

Rapport du projet "Poker" de L2INFO

URL : https://github.com/BouquetTristan/Poker_L2_Project

POUPIN Romain
BOUQUET Tristan
NJOFANG NGASSAM William

6 avril 2018

Table des matières

1	Introduction	2
2	Analyse	2
2.1	Règles du jeu	2
2.2	Cahier des charges	3
3	Organisation du travail	3
3.1	Répartition du travail	3
3.1.1	Romain	3
3.1.2	Tristan	3
3.1.3	William	3
3.2	Méthode de collaboration	4
3.3	Planning prévisionnel	5
3.4	Planning final	6
4	Conception et réalisation	7
4.1	Organisation des fichiers	7
4.2	Codage	7
4.2.1	Structures	7
4.2.2	Primitives	9
4.2.3	Exploitation des structures	10
4.2.4	Déroulement du jeu	12
4.2.5	Déroulement du jeu version William	15
4.2.6	Interface utilisateur	16
4.2.7	Multijoueur en réseau	17
4.2.8	Assistant d'installation/désinstallation	18

5 Conclusion	19
5.1 Existant	19
5.2 Critique	23
5.3 Perspectives d'amélioration	23

1 Introduction

L'objectif de ce projet vise à créer un jeu de poker selon les règles du "Five Draw". Les joueurs, au nombre maximum de cinq, parient leurs jetons et échangent leurs cartes pour chercher à obtenir la meilleure main possible et remporter tous les jetons pour gagner la partie.

2 Analyse

Avant de nous lancer dans l'écriture du code, nous avons commencé par définir en commun les règles et contraintes de notre version de poker, afin que tous les membres du groupent puissent avoir une idée générale du programme et ainsi, éviter les incohérences entre les parties programmées par différentes personnes.

2.1 Règles du jeu

Le poker se joue avec un paquet de 52 cartes et requiert d'obtenir la meilleure combinaison de cartes pour remporter les jetons misés par les joueurs. Celui qui finit par remporter tous les gains à gagné la partie. Dans le cadre de notre projet, nous avons choisis d'implémenter le jeu selon les règles du "Five Draw", une des nombreuses variantes du poker. Elle se distingue des règles classiques par le fait que chaque joueur possède un nombre fixe de 5 cartes qu'il peut échanger au cours de la partie et comporte 3 tours de mise.

En début de partie, le jeu de cartes est mélangé et distribué à chaque joueur. Après que les joueurs aient reçu leurs cinq cartes, le premier tour de mise démarre : le premier joueur commence à miser, les joueurs suivants ont le choix entre suivre, relancer ou se coucher. Ce premier tour s'arrête quand une mise est sans relance. Il peut y avoir autant de relances et de tours de table que l'on veut. Les joueurs encore dans le coup peuvent se débarrasser de quatre cartes maximum et recevoir de nouvelles cartes tirées du haut de la pioche.

Ensuite, on procède au second, puis au troisième et dernier tour de mise selon les même règles que le premier. A l'issue du dernier tour de mise, la meilleure main l'emporte. En cas d'égalité, les joueurs se partagent les jetons misés. Les joueurs ne disposant plus de jetons sont éliminés au fur et à mesure du jeu et le vainqueur est déterminé comme étant le dernier joueur qui détient donc tous les jetons.

2.2 Cahier des charges

D'une part, le jeu devra répondre aux règles exposées ci-dessus, et d'autre part, il devra proposer une interface graphique, permettre de choisir la quantité de jetons de départ pour tous les joueurs en début de partie, saisir le nombre de joueurs et configurer leurs pseudonymes.

3 Organisation du travail

3.1 Répartition du travail

Afin de travailler le plus efficacement possible, nous nous sommes réparties les tâches d'abord en fonction des compétences, puis de la motivation de chacun. Certains se sont retrouvés avec beaucoup de travail en programmation et d'autres à effectuer des tests et séries de débogages principalement sur le déroulement d'une partie.

3.1.1 Romain

- Création et gestion de l'architecture globale du logiciel
- Création des structures qui modélisent les objets nécessaires à manipuler dans le jeu (carte, jeu de cartes, joueur).
- Création des primitives bas niveau (instanciation, modification et destruction des structures).
- Création des fonctions plus haut niveau de manipulation et d'exploitation des structures
- Création de l'interface graphique et gestion des effets sonore en SDL
- Assistant graphique d'installation/désinstallation du jeu et makefile

3.1.2 Tristan

- Création de la fonction destinée à calculer la valeur d'une main
- Test de cette fonction pour prévoir tous les cas d'erreurs
- Création d'un serveur TCP acceptant la connexion de cinq personnes différentes au maximum
- Intégration du serveur et du client dans le déroulement du jeu

3.1.3 William

- Création du déroulement du jeu à l'aide des fonctions de manipulation du jeu et des cartes
- Intégration de la fonction de calcul de la main dans le déroulement du jeu
- Tests et débogages des fonctions du déroulement de jeu

3.2 Méthode de collaboration

Pour collaborer facilement, nous avons utilisé l'outil de gestion de version **Git** qui travaille depuis un dépôt distant hébergé sur **Github**. Nous avions pour consigne de ne jamais envoyer un travail qui contient des erreurs fatales pour ne pas bloquer le reste du programme. Nous avions également commencé à utiliser l'outil de planification des tâches proposé par **Github**, mais nous nous sommes finalement retourné vers une méthode plus classique sous la forme de briefings hebdomadaires en groupe (méthode agile).

Aussi, nous avons privilégié le temps passé en séances de TP pour profiter d'être tous réunis pour pouvoir fusionner le travail plus facilement.

3.3 Planning prévisionnel

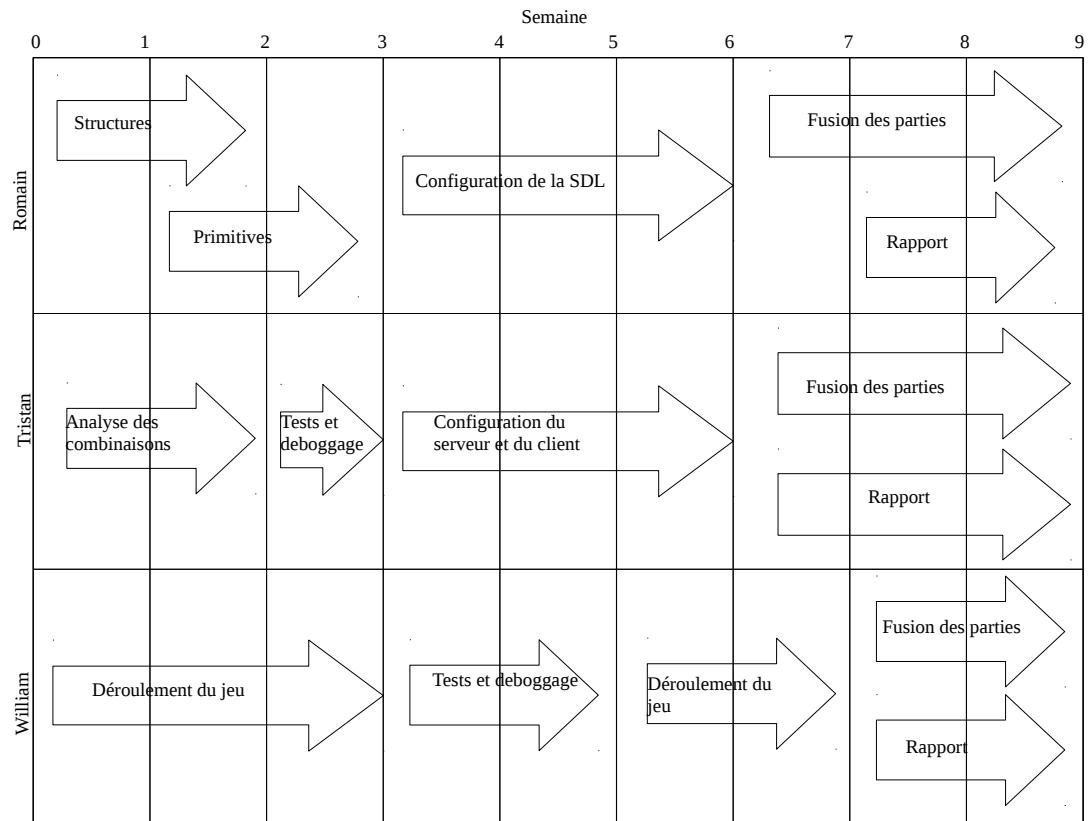


FIGURE 1 – répartition prévisionnelle du travail dans le temps

3.4 Planning final

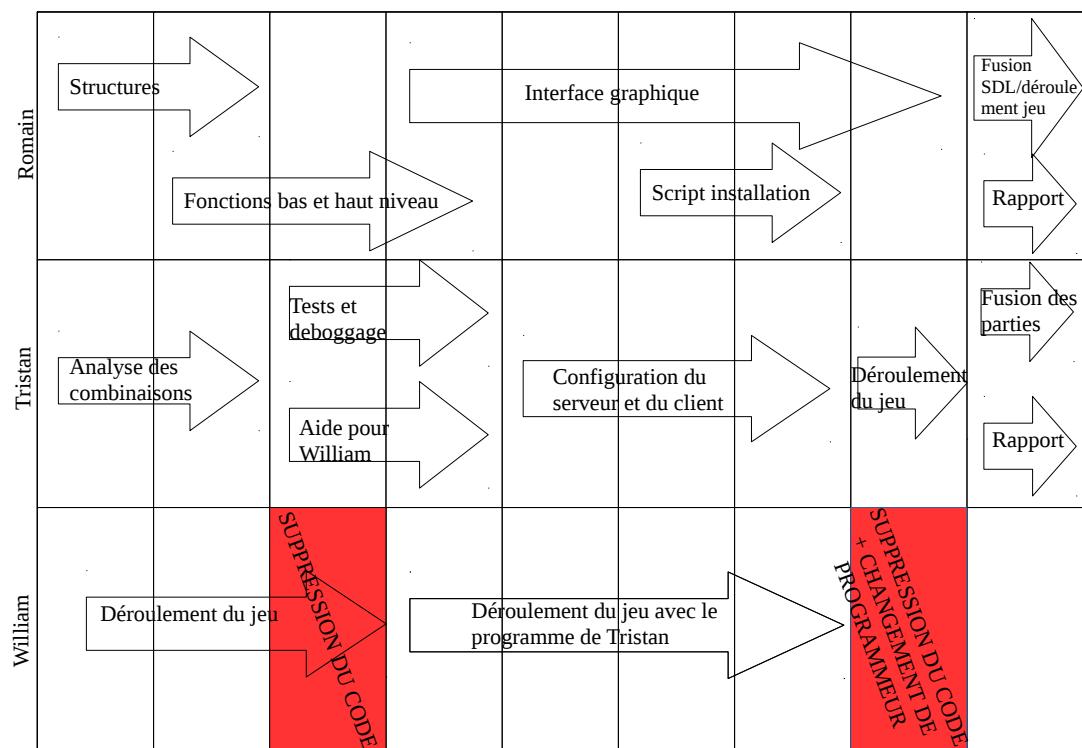


FIGURE 2 – répartition final du travail dans le temps

4 Conception et réalisation

4.1 Organisation des fichiers

Le logiciel est organisé selon l'arborescence suivante :

```
root folder
  └── doc
      └── ..... documentation et autres
  ├── font
  │   └── GODOFWAR.ttf ..... police
  ├── img
  │   └── *.png, *.jpg ..... textures
  ├── include
  │   └── *.h ..... bibliothèques
  ├── sound
  │   └── *.mp3, *.wav ..... audios
  ├── src
  │   └── *.c ..... sources
  ├── LICENSE ..... GPL3
  ├── README.md ..... notice d'utilisation
  ├── install.sh ..... script shell d'installation
  ├── makefile ..... compilation séparée
  ├── poker ..... exécutable
  └── uninstall.sh ..... script shell de désinstallation
```

4.2 Codage

4.2.1 Structures

Les structures suivantes constituent les trois objets fondamentaux manipulés partout dans le logiciel, à savoir, les cartes, le jeu de cartes et les joueurs.

- `carte_t` : structure composée de trois chaînes de caractères pour renseigner la couleur, la hauteur et le propriétaire actuel d'une carte.

attribut	exemple
<code>char * couleur</code>	"trefle"
<code>char * hauteur</code>	"valet"
<code>char * owner</code>	"pioche"

FIGURE 3 – description `carte_t`

- **jeu_t** : il s'agit d'un tableau de N pointeurs de **carte_t**. Grâce à l'attribut **owner** des cartes, il devient donc facile de déterminer l'emplacement de n'importe quelle carte du jeu.

attribut
carte_t * liste[N]

FIGURE 4 – description **jeu_t**

On notera que l'ordre des cartes de la pioche (quand **carte_t.owner = "pioche"** donc) est déterminé par leur indice dans le tableau **liste[N]** : plus l'indice est faible, plus la carte est haute dans la pioche, et inversement. Concernant les autres cartes, l'indice n'a pas de signification.

Ainsi, d'après l'exemple ci-dessous, on déduit que le valet de trèfle est la première carte de la pioche car son propriétaire est "pioche" et qu'elle se situe dans la case d'indice le plus faible du tableau **liste[N]**

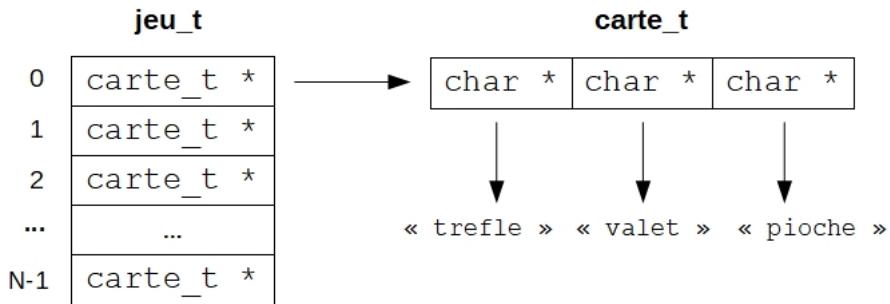


FIGURE 5 – exemple **jeu_t**

- **player_t** : contient les informations du joueur concernant sa cagnotte personnelle, nombre de jetons misés en cours, statut (en jeu ou couché), pseudonyme et sa main qui compte un nombre fixe de 5 cartes.
Main qui est donc modélisée par un tableau de 5 pointeurs de type **carte_t** sur les cartes du paquet dont le propriétaire correspond au pseudo du joueur.

attribut
carte_t * main[5]
int jetons_stock
jetons_mise
char * pseudo
int actif

FIGURE 6 – description player_t

4.2.2 Primitives

Cette partie a pour but de présenter le fonctionnement des procédures de **création** et de **destruction** des structures : il s’agit des briques de base de manipulation des objets **carte_t**, **jeu_t** et **joueur_t**.

Aussi, parce que le mécanisme de ces primitives reste semblable d’une structure à l’autre, nous détaillerons donc d’abord leur fonctionnement type puis détaillerons leurs différences et particularités en fonctions des structures.

— **création :**

entrée : rien

sortie : pointeur sur l’objet créé

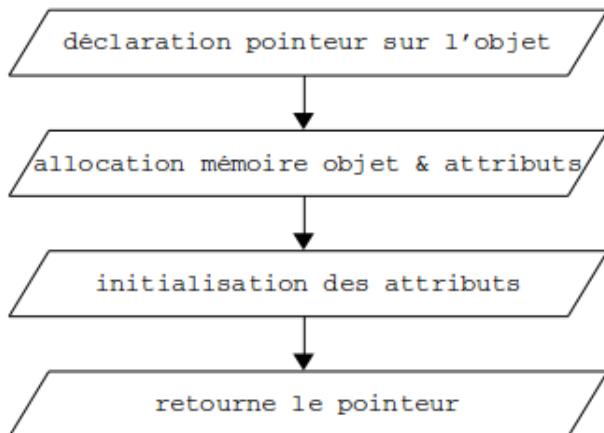


FIGURE 7 – algorithme création objet

— **destruction :**

entrée : pointeur sur l’objet à détruire

sortie : rien

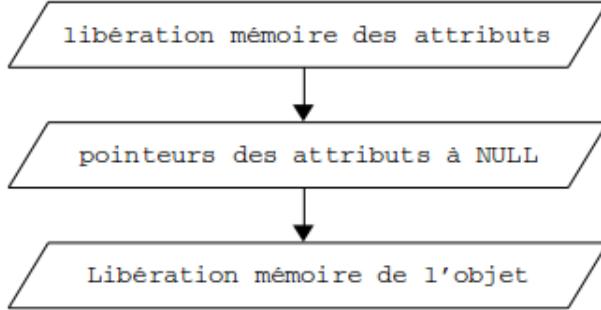


FIGURE 8 – algorithme destruction objet

Particularités et remarques :

- `carte_t` : les cartes sont initialisées dans la fonction `jeu_initialiser` (voir section 4.2.3).
- `jeu_t` : on appelle la primitive `carte_creer` pour toutes les cartes du jeu dans le tableau `liste[N]`, et inversement pour détruire le jeu de cartes à l'aide de `carte_detruire`.
- `joueur_t` : ici, pas besoin d'appeler les primitives de création et de destruction d'une carte pour la main des joueurs, il s'agit uniquement de pointeurs sur les cartes déjà créées dans l'objet `jeu_t`.

4.2.3 Exploitation des structures

Pour faire suite à la partie précédente, nous exposerons ici les fonctions plus avancées de manipulation et d'exploitation des structures. Les fonctions concernant le déroulement du jeu sont abordées dans les sections propres à cette partie du logiciel (voir section 4.2.4 et 4.2.5).

- `int indice_hauteur(carte_t * carte) :`
rôle : retourne l'indice de la case du tableau `tab_hauteur` correspondante à celle de la carte (utilisée plus loin dans les fonctions d'analyse des combinaisons des cartes des joueurs).
- `void jeu_initialiser(jeu_t * jeu)) :`
rôle : initialise la hauteur, la couleur et le propriétaire de toutes les cartes du paquet. "pioche" est le propriétaire par défaut.
- `void carte_echanger(carte_t * carte1, carte_t * carte2)) :`
rôle : échange la position de deux cartes dans le paquet
détails : permute les pointeurs (de type `carte_t`) contenus dans les cases du tableau `jeu_t.liste[N]`.
- `void jeu_melanger(jeu_t * jeu)) :`
rôle : mélange le paquet de cartes.

- détails :** appelle N fois `carte_echanger` avec comme paramètres deux cartes choisis aléatoirement depuis le jeu de cartes à l'aide de `rand()`.
- int carte_distribuer(jeu_t * jeu, player_t * joueur, int i) :**
rôle : distribue une carte (d'indice i) de la pioche vers la main d'un joueur
détails : renseigne le nouveau propriétaire de la carte par le pseudonyme du joueur et cherche un pointeur libre (= `NULL`) dans la main du joueur pour diriger ce pointeur vers celui correspondant à la carte dans `jeu_t.liste[N]`. Si le joueur possède déjà le nombre maximum de 5 cartes en main ou que la carte à distribuer n'appartient pas à la pioche, la carte n'est pas distribuée et le code erreur -1 est retourné.
- int carte_debarasser(jeu_t * jeu, player_t * joueur, int i) :**
rôle : défausse la carte d'indice i de la main du joueur vers la poubelle
détails : fonctionnement inverse à `carte_distribuer`. (met à `NULL` le nouveau pointeur libre dans la main du joueur et remplace le propriétaire de la carte défaussée par "poubelle")
- int top_game_card(jeu_t * jeu) :**
rôle : retourne l'indice de la première carte de la pioche dans le tableau `jeu_t.liste[N]`
détails : parcourt l'intégralité du jeu de cartes et retourne l'indice de la première carte trouvée dont le propriétaire est "pioche". Si la pioche est vide, le code erreur -1 est retourné.
- void main_trier_desc(player_t * joueur) :**
rôle : tri la position des cartes de la main du joueur en fonction de la valeur des cartes (ordre décroissant)
détails : tri requis pour la fonction `main_analyser` et plus ergonomique à l'affichage. Cette fonction associe d'abord une valeur en fonction de la hauteur de chaque carte que possède le joueur, effectue un tri par sélection des valeurs obtenues, puis réorganise la position des pointeurs des cartes de la main du joueur.
- void distribution_cartes(jeu_t * jeu, player_t * joueurs[], int nb_joueurs) :**
rôle : distribue 5 cartes à tous les joueurs et respecte la convention qui veut qu'une seule carte soit distribuée à la fois par joueur
détails : pour chaque carte distribuée, on récupère l'indice de la première carte de la pioche à l'aide de `top_game_card`, la distribue avec `carte_distribuer` puis tri les cartes de la main du joueur avec `main_trier_desc`.

4.2.4 Déroulement du jeu

Le déroulement du jeu est concentré dans les fichiers `mise.c` et `partie.c`. Le premier contient l'ensemble des fonctions qui régissent le déroulement d'un tour de mise tandis que, le deuxième, se place à un niveau au dessus puisqu'il veille au bon déroulement de la partie.

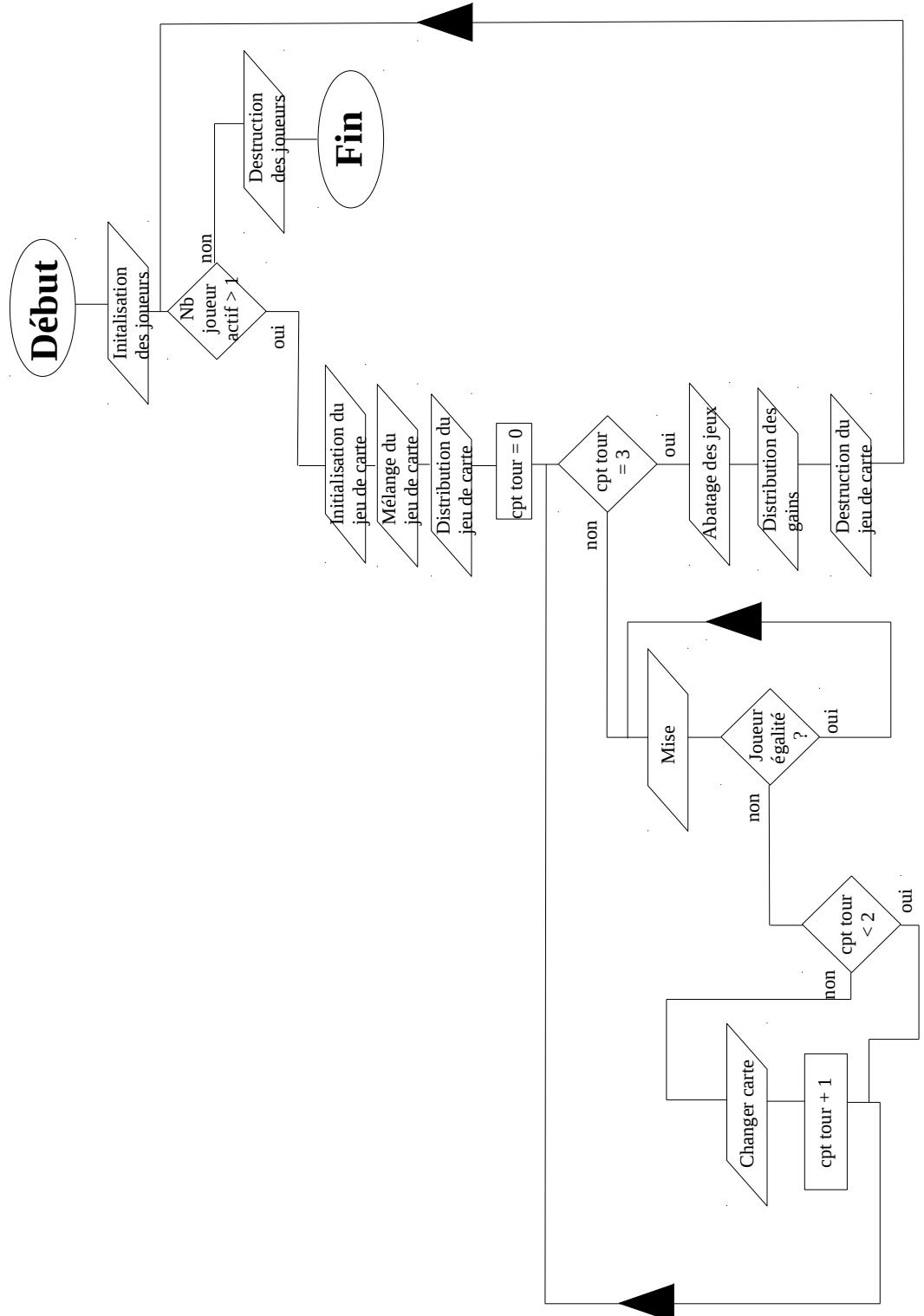


FIGURE 9 – logigramme du déroulement du jeu

Le programme du déroulement d'une partie est construit de façon à exécuter les 3 tours sans sortir. Au commencement, la fonction "turnOfBet" était récursive avec une condition d'arrêt remplie après 3 tours de jeu cependant, l'utilisation de la variable était difficile et j'ai finalement décidé de créer une boucle for vu que je connaissais le nombre de tour et qu'il ne changerait jamais. La fonction fait donc 3 tours et trouve le premier joueur de la liste des joueurs qui est encore debout et utilise la fonction "maj_jetons" pour chaque joueur pour modifier les jetons du joueur.

Cette fonction concentre toutes les modifications des jetons. Elle envoie le joueur vers un menu pour choisir sa mise et récupère un nombre de jeton. Ce nombre de jeton est la mise du joueur et la fonction doit le soustraire au nombre de jeton en stock et l'ajouter au nombre de jeton misé ce tour ci. Pour éviter des problèmes de relance (si un joueur a déjà misé 10 et relance de 20 pour atteindre 30 jetons, il ne doit pas misé 40 mais 30 jetons), on soustrait les jetons déjà misé avec ceux que l'on vient de miser. Enfin, un tour des joueurs est effectué et on ajoute au pot commun le cumul de toutes les mises des joueurs.

Le joueur mise grâce à la fonction "bet" qui, avec un switch, laisse 4 choix à chaque joueur, la différence étant que le premier ne peut pas suivre le précédent. Ses choix sont de "miser", "check", "tapis", "se coucher". Le premier permet de misé un certain nombre de jeton compris entre 0 et le nombre maximum de jeton. Le deuxième permet de misé 0 jeton tandis que le troisième permet de misé tous les jetons du joueur. Le dernier choix fait abandonné la partie pour le joueur, sans lui rendre ses jetons déjà misés, et le laissera revenir pour la prochaine.

Si un joueur a déjà misé avant, le joueur dispose de 2 autres choix à la place de "miser" et "check" pour ceux de "suivre" et "relance". Le choix de suivre récupère le nombre de jeton misé du joueur précédent encore dans la partie pour lui-même. Le choix de relancer utilise aussi cette fonction mais demande un nombre de jeton au joueur compris entre 0 et son nombre maximum de jeton moins le nombre de jeton que le joueur précédent a misé. Cette fonction renvoie un nombre de jeton égale à au prédecesseur du joueur plus un nombre de jeton que le joueur a décidé.

Après un tour des joueurs pour misé, on continue jusqu'à ce que tous les joueurs encore dans la partie (ceux qui ne se sont pas couchés) soient à égalité, après cela les joueurs peuvent échanger leur cartes.

Le déroulement du jeu se poursuit et fait le tour de chaque joueur et si celui-ci est encore en jeu alors la fonction "askCard" est utilisée. Cette dernière demande quelles sont les cartes que le joueur voudrait échanger. L'échange permet de sélectionné entre 0 et 4 cartes en empêchant le joueur de sélectionné deux fois la même carte. Une fois la sélection passée, une autre fonction du nom de "changer_card" est appelée qui va faire jeter les cartes sélectionnées et donner un nombre de nouvelle carte égale au nombre de carte débarrassé.

Le jeu continue et un nouveau tour de mise est effectué, puis un tour pour

changer les cartes de nouveau et enfin, un dernier tour de mise. Après cela, on calcule la valeur de la main de chaque joueur et on range cette valeur ainsi que l'indice du joueur correspondant dans un tableau. Le tableau est trié par ordre décroissant de valeur de main ainsi, le premier du tableau est le joueur possédant la meilleure main. Bien sûr, je dois prendre en compte les égalités entre les joueurs. On crée une boucle qui implémente un indice pour chaque valeur égale à la plus haute valeur du tableau. Enfin, on redistribue les jetons du pot à tous les joueurs possédant la valeur de main la plus élevée en divisant le nombre de jeton donnée à chaque joueur par le nombre de joueur ayant gagné.

Voilà qui conclue le déroulement d'une partie de notre jeu. Cependant, un jeu de poker se compose de beaucoup de partie, pour cela la fonction "partie" permet de lancer les parties.

Cette fonction récupère, depuis le main, la structure de joueur et le nombre de joueur initialisé. Un jeu de 52 cartes est ensuite initialisé puis mélangé et distribuer aux joueurs de la partie. Avant de lancer la partie, une vérification est faite pour savoir s'il reste au moins plus qu'un joueur ayant encore des jetons. Passé cela, la partie est débuté avec les joueurs et le jeu de carte précédemment distribué sinon un message félicite le joueur pour avoir gagné. Une fois La partie finit, le jeu est détruit et la fonction s'appelle elle même avec les même paramètres.

4.2.5 Déroulement du jeu version William

1- Conception et analyse de la mise

Le poker est un jeu de mise. La mise en elle même comporte plusieurs étapes. Ainsi la mise au poker consiste à :

- Suivre (call) : c'est égaliser la mise mise faite par un autre joueur ayant agi et misé avant. La fonction qui la représentera sera la fonction "follow".
- Relancer(raise) : ici il s'agit d'augmenter la main de la mise faite par un autre. L'action consiste à placer suffisamment de jetons pour faire une relance légale en une seule fois. Une relance légale est une relance où vous n'avez pas le droit de mettre des jetons puis de retourner la main à votre stock pour en prendre plus, sauf si vous avez clairement annoncé votre intention en terme de montant avant votre mouvement. La fonction en question est "reflate".
- Tapis(Push) : l'action consiste à pousser tout ce que vous avez en avant de la table. Elle est représentée par la fonction "all_in".
- Coucher(Check) : elle se fait lorsqu'aucune mise n'a été effectuée, le joueur décide alors de passer son tour. L'action au poker peut être faite soit en tapant du doigt ou de la main deux fois sur la table soit en l'annonçant à haute voix. Dans le projet il faudra juste choisir l'option coucher.

2- Répartition des fichiers

Les différents fichiers utilisés sont :

- mise.c : elle possède toutes les fonctions de l'implémentation des fonctions.
- mise.h : il s'agit du point h déclarant les fonctions présentent dans le fichier mise point c.
- joueur.h : contient les primitives nécessaires à la manipulation des données du jeu comme par exemple le nombre de jetons en stock, le nombre de jetons misés, le pseudo ou encore la main de carte.
- poker.c : il permettra grâce à une implémentation des fonctions de la mise d'intégrer le code du fichier mise point c.

3- Implémentation des fonctions

Nous avons huit fonctions principales ; la fonction "follow" qui a pour but d'attribuer au joueur le montant de la mise du joueur précédent. La fonction "reflate", La fonction "reflate" permet de relancer la mise. C'est à dire augmenter la mise. Aussi il faut noter que la fonction prendra en compte la relance en l'ajoutant à la mise précédemment faite. Ensuite la fonction "all_in" quant à elle, donne l'indication que le joueur met tous les jetons qui lui restent sur la table et par conséquent vide ses jetons en stock. Nous poursuivons avec la fonction "sleep" : ici la fonction montre que le joueur passe son tour, ainsi sa mise est de zéro et la main est donnée au joueur suivant. La fonction "egalite" vérifie si les joueurs ont la même mise (même nombre de jetons) et à la suite remet à zéro leur mise pour le prochain tour. La fonction initialisation met le compteur de mise à zéro et le compteur de jetons en stock à cent par exemple. En effet cent est la valeur du nombre de jetons attribuée à chaque joueur. Elle est choisie aléatoirement. Bet est la fonction de mise ; Elle comporte les différents choix liés au jeu et de ce fait comprend les fonctions "follow", "reflate", "all_in" et "sleep". Et enfin la fonction "turnOfBet" cette fonction se distingue par le fait qu'elle possède la fonction initialisation et les fonctionnalités liées au déroulement du jeu.

4- Déroulement du jeu

Dans cette partie, il sera important d'expliquer les fonctionnalités du jeu. Le jeu est initialisé à cinq joueurs et un nombre de jetons de cent chacun. Dans le jeu, il y a trois tours de mise. Chaque tour se termine lorsque chaque joueur a effectué l'une des actions citées précédemment dans la conception et analyse. Chaque joueur prend la main après que l'action du joueur précédent soit terminée ou que l'un des joueurs se soit couché.

4.2.6 Interface utilisateur

Le jeu propose une interface graphique codée en **SDL** (version 1.2.15). Les images sont gérées à l'aide de **SDL_image**, le texte avec **SDL_ttf** et le son grâce

à `SDL_mixer`. Le code de l'interface graphique est séparé du reste pour plusieurs raisons :

- Il est plus facile de vérifier du code s'il est dans une fonction indépendante plutôt que caché dans le code de l'interface graphique.
- Ce code peut être utilisé ailleurs dans le programme
- Indispensable de séparer les calculs de l'interface graphique pour tester indépendamment les fonctionnalités du programme
- Permet d'améliorer la cohérence et la lisibilité du code

un écran de jeu = une fonction = une fenêtre

C'est la règle sur laquelle repose l'interface graphique du jeu. Ce choix de conception permet, d'une part, de faciliter les interactions entre les différents menus puisqu'il suffit simplement d'appeler leurs fonctions respectives pour y accéder, et d'autre part d'assurer l'indépendance de chacun des écrans du jeu. Ci-dessous, le principe de fonctionnement commun à toutes les fenêtres du jeu :

1. déclaration :
 - des surfaces à afficher à l'écran (textures, texte)
 - des polices de caractères utilisées pour le texte
 - des sons dans leurs canaux audio respectifs
2. initialisation :
 - chargement des images et du texte à insérer dans les surfaces
 - de la taille et de la position initiale des surfaces dans la fenêtre
 - chargement des fichiers audio associés aux sons déclarés plus haut et réglages des paramètres de chaque canal audio
3. boucle d'événements :
 - modifie le contenu affiché en fonction de la saisie utilisateur
 - affiche en boucle toutes les surfaces pour rafraîchir l'écran en cas de modification
4. destruction :
 - exécution du traitement prévu en fonction de la condition qui a provoqué la sortie de la boucle événementielle
 - libération mémoire de tous les objets `SDL` créés

4.2.7 Multijoueur en réseau

Pour permettre aux joueurs de jouer au poker sans utiliser le même ordinateur, nous avons décidé d'implémenter une partie réseau qui donne la possibilité aux joueurs de pouvoir jouer sur différent ordinateur. Pour se faire, j'ai utilisé une connexion TCP permettant l'envoie des données aux clients, tout en permettant une connexion multiple sur le serveur. Le serveur possède un tableau capable d'enregistrer 5 adresses différentes, une fois le tableau rempli le serveur rejette toute autre demande de connexion en spécifiant que le serveur est plein. Pour les cinq joueurs connectés, le tableau stocke leur socket pour pouvoir communiquer avec eux.

Le serveur devait suivre le déroulement du jeu et envoyer les informations pour que le programme du client affiche les fenêtres correspondantes comme le menu principal ou encore le rafraîchissement de l'écran de jeu. Le client devait communiquer avec le serveur en envoyant son choix du menu de la partie de déroulement du jeu.

Cependant, suite au retard de la conception de la partie du déroulement de jeu, la partie réseau est mis en pause. Le programme est construit sans utiliser de connexion en réseau pour ne pas provoquer d'erreur dans le programme et l'algorithme du serveur est abandonné.

4.2.8 Assistant d'installation/désinstallation

En plus d'un `makefile`, le jeu propose un script graphique d'installation et de désinstallation automatique. Ce qui permet aux utilisateurs les plus novices d'installer et utiliser le jeu sans taper une seule ligne de commande : une fonctionnalité non négligeable qui répond bien à l'attente de la majorité des utilisateurs d'un jeu vidéo.

L'assistant est découpé en deux scripts `shell` : un pour installer et l'autre pour désinstaller. On notera que celui d'installation requiert une connexion internet et que le gestionnaire de paquet `apt` soit installé sur le système cible dans le cas où le script détecte des paquets manquants à installer. Ci-dessous, les grandes étapes du scripts d'installation :

1. vérifie si le script est lancé dans un terminal (requis pour certaines opérations)
2. vérifie si le jeu n'est pas déjà installé
3. vérification des paquets sdl requis (`dpkg -s`)
4. installation des paquets manquants (`apt`)
5. sélection emplacement de destination (vérification droits écritures)
6. copie des fichiers du jeu, compilation et nettoyage pour ne conserver que l'exécutable et les ressources du jeu (images, sons et textures)
7. ajout raccourci vers bureau par défaut (demande un emplacement personnalisé si introuvable)
8. ajout raccourci du jeu et du script de désinstallation dans le lanceur d'applications du système

Pour plus de détails, voir la vidéo de démonstration avec plusieurs cas d'erreurs testés : <https://www.youtube.com/watch?v=CashnKusGLs>

5 Conclusion

Après 9 semaines de projet, notre sentiment est quelque peu mitigé car le travail réalisé dans certaines parties est très abouti tandis qu'il est largement insuffisant dans d'autres.

5.1 Existant

Le déroulement du jeu fonctionne bien dans la globalité hormis quelques bugs liés au contexte dans lequel il a été développé (changement de programmeur et refonte du code en urgence). Des séries de tests et de débogages sont donc à prévoir pour corriger cette partie du programme. Notamment quand tous les joueurs décident de se coucher par exemple. L'interface graphique est aboutie, soignée et affiche toutes les informations correctement. La gestion du son est parfaitement intégrée à l'interface graphique et permet de diffuser des musiques d'ambiance et des effets sonores simultanément. L'assistant d'installation/désinstallation qui n'était pas prévu fonctionne correctement et couvre une très grande partie des erreurs de manipulation de l'utilisateur. Ainsi, malgré un travail minutieux et acharné, l'existant demeure non fonctionnel, il reste à corriger le déroulement du jeu pour permettre de terminer son intégration dans la couche graphique. Ci-dessous, des captures d'écrans des menus et fonctionnalités du jeu :



FIGURE 10 – menu de sélection du mode de l'écran



FIGURE 11 – menu principal



FIGURE 12 – menu crédits



FIGURE 13 – menu de sélection des règles



FIGURE 14 – menu de sélection des pseudonymes

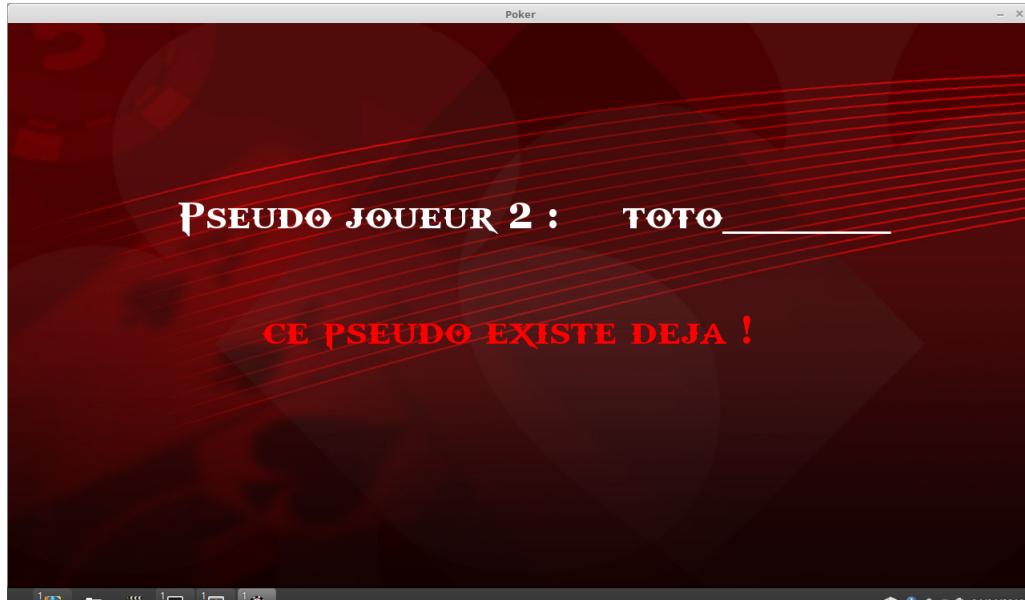


FIGURE 15 – menu de sélection des pseudonymes - cas d'erreur



FIGURE 16 – écran du jeu

5.2 Critique

La répartition du travail que nous avions planifié (voir section 3.1) s'est vue chamboulée vers la dernière phase du projet car le retard accumulé concernant une partie du programme a entraîné un retard global sur l'ensemble du projet. En effet, parce que cette partie n'a pas été réalisée dans les temps et que le code produit était quasi-inexistant, nous étions dans l'obligation de changer de programmeur pour la réécrire. Cet erreur de gestion de projet est la cause principale d'un programme non fonctionnel. Très frustrant donc, car le reste du programme fonctionne très bien et compte zéros fuites mémoires (en oubliant celles causées de façon incontrôlables par la SDL).

5.3 Perspectives d'amélioration

Les idées pour enrichir le projet ne manquent pas. Implémenter un système de jeu en réseau local sur différentes machines, inclure d'autres variantes du poker comme le "texas holdem", ou bien instaurer un système de profil qui permet aux joueurs de consulter leurs statistiques (nombre de parties jouées, perdues et gagnées avec quelles cartes en main et quel score).