STIZ Romain
POUZIN Pierre-Emmanuel

RAPPORT FINAL
Systeme et Reseau

Table des matières

1)Présentation et règle du jeu

a) Présentation

Tirer du sujet donner : L'objectif de ce projet est d'illustrer l'utilisation des concepts et mécanismes abordés en cours de Systèmes & Réseaux au travers du développement du jeu de carte 6 qui prend sur ordinateur en permettant aux utilisateurs de jouer en ligne `à travers un terminal avec d'autres joueurs et/ou processus robots. 6 qui prend est un jeu de cartes de type "défausse" créé par Wolfgang Kramer en 1994. En voici une description issue du site Gigamagic : "dans ce jeu culte, il vous faudra placer vos cartes dans les différentes rangées mais sans jamais poser la 6ème ! Les choix des cartes étant secrets et simultanés, il vous faudra être un poil calculateur pour ne pas finir avec le plus gros troupeau. Alors gare aux vacheries !"

b) Placement de carte

Pour le placement des cartes on a respecté les règles du jeu,lorsque les joueurs jouent une carte celle-ci est placée dans une liste, nous avons un tri a bulle sur cette liste pour trié par ordre croissant.
Pour les joueurs la carte est positionnée sur la ligne ou la différence est la plus petite avec sa carte, mais celle-ci est toujours supérieure à la carte sur le plateau.
Pour les ordinateurs basiques il joue une carte aléatoire et celle- ci est positionnée de façon aléatoire sans chercher à la positionner de manière optimale, on regarde juste qu'elle soit supérieur.
Avant de commencer à jouer des cartes, on place une carte sur chaque ligne du plateau.

c) Mise à jour des listes

Comme dit précédemment les cartes sont placé dans une liste qui est affiché à chaque début de tour pour pouvoir suivre correctement le Jeu .De même sur le plateau c'est mis a jour des qu'on pose une carte, pour la main des joueur quand il joue la liste de carte est mis à jour et la carte leur est retiré.

d) 6eme carte

Quand on essaye de jouer une carte et que celle-ci doit aller a la 6eme position sur une ligne,alors on calcul le score des têtes de boeuf des cartes sur la même ligne, on ajoute ce score au joueur qui vient de jouer la 6eme carte, on retire les cartes et on pose la sienne sur la première colonne.

e) Plus petite carte

Un peu de la même manière quand on a une carte qui ne peut pas être posée car elle est trop petite :
-Pour le joueur humain,il choisit un ligne qu'il veut retirer pour poser sa carte et l'ajout du score est fait
-Pour les robots basiques il choisit une ligne de façon aléatoire et le score lui est ajouté.

f) Arrêt

La condition d'arrêt du jeu se fait quand un joueur ou un robot atteint un score de 66 ou plus.

g) Score

Le score est calculé à chaque fois qu'un joueur ou un robot ramasse des têtes et celui-ci est affiché à la fin de chaque tour pour pouvoir suivre correctement le jeu.

2) Fonctionnalité additionnelle

a) Robot intelligent

Nous avons choisi comme fonctionnalité supplémentaire d'améliorer l'intelligence des robots pour les rendre plus difficiles.Cela nous a demandé de modifier quelques algorithmes comme celui où il joue une carte.

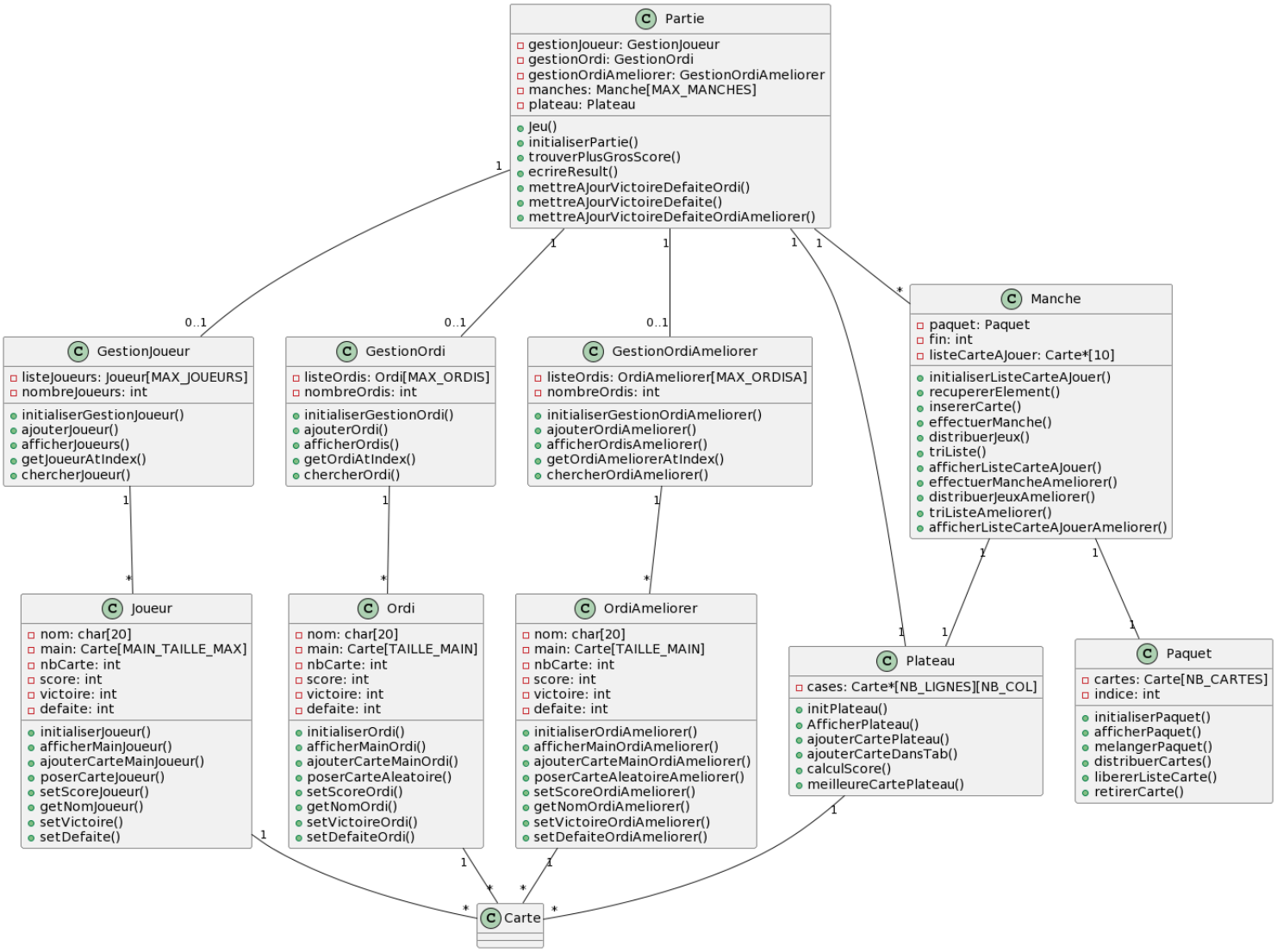
```
struct Carte poserCarteAleatoireAmeliorer(struct OrdiAmeliorer *ordi,
struct Plateau *p) {
	int choix = 0;
	int indexCarte = -1;
	struct Carte cartesJouer[ordi->nbCarte];
	for (int i = 0; i < ordi->nbCarte; ++i) {
		struct Carte c = meilleureCartePlateau(p, &ordi->main[i]);
		if (c.numero != 0) {
			cartesJouer[choix] = c;
			choix++;
		}
	}
	if (choix == 0) {
		// Aucune carte jouable, choisir une au hasard
```

```
            indexCarte = rand() % ordi->nbCarte;
            for (int i = indexCarte; i < ordi->nbCarte - 1; ++i) {
                  ordi->main[i] = ordi->main[i + 1];
            }
            //  Si  aucune  carte  n'est  jouable,  marquer  le  dernier
emplacement comme vide
            ordi->main[ordi->nbCarte - 1].numero = 0;
            ordi->main[ordi->nbCarte - 1].teteDeBoeuf = 0;
            ordi->nbCarte--;
            return ordi->main[indexCarte];
      } else {
            // Choisir une carte parmi celles jouables
            indexCarte = rand() % choix;
            // Retrouver l'indice de la carte dans la main originale

            int index = -1;
            for (int i = 0; i < ordi->nbCarte; ++i) {
                  if          (cartesJouer[indexCarte].numero          ==
ordi->main[i].numero) {
                        index = i;
                        break;
                  }
            }
            if (index != -1) {
                  // Déplacer les cartes jouées vers la fin de la main
                  for (int i = index; i < ordi->nbCarte - 1; ++i) {
                        ordi->main[i] = ordi->main[i + 1];
                  }
                  // Marquer le dernier emplacement comme vide
                  ordi->main[ordi->nbCarte - 1].numero = 0;
                  ordi->main[ordi->nbCarte - 1].teteDeBoeuf = 0;
                  // Réduire le compte après le déplacement
                  ordi->nbCarte = ordi->nbCarte - 1;
                  return cartesJouer[indexCarte];
            }
      }
      ordi->main[ordi->nbCarte - 1].numero = 0;
      ordi->main[ordi->nbCarte - 1].teteDeBoeuf = 0;
      // Réduire le compte après le déplacement
      ordi->nbCarte = ordi->nbCarte - 1;
      // Retourner une carte vide
      return ordi->main[0];
}
```

Cet algo est un peu long, pour résumer si on regarde si l'ordi peut jouer une carte mais maintenant on regarde que cette carte est la plus petite différence avec la carte sur le plateau, si il a plusieurs carte qui ont la même différence alors il en choisit une au hasard.
Quand il doit prendre un ligne il choisit celle avec le moins de têtes, pareil quand il doit poser la plus petite carte.Le choix de la carte se fait dans la méthode meilleureCartePlateau()

3)Structure



Il manque un lien entre Paquet et carte.

Voici le diagramme de classe de notre projet pour mieux comprendre la structure et la logique derrière.
On a choisi une structure par acteur et gestion pour gérer ses structures quand on a plusieurs acteurs, ce qui nous paraissait le plus adapter.

On a fait le choix de séparer les ordis normaux et améliorer car pour nous il s'agissait de deux acteur différents qui n'ont pas la même logique.

La structure Plateau est obligatoire, car c'est là qu'on va pouvoir stocker les cartes de la partie et afficher le plateau pour une meilleure compréhension.

La structure Paquet comme son nom l'indique contiendra le Paquet de carte de la manche et se "reset" à chaque début de manche avant distribution d'où l'utilité d'avoir une structure.
La structure Manche contient le Plateau et le Paquet, cela aurait pu être fait dans la structure Partie , mais pour des raisons de clarté et de compréhension cela parait plus agréable d'avoir un structure dédiée de même que l'on peut avoir plusieur manche tant que le score n'atteint pas 66 .

La structure Partie nous permet de mettre en place le jeu de créer les objets Gestions Joueurs ,qui va contenir les joueur humain, ou autre en fonction du nombre de personnes présente dans la partie, c'est la méthode initialiserPartie en fonction de ce qu'on entrer dans le terminal qui va créer la partie en fonction de nos besoin .Nous permet de mettre la condition d'arrêt du jeu.D'écrire les résultats de la partie.

4)Réalisation

a) Processus

Par manque de temps nous n'avons pas de processus pour les ordinateurs et les joueurs , cela aurait dû être notre réalisation suivante.
Mais nous avons quand même plusieur processus qui sont lancer en parallèle pour faire les statistique et le fichier PDF.

```
       char awkCommandStats[] = "awk -F'[:,]' '{player_sum[$1] += $2;
player_count[$1] += 1; victories[$1] += $4; defeats[$1] += $6;} \
                              END {printf \"Moyenne des scores :\\n\"; \
                                   for (player in player_sum) { \
                                               avg_score = player_sum[player] /
player_count[player]; \
                                          printf \"%s : %.2f points, Victoires :
%d, Défaites : %d\\n\", \
                                                                  player, avg_score,
victories[player], defeats[player]; \
                                       }}' resultats_formates.txt > stat.txt 2>
awk_error.log";
       // Ouvre un processus pour exécuter la commande awk
       FILE *awkProcessStats = popen(awkCommandStats, "w");
```

Ici on a un premier processus lancer avec la commande popen , qui va executer un script en awk pour calculer les statistiques de la partie.

```
const char *c = "pandoc jeu.txt -o jeuPdf.pdf --pdf-engine=pdflatex";
       // Exécution de la commande via le shell
       int r = system(c);
       // Vérification du code de retour
       if (r == 0) {
           printf("La conversion a réussi.\n");
       } else {
           printf("Erreur lors de la conversion. Code de retour : %d\n", r);
       }
```

Ici on a un deuxième processus qui lance une commande système pour convertir un fichier PDF.

b) Communication

```
FILE *file = fopen("resultats_formates.txt", "a");
       if (file == NULL) {
             perror("Erreur lors de l'ouverture du fichier resultats_formates.txt");
             exit(EXIT_FAILURE);
       }
       // Écrire les résultats des joueurs dans le fichier avec le nombre de victoires et de défaites
       for (int i = 0; i < gj->nombreJoueurs; ++i) {
             fprintf(file, "%s : %d points, Victoires : %d, Défaites : %d\n",
                          gj->listeJoueurs[i].nom,
                          gj->listeJoueurs[i].score,
                          gj->listeJoueurs[i].victoire,
                          gj->listeJoueurs[i].defaite);
       }
       // Écrire les résultats des ordinateurs dans le fichier avec le nombre de victoires et de défaites
       for (int i = 0; i < go->nombreOrdis; ++i) {
             fprintf(file, "%s : %d points, Victoires : %d, Défaites : %d\n",
                          go->listeOrdis[i].nom,
                          go->listeOrdis[i].score,
                          go->listeOrdis[i].victoire,
                          go->listeOrdis[i].defaite);
       }
       // Écrire les résultats des ordinateurs dans le fichier avec le nombre de victoires et de défaites
       for (int i = 0; i < gm->nombreOrdis; ++i) {
             fprintf(file, "%s : %d points, Victoires : %d, Défaites : %d\n",
                          gm->listeOrdis[i].nom,
```

```
                        gm->listeOrdis[i].score,
                        gm->listeOrdis[i].victoire,
                        gm->listeOrdis[i].defaite);
        }
        // Fermer le fichier
        fclose(file);
```

On a une communication pour écrire dans le fichier et ensuite faire les statistique dessus, car le processus awk doit attendre la fin de l'écriture.Pour éviter le problème de zone critique on lance le processus awk uniquement à la fin de l'écriture du fichier ce qui évite d'implémenter un système de verrou ou autre.

Nous n'avons pas de communication directe entre deux processus.

c) Têtes de boeuf

```
void initialiserPaquet(struct Paquet *paquet) {
    int i;
    paquet->indice=0;
    for (i = 0; i < NB_CARTES; ++i) {
        paquet->cartes[i].numero = i + 1;
        if ((i + 1) % 10 == 0) {
            paquet->cartes[i].teteDeBoeuf = 3;  // Cartes se terminant par 0
        }else if ((i + 1) == 55) {
            paquet->cartes[i].teteDeBoeuf = 7;  // Carte numéro 55
        }else if ((i + 1) % 10 == 5) {
            paquet->cartes[i].teteDeBoeuf = 2;  // Cartes se terminant par 5
        } else if ((i + 1) % 11 == 0) {
            paquet->cartes[i].teteDeBoeuf = 5;  // Doubles (comme 11, 22, etc.)
        } else {
            paquet->cartes[i].teteDeBoeuf = 1;  // Autres cartes
        }
    }
}
```

Pour les têtes nous regardons le numéros de la carte et lui attribuons la tête qui convient en suivant les règles du jeu.

d) Max Joueurs

Le maximum de joueur est géré avant le début de la partie , il y a un total de 10 personnes maximum car nous avons que 104 cartes (10 par /p + 4 pour le plateau) , on peut avoir 10 joueurs , ou dix robots normaux/intelligent pour faire une partie uniquement de robots ou de joueurs , mais on peut mélanger les joueurs et les robots.

```
Bienvenue sur le 6 Qui Prend

Veuillez choisir le nombre de Personnes total de la Partie:

10
Veuillez choisir le nombre de Joueurs afin de combler le reste par des Ordinateurs

5
Voulez-vous comblez la partie avec des Ordis Améliorés?

1:Oui                       2:Non

2
```

On peut voir qu'on remplit automatiquement le nombre d'ordinateurs car on le nombre de personnes et de joueurs.

e) Affichage

Exemple d'affichage pour une partie uniquement de Robots :

```
Début de la Manche:


Début du tour 1:

| 2* 5 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 3 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 14 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 73 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |




Liste des cartes à jouer :
| 1* 17 | | 1* 47 | | 1* 54 |

| 2* 5 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 3 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 14 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 73 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 2* 5 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 3 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 14 | | 1* 17 | | 1* 47 | | 1* 54 | |  vide  ||  vide  |
| 1* 73 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |


nom : Bot n°1 , score : 0
nom : Bot n°2 , score : 0
nom : Bot n°3 , score : 0
```

Exemple d’affichage pour une partie avec des joueurs :

```
Début du tour 1:

| 1* 87 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 96 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 3* 10 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 68 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
Main du Joueur PE :
Carte 0 - Numéro : 18, Têtes de Bœuf : 1 / PE
Carte 1 - Numéro : 104, Têtes de Bœuf : 1 / PE
Carte 2 - Numéro : 36, Têtes de Bœuf : 1 / PE
Carte 3 - Numéro : 42, Têtes de Bœuf : 1 / PE
Carte 4 - Numéro : 72, Têtes de Bœuf : 1 / PE
Carte 5 - Numéro : 37, Têtes de Bœuf : 1 / PE
Carte 6 - Numéro : 62, Têtes de Bœuf : 1 / PE
Carte 7 - Numéro : 88, Têtes de Bœuf : 5 / PE
Carte 8 - Numéro : 92, Têtes de Bœuf : 1 / PE
Carte 9 - Numéro : 55, Têtes de Bœuf : 7 / PE

Choisir une carte à jouer :

0

Main de l'ordinateur Bot Amélioré n°1 :
Carte 0 - Numéro : 9, Têtes de Bœuf : 1
Carte 1 - Numéro : 6, Têtes de Bœuf : 1
Carte 2 - Numéro : 70, Têtes de Bœuf : 3
Carte 3 - Numéro : 1, Têtes de Bœuf : 1
Carte 4 - Numéro : 48, Têtes de Bœuf : 1
Carte 5 - Numéro : 97, Têtes de Bœuf : 1
Carte 6 - Numéro : 21, Têtes de Bœuf : 1
Carte 7 - Numéro : 20, Têtes de Bœuf : 3
Carte 8 - Numéro : 34, Têtes de Bœuf : 1
Carte 9 - Numéro : 31, Têtes de Bœuf : 1

Carte de l'ordinateur Bot Amélioré n°1 : 48;1


Liste des cartes à jouer :
| 1* 18 | | 1* 48 |
```

```
Liste des cartes à jouer :
| 1* 18 | | 1* 48 |

| 1* 87 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 96 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 3* 10 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 68 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
nom : PE , score : 0
nom : Bot Amélioré n°1 , score : 0

| 1* 87 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 1* 96 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |
| 3* 10 | | 1* 18 | | 1* 48 | |  vide  ||  vide  ||  vide  |
| 1* 68 | |  vide  ||  vide  ||  vide  ||  vide  ||  vide  |

Début du tour 2:
```

f) Statistiques

Le processus awk qui fait les statiques a été montré au dessus il recupere les resultats écrit dans un fichier et calcul les statistique a partir de ses données.

Données de départ :

```
Bot n°1 : 46 points, Victoires : 1, Défaites : 0
Bot n°2 : 6 points, Victoires : 1, Défaites : 0
Bot n°3 : 71 points, Victoires : 0, Défaites : 1
Bot n°4 : 10 points, Victoires : 1, Défaites : 0
Bot n°5 : 25 points, Victoires : 1, Défaites : 0
NOM : 66 points, Victoires : 1, Défaites : 0
NOM1 : 60 points, Victoires : 1, Défaites : 0
Bot Amélioré n°1 : 19 points, Victoires : 1, Défaites : 0
Bot Amélioré n°2 : 19 points, Victoires : 1, Défaites : 0
Bot Amélioré n°3 : 17 points, Victoires : 1, Défaites : 0
Bot n°1 : 45 points, Victoires : 1, Défaites : 0
Bot n°2 : 17 points, Victoires : 1, Défaites : 0
Bot n°3 : 30 points, Victoires : 1, Défaites : 0
Bot n°4 : 22 points, Victoires : 1, Défaites : 0
Bot n°5 : 76 points, Victoires : 0, Défaites : 1
Bot n°1 : 21 points, Victoires : 1, Défaites : 0
Bot n°2 : 66 points, Victoires : 1, Défaites : 0
Bot n°3 : 15 points, Victoires : 1, Défaites : 0
```

Statistiques:

```
Moyenne des scores :
Bot n°3  : 38.67 points, Victoires : 2, Défaites : 1
NOM  : 66.00 points, Victoires : 1, Défaites : 0
Bot n°4  : 16.00 points, Victoires : 2, Défaites : 0
Bot Amélioré n°1  : 19.00 points, Victoires : 1, Défaites : 0
Bot n°5  : 50.50 points, Victoires : 1, Défaites : 1
Bot Amélioré n°2  : 19.00 points, Victoires : 1, Défaites : 0
Bot Amélioré n°3  : 17.00 points, Victoires : 1, Défaites : 0
NOM1  : 60.00 points, Victoires : 1, Défaites : 0
Bot n°1  : 37.33 points, Victoires : 3, Défaites : 0
Bot n°2  : 29.67 points, Victoires : 3, Défaites : 0
```

On affiche le score moyen , le nombre de victoires et de défaites du joueur.
On part du principe que dans un partie si la personne n'a pas perdu cette partie elle l'a donc gagnée donc il peut y avoir plusieur gagnant par partie mais un seul perdant.

5)Exemple d'une partie de jeu PDF

Il n'y aura pas d'exemple ici car le rapport ferait beaucoup trop de pages , mais un fichier pdf comme demandé dans le sujet est attaché au projet et contient un set de plusieurs parties effectuer et enregistrer , celui-ci se met a jours apres chaque partie de jeu.

6)MakeFile

Nous avons fait deux makefiles différents: un pour windows et un pour linux.Comme montré lors de la présentation nous utilisons principalement celui sur windows car c'est sur ce système que nous avons travaillé.

```
CC = gcc
CFLAGS = -std=c11
SRC    =    main.c    Joueur.c    Ordi.c    OrdiAmeliorer.c
GestionJoueur.c    GestionOrdi.c    GestionOrdiAmeliorer.c
Manche.c Paquet.c Carte.c Partie.c Affichage.c Plateau.c
OBJ = $(SRC:.c=.o)
EXEC = 6Quiprend3.exe

all: $(EXEC)

$(EXEC): $(OBJ)
	$(CC) $(CFLAGS) -o $@ $^

%.o: %.c %.h
	$(CC) $(CFLAGS) -c $<

clean:
	del $(OBJ) $(EXEC)
```

CC = gcc : Définit la variable CC comme le compilateur à utiliser, ici GCC.
CFLAGS = -std=c11 : Définit la variable CFLAGS comme les options de compilation, ici -std=c11 spécifie le standard C11.

SRC = main.c Joueur.c Ordi.c OrdiAmeliorer.c GestionJoueur.c GestionOrdi.c GestionOrdiAmeliorer.c Manche.c Paquet.c Carte.c Partie.c Affichage.c Plateau.c : Définit la variable SRC comme la liste des fichiers source du programme.

OBJ = $(SRC:.c=.o) : Définit la variable OBJ comme la liste des fichiers objets correspondant aux fichiers source.

EXEC = 6Quiprend3.exe : Définit la variable EXEC comme le nom de l'exécutable à générer.
all: $(EXEC) : Règle par défaut, qui indique que la cible par défaut (all) dépend de l'exécutable.

$(EXEC): $(OBJ) : Règle pour générer l'exécutable à partir des fichiers objets.

$(CC) $(CFLAGS) -o $@ $^ : Commande pour compiler l'exécutable. $@ représente la cible (ici, l'exécutable), et $^ représente toutes les dépendances (les fichiers objets).

%.o: %.c %.h : Règle générique pour générer un fichier objet à partir d'un fichier source et de son en-tête correspondant.
$(CC) $(CFLAGS) -c $< : Commande pour générer un fichier objet à partir d'un fichier source.

clean: del $(OBJ) $(EXEC) : Règle pour supprimer les fichiers objets et l'exécutable générés.

7)Conclusion

Beaucoup d'heure on été utilisé à la mise en place de ce projet, qui nous aura permit de prendre conscience de certain point ou nous sommes moins à l'aise comme les "pointeurs" et nous aura fait travailler notre algorithmique car de nombreux algorithme on du être réalisé pour l'optimisation du jeu.
Le plus gros du temps a été passé au débogage du jeu.
Avec le temps qui nous manquait, nous aurions bien aimé travailler sur la partie communication des processus ou encore au moins ajouter des threads.
Au final le projet nous a permis de travailler sur une majorité de nos connaissances en informatique et la cohésion d'équipe.