

Informatik Dokumentation

"Schach.py"

Julian Sehbaoui

18. April 2021

https://github.com/JSehbaoui/Chess_Python



Made with L^AT_EX

Inhaltsverzeichnis

1	Vorwort	3
2	Das Ziel	3
3	Ideenfindung	3
4	Benutze Pakete	3
4.1	pygame	3
4.1.1	Warum pygame?	3
4.2	Stockfish	4
4.2.1	Warum Stockfish?	4
5	Installation	4
6	Bedienung des Programmes	4
6.1	Wie spiele ich?	4
6.2	Overlays	5
6.2.1	Spielerinformationstabs	5
6.2.2	Spielhistorie	5
6.2.3	Buttons	5
7	Code	6
7.1	Klassenbeziehungen	7
7.2	Probleme und Problemlösungen	8
7.2.1	Der König, das Problemkind	8
7.2.2	Foresight	8
7.2.3	Stockfish	10
8	Danke an . . .	10
9	Schlusswort und Fazit	11
9.1	Zukünftige Aussichten	11

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 python3 geschrieben.

2 Das Ziel

Mein Ziel war es ein vollfunktionalles Schachspiel ausschließlich in Python zu erschaffen, welches alle klassischen Regeln des Schachs beachtet. Hierbei wollte ich auf jegliche äußerliche Hilfe, wie zum Beispiel API's oder schon existierende Algorithmen verzichten, mit Ausnahme eines Schachcomputers. Außerdem soll man zwischen PvP (Player versus Player) und PvE (Player versus)

3 Ideenfindung

Ich persönlich habe mich schon seit dem ich klein bin für Schach interessiert. Es ist ein recht simples Spiel, welches aber durch seine zahlreichen Zugmöglichkeiten zu einem sehr tiefgründigen Spiel 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, nicht viele Ausnahmen existieren und ich es wie schon erwähnt sehr gerne spiele.

4 Benutze Pakete

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

4.1 pygame

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

4.1.1 Warum pygame?

„pygame“ ist aufgrund der großen Community sehr gut dokumentiert. Außerdem bietet es eine wundervolle SDL (Simple Direct Media) Bibliothek. 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 leicht-verständliches Prinzip, da man hier mit der Syntax von entweder Java oder Python

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. Des Grund warum ich mich dennoch für pygame entschieden habe war, dass man in pygame Objektorientiert arbeiten kann. Kurz gesagt verwendet man bei der Objektorientierten Programmierung Grundgerüste (sogenannte Klassen), mitwelchen sich Objekte mit durch die Klasse definierten Attributen erstellen lassen. Dieses Konzept ist hilfreich bei der Programmierung eines Schachspiels, da sich alle Figuren an den selben „Bauplan“halten und sich über das Prinzip der Vererbung sich die Figurtypischen Eigenschaften in dieses Grundgerüst implementieren lassen.

4.2 Stockfish

Stockfish ist eine berühmte Schachengine, welche von vielen großen Schachanalysten und Schachwebsites benutzt wird.

4.2.1 Warum Stockfish?

Stockfish hat eine gute Python Unterstützung. Außerdem ließ es sich nach ein paar wenigen Komplikationen auch gut benutzen und implementieren.

5 Installation

Mein Skript ist kostenlos von der Seite [github](#) heruntergeladen werden. Offensichtlicher Weise muss man um das Python-Skript auszuführen, [python3](#) herunter laden. Um dieses dann zu starten sind die Pakete „Stockfish“, „pygame“ und "pygame-widgets"über [pip](#) herunter zuladen. Dies ist über die Konsole mit den Befehlen

```
pip install pygame
```

```
pip install stockfish
```

```
pip install pygame-widgets
```

zu erreichen.

6 Bedienung des Programmes

6.1 Wie spiele ich?

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

Die beiden Modi die hier zur Auswahl stehen sind einmal die Standard-Schach-Variante und die Chess 960 Variante, 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. 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. 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. Wenn man anschließend auf ein Feld klickt, welches die Figur betreten kann, bewegt sich die ausgewählte Figur auf das gewünschte Feld.

6.2 Overlays

6.2.1 Spielerinformationstabs

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

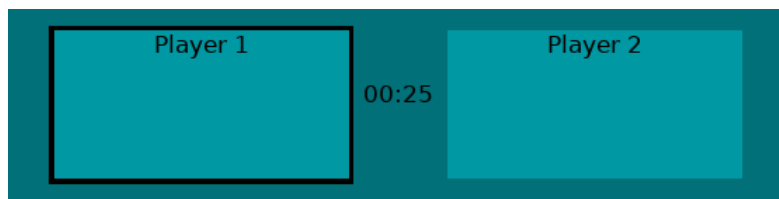


Abbildung 1: Informations-Tabs und Spielzeit

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

6.2.2 Spielhistorie

In dem rechten Teil des Fensters befindet sich die Spielhistorie. Diese zeigt die bereits gezogenen Spielzüge an.

6.2.3 Buttons

Oben im rechten Eck finden sich insgesamt drei Buttons mit folgenden Funktionen:

- „Quit“-Button
 - Der Quit-Button beendet sofort das Spiel und schließt das Fenster.



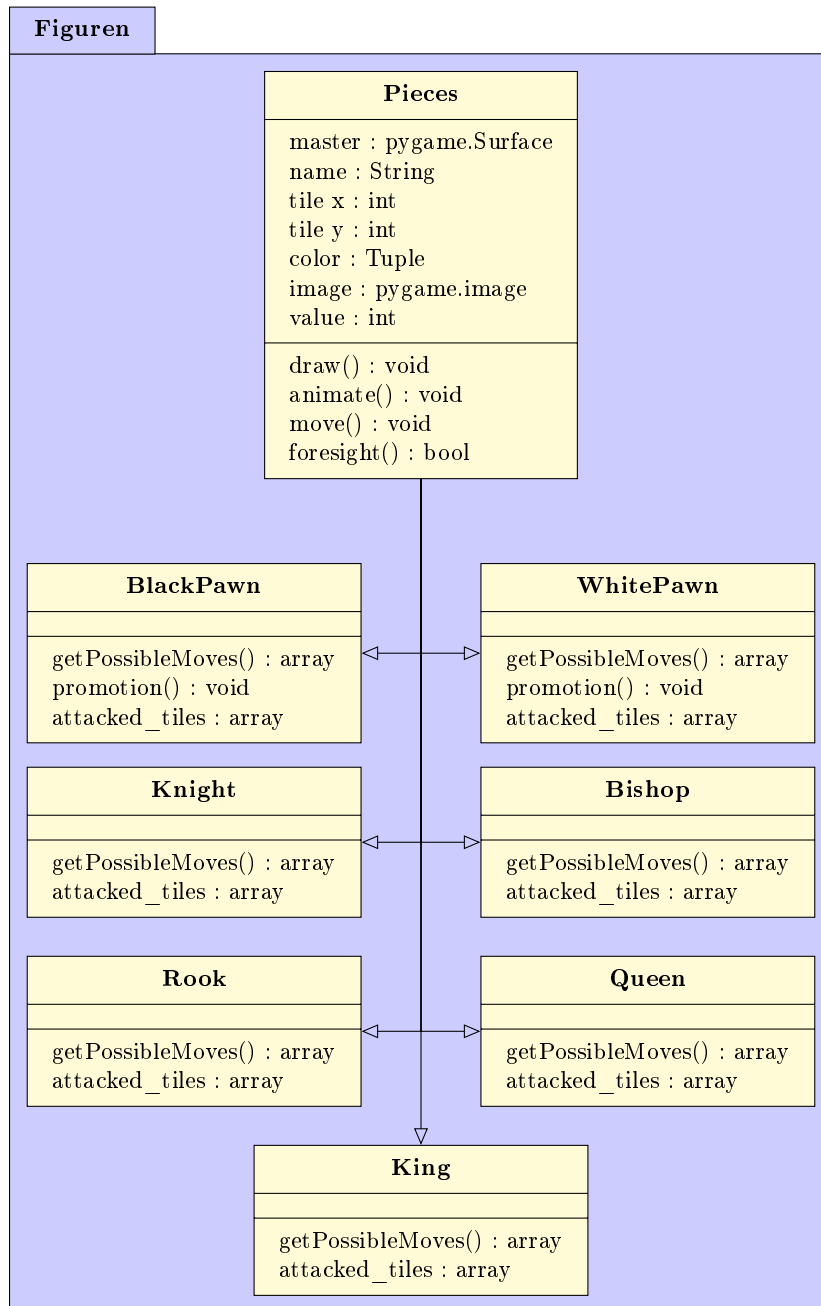
Abbildung 2: Bedienungs-Knöpfe

- „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.

7 Code

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

7.1 Klassenbeziehungen



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 enthält, die für eine beliebige Schachfigur vonnöten sind. Von dieser Klasse erben insgesamt 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 war 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.

7.2 Probleme und Problemlösungen

Während des Programmierens bin ich auf viele Probleme gestoßen, die mich an meine Grenzen trieben, 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.

7.2.1 Der König, das Problemkind

Das wohl größte Problem war der König. Dieser muss nämlich abhängig von der gegnerischen Position der Figuren 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 einher. Der König konnte sich selbst blockieren bzw. das Schach im nächsten Zug verhindern, was wie folgt aussah:

Um dieses Problem zu lösen hatte ich zunächst die Idee den König zu einer "transparenten Figur" zu machen, sodass der Angriffszug auch durch diesen hindurch wirkt, doch ich sah immer mehr Probleme kommen, die wieder aufwand gemacht hätten zu lösen, wie zum Beispiel, dass man die Figur, die vor dem König steht und das Schach verhindert nicht wegbewegt werden darf. Dann kam mir aber die Idee, wie ich all diese Probleme auf einen Schlag lösen konnte.

7.2.2 Foresight

Wenn man es schaffen könnte, in die Zukunft nach dem nächsten Zug 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):
```

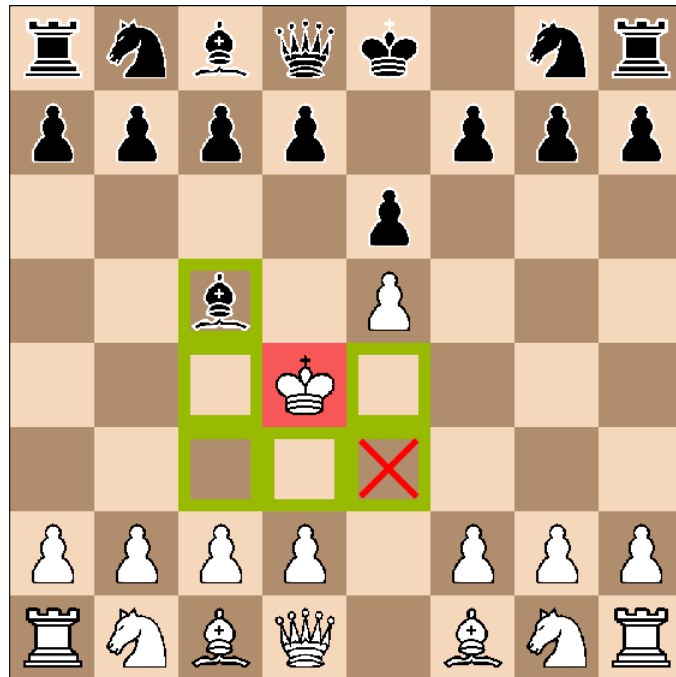



Abbildung 3: Bug, König verhindert Schach

```

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
for piece in Pieces.all_pieces_list:
    if (self.x, self.y) == (piece.x, piece.y) and not (piece == self):
        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 = bool(self.farbe == (255,255,255))
black_bool = bool(self.farbe == (0,0,0))

#returning if the move is legal or not
return (white_bool and not white_check) or (black_bool and not

```

```
black_check)
```

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

7.2.3 Stockfish

Zunächst hatte ich noch an die Leinfachheit der Implementierung von Stockfish in python geglaubt, aber ich sollte falsch liegen. Zunächst hatte 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 selbe Problem hatten. Glücklicher Weise gab es die Möglichkeit eine .exe Datei herunterzuladen und den Bot von dort aus zu starten.

Ich fand die Lösung war nun nicht die Eleganteste, aber da das Programm für jeden auszuführen sein soll, wollte ich nichts bei mir lokal bei dem Paket „Stockfish“ ändern.

8 Danke an...

Ich möchte zunächst dem Luisengymnasium und dem Kollegium, 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 mich 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.

9 Schlusswort und Fazit

Dieses Projekt war sehr anspruchsvoll, da ich das Schachprogramm „from Scratch“ gestartet habe 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.

9.1 Zukünftige Aussichten

Wie eben erwähnt habe ich vor die mir fehlende Regel einzubauen. Ebenfalls hatte ich vor den Code noch einmal aufzuräumen und zu kommentieren, um ihn für Interessierte nachvollziehbarer zu gestalten. Eine weitere Idee von mir war, dass ich eventuell das Programm auf meine eigene Website hochlade und somit ein Onlinespiel zu schaffen.