Rapport Projet Logiciel-Devin

LE DEVIN

Professeur Encadrant

M. Nacer-Eddine ZERGAINOH

Equipes

Gaël LEMIERE
Hugues KOUAKOU
Octavian OLIVIER
Robin FARGES

IESE 3-GROUPE 1

INTRODUCTION

Objectifs

Développer un jeu, comportant une interface graphique, où l'ordinateur devine le personnage auquel l'utilisateur pense. L'ordinateur posera une suite de questions et tentera d'en déduire le choix de personnage du joueur.
Il sera possible d'ajouter manuellement un personnage dans le jeu. Cela permettra d'étoffer notre base de donnée au fur et à mesure.

Pour une meilleure organisation et pour partager des différentes versions de code, nous avons travaillé sur Discord et Github (pour les dépôts des codes).

PRÉSENTATION DU LOGICIEL

Le code a été découpée en plusieurs fonctions pour faciliter l'implémentation des fonctions et avoir un programme plus lisible pour tracker facilement les bugs et erreurs de compilation.
Le programme se comporte de différents fichiers: main.h, main.c, Populations.txt, questions.txt et historique.txt.

```
3   Cristiano Ronaldo |5 1 1 5 5 1 5 1 1 2 2 5 3 4 1 5 1 5 1 5 1 5 5 5 0 0 -1
4   Lionel Messi      |5 1 1 5 5 1 5 1 1 5 1 5 1 2 1 5 1 5 1 5 1 5 5 5 0 0 -1
5   Rafael Nadal      |5 1 1 5 5 1 5 1 2 1 5 2 4 2 4 2 5 1 5 5 1 5 5 5 0 0 -1
6   Roger Federer     |5 1 1 5 5 1 5 1 2 1 5 2 4 2 5 1 5 1 5 5 1 5 5 5 0 0 -1
7   Albert Einstein   |5 1 5 5 5 1 5 1 5 5 5 4 2 1 5 5 3 3 5 2 5 5 5 2 0 0 -1
8   Marie Curie       |5 1 5 5 1 1 5 2 5 4 5 1 5 1 1 3 3 3 5 4 5 1 3 2 0 0 -1
9   Jamy Gourmaud     |5 1 1 5 5 1 5 2 5 5 2 1 2 5 5 1 5 1 3 4 5 1 5 1 0 0 -1
10  Stephen Hawking   |5 1 5 5 5 1 5 1 5 5 1 5 1 5 5 2 5 1 4 4 5 5 5 1 0 0 -1
11  Rick Sanchez      |1 5 2 5 5 1 5 2 5 1 5 4 2 2 3 3 5 1 4 4 4 5 5 4 0 0 -1
12  Donald Trump      |5 1 1 5 5 1 5 1 5 1 5 1 5 2 1 5 5 1 2 1 4 5 3 5 0 0 -1
13  Emmanuel Macron   |5 1 1 5 5 1 5 2 2 4 2 5 2 5 5 1 1 5 4 1 5 1 5 5 0 0 -1
14  Vladimir Poutine  |5 1 1 5 5 1 4 1 5 5 1 4 2 5 2 2 1 5 5 1 2 5 5 5 0 0 -1
```

```
1   Votre personnage est-il fictif ?
2   Votre personnage est-il humain ?
3   Votre personnage est-il vivant ?
4   Votre personnage est-il mort depuis longtemps (+100ans) ?
5   Votre personnage est-il une femme ?
6   Votre personnage est-il de type caucasien ?
7   Votre personnage est-il de type asiatique ?
8   Votre personnage est-il célèbre ?
9   Votre personnage est-il jeune (-35ans) ?
10  Votre personnage est-il grand (+1,85m) ?
11  Votre personnage est-il petit (-1,65m) ?
12  Votre personnage est-il de corpulence forte (90kg<poids) ?
```

Les bases de données Populations.txt et questions.txt contiennent respectivement les personnages et les questions qui seront posées au fur et à mesure de l'avancement du jeu.
Le fichier historique.txt quant à lui, permet de sauvegarder l'historique du jeu à savoir: le nombre de parties jouées, le nombre de personnages bien devinés et le nombre de personnages rajoutés dans la base de données Populations.txt.

Avant de passer à la description des fonctions

L'idée était dans un premier temps de définir les structures d'enregistrement PERSONNAGE et QUESTION.

```
typedef struct personnage{
    int nbre_points;
    int num_personnage;
    struct personnage * suivant;
}PERSONNAGE;


typedef struct question{
    int nbre_points;
    int num_question;
    struct question * suivant;
}QUESTION;
```

Un PERSONNAGE est définit par son nombre de points, son numéro dans la base de données Populations.txt et un pointeur vers le PERSONNAGE suivant : une liste simplement chaînée nous a donc été utile pour stocker tous les personnages.

Une QUESTION est définit par son nombre de points (pertinence), son numéro dans la base de données questions.txt et un pointeur vers la QUESTION suivante : une liste simplement chaînée nous a donc été utile pour stocker toutes questions.

Le principe est de sélectionner la meilleure question (la plus pertinente), de la poser puis de calculer le nombre de points de chaque personnage en fonction de la réponse renseignée par l'utilisateur (entre 0 et 5). Une fois une question posée, on la supprime de la liste chaînée qui stocke les questions pour ne pas la reposer. Au fur et à mesure que la partie avance, on élimine les personnages dont le score est très faible (<=10) on présente le personnage au meilleur score à l'utilisateur qui dira si le programme a trouvé juste ou pas: 2 cas se présente dès lors.

Cas 1: Le programme a trouvé juste
Dans ce cas on met à jour le vecteur réponses correspondant au personnage deviné (remplacement des 0 par les réponses renseignées), on lui propose de rejouer.

Cas 2: Le programme n'a pas trouvé juste
Dans ce cas, on lui propose de continuer sur une seconde série de questions et le cycle reprend. S'il arrive qu'il n'y ait plus aucune question à poser ou qu'il reste un seul personnage en lice et que le programme n'a toujours pas trouvé le bon personnage alors, on

affiche à l'utilisateur tous les noms des personnages dans notre base de données: Si son personnage s'y trouve alors on lui demande quelle question aurait permis de deviner son

personnage et si son personnage n'est pas dans la liste alors on lui demande de saisir le nom de son personnage.
Suite à la saisie de son personnage, le vecteur réponse correspondant au personnage sont automatiquement mis à jour avec les réponses renseignées durant la partie.

Ci-dessous tous les prototypes de fonctions codées dans le cadre de ce projet:

```
39  int nombre_questions();
40  int nb_rep (int question, int rep);
41  void ajout_personnage(LISTE_SIMPLE *L,int n);
42  int nb_personnages();
43  void init_point_par_personnage(LISTE_SIMPLE *L);
44  void affichage(LISTE_SIMPLE L);
45  void suppression_personnage(LISTE_SIMPLE *L, int rang);
46  int nbre_a_eliminer(LISTE_SIMPLE L);
47  void Eliminer (LISTE_SIMPLE *L);
48  int taille(LISTE_SIMPLE L);
49  void point(PERSONNAGE*ptr,int num,int reponse,int meill_question);
50  int** calcul_point_par_personnage(LISTE_SIMPLE *L,int meill_question,int** tab);
51  void ajout_question(ListeSimple *L,int n);
52  void init_point_par_question(ListeSimple *L);
53  int nb_questions();
54  int nb_question_chaine(QUESTION *tete);
55  void affichage_questions(ListeSimple L);
56  void suppression_question(ListeSimple *L, int num);
57  void comptage (ListeSimple *L);
58  int nb_rep (int question, int rep);
59  int choix_question(ListeSimple *L);
60  void poser_question(int rang);
61  int personnage_trouve(LISTE_SIMPLE *L, int** tab);
62  void historique(int partie_joue,int bien_devine,int rajoute);
63  void affich_nom_personn(int num_pers);
64  void affich_list_pers();
65  void mise_a_jour_rep(int** tab, int num_pers);
66  void copy_with_0();
67  void ajouter_question_fichier();
68  void ajouter_personnage(int** tab,int nb_quiz);
69  void clean_stdin() ;
70  void remplir_personnage();
71  void affich_statistique();
```

Descriptions des fonctions

int nombre_questions()
Cette fonction sans paramètre retourne le nombre de questions dans la base de données questions.txt
int nb_rep(int question, int rep)
Elle prend en paramètres le numéro de la question (int question) et int rep (entre 0 et 5) puis retourne le nombre de personnages ayant la réponse int rep.
ex: int nb_rep(3, 4) va retourner le nombre de personnages ayant la réponse 4 à la 3ème question du fichier questions.txt
void ajout_personnage (LISTE_SIMPLE *L, int n)
Elle permet de rajouter le personnage n à la liste chaînée L.

int nb_personnages()
Cette fonction retourne le nombre de personnages dans la base de données Populations.txt

void init_point_par_personnage (LISTE_SIMPLE *L)
Permet de rajouter tous les personnages du fichier Populations.txt dans la liste chaînée L. Le nombre de point de chaque personnage est initialisé à 0.
void affichage (LISTE_SIMPLE L)
C'est une fonction teste qui a permis de bien vérifier le contenu de la liste chaînée de personnages.
void suppression_personnage (LISTE_SIMPLE *L, int rang)
Cette fonction permet de supprimer le personnage numéro int rang de la liste des personnages L.
int nbre_a_eliminer (LISTE_SIMPLE L)
Retourne le nombre de personnages à éliminer (critère d'élimination d'un personnage: nombre de points <=10)
void Eliminer (LISTE_SIMPLE *L)
Permet de supprimer tous les personnages dont le nombre de points <=10.
int taille (LISTE_SIMPLE L)
Retourne le nombre de personnages en lice
void point(PERSONNAGE*ptr,int num,int reponse,int meill_question)
cette fonction prend en paramètre un pointeur sur un PERSONNAGE, le N° du PERSONNAGE dans le fichier Population.txt
ainsi que la réponse entrée par le joueur(entre 0---5) puis calcule et met à jour son nombre de point suite à une question posée
int** calcul_point_par_personnage(LISTE_SIMPLE *L, int meill_question, int** tab)
Calcul et met à jour le nombre de points de chaque personnage après une question puis retourne le tableau tab2d qui contient les numéros des questions posées sur une colonne puis les réponses renseignées par l'utilisateur sur une colonne.
void ajout_question(ListeSimple *L,int n)
Rajoute la question n à la liste chaînée question
void init_point_par_question(ListeSimple *L)
Crée une liste simplement chaîné de toutes les questions du fichier questions.txt
int nb_questions()
Retourne le nombre de questions dans le fichier questions.txt
int nb_question_chaine(QUESTION *tete)
Récupère le nombre de questions qu'il y a dans la liste chainée
void affichage_questions(ListeSimple L)
Affiche le contenu de la liste chaînée des questions
void suppression_question(ListeSimple *L, int num)
Supprime la question numéro num de la liste chaînée des questions, une fois la question posée.
void comptage (ListeSimple *L)
Calcul et met à jour le nombre de points pour chaques questions de la liste chainée
int nb_rep (int question, int rep)

Retourne le nombre de personnages qui ont la réponse rep à la question numéro :int question

int choix_question(ListeSimple *L)

Retourne le rang de la question avec le nombre de points le plus élevé

void poser_question(int rang)

Affiche la question numéro rang

int personnage_trouve(LISTE_SIMPLE *L, int** tab)

Prend en paramètre la liste chaînée des personnages en lice ainsi que le tableau 2d tab contenant les réponses renseignées par l'utilisateur aux différentes questions.

retourne 1: si le personnage a bien été deviné

retourne 0: si le personnage n'a pas bien été deviné

retourne -1 si ouverture impossible du fichier

L'affichage du nom du personnage trouvé par le programme et la mise à jour du fichier historique.txt (incrémente de 1 le nombre de personnages bien devinées si le programme a trouvé juste) se font ici.

void historique(int partie_joue,int bien_devine,int rajoute)

historique(1,0,0) incrémente le nombre de parties jouées

historique(0,1,0) incrémente le nombre de personnages bien devinés

historique(0,0,1) incrémente le nombre de personnages rajoutés

void affich_nom_personn(int num_pers)

Affiche le nom du personnage N° num_pers

void affich_list_pers()

Affiche la liste des noms de tous les personnages contenus dans le fichier Populations.txt

void mise_a_jour_rep(int** tab, int num_pers)

Prend en paramètre le tableau 2d, contenant sur une colonne les numéros de questions puis sur l'autre les réponses renseignées par l'utilisateur à chaque question ainsi que le numéro d'un personnage (int num_pers).

Elle permet de mettre à jour le vecteur réponses du personnage N°num_pers au cas où le programme aurait bien deviné le personnage de l'utilisateur.

La mise à jour ici consiste à remplacer les 0 par les réponses renseignées par l'utilisateur.

void copy_with_0()

Permet de mettre la réponse correspondante à la nouvelle question rajoutée à 0 pour tous les vecteurs réponses

void ajouter_question_fichier()

Permet d'ajouter une question au fichier au fichier questions.txt et de mettre à jour dans le fichier Populations.txt la réponse correspondante à la question ajoutée (0).

void ajouter_personnage(int** tab,int nb_quiz)

Permet de rajouter un personnage: les réponses correspondant au personnage rajouté sont celles renseignées au cours de la partie

Les réponses aux questions qui n'ont pas été posées sont mises à 0

void clean_stdin()

Permet de vider le buffer, utile entre 2 scanf

void remplir_personnage()

Permet à l'utilisateur de rentrer un personnage, il devrait répondre aux différentes questions du fichier questions.txt afin de construire le vecteur réponses de son personnage

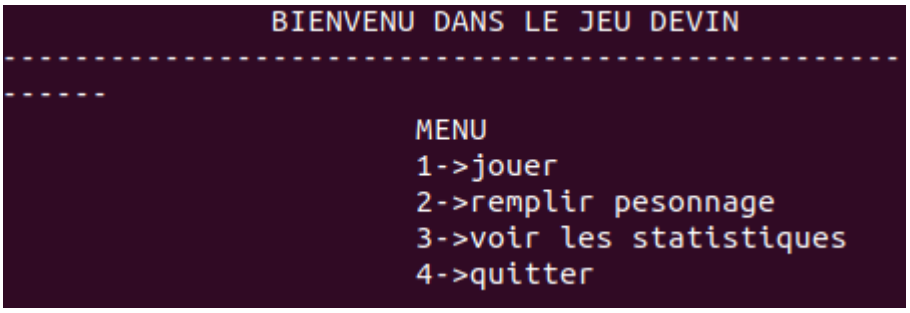
void affich_statistique()

Permet d'afficher les statistiques du jeu, nombre de parties jouées, nombre de personnages bien devinés, nombre de personnages rajoutés

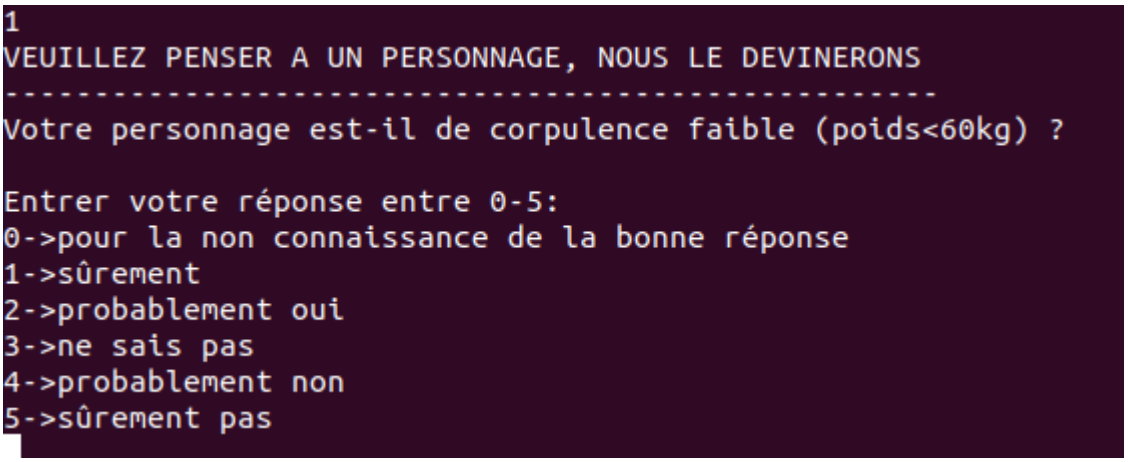
int main()

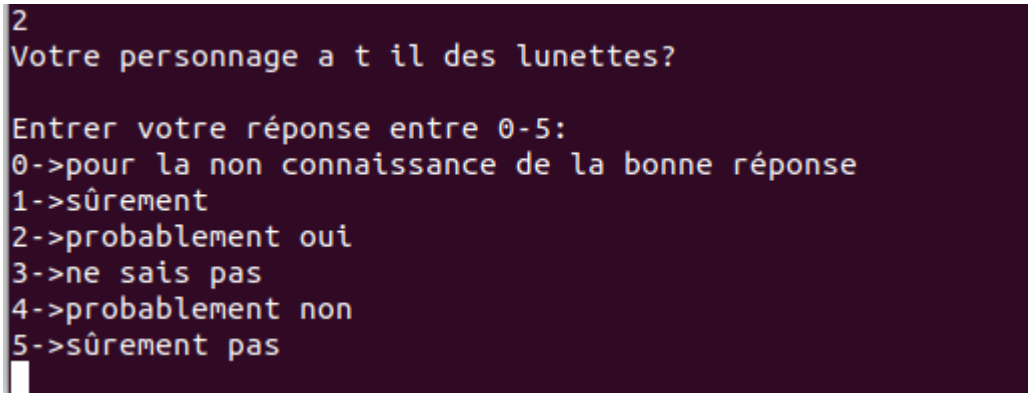
Au début du main on demande à l'utilisateur de choisir en 4 options à l'aide d'un switch-case:

1-> jouer
2-> remplir personnage
3-> voir les statistiques
4-> quitter

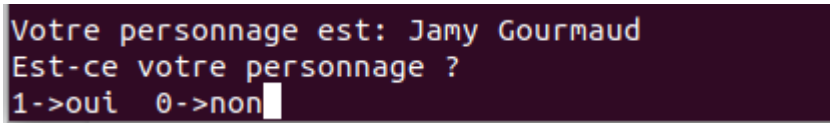


Les écrans qui s'affichent si l'utilisateur sélectionne l'option 1

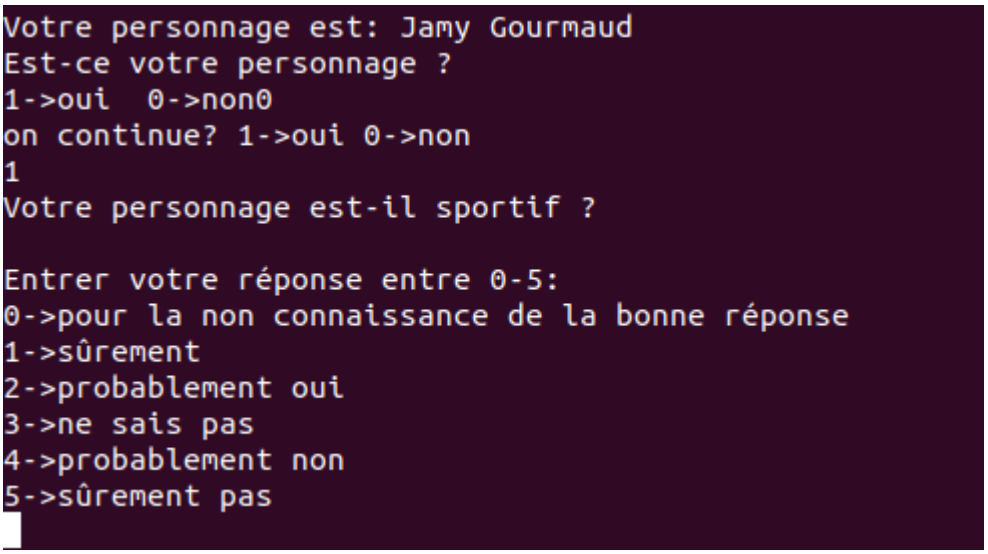




Le programme présente le personnag trouvé suite à une série de questions

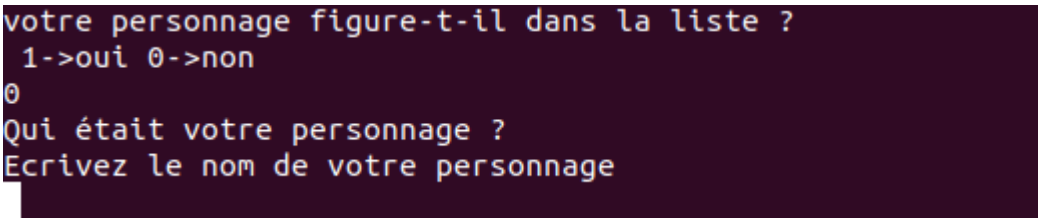


On sélectionne 0, on part alors sur une seconde série de questions



Tous les personnages ont été éliminés ou il n'y a plus de questions à poser, on affiche la liste des personnages

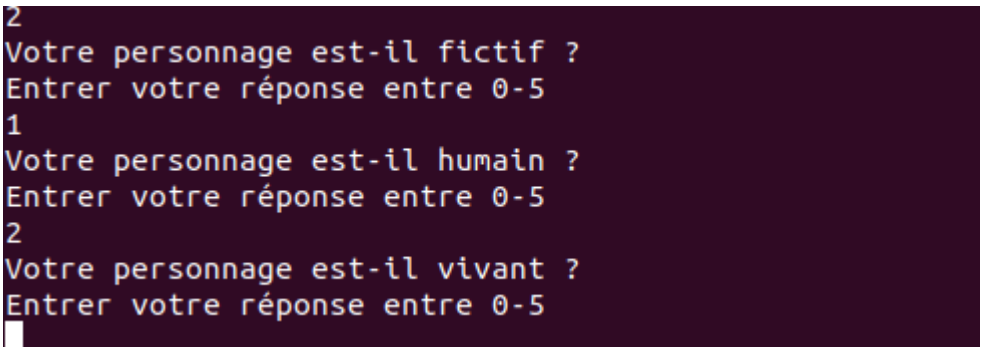
On sélectionne le 1 pour dire que le personnage figure dans la liste, on est alors invité à saisir le nom du personnage.



Si on avait sélectionner 0, alors l'utilisateur aurait été invité à saisir la question qui aurait permise de deviner son personnage.

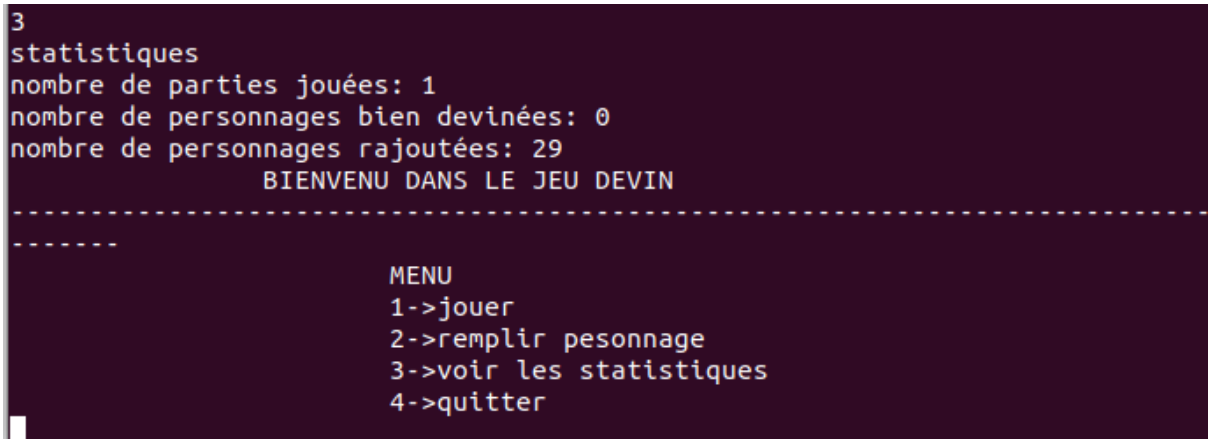
Les écrans qui s'affichent si l'utilisateur sélectionne l'option 2

On demande à l'utilisateur de renseigner la réponse pour chaque question



On l'invite à saisir le nom de son personnage pour finir, le personnage est alors rajouté à la base de données Populations.txt

Les écrans qui s'affichent si l'utilisateur sélectionne l'option 3



Une fois les statistiques affichées, on revient automatiquement dans le menu (comme toutes les options 1, 2)

COMPILATION

Makefile

```
CC=gcc

# Les différents répertoires contenant respectivement les fichiers :*.c,*.h,*.o,l'exécutable
SRCDIR=src
HEADDIR=include
BINDIR=bin


CFLAGS= -I$(HEADDIR) -g -Wall
GLLIBS = -lm

# L'exécutable
BIN=code

# trouver les différents sources *.c qu'il faudra compiler pour créer les objets correspondants
SRC= $(wildcard $(SRCDIR)/*.c)
OBJ= $(SRC:$(SRCDIR)/%.c=$(BINDIR)/%.o)

all: $(BIN)

#Création de l'exécutable
code: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS) $(GLLIBS)

# Créaation des différents *.o Ã  partir des *.c
$(BINDIR)/%.o: $(SRCDIR)/%.c $(HEADDIR)/%.h
    $(CC) -o $@ -c $< $(CFLAGS)


# Nettoyage des objets => Tout sera recompiler !
clean:
    rm $(BINDIR)/*.o

# Nettoyage complet => clean + effacement du l'exécutable
Clean: clean
    rm $(BINDIR)/*
```

Pour compiler:

1. make
2. ./code

Pour effacer les fichiers objets générés par la compilation: make clean

CONCLUSION

Ce projet nous a permis de mettre en oeuvre toutes les connaissances acquises au cours de l'année, tout en travaillant en équipe. Tout le partage du code s'est fait sur Github, lequel

nous avons appris à s'en servir avant de démarrer le projet. Nous avons pu réaliser un programme qui fonctionne, en accord avec le cahier des charges et avons ajouté quelques fonctions supplémentaires comme aller sur une seconde série de questions quand le programme n'a pas trouvé le personnage, garder l'historique du jeu...Nous sommes globalement satisfait du projet, cependant nous n'avons pas pu lier l'interface graphique codée en Python au programme en C de sorte à offrir une meilleure expérience utilisateur: cette partie peut s'inscrire dans les possibilités d'extension.

POSSIBILITÉS D'EXTENSION

1. Lier le code C à l'interface graphique qu'on a codée en Python pour avoir une meilleure expérience utilisateur.
   Nous avons réalisé notre ébauche d'interface graphique grâce à la bibliothèque libre Tkinter sur Python comme recommandé par notre professeur.
   Nous avons créé différents écrans avec différents boutons cliquables:
   - un écran menu :

   - un écran de jeu :

- <u>un écran d'ajout de personnage :</u> avec une case où taper le nom du personnage

- <u>un écran regroupant regroupant les statistiques du jeu:</u>

Nous avons malheureusement pas pu réaliser à temps la jonction entre notre jeu, programmé en C, et notre interface graphique programmée en python.

2. Construire un avatar à chaque personnage puis l'afficher lorsque le programme a trouvé le bon personnage (mais là un problème se pose, lorsque l'utilisateur rajoutera lui-même son personnage à la base de données aucun avatar ne lui correspondra).