Rapport
Tetris

Geoffray LECATHELINAIS
Antoine JOURDAN
Kilian CAILLOT
Paul GOSSE

Avril 2019

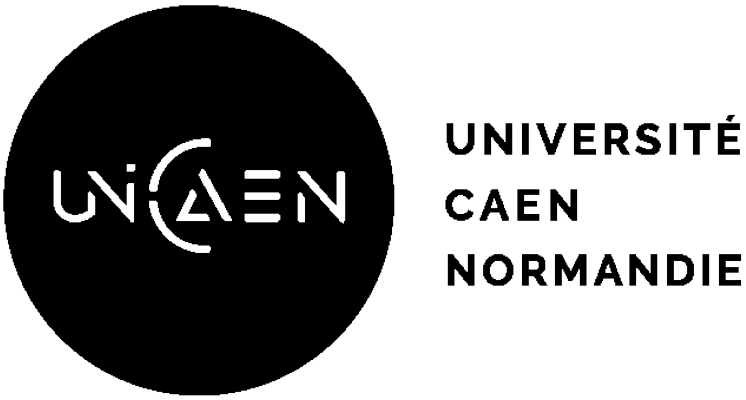


Table des matières

1 Introduction

1.1 Description du projet

Nous avons choisi de réaliser le jeu tetris. Le jeu tetris est un jeu qui a été inventé en 1984, le but du jeu est de réaliser des lignes complètes sans aucun trou à partir de formes géométriques prédéfinies qui nous sont distribuées à chaque fois que l'on utilise une des pièces. Avec des bonus si l'on réalise plusieurs lignes d'un seul coup.

1.2 Premières idées

Lors du commencement de notre projet nous ne savions pas si l'on allait rester proche de l'idée de base que se font les personnes d'un tetris ou si l'on devait plutôt s'orienter vers des fonctionnalités étrangères au tetris. Finalement nous avons décidé d'utiliser comme base un tetris très primaire du point de vu de ses fonctionnalités et d'y ajouter les objectifs requis tels qu'un mode un contre un avec des malus, un système de visualisation de la pièce (mode fantôme).

2 Fonctionnalités implémentées

2.1 Description des fonctionnalités de notre jeu

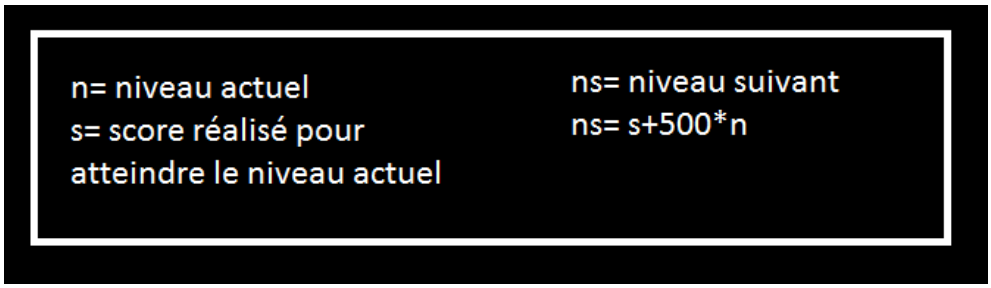
Lors du lancement de notre programme, un menu principal est mis à disposition du joueur ou des joueurs. Dans ce menu il est proposé un mode solo ou un mode duo. Ensuite une fois ce choix réalisé, les touches sont expliquées et puis il peut se passer deux choses différentes:

- S'il n'y a qu'un seul joueur il lui sera directement proposé s'il veut ou non jouer avec les pièces fantômes; une pièce fantôme est la projection de la pièce qui est en train de chuter à l'endroit où elle est supposée atterrir si elle continue dans la même direction et avec la même position.

- S'il y a deux joueurs alors ils leur sera possible de choisir s'ils veulent jouer avec les malus; les malus s'activent lorsque l'un des deux joueurs détruit plusieurs lignes d'un seul coup. Les différents malus sont:

- Deux lignes: accélère la vitesse de chute sur la partie de l'autre joueur.
- Trois lignes: empêchera la rotation de la pièce.
- Quatre lignes: fera apparaître en pièce suivante une pièce particulière que l'on a décidé de nommer "L'escalier".

Pendant la partie, le joueur peut observer sur un tableau d'affichage l'avancée de la partie de façon chiffrée; son score, son niveau et son nombre de lignes détruites. Son niveau augmente en fonction de son score de telle façon:
Le passage au deuxième niveau se réalise à un score de 500, le troisième à 1500 (500+500*2) , le quatrième à 3000 (1500+500*3), et le énième niveau au score du niveau actuel additionné à 500 multiplié par le niveau actuel.



Le score augmente de cette façon:

- Si une ligne est complétée alors le score est égale à 50* le niveau actuel.
- Deux lignes d'un coup: 150* le niveau actuel.
- Trois lignes d'un coup: 350* le niveau actuel.
- Quatre lignes d'un coup: 1000* le niveau actuel.

De plus à chaque fois qu'une pièce est posée le score augmente de 10* le niveau actuel.La partie est perdue si le joueur se laisse submerger par les pièces et que celles-ci finissent par sortir du cadre.

En mode duo lorsqu'un joueur a perdu son cadre se remplit de pièce qui forme un "game over" et l'autre joueur peut continuer à jouer jusqu'au moment où il perdra à son tour et la même chose lui arrivera. Et enfin un message global qui prendra tout l'écran divulguera qui a gagner la partie.

2.2 Organisation

Après la décision du projet que nous avons faites, nous nous sommes réparti les tâches en fonction des points forts de chacun pour que le projet soit établi au mieux. Ainsi, nous avons divisé les tâches de la manière suivante:

- programmation du mode 1vs1.
- programmation du mode solo.
- programmation du mode fantôme et des malus.
- design de l'identité graphique.

Puis à chaque séance nous avancions dans nos tâches respectives tout en implémentant ces fonctions au coeur du programme principal.

3 Éléments techniques

3.1 Algorithmes

Les principaux algorithmes pour la réalisation de ce projet ont consisté à l'évolution des pièces dans l'espace. C'est à dire qu'il fallait que le programme soit capable de savoir si la pièce chute normalement ou si quelque chose comme une précédente pièce par exemple était rentrée en collision avec la pièce actuellement en chute pour que celle-ci ne la traverse pas, de même pour qu'elle ne sorte pas du cadre de la partie.

De ce fait la création d'une fonction qui retournait si la pièce était encore à l'intérieur du terrain et si elle n'était pas entrée en collision était importante.

```
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
```

Cette fonction prend donc naturellement en variables le board, la pièce en question , “adjX” et “adjY”, alors adjX et adjY permettent de savoir l’évolution de la pièce, en utilisant les mêmes coordonnées que l’on rencontre dans un plan; c’est à dire x=abscisse et y=ordonnée. Si leur valeur est égal à 0 alors cela est traduit dans le programme par une immobilisation de la pièce de façon horizontale ou verticale car cela correspond à la position actuelle de la pièce et si c’est égal à 1 alors la pièce chute ou se décale d’une place (appelée une box dans le programme).

Cette fonction permet donc de savoir si un mouvement voulu par l’utilisateur est possible ou non, par exemple s’il veut bouger sa pièce horizontalement, nous allons tester la fonction “isValidPosition” avec adjX=-1 pour savoir si un mouvement d’une box vers la gauche est possible. Si cela est possible alors la fonction “isValidPosition” va retourner “True” et activer l’event correspondant en faisant chuter la pièce d’une box vers la gauche avec le “fallingPiece[‘x’] - =1.

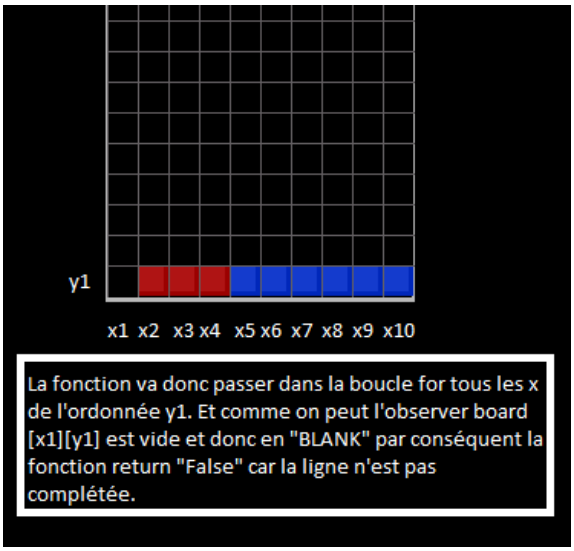
```
# Vérifie si le mouvement effectué par le joueur est possible ou non, ici pour
if (event.key == K_LEFT) and isValidPosition(board, fallingPiece, adjX=-1):
    fallingPiece['x'] -= 1
    movingLeft = True # Active l'évènement correspondant
    movingRight = False # Désactive les autres évènements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

De façon générale la fonction continue de vérifier s’il y a un obstacle ou non en prenant en compte la position de la pièce actuelle avec le “piece[‘x’] et [‘y’]” en faisant “continue” si la box désirée est libre donc en “BLANK” et elle continue de tourner avec les “x” et “y” qui évoluent selon le “TEMPLATEWIDTH” jusqu’à un “False” car la pièce chute sans l’intervention de l’utilisateur (sauf s’il veut accélérer la chute ou bouger horizontalement) jusqu’au moment où il y a un obstacle et dans ce cas l’un des deux autres “if” retourne false selon la situation est sort de la fonction pour se relancer avec la pièce suivante.

Un algorithme permet au programme de savoir si une ligne est complétée sans trou et s'il y'en a plusieurs. Pour cela une fonction permet de vérifier s'il y a encore des trous aux coordonnées x et y.

```
def isCompleteLine(board, y):
    # Retourne True si la ligne a été com
    for x in range(BOARDWIDTH):
        if board[x][y] == BLANK:
            return False
    return True
```

Cette fonction va donc tester pour chaque y en fonction d'un x si cette box est vide d'où le "board[x][y]" qui permet cela, et pour chacune de ces coordonnées la fonction va vérifier si elle est égale à "BLANK" qui traduit une box vide. Par conséquent s'il y a un des x pour un même y qui est "BLANK" alors la fonction return "False".



3.2 Structures de données

Pour structurer les données, nous avons principalement utilisé des listes et des dictionnaires. Comme par exemple pour dessiner nos pièces c'est une liste qui contient que des points et des 0 où les 0 permettent de dessiner la forme de la pièces et pour avoir les rotations d'une même pièce regroupées au même endroit nous avons utilisé des listes de liste.

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

Et ensuite un dictionnaire permet de regrouper toutes ces pièces au même endroit.

```
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
          }
```

De plus nous avons rédigé le “game over” dans le cadre à la fin de la partie en plaçant les pièces dédiées au “game over’ directement avec les coordonnées qu’elles doivent occuper.

```
G = {'shape': "G_GAMEOVER",
                'rotation': 0,
                'x': 0,
                'y': 0,
                'color': 2}
A = {'shape': "A_GAMEOVER",
                'rotation': 0,
                'x': 0,
                'y': 5,
                'color': 2}
M = {'shape': "M_GAMEOVER",
                'rotation': 0,
                'x': 0,
                'y': 10,
                'color': 2}
```

Et ensuite les pièces du “game over” avec leur position et leur couleur ont été regroupées dans une même liste.

Et en ce qui concerne le programme lui même, il est divisé en plusieurs fichiers pour le mode solo et duo et un autre qui permet de lancer le mode solo ou duo selon le choix de l’utilisateur.

3.3 Librairies

Pour ce qui est des librairies nous en utilisons deux:

- Pygame
- Python
- Time
- Sys

Sans la librairie pygame la réalisation de ce logiciel n’aurait pas été possible car elle est à la base de celui-ci, elle permet de réaliser l’interface utilisateur.

Et les fonctions qu'elle met à notre disposition permettent d'initialiser le jeu et à l'utilisateur d'être actif selon ses actions qu'il va réaliser grâce à son clavier et qui vont être récupérer par pygame.

La librairie random permet de choisir aléatoirement une pièce dans la liste ou celles-ci se trouvent pour qu'elle soit par la suite générer sur le board.

La librairie sys permet un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier.

La librairie time permet d'exprimer le temps.

4 Architecture

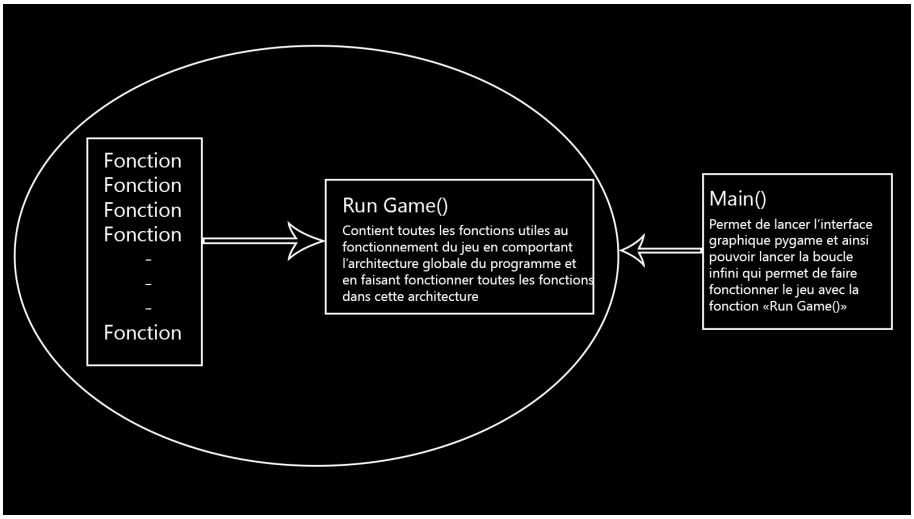
4.1 Structuration des fonctions

L'architecture de notre projet est plutôt simple mais très étalée car nous avons choisi de ne pas utiliser de class qui aurait pu regrouper des fonctions qui ont un même thème et ainsi elles auraient pu partager des variables globales seulement entre elles. Mais nous avions pas trouvé de réels thèmes en commun car finalement dans un tétris les fonctions partagent presque toutes les mêmes variables principales.

Par conséquent nous avons des variables globales à toutes les fonctions telles les couleurs, les pièces, les borders...

4.2 Traitement

Chaque fonction créée va être appelée dans notre fonction "runGame()" qui va diriger ces fonctions et donc la façon dont elles doivent être utilisées les unes par rapport aux autres avec tous ceux qui est ajoutés pour le fonctionnement du jeu. Puis la fonction "main()" qui permet de lancer la fonction "runGame()".



5 Applications et usages

5.1 Images

Pour l'identité graphique, toutes les images utilisées sont des créations faites par les membres du groupe avec le logiciel PHOTOSHOP. Une importation de police externe fût nécessaire pour la conception du logo du jeu et du menu.

Ce programme est composé de 6 images représentant l'explication des commandes où bien celle des différents modes tels que les malus et les pièces fantômes.

Image du menu avec deux polices d'écriture importées

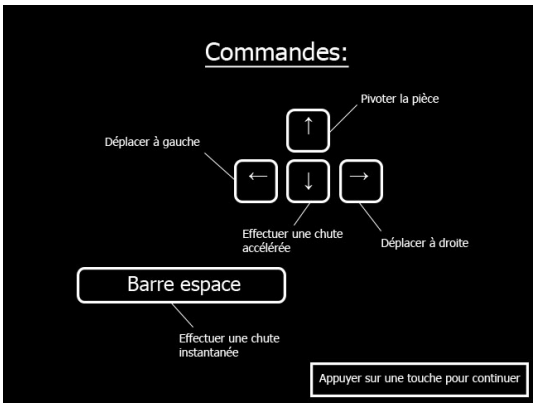


Image d'explication des commandes

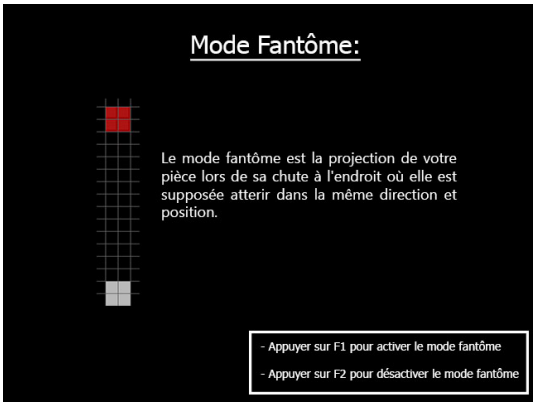


Image d'explication du mode fantôme

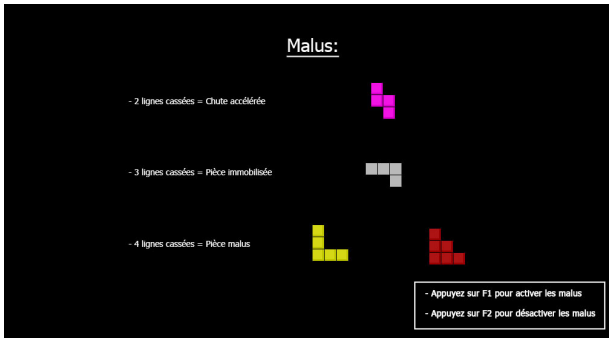


Image d'explication du mode fantôme

5.2 Performances

Le programme fonctionne sur tout type de machines, et ne possède aucun bug à notre connaissance ou aucune lenteur concernant l'utilisation. La boucle infinie ne posent aucun soucis d'utilisation et est gérée par des évènements Pygame.

6 Conclusion

6.1 Problèmes rencontrés

Lors de la réalisation de notre projet, nous avons rencontré plusieurs problèmes. Parmi ceux-ci, nous en avons retenu certains. Par exemple un problème majeur était les pièces qui sortaient du cadre. Cela était dû au fait que les coordonnées de la pièce étaient mal prises en compte par la fonction qui régissait sur la validation de la position d'une pièce et par conséquent elle considérait que les coordonnées de la pièces étaient toujours valides.

De plus, lors de la réalisation du mode 1v1 le joueur qui avait perdu en premier pouvait toujours bouger sa dernière pièce n'importe où sur l'écran et ainsi la faire aller sur la partie de jeu de son adversaire. Cela était dû au fait que lors de la défaite du joueur certaines fonctions étaient désactivées pour qu'il ne puisse plus jouer mais pas toutes, et par conséquent la fonction "isValidPosition" était désactivée mais pas celles qui lui permettaient de bouger sa pièce et ainsi il pouvait la faire aller n'importe où car elle ne détectait aucun obstacle.

Et un autre problème était lorsque le joueur 1 faisait trois lignes d'un coup et que le joueur 2 faisait tomber sa pièce au même moment le programme s'arrêtait de fonctionner car il y avait une erreur. Cette erreur était due au "fallingPiece" qui passait à "none" et qui n'était régénéré qu'au début de la boucle, or le malus dépend du "fallingPiece" adverse et par conséquent comme il était en "none" , il était impossible au malus de se lancer ce qui retournait une erreur et interrompait le programme.

6.2 Améliorations possibles

Une des améliorations possibles auraient été d'ajouter les statistiques d'apparitions des différentes pièces qui auraient pu permettre une génération plus équilibrée des pièces pour ne pas obtenir plusieurs fois d'affilée la même pièces.

Nous avions envisagé d'ajouter le mode en lignes pour que deux utilisateurs puissent jouer de deux ordinateurs différents mais nous y sommes pas parvenus.

L'ajout d'une IA pour que l'utilisateur puisse s'entraîner car en mode solo il ne peut pas subir les malus.

Enfin, le stockage d'une pièce, pour que l'utilisateur ait la possibilité de l'utiliser quand il le désire, ou le choix des touches.