

# Informatik Dokumentation "Schach.py"

Julian Sehbaoui

27. April 2021

[https://github.com/JSehbaoui/Chess\\_Python](https://github.com/JSehbaoui/Chess_Python)



Made with L<sup>A</sup>T<sub>E</sub>X

## Inhaltsverzeichnis

<b>1</b>	<b>Vorwort</b>	<b>3</b>
<b>2</b>	<b>Mein Ziel</b>	<b>3</b>
<b>3</b>	<b>Ideenfindung</b>	<b>3</b>
<b>4</b>	<b>Die Regeln im Schach</b>	<b>3</b>
<b>5</b>	<b>Benutze Pakete</b>	<b>3</b>
5.1	pygame . . . . .	4
5.1.1	Warum pygame? . . . . .	4
5.2	Stockfish . . . . .	4
5.2.1	Warum Stockfish? . . . . .	4
<b>6</b>	<b>Installation</b>	<b>5</b>
<b>7</b>	<b>Bedienung des Programmes</b>	<b>6</b>
7.1	Wie spiele ich? . . . . .	6
7.2	Overlays . . . . .	7
7.2.1	Spielerinformationstabs . . . . .	7
7.2.2	Spielhistorie . . . . .	8
7.2.3	Buttons . . . . .	8
<b>8</b>	<b>Code</b>	<b>9</b>
8.1	Klassenbeziehungen . . . . .	9
8.2	Probleme und Problemlösungen . . . . .	10
8.3	Das Verrücken des Spielbretts . . . . .	10
8.3.1	Der König, das Problemkind . . . . .	11
8.3.2	Foresight . . . . .	12
8.3.3	Stockfish . . . . .	13
<b>9</b>	<b>Danke an...</b>	<b>13</b>
<b>10</b>	<b>Schlusswort und Fazit</b>	<b>14</b>
10.1	Was habe ich gelernt? . . . . .	14
10.2	Zukünftige Aussichten . . . . .	14

## 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 Mein Ziel

Mein Ziel war es, ein vollfunktionales Schachspiel ausschließlich in Python zu erschaffen, welches alle klassischen Regeln von Schach beinhaltet. Hierbei wollte ich auf jegliche äußerliche Hilfe, wie zum Beispiel API's oder schon existierende Algorithmen, verzichten, mit Ausnahme eines Schachcomputers, gegen den man spielen kann. Außerdem soll man zwischen PvP (Player versus Player) und PvE (Player versus Environment) 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 bin 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 es wenige Sonderregeln existieren. Außerdem wollte ich mich immer weiter vom Terminal entfernen und mehr mit GUI's (Graphical User Interfaces) arbeiten. Ein weiterer Grund für meine Entscheidung war, dass mir Schach als Herausforderung in den Sinn kam, um meine aktuellen Fertigkeiten unter Beweis zu stellen und mich selbst zu fordern.

## 4 Die Regeln im Schach

Die Regeln des Schachspiels finden sich unter

"\dokumentation\rules.pdf"

## 5 Benutze Pakete

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

## 5.1 pygame

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

### 5.1.1 Warum pygame?

„pygame“ ist aufgrund der großen Community sehr gut dokumentiert. Außerdem bietet es eine wundervolle SDL-Bibliothek(Simple Direct Media). 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 „Scetchbook“, 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. Der Grundm, 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), mittels welchen 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 über das Prinzip der Vererbung die figurtypischen Eigenschaften in dieses Grundgerüst implementieren lassen.

## 5.2 Stockfish

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

### 5.2.1 Warum Stockfish?

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

1. Die KI braucht eine python-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 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 (Eine Schach-KI von Google) 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.

## 6 Installation

Mein Skript kann kostenlos von der Seite [github](#) heruntergeladen werden. Natürlich muss man, um das Python-Skript auszuführen, [python3](#) installiert haben. Hierbei ist jede Version von python3 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.

## 7 Bedienung des Programmes

### 7.1 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.

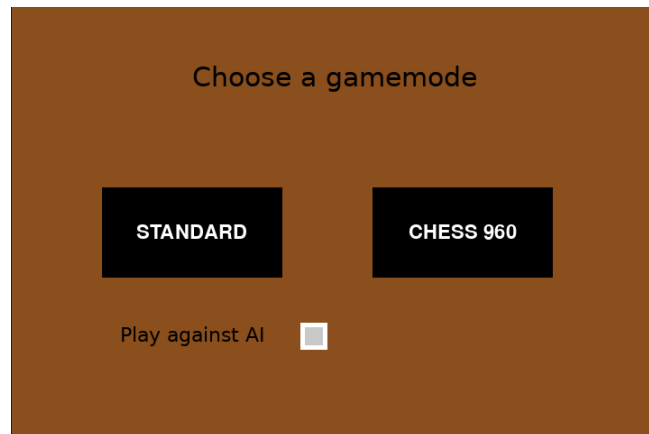


Abbildung 1: Startbildschirm (*Eigenaufnahme*)

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. 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*)

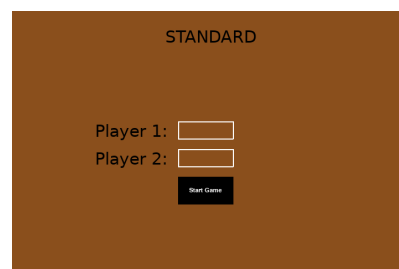


Abbildung 3: Modus ohne Bot (*Eigenaufnahme*)

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.

## 7.2 Overlays

### 7.2.1 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.

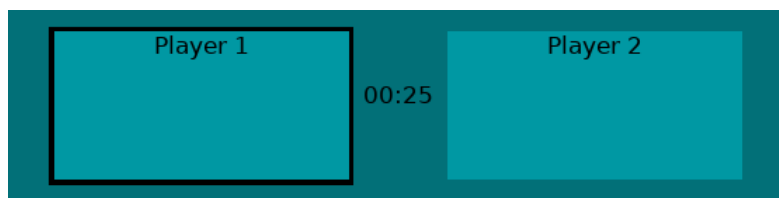


Abbildung 5: Informations-Tabs und Spielzeit

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.

### 7.2.2 Spielhistorie

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

### 7.2.3 Buttons

In der rechten, oberen Ecke finden sich insgesamt drei 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



Abbildung 6: Bedienungs-Knöpfe



## 8 Code

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

### 8.1 Klassenbeziehungen

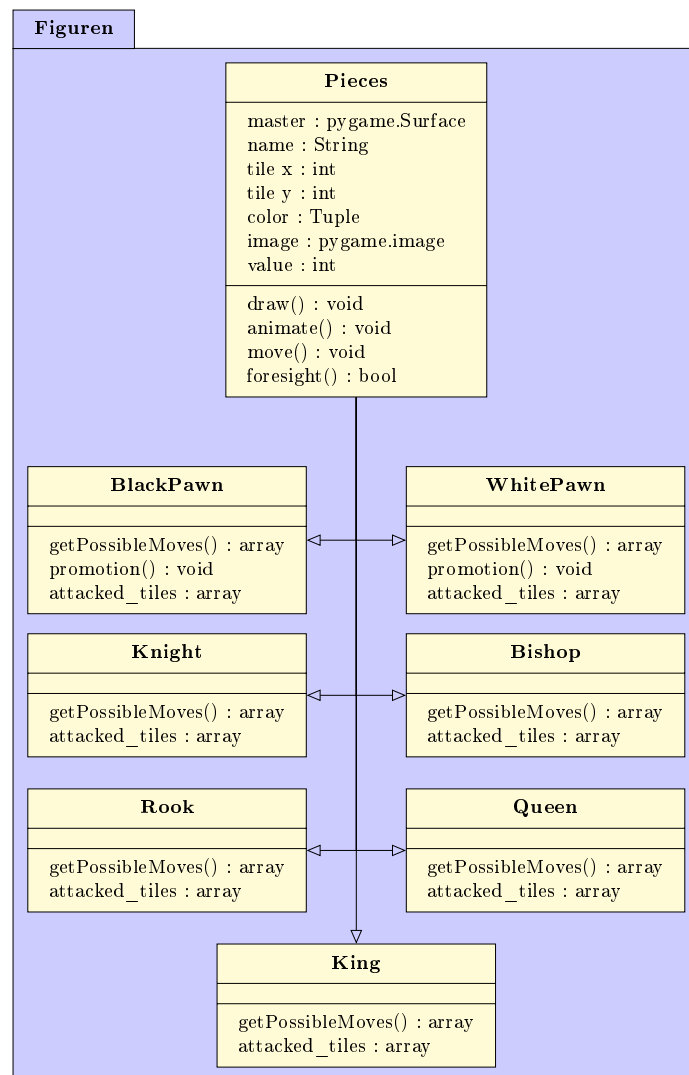


Abbildung 7: UML-Diagramm der Figurenklassen

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 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 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.

## 8.2 Probleme und Problemlösungen

Während des Programmierens 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.

## 8.3 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.

### 8.3.1 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:

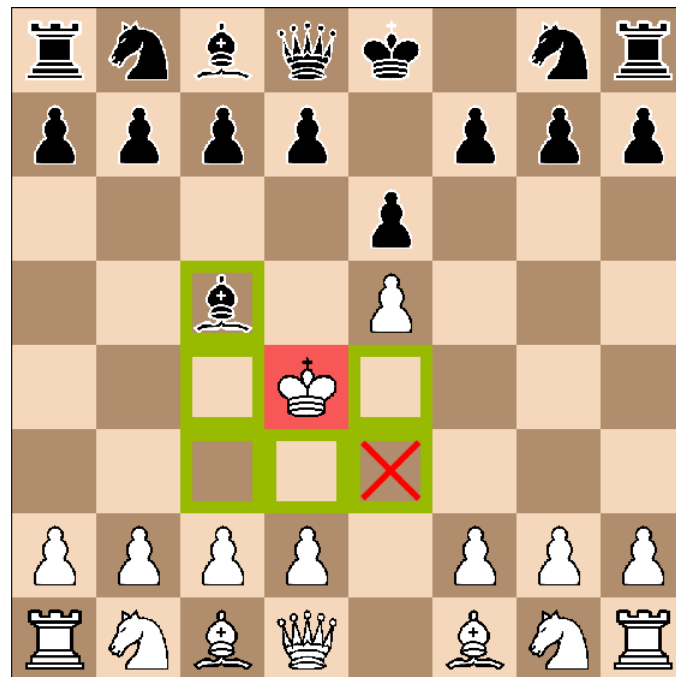


Abbildung 9: Bug, König verhindert Schach

Um dieses Problem zu lösen, hatte ich zunächst 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.

### 8.3.2 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.

### 8.3.3 Stockfish

Zunächst glaubte ich noch an die Einfachheit der Implementierung von Stockfish in python, 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 extreme 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 auszuführen sein soll. Deshalb wollte ich nichts bei mir lokal bei dem Paket „Stockfish“ ändern.

## 9 Danke an...

Ich möchte zunächst dem Kollegium des Luisengymnasiums, 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.

## 10 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 einwandfrei zu benutzen ist.

### 10.1 Was habe ich gelernt?

Zum einen war dies eines meiner ersten, großen Projekte, welches ich alleine bewältigen musste.

Zum anderen habe ich gelernt mit Git umzugehen, zu debuggen, meinen Code zu organisieren und zu strukturieren, mit GUI's zu arbeiten, und vieles mehr.

### 10.2 Zukünftige Aussichten

Wie bereits zuvor erwähnt habe ich vor, die mir fehlende Regel einzubauen. Ebenfalls hatte ich vor, den Code nochmal 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 anbieten zu können.