Mini-projets
Jeux

Avant tout…

I. Environnement technique

Pour ce projet, vous aurez besoin :

- à minima de Python ou de la suite Anaconda (qui inclut Python et Jupyter Notebook)
- d'un environnement de développement (l'un ou l'autre, à votre convenance) :
  - Jupyter Notebook (par habitude...)
  - Thonny (si vous préférez un IDE plus complet mais très simple)
  - VSCodium (si vous souhaitez un IDE complet), avec les modules suivants :
    - Python (essentiel)
    - French Langage pack (conseillé)
    - Live server (conseillé)
    - Bracket Pair colorizer (conseillé)
- d'un fichier au format CSV, disponible sur Gitlab
- d'outils numériques complémentaires (optionnels mais souvent utiles) :
  - de communication (à votre convenance, mais si possible open source)
  - de stockage / partage (Gitlab ou e-lyco)
  - d'organisation de projet (ex : Gitlab, Trello,…)
  - de forge, un système de gestion de versions logicielles (Gitlab)

II. Objectifs pédagogiques

Pour ce projet, vous aurez besoin :

- de **définir un cahier des charges**, qui dépendra :
  - du temps que vous souhaitez y consacrer
  - de l'ambition que vous misez sur ce projet et, plus généralement, sur vos études
  - des compétences actuelles que vous maîtrisez en programmation
- de **vous organiser en équipe** :
  - de trois (c'est l'idéal)
  - de quatre (pour des projets encore plus ambitieux)
- de **respecter les consignes** de votre enseignant :
  - respect du délai
  - respect du formatage de rendu de projet (ex : nommage et format des fichiers)
  - respect du cahier des charges

Utilisation des modules Python :

- Il sera indispensable d'utiliser le module random, pour gérer l'aléatoire des parties.
- Il est **interdit d'utiliser le module csv** pour gérer l'export du fichier CSV.
- Plus généralement, il faudra limiter l'utilisation des modules à leur strict minimum.

III. Instructions Python utiles

Afin d'exploiter les données contenues dans le fichier CSV fourni (« Characters.csv »), vous aurez probablement besoin des instructions et des méthodes suivantes :

- with open(paramètres à insérer ici) as f:
  - qui permet d'ouvrir un fichier et d'en affecter son contenu à un objet f.
- chaine_de_caractère = f.readline()
  - qui permet de lire une ligne dans l'objet f crée à partir du fichier.
- liste = f.readlines()
  - qui permet de lire toutes les lignes contenues dans l'objet f crée à partir du fichier.
- for ligne in f :
  - qui est une autre façon de lire, une à une, chaque ligne du fichier.
- nouvelle_liste = chaine_de_caractere.split(séparateur)
  - qui permet de décomposer une chaîne de caractère d'après un séparateur.
- nouvelle_chaine = chaine_de_caractere.strip()
  - qui permet de supprimer les espaces en début et fin de chaîne de caractère.

IV. Gestion de projet

Avant de vous plonger tête baissée dans ce projet, il faut absolument au préalable :

- réfléchir ensemble à la structure de votre programme : faire un algorithme global
- réfléchir aux meilleurs types et structures de données à utiliser (booléens, listes, dictionnaires, classes,…)
- se mettre d'accord sur la modularité de votre programme (fonctions, procédures,…)
- se répartir le travail, en fonction des compétences de chacun et de sa disponibilité / motivation.
- vérifier régulièrement que votre travail respecte le cahier des charges, c'est à dire les consignes longuement décrites dans ce document.

V. Évaluation

Critères d'évaluation (note sur 10 points) :

- Respect du cahier des charges écrit dans les consignes (6 points)
  - Règles du jeu
  - Contraintes techniques (obligation ou interdiction d'utiliser telle ou telle fonction, structure de donnée, bibliothèque, ...)
  - Normalisation du rendu de projet (date, zone de dépôt, type et nom de fichier,…)
- Bonnes pratiques PEP-8 (1 point) (test possible sur pythonchecker.com)
  - Commentaires du code bien dosé (ni trop, ni trop peu)
  - Noms de variables explicites
  - Espaces bien placés
  - Lignes pas trop longues (en théorie, 80 caractères maximum)
- Implication globale, organisation du travail en équipe (1 point)
- Utilisation d'outils collaboratifs numériques permettant une organisation et un suivi du projet (Gitlab, Trello,…) (1 point)
- Point « subjectif » (1 point)
  - pas de critère particulier, c'est subjectif j'ai dit !

- Bonus (+ 2 points envisageables en fonction de l'ampleur du bonus)

VI. Rendu de projet

- Enregistrer le programme sous le format : Nom_du_jeu_NOM1_NOM2_NOM3.py
- Ajouter le fichier Characters.csv et tout autre fichier dont vous pourriez avoir besoin.
- Dans le cas où vous auriez crée un dossier, compresser l'ensemble de vos fichiers sous la forme Nom_du_jeu_NOM1_NOM2_NOM3.zip pour conserver l'arborescence.
- Votre projet doit être déposé sur l'espace de travail NSI d'e-lyco (« Vue d'ensemble », « tâches »), en temps et en heure.
- Il faut créer le groupe (ajout des noms de tous les participants) au moment du dépôt de fichiers.

VII. Niveau de difficulté

Des * sont notées à côté du nom du jeu. Elles estiment le niveau de difficulté.

Quidditch ****

I. Règles du jeu

On simule virtuellement une partie de Quidditch sur laquelle des élèves de Poudlard vont parier.

La partie est divisée en tranches de 5 minutes, pendant lesquelles :

- chaque équipe a 1 chance sur 3 de marquer un but (10 pts).
- chaque équipe a une chance sur 50 d'attraper le vif d'or, ce qui lui apporte 150 points et met fin à la partie.

A l'issue de la partie, les personnages se verront distribuer des gains en cas de victoire de leur « Maison ».

II. Cahier des charges

1. Coder une fonction permettant de jouer deux parties (Griffondor - Serpentard et Poufsouffle - Serdaigle).
   - L'issue de chaque partie doit déterminer l'équipe gagnante et le score.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de tirer au sort les paris de chaque personnage.
   - Chaque personnage parie sur sa propre maison (donc pour un seul match).
   - Chaque personnage doit parier sur un score (que vous devez rendre « crédible »)
4. Coder une fonction permettant d'attribuer les gains aux personnages.
   - Chaque personnage commence avec 2 points.
   - Pour participer, il doit utiliser 1 point (il doit donc toujours avoir des points pour jouer !)
   - Chaque pari gagné (équipe gagnante) fait remporter 5 points au personnage.
   - Le personnage se rapprochant le plus du score final de la partie (concept à définir précisément) gagne un bonus de 30 points (à condition que son équipe soit gagnante).
5. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire parier l'ensemble des personnages 10 fois (sur 10 match donc).
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Proposer une interface graphique.
- Donner la durée de chaque partie.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Bataille navale ***

I. Règles du jeu

On simule virtuellement une grille de bataille navale de 5x5 cases. Deux bateaux, symbolisés chacun par une seule case, sont placés aléatoirement dans cette grille. Une case ne peut contenir qu'un seul bateau.

Si le joueur choisit une case contenant un bateau, celui-ci est considéré « coulé » (le gain est alors de 10 points). Si un joueur choisit une case située sur la même ligne ou sur la même colonne qu'un bateau, on considère ce bateau « en vue » (le gain est alors de 2 points).

II. Cahier des charges

1. Coder une fonction permettant de jouer un tour de jeu aléatoire.
   - Un tour de jeu comprend trois coups au plus (il faut tenir compte de l'hypothèse des deux premiers coups gagnants, ce qui rendrait le troisième coup inutile), donc trois choix (différents) de cases au maximum.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Ajouter des bateaux de différents types (2, 3 ou 4 cases), sur une grille plus grande.
- Proposer une interface graphique pour afficher votre jeu (grille(s), coups, scores,...).
- Proposer un jeu à deux, chaque joueur intervenant l'un après l'autre.
- Proposer une IA (Intelligence Artificielle) permettant de jouer contre l'ordinateur.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Mastermind **

I. Règles du jeu

Le maître de jeu tire au hasard une suite de quatre nombres entiers différents, compris entre 0 et 9.

Le joueur tire aussi quatre nombres entiers différents, sans voir la combinaison du maître de jeu.

Le maître de jeu calcule le score en fonction du nombre d'éléments communs aux deux ensembles :

- 1 point si un élément est trouvé
- 3 points si deux éléments sont trouvés
- 8 points si trois éléments sont trouvés
- 20 points si tous les éléments sont trouvés

II. Étapes à suivre

1. Coder une fonction permettant de jouer un tour de jeu aléatoire.
   - Un tour de jeu comprend un tirage au sort du maître du jeu et un tirage au sort du joueur.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Utiliser des couleurs à la place des nombres.
- Proposer une interface graphique pour afficher votre jeu.
- Proposer un jeu à deux, avec les règles officielles : un joueur et un maître de jeu.
- Proposer une IA permettant de jouer contre l'ordinateur (celui-ci étant maître de jeu).

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Yam's ***

I. Règles du jeu

On lance cinq dés à six faces une seule fois.

Le score dépend de la combinaison de ce seul tirage :

- 1 point pour une paire (2 dés identiques)
- 2 points pour une paire (2 fois 2 dés identiques)
- 3 point pour un brelan (3 dés identiques)
- 5 points pour un carré (4 dés identiques)
- 7 points pour un full (3 puis 2 dés identiques)
- 9 points pour une suite (5 dés qui se suivent)
- 15 points pour un yams (5 dés identiques)

II. Étapes à suivre

1. Coder une fonction permettant de jouer un tour de jeu aléatoire.
   - Un tour de jeu comprend un lancé de dé par un joueur.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

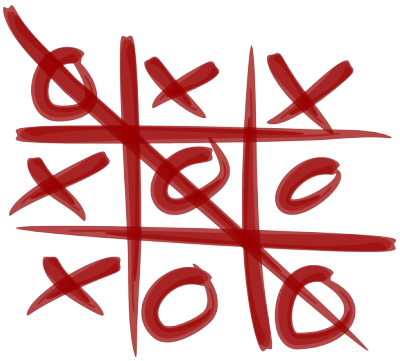
- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- S'approcher de la règle de jeu officielle en jouant chaque tour en trois étapes (garder au moins une combinaison valide, si elle(s) existe(nt), pour l'étape suivante, puis relancer les autres dés).
- Proposer une interface graphique pour afficher votre jeu (plateau de jeu, coups, scores,...).
- Proposer un jeu à deux, chaque joueur intervenant l'un après l'autre (avec un choix possible des dés à conserver).
- Proposer une IA (Intelligence Artificielle) permettant de jouer contre l'ordinateur
- Proposer une interface pour deux joueurs distants

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Tic tac toe **

I. Règles du jeu

Sur une grille 3x3, un joueur et une intelligence artificielle placent, l'un après l'autre, une croix ou un cercle. Le gagnant est celui ayant réussi à créer une ligne avec trois de ses motifs.



Le score du joueur se définit ainsi :

- 1 point en cas de défaite
- 5 points en cas de match nul
- 10 points en cas de victoire

II. Étapes à suivre

1. Coder une fonction permettant de jouer un tour de jeu aléatoire.
   - Un tour de jeu comprend une grille complète (ou incomplète si une ligne est obtenue).
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

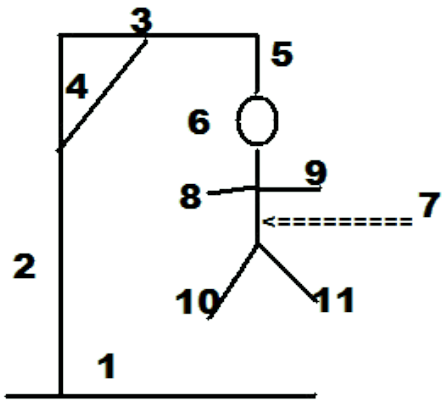
- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Proposer une interface graphique pour afficher votre jeu (grille, coups, scores,...).
- Proposer un jeu à deux, chaque joueur intervenant l'un après l'autre.
- Proposer une IA (Intelligence Artificielle) crédible (et non plus aléatoire) permettant de jouer contre l'ordinateur.
- Proposer une interface pour deux joueurs distants.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Pendu ***

I. Règles du jeu

Un joueur doit trouver un mot inconnu (ici, le mot « NSI ») en proposant une succession de lettres. Chaque proposition de lettre non présente dans le mot mystère est sanctionnée par l'ajout d'un élément du dessin du pendu. Au onzième et dernier élément dessiné, le joueur a perdu. Au contraire, s'il complète le mot avant ce onzième élément, le joueur a gagné.



Le score du joueur se définit ainsi :

- 22 point si le mot est trouvé sans erreur
- 20 points si le mot est trouvé avec une erreur
- 18 points si le mot est trouvé avec deux erreurs
- …
- 0 point si le mot n'est pas trouvé
- Dans ce dernier cas, on attribue 1 point par lettre trouvée

II. Étapes à suivre

1. Coder une fonction permettant de jouer un tour de jeu aléatoire.
   - Un tour de jeu comprend un tirage au sort successif de lettres différentes par un joueur, jusqu'à trouver mot mystère « NSI » ou jusqu'à… une défaite (11 lettres tirées) !
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Proposer une interface graphique pour afficher votre jeu (mot mystère, pendu, scores,...).
- Proposer une interface permettant de jouer réellement en choisissant les lettres.
- Intégrer une liste de mots prédéfinis, permettant de choisir un mot mystère différent à chaque tour.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Tennis ***

I. Règles du jeu

Au tennis, la façon de compter les points est... particulière !

Faire une recherche sur cette comptabilisation (Points, jeux, sets et match).

Le match sera remporté en deux sets gagnants.

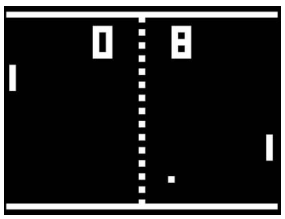
II. Étapes à suivre

1. Coder une fonction permettant de jouer un match aléatoire.
   - Un match est terminé en deux sets gagnants.
   - Le score renvoyé devra contenir les résultats de chaque set (ex : 6-4 3-6 7-6)
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : nombre de victoires), par ordre de nombre de victoires décroissant.
   - d'afficher la moyenne de l'ensemble nombre de victoires en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Pondérer les chances de gain de chaque points en fonction du niveau des joueurs (affecter un coefficient « ATP » à chaque joueur)



- Pong est un jeu vidéo datant de plus de quarante ans (sorti en 1972). Il propose une interface simplifiée où deux joueurs de tennis tentent de se renvoyer la balle. Deux choix très différents vous sont proposés :
  - Coder une interface graphique dynamique de ce jeu (deux joueurs).
  - Coder une interface graphique au coup par coup. Dans cette version très simplifiée, chaque côté est divisé en trois parties (Haut (H) / Milieu (M) / Bas (B)). Chaque joueur envoie à son tour la balle dans une direction donnée (H/M/B) et son adversaire ne renvoie la balle que s'il est placé dans cette même position.
- Intégrer le vrai décompte de points du tennis à votre Pong.
- Proposer une IA (Intelligence Artificielle) permettant de jouer contre l'ordinateur. Cette évolution est possible pour les deux cas décrits précédemment.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Démineur **

I. Règles du jeu

Démineur est un jeu vidéo très connu car il est fourni gratuitement avec l'OS Windows depuis plus de 20 ans.

Il consiste à dévoiler toutes les cases d'une grille, à l'exception de celles masquant une mine. Chaque case déjà dévoilée permet de connaître le nombre de mines présentes dans son environnement direct (cases adjacentes).

Dans notre version très simplifiée, on considérera une grille de 4x4 cases. Trois mines y sont placées de façon aléatoire. Le joueur gagne s'il dévoile les 13 cases ne cachant pas de mines (cases « libres »).

Le score du joueur se définit ainsi :

- 50 points si les 13 cases libres sont trouvées.
- Si le joueur dévoile une mine avant la fin, il gagne 1 point par case libre dévoilée.

II. Étapes à suivre

1. Coder une fonction permettant de jouer une partie aléatoire.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

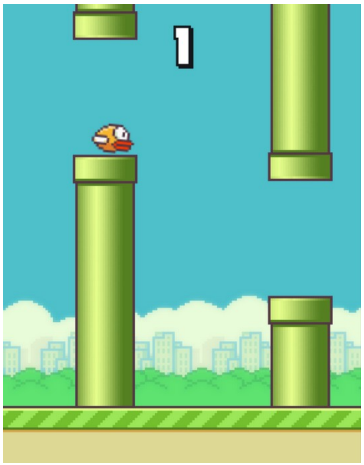
En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Donner le nombre de mines cachées sous les cases adjacentes de chaque case libre dévoilée.
- Proposer une interface graphique (grille, coups, scores,...), avec entrée des coups au clavier.
- Proposer une interface graphique, avec entrée des coups à la souris.
- Proposer une interface se rapprochant le plus possible du jeu original.

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.

Flappy bird *

I. Règles du jeu



Flappy bird est un jeu vidéo d'obstacles sorti en 2013.

Il consiste à donner suffisamment d'altitude à un oiseau pour qu'il passe à travers des obstacles (des tuyaux) lors d'un défilement horizontal continu. La partie est terminée lorsqu'un tuyau est touché par l'oiseau.

Notre version simplifiée consiste à définir trois zones pour chaque obstacle (Haut, Milieu et Bas). A chaque obstacle, la zone de passage est choisie aléatoirement. L'oiseau tente un passage aléatoire en volant au niveau d'une de ces trois zones. Si les zones de l'obstacle et de l'oiseau concordent, il peut passer à l'obstacle suivant, sinon le jeu se termine.

Le score du joueur se définit ainsi :

- 50 points si 4 obstacles sont franchis (score maximal).
- Sinon, 10 point par obstacle franchi.

II. Étapes à suivre

1. Coder une fonction permettant de jouer une partie aléatoire.
2. Coder l'importation des personnages inclus dans le fichier « Characters.csv ».
   - Cet import doit aboutir, à minima, à un tableau incluant les noms des personnages.
3. Coder une fonction permettant de faire jouer tous les personnages, tour à tour.
   - Chaque score doit être conservé, en relation avec le personnage qui vient de jouer.
4. Coder une IHM (Interface Homme-Machine) permettant :
   - de faire jouer l'ensemble des personnages 10 fois.
   - d'afficher le classement final des personnages (Nom : score), par ordre de score décroissant.
   - d'afficher la moyenne de l'ensemble scores en fin de classement.
   - de proposer à l'utilisateur de relancer le programme pour refaire une série de parties.

III. Bonus optionnels

En fonction de votre motivation, de vos compétences et du temps que vous souhaitez y consacrer, voici quelques pistes optionnelles :

- Effectuer un import plus large des personnages (incluant leurs caractéristiques).
- Afficher des caractéristiques du gagnant (sa « Maison », sa baguette,…).
- Proposer un classement par « Maison » (Griffondor, Serpentard,…).
- Donner le choix au joueur de sa position à chaque passage d'obstacle.
- Proposer une interface graphique simple (position de l'oiseau par rapport à l'obstacle, sans défilement continu), avec entrée des coups au clavier.
- Proposer une interface graphique, avec entrée des coups à la souris (1 clic pour la position basse, 2 pour le milieu et 3 pour la position haute).
- Proposer une interface se rapprochant le plus possible du jeu original.
- En proposer une version Android

Les bonus ne se substituent pas au cahier des charges. Si vous ajoutez des bonus, ils doivent s'ajouter aux fonctionnalités de base, quitte à gérer deux modes de jeu dans une IHM adaptée.