PRÉSENTATION PROGRAMMATION
SUPER MARIO ALLUMETTES

I)Expression des besoins impératifs:

→Le but du projet est de présenter un jeu fonctionnel, reprenant le principe du jeu d'allumettes et toutes les problématiques liées à celui-ci et listées dans les consignes.

→Le deuxième objectif est de rendre le dit jeu des allumettes plus “esthétique”, c’est-à-dire transformer le jeu de base pour le rendre plus attrayant pour les utilisateurs.Nous entendons par là qu’il faut translater le jeu des allumettes à un univers existant ou de notre imagination qui pourrait intéresser l'utilisateur.

→ Le troisième besoin est lié au niveau de difficulté choisie: le niveau 3, outre les problématiques techniques que nous énonçons ci-dessous, nous impose de modéliser les allumettes par “tas”, il faut donc trouver ou créer un univers nous permettant de symboliser ces tas.

II)Description de l’idée générale de l’environnement du jeu :

- Pour répondre aux trois besoins impératifs du jeu nous avons imaginé le jeu suivant (pour les problématiques techniques voir cahier des charges/maîtrise d’oeuvre) :

  →Pour rendre le jeu attractif,nous avons décidé de nous rattacher à l’univers de Mario, licence ayant une grande cote de popularité auprès des joueurs du monde entier et que tout le monde connaît.

  →Dans le jeu que nous avons décidé d’appeler “Super Mario allumettes”, mario (l’utilisateur) affronte bowser (l’ordinateur) autour d’un mini jeu similaire au jeu des allumettes de base:
  - Les allumettes sont associées à des goombas (petits champignons marrons de l’univers mario).

    
  - les goombas sont répartis sur des étages de briques (symbolisation des tas)
  - L’utilisateur commence.
  - L’utilisateur (mario) choisit l’étage et le nombre de goombas (attention règle) qu’il souhaite enlever
  - L’ordinateur (bowser) choisit de manière aléatoire le tas et le nombre de goomba à supprimer
  - Le joueur qui prend la dernière allumette a gagné

III)Répartition du travail

- Il est important dans cette section, qu'il y a au sein du binôme une différence de niveau importante, Loïc ayant plus d'expérience en codage que Rémi:

Rémi:
→ Création des fonctions liées au tracé des personnages
→ importation du fond d'écran, photoshop
→ tests utilisateur
→ Rédaction du rapport

Loïc:
→ création des fonctions liées à l'interaction graphique à l'aide des flèches directionnelles
→ création des fonctions de l'interface d'accueil (menu règle, singleplayer, crédits)
→ application de la partie graphique au jeu.

IV)Déroulement du projet suivant la structure du code

Phase 1: **codage du jeu des allumettes niveau 3 de "base"** sans interface graphique, choix de l'univers de mario et de la représentation des allumettes

Phase 2: codes des fonctions setpersonnages_…permettant de tracer les personnages:

Problème 1: impossibilité de tracer les personnages à l'aide des fonctions turtle de base:
- Tout d'abord parce que le travail d'utiliser les commandes de base de turtle est très fastidieux et ne permet pas de réaliser des formes très précises.
- D'autre part, même en utilisant speed(0), le chargement de tous les personnages aurait été beaucoup trop long.

Solution 1: Pour résoudre ce problème nous avons décidé de "réencoder" des images pixélisées pour pouvoir les tracer en python:
- noter les codes RVB des couleurs des photos pixélisées (goombas et autres personnages) et les stocker dans une variable **couleurs**
- décomposition des photos en pixels stockés dans une matrice, contenue dans une variable **personnages**

Fonctions correspondantes à cette phase:
-print_personnage() affiche 1 personnage i de taille s (largeur d'un pixel d'image en pixels)

-**set_goomba()** affiche le personnage de coordonnées (j,i) sur la fenêtre, j est le numéro de tas, i est l'indice du personnage dans le tas. perso est le numéros de personnage à afficher
-setpersonnages() appelle print_personnages() pour tracer les autres briques et set_goomba() pour tracer les autres goombas
- **setpersonnages_joueur()** affiche mario et bowser aux coordonnées voulues.
-set_fleche_joueur() affiche la flèche au dessus de mario et bowser
- **set_fleche_goomba()** place la flèche sur les tas de goomba

Phase 3: import du fond d'écran

Problème 2: la méthode des pixels impossible à appliquer au fond d'écran (trop long à tracer et rend le jeu injouable)

Solution 2 : import d'un fond d'écran **bgrd.gif** (modifier au préalable avec photoshop)

Fonctions correspondantes à cette phase:

- **setenv()** qui crée l'affichage de la fenêtre de jeu (taille fenêtre, icône de la page,...)
- **hide_title()** fonction permettant de cacher le titre du jeu (importation du fond d'écran modifiée dans la phase pour avoir 1 seule importation d'image permettant de faire l'affichage du menu principal et du menu)

Phase 4: application du programme de jeu allumette à l'environnement graphique

Problème 3 : la fonction set_goomba permet de tracer les goombas. Le problème étant qu'au moment de tester l'animation de mort des goombas, nous nous sommes aperçus que la tortue ne peut pas effacer un seul goomba (elle efface donc tous les goombas!)

*Solution 3 :*dans la fonction set_goomba() une tortue trace un goomba (ex: si il y a 15 goombas à l'écran, 15 tortues seront dédié à l'affichage des goombas)

→Création d'un écran "you win" en case de victoire et d'un écran "game over" en cas de défaite

Problème 4 : lorsque je presse play again sur l'affichage de victoire ou de défaite, les tortues des anciennes parties entrent en conflit avec les tortues de la nouvelle partie (goombas non supprimés, briques partiellement détruites…)

Solution 4: Résolution du conflit en effaçant le travail de toutes les tortues et en relançant single_player

Fonctions correspondantes à cette phase:

- **singleplayer()** amorçant le jeu
- **tour_ordi():** qui permet le tour de l'ordinateur
- **select()** fonction que nous avons défini dans la partie 5, c'est elle qui permet réellement de jouer

Phase 5: définitions des fonctions de contrôles directionnels à l'aide des flèches

→Pour rendre le jeu plus agréable à jouer l'interaction ne se fait plus au niveau du terminal, mais au clavier.

Fonctions correspondantes à cette phase:

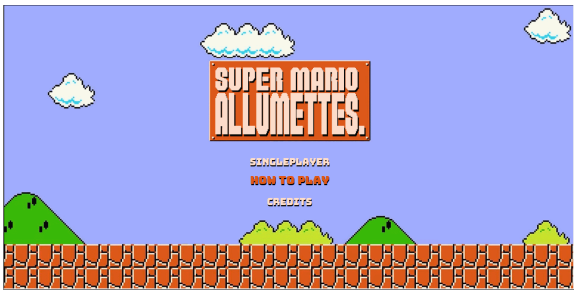
- **up_down** permettant de changer de tas, fonction modifiée dans la phase 6 pour pouvoir naviguer dans l'interface d'accueil.
- **left_right()** permettant de faire varier le nombre d'allumettes.
- **select()** permettant de valider la sélection du nombre de goombas et le menu auquel on veut accéder

Phase 6: création de l'interface d'accueil:

→Modification des fonctions up_down, left_right et sélect pour pour pouvoir créer l'environnement graphique de départ.

Fonctions associées à cette phase:

- **mainmenu()** préparation de la page de jeu
- **menuregle()** permettant d'accéder à la page des règles
- **menucredit()** génère l'interface des crédits
- **update_menu()** qui permet de changer de l'interface du menu aux interfaces singleplayer/règles et menu



Phase 7 perfectionnement du code et de certains détails:

Problème 4: Après une série de test, nous nous sommes rendu compte que dans le cas où il y avait moins de goombas sur un tas que du nombre minimum de goomba de la règle, il était impossible d'éliminer les goombas de ce tas, ce qui nous empêchait de finir le jeu.

Solution 4 : Pour régler ce problème, on a décidé de substituer le nombre de goombas du tas non sélectionnable au nombre de goomba sélectionné sur un tas jouable.
ex: ma règle de jeu commence à 2 allumettes, sur un tas après sélection il ne reste que 1 goomba sur le tas, je ne peux plus sélectionner ce tas, si je sélectionne un autre tas avec 4 goomba et j'en supprime 3, cela tuera 1 goomba sur ce tas, et les goombas sur le tas non sélectionnable)

Problème 5: Dans la version actuelle,on ne voit pas l'animation du tour ordinateur, la flèche se met directement sur le tas sélectionné, supprime le goomba, le tout en une fraction de seconde, passant directement au tour joueur.

Solution 5 : Pour simuler le tour de l'ordinateur, on a créé une boucle while, tant qu'on est pas sur le tas choisi par l'ordinateur (décidé de manière aléatoire), la flèche des tas se déplace de manière aléatoire de bas en haut jusqu'à atterrir sur le tas sélectionner. Cela crée l'illusion que l'ordinateur joue.

V)Nouvelles versions envisagées (travail d'amélioration envisagé)

→ Si l'on voulait améliorer le code, l'idée principale serait de programmer le tour ordinateur pour que celui ne joue plus de manière aléatoire,mais de façon intelligente. Cependant, cela dépasse les conditions attendues et de manière générale cela dépasse nos capacités. (développement d'une IA)

→ Trouver un moyen d'importer la police d'écriture nécessaire sans devoir l'installer : aucune méthode cross-platform n'a été trouver lors de la réalisation du projet, cependant, le module Pillow pourrait résoudre le problème.
→Par soucis d'esthétisme,nous avons limité le nombre de tas et le nombre d'allumettes dans les tas, sinon les tas dépasseraient la fenêtre turtle et les goombas seraient tracés sur les personnages. L'idée ici, serait de réduire ou d'augmenter la taille des goombas/briques en fonction du nombre de tas et de goombas demandés. Le problème étant que cette fonctionnalité serait difficile à mettre en place au vu du placement des goombas avec des coordonnées écrites "en dur".

→pouvoir modifier la taille de la fenêtre: en effet, la taille de la fenêtre est fixe (1440x720) car : - le fond d'écran a cette résolution et un redimensionnement dynamique de celui-ci nécessiterait l'utilisation du module Pillow.
- les goombas sont placé à des coordonnées fixes.

→Les idées suivantes sont d'ordre esthétique et non prioritaire:

- faire une flèche intermédiaire lors du changement des tas pour plus de dynamisme (Fait)
- aligner la flèche en fonction de la longueur du tas
- créer une animation des personnages au moment du changement de joueur (Fait: une flèche se déplace au dessus du joueur en train de jouer)