Informatik Dokumentation „Schach.py“

Julian Sehbaoui S4 MINT

28. Mai 2021

Besondere Lernleistung: Fachbereich Informatik

https://github.com/JSehbaoui/Chess_Python

Made with LaTeX

Inhaltsverzeichnis

1 Vorwort

Dieses Projekt findet im Rahmen des Informatikunterrichts des Luisengymnasiums Bergedorf statt und wird als zusätzliche Lernleistung entsprechend gewertet. Ich bin Julian Y. Sehbaoui und habe ein Schachprogramm „from scratch“ in Python 3 geschrieben.

2 Mein Ziel

Mein Ziel war es, ein vollfunktionales Schachspiel ausschließlich in Python 3 zu erschaffen, welches alle klassischen Regeln von Schach beinhaltet. Hierbei wollte ich auf jegliche äußerliche Hilfe, wie zum Beispiel API's oder schon existierende Algorhytmen, verzichten, mit Ausnahme eines Schachcomputers, gegen den man spielen kann. Außerdem soll man zwischen PvP[1] und PvE[2] wählen können, um das Endprodukt entweder mit Freunden oder auch alleine benutzen zu können.

3 Ideenfindung

Ich persönlich habe mich schon seit dem ich klein war für Schach interessiert. Es ist ein Spiel mit relativ simplen Regeln, welches aber durch seine zahlreichen Zugmöglichkeiten schnell sehr komplex werden kann. Als ich mich nach einem neuen Projekt umgesehen habe, kam mir direkt Schach in den Sinn, weil es an strikte Regeln gebunden ist und ich zu dem Zeitpunkt dachte, dass wenige Sonderregeln existieren. Außerdem wollte ich mich immer weiter vom Terminal entfernen und mehr mit GUI's[3] arbeiten. Ein weiterer Grund für meine Entscheidung war, dass mir Schach als Herausforderung in den Sinn kam, um meine aktuellen Fertigkeiten unter Beweiß zu stellen und mich selbst zu fordern.
Außerdem wollte ich eine Offline Schachversion für meinen Laptop haben damit ich nicht immer zu einer webbsierten Version zurückgreifen muss. Da mir aber die Alternativen nicht gefielen, habe ich mich an meine eigene Version gemacht. Sie sollte alles können, was die Onlinespiele können, also die Basisregeln beachten und eine benutzerfreundliche Oberfläche bieten.

4 Warum Schach?

Warum schreibe ich mein eigenes Schachprogramm, wenn es diese zuhauf online gibt und die meisten Versionen kostenlos sind? Zu nächst wollte ich mich wie bereits erwähnt mit einem neues Projekt fordern und vorallem mich mit GUI's vertraut machen. Zum anderen gibt es Features, die ich gerne bei der „Konkurenz“ gesehen hätte, zum Beispiel den Testmodus (später erläutert).

[1] Player versus Player
[2] Player versus Enviroment
[3] Graphical User Interfaces

5 Die Regeln im Schach

Die Regeln des modernen Schachspiels sind dokumtentiert und online abrufbar[5].

6 Benutze Pakete

Im Folgenden finden sich die verwendeten Pakete sowie deren Vorteile, welche mich zu der Entscheidung geführt haben.

6.1 pygame

„pygame“ [4] ist eine Ansammlung von Python 3-Modulen, welche auf das programmieren von Videospielen ausgelegt sind. Es ist auf sämtlichen OS verfügbar und hat schon mehrere Millionen Benutzer.

Warum pygame?

„pygame“ ist aufgrund der großen Community sehr gut dokumentiert. Außerdem basiert es auf SDL [1], was eine gute Input-Abfrage ermöglicht. Ein weiterer Grund für die Verwendung von „pygame“ war, dass ich schon ein wenig Erfahrung mit dem Modul sammeln konnte und daher für mich die erste oberflächliche Benutzung erleichtert wurde. Eine Alternative für mich war die „Processing“. „Processing“ ist ein „Sketchbook “ , welches einem erlaubt, auf einer graphischen Oberfläche zu programmieren. Es ist ein leichtverständliches Prinzip, da man hier mit der Syntax von entweder Java oder Python 3 funktionell programmieren kann. Außerdem, da Processing für graphische Arbeiten und kleine Spiele geschaffen wurde, lässt sich schnell und einfach die Mausposition und weitere Inputs abfragen und Figuren und Bilder auf der graphischen Oberfläche abbilden. Der Grund, warum ich mich dennoch für „pygame“ entschieden habe, war, dass man in „pygame“ objektorientiert [2] arbeiten kann. Dieses Konzept ist hilfreich bei der Programmierung eines Schachspiels, da sich alle Figuren an den selben „Bauplan“ halten und über das Prinzip der Vererbung die figurtypischen Eigentschaften in dieses Grundgerüst implementieren lassen.

Was habe ich verwendet?

Von dem Paket „pygame“ habe ich die „Display“-Methode benutzt, welche das Fenster erstellt hat. Außerdem habe ich die Funktion „draw“ benutzt, um Formen, wie zum Beispiel die einzelnen Quadrate des Bretts, Bilder, wie zum Beispiel die der Figuren, und Schriftzüge auf das Display zu bringen. Ebenfalls hilfreich war die Klasse „Surface“, mit der man einzelne kleine Bereiche abgrenzen konnte, um so mehr Struktur in das GUI zu bringen.

6.2 pygame-widgets

„pygame-widgets“ ist eine Art Erweiterung für „pygame“, welche Klassen für zum Beispiel Buttons und Schieberegler mitsich bringt.

Warum pygame-widgets

Die Erweiterung war sehr hilfreich, da ich somit einzelne Widgets nicht selber schreiben musste.

Was habe ich verwendet?

Von „pygame-widgets“ habe ich nur den Schieberegler für die Anpassung der Computerschwierigkeit verwendet. Obwohl ich Buttons verwendet habe, programmierte ich diese selber, da ich so viel mehr Freiheit hatte, was Funktion und Design anging. Für den Schieberegler hingegen war das Paket jedoch ausreichend, weswegen ich mir die Arbeit des Selberschreibens hier gespart habe.

6.3 Stockfish

„Stockfish“ [6] ist eine berühmte Schachengine, welche von vielen großen Schachanalysten und Schachwebsites, wie zum Beispiel `lichess.org` oder `chess.com`, benutzt wird.

Warum Stockfish?

Auf der Suche nach einer Schach-KI hatte ich gewisse ansprüche.

1. Die KI braucht eine Python 3-Unterstützung
2. Der Service soll nicht über eine API laufen, da ich das Spiel auch offline benutzen möchte
3. Die KI muss Open-Source sein.

Von meinen bisherigen Schacherfahrungen aus der Vergangenheit blieb mir der Name „Stockfish“ im Kopf. Als ich recherchierte sah ich, dass die „Stockfish“-Engine nicht nur Open-Source ist, sondern sogar in pip[4] vorhanden war. Somit hatte „Stockfish“ alle meine Anforderungen erfüllt und ich hatte keinen Grund mehr, nach Alternativen zu suchen. Zudem ist „Stockfish“ sehr populär unter den Schachspielern, weswegen ich mir eine gute Dokumentation erhoffte.
Alternativen zu „Stockfish“ wären noch Alpha Zero[5] sowie viele kleine selbstgeschriebene Engines. Jedoch überzeugte „Stockfish“ auch mit seiner hervorragenden Informationsübermittlung, da man zur Kommunikation zwischen meinem Skript und „Stockfish“ nur das Format geringfügig ändern musste und schon funktionierte die Engine einwandfrei.

[4] **P**ackage **I**nstaller for **P**ython
[5] Eine Schach-KI von Google

Was habe ich verwendet?

Von Stockfish habe ich die Klasse „Stockfish“ verwendet. Von dieser wiederum habe ich die Methode „get_best_move“ verwendet, um einen legalen und taktisch sinnvollen Zug für den Computer zu ermitteln.

7 Installation

Mein Skript kann kostenlos von der Seite github herutergeladen werden. Natürlich muss man, um das Python 3-Skript auszuführen, python3 installiert haben. Hierbei ist jede Version von Python 3 möglich, jedoch arbeitete ich mit Python 3.8.X, weswegen ich diese Version empfehlen würde. Um dieses dann zu starten sind die Pakete „Stockfish“, „pygame“ und "pygame-widgets"über pip herunterzuladen. Dies ist über die Konsole mit den Befehlen

```
pip install pygame
```

```
pip install stockfish
```

```
pip install pygame-widgets
```

zu erreichen.

8 Bedienung des Programmes

Im Folgenden erläutere ich die Bedienung des Programmes.

8.1 Wie spiele ich?

Um das Schachprogramm zu starten, muss man die „start.py“-Datei in Python 3 starten. Anschließend öffnet sich ein Fenster, in welchem man zwischen verschiedenen Spielmodi wählen kann.

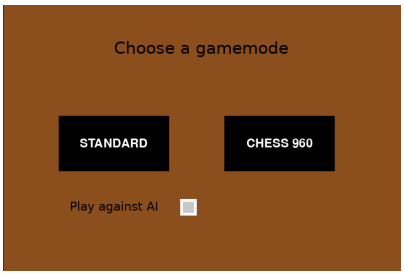

Abbildung 1: Startbildschirm *(Eigenaufnahme)*

Die beiden Modi, die hier zur Auswahl stehen, sind einmal die Standard-Schach-Variante und die Chess 960 Variante [3], bei der die Figuren in der hinteren Reihe zufällig angeordnet werden. Für jeden der beiden Modi lässt sich optional ein zweiter, virtueller Spieler hinzufügen, falls man keinen realen Gegner bei sich hat. Das kann durch das Klicken des grauen Feldes neben der Nachricht „Play against AI“ erreicht werden. Nachdem man sich für einen Modus entschieden hat, lässt sich der Spielername, und je nach Option die Computerschwierigkeit oder der Name des zweiten Spielers, einstellen.

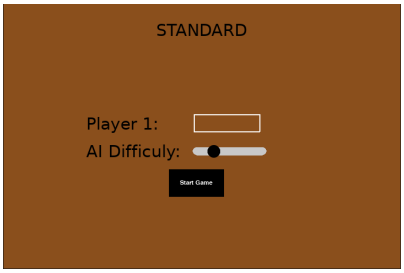

Abbildung 2: Modus mit Bot *(Eigenaufnahme)*

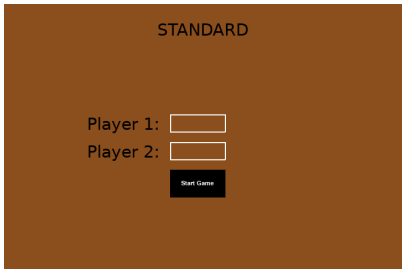

Abbildung 3: Modus ohne Bot *(Eigenaufnahme)*

Nun kann das Spiel durch den Knopf unterhalb der Einstellungen gestartet werden und das tatsächliche Spiel kann beginnen. Sobald man auf eine Figur klickt, öffnen sich auf dem Feld alle legalen Zugoptionen.

Abbildung 4: Das Spielbrett nach klicken auf e2

Wenn man anschließend auf ein Feld klickt, welches die Figur betreten kann, bewegt sich die ausgewählte Figur auf das gewünschte Feld.

8.2 Overlays

Spielerinformationstabs

Im oberen Bereich des Spielfelds befindet sich zum einen eine Uhr, welche die aktuelle Spielzeit anzeigt. Links und rechts danaben befinden sich die Spielerinformationen.

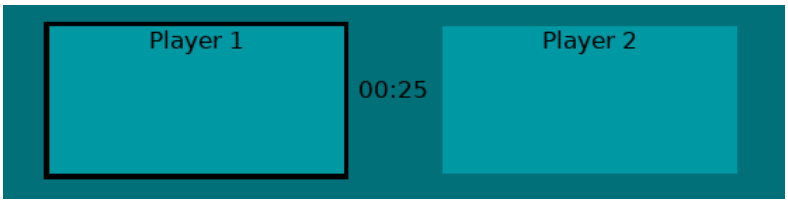

Abbildung 5: Informations-Tabs und Spielzeit

In diesen werden die geschlagenen Figuren sowie der Spielername angezeigt. Außerdem wird das Spielerinformationsfeld von dem Spieler hervorgehoben, welcher an der Reihe ist, den nächsten Zug zu machen.

Spielhistorie

In dem rechten Teil des Fensters befindet sich die Spielhistorie. Diese zeigt die bereits gezogenen Spielzüge in der „Ausführlichen Notation“ an.

Buttons

In der rechten, oberen Ecke finden sich insgesammt vier Buttons mit folgenden Funktionen:

- „Quit“-Button
  - Der Quit-Button beendet sofort das Spiel und schließt das Fenster
- „Resign“-Button
  - Beendet das Spiel mit einem Sieg für den anderen Spieler
- „Takeback“-Button
  - Macht den letzten Zug rückgängig
  - Ist nur zu benutzen, wenn keine Figur im letzten zug geschlagen wurde, um große Patzer zu bestrafen
- „Testmodus“
  - Bei erstmaligem Drücken betritt man einen Testmodus, indem es einem möglich einem möglich ist, Züge auszutesten.
  - alle Züge revidiert und die Figuren auf ihre Ausgangsposition zu beginn des Testens zurückgestellt

Abbildung 6: Bedienungs-Knöpfe

9 Code

In dem folgenden Kapitel werde ich das Konzept meines Codes erläutern und die Entwicklung darstellen.

9.1 Klassenbeziehungen

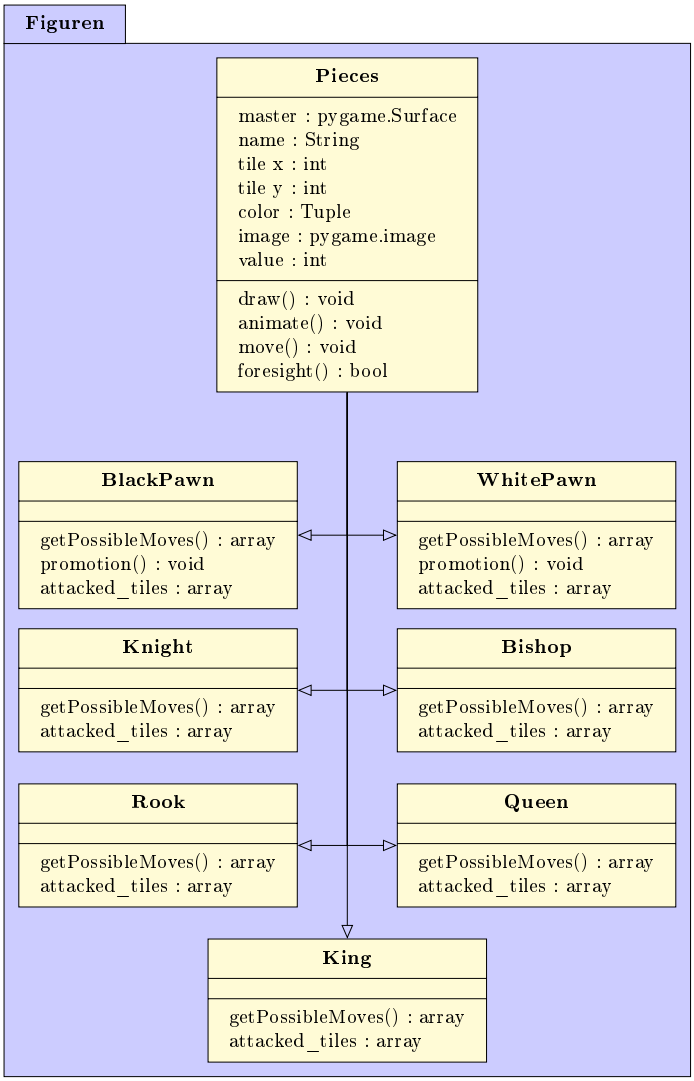

Abbildung 7: UML-Diagram der Figurenklassen

Oben zu sehen ist ein UML-Diagramm, um die Beziehungen von den verschiedenen Klassen zu visualisieren. Zu erkennen ist eine Parentclass „Pieces“, welche alle Methoden enthällt, die für eine beliebige Schachfigur von nöten sind. Von dieser Klasse erben insgesammt sieben andere Klassen. Jede Klasse hat ihre eigene Methode, um die möglichen Züge zu berechnen. Getrennt sind hierbei die Bauern, in „BlackPawn“ und „WhitePawn“. Das ist nötig, da sich die Bauern der verschiedenen Teams als einzige Figur nicht exakt so wegbewegt, wie die des gegnerischen Teams. So bewegt sich der weiße Bauer aufsteigend der Reihen, wohingegen sich der schwarze Bauer absteigend der Reihen bewegt.

9.2 Probleme und Problemlösungen

Während des Programierens bin ich auf viele Probleme gestoßen, die mich an meine Grenzen gebracht haben, welche ich aber immer wieder überwinden konnte. Das wohl größte Problem war, dass viele Regeln Voraussetzungen mit sich bringen, die man beim normalen Spielen gar nicht aktiv bedenkt, welche aber beim Implementieren eine Hürde darstellten.

Das Verrücken des Spielbretts

Als ich grob mit dem Aufbau meines Schachprogramms fertig war, also die Figuren, welche sich noch ohne Einschränkungen bewegen konnten, auf einem richtig koloriertem Feld standen, wollte ich das Schachbrett verrücken, um Platz für die zukünftigen Overlays zu schaffen. Die ersten Versuche hatten zur Folge, dass die Figuren nicht mehr auf die Events, wie zum Beispiel einen Mausklick, reagierten.
Als das Problem behoben war, traten aber viele visuelle Probleme auf.

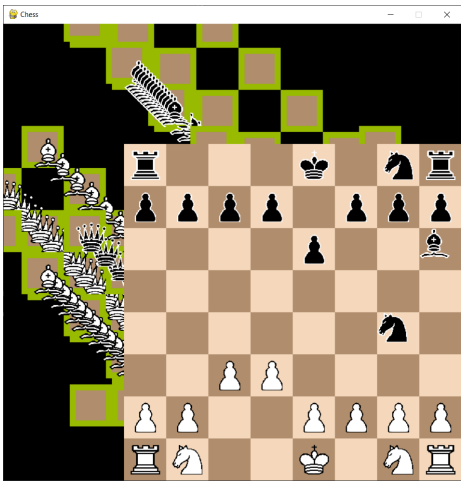

Abbildung 8: Graphischer Fehler

Das Problem ließ sich durch die weiteren Einschränkungen der Bewegungsfreiheit der Figuren, sowie das Hinzufügen eines Anker-Punktes an der oberen-rechten Ecke des Spielbretts beheben.

Der König, das Problemkind

Das wohl größte Problem war der König. Dieser muss nämlich, abhängig von der Stellung des Bretts, in seinen Bewegungen eingeschränkt werden. Die Lösung der Felderbegrenzung konnte durch einen simplen Filter gelöst werden. Das erste, große Problem kam nun aber mit dem Blocken eines Schachs. Wenn eine Figur zwischen König und der angreifenden Figur steht, dann ist das Schach aufgelöst. Nur was bedeutet nun „dazwischen “? Um dieses Problem zu lösen, musste ich den König entlang der Angriffslinien „tracken“. Als ich nach langer Zeit das Problem gelöst hatte, kam nun leider ein weiteres Problem hinzu. Der König konnte sich selbst blockieren bzw. das Schach im nächsten Zug verhindern, was wie folgt aussah:

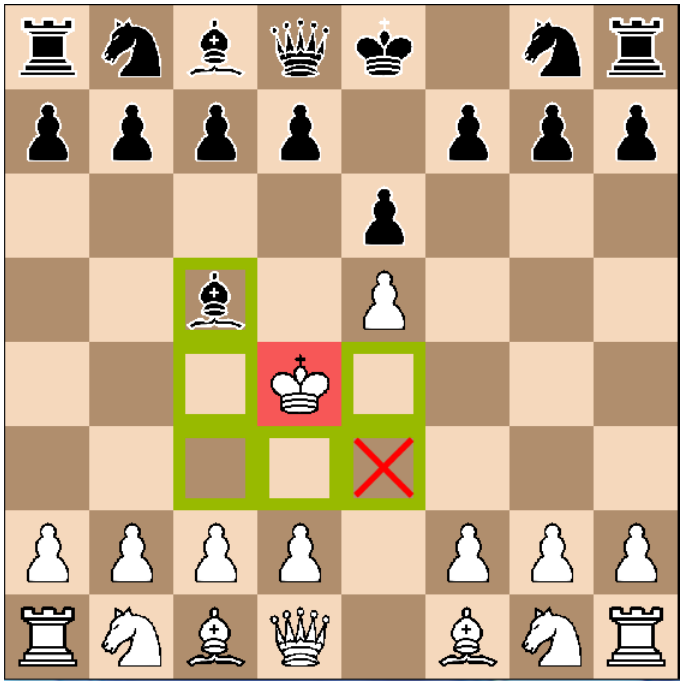

Abbildung 9: Bug, König verhindert Schach

Um dieses Problem zu lösen, hatte ich erst die Idee, den König zu einer „transparenten Figur“ zu machen, sodass der Angriffszug auch durch diesen hindurch geht. Als ich jedoch dabei war die dieses Problem zu lösen, taten sich viele weitere Bugs auf, wie zum Beispiel, dass man die Figur, die vor dem König steht und das Schach verhindert

wegbewegt werden darf, obwohl das kein legaler Zug ist. Aufgrunddessen suchte ich nach einer Lösung, welche alle nicht-legalen Züge verhindern würde.

Foresight

Wenn man es schaffen könnte, nach dem nächsten Zug in die Zukunft zu sehen und erkennen könnte, ob der Zug als Konsequenz hat, dass der König im Schach steht, könnte man diesen Zug verbieten. Genau das war der Gedanke hinter dieser Lösung. Ich lasse, bevor ich die möglichen Züge einer Figur exportiere, die Liste durch einen Filter laufen, der oben Erwähntes prüft. Der Filter sieht folgendermaßen aus:

```
def filter_method_foresight(self, move):
        self.x, self.y = move #setting the position of the piece to the move,
            that you want to check
        ignoring_piece = None #currently no piece has to be ignored

        ### if you want to simulate to take a piece,
        # without actually taking it, you can just
        ### ignore it in the .detectingCheck method
        same_pos = (self.x, self.y) == (piece.x, piece.y)
        same_piece = piece == self
        for piece in Pieces.all_pieces_list:
                if same_pos and not same_piece:
                        ignoring_piece = piece

        #checking if the king is checked, after the move
        Pieces.detectingCheck(ignoring_piece = ignoring_piece)

        white_check = bool(Pieces.white_is_checked)
        black_check = bool(Pieces.black_is_checked)
        white = bool(self.farbe == (255,255,255))
        black = bool(self.farbe == (0,0,0))

        #returning if the move is legal or not
        return (white and not white_check) or (black and not black_check)
```

Nachdem ich diesem Filter für jede Figur angewendet hatte, gab es keinerlei Probleme mehr mit illegalen Zügen.

Stockfish

Vorerst glaubte ich noch an die Einfachheit der Implementierung von Stockfish in Python 3, aber diese Vermutung erwies sich als falsch. Zunächst habe ich Stockfish wie in der Dokumentation beschrieben installiert und importiert. Als ich dann ein Objekt der Klasse „Stockfish“ erstellen wollte, bekam ich folgende Fehlermeldung:

```
Traceback (most recent call last):
File "D:\Julian\Coding\Sprachen\python\lib\site-packages\
    stockfish\models.py", line 33, in __init__
self.stockfish = subprocess.Popen(
File "D:\Julian\Coding\Sprachen\python\lib\subprocess.py", line 947,
    in __init__
self._execute_child(args, executable, preexec_fn, close_fds,
File "D:\Julian\Coding\Sprachen\python\lib\subprocess.py", line 1416,
    in _execute_child
hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
FileNotFoundError: [WinError 2] Das System kann die angegebene Datei
    nicht finden
Exception ignored in: <function Stockfish.__del__ at
    0x000001F7B592D820>
Traceback (most recent call last):
File "D:\Julian\Coding\Sprachen\python\lib\site-packages\
    stockfish\models.py", line 270, in __del__
self.stockfish.kill()
AttributeError: 'Stockfish' object has no attribute 'stockfish'
```

Als ich nach Lösungen gesucht hatte, war ich in einem Forum auf User gestoßen, die das gleiche Problem hatten. Glücklicherweise gab es die Möglichkeit, eine extrene Konsole in Form einer .exe Datei herunterzuladen und den Bot von dort aus zu starten.
Obwohl mit die Lösung nicht als sehr elegant erschien, wollte ich keine Änderungen vornehmen, da das Programm für jeden ausführbar sein sollte. Deshalb wollte ich nichts bei mir lokal in dem Paket „Stockfish“ ändern.

Rochade

Die Rochade [5] stellte sich anfangs als großes Problem heraus, da viele Bedingungen erfüllt sein müssen, um eine Rochade durchzuführen. Meine Lösung für das Problem war, dass ich für jede Bedingung welche erfüllt sein muss einen Boolean[6] erstellte, welchen ich vorerst auf „True“ setze. Danach ließ ich ein Skript jede Bedingung überprüfen und den Boolean auf „False“ ändern, wenn die Bedingung nicht erfüllt war. Ob eine Rochade dann möglich war, entschied ein entgültiger Boolean, welcher eine Konjunktion aller „Bedingungs-Boolean“ war.

[6] Zustandsvariable

10 Danke an...

Ich möchte zunächst dem Kollegium des Luisengymnasium, insbesondere Frau Swiebodzinski, dafür danken, dass ich das Projekt in diesem Rahmen bearbeiten durfte. Ein besonderer Dank geht an zwei Mitschüler, Gero Beckmann und Vincent Piegsa, da ich bei Fragen immer die Möglichkeit hatte, mich an sie zu wenden.
Außerdem möchte ich den Usern der Medien Youtube, Reddit, Stackoverflow und GitHub bedanken, da ich mir auch dort Hilfe holen konnte.

11 Schlusswort und Fazit

Dieses Projekt war sehr anspruchsvoll, da ich das Schachprogramm „from Scratch“ gestartet und auf jegliche Hilfen von API's verzichtet habe. Dennoch war dieses Projekt sehr umfangreich und spannend, da ich das erste Mal im Frontend entwickelt habe und ich, trotz vieler Hürden, meinen Spaß daran hatte.
Ich bin mit dem Endresultat sehr zufrieden, obwohl mir es bis Dato () nicht gelungen ist, die Regel „en passant“ zu implementieren. Somit habe ich mein Ziel, welches ich mir am Anfang gesetzt hatte, nicht erreicht, dennoch bin ich stolz auf das Produkt, da ich viele Hürden überwinden musste, um dieses Programm zu schreiben und es nun funktioniert.

11.1 Was habe ich gelernt?

Zum einen war dies eines meiner ersten, großen Projekte, welches ich alleine bewältigen musste. Dadurch erlernte ich nicht nur ein gutes Zeitmanagement, sondern auch Sorgfalt und Stuktur im Code zu haben und eine organisierte Ordnerstruktur zu pflegen.
Zum anderen habe ich gelernt mit Git[7] umzugehen, zu debuggen, mit GUI's zu arbeiten, und vieles mehr.

11.2 Zukünftige Aussichten

Wie bereits zuvor erwähnt, habe ich vor, die fehlende Regel einzubauen. Ebenfalls habe ich vor, den Code nocheinmal aufzuräumen und zu kommentieren, um ihn für Interessierte nachvollziebarer zu gestallten.

[7] https://git-scm.com

12 Quellen

Nicolas Kielbasiewicz, UMA/POems, ENSTA - ParisTech, TikZ- UML - Package perso.ensta-paris.fr

Literatur

[1] *About - SDL*. WebPage. URL: https://www.libsdl.org.

[2] Gregor Rayman Bernhard Lahres. *Objektorientierte Programmierung*. Rheinwerk Verlag GmbH 2009. URL: https://openbook.rheinwerk-verlag.de/oop/index.htm.

[3] User: Lukas. *Chess960 – Aufbau, Regeln und Geschichte*. WebPage. URL: https://schachmatt.net/ratgeber/chess960/.

[4] pygame. *About - pygame*. WebPage. URL: https://www.pygame.org/wiki/about.

[5] Deutscher Schachbund. *Schachregeln und Spielweise*. WebPage. URL: https://cutt.ly/FbEV9EO.

[6] Daylen Yang. *About - Stockfish*. WebPage. 2010-2021. URL: https://stockfishchess.org/about.