TELECOM Nancy

Projet PPII

Wordle et solveur

Poiron Léa
Peronnin Edgar
Laurent Noé-Laurent
Galkowski Anna

Responsables de module :
Festor Olivier
Oster Gérald

Table des matières

Chapitre 1

Introduction

1.1 Contexte du projet

Ce projet a été réalisé dans le cadre du PPII (Projet Pluridisciplinaire d'Informatique Intégrative) au programme de la première année de la formation initiale sous statut étudiant (FISE) du cycle ingénieur de TELECOM Nancy.

Le projet est subdivisé en deux parties distinctes.
Le but de la première est de concevoir et d'implémenter une application Web reproduisant le populaire jeu en ligne Wordle. Le joueur doit avoir la possibilité de paramétrer ses parties en choisissant la longueur du mot à deviner et le nombre d'essais autorisés.
L'objectif de la seconde partie du projet est de réaliser un solveur interactif de ce jeu en langage C.

La réalisation de cette application et de ce solveur a été réalisée en mobilisant nos connaissances en matière de gestion de projet, d'algorithmique, de bases de données et de développement Web.

1.2 Organisation du document

Le chapitre suivant est un état de l'art qui présente le concept du jeu Wordle.

Le chapitre 3 est consacré à la partie conception et à l'implémentation de l'application Web. Il se compose d'une section dédiée à la création de la base de données nécessaire au fonctionnement de notre application Web, une autre à la partie développement Web et enfin une dernière à l'algorithmique.

Le chapitre 4, portant sur la partie solveur du projet, présente les structures de données et les algorithmes utilisés ainsi que les tests réalisés.

Le chapitre 5 est dédié à la gestion de projet. Nous y présenterons les différents outils utilisés pour le mener à bien.

Enfin dans un dernier chapitre nous allons effectuer un bilan général puis individuel du projet.

Certains document complémentaires concernant la gestion de projet seront disponibles en annexes.

1.3 Présentation du projet

Notre application Web est une version du jeu Wordle dans laquelle le joueur peut décider de la longueur du mot et du nombre de tentatives permises pour le trouver. L'application possède un système de login qui permet au joueur de créer un compte et ainsi de sauvegarder ses statistiques de jeu (nombre de parties jouées et nombre de parties gagnées). Il est également possible de jouer sans posséder de compte.

Le solveur implémenté en C proposera successivement à l'utilisateur des mots, ce dernier devant répondre par un pattern composé de 0, de 1 et de 2 pour indiquer la correspondance des lettres avec le mot auquel il pense (0 si la lettre n'est pas présente dans le mot, 1 si elle est mal placée et 2 si elle est bien placée).

Chapitre 2

État de l'art

2.1 Concept du jeu

Wordle est un jeu de lettres en ligne créé par Josh Wardle et sorti en octobre 2021. Le nom du jeu est un jeu de mots basé sur le nom de son créateur et le mot pour mot en anglais - word.

Le jeu consiste à deviner un mot de cinq lettres. L'utilisateur bénéficie de six chances, et, pour chaque tentative infructueuse, le jeu l'informe sur les lettres grâce à un système de couleurs :

- si la lettre n'appartient pas au mot à deviner : elle devient grise,
- si la lettre appartient au mot à deviner mais n'est pas bien placée : elle devient jaune,
- si la lettre appartient au mot à deviner et est bien placée : elle devient verte.

Chaque jour, tous les joueurs essayent de deviner un même mot-réponse, qui est sélectionné parmi les 2315 mots choisis par le développeur et stockés dans une base de données.

2.2 Un énorme succès

En quelques mois, ce jeu a rencontré une grande popularité. En effet, le nombre de joueurs quotidiens est passé de 90 le 1er novembre 2021 à plus 2 millions une semaine plus tard. Au cours des deux premières semaines de 2022, plus de 1,2 million de mentions du jeu sont apparues sur Twitter.

En plus de la gratuité et de l'absence de publicité, cette popularité est notamment due à la fonction partage sur les réseaux sociaux qui en ont fait une mode unissant des joueurs du monde entier.

Ce succès phénoménal a mené au développement de plus d'une centaine d'autres versions du jeu. Parmi elles, des traductions du jeu dans une soixantaine de langues mais aussi des versions sur un thème donné grâce à des bases de données plus réduites.

Chapitre 3

Jeu

3.1 Base de données

3.1.1 Tables

user

pseudo	mdp	parties_jouees	parties_gagnees

clef primaire : pseudo

pseudo (varchar(50)) : chaîne de caractère unique attribuée à l'utilisateur, monovaluée et obligatoire
mdp (varchar(50)) : mot de passe de l'utilisateur, monovalué et obligatoire
parties_jouees (entier) : nombre de parties jouées par l'utilisateur, monovalué et obligatoire
parties_gagnees (entier) : nombre de parties gagnées par l'utilisateur, monovalué et obligatoire

dico

mot	longueur

clef primaire : mot

mot (texte) : mot du dictionnaire, unique, obligatoire et monovalué
longueur (entier) : nombre de lettres du mot, monovalué et obligatoire

3.1.2 Forme normale de la base de données

Rappel

Une relation est en 1NF si chacun de ses attributs est atomique (non composé) et mono-valué.

Une relation R munie d'une clé primaire est en 2NF si

- elle est en 1NF
- tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé (DF partielle)

Une relation munie d'une clé primaire est en 3NF si

- elle est en 2NF
- tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut n'appartenant pas à la clé

user

user(pseudo, mdp, parties_jouees, parties_gagnees)

Chacun des attributs est atomique et monovalué, la relation user est donc en 1NF.
De plus, tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé, la relation user est donc en 2NF.
Finalement, tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut n'appartenant pas à la clé. En effet, seul un attrivut n'appartient pas à la clé, la relation user est donc en 3NF.

dico

dico(mot, cat1, cat2, cat3, longueur)

Chacun des attributs est atomique et monovalué, la relation dico est donc en 1NF.
De plus, tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé, la relation dico est donc en 2NF.
Finalement, tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut n'appartenant pas à la clé. En effet, seul un attribut n'appartient pas à la clé, la relation dico est donc en 3NF.

3.2 Web

On présente l'enchaînement globale des pages par un schéma :

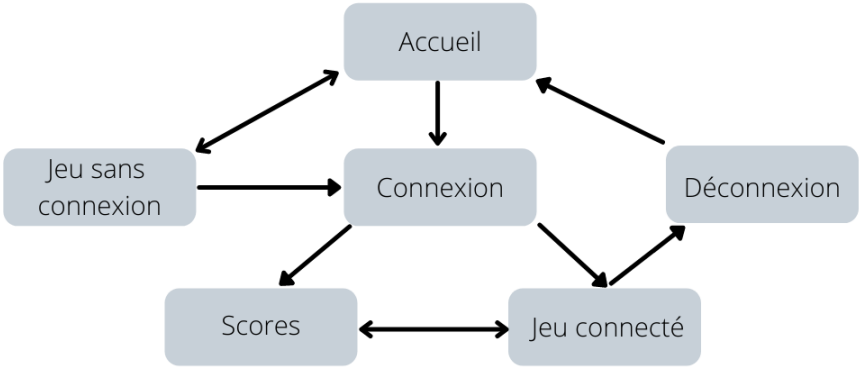

FIGURE 3.1 – Schéma de l'enchaînement des pages principales

Route	Commentaire
/	Redirige vers la route '/jeusanslogin'
/inscription	Page permettant au joueur de s'inscrire, possibilité d'accéder à la page de jeu sans connexion et à la page de connexion
/register	Route enregistrant l'inscription dans la base de données et renvoyant un message d'erreur en cas d'erreur et un message de validation si l'inscription est réussie
/login	Page permettant au joueur de se connecter, possibilité d'accéder à la page de jeu sans connexion et à la page d'inscription
/loginbis	Route vérifiant les données rentrées pour la connexion et renvoyant vers la page de jeu après connexion si la connexion est réussie et renvoie un message d'erreur en cas d'erreur d'autentification
/jeulogin	Page permettant de jouer après s'être connecté, elle permet également d'accéder à la page de consultation des scores et de se déconnecter
/regleco	Page proposant les règles du jeu et permettant d'accéder au jeu, aux scores et à la déconnexion
/historique_score	Page affichant les scores et permettant de retourner au jeu ou de se déconnecter
/deconnexion	Redirige vers la route '/jeusanslogin'
/deco	Route qui déconnecte l'utilisateur s'il accède à une page sans connexion
/règles	Page proposant les règles du jeu et permettant d'accéder au jeu, à l'inscription et à la connexion
/jeusanslogin	Permet de jouer à Wordle sans se connecter, possibilité d'accéder à la page de connexion et à la page d'inscription
/jslprov	Permet de réinitialiser la partie en cliquant sur jouer sans se connecter
/tente	Recharge la page pour actualiser les paramètres
/recupelettre	Récupère les entrées utilisateurs en Javascript (lettres, supressions, validation)

3.3 Algorithmique

La fonction **prop** prend en argument :

- le mot proposé par l'utilisateur (string)
- le mot à deviner (string)
- la longueur du mot à trouver (int)
- les lettres déja trouvées parmi les précédents essais (list)

Elle analyse le mot donné par l'utilisateur afin de trier les bonnes lettres, les mals placées et les mauvaises lettres.

La fonction retourne 4 listes :

- bonnes : Liste contenant les lettres bonnes depuis le début de la partie
- bonnesponctuel : Liste contenant les lettres bonnes sur cet essai
- malponctuel : Liste contenant les lettres mals placées sur cet essai
- faussesponctuel : Liste contenant les lettres fausses sur cet essai

Par conséquent, les listes bonnesponctuel, malponctuel et faussesponctuel sont complémentaires.

La fonction **prop** contient une fonction **malp** qui gère les lettres mal placées. Celle-ci prend en argument :

- les lettres mal placées (list)
- les lettres fausses (list)
- le mot proposé (list)
- le mot à trouver (list)
- une copie du mot proposé (list)
- la profondeur (int) (la fonction est récursive)

La fonction ne retourne rien puisqu'elle influe directement sur les listes.

Enfin, la fonction **couleurClavier** permet de gérer l'affichage des couleurs sur le clavier virtuel du jeu.

Elle prend en argument :

- l'alphabet (list)
- le clavier actuel (list)
- les bonnes lettres (list)
- les lettres mal placées (list)
- les fausses lettres (list)
- le mot proposé (list)

La fonction retourne le nouvelle liste clavier qui contient les couleurs des lettres dans l'ordre alphabétique.

3.4 Tests

Méthode

Les tests des algorithmes ont été réalisés à l'aide de pytest.

1. prop

Tests :

- -Si le mot à deviner est "deux" et que l'on propose "deux", alors la partie est bien gagné.
- -Si le mot à deviner est "bleu" et que l'on propose "deux", alors l'algorithme renvoie "e" et "u" comme lettres mals placées et "d" et "x" comme lettres non présentes dans le mot.
- -Si le mot à deviner est "protocole" et que l'on propose "appareils", alors l'algorithme renvoie "l" comme lettre bien placée, "p", "r" et "e" comme lettres mals placées et "a", "p", "a", "i" et "s" comme lettres non présentes dans le mot.
- -Si le mot à deviner est "minou" et que l'on propose "plage", alors l'algorithme renvoie "p", "l", "a", "g" et "e" comme lettres non présentes dans le mot.
- -Si dans ce même cas on propose "plage" comme mot alors que les lettres "m", "i" et "o" ont déjà été trouvées au bon emplacement, alors l'algorithme renvoie les lettres "m", "i" et "o" comme bien placées et les lettres "p", "l", "a", "g" et "e" comme lettres non présentes dans le mot.

2. couleurClavier

Tests :

- -Si le mot à deviner est "bleu" et que l'on propose "deux", alors l'algorithme colore "e" et "u" en orange et "d" et "x" en gris.
- -Si le mot à deviner est "protocole" et que l'on propose "appareils", alors l'algorithme colore "l" en vert, "p", "r" et "e" en orange et "a", "p", "a", "i" et "s" en gris.
- -Si le mot à deviner est "inégaux" et que l'on propose "iceberg", alors l'algorithme colore "i" en vert, "g" en orange et "c", "e", "b" et "r" en gris.
- -Si le mot à deviner est "inégaux" et que l'on propose "inactif", alors l'algorithme colore "i" et "n" en vert, "a" et "g" en orange et "c", "t", "i" et "f" en gris.
- -Si le mot à deviner est "inégaux" et que l'on propose "inégaux", alors l'algorithme colore "i", "n", "e", "g", "a", "u" et "x" en vert.

3.5 Complexité

sutom.py

prop

La complexité de la fonction prop est en O(n) où n est la longueur du mot proposé.

couleurClavier

La complexité de la fonction couleurClavier est en 0(n) où n est la longueur du mot proposé.

app.py

jeusanslogin / jeulogin

La complexité du jeu est en 0(n) où n est la longueur du mot proposé.

Chapitre 4

Solveur

4.1 Specification algébrique de la structure de données

4.1.1 Structures

On précise également que la structure principale, celle qui constitue notre dictionnaire est la structure : Liste de chaîne de caractères.

Liste d'entiers

elementint_t :

- int value
- elementint_t *next

listint_t :

- elementint_t *head

Liste de chaînes de caractères

element_t :

- char* ch1
- element_t *next

list_t :

- element_t *head

Liste de caractères

elementchar_t :

- char letter
- elementchar_t *next

listchar_t :

— elementchar_t *head

4.1.2 Signatures

On donne ici le type de la structure comme type de retour des opérations internes, cependant dans le code ces fonctions auront un type de retour void pour faciliter l'implémentation.
On considérera également que la fonction de création du dictionnaire ne fait que réserver l'espace et renvoyer un pointeur.
Dans le code source elle rempliera aussi le dictionnaire avec les mots de la taille spécifiée dans le wsolf.txt.

Liste d'entiers

Fonction	Paramètres	Type de retour	Type
create_listint	void	listint_t*	Opération interne
indexlistint	listint_t* X int	int	Observateur
listint_append	listint_t* X int	listint_t*	Opération interne
removeelementint	listint_t* X int	listint_t*	Opération interne
indexofmin	listint_t*	int	Observateur
max	listint_t*	int	Observateur
addone	listint_t* X int	listint_t*	Opération interne
findelementint	listint_t* X int	elementint_t*	Observateur

Liste de chaînes de caractères

Fonction	Paramètres	Type de retour	Type
create_dico	void	list_t	Opération interne
ajout_dico	char* X list_t*	list_t*	Opération interne
dico_print	list_t*	void	Observateur
isEmpty	list_t*	bool	Observateur
length	list_t	int	Observateur
findelement	list_t* X int	element_t*	Observateur
retire	element_t* X list_t*	list_t*	Opération interne
list_get	list_t* X int	char*	Observateur
list_index_of	list_t* X char*	int	Observateur
reduction_dico	char* X char* X list_t*	list_t*	Opération interne
char* wordfinder	list_t* X int	char*	Observateur

Liste de caractères

Fonction	Paramètres	Type de retour	Type
create_alphalist	void	listchar_t*	Opération interne
indexlistchar	listchar_t* X char	int	Observateur
listchar_get	listchar_t* X int	char*	Observateur
listchar_append	listchar_t* X char	listchar_t*	Opération interne
retirechar	int X listchar_t*	listchar_t*	Opération interne
printlistchar	listchar_t*	listchar_t*	Observateur

4.1.3 Préconditions

Liste d'entiers

indexlist_int(liste,entier) défini si entier appartient à *liste

indexofmin(liste) défini si *liste est non vide

max(liste) défini si *liste est non vide

findelementint(liste, index) défini si index est inférieur à la longueur de la *liste

Liste de chaînes de caractères

findelement(liste,index) défini si index inferieur à longueur liste

list_index_of(liste,mot) défini si mot appartient à liste

list_get(liste, indice) défini si la longueur de liste est supérieure à indice demandé

wordfinder(liste) est défini si liste est non vide

Liste de caractères

indexlistchar(liste,char) défini si char appartient à liste

listchar_get(liste,index) défini si index inférieur à logueur de liste

4.1.4 Axiomes

Liste d'entiers

- indexlist_int(create_listint(),0)=violation de précondition
- indexlist_int(listint_append(create_listint(),2),2)=0
- indexlist_int(removeelementint(listint_append(create_listint(),2),2),0)=violation de précondition
- indexlist_int(addone(listint_append(create_listint(),2),0),3)=0

- indexofmin(create_listint())=violation de précondition
- indexofmin(listint_append(create_listint(),2))=0
- indexofmin(removeelementint(listint_append(create_listint(),2),0))=violation de précondition
- indexofmin(addone(listint_append(create_listint(),2),0))=0
- max(create_listint())=violation de précondition
- max(listint_append(create_listint(),1))=1
- max(removeelementint(listint_append(create_listint(),1),0))=violation de précondition
- max(addone(listint_append(create_listint(),2),0))=3
- findelementint(create_listint(),0)=violation de précondition
- findelementint(listint_append(create_listint(),2),0)=2
- findelementint(removeelementint(listint_append(create_listint(),2),0),0)=violation de précondition
- findelementint(addone(listint_append(create_listint(),2),0),0)=3

Liste de chaînes de caractères

- dico_print(create_dico())="C'est vide"
- dico_print(ajout_dico("arbre",create_dico()))=[arbre]
- dico_print(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()))="C'est vide"
- dico_print(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())))=[arbre]
- isEmpty(create_dico())=true
- isEmpty(dico_print(ajout_dico("arbre",create_dico()))=false
- isEmpty(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()))=true
- isEmpty(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())))=false
- length(create_dico())=0
- length(ajout_dico("arbre",create_dico()))=1
- length(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()))=0
- length(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())))=1
- findelement(create_dico(),0)=violation de précondition
- findelement(ajout_dico("arbre",create_dico()),0)=pointeur vers l'élément contenant "arbre"
- findelementretire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()),0)=violation de précondition
- findelement(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())),0)=pointeur vers l'élément contenant "arbre"
- list_get(create_dico(),0)=violation de précondition
- list_get(ajout_dico("arbre",create_dico())),0)="arbre"
- list_get(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()))=violation de précondition
- list_get(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())),0)=arbre
- list_index_of(create_dico(),"arbre")=violation de précondition
- list_index_of(ajout_dico("arbre",create_dico())),"arbre")=0
- list_index_of(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()),"arbre")=violation de précondition

- list_index_of(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico())),"arbre")=0
- wordfinder(create_dico())=violation de précondition
- wordfinder(ajout_dico("arbre",create_dico()))="arbre"
- wordfinder(retire(findelement(ajout_dico("arbre",create_dico()),0),ajout_dico("arbre",create_dico()))=violation de précondition
- wordfinder(reduction_dico("arbre","22222",ajout_dico("arbre",create_dico()))="arbre"

Liste de caractères

- printlistchar(create_alphalist())="abcdefghijklmnopqrstuvwxyz"
- printlistchar(listchar_append(create_alphalist(),'a'))="abcdefghijklmnopqrstuvwxyza"
- printlistchar(retirchar(0,listchar_append(create_alphalist(),a)))="bcdefghijklmnopqrstuvwxyza"
- indexlistchar(create_alphalist(),'a')=0
- indexlistchar(create_alphalist(),'a')=0
- indexlistchar(retirchar(0,create_alphalist()),'a')=violation de précondition
- listchar_get(create_alphalist(),0)='a'
- listchar_get(listchar_append(create_alphalist(),'a'),0)='a'
- listchar_get(retirchar(0,listchar_append(create_alphalist(),a)),0)='b'

4.2 Algorithmique

Nom de la fonction	Objectif	Valeurs d'entrée	Type de retour
recupnb	Récuperer la longueur du mot dans wsolf.txt	Aucune	Entier
nb_letters	Récupérer le nombre de lettres dans un mot	Une chaîne de caractère (char*)	Entier
create_dico	Créer une liste de char* et y ajouter tout les mots de la longueur spécifiée dans le wsolf.txt	Aucune	list_t*
ajout_dico	Ajoute un mot dans une liste de char*	Un mot (char*) et un dictionnaire (list_t*)	void
dico_destroy	Libère toute la mémoire prise par un dictionnaire	Un dictionnaire (list_t*)	void
element_print	Affiche la valeur d'un élement d'un dictionnaire	Un élement du dictionnaire (element_t*)	void
dico_print	Affiche l'entièreté d'un dictionnaire	Un dictionnaire (list_t*)	void
isEmpty	Indique si un dictionnaire est vide ou non	Un dictionnaire (list_t*)	Booléen
length	Retourne la longueur d'un dictionnaire	Un dictionnaire (list_t*)	int
retire	Enlève un élement d'un dictionnaire	Un élement (element_t*) et un dictionnaire (list_t*)	void
list_get	Récupère le mot en i-ème position dans un dictionnaire	Un dictionnaire (list_t*) et un indice (int)	char*
list_index_of	Récupère l'indice d'un mot dans un dictionnaire	Un dictionnaire (list_t*) et un mot (char*)	Entier
list_element_of	Récupère l'élement associé à un mot dans un dictionnaire	Un dictionnaire (list_t*) et un mot (char*)	element_t*
occurrences	Compte le nombre de fois qu'une lettre apparaît dans un mot	Un mot (char*) et une lettre (char)	Entier
reduction_dico	A partir d'une réponse de l'utilisateur (le "pattern"), enlèvee tous les mots impossibles du dictionnaire	Un mot (char*), un pattern (char*), et un dictionnaire (list_t*)	list_t*
inttochar	Transforme un entier en une chaîne de caractère	Un nombre (int)	char*
wordfinder	Trouve le mot le plus efficace à proposer dans un dictionnaire	Un dictionnaire (list_t*) et le nombre d'essais (int)	char*
in	Indique si une lettre est dans un mot ou non	Un mot (char*) et une lettre (char)	Booléen
hereorafter	Donne le mot constitué des lettres situées après la i-ème lettre (incluse)	Un mot (char*) et un indice (int)	char*
hereorbefore	Donne le mot constitué des lettres situées avant la i-ème lettre (non incluse)	Un mot (char*) et un indice (int)	char*
indexletter	Retourne l'indice d'une lettre dans l'alphabet	Un mot (char*) et une lettre (char)	int
indexlistint	Retourne l'indice d'une valeur dans une liste d'entier	Une liste d'entiers (listint_t*) et une valeur (int)	int

findelementint	Retourner l'élement d'une liste d'entier associé à un indice	Une liste d'entiers (listint$_t$_$*$) et un indice (int)	elementint_t*
findelement	Retourner l'élement d'un dictionnaire associé à un indice	Un dictionnaire (list_t*) et un indice (int)	element_t*
elementint_print	Afficher la valeur d'un élement d'une liste d'entiers	Un élement de la liste d'entiers (elementint_t*)	void
printcountlist	Afficher l'entièreté d'une liste d'entiers	Une liste d'entiers (listint_t*)	void
elementchar_print	Afficher la valeur d'un élement d'une liste de caractères	Un élement de la liste de caractères (elementchar_t*)	void
printlistchar	Afficher l'entièreté d'une liste de caractères	Une liste de caractères (listchar_t*)	void
addone	Ajouter 1 à la i-ème valeur d'une liste d'entiers	Une liste d'entiers (listint_t*) et un indice (Entier)	void
indexofmin	Renvoyer l'indice du minimum dans une liste d'entiers	Une liste d'entiers (listint$_t*$)	int
max	Renvoyer la valeur du max dans une liste d'entiers	Une liste d'entiers (listint$_t*$)	int
removeelementint	Enlever le i-ème élement d'une liste d'entiers	Une liste d'entiers (listint_t*) et un indice (int)	void
removeletterinalphabet	Enlever le i-ème caractère d'une chaine de caractères	Un mot (char*) et un indice (int)	char*
best_letters	Chercher dans un dictionnaire les lettres les plus repétées et les classe dans une liste	Un dictionnaire (list_t*)	listchar_t*
create_alphalist	Créer une liste contenant les lettres de l'alphabet	Aucune	listchar_t*
listchar_get	Récupérer la i-ème lettre d'une liste de caractères	Une liste de caractères (listchar_t*) et un indice (int)	char*
retirechar	Enlever le i-ème élement d'une liste de caractères	Une liste de caractères (listchar_t*) et un indice (int)	void
listchar_append	Ajouter une lettre à une liste de caractères	Une liste de caractères (listchar_t*) et une lettre (char)	void
listint_append	Ajouter une lettre à une liste d'entiers	Une liste d'entiers (listint_t*) et une valeur (int)	void
listint_destroy	Libérer l'espace mémoire occupé par une liste d'entiers	Une liste d'entiers (listint_t*)	void
listchar_destroy	Libérer l'espace mémoire occupé par une liste de caractères	Une liste de caractères (listchar_t*)	void
savedwords	Donner le premier mot de chaque partie selon la longueur du mot	Une longueur (int)	char*
create_listint	Allouer la mémoire pour une liste d'entiers	Aucune	listint_t*

4.3 Analyse des structures de données

L'ensemble de nos structures de données représente donc 3 types de listes chaînées différentes. Nous ferons donc la même analyse pour toutes nos structures.

Avantages	Inconvénients
Place en mémoire → allocation à la "volée" Insertion/suppression	Place en mémoire → utilisation de cellules Accès aux données

4.4 Analyse des fonctions du solveur

4.4.1 Analyse en mémoire

Nous effectuerons cette analyse sur le dictionnaire comprenant tous les mots en 6 lettres de la langue française. Il y en a ici 17 991.

list_t* create_dico :

- Mémoire allouée : 575 712 octets (35 982 éléments), 8 octets (1 liste)
- Mémoire libérée : 287 856 octets (17 991 éléments)

void ajout_dico(char* mot, list_t *dico) :

- Mémoire allouée : 16 octets (1 élément)

void dico_destroy(list_t *dico) :

- Mémoire libérée : 287 856 octets (17 991 éléments), 8 octets (1 liste)

void retire(element_t *element,list_t* dico) :

- Mémoire libérée : 16 octets (1 élément)

list_t* reduction_dico(char* mot,char* pattern, list_t* dico) :

Imaginons que l'on retire 1 000 mots.

- Mémoire allouée : 271 856 (16 991 éléments), 8 octets (1 liste)
- Mémoire libérée : 287 856 octets (17 991 éléments), 8 octets (1 liste)

char* wordfinder(list_t *dico,int trynumber) :

- Mémoire allouée : 16 octets (2 listes)
- Mémoire libérée : 16 octets (2 listes)

void removeelementint(listint_t *countlist,int index) :

- Mémoire libérée : 16 octets (1 élément)

listchar_t* best_letters(list_t *dico) :

- Mémoire allouée : 416 octets (26 éléments), 24 octets (3 listes)
- Mémoire libérée : 16 octets (2 listes)

listchar_t* create_alphalist() :

- Mémoire allouée : 432 octets (27 éléments), 24 octets (3 listes)
- Mémoire libérée : 16 octets (2 listes)

listchar_t* create_alphalist() :

- Mémoire allouée : 416 octets (26 éléments), 8 octets (1 liste)

void retirechar(int index,listchar_t* listchar) :

- Mémoire libérée : 16 octets (1 élément)

void listchar_append(listchar_t* listchar, char letter) :

- Mémoire allouée : 16 octets (1 élément)

void listint_append(listint_t* wordscores,int score) :

- Mémoire allouée : 16 octets (1 élément)

void listint_destroy(listint_t *listint) :

- Mémoire libérée : 16*N octets (N éléments dans la liste), 8 octets (1 liste)

void listchar_destroy(listchar_t* listchar) :

- Mémoire libérée : 16*N octets (N éléments dans la liste), 8 octets (1 liste)

4.4.2 Analyse en complexité

Nom de la fonction	Complexité : Meilleur cas	Complexité : Pire cas
recupnb	O(1)	
nb_letters	O(1)	
create_dico	$O(N^2)$ où N est la taille du dictionnaire français	
ajout_dico	O(k) où k est la taille du dictionnaire en création	
dico_destroy	O(n) où n est la taille du dictionnaire	
element_print	O(1)	
dico_print	O(n) où n est la taille du dictionnaire	
isEmpty	O(1)	
length	O(n) où n est la taille du dictionnaire	
retire	O(1)	O(n) où n est la taille du dictionnaire
list_get	O(1)	O(n) où n est la taille du dictionnaire
list_index_of	O(1)	O(n) où n est la taille du dictionnaire
list_element_of	O(1)	O(n) où n est la taille du dictionnaire
occurrences	O(m) où m est le nombre de lettres du mot	
reduction_dico	O(nm)	$O(nm^2)$
inttochar	O(m) où m est le nombre de lettres du mot	
wordfinder	$O(n^2ma)$ où a est la taille de l'alphabet	
in	O(m) où m est le nombre de lettres du mot	
hereorafter	O(1)	
hereorbefore	O(1)	
indexletter	O(a) où a est la taille de l'alphabet	
indexlistint	O(n) où n est la longueur de la liste	
indexlistchar	O(n) où n est la longueur de la liste	
findelementint	O(n) où n est la valeur d'index	
findelement	O(n) où n est la valeur d'index	
elementint_print	O(1)	
printcountlist	O(n) où n est la longueur de la liste	
elementchar_print	O(1)	
printlistchar	O(n) où n est la longueur de la liste	
addone	O(n) où n est la valeur d'index	
indexofmin	O(n) où n est la longueur de la liste	
max	O(n) où n est la longueur de la liste	
removeelementint	O(1)	O(n) où n est la longueur de la liste ou la valeur d'index
removeletterinalphabet	O(1)	
best_letters	$O(n^2ma)$ où a est la taille de l'alphabet	
create_alphalist	O(a) où a est la taille de l'alphabet	
listchar_get	O(n) où n est la valeur d'index	
retirechar	O(1)	O(n) où n est la longueur de la liste ou la valeur d'index

Nom de la fonction	Complexité : Meilleur cas	Complexité : Pire cas
listchar_append	O(1)	O(n) où n est la longueur de la liste
listint_append	O(1)	O(n) où n est la longueur de la liste
listint_destroy	O(n) où n est la longueur de la liste	
listchar_destroy	O(n) où n est la longueur de la liste	
savedwords	O(1)	
create_listint	O(1)	

4.5 Tests

Les tests sont effectués grâce à la fonction assert. Pour chaque fonction nous testerons les cas classiques d'utilisation et les cas limites lorsque cela est pertinent.

La fonction **nb_letters** renvoie bien le nombre de lettre du mot (même s'il est vide) :

- assert(nb_letters("plage")== 5)
- assert(nb_letters("")==0)

La fonction **occurences** renvoie bien le nombre d'occurences d'une lettre dans un mot, que la lettre soit absente, présente :

- assert(occurrences("plage",'o')==0)
- assert(occurrences("monter",'o')==1)
- assert(occurrences("fauteuil",'u')==2)

La fonction **in** verifie bien qu'une lettre est présente dans un mot :

- assert(in('a',"plage"))
- assert(!in('o',"canape"))

La fonction **hereorafter** donne bien la chaîne de caractères à partir du rang demandé

- assert(strcmp(hereorafter("bonjour",0),"bonjour")==0)
- assert(strcmp(hereorafter("abcdefghijklmnopqrstuvwxyz",5),"fghijklmnopqrstuvwxyz")==0)
- assert(strcmp(hereorafter("plus",4),"")==0)

La fonction **hereorbefore** donne bien la chaîne de caractères de 0 au rang demandé :

- assert(strcmp(hereorbefore("abcdefghijklmnopqrstuvwxyz",5),"abcde")==0)
- assert(strcmp(hereorbefore("bonjour",7),"bonjour")==0) ;
- assert(strcmp(hereorbefore("bonjour",0),"")==0)

La fonction **indexletter** renvoie bien le rang de la lettre demandée dans la chaîne de caractère :

- assert(indexletter("abcdefghijklmnopqrstuvwxyz",'e') == 4)

La fonction **removeletterinalphabet** retire bien l'élément d'indice demandé

- assert(strcmp(removeletterinalphabet("abcdefghijklmnopqrstuvwxyz",5),"abcdeghijklmnopqrstuvwxyz")==0)

On considère pour la suite qu'on travaille sur les mots de longueur 5

La fonction **create_dico()** crée bien un dictionnaire non vide de la taille de la liste des mots de longueur 5, on verifie également ici le bon fonctionnement de la fonction length et de la fonction isEmpty.

— list_t* dico = create_dico() ;

assert(!isEmpty(dico)) ;

assert(length(dico)==7087) ;

La fonction **ajout_dico** ajoute bien un mot au dictionnaire, on le verifie grâce à la fonction length.

— char* mot = "test" ;

ajout_dico(mot,dico) ;

assert(length(dico)==7088) ;

Chapitre 5

Gestion de projet

5.1 Equipe de projet

L'équipe de projet se compose de 4 étudiants en première année :

- Galkowski Anna
- Laurent Noé-Laurent
- Peronnin Edgar
- Poiron Léa

Noé est désigné chef de projet, il a la responsabilité d'animer les réunions et de suivre l'avancée du projet.

L'équipe s'est réunie une fois par semaine minimum à partir du choix de l'application à développer afin de définir les objectifs et de se répartir le travail. Durant les vacances scolaires ces réunions ont eu lieu par le biais de la plateforme Discord.

Les documents relatifs au projet ont principalement été rédigés sur le serveur leaf de l'école. Cependant le groupe a également accès à un drive contenant les autres documents.

5.2 Analyse du projet

5.2.1 Définition des objectifs

Les objectifs ont été définis à l'aide de la méthode SMART :

S	Spécifique	L'objectif est simple et précis
M	Mesurable	L'objectif est vérifiable de manière qualitative ou quantitative
A	Accepté	L'objectif doit être défini avec la personne qui le réalise et non imposé
R	Réaliste	L'objectif doit être motivant sans se mettre en situation d'échec
T	Temporellement défini	L'objectif doit être inscrit dans le temps, avec une échéance précise

5.2.2 Analyse des risques : Matrice SWOT

Nous avons résumé les risques et les avantages de notre projet grâce à une matrice SWOT (Strength, Weaknesses, Opportunities, Threats), présentée en figure 7.1.

FIGURE 5.1 – Matrice SWOT

5.3 Organisation du projet

Le projet se déroule du mois d'avril 2022 jusqu'à la fin du mois de mai 2022. Une étape intermédiaire de présentation du site web est effectuée début mai.

5.3.1 Jalons du projet : GANTT

Nous avons réalisé deux diagrammes de GANTT afin d'organiser notre travail. Ces derniers sont présentés en figure 7.2.

FIGURE 5.2 – GANTT

5.3.2 Répartition des tâches : Matrice RACI

Jeu

Membres	Anna Galkowski	Noé-Laurent Laurent	Edgar Peronnin	Léa Poiron
Web : Squelette	CI	ACI	CI	R
Algo : Fonctions de base	CI	R	ACI	R
BD : Création des tables	R	CI	CI	ACI
Layout	ACI	CI	R	R
Login	CI	ACI	R	CI
Paramétrage	CI	R	ACI	CI
Connexion Web Algo	R	R	CI	ACI
Affichage Dynamique	CI	ACI	R	R
Connexion Web BD	R	CI	ACI	R
Tests	RCI	ARCI	RCI	RCI

Solveur

Membres	Anna Galkowski	Noé-Laurent Laurent	Edgar Peronnin	Léa Poiron
Structures de données	CIR	CIR	CIR	ACI
Fonction de bases	ACIR	CIR	CIR	CIR
Wordfinder	CI	CIR	CIA	CI
reducdico	CI	CIA	CIR	CIR
main	CIR	CIA	CI	CI
tests	CI	CIR	CIR	CIA

5.4 Réunions

Les comptes rendus des réunions sont présentés en annexe.

Chapitre 6

Bilan du projet

6.1 Bilan global

	Points positifs	Difficultés
Partie Base de Données	Structure simple	Aucune
Partie Web	Meilleure efficacité que sur le premier projet Connaissances en JavaScript	Liaison entre les différents langages (Python/Javascript)
Partie Algorithmique	Bonne maîtrise de python	Adaptation pour coller aux besoins du web et de JavaScript
Partie Solveur	Entraide pour implémenter et corriger les programmes	Gestion des erreurs en C
Partie Gestion de Projet	Meilleure efficacité que sur le premier projet	Gestion des délais

6.2 Bilan individuel

Léa Poiron

Points positifs	Meilleure compréhension de l'utilisation des structures de données
Difficultés rencontrées	Correction des fuites mémoire en C
Expériences personnelles	Développement des compétences en CSS Progression en C
Axes d'amélioration	Mieux comprendre les erreurs en C et les éviter

Edgar Peronnin

Points positifs	Apprentissage d'un nouveau langage
Difficultés rencontrées	Trouver les erreurs en C
Expériences personnelles	Découverte de JavaScript Amélioration en C
Axes d'amélioration	Une meilleure efficacité et compréhension du langage C

Noé-Laurent Laurent

Points positifs	Plus grande efficacité pour la partie web Découverte de deux nouveaux langages Complémentarité des rôles par rapport au premier projet
Difficultés rencontrées	Recherche des erreurs en C
Expériences personnelles	Découverte de JavaScript Amélioration en C Première fois en tant que chef de projet
Axes d'amélioration	Meilleure gestion du temps

Anna Galkowski

Points positifs	Approfondissement des connaissances en BD et en Web Initiation à de nouveaux langages
Difficultés rencontrées	Rechercher des erreurs en C
Expériences personnelles	Aide à la compréhension du langage C Découverte de JavaScript
Axes d'amélioration	Meilleure gestion du temps Mieux comprendre le langage C

6.3 Temps de travail

Composante	Anna Galkowski	Noé-Laurent Laurent	Edgar Peronnin	Léa Poiron
GDP	11h15	7h	8h	11h45
Réunions	3h10	3h10	3h10	3h10
Web	8h30	20h	20h55	20h05
BD	3h	15min	15min	15min
Algo	1h	3h15	1h	1h15
Solveur	10h30	37h25	13h	12h40
Total	35h35	70h	46h20	49h

Chapitre 7

Bibliographie

Etat de l'art

— https ://fr.wikipedia.org/wiki/Wordle

— https ://wordle.global/

— https ://rwmpelstilzchen.gitlab.io/wordles/

Développement

— https ://docs.python.org/fr/3/

— https ://stackoverflow.com/

Chapitre 8

Annexes

8.1 Comptes rendus des réunions

8.1.1 21 mars 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
19h20	50min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Ordre du jour

— Définition des objectifs
— Travail réalisé
— Travail à faire avant la prochaine réunion

Définition des objectifs

— Finir la partie gestion de projet en 3 jours
— Finir l'application web d'ici la fin de la première semaine de vacances (la semaine du 04 avril)
— Terminer les tests des algorithmes Python avant de commencer la partie solveur (ça devrait prendre moins d'une semaine)

Travail réalisé

— Cahier des charges (tout le monde)(fait à 100%)
— Début de la matrice SWOT (tout le monde)(fait à 75%)

Travail à faire avant la prochaine réunion

— Faire la matrice RACI (Noé)
— Faire le diagramme de GANTT (Edgar)
— Finir la matrice SWOT (Léa)
— Faire l'état de l'art (Anna)
— Faire la documentation : Web (Léa), BD (Anna), Algorithmes (Noé)

8.1.2 29 mars 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
16h	40min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail fait depuis la dernière réunion

- Faire la matrice RACI (Noé) : fait
- Faire le diagramme de GANTT (Edgar) : fait
- Finir la matrice SWOT (Léa) : fait
- Faire l'état de l'art (Anna) : 90 %, il sera fini complètement à la fin du projet
- Faire la documentation :
  - Web (Léa) : les chemins sont faits
  - BD (Anna) : fait
  - Algorithmes (Noé) : étude des algorithmes existants

Edgar est en train d'apprendre à utiliser JavaScript pour faire l'affichage dynamique

Pour la prochaine réunion qui se déroulera le vendredi 08/04 :

- envoyer mail pour Festor grand max avant ce week end
- Squelette du Web + Layout (Léa)
- Créer Base de données (Anna)
- Faire l'algo et commmencer à implémenter dans Web (Noé)
- Faire le système de login + Javascript (Edgar)

8.1.3 8 avril 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
14h15	23min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail fait depuis la dernière réunion :

- Edgar : avancement du Javascript (50%), n'a pas encore vu comment relier javascript et python ; a avancé les fonctions qui permettent d'afficher un clavier à l'écran
- Léa : a fait du css (fait à 55%), rencontre un petit problème au niveau des liens, a commencé à relire le programme Python, le squelette WEB est bon (fait à 90%)
- Anna : a complété l'état de l'art, a créé un fichier CSV avec tous les mots ainsi que la base de données qui n'est pas encore push car problème de branches sur git
- Noé : a fini sa partie algo, a commencé la partie solveur qui est assez complexe

Il est décidé que le document avec toutes les informations dans le mail à envoyer à Monsieur Festor sera sous format PDF.

Pour la prochaine réunion qui se déroulera le 15/08 :

- Envoyer le mail : tout le monde
- Connexion base de données/web : Léa et Anna
- Connexion web/algo : Noé et Anna
- Paramétrage : Noé
- Login et Javascript : Edgar
- CSS : Léa
- Documentation web : Léa
- Réfléchir au solveur : tout le monde

8.1.4 15 avril 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
21h30	30 min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail fait depuis la dernière réunion :

- Edgar : a fait en sorte que la touche entrée marche sur clavier virtuel, n'a pas encore réussi à lier javascript et python
- Léa : documentation web faite, connexion web algo faite, CSS (fait à 85%)
- Noé : a travaillé sur le solveur qui fonctionne, a réglé un pb, lié l'algo à au WEB et va le lier avec la BD, on peut jouer

Travail à faire avant prochaine réu (mercredi 20 avril)

- javascript (Edgar et Anna)
- login (Léa)
- css à finir (Léa)
- tests à faire en fin de semaine prochaine
- guide pour lancer l'appli (Léa)

8.1.5 20 avril 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
15h00	25min	Présentiel	Léa Poiron (secrétaire), Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski	

Travail réalisé depuis la dernière fois

- Léa CSS avancé mais pas fini car les modification ne s'affiche pas instantanément (fait à 90%) + login fini + guide pour lancer l'appli fini
- Edgar réglage de problèmes avec le code java script, pb de gestion des paramètres
- Anna CSS clavier et cases (90%) problème sur les cases à régler + correction de fautes dans le README
- Noé à lié le script à la page web et à géré des problèmes au niveau des pages html + première version du solveur en python (fait à 95%)

Travail à faire avant la prochaine réunion

- Connexion javascript python pour gérer les paramètres (Tout le monde)
- Mettre les couleurs sur les cases (Anna)(après connexion avec python)
- Finir le CSS (Léa)
- Correction de bug (Noé)

8.1.6 24 avril 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
20h36	15min	Distantiel	Léa Poiron (secrétaire), Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski	

Travail fait depuis la dernière réunion

- CSS (Léa) : certaines modifs ne sont pas prises en compte ce qui rend le travail plus long que prévu (fait à 95%)
- Liaison JavaScript et python (Anna et Edgar) : sans succes (fait à 0%)
- Faire en sorte que cliquer sur les cases du clavier à l'écran affiche les lettres(fait à 5%)
- Recherche de Bibliothèque python pour lier avec JavaScriptn (Noé)
- Création d'une version fonctionnant sans JavaScript au cas où la liaison serait impossible (Noé)(fait à 100%)
- Relire le programme python (Edgar)(Fait à 100%)

Pour la prochaine réunion qui se déroulera le 04/05 :

- Effectuer les tests (Tout le monde)

8.1.7 4 mai 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
18h	24min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail fait depuis la dernière réunion

— Début des tests (Léa) : problème pour les lancer (fait à 20%)

Organisation du travail

On décide de se reunir 2 fois par semaine pour pouvoir mettre les idées en commun (Il ne s'agira pas nécessairement de réunions, cela peut-être des séances de travail)

3 principaux objectifs pour tout le monde :

— Finir les tests des algos

— Travailler le C

— Commencer la rédaction du rapport

Travail à faire pour la prochaine séance de travail (jeudi 12 mai)

— Se documenter sur l'implementation d'un solveur en C

— Se mettre au clair sur la répartition pour la rédaction du rapport

— Echanger le plus possibles sur les idées et questionnements

8.1.8 11 mai 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
11h21	45min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail effectué depuis la dernière réunion

- Se documenter sur l'implementation d'un solveur en C
- Se mettre au clair sur la répartition pour la rédaction du rapport (100%)

On commence par réaliser un plan du rapport de projet pour noter la répartition les différentes parties à rédiger.

- **Intro**
- **Site :**
  - Web à compléter : Léa
  - BD déjà fait par Anna
  - Algo : Noé
  - Tests : Edgar
- **Solveur :**
  - Algo
  - Tests
- **GDP**

Ensuite, nous réfléchissons à la conception du solveur. Nous organisons une séance de travail le jeudi 12/05 (le lendemain) pour décider de la méthode et implémenter les premières fonctions.

8.1.9 22 mai 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
18h	30 min	Distanciel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail effectué depuis la dernière réunion

- Léa : a écrit la fonction qui permet de réduire le dico (70%), et mis à jour la documentation Web (100%)
- Edgar : a travaillé sur le solveur
- Anna : a fait l'intro du projet et une partie de la bibliographie (50%)
- Noé : a créé des fonctions du solveur, a fait la documentation algorithmique

On va adapter le solveur en python que l'on a en langage C.

Travail à faire pour le solveur :

- Finir la fonction qui réduit le dico (d'ici mercredi soir) (Léa, Edgar)
- Créer la fonction qui trouve le mot à donner (Noé)
- Faire le main (Anna)
- Faire la documentation des tests des algorithmes (Edgar)

Il faudrait finir la partie Gestion de Projet du rapport d'ici jeudi 26 mai pour ensuite se focaliser entièrement sur le solveur.

8.1.10 1er juin 2022

Heure	Durée	Présentiel ou distanciel	Présent	Absent
12h	20min	Présentiel	Léa Poiron, Noé-Laurent Laurent, Edgar Peronnin, Anna Galkowski (secrétaire)	

Travail effectué depuis la dernière réunion

— Léa, Edgar et Noé : fonction qui réduit le dico (100%)

— Noé : Créer la fonction qui trouve le mot à donner (100%)

— Anna : Faire le main (100%)

— Edgar : Faire la documentation des tests des algorithmes du site web (100%)

Travail à faire

— Documentation des algorithmes : Noé

— Spécification algébrique et documentation des tests : Léa

— Finir les tests (reduc_dico) : Anna

— Analyse en complexité et en mémoire : Edgar