Programmeringsprojekt Del 2

Celine S. Fussing
cefus23@student.sdu.dk

Sandra K. Johansen
sjoha23@student.sdu.dk

Sofie Løfberg
soloe23@student.sdu.dk

14. december 2023

Contents

1 Introduktion

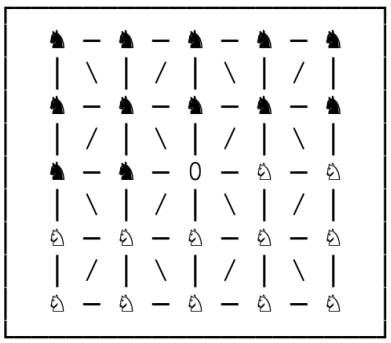
Denne rapport er udarbejdet i november og december 2023 til et projekt på Syddansk Universitet (Odense) i kurset DM574. I denne fase har vi udviklet et modul, alquerque.py, med hensigt på at kunne spille Alquerque i terminalen. Vi bruger vores egne moduler, board.py og move.py, fra fase 1 af projektet.

2 Designvalg

I dette afsnit uddybes nogle af de designvalg, der ligger til grund for oplevelsen af vores spil.

2.1 Udseende

Vi har valgt, at alquerque.py skal have et tema. Vi landede på et middelalder tema, da det passede godt til det gamle Alquerque-spil. Derfor ville vi gerne repræsentere vores brikker som andet end de klassiske brikker - runde, sorte og hvide brikker. Brikkerne endte som henholdvis sorte og hvide springere - ligesom i skak. Se figur 1. Dette gøres med UNICODE. Metoden uddybes under afsnittet "3 Teknisk beskrivelse af programmet".



Figur 1

Sproget skulle matche vores tema; den kliniske tilgang i en allerede minimalistisk terminal virkede kedelig. Derfor er vores sprog inspireret af Shakespeares værker. Det skal skabe associationer til en engelsk slagmark i middelalderen. Vi har dog aktivt undgået at rime, da det ville være mindre læseligt.

2.2 Brugervenlighed

Der er en ulempe i opsætningen af vores bræt i terminalen. Terminalen, baseret på hvilke indstillinger brugeren har, kan bytte rundt på farverne af vores springere. Dette vil sige, at har man mørk tilstand, vil de sorte brikker fremstå hvide

og omvendt. Brættet kan også ende med at se småt ud. Vi erfarede, at lys tilstand i terminalen kombineret med Courier New-fonten størrelse 18 skabte en god oplevelse. Her var springerne den korrekte farve, og brættet bliver printet så læsbart som muligt.

Herudover har vi lavet ét særligt vigtigt designvalg - nemlig at man på ethvert givent tidspunkt kan afslutte spillet uden at skulle lukke terminalen ned. Dette implementeres ved, at brugeren kan indtaste "q" for "Quit".

3 Teknisk Beskrivelse af Programmet

Programmet består af vores moduler board.py og move.py fra del 1 af projektet samt filen minimax.py. Derudover har vi lavet en ny fil, alquerque.py, som indeholder 6 funktioner.

3.1 _convert

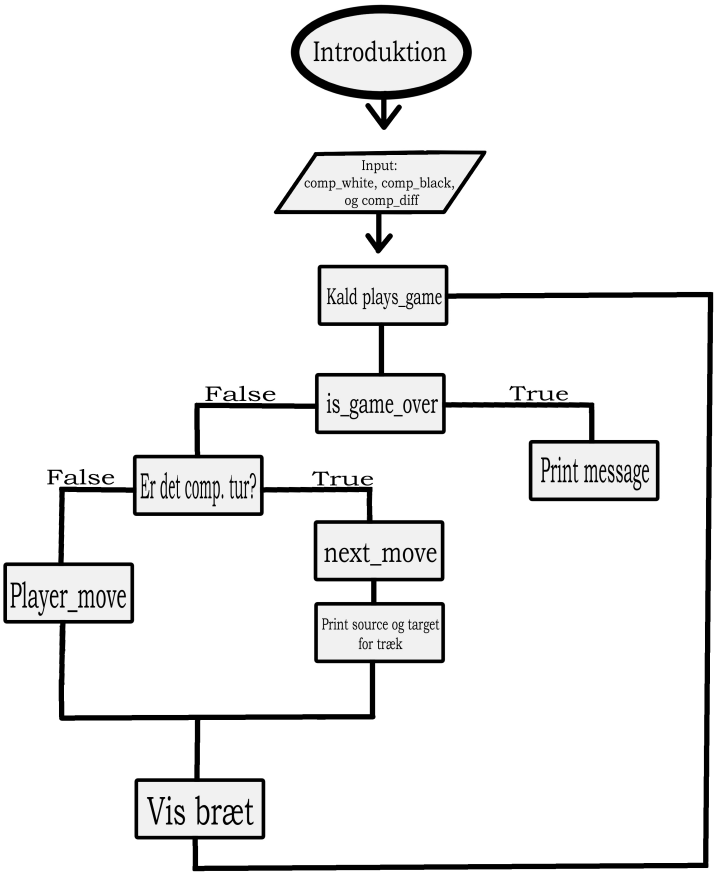
Som beskrevet under afsnittet "Designvalg" bruger vi springere som repræsentation af vores brikker. Dette er gjort ved brug af UNICODE, hvor hjælpefunktionen _convert anvender black og white funktionerne fra board.py-modulet. _convert konverterer det givne indeks til den korrekte UNICODE-string alt efter hvilket tal, der er på det indeks.

3.2 start_game

I funktionen præsenteres spillet og kalder de funktioner, som viser vores bræt; indices_board og show_board. Herefter definerer vi tre variabler: comp_white, comp_black og comp_diff. Heri gemmes spillerens valg for, om computeren spiller hvid, sort eller ingen. Hvis computeren spiller, vælges der også en sværhedsgrad mellem 1 og 7. Desuden har vi implementeret det således, at både "y" og "Y" godtages som input ved hjælp af string-modifikationsmetoden, lower. Afslutningsvist kalder start_game funktionen, plays_game.

3.3 plays_game

Funktionen plays_game kalder de korrekte funktioner, der gør, at spillet kører, som det skal. Den starter med at tjekke, om spillet stadig er i gang. Hvis det er tilfældet, tjekker vi, om det er computeren, der skal lave næste træk. Computerens træk defineres som variablen new_comp_move. Trækket udføres ved brug af move-funktionen fra board.py. Herefter printes hvilken brik der rykkes hvorhen.



Figur 2

3.4 player_move

Funktionen player_move udfører en spillers træk. Den tager to input fra brugeren (tal mellem 1 og 25) og definerer dem som variablerne s (source) og t (target). Variablerne bruges som input i make_move-funtionen. Det nye træk udføres, hvis det er lovligt. Hvis trækket ikke er lovligt printes en besked, hvorefter spillet gentager turen. Til sidst i funktionen benyttes show_board til at printe et opdateret bræt. Herefter kaldes funktionen rekursivt, så det næste træk kan udføres.

3.5 indices_board

Funktionerne indices_board og show_board er essentielle for spillet. De viser henholdvist repræsentationen af brættet med indekser samt selve brættet med de sorte og hvide springere. Indeks-repræsentationen giver spilleren overblik over positionerne på brættet, da vi kræver to tal-input for at lave et træk. Dette board bliver vist én gang ved begyndelsen af spillet. show_board bruges udelukkende efterfølgende.

4 Overvejelser

I dette afsnit redegøres der for vores overvejelser om blandt andet brættes repræsentation og interaktionen mellem terminalen og spilleren.

4.1 Brugervenlighed

I vores oprindelige repræsentation af brættet ville vi bruge indekser ved tomme pladser. Dette medførte, at brættet blev asymmetrisk. Derfor bruger vi 0'er i stedet. Vi forudsætter, at folk kan tælle til 25.

Vi prøvede derudover at bruge match i start_game, når der skulle vælges spillere. Dette skyldes, at match tager tastefejl mere seriøst. Vi løb hurtigt ind i problemet, at vi ikke kunne bryde ud af match igen uden aktivt bruger-input. Derved gik vi tilbage til if-statements.

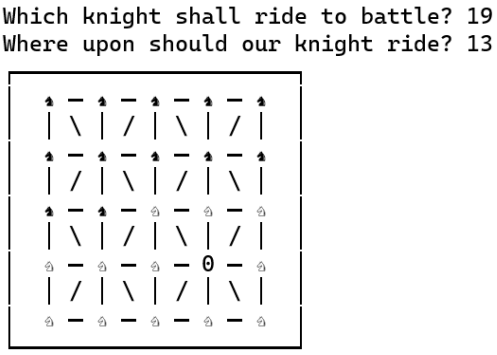
For brugervenlighedens skyld ville vi gerne have delt start_game op i flere hjælpefunktioner. Hvis der opstod en tastefejl, ville det være ideelt at kunne gentage enkelte dele af programmet. Dette havde dog konsekvenser for vores brug af variabler og for læseligheden af vores program. Vi endte med at slå start_game sammen igen. Dette betyder dog, at hvis der har været en tastefejl, skal flere værdier indtastes på ny.

4.2 Terminalen

Et sidste problem er størrelsen på terminalen, da vores bræt i sig selv fylder meget, hvis man bruger skrifttype 18. Selve introduktionen til spillet bliver, i den normale terminal størrelse, "skåret af", og brugeren skal selv gå lidt op for at se introduktionsteksten. Dette er ikke ideelt, da hvordan man skal interagere med spillet står øverst. Løsningen kunne være at anvende terminalen som et fuld-skærmsvindue eller et større-end-normalt vindue. Der er selvfølgelig også mulighed for en mindre skrifttype, men vi oplevede, at dette gjorde brættet meget lille. I sidste ende giver det ingen problemer spilmæssigt, da man stadig bliver guidet igennem hele spillet løbende.

5 Test af Programmet

Ligesom i første del af projektet var det vigtigt løbende at teste vores funktioner og program. Så vidt muligt har vi forsøgt at teste vores funktioner enkeltvist. Dette har dog ikke kunne lade sig gøre flere steder. Stort set alle vores funktioner returnerer ikke noget, og flere af funktionerne kalder andre funktioner eller fungerer kun i sammenhæng. Det har været muligt at teste show_board, indices_board og _convert. Dette gjorde vi som det første, da brættet skulle fremstå velafgrænset og utvetydigt. Førnævnt var især vigtigt, da vi arbejder med UNICODE. Herudover har det været muligt at teste player_move, hvilket ses nedenfor af figur 3. Dette gør vi ved at kalde player_move og herefter show_board. Vi kan derved tjekke, at brættet opdateres korrekt.



Figur 3

Eftersom vi ikke har kunne teste alle vores funktioner separat, har det været nødvendigt at "teste" gennem læsning. I sidste, men fundamentale, testfase kører vi spillet. Her testes funktionen plays_game og start_game. Dette gøres ved at se, om der skiftes tur, om brættet opdateres, og om spillet slutter. Herudover tester vi især med tanke for: Hvordan oplever brugeren spillet? Er der noget generende i vores opsætning? Her opdagede vi blandt andet, at det fremmede overblikket for brugeren, hvis der ikke var for meget tekst. Til dette kan det tilføjes, at vi også gerne vil imødekomme spillerfejl såsom tastefejl eller ugyldige træk.

6 Konklusion

På baggrund af rapporten kan vi konkludere, at vi har formået at indfri vores fundamentale ønsker for alquerque.py. Vi har lavet et modul, som opfylder spillets opsætning. Vi har skabt en repræsentation af spillet, som rækker udover den kliniske tilgang. Først og fremmest har vi designet alquerque.py således, at den kalder funktionerne fra del 1 i den rigtige rækkefølge Herudover har vi arbejdet med sproget og repræsentationen af brættet for at give det noget mere personlighed. I sidste ende voldte det os de største udfordringer at få lavet velovervejede og isolerede test. Dette formåede vi delvist. Vi endte med at teste enkelte funktioner separat og andre i sammenhæng. Dette foregik dog med øje for, hvilke fejlgrupper der kunne være ved disse testformer. Vi kan hermed konkludere, at det nye top-modul alquerque.py fungerer som et uafhængigt spil i terminalen, der udelukkende modtager input fra tastaturet.

7 Appendix

```
from board import *
from move import *
from minimax import next_move


def start_game() -> None:
    """Start the game"""
    print("Hear ye hear ye!"
          "'Tis thine ancient game… "
          "Thou shalt journey, proud adventurer; "
          "Be brave as you go on! this game of Alquerque may last long! "
          "'Tis the battefield upon which you will find your glory or perhaps your defeat! "
          "Take notice of thine possible moves "
          "and if thou wish to lay down thine arms, enter q upon thine board of keys. " )
    indices_board()
    show_board()
    comp_white = (input("Does thou wish the magical machine to play white? "
                        "a single y for yay, or an n for nay! ").lower())
    if comp_white == 'n':
        comp_white = False
    elif comp_white == 'y':
        comp_white = True
    elif comp_white == 'q':
        exit()
    else:
        print("Dear friend, 'twas not a choice. Give it another attempt! ")
        start_game()
    comp_black = input("Does thou wish the magical machine to play black? y or n? ").lower()
    if comp_black == 'n':
        comp_black = False
    elif comp_black == 'y':
        comp_black = True
    elif comp_black == 'q':
        exit()
    else:
        print("Dear friend, 'twas not a choice. Give it another attempt! ")
        start_game()
    if (comp_white or comp_black):
        comp_diff = input("Thine enemy be quick to strike, "
                          "but HOW quick? "
                          "Thine decision may be made upon entering, "
                          "to your board of keys, "
                          "a numeral that exists between 1 and 7! ").lower()
        if comp_diff == 'q':
```

```
            exit()
        elif not (0 < int(comp_diff) ≤ 7):
            print("Dear friend, 'twas not a choice. Give it another attempt! ")
            start_game()
        plays_game(comp_white, comp_black, int(comp_diff))
    else:
        plays_game(comp_white, comp_black, 0)

def plays_game(comp_white: bool, comp_black: bool, comp_diff: int) -> None:
    """Play the game"""
    if is_game_over(b):
        if black(b) == []:
            print("Brave battles were fought and brave battles were lost! "
                  "Here I must announce, that our White Knight has won! ")
        elif white(b) == []:
            print("Brave battles were fought and brave battles were lost! "
                  "Here I must announce, that our Black Knight has won! ")
        else:
            print("Battles are strenuous, and strategy flows a plenty. "
                  "Alas at this conjecture, our Knights are wounded and weary. "
                  "Neither may win and yet, neither may lose. The adventure is over,"
                  "pick up your boots! ")
    else:
        if ((white_plays(b) and comp_white) or (not white_plays(b) and comp_black)):
            new_comp_move = next_move(b, comp_diff)
            move(new_comp_move, b)
            print("The magical machine moves knight", source(new_comp_move),
            "to field", target(new_comp_move))
        else:
            player_move()
        show_board()
        plays_game(comp_white, comp_black, comp_diff)

def player_move() -> None:
    """Ask the player for a move, and update the board by making that move"""
    s = input('Which knight shall ride to battle? ').lower()
    if s == 'q':
        exit()
    t = input('Where upon should our knight ride? ').lower()
    if t == 'q':
        exit()
    upcoming_move = make_move(int(s), int(t))
    if is_legal(upcoming_move, b):
        move(upcoming_move, b)
    else:
        print("Thine knight is lacking the bravery necessary for thy move, try another! ")
```

```
def show_board() -> None:
    """Print a visual representation of the board"""
    c = _convert
    print("┌─────────────────────┐")
    print("│ ",c(1),"─",c(2),"─",c(3),"─",c(4),"─",c(5)," │")
    print("│  │ \ │ / │ \ │ / │  │")
    print("│ ",c(6),"─",c(7),"─",c(8),"─",c(9),"─",c(10)," │")
    print("│  │ / │ \ │ / │ \ │  │")
    print("│ ",c(11) ,"─",c(12) ,"─",c(13) ,"─",c(14) ,"─",c(15) ," │")
    print("│  │ \ │ / │ \ │ / │  │")
    print("│ ",c(16) ,"─",c(17) ,"─",c(18) ,"─",c(19) ,"─",c(20) ," │")
    print("│  │ / │ \ │ / │ \ │  │")
    print("│ ",c(21) ,"─",c(22) ,"─",c(23) ,"─",c(24) ,"─",c(25) ," │")
    print("└─────────────────────┘")

def indices_board() -> None:
    """Print the board with the indices"""
    print("┌─────────────────────────┐")
    print("│ ",1," ─",2," ─",3," ─",4," ─",5," │")
    print("│  │  \ │ /  │ \  │ /  │  │")
    print("│ ",6," ─",7," ─",8," ─",9,"─",10," │")
    print("│  │  / │ \  │ /  │ \  │  │")
    print("│ ",11,"─",12,"─",13,"─",14,"─",15,"│")
    print("│  │  \ │ /  │ \  │ /  │  │")
    print("│ ",16,"─",17,"─",18,"─",19,"─",20,"│")
    print("│  │  / │ \  │ /  │ \  │  │")
    print("│ ",21,"─",22,"─",23,"─",24,"─",25,"│")
    print("└─────────────────────────┘")

def _convert(n: int) -> str:
    """Convert black or white to their respective knight characters"""
    if n in black(b):
        return '\u265E'
    if n in white(b):
        return '\u2658'
    else:
        return 0

start_game()
```