DOSSIER DE DÉVELOPPEMENT LOGICIEL:

John LI, Nassim BEN DAALI
Groupe : 105

Table des matières :

1.Présentation brève du projet

Le projet en question est la SAé S1.02 effectué en période B à l'Université Rives-de-Seine en formation BUT Informatique, par les étudiants de 1er année (2022-2023) John LI et Nassim BEN DAALI, Groupe 105.

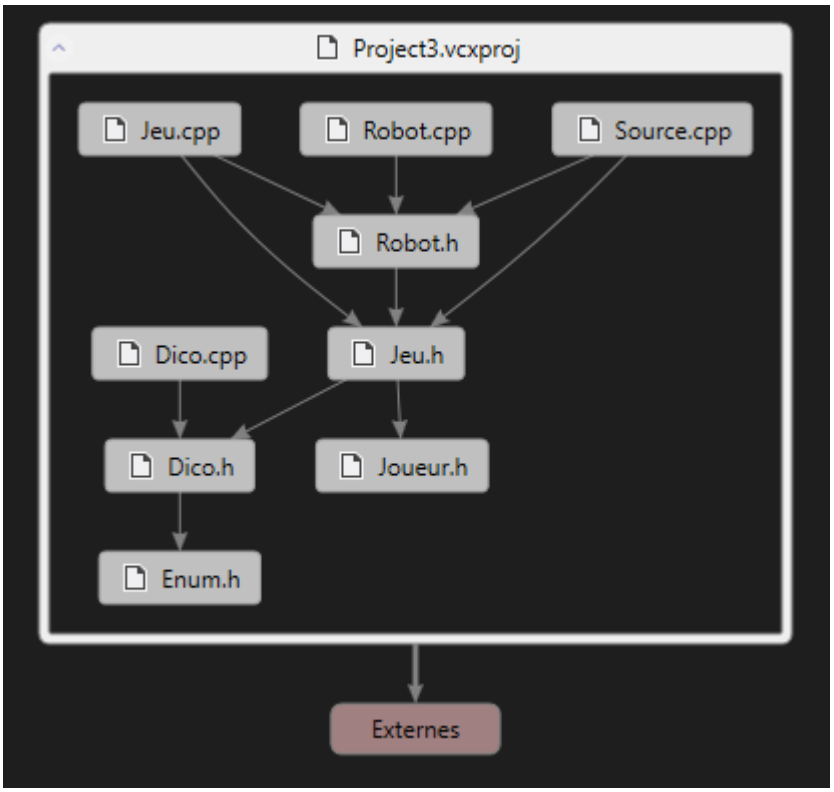
L'objectif de ce projet consistait à développer un logiciel permettant à un ensemble de joueurs de disputer une partie de quart de singe. L'application doit veiller au respect des règles du jeu et gérer la totalité du déroulement de la partie jusqu'à l'annonce du perdant tout en respectant un cahier des charges.

Les fonctions à prendre en compte vont permettre de définir et initialiser la structure du jeu, du dictionnaire, des joueurs avec leurs attributs spécifiques. Ainsi, les joueurs ont accès à différentes possibilités d'actions telles que l'abandon d'une manche, le bluff ou la saisie d'un caractère préférablement une lettre. Le jeu affiche à chaque manche l'état de la partie selon les actions réalisées et vérifie d'abord si la manche est finie, puis en conséquence si la partie doit prendre fin ou non.

Le joueur pourra en entrée saisir un caractère de formats attendus par les commandes en utilisant soit l'entrée standard (le clavier) pour les humains ou une valeur de retour depuis des fonctions pour les robots.
A partir des entrées, l'application devra faire l'appel des fonctions et produire les résultats en sortie. (l'écran)

A la suite de ces tâches, la nécessité de rédiger un dossier de développement logiciel afin de rendre compte du travail effectué.

2.Graphe de dépendance des fichiers sources



3. Code source des tests unitaires

```
	Jeu jeu;
	Dico dico;
	init_dico(dico);
	initialiser(jeu, argv);
	srand(time(NULL)); //initialise un générateur de nombres aléatoires
	char c;
	for (unsigned int i = 0; i < 4; ++i) {
		if (i == 0) {
			c = 'a';
			jeu.mot[jeu.nbLettre] = c;
			++jeu.nbLettre;
			Copie(jeu.mot, jeu.nbLettre);
			verif_dichotomique(dico, dico.nbMot, jeu.mot);
			assert(verif_dichotomique(dico, dico.nbMot, jeu.mot) == false);
			char a[6] = "table";
			verif_dichotomique(dico, dico.nbMot, a);
			assert(verif_dichotomique(dico, dico.nbMot, jeu.mot) == true);
		}
		if (i == 1) {
			c = '?';
			float point = 0;
			bluff(jeu, i, dico);
			assert(jeu.joueurs[i].nbPoint - point == 0.25 || jeu.joueurs[i - 1].nbPoint -
point == 0.25);
		}
		if (i == 2) {
			c = '!';
			float point = 0;
			surrend(jeu, jeu.joueurs, i);
			assert(jeu.joueurs[i].nbPoint - point == 0.25);

		}
		if (i == 3) {
			c = robot_choix(jeu, dico);
			assert(isalpha(c));
		}
	}
```

4.Bilan de projet :

Les difficultés rencontrées :

La première difficulté a été de réfléchir sur la création des structures permettant de faire une partie : “Jeu”, “Joueurs”, “Dico”. Également, réfléchir aux fonctions nécessaires pour la partie (initialisation des joueurs, du jeu, du dico), le robot, les vérifications du mot de la manche, les différents affichages en cours de la partie… Ensuite, mettre en place l’allocation dynamique pour stocker les mots du dictionnaire. Implémenter la recherche dichotomique du dictionnaire et les fonctions liées au robot notamment ses choix. Pour finir, il a fallu faire attention au passage au joueur précédent dans le cas du 1er joueur.

Les réussites :

Les réussites ont été l’utilisation des références pour accéder aux différentes structures.
Également, la création de fonctions intermédiaires afin d’éviter la redondance notamment pour la vérification de façon dichotomique des mots et des affichages.
Une autre réussite a été l’implémentation d’un joueur robot fonctionnel qui arrive à prendre des choix cohérents et stratégiques.
La réussite majeure a été l’aboutissement du programme complet avec toutes les fonctions nécessaires au fonctionnement du jeu “quart de singe”, ce qui signifie avoir réussi à surmonter toutes les difficultés rencontrées lors de ce projet.

Les améliorations possibles :

Les améliorations évidentes sont la création de fonctions intermédiaires supplémentaires.
Également, on aurait pu éviter de nombreuses redondances sur les affichages. Réduire la longueur de notre main. Diviser notre code source en plus de .h et .cpp. Rendre le code du robot plus propre et le rendre plus intelligent par exemple en tenant des pièges aux joueurs. Éviter au maximum d'utiliser des nombres magiques comme pour le seuil qui est uniquement arbitraire. On aurait des structures abstraites comme la file circulaire afin de nous éviter les problèmes au passage au joueur précédent. Créer une structure Mot à part entière pour le mot courant de la manche.
Pour finir, on aurait pu éliminer tous les avertissements de Visual Studio.

4.Annexes :

a. Listing du code source avec l'extension (Code Blocks)

b. LES CODES SOURCES DES .h :

Enum.h :

```
#ifndef ENUM_H
#define ENUM_H
/**
* @file Enum.h
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Constantes nécessaires au jeu, dico, robot
* @date Semestre 1 Période B
*/



enum {
	TAILLE_MINIMALE = 2, // taille minimale d'un mot du dico
	PAS = 2, //Le pas pour l'extension du dico
	ALPHABET = 26,
	SEUIL = 20 //Modifiable, le seuil à laquel le robot "?" ou pas
};


#endif
```

Joueur.h :

```
#ifndef JOUEUR_H
#define JOUEUR_H
/**
* @file Joueur.h
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Structure du joueur
* @date Semestre 1 Période B
*/

struct Joueur {
	char type;
```

```
	unsigned int NumJoueur;
	float nbPoint;
};


#endif
```

Dico.h :

```
#ifndef DICO_H
#define DICO_H
/**
* @file Dico.h
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composant de dictionnaire et recherche dichotomique
* @date Semestre 1 Période B
*/

#include <ctype.h>
#include <cctype>
#include <iostream>
#include <cstring>
#include <fstream> // pour ifstream
#include <iomanip>
#include "Enum.h"
using namespace std;

struct Mot {
	char mot[ALPHABET];
};

struct Dico {
	Mot* dico;
	unsigned int nbMot = 0;
	unsigned int capacite = 1;
	unsigned int pasExtension = PAS;
};



/**
* @brief Initialise le dictionnaire en ajoutant tous les mots du fichier
* @param[in-out] Dico& dico, référence du dico
* @return void
*/
void init_dico(Dico& dico);
```

```
/**
* @brief Réalise une recherche dichotomique dans le dico avec un mot à trouver
* @param[in-out] Dico& dico, référence du dico
* @param[in] int taille, taille du mot
* @param[in] char mot[ALPHABET], le mot à trouver
* @return bool , true si le mot existe dans le dico , false sinon
*/
bool verif_dichotomique(Dico& dico, int taille, char mot[ALPHABET]);


#endif
```

Jeu.h :

```
#ifndef JEU_H
#define JEU_H

/**
* @file Jeu.h
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composants pour le déroulement du jeu
* @date Semestre 1 Période B
*/
#include <ctype.h>
#include <cctype>
#include <iostream>
#include <cstring>
#include "Dico.h"
#include "Joueur.h"
using namespace std;



struct Jeu {
	unsigned int nbLettre = 0;
	char* mot;
	unsigned int nbJoueur;
	Joueur* joueurs;
};


/**
* @brief Initialise la partie et les joueurs
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] char** argv, pour obtenir le nombre de joueurs
* @return void
```

```
*/
void initialiser(Jeu& jeu, char** argv);


/**
* @brief Initialise la partie et les joueurs
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] char** argv, pour obtenir le nombre de joueurs
* @return void
*/
void initialiser(Jeu& jeu, char** argv);


/**
* @brief Entrez le mot à saisir et affiche si le mot existe ou pas ou si le mot commence par les mêmes
lettres
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @param[in] unsigned int& i, position du joueur
* @return int, 0 pour sortir de la fonction
*/
void bluff(Jeu& jeu, unsigned int& i, Dico& dico);


/**
* @brief Affiche que le mot existe s'il a été trouvé
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] unsigned int& i, position du joueur
* @return void
*/
void trouver(Jeu& jeu, unsigned int i);


/**
* @brief Vérifie si un joueur à 4 quart de singe
* @param[in-out] Jeu& jeu, référence du jeu
* @return bool, true si un joueur à 4 quarts de singe et false sinon
*/
bool eliminer(Jeu& jeu);


/**
* @brief Affiche que le joueur abandonne la manche
* @param[in-out] Jeu& jeu, référence du jeu, Joueur* joueurs, pointeur sur les joueurs
* @param[in] unsigned int i, position du joueur
* @return void
*/
void surrend(Jeu& jeu, Joueur* joueurs, unsigned int i);

/**
* @brief Affiche la situation actuelle de la partie avec le mot et le joueur courant
* @param[in-out] Jeu& jeu, référence du jeu
```

```
* @param[in] unsigned int i, la position du joueur
* @return void
*/
void afficher(Jeu& jeu, unsigned int i);

/**
* @brief Affiche la situation actuelle de la partie avec le nombre de points des joueurs
* @param[in-out] Jeu& jeu, référence du jeu
* @return void
*/
void afficher_manche(Jeu& jeu);

/**
* @brief Devient le joueur précédent
* @param[in-out] Jeu& jeu, référence du jeu, unsigned int& i, position du joueur
* @return void
*/
void joueurPrecedent(Jeu& jeu, unsigned int& i);


/**
* @brief Vérifie si un joueur à 4 quart de singe
* @param[in-out] Jeu& jeu, référence du jeu
* @return bool, true si un joueur à 4 quarts de singe et false sinon
*/
bool eliminer(Jeu& jeu);

/**
* @brief Réalise une copie du mot de la manche
* @param[in-out] char*& mot, référence du mot de la manche
* @param[in] int taille, taille du mot
* @return void
*/
void Copie(char*& mot, int taille);


/**
* @brief Vérifie si le mot rentré à bien les mêmes lettres que le mot de la manche
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] char mot[ALPHABET], le mot rentré
* @return bool , true si le mot commence par les mêmes lettres , false sinon
*/
bool verif2(char mot[ALPHABET], Jeu& jeu);


/**
* @brief Réinitialise le mot de la manche
* @param[in-out] Jeu& jeu, référence du jeu
* @return void
*/
void reset_mot(Jeu& jeu);
```

```
#endif
```

Robot.h :

```
#ifndef ROBOT_H
#define ROBOT_H

/**
* @file Robot.h
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composants pour les choix du robot
* @date Semestre 1 Période B
*/

#include <cstdlib>
#include <ctime>
#include <stdlib.h>     /* srand, rand */
#include <time.h>
#include "Jeu.h"



/**
* @brief Renvoie le choix du robot selon l'état de la partie
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @return char : une lettre ou "?" pour appeller bluff
*/
char robot_choix(Jeu& jeu, Dico& dico);


/**
* @brief Renvoie un caractère selon le nombre de mots que l'on peut former avec le mot du jeu
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @return char, un caractère
*/
char verif_robot(Jeu& jeu, Dico& dico);


/**
* @brief Renvoie le choix du robot lorsqu'il se fait bluff
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @param[in] unsigned int y, position du joueur précédent
* @return char*, le mot trouvé dans le dico ou "!" si le robot ne trouve rien
*/
char* robot_choix_bluff(Jeu& jeu, Dico& dico, unsigned int y);
```

```
#endif
```

c. LES CODES SOURCES DES .cpp :

Dico.cpp :

```
/**
* @file Dico.cpp
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composant de dictionnaire et recherche dichotomique
* @date Semestre 1 Période B
*/


#include "Dico.h"


/**
* @brief Initialise le dictionnaire en ajoutant tous les mots du fichier
* @param[in-out] Dico& dico, référence du dico
* @return void
*/
void init_dico(Dico& dico) {
	dico.dico = new Mot[dico.capacite];
	ifstream in("ods4.txt"); // on ouvre le fichier
	if (!in) {
		cout << "le dictionnaire n'a pu etre ouvert" << endl;
		exit(2);
	}
	const int MAX = ALPHABET;
	char mot[ALPHABET];
	unsigned int i = 0;
	in >> setw(MAX) >> mot; // on lit le premier mot

	// on utilise l'allocation dynamique pour étendre la taille du dico autant que nécessaire
	while (in) {
		if (i >= dico.capacite) {
			/* Stratégie de réallocation proportionnelle au pas d'extension :
			 * initialisez la nouvelle taille du conteneur (newTaille)
			 * à i * c.pasExtension */
			unsigned int newTaille = i * dico.pasExtension;
```

```
                    /* Allouez en mémoire dynamique un nouveau tableau (newT)
                     * à cette nouvelle taille*/
                    Mot* newT = new Mot[newTaille];

                    /* Recopiez les items déjà stockés dans le conteneur */
                    for (unsigned int j = 0; j < dico.capacite; ++j)
                           newT[j] = dico.dico[j];

                    /* Désallouez l'ancien tableau */
                    delete[] dico.dico;

                    /* Actualiser la mise à jour du conteneur en mémoire dynamique
                    * Faites pointer le tableau support du conteneur
                    * sur le nouveau tableau en mémoire dynamique */
                    dico.dico = newT;

                    /* Actualisez la taille du conteneur */
                    dico.capacite = newTaille;
             }
             ++dico.nbMot;
             strcpy_s(dico.dico[i].mot, mot); //on copie le mot dans le dico
             in >> setw(MAX) >> mot; // on lit le mot suivant
             ++i;
       }
       in.close(); // on ferme le fichier
}


/**
* @brief Réalise une recherche dichotomique dans le dico avec un mot à trouver
* @param[in-out] Dico& dico, référence du dico
* @param[in] int taille, taille du mot
* @param[in] char mot[ALPHABET], le mot à trouver
* @return bool , true si le mot existe dans le dico , false sinon
*/
bool verif_dichotomique(Dico& dico, int taille, char mot[ALPHABET]) {
       int debut = 0; // indice de debut
       int fin = taille - 1; // indice de fin

       while (debut <= fin) // boucle de recherche
       {
             int milieu = (debut + fin) / 2; // indice du milieu
             int resultat = strcmp(dico.dico[milieu].mot, mot); //permet de comparer deux chaînes de
caractères et de savoir si la première est inférieure, égale ou supérieure à la seconde
             //ce qui permet de rechercher si le mot se trouve dans la première moitié du dico ou
seconde moitié
             if (resultat == 0 && strlen(dico.dico[milieu].mot) > TAILLE_MINIMALE) //si le mot existe
et que la taille du mot est supérieure à deux lettres
             {
                    return true; //le mot existe dans le dico
```

```
            }
            else if (resultat < 0) { //si le mot du milieu est inférieure au mot recherché,
                  //l'indice de debut devient l'indice de milieu, ainsi l'intervalle de recherche
est restreint
                  debut = milieu + 1;
            }
            else {//si le mot du milieu est supérieur au mot recherché,
                  //l'indice de fin devient l'indice de milieu, ainsi l'intervalle de recherche est
restreint
                  fin = milieu - 1;
            }
      }
      return false; //le mot n'existe pas dans le dico
}
```

Jeu.cpp :

```
/**
* @file Jeu.cpp
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composants pour le déroulement du jeu
* @date Semestre 1 Période B
*/


#include "Jeu.h"
#include "Robot.h"

/**
* @brief Initialise la partie et les joueurs
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] char** argv, pour obtenir le nombre de joueurs
* @return void
*/
void initialiser(Jeu& jeu, char** argv) {
      jeu.mot = NULL; //Initialise le mot pour la partie de jeu
      jeu.mot = new char[jeu.nbLettre + 1]; //Initialise dynamiquement le mot à une taille d'une
lettre
      jeu.nbJoueur = strlen(argv[1]); //Initialise le nombre de joueurs
      jeu.joueurs = new Joueur[jeu.nbJoueur]; //Initialise dynamiquement le nombre de joueurs
      // attribue aux joueurs leur type, leur numéro, leur nombre de points
      for (unsigned int i = 0; i < jeu.nbJoueur; ++i) {
            jeu.joueurs[i].type = argv[1][i];
            jeu.joueurs[i].NumJoueur = i + 1;
```

```
            jeu.joueurs[i].nbPoint = 0;
      }
}


/**
* @brief Entrez le mot à saisir et affiche si le mot existe ou pas ou si le mot commence par les
mêmes lettres
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @param[in] unsigned int& i, position du joueur
* @return void
*/
void bluff(Jeu& jeu, unsigned int& i, Dico& dico) {
      unsigned int y = i;
      joueurPrecedent(jeu, y);
      cout << jeu.joueurs[y].NumJoueur << jeu.joueurs[y].type << ",  saisir le mot > ";
      char* tmp = new char[ALPHABET];
      if (jeu.joueurs[y].type == 'H') { //si le type du joueur est humain on fait un cin
            cin >> tmp;
            cin.ignore(INT_MAX, '\n');
      }
      else { //si le type du joueur est robot , on appelle la fonction robot_choix_bluff(jeu,
dico,y)
            tmp = robot_choix_bluff(jeu, dico, y); // tmp est égal au mot trouvé
            //si le robot ne trouve pas de mots dans le dico, il renvoie "!" pour signifier sa
défaite

      }

      for (unsigned int y = 0; y < strlen(tmp); ++y)
            tmp[y] = toupper(tmp[y]); // met en majuscule
      if (verif2(tmp, jeu)) { //si le mot commence par les mêmes lettres que le mot du jeu
            if (verif_dichotomique(dico, dico.nbMot, tmp)) { // et si le mot existe dans le dico,
on affiche le message correspondant
                  cout << "le mot " << tmp << " existe, " << jeu.joueurs[i].NumJoueur <<
jeu.joueurs[i].type << " prend un quart de singe" << endl;
                  jeu.joueurs[i].nbPoint += 0.25; // ajoute au joueur courant un quart de singe
            }
            else {
                  cout << "le mot " << tmp << " n existe pas, " << jeu.joueurs[y].NumJoueur <<
jeu.joueurs[y].type << " prend un quart de singe" << endl;
                  jeu.joueurs[y].nbPoint += 0.25; // ajoute au joueur précédent un quart de singe
                  i = y; // la position du joueur repart du joueur précédent
            }
      }
      else {//si le mot ne commence pas par les mêmes lettres que le mot du jeu
            cout << "le mot " << tmp << " ne commence pas par les lettres attendues, " <<
jeu.joueurs[y].NumJoueur << jeu.joueurs[y].type << " prend un quart de singe" << endl;
            jeu.joueurs[y].nbPoint += 0.25;// ajoute au joueur précédent un quart de singe
            i = y;// la position du joueur repart du joueur précédent
      }
```

```
	//fin de la manche quand un joueur se prend un quart de singe
	afficher_manche(jeu); //appel la fonction afficher_manche(jeu) pour afficher la situation actuelle de la partie avec le nombre de points des joueurs
}


/**
* @brief Affiche que le mot existe s'il a été trouvé
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] unsigned int& i, position du joueur
* @return void
*/
void trouver(Jeu& jeu, unsigned int i) {
	cout << "le mot " << jeu.mot << " existe, " << jeu.joueurs[i].NumJoueur << jeu.joueurs[i].type << " prend un quart de singe" << endl;
	jeu.joueurs[i].nbPoint += 0.25; // ajoute au joueur courant un quart de singe
	//fin de la manche quand un joueur se prend un quart de singe
	afficher_manche(jeu);//appel la fonction afficher_manche(jeu) pour afficher la situation actuelle de la partie avec le nombre de points des joueurs
}



/**
* @brief Affiche que le joueur abandonne la manche
* @param[in-out] Jeu& jeu, référence du jeu, Joueur* joueurs, pointeur sur les joueurs
* @param[in] unsigned int i, position du joueur
* @return void
*/
void surrend(Jeu& jeu, Joueur* joueurs, unsigned int i) {
	cout << "le joueur " << joueurs[i].NumJoueur << joueurs[i].type << " abandonne la manche et prend un quart de singe" << endl;
	joueurs[i].nbPoint += 0.25;
	//fin de la manche quand un joueur se prend un quart de singe
	afficher_manche(jeu);//appel la fonction afficher_manche(jeu) pour afficher la situation actuelle de la partie avec le nombre de points des joueurs
}

/**
* @brief Affiche la situation actuelle de la partie avec le mot et le joueur courant
* @param[in-out] Jeu& jeu, référence du jeu
* @param[in] unsigned int i, la position du joueur
* @return void
*/
void afficher(Jeu& jeu, unsigned int i) {
	if (jeu.mot == NULL) //si le mot est vide, affiche en début de partie ou réinitialisation du mot
		cout << jeu.joueurs[i].NumJoueur << jeu.joueurs[i].type << ", ( ) > ";
	else //sinon affiche le mot de la manche et le joueur courant
	{
		cout << jeu.joueurs[i].NumJoueur << jeu.joueurs[i].type << ", (";
```

```
            for (unsigned int y = 0; y < jeu.nbLettre; ++y) {
                  cout << jeu.mot[y];
            }
            cout << ") > ";
      }
}


/**
* @brief Affiche la situation actuelle de la partie avec le nombre de points des joueurs
* @param[in-out] Jeu& jeu, référence du jeu
* @return void
*/
void afficher_manche(Jeu& jeu) {
      const char* virgule = "";
      //affiche les joueurs et leur nombre de points
      for (unsigned int i = 0; i < jeu.nbJoueur; ++i) {
            cout << virgule << jeu.joueurs[i].NumJoueur << jeu.joueurs[i].type << " : " <<
jeu.joueurs[i].nbPoint;
            virgule = "; ";
      }
      cout << endl;
      reset_mot(jeu); //réinitialise le mot de la manche
}

/**
* @brief Réalise une copie du mot de la manche
* @param[in-out] char*& mot, référence du mot de la manche
* @param[in] int taille, taille du mot
* @return void
*/
void Copie(char*& mot, int taille) {
      /* Alloue en mémoire dynamique un nouveau tableau
      * à cette taille*/
      char* copie = new char[taille + 1];
      // copie le mot dans le nouveau tableau copie
      for (int i = 0; i < taille; i++)
            copie[i] = mot[i];
      copie[taille] = '\0'; //rajouté pour pouvoir comparer avec les mots du dico
      delete[] mot; //désalloue l'ancien tableau pour le mot

      /* Actualise la mise à jour du conteneur en mémoire dynamique
      * Faites pointer le tableau support du conteneur
      * sur le nouveau tableau en mémoire dynamique */
      mot = copie;
}


/**
* @brief Vérifie si le mot rentré à bien les mêmes lettres que le mot de la manche
* @param[in-out] Jeu& jeu, référence du jeu
```

```
* @param[in] char mot[ALPHABET], le mot rentré
* @return bool , true si le mot commence par les mêmes lettres , false sinon
*/
bool verif2(char mot[ALPHABET], Jeu& jeu) {
	for (unsigned int i = 0; i < jeu.nbLettre; ++i) {
		if (mot[i] != jeu.mot[i]) { //vérifie si pour chaque position du mot rentré et du mot de la manche, on a bien les mêmes lettres
			return false; //le mot ne commence pas par les mêmes lettres
		}
	}
	return true; //le mot commence par les mêmes lettres
}


/**
* @brief Réinitialise le mot de la manche
* @param[in-out] Jeu& jeu, référence du jeu
* @return void
*/
void reset_mot(Jeu& jeu) {
	delete[] jeu.mot; // désalloue le tableau du mot de la manche
	//initialise le mot de la manche et le nombre de lettres à 0
	jeu.mot = NULL;
	jeu.nbLettre = 0;
	jeu.mot = new char[jeu.nbLettre + 1]; //Initialise dynamiquement le mot à une taille d'une lettre
}

/**
* @brief Devient le joueur précédent
* @param[in-out] Jeu& jeu, référence du jeu, unsigned int& i, position du joueur
* @return void
*/
void joueurPrecedent(Jeu& jeu, unsigned int& i) {
	if (i == 0) //si on est à la position du premier joueur
		i = jeu.nbJoueur - 1; // la position du joueur devient le dernier
	else
		--i; //sinon la position du joueur devient celle du précédent
}


/**
* @brief Vérifie si un joueur à 4 quart de singe
* @param[in-out] Jeu& jeu, référence du jeu
* @return bool, true si un joueur à 4 quarts de singe et false sinon
*/
bool eliminer(Jeu& jeu) {
	for (unsigned int i = 0; i < jeu.nbJoueur; ++i) {
		if (jeu.joueurs[i].nbPoint == 1) { //vérifie le nombre de points de chaque joueur est différent de 4 quarts de singe
			cout << "La partie est finie" << endl;
```

```
                    return true; //renvoie si un joueur a 4 quarts de singe
                }
        }
        return false; //renvoie si aucun joueur n'a 4 quarts de singe
}
```

Robot.cpp :

```
/**
* @file Robot.cpp
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @brief Composants pour les choix du robot
* @date Semestre 1 Période B
*/


#include "Robot.h"
#pragma warning(disable: 4996);

/**
* @brief Renvoie le choix du robot selon l'état de la partie
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @return char : une lettre ou "?" pour appeller bluff
*/
char robot_choix(Jeu& jeu, Dico& dico) {
        if (jeu.nbLettre == 0) { //si c'est le robot qui doit mettre la première lettre
                return 'A' + rand() % ('Z' - 'A' + 1); //renvoie une lettre aléatoire
        }
        char result = verif_robot(jeu, dico); //appel fonction verif_robot(jeu, dico) pour avoir le
choix du robot
        if (isalpha(result)) { //si le caractère est une lettre
                return result; // renvoie la lettre
        }
        else { //sinon si le caractère renvoyé n'est pas une lettre
                return '?'; //renvoie "?" pour que le robot bluff
        }


}

/**
* @brief Renvoie un caractère selon le nombre de mots que l'on peut former avec le mot du jeu
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @return char, un caractère
*/
char verif_robot(Jeu& jeu, Dico& dico) {
        //Un tableau de 26 entiers, un pour chaque lettre de l'alphabet
```

```
	int lettreCount[ALPHABET] = { 0 };
	int max_lettre = lettreCount[0];
	int taille = strlen(jeu.mot); //pour avoir la taille du préfixe du mot dans le jeu
	for (unsigned int i = 0; i < dico.nbMot; ++i) {
		if (strncmp(dico.dico[i].mot, jeu.mot, taille) == 0) { //permet de comparer deux
chaînes de caractères et de savoir si la première est égale à la seconde selon une taille précise
			char c = dico.dico[i].mot[taille]; //lettre juste après la taille du préfixe
			lettreCount[c - 'A']++; //plus un à la position du tableau de la lettre trouvée
		}
	}
	//On cherche la lettre avec le plus d'occurrence
	for (int i = 0; i < ALPHABET; i++) {
		if (max_lettre < lettreCount[i]) {
			max_lettre = lettreCount[i];
		}
	}
	//le compteur vérifie s'il y a une lettre ou plus dans le tableau
	int cpt = 0;
	for (int i = 0; i < ALPHABET; i++) {
		if (lettreCount[i] > 0)
			cpt++;
	}
	//si le cpt est égal à 0 ou 1, on ne pourra former un mot avec qu'une seule lettre

	if (max_lettre > SEUIL) { // le seuil à laquel le robot "?" ou pas
		for (int i = 0; i < ALPHABET; i++) {

			if (max_lettre == lettreCount[i]) { //le robot essaie de renvoyer la lettre avec
le plus d'occurrence
				char result[ALPHABET] = "";
				strcpy(result, jeu.mot);
				result[jeu.nbLettre] = char('A' + i); //forme le mot du jeu avec la lettre
				if (verif_dichotomique(dico, dico.nbMot, result) == false || cpt <= 1)
//vérifie si le mot formé existe ou pas et que l'on peut former un mot avec plus qu'une seule lettre
					return char('A' + i); //renvoie la lettre avec le plus d'occurrence

				else { //sinon le robot essaie de renvoyer une lettre aléatoire avec une
occurrence non nulle

					int j = rand() % ALPHABET + 1; // position de la lettre choisie
aléatoirement...
					int cpt1 = 0;
					while (verif_dichotomique(dico, dico.nbMot, result) == true) {
//vérifie, tant que le mot formé existe, on utilise une lettre différente
						if (lettreCount[j] > 0) { //si la lettre a au moins une
occurrence
							result[jeu.nbLettre] = char('A' + j); //teste la
lettre dans le mot formé
							cpt1++;// compte le nombre de tour de boucle
						}
						else {
```

```
                                                j = rand() % ALPHABET + 1; // position d'une lettre
aléatoire
                                          }
                                          if (cpt1 >= SEUIL) { // rand n'est pas complétement aléatoire
donc
                                                //cela pose un problème quand le programme ne tombe
jamais sur une lettre avec au moins une occurrence
                                                //on arrête selon un seuil le choix semi-aléatoire et
on parcours juste le tableau jusqu'à tomber sur une lettre avec au moins une occurrence
                                                for (unsigned int j = 0; j < ALPHABET; ++j) {
                                                      if (lettreCount[j] > 0) {
                                                            result[jeu.nbLettre] = char('A' + j);
                                                            if (verif_dichotomique(dico, dico.nbMot,
result) == false) {//vérifie si le mot formé existe ou pas
                                                                  char resulta = char('A' + j);
                                                                  j = 0;
                                                                  return resulta;//renvoie la
première lettre trouvée avec au moins une occurrence
                                                            }

                                                      }
                                                }
                                          }
                                    }
                                    char resulta = char('A' + j);
                                    j = 0;
                                    return resulta; //renvoie la lettre aléatoirement trouvée
                              }
                        }
                  }
            }
            else {
                  return char(max_lettre); // renvoie un caractère, préférablement qui n'est pas une
lettre
            }
}



/**
* @brief Renvoie le choix du robot lorsqu'il se fait bluff
* @param[in-out] Jeu& jeu, Dico& dico, référence du jeu et du dico
* @param[in] unsigned int y, position du joueur précédent
* @return char*, le mot trouvé dans le dico ou "!" si le robot ne trouve rien
*/
char* robot_choix_bluff(Jeu& jeu, Dico& dico, unsigned int y) {
      int taille = strlen(jeu.mot);
      for (unsigned int i = 0; i < dico.nbMot; ++i) {
            //permet de comparer deux chaînes de caractères et de savoir si la première est égale à
la seconde selon une taille précise
            //si un mot existe avec ce préfixe (= le mot du jeu) et que la taille du mot est
```

```
supérieure à deux lettres
            if (strncmp(dico.dico[i].mot, jeu.mot, taille) == 0 && strlen(dico.dico[i].mot) >
TAILLE_MINIMALE) {
                  return dico.dico[i].mot; //renvoie le mot trouvé dans le dico
            }
      }
      char str[] = "!";
      return str; //renvoie "!" pour abandonner si le robot ne trouve rien
}
```

Source.cpp :

```
/**
* @file Source.cpp
* @brief Projet Saé S1.02: Quart de singe
* @author John LI et Nassim BEN DAALI Grp 105
* @version 1.0
* @date Semesstre 1 Période B

*/

#include "Jeu.h"
#include "Robot.h"


/**
* @brief Lit à partir d'une entrée standard puis fait l'appel des fonctions entrées. Doit produire
des résultats en sortie
* prend comme paramètres argc et argv
*/
int main(int argc, char** argv) {
      char b; // le caractère stocké
      Jeu jeu;
      Dico dico;
      init_dico(dico);
      initialiser(jeu, argv);
      srand(time(NULL)); //initialise un générateur de nombres aléatoires
      do {
            for (unsigned int i = 0; i < jeu.nbJoueur; ++i) {
                  if (eliminer(jeu) == true) { //si un joueur a été éliminé pour avoir obtenu 4
quarts de singe
                        delete[] jeu.mot; //quand la partie se finit, on peut désallouer le mot de
la manche, le dico et les joueurs
                        jeu.mot = NULL;
                        delete[] dico.dico;
                        delete[] jeu.joueurs;
```

```
                        jeu.joueurs = NULL;
                        return 0; //fin de partie

                }
                afficher(jeu, i);
                if (jeu.joueurs[i].type == 'H') { //si le type du joueur est humain on lit à
partir d'une entrée standard
                        cin >> b;
                        cin.ignore(INT_MAX, '\n'); // pour prendre en compte que le premier
caractère
                }
                else {//si le type du joueur est robot on appelle la fonction
robot_choix(jeu,dico)
                        b = robot_choix(jeu, dico);
                }
                b = toupper(b); // met en majuscules le caractère

                if (b == '!') { // si le caractère est "!" alors on appelle surrend(jeu,
jeu.joueurs, i) pour abandonner la manche
                        surrend(jeu, jeu.joueurs, i);
                        --i;
                }
                else if (b == '?') { // si le caractère est "?" alors on appelle bluff(jeu, i,
dico) pour bluff le mot
                        bluff(jeu, i, dico);
                        --i;
                }
                else { // si le caractère est une lettre ou autre
                        jeu.mot[jeu.nbLettre] = b; // ajoute le caractère au mot de la manche
                        ++jeu.nbLettre;
                        Copie(jeu.mot, jeu.nbLettre);
                        if (verif_dichotomique(dico, dico.nbMot, jeu.mot)) { //vérifie que le mot
de la manche n'existe pas dans le dico
                                trouver(jeu, i); //si true , appel la fonction trouver(jeu, i) car
le mot a été trouvé
                                i = i - 1;
                        }
                }
        }
    } while (1); //tant qu'ancun joueur n'a pas obtenu 4 quarts de singe la partie continue
}


//TEST UNITAIRE
//int main(int argc, char** argv) {
       // le caractère stocké
      //Jeu jeu;
      //Dico dico;
      //init_dico(dico);
      //initialiser(jeu, argv);
      //srand(time(NULL)); //initialise un générateur de nombres aléatoires
```

```
	//char c;
	//for (unsigned int i = 0; i < 4; ++i) {
	//	if (i == 0) {
	//			c = 'a';
	//			jeu.mot[jeu.nbLettre] = c;
	//			++jeu.nbLettre;
	//			Copie(jeu.mot, jeu.nbLettre);
	//			verif_dichotomique(dico, dico.nbMot, jeu.mot);
	//			assert(verif_dichotomique(dico, dico.nbMot, jeu.mot) == false);
	//			char a[6] = "table";
	//			verif_dichotomique(dico, dico.nbMot, a);
	//			assert(verif_dichotomique(dico, dico.nbMot, jeu.mot) == true);
	//	}
	//	if (i == 1) {
	//			c = '?';
	//			float point = 0;
	//			bluff(jeu, i, dico);
	//			assert(jeu.joueurs[i].nbPoint - point == 0.25 || jeu.joueurs[i - 1].nbPoint -
point == 0.25);
	//	}
	//	if (i == 2) {
	//			c = '!';
	//			float point = 0;
	//			surrend(jeu, jeu.joueurs, i);
	//			assert(jeu.joueurs[i].nbPoint - point == 0.25);
	//
	//	}
	//	if (i == 3) {
	//			c = robot_choix(jeu, dico);
	//			assert(isalpha(c));
	//	}
	//}
```