Coursework Report

Marco Moroni
40213873@live.napier.ac.uk
Edinburgh Napier University - Algorithms & Data Structures (SET09117)

Abstract

The goal of this coursework is to implement the classic board game of checkers in an arbitrary computer language demonstrating a correct use of data structures. The language chosen here is Python and the game can be played from the console.

Keywords – algorithms, data structures, Python, checkers, draughts

1 Introduction

2 Implementation

2.1 Overwiew

The board is made by a 2D list, in which every item can be either empty or a piece.
Every piece know its *rank* (man or king) and what player they belong to.
Moves are recorded in a stack called `moves`, which is used when undoing. There is another similar stack, `redoMoves`, used to store moves that can be redone.
The game uses a while loop as a game loop. This loop runs until there is a winner.
Finally, the AI works by choosing random moves.

2.2 Board

This project started by creating a simple board made by a 2D array. In Python this is archived by using a list of lists, which means a list of rows where each row is a list os sqaures.
There are 8 rows and 8 columns and each sqaure of the board is initially empty (`None` in Python).

2.3 Pieces

In checkers a piece can be:

- either black or white;
- either a man or a king.

Considering that, the game uses a `Piece` class that knows:

- its player (of type `Player`);
- its *rank*, that is wheter it is a man or a king.

The ranks are enumerated elements of a class `PieceRank`: the values are `PieceRank.MAN` and `PieceRank.KING`. Initially, a piece also knew its position it had at the beginning of the game. This information could have been used in an early version of the `replay` funtion, where there was a board reset to its inital state. Later on this information became useless, because `replay` now resets the board by undoing every move.

2.4 Players

A player is normally identified by being either black or white. In this implementation, a player is an instance of a class `Player`. This class has:

- a dictionary of symbols (*symbols* are the characters printed in the console to represent a piece). The keys are `PieceRanks` and the values are a character (eg. a white/black dot for men, or a white/black sqaure for kings);
- a boolean `isFacingUp`. This is used in two occasions: when setting up the pieces (see *Prelude*) and when getting all the legal displacements of a piece (see *Getting legal displacements*);
- a boolean `cpu`;
- all the functions used by AI (called when the player is not human) (see *AI*).

2.5 Moves

...

3 Formatting

Some common formatting you may need uses these commands for **Bold Text**, *Italics*, and <u>underlined</u>. `Inline code`.

3.1 Referencing

You should cite References like this: [1]. The references are saved in an external .bib file, and will automatically be added to the bibliography at the end once cited.

3.2 LineBreaks

Here is a line

Here is a line followed by a double line break. This line is only one line break down from the above, Notice that latex can ignore this

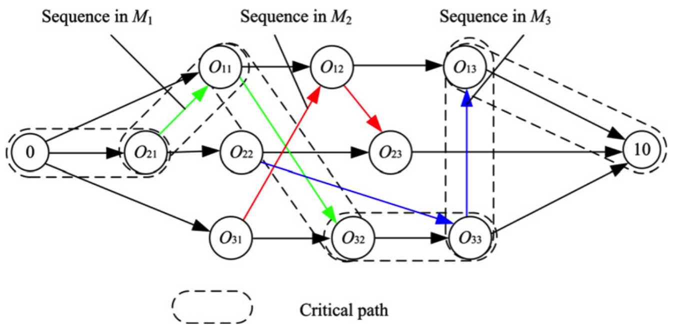

Figure 1: **ImageTitle** - Some Descriptive Text

We can force a break
with the break operator.

3.3 Maths

Embedding Maths is Latex's bread and butter

$$J = \left[\frac{\delta e}{\delta \theta_0} \frac{\delta e}{\delta \theta_1} \frac{\delta e}{\delta \theta_2}\right] = e_{current} - e_{target}$$

3.4 Code Listing

You can load segments of code from a file, or embed them directly.

Listing 1: Hello World! in c++

```cpp
#include <iostream>

int main() {
   std::cout << "Hello World!" << std::endl;
   std::cin.get();
   return 0;
}
```

Listing 2: Hello World! in python script

```python
print "Hello World!"
```

3.5 PseudoCode

```
for $i = 0$ to $100$ do
    print_number = true;
    if i is divisible by 3 then
        print "Fizz";
        print_number = false;
    end
    if i is divisible by 5 then
        print "Buzz";
        print_number = false;
    end
    if print_number then
        print i;
    end
    print a newline;
end
```

Algorithm 1: FizzBuzz

4 Conclusion

References

[1] S. Keshav, "How to read a paper," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 83–84, July 2007.