



**UNIVERSITÉ
CAEN
NORMANDIE**

RAPPORT PROJET CONCEPTION LOGICIELLE

PYPUZZLE

Romuald GAFFE
Yanis COSNEFROY
Nguyen Phuong Vy VU
Alexandre BOURGOIN

28 avril 2020

Table des matières

I	Objectifs du projet	3
0.1	Présentation du concept	4
0.2	Cahier des charges	4
0.3	Exemples du jeux publiés	4
II	Fonctionnalités implémentés	6
0.4	Mode solo	7
0.5	Mode multijoueur	8
0.5.1	Joueur vs Intelligence artificielle	8
0.5.2	Joueur vs Joueur en local	9
0.5.3	Joueur vs Joueur en ligne	9
III	Éléments techniques	10
0.6	Les pièces	11
0.7	La grille	11
0.8	L'aléatoire	12
0.9	L'intelligence artificielle	12
0.10	Le réseau	12
0.11	L'interface graphique	13
0.11.1	Le menu	13
0.11.2	Gameplay	15
IV	Expérimentations et usages	16
0.12	Capture écrans	17
0.13	Mesures de performance	18
0.13.1	Interfaces graphiques	18
0.13.2	Effet sonore	18
V	Architecture du projet	20
0.14	Diagrammes des classes	21
0.15	Arborescences des dossiers	22
0.16	Modules utilisés	22

VI	Organisation	23
VII	Conclusion	25

Première partie

Objectifs du projet

0.1 Présentation du concept

Tout d’abord, nous avons choisi de créer un puzzle block puisque nous avons plus d’affinité et d’idées en rapport avec ce sujet. Nous voulions que ce jeu soit simple d’utilisation afin que tout le monde puisse y jouer. Le jeu que nous avons développé est composé de 4 modes de jeux, le premier étant le mode de jeu solo, il se présente sous forme d’une grille carrée de 10 cases de côté. Le joueur obtient 3 pièces à placer sur la grille tirées aléatoirement parmi un total de 30 pièces (toutes orientations incluses). Les pièces qui apparaissent doivent toutes être placées dans la grille avant que les suivantes soient tirées. Si le tirage courant du joueur ne peut pas être placé alors un écran de fin de jeu s’affiche, montrant le score de la personne. Sur ce menu, le joueur aura la possibilité de retourner au menu principal ou de recommencer une partie. Ensuite le jeu se compose d’un mode multijoueur. Le premier étant contre une intelligence artificielle ayant le même mode de fonctionnement que le mode solo. Seul le score actuel de l’intelligence artificielle figure sur l’écran du joueur afin d’éviter d’en copier la stratégie. Le second mode multijoueur est, lui, un mode joueur contre joueur en local sur une même machine. Les grilles des joueurs s’affichent l’une après l’autre afin de limiter, de même, l’imitation de la stratégie adverse. Le dernier mode de jeu est un mode joueur contre joueur également mais en ligne où le but est de faire le plus de points.

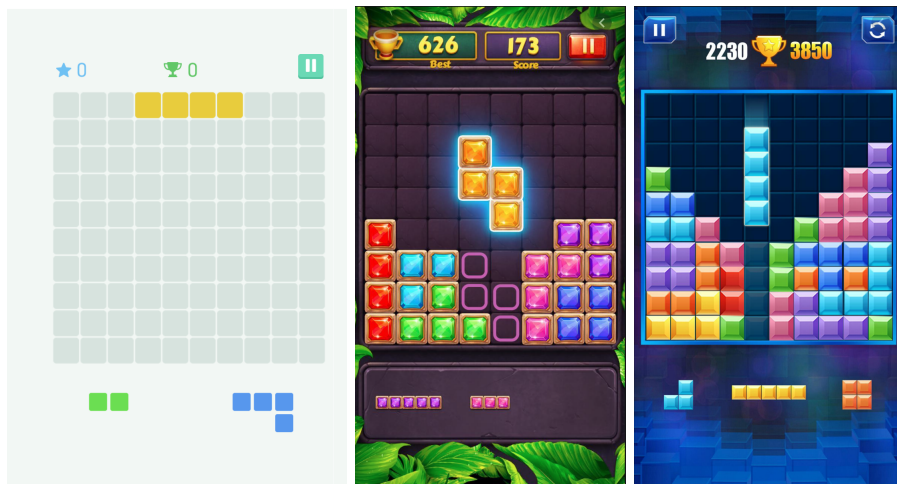
0.2 Cahier des charges

Comme dit précédemment, nous voulions que le jeu soit simple d’utilisation afin qu’il puisse être accessible au plus grand nombre de personnes, nous voulions également que le jeu puisse être joué peu importe si on est seul ou plusieurs, pour cela, nous avons décidé d’implémenter en accord avec le sujet du projet :

- 4 modes de jeu :
 - Mode de jeu solo.
 - Mode de jeu multijoueur joueur contre une intelligence artificielle.
 - Mode de jeu multijoueur joueur contre joueur en local.
 - Mode de jeu multijoueur joueur contre joueur en ligne.
- Un système de tirage pseudo-aléatoire des pièces.
- Une interface simple et ergonomique pour que l’utilisateur ne se perde pas.
- Un programme divisé en plusieurs modules.

0.3 Exemples du jeux publiés

Voici plusieurs block-puzzle provenant de jeux.fr, et des applications pour android :



On peut y retrouver les mêmes caractéristiques que notre projet. Notre projet a été conçu sur une base classique de block-puzzle qui réunit ces trois images montrées précédemment. Les différences entre ces block-puzzles et le nôtre sont l'aspect graphique ainsi que la sauvegarde du meilleur score établis sur le jeu par le joueur. On peut constater à l'essai de notre block-puzzle uniquement l'apparition des scores qui vont être actifs en fonction du jeu du joueur. Cependant nos règles sont un peu différentes par rapport aux jeux déjà publiés.

Deuxième partie

Fonctionnalités implémentées

0.4 Mode solo

Tout d'abord la structure du jeu en mode solo est une structure de jeu plutôt classique comme on peut le voir dans l'image présente ci-contre.

Comme on peut le constater à l'image du projet, il y a une grille placée au centre de la fenêtre qui possède une taille fixe. Autour de cette grille, va se trouver des informations essentielles pour le joueur comme son score actuel qui est placé en haut à gauche de la fenêtre. À droite, va se trouver le joueur qui joue. En dessous de la grille, on va retrouver les 3 pièces que le joueur devra placer sur la grille. Enfin, tout en bas à droite, il va y avoir un bouton pour permettre à l'utilisateur de retourner au menu du jeu.

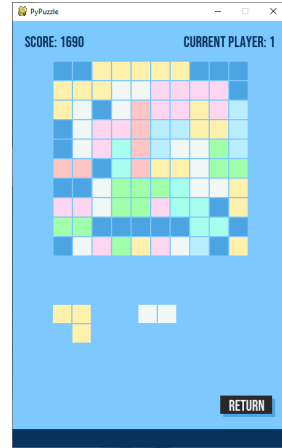


FIGURE 1 – Gameplay solo

Ensuite, les règles du jeu vont s'appliquer au fur et à mesure du jeu du joueur. Quand un block sera placé dans la grille, il va y avoir un ajout de 30 points à son score. Et dès qu'une ligne ou une colonne sera remplie, il va y avoir une disparition de la ligne ou de la colonne voir même les deux si les conditions sont réunies. Le bonus de point sera alors de 100 points lorsqu'une ligne ou une colonne est construite par l'utilisateur. Le joueur ne peut pas mettre sa partie en pause. Cependant, il n'y a pas de limite de temps. Dès que les trois blocs sont placés dans la grille, trois autres blocs vont réapparaître avec une probabilité de difficultés qui est calculée en fonction du score.

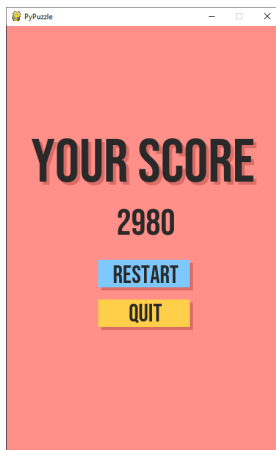


FIGURE 2 – Gameplay solo

Enfin, si le joueur perd en étant bloqué sur le placement impossible d'une pièce dans la grille, alors le jeu s'arrêtera et affichera un Game Over montré ci-contre avec les informations de jeu du joueur :

Comme on peut le voir sur cette photo, le score du joueur est affiché. Afin d'éviter au joueur de repartir dans le menu pour relancer une partie, il a la possibilité de la relancer directement grâce au bouton "restart". Il a aussi la possibilité de quitter le jeu en cliquant sur le bouton "quit". Si le joueur veut retourner au menu, alors il devra d'abord recommencer une partie et cliquer sur le bouton "return" se trouvant en bas à droite de la fenêtre.

0.5 Mode multijoueur

Différents modes de multijoueur sont présents dans ce projet. On peut y retrouver un multijoueur où un joueur réel va jouer contre une intelligence artificielle, un multijoueur où deux joueurs vont s'affronter sur un même écran. Et enfin un multijoueur en réseau où deux joueurs vont pouvoir s'affronter sur deux écrans séparés. En dessous se trouve une image représentant le menu du multijoueur. On peut également y retrouver la possibilité d'un retour au menu principal dans le menu si le joueur veut jouer seul.

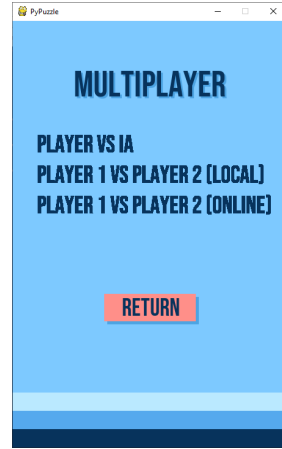


FIGURE 3 – Menu multijoueur

0.5.1 Joueur vs Intelligence artificielle

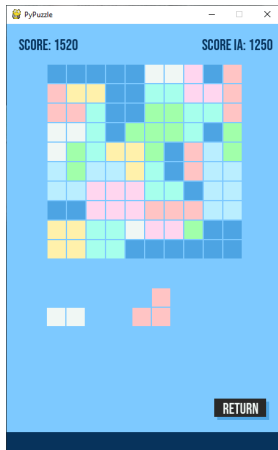


FIGURE 4 – Joueur vs IA

Tout d'abord, il faut savoir que le mode multijoueur avec le joueur contre l'intelligence artificielle se joue comme le mode solo. Le joueur va pouvoir jouer comme il voudra. Il n'aura pas accès à la grille de l'intelligence artificielle pour éviter de la tricherie. L'utilisateur va avoir comme information sa grille, ses points, les points de son adversaire. Le joueur et l'intelligence artificielle vont tout les deux avoir les mêmes blocs pour que la difficulté soit la même de chaque côté. Le joueur va pouvoir retourner au menu si il le souhaite, toujours grâce au bouton "return" se trouvant en bas à droite. Ci-contre, va se présenter une photo de la grille du jeu avec les différentes informations énoncées précédemment que le joueur va avoir.

Ensuite les règles du jeu sont les mêmes que pour le solo. Il n'y a pas de mise en difficulté pour qui que ce soit car si l'un remplit une ligne ou une colonne, l'autre ne va pas recevoir de pièce qui peut le mettre en difficulté. Ici il y a juste une concurrence au niveau des points qui va justement permettre de départager la victoire entre le joueur et l'intelligence artificielle.

Enfin, si le joueur perd en étant bloqué sur le placement impossible d'une pièce dans la grille, alors le jeu s'arrêtera et affichera un Game Over comme montré ci-dessous (dans la partie Joueur VS Joueur local) avec les informations de jeu du joueur. Si le jeu ne s'arrête pas c'est qu'il y a encore une possibilité de jouer.

Comme on peut le voir sur cette photo, le score du joueur ainsi que celui de l'intelligence artificielle sont affichés. De plus on a décidé de rajouter une ligne pour faire ressortir le gagnant de la partie. Dans cette image, Player1 est le joueur se trouvant derrière l'écran et Player2 est l'intelligence artificielle. Afin d'éviter au joueur de repartir dans le menu pour relancer une partie, il a la possibilité de la relancer directement grâce au bouton "restart". Il a aussi la possibilité de quitter le jeu en cliquant sur le bouton "quit". Si le joueur veut retourner au menu, alors il devra d'abord recommencer une partie et cliquer sur le bouton "return" se trouvant en bas à droite de la fenêtre.

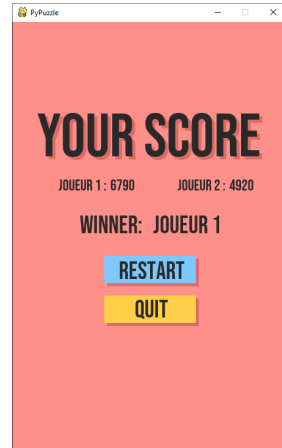


FIGURE 5 – Fin du jeu multi

0.5.2 Joueur vs Joueur en local

Le concept est le même que celui contre l'intelligence artificielle, mais cette fois-ci il se fera contre un autre joueur sur un même ordinateur, chaque joueur joue l'un après l'autre, le but étant de pouvoir placer les pièces tirées le plus longtemps possible, le premier à ne plus pouvoir placer son tirage se voit perdre. Le nombre de point n'est donc pas très important à prendre en compte dans ce mode de jeu.

Il y a toujours la possibilité de quitter la partie à tout moment avec le bouton "return" situé en bas à droite.

0.5.3 Joueur vs Joueur en ligne

Le mode de jeu en ligne se démarque un peu des autres modes de jeu multijoueur pour se rapprocher un peu plus du mode de jeu solo puisque le but ici est d'avoir le meilleur score, le tirage des deux joueurs est indépendant, ce qui ajoute une part de chance à la partie. Dans ce mode de jeu, si un joueur perd, l'autre joueur continue de jouer quand même, la partie n'est réellement fini que quand les deux joueurs ne peuvent plus jouer.

Troisième partie

Éléments techniques

0.6 Les pièces

Les pièces sont représentées grâce à deux classes, il y a d'abord la classe **Pieces** qui a comme propriété un tableau à deux dimensions, chaque lignes représentent une pièce différente et chaque colonnes représentent une "couche" de la pièce, comme toutes les pièces rentrent dans un carré de 5 par 5, il y a donc 5 couches. Une couche est une chaîne de caractères contenant des 0 et des 1. Chaque 1 représente une case constituant la pièce et les 0 représentent le vide. L'objet **Pieces** est également chargé du tirage des pièces grâce aux propriétés **history** où sont stockées les prochaines pièces tirées ainsi que la méthode **alea** qui est la méthode de tirage pseudo-aléatoire.

La pièce en elle-même est représentée par la classe **Piece** qui hérite de **Pieces**, elle contient comme propriété sa couleur, un numéro correspondant à sa place dans le tableau **pieces** contenant tous les pièces vues précédemment, sa position sur l'affichage quand elle est affichée dans le tirage ou déplacée par l'utilisateur, un booléen qui contient **True** si la pièce est actuellement saisie par l'utilisateur et **False** si elle ne l'est pas. La couleur des pièces dépend de la valeur de la chaîne de caractères de représentation de la pièce, par défaut nous avons vu que les couches des pièces sont représentées par des 1 et des 0, la méthode **applyColor** va changer chaque couche de la pièce en entière et la multiplier par la valeur correspondante à la couleur voulue. Cependant le passage de la chaîne de caractère à un entier va "dégrader" la pièce puisque tous les 0 situés avant les 1 dans la chaîne de caractères vont être supprimés, pour remédier à cela, il existe une autre méthode : **repair** qui permet de rajouter à nouveau les 0 et donc "réparer" la pièce.

COUCHE PAR DÉFAUT	APRÈS PASSAGE PAR LA FONCTION INT()	RÉSULTAT APRÈS LA MULTIPLICATION POUR OBTENIR LA COULEUR 7 (MULTIPLICATION)	RÉTABLISSEMENT DE LA FORME STANDARD APRÈS PASSAGE PAR LA FONCTION "REPAIR"
«01110»	==> 1110	==> 7770	==> «07770»

0.7 La grille

La grille est mise sous la forme d'un objet contenant comme propriétés sa taille, une grille où on placera les pièces, un objet **Pieces** qu'on a décrit précédemment ainsi qu'un tableau où sont listées les lignes et colonnes complétées dans la grille.

L'objet **Grid** contient également des méthodes permettant de la modifier ou de voir son état :

- **init** : Cette méthode permet d'initialiser la grille c'est à dire ajouter les listes correspondantes aux colonnes avec la taille spécifiée.
- **print** : Permet d'avoir l'état de la grille affichée dans la console.
- **definePhysicalLimits** : Permet d'ajouter tout autour de la grille des 1 afin d'empêcher le joueur de placer des pièces en dehors de la grille.

- *isPiecePlaceable* : Permet de voir si une pièce peut être placée à des coordonnées données.
- *putPiece* : Permet de placer une pièce dans la grille.
- *isThereAlignment* : Permet d'ajouter à la propriété *linesCompleted* les lignes et colonnes complétées.
- *eraseAlignment* : Supprime les alignements dans la grille.
- *isDrawPlaceable* : Vérifie si le tirage d'un joueur donné peut être placé dans la grille.

0.8 L'aléatoire

0.9 L'intelligence artificielle

L'intelligence artificielle se présente comme une dérivée de la classe *Player* car elle a les mêmes priorités qu'un joueur comme ses points ou un tirage, le changement se fait avec la méthode *determineWhatToPlay* qui retourne le placement le plus judicieux à effectuer sous la forme d'un tableau contenant le poids attribué au mouvement, les coordonnées où il doit se faire et la pièce à jouer.

La méthode va tester pour chaque case de la grille si chaque pièce peut être placée, dans ce cas le poids du mouvement sera incrémenté de 1, ensuite si le mouvement crée une ou plusieurs lignes, il rajoutera au poids le nombre de lignes remplies. Pour deux poids égaux, ce sera la mouvement le plus en bas à droite de la grille qui sera compté.

Sur 100 parties solo jouées par l'intelligence, voici les statistiques que nous avons pu récolter (via le programme "IAStats.py" situé à la racine du répertoire git) :

Nombre de rounds :

Moyenne : 51.08

Maximum : 86

Minimum : 26

Nombre de points :

Moyenne : 2317.4

Maximum : 4400

Minimum : 780

0.10 Le réseau

Le jeu PyPuzzle possédé par le joueur ne fait office que de client pour le jeu en réseau donc les personnes possédants le jeu n'auront pas la possibilité d'héberger leur propre partie à partir de leur client mais devront rejoindre un serveur.

La partie client est un jeu solo classique comme décrit précédemment, le client va recevoir à chaque tour de boucle l'état de la partie qui peut être soit :

- "wait" : En attente d'un second joueur.
- "running" : La partie est en cours.
- "quit" : Un des client a quitté la partie, l'autre client retourne au menu principal.
- "gameover" : Un des deux joueurs a fini de jouer.

Il va également demander l'état des points des deux joueurs afin de les afficher. Tout se fait grâce à l'objet **Network** permettant de simplifier la connexion et l'envoi de donnée au serveur.

Côté serveur, chaque connexion client incrémente une variable *idCount* qui va permettre d'attribuer un joueur se connectant à une partie. Chaque partie dispose d'un *gameID* qui permet de l'identifier dans un dictionnaire : *games*. Chaque partie de jeu dans le dictionnaire contient le nombre de points des deux joueurs ainsi que l'état dans lequel est la partie. Lorsque qu'une partie se termine, les joueurs se déconnectent, *idCount* est décrémenté et les données de la partie dans le dictionnaire sont supprimées afin de libérer de la mémoire.

0.11 L'interface graphique

Pour l'interface du jeu, on utilise **Pygame**, une bibliothèque graphique dédiée aux jeux vidéos. Pygame n'est pas fourni avec Python. Donc, pour jouer, vous devez télécharger et installer Pygame à la place. Pour appliquer Pygame :

```
|| import pygame as pg
```

On va utiliser pygame pour faire et designer l'interface du jeu, pas utiliser les images déjà faites et les mettre dans le jeu. Et on va appliquer en même temps une méthode pour faire le menu démarrer, le menu multijoueur et le menu de la fin du jeu.

0.11.1 Le menu

D'abord, il faut faire un réglage des tailles des positions. *pygame.Rect()* n'est pas dessiné un rectangle, mais va plutôt servir à juste déterminer les tailles et les positions d'un rectangle.

```
|| #tailles de fenetre
|| SCREENHEIGHT = 700
|| SCREENWIDTH = 450
|| screensize = (SCREENWIDTH, SCREENHEIGHT)
|| #positions de la grille
|| boardX = 65
|| boardY = 65
|| #tailles et positions des boutons
|| soloButtonRect = pg.Rect(150, 318, 150, 50)
|| multiButtonRect = pg.Rect(150, 383, 150, 50)
|| quitButtonRect = pg.Rect(150, 448, 150, 45)
```

Pour commencer le jeu, on doit appeler la fonction *pygame.init()*, elle doit être appelée en premier pour que de nombreuses fonctions Pygame fonctionnent

```
|| pg.init()
```

Ensuite, on doit construire une boucle pour tout menu. Si une boucle est toujours vrai, alors elle fonctionne pour toujours.

```
def menu():
    doContinue = True
    while doContinue:
        for event in pg.event.get():
            if event.type == pg.QUIT:
                fnc.quitGame() #dans fonction.py pour quitter
```

Puis, on ajoute des boutons, on utilise *.collidepoint()* pour déterminer les positions des boutons.

```
elif event.type == pg.MOUSEBUTTONDOWN:
    if soloButtonRect.collidepoint(event.pos):
        solo() #autre menu pour basculer entre eux
    if multiButtonRect.collidepoint(event.pos):
        multiMenu() #autre menu pour basculer entre eux
    if quitButtonRect.collidepoint(event.pos):
        fnc.quitGame() #dans fonction.py pour quitter
pg.display.flip() #pour mise a jour
```

Ensuite, on construit une interface d'affichage dans un autre fichier appelé *display.py*. Pour les textes, on utilise *pygame.font* avec les sous-éléments *pygame.font.Font()*, *"nom".render()* et l'élément *"screen".blit()*. Pour les rectangles (boutons), on utilise *pygame.draw.rect()*. Et pour colorer le fond, on utilise *"screen".fill()*.

```
NAVY = (7, 51, 92)
pg.font.init()
bigFont = pg.font.Font('assets/BebasNeue-Regular.ttf', 100)
pypuzzle = bigFont.render("PyPuzzle", True, NAVY)

def displayMenu(win):
    win.fill(BACKGROUND_COLOR)
    pg.draw.rect(win, BOARD_COLOR, (155, 323, 150, 45))
    pg.draw.rect(win, BOARD_COLOR, (155, 388, 150, 45))
    pg.draw.rect(win, BOARD_COLOR, (155, 453, 150, 45))
    #titre
    win.blit(pypuzzleShadow, (82, 170))
    win.blit(pypuzzle, (78, 165))
    #decoration
    pg.draw.rect(win, LIGHTBLUE, (0, 610, 450, 30))
    pg.draw.rect(win, BLUE, (0, 640, 450, 30))
    pg.draw.rect(win, NAVY, (0, 670, 450, 30))
```

Pour les sons, on utilise *pygame.mixer* avec les sous-éléments *pygame.mixer.Sound()*, *"nom".play()* et *"nom".stop()*. Type de fichier est .wav (Les formats de fichiers audio pris en charge par Pygame sont OGG, WAV et MP3.)

```
pg.mixer.init() #pour fonctionner
soundMenu = pg.mixer.Sound("assets/menu.wav")
soundMenu.play(-1, 0, 0) #-1 signifie que la musique se repetera
indefinement
```

On utilise juste une animation, c'est le survol quand le curseur de la souris entre dans les cases des boutons. Ici, on utilise *pygame.mouse* avec le sous-élément *pygame.mouse.get_pos()*. pour déterminer les positions de la souris.

```
# HOVER
pos = pg.mouse.get_pos()
if 340 + 85 > pos[0] > 340 and 615 + 30 > pos[1] > 615:
```

```
pg.draw.rect(screen, dsp.YELLOW, (340, 615, 85, 30))
screen.blit(dsp.returnMenuText, (354, 617))
else:
pg.draw.rect(screen, dsp.GRAY, (340, 615, 85, 30))
screen.blit(dsp.returnMenuText1, (354, 617))
```

0.11.2 Gameplay

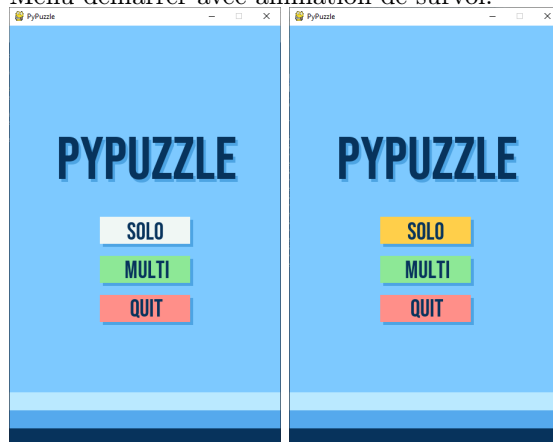
Quatrième partie

Expérimentations et usages

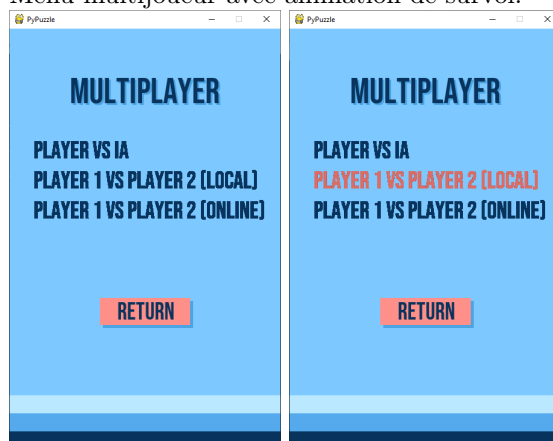
0.12 Capture d'écrans

Les captures ci-dessous rendront le joueur passionné de jouer au jeu.

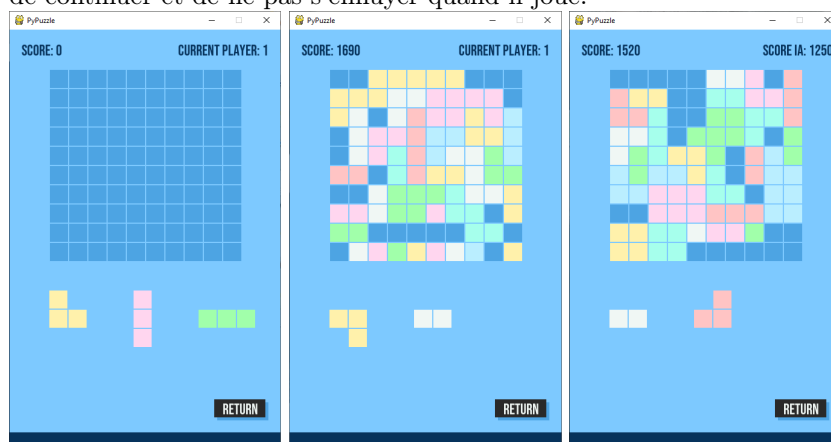
1. Menu démarrer avec animation de survol.



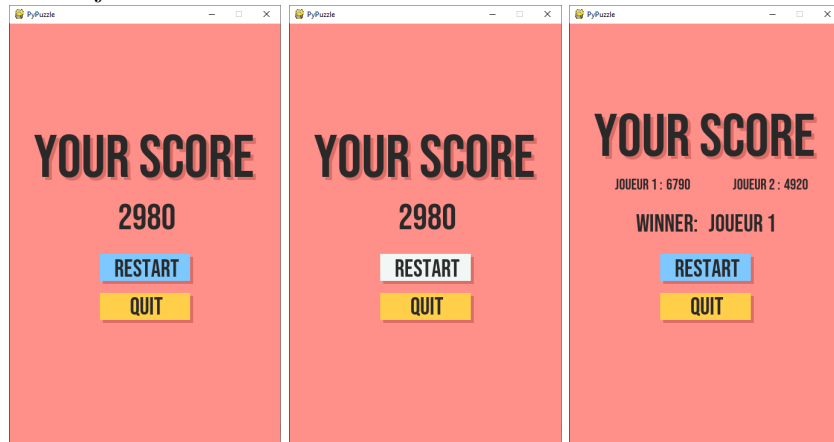
2. Menu multijoueur avec animation de survol.



3. Gameplay (solo et multi). De nombreuses couleurs permettent au joueur de continuer et de ne pas s'ennuyer quand il joue.



4. Fin du jeu avec animation de survol.



0.13 Mesures de performance

0.13.1 Interfaces graphiques

Quelle est l'interface du jeu pour vous ? Selon nous, d'après le contenu du jeu, l'interface du jeu est la plus importante pour attirer les joueurs. C'est un jeu pour tout le monde, on choisit le minimalisme avec une palette des couleurs pastels car ce sont des couleurs douces et non sombres. Elles procurent une atmosphère relaxée quand on joue. On échoue, mais sans s'énerver.

- Font : Bebas Neue Regular (Free Copyright)
- Couleur principale : **bleu(125, 201, 255)**
- Couleurs des pièces : les couleurs pastels



- Couleurs de menu



0.13.2 Effets sonores

On voulait ajouter de la musique à notre jeu, car on pensait que cela attirerait les joueurs. Comme la musique est quelque chose de très populaire à partir de maintenant, on pensait que les joueurs aimeraient qu'on ajoute de la musique au jeu que nous avons créé.

Au début de la conception du jeu, il n'y avait pas de bruit tonitruant, ni de sons nerveux. On souhaitait mettre le confort et la joie en avant lors d'une partie. C'est pourquoi on a choisit l'accord de base au piano C(Do) majeur. Ensuite, le son des boutons est la note F(Fa) correspond à la première note de son du menu. Puis, quand on joue, il n'y a pas de son de fond, il existe que le son des pièces pour que l'on se sente concentré. Afin de s'investir un minimum dans

le jeu, le son du bongo apparait quand on drag and drop les pièces. Enfin, on choisit l'accord guitare C(Do) mineur et on pense à créer une synchronisation de la musique.

Cinquième partie

Architecture du projet

0.14 Diagrammes des classes

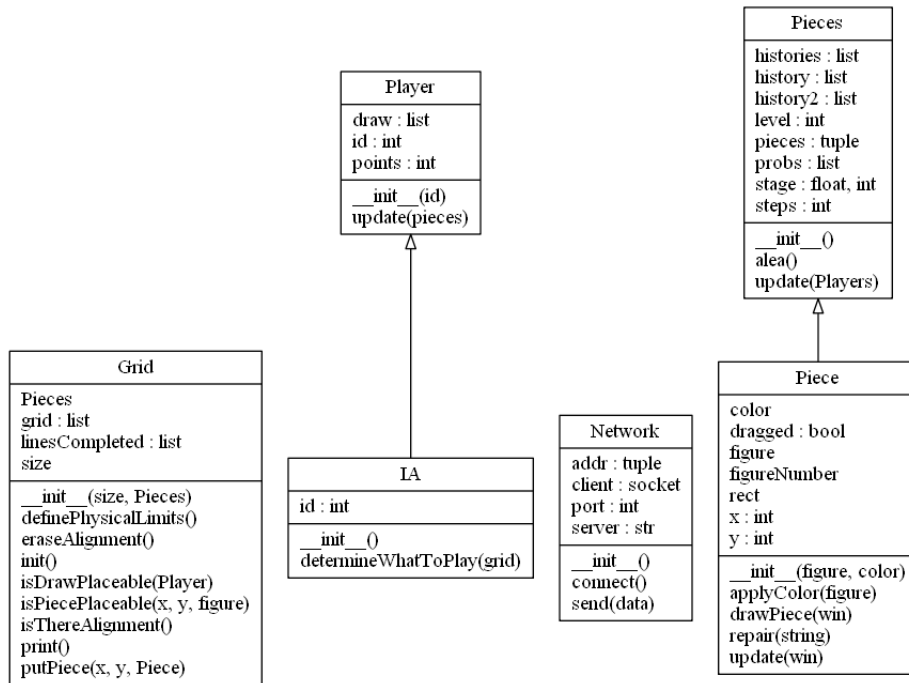
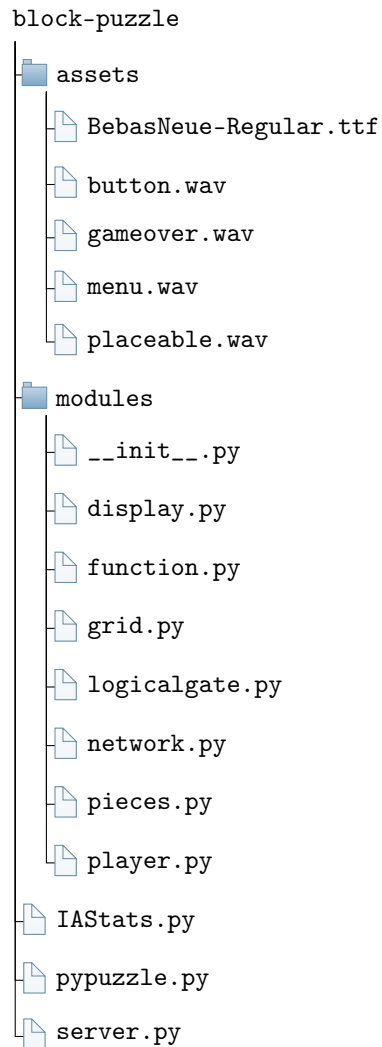


FIGURE 6 – Classes du dossier **modules** des fichiers *grid.py*, *network.py*, *pieces.py* et *player.py*.

FIGURE 7 – Classe de fichier *server.py*

Nous n'utilisons pas des classes pour les fichiers *IAStats.py*, *pypuzzle.py*, *display.py*, *function.py* et *logicalgate.py*.

0.15 Arborescences des dossiers



0.16 Modules utilisés

```
import pygame as pg
import random as rnd
import socket
import threading
import pickle
import modules.logicalgate as lg
import modules.grid as gd
import modules.display as dsp
import modules.player as player
import modules.functions as fnc
import modules.pieces as pcs
import modules.network as ntw
```

Sixième partie

Organisation

Fonctions/classes	Fichiers	Personnes
updates	pypuzzle.py	Alexandre
updatesMultiLocal	pypuzzle.py	Alexandre
menu	pypuzzle.py	Romuald, Vy, Yanis
multiMenu	pypuzzle.py	Alexandre, Vy
gameOverSolo	pypuzzle.py	Alexandre, Vy
gameOverMulti	pypuzzle.py	Vy, Yanis
solo	pypuzzle.py	Alexandre, Romuald, Yanis, Vy
multiLocal	pypuzzle.py	Alexandre, Vy
multiIA	pypuzzle.py	Alexandre, Vy, Romuald
multiOnlineClient	pypuzzle.py	Alexandre, Vy
	server.py	Alexandre
	IAStats.py	Alexandre
displayMenu	modules/display.py	Vy
displayMulti	modules/display.py	Vy
displayBoard	modules/display.py	Alexandre, Vy
displayDrawPieces	modules/display.py	Alexandre
displayTexts	modules/display.py	Alexandre, Vy
displayTextsIA	modules/display.py	Alexandre
displayTextsOnline	modules/display.py	Alexandre
displayGameOverSolo	modules/display.py	Alexandre, Yanis, Vy
displayGameOverMulti	modules/display.py	Yanis, Vy
displaygameOverMultiOnline	modules/display.py	Alexandre, Yanis, Vy
displayWaitPlayers	modules/display.py	Alexandre
	modules/functions.py	Alexandre
	modules/logicalgate.py	Alexandre
Grid	modules/grid.py	Alexandre, Yanis
Network	modules/network.py	Alexandre
Pieces	modules/pieces.py	Alexandre, Romuald
Piece	modules/pieces.py	Alexandre
Player	modules/player.py	Alexandre
IA	modules/player.py	Alexandre

Numéros de téléphone :
 Romuald GAFFE :
 Yanis COSNEFROY : 07 60 68 78 78
 Nguyen Phuong Vy VU : 06 65 77 58 80
 Alexandre BOURGOIN :

Septième partie

Conclusion

Pour conclure, nous voulons remercier Mme.LAMBERT de nous avoir encadrée pendant les cours de travaux pratiques ainsi que M.BONNET pour ses cours magistraux. Merci d'avoir pris le temps de lire ce rapport.