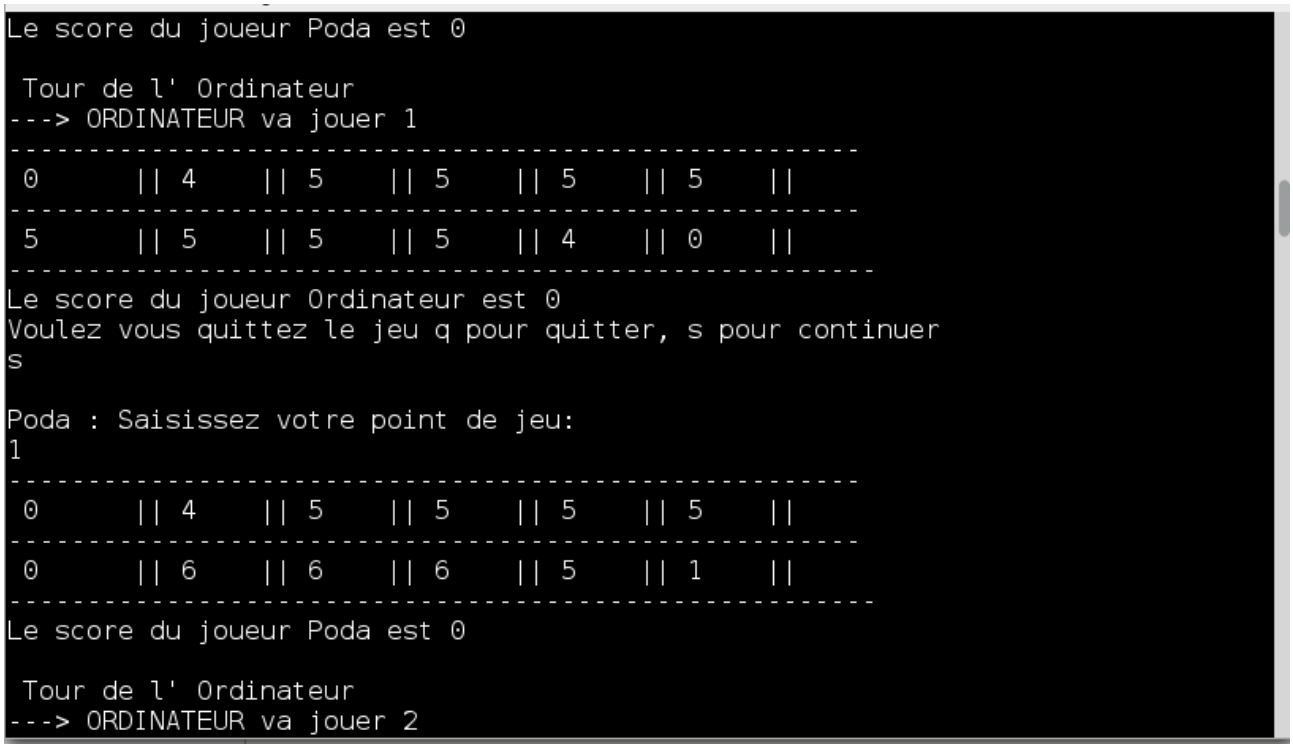


Figure 6 : Image présentant une partie avec l'ordinateur.

VII. Conclusion

Au terme de notre analyse force est de constater que les différents algorithmes et méthodes de programmation vu au cours et nos différentes recherches nous ont permis de bien comprendre les notions pour pouvoir réaliser à bien notre projet .Le projet respecte les objectifs assignes dans notre cahier de charge. Soulignons que d'autres amélioration pouvaient être réalisé telles que : l'intelligence artificielle de l'ordinateur et le développement d'une interface graphique.

Table des matières