Checkers Design Doc

Daniel Sinkin
danSinkin@protonmail.com

January 10, 2023

Contents

1 Introduction: What is this?

1.1 Motivation: Why are we doing this?

> A journey of a thousand miles
> begins with a single step
>
> *Daniel Lauzi*

Checkers is among the most popular board games and unlike chess it isn't particularly complicated. As such it's a great choice for a first project, it's reasonably complex to actually have to think through different engineering questions, but not so complex that one gets bogged down in minutae.

1.2 Goal: What do we want to achieve?

Once completed this project will represent a entirely playable and polished implementation of the classic game, which can be enjoyed either alone, against a (rudimentary) AI opponent or against other players via the internet.

1.3 What comes afterwards?

The underlying gameplay and networking logic can be used to build a more complicated board game like chess, which would be a bigger programming challenge.

A very rudimentary AI could be implemented by the opponent randomly choosing legal move. This can be improved (significantly) by implementing a heuristic based minimax algorithm, and of course by leveraging existing (or even training a new) AI models, possibly of different difficulties.

1.4 Glossary

Term	Description
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
ORM	Object Relational Mappe"
SQLAlchemy	SQL Toolkit and ORM for Python
Web Framework	Software architecture that for developping web applications
Flask	A web framwork written in Python
WSGI	"Web Server Gateway Interface". An interface between the web server and the web application that Flask uses
Werkzeug	WSGI toolkit used for implementing requests, response objects, and other utilities
Jinja2	Templating Engine used by Flask
Flask-SQLAlchemy	Flask Extension to be able to use SQLAlchemy

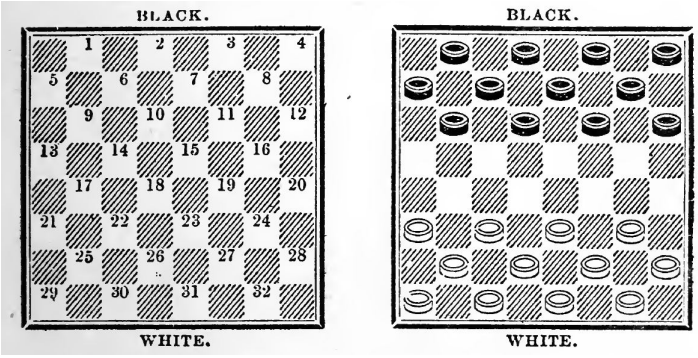
2 Game Logic

Given that checkers (also known as draughts among many other names) is an ancient game (c.f. [2], [3]) there are many different subtle (and not so subtle) difference between the rules of the game, the version that we will implement will be based on [1], [4] and [6].

The actual game logic will be implemented in Python as computational performance is not a concern with this game. There will be two versions, a terminal version and a website version with a functioning GUI.

2.1 Encoding and Rendering

While usually played on a 8×8 chessboard the actual game of checkers only occurs on the white spaces, so there are 32 potential spaces, following the enumeration in [1] these have the following form:



White is always the starting player. There are 8, we enumerate them as $0, \ldots, 7$ from top to bottom, so the pieces start from the 2. position in every even position and in the 1. position on odd indices.

A empty spot can have 5 different states

'.'	Empty space
'w'	White piece
'W'	White king
'b'	Black piece
'B'	Black king

Given the small amount of datasets any further optimizing of the internal represen-

tation at the cost of useability isn't going to be worth it.

A `board` is a 32 list of chars (in Python strings 1 element strings have a size of 50, so it's 1600 `bit` ≈ 0.2 `kB`).

2.2 Game Loop

There are five types of squares

Type	Squares)
Interior	$\{6, ..., 26, 27\}$
Left Side	$\{5, 13, 21\}$
Right Side	$\{12, 20, 28\}$
White End	$\{1, 2, 3\}$
Black End	$\{30, 31, 32\}$
Corner 1	$\{4\}$
Corner 2	$\{29\}$

Note that the categories are chosen such that they are disjoint. White is the first to move then the players alternate until the game ends. A player can either move or (if possible) capture a piece.

2.2.1 Moving

A given piece can moves into one of its empty neighbors, white pieces always have to decrease their position, black pieces increase it.

The neighbors are precomputed with `neighbors.py` script which writes the neighbors in a python compatible format into a text file, which then got pasted in to the main code. As there are no plans to support other board sizes (6×6, 10×10, 12×12 and so on) the script is not written in the most general possible way.

2.2.2 Capturing

If a valid move neighbor is occupied by a piece and the slot behind it is empty the piece can jump over (/capture) this piece to the free spot. If then after jumping there is the same thing again then the piece jumps again.

If there is the possibility to capture then the player has to capture, although he is free to choose with which piece he captures if there are multiple possibilities.

2.2.3 Kings and Promotion

If a white (resp. black) piece reaches the black (resp. white) end it will be promoted to a king (encoded by a uppercase letter). A king can increase and decrease their position by moving and capturing, i.e. they can move fowards and backwards. But still just one step.

2.2.4 End of Game

A player has won if either the opposing player has no more pieces left or has no valid moves. There are no draws.

2.3 IO

2.3.1 Saving

A board can be saved in text file with the following encoding: The first line is the name of the boardstate, the second line encodes the current turn (starting from 1, so if it's odd then white is the current player), the name of the white and of the black player. The 3rd to 10th line is for the current boardstate.

```
    startingPosition.txt    alma_NO_MOVES.txt
------------------------    ---------------------
    Starting Position       Starting Position
    1,Alice,Bob             6,Alma,Bernd
    bbbb                    bb.b
    bbbb                    bbbb
    bbbb                    bbbb
    ....                    ..b.
    ....                    w.w.
    wwww                    w..w
    wwww                    wwww
    wwww                    wwww
```

These will be rendered as

```
    .b.b.b.b                .b.b...b
    b.b.b.b.                b.b.b.b.
    .b.b.b.b                .b.b.b.b
    ........                ..b.....
```

```
........                    .W...W..
w.w.w.w.                    w.....w.
.w.w.w.w                    .w.w.w.w
w.w.w.w.                    w.w.w.w.
```

At the end of the encoding, optionally, the sequence of moves that was played is written down. Following the notation of [1] the steps are encoded as `"10 15"`.

```
alma.txt
--------------------
    Starting Position
    6,Alma,Bernd
    bb.b
    bbbb
    bbbb
    ..b.
    w.w.
    w..w
    wwww
    wwww
    11 15,23 19,8 11,22 17,3 8
```

2.3.2 Loading

The loading function imports boards which have the exact formatting given in the Saving section.

3 Web development

We will be using Flask to develop the website.

4 Database

The database will be managed by the Flask extension for SQLAlchemy.

5 References

1. A complete guide to the game of draughts or checkers, J. W. Dawson
    - https://archive.org/details/completeguidetog00lees/
2. Gameplay and design, K. Oxland
    - https://archive.org/details/gameplaydesign0000oxla
3. https://www.worldchampionshipcheckers.com/2018/04/19/the-history-of-the-game-checkers/
4. https://checkerslounge.com/rules-of-checkers.html
5. https://www.officialgamerules.org/checkers
6. How to Win at Checkers, F. Reinfeld

5.1 Tutorials and Educational Material

- Design Doc
    - https://www.slideshare.net/RaviYasas/example-for-sds-document-in-software-engineer
    - https://www.youtube.com/watch?v=FLtqAi7WNBY
- Flask
    - https://www.youtube.com/watch?v=dam0GPOAvVI

A Project debriefing and retrospective

Note that this is not part of the actual design document, but rather personal commentary on the project itself.

A.1 What has been learned

A.2 What can be done better

A.3 Where to go from here?