S1.02 : Comparaison d'approches algorithmiques

Le quart de singe

Groupe 104

Romain Millet - Ariane Luc

6 janvier 2023

Table des matières

Introduction du projet

Dans ce projet, notre rôle était de développer une application où l'on peut y jouer le "Quart de Singe", plusieurs joueurs, qu'ils soient des humains ou des robots, peuvent effectuer une partie.

Quelles sont les règles du jeu ?

Chaque joueur introduit une lettre, qui formera au fur et à mesure un mot. Si ce mot existe déjà dans le dictionnaire du Scrabble, le joueur ayant complété en dernier le mot recevra, un Quart de Singe équivalent à 0.25 points. Deux autres entrées sont autorisées : En premier le ? Cela signifie que le joueur demande au joueur précédent à quel mot il pense. Une deuxième entrée est possible et c'est le point d'exclamation ! signifiant que le joueur abandonne la manche actuelle ; tout en se prenant un quart de singe. La partie prend fin lorsqu'un joueur atteint 1 point.

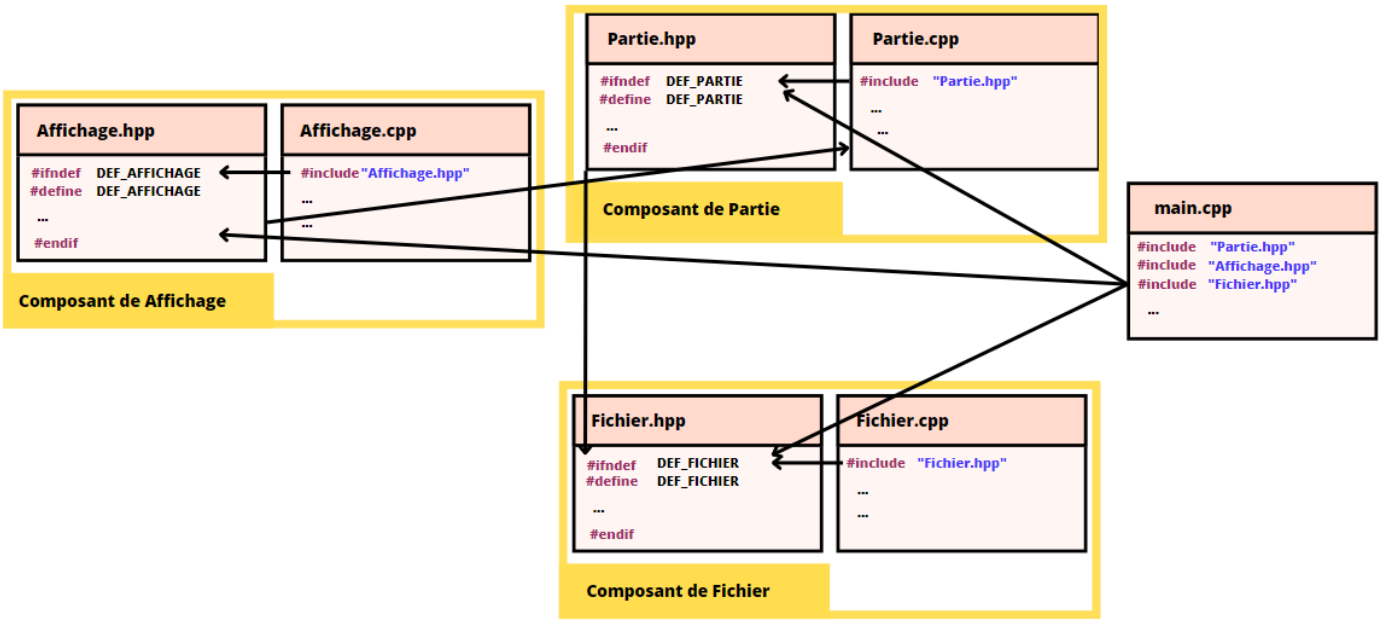
Conditions pour se prendre un Quart de Singe :

- Trouve un mot du dictionnaire qui existe déjà
- Par suite du "?" Le joueur ne propose pas un mot qui existe dans le dictionnaire
- À la suite du "?" Le joueur écrit mal le mot proposé
- Le joueur abandonne la manche avec "!"

Interdictions lors de ce projet :

- Utilisation du type string
- Utilisation conteneurs de la bibliothèque standard

Le graphe de dépendance des fichiers sources



Le code source des tests unitaires

- 1 er test : Comparaison entre les fichiers in-out donnée par le professeur.

```
C:\Users\Ariane\Desktop\qds>qds.exe HHHHHH <in.txt> res.txt

C:\Users\Ariane\Desktop\qds>fc out.txt res.txt
Comparaison des fichiers out.txt et RES.TXT
FC : aucune différence trouvée
```

Un bilan du projet

Difficultés rencontrées :

- Lorsqu'on effectue le point d'interrogation et que l'on demande au joueur précédent qui s'avère être un robot au mot qu'il pense:
    - Nous ne voulions pas que notre robot sorte aléatoirement des lettres qui n'ont pas de sens, mais plutôt un robot, qui, s'il le peut, fera en sorte de gagner en disant un mot qui existe dans le dictionnaire. Pour cela nous devions effectuer une vérification entre le début des mots du dictionnaire et le mot formé tout au long de la manche.
    - **Résolution :** On a comparé les deux chaînes de caractères avec strlen pour vérifier que le mot du dictionnaire était bien supérieur au mot de la manche. Avant d'effectuer une vérification de la validité des deux mots, si oui alors on copiait le mot du dictionnaire trouvé, dans un tableau qui est renvoyé par notre fonction.
- Au bout de la 7ème lettre enregistrée dans notre mot le programme se ferme subitement.

  1H, () > A

  2H, (A) > D

  3H, (AD) > V

  4H, (ADV) > E

  5H, (ADVE) > R

  6H, (ADVER) > S

1H, (ADVERS) > I

2H, (ADVERSI) > T

3H, (ADVERSIT—) >

PS C:\Users\Ariane\Desktop\qds>

Résolution : Il s'avère qu'à chaque round lorsqu'on allouait une nouvelle lettre, la taille d'allouement était mauvaise, en effet on ne prenait pas en compte le \0 ce qui a engendré ses erreurs. Au lieu de `tmp.mot = new char[strlen(partie.mot)+1];` on a mit `tmp.mot = new char[strlen(partie.mot)+2];`

- Il y avait également un problème avec le compteur "ijoueur" pour la position du joueur, il y avait une mauvaise incrémentation, +1 ou -1 lorsqu'il ne fallait pas et inversement.

  Résolution : On a changé tous nos index de joueurs.
- Certaines vérifications essentielles au bon fonctionnement du jeu n'étaient pas effectuées, et par conséquent le jeu était faussé. Il a donc fallu intelligemment placer les vérifications nécessaires telles que la vérification des points des joueurs, lorsque l'un atteignait 4 quarts de singe soit 1 point.
- Un petit inconvénient nous obligeait à entrer chaque caractère en majuscule. Nous avons donc pour chaque entrée effectué une conversion grâce à "toupper()" dans la fonction "conv(char t[]);".

Réussite :

- On a réussi à afficher et aligner correctement le contenu du jeu (Le mot, le score..)
- Nos lettres se stocks bien dans le conteneur, qui s'initialise à chaque manche et se détruit à la fin de la partie
- Nos compteurs sont optimisés on a fait en sorte que même si le joueur 1 effectue un ? ce sera bien le dernier joueur qui sera interrogé
- Générer des lettres en majuscules aléatoirement et

Amélioration

- Optimisation des robots en faisant en sorte qu'ils gagnent quasi tout le temps
- Réduire la taille de main, (cause du problème = manque de temps)
- Créer des fichiers qui sont plus cohérents
- Créer des meilleurs liens entre chaque fichiers

-

Annexe

Le code complet de nos sources .cpp et .hpp

Fichier.hpp

```
#ifndef DEF_FICHIER
#define DEF_FICHIER

#include <iostream>
#include "Partie.hpp"

/*PROTOTYPES*/
int recherche(char mot[]);
int robot(Partie &partie, char t[]);

#endif // !DEF_FONCTIONS
```

Fichier.cpp

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <fstream>
#include "Fichier.hpp"

using namespace std;

/**
  * Recherche le mot donné dans un fichier dictionnaire.
  *
  * @param mot Le mot à rechercher.
  * @return 0 si le mot est trouvé dans le dictionnaire, 1 s'il n'est pas trouvé,
2 si le fichier du dictionnaire n'a pas pu être ouvert.
  */
int recherche(char mot[]){
    std::ifstream in("ods4.txt"); // on ouvre le fichier
    if (!in) { //Si le fichier ne peux pas s'ouvrir
        std::cout << "le dictionnaire n'a pu etre ouvert" << endl; //Message
d'erreur
        return 2; //retourne 2 pour indiquer l'erreur
    }

    const int MAX = 26; //La longueur maximum d'un mot dans le dictionnaire
```

```
    char motdic[MAX]; //Un buffer pour lire les mots du dico
    in >> setw(MAX) >> motdic; // on essaye de lire le premier mot
    while (in) {
        if (strcmp(motdic, mot) == 0){ //Si le mot match avec celui que l'on
cherche
            return 0; //retourne 0 pour indiquer que le mot est trouvé
        }
        in >> setw(MAX) >> motdic; // on essaye de lire le mot suivant
    }
    in.close(); // on ferme le fichier
    return 1; //retourne 1 pour indiquer que le mot n'a pas été trouvé
}

/**
  * Trouve un mot dans un dictionnaire qui commence par un mot donné.
  *
  * @param partie L'objet de jeu dont le mot sera utilisé comme préfixe.
  * @param t Le mot trouvé sera stocké dans cette chaîne.
  * @return 1 si un mot est trouvé, 2 si le fichier dictionnaire ne peut pas être
ouvert, sinon 0.
  */

int robot(Partie &partie, char t[]){
    ifstream in("ods4.txt"); // on ouvre le fichier
    if (!in) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 2;
    }
    const int MAX = 26;
    char motdic[MAX];
    bool a = false;
    in >> setw(MAX) >> motdic; // on essaye de lire le premier mot
    while (in) {
        if (strlen(motdic) > strlen(partie.mot)){
            if(verifmot(partie, motdic)){
                strcpy(t, motdic);
                a = true;
                break;
            }
        }
        in >> setw(MAX) >> motdic; // on essaye de lire le mot suivant
    }
    if(a==false){
        strcpy(t, "!");
    }
```

```
    in.close(); // on ferme le fichier
    return 1;
}
```

Affichage.hpp

```
#ifndef DEF_AFFICHAGE
#define DEF_AFFICHAGE

#include <iostream>
#include "Partie.hpp"

/*PROTOTYPES*/
void afficher(Partie &partie, unsigned int nbjoueur);
void affichermot(char mot[]);
char afficherNature(natureJ natureJoueur);
void afficheTous(unsigned int ijoueur, Partie &partie, unsigned int nbjoueur);
void afficheScore(Partie &partie, unsigned int i, unsigned int nbjoueur);

#endif // !DEF_FONCTIONS
```

Affichage.cpp

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <fstream>
#include "Affichage.hpp"

using namespace std;

/**
  * Imprime les informations du joueur pour un objet de jeu.
  *
  * @param partie L'objet de jeu dont les informations sur le joueur seront
imprimées.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */
void afficher(Partie &partie, unsigned int nbjoueur){
  for (unsigned int i=0; i<nbjoueur; ++i){
    cout << partie.joueur[i].natureJoueur << ", " << partie.joueur[i].numJoueur
<< ", " << partie.joueur[i].scoreJoueur << endl;
  }
```

```
}

/**
  * Affiche le mot donné sur la sortie standard.
  *
  * @param mot Le mot à afficher.
  */
void affichermot(char mot[]){
  unsigned int taille = strlen(mot);
  for (unsigned int i=0; i<taille; ++i){
    cout << mot[i];
  }
}

/**
  * Renvoie la représentation du personnage du type de joueur donné.
  *
  * @param natureJoueur Le type de joueur à représenter en tant que personnage.
  * @return 'H' si le type de joueur est H (humain), 'R' si le type de joueur est
R (robot).
  */
char afficherNature(natureJ natureJoueur){
  if(natureJoueur == 0)
    return 'H';
  else if (natureJoueur == 1)
    return 'R';
}

/**
  * Affiche le joueur dont c'est le tour, le mot en cours de lecture et une
invite de saisie.
  *
  * @param ijoueur L'index du joueur dont c'est le tour.
  * @param partie L'objet de jeu contenant les joueurs et le mot courant.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */
void afficheTous(unsigned int ijoueur, Partie &partie, unsigned int nbjoueur){
  tourdejeu(ijoueur, partie, nbjoueur);//Affiche le type de joueur et son numéro
  cout << ", (";
  affichermot(partie.mot);//Affiche le mot actuel
  cout << ") > ";
}

/**
  * Affiche les scores de tous les joueurs du jeu.
```

```
  *
  * @param partie L'objet de jeu contenant les joueurs et leurs scores.
  * @param i L'index du lecteur actuel.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */
void afficheScore(Partie &partie, unsigned int i, unsigned int nbjoueur) {
  for (int i = 0; i < nbjoueur; i++) { //Pour tout les joueurs
    if (i == nbjoueur-1) { //Si c'est le dernier joueur
      cout << partie.joueur[i].numJoueur <<
afficherNature(partie.joueur[i%nbjoueur].natureJoueur)<< " : " <<
partie.joueur[i].scoreJoueur <<endl;
      break;
    }
    cout << partie.joueur[i].numJoueur <<
afficherNature(partie.joueur[i%nbjoueur].natureJoueur)<< " : " <<
partie.joueur[i].scoreJoueur << "; ";
  }

}
```

Partie.hpp

```
#ifndef DEF_PARTIE
#define DEF_PARTIE

#include <iostream>

/**
  * Une énumération des types de joueurs.
  *
  * H - Joueur humain
  * R - Joueur robot
**/

enum natureJ{H, R};
const unsigned int tailleMax = 25;

/**
  * Une structure pour le stockage temporaire d'un mot.
*/

typedef struct{
  char *mot;
}Tmp;
```

```
/**
  * Une structure pour un joueur dans le jeu.
  */

typedef struct {
  natureJ natureJoueur; //Nature joueur Humain ou Robot
  unsigned int numJoueur;
  float scoreJoueur;
}Joueur;

/**
  * Une structure pour un jeu.
  */

typedef struct{
  Joueur *joueur; //Les joueurs de la partie
  unsigned int nbjoueur; //nombre de joueur
  char *mot; //Stock le mot
}Partie;

void initialisation(const char*argv[], Partie &partie, unsigned int nbjoueur);
void tourdejeu(unsigned int ijoueur, Partie &partie, unsigned int nbjoueur);
bool verifmot(Partie &partie, char mot[]);
void generateurLettre(char a[]);
void conv(char t[]);
void perdu(Partie &partie, unsigned int ijoueur, unsigned int nbjoueur);
void destru(Partie &partie, Tmp &tmp);

#endif // !DEF_FONCTIONS
```

Partie.cpp

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <fstream>
#include "Partie.hpp"
#include "Affichage.hpp"

using namespace std;

/**
  * Initialise les joueurs et le mot pour un nouveau jeu.
```

```
  *
  * @param argv Les arguments de la ligne de commande passés au programme. Le
premier argument devrait contenir les types de joueurs (H pour humain, R pour
robot).
  * @param partie Une référence au jeu à initialiser.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */
void initialisation(const char*argv[], Partie &partie, unsigned int nbjoueur) {
  unsigned int j=0;
  partie.joueur  = new Joueur[nbjoueur];
  partie.mot = new char[2];
  while (argv[1][j]){
    partie.nbjoueur = nbjoueur;
    if (argv[1][j] == 'H')
      partie.joueur[j].natureJoueur = H;
    else if (argv[1][j] == 'R')
      partie.joueur[j].natureJoueur = R;
    partie.joueur[j].numJoueur = j+1;
    partie.joueur[j].scoreJoueur = 0.;
    ++j;
  }

}

/**
  * Affiche le nombre et le type de joueur dont c'est le tour.
  *
  * @param ijoueur L'index du joueur dont c'est le tour.
  * @param partie L'objet de jeu contenant les joueurs.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */
void tourdejeu(unsigned int ijoueur, Partie &partie, unsigned int nbjoueur){
  cout << partie.joueur[ijoueur%nbjoueur].numJoueur <<
afficherNature(partie.joueur[ijoueur%nbjoueur].natureJoueur);
}

/**
  * Vérifie que le mot donné est une continuation valide du mot courant.
  *
  * @param partie L'objet de jeu contenant le mot courant.
  * @param mot Le mot à vérifier comme suite valide.
  * @return true si le mot donné est une continuation valide, false sinon.
  */
bool verifmot(Partie &partie, char mot[]){
  for (int i=0; i<strlen(partie.mot); ++i){
```

```
    if(partie.mot[i] != mot[i]) //si le caractère à la même position que le mot
donnée est différents
      return false; //return faux
  }
  return true; //return true si tous les caractères match
}

/**
  * Génère une lettre aléatoire.
  *
  * @param a Le tableau pour stocker la lettre générée.
  */
void generateurLettre(char a[]) {
  srand(time(NULL)); //Initialisation du générateur de lettre
  *a = rand() % ('Z' - 'A' + 1) + 'A'; //génère une lettre entre 'A' et 'Z'
}

/** Convertit tous les caractères de la chaîne donnée en majuscules.
  *
  * @param t Une chaîne terminée par un caractère nul à convertir en majuscules.
  * La chaîne est modifiée sur place.
  */

void conv(char t[]){
  for (int i = 0; i < strlen(t); i++) {
  t[i] = toupper(t[i]);
  }
}

/**
  * Affiche le joueur qui a perdu la partie.
  *
  * @param partie L'objet de jeu contenant les joueurs.
  * @param ijoueur L'index du joueur qui a perdu la partie.
  * @param nbjoueur Le nombre de joueurs dans la partie.
  */

void perdu(Partie &partie, unsigned int ijoueur, unsigned int nbjoueur){
  tourdejeu(ijoueur, partie, nbjoueur); //Affiche le numéro et type de joueur
  cout << " a perdu la partie" << endl; //output message pour indiquer le joueur
perdant
}

/** Supprime la mémoire allouée dynamiquement pour un objet de jeu.
  *
```

```
  * @param partie L'objet de jeu dont la mémoire allouée dynamiquement sera
supprimée.
  * @param tmp L'objet de jeu dont la mémoire allouée dynamiquement sera
supprimée.
  */
void destru(Partie &partie, Tmp &tmp) {
  delete[] partie.joueur;
  delete[] partie.mot;
  delete[] tmp.mot;
}
```

Main.cpp

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <fstream>
#include <random> //generer mot
#include <limits.h>
#include "Fichier.hpp"
#include "Affichage.hpp"
#include "Partie.hpp"

using namespace std;

int main(int argc, const char* argv[]) {
    unsigned int nbjoueur = strlen(argv[1]);
    //cout << te << endl;
    if (nbjoueur >= 2) {
        Partie partie;
        initialisation(argv, partie, nbjoueur);
        //unsigned int tailleinit=2;
        strcpy(partie.mot, "");

        Tmp tmp;
        unsigned int ijoueur=0;
        char c[2];
        while (1){
            afficheTous(ijoueur%nbjoueur, partie, nbjoueur);

            tmp.mot = new char[strlen(partie.mot)+2];
            strcpy(tmp.mot, partie.mot);

            //si humain insérer input
            if(afficherNature(partie.joueur[ijoueur%nbjoueur].natureJoueur) ==
'H') {
```

```
                cin >> setw(2) >> c;
                cin.ignore(INT_MAX, '\n');
            }
            //Sinon générer une lettre aléatoirement
            else {
                char a[2];
                //GENERER UN MOT
                generateurLettre(a);
                cout << a[0] << endl;
                c[0] = a[0];
            }
            conv(c);

            strcat(tmp.mot, c);
            if(strcmp(c,"?")==0){
                char idee[tailleMax+1];
                //Demande mot
                tourdejeu(ijoueur-1, partie, nbjoueur);
                cout << ", saisir le mot > ";

                if(afficherNature(partie.joueur[(ijoueur-
1)%nbjoueur].natureJoueur) == 'H') {
                    cin >> setw(tailleMax+1) >> idee;
                    cin.ignore(INT_MAX, '\n');
                }
                //Sinon générer une lettre aléatoirement
                else {
                    robot(partie,idee);
                    affichermot(idee);
                    cout << endl;
                }
// et modif dans robot avec bool true et false
                if(strcmp(idee,"!")==0) {
                    cout << "le joueur ";
                    tourdejeu(ijoueur-1, partie, nbjoueur);
                    cout << " abandonne la manche et prend un quart de singe" <<
endl;
                    //Le joueur se prend un quart de singe
                    partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur += 0.25;
                    //fin de la manche donc on rénitialise le mot
                    afficheScore(partie, ijoueur, nbjoueur);
                    //cout << endl;

                    if(partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur == 1.){
```

```
                        //il faut detruire (mémoire) après le boucle while()
juste avant return 0
                        break;
                    }
                    else{
                        partie.mot = new char[2];
                        strcpy(partie.mot, "");
                    }
                }else{
                    conv(idee);
                    if(verifmot(partie, idee)){
                        //cout << "mot identique" << endl;
                        if(recherche(idee)==0){
                            cout << "le mot ";
                            affichermot(idee);
                            cout << " existe, le joueur ";
                            tourdejeu(ijoueur, partie, nbjoueur);
                            cout << " prend un quart de singe" << endl;
                            partie.joueur[ijoueur%nbjoueur].scoreJoueur += 0.25;
                            afficheScore(partie, ijoueur, nbjoueur);

                            if(partie.joueur[ijoueur%nbjoueur].scoreJoueur ==
1.){
                                //il faut detruire (mémoire) après le boucle
while() juste avant return 0
                                break;
                            }else{
                                partie.mot = new char[2];
                                strcpy(partie.mot, "");
                            }

                        }else{
                            cout << "le mot ";
                            affichermot(idee);
                            cout << " n'existe pas, le joueur ";
                            tourdejeu(ijoueur-1, partie, nbjoueur);
                            cout << " prend un quart de singe" << endl;
                            partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur +=
0.25;
                            //affiche score
                            afficheScore(partie, ijoueur, nbjoueur);

                            if(partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur ==
1.){
```

```
                                //il faut detruire (mémoire) après le boucle
while() juste avant return 0
                                break;
                            }else{
                                partie.mot = new char[2];
                                strcpy(partie.mot, "");
                            }
                            ijoueur -=1;
                        }
                    }else{
                        cout << "le mot ";
                        affichermot(idee);
                        cout << " ne commence pas par les lettres attendues, le
joueur ";
                        tourdejeu(ijoueur-1, partie, nbjoueur);
                        cout << " prend un quart de singe" << endl;
                        partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur += 0.25;
                        //Affiche score
                        afficheScore(partie, ijoueur, nbjoueur);

                        if(partie.joueur[(ijoueur-1)%nbjoueur].scoreJoueur ==
1.){
                            //perdu(partie, ijoueur-1, nbjoueur);
                            //il faut detruire (mémoire) après le boucle while()
juste avant return 0
                            break;
                        }else{
                            partie.mot = new char[2];
                            strcpy(partie.mot, "");
                        }
                        ijoueur -=1;
                    }
                }
                strcpy(idee, "");
            }
            else if(strcmp(c,"!")==0) {
                cout << "le joueur ";
                tourdejeu(ijoueur, partie, nbjoueur);
                cout << " abandonne la manche et prend un quart de singe" <<
endl;
                //Le joueur se prend un quart de singe
                partie.joueur[ijoueur%nbjoueur].scoreJoueur += 0.25;
                //fin de la manche donc on rénitialise le mot
                afficheScore(partie, ijoueur, nbjoueur);
                //cout << endl;
```

```
                if(partie.joueur[ijoueur%nbjoueur].scoreJoueur == 1.){
                    //il faut detruire (mémoire) après le boucle while() juste
avant return 0
                    break;
                }
                else{
                    partie.mot = new char[2];
                    strcpy(partie.mot, "");
                }
            }
            else if (strlen(partie.mot) < 2 || (strlen(partie.mot) >= 2 &&
recherche(tmp.mot) == 1)){
                partie.mot = new char[strlen(partie.mot)+2];
                strcpy(partie.mot, tmp.mot);
                ijoueur += 1;
            }
            else{
                cout << "le mot ";
                affichermot(tmp.mot);
                cout << " existe, le joueur ";
                tourdejeu(ijoueur, partie, nbjoueur);
                cout << " prend un quart de singe" << endl;
                //ajout point
                partie.joueur[ijoueur%nbjoueur].scoreJoueur += 0.25;
                //Affiche score
                afficheScore(partie, ijoueur, nbjoueur);
                //cout << endl;

                if(partie.joueur[ijoueur%nbjoueur].scoreJoueur == 1.){
                    //il faut detruire (mémoire) après le boucle while() juste
avant return 0
                    break;
                }
                else{
                    partie.mot = new char[2];
                    strcpy(partie.mot, "");
                }
            }
        }
        cout << "La partie est finie";
        destru(partie, tmp);
    }else{
        cout << "Les parametre son incorrects" << endl;
    }
```

```
    //détruire mémoire ici
    return 0;
}
```