Alquerque

Eksamensprojekt DM574
Del 2

Vejleder:
Luís Cruz-Filipe

Lavet af:
Søren Rosendahl Christensen
soerc23@student.sdu.dk
project group 20

Resumé

Denne rapport handler om hvordan jeg har skabt et interaktivt program der lader brugeren spille det forhistoriske spil alquerque, i terminalen. Klienten bruger udbydermodulet implementeret de andre dele af projektet til at klarer størstedelen af arbejdet for os og derfor håndterer klientmodulet mest brugerinput og en visual repræsentation af spillet i terminalen.

Abstract

This report is about how I have created an interactive program that lets the user play the ancient game of alquerque, in the terminal. The client uses the provider module implemented by the other parts of the project to handle most of the work for us and therefore the client module handles mostly user input and a visual representation of the game in the terminal.

Forord

Rapporten er lavet på første semester af uddannelsen computer-science på kurset DM574 'Introduktion til programmering' og er i den forbindelse eksamensprojektets anden del ud af tre og vil være baggrund for det mundtlige forsvar af projektet til januar 2024.

Det må noteres at som et gruppeprojekt har dette ikke været den største success, hvilket kan skyldes flere grunde. Jeg har har forsøgt at motivere de to andre til at deltage og tage initiativ både i forbindelse med det faglige men også til sociale sammenkomster *(for projektets skyld)*.

Indhold

1 Indledning **4**
1.1 Problemformulering 4
1.2 Kravspecifikation 4
1.3 Projektafgrænsning 4

2 Implementering **5**
2.1 Interface 5
2.2 Funktioner 6
2.3 Håndtering af I/O 6
2.3.1 Generelt om inputtet 6
2.3.2 Generelt om outputtet 7
2.3.3 Tilstande 7
2.3.4 Tur 7
2.3.5 Visning af brættet 7
2.3.6 Slut spil 7

3 Testning **7**

4 Konklussion **8**

5 Bilag **9**

1 Indledning

Projektets formål i denne del er at udvikle et klientprogram der benytter udbydermodulerne der udvilkes i de andre faser. I det følgende kapitel fastlægges projektets formål og problemformulering, som danner grundlaget for projektet. Rapporten afspejler og dokumenterer det udførte projektarbejde på 1. semester for projektets anden fase i kurset DM574 "Introduktion til programmering".

1.1 Problemformulering

Formålet med projektet er at undersøge, hvordan adgangspunktet til programmet kan implementeres i topmodulet alquerque.py. Løsningen skal opbygges som en modulær løsning, der benytter de underlæggende moduler `board.py` og `minimax.py` udviklet i de andre dele af projektet. Størstedelen af spilogikken er implementeret i disse moduler, og topmodulet `alquerque.py` skal dermed implementeres ved at udnytte dette. Det betyder at alquerque.py for det meste skal kalde de rigtige funktioner i en passende rækkefølge.

1.2 Kravspecifikation

- Brugeren vælger først hvilke spillere er menneskestyret og hvilke er computerstyret.
- Ved en menneskestyret spillers tur skal programmet bede spilleren om at specificere sit træk.
- Ved en computerstyret spillers tur skal programmet informerer om dens træk.
- Efter hver tur skal brættets nuværende tilstand vises.
- Spillet slutter når en spiller har vundet, eller ikke har flere træk.
- Alt I/O foregår i terminalen.

1.3 Projektafgrænsning

Projektet er ikke udviklet til et slutprodukt. I stedet udvikles et *Proof of Concept*, som kalder de rigtige funktioner i en passende rækkefølge. Derfor kan vi f.eks. antage at brugerinput har den rigtige type når programmet kører. Det noteres også at brugerfladen vil være brugbar, men ikke perfekt.

2 Implementering

Implementeringen kan opdeles i to kategorier, input og output. Det er udviklet ud fra interfacet i board.py modulet som repræsenterer brættet og den tilhørende funktionalitet. Mit program er derfor begrænset til denne implementation. I dette kapitel redegører jeg kort for top-modulets implementation, og de dele af interfacet der benyttes.

2.1 Interface

Interfacet udgør grundlages for hvordan mit program er implementeret og det er derfor relevant at komme med en beskrivelse af følgende funktioner som jeg benytter:

- `make_board` returnere et nyt bræt klar til at spille.
- `white` Retunere en liste med hvid spillers positioner.
- `black` Retunere en liste med sort spillers positioner.
- `legal_moves` Retunerer en liste med spilerens lovlige træk.
- `next_move` Returnere et træk blandt de mulige træk.
- `white_plays` Fortæller hvem har næste tur.
- `move` Opdaterer brættet med et givent træk.
- `is_game_over` Fortæller om spillet er slut.

Udover ovennævnte findes flere funktioner, men disse er ikke benyttet direkte i min implementation, og vises derfor ikke.

2.2 Funktioner

Funktionaliteten er forsøgt encapsuleret, så programmet arbejder ikke med globale variabler, og hver funktion tager relevante argumenter for opgaven den udfører. Det gør progammet nemmere at forstå og minimere risikoen for fejl i prorgammet.

Programmet definerer hovedfunktionerne:

- `game` Initialisere spillet og starter spillet. Benytter `make_board`.
- `play` Spiller det valgte spil alquerque. Benytter `move` og `next_move`.

For output:

- `print_rules` Viser spilreglerne.
- `print_menu` Viser spil-menuen.
- `print_board` Viser brættet.
- `print_moves` Viser mulige træk. Benytter `legal_moves`.

Er for bruger input:

- `get_move` Tager input fra spilleren og retunere det valgte træk. Benytter `legal_moves`.
- `get_mode` Tager input fra spilleren og retunere det valgte game mode.
- `get_choice` Tager input fra spilleren om valget mellem at fortsætte eller ændre game mode.

Hjælpefunktioner:

- `_game_result` Returnere en string om spillets resultat. Benytter `white` og `black`.
- `_ai_plays` Returnere en boolsk værdi om det er computerens tur. Benytter `white_plays`.
- `_board_list` Returnere en liste der repræsentere brættet. Benytter `white` og `black`.
- `_player_color` Returnere en string med nuværende spillers farve. Benytter `white_plays`.

2.3 Håndtering af I/O

Interaktionen mellem programmet og brugeren består af veksling af information gennem terminalen. Helt generelt er denne funktionalitet opdelt i programmet.

2.3.1 Generelt om inputtet

Input til programmet består af tal der hører til besteme valg vist i terminalen. Denne beslutning er taget for at minimere muligheden for indstastningsfejl og simplificere interaktionen for brugeren. Kort sagt bliver brugeren præsentereret for nogle valg og vælger ved at indtaste det tal der hører til valget.

2.3.2 Generelt om outputtet

Output til brugeren vises formateret i terminalen. Det er altsammen indrykket en tab for at skabe er 'polstret' indtryk af brættet. Brættet samt den nuværende spiller vises på skærmen inden hver tur, hvilket sikre at en opdateret version vises og at man kan se hvem der har tur. Efter hver tur vises både spiller og træk, og om det er en person eller computeren der har rykket.

2.3.3 Tilstande

Brugeren får valget mellem fem muligheder i menuen, de fire måder to farver kan kombineres med to spillere, samt en mulighed for at afbryde programmet.
Menuen vises ved at kalde `print_menu`, hvorefter `get_mode` kaldes som retunere informationen til programmet. Dette valg har betydning for programmets kontrol flow i og med vi skal have input fra spilleren. Denne information gemmes i variablen `game_mode` og kan ændres undervejs.

2.3.4 Tur

Under spillet skal programmet spørge spilleren om at specificere sit træk, ellers skal computeren vælge et træk. Dette opnås baseret på værdien i `game_mode` i. Hjælpefunktionen `_ai_plays` er vedhæftet i Bilag 1.

Programmet spørger først spilleren om at specificere sit træk, eller ændre spillets tilstand:

- Hvis man vælger at specifiere trækket, vises mulige træk, hvorefter `get_move` kaldes.
- Hvis man vælger at ændre tilstanden, vises menuen, hvorefter `get_mode` kaldes.

Når det er computerens tur, bruges funktionen `next_move` fra interfacet. Brættet opdateres hver gang ved at benytte `move`.

2.3.5 Visning af brættet

Brættet vises vha. af `board_list` der konstruere en liste med 25 elementer hvor hver plads svare til den korresponderende plads på brættet. Her tages højde for at listen er 0-indekseret. Dette valg simplificere funktionen `print_board` som viser brættet i terminalen. Hjælpefunktionen `_board_list` er vedhæftet i Bilag 1.

2.3.6 Slut spil

Spillet bryder ud af game loopen, når spillet er slut eller tilstanden er 0. Dvs. når `is_game_over` returnere `False` eller brugeren vælger at afbryde.

3 Testning

Funktioner der retunere værdier har doctests der viser at de virker korrekt og alle funktioner har docstrings der beskriver hvad de gør. Et eksempel kan ses i Bilag 1.

4 Konklussion

I denne rapport har jeg undersøgt og redegjordt for hvordan modulet `alquerque.py` kan implementeres ved at tage begrænsninger fra undermodulerne i betragning. Der er plads til at gøre det mere udførligt, men overordnet set er det vigtigste med. F.eks. udfordringen med at vise brættet i terminalen og hvorfor programmets kontrol flow styres af den valgt tilstand.

Udfordringer:

- En udfordring har været at bruge globale variabler, hvilken havde effekten at programmet blev mere kompleks/svært at forstå. Det kan være en fordel at gøre funktionernes parametre endnu mere specifikke. F.eks. tager nogle funktioner datatypen Board, men bruger ikke brikkernes positioner. Således kan programmet gøres mere forståligt.
- En anden udfordring har været navngivningen af variabler. Det skulle være ensartet gennem hele programmet, og alle navnene skulle gøre klart hvad de betyder i deres sammenhæng. Jeg synes det lykkedes rimeligt.
- En tredje udfordring, som ikke er relateret til selve projektet, er manglende feedback fra den første del af projektet.

Således er det lykkedes at implementere `alquerque.py`, der lever op til kravende. Rapporten introducerer og forklarer implementationen. Nu ser jeg frem til feedback, så jeg kan gøre det endnu bedre næste gang. Jeg vil gerne takke læseren for opmærksomheden, og håber at det været lige så fedt at læse den som det har været for mig at lave den.

5 Bilag

```
import board as b
import minimax

def print_menu() -> None:
    """Prints the select game mode menu in the console."""
    print("\t********** SELECT GAME MODE **********\n" +
          "\t|(1) PvP                              |\n" +
          "\t|(2) White vs CPU                     |\n" +
          "\t|(3) Black vs CPU                     |\n" +
          "\t|(4) CPU vs CPU                       |\n" +
          "\t|(0) Quit                             |\n" +
          "\t|      ~made by soren rosendahl~      |\n" +
          "\t***************************************")

def print_board(board: b.Board) -> None:
    """Prints the board in the console."""
    positions = _board_list(board)
    print("\t-----------       BLACK      ------------")
    for i in range(25):
        if i != 0 and i % 5 == 0:
            print("\t|\n" +
                  "\t|\t|\t|\t|\t|\t|")
            print(f"\t|{i+1}:{positions[i]}", end="")
        else:
            print(f"\t|{i+1}:{positions[i]}", end="")
    print("\t|")
    print("\t-----------       WHITE      ------------")

def print_rules() -> None:
    """Prints the rules of alquerque in the console."""
    print("\tGAME RULES:")
    print("\t*The purpose of the game is to capture all enemy pieces.\n" +
          "\t*Each player starts with 12 pieces.\n" +
          "\t*All moves are straight or diagonal on designated lines.\n" +
          "\t*Normal distance is 1 field\n" +
          "\t*Capture distance is 2 fields and posibble\n" +
          "\tif and only if the enemy is between the capturing move.\n" +
          "\t*Normal moves is only permitted to go forward.\n" +
          "\t*Captures is permitted in any direction.\n" +
          "\t*Game ends when the has lost all its pieces\n" +
          "\tor there are no more moves.")

def print_moves(board: b.Board) -> b.Move:
    """Prints the legal moves in the console."""
    print(f"\tDecisions:")
    moves = b.legal_moves(board)
    for i in range(len(moves)):
        if i != 0 and i % 3 == 0:
            print()
        move = moves[i]
        print(f"\t[{i+1}]: {move[0]} to {move[1]}", end="")

def get_move(board: b.Board) -> b.Move:
    """Asks the user to make a move returns it
    >>> get_move(b.make_board())
            Make your move: 1
```

```
    (17, 13)
    >>> get_move(b.make_board())
            Make your move: 0
            try again:
    """
    moves = b.legal_moves(board)
    move = int(input("\tMake your move: "))
    while move < 1 or move > len(moves):
        move = int(input("\ttry again: "))
    return moves[move - 1]

def get_mode() -> int:
    """Asks the user to select game mode and returns it
    >>> get_mode()
            Which one is it?: 0
    0
    >>> get_mode()
            Which one is it?: 5
            try again:
    """
    game_mode = int(input("\tWhich one is it?: "))
    while game_mode < 0 or game_mode > 4:
        game_mode = int(input("\ttry again: "))
    return game_mode

def get_choice() -> int:
    """Asks the user about the flow of the game.
    >>> get_choice()
            [1]: Make move
            [0]: Change game mode
            Choose: 0
    0
    >>> get_choice()
            [1]: Make move
            [0]: Change game mode
            Choose: 2
            try again:
    """
    choice = int(input("\t[1]: Make move\n" +
                       "\t[0]: Change game mode\n" +
                       "\tChoose: "))
    while choice < 0 or choice > 1:
        choice = int(input("\ttry again: "))
    return choice

def _game_result(board: b.Board) -> str:
    """Print the final result of the game in the console.
    >>> _game_result(b.make_board())
    'DRAW'
    >>> _game_result(b.Board(board=[[0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0],
                                    [0, 0, 0, 1, 0],
                                    [0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0]], player=2))
    'WHITE wins'
    >>> _game_result(b.Board(board=[[0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0],
                                    [0, 0, 0, 2, 0],
```

```
                                       [0, 0, 0, 0, 0],
                                       [0, 0, 0, 0, 0]], player=1))
    'BLACK wins'
    """
    msg = ""
    if (b.black(board) != [] and b.white(board) != []):
        msg = "DRAW"
    elif b.white(board) == []:
        msg = "BLACK wins"
    else:
        msg = "WHITE wins"
    return msg

def _player_color(board: b.Board) -> str:
    """Returns the current player turn.
    >>> _player_color(b.make_board())
    'WHITE'
    """
    if b.white_plays(board):
        return "WHITE"
    else:
        return "BLACK"

def _ai_plays(game_mode: int, board: b.Board) -> bool:
    """Determines whether it's the AI's turn to make a move.
    >>> _ai_plays(1, b.make_board())
    False
    >>> _ai_plays(2, b.make_board())
    False
    >>> _ai_plays(3, b.make_board())
    True
    >>> _ai_plays(4, b.make_board())
    True
    """
    return (game_mode == 4 or
            (game_mode == 2 and not b.white_plays(board)) or
            (game_mode == 3 and b.white_plays(board)))

def _board_list(board: b.Board) -> list[str]:
    """Returns a list representation of the board where
    each index corresponds to a position on the board.
    >>> _board_list(b.make_board())
    ['B', 'B', 'B', 'B', 'B',
      'B', 'B', 'B', 'B', 'B',
        'B', 'B', '', 'W', 'W',
          'W', 'W', 'W', 'W', 'W',
            'W', 'W', 'W', 'W', 'W']
    """
    positions = ["" for i in range((25))]
    for pos in b.white(board):
        positions[pos - 1] = "W"
    for pos in b.black(board):
        positions[pos - 1] = "B"
    return positions

def play(game_mode: int, board: b.Board) -> None:
    """Plays alquerque in the console."""
    while b.is_game_over(board) == False and game_mode != 0:
```

```
        print(f"\t{_player_color(board)} plays ***************************")
        print_board(board)
        if _ai_plays(game_mode, board):
            move = minimax.next_move(board)
            print(f"\tAI {_player_color(board)} moved from {move[0]} to {move[1]}")
            b.move(move, board)
        else:
            print("\t----------------------------------------")
            match get_choice():
                case 0:
                    print_menu()
                    game_mode = get_mode()
                    print(f"\tGame mode changed to {game_mode}")
                case 1:
                    print_moves(board)
                    print()
                    move = get_move(board)
                    print(f"\tHUMAN {_player_color(board)} moved from {move[0]} to {
    move[1]}")
                    b.move(move, board)

def game() -> None:
    print_rules()
    print("\t~~~~~~~~ Welcome to Alquerque ~~~~~~~~~")
    board = b.make_board()
    print_menu()
    game_mode = get_mode()
    print()
    play(game_mode, board)
    print("\tFinal state:")
    print_board(board)
    print(f"\t------ GAME OVER ------ result is {_game_result(board)}")
    print("\t**** Thanks for playing Alquerque! ****")

game()
```

Listing 1: kildekode