I) CONTEXTE

Ce travail **individuel** fait partie de la note finale du second quadrimestre.
Nos objectifs sont la mise en application des concepts :

- d'analyse et de modélisation,
- fonctions et procédures,
- spécification, assertion et programmation défensive,
- documentation auto générée,
- bibliothèques et `makefile`,
- tests unitaires,
- tableaux, pointeurs, structures,
- lecture/écriture de fichiers.

II) CRITERES D'EVALUATION

Les critères d'évaluation sont les suivants :

- une rédaction soignée respectant les consignes du cours de communication,
- des algorithmes respectant les préconisations vues en méthodes de programmation :
  - corrects dans tous les cas *(généraux et spécifiques)*,
  - lisibles et simples avec des structures de contrôle bien construites,
  - efficaces *(vous éviterez tout gaspillage mémoire ou temporel)*,
- des variables dont les portées sont minimales et les identificateurs explicites *(porteur de sens)*,
- une découpe en sous-problèmes séparés par des vérifications et/ou assertions,
- des commentaires pertinents et non redondants, ainsi qu'une documentation auto générée,
- des scénarii de tests pertinents implémentés via une batterie de tests unitaires,
- les compilations ne remontent aucun avertissement *(toutes options de compilation activées, voir cours)*,
- le respect inconditionnel des échéances.

III) ECHEANCES

Deux parties sont à rendre, chacune ayant son propre dépôt sur Moodle. Un seul fichier est autorisé par dépôt, il devra respecter la convention de nommage suivante :

`<VotreNuméroDeGroupe>_<VotreNomEnMajuscules>_<VotrePrénom>.zip`
`Ex : 1_BRACAME_Edouard.zip`

Echéance intermédiaire :

avant le **`samedi 5 avril 2025 à 18h00`**,
postez votre **étude préliminaire du projet** sur moodle.

Echéance finale :

avant le **`samedi 10 mai 2025 à 18h00`**,
postez le **projet** dans sa **version finale** sur moodle.

En cas de défaillance de Moodle, votre fichier est à envoyer par mail avec votre **adresse étudiante** à : benoit.burlion@hers.be . Quel que soit le média utilisé, **les échéances doivent être respectées**.

IV) AVERTISSEMENTS

Le travail à réaliser est une production personnelle. Dans le cas où plusieurs travaux seraient fortement similaires, l'enseignant se réserve le droit de **diviser la note par le nombre d'étudiants concernés**. Dans le cas où ce travail ne semblerait pas personnel (par exemple, copie de code source trouvé/**généré** sur Internet), l'enseignant se réserve le droit **de réduire la note pour qu'elle corresponde à la production de l'étudiant**.

V) PROJET PERSONNEL

Une grande liberté vous est laissée dans ce projet, aussi bien dans les choix de conception que d'implémentation. En revanche, les attentes du client doivent impérativement être respectées. Nous pourrions synthétiser cet état d'esprit par : *« **tout ce qui n'est pas défini est à votre discrétion** »* et *« **répondez d'abord à l'ensemble des exigences du client** »*.

Nous vous conseillons donc vivement **de ne pas échanger entre vous sur ce projet** *(le pair programming est ici totalement exclu)*. Ce faisant, vous comprenez qu'il nous semble improbable de recevoir deux projets fortement similaires (voir avertissements).

VI) LIMITES DE L'ENONCÉ

Pour vous laisser une grande latitude dans vos choix, notre énoncé est volontairement très ouvert. En contrepartie, plusieurs aspects du projet n'ont pas été spécifiés. Vous devrez donc faire part **de bon sens** et de bienveillance envers vos utilisateurs afin que votre jeu soit ergonomique.

VII) Projet de second quadrimestre en langage C

L'objectif est de programmer **une variante** du jeu ***Lettre manquante***[1]. Vous trouverez ci-dessous le principe du jeu ainsi que les détails des travaux attendus : l'étude préliminaire pour l'échéance intermédiaire et la note de cadrage pour l'échéance finale.

Principe du jeu

L'objectif est de retrouver la/les lettres manquantes dans les mots.
Le jeu propose pour chaque tour : entre deux et quatre mots avec une à trois lettres à retrouver.
Les mots n'ont pas nécessairement la même longueur.
Lorsqu'une lettre est cachée, toutes ses occurrences le sont également avec le même masque.
Si plusieurs lettres sont cachées, chacune a son propre masque.
Lorsque plusieurs lettres sont à retrouver, c'est toujours dans le même ordre des masques.
La difficulté du jeu croît de manière progressive à travers trois niveaux de difficulté.
Le nombre d'essais infructueux est limité. Si le joueur trouve la bonne lettre, le nombre d'essais reste inchangé, mais sinon il est décrémenté.
Le joueur capitalise des points à chaque fois qu'il découvre une lettre.
Il est possible de passer un tour, mais dans ce cas le joueur perd des points et/ou un essai.
Le jeu se termine soit lorsque le joueur : gagne le jeu ou qu'il ne dispose plus d'essai ou qu'il décide de quitter la partie.

Etude Préliminaire

Cette étude sera composée en deux parties : un rapport d'étude et une implémentation minimale.

Le rapport d'étude

Il contiendra **à minima** et dans l'ordre :

1. une page de garde conventionnelle, un sommaire, une introduction,
2. une maquette du jeu, en expliquant quelles touches seront utilisées par le joueur,
3. les particularités de votre implémentation minimale, et notamment :
   les représentations graphiques des données manipulées,
4. les explications de vos choix *(j'ai choisi : cette variable pour …, de vérifier la fin de partie …)*,
5. la désignation des *(au moins 2)* fonctions « essentielles » qui bénéficieront de **tests unitaires**,
6. le principe de fonctionnement des tirages pseudo-aléatoires
   (qu'allez-vous tirer au sort ? Comment éviter les tirages inutiles ?),
7. le principe des trois niveaux de difficulté,
8. l'explication sur la façon dont vous mettrez en place l'allocation dynamique
   (en quoi est-elle utile ?),
9. toutes notes utiles à la compréhension *(en restant concis)*.

Le rapport d'étude sera remis au **format pdf** et devra contenir **6 à 12 pages**. La numérotation commencera à l'introduction.

[1] https://cerebral.evalquiz.com/vitesselettres

Implémentation minimale

Il s'agit de réaliser une preuve de concept *(POC – Proof Of Concept [1])*. L'objectif est d'implémenter rapidement les principes fondamentaux du jeu sur une unique partie *(avec tirage pseudo-aléatoire)*.

En termes de code source, allez à l'essentiel :

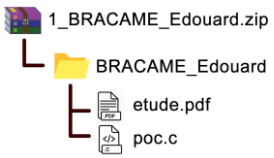
- ne demandez pas le pseudonyme du joueur,
- limitez la création de bibliothèques externes,
- n'enregistrez aucune donnée,
- faites au plus simple[2].

---

a) *Un POC c'est du code source « d'essai », « de prototypage ». L'objectif est ici de vous permettre d'expérimenter de manière empirique le passage de vos idées et de vos schémas sur feuille au code source. Précision : un prototype doit être de qualité. Même si vous faites au plus simple, rien ne vous empêche de soigner votre code source.*

Arborescence

Vous déposerez votre étude préliminaire en respectant l'exemple d'arborescence ci-contre :



Retour sur l'étude préliminaire

Un feed-back [a,b] sur l'étude préliminaire vous sera rendu. Il comprendra des observations, **d'ordre consultatif**.

Remarques :

a) *Nous vous conseillons de ne pas attendre notre retour pour commencer à raffiner (refactoring) votre prototype (POC). Vous pourriez vous intéresser à la généricité de vos algorithmes (que se passe-t-il si la taille des mots change ?) et commencer à diviser votre prototype en plusieurs bibliothèques documentées. Cela vous permettra de créer les tests unitaires relatifs aux fonctions essentielles du jeu. Dans cette optique, nous vous conseillons de dupliquer votre prototype pour le modifier dans un nouveau répertoire (ex : beta, v0.2 …). Vous conserverez ainsi la version originale telle que déposée à la première échéance.*

b) *Le cours sur les fichiers n'est pas encore abordé. Cela ne pose pas de problème dès l'instant que vous programmez dans un ordre logique, et de façon modulaire : en partant du « noyau » du jeu puis en ajoutant progressivement des fonctionnalités. Les fonctionnalités liées aux fichiers seront alors progressivement ajoutées au projet.*

---

[2] Prenez le temps de **bien choisir vos structures de données**, faites des essais sur papier pour vérifier qu'elles soient pertinentes.

Note de cadrage

Le jeu sera :

- implémenté en mode console,
- pour un terminal de taille comprise entre : 80 x 24 (standard) et 211 x 56 (maximum),
- pour 1 seul joueur (humain),
- pour chaque tour, les mots et lettres à deviner seront choisi de manière pseudo-aléatoire *(vous éviterez les tirages inutiles)*,
- codé en anglais *(code source et commentaires)*,
- avec une interface en français *(pour le joueur)*,
- économe en ressource *(vous éviterez tout gaspillage mémoire ou temporel)*,
- robuste *(tolère les erreurs de saisies, accepte aussi bien majuscule/minuscule …)*,
- composé d'une succession d'écrans *(effacement console à chaque nouvel écran)*,
- intuitif à prendre en main *(en termes d'ergonomie, un **enfant de 8 ans** doit pouvoir y jouer sans éprouver de difficulté. Gardez à l'esprit que **« l'erreur est humaine »**)*.

À l'exécution le jeu devra proposer :

- un écran de démarrage *(splash screen enregistré dans un fichier au format texte)*,
- un menu proposant :
  - l'affichage des règles du jeu,
  - d'entrer le pseudonyme du joueur,
  - de lancer une nouvelle partie,
  - de consulter l'historique des 20 dernières parties,
  - de quitter le jeu.

Historique des parties

Chaque partie terminée intègre l'historique des 20 dernières parties jouées *(triées par ordre décroissant de score)*. Le jeu sauvegardera : la date, le pseudonyme du joueur, le niveau atteint et son score.
Attention : la gestion du fichier de l'historique des parties jouées doit être réalisée en **mode binaire**.

Conventions de codage

Vos fichiers devront respecter les bonnes pratiques en termes de rédaction de code source. Vous devrez également respecter les préconisations suivantes :

- identificateurs nommés selon la convention « snake case »,
- fichiers sources inférieurs à 200 lignes,
- fonctions inférieures à 60 lignes *(spécifications et commentaires inclus)*.

Version finale et livrables (2)

Nous attendons un ensemble cohérent comprenant notamment le code source ainsi que la documentation auto générée *(par doxygen via la commande : « make doc »)*, réparti dans une arborescence logique.
Implémentations et **tests unitaires** doivent compiler via la commande « `make` ». Un fichier d'accueil du type « `readme.txt` » serait le bienvenu à la racine du projet. Aucun exécutable ne sera livré *(puisque nous compilerons votre projet)*.

Enfin, un dernier fichier nommé : « **`retour_experience.txt`** » sera placé en racine du projet. Vous y expliquerez succinctement :

- ce que ce projet vous à « appris »,
- ce que vous feriez différemment si ce projet était à refaire.