Projet 1DEV :

I. Pourquoi avons-nous choisi Python avec Pygame ?

Nous avons choisi le langage python car tous les membres du projet possédaient déjà des bases dans ce langage. De plus python est un langage pour lequel on trouve beaucoup de documentations et il possède beaucoup de librairies faciles à utiliser.
L'utilisation de pygame était pour nous une évidence car cette librairie d'interface graphique est rapide à prendre en main et a déjà été utilisé par plusieurs membres, elle nous permet de respecter une des contraintes qui est d'être compatible sur les différents OS.

II. Explication de nos algorithmes :

L'interface :

1. Taille de l'interface :

L'interface est initialisée à 1200 pixels en longueur et 800 pixels en hauteur.

```
#paramètres de la fenetre
fenetre = pygame.display.set_mode((1200,800))
pygame.display.set_caption("Mr.Driller")
pygame.time.Clock().tick(60)
pygame.key.set_repeat(1,15)
font=pygame.font.Font(None, 24)
font2=pygame.font.Font(None, 50)
```

La seconde ligne permet de nommer la fenêtre où se déroulera le jeu, la suivante donne au jeu 60 FPS (frame per second) puis celle d'après permet de prendre en compte le fait de rester appuyer sur des touches.
Enfin, les deux dernières instaurent des polices d'écriture. Celles-ci seront utilisés pour l'affichage de texte en jeu ou dans le menu.

2. Code de l'horloge :

L'horloge est uniquement décorative et servira de décor. Elle sera affichée dans le hud, la partie droite de la fenêtre en jeu.

On stocke dans une liste les 60 images de la montre.

```
#images du timer
horloge = []
horloge.append(pygame.image.load("horloge/1.png").convert_alpha())
horloge.append(pygame.image.load("horloge/2.png").convert_alpha())
horloge.append(pygame.image.load("horloge/3.png").convert_alpha())
```

On fait apparaître une image par seconde et faisant bien attention qu'elle se suive :

```
fenetre.blit(horloge[secondes],(1000,50))
```

Les images doivent être de la même taille pour qu'on ait bien l'impression que la montre défile naturellement.
Arrivé à 59 secondes, les secondes passent à zéro et le chrono redémarre.
On peut donc enchaîner sur la fonction qui nous donne le temps :

```
#gestion du temps
def theTimer(start,temps,saveTemps,secondes,minutes,timer):
    now = time.time()
    temps = int(now-start)

    if temps!=saveTemps:
          secondes += 1
          if secondes == 60:
                secondes=0
                minutes += 1
          timer = font.render(("Timer     " + str(minutes) + "min " + str(secondes) + "sec"),1,(255,255,255))
    saveTemps = temps
    return temps,saveTemps,secondes,minutes,timer
```

On crée une boucle où on l'on vérifie que le temps ne s'est pas arrêté et où on affiche les minutes dès qu'on a atteint 60 secondes.
Avec str(), on met les variables sous une chaîne de caractères pour pouvoir les afficher.

3. Code de la vie :

```
#fonction d'affichage des éléments du hud
def affichageBarreDeVie(barreVie, nbrVie, vie, saveSecondes, secondes, horloge, timer):
    if vie > 100:
        vie = 100
    if saveSecondes != secondes:
        saveSecondes = secondes
        vie-=1

    barreVie = font.render((str(vie) + "%"),1,(0,0,0))
    afficherProfondeur = font.render(("Profondeur :  "+str(CasePersoY-3+103*level)),1,(255,255,255))
    afficherScore = font.render(("Score :  "+str(score)),1,(255,255,255))
    fenetre.blit(journal,(750,500))
    fenetre.blit(timer,(800,100))
    fenetre.blit(horloge[secondes],(1000,50))
    fenetre.blit(cadre,(950,0))
    fenetre.blit(afficherProfondeur, (800,200))
    fenetre.blit(afficherScore, (800,150))

    if vie >=0:
        pygame.draw.circle(fenetre, (114,103,103), (875,450), 85)
        pygame.draw.circle(fenetre, (0,0,0), (875,450), 85, 5)
        pygame.draw.circle(fenetre, (143,22,22), (875,450), (round(vie*0.8)))
        fenetre.blit(barreVie,(860,445))
    else:
        nbrVie -= 1
        sonMort.play()
        vie = 100

    return barreVie, nbrVie, vie, saveSecondes
```

Dans un premier temps, on fait en sorte que la vie ne dépasse pas 100%, puisque c'est mathématiquement faux.
Ensuite, le code permet que l'on perde une vie toutes les secondes, puis on intègre cette variable vie dans l'interface et on dessine un disque rouge qui s'ajuste en fonction du pourcentage de vie avec un cercle noir autour.
On a aussi le code qui fait apparaître l'horloge et le temps.
Quand le personnage perd une vie, on active un son et on remet sa vie à 100 tout en lui enlevant un cœur.

4. Code des cœurs :

```
#systeme de vies
def lesVies(coeurRouge,coeurGris,nbrVie, font, musique, game, fin):
    if nbrVie == 3:
        fenetre.blit(coeurRouge,(800,270))
        fenetre.blit(coeurRouge,(850,270))
        fenetre.blit(coeurRouge,(900,270))
    elif nbrVie == 2:
        fenetre.blit(coeurRouge,(800,270))
        fenetre.blit(coeurRouge,(850,270))
        fenetre.blit(coeurGris,(900,270))
    elif nbrVie == 1:
        fenetre.blit(coeurRouge,(800,270))
        fenetre.blit(coeurGris,(850,270))
        fenetre.blit(coeurGris,(900,270))
    elif nbrVie == 0:
        barreVie = font.render("0%",1,(0,0,0))
        pygame.draw.circle(fenetre, (114,103,103), (875,450), 85)
        pygame.draw.circle(fenetre, (0,0,0), (875,450), 85, 5)
        pygame.draw.circle(fenetre, (143,22,22), (875,450), 0)
        fenetre.blit(barreVie,(860,445))
        fenetre.blit(coeurGris,(800,270))
        fenetre.blit(coeurGris,(850,270))
        fenetre.blit(coeurGris,(900,270))
        pygame.display.flip()
        roblox.play()
        pygame.time.delay(500)
        pygame.mixer.music.stop()
        musique = False
        game = False
        fin = True

    return musique, game, fin
```

En fonction de la variable “nbrVie” on va afficher un certain nombre de coeur rouge ou gris. “fenetre.blit” désigne notre fenêtre où on affiche l’image, “coeurRouge” ou “coeurGris” désignent l’image à afficher, et les deux nombres à côtés désignent les pixels où on doit afficher l’image.
La ligne roblox.play() met le son quand on meurt, le time.delay fait une pause de 500 millisecondes dans le jeu pour un meilleur visuel, on arrête la musique et on la met en False, pareil pour le jeu et on active la fin qui est de rentrer son nom pour le score.
On retourne aussi les variables.

5. Code du score :

Classement des scores dans le menu :

Avec la fonction “NewScore”, on ajoute le pseudo et le score dans le fichier score.txt
Le pseudo sera demandé au joueur lorsque sa partie sera terminée. Pour ce faire, nous avons créé la fonction Inputer, qui à chaque appui d’une touche, ajoute le caractère à la variable “monInput” que la fonction finira par retourner.

```
monInputGraph = font2.render(("Votre nom est :  " + monInput),1,(255,255,255))
disp_score = font2.render(("Score :  " + str(score)),1,(255,255,255))
fenetre.blit(monInputGraph,(480,500))
fenetre.blit(disp_score,(500, 350))
return monInput,maj,continuer,saisieDuPseudo, reinitialiser
```

Afin de lier le score aux pseudos, on utilise un dictionnaire que l'on crée dans la fonction "Scorelist". On ouvre donc score.txt et on utilise une liste qui contiendra tous les scores et pseudos pour enfin remplir le dictionnaire "dic". Dans celui-ci, les clés sont les pseudos et les valeurs leur correspondant sont les scores. On termine par trier par ordre décroissant le dictionnaire selon les scores.

```
#on recupere les scores sous forme de dictionnaire
def Scorelist():
    scoretxt = open("score.txt", "r")
    liste = scoretxt.readline().split(";")
    dic = {}
    for i in range(0, len(liste), 2):
        dic[liste[i]] = int(liste[i+1])
    dic = sorted(dic.items(), reverse=True, key=lambda t: t[1])
    return dic
```

Enfin, la fonction "AfficherScore" affichera simplement les 10 premières clés et valeurs du dictionnaire précédemment créé, ce qui correspondra aux 10 meilleurs scores.

6. <u>Bouton pause :</u>

```
#boucle de pause
elif pause:
      for event in pygame.event.get():

            if event.type == QUIT:
                  continuer = 0

            if event.type == KEYUP:

                  if event.key == K_ESCAPE:
                        continuer = 0

                  elif event.key == K_p:
                        game = True
                        pause = False
                        fondBlit = 0

      fenetre.blit(imagePause, (0,0))

      if fondBlit < 8:
            fenetre.blit(fondPause, (0,0))
            fondBlit += 1
```

La pause en jeu est une nouvelle boucle, qui nous fera sortir de la boucle de jeu si la touche P est appuyée. On affiche alors une fenêtre indiquant que le jeu est en pause et derrière un

masque gris pour créer un “flou” sur le jeu en pause. On attend bien évidemment l’appui de la touche P pour sortir de la boucle pause pour reprendre la partie, techniquement on traduit cela par le passage de la variable game à True et pause à False.

10. Déplacement du personnage :

```
#mouvements du perso
if direction == 1:
      fenetre.blit(droite[D],(persoX,persoY))
elif direction == 2:
      fenetre.blit(gauche[G],(persoX,persoY))
elif direction == 3:
      if bas == 1:
            fenetre.blit(basDroite,(persoX,persoY))
      elif bas == 2:
            fenetre.blit(basGauche,(persoX,persoY))
elif direction == 4:
      fenetre.blit(haut,(persoX,persoY))
if score < 0 or (CasePersoY-3+103*level) == 0:
      score = 0
```

Pour déplacer le personnage, on affiche à la suite les différentes images de gauche et de droite qui sont stockées dans des listes afin d’avoir un effet de mouvement.
Si le personnage tombe on affiche l’image du personnage vers le bas.

Fonctionnement des blocs :

1. Comment apparaissent-ils aléatoirement :

```
#generation des niveaux aleatoirement avec des probas différentes
def GenerateLevel():
      list=["niveaux/niveau1.txt", "niveaux/niveau2.txt", "niveaux/niveau3.txt", "nive
```

Au début, on stocke dans une liste les noms des fichiers texte de chaque niveau.

On va ensuite créer des listes “elements_level” pour chaque niveau afin de personnaliser les probabilités d’apparition de chaque bloc dans celui-ci.
Chaque bloc est représenté par un chiffre dans les fichiers texte générés par la fonction, plus ce chiffre est présent, plus la probabilité d’apparition du bloc associé est élevée. Ainsi, tous les niveaux sont générés aléatoirement et nous pouvons augmenter la difficulté des niveaux en réduisant, par exemple, la probabilité d’apparition des seringues.

```
#Listes probabilités des types de blocs

elements_level_1= ['1','2','3', '3', '3', '3', '3', '3', '3', '4', '4', '4', '4', '4', '4', '5', '5'

elements_level_2= ['1', '1', '2', '2', '3', '3', '3', '3', '3', '3', '3', '4', '4', '4', '4', '4', '

elements_level_3= ['1', '4', '4', '4', '4', '4', '4', '5', '5', '5', '5', '5']
```

On peut alors créer les 10 fichiers de niveaux grâce à une boucle. Dans chaque fichier, on commence par écrire 4 lignes de 0 pour séparer les niveaux qui s’enchaineront. Ensuite on écrit sur un ligne 7 caractères qui sont sélectionnés aléatoirement dans la liste d'élément

correspondant au niveau actuel. On termine la ligne par un 0 pour délimiter l'espace de jeu. On exécute 100 fois cette ligne afin que les niveaux aient bien 100 blocs de profondeur.

```
for j in range(0,10):
      level = open(list[j], "w")
      level.write("00000000\n")
      level.write("00000000\n")
      level.write("00000000\n")
      level.write("00000000\n")
      for i in range(100):
            for y in range(7):
                  if j ==0:
                        level.write(random.choice(elements_level_1))
                  elif j ==1:
                        level.write(random.choice(elements_level_2))
                  elif j ==2:
                        level.write(random.choice(elements_level_3))
                  elif j ==3:
                        level.write(random.choice(elements_level_4))
                  elif j ==4:
                        level.write(random.choice(elements_level_5))
```

2. Comment fonctionne la gravité et les collisions :

Collisions :

On commence par récupérer les bords de la "hitbox" du personnage.

```
#recuperation de la position du personnage dans notre "niveau" (coordonnees dans liste "niveau")
def collision(structure_niveau, persoX, persoY, hauteur):
    Droite = persoX + 83
    Gauche = persoX + 17
    Haut = hauteur - 500 + 66
    Bas = hauteur - 500 + 99
```

Une fois les bords récupérés, on calcule ensuite la position des différents côtés de la hitbox dans la liste "structure_niveau".

```
    CasePersoX = (persoX+50)//100
    CasePersoY = -((hauteur - 400) //100) + -1
    PersoDroite = Droite//100
    PersoGauche = Gauche//100
    PersoHaut = -(Haut//100) - 1
    PersoBas = -(Bas//100) - 1

    return CasePersoX, CasePersoY, PersoDroite, PersoGauche, PersoHaut, PersoBas
```

Gravité :

Le personnage chute de 4 pixels par tour de boucle, si on arrive à une certaine profondeur on change de niveau et si le bloc sous le personnage est une seringue, il l'a récupère automatiquement et tombe.
On récupère aussi une variable "tombe" pour pouvoir détecter si le personnage est en train de tomber.

```
#gravite
if CasePersoY+1 >= 105: #changement de niveau
      direction = 3
      hauteur -= 5
      score+=5000
      tombe = True
      CasePersoY = 0
      level += 1
      hauteur = 400
      game = False
      changementNiveau = True
elif structure_niveau[CasePersoY+1][CasePersoX] == 0:
      direction = 3
      hauteur -= 5
      score+=1
      tombe = True
elif structure_niveau[CasePersoY+1][CasePersoX] == 1:
      structure_niveau[CasePersoY+1][CasePersoX] = 0
      vie+=20
      score+=500
else :
      tombe = False
```

3. Les différents types de blocs et leur destruction :

De base, on charge les images des blocs dans des variables :

```
#blocs
oxygene = pygame.image.load("blocs/oxygene.png").convert_alpha()
bleu = pygame.image.load("blocs/blocBleu.png").convert_alpha()
rouge = pygame.image.load("blocs/blocRouge.png").convert_alpha()
vert = pygame.image.load("blocs/blocVert.png").convert_alpha()
jaune = pygame.image.load("blocs/blocJaune.png").convert_alpha()
brune = pygame.image.load("blocs/caisseBrune.png").convert_alpha()
blanche = pygame.image.load("blocs/caisseBlanche.png").convert_alpha()
tnt = pygame.image.load("blocs/tnt.png").convert_alpha()
tnt1 = pygame.image.load("blocs/tnt1.png").convert_alpha()
tnt2 = pygame.image.load("blocs/tnt2.png").convert_alpha()
```

Ensuite, on lui dit d'afficher le bon bloc en fonction de la valeur qui sera dans le fichier niveau.
On analyse ligne par ligne le fichier texte, et on blit l'image à la bonne ligne et la bonne colonne.

```
#affichage des blocs selon la liste creee
def affichage(structure_niveau):
      for y in range(len(structure_niveau)):
            if niveau == "niveaux/niveaul.txt" and level == 0:
                  fenetre.blit(terre,(0,y*100+hauteur+400))
            else:
                  fenetre.blit(terre,(0,y*100+hauteur-400))
      for y in range(len(structure_niveau)):
            for x in range(len(structure_niveau[0])):
                  if structure_niveau[y][x]==00:
                        deplacement=True
                  elif structure_niveau[y][x]==1:
                        fenetre.blit(oxygene,(x*100,y*100+hauteur))
                  elif structure_niveau[y][x]==2:
                        fenetre.blit(tnt,(x*100,y*100+hauteur))
                  elif structure_niveau[y][x]==3:
                        fenetre.blit(rouge,(x*100,y*100+hauteur))
                  elif structure_niveau[y][x]==4:
                        fenetre.blit(bleu,(x*100,y*100+hauteur))
```

Pour détruire un bloc, on va détecter le bloc à côté du personnage et passer sa valeur à 0.
On répète ceci pour les 4 directions possibles (haut, bas, gauche, droite).
On analyse si le bloc à côté n'est pas de la vie, puis on passe le bloc à 0 pour qu'il disparaisse.
En fonction des blocs, on agit aussi sur d'autres variables :
Pour la TNT, il faut que le personnage lui mette 5 coups, on fait changer à chaque fois l'image dès qu'il agit sur le bloc jusqu'à la faire disparaître en la mettant à 0.
On soustrait aussi 20 à la variable de vie et 500 au score.
Pour la caisse brune avec un temps de latence, on va simplement faire un pygame.delay de 1sec.
Pour les blocs normaux, on va juste les passer à 0 et faire appel à la fonction "detruireAutour".

```
#destruction des blocs en appuyant sur espace
elif event.key == K_SPACE and not tombe:
      pygame.time.delay(50)
      sonBloc.play()
      if direction == 4:
            if structure_niveau[CasePersoY-1][CasePersoX] != 1:
                  if structure_niveau[CasePersoY-1][CasePersoX] == 2:
                        structure_niveau[CasePersoY-1][CasePersoX] = 9
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 9:
                        structure_niveau[CasePersoY-1][CasePersoX] = 10
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 10:
                        structure_niveau[CasePersoY-1][CasePersoX] = 11
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 11:
                        structure_niveau[CasePersoY-1][CasePersoX] = 12
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 12:
                        structure_niveau[CasePersoY-1][CasePersoX] = 0
                        vie-=20
                        score-=500
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 7:
                        structure_niveau[CasePersoY-1][CasePersoX] = 0
                        score += 5
                  elif structure_niveau[CasePersoY-1][CasePersoX] == 8:
                        pygame.time.delay(1000)
                        score += 5
                        structure_niveau[CasePersoY-1][CasePersoX] = 0
                  else:
                        couleur = structure_niveau[CasePersoY-1][CasePersoX]
                        structure_niveau[CasePersoY-1][CasePersoX] = 0
                        score += 10
                        monBlocX = CasePersoX
                        monBlocY = CasePersoY-1
                        structure_niveau, score = detruireAutour(structure_niveau,monBlocX,monBlocY,couleur,score)
```

Les fonctions detruireAutour et detecterAutour :

“detruireAutour” sert à détruire les blocs qui sont liés au premier bloc que le personnage détruit.

```
#destruction de tout les blocs d'une même couleur cote a cote
def detruireAutour(structure_niveau,monBlocX,monBlocY,couleur,score):
      if couleur != 0 and couleur !=1 and couleur !=2 :

            if structure_niveau[monBlocY-1][monBlocX] == couleur and monBlocY-1 != -1:
                  structure_niveau[monBlocY-1][monBlocX] = 0
                  score += 10
                  structure_niveau, score = detruireAutour(structure_niveau,monBlocX,monBlocY-1,couleur,score)
            if structure_niveau[monBlocY+1][monBlocX] == couleur and monBlocY+1 != 105:
                  structure_niveau[monBlocY+1][monBlocX] = 0
                  score += 10
                  structure_niveau, score = detruireAutour(structure_niveau,monBlocX,monBlocY+1,couleur,score)
```

On vérifie que ce soit bien des blocs et pas de l’air, de la vie ou de la TNT.
On vérifie la couleur et on fait en sorte qu’il analyse pas en dehors de la liste.
On ajoute 10 au score pour la destruction du bloc et on rappelle la même fonction pour analyser si il y a un bloc suivant à détruire.
Cette fonction va tourner en boucle jusqu’à qu’il n’y ait plus de bloc à détruire.

```
#detection du nombre de blocs fusionnes
def detecterAutour(structure_niveau,monBlocX,monBlocY,couleur,nbrBlocs):
      if couleur != 0 and couleur !=1 and couleur !=2 :

            if structure_niveau[monBlocY-1][monBlocX] == couleur and monBlocY-1 != -1:
                  structure_niveau[monBlocY-1][monBlocX] = 0
                  nbrBlocs += 1
                  nbrBlocs = detecterAutour(structure_niveau,monBlocX,monBlocY-1,couleur,nbrBlocs)
            if structure_niveau[monBlocY+1][monBlocX] == couleur and monBlocY+1 != 105:
                  structure_niveau[monBlocY+1][monBlocX] = 0
                  nbrBlocs += 1
                  nbrBlocs = detecterAutour(structure_niveau,monBlocX,monBlocY+1,couleur,nbrBlocs)
```

“détecterAutour” intervient quand les blocs chutent et doivent se détruire automatiquement sans l’intervention du personnage.
La fonction suit le même principe que la fonction précédente et va s'exécuter quand les blocs tombent.

```
#chute des blocs
if saveTempsCubes+0.7 < (time.time()-start):
      saveTempsCubes = time.time()-start

      for y in range(len(structure_niveau)-1,0,-1):
            for x in range(len(structure_niveau[0])-1,-1,-1):
                  if structure_niveau[y][x] == 0 and y != 104 and x != 7 and structure_niveau[y-1][x] == 1:
                        structure_niveau[y][x] = structure_niveau[y-1][x]
                        structure_niveau[y-1][x] = 0
                  if structure_niveau[y][x] == 0 and y != 104 and x != 7 and (not structure_niveau[y-1][x] == structure_niveau[y-1][x-1] or structu
                        structure_niveau[y][x] = structure_niveau[y-1][x]
                        structure_niveau[y-1][x] = 0
                  elif structure_niveau[y][x] == 0 and y != 104 and x != 7 and (structure_niveau[y-1][x] == structure_niveau[y-1][x-1] or structure
                        nbrBlocs = 1
                        copieStructure = deepcopy(structure_niveau)
                        nbrBlocs = detecterAutour(copieStructure,x,(y-1),(structure_niveau[y-1][x]),nbrBlocs)
                        if nbrBlocs >= 4:
                              couleur = structure_niveau[y-1][x]
                              structure_niveau[y-1][x] = 0
                              structure_niveau, score = detruireAutour(structure_niveau,x,(y-1),couleur,score)

            if structure_niveau[CasePersoY][CasePersoX] != 0 and structure_niveau[CasePersoY][CasePersoX] != 1:
                  vie -= 100
                  score -= 1000
                  for i in range(CasePersoY):
                        structure_niveau[CasePersoY-i][CasePersoX]=0
                        structure_niveau[CasePersoY-i][CasePersoX+1]=0
                        structure_niveau[CasePersoY-i][CasePersoX-1]=0
```

4. Explication de la fonction lecture :

La fonction lecture utilise la variable “niveau” et l’interprète en une nouvelle variable “structure niveau” qui elle est lisible par Pygame.
Tout d’abord, on ouvre le fichier texte qui contient le niveau et on crée une liste vide.

```python
#fonction de lecture des fichiers textes
def lecture(niveau):
    with open(niveau, "r") as fichier:
        structure_niveau = []
```

Ensuite, on parcourt les caractères du fichier et on les ajoute dans une liste à deux dimensions afin d'avoir le contenu du fichier texte dans la liste qui est plus facilement lisible en python.

```python
        for ligne in fichier:
            ligne_niveau = []
            #On parcourt les sprites (caracteres) contenus dans le fichier
            for sprite in ligne:
                #On ignore les "\n" de fin de ligne
                if sprite != '\n':
                    #On ajoute le sprite à la liste de la ligne
                    ligne_niveau.append(int(sprite))
            #On ajoute la ligne à la liste du niveau
            structure_niveau.append(ligne_niveau)
        return(structure_niveau)
```