Soutenance

- Geoffray Lecathelinais, Antoine Jourdan, Kilian Caillot, Paul Gosse
- Licence 1 Informatique - 3A
- Projet choisi: Tetris
- Premières idées:
  - Récupération d’un tetris primaire: Tetromino
  - ajout de mode:
    - 1 contre 1
    - malus
    - fantôme

Éléments techniques

Algorithme:

Permettant de vérifier en continue si quelque chose entre en collision avec la pièce actuellement en chute.

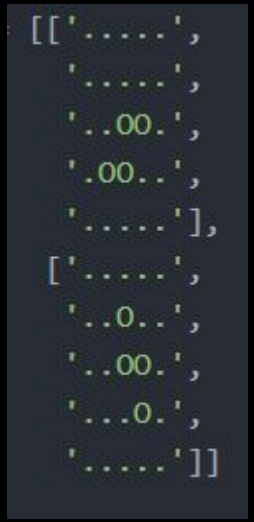
```
def isValidPosition(board, piece, adjX=0, adjY=0):
    # Retourne si la pièce est à l'intérieur du terrain et si elle n'est pas rentré en collision
    for x in range(TEMPLATEWIDTH):
        for y in range(TEMPLATEHEIGHT):
            isAboveBoard = y + piece['y'] + adjY < 0
            if isAboveBoard or PIECES[piece['shape']][piece['rotation']][y][x] == BLANK:
                continue
            if not isOnBoard(x + piece['x'] + adjX, y + piece['y'] + adjY):
                return False
            if board[x + piece['x'] + adjX][y + piece['y'] + adjY] != BLANK:
                return False
    return True
```

Exemple d’utilisation de cet algorithme:

```
# Vérifie si le mouvement effectué par le joueur est possible ou non, ici pour
if (event.key == K_LEFT) and isValidPosition(board, fallingPiece, adjX=-1):
    fallingPiece['x'] -= 1
    movingLeft = True # Active l'évènement correspondant
    movingRight = False # Désactive les autres évènements
    lastMoveSidewaysTime = time.time()
```

Éléments techniques

Structure de données:



```
[['.....',
  '.....',
  '..OO.',
  '.OO..',
  '.....'],
 ['.....',
  '..O..',
  '..OO.',
  '...O.',
  '.....']]
```

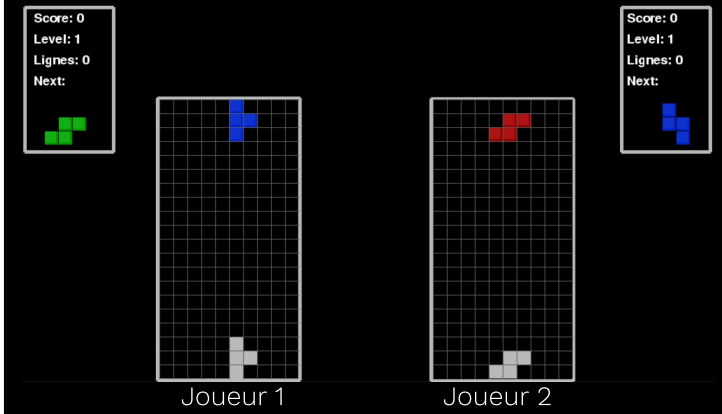
```
PIECES = {'S': S_SHAPE_TEMPLATE,
          'Z': Z_SHAPE_TEMPLATE,
          'J': J_SHAPE_TEMPLATE,
          'L': L_SHAPE_TEMPLATE,
          'I': I_SHAPE_TEMPLATE,
          'O': O_SHAPE_TEMPLATE,
          'T': T_SHAPE_TEMPLATE,

          'ESCALIER':ESCALIER_SHAPE_TEMPLATE,
          'COIN'   :COIN_SHAPE_TEMPLATE,

          "G_GAMEOVER": G_GAMEOVER,
          "A_GAMEOVER": A_GAMEOVER,
          "M_GAMEOVER": M_GAMEOVER,
          "E_GAMEOVER": E_GAMEOVER,
          "O_GAMEOVER": O_GAMEOVER,
          "V_GAMEOVER": V_GAMEOVER,
          "R_GAMEOVER": R_GAMEOVER
          }
```

Fonctionnalités implémentées

Mode 1 contre 1:



Règles :

- Règle Tetris
- Victoire au score
- Malus si plusieurs lignes d'un coup (si activé)

```
def convertToPixelCoords(boxx, boxy,is_right):
    # Transforme les coordonnées données par rapport au  board en coordonnées xy par rapport à la fenêtre
    # Coordonnées de l'emplacement sur l'écran
    if is_right:
        return (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))
    else:
        return (XMARGIN2 + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))

def drawBox(boxx, boxy, color,is_right, pixelx=None, pixely=None):
    # Affiche une box
    # Au coordonnées xy sur le board
    # Si pixelx et pixely sont spécifiés, dessine en fonction des coordonnées stockées dans pixelx et pixely
    # (Utilisée pour la prochaine pièce "Next piece")
    if color == BLANK:
        return
    if pixelx == None and pixely == None:
        pixelx, pixely = convertToPixelCoords(boxx, boxy,is_right)
    pygame.draw.rect(DISPLAYSURF, COLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 1, BOXSIZE - 1))
    pygame.draw.rect(DISPLAYSURF, LIGHTCOLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 4, BOXSIZE - 4))
```

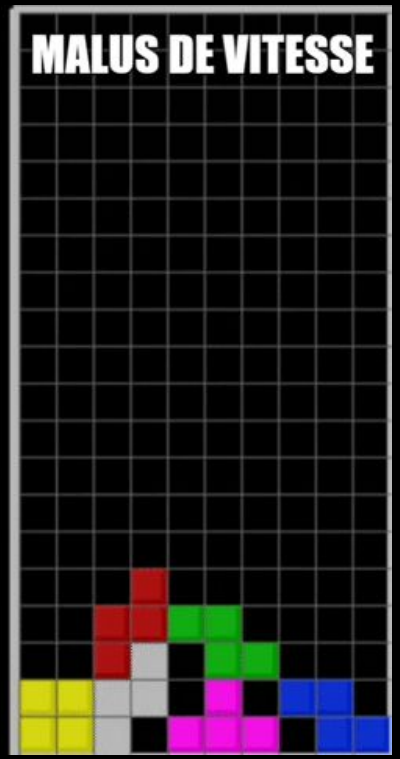
is_right : True → Joueur 1
False → Joueur 2

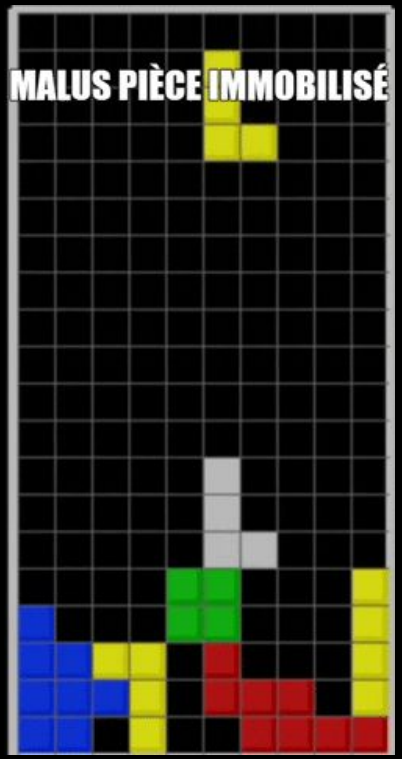
Sert à sélectionner le joueur

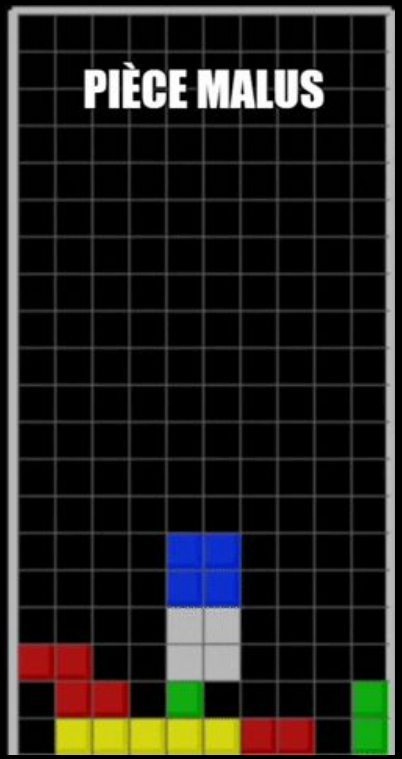
```
for x in range(BOARDWIDTH):
    for y in range(BOARDHEIGHT):
        drawBox(x, y, board[x][y],True)

for x in range(BOARDWIDTH):
    for y in range(BOARDHEIGHT):
        drawBox(x, y, board2[x][y],False)
```

Fonctionnalités implémentées



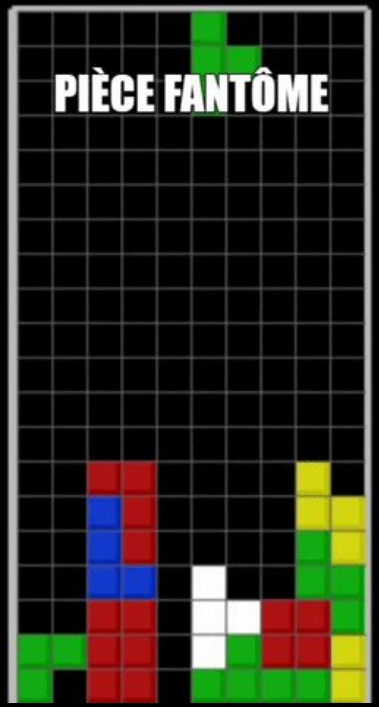




```
if malusOn:
    if ligneCompleted==2: # active un malus qui
        fallFreq2=fallFreq2/4
        fallingPiece2["color"]=4
    if ligneCompleted==3:
        if fallingPiece2["shape"]!="O":
            fallingPiece2["color"]=5
            malusJoueur2=1 #active un malus empê
        else:
            fallFreq2=fallFreq2/5
            fallingPiece2["color"]=4
    elif ligneCompleted==4: # active un malus cr
        nextPiece2=getNewPiece(is_normal=False)
```

```
elif (event.key == K_UP):
    if not(malusOn and malusJoueur1==1):
```

Fonctionnalités implémentées




```
def getFantomPiece(piece,board):
    fantomPiece={"shape":piece["shape"],
                 "rotation":piece["rotation"],
                 "x":piece["x"],
                 "y":piece["y"],
                 "color":len(COLORS)-1}
    for i in range(1, BOARDHEIGHT):
        if not isValidPosition(board, fantomPiece, adjY=i):
            break
    fantomPiece["y"]+=i-1
    return(fantomPiece)
```

Fonctionnalités implémentées

Niveaux / Difficulté progressive:

n= niveau actuel
s= score réalisé pour atteindre le niveau actuel

ns= niveau suivant
ns= s+500*n

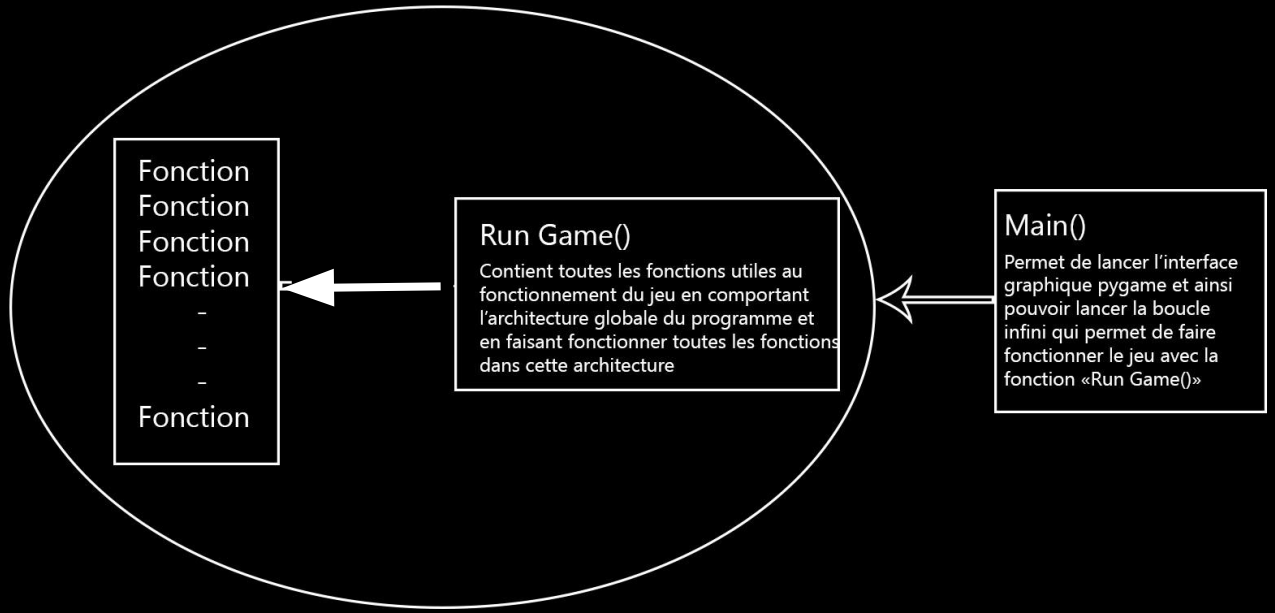
```
LEVELS=[]
scoreLevel=0
for i in range(30):
    scoreLevel+=500*i
    LEVELS.append(scoreLevel)
```

```
def calculateLevelAndFallFreq(score,level):
    # Fréquence de chute calculée selon le niveau du joueur qui lui même est calculé selon son score

    while score > LEVELS[level]:
        level+=1
    fallFreq = 0.50 *(0.9**level)
    return level, fallFreq
```

Architecture

Structuration et traitement des fonctions:



Conclusion

Problèmes rencontrés

- Problème réseau
- Gestion de la défaite d'un des joueurs
- Erreur lors d'une certaine manoeuvre

Améliorations possibles

- Jeu en réseau
- Optimiser le code avec une boucle for pour le mode 1 contre 1
- Statistique d'apparition des différentes pièces pour une plus grande équité
- Ajout d'une IA
- Stockage d'une pièce