Rapport projet S6 : Jeu des Amazons

Yannis Chappet-Juan, César Larragueta
Corentin Perdrizet, Joris Rousere

Avril 2023

FIGURE 1 – Image tirée de la page du projet

Table des matières

1 Introduction

Le Jeu des Amazones est un jeu de duel sur un plateau d'échecs. Dans ce jeu, les pièces sont appelées les "Reines", et elles se déplacent selon les mêmes règles que la Dame aux échecs, c'est-à-dire en lignes droites dans les huit directions cardinales et diagonales. À chaque tour, les joueurs déplacent une Reine et tirent une flèche sur une case accessible depuis sa position d'arrivée. Cette case devient alors définitivement impraticable pour le reste de la partie. Le but est de bloquer les mouvements de l'adversaire tout en conservant les siens. La partie se termine lorsque l'un des joueurs ne peut plus faire aucun mouvement.

Ce jeu est un sujet de recherche important en informatique théorique en raison de sa complexité, par exemple, il est difficile de déterminer le gagnant d'une partie. En conséquence, il est possible de développer différentes heuristiques pour estimer des stratégies optimales. C'est précisément l'objectif de ce projet dans lequel seront implémentés en C deux clients et un serveur qui joueront au jeu des amazones.

2 Description du projet

2.1 Modélisation du jeu

Les plateaux de jeu sur lesquels les parties d'amazones seront jouées dans ce projet seront modélisés par des graphes. Un tel graphe contient autant de sommet qu'il n'y a de cases sur le pleateau, on note *num_vertices* (appelé N dans la suite). Ce graphe est implémenté en machine à l'aide d'une matrice d'adjacence de taille $N * N$, le coefficient (i, j) de cette matrice indique s'il existe une arrête entre les sommets i et j du graphe. Dans notre cas le coefficient (i, j), est à 0 si aucune arête n'existe entre i et j, s'il en existe une le coefficient est un entier qui renseigne sur la direction de la relation qui lie ces deux sommets ensemble, selon l'enum ci-dessous :

```
enum dir_t {
  NO_DIR=0,
  DIR_NORTH=1, DIR_NE=2,
  DIR_EAST=3,  DIR_SE=4,
  DIR_SOUTH=5, DIR_SW=6,
  DIR_WEST=7,  DIR_NW=8,
  FIRST_DIR=DIR_NORTH, LAST_DIR=DIR_NW,
  NUM_DIRS=8,
  DIR_ERROR
};
```

Ces plateaux pourront prendre 4 formes différentes que voici (**Figure 2**) :

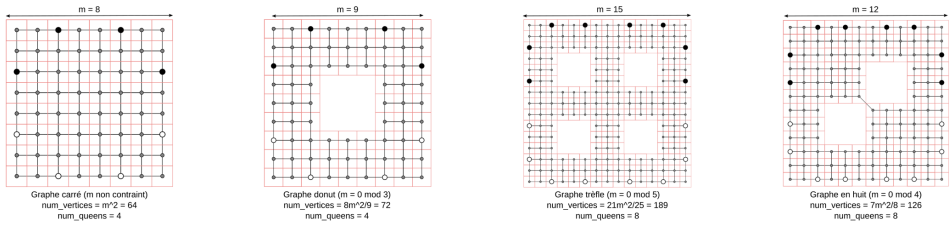


FIGURE 2 – 4 différentes formes de plateaux, tirée du rapport

Sur ces différents plateaux les joueurs disposent de *num_ queens* reines, ce nombre est fixé à $4(m/10+1)$, disposées sur le plateau comme précisé ci dessus en **Figure 2**.

2.2 Architecture client serveur

Lors de la mise en œuvre de ce projet informatique, nous avons fait l'exploration d'une nouvelle approche : l'architecture client/serveur. Nous avons dû nous familiariser avec cette approche modulaire qui, bien que complexe au premier abord, offre de multiples avantages en termes de flexibilité et d'évolutivité. L'utilisation des bibliothèques partagées fournies par dlfcn, et plus particulièrement de fonctions comme dlopen, a nécessité une phase d'apprentissage et de compréhension de notre part. Ayant acquis une certaine aisance avec ces concepts, nous avons adopté une architecture server / client modulaire pour notre projet. Ces bibliothèques partagées agissent comme une interface entre les différentes composantes de notre projet. Cette structure offre plusieurs avantages en termes de flexibilité et d'évolutivité.

2.2.1 Rôle du serveur

Le serveur a pour principale mission de gérer la logique du jeu et de coordonner les actions des clients. Il est responsable de la mise en place de la partie, de la validation des mouvements des joueurs, de la vérification des coups autorisés, ainsi que de la détection d'un gagnant éventuel. Le serveur sert également de point central de communication entre les clients, en transmettant les informations relatives aux mouvements effectués par chaque joueur et en mettant à jour l'état du jeu en conséquence.

2.2.2 Rôle des clients

Les clients, quant à eux, sont principalement chargés de la transmission des actions effectuées par les joueurs au serveur. Les clients reçoivent également les mises à jour du serveur concernant les mouvements des autres joueurs et adaptent leur donnée en conséquence.

2.2.3 Avantages de l'architecture serveur/client

L'utilisation d'une architecture serveur-client dans un projet informatique présente plusieurs avantages. Premièrement, elle facilite la scalabilité : on peut ajouter autant de clients que nécessaire, permettant ainsi de gérer un grand nombre d'utilisateurs simultanément. Deuxièmement, elle favorise la modularité et la réutilisabilité, car chaque client peut être développé et maintenu indépendamment.

Dans le cadre de notre projet, ces avantages se traduisent concrètement par la possibilité de créer plusieurs clients et de les faire interagir entre eux, enrichissant ainsi l'expérience de jeu. De plus, grâce à cette architecture, nous pouvons intégrer facilement des clients développés par d'autres programmeurs, permettant ainsi des affrontements diversifiés et stimulants.

3 Organisation du travail

3.1 Répartition du travail

En ce qui concerne la répartition des tâches au sein du groupe, nous avons adopté une approche collaborative en utilisant la technique du pair programming, où deux programmeurs travaillent simultanément sur un même code. Cette méthode de travail a permis une répartition efficace des

tâches et a favorisé une collaboration étroite entre les membres du groupe, permettant ainsi le partage des connaissances et des idées, et une détection et correction rapides des erreurs.

Ce projet a été une occasion pour nous de découvrir de nouvelles techniques de développement logiciel qui étaient jusqu'alors inconnues pour nous, mais qui sont utilisées dans le monde professionnel. Afin de garantir la cohérence et la lisibilité du code, nous avons convenu d'utiliser la convention de codage *snake_ case* et d'adopter l'anglais pour tous les noms de variables, structures, fonctions etc.

3.2 Outils de travail

Au cours de ce projet, nous avons eu à utiliser la bibliothèque c *GSL (GNU Scientific Library)*, qui fournit un ensemble complet de fonctions et d'outils pour la résolution de problèmes mathématique dans le code relatif aux matrices. L'outil de génération automatique de documentation pour les projets logiciels nous a été utile pour réaliser les graphes de dépendances entre les différents fichiers. *Git* a permis de prevenir les problèmes de conflits de versions du projet entre les différents membres du groupe. Nous avons utilisé DL open, une fonctionnalité de la librairie *dlfcn.h*, afin de charger dynamiquement une bibliothèque partagée, ce qui nous a permis de rendre les clients interopérables avec le serveur en offrant une flexibilité dans l'ajout et la mise à jour de fonctionnalités pendant l'exécution du jeu des amazones.

4 Implémentation du code

4.1 Organisation du code

4.1.1 Arborescense de fichier

- Le repertoire *src/* contient :
  - Les deux fichiers *client1.c* et *client2.c* qui contiennent les fonctions d'initialisation, la fonction play et ses sous fonctions pour chacun des joueurs.
  - Le fichier *graph.c* dans lequel sont implémentés toutes les fonctions relatives à la creation et la libération des graphes utilisés.
  - Dans le fichier *neighbors.c* est seulement codée une fonction qui permet de savoir si un sommet dub graphe admet des sommets voisins libres.
  - Le fichier *server.c* permet de charger les bibliothèques partagées des clients, et contient les fonctions liées au déroulé de la partie
- Répertoire *tst/* contient tous les fichiers qui testent le bon fonctionnement du code relatif au repertoire *src/*.
- Répertoire *install/* contient, après compilation, les bibliothèques partagées liées aux fichiers clients.

4.1.2 Dépendances entre les fichiers

Coté client

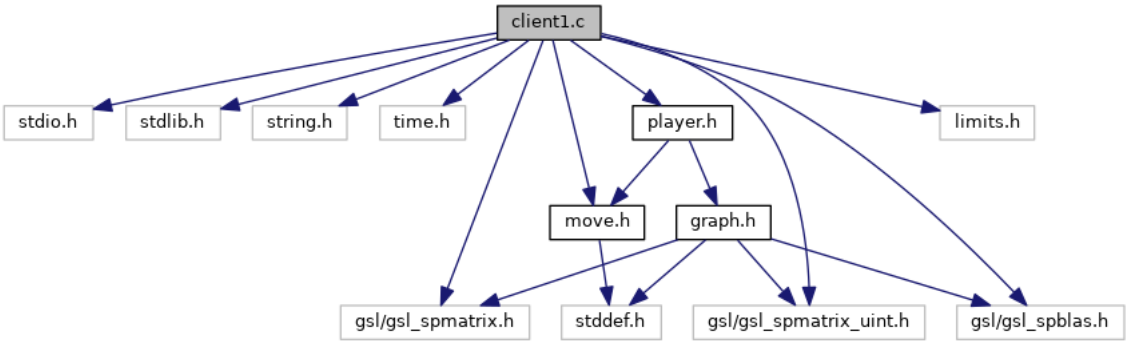


FIGURE 3 – Graphe de dépendances du fichier client

Coté serveur

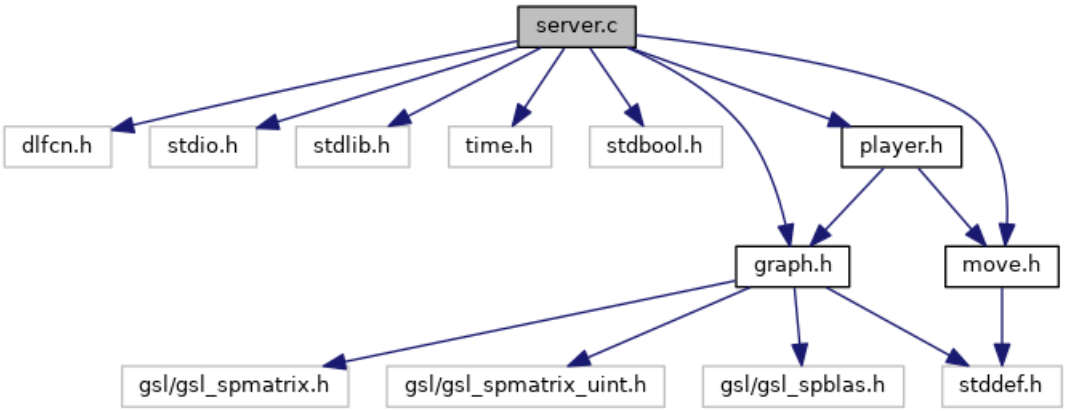


FIGURE 4 – Graphe de dépendances du fichier serveur

4.2 Les différentes structures de données du projet

La définition précise et efficace des structures de données est un facteur essentiel dans tout projet informatique, car elle détermine la manière dont l'information est stockée, organisée et manipulée. Une bonne gestion des structures de données peut grandement contribuer à l'efficacité et à la clarté du code, ce qui facilite à la fois le développement et la maintenance du projet.

Dans le cadre de notre projet du jeu des Amazones, la gestion des structures de données est effectuée de manière parallèle par le serveur et les clients. Chacun de ces acteurs maintient sa propre version de l'ensemble des structures de données, reflétant l'état actuel du jeu. À chaque tour, lorsque un coup est joué, le serveur et les clients mettent à jour leurs structures de données en conséquence. Cette synchronisation constante garantit que toutes les parties impliquées ont une vue cohérente et à jour du jeu, permettant ainsi une interaction fluide et précise entre le serveur et les clients.

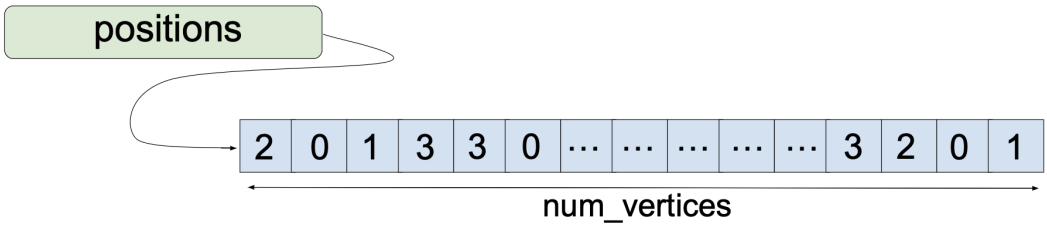
Tout d'abord, certaines constantes sont définies pour faciliter la compréhension du code. Les constantes utilisées sont les suivantes :

— num_vertices : représente le nombre total de cases sur le plateau.
— num_queens : indique le nombre de reines par joueur.
— num_players : représente le nombre total de joueurs.

Pour faciliter l'indexation des tableaux, nous utilisons la convention suivante, on indexe les cases du plateau par ordre croissant de gauche à droite et de haut en bas en commençant par 0, comme le montre l'exemple suivant pour un plateau de taille 8x8 :

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Pour représenter l'état du plateau de jeu à un instant donné, nous utilisons le pointeur vers un tableau appelé positions, de taille num_vertices. Chaque élément de ce tableau représente une case du plateau. La première case est représentée par le premier élément du tableau, la deuxième case par le deuxième élément, et ainsi de suite. Chaque élément du tableau contient une valeur qui peut être soit 0, indiquant que la case est vide, soit 1, indiquant la présence d'une reine alliée, soit 2, indiquant la présence d'une reine ennemie, soit 3, représentant un bloc. Chaque joueur possède un tableau adapté à sa configuration de plateau, où 1 désigne ses propres reines et 2 désigne les reines adverses. Voici un exemple illustrant cette structure :

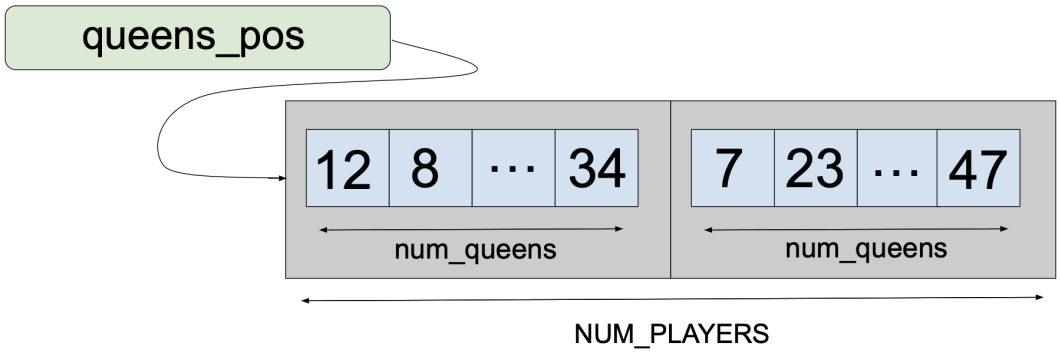


Le plateau de jeu représenté par ce tableau positions présente la configuration suivante : la première case contient une reine appartenant au joueur adverse (l'adverse du joueur qui stocke ce tableau). La deuxième case est vide, tandis que la troisième case abrite une reine du joueur qui a stocké ce tableau. Sur la quatrième case, on trouve un bloc, et ainsi de suite pour les cases suivantes.

Par la suite, nous employons une matrice d'adjacence pour symboliser les relations entre les cases du plateau. Cette structure, qui demeure invariable tout au long de la partie, sert à identifier les mouvements valides pour les reines. Pour sa création et sa manipulation, nous avons recours à la bibliothèque *GSL* (GNU Scientific Library). Cette bibliothèque, largement utilisée dans le domaine scientifique, offre une gamme d'outils et de structures de données robustes pour les calculs

numériques, y compris les matrices. De plus en utilisant *GSL*, nous bénéficions d'une gestion efficace de la mémoire, ce qui contribue à la fiabilité de notre programme.

Enfin, la structure de données queens_pos est utilisée pour obtenir directement les positions des reines. Il s'agit d'un tableau de pointeurs de taille num_players, représentant les joueurs. Pour simplifier, nous considérerons ici une partie avec deux joueurs. Chaque élément du tableau est lui-même un tableau de taille num_queens, contenant les index des reines pour chaque joueur. Par exemple, le premier élément du tableau représente le joueur 1 et contient les index des reines appartenant à ce joueur, tandis que le deuxième élément représente le joueur 2 (ennemi). Cette structure permet d'accéder rapidement aux positions des reines pour chaque joueur.



En résumé, les principales structures de données utilisées dans ce projet sont un tableau position représentant l'état du plateau, une matrice d'adjacence pour les relations entre les cases et un tableau queens_pos permettant d'obtenir directement les positions des reines pour chaque joueur.

4.3 Déroulement d'une partie

Au début de chaque partie, le serveur charge l'ensemble des fonctions nécessaires pour garantir une progression fluide du jeu. Ces fonctions incluent notamment la vérification de la validité d'un coup et l'initialisation des structures de données.

Une fois les structures de données initialisées et les reines correctement positionnées sur le plateau, la partie peut débuter.

C'est d'abord au joueur 1 de commencer. Pour décider du coup à jouer, le système calcule l'ensemble des cases accessibles à la reine. Une case est choisie de manière aléatoire, et la reine y est déplacée. Ce processus est répété une fois la reine sur sa nouvelle case, pour déterminer où elle tirera sa flèche. En pratique, un tableau `possible_cell` de taille `num_vertices` est initialisé à 0 et pour une reine donnée, toutes les directions sont explorées en ligne droite autour de la reine, si une case i est libre, l'élément i du tableau est remplacé par 1. Les cases atteignables par la reine sont donc celles qui sont indexées par 1 dans le tableau.

Le coup est ensuite vérifié par le client, puis transmis au serveur. Le client met à jour ses données pour refléter le nouvel état du jeu.

Le serveur vérifie à son tour la validité du coup, puis le transmet au second client, qui devient alors le joueur actif.

A chaque coup, le serveur vérifie si un gagnant peut être déclaré. La partie continue de cette manière jusqu'à l'identification d'un gagnant.

4.4 Makefile et compilation

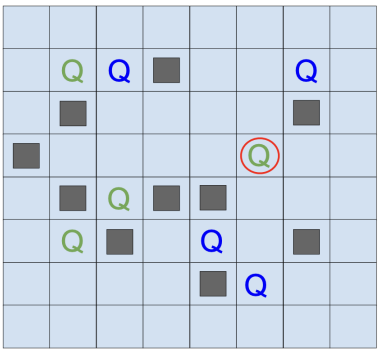
Le Makefile joue un rôle crucial dans la compilation du projet du jeu des amazones, qui suit une architecture client/serveur. Il contient des règles spécifiques pour compiler différentes parties du projet de manière séparée. Il nous permet de compiler le fichier *client.c* en une bibliothèque partagée *client.so* en spécifiant les dépendances nécessaires et les options de compilation appropriées. De plus, il offre la possibilité de compiler les tests unitaires de manière distincte pour assurer la qualité du code, facilitant ainsi le processus de développement et de maintenance du jeu des amazones.

5 Tests de validation

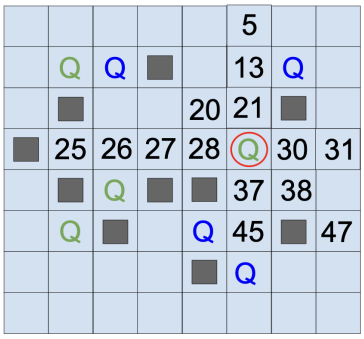
Dans le domaine de la programmation, l'importance des tests ne peut être sous-estimée. Ces derniers constituent un aspect crucial du développement de tout projet informatique. Les tests permettent de vérifier le bon fonctionnement des différentes parties d'un programme, de déceler les erreurs et les défauts, et de s'assurer que le programme répond bien aux exigences et aux attentes. Ils jouent un rôle prépondérant dans l'assurance de la qualité du programme, contribuant à sa fiabilité et sa robustesse.

Dans le cadre de notre projet, nous avons mis en place une batterie de tests pour évaluer divers aspects du programme. Nous avons commencé par effectuer des tests permettant la vérification du calcul des mouvements possibles au sein du jeu, et l'authentification des fonctions responsables de la validation d'un coup permis. Ces tests sont fondamentaux pour garantir l'équité du jeu et la précision des mouvements des reines sur le plateau.

Pour effectuer cela, on se place dans une situation précise et on analyse les calculs permettant de déplacer une reine. Supposons que nous sommes dans la situation suivante et que nous analysons la reine entouré en rouge :



Nous procédons alors à l'analyse de la situation en déterminant à la main quelles cases sont accessibles à la reine concernée. Sur le schéma suivant, nous illustrons les cases accessibles en indiquant leurs indices respectifs :



Après avoir déterminé les indices des cases accessibles, nous remplissons le tableau `possible_cell`, mentionné dans la section 4.3, en utilisant nos fonctions. Nous veillons ensuite à ce que le tableau résultant corresponde bien à nos attentes. Par la suite, nous invoquons notre fonction de vérification de la faisabilité d'un déplacement. De la même manière, nous procédons à des tests pour nous assurer de la fiabilité de cette vérification.

Nous avons également assuré la validation de l'initialisation appropriée des structures de données. Cette démarche s'est déroulée de manière relativement directe. Après avoir configuré un plateau de jeu dans son état de départ attendu, nous vérifions que les reines sont bien positionnées dans le tableau `positions`. Par ailleurs, nous avons contrôlé la bonne initialisation de la matrice d'adjacence, en nous assurant que chaque ligne (représentant chaque case du plateau) contient bien huit relations directionnelles pour les cases centrales, cinq pour celles situées sur les arêtes et trois pour celles situées aux coins.

6 Conclusion

6.1 Limites et potentiel d'évolution du projet

Il est important de mentionner que l'une des limitations du projet est l'absence d'une heuristique spécifique permettant à nos joueurs de jouer au jeu des amazones selon une stratégie particulière. En raison des contraintes de temps, nous n'avons pas pu développer une heuristique sophistiquée. Nos joueurs choisissent donc leurs futurs coups aléatoirement ce qui leur assure pas une place de choix sur le ladder.

6.2 Apports du projet en termes d'apprentissage et d'expérience

Ce projet nous a permis de tirer plusieurs apprentissages précieux. Tout d'abord, nous avons réalisé l'importance d'une planification et d'une gestion efficace du temps. En constatant les contraintes temporelles auxquelles nous étions confrontés, nous avons dû prendre des décisions stratégiques pour prioriser les taches. Cela nous a aidés à développer nos compétences en gestion de projet et à mieux comprendre l'importance de l'organisation et de la répartition des tâches au sein de l'équipe.

De plus, ce projet nous a confrontés aux défis de la collaboration et de la coordination au sein d'une équipe de développement. Nous avons dû partager efficacement les responsabilités, communiquer nos progrès et résoudre les problèmes de manière collaborative. Cela nous a permis d'améliorer nos compétences en travail d'équipe, en communication et en résolution de problèmes.

En outre, nous avons acquis une expérience concrète dans le développement d'applications utilisant une architecture client/serveur. Nous avons appris à concevoir des interfaces entre le client et le serveur, à échanger des données et à coordonner les interactions entre les deux parties.

Dans l'ensemble, ce projet nous a apporté une expérience précieuse en matière de gestion de projet, de collaboration, en nous fournissant des compétences et des connaissances pratiques pour aborder de futurs projets complexes.