Le Mans Université

Licence Informatique *2ème année*

Conduite de Projet

Jeu du Okey

Samuel DURAN
Valentin GIROD
Taner CELIK

GitHub Okey

14 Mai 2019

Table des matières

1 Introduction

Dans le cadre du quatrième semestre de la deuxième année de Licence Informatique, il est demandé de créer un jeu en langage C. Outre les sujets proposés par les enseignants, le choix de notre projet se porte sur le jeu du Okey.

Ce rapport explique le principe du jeu en évoquant son objectif et ses règles. Seront ensuite décrits l'organisation du travail, c'est-à-dire la planification des tâches et l'utilisation d'outils de gestion. Puis, la partie conception donne des éléments sur l'analyse du cahier des charges et dévoile les fonctionnalités associées. La partie développement détaille les méthodes, les algorithmes spécifiques, les principales structures de données, les outils réseaux et l'utilisation de la bibliothèque graphique SDL. Enfin, le rapport indique les résultats obtenus,le bilan du projet et les améliorations suggérées.

1.1 Objectif

Le jeu du Okey est un jeu de réflexion et de hasard, d'origine turque[1]. Il se joue principalement dans des cafés de 2 à 4 joueurs. Celle-ci a plusieurs variantes comme le Rami ou bien le Rummikub, plus connus dans le monde occidental.

FIGURE 1 – Chevalet du jeu Okey

L'Okey comporte 106 tuiles et 4 chevalets qui peuvent accueillir de 14 à 15 tuiles. Chaque tuile est numérotée de 1 à 13 et est parmi l'une des 4 couleurs suivantes : jaune, rouge, noire ou bleu. Ces tuiles sont présentes en 2 séries et les 2 dernières tuiles restantes sont des jokers. Le plateau du jeu dispose

de 4 piles proches des joueurs et au centre, une pioche et un emplacement qui est la tuile Okey.

1.2 Règles du jeu

Lors de la distribution, les 4 chevalets reçoivent 14 tuiles chacune. Une tuile est alors prise dans la pioche et celle-ci désigne le Okey, qui représente une tuile qui remplace la valeur des jokers par sa valeur + 1 et sa couleur. Une tuile supplémentaire est donnée à l'un des 4 joueurs qui lui permet de commencer la partie, celui-ci doit alors faire son choix et défausser une tuile pour que le joueur suivant puisse jouer.

Le sens du jeu est anti-horaire. Le déroulement d'un tour est défini comme suit : le joueur pioche ou prend une tuile de la pile à sa gauche puis défausse une de ses tuiles sur la pile à sa droite.

L'objectif du jeu est d'obtenir un chevalet comportant 14 tuiles appartenant à <une combinaison de règles. On dit alors que c'est un chevalet gagnant.
Pour obtenir ce chevalet gagnant, les tuiles doivent appartenir à l'une des deux combinaisons suivantes :

- soit une combinaison de série de couleurs : il faut avoir dans ce cas une série de 3 ou 4 couleurs différentes d'un même chiffre
- soit une combinaison de suite d'entiers : il faut avoir dans ce cas une suite de 3 ou plus d'entiers de même couleur, sachant qu'il y a la possibilité de boucler les entiers (exemple : la combinaison décrite comme "12/13/1" est valide).

Il faut alors éviter d'accueillir de mauvaises tuiles et d'essayer de combler des tuiles seules[2].

Chaque jeu d'Okey laisse aux joueurs le libre choix de son système de points. Pour ce projet, le choix est simple : si l'un des joueurs remporte une partie, il gagne 1 point et les autres 0 point.

2 Organisation du travail

Initialement, les membres du projet se sont répartis les tâches suivant 3 grands axes :

- le back-end, qui concerne le développement de toute la structure et méthode logique du jeu

— le front-end, qui concerne les affichages graphiques et sous terminal
— la partie réseau, qui concerne la communication entre deux machines distantes

2.1 Répartition des tâches

Le groupe est constitué de trois personnes dont chacun s'est convié une principale tâche au début du projet : Samuel DURAN s'occupe du back-end, Taner CELIK du front-end et Valentin GIROD du réseau. Cette décision intervenait au moment où les membres du projet n'avaient encore jamais pris ces responsabilités et qu'ils voulaient apprendre et pratiquer dans ces domaines. Chacun participait également à la réalisation des tests des programmes codés par les autres.

Au fur et à mesure de l'avancement du projet, la progression des tâches principales se heurtait à des freins organisationnels. Le choix de la méthode de travail a été changé à partir de ce moment. Il se basait alors sur l'accomplissement de missions prédéfinies que les membres du groupe pouvaient choisir. C'était un travail basé sur un planning micro-prévisionnel dont toute la progression est disponible sur les outils de gestion en ligne.

2.2 Outils de gestion

Plusieurs outils de gestion de projet ont été utilisés :

— **le planning micro-prévisionnel**
Trello a été le choix pour mettre en place ce planning. Il était pratique pour définir les tâches à faire, celles en cours et celles qui ont été terminées, tout en indiquant la date limite imposée par chaque membre du groupe et l'historique des actions.

— **la vue macro-prévisionnelle**
Le projet possède une vue macroscopique sous forme de diagramme de Gantt, disponible sur le site onedrive.live.com. Elle décrit l'avancement des principaux jalons, l'affectation des tâches d'origine, la catégorie, la date de début et le nombre de jours écoulés. Une description de couleur accompagnée de légende est présente sur l'encadré correspondant à un calendrier sous fenêtre de 56 jours.

— **la communication**
La communication se fait à travers l'application Discord entre les membres du projet et auprès de notre tuteur sur Mattermost.

Le dépôt de l'ensemble du travail du projet Okey, qui inclut le code des développements réalisés, les dossiers, les fichiers et leurs descriptions se trouve

sur Github.

3 Conception

La conception du projet découle de contraintes détaillées par un cahier des charges. Les fonctionnalités présentent les programmes utilisables dans la branche mère (master) et le dossier Test du jeu.

3.1 Le cahier des charges

Pour la réalisation du jeu du Okey, des contraintes sont définies par le projet. Tout d'abord, il faut que le programme principal du jeu fonctionne sous un terminal. Ajouté à cela, il faut avoir la possibilité d'afficher et de jouer au jeu sous ce terminal avec une interface graphique, ce qui nécessitait l'utilisation des librairies SDL. Ensuite, le jeu peut comporter au minimum un mode de jeu. Il faut qu'un joueur puisse jouer de manière autonome sans erreur avec des joueurs IA. La dernière charge est allouée au réseau : le programme doit posséder des fonctionnalités pour pouvoir connecter deux machines entre elles et pouvoir démarrer une partie.

3.2 Fonctionnalités

L'Okey comporte 3 modes de jeu :
— un mode de jeu avec 4 joueurs
— un mode de jeu avec 1 joueur et 3 IA sur une machine
— un mode de jeu avec 2 joueurs (jouant sur 2 ordinateurs différents) et 2 IA en réseau

Le joueur peut effectuer une nouvelle partie en créant une sauvegarde qui comporte les scores des joueurs. Il peut également charger une sauvegarde à partir d'un fichier existant.

Durant une partie, le joueur peut défausser une tuile de son chevalet, choisir une tuile à partir de la pioche ou de la pile à sa gauche. À la fin d'un tour, le jeu propose de pouvoir ranger les tuiles d'un chevalet. Le joueur peut alors ranger ses tuiles soit par un tri rapide, soit un tri manuel, tout en ayant la possibilité d'annuler l'action de rangement. Une consultation des règles est disponible sur une rubrique du menu principal.

Le dossier Test est constitué de programmes qui permettent de tester si les fonctions sont opérationnelles. Celles-ci sont par la suite récupérées et

utilisées dans la branche principale.

4 Développement

Le développement du jeu est décomposé comme suit : la branche principale comporte tous les dossiers, excepté le dossier Test, qui appartient à la construction du jeu. L'interface graphique se trouve dans le dossier Affichage. La gestion de l'IA se trouve dans le dossier Jeu et le réseau dans le dossier éponyme. Voici l'arborescence simple du projet :

- Affichage
- Initialisation
- Jeu
- Menu
- Règle
- Réseau
- Sauvegarde
- Test

Le dossier Initialisation comporte les différents éléments de l'initialisation d'une partie : le chevalet et la distribution des tuiles. Le dossier Menu contient les différents modes de jeu qu'offre le programme. Le dossier Règle regroupe les différentes combinaisons de règles possibles pour valider un ensemble de tuiles et la sélection de celles-ci. Enfin, le dossier Sauvegarde comporte les fonctionnalités de création et de chargement de sauvegardes de jeu.

4.1 Gestion du code

Pour pouvoir développer le jeu du Okey, il est nécessaire d'avoir des structures de données qui permettent de représenter la tuile, la couleur de la tuile, l'élément d'une pile, une pile de tuiles et le fichier de sauvegarde.

```c
typedef enum {jaune,rouge,noire,bleu} t_couleur;

typedef struct s_tuile {
  int nbr;
  t_couleur clr;
} t_tuile;

typedef struct s_element {
  t_tuile * tuile;
  struct s_element * suivant;

typedef struct s_pile {
  t_element * premier ;
} t_pile;

typedef struct s_fichier {
  FILE * fichier;
  char * nom; /
} t_fichier;
```

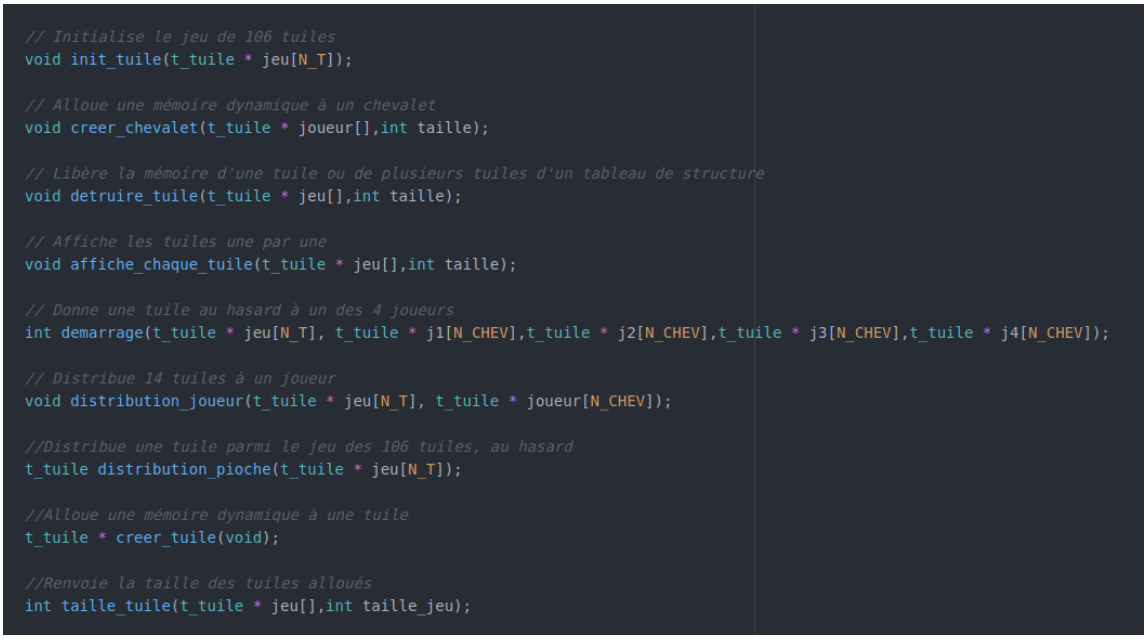
FIGURE 2 – Les structures de données

Il est tout autant utile d'avoir des variables globales qui définissent notamment le nombre de tuiles du jeu, la capacité maximale d'un chevalet, la valeur d'un Okey, la valeur d'une tuile supprimée et une tuile considérée sans valeur.

```c
// Nombre de tuiles d'un chevalet
#define N_T 106

// Nombre de tuiles d'un chevalet
#define N_CHEV 15

// Valeur d'une tuile de okey alias joker
#define V_OKEY 13

// Valeur d'une tuile supprimée
#define V_DEL 100

// Valeur tampon qui sert pour la fonction combinaison suite entier
#define NO_VALUE 99
```

FIGURE 3 – Les variables globales

La suite du développement se base sur des fonctions d'initialisation des structures pour former les premières édifices de la préparation du jeu. Il existe également des fonctions manipulant les tours des joueurs ou des joueurs IA et l'affichage des tuiles du chevalet.



```c
// Initialise le jeu de 106 tuiles
void init_tuile(t_tuile * jeu[N_T]);

// Alloue une mémoire dynamique à un chevalet
void creer_chevalet(t_tuile * joueur[],int taille);

// Libère la mémoire d'une tuile ou de plusieurs tuiles d'un tableau de structure
void detruire_tuile(t_tuile * jeu[],int taille);

// Affiche les tuiles une par une
void affiche_chaque_tuile(t_tuile * jeu[],int taille);

// Donne une tuile au hasard à un des 4 joueurs
int demarrage(t_tuile * jeu[N_T], t_tuile * j1[N_CHEV],t_tuile * j2[N_CHEV],t_tuile * j3[N_CHEV],t_tuile * j4[N_CHEV]);

// Distribue 14 tuiles à un joueur
void distribution_joueur(t_tuile * jeu[N_T], t_tuile * joueur[N_CHEV]);

//Distribue une tuile parmi le jeu des 106 tuiles, au hasard
t_tuile distribution_pioche(t_tuile * jeu[N_T]);

//Alloue une mémoire dynamique à une tuile
t_tuile * creer_tuile(void);

//Renvoie la taille des tuiles alloués
int taille_tuile(t_tuile * jeu[],int taille_jeu);
```

FIGURE 4 – Les fonctions de structure

Les fonctions de piles sont également des structures de données non négligeables, qui permettent de gérer les éléments, c'est-à-dire les tuiles d'une pile. Elles sont créées notamment avec l'aide du cours d'algorithme du semestre 3.

```c
// Initialise la pile
void init_pile(t_pile * pile);

// Vérifie si la pile est vide
int pile_vide(t_pile * pile);

//Empile une tuile de la pile
void empiler(t_pile * pile,t_tuile * t);

//void depiler(t_pile * pile)
void depiler(t_pile * pile);

//Récupère la tuile du sommet de la pile
void sommet_pile(t_pile * pile,t_tuile ** c);

//Compte le nombre d'élément d'une tuile
int compte_element(t_pile * pile);
```

FIGURE 5 – Les fonctions de piles

Finalement, les fonctions de combinaison gèrent la résolution d'un che-

valet de 14 tuiles pour vérifier que toutes ces tuiles appartiennent à une combinaison de couleurs ou bien de suite d'entiers.

```
//Vérifie les ensembles de combinaisons à partir de 3 suites d'entiers
void combinaison_suite(t_tuile * chev[]);

// Vérifie les ensembles de combinaisons de 3 ou 4 couleurs
void combinaison_coul(t_tuile * chev[]);
```

FIGURE 6 – Les fonctions de combinaison

La méthode de développement est de tester des programmes utilisant en majorité les fonctions ci-dessus. Le prototype de chaque fonction est défini dans le dossier Test. Quand celle-ci est valide, c'est-à-dire quand le programme réalise son objectif et qu'il n'y ait aucune mauvaise gestion de la mémoire ou d'autres erreurs, les fonctions sont alors incorporées dans la branche principale, dans leur dossier respectif.

10 programmes sont présents dans le dossier test et un *make all* est nécessaire pour obtenir leurs exécutables. Voici les programmes :

- **Tests des structures d'éléments**
  Ce premier programme permet de tester la création des 106 tuiles du jeu et de les afficher. Il est appelé *prog_structure*.
- **Tests de la pile**
  Ce programme permet de tester l'empilement et le dépilement d'une tuile, ainsi que la récupération du sommet de la pile et l'affichage de sa valeur. Il est appelé *prog_pile*.
- **Tests d'affichage de la tuile**
  Ce programme permet l'affichage des 15 tuiles d'un tableau de tuiles. Il est appelé *prog_affichage_tuile*.
- **Tests de la sélection de tuiles**
  Ce programme permet d'effectuer le déroulement d'un tour, c'est-à-dire le retrait d'une tuile et le choix d'une autre tuile à partir de la pioche ou d'une pile de tuiles. Il est appelé *prog_selection_tuile*.
- **Tests de la combinaison de couleurs**
  Ce programme permet de tester une série de 3 ou 4 tuiles de couleurs différentes à partir d'un chevalet et de vérifier la suppression d'une tuile de la série. Il est appelé *prog_combinaison_couleur*.
- **Tests de la combinaison de suites d'entiers**
  Ce programme permet de tester une suite de 3 entiers minimum d'une même couleur de tuile, il est appelé *prog_combinaison_suite_entiers*.
- **Tests de la partie du jeu**
  Ce programme permet de tester le déroulement d'une partie avec 4

joueurs à tour de rôle sur une même machine, il est appelé *prog_ partie*.

— **Tests de la partie avec l'IA**
Ce programme permet de tester le déroulement d'une partie avec un joueur et 3 autres joueurs IA sur une même machine à tour de rôle. Il est appelé *prog_ partie_ IA*.

— **Tests de la sauvegarde du jeu**
Ce programme permet de tester la création et le chargement d'une sauvegarde via un fichier local. Il est appelé *prog_ sauvegarde*. Le nom du fichier est défini au format *ANNEE-MOIS-JOUR-MINUTES-SECONDES.txt*. Exemple : 2020-05-04-15-53.txt.

— **Tests de la connexion client-réseau (jeu en réseau)**
Ce programme non fonctionnel devait tester le déroulement de la partie entre une machine cliente et serveur. A l'heure actuelle, seule la connexion entre le client et le serveur est opérationnelle. Ils sont appelés *prog_ client* et *prog_ serveur*.

Un exemple de débogage est présent dans le dossier Test, celui-ci concerne la recherche d'un bug (résolu) dans la fonction de combinaison de suite d'entiers, quand un chevalet possède 3 ou 4 fois la même tuile, ce qui inclus les tuiles jokers, la fonction n'arrive pas à vérifié ces combinaisons de suite d'entier.

4.2 Interface graphique

Concernant l'affichage via les librairies SDL, il est découpé en deux parties : un fichier *fonctions_ sdl.c*, qui contient les fonctions plutôt basiques qui vont être utilisées par la quasi-totalité des fonctions du deuxième fichier :

— une fonction *drawImage* qui affiche une image d'une taille donnée à des coordonnées.
— une fonction *loadImages* qui charge toutes les images d'un répertoire et les met dans un tableau de textures, pour les garder en mémoire et les utiliser dans *drawImage*.
— une fonction *init_ affichage* qui ouvre une fenêtre en plein écran, enregistre la taille de l'écran, charge la police de caractères et lance le chargement des images.
— une fonction *drawText* qui écrit du texte donné, d'une taille donnée à des coordonnées données.
— une fonction *unloadImages* qui supprime les textures des images.
— une fonction *fond_ blanc* qui fait un fond blanc (ayant besoin d'un fond blanc, pour le menu de sélection).
— une fonction *quitter_ affichage* pour quitter la SDL.

Le fichier *affichage_sdl.c* est le second fichier contenant les fonctions d'affichage et de gestion des menus, et qui utilise les fonctions du premier fichier. Ces fonctions sont les suivantes :

— *afficher_sauvegarde* permet d'accéder au menu de chargement de partie, et de lister toutes les sauvegardes (ce menu peut afficher BEAUCOUP de sauvegardes).

FIGURE 7 – Affiche les sauvegardes

— *tuileVersImage* prend en paramètre une tuile et renvoie une chaîne de caractère correspondant au nom de l'image liée à la tuile.

— *affiche_chevalet* et *affiche_chevalet_sans_rendu* prennent en paramètre un chevalet et l'affichent, l'une en mettant à jour l'écran, et l'autre sans mise à jour de l'écran.

— *select_tuile* prend en paramètre un booléen qui autorise ou non d'appuyer sur le bouton de fin de tour, et qui détecte les clics sur les tuiles pour renvoyer le numéro de la tuile cliquée. Cette fonction est utilisée pour sélectionner une tuile à jeter mais aussi pour sélectionner une tuile à changer de place, ce qui explique la nécessité du booléen.

FIGURE 8 – Défausse une tuile

— *select_tuile_to_replace* prend en paramètre la tuile sélectionnée avec *select_tuile* et le chevalet. La fonction affiche le chevalet et la tuile en main jusqu'à ce que l'utilisateur clique sur la tuile à échanger. La fonction renvoie le numéro de la tuile à échanger.

— *selection_tuile* prend en paramètre toutes les piles y compris la pioche, le chevalet du joueur, et la tuile joker. Cette fonction attend un clic soit sur la pioche, soit sur la pile à gauche du joueur, et ajoute une

tuile dans le chevalet du joueur en conséquence, puis attend un clic sur une des tuiles du chevalet pour la jeter.

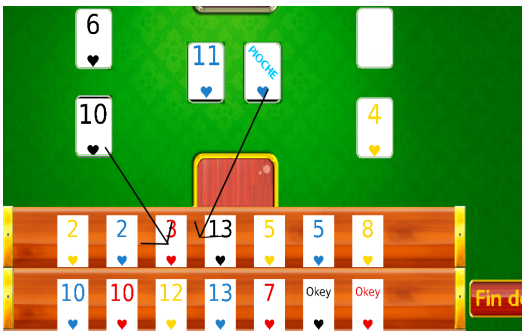


FIGURE 9 – Sélectionne une tuile

— *afficher_menu* prend en paramètre un tableau de chaînes de caractères et affiche un menu à l'écran, en conséquence, elle renvoie le numéro du bouton cliqué.
— *choixtri* utilise *select_tuile* et *select_tuile_to_replace* pour trier les tuiles à la main. En terminal, cette fonction permet de choisir vraiment entre un tri manuel et un tri rapide.
— *affiche_sommet_pile* prend en paramètre une pile et son numéro et l'affiche sur le plateau.
— *affiche_piles* prend en paramètre les piles, la pioche et la tuile joker, et les affiche.

FIGURE 10 – Affiche les piles, le Okey et la pioche

Les fonctions du fichier *affichage_sdl.c* vont être directement utilisées dans les fichiers qui gèrent le déroulement du jeu, tandis que celles du premier fichier ne sont que des outils de base, utilisées par le deuxième fichier ou bien au lancement et à la fin du jeu.

Les fichiers qui gèrent le déroulement du jeu sont indépendants de l'affichage car ils ne font qu'utiliser les fonctions d'affichage des fichiers décrits

plus haut. Ils sont compilés seulement après l'affichage, et cette indépendance de l'affichage a pu être testée en créant une version terminale du jeu avec le même fichier d'en-tête que les fonctions d'affichage SDL, mais des corps de fonctions différentes. Cela permet de compiler la version terminale et SDL d'en changer quoi que ce soit à part la commande du makefile (*make sdl* ou *make terminal*) et en utilisant les mêmes fichiers de jeu.

Dans les fichiers de gestion du déroulement du jeu en SDL (ou en version terminale avec *ncurses*), on a donc un fichier *main.c*, qui utilise les fonctions d'initialisation de l'affichage du menu.

Dans le fichier *menuprincipal.c*, on utilise *fondblanc*, on crée un tableau de chaînes de caractères qui contient le texte à afficher pour chaque bouton, puis on utilise *affichermenu* avec ce tableau. Selon ce que renvoie *affichermenu*, la fonction peut :

— démarrer la partie en choisissant de jouer avec des joueurs ou des IA ;
— charger une partie ;
— consulter les règles du OKey ;
— ou quitter la partie et l'affichage.

FIGURE 11 – Le menu

4.3 Intelligence Artificielle

La programmation de l'IA se base sur le choix qu'elle porte pour pouvoir défausser une tuile mais aussi le choix de retirer une tuile de la pioche ou de la pile à sa gauche.

La logique est la suivante : si l'IA possède 15 tuiles, elle défausse la tuile

ayant le moins de combinaison possible. Elle choisit alors la prochaine tuile en copiant la tuile de la pile à sa gauche. Si celle-ci obtient un nombre de combinaisons important (minimum 0 et maximum 14) et que d'autres tuiles n'en ont pas beaucoup, alors elle choisit cette tuile et la tuile ayant le moins de combinaison est délaissée. Si plusieurs tuiles ayant le moins de combinaison existent, l'IA en choisit une au hasard.

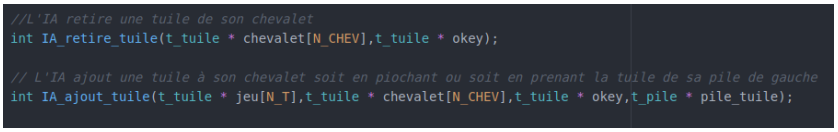


FIGURE 12 – Les fonctions de combinaison

4.4 Réseau

La partie Réseau a pour objectif de pouvoir jouer une partie en LAN (Local Area Network) entre deux utilisateurs. L'un des utilisateurs hébergera la partie tandis que l'autre sera le client. Les machines doivent appartenir aux mêmes adresses réseau. La donnée communiquée entre les machines est un entier qui correspond à une tuile existante, supprimée, sans valeur ou joker.

Le réseau en C fonctionne à l'aide de la librairie *socket* dont on utilise le protocole TCP pour maintenir la donnée à destination. Autrement dit, il est plus fiable que le protocole UDP pour le transport de paquets. D'ailleurs, les tutoriels disponibles sur UMTICE ainsi que les ressources pour apprendre la librairie optent pour TCP, ce qui a influencé sur le choix de la communication entre machines. Le programme réseau fonctionne à partir d'un menu qui opère différents choix suivant la figure ci-dessous :

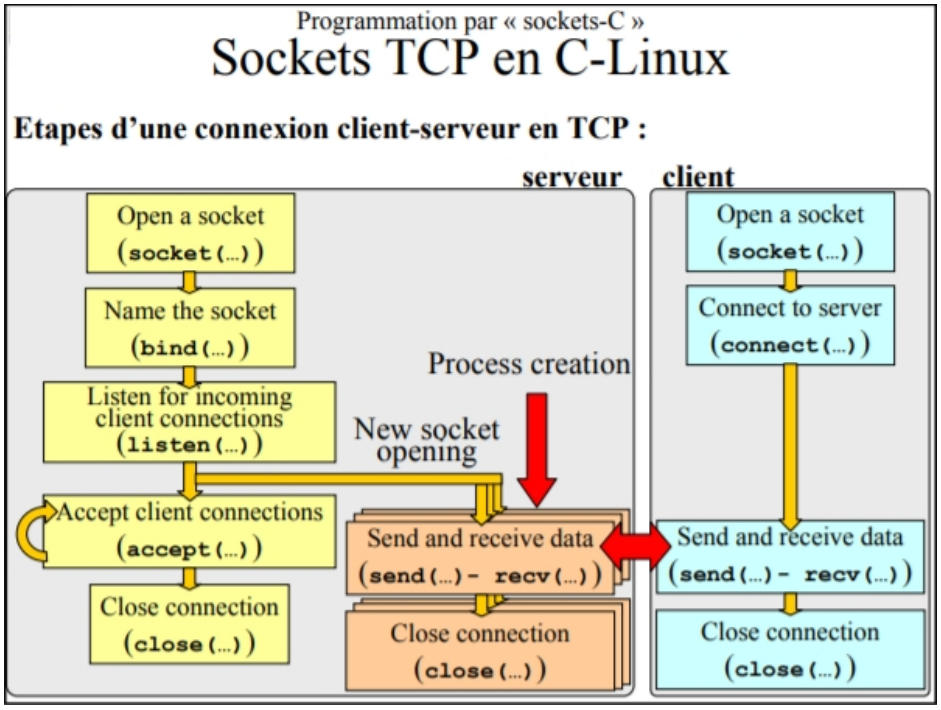


FIGURE 13 – Représentation du fonctionnement de la socket TCP. .

Tout d'abord, on définit si la machine est cliente ou serveur. S'il prend le rôle de serveur, on doit saisir son adresse IP et il en va de même pour la machine cliente. Un numéro de port doit être réquisitionné, devant être au-dessus de 1024 sans prendre un numéro déjà occupé. Par défaut, le numéro est 15000. Le socket du serveur se lie avec son adresse IP et son port et passe en écoute, dans l'attente de la connexion du client. Les opérations comme *send*, *recv*, *shutdown* et *close* seront rappelées par la suite.

La gestion de ces flux de données étant une impasse, le mode de jeu en réseau n'est pas fonctionnelle.

5 Conclusion

Dans cette partie, les résultats des programmes et de l'avancement du jeu seront exprimés ainsi que les difficultés rencontrées et les améliorations possibles.

5.1 Résultats

Dans la globalité du projet, le jeu sous terminal et sous SDL sont fonctionnels, c'est-à-dire la possibilité d'utiliser les modes de jeu avec 4 joueurs différents ainsi que le mode 1 joueur et 3 joueurs IA. Seule la partie du jeu en réseau ne fonctionne pas, mise à part la connexion et la déconnexion entre une machine cliente et une machine serveur.

Le choix des tuiles effectué par le joueur IA est bon en début de partie, mais en fin de jeu, il a du mal à choisir les tuiles à défausser : il manque notamment à effectuer une fonction qui lui donne un meilleur choix.

5.2 Difficultés rencontrées

Notre groupe a rencontré beaucoup de difficultés, notamment sur la réalisation des tâches à faire et à respecter sur un délai. La répartition de la charge de travail est largement inégale et le souci de matériel pour effectuer le réseau pose problème. D'ailleurs, plus le jeu avance, et plus il est nécessaire d'avoir un testeur pour gagner du temps.

5.3 Améliorations possibles

Dans une situation où le jeu serait complètement fini, il est possible d'améliorer l'IA en profondeur, c'est-à-dire d'utiliser complètement tout le jeu. Pour cela, il faut que les 3 IA communiquent ensemble pour établir une stratégie, et ainsi définir par exemple quelle serait l'IA ayant le plus de chance de gagner. Si l'une des IA a le plus de chance de gagner, alors les autres IA l'aideront. Par exemple, ils peuvent aider en donnant les tuiles nécessaires pour avoir un chevalet gagnant.

D'ailleurs, il faudrait que la distribution des tuiles fonctionne en mélangeant les tuiles et en donnant la tuile du dessus de la pioche au lieu qu'elle soit donnée au hasard.

Une autre amélioration possible est à travers les modes de jeu qui pourraient être en fonction du nombre de joueurs et d'IA participant à une partie.

6 Bibliographie

Références

[1] Page Wikipédia du jeu de Okey. https://fr.wikipedia.org/wiki/Okey

[2] Voici les règles du jeu de Okey ou rami turc | Rami France https://www.ramif.com/regles-okey-rami-turc.php