1 Présentation de l'environnement PyGame

1.1 Intérêt d'utilisation de la bibliothèque PyGame

PyGame est une bibliothèque libre multi-plateforme qui facilite le développement de jeux vidéo en temps réel avec le langage de programmation Python, langage utilisé dans le cadre de ce projet. Cette bibliothèque permet principalement de développer une interface graphique par l'intermédiaire du langage Python.

FIGURE 1 – Logo PyGame

Il est intéressant de se demander pourquoi avoir choisi une telle bibliothèque pour le développement d'une interface graphique.

Celle-ci possède plusieurs points forts qui peuvent être énumérés ci-dessous :
- Une portabilité sur divers systèmes d'exploitation
- Facilité d'utilisation avec la multitude de fonctionnalités offerte par une telle libraire
- Construction sur la bibliothèque logicielle libre Simple DirectMedia Layer (SDL) qui supporte plusieurs OS
- Gestion de l'affichage vidéo, de périphériques communs avec le clavier et la souris, de l'audio numérique

1.2 Exemples d'utilisation

Pour se familiariser avec les fonctionnalités proposées par la bibliothèque PyGame, il peut être intéressant de pouvoir découvrir ce qu'il en est à travers différents exemples d'utilisation qui seront présentés à la suite.

Le programme 1 permet de construire une fenêtre d'affichage donnée en spécifiant ses dimensions (largeur,hauteur) exprimées en pixels. Après exécution du programme, une fenêtre d'affichage noire se lance et change de couleur après appui sur une touche de clavier par l'utilisateur, la couleur de la fenêtre d'affichage bascule du noir vers le blanc et vice-versa. Le code source et les résultats obtenus sont illustrés sur la page suivante.

```python

import pygame


noir = (0,0,0)         ## Code couleur RVB associé au Noir
blanc = (255,255,255)  ## Code couleur RVB associé au Blanc

pygame.init()

## Création d'une fenêtre de dimension 640 * 480 pixels

fenetre = pygame.display.set_mode((640,480))

execution = True

while execution:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:   ## Evenement : Fermer la fenêtre d'affichage après execution du programme
            execution = False
        if event.type == pygame.KEYDOWN: ## Evenement : Appuyer sur une touche du clavier
            noir,blanc = blanc,noir      ## Permutation des couleurs après l'évenement pygame.KEYDOWN

            ## La fenêtre d'affichage change de couleur dès lorsque l'on appuie sur une touche du clavier
            ## en passant du noir au blanc

    fenetre.fill(noir)     ## Fenêtre d'affichage noire après execution de ce programme
    pygame.display.flip()
```

FIGURE 2 – Code source du programme 1

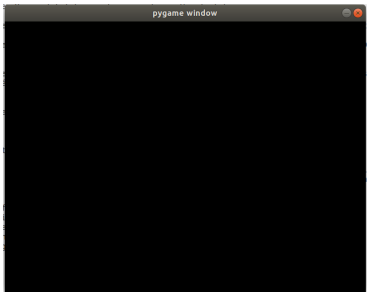
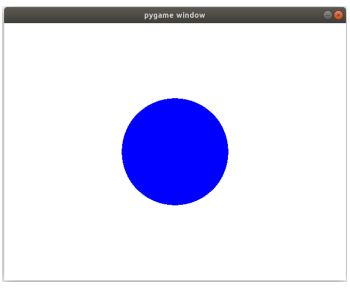


FIGURE 3 – Fenêtre d'affichage avant l'appui sur une touche de clavier

FIGURE 4 – Fenêtre d'affichage après l'appui sur une touche de clavier

Intéressons nous à présent à la construction de figures géométriques, la bibliothèque PyGame compte une multitude de fonctions permettant de réaliser des figures géométriques à partir d'une fenêtre d'affichage. L'exemple ci-dessous concerne la construction d'un cercle avec un rayon spécifié, le code source correspondant se base en partie de celui du premier programme.

```python

import pygame


blanc = (255,255,255)         ## Code couleur RVB associé au Blanc
bleu = (0,0,255)              ## Code couleur RVB associé au Bleu

pygame.init()

## Création d'une fenêtre de dimension 640 * 480 pixels

fenetre = pygame.display.set_mode((640,480))

execution = True

while execution:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:   ## Evenement : Fermer la fenêtre d'affichage après execution du programme
            execution = False
    fenetre.fill(blanc)   ## Mise à jour de la fenêtre d'affichage

    pygame.draw.circle(fenetre,bleu,(320,240),100) ## Dessin d'une figure géométrique (Cercle de rayon 100 px)
                                                   ## Position géométrique du cercle dessiné en pixels (320 px ; 240 px)

    pygame.display.flip()                          ## Mise à jour de la fenêtre d'affichage
```



Le troisième et dernier exemple d'utilisation va concerner la création de boutons. Cet exemple consiste à créer une suite de boutons ayant une forme géométrique donnée, après un clic sur un des boutons une animation est déclenchée (disparition du bouton / changement de couleur d'un bouton).

```python
import pygame

pygame.init()
clock = pygame.time.Clock()

bleu_ciel = (0, 200, 255)
blanc = (255,255,255)
vert = (0,255,0)
rouge = (255,0,0)


## Création d'une fenêtre de dimension 640 * 480 pixels

fenetre = pygame.display.set_mode((640,480))

execution = True
while execution:
    background = pygame.Surface(fenetre.get_size())
    background.fill(bleu_ciel)
    # Ajout du fond dans la fenêtre d'affichage
    fenetre.blit(background,(0,0))

    # Dessin d'un rectangle de contour blanc
    # (x = 75 ; y = 10) coordonnées de position du rectangle blanc par rapport au coin supérieur gauche de la fenêtre
    # Largeur = 100 pixels  Hauteur = 50 pixels
    rectangle_blanc = pygame.draw.rect(fenetre,blanc,[75,10,100,50])

    # Dessin d'un rectangle vert
    rectangle_vert = pygame.draw.rect(fenetre,vert,[250,10,100,50])

    # Position du curseur par rapport à la fenêtre d'affichage
    # Retourne 1 si le curseur est au-dessus d'un rectangle
    mouse_xy = pygame.mouse.get_pos()
    over_blanc = rectangle_blanc.collidepoint(mouse_xy)
    over_vert = rectangle_vert.collidepoint(mouse_xy)

    for event in pygame.event.get():
        if event.type == pygame.QUIT: # si fermeture de la fenêtre avec un clic de la souris
            execution = False
        # Si clic gauche de la souris sur le rectangle vert, celui-ci devient de couleur rouge
        elif event.type == pygame.MOUSEBUTTONDOWN and over_vert:
            vert = rouge

        # Si clic gauche de la souris sur le rectangle blanc, celui-ci disparaît
        elif event.type == pygame.MOUSEBUTTONDOWN and over_blanc:
            blanc = bleu_ciel

    # Actualisation de l'affichage
    pygame.display.flip()
    # 10 images par seconde
    clock.tick(10)
```

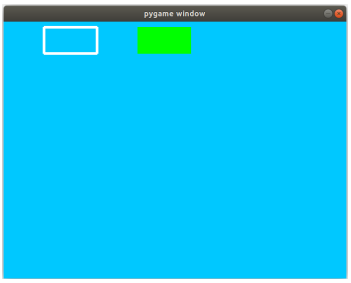


FIGURE 5 – Fenêtre d'affichage

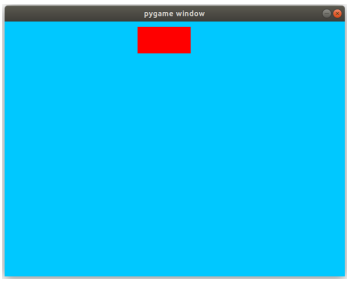


FIGURE 6 – Animations après clic

La bibliothèque PyGame propose davantage de fonctionnalités, de nombreux tutoriels sont présents afin de se familiariser avec cet environnement. La documentation de PyGame est consultable via le lien ci-dessous :
`https://www.pygame.org/docs/`

1.3 Interface graphique proposée pour le Puissance 4

Il est à présent important de se focaliser sur le projet concerné, il sera question ici de présenter et d'expliquer le développement de l'interface graphique proposée pour le jeu du Puissance 4.
Premièrement, il a été nécessaire de définir correctement l'environnement de jeu. Le code ci-dessous décrit la définition des paramètres géométriques sur l'environnement concerné. Il a été question de fixer une taille pour la grille de jeu adaptée pour une dimension 6 * 7, cette taille a été choisie de manière à ce que cela soit adaptée avec la dimension souhaitée.
Il a fallu de plus, définir le rayon du cercle autrement dit l'emplacement nécessaire à l'insertion des jetons. Cela a été défini en tenant compte de la propriété géométrique d'une case (rectangle avec un cercle inscrit).
Pour rendre un tel environnement jouable, il a fallu ajouter à cela des jetons de jeu (jaune et rouge), des pions utilisées pour des fonctions prochaines.
La fenêtre de jeu doit être adaptée avec le dimensionnement de la grille, d'où la définition de largeur et de hauteur sur le code ci-dessous.

```
243  ####################  INTERFACE GRAPHIQUE  ####################
244
245  taille_case = 100                    ## Taille en pixels d'une case de la grille de jeu
246  rayon_cercle = int(taille_case/2 - 4) ## Rayon du cercle
247  cadre = (0, 0, 255)                  ## Cadre bleue de la grille de jeu
248  blanc = (255, 255, 255)              ## Case initialement non remplie
249  jeton_rouge = (255, 0, 0)            ## Jeton utilisé par l'utilisateur
250  jeton_jaune = (255, 255, 0)          ## Jeton utilisé par l'IA
251  pion_joueur = 2                      ## pion utilisé par l'utilisateur
252  pion_IA = 1                          ## pion utilisé par l'IA
253
254  humain = 0                           ## Variables 'humain' et 'IA' utilisées pour définir les tours de jeu
255  IA = 1
256
257
258  largeur = len(etat[0]) * taille_case         ## Dimensions de la grille de jeu
259  hauteur = (len(etat) + 1) * taille_case
```

FIGURE 7 – Spécifications Géométriques

Une fois avoir présenté l'environnement de jeu, il est maintenant important d'aborder les fonctions nécessaires à l'évolution du jeu.
La première fonction*inserer jeton* prend comme paramètres :

- Un état du jeu qui est la grille
- i étant l'indice de ligne de la case utilisée pour l'insertion d'un jeton
- j étant l'indice de colonne de la case pour laquelle on veut insérer un jeton

Celle-ci effectue au cours du jeu les insertions de jeton pour l'humain et l'IA, on retrouvera cette fonction dans la boucle de jeu qui sera abordée plus tard.
La fonction suivante *emplacementValide* prend comme paramètres l'état de jeu et l'indice de la colonne de la case souhaitée pour effectuer l'insertion d'un jeton de jeu. L'emplacement est valide lorsqu'un jeton occupe la postion la plus basse possible sur une colonne donnée. Cette fonction renvoie un booléen pour indiquer si l'insertion est permise ou non.

La troisième et dernière fonction de jeu *rangeeSuivante* prend les mêmes paramètres que la fonction précédente. En revanche, celle-ci renvoie l'indice le plus grand de la ligne d'un état de jeu pas encore complétée de jetons de jeu. Exemple, si la dernière rangée est complétée de jetons de jeu il suffira de rechercher l'indice le plus grand de la rangée non encore complétée, étant donné qu'un jeton à insérer doit occuper la position la plus basse possible sur une colonne précise de la grille de jeu.

```
264  def inserer_jeton(etat,i,j,pion):  ## Fonction permettant l'insertion d'un jeton de joueur (Utilisateur / IA)
265      etat[i][j] = pion              ## etat : grille de jeu , i : abscisse du jeton inséré
266                                     ## j : ordonnée du jeton inséré , pion : 1 ou 2
267
268
269  def emplacementValide(etat,colonne):     ## Fonction vérifiant  si une case de la grille de jeu est inoccupée avant l'inserion d'un jeton de joueur
270      if etat[len(etat)-1][colonne] == 0:  ## Le jeton inséré doit occuper la position la plus basse sur une colonne (gravité)
271          return True                      ## colonne : ordonnée de la case donnée
272      else:
273          return False
274
275
276  def rangeeSuivante(etat, colonne):   ## Fonction recherchant une ligne non occupée de la grille de jeu
277      for ligne in range(len(etat)):
278          if etat[ligne][colonne] == 0:
279              return ligne
```

FIGURE 8 – Fonctions de jeu

La fonction suivante est l'une des fonctions principales pour le développpement de l'interface graphique. Celle-ci va s'occuper de l'affichage de la grille de jeu, la grille de jeu se construit de manière itérative. En parcourant toute la grille qu'on peut assimiler à une matrice, les cases se construisent progressivement. Chaque case est représentée par un rectangle avec des dimensions choisies (largeur,hauteur) et des coordonées de position (x ;y) par rapport à la grille de jeu. Chaque rectangle contient un cercle ayant un rayon défini et aussi des coordonées de position (x,y) par rapport à la grille de jeu.
Les dimensions proposées dans le code ci-dessous ont été choisies de manière à ce que cela s'adapte au mieux aux dimensions de la grille de jeu.
Elle se construit ainsi progressivement dès lorsque des insertions de jetons de jeu ont été effectuées, cette fonction fera l'objet de plusieurs appels car à chaque insertion de jetons il est nécessaire de mettre à jour l'affichage de la grille de jeu.

```
284  def grilleApparition(etat):
285      for ligne in range(len(etat)):
286          for colonne in range(len(etat[0])):
287              ## Construction des emplacements pour l'insertion des jetons
288              pygame.draw.rect(grille, cadre, (colonne*taille_case, (ligne+1)* taille_case, taille_case, taille_case))
289              pygame.draw.circle(grille, blanc, (int(colonne*taille_case + taille_case/2), int((ligne+1)*taille_case + taille_case/2)), rayon_cercle)
290
291      for ligne in range(len(etat)):
292          for colonne in range(len(etat[0])):
293              # Si c'est au tour de l'humain d'insérer un jeton, création d'un jeton rouge à insérer dans la case souhaitée
294              if etat[ligne][colonne] == pion_joueur:
295                  pygame.draw.circle(grille, jeton_rouge, (int(colonne*taille_case + taille_case/2), hauteur-int((ligne)*taille_case + taille_case/2)), rayon_cercle)
296              # Si c'est au tour de l'IA d'insérer un jeton, création d'un jeton jaune à insérer dans la case souhaitée
297              elif etat[ligne][colonne] == pion_IA:
298                  pygame.draw.circle(grille, jeton_jaune, (int(colonne*taille_case + taille_case/2), hauteur-int((ligne)*taille_case + taille_case/2)), rayon_cercle)
299      # Mise à jour de la grille de jeu
300      pygame.display.update()
```

FIGURE 9 – Création de la grille de jeu

La dernière étape ayant permis la réalisation de l'interface graphique est la fonction jeu(). Cette fonction peut se décomposer en plusieurs phases :

— Gestion des tours de jeu
— Insertion d'un jeton de jeu dans le cas où c'est au tour de l'utilisateur de débuter la partie
— Insertion d'un jeton de jeu dans le cas où c'est au tour de l'IA de commencer
— Détermination de l'état final du jeu (Victoire de l'utilisateur / Victoire de l'IA / Match nul)

Les différentes phases sont décrites par les deux illustrations qui suivent.

```
311  def jeu():
312      etat = [[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0]] ## Grille de jeu 6 * 7
313      fin_partie = False
314
315
316      myfont = pygame.font.SysFont("comicsansms", 40) ## Police d'écriture utilisée pour l'affichage du gagnant de la partie / égalité
317
318      tour = random.randint(humain,IA)    ## Gestion des tours de jeu
319
320
321      while not fin_partie and len(possibles(etat)) != 0:
322
323          for event in pygame.event.get():
324              ## Si clic sur la croix de la fenêtre d'affichage, celle-ci se ferme
325              if event.type == pygame.QUIT:
326                  sys.exit()
327
328              if event.type == pygame.MOUSEMOTION: ## Mouvement du curseur
329                  pygame.draw.rect(grille, blanc, (0,0, largeur, taille_case)) ## Bande horizontale blanche au-dessus de la grille de jeu
330                  posx = pygame.mouse.get_pos()[0]  ## Récupération des coordonnées de position (x;y) du curseur de la souris
331                  if tour == humain:  ## Si c'est au tour de l'humain de jouer, celui-ci va devoir insérer un jeton rouge dans la case désirée
332                      pygame.draw.circle(grille, jeton_rouge, (posx, int(taille_case/2)), rayon_cercle)
333                  else:  ## Jeton jaune dans le cas où c'est à l'IA de commencer la partie
334                      pygame.draw.circle(grille, jeton_jaune, (posx, int(taille_case/2)), rayon_cercle)
335
336
337              pygame.display.update()  ## Actualisation de la fenêtre d'affichage
338
339
340              if event.type == pygame.MOUSEBUTTONDOWN:  ## Evenement : clic gauche de la souris sur un emplacement inoccupée de la grille de jeu
341                  pygame.draw.rect(grille, blanc, (0,0, largeur,taille_case))
342
343                  if tour == humain:
344                      posx = pygame.mouse.get_pos()[0]   ## Position en abscisse du curseur par rapport à la fenêtre d'affichage
345                      col = posx // taille_case          ## Position en ordonnée du curseur par rapport à la fenêtre d'affichage
346
347                      if emplacementValide(etat, col):
348                          ligne = rangeeSuivante(etat, col)  ## Insertion du jeton de l'utilisateur dans le cas où la situation est favorable
349                          inserer_jeton(etat, ligne, col, 2)
350
351                          if estGagnant(etat,3,"H"):     ## Si l'utilisateur remporte la partie, un message s'affiche en haut de la grille
352                              label = myfont.render("Victoire de l'humain !", 1, jeton_rouge)
353                              grille.blit(label, (40,10))
354                              fin_partie = True
355
356                          tour += 1         ## Incrémentation des tours pour permettre aux joueurs de jouer l'un après l'autre
357                          tour = tour % 2
358
359                          ## Mise à jour de la grille de jeu après plusieurs insertions de jetons
360                          grilleApparition(etat)
```

FIGURE 10 – Partie 1 de la fonction jeu()

```
362                 if tour == IA and not fin_partie:
363                     longueur = len(possibles(etat))
364                     coup = possibles(etat)[longueur // 2] ## Jeton placé sur la 4e colonne de la grille
365                     colonne = coup[1]
366 
367                     if emplacementValide(etat, colonne):
368                         ligne = rangeeSuivante(etat, colonne)
369                         inserer_jeton(etat, ligne, colonne, 1)
370 
371                         if estGagnant(etat, 4, "IA"):
372                             ## Si l'IA remporte la partie, un message s'affiche en haut de la grille
373                             label = myfont.render("Victoire de l'IA !", 1, jeton_jaune)
374                             grille.blit(label, (40, 10))
375                             fin_partie = True
376 
377 
378 
379 
380                         tour += 1
381                         tour = tour % 2
382 
383                         grilleApparition(etat)
384 
385                 if(possibles(etat) == [] and not (estGagnant(etat, 4, "IA") == True) and not (estGagnant(etat, 4, "H") == True)):
386                     ## Message affiché sur la bande horizontale blanche, dans le cas d'une saturation de la grille
387                     label = myfont.render("MATCH NUL", 1, (0,0,255))
388                     grille.blit(label, (40, 10))
389                     pygame.time.wait(3000)
390 
391                 if fin_partie:
392                     pygame.time.wait(3000)
393                     ## La fenêtre d'affichage se ferme 3000 ms après que la partie soit terminée
394 
395 
396 jeu()
```

FIGURE 11 – Partie 2 de la fonction jeu()

1.4 Tests effectués sur une grille traditionnelle 6 * 7

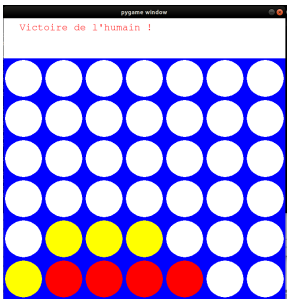


FIGURE 12 – Victoire de l'utilisateur

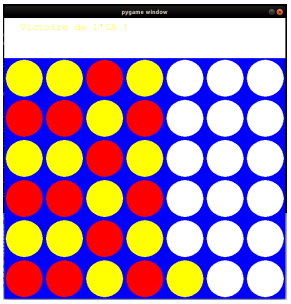


FIGURE 13 – Victoire de l'IA

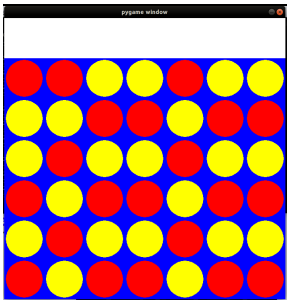


FIGURE 14 – Saturation de la grille