

ALGORİTMA VE PROGRAMLAMA I

**Kare Kapan
Oyunu Simülasyonu**

**Enes Adışen
05210000940**
**Barış Mehmet Balcı
05210000933**

içindekiler

◆ Programcı kataloğu

- Fonksiyonlar/modüller
- Projenin üretim aşamaları

◆ Kullanıcı katlogu

- Kare kapan oyunu
- Kullanıcı talimatları

Programcı kataloğu\ fonksiyonlar

```
def start_game():
    row_number = int(input("Please enter the number of horizontal lines [3,7]:"))
    while row_number < 1 or row_number > 7:
        print("Please enter a number between 3 and 7")
        row_number = int(input("Please enter the number of horizontal lines [3,7]:"))
    column_number = row_number + 1
    return row_number, column_number
```

start_game():

Oyunun oynanacağı tahtanın büyülüüğünü kullanıcıdan alır ve satır/sütun sayısını döndürür.

```
def input_translation(input_o):
    dict_x, dict_y = import_dict()
    x = dict_x[input_o[0]]
    y = dict_y[input_o[1]]
    output = f"{x}{y}"
    return output
```

```

def import_dict():
    dict_y = {
        "A": "0",
        "B": "1",
        "C": "2",
        "D": "3",
        "E": "4",
        "F": "5",
        "G": "6",
        "H": "7"
    }
    dict_x = {
        "1": "0",
        "2": "1",
        "3": "2",
        "4": "3",
        "5": "4",
        "6": "5",
        "7": "6"
    }
    return dict_x, dict_y

```

import_dict():
Kullanıcının girdiği koordinatlar ile listelerin indekslerini sözlük yardımıyla eşleştirir.
Oluşan sözlükleri döndürür.

```

def input_division(initial_input):
    output_one = f"{initial_input[0]},{initial_input[1]}"
    output_two = f"{initial_input[3]},{initial_input[4]}"
    output_one_translated = input_translation(output_one)
    output_two_translated = input_translation(output_two)
    return output_one_translated, output_two_translated

```

input_division(initial_input):
Kullanıcıdan alınan birden fazla girişlerin ayrıstırılıp programa uygun çevrilmesini sağlar.

```
def check_space(pieces):
    is_board_empty = False
    a = 0
    while a < len(pieces):
        if 0 in pieces[a]:
            is_board_empty = True
        a += 1
    return is_board_empty
```

check_space(pieces):
Tahtada boş bir nokta olup olmadığını inceler.
Bir boolean döndürür.

```
def check_specific_space(pieces, selection):
    x = int(selection[0])
    y = int(selection[1])
    return pieces[x][y]
```

check_space(pieces, selection):
Parametre olarak taşların listesini ve seçilen
noktayı alır. Seçili bir noktanın hangi taşı
ait veya boş olduğunu döndürür.

```
def set_pieces(row, column):  
    initial_pieces = [0] * row  
    for i in range(row):  
        initial_pieces[i] = [0] * column  
    return initial_pieces
```

set_pieces(row, column):

Parametre olarak satır ve sütun sayılarını alır.
Taşların barınacağı boş listeyi oluşturup geri döndürür.

```
def insert_piece(the_piece, pieces, color):  
    x = int(the_piece[0])  
    y = int(the_piece[1])  
    if pieces[x][y] == 0:  
        pieces[x][y] = color
```

insert_piece(the_piece, pieces, color):

Parametre olarak noktanın koordinatını (the_piece),
taşların listesini (pieces) ve hangi renge ait
olduğunu (color) alır. İstenen koordinata taşı yerleştirir.

```
def remove_piece(the_piece, pieces):
    # removes a piece from the board.
    # the_piece: piece to be removed , pieces: list of all pieces.
    x = int(the_piece[0])
    y = int(the_piece[1])
    if pieces[x][y] != 0:
        pieces[x][y] = 0
```

remove_piece(the_piece, pieces):

Parametre olarak noktanın koordinatını (the_piece) ve taşların listesini (pieces) alır. İstenen koordinattan taşı kaldırır.

```
def count_squares(pieces, color):
    # Takes the positions and the player as parameter, returns an integer (number of squares) back.
    square_count = 0
    for i in range(len(pieces)):
        for k in range(len(pieces[i])):
            try:
                if pieces[i][k] == pieces[i][k + 1] == pieces[i + 1][k] == pieces[i + 1][k + 1] == color:
                    square_count += 1
            except:
                pass
    return square_count
```

count_squares(pieces, color):

Parametre olarak taşların listesini (pieces) ve taşların rengini (color) alır. Toplam oluşan kare sayısını döndürür.

```
def is_square(pieces, the_piece):
    # Checks if a piece belongs to a square or not.
    check = False
    x = int(the_piece[0])
    y = int(the_piece[1])
    try:
        if pieces[x][y] == pieces[x][y + 1] == pieces[x + 1][y] == pieces[x + 1][y + 1]:
            check = True

    except:
        pass
    try:
        if pieces[x - 1][y] == pieces[x - 1][y + 1] == pieces[x][y] == pieces[x][y + 1]:
            check = True

    except:
        pass
    try:
        if pieces[x][y - 1] == pieces[x][y] == pieces[x + 1][y - 1] == pieces[x + 1][y]:
            check = True

    except:
        pass
    try:
        if pieces[x - 1][y - 1] == pieces[x - 1][y] == pieces[x][y - 1] == pieces[x][y]:
            check = True

    except:
        pass

    return check
```

is_square(pieces, the_piece):
Parametre olarak taşların listesini (pieces) ve ilgili koordinatı (the_piece) alır. Koordinatta bulunan taşın bir kareye ait olup olmadığını döndürür.

```

def draw_board(a_pieces):
    pieces = list(a_pieces)
    row_number = len(pieces)
    column_number = row_number + 1
    dict = {
        1: "A",
        2: "B",
        3: "C",
        4: "D",
        5: "E",
        6: "F",
        7: "G",
        8: "H"
    }
    space = " "
    twospace = " "
    bigspace = " "
    for i in range(1, row_number + 2):
        if i != 1:
            for element in range(1, column_number + 1):
                if element == column_number:
                    print(pieces[i - 2][element - 1], end="", sep='')
                elif element != 1:
                    print(pieces[i - 2][element - 1], "---", end="", sep='')
                else:
                    print(space, pieces[i - 2][element - 1], "---", end="", sep='')
        if i == 1:
            for element in range(1, column_number + 1):
                if element != 1:
                    print(bigspace, dict[element], end="", sep='')
                else:
                    print(twospace, dict[element], end="", sep='')

        if i != row_number + 1:
            print("")
            if i != 1:
                line = "|"
                print(twospace + line * (row_number + 1))
            print(i, end="")
        if i == row_number + 1:
            pass
    print("")

```

draw_board(a_pieces):
Parametre olarak
taşların listesini alır
ve listeye uygun olarak
taşları koordinatlarla
birlikte yazdırır.

```
def initial_insertion_phase(pieces):
    player_turn = 1
    while check_space(pieces) == True:
        if player_turn == 1:
            draw_board(pieces)
            selection = input("Select an empty space for your piece (Player One):")
            selection = input_translation(selection.upper())
            while check_specific_space(pieces, selection) != 0:
                selection = input("Select an empty space for your piece (Player One):")
                selection = input_translation(selection.upper())
            insert_piece(selection, pieces, "W")
            player_turn += 1
        else:
            draw_board(pieces)
            selection = input("Select an empty space for your piece (Player Two):")
            selection = input_translation(selection.upper())
            while check_specific_space(pieces, selection) != 0:
                selection = input("Select an empty space for your piece (Player Two):")
                selection = input_translation(selection.upper())
            insert_piece(selection, pieces, "B")
            player_turn -= 1
```

initial_insertion_phase(pieces):
Parametre olarak taşların listesini alır ve diğer fonksiyonları kullanarak yerleştirme aşamasını oynatır.

```
def is_there_any_piece_to_take(pieces, color):
    can_take = False
    non_square_piece = 0
    for i in range(len(pieces)):
        for j in range(len(pieces[i])):
            if pieces[i][j] == color:
                piece = f"{i}{j}"
                if is_square(pieces, piece) == False:
                    non_square_piece += 1
    if non_square_piece > 0:
        can_take = True
    return can_take
```

is_there_any_piece_to_take(pieces, color):

Sırası gelen oyuncunun kaç tane rakip taşını dışarı çıkarabileceğini bulur. Eğer hiç taş yoksa False, varsa True döndürür.

```

def piece_removal_by_player(pieces, color, piece_amount=1):
    how_many_removes = piece_amount
    if color == "W":
        player = "one"
        opponent = "B"
    else:
        player = "two"
        opponent = "W"
    can_take = is_there_any_piece_to_take(pieces, opponent)
    if can_take == False:
        print(f"Player {player} cant take any piece!")
    else:
        if how_many_removes > 0:
            while 0 < how_many_removes:
                if how_many_removes == 1:
                    is_it_plural = "piece"
                else:
                    is_it_plural = "pieces"
                draw_board(pieces)
                print(f"Player {player} can remove {how_many_removes} {is_it_plural}!")
                is_it_square = False
                is_it_opponents = False
                while is_it_opponents == False or is_it_square == False:
                    selection = input(
                        f"Please select a piece which isn't a part of a square to remove from opponent (Player {player}):")
                    selection = input_translation(selection.upper())
                    if check_specific_space(pieces, selection) == color or check_specific_space(pieces, selection) == 0:
                        is_it_opponents = False
                        print("Selected space doesn't have any piece of opponent's.")
                    else:
                        is_it_opponents = True
                    if is_square(pieces, selection):
                        is_it_square = False
                        print("Selected space has a piece which is a part of a square.")
                    else:
                        is_it_square = True
                    remove_piece(selection, pieces)
                    how_many_removes -= 1
                else:
                    print(f"Player {player} used all of his/her removals!")
            else:
                print(f"Player {player} has no square!")

```

piece_removal_by_player(pieces, color ,piece_amount=1)
Parametre olarak taşların listesini (pieces), rengi (color) ve kaldırılacak taş sayısını alır. Taş kaldırma aşamasını diğer fonksiyonları kullanarak oynatır.

```
def piece_movement(pieces, the_piece, where_to, color):
    remove_piece(the_piece, pieces)
    insert_piece(where_to, pieces, color)
```

piece_movement(pieces, the_piece, where_to, color):
Parametre olarak taşların listesini (pieces), ilgili koordinatı (the_piece), gidilecek koordinatı (where_to) ve rengi (color) alır. insert_piece ve remove_piece fonksiyonlarını kullanarak taşı hareket ettirir.

```
def how_many_pieces_left(pieces):
    whites_amount = 0
    blacks_amount = 0
    for i in range(len(pieces)):
        for j in range(len(pieces[i])):
            piece = f"{i}{j}"
            if check_specific_space(pieces, piece) == "W":
                whites_amount += 1
            elif check_specific_space(pieces, piece) == "B":
                blacks_amount += 1
            else:
                pass
    return whites_amount, blacks_amount
```

how_many_pieces_left(pieces):

Tüm koordinatları kontrol ederek iki oyuncunun kaç tane toplam taşı kaldığını geri döndürür.

```
def can_move(pieces, the_piece):
    can_move = True
    count = 0
    x = int(the_piece[0])
    y = int(the_piece[1])
    up = f"{x - 1}{y}"
    down = f"{x + 1}{y}"
    left = f"{x}{y - 1}"
    right = f"{x}{y + 1}"
    if x == 0:
        up = f"{x}{y}"
    elif x == len(pieces) - 1:
        down = f"{x}{y}"
    if y == 0:
        left = f"{x}{y}"
    elif y == len(pieces[0]) - 1:
        right = f"{x}{y}"
    if pathway_check(pieces, the_piece, up) == False:
        count += 1
    if pathway_check(pieces, the_piece, down) == False:
        count += 1
    if pathway_check(pieces, the_piece, left) == False:
        count += 1
    if pathway_check(pieces, the_piece, right) == False:
        count += 1
    if count == 4:
        can_move = False
    return can_move
```

can_move(pieces, the_piece):

Parametre olarak taşların listesini (pieces), ilgili koordinatı (the_piece) ve gidilecek koordinatı (where_to) alır. Taşın istenilen hareketi yapmasında bir engel olup olmadığını boolean olarak döndürür.

```

def gameplay_phase(pieces, color):
    draw_board(pieces)
    if color == "W":
        player = "one"
    else:
        player = "two"
    available = False
    movable = False
    while available == False or movable == False:
        initial_input = input(f"Please select a piece and a position to move (Player {player}):")
        initial_input = initial_input.upper()
        selection, where_to = input_division(initial_input)
        if check_specific_space(pieces, selection) != color or can_move(pieces, selection) == False:
            print("Please select your own available pieces!")
        else:
            available = True
        if pathway_check(pieces, selection, where_to) == False:
            print("Please select a movable space!")
        else:
            movable = True
    piece_movement(pieces, selection, where_to, color)
    if is_square(pieces, where_to) == True:
        piece_removal_by_player(pieces, color)
    else:
        piece_amount = 0
        piece_removal_by_player(pieces, color, piece_amount)

```

gameplay_phase(pieces,color):
Parametre olarak taşların listesini (pieces) ve rengi (color) alır. İlk yerleştirme ve taş kaldırma aşaması bittikten sonra oyunu devam ettirir

```

def main():
    player_turn = 1
    pieces = set_pieces(*start_game())
    initial_insertion_phase(pieces)
    piece_removal_by_player(pieces, "W", count_squares(pieces, "W"))
    piece_removal_by_player(pieces, "B", count_squares(pieces, "B"))

    if stuck_at_start(pieces) == True:
        print("Both player got stuck at the beggining. Player one will take first piece.")
        is_it_opponents = False
    while is_it_opponents == False:
        draw_board(pieces)
        selection = input(f"Please select a piece remove from opponent (Player one):")
        selection = input_translation(selection.upper())
        if check_specific_space(pieces, selection) == "W" or check_specific_space(pieces, selection) == 0:
            is_it_opponents = False
            print("Selected space doesn't have any piece of opponent's.")
        else:
            is_it_opponents = True
        remove_piece(selection, pieces)
        player_turn += 1

    black_pieces = 4
    white_pieces = 4
    while black_pieces > 3 and white_pieces > 3:
        if player_turn == 1:
            color = "W"
            player_turn += 1
        else:
            color = "B"
            player_turn -= 1
        gameplay_phase(pieces, color)
        white_pieces, black_pieces = how_many_pieces_left(pieces)
        draw_board(pieces)
    if white_pieces > 3:
        print("Player one has won with whites!!!")
    else:
        print("Player two has won with blacks!!!")

```

main():
Yazılmış tüm fonksiyonları kullanarak oyunun oynatılmasını sağlayan main() fonksiyonu.

Programcı kataloğu \ Projenin üretim aşamaları

Analiz ve Tasarım 6 saat

Problem analiz edildi. Oyunu en iyi şekilde simüle edebilmek için üstüne düşünüldü. Çözmek için gerekli olan algoritmalar düşünüldü ve önce pseudo-code vasıtasyyla kağıt üzerinde geliştirildi.

Gerçekleştirme 16 saat

Taslak oluştuktan sonra program python üzerinde oluşturulmaya başlandı. Örnek girdiler durumlar oluşturuldu, durumlara göre gerekli fonksiyonlar oluşturuldu.

Test 6 saat

Olası sorunları düzeltmek için program test edildi. Gerekli yerlerde de bug fix aracılığıyla algoritmalar denenedi. Mantıksal hatalar giderildi.

Rapor 3 saat

Program sorunsuz şekilde oluşturulduktan sonra, süreç ve adımlar raporlandı.

Kullanıcı katologu \ kare kapan oyunu

Kare Kapan oyunu 2 kişi ile oynanan bir strateji oyunudur. Oyun alanında, 7 yatay ve 8 dikey çizginin oluşturduğu 42 adet kare bulunmaktadır. Oyun, 28 beyaz ve 28 siyah taş ile oynanır.

Oyunun amacı rakibin taş sayısını 3'e indirmektir.

Beyaz taşlarla oynayan oyuncu önce başlar. Sırası gelen oyuncu çizgilerin kesiştiği herhangi bir noktaya taş yerleştirerek kare elde etmeye çalışır. Oyuncuların ellerindeki taşlar bitene kadar sıra ile taşlar yerleştirilmeye devam edilir. Bütün taşlar yerleştirilinceye kadar oyunda hiçbir taş hareket ettirilemez veya taşlarla hamle yapılamaz.

Bütün taşlar yerleştirildikten sonra oyuncular elde ettikleri kareleri sayarlar ve elde ettikleri kare sayısı kadar seçecekleri rakip taşını oyun dışına çıkarırlar. Eğer hiçbir oyuncu kare elde edememişse, ilk oyuncu (beyaz taşlarla oynayan oyuncu) rakip taşlarından birini oyun dışına çıkarır. Oyunun herhangi bir aşamasında, dışarı çıkarılacak bir rakip taşı seçilirken oluşmuş kareler bozulamaz.

Daha sonra hamle sırası gelen oyuncu, kendi taşlarından birisini hareket ettirerek kare oluşturmaya çalışır. Önü boş olan taş istenildiği kadar yatay ya da dikey hareket ettirilebilir ancak taşların üzerinden hiçbir durumda atlanamaz. Yeni bir kare oluşturulduğu an, seçilen bir rakip taşı dışarı çıkarılır. Rakibin taş sayısını 3'e düşüren oyuncu oyunu kazanır.

Kullanıcı katoloğu \ kullanıcı talimatları

Please enter the number of horizontal lines [3,7]:

Oyuncular 3 ile 7 arasında bir tam sayı girerek oyun tahtasını oluşturabilirler.

	A	B	C	D					
1	0	---	0	---	0	---	0		
2	0	---	0	---	0	---	0	---	0
3	0	---	0	---	0	---	0	---	0

Select an empty space for your piece (Player One):

Oyun tahtası oluştuktan sonra oyuncular ilk satır numarası sonra da sütun harfi olacak şekilde (1A gibi) belli koordinatları girerek taşlarını sırayla yerleştirirler.

	A	B	C	D					
1	W	---	W	---	B	---	B	---	B
2	W	---	W	---	B	---	B	---	B
3	B	---	W	---	B	---	B	---	W

Player one can remove 1 piece!

Please select a piece which isn't a part of a square to remove from opponent (Player one):

Tüm taşlar yerleşikten sonra yine koordinat girerek rakibin taşları oyun dışına çıkarılabilir.

	A	B	C	D			
1	W	---	W	---	B	---	B
2	W	---	W	---	B	---	B
3	0	---	0	---	B	---	W

Please select a piece and a position to move (Player one):|

Oyuncular öncelikle hareket ettirecekleri taşın sonra da gitmesini istedikleri yerin koordinatını bu sırayla (A1 A2 gibi) girerek taşlarını hareket ettirebilir.

	A	B	C	D			
1	W	---	W	---	B	---	B
2	W	---	W	---	B	---	B
3	0	---	0	---	B	---	W

Player two can remove 1 piece!

Please select a piece which isn't a part of a square to remove from opponent (Player two):|

Eğer taşı hareket ettirdikten sonra kare oluştururlarsa oyuncular yine rakiplerinin bir taşını oyun dışına çıkarır. Son üç taşı kalan oyuncu oyunu kaybeder.

Player two used all of his/her removals!

	A	B	C	D			
1	W	---	0	---	B	---	0
2	W	---	0	---	B	---	B
3	W	---	0	---	B	---	B

Player two has won with blacks!!