Colegiul Național „Cantemir-Vodă”

PROIECT PENTRU OBȚINEREA ATESTĂRII PROFESIONALE ÎN INFORMATICĂ

TETRIS

Elev : Mișicu Laura - Teodora

Profesor îndrumător : Mocrienco Florentina

Mai 2021

CUPRINS

1. Prezentarea temei

Tetris este un joc video creat de Alexei Pajitnov în iunie 1985, fiind considerat în zilele noastre unul dintre cele mai faimoase jocuri retro. Acesta este un joc simplu și relaxant care are la bază 7 piese de forme diferite. Scopul jocului este acela de a completa cât mai multe linii folosind pisele furnizate aleator, jucătorul putând să le mute și să le rotească. Acesta a pierdut jocul în clipa în care a pus o piesă care depășește înălțimea spațiului de joc delimitat. Odată ce o linie este completată, aceasta este ștearsă, iar jucătorul primește punctajul aferent, însă cu cât trece mai mult timp, cu atât viteza de cădere a pieselor.

Acest joc mi-a captat atenția în copilărie: chiar dacă aveam voie să petrec o singură oră pe zi la calculator, acel timp era mai mereu umplut de încercările mele de a bate highscore-ul fratelui meu. Este un joc ce necesită o foarte bună concentrare și viteză de reacție, acest fapt făcându-l mereu o provocare interesantă. Chiar și în prezent revizitez jocul în timpul liber, iar ocazional urmăresc campionatele mondiale de Tetris.

Așadar, am ales să realizez acest proiect atât pentru a înțelege mai bine unul dinre jocurile mele preferate, cât și pentru a-mi extinde cunoștințele în limbajul de programare Python și pentru a vedea câtă muncă se află în spatele unui joc destul de simplu în aparență.

2. Resurse necesare

Datorită realizării proiectului în Python, acesta poate fi rulat atât pe Windows (minim Windows 7), cât și pe Linux. Jocul nu necesită programe auxiliare sau conexiune la internet, însă pentru rularea acestuia este nevoie de fișierul de tip text „scores.txt”. În cazul absenței sau ștergerii acestuia, va fi creat automat un nou fișier, high score-ul înregistrat repornind de la 0.

De asemenea, recomand ca resurse minime pentru funcționarea corespunzătoare a jocului:

- Procesor : Intel i3-4340
- Memorie (RAM) : 4 GB
- Spațiu liber de stocare : 11 MB

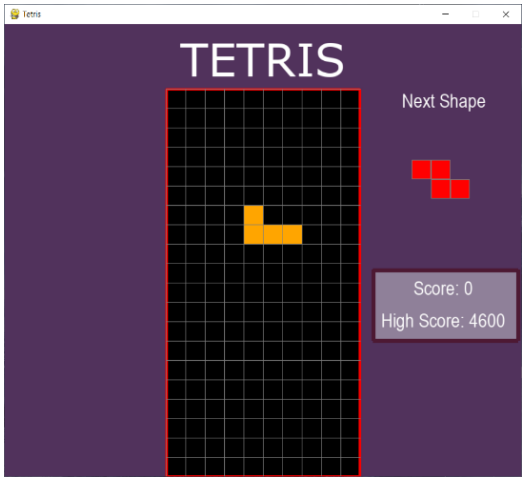
3. Minighid al utilizării aplicaţiei

La deschiderea aplicației, utilizatorului îi va fi prezentat următorul ecran. După apăsarea oricărei taste, jocul va începe.

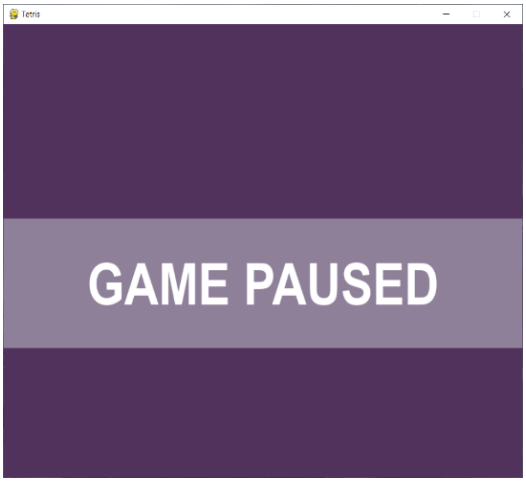
Odată cu începerea jocului, în mijlocul ecranului va fi amplasat spațiul de joc, în colțul din dreapta sus al ecranului va fi afișată piesa care urmează să cadă, iar dedesubt se vor putea observa scorul actual și cel mai mare scor înregistrat până la acel moment de către utilizator.

Tastele utilizate pe parcursul jocului sunt următoarele:

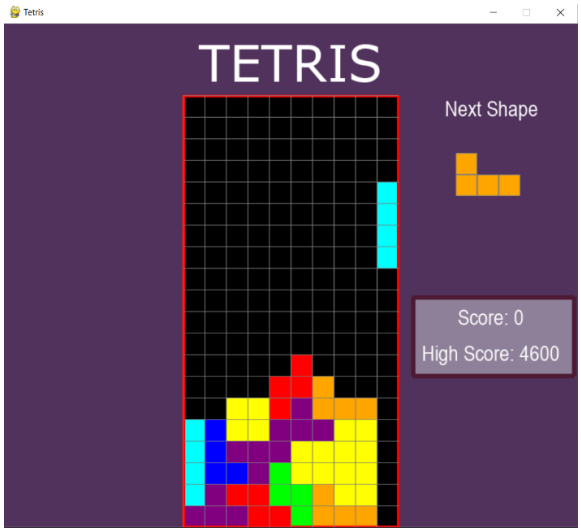
- Pentru a muta piesa pe orizontala se folosesc săgețile spre stânga și spre dreapta de pe tastatură.
- Pentru a roti piesa se folosește tasta săgeata în sus.
- Pentru a deplasa mai repede piesa în jos se folosește tasta săgeată în jos.
- Pentru a face piesa să cadă instant se folosește tasta space.
- Pentru a pune jocul pe pauză se folosește tasta „p”.

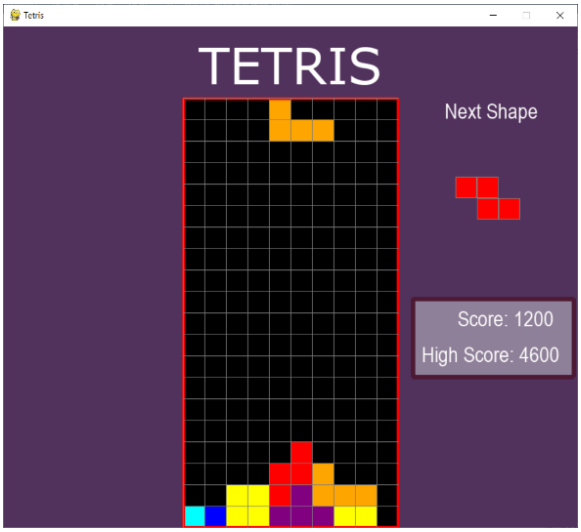


După apăsarea tastei „p”, jocul va fi pus pe pauză până la reapăsarea acesteia, iar ecranul va fi acoperit (așa cum se poate observa în imaginea alăturată) pentru a evita trișatul.



În cele două imagini de mai jos observăm un exemplu de completare a patru linii în același timp: cele 4 linii sunt înlăturate, restul liniilor se mută mai jos și punctajul aferent este adăugat la scor.

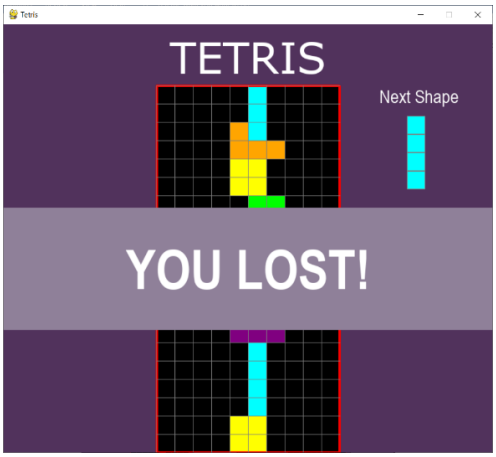




Punctajul îi este acordat jucătorului în funcție de combo-ul efectuat, acesta fiind unul dintre următoarele 4:

- Single (o linie completată): 40 puncte
- Double (două linii completate): 100 puncte
- Triple (trei linii completate): 300 puncte
- Tetris (patru linii completate): 1200 puncte

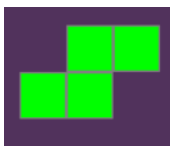
Jucătorul poate continua să adune puncte atâta timp cât completează linii în cadrul zonei de joc (delimitată cu roșu). Odată cu plasarea unei piese care depășește zona indicată, acesta a pierdut (fapt semnalat cu un mesaj ca în imaginea alăturată), punctajul va fi înregistrat ca high score în cazul în care îl depășește pe cel anterior, iar după două secunde jocul o va lua de la capăt.

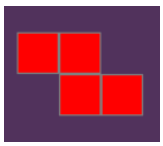


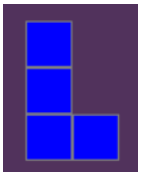
4. Realizarea concretă a aplicației

a. Piesele de joc

Pe parcursul jocului, utilizatorul va primi aleator câte una din cele șapte piese existente. Datorită formelor sale sugestive, fiecărei piese îi poate fi asociată o literă din alfabet.

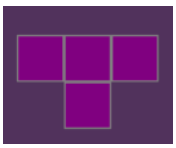

Piesa S

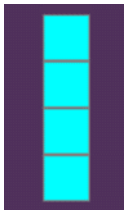

Piesa Z


Piesa L


Piesa J


Piesa O


Piesa T


Piesa I

Astfel, piesele și toate rotațiile posibile ale acestora sunt stocate în liste de matrice după modelul următor:

```
J = [['.....',
      '.0...',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..00.',
      '..0..',
      '..0..',
      '.....'],
```

```
    ['.....',
     '.....',
     '.000.',
     '...0.',
     '.....'],
    ['.....',
     '..0..',
     '..0..',
     '.00..',
     '.....']]
```

Mai apoi, acestea sunt ținute minte în lista „`shapes`”, iar culorile lor în lista „`shape_colors`”.

```
shapes = [S, Z, I, O, J, L, T]
shape_colors = [(0, 255, 0), (255, 0, 0), (0, 255, 255), (255, 255, 0),
(255, 165, 0), (0, 0, 255), (128, 0, 128)]
```

Pentru a ne referi ușor la o piesă și la atributele acesteia vom folosi clasa „`Piece`”, unde am definit următorul constructor:

```
class Piece(object):
    def __init__(self, x, y, shape):
        self.x = x
        self.y = y
        self.shape = shape
        self.color = shape_colors[shapes.index(shape)]
        self.rotation = 0
```

b. Funcțiile utilizate

Înainte de toate, acestea sunt câteva variabile globale de care ne vom folosi pe parcursul codului.

```
#dimensiunile ferestrei
s_width = 800
s_height = 700

#dimensiunile spațiului de joc
play_width = 300
play_height = 600

#dimensiunea unui pătrățel
block_size = 30

#coordonatele spațiului de joc
top_left_x = (s_width - play_width) // 2
top_left_y = s_height - play_height
```

Funcția `create_grid` este folosită pentru a reține unde se află pisele pe parcursul jocului. Acesta creează o matrice în care sunt stocate culori. Inițial toate spațiile vor fi negre, urmând să fie umplute cu culorile corespunzătoare pieselor poziționate de către jucător.

```
def create_grid(locked_positions={}):
    grid = [ [(0,0,0) for x in range(10)] for x in range(20)]

    for i in range(len(grid)):
        for j in range(len(grid[i])):
```

```
            if (j, i) in locked_positions:
                c = locked_positions[(j,i)]
                grid[i][j] = c
    return grid
```

Funcția `convert_shape_format` face trecerea piselor din stadiul de matrice formate din „0” și „ . ” la stadiul de coordonate concrete cu care putem lucra mai departe.

```
def convert_shape_format(shape):
    positions = []
    format = shape.shape[shape.rotation % len(shape.shape)]

    for i, line in enumerate(format):
        row = list(line)
        for j, column in enumerate(row):
            if column == '0':
                positions.append((shape.x + j, shape.y + i))

for i, pos in enumerate(positions):
        positions[i] = (pos[0] - 2, pos[1] - 4)

    return positions
```

Pentru a verifica dacă spațiul în care utilizatorul dorește să mute piesa este unul valid (adică se încadrează în spațiul de joc și nu se suprapune cu o piesă anterioară), folosim următoarea funcție:

```
def valid_space(shape, grid):
    accepted_pos = [[(j, i) for j in range(10) if grid[i][j] == (0,0,0)]
for i in range(20)]
    accepted_pos = [j for sub in accepted_pos for j in sub]

    formatted = convert_shape_format(shape)

    for pos in formatted:
        if pos not in accepted_pos:
            if pos[1] > -1:
                return False
    return True
```

Pierderea jocului se verifică foarte simplu: dacă o piesă este fixată depășind limita superioară a spațiului de joc, atunci utilizatorul a pierdut. Acest lucru este verificat prin funcția `check_lost`.

```
def check_lost(positions):
    for pos in positions:
        x, y = pos
        if y < 1:
            return True

    return False
```

La fel de simplu, hotărârea piesei ce urmează să cadă se face prin următoarea funcție, aceasta folosind biblioteca random.

```
def get_shape():
    return Piece(5, 0, random.choice(shapes))
```

Funcția de mai jos este folosită pentru a scrie un mesaj în mijlocul ecranului, aceasta urmând să fie apelată în cazul deschiderii jocului, al pauzei și al pierderii jocului.

```
def draw_text_middle(surface, text, size, color):
   font = pygame.font.SysFont('arial', size, bold = True)
   label = font.render(text, 1, color)

   surface.blit(label, (top_left_x + play_width/2 -(label.get_width()/2),
top_left_y + play_height/2 - (label.get_height()/2)))
```

Pentru eliberarea liniilor după completrea acestora folosim funcția `clear_rows`, descrisă mai jos. Aceasta parcurge pe linii, de jos în sus, pătrățelele spațiului de joc, iar în cazul unei linii pline (pe care nu se găsește niciun pătrățel negru) crește numărul din `inc`, urmând să deplaseze restul liniilor cu `inc` poziții mai jos. Această metodă asigură ștergerea corespunzătoare a liniilor și în cazurile în care între două linii completate se află linii necopletate.

```
def clear_rows(grid, locked):

    inc = 0
    for i in range(len(grid)-1, -1, -1):
        row = grid[i]
        full = False
        if (0,0,0) not in row:
            inc += 1
            full = True
        if inc > 0 and full == False:
            for j in range(len(row)):
                if (j , i) in locked:
                    locked[(j,i + inc)] = locked.pop((j,i))
                else:
                    locked[(j, i + inc)] = (0,0,0)
    return inc
```

Pentru afișarea în colțul din stânga sus a piesei care urmează, folosim următoarea funcție:

```
def draw_next_shape(shape, surface):
    font = pygame.font.SysFont('arial', 30)
    label = font.render('Next Shape', 1, (255, 255, 255))

    #variabile utile pentru poziționarea desenului piesei
    sx = top_left_x + play_width + 50
    sy = top_left_y + play_height - 550
    format = shape.shape[shape.rotation % len(shape.shape)]

    for i, line in enumerate(format):
        row = list(line)
        for j, column in enumerate(row):
            if column == '0':
                pygame.draw.rect(surface, shape.color, (sx + j *
block_size, sy + i * block_size, block_size, block_size), 0 ) #desenarea
piesei
                pygame.draw.rect(surface, (128, 128, 128), (int(sx + j *
30), int(sy + i * 30), 30, 30), 1) #desenarea grilei gri suprapuse pe piesă
    surface.blit(label, (sx + 15, sy - 50))
```

Schimbarea valorilor scorului actual și a high score-ului se face cu ajutorul următoarelor funcții. Este de menționat că acestea depind de fișierul „scores.txt”, însă în cazul în care acesta nu există sau a fost mutat, este creat unul nou în același loc în care se află jocul.

```
def update_score(new_score, missing_file=False):
    if missing_file == True:
        with open('scores.txt', 'w') as f:
            f.write(str(new_score))
    else:
        score = high_score()
        with open('scores.txt', 'w') as f:
            if int(score) > new_score:
                f.write(str(score))
            else :
                f.write(str(new_score))

def high_score():
    if os.path.isfile('scores.txt'):
        with open('scores.txt', 'r') as f:
            lines = f.readlines()
            score = lines[0].strip()
        return score
    else:
        update_score(0, True)
```

Funcția `draw_window` se ocupă cu afișarea în ecranul principal al jocului a elementelor esențiale acestuia (precum spațiul de joc și scorurile).

```
def draw_window(surface, grid, score=0, last_score=0):
    surface.fill((81, 50, 92))

    pygame.font.init()
    font = pygame.font.SysFont('verdana', 70)
    label = font.render('TETRIS', 1, (255, 255, 255))

    surface.blit(label, (top_left_x + play_width / 2 - label.get_width() /
2, 10))

    #desenarea casetei pentru scor
    pygame.draw.rect(surface, (143, 128, 153), (570, 380, 225, 110))
    pygame.draw.rect(surface, (77, 25, 51), (570, 380, 225, 110), 6)


    #afișarea scorului curent
    font = pygame.font.SysFont('arial', 30)
    label = font.render('Score: ' + str(score), 1, (255, 255, 255))

    sx = top_left_x + play_width + 65
    sy = top_left_y + play_height - 450

    surface.blit(label, (sx + 18, sy + 140))

    #afișarea high score-ului
    label = font.render('High Score: ' + str(last_score), 1, (255, 255,
255))

    sx = top_left_x + play_width + 15
    sy = top_left_y + play_height - 400
    surface.blit(label, (sx + 18, sy + 140))
```

```
    #creearea spațiul de joc
    for i in range(len(grid)):
        for j in range(len(grid[i])):
            pygame.draw.rect(surface, grid[i][j], (top_left_x +
j*block_size, top_left_y + i*block_size, block_size, block_size), 0)

    #indicarea cu roșu a marginii spațiului de joc
    pygame.draw.rect(surface, (250, 0, 0), (top_left_x, top_left_y,
play_width, play_height), 4)

    #suprapunerea grilei gri peste spațiul de joc
    sx = top_left_x
    sy = top_left_y

    for i in range(len(grid)):
        pygame.draw.line(surface, (128, 128, 128), (sx, sy + i *
block_size), (sx + play_width, sy + i * block_size))
        for j in range (len(grid[i])):
            pygame.draw.line(surface, (128, 128, 128), (sx + j *
block_size, sy), (sx + j * block_size, sy + play_height))
```

Jocul poate fi trecut în modul de pauză folosind tasta „p”. În acest caz, activitatea este oprită, iar ecranul este acoperit pentru a evita „pauzele de gândire”, menținând scopul jocului de a testa viteza cu care utilizatorul poate lua decizii. Funcția responsabilă pentru acest proces este descrisă mai jos.

```
def pause(surface):
    global stop
    global run
    while stop:
        surface.fill((81, 50, 92))
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.display.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_p:
                    stop = False
        pygame.draw.rect(surface, (143, 128, 153), (0, top_left_y +
play_height / 2 - 100, 800, 200))
        draw_text_middle(surface, "GAME PAUSED", 90, (255, 255, 255))
        pygame.display.update()
```

În `main` sunt puse cap la cap toate funcțiile descrise anterior. Pentru a nu îl segmenta prea mult, o să comentez funcționarea acestuia prin comentarii clasice direct pe cod (acestea sunt precedate de semnul „#”).

```
def main(win):
    last_score = high_score()
    locked_positions = { }
    grid = create_grid(locked_positions)

    change_piece = False
    run = True
    current_piece= get_shape()
    next_piece = get_shape()
    clock = pygame.time.Clock()
    fall_time = 0
```

```
    fall_speed = 0.5
    level_time = 0
    score = 0
    global stop

    while run:
        grid = create_grid(locked_positions)

        #asigurăm aceeași viteză de cădere a pieselor indiferent de
performanța calcuatorului (folosirea de fps ar fi putut duce la diferențe)
        fall_time += clock.get_rawtime()
        level_time += clock.get_rawtime()
        clock.tick()

        #creștem treptat viteza cu care cad piesele
        if level_time/1000 > 5v
            level_time = 0
            if fall_speed > 0.12:
                fall_speed -= 0.005

        #efectuăm căderea piesei și trecerea la următoarea în cazul în care
aceasta nu mai are unde să cadă
        if fall_time/1000 > fall_speed:
            fall_time = 0
            current_piece.y += 1
            if not(valid_space(current_piece, grid)) and current_piece.y>0:
                current_piece.y -= 1
                change_piece = True

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.display.quit()
                sys.exit()

            #pentru a mișca piesele folosim următorea secvență de cod,
verificând la fiecare mișcare că spațiul dorit este unul valid
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    current_piece.x -=1
                    if not(valid_space(current_piece, grid)):
                        current_piece.x +=1
                if event.key == pygame.K_RIGHT:
                    current_piece.x +=1
                    if not(valid_space(current_piece, grid)):
                        current_piece.x -=1
                if event.key == pygame.K_DOWN:
                    current_piece.y +=1
                    if not(valid_space(current_piece, grid)):
                        current_piece.y -=1
                if event.key == pygame.K_UP:
                    current_piece.rotation +=1
                    if not(valid_space(current_piece, grid)):
                        current_piece.rotation -=1
                if event.key == pygame.K_SPACE:
                    while current_piece.y < 21 and
(valid_space(current_piece, grid)):
                        current_piece.y += 1
                    if not (valid_space(current_piece, grid)):
                        current_piece.y -= 1
                if event.key == pygame.K_p:
```

```
                    stop = True
                    pause(win)
        shape_pos = convert_shape_format(current_piece)

        #umplerea matricei cu culorile corespunzătoare piesei actuale
        for i in range(len(shape_pos)):
            x, y = shape_pos[i]
            if y > -1:
                grid[y][x] = current_piece.color

        #trecerea la o nouă piesă
        if change_piece:
            for pos in shape_pos:
                p = (pos[0], pos[1])
                locked_positions[p] = current_piece.color
            current_piece = next_piece
            next_piece = get_shape()
            change_piece = False

            #calcularea punctajului în funcție de numărul de linii
completate
            combo = clear_rows(grid, locked_positions)
            if combo == 1:
                score += 40
            if combo == 2:
                score += 100
            if combo == 3:
                score += 300
            if combo == 4:
                score += 1200

        draw_window(win, grid, score, last_score)
        draw_next_shape(next_piece, win)
        pygame.display.update()

        #pe parcursul jocului verificăm constant că jucătorul încă nu a
pierdut, iar în caz contrar, utilizatorul este anunțat ca a pierdut și la
scurt timp, jocul reîncepe
        if check_lost(locked_positions):
            pygame.draw.rect(win, (143, 128, 153), (0, top_left_y +
play_height/2 - 100, 800, 200))
            draw_text_middle(win, "YOU LOST!", 90, (255, 255, 255))
            pygame.display.update()
            pygame.time.delay(2000)
            run = False
            update_score(score)
```

Iar nu în cele din urmă, funcția `main_menu` creează ecranul de start, acesta apărând și după fiecare pierdere a jocului.

```
def main_menu(win):
    run = True
    while run:
        win.fill((0,0,0))
        draw_text_middle(win, 'Press any key to start', 80, (255, 255,
255))
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
```

```
            if event.type == pygame.KEYDOWN:
                main(win)

    pygame.display.quit()
```

5. Posibile extinderi ale aplicației

Consider că, deși jocul în forma sa actuală este bine făcut și oferă o experiență plăcută, acestuia i se pot aduce multiple îmbunătățiri, atât la nivel vizual, cât și la nivel funcțional.

Jocul are aspect clasic, respectând schematica standard de culori pentru piese, însă, pe viitor, pot fi adăugate mai multe opțiuni de culori sau modele dintre care utilizatorul să poată alege sau mai multe fundaluri pentru utilizatorii care consideră că un model mai complex nu i-ar distrage.

Pe lângă actualizările vizuale care pot fi aduse jocului, o funcție utilă care mai poate fi implementată este opțiunea de indicator al locului unde urmează să cadă piesa, acest fapt scăzând dificultatea jocului prin ușurarea alinierii pieselor. Această adiție ar crește viteza jucătorului, permițând continuarea jocului pe o perioadă mai lungă. De asemenea, un mod de joc de tip „Quick play”, în care jucătorul trebuie să adune cât mai multe puncte într-o anumită limită de timp, ar fi potrivit pentru persoanele care doar doresc să se distreze rapid, într-o pauză, fără a investi prea mult timp în joc.

6. Considerații finale

Realizarea acestui proiect a fost o experiență interesantă și o mare oportunitate de învățare. Deși începutul a fost puțin greu deoarece nu eram familiarizată cu limbajul de programare Python, acesta are o structură intuitivă pentru începători, iar cunoștiințele de C/C++ dobândite pe timpul liceului au ușurat mult procesul. De asemenea, un astfel de proiect practic te motivează să încerci până vezi rezultate, iar toate orele de chin pentru rezolvarea erorilor devin complet meritate odată ce ai ajuns la varianta finală.

7. Bibliografie

https://en.wikipedia.org/wiki/Tetris

https://tetris.fandom.com/wiki/Scoring

https://www.w3schools.com/python/default.asp

https://www.youtube.com/watch?v=8zXlWbEgfiY

https://www.youtube.com/watch?v=zfvxp7PgQ6c

https://datatofish.com/executable-pyinstaller/