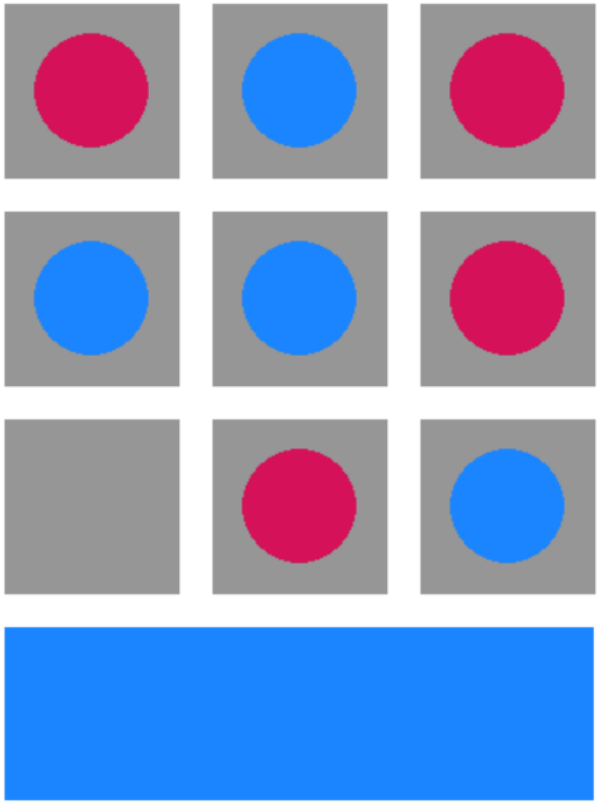
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

PRÀCTICA PA1

GRAU EN INTEL·LIGÈNCIA ARTIFICIAL



Alumnes: Óscar Lu i Chenhui Lucía Wu

Professor: Jordi Delgado

Assignatura: Programació i algoritme 1

ÍNDEX

INTRODUCCIÓ

Aquest treball s'ha centrat a completar la implementació del fitxer **abs_board_h.py** perquè inclogui funcions específiques que permetin jugar al clàssic joc de 3-en-ratlla.

L'enunciat inicial proporcionava aquest fitxer incomplet **abs_board_h.py**, que contenia les capçaleres de les funcions necessàries per a la gestió del joc i els conceptes bàsics per estructurar-lo. A més, incloïa dos fitxers principals: **main_gui.py**, que implementava el joc utilitzant la llibreria Pygame per proporcionar una interfície gràfica, i **main_txt.py**, una versió basada en text que permet interactuar amb el joc a través del terminal.

El nostre objectiu amb aquesta pràctica era aconseguir que el fitxer funcionés correctament de dues maneres:

- En el terminal, d'una manera més simple i textual
- Amb Pygame, d'una manera més visual i interactiva

En aquest informe, primerament, explicarem com hem completat el fitxer **abs_board_h.py** i destacarem algunes de les relacions importants amb els fitxers **main_gui.py** i **main_txt.py**. Posteriorment, mostrarem els resultats del programa en les dues modalitats: al terminal i mitjançant Pygame. Finalment, esmentarem algunes de les dificultats que hem trobat durant el procés i donarem les nostres conclusions.

EXPLICACIÓ DEL CODI

Abans d'explicar res, hem de saber que:

- El jugador 1 està representat pel 0 i utilitza la 'X' com a fitxa.
- El jugador 2 està representat pel 1 i utilitza la 'O' com a fitxa.
- Cada jugador té 4 fitxes
- Les columnes i les files de la taula es comencen a comptar des de 0, és a dir, en una taula de 3x3, la posició de dalt a l'esquerra serà (0,0).

(Aconsellem tenir el programa a prop per poder seguir l'explicació de manera més senzilla)

Inicialització de variables locals en set_board_up()

Per començar, inicialitzem unes variables locals:

- **board**: Representa el tauler com una matriu de dimensions , però en realitat es tracta d'una llista de llistes, on cada element inicialment està buit. Posteriorment, aquestes posicions podran contenir els valors '**X**', '**O**' o continuar sent buides (representades per un espai en blanc **" "**).

```
    board = [[" "]*BSIZ for _ in range(BSIZ)]
```

- **played_stones**: Aquesta variable és una llista que emmagatzema les pedres jugades, representades mitjançant tuples **(x, y, color)**, utilitzant el **namedtuple** Stone. Cada element de la llista és un objecte de tipus **Stone** que conté les coordenades de la casella (x, y) on es va jugar la pedra i el color de la pedra.

  Exemple: Després del primer moviment del jugador 1 a les coordenades (0, 0), la llsita **played_stones** seria: **[Stone(0, 0, PLAYER_COLOR[curr_player])]**.

```
played_stones = []
```

- **curr_player**: Variable que determina quin jugador que està jugant en cada moment. El valor 0 correspon al jugador 1 i el valor 1 correspon al jugador 2.

```
    curr_player = 0
```

- **`stone_selected`**: Variable booleana que indica si una pedra ha estat seleccionada per moure. El valor serà **True** quan una pedra estigui seleccionada i **False** quan no ho estigui.

```
    stone_selected = True
```

- **`total_stones`**: Variable que emmagatzema la quantitat total de pedres disponibles en el joc, sumant les pedres de tots dos jugadors. En aquest cas, si cada jugador comença amb 4 pedres, el valor inicial **`total_stones`** serà 8 (4 pedres per jugador en una taula de 3x3).

```
    total_stones = stones_per_player * 2
```

- **`stone_itself`**: Variable que emmagatzema les coordenades **`(x, y)`** de la pedra seleccionada pel jugador per moure. Inicialment, el valor és **`(None, None)`** perquè no hi ha cap pedra seleccionada.

```
    stone_itself = (None, None)
```

Funcions anidades a la funció set_board_up

stones()

Aquesta funció bàsicament retorna un iterable amb les pedres que ja han estat jugades, que seria retornar la llista **`played_stones`**.

```
    def stones():
            return played_stones
```

select_st()

Aquesta funció rep com a paràmetres formals **`i`** i **`j`** que corresponen a les coordenades de la casella (fila **`i`** i columna **`j`**) on es troba la pedra seleccionada, respectivament.

```
def select_st(i, j):
```

Seguidament, utilitzem **`nonlocal`** per declarar les variables que havíem definit prèviament fora de la funció, de manera que puguem utilitzar-les i modificar-les dins d'aquest funció.

```
        nonlocal curr_player, total_stones, stone_itself, stone_selected
```

A continuació, fem dues coses, la primera és comprovar quin jugador està jugant (jugador 1 o jugador 2), i la segona és assegurar-nos que les coordenades de la pedra seleccionada estiguin dins dels límits del tauler.

```
        if curr_player == 0 and 0 <= i < BSIZ and 0 <= j < BSIZ:
                pass

        elif curr_player == 1 and 0 <= i < BSIZ and 0 <= j < BSIZ:
                pass
```

Ara bé, si es tracta del jugador 1 o del jugador 2, caldria comprovar si la pedra que ha seleccionat li pertany.

```
        if curr_player == 0 and 0 <= i < BSIZ and 0 <= j < BSIZ:
            if board[i][j] == 'X':
                pass

        elif curr_player == 1 and 0 <= i < BSIZ and 0 <= j < BSIZ:
            if board[i][j] == 'O':
                pass
```

Si es tractés de la seva pròpia pedra, guardarem les coordenades de la pedra seleccionada a la variable **`stone_itself`**, incrementarem **`total_stones`** +1, ja que la funció **`select_st()`** només serà cridada quan ja no hi hagi més pedres disponibles per jugar, és a dir, quan tots dos jugadors hagin jugat les seves pedres i cap d'ells hagi guanyat.

A continuació, establirem **stone_selected = True** per indicar que s'ha seleccionat una pedra correctament i finalment retornarem **True**. En cas contrari, retornarem **False**.

```
        if curr_player == 0 and 0 <= i < BSIZ and 0 <= j < BSIZ:
            if board[i][j] == 'X':
                stone_itself = (i, j)
                total_stones += 1
```

```
                stone_selected = True
                return True

            return False

        elif curr_player == 1 and 0 <= i < BSIZ and 0 <= j < BSIZ:
            if board[i][j] == 'O':
                stone_itself = (i, j)
                total_stones += 1
                stone_selected = True
                return True

            return False
```

end()

Aquesta funció comprova si s'ha fet 3-en-ratlla, tant de manera vertical, horitzontal com diagonal.

Per cada fila del tauler, comprovem si s'ha fet 3-en-ratlla horitzontalment:

```
        for i in range(BSIZ):
            if all(board[i][j] == board[i][0] and board[i][j] != " " for j
in range(BSIZ)):

                return True
```

Aquest codi funciona d'una manera senzilla. Anem a posar un exemple. Suposem que **`i = 0`**, és a dir, que estem a la primera fila. A continuació, agafem l'element que es troba a la primera fila i a la primera columna (**`board[i][0]`**) i comprovem si és igual a les pedres que estan a la mateixa fila però a les següents columnes (**`board[i][j]`**). A més, ens assegurem que les caselles no estiguin buides (**`board[i][j] != " "`**). Si alguna de les files compleix aquesta condició, retornarem **`True`**.

Ara fem el mateix per a les columnes. Comprovem, per a cada columna **j**, si tots els elements de la columna són iguals al primer element de la mateixa columna (**board[0][j]**) i, a més, ens assegurem que les caselles no estiguin buides (**board[i][j] != “ ”**) per a totes les files **i**.

```
        for j in range(BSIZ):
            if all(board[i][j] == board[0][j] and board[i][j] != " " for i
in range(BSIZ)):

                return True
```

Aquesta part del codi comprova si tots els elements de la diagonal principal (és a dir, els elements **board[i][i]**, on **i** va de **0** fins a **BISZ-1**) són iguals al primer element de la diagonal (**board[0][0]**) i, a més, ens assegurem que les caselles no estiguin buides (**board[i][i] != “ ”**).

```
        if all(board[i][i] == board[0][0] and board[i][i] != " " for i in
range(BSIZ)):

            return True
```

A més a més de la diagonal principal també hem de comprovar la diagonal secundària. El següent codi comprova si tots els elements de la diagonal secundària (és a dir, els elements **board[i][BSIZ – 1 - i]**, on **i** va de **0** fins a **BSIZ - 1**) són iguals al primer element de la diagonal secundària (**board[0][BSIZ – 1]**) i, a més, ens assegurem que les caselles no estiguin buides (**board[i][BSIZ -1 – i] != “ ”**).

```
       if all(board[i][BSIZ - 1 - i] == board[0][BSIZ - 1] and board[i][BSIZ
- 1 - i] != " " for i in range(BSIZ)):

            return True
```

Si cap de les condicions anteriors es compleix, vol dir que no s'ha aconseguit un 3-en-ratlla en cap de les tres direccions (horitzontal, vertical o diagonal). En aquest cas, retornem **False** per indicar que el joc no ha acabat i que ningú ha guanyat.

```
        return False
```

move_st(i, j)

Aquesta funció rep com a paràmetres formals **i** i **j** que corresponen a les coordenades de la casella (fila **i** i columna **j**) on es vol moure la pedra, respectivament.

```
def move_st(i, j):
```

Seguidament, utilitzem **nonlocal** per declarar les variables que havíem definit prèviament fora de la funció, de manera que puguem utilitzar-les i modificar-les dins d'aquest funció.

```
        nonlocal curr_player, stone_selected, total_stones, stone_itself
```

Després d'haver seleccionat una pedra amb la funció **select_st(i, j)**, guardem les coordenades de la pedra seleccionada a la variable **stone_itself**. Aquestes coordenades x i y, dins de la funció **move_st(i, j)**, es desestructuren i s'assignen a les variables **x** i **y** respectivament per poder moure la pedra.

```
        x, y = stone_itself
```

Però, atenció: si mai s'ha cridat a la funció **select_st(i, j)**, recorda que les coordenades **stone_itself** seran **(None, None)**.

Més endavant, comprovem si les coordenades introduïdes pel jugador estan dins dels límits del tauler. Si no és així, mostrem un missatge d'error al jugador indicant-li que les coordenades que ha escollit no són vàlides.

```
        if not(0 <= i < BSIZ and 0 <= j < BSIZ):
            print("Las coordenadas que has introducido están mal.")
            return True, curr_player, end()
```

Si això fos cert, retornem **True, curr_player i end()**. Però, per què retornem **True**? La raó és que al fitxer **main_txt.py** aquests tres valors es guarden a les variables **stone_selected, curr_player i end**, respectivament. Com que hem retornat que **stone_selected = True**, això fa que el programa segueixi dins del **while stone_selected and not end**, i així el jugador hauria de tornar a introduir les coordenades de la casella con vol moure la seva pedra.

A més a més, també ens assegurem que la casella seleccionada pel jugador estigui buida. Si la casella ja conté una pedra (és a dir, si no està buida), es mostrarà un missatge d'error al jugador indicant que ja hi ha una fitxa en aquestes coordenades.

```
        if board[i][j] != " ":
            print("Ya hay una ficha en esas coordenadas")
            return True, curr_player, end()
```

Si cap de les condicions anteriors es compleix, això vol dir que les coordenades introduïdes pel jugador són vàlides. Per tant, podem procedir a moure la pedra del jugador a la casella indicada.

```
        if stone_selected:
            pass
```

A continuació, mirem si les coordenades **x** i **y** són diferents de **None**. Si fos cert, voldrà dir que s'ha cridat a la funció **select_st(i, j)** per seleccionar una pedra i després moure-la. Per tant, eliminem la pedra seleccionada i la removem de la llista **played_stones** i fem que la casella on es trobava la pedra estigui buida.

```
        if stone_selected:

            if x != None and y != None:
                played_stones.remove(Stone(x, y, PLAYER_COLOR[curr_player]))
                board[x][y] = " "
```

Seguidament, comprovem quin jugador (jugador 1 o jugador 2) está intentant moure la seva pedra. Dependrà de qui sigui el jugador actual si col·loquem una fitxa '**X**' (per al jugador 1) o una fitxa '**O**' (per al jugador 2) en aquelles coordenades del tauler.

```
            if curr_player == 0:
                board[i][j] = 'X'

            elif curr_player == 1:
                board[i][j] = 'O'
```

Després de moure la pedra, afegim les noves coordenades de la pedra a la llista **played_stones**. A continuació, canviem de jugador (canviant el valor de **curr_player**) i restem **1** al nombre total de pedres disponibles (**total_stones**).

```
            played_stones.append(Stone(i, j, PLAYER_COLOR[curr_player]))
            curr_player = 1 - curr_player
            total_stones -= 1
```

Si el nombre de total de pedres disponibles és zero (**total_stones == 0**), canviem el valor de la variable **stone_selected** a **False** per indicar que ja no es poden moure més pedres, i que ara s'ha de seleccionar una nova pedra per moure-la en el següent torn.

```
            if total_stones == 0:
                stone_selected = False
```

Per últim, retornem **stone_selected, curr_player i end()**.

```
        return stone_selected, curr_player, end()
```

draw_txt()

Aquesta funció s'utilitza per imprimir el tauler a la terminal, sempre que el joc no hagi finalitzat.

```
    def draw_txt(end = False)
```

Seguidament, la funció entra en un **bucle principal** que recorre totes les files i columnes del tauler. Aquest bucle està format per dues iteracions anidades:

Primera iteració (files): El bucle principal utilitza la instrucció **for row in range(BSIZ)** per iterar per cada fila del tauler. La variable **row** representa l'índex de la fila actual.

Segona iteració (columnes): Dins de cada fila, el bucle intern **for col in range(BSIZ)** recorre cada columna. La variable **col** representa l'índex de la columna actual dins de la fila.

```
        for row in range(BSIZ):
            for col in range(BSIZ):
                pass
```

Després d'iterar pel tauler, la funció imprimeix el contingut de les caselles, diferenciant entre dues situacions segons si la casella actual es troba o no a l'última columna de la fila.

Celles que no són de l'última columna:

El contingut de la cel·la actual, que pot ser '**X**', '**O**', o estar buida (**board[row][col]**), es mostra seguit d'una barra vertical (**|**) per separar-la de la següent casella.

```
                if col < BSIZ - 1:
                    print("", board[row][col], "|", end="")
```

Celles que són de l'última columna:

Simplement es mostra el contingut de la cel·la sense afegir cap barra vertical al final.

```
                else:
                    print("", board[row][col], end="")
```

Després d'imprimir totes les caselles d'una fila del tauler, la funció realitza un salt de línia per separar les files entre elles. Això es fa utilitzant una simple instrucció **print()**:

```
            print()
```

Finalment, es mostra una línia separadora composta per guions (**-**). Aquesta línia serveix per distingir clarament les files entre si.

La longitud de la línia separadora es calcula com **BSIZ * 4 – 1**.

```
            if row < BSIZ - 1:
                print("-" * (BSIZ * 4 - 1))
```

En l'apartat de resultat es podrà observar com aquesta funció imprimeix el tauler amb les línies separadores corresponents.

RESULTATS

En aquest apartat es descriuen les dues formes disponibles per jugar al joc: mitjançant la terminal i amb l'ús de Pygame. Cadascuna d'aquestes modalitats ofereix una experiència diferent, però comparteixen la mateixa lògica del joc.

Primer, explicarem els passos necessaris per inicialitzar el joc en cada modalitat. A continuació, es mostrarà el funcionament del programa, destacant les diferències visuals i interactives entre la terminal i la interfície gràfica proporcionada per Pygame.

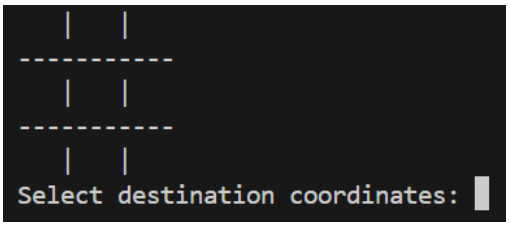
Amb main_txt.py

Passos per inicialitzar el programa:

1. Obre un terminal
2. Accedeix al directori dels fitxers
3. Executa el programa escrivint la següent comanda: **`python3 main_txt.py`**
4. Comença a jugar

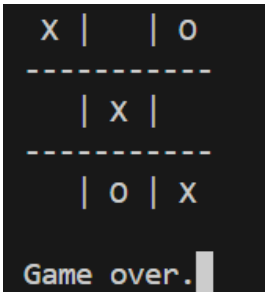
CAS 1: Sense cap equivocació. En aquest cas, els jugadors segueixen les regles del joc correctament, sense cometre cap error, i un dels jugadors acaba guanyant sense haver de seleccionar la pedra, o sigui, cridar a la funció **`select_st()`**.

Al començar en la terminal sortirà això:



Llavors, els jugadors seleccionen les coordenades on volen col·locar les seves fitxes. Recorda: el jugador 1, amb la fitxa '**X**', serà el primer en jugar, seguit pel jugador 2, amb la fitxa '**O**'. Els jugadors segueixen introduint les coordenades fins que algú guanyi.

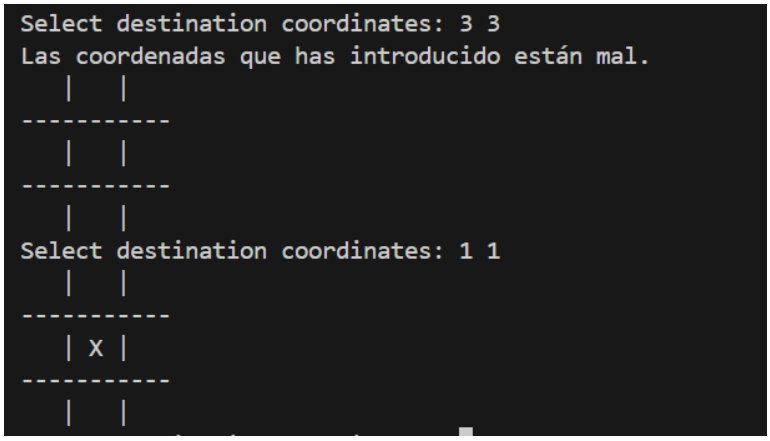
En aquest cas ha guanyat el jugador 1.



CAS 2: En aquest cas, un jugador cometi algun error durant el joc. Els errors més comuns poden ser: **`seleccionar coordenades incorrectes, seleccionar una pedra que no és la seva i posar una fitxa en una casella ocupada`**.

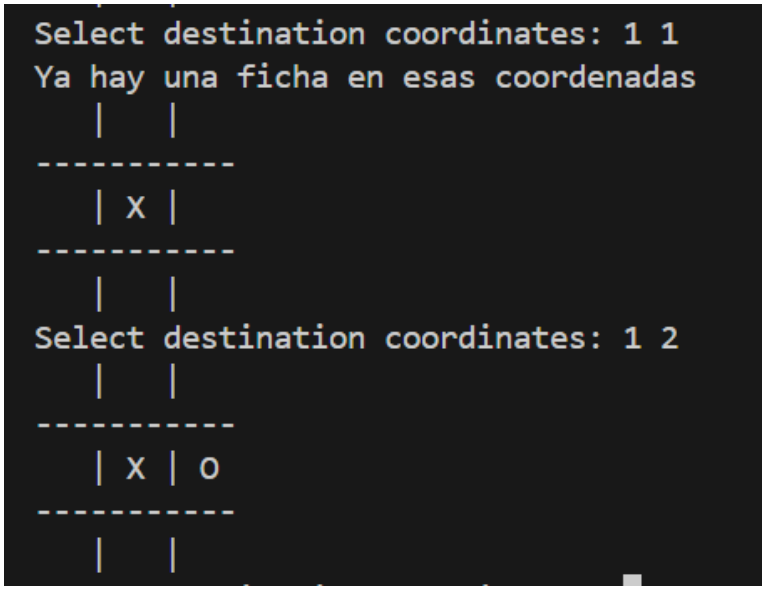
En qualsevol d'aquests casos, el programa retorna un missatge d'error i torna a demanar que es seleccionin noves coordenades vàlides.

Coordenades fora del tauler :



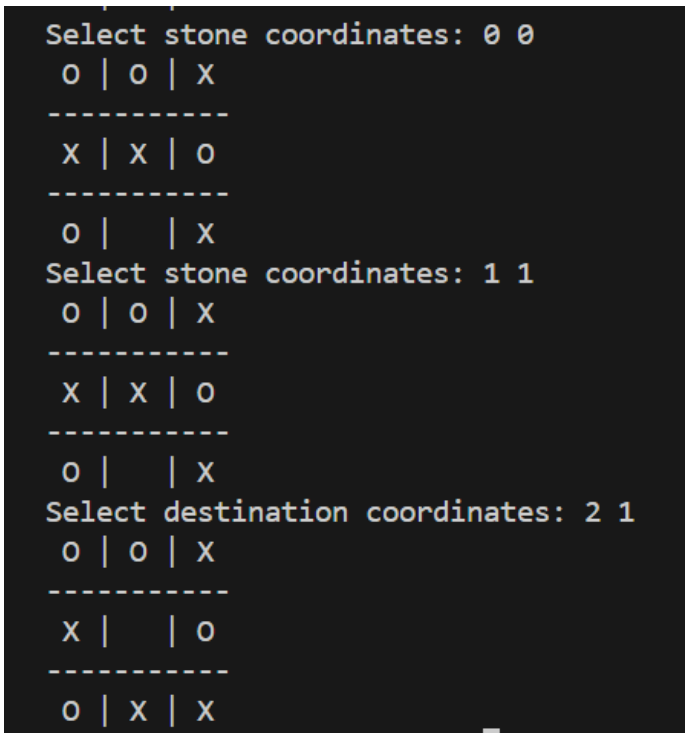
Ha escollit una coordenada inexistent (3,3).

Coordenades a una posició ja ocupada:



Ha escollit unes coordenades que ja estaven ocupades per una altra fitxa.

Quan selecciona una pedra que no és la seva:



En aquest cas, el jugador 1 (**'X'**) intenta moure una pedra, però selecciona una pedra del jugador 2 (**'O'**). Per tant, el programa li permetrà seleccionar novament unes coordenades vàlides per seleccionar una pedra que sigui seva.

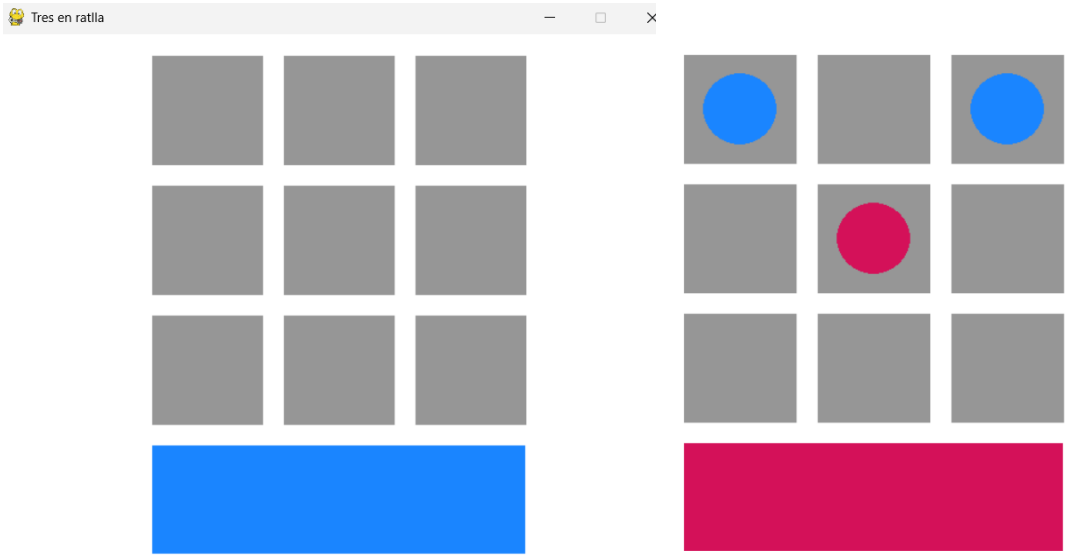
Després d'aquesta correcció, el jugador 1 escull unes noves coordenades, com ara **(1, 1)**, per seleccionar una pedra pròpia. Finalment, mou la pedra a una casella buida i el joc continua.

Amb main_gui.py

Passos per inicialitzar el programa:

1. Obrim un terminal
2. Instal·lem Pygame si no el tenim instal·lat: **`pip install pygame`**
3. Anem al directori on es troba els programes
4. Escrivim la comanda per executar el joc: **`python3 main_gui.py`**
5. Juguem

Quan comencem el joc amb la interfície gràfica utilitzant Pygame, veurem una finestra amb el tauler de joc (en aquest cas un tauler de 3x3) amb la configuració inicial. El tauler estarà buit, esperant que els jugadors facin les seves jugades.



A la part inferior del tauler, veiem un rectangle que mostra quin jugador té el torn per jugar. El jugador 1 serà representat amb el color **`blau`** i el jugador 2 amb el color **`vermell`**. Aquest rectangle es canvia cada vegada que un jugador fa una jugada, indicant qui ha de moure a continuació.

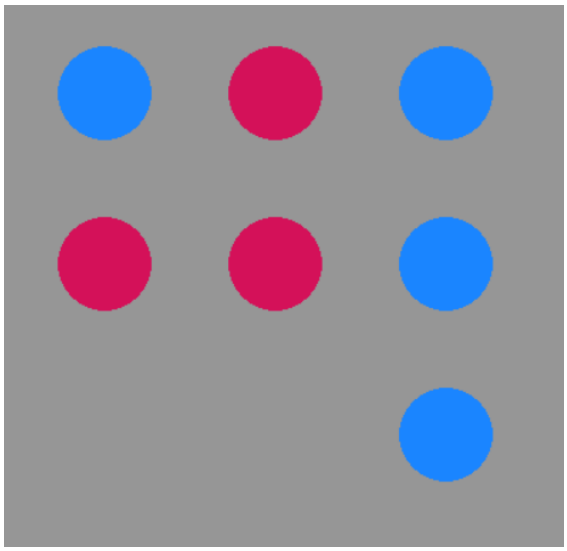
Per col·locar una fitxa en el tauler, el jugador ha de fer **`clic`** sobre una casella buida. Si el jugador fa clic en una casella buida, es col·locarà la fitxa del jugador a aquesta casella. Si el jugador fa clic en una casella fora del tauler, no es farà cap acció.

Per moure una fitxa ja col·locada, el jugador ha de **`clicar sobre la seva pròpia fitxa`**. Quan el jugador fa clic sobre una pedra que li pertany, aquesta pedra es selecciona, i el jugador pot moure-la a una casella buida.

Si el jugador fa clic sobre una pedra que **`no és seva`**, no passarà res.

La millor manera de comprovar-lo és que ho proveu!

Finalment, quan algú guanya, es desactiven les interaccions i es canvia el fons a gris per indicar que el joc ha acabat.



CONCLUSIONS

Després d'haver implementat i completat el projecte, creiem que hem aconseguit els objectius proposats: fer que el joc funcioni tant en el terminal com en una interfície gràfica amb Pygame.

Des del punt de **`vista acadèmic`**, creiem que hem après conceptes nous i hem consolidat coneixements que no teníem tan assimilats anteriorment. Per exemple, **`namedtuple`**, hem après a utilitzar aquest tipus de dada per representar informació estructurada de manera eficaç, com les coordenades de les pedres en el tauler. O també la funció **`all()`** per comprovar si algú ha guanyat, entre d'altres.

Com és normal en qualsevol projecte, ens hem trobat amb diversos obstacles. Al principi, ens va costar saber com començar el projecte, ja que havia diversos aspectes del codi que no teníem clars. Tot i això, vam poder avançar poc a poc gràcies a la recerca d'informació a Internet, preguntant a altres companyes o al professor.

Un altre problema va ser el problema amb la instal·lació de Pygame. Inicialment, vam tenir dificultats en executar el joc perquè el mòdul Pygame no estava instal·lat correctament. Després de diverses comprovacions i solucions, com desinstal·lar i tornar a instal·lar Pygame, i actualitzar la versió de **`pip`**, vam aconseguir solucionar aquest problema.

En resum, pensem que hem completat satisfactòriament aquest projecte, tot aprenent diverses eines i tècniques que no teníem tan assimilades abans.

OBSERVACIONS

Altre versió a GitHub

També hem desenvolupat una versió alternativa del projecte utilitzant **`diccionaris`** i la constant **`NO_PLAYER`**. Aquí teniu l'enllaç al nostre repositori de GITHUB. És el fitxer **`diccionaris.py`**.

https://github.com/bboscar248/PA1.git

Comprovació del funcionament amb taulers de dimensions més grans.

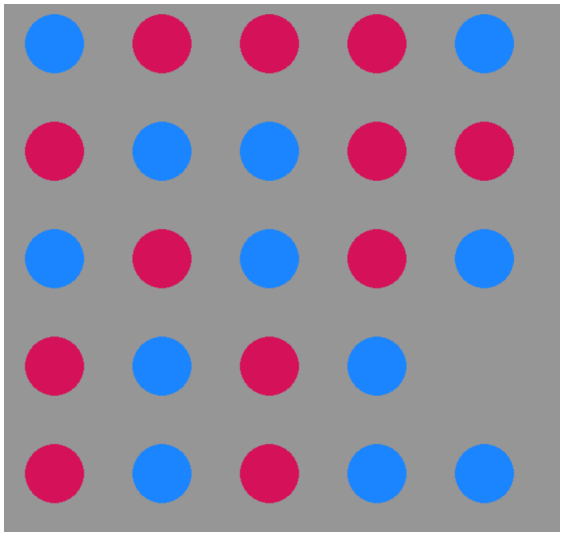
També hem provat el programa amb taulers de dimensions més grans i hem comprovat que funciona correctament, independentment de la mida del tauler. Cal destacar que, per al correcte funcionament del joc, el número total de pedres ha de ser senar, ja que això garanteix que els jugadors tinguin el mateix nombre de pedres.

Hem modificat els fitxers **`constants.py`** i **`abs_board.py`** per adaptar el programa a taulers de dimensions més grans. En el fitxer **`constants.py`**, hem actualitzat la constant **`BSIZ`** per posar les dimensions que vulguem. A més, al fitxer **`abs_board.py`**, hem ajustat la constant **`stones_per_player`** per definir el número corresponent de pedres per jugador, tenint en compte la mida del tauler.

Ex: **`BSIZ = 5 i Stones_per_player = 12`**

```
BSIZ = 5
def set_board_up(stones_per_player = 12):
```

Amb Pygame:



En el terminal: