2026

ATOLL PROJEKAT

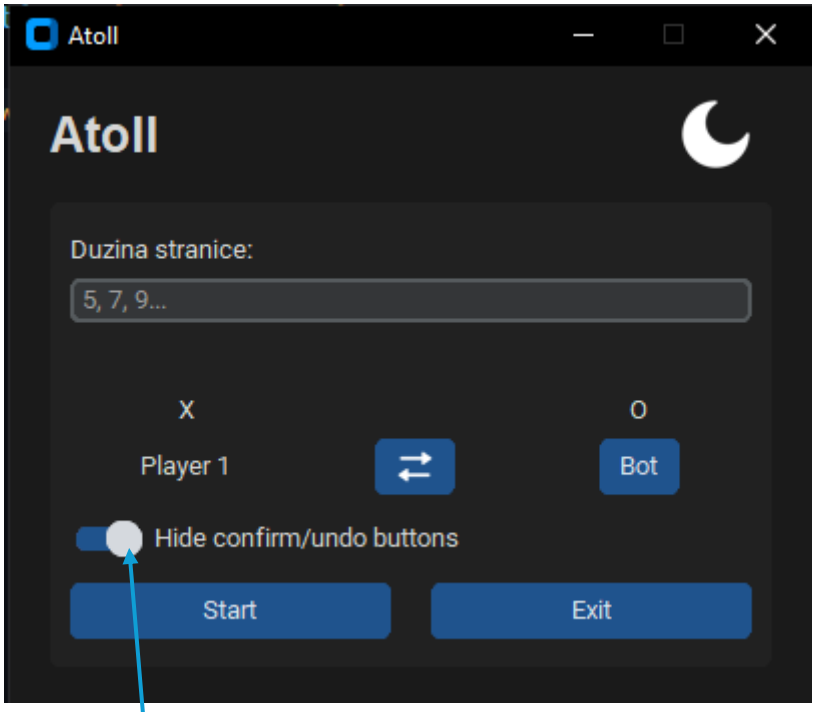
DRUGA FAZA

STEFAN CVETKOVIC 19461 | LAZAR STANKOVIĆ 19375

Contents

IZGLED IGRE

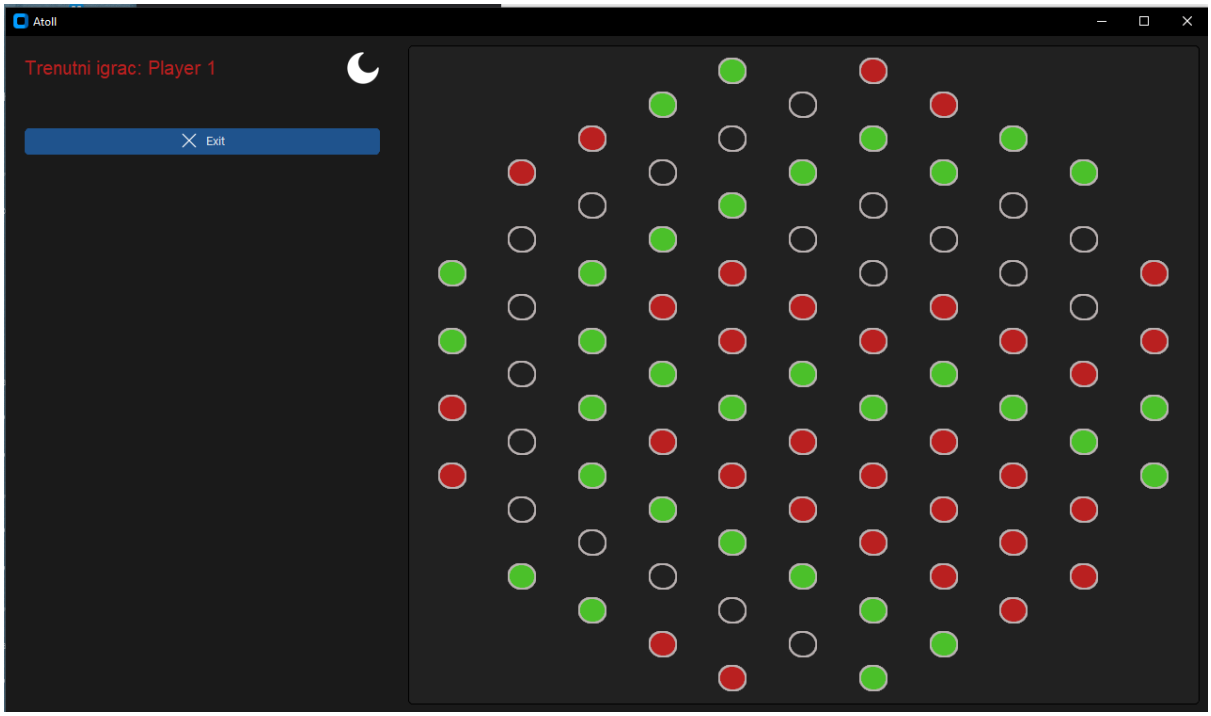
Početni prozor



Sakviranje confirm/undo dugmića na tabli

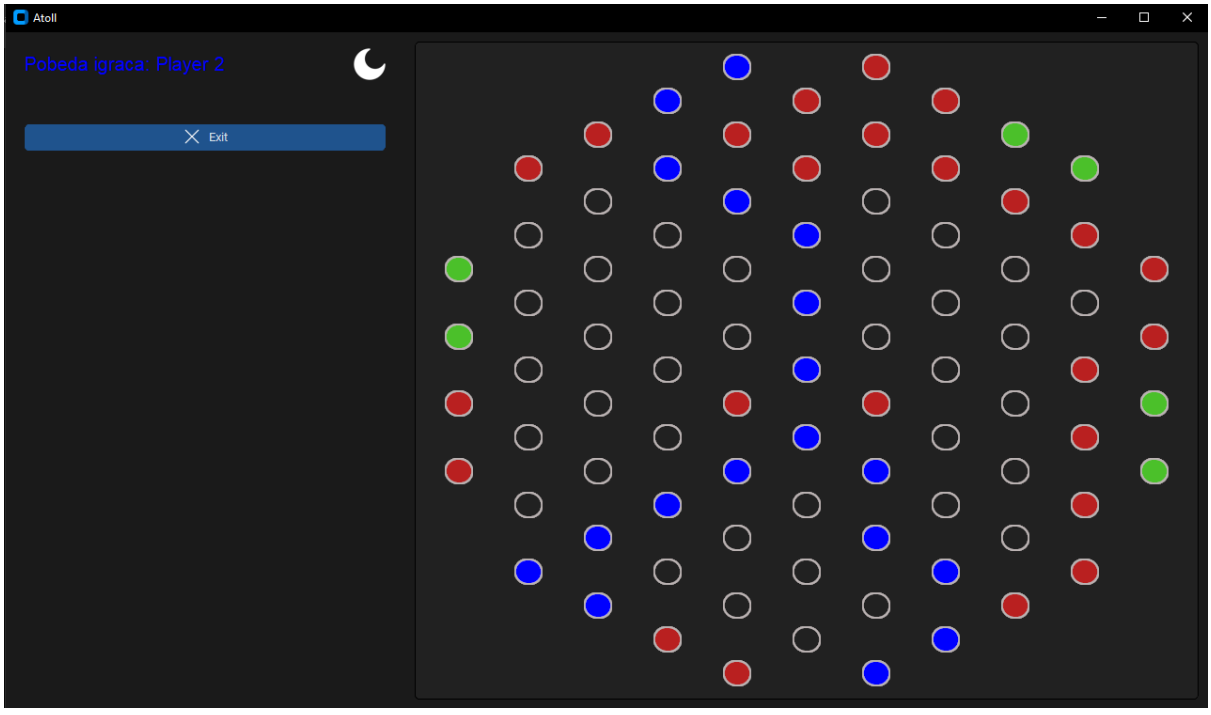
Tabla za igranje

Skriveni confirm/undo dugmići



Automatski se vrši potvrda klikom na polje, gubi se mogućnost poništavanja poteza ali se gubi potreba za potvrdom svakog poteza.

Izgled pobede



FUNKCIJE IGRE

enums.py

```
class eTile(Enum):
    Invalid = -1,
    Playable = 0,
    Player1 = 1,
    Player2 = 2,
    Connected = 3,
```

Dodata vrednost “Connected” koja se koristi za pamćenje povezanih polja u algoritmu provere.

main.py

```
switchVar = customtkinter.StringVar(value = 'off')
    switchEl = customtkinter.CTkSwitch(buttonsFrame, text='Hide confirm/undo
buttons', variable=switchVar, onvalue='on', offvalue='off')
    switchEl.grid(row = 0, column = 0, padx=10, columnspan=2, sticky='w')
```

Dodat “switch” (checkbox dugme) na početnoj stranici koji sakriva confirm i undo dugmice na stranici za igru.

```
hideButtons = switchVar.get() == 'on'
openGame(n, isBot, swapped, app, customtkinter.get_appearance_mode().lower(),
hideButtons)
```

Kod koji uzima vrednost switch-a i salje je funkciji za pokretanje igre.

board.py

Sakrvianje confirm/undo dugmića

```
if hideButtons:
        confirmButton.grid_forget()
        undoButton.grid_forget()
```

Ukoliko je kao argument poslat hideButtons poziva se funkcija grid_forget() koja sakriva elemente.

```
if hideButtons:
        confirmClick()
    else:
        print(f"{getCurrentPlayerName()} je odabrao: {chr(ord('A') - 1 + c)}
{(r + c - n + 1)//2}")
```

Ovaj kod se izvršava pri kliku na polje, ukoliko je poslat hideButtons automatski poziva funkciju za potvrdu odigranog polja, u suprotnom štampa kako je igrač odabrao polje.

Kreiranje grafova grupa

```
def startPage():
    global buttons
    global matrix
    global groups
    matrix, matrixN, matrixM = createTable(n)
    groups = createGraphs(matrix, matrixN, matrixM)
```

Funkcija se poziva nakon kreiranja matrice polja.

createGraphs(matrix, matrixN, matrixM)

```
hexDirections = [(-2, 0), (-1, 1), (1, 1), (2, 0), (1, -1), (-1, -1)]
def createGraphs(matrix, matrixN, matrixM):
    groups = {}
    currentGroup = 1
    processed = set()

    directions = [(1, -1), (2, 0), (1, 1)]

    for i in range(matrixN):
        for j in range(matrixM):
            if matrix[i][j] in [eTile.Invalid.value[0],
eTile.Playable.value[0]] or (i, j) in processed:
                continue

            currentTile = matrix[i][j]
            processed.add((i, j))
            group = [(i, j)]

            for di, dj in directions:
                ni = i + di
                nj = j + dj
                while 0 <= ni < matrixN and 0 <= nj < matrixM:
                    if matrix[ni][nj] != currentTile:
                        break

                    processed.add((ni, nj))
                    group.append((ni, nj))
                    ni += di
                    nj += dj

            groups[currentGroup] = (group, currentTile)
            currentGroup += 1

    return groups
```

Funkcija koja kreira dictionary koji čuva grupe, tačnije ostvra, na samoj tabeli igre. Funkcija prolazi kroz matricu nakon kreiranja nje. Na samom početku je matrica prazna tako da su polja igrača samo početne grupe ostvra. Nakon što funkcija naiđe na prvo polje, ona ide dijagonalno i dole dok ne naiđe na polje koje se ne slaže. Nakon pronalaska polja koje se ne slaže znamo da smo došli do kraja tog ostrva. Funkcija pamti sva polja kroz koja je prošla kako ne bi kreirala duplikate.

Provera kraja igre

```
def confirmClick():
    global currentPlayerLabel, lastMove, currentPlayer, n
    print(f"{getCurrentPlayerName()} je odigrao: {chr(ord('A') - 1 +
lastMove[1])} {(lastMove[0] + lastMove[1] - n + 1)//2}")
    status, board = isGameFinished()
```

Kraj igre se provereva nakon potvrđenja odigranog poteza. Kraj igre se proverava pozivom funkcije isGameFinished()

```
    if status == True or isGameDraw():
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == eTile.Connected.value[0]:
                    buttons[i][j].configure(fg_color = 'blue')
                if buttons[i][j] is not None:
                    buttons[i][j].configure(state = 'disabled')
                    if status != True:
                        buttons[i][j].configure(fg_color = 'yellow')

        print(f'Pobeda igraca: {getCurrentPlayerName()}' if status == True
else 'Igra je neresena')
        if status == True:
            currentPlayerLabel.configure(text = f'Pobeda igraca:
{getCurrentPlayerName()}', text_color= 'blue')
        else:
            currentPlayerLabel.configure(text = 'Igra je neresena',
text_color= 'yellow')
    else:
        currentPlayer = (currentPlayer + 1) % 2
        lastMove = None
        currentPlayerLabel.configure(text = getCurrentPlayer(), text_color=
playerColors[currentPlayer])
```

Ukoliko je meč završen ili nerešen, svako dugme se gasi. Dugmići koji prave put od jednog ostvra do drugog se boje u plavu boju. U slučaju kada je igra nerešena svako polje se boji u žutu boju. Label koji prikazuje koji igrač igra se pretvara u label koji pokazuje koji igrač je pobedio ili obaveštava da je bilo nerešeno.

isGameFinished()

```
def isGameFinished():
    processedGroups = set()
    for group in groups:
        ni, nj = groups[group][0][0]
        player = matrix[ni][nj]
        if player != players[currentPlayer] or group in processedGroups:
            continue

        boardCopy = [row[:] for row in matrix]
        dfs(boardCopy, ni, nj, player, eTile.Connected.value[0])
        processedGroups.add(group)

        connectedGroups = getConnectedGroups(group, boardCopy)
        lowest = iterateThroughGroups(connectedGroups, boardCopy)

        for conGroup in connectedGroups:
            processedGroups.add(conGroup)

        if lowest >= len(groups) / 2 + 1:
            return (True, boardCopy)

    return (False, [])
```

Funkcija koja nam vraća da li je igra završena. Funkcija proverava samo polja trenutnog igrača kako ne bi svakog puta proveravala sva ostrva. Funkcija prolazi kroz grupe ostrva i izvršava DFS popunjavanje kopije matrice polja. Koristi se **FloodFill** algoritam kako bi se popunila sva polja koja se spajaju sa početnom tačkom. Na taj način kreiramo put između svih polja čime možemo da proverimo samo rezultat igre. Nakon kreiranja tabele se vraćaju sva ostvra koja su povezana i za njih se vraća najmanji put. Ako je taj put manji od **d = m / 2 + 1** onda igra nije završena.

dfs(board, x, y, oldTile, newTile)

```
def dfs(board, x, y, oldTile, newTile):
    if (x < 0 or x >= len(board) or
        y < 0 or y >= len(board[0]) or
        board[x][y] != oldTile):
        return

    board[x][y] = newTile
    for hdi, hdj in hexDirections:
        dfs(board, x+hdi, y+hdj, oldTile, newTile)
```

Rekurzivna funkcija koja prolazi kroz svaka polja koja dodiruju trenutno polje i menja im vrednost u enum Connected.

getConnectedGroups(group, board)

```
def getConnectedGroups(group, board):
    res = []
    for curGroup in groups:
        if groups[curGroup][1] != groups[group][1] or curGroup == group:
            continue

        x, y = groups[curGroup][0][0]
        if board[x][y] != eTile.Connected.value[0]:
            continue

        res.append(curGroup)

    return res
```

Funkcija koja vraća listu ostvra koja su povezana. S obzirom na to da FloodFill boji sva polja koja se dodiruju ovo takođe boji i početna ostrva. Zbog toga nema potrebe da prolazimo kroz prvi krug tabele.

iterateThroughGroups(connectedGroups, board)

```
def iterateThroughGroups(connectedGroups, board):
    lowest = -1
    orderedGroups, reverseOrderGroups = getOrderedGroups()
    for connectedGroup in connectedGroups:
        startIndex = orderedGroups.index(connectedGroup)
        reverseStartIndex = reverseOrderGroups.index(connectedGroup)

        firstOrderedGroups = orderedGroups[startIndex:] +
orderedGroups[:startIndex]
        reverseOrderGroups = reverseOrderGroups[reverseStartIndex:] +
reverseOrderGroups[:reverseStartIndex]
        curLowest = min(processGroup(firstOrderedGroups, board),
processGroup(reverseOrderGroups, board))
        if lowest == -1:
            lowest = curLowest

        lowest = min(lowest, curLowest)

    return lowest
```

Funkcija koja prolazi kroz grupe spojenih ostvra. Preko funkcije **getOrderedGroups** dobija niz ostvra poredjan po obodnom putu. Nakon toga funkcija prolazi kroz sve spojene grupe i za svaku od njih pronalazi sebe u nizu. Nakon toga pravi niz koji počinje od tog ostrva i ide do kraja. Pored tog niza takođe i pravi obrnuti niz koji će se koristi za suprotan smer. Nakon toga poziva funkciju koja precosuira grupe i vraća broj ostvra koja su povezana obodnim putem. Uzima minimalnu vrednost i vraća je.

getOrderedGroups()

```
def getOrderedGroups():
    orderedGroups = [1]
    groupAmount = len(groups)
    orderedGroups.extend(range(2, groupAmount + 1, 2))
    orderedGroups.extend(range(groupAmount - 1, 2, -2))

    return orderedGroups, orderedGroups[::-1]
```

Vraća niz obodnih puteva u oba smera.

processGroup(orderedGroups, board)

```
def processGroup(orderedGroups, board):
    currentIslandCount = 0
    finalIslandCount = 0

    for group in orderedGroups:
        currentIslandCount += 1
        x, y = groups[group][0][0]

        if board[x][y] == eTile.Connected.value[0]:
            finalIslandCount = currentIslandCount

    return finalIslandCount
```

Funkcija koja procesira ostvra i vraća broj povezanih ostvra po određenom putu. Kao parametar prima niz koji se prethodno kreirao i koji počinje sa prvim ostrvom od kojeg krećemo. On broji svako ostvro od početnog ostrva. Ukoliko je jedno od ostrva do kojeg je stigao **Connected** onda znamo da je sve pre toga bilo povezano. Zbog toga možemo na ovaj način odrediti dužinu puteva po obodnim ostrvima. Nakon završetka niza vraćamo poslednju vrednost do koje je stala petlja kada je naišla na poslednje povezano ostrvo.

isGameDraw()

```
def isGameDraw():
    for row in matrix:
        if eTile.Playable.value[0] in row:
            return False

    return True
```

Proverava da li je rezultat nerešen. Ovo nije tehnički sam rad funkcije jer ona proverava da li je tabla skroz popunjena. Svakako u samom rešenju služi za to jer se koristi nakon što nam funkcija vrati da nije došlo do kraja. Ukoliko znamo da nije došlo do dobrog povezivanja ostrva i da je sve popunjeno onda je automatski rezultat nerešen.

INFORMACIJE O IGRI

Tehnologije

Za pronalaženje puteva je korišćen algoritam **FloodFill** koji pravi puteve između povezanih ostvra. Ovaj algoritam kao bazu koristi **DFS** algoritam (**Depth First Search**) koji je u ovom slučaju rađen preko rekurzije.

Reference

Implementacija za FloodFill - https://www.geeksforgeeks.org/dsa/flood-fill-algorithm/