BUT INFORMATIQUE - PARIS DESCARTES

Comparaison d’approches algorithmiques

Projet réalisé par Yohann PECH et Mehdi Bouziane du groupe 106

2022-2023

Table des matières

Présentation du projet :

Cet SAE d'initiation au développement correspond à notre deuxième travail en groupe en rapport à la programmation, mais notre premier en C++.

L'objectif du projet était de réaliser un jeu qui permettrait l'affrontement entre humain/humain, humain/robot ou encore robot/robot.

Le but du jeu était de former un mot du dictionnaire de la langue française en annonçant chacun son tour une lettre. Au fur et à mesure, les joueurs forment un mot, et si un joueur a un doute, il peut demander au joueur précèdent le mot auquel il pensait en saisissant le caractère '?'.

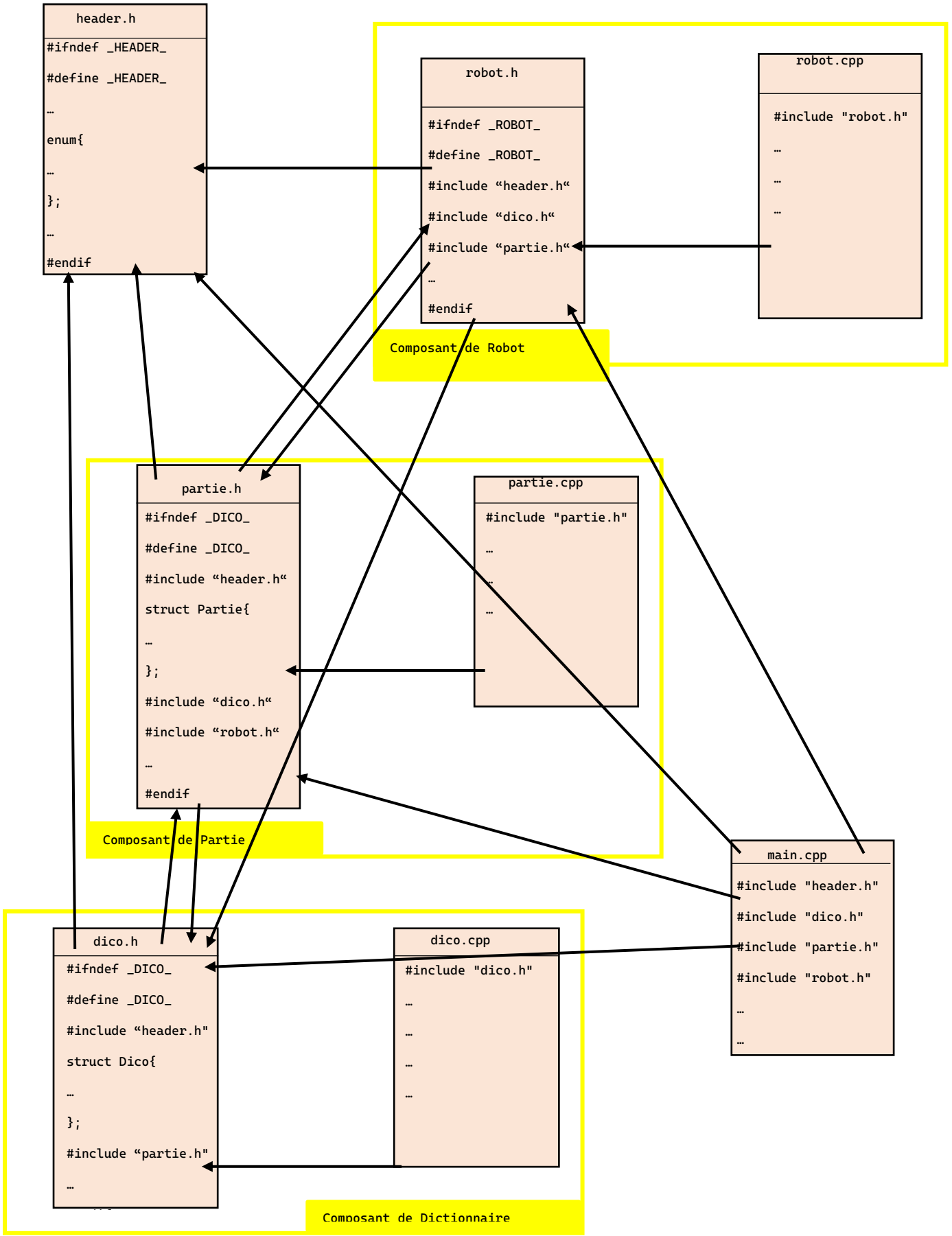
Le jeu se termine lorsqu'un seul joueur obtient quatre quarts de singe et dans ce cas-là, il perd. Un quart de singe peut s'obtenir :

- En donnant la dernière lettre de n'importe quel mot existant dans le dictionnaire fourni. (Par exemple, le mot formé dans la partie est 'ABR', et je pense au mot 'ABRICOT'. Alors je mets un 'I' mais la conséquence est que je prends un quart de singe, car j'ai formé le mot 'ABRI', qui est un mot existant dans le dictionnaire.
- En abandonnant la manche par la saisie du caractère ' !'
- En saisissant le caractère ' ?' et dans ce cas-là, le joueur demande au joueur précédemment le mot auquel il pensait, et il y a trois possibilités :
    - S'il renseigne un mot qui ne commence pas par les mêmes lettres que le mot formé dans la partie, alors le joueur précèdent obtient un quart de singe
    - Si le mot saisi existe dans le dictionnaire, alors le joueur qui a saisi ' ?' obtient un quart de singe
    - Si le mot saisi n'existe pas dans le dictionnaire, alors le joueur précèdent obtient un quart de singe

L'affichage du jeu se déroule de la manière suivante :

L'utilisateur doit saisir un nombre de joueurs qu'il précise avec la nature 'H' pour *humain* et 'R' pour *robot*. En fonction des joueurs saisi, le programme va les identifier : leur numéro suivi de leur nature. Par exemple si l'utilisateur a saisi 'HRH', alors la partie sera composée de 3 joueurs, le premier est un humain et sera nommé 1H, le deuxième est un robot nommé 2R, et la troisième 3H.

Graphe de dépendance des fichiers sources



Code source des tests unitaires :

Pour vérifier et tester notre programme, nous avons créé un code de test unitaire pour :

1) Le dictionnaire (dico.cpp et dico.h – voir fichier test_unitaire_dico.cpp dans le répertoire tests_unitaires) :

```
int main() {

  //Création du dictionnaire
  Dico dico;
  remplirDico(dico, "test_Dico.txt"); //Dictionnaire de test qui comporte quelques mots

  //Vérification du remplissage du dictionnaire
  // (qui contient des sauts de ligne, tabulation, ou espaces entre les mots
  // et des mots en minuscules et majuscules)
  assert(dico.tailleDico == 5);
  assert(strcmp(dico.mots[0], "AUREVOIR") == 0);
  assert(strcmp(dico.mots[1], "BONJOUR") == 0);
  assert(strcmp(dico.mots[2], "HELLO") == 0);
  assert(strcmp(dico.mots[3], "WORD") == 0);
  assert(strcmp(dico.mots[4], "ZOO") == 0);
  char mot[MAX_CHAR] = "Bonjour";
  unsigned int valeur = trouverMotDansDico(dico, mot);
  assert(valeur == 1);

  desallouerDico(dico);
  assert(dico.mots == NULL);
}
```

2) Le déroulement de la partie (partie.cpp et partie.h – voir fichier test_unitaire_partie.cpp dans le répertoire tests_unitaires) :

```
int main() {

  Partie p;
  char joueurs[MAX_CHAR] = "hhhh";
  // on teste avec une partie de 4 humains
  init(p, joueurs);

//Vérification des paramètres de la partie et des fonctions utiles au bon déroulement //de la
partie

  assert(p.nbJoueurs == strlen(joueurs));
  assert(p.taillemot == 0);
  assert(p.nouveauTour == FAUX);
  assert(p.dernierJoueurPerdant == -1);

  for (unsigned int i = 0; i < p.nbJoueurs; i++) {
    assert(p.score[i] == 0.);
    assert(p.ordre[i] == toupper(joueurs[i]));
  }
  int valeur3 = ajoutLettre(p, '!');
  assert(valeur3 == 2);
  valeur3 = ajoutLettre(p, '?');
  assert(valeur3 == 1);

  for (int i = 0; i < MAX_LENMOT; i++) {
    ajoutLettre(p, 'A' + i);

  for (unsigned int i = 0; i < MAX_LENMOT; i++) {
    assert(p.mot[i] == 'A' + i);
  }
  unsigned int valeur1 = verifMot(p, "test");
  assert(valeur1 == 1);

  for (unsigned int i = 0; i < 4; i++) {
    p.score[1] += 0.25;
  }
  unsigned int valeur2 = finduJeu(p);
  assert(valeur2 = 1);
```

```
  strcpy(p.motJoueurDeviner, "test");
  resetMotJoueur(p);
  for (unsigned int i = 0; i < MAX_LENMOT; i++) {
    assert(p.motJoueurDeviner[i] == CHAINE_VIDE);
  }

  detruireJeu(p);
  assert(p.ordre == NULL);
  assert(p.score == NULL);
}
```

3) Les robots (robot.cpp et robot.h – voir fichier test_unitaire_robot.cpp dans le répertoire tests_unitaires) :

```
int main() {

  char joueurs[MAX_CHAR] = "rrr";
  //on teste la partie avec 3 robots

  //Initialisation d'un dictionnaire et d'une partie pour tester les fonction utiles
  //aux robots
  Partie p;
  Dico dico;
  remplirDico(dico, "test_Dico.txt");

  init(p, joueurs);

  ajoutLettreRobot(p, 'A');
  assert(p.taillemot == 1);
  assert(strcmp(p.mot, "A") == 0);
  assert(p.motRobot[0] == CHAINE_VIDE);
  int indexMot = 0;
  unsigned int valeur1 = ChercherMotDico_Robot(dico, p, indexMot);
  assert(valeur1 == 0);
  assert(indexMot == 0);

  char* listeLettre = new char[MAX_CHAR];
  resetListeLettre(listeLettre);

  for (unsigned int i = 0; i < MAX_CHAR; i++) {
    assert(listeLettre[i] == CHAINE_VIDE);
  }

  unsigned int valeur3 = VerifLettreUtil('A', listeLettre);
  assert(valeur3 == 0);

  unsigned int valeur2 = AjoutLettreUtil('A', listeLettre);
  valeur3 = VerifLettreUtil('A', listeLettre);
  assert(valeur2 == 0);
  assert(valeur3 == 1);

  valeur2 = AjoutLettreUtil('A', listeLettre);
  assert(valeur2 == 1);
  assert(valeur3 == 1);
}
```

Bilan du projet :

Pour un deuxième projet, nous sommes satisfaits car nous pensons avoir accompli tous les objectifs demandés. Ce projet était d'une certaine complexité au niveau de l'algorithmie mais était aussi l'occasion de corriger les erreurs du projet précédent en nous initiant à la compilation séparée avec plusieurs fonctions sur plusieurs fichiers.

Nous avons rencontré des difficultés au niveau de la compréhension du sujet, par la présence de nombreuses règles à respecter. Le commencement était très rude car lorsqu'une partie du programme était fausse, l'erreur était très compliquée à trouver cependant l'utilisation du débuggeur nous a beaucoup servis.

De plus, la création des robots, plus précisément le fait de vouloir créer le meilleur robot qui soit afin qu'il ne perde le moins possible, était extrêmement complexe.

Mais nous avons su nous adapter, nous pensons avoir donnés le meilleur de nous-même afin de créer le meilleur quart de singe possible.

Après avoir énoncé la complexité du projet, nous sommes fiers de certaines choses : La création de la fonction *RemplirDico* est très utile, car elle nous permet de dupliquer le fichier texte qui contient le dictionnaire dans un tableau de pointeurs propre au programme.

La création de la fonction *LettreRobot qui* permet de générer une lettre aléatoire, en fonction du mot formé durant la partie, et des mots présents dans le dictionnaire qui est dans la structure *Dico.*

Pour conclure, nous sommes satisfaits car nous avons réussi à réaliser des parties entre humain ou entre robots et ainsi valider le test fournis par le professeur.

Annexe Dictionnaire :

dico.cpp

```
#include <fstream> // pour ifstream
#include <iomanip> // pour setw

#include "dico.h"

/**
 * @brief Remplir le dictionnaire
 * @param[in-out] idico : Le dictionnaire.
 * @param[in] fileName : Le fichier contenant le dictionnaire, si NULL le fichier est
ods4.txt.
 */
void remplirDico(Dico& idico, const char* fileName) {
  int TailleFixe = MAX;
  int Max = TailleFixe;
  int index = 0;
  char** pListMots = new char* [Max];

  if (fileName == NULL) {
    fileName = "ods4.txt";
  }
  ifstream in(fileName); // on ouvre le fichier

  char* pMot = new char[MAX_CHAR];
  in >> setw(MAX_CHAR) >> pMot;

  while (in) {
    if (strlen(pMot) > MIN_MOT) {
      for (unsigned int i = 0; i < strlen(pMot); i++) {
        pMot[i] = toupper(pMot[i]);
      }
      pListMots[index] = pMot;
      index++;
    }

    if (index == Max) {
      // On a atteint ma taille max, on réalloue un autre tableau avec la taille 2 x Max
      int NewMaw = Max + TailleFixe;
      char** pNewListMots = new char* [NewMaw];

      // On copie les anciens
      for (int i = 0; i < Max; i++)
        pNewListMots[i] = pListMots[i];
      // On détruit l'ancien
      delete[] pListMots;
      // On réaffecte le nouveau
      pListMots = pNewListMots;
      Max = NewMaw;
    }
    pMot = new char[MAX_CHAR];
    in >> setw(MAX_CHAR) >> pMot;
    if (!in)break;
  }
  idico.tailleDico = index;
  idico.mots = pListMots;
  in.close(); // on ferme le fichier
}

/**
 * @brief vérifie si le mot entré par le joueur existe dans le dictionnaire
 * @param[in-out] idico : Le dictionnaire
 * @param[in] trouve_mot : Le mot à comparer avec les mots du dictionnaire
 * @return oreturn : 1 si le mot a été trouve dans le dictionnaire, sinon 0 .
 */
unsigned int trouverMotDansDico(Dico& idico, char trouve_mot[]) {
  unsigned int oreturn = 0;
  for (unsigned int i = 0; i < strlen(trouve_mot); ++i)
```

```
    trouve_mot[i] = toupper(trouve_mot[i]);

  if (TRACE)
    cout << "trouve_Dico trouve_mot = " << trouve_mot << endl;

  for (int i = 0; i < idico.tailleDico; i++) {
    if (strncmp(trouve_mot, idico.mots[i], strlen(idico.mots[i])) == 0) {
      oreturn = 1;
      break;
    }
  }
  return oreturn;   // return 0 si le mot existe pas
}

/**
 * @brief Vérifier s'il existe un mot dans le dictionnaire qui commence par les mêmes lettres
que le mot passe en paramètre
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in] mot : Le mot a tester
 * @return oreturn : 1 s'il existe un mot commençant par les mêmes lettres, sinon 0.
 */
unsigned int motDebutExiste(Dico& idico, Partie& partie, const char mot[]) {
  unsigned int oreturn = 0;
  for (int i = 0; i < idico.tailleDico; ++i) {
    char* motDico = idico.mots[i];
    if (strncmp(mot, motDico, partie.taillemot + 1) == 0) {
      oreturn = 1;
      break;
    }
  }
  return oreturn;
}

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique
 * @param[in-out] idico : Le dictionnaire.
 */
void desallouerDico(Dico& idico) {
  for (int i = 0; i < idico.tailleDico; i++)
    delete[] idico.mots[i];
  delete[] idico.mots;
  idico.mots = NULL;

}
```

dico.h

```
#ifndef _DICO_
#define _DICO_

#include "header.h"

struct Dico {
  int tailleDico = 0; // le nombre de mots du dictionnaire
  char** mots = NULL; // le dictionnaire

};

#include "partie.h"
/**
 * @file dico.h
 * @author Yohann PECH - Mehdi BOUZIANE - Grp 106
 * @brief Toutes les fonctions relatives au dictionnaire.
 */


 /**
  * @brief Remplir le dictionnaire
  * @param[in-out] idico : Le dictionnaire.
```

```
   * @param[in] fileName : Le fichier contenant le dictionnaire, si NULL le fichier est
ods4.txt.
 */
void remplirDico(Dico& idico, const char* fileName);

/**
 * @brief verifie si le mot entré par le joueur existe dans le dictionnaire
 * @param[in-out] idico : Le dictionnaire
 * @param[in] trouve_mot : Le mot à comparer avec les mots du dictionnaire
 * @return oreturn : 1 si le mot a ete trouve dans le dictionnaire, sinon 0.
 */
unsigned int trouverMotDansDico(Dico& idico, char trouve_mot[]);

/**
 * @brief Vérifier s'il existe un mot dans le dictionnaire qui commence par les mêmes lettres
que le mot passe en paramètre
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in] mot : Le mot a tester
 * @return oreturn : 1 s'il existe un mot commençant par les mêmes lettres, sinon 0.
 */
unsigned int motDebutExiste(Dico& idico, Partie& partie, const char mot[]);

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique
 * @param[in-out] idico : Le dictionnaire.
 */
void desallouerDico(Dico& idico);

#endif
```

Annexe Partie :

partie.cpp

```
#include "partie.h"

#pragma warning(disable: 4996)

/**
 * @brief Afficher le score de la partie
 * @param[in] partie : La partie en cours
 * @param[in] indice : Indice du joueur prenant un quart de singe
 */
void afficheScore(const Partie& partie, const unsigned int indice) {
  partie.score[indice] += QDS;
  for (unsigned int i = 0; i < partie.nbJoueurs - 1; ++i)
    cout << i + 1 << partie.ordre[i] << " : " << partie.score[i] << "; ";
  cout << partie.nbJoueurs << partie.ordre[partie.nbJoueurs - 1] << " : " <<
partie.score[partie.nbJoueurs - 1] << endl;
}

/**
 * @brief Afficher le mot formé.
 * @param[in] partie : La partie en cours.
 */
void afficheTexte(const Partie& partie) {
  unsigned int i = 0;
  while (partie.mot[i] != CHAINE_VIDE) {
    cout << partie.mot[i];
    i++;
  }
}

/**
 * @brief Afficher le message d'abandon du joueur.
 * @param[in] partie : La partie en cours.
 * @param[in] indice : Indice du joueur qui abandonne
 */
void afficheAbandonJoueur(Partie& partie, unsigned int indice) {
  if (indice == partie.nbJoueurs)
    indice -= (partie.nbJoueurs - 1);
  cout << "le joueur " << indice + 1 << partie.ordre[indice] << " abandonne la manche et prend
un quart de singe" << endl; // si l'utilisateur rentre '!'
  partie.dernierJoueurPerdant = indice;
}

/**
 * @brief Afficher le message disant que le mot saisi n'existe pas
 * @param[in] partie: La partie en cours
 * @param[in] indice : Indice du joueur prenant un quart de singe
 * @param[in] mot : Mot qui n'existe pas dans le dictionnaire.
 */
void AfficheMotExistePas(Partie& partie, unsigned int indice, char mot[]) {
  if (indice + 1 == 1)
    indice = partie.nbJoueurs;
  cout << "le mot " << mot << " n'existe pas, " << indice << partie.ordre[indice - 1] << "
prend un quart de singe" << endl;
  partie.dernierJoueurPerdant = indice - 1;
}

/**
 * @brief Affiche le message disant que le mot existe dans le dictionnaire
 * @param[in] partie : La partie en cours
 * @param[in] indice : Indice du joueur qui va prendre un quart de singe
 * @param[in] trouve_mot : Mot qui existe dans le dictionnaire.
 */
void afficheMotExiste(Partie& partie, unsigned int indice, const char trouve_mot[]) {
  if (indice == partie.nbJoueurs)
```

```
    indice -= (partie.nbJoueurs);
  cout << "le mot " << trouve_mot << " existe, le joueur " << indice + 1 <<
partie.ordre[indice] << " prend un quart de singe" << endl;
  partie.dernierJoueurPerdant = indice;
}

/**
 * @brief Afficher le message disant que le mot saisi ne commence pas par les lettres
attendues
 * @param[in] partie : La partie en cours
 * @param[in] indice : Indice du joueur qui prend un quart de singe
 * @param[in] mot : Mot incorrect
 */
void AfficheDebutMotFaux(Partie& partie, unsigned int indice, const char mot[]) {
  if (indice + 1 == 1)
    indice = partie.nbJoueurs;
  cout << "le mot " << mot << " ne commence pas par les lettres attendues, le joueur " <<
indice << partie.ordre[indice - 1] << " prend un quart de singe" << endl;
  partie.dernierJoueurPerdant = indice - 1;
}

/**
 * @brief Afficher la fin du jeu.
 */
void AfficheGameOver() {
  cout << "La partie est finie" << endl;
}

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique
 * @param[in] idico : Le dictionnaire
 * @param[in] partie : La partie en cours.
 */
void EffacerJeu(Dico& idico, Partie& partie) {
  detruireJeu(partie);
  desallouerDico(idico);
}

/**
 * @brief Verifier si un joueur a obtenu quatre quarts de singe
 * @param[in] partie: La partie en cours.
 * @return oreturn : 1 si un joueur a obtenu quatre quarts de singe, sinon 0
 */
unsigned int finduJeu(Partie& partie) {
  unsigned int oreturn = 0;
  for (unsigned int i = 0; i < partie.nbJoueurs; ++i) {
    if (partie.score[i] == 1) {
      oreturn = 1;
      break;
    }
  }
  return oreturn;
}

/**
 * @brief Initialiser les structure
 * @param[in] partie : La partie en cours
 * @parame[in] pl : Chaine de caractères entrée sur la ligne de commande.
 */
void init(Partie& partie, char pl[]) {
  for (unsigned int i = 0; i < MAX_CHAR; i++) {
    partie.mot[i] = CHAINE_VIDE;
    partie.motJoueurDeviner[i] = CHAINE_VIDE;
  }
  partie.taillemot = 0;
  partie.nbJoueurs = 0;
  partie.dernierJoueurPerdant = -1;
  partie.nouveauTour = FAUX;
  partie.nbJoueurs = (unsigned int)strlen(pl);
  // pl est la chaine de caractères passée en paramètre de la fonction, et qui correspond à
argv[1]
  if (partie.nbJoueurs < MIN_JOUEURS) {
    cerr << "Pour jouer au quart de singe, il faut etre au moins 2 joueurs." << endl;
    exit(0);
  }
  partie.ordre = new char[partie.nbJoueurs];
```

```
  // tableau de char qui stockera le joueur suivant son type ( H ou R )
  partie.score = new float[partie.nbJoueurs];
  // tableau de float qui stockera les quart de singe
  for (unsigned int i = 0; i < partie.nbJoueurs; ++i) {
    partie.score[i] = 0.;
    partie.ordre[i] = toupper(pl[i]);
    if (partie.ordre[i] != HUMAIN && partie.ordre[i] != ROBOT) {
      cerr << "Le type de joueurs est incorrect." << endl << "Renseignez 'H' pour un Humain,
'R' pour un Robot." << endl;
      exit(0);
    }
  }
}

/**
 * @brief Permet a l'humain de jouer et de saisir une lettre ou '?' ou '!'
 * @param[in] partie : La partie en cours
 * @param[in] lettre : caractère que le joueur a saisi
 * @return oreturn : 0 si le joueur a saisi une lettre, 1 si le joueur a saisi '?' ou 2 si le
joueur a saisi '!'.
 */
int ajoutLettre(Partie& partie, char lettre) {
  unsigned int oreturn = 0;
  unsigned int i = 0;

  if (TRACE)
    cout << "ajoutLettre mot: " << lettre << endl;

  //le joueur a saisi '?'
  if (lettre == FINISH1) {

    if (TRACE)
      cout << "ajoutLettre mot == FINISH1" << endl;

    oreturn = 1;
  }
  else
    //le joueur a saisi '!'
    if (lettre == FINISH2)
      oreturn = VALUE2;
  if (oreturn == 0)
    AjoutLettreMot(partie, lettre);
  return oreturn;
}

/**
 * @brief Ajouter la lettre saisie au mot de la partie
 * @param[in-out] partie : La partie en cours
 * @param[in] lettre : La lettre à ajouter au mot.
 */
void AjoutLettreMot(Partie& partie, char lettre) {
  unsigned int i = 0;
  while (partie.mot[i] != CHAINE_VIDE) {
    i++;
  }
  if (partie.mot[i] == CHAINE_VIDE) {
    partie.mot[i] = toupper(lettre);
    partie.taillemot++;
  }
  partie.mot[i + 1] = '\0';
}

/**
 * @brief Affichage et gestion de la nouvelle manche
 * @param[in] idico : Le dictionnaire
 * @param[in] partie : La partie en cours.
 */
void nouvelleManche(Dico& idico, Partie& partie) {
  for (unsigned int i = 0; i < MAX_CHAR; i++) {
    partie.mot[i] = CHAINE_VIDE;
    partie.taillemot = 0;
  }
  int indexmotdudico = 0;
  if (partie.dernierJoueurPerdant + 1 > 1) {
    for (unsigned int i = partie.dernierJoueurPerdant; i < partie.nbJoueurs; ++i) {
      if (partie.nouveauTour == VRAI) {
```

```
        i = 0;
        partie.nouveauTour = FAUX;
      }

      if (finduJeu(partie)) {
        EffacerJeu(idico, partie);
        AfficheGameOver();
        exit(0);
      }


      cout << i + 1 << partie.ordre[i] << ", (";
      afficheTexte(partie);
      cout << ") > ";
      int letter = 0;
      if (partie.ordre[i] == HUMAIN) {

        char lettre;
        cin >> lettre;
        cin.ignore(INT_MAX, '\n');

        letter = ajoutLettre(partie, lettre);
      }
      else {
        letter = LettreRobot(idico, partie, i, indexmotdudico);
      }
      switch (letter) {
      case 0:
        if (trouverMotDansDico(idico, partie.mot)) {
          afficheMotExiste(partie, i, partie.mot);
          afficheScore(partie, i);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        break;
      case 1:
        // si le joueur a entré '?'
        saisirMot(idico, partie, i);
        if (verifMot(partie, partie.motJoueurDeviner)) {
          AfficheDebutMotFaux(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, partie.dernierJoueurPerdant);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        else if (trouverMotDansDico(idico, partie.motJoueurDeviner)) {
          afficheMotExiste(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, i);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        else {
          AfficheMotExistePas(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, partie.dernierJoueurPerdant);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        break;
      case 2:
        //si le joueur a entré '!'
        afficheAbandonJoueur(partie, i);
        afficheScore(partie, partie.dernierJoueurPerdant);

        indexmotdudico = 0;
        nouvelleManche(idico, partie);
        break;
      }
    }
  }
  // on met le bool a vrai
  partie.nouveauTour = VRAI;
}
```

```
/**
 * @brief cherchez si deux mots commencent par les mêmes lettres
 * @param[in] mot : le mot à comparer avec le mot de la partie
 * @return oreturn : 1 si les deux mots ne commencent pas par les mêmes lettres, sinon 0.
 */
int verifMot(Partie& partie, const char mot[]) {
  unsigned int oreturn = 0;
  for (unsigned int i = 0; i < strlen(partie.mot); ++i) {
    if (partie.mot[i] != mot[i]) {
      oreturn = 1;
      break;
    }
  }
  return oreturn;
}

/**
 * @brief Manche de la partie en cours
 * @param[in] idico : Le dictionnaire
 * @param[in] partie : La partie en cours.
 */
void mancheJeu(Dico& idico, Partie& partie) {
  int indexmotdudico = 0;
  while (partie.taillemot <= MAX_LENMOT) {
    for (unsigned int i = 0; i < partie.nbJoueurs; ++i) {

      if (TRACE)
        cout << "mancheJeu nouveauTour : " << partie.nouveauTour << endl;

      if (partie.nouveauTour == VRAI) {
        i = 0;
        partie.nouveauTour = FAUX;
      }

      if (finduJeu(partie)) {
        EffacerJeu(idico, partie);
        AfficheGameOver();
        exit(0);
      }


      cout << i + 1 << partie.ordre[i] << ", (";
      afficheTexte(partie);
      cout << ") > ";

      int letter = 0;
      if (partie.ordre[i] == HUMAIN) {

        char lettre;
        cin >> lettre;
        cin.ignore(INT_MAX, '\n');

        letter = ajoutLettre(partie, lettre);
      }
      else {

        if (TRACE)
          cout << "mancheJeu lettreRobot index: " << partie.ordre[i] << endl;

        letter = LettreRobot(idico, partie, i, indexmotdudico);
      }
      switch (letter) {
      case 0:

        if (trouverMotDansDico(idico, partie.mot)) {
          afficheMotExiste(partie, i, partie.mot);
          afficheScore(partie, i);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        break;
      case 1:
        // si le joueur a entré '?'
        // saisirMotHumain renvoie la chaine de caractères écrit par le joueur
        saisirMot(idico, partie, i);
```

```
        // réaffecte partie.motJoueurDeviner
        if (verifMot(partie, partie.motJoueurDeviner)) {
          AfficheDebutMotFaux(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, partie.dernierJoueurPerdant);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        else if (trouverMotDansDico(idico, partie.motJoueurDeviner)) {
          afficheMotExiste(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, i);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);

        }
        else {
          AfficheMotExistePas(partie, i, partie.motJoueurDeviner);
          afficheScore(partie, partie.dernierJoueurPerdant);

          indexmotdudico = 0;
          nouvelleManche(idico, partie);
        }
        break;
      case 2:
        //si le joueur a entré '!'
        afficheAbandonJoueur(partie, i);
        afficheScore(partie, partie.dernierJoueurPerdant);

        indexmotdudico = 0;
        nouvelleManche(idico, partie);
        break;

      }
    }
  }
}

/**
 * @brief Réinitialiser le mot à deviner dans la structure 'partie'
 * @param partie : La partie en cours.
 */
void resetMotJoueur(Partie& partie) {
  for (unsigned int i = 0; i < MAX_CHAR; ++i)
    partie.motJoueurDeviner[i] = CHAINE_VIDE;
}

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique dans la structure
'partie'
 * @param partie : La partie en cours.
 */
void detruireJeu(Partie& partie) {
  delete[] partie.ordre;
  partie.ordre = NULL;
  delete[]partie.score;
  partie.score = NULL;
}

/**
 * @brief Saisir un mot si le joueur précédent a saisi '?'
 * @param[in] idico : Le dictionnaire
 * @param[in] partie : La partie en cours
 * @param[in] indice : Indice du joueur qui doit saisir le mot.
 */
void saisirMot(Dico& idico, Partie& partie, unsigned int indice) {
  char trouve_mot[MAX_CHAR];
  // le premier joueur doit demander au dernier joueur
  if (indice == 0) {
    while (partie.ordre[indice] == HUMAIN || partie.ordre[indice] == ROBOT) {
      indice++;
    }
  }
  cout << indice << partie.ordre[indice - 1];
  cout << ", saisir le mot > ";
  if (partie.ordre[indice - 1] == HUMAIN) {
```

```
    cin >> trouve_mot;

    for (unsigned int i = 0; i < strlen(trouve_mot); ++i) {
      trouve_mot[i] = toupper(trouve_mot[i]);
    }
    resetMotJoueur(partie);
    for (unsigned int i = 0; i < strlen(trouve_mot); ++i) {
      partie.motJoueurDeviner[i] = trouve_mot[i];
    }

    return;
  }
  else {
    trouveMotDico_Robot(idico, partie);
    return;
  }
}
```

partie.h

```
#ifndef _PARTIE_
#define _PARTIE_

/**
 * @file partie.h
 * @author Yohann PECH - Mehdi BOUZIANE - Grp 106
 * @brief Toutes les fonctions relatives au bon fonctionnement du jeu.
 */

#include "header.h"

struct Partie {
  char mot[MAX_LENMOT]; // le mot de la manche
  unsigned int taillemot; // la taille du mot de la manche
  unsigned int nbJoueurs; // le nombre de joueurs de la partie
  unsigned int dernierJoueurPerdant; // le dernière joueur qui a écope un quart de singe
  BOOL nouveauTour;
  char motJoueurDeviner[MAX_LENMOT]; // mot a deviner
  char* ordre; // tableau qui stocke l'ordre des joueurs
  float* score; // tableau qui stocke les valeurs des quarts de singe
  char motRobot[1]; // la lettre du robot
};

#include "dico.h"
#include "robot.h"
/**
 * @brief Afficher le score de la partie
 * @param[in-out] partie : La partie en cours
 * @param[in] indice : Indice du joueur prenant un quart de singe
 */
void afficheScore(const Partie& partie, const unsigned int indice);

/**
 * @brief Afficher le mot formé.
 * @param[in-out] partie : La partie en cours.
 */
void afficheTexte(const Partie& partie);

/**
 * @brief Afficher le message d'abandon du joueur.
 * @param[in-out] partie : La partie en cours.
 * @param[in] indice : Indice du joueur qui abandonne
 */
void afficheAbandonJoueur(Partie& partie, unsigned int indice);

/**
 * @brief Afficher le message disant que le mot saisi n'existe pas
 * @param[in-out] partie: La partie en cours
 * @param[in] indice : Indice du joueur prenant un quart de singe
 * @param[in] mot : Mot qui n'existe pas dans le dictionnaire.
 */
```

```
void AfficheMotExistePas(Partie& partie, unsigned int indice, char mot[]);

/**
 * @brief Affiche le message disant que le mot existe dans le dictionnaire
 * @param[in-out] partie: La partie en cours
 * @param[in] indice : Indice du joueur qui va prendre un quart de singe
 * @param[in] trouve_mot : Mot qui existe dans le dictionnaire.
 */
void afficheMotExiste(Partie& partie, unsigned int indice, const char trouve_mot[]);

/**
 * @brief Afficher le message disant que le mot saisi ne commence pas par les lettres
attendues
 * @param[in-out] partie: La partie en cours
 * @param[in] indice : Indice du joueur qui prend un quart de singe
 * @param[in] mot : Mot incorrect
 */
void AfficheDebutMotFaux(Partie& partie, unsigned int indice, const char mot[]);

/**
 * @brief Afficher la fin du jeu.
 */
void AfficheGameOver();

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie: La partie en cours .
 */
void EffacerJeu(Dico& idico, Partie& partie);

/**
 * @brief Vérifier si un joueur a obtenu quatre quarts de singe
 * @param[in-out] partie : La partie en cours.
 * @return oreturn : 1 si un joueur a obtenu quatre quarts de singe, sinon 0
 */
unsigned int finduJeu(Partie& partie);

/**
 * @brief Initialiser les structures
 * @param[in-out] partie : La partie en cours
 * @parame[in] pl : Chaine de caractères entrée sur la ligne de commande.
 */
void init(Partie& partie, char pl[]);

/**
 * @brief Permet a l'humain de jouer et de saisir une lettre ou '?' ou '!'
 * @param[in] partie : La partie en cours
 * @param[in] lettre : caractère que le joueur a saisi
 * @return oreturn : 0 si le joueur a saisi une lettre, 1 si le joueur a saisi '?' ou 2 si le
joueur a saisi '!'.
 */
int ajoutLettre(Partie& partie, char lettre);

/**
 * @brief Ajouter la lettre saisie au mot de la partie
 * @param[in-out] partie : La partie en cours
 * @param[in] lettre : La lettre à ajouter au mot.
 */
void AjoutLettreMot(Partie& partie, char lettre);

/**
 * @brief Affichage et gestion de la nouvelle manche
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours.
 */
void nouvelleManche(Dico& idico, Partie& partie);

/**
 * @brief cherchez si deux mots commencent par les mêmes lettres
 * @param[in-out] partie : La partie en cours
 * @param[in] mot : le mot à comparer avec le mot de la partie
 * @return oreturn : 1 si les deux mots ne commencent pas par les mêmes lettres, sinon 0.
 */
int verifMot(Partie& partie, const char mot[]);
```

```
/**
 * @brief Manche de la partie en cours
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie: La partie en cours.
 */
void mancheJeu(Dico& idico, Partie& partie);

/**
 * @brief Reinitialiser le mot a deviner dans la structure
 * @param[in-out] partie : La partie en cours.
 */
void resetMotJoueur(Partie& partie);

/**
 * @brief Désallouer les zones mémoires allouées en mémoire dynamique dans la structure
'partie'
 * @param partie[in-out] : La partie en cours.
 */
void detruireJeu(Partie& partie);


/**
 * @brief Saisir un mot si le joueur précédent a saisi '?'
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in] indice : Indice du joueur qui doit saisir le mot.
 */
void saisirMot(Dico& idico, Partie& partie, unsigned int indice);

#endif
```

Annexe Robot :

robot.cpp

```
#include "robot.h"

#pragma warning(disable: 4996)

/**
 * @brief Trouver un mot pour le robot si le joueur précédent a entré '?', et
l'afficher
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie: La partie en cours
 */
void trouveMotDico_Robot(Dico& idico, Partie& partie) {
  int i = 0;
  for (i = 0; i < idico.tailleDico; ++i) {
    if (strncmp(idico.mots[i], partie.mot, partie.taillemot) == 0) {
      resetMotJoueur(partie);
      strcpy(partie.motJoueurDeviner, idico.mots[i]);
      cout << idico.mots[i] << endl;
      return;
    }
  }
  if (i == idico.tailleDico) {
    resetMotJoueur(partie);
    strcpy(partie.motJoueurDeviner, partie.mot);
    cout << partie.mot << endl;
  }
}

/**
 * @brief Ajouter la lettre du robot au mot de la partie
 * @param[in-out] partie : La partie en cours.
 * @param[in] LettreAleatoire : La lettre generee par le robot
 */
void ajoutLettreRobot(Partie& partie, const char LettreAleatoire) {
  unsigned int i = 0;
  while (partie.mot[i] != CHAINE_VIDE) {
    i++;
  }
  partie.taillemot++;
  partie.mot[i] = LettreAleatoire;
  partie.motRobot[0] = CHAINE_VIDE;
}

/**
 * @brief Générer une lettre aléatoire si le robot joue en premier
 * @param[in-out] partie: La partie en cours
 */
void LettreAleatoireRobot(Partie& partie) {
  char LettreAleatoire = 'A' + rand() % MAX_CHAR;
  cout << LettreAleatoire << endl;
  ajoutLettreRobot(partie, LettreAleatoire);
}

/**
 * @brief Permet au robot de joueur une lettre ou de demander au joueur precedent le mot qu'il
pensait
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in] indice : Indice du robot
```

```
 * @param[in-out] indexmotdudico : Index du mot dans le dictionnaire
 * @return num : 0 si le robot a pu genere une lettre, sinon 1.
 */
int LettreRobot(Dico& idico, Partie& partie, unsigned int indice, int& indexmotdudico) {
  if (partie.taillemot == 0) {
    LettreAleatoireRobot(partie);
    return 0;
  }
  int num = ChercherMotDico_Robot(idico, partie, indexmotdudico);
  if (num == 0) {
    cout << partie.motRobot[0] << endl;
    ajoutLettreRobot(partie, partie.motRobot[0]);
  }
  else {
    cout << "?" << endl;
  }
  return num;
}

/**
 * @brief Permet au robot de chercher une lettre qui completera le mot deja forme.
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in-out] indexmotdudico : Index du mot dans le dictionnaire
 * @return oreturn : 0 si le robot a reussi a trouve une lettre, sinon 1
 */
int ChercherMotDico_Robot(Dico& idico, Partie& partie, int& indexmotdudico) {
  unsigned int oreturn = 1;
  int i = indexmotdudico;
  while (i < idico.tailleDico) {
    char* pMotdudico = idico.mots[i];
    if (strlen(pMotdudico) > partie.taillemot) {
      char* motDico = new char[MAX_CHAR];
      strncpy(motDico, pMotdudico, partie.taillemot);

      if (strncmp(motDico, partie.mot, partie.taillemot) == 0) {
        indexmotdudico = i;

        if (TRACE)
          cout << "motDico trouve = " << motDico << " indexmotdudico: " << indexmotdudico <<
endl;
        unsigned int Compteur = 0;

        char* listeLettre = new char[MAX_CHAR];
        resetListeLettre(listeLettre);

        int motExiste = construireMot_Robot(idico, partie, motDico, Compteur, listeLettre);
        delete[] listeLettre;

        if (motExiste) {
          delete[] motDico;
          oreturn = 0;
          break;
        }
        else {
          delete[] motDico;
          oreturn = 0;
          break;
        }
      }
      delete[] motDico;
    }
    i++;
  }
  return oreturn;   // return 1 si le mot existe pas
}

/**
 * @brief Reinitialiser la liste qui comporte les lettres generees aleatoirement
 * @param[in-out] listeLettre : La liste .
 */
void resetListeLettre(char* listeLettre) {
  for (int i = 0; i < MAX_CHAR; ++i)
    listeLettre[i] = CHAINE_VIDE;
}
```

```
/**
 * @brief Verifier si les lettres generees aleatoirement n'ont pas encore ete generes
 * @param[in] lettre : La lettre generee aleatoirement
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si la lettre a deja ete generee, sinon 0.
 */
int VerifLettreUtil(char lettre, char* listeLettre) {
  unsigned int oreturn = 0;
  for (int i = 0; i < MAX_CHAR; ++i) {
    if (listeLettre[i] == lettre) {
      oreturn = 1;
      break;
    }
  }
  return oreturn;
}

/**
 * @brief Ajouter la lettre alatoire a listeLettre
 * @param[in] lettre : La lettre aleatoire
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si la lettre a deja ete generee, sinon 0.
 */
int AjoutLettreUtil(char lettre, char* listeLettre) {
  unsigned int oreturn = 0;
  unsigned int i = 0;
  while (listeLettre[i] != CHAINE_VIDE) {
    i++;
  }
  if (VerifLettreUtil(lettre, listeLettre))
    oreturn = 1;
  else {
    listeLettre[i] = lettre;
    oreturn = 0;
  }
  return oreturn;
}

/**
 * @brief Permet de generer une lettre aleatoire et de tester si la lettre construit un mot
existant.
 *
 * @param[in-out] idico : Le dicctionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in-out] mot : le mot du dictionnaire qui commence par les memes lettres que le mot
forme dans la partie
 * @param[in-out] nbNombre : Nombre de fois que la fonction va s'executer si la lettre generee
n'est pas bonne
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si le robot a reussi a construire un mot avec une lettre, sinon 0.
 */
int construireMot_Robot(Dico& idico, Partie& partie, char mot[], unsigned int& nbNombre, char*
listeLettre) {
  unsigned int oreturn = 0;
  char lettreRand = 'A' + rand() % MAX_CHAR;
  int lettreExiste = AjoutLettreUtil(lettreRand, listeLettre);
  int nbRand = 0;
  // lettreExiste == 1 si la lettre a deja été générée
  while (lettreExiste) {
    if (nbRand == MAX_CHAR)
      return 1;
    lettreRand = 'A' + rand() % MAX_CHAR;
    nbRand++;
    lettreExiste = AjoutLettreUtil(lettreRand, listeLettre);

    if (TRACE)
      cout << "verifDebutMot lettreExiste = " << lettreRand << endl;

  }
  if (mot)
    mot[partie.taillemot] = lettreRand;
  partie.motRobot[0] = lettreRand;
  if (nbNombre == MAX_CHAR)
    return 1;

  unsigned int motExiste = motDebutExiste(idico, partie, mot);
```

```
  // voir si il y a un mot dans le dictionnaire qui commencence par "mot"
  if (motExiste) {

    int motTrouve = trouverMotDansDico(idico, mot);
    if (motTrouve) {
      nbNombre++;
      oreturn = construireMot_Robot(idico, partie, mot, nbNombre, listeLettre);
    }
    else {
      return 1;
    }
  }
  else {
    nbNombre++;
    oreturn = construireMot_Robot(idico, partie, mot, nbNombre, listeLettre);
  }
  return oreturn;
}
```

robot.h

```
#ifndef _ROBOT_
#define _ROBOT_

/**
 * @file robot.h
 * @author Yohann PECH - Mehdi BOUZIANE - Grp 106
 * @brief Toutes les fonctions relatives au robot.
 */

#include "header.h"
#include "dico.h"
#include "partie.h"

/**
 * @brief Trouver un mot pour le robot si le joueur précédent a entré '?', et l'afficher
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie: La partie en cours
 */
void trouveMotDico_Robot(Dico& idico, Partie& partie);

/**
 * @brief Ajouter la lettre du robot au mot de la partie
 * @param[in-out] partie : La partie en cours.
 * @param[in] LettreAleatoire : La lettre generee par le robot
 */
void ajoutLettreRobot(Partie& partie, const char LettreAleatoire);

/**
 * @brief Générer une lettre aléatoire si le robot joue en premier
 * @param[in-out] partie: La partie en cours
 */
void LettreAleatoireRobot(Partie& partie);

/**
 * @brief Permet au robot de joueur une lettre ou de demander au joueur precedent le mot qu'il
pensait
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in] indice : Indice du robot
 * @param[in-out] indexmotdudico : Index du mot dans le dictionnaire
 * @return num : 0 si le robot a pu genere une lettre, sinon 1.
 */
int LettreRobot(Dico& idico, Partie& partie, unsigned int indice, int& indexmotdudico);

/**
 * @brief Permet au robot de chercher une lettre qui completera le mot deja forme.
 * @param[in-out] idico : Le dictionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in-out] indexmotdudico : Index du mot dans le dictionnaire
 * @return oreturn : 1 si le robot a reussi a trouve une lettre, sinon 0
```

```
 */
int ChercherMotDico_Robot(Dico& idico, Partie& partie, int& indexmotdudico);

/**
 * @brief Reinitialiser la liste qui comporte les lettres generees aleatoirement
 * @param[in-out] listeLettre : La liste .
 */
void resetListeLettre(char* listeLettre);

/**
 * @brief Verifier si les lettres generees aleatoirement n'ont pas encore ete generes
 * @param[in] lettre : La lettre generee aleatoirement
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si la lettre a deja ete generee, sinon 0.
 */
int VerifLettreUtil(char lettre, char* listeLettre);

/**
 * @brief Ajouter la lettre alatoire a listeLettre
 * @param[in] lettre : La lettre aleatoire
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si la lettre a deja ete generee, sinon 0.
 */
int AjoutLettreUtil(char lettre, char* listeLettre);

/**
 * @brief Permet de generer une lettre aleatoire et de tester si la lettre construit un mot
existant.
 *
 * @param[in-out] idico : Le dicctionnaire
 * @param[in-out] partie : La partie en cours
 * @param[in-out] mot : le mot du dictionnaire qui commence par les memes lettres que le mot
forme dans la partie
 * @param[in-out] nbNombre : Nombre de fois que la fonction va s'executer si la lettre generee
n'est pas bonne
 * @param[in-out] listeLettre : La liste de lettres
 * @return oreturn : 1 si le robot a reussi a construire un mot avec une lettre, sinon 0.
 */
int construireMot_Robot(Dico& idico, Partie& partie, char mot[], unsigned int& nbNombre, char*
listeLettre);

#endif
```

Annexe main et header

main.cpp

```cpp
#include <iostream>
#include <cassert>

#include "header.h"
#include "dico.h"
#include "partie.h"
#include "robot.h"

#pragma warning(disable:4996)

/**
 * @file main.cpp
 * @author Yohann PECH - Mehdi BOUZIANE - Grp 106
 * @brief fichier source.
 */

int main(int argc, char* argv[]) {
  srand((unsigned int)time(NULL));

  Dico dico;
  Partie p;
  remplirDico(dico, NULL);

  init(p, argv[1]);
  mancheJeu(dico, p);

}
```

header.h

```cpp
#ifndef _HEADER_
#define _HEADER_

/**
 * @file Header.h
 * @author Yohann PECH - Mehdi BOUZIANE - Grp 106
 * @brief Tout les types et structures utiles au jeu.
 */

#include <iostream>

using namespace std;

enum {
  MAX_CHAR = 26,
  MAX_LENMOT = 25,
};
#define FINISH1 '?'
#define FINISH2 '!'
#define HUMAIN 'H'
#define ROBOT 'R'
#define CHAINE_VIDE '\0'
#define VRAI 1
#define FAUX 0
#define QDS 0.25
#define MIN_JOUEURS 2
#define MAX 100
#define MIN_MOT 2
#define VALUE2 2
```

```
typedef int BOOL;

#define TRACE 0

#endif
```