TPK4186 - Advanced Tools for Performance Engineering Spring 2023

Assignment 2: Chess Games

Done by Christian G Kartveit

NTNU

Kunnskap for en bedre verden

Introduction

To ensure you can run this program, you need to use the python virtual environment.
To activate use:

- Windows `source venv/Scripts/activate`
- Mac/linux `source venv/bin/activate`

Then run `pip install -r requirements.txt`

This will install the required packages to run the application

Assumptions

1. You need to run all the scripts in the requirements file to run the application
2. You need to comment out the circular imports in Database.py and TreeStructure.py. This is shown in the code. It will provide an error msg if not done correctly.

Games

Most of the tasks from games, is made done using the class ChessGame.py

```
def __init__(self, metadata, moves):
    self.metadata = metadata
    self.moves = moves
    self.winner = None
```

It contains the metadata as dictionary, and all the moves as a list

Moves contains Move objects from Move.py:

```
class Move():

    def __init__(self, move1, move2):
        # Move 1 is the white move, move 2 is the black move
        self.move1 = move1
        self.move2 = move2

    def getMoves(self):
        return self.move1, self.move2

    def getFirstMove(self):
        return self.move1

    def getSecondMove(self):
        return self.move2

    def __str__(self) -> str:
        if self.move2 != None:
            return str(self.move1 + " " + self.move2)
        else:
            return str(self.move1)
```

Task 2 and 3 is done the same way as PGN files. This is implemented in Database.py and supports both versions. You can export to a file from ChessGame.py.

```python
#Task 3
def exportToText(self, file):
    with open(file, 'w') as f:
        f.write('[Event "' + self.getMetadata().get('Event') + '"]\n')
        f.write('[Site "' + self.getMetadata().get('Site') + '"]\n')
        f.write('[Date "' + self.getMetadata().get('Date') + '"]\n')
        f.write('[Round "' + self.getMetadata().get('Round') + '"]\n')
        f.write('[White "' + self.getMetadata().get('White') + '"]\n')
        f.write('[Black "' + self.getMetadata().get('Black') + '"]\n')
        f.write('[Result "' + self.getMetadata().get('Result') + '"]\n')
        f.write('[ECO "' + self.getMetadata().get('ECO') + '"]\n')
        f.write('[Opening "' + self.getMetadata().get('Opening') + '"]\n')
        if self.getMetadata().get('Variation') != None:
            f.write('[Variation "' + self.getMetadata().get('Variation') + '"]\n')
        f.write('[PlyCount "' + self.getMetadata().get('PlyCount') + '"]\n')
        f.write('[WhiteElo "' + self.getMetadata().get('WhiteElo') + '"]\n')
        f.write('[BlackElo "' + self.getMetadata().get('BlackElo') + '"]\n\n')

        for i,move in enumerate(self.getMoves()):
            i = i + 1
            if i % 5 == 0:
                f.write("\n")
            f.write(str(i)+". " + str(move) + ' ')
        f.write(self.getMetadata().get('Result'))
```

It is written in the same format as PGN.

Task 4 extended the class to Database.py, and it contains most of the tasks in this exercise.
I used Regex search, as it was way quicker than any other method to find all the data needed to create 2600 Chess Games.

Statistics

Task 6 is done through DocumentWriter.py, it uses python-docx and I made my own defined methods to add Headings, Paragraphs, Pictures and Tables. It will create a Word document with the selected filename. All of the figures and tasks are done in the Chess Database.docx file.

```python
class DocumentWriter:

    def __init__(self, title, filename):
        self.document = Document()
        self.filename = filename
        self.current_section = None

        # Change font size
        self.document.styles['Normal'].font.name = 'Times New Roman'
        # Increase font size for all headings
        self.document.styles['Heading 1'].font.size = Pt(20)

        self.document.add_heading(
            title, 0).paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
        self.document.add_heading(
            "TPK4186 – Advanced Tools for Performance Engineering \n\n Assingment 2: Chess Game").paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
        self.document.add_heading(
            '\nCreated by: Christian G Kartveit', 2).paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

Openings

This is solved with a Node class

```python
class Node:
    _id_counter = 0

    def __init__(self, move, turn=None, depth=0, parent=None, opening=None):
        self.id = Node._id_counter
        Node._id_counter += 1
        self.turn = turn
        self.parent = parent
        self.children = {}
        self.outcomes = {'white': 0, 'black': 0, 'draw': 0}
        self.move = move
        self.previousMoves = (self.parent.previousMoves +
                              [self.move]) if self.parent is not None else [self.move]
        self.depth = depth
        self.opening = opening
```

Tree class

```python
class Tree:
    _id_counter = 0

    def __init__(self, rootNode):
        self.root = rootNode
        rootNode.setTurn('white')
        #
        self.root.id = Tree._id_counter
        Tree._id_counter += 1
        self.nodes = {}  # {Node: depth}
        self.rootOutcome = None
```

And a TreeManagement class to process multiple trees.

Task 11 figures are provided in Chess Database.docx, and consists of all openings with a depth of 5. This is specified in the method, and can be decided by the user.

```
class TreeManagement:
    def __init__(self) -> None:
        self.trees = {}
```

Task 12 figure is also provided in Chess Database.docx, and is a table of all openings played more than x times. In this table, it is set to N = 50.