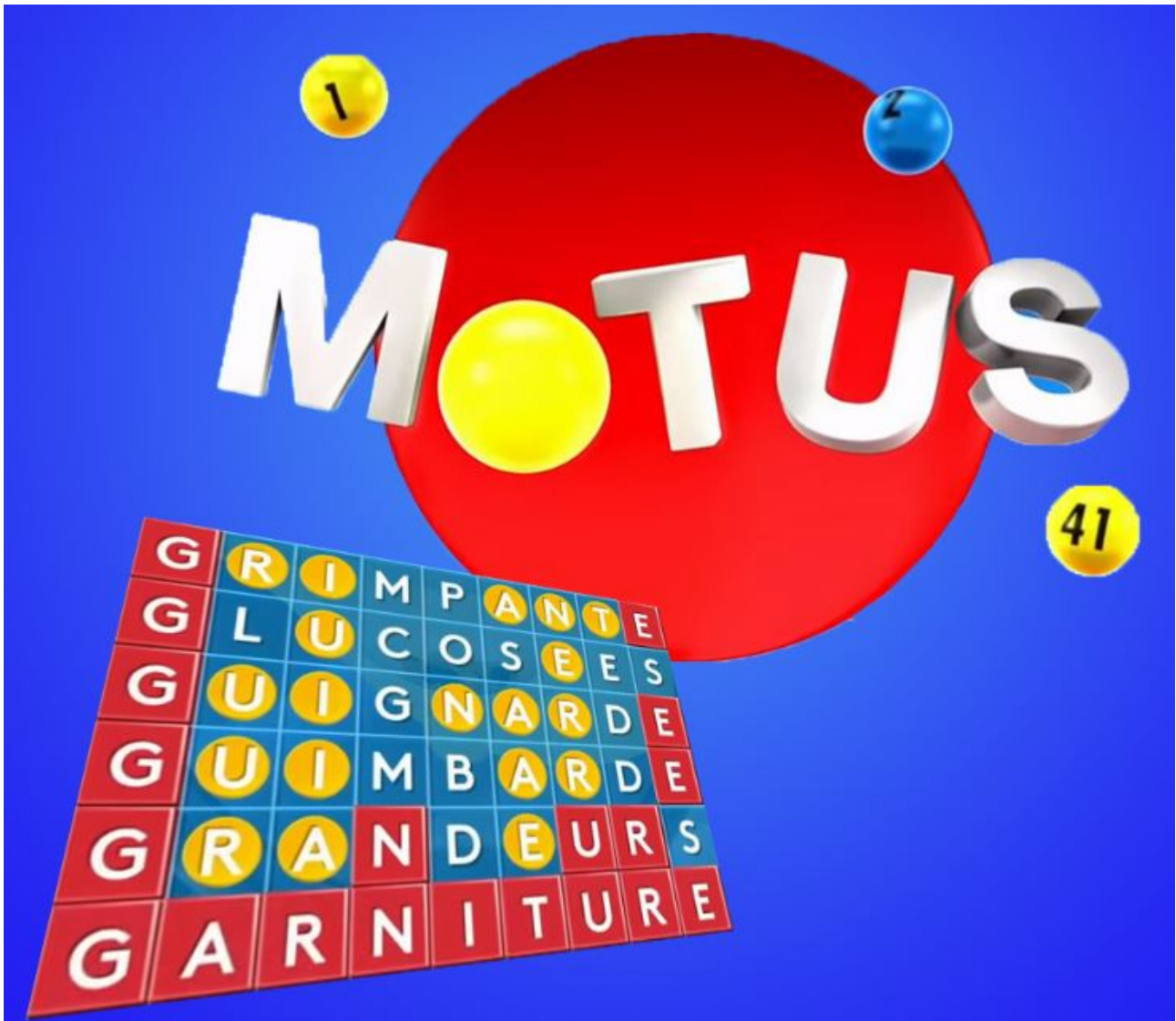


Projet ISN :

Jeu de Motus



Professeur : Mr MENENDEZ

Année : 2018/2019

Sommaire

I. Présentation de l'équipe.....	3
II. Présentation du projet.....	3
III. Analyse des besoins.....	3
IV. Recherche d'idées.....	3
V. Répartition des tâches.....	3
VI. Réalisation.....	4
A) Le langage et les règles de codage utilisées.....	4
B) Les librairies utilisées.....	4
C) Les outils de développement utilisés.....	5
1) Visual Studio.....	5
2) Qt Designer.....	6
D) L'architecture de notre application.....	7
1) Le Module « common », commun aux deux programmes.....	7
2) Programme de création du dictionnaire « subdicos_generator ».....	7
3) Notre jeu Motus.....	8
a) Le module « game_engine ».....	8
b) Le module « ui ».....	9
c) La communication entre les 2 modules.....	10
VII. Conclusion.....	11
VIII. Annexes (code source).....	12
A) Module common.....	12
1) Parameters.py.....	12
2) Utils.py.....	12
B) Motus.....	13
1) Motus.py.....	13
2) game_engine.....	13
a) Game.py.....	13
b) GamesManager.py.....	16
c) Options.py.....	17
d) Utils.py.....	18
3) Ui.....	18
a) GameWidget.py.....	18
b) GameWidget.ui.....	20
c) MainWindow.py.....	22
d) MainWindow.ui.....	23
e) MotusTableWidget.py.....	24
f) OptionsDialog.py.....	25
g) OptionsDialog.ui.....	26
C) subdicos_generator.....	28
1) subdicos_generator.py.....	28
IX. Index alphabétique.....	29
X. Index des illustrations.....	29

I. Présentation de l'équipe

Le groupe pour notre projet « Jeu de Motus » se compose de Gabriel NAVARRO et Valentin BERRON, tous les deux faisant partie de la classe Terminale SI2 durant l'année 2018-2019

II. Présentation du projet

Les règles du motus sont assez simples. Motus est un jeu de lettres où en procédant par essais et erreurs on essaye de deviner un mot dont seule la première lettre est donnée. Au début de la partie le joueur choisit le nombre de lettres qui composent le mot à découvrir. Le joueur dispose d'un temps limité pour proposer un mot, et les propositions sont elles aussi limitées pour trouver le mot choisi par l'ordinateur. Pour aider le joueur à trouver le mot inconnu, en plus de donner la première lettre du mot, à chaque proposition l'ordinateur va marquer en rouge les lettres qui sont dans la bonne position et en jaune les lettres qui se trouvent dans le mot mais qui ne sont pas dans la bonne position. Quand le joueur a trouvé le mot inconnu toutes les lettres sont en rouge et il a gagné la partie. Nous hébergerons le projet sur <https://github.com/kyvakl/Motux>.

III. Analyse des besoins

Le but de notre projet était de réaliser notre propre version du jeu de Motus, avec une interface simple mais intuitive, de cette manière le joueur pourrait faire une partie quand bon lui semble et ainsi s'amuser le temps de quelques parties.

Il nous a paru important de faire en sorte que le jeu reste cependant proche de celui vu à la télé.

IV. Recherche d'idées

Dans un premier temps nous avons réfléchi à réaliser comme projet d'ISN un jeu de pendu mais très vite nous nous sommes demandés si c'était approprié pour un projet scolaire donc nous avons abandonné cette idée. Mais assez rapidement nous avons trouvé notre sujet que nous avons décidé de réaliser cette année. C'est une version personnelle du célèbre jeu télévisé motus. Notre version du motus se nomme MOTUX. Nous avons choisi de réaliser une variante de ce jeu car c'est un jeu que nous apprécions tous les deux et que nous avons regardé de nombreuses heures à essayé de trouver le mot inconnu.

V. Répartition des tâches

Pour séparer de manière égale les tâches menant à la complétion de notre projet, nous avons décidé de séparer la programmation du jeu en 2 parties :

- Le code rendant le jeu fonctionnel, fait par Valentin BERRON
- L'Interface Utilisateur (ou IHM Interface Homme Machine), fait par Gabriel NAVARRO

De cette manière, nos tâches ne se superposent pas et nous ne pouvons pas nous empêcher l'un l'autre de continuer sa propre tâche, car ces 2 parties sont indépendantes du côté programmation.

Cependant les 2 parties doivent être chacune associées l'une à l'autre pour permettre le fonctionnement du programme.

VI. Réalisation

Pour la réalisation de ce projet en Python, nous nous sommes tout d'abord fixés des règles.

Nous avons ensuite cherché une librairie intéressante pour pouvoir notamment afficher notre jeu.

Nous avons ensuite choisi des outils de développement pour nous faciliter la tâche, tant au niveau du code Python que des interfaces utilisateur.

Nous avons ensuite défini l'architecture de notre application, avant de l'implémenter.

A) Le langage et les règles de codage utilisées

Nous avons décidé d'utiliser le langage Python, l'un des plus utilisés, pour sa simplicité d'écriture.

Les mots clé du langage Python étant en anglais, nous avons décidé que les noms de variables, de fonctions ou de classes seraient des mots anglais. Cela permet de mieux lire le programme, en créant des phrases cohérentes.

- Les commentaires sont en Français.
- Les noms de classes commencent par une majuscule.
- Les noms des variables et des fonctions commencent par une minuscule.
- Les noms des variables membres d'une classe sont préfixées par « m_ » ; par exemple, m_game.
- Enfin, comme nous l'expliquerons plus en détail plus loin dans ce document, nous avons décidé de séparer les traitements des interfaces utilisateur.

B) Les librairies utilisées

Pour la partie interface utilisateur (ou IHM, pour Interface Homme Machine), nous avons décidé d'utiliser Qt¹ pour Python.

Outre le développement de l'IHM, Qt est une librairie qui contient de nombreuses fonctionnalités, comme par exemple jouer des sons ou compter le temps écoulé.

Nous avons trouvé les fichiers son sur <https://codes-sources.commentcamarche.net/source/53651-motus-en-python-2-6-avec-tksnack>, que nous avons consulté.

1 Qt pour Python: https://wiki.qt.io/Qt_for_Python

C) Les outils de développement utilisés

Bien qu'il soit possible de développer un programme Python avec un simple éditeur de texte, tel que notepad++, des outils plus puissants permettent de faciliter ce développement, pour le code Python et pour la création de l'IHM.

1) Visual Studio

Pour développer le code python de notre application de façon modulaire, nous avons utilisé Visual Studio Community², qui est distribué gratuitement par Microsoft à des fins non commerciales, et qui intègre le langage Python.

En plus de faciliter l'intégration et le développement de l'application, cet outil nous a permis de la déboguer plus facilement, en permettant de l'exécuter en pas à pas.

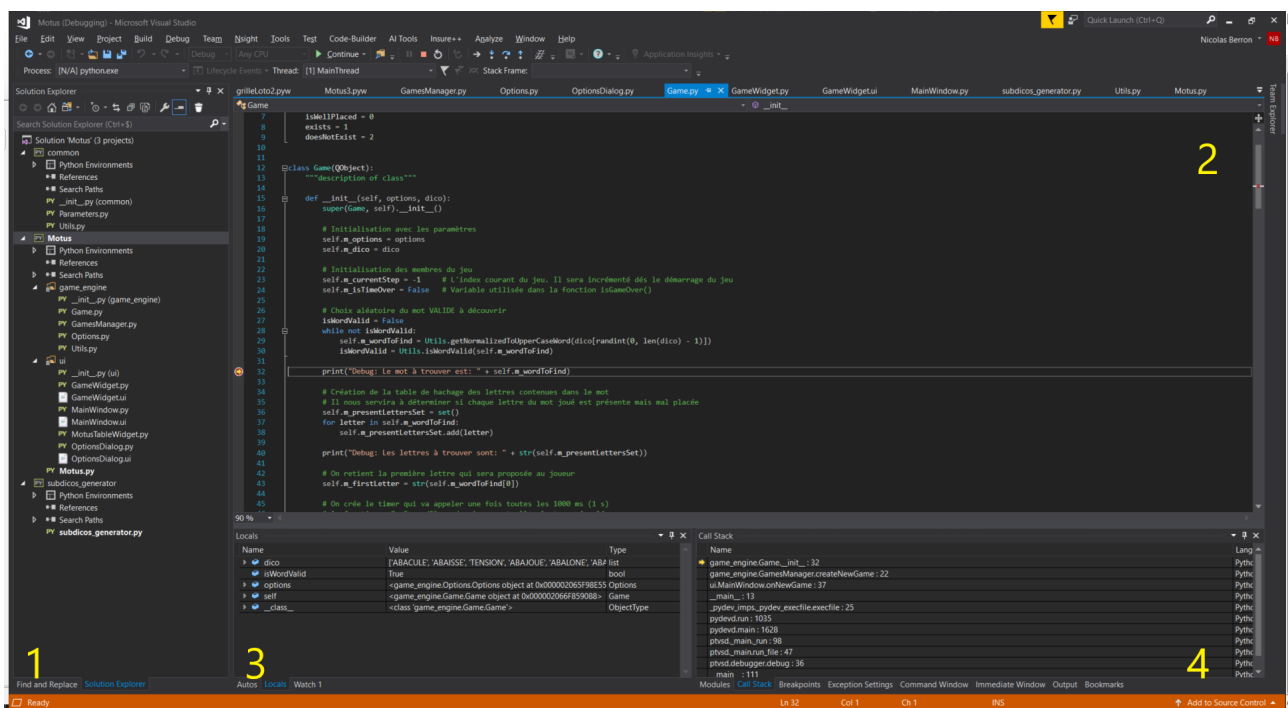


Illustration 1: Visual Studio Community

Cette capture d'écran montre les différentes fonctionnalités :

1. L'explorateur de solution : Il contient l'arborescence des programmes et des modules de notre application.
2. L'éditeur de programmes. Ici, un point d'arrêt a été atteint.
3. Les valeurs des variables courantes sont affichées, ce qui permet d'analyser le bon déroulement du programme.
4. La pile d'appel des fonctions

² Visual Studio Community : <https://visualstudio.microsoft.com/fr/vs/community/>

2) Qt Designer

Qt designer est un outil livré avec Qt, qui permet de développer facilement grâce à la souris les interfaces utilisateur.

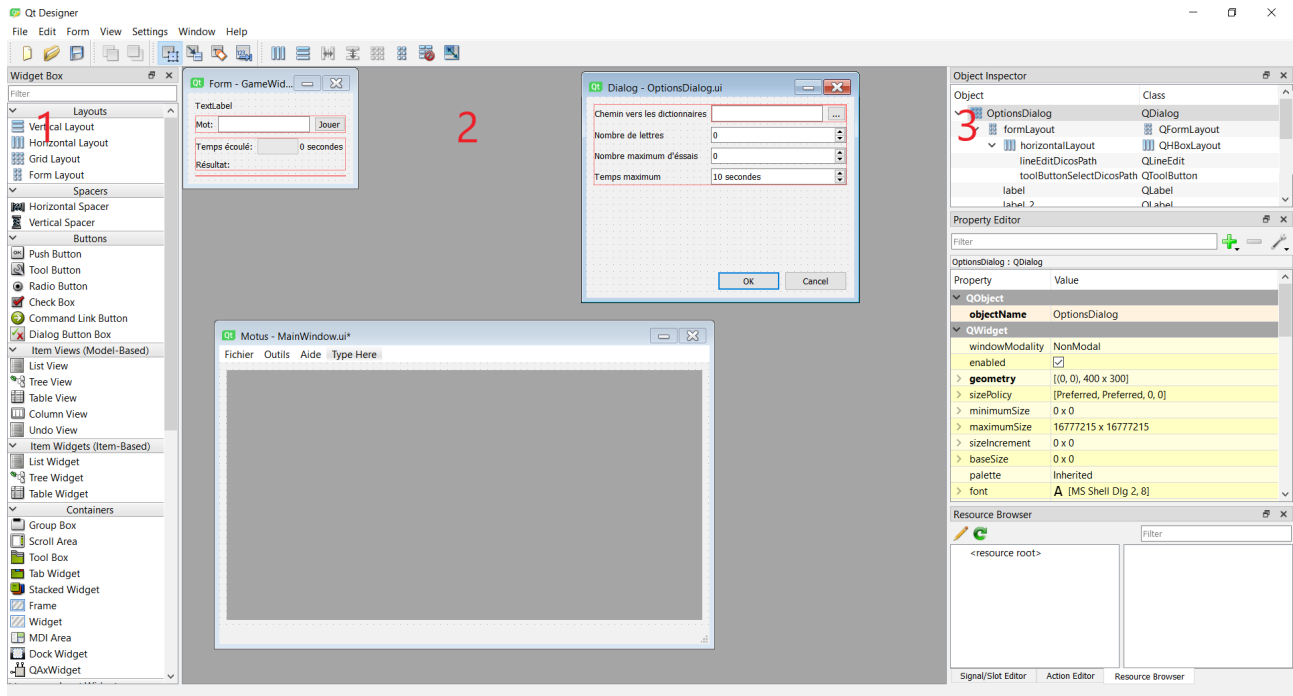


Illustration 2: Qt Designer

Les différentes régions de Qt Designer sont :

1. Ce panneau contient tous les Widgets (Windows Gadgets), qui sont les boutons, les zones de saisie, etc.
2. La zone de travail, où nous pouvons créer des fenêtres, des boîtes de dialogues, etc. Ici, nous avons la fenêtre principale, la fenêtre du jeu proprement dite, et la boîte de dialogue pour modifier les options du jeu.
3. Ce panneau permet de nommer et modifier les propriétés de chaque Widget.

D) L'architecture de notre application

Notre application se compose de deux programmes : « subdicos_generator », qui va nous permettre de créer notre dictionnaire de mots, et notre jeu Motus.

Ces deux programmes utilisent un module commun, nommé « common ».

1) Le Module « common », commun aux deux programmes

Ce module contient les deux fichiers suivant :

1. **Parameters.py**, qui contient des paramètres communs aux 2 programmes.
2. **Utils.py**, qui contient des utilitaires communs aux 2 programmes.

2) Programme de création du dictionnaire « subdicos_generator »

Nous avons trouvé un dictionnaire à l'adresse <http://infolingu.univ-mlv.fr/DonneesLinguistiques/Dictionnaires/telechargement.html>.

C'est un fichier XML, que notre programme, en ligne de commande, va lire pour en extraire les mots, les transformer en majuscules et sans accents, et ensuite écrire un fichier pour chaque nombre de lettres :
























Name	Date modified	Type	Size
 3.dico	4/26/2019 5:16 PM	DICO File	7 KB
 4.dico	4/26/2019 5:16 PM	DICO File	26 KB
 5.dico	4/26/2019 5:16 PM	DICO File	68 KB
 6.dico	4/26/2019 5:16 PM	DICO File	124 KB
 7.dico	4/26/2019 5:16 PM	DICO File	189 KB
 8.dico	4/26/2019 5:16 PM	DICO File	243 KB
 9.dico	4/26/2019 5:16 PM	DICO File	268 KB
 10.dico	4/26/2019 5:16 PM	DICO File	261 KB
 11.dico	4/26/2019 5:16 PM	DICO File	226 KB
 12.dico	4/26/2019 5:16 PM	DICO File	185 KB
 13.dico	4/26/2019 5:16 PM	DICO File	141 KB
 14.dico	4/26/2019 5:16 PM	DICO File	102 KB
 15.dico	4/26/2019 5:16 PM	DICO File	72 KB
 16.dico	4/26/2019 5:16 PM	DICO File	45 KB
 17.dico	4/26/2019 5:16 PM	DICO File	29 KB
 18.dico	4/26/2019 5:16 PM	DICO File	18 KB
 19.dico	4/26/2019 5:16 PM	DICO File	12 KB
 20.dico	4/26/2019 5:16 PM	DICO File	6 KB
 21.dico	4/26/2019 5:16 PM	DICO File	4 KB
 22.dico	4/26/2019 5:16 PM	DICO File	2 KB
 23.dico	4/26/2019 5:16 PM	DICO File	1 KB
 24.dico	4/26/2019 5:16 PM	DICO File	1 KB
 25.dico	4/26/2019 5:16 PM	DICO File	1 KB

Illustration 3: Liste des sous dictionnaires générés

Le fichier « 3.dico » contient tous les mots de 3 lettres, et ainsi de suite jusqu'à « 25.dico », qui en contient 25.

Il est ainsi facile, selon l'option de nombre de lettres, de charger le dictionnaire correspondant.

3) Notre jeu Motus

Nous avons pris soin de séparer les traitements (c'est à dire les programmes concernant le moteur du jeu proprement dit) des interfaces utilisateur. Ainsi, il est possible d'afficher le jeu de façons différentes, sans modifier le fonctionnement du jeu. Nous avons donc 2 modules : le module « game_engine » et le module « ui »

a) Le module « game_engine »

C'est le module central du jeu, qui contient « l'intelligence » de ce dernier :

1. **Utils.py**, qui contient des utilitaires permettant de récupérer des informations à propos du dictionnaire (son chemin d'accès, si il est valide) ou encore son contenu.
2. **Options.py**, qui contient les options du jeu : le chemin vers les dictionnaires, le nombre de lettres du mot, la durée maximale du jeu, le nombre maximum d'essais, la couleur des lettres bien placées, des lettres existantes, et des lettres inexistantes.
3. **GameManager.py**, qui permet de créer un nouveau jeu et de centraliser les options avec ce dernier.
4. **Game.py**, qui est le jeu lui même. C'est dans ce fichier que l'algorithme du jeu a été implémenté.

Algorithme en pseudo code :

```

MotATrouver := PrendreUnMotAuHasardDansDico
LancerChronomètre
ProposerLaPremièreLettre
AGagné:= Faux
NombreDEssaisRestants := 10
TantQue: Le temps n'est pas écoulé ET AGagné = Faux ET NombreDEssaisRestants > 0
    Initialiser TableauDeLettresResultat à bleu
    Quand: MotEntré

        NombreDEssaisRestants:= NombreDEssaisRestants - 1

        Si MotEntré = MotATrouver
            Agagné := Vrai
        Sinon:
            Pour Index < Longueur MotATrouver
                Si MotEntré[Index] = MotATrouver[Index]:
                    TableauDeLettresResultat[Index]:= Rouge
                SinonSi La lettre existe dans le mot cherché:
                    TableauDeLettresResultat[Index]:= Jaune
                Sinon:
                    TableauDeLettresResultat[Index]:= Bleu
            FinSi
        FinPour

        Afficher(MotEntré, TableauDeLettresResultat)
    FinSi
    Index := Index + 1
FinQuand
FinTantQue

Si Agagné :
    Afficher(« Gagné »)
Sinon :
    Afficher(« Perdu »)
FinSi

```


b) Le module « ui »

Il contient toutes les IHM de notre jeu :

Les fichiers « *.ui » sont les fichiers XML générés par Qt Designer, qui décrivent l'interface utilisateur. Les fichiers « *.py » sont les implémentations Python de chaque widget

1. **MainWindow.ui et MainWindow.py**: la fenêtre principale, avec les menus Fichier (qui contient les sous-menus Nouveau jeu et Quitter » et Outils (qui contient le sous-menu Options) :

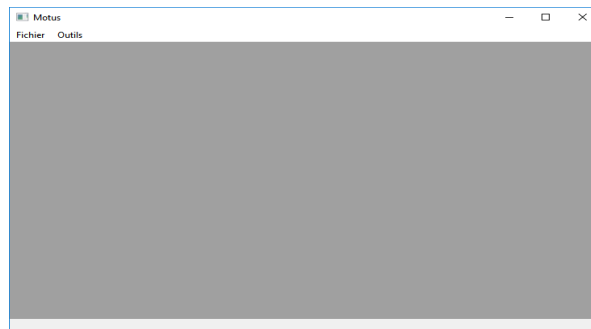


Illustration 4: La fenêtre principale du jeu

2. **OptionsDialog.ui et OptionsDialog.py** : la boîte de dialogue permettant de modifier les options :

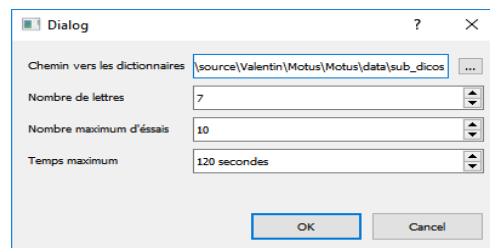


Illustration 5: La boîte d'options

3. **MotusTableWidget.py** : C'est une spécialisation de QTableWidgetItem, qui sert à éditer une table de type excel (par exemple). Ici, Une table pour un Motus de 7 lettres et 10 coups :

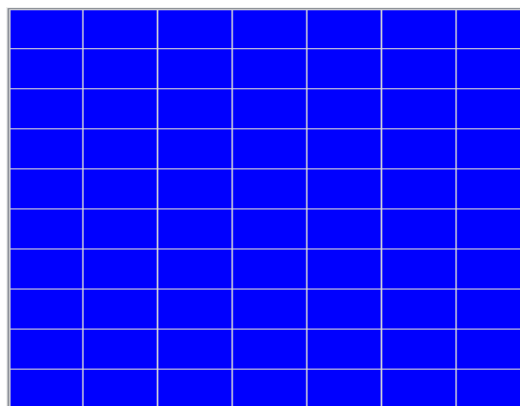


Illustration 6: Table 7 lettres 10 essais

4. **GameWidget.ui et GameWidget.py** : C'est la fenêtre du jeu proprement dit. Elle permet de saisir les mots (la zone de texte est rouge tant que le mot est incomplet, et devient blanche quand il le devient), de jouer ce mot, de voir le temps écoulé ainsi que le résultat, et enfin d'afficher la grille.

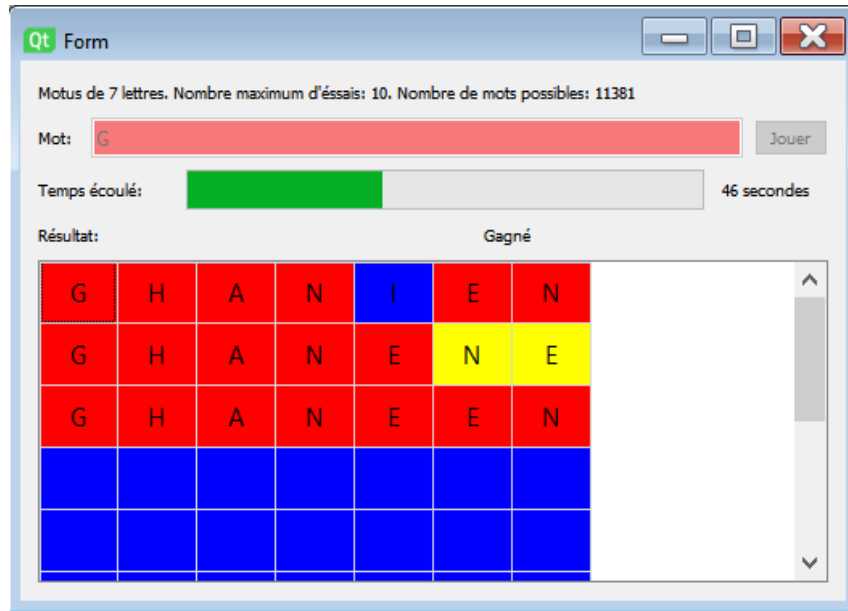


Illustration 7: Le widget de jeu

c) **La communication entre les 2 modules**

Comme nous l'avons dit, le jeu et son affichage sont séparés. Le jeu ne connaît pas l'affichage (il se consacre aux seuls traitements). Par contre, l'affichage connaît le jeu ; puisqu'il est fait pour lui.

Qt permet de communiquer à travers des « **signals** » et des « **slots** ».

Les slots sont des fonctions, qui, lorsqu'elles sont connectées à un signal, sont appelées lorsque ce signal est émis.

Ainsi, **Game** contient les signaux suivants :

- **proposeFirstLetterSignal**, qui est émis en début de jeu et à chaque itération, envoyant la première lettre.
- **playedSignal**, qui est émis quand un mot a été joué et qu'il a été analysé, envoyant l'index du mot essayé, le mot lui-même, ainsi qu'une liste qui décrit, pour chaque lettre, leur validité (existante, inexistante ou bien placée)
- **progressSignal**, qui est émis chaque seconde, envoyant le temps écoulé.
- **gameOverSignal**, qui est émis en fin de partie, envoyant vrai si le joueur a gagné, faux si il a perdu (nombre d'essais atteint ou temps écoulé)

De son côté, **GameWidget** connecte les signaux de la façon suivante :

- **proposeFirstLetterSignal** est connecté à **onFirstLetterProposed**, qui se charge de mettre la première lettre dans la zone de saisie du mot.
- **playedSignal** est connecté à **onPlayed**, qui se charge de remplir la grille avec le mot joué et d'assigner les bonnes couleurs.
- **progressSignal** est connecté à **onDurationChanged**, qui met à jour la barre de progrès du temps écoulé.
- **gameOverSignal** est connecté à **onGameOver**, qui affiche « Gagné » ou « Perdu », et de jouer le son correspondant.

Quand le bouton « Jouer » est cliqué, la fonction **play()** du **Game** est appelée.

VII. Conclusion

L'architecture que nous avons utilisé, séparant les traitements de l'IHM, nous a également permis de travailler en parallèle, en se consacrant chacun à notre développement.

Il a ensuite été facile d'intégrer notre travail. C'est un autre avantage de séparer les traitements de IHM.

Voici un aperçu du programme terminé, avec un jeu gagné et un jeu en perdu (qui n'a en fait pas été joué, et tout le temps s'est écoulé):

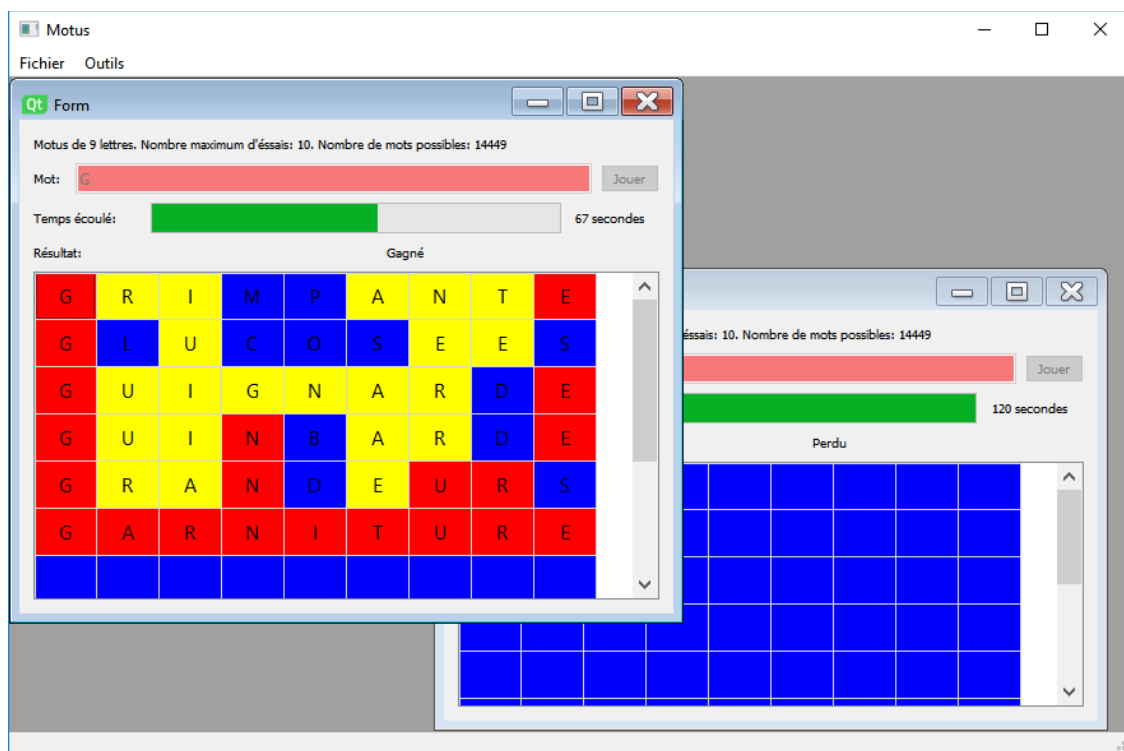


Illustration 8: Le jeu terminé

VIII. Annexes (code source)

A) Module common

1) Parameters.py

```
class Parameters(object):
    """Cette classe contient des paramètres généraux"""

    # Le nombre minimal de mots dans un dictionnaire
    @staticmethod
    def minimumWordsInDico():
        return 3
```

2) Utils.py

```
import unicodedata

class Utils(object):
    """Cette classe contient des utilitaires généraux"""

    # Renvoie un mot en majuscules et sans accent
    @staticmethod
    def getNormalizedToUpperCaseWord(word):
        bytes = str(unicodedata.normalize('NFKD', word.upper()).encode('ASCII', 'ignore'))
        w = ''
        for i in range(2, len(bytes) - 1):
            w = w + bytes[i]
        return w;

    # Renvoie si un mot est valide ou pas
    @staticmethod
    def isWordValid(word):
        # On enlève les mots du type "L'OPERA"
        for letter in word:
            if letter < 'A' or letter > 'Z':
                return False

        return True
```

B) Motus

1) Motus.py

```
import sys
from PySide2.QtWidgets import QApplication

import ui.MainWindow as MainWindow
import game_engine.GameManager as GameManager

if __name__ == "__main__":
    # Création de l'application Qt
    app = QApplication(sys.argv)

    # Création de la fenêtre principale
    mainWindow = MainWindow.MainWindow(GameManager.GameManager())

    # Affichage de la fenêtre à l'écran
    mainWindow.show()

    # Exécution de l'application
    sys.exit(app.exec_())
```

2) game_engine

a) Game.py

```
from game_engine.Utils import Utils
from random import *
from PySide2.QtCore import QObject, Signal, Slot, QTimer
from enum import Enum

# Enumeration des status des lettres
class LetterPlacement(Enum):
    isWellPlaced = 0
    exists = 1
    doesNotExist = 2

class Game(QObject):
    """Cette classe contient l'algorithme de Motus"""

    def __init__(self, options, dico):
        super(Game, self).__init__()

        # Initialisation avec les paramètres
        self.m_options = options
        self.m_dico = dico

        # Initialisation des membres du jeu
        self.m_currentStep = -1 # L'index courant du jeu. Il sera incrémenté dès le démarrage du jeu
        self.m_isTimeOver = False # Variable utilisée dans la fonction isGameOver()

        # Choix aléatoire du mot VALIDE à découvrir
        isWordValid = False
        while not isWordValid:
            self.m_wordToFind = Utils.getNormalizedToUpperCaseWord(dico[randint(0, len(dico) - 1)])
            isWordValid = Utils.isWordValid(self.m_wordToFind)

        print("Debug: Le mot à trouver est: " + self.m_wordToFind)

        # Création de la table de hachage des lettres contenues dans le mot
        # Il nous servira à déterminer si chaque lettre du mot joué est présente mais mal placée
        self.m_presentLettersSet = set()
        for letter in self.m_wordToFind:
            self.m_presentLettersSet.add(letter)

        print("Debug: Les lettres à trouver sont: " + str(self.m_presentLettersSet))
```

```

# On retient la première lettre qui sera proposée au joueur
self.m_firstLetter = str(self.m_wordToFind[0])

# On crée le timer qui va appeler une fois toutes les 1000 ms (1 s)
# la fonction onOneSecondElapsed qui va contrôler le temps écoulé
self.m_timer = QTimer()
self.m_elapsedTime = 0
self.m_timer.timeout.connect(self.onOneSecondElapsed)
self.m_timer.start(1000)

# --- Signaux

# Ce signal sera émis au moment de donner la main au joueur
# Son paramètre est une chaîne de caractère contenant la première lettre
proposeFirstLetterSignal = Signal(str)

# Ce signal sera émis juste après l'évaluation du mot joué
# Paramètre 1: L'index du coup joué
# Paramètre 2: Le mot joué
# Paramètre 3: La liste du type de placement de chaque lettre du mot (LetterPlacement)
playedSignal = Signal(int, str, list)

# Ce signal est émis chaque seconde. Son paramètre est le nombre de secondes écoulées
progressSignal = Signal(int)

# Ce signal est émis à la fin du jeu, soit parce que le temps est écoulé,
# soit parce que le nombre maximum de coups a été atteint, soit parce que le mot a été trouvé
# Son paramètre est True si le mot est trouvé, False sinon
gameOverSignal = Signal(bool)

# --- Slots

# Cette méthode est appelée toutes les secondes
@Slot()
def onOneSecondElapsed(self):
    # On incrémente le nombre de secondes écoulées
    self.m_elapsedTime = self.m_elapsedTime + 1

    # On informe les objets connectés à ce signal
    # (par exemple une QProgressBar) de l'avancement
    self.progressSignal.emit(self.m_elapsedTime)

    # Si on a dépassé le temps imparti:
    if self.m_elapsedTime >= self.m_options.duration:
        # On spécifie que le temps est terminé, et donc que le jeu est terminé pour cette raison
        self.m_isTimeOver = True

        # On arrête le timer
        self.m_timer.stop()

        # On informe les objets connectés à ce signal que le jeu est fini non gagnant
        self.gameOverSignal.emit(False)

# --- Propriétés

# Permet d'accéder aux options depuis l'extérieur
@property
def options(self):
    return self.m_options

# Permet d'accéder au dictionnaire depuis l'extérieur
@property
def dico(self):
    return self.m_dico

```

```

# --- Méthodes
# Cette méthode permet de démarrer le jeu
def start(self):
    self.playNextStep()

# Cette méthode joue le coup suivant
def playNextStep(self):
    # On propose la première lettre aux objets connectés au signal proposeFirstLetterSignal
    self.proposeFirstLetterSignal.emit(self.m_firstLetter)

    # On incrémente l'index du jeu
    self.m_currentStep = self.m_currentStep + 1

# Cette méthode est le point d'entrée pour jouer un mot
def play(self, word):
    # On informe les objets connectés à ce signal de l'index du jeu, du mot joué
    # et du résultat de l'analyse, lettre par lettre
    self.playedSignal.emit(self.m_currentStep, word, self.getResult(word))

    # Si le jeu n'est pas terminé, on joue le coup suivant
    if not self.isGameOver():
        self.playNextStep()
    else:
        self.gameOverSignal.emit(False)

# --- Functions
# Renvoie si un mot est valide ou pas
def isValidWord(self, word):
    if len(word) < self.m_options.lettersCount:
        return False

    for letter in word:
        if letter < 'A' or letter > 'Z':
            return False

    return True

# Le jeu est terminé si le temps est écoulé, ou qu'on atteint le nombre maximum d'essais
def isGameOver(self):
    return self.m_isTimeOver or self.m_currentStep >= self.m_options.m_maximumTriesCount - 1

# Cette fonction analyse le mot joué pour renvoyer, lettre par lettre cette analyse
def getResult(self, word):
    allGood = True
    l = [] # Liste des analyses

    # On parcourt le mot avec son index
    for i in range(len(word)):

        # Si les lettres des 2 mots sont les mêmes
        if word[i] == self.m_wordToFind[i]:
            l.append(LetterPlacement.isWellPlaced)
        else:
            allGood = False # Toutes les lettres ne sont pas bien placées

        # Si la lettre se trouve dans la table de hachage qu'on a créée dans le constructeur
        if word[i] in self.m_presentLettersSet:
            # La lettre existe, mais n'est pas bien placée
            l.append(LetterPlacement.exists)
        else:
            # Sinon, elle n'existe pas
            l.append(LetterPlacement.doesNotExist)

    # Si toutes les lettres sont bien placées, le jeu est terminé
    if allGood:
        # On arrête le timer
        self.m_timer.stop()
        # Et on informe que le jeu est terminé gagnant
        self.gameOverSignal.emit(True)

    # On retourne enfin la liste des types de placement de lettres
    return l

```

b) GamesManager.py

```
import game_engine.Options as Options
import game_engine.Game as Game

class GamesManager(object):
    """Cette classe est destinée à créer un jeu"""

    def __init__(self):
        self.m_minimumWordsInDico = 3
        self.m_options = Options.Options()

    # --- Propriétés

    # Options
    @property
    def options(self):
        return self.m_options

    @options.setter
    def options(self, value):
        self.m_options = value

    # Méthodes

    # Création d'un jeu
    def createNewGame(self, dico):
        return Game.Game(self.m_options, dico)
```


c) Options.py

```
import os
class Options(object):
    """Cette classe contient toutes les options du jeu"""
    def __init__(self):
        self.m_pathToDicos = os.getcwd() + '\\data\\sub_dicos'
        self.m_lettersCount = 7
        self.m_maximumTriesCount = 10
        self.m_duration = 120 # secondes
        self.m_letterWellPlacedColor = 255, 0, 0 # Rouge
        self.m_letterExistsColor = 255, 255, 0 # Jaune
        self.m_letterDoesNotExistColor = 0, 0, 255 # Bleu
    # Propriétés
    # Chemin vers les dictionnaires
    @property
    def pathToDicos(self):
        return self.m_pathToDicos

    @pathToDicos.setter
    def pathToDicos(self, value):
        self.m_pathToDicos = value

    # Nombre de lettres du mot
    @property
    def lettersCount(self):
        return self.m_lettersCount

    @lettersCount.setter
    def lettersCount(self, value):
        self.m_lettersCount = value

    # Durée maximale du jeu
    @property
    def duration(self):
        return self.m_duration

    @duration.setter
    def duration(self, value):
        self.m_duration = value

    # Nombre maximum d'essais
    @property
    def maximumTriesCount(self):
        return self.m_maximumTriesCount

    @maximumTriesCount.setter
    def maximumTriesCount(self, value):
        self.m_maximumTriesCount = value

    # Couleur des lettres bien placées
    @property
    def letterPlacementWellPlacedColor(self):
        return self.m_letterWellPlacedColor

    @letterPlacementWellPlacedColor.setter
    def letterPlacementWellPlacedColor(self, value):
        self.m_letterWellPlacedColor = value

    # Couleur des lettres existantes
    @property
    def letterExistsColor(self):
        return self.m_letterExistsColor

    @letterExistsColor.setter
    def letterExistsColor(self, value):
        self.m_letterExistsColor = value

    # Couleur des lettres inexistantes
    @property
    def letterDoesNotExistColor(self):
        return self.m_letterDoesNotExistColor

    @letterDoesNotExistColor.setter
    def letterDoesNotExistColor(self, value):
        self.m_letterDoesNotExistColor = value
```

d) *Utils.py*

```
import pickle
from common.Parameters import Parameters

from common.Utils import Utils as BaseUtils

class Utils(BaseUtils):
    """Cette classe contient des utilitaires spécifiques au jeu"""

    # Renvoie le chemin complet vers les dictionnaires
    @staticmethod
    def getDicoFullPath(path, lettersCount):
        return path + "\\\" + str(lettersCount) + ".dico"

    # Lit et renvoie le contenu d'un dictionnaire
    @staticmethod
    def getDicoContents(path, lettersCount):
        dicoPath = Utils.getDicoFullPath(path, lettersCount)
        try:
            with open(dicoPath, 'rb') as pickle_file:
                return True, pickle.load(pickle_file), ''
        except Exception as error:
            message = "Le chargement du dictionnaire '" + dicoPath + "' a échoué, car \n" + str(error.args)
            print('Erreur: ' + message)
            return False, [], message

    # Renvoie si le dictionnaire est valide ou pas
    @staticmethod
    def isDicoValid(path, lettersCount):
        isValid, dicoContents, errorMessage = Utils.getDicoContents(path, lettersCount)

        if not isValid:
            return False, errorMessage

        if len(dicoContents) < Parameters.minimumWordsInDico():
            message = "Le dictionnaire '" + Utils.getDicoFullPath(path, lettersCount) + "' ne contient que "
                + str(len(dicoContents)) + " mots; le minimum est " + str(Parameters.minimumWordsInDico())
            print('Erreur: ' + message)
            return False, message

        return True, ''
```

3) Ui

a) *GameWidget.py*

```
from PyQt5 import uic, QtWidgets, QtGui
from PySide2.QtCore import Slot, Qt
from PySide2.QtMultimedia import QSound
import ui.MotusTableWidget as MotusTableWidget
from game_engine.Game import LetterPlacement, Utils

class GameWidget(QtWidgets.QWidget):
    """Cette classe implémente l'IHM du jeu"""

    def __init__(self, game):
        super(GameWidget, self).__init__()
        uic.loadUi('UI/GameWidget.ui', self)

        self.m_game = game
        self.labelInfo.setText('Motus de ' + str(self.m_game.options.lettersCount)
            + " lettres. Nombre maximum d'essais: "
            + str(self.m_game.options.maximumTriesCount)
            + ". Nombre de mots possibles: " + str(len(game.dico) - 1))

        self.lineEditWord.setMaxLength(self.m_game.options.lettersCount)

        self.m_motusTableWidget = MotusTableWidget.MotusTableWidget(game)
        self.progressBar.setMaximum(game.options.duration)
```

```

# Sons
self.beepSound = QSound("data/sounds/beep-3.wav")
self.newGameSound = QSound("data/sounds/nouveau_mot.wav")
self.winSound = QSound("data/sounds/kultur0407.wav")
self.lostSound = QSound("data/sounds/incorrect.wav")

self.verticalLayoutGameComponents.addWidget(self.m_motusTableWidget)

# Connections
self.toolButtonPlayWord.clicked.connect(self.onPlay)
self.lineEditWord.textChanged.connect(self.onLineEditTextChanged)

self.m_game.proposeFirstLetterSignal.connect(self.onFirstLetterProposed)
self.m_game.playedSignal.connect(self.onPlayed)
self.m_game.progressSignal.connect(self.onDurationChanged)
self.m_game.gameOverSignal.connect(self.onGameOver)

# --- Slots

# Appel   quand le contenu du mot jou   est modifi  
@Slot(str)
def onLineEditTextChanged(self, word):
    self.lineEditWord.textChanged.disconnect(self.onLineEditTextChanged)
    w = Utils.getNormalizedToUpperCaseWord(word)
    if self.m_game.isWordValid(word):
        self.lineEditWord.setStyleSheet("QLineEdit { background-color: rgb(255, 255, 255); } ")
    else:
        self.lineEditWord.setStyleSheet("QLineEdit { background-color: rgb(255, 0, 0, 127); } ")

    self.lineEditWord.setText(w)
    self.lineEditWord.textChanged.connect(self.onLineEditTextChanged)

# Appel   quand un on clique sur le bouton jouer
@Slot()
def onPlay(self):
    self.m_game.play(self.lineEditWord.text())

# Appel   quand la premi  re lettre du mot est propos  e par le jeu
@Slot()
def onFirstLetterProposed(self, firstLetter):
    self.lineEditWord.setText(firstLetter)
    self.newGameSound.play()

# Appel   quand un essai a   t   valid   par le jeu
@Slot(int, str)
def onPlayed(self, row, word, result):
    for column in range(len(word)):
        self.m_motusTableWidget.setLetter(row, column, str(word[column]), result[column])

# Appel   quand le jeu signale une seconde   coul  e
@Slot(int)
def onDurationChanged(self, duration):
    self.progressBar.setValue(duration)
    self.beepSound.play()

# Appel   quand le jeu est termin  
@Slot(bool)
def onGameOver(self, isWinner):
    self.lineEditWord.setEnabled(False)
    self.toolButtonPlayWord.setEnabled(False)

    if isWinner:
        self.labelResult.setText("Gagn  ")
        self.winSound.play()
    else:
        self.labelResult.setText("Perdu")
        self.lostSound.play()

```

b) GameWidget.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>GameWidget</class>
<widget class="QWidget" name="GameWidget">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>247</width>
      <height>136</height>
    </rect>
  </property>
  <property name="windowTitle">
    <string>Form</string>
  </property>
  <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
      <widget class="QLabel" name="labelInfo">
        <property name="sizePolicy">
          <sizepolicy hsize="Preferred" vsize="Fixed">
            <horstretch>0</horstretch>
            <verstretch>0</verstretch>
          </sizepolicy>
        </property>
        <property name="text">
          <string>TextLabel</string>
        </property>
      </widget>
    </item>
    <item row="1" column="0">
      <layout class="QHBoxLayout" name="horizontalLayout">
        <item>
          <widget class="QLabel" name="label">
            <property name="text">
              <string>Mot:</string>
            </property>
          </widget>
        </item>
        <item>
          <widget class="QLineEdit" name="lineEditWord"/>
        </item>
        <item>
          <widget class="QToolButton" name="toolButtonPlayWord">
            <property name="text">
              <string>Jouer</string>
            </property>
          </widget>
        </item>
      </layout>
    </item>
    <item row="2" column="0">
      <layout class="QFormLayout" name="formLayout">
        <item row="0" column="0">
          <widget class="QLabel" name="label_2">
            <property name="text">
              <string>Temps écoulé:</string>
            </property>
          </widget>
        </item>
        <item row="0" column="1">
          <widget class="QProgressBar" name="progressBar">
            <property name="value">
              <number>0</number>
            </property>
            <property name="format">
              <string>%v secondes</string>
            </property>
          </widget>
        </item>
      </layout>
    </item>
  </layout>
</widget>
</ui>

```

```
<item row="1" column="0">
  <widget class="QLabel" name="label_3">
    <property name="text">
      <string>Résultat:</string>
    </property>
  </widget>
</item>
<item row="1" column="1">
  <widget class="QLabel" name="labelResult">
    <property name="text">
      <string/>
    </property>
    <property name="alignment">
      <set>Qt::AlignCenter</set>
    </property>
  </widget>
</item>
</layout>
</item>
<item row="3" column="0">
  <layout class="QVBoxLayout" name="verticalLayoutGameComponents"/>
</item>
</layout>
</widget>
<resources/>
<connections/>
</ui>
```

c) MainWindow.py

```

from PyQt5 import uic, QtWidgets
import sys
import ui.OptionsDialog as OptionsDialog
import ui.GameWidget as GameWidget
from PySide2.QtCore import Slot
from game_engine.Utils import Utils
from PyQt5 import QtWidgets

class MainWindow(QtWidgets.QMainWindow):
    """Cette classe implémente la fenêtre principale de l'application"""

    def __init__(self, gamesManager):
        super(MainWindow, self).__init__()
        uic.loadUi('UI/MainWindow.ui', self)

        # Initialisation des membres
        self.m_gamesManager = gamesManager
        self.optionDialog = OptionsDialog.OptionsDialog(gamesManager)

        # Connection des actions à leurs slots
        self.actionOptions.triggered.connect(self.onOptionDialog)
        self.actionExit.triggered.connect(self.onExit)
        self.actionNew_game.triggered.connect(self.onNewGame)

        # Appelé quand le menu Options est activé
        @Slot()
        def onOptionDialog(self):
            self.optionDialog.exec()

        # Appelé quand le menu 'Nouveau jeu' est activé
        @Slot()
        def onNewGame(self):
            isDicoValid, dicoContents, errorMessage =
Utils.getDicoContents(self.m_gamesManager.options.pathToDicos, self.m_gamesManager.options.lettersCount)
            if isDicoValid:
                game = self.m_gamesManager.createNewGame(dicoContents)
                gw = GameWidget.GameWidget(game)
                self.mdiArea.addSubWindow(gw)
                gw.show()
                game.start()
            else:
                QtWidgets.QMessageBox.critical(self,
                                                "Erreur dans Motus",
                                                "Le dictionnaire de mots de " +
str(self.m_gamesManager.options.lettersCount) + " lettres est invalide:\n" + errorMessage)

        # Appelé quand le menu Quitter est activé
        @Slot()
        def onExit(self):
            sys.exit(0)

```

d) MainWindow.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>732</width>
        <height>446</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Motus</string>
    </property>
    <widget class="QWidget" name="centralwidget">
      <layout class="QGridLayout" name="gridLayout">
        <item row="0" column="0">
          <widget class="QMdiArea" name="mdiArea"/>
        </item>
      </layout>
    </widget>
    <widget class="QMenuBar" name="menubar">
      <property name="geometry">
        <rect>
          <x>0</x>
          <y>0</y>
          <width>732</width>
          <height>26</height>
        </rect>
      </property>
      <widget class="QMenu" name="menuFile">
        <property name="title">
          <string>Fichier</string>
        </property>
        <addaction name="actionNew_game"/>
        <addaction name="separator"/>
        <addaction name="actionExit"/>
      </widget>
      <widget class="QMenu" name="menuTools">
        <property name="title">
          <string>Outils</string>
        </property>
        <addaction name="actionOptions"/>
      </widget>
      <addaction name="menuFile"/>
      <addaction name="menuTools"/>
    </widget>
    <widget class="QStatusBar" name="statusbar"/>
    <action name="actionOptions">
      <property name="text">
        <string>Options</string>
      </property>
    </action>
    <action name="actionNew_game">
      <property name="text">
        <string>Nouveau jeu</string>
      </property>
      <property name="toolTip">
        <string>Nouveau jeu</string>
      </property>
    </action>
    <action name="actionExit">
      <property name="text">
        <string>Sortir</string>
      </property>
      <property name="toolTip">
        <string>Sortir</string>
      </property>
    </action>
  </widget>
</ui>

```

```

</action>
<action name="actionAbout">
  <property name="text">
    <string>A propos</string>
  </property>
  <property name="toolTip">
    <string>A propos</string>
  </property>
</action>
</widget>
</resources/>
</connections/>
</ui>

```

e) *MotusTableWidget.py*

```

from PyQt5 import uic, QtWidgets, QtGui
from PySide2.QtCore import Slot, Qt
from game_engine.Game import LetterPlacement

class MotusTableWidget(QtWidgets.QTableWidget):
    """Cette classe implémente l'IHM de la grille de mots"""

    def __init__(self, game):
        super(MotusTableWidget, self).__init__()

        self.m_game = game
        self.m_lettersColor = [self.m_game.options.letterPlacementWellPlacedColor,
                               self.m_game.options.letterExistsColor,
                               self.m_game.options.letterDoesNotExistColor ]

        self.setColumnCount(self.m_game.options.lettersCount)
        self.setRowCount(self.m_game.options.maximumTriesCount)
        self.horizontalHeader().hide()
        self.verticalHeader().hide()
        for column in range(self.m_game.options.lettersCount):
            self.setColumnWidth(column, self.rowHeight(0)) # Les cellules seront carrées
            for row in range(self.m_game.options.maximumTriesCount):
                self.setLetter(row, column, "", LetterPlacement.doesNotExist)

    # Méthodes

    # Cette méthode place une lettre dans la grille et assigne la couleur en fonction de son placement
    def setLetter(self, row, column, letter, letterPlacement):
        cell = QtWidgets.QTableWidgetItem()
        cell.setTextAlignment(Qt.AlignCenter)
        cell.setText(letter)
        self.setItem(row, column, cell)
        cell.setBackground(QtGui.QColor(
            self.m_lettersColor[letterPlacement.value][0],
            self.m_lettersColor[letterPlacement.value][1],
            self.m_lettersColor[letterPlacement.value][2]
        ))

```


f) OptionsDialog.py

```

import sys
from PyQt5 import uic, QtWidgets
from PySide2.QtCore import Slot
from game_engine.Utills import Utills
from common.Parameters import Parameters

class OptionsDialog(QtWidgets.QDialog):
    """Cette classe implémente l'IHM de la boîte de dialogue d'options"""
    def __init__(self, gamesManager):
        super(OptionsDialog, self).__init__()
        uic.loadUi('UI/OptionsDialog.ui', self)
        self.m_gamesManager = gamesManager

        # Initialisations
        self.spinBoxLettersCount.setMinimum(Parameters.minimumWordsInDico())
        self.spinBoxLettersCount.setValue(self.m_gamesManager.options.lettersCount)
        self.spinBoxDuration.setValue(self.m_gamesManager.options.duration)
        self.spinBoxMaxTriesCount.setValue(self.m_gamesManager.options.maximumTriesCount)
        self.lineEditDicosPath.setText(self.m_gamesManager.options.pathToDicos)

        # Connections
        self.buttonBox.accepted.connect(self.onApplyNewValues)
        self.buttonBox.rejected.connect(self.onCancel)
        self.spinBoxLettersCount.valueChanged.connect(self.onLettersCountChanged)

        # Slots

        # Appelé quand le nombre de lettres a changé
        @Slot()
        def onLettersCountChanged(self, count):
            isDicoValid, errorMessage = Utills.isDicoValid(self.lineEditDicosPath.text(),
self.spinBoxLettersCount.value())
            if isDicoValid:
                self.buttonBox.button(QtWidgets.QDialogButtonBox.Ok).setEnabled(True)
                self.labelStatus.setText("")
            else:
                self.buttonBox.button(QtWidgets.QDialogButtonBox.Ok).setEnabled(False)
                self.labelStatus.setText(errorMessage)

        # Appelé quand on presse le bouton OK
        @Slot()
        def onApplyNewValues(self):
            options = self.m_gamesManager.options
            options.pathToDicos = self.lineEditDicosPath.text()
            options.lettersCount = self.spinBoxLettersCount.value()
            options.duration = self.spinBoxDuration.value()
            options.maximumTriesCount = self.spinBoxMaxTriesCount.value()
            self.m_gamesManager.options = options

        # Appelé quand on presse le bouton Cancel
        @Slot()
        def onCancel(self):
            self.buttonBox.button(QtWidgets.QDialogButtonBox.Ok).setEnabled(True)
            self.labelStatus.setText("")

```

g) OptionsDialog.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>OptionsDialog</class>
<widget class="QDialog" name="OptionsDialog">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>400</width>
      <height>300</height>
    </rect>
  </property>
  <property name="windowTitle">
    <string>Dialog</string>
  </property>
  <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
      <layout class="QFormLayout" name="formLayout">
        <item row="0" column="0">
          <widget class="QLabel" name="label_2">
            <property name="text">
              <string>Chemin vers les dictionnaires</string>
            </property>
          </widget>
        </item>
        <item row="0" column="1">
          <layout class="QHBoxLayout" name="horizontalLayout">
            <item>
              <widget class="QLineEdit" name="lineEditDicosPath"/>
            </item>
            <item>
              <widget class="QToolButton" name="toolButtonSelectDicosPath">
                <property name="text">
                  <string>...</string>
                </property>
              </widget>
            </item>
          </layout>
        </item>
        <item row="1" column="0">
          <widget class="QLabel" name="label">
            <property name="text">
              <string>Nombre de lettres</string>
            </property>
          </widget>
        </item>
        <item row="1" column="1">
          <widget class="QSpinBox" name="spinBoxLettersCount">
            <property name="maximumSize">
              <size>
                <width>16777215</width>
                <height>16777215</height>
              </size>
            </property>
          </widget>
        </item>
        <item row="2" column="0">
          <widget class="QLabel" name="label_3">
            <property name="text">
              <string>Nombre maximum d'essais</string>
            </property>
          </widget>
        </item>
        <item row="2" column="1">
          <widget class="QSpinBox" name="spinBoxMaxTriesCount"/>
        </item>
        <item row="3" column="0">
          <widget class="QLabel" name="label_4">
            <property name="text">
              <string>Temps maximum</string>
            </property>
          </widget>
        </item>
      </layout>
    </item>
  </layout>
</widget>
</ui>

```

```

        </property>
    </widget>
</item>
<item row="3" column="1">
    <widget class="QSpinBox" name="spinBoxDuration">
        <property name="suffix">
            <string> secondes</string>
        </property>
        <property name="minimum">
            <number>10</number>
        </property>
        <property name="maximum">
            <number>99999999</number>
        </property>
    </widget>
</item>
</layout>
</item>
<item row="1" column="0">
    <widget class="QLabel" name="labelStatus">
        <property name="text">
            <string/>
        </property>
    </widget>
</item>
<item row="2" column="0">
    <widget class="QDialogButtonBox" name="buttonBox">
        <property name="orientation">
            <enum>Qt::Horizontal</enum>
        </property>
        <property name="standardButtons">
            <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
        </property>
    </widget>
</item>
</layout>
</widget>
<resources/>
<connections>
    <connection>
        <sender>buttonBox</sender>
        <signal>accepted()</signal>
        <receiver>OptionsDialog</receiver>
        <slot>accept()</slot>
        <hints>
            <hint type="sourcelabel">
                <x>248</x>
                <y>254</y>
            </hint>
            <hint type="destinationlabel">
                <x>157</x>
                <y>274</y>
            </hint>
        </hints>
    </connection>
    <connection>
        <sender>buttonBox</sender>
        <signal>rejected()</signal>
        <receiver>OptionsDialog</receiver>
        <slot>reject()</slot>
        <hints>
            <hint type="sourcelabel">
                <x>316</x>
                <y>260</y>
            </hint>
            <hint type="destinationlabel">
                <x>286</x>
                <y>274</y>
            </hint>
        </hints>
    </connection>
</connections>
</ui>

```

C) subdicos_generator

1) subdicos_generator.py

```
import sys
from lxml import etree
import pickle
from common.Utils import Utils
from common.Parameters import Parameters

dictionnaireList = []

def getListOfWords(dicoEntry):
    inputList = dicoEntry.split() # Par défaut, la séparation se fait sur les espaces, ce qui est notre cas
    outputList = []

    # Certains mots, comme "100-mètres" contiennent un '\'. Nous le supprimons
    for word in inputList:
        word = Utils.getNormalizedToUpperCaseWord(word)
        if Utils.isWordValid(word):
            outputList.append(word)

    return outputList

def addWords(wordsList):
    # On calcule la longueur du mot le plus grand
    longestWordLength = 0
    for word in wordsList:
        if len(word) > longestWordLength:
            longestWordLength = len(word)

    # On prépare le tableau de dictionnaires de sortie:
    while len(dictionnaireList) < longestWordLength:
        dictionnaireList.append([])

    # On rajoute chaque mot à sa liste (numérotée par longueur de mots)
    for word in wordsList:
        index = len(word) - 1 # Le premier index est 1
        if word not in dictionnaireList[index]:
            dictionnaireList[index].append(word)

# Cette fonction écrit les listes dans un fichier.
# Pour les relire, utiliser: itemlist = pickle.load(fp)
def writeSubDico(listOfWord, charsCount, folder):
    name = folder + '\\' + str(charsCount) + ".dico"
    with open(name, 'wb') as fp:
        pickle.dump(listOfWord, fp)

def readDico(inputPath, outputPath):
    print("Lecture du fichier " + inputPath)
    tree = etree.parse(inputPath)
    print("Le fichier " + inputPath + "a bien été lu")

    wordsList = tree.xpath("/dico/entry/lemma")

    index = 0
    for word in wordsList:
        addWords(getListOfWords(word.text))
        index = index + 1
        print('\r', end='') # On revient au début de la ligne
        print("Analyse en cours. " + str(int(100 * index / len(wordsList))) + " % effectué.", end="")

    print("") # On saute une ligne

    for i in range(0, len(dictionnaireList)):
        l = dictionnaireList[i]
        wordsCount = len(l)
        print("Le dictionnaire " + str(i) + " contient " + str(len(l)) + " mots de " + str(i + 1) + " caractères")
        if wordsCount >= Parameters.minimumWordsInDico():
            writeSubDico(l, i + 1, outputPath)

if __name__ == "__main__":
    # Le premier argument est le chemin vers CE programme
    # C'est donc le deuxième argument qui doit contenir le chemin vers le fichier xml d'entrée
    # Et le troisième le répertoire où écrire les sous-dictionnaires
    if len(sys.argv) < 3:
        print("Usage: python dico_generator.py chemin_vers_dictionnaire_xml repertoire_d_enregistrement_des_sous_dictionnaires")
        sys.exit(1)

    # Le premier argument est 0, donc le deuxième est 1, ...
    readDico(sys.argv[1], sys.argv[2])
```

IX. Index alphabétique

IHM: Interface Homme Machine.....	4
Qt signals: Signaux émis par Qt.....	10
Qt slots: Fonctions spéciales Qt connectable à un signal, et appelée quand ce signal est émis.....	10
Widgets: "Windows Gatdget" (boutons, zones de textes, ...).....	6

X. Index des illustrations

Illustration 1: Visual Studio Community.....	5
Illustration 2: Qt Designer.....	6
Illustration 3: Liste des sous dictionnaires générés.....	7
Illustration 4: La fenêtre principale du jeu.....	9
Illustration 5: La boîte d'options.....	9
Illustration 6: Table 7 lettres 10 essais.....	9
Illustration 7: Le widget de jeu.....	10
Illustration 8: Le jeu terminé.....	11