python-chess_evaluation

March 26, 2019

```
In [1]: from IPython.core.display import HTML, display
        HTML("""
        <style>
        svg {
            width:40% !important;
            height:40% !important;
        }

        .container {
            width:100% !important;
        }
        </style>
        """)
```

```
Out[1]: <IPython.core.display.HTML object>
```

```
In [2]: # %autosave 0
        %config IPCompleter.greedy=True
```

1 Python-Chess Evaluierung

In diesem Notebook wird die "python-chess-library", oder auch "chess core" genannt, Bibliothek evaluiert. Der "chess core" soll Funktionen zum Erstellen des Schachbretts, berechnen der erlaubten/möglichen Züge, durchführen der Züge etc. bereitstellen.

Dabei wird jede notwendige Funktion getestet und dessen Realisierung dokumentiert. Alle notwendigen Funktionen sind folgend gelistet.

- Schachbrett erstellen
- Schachbrett als ASCII ausgeben
- Züge auf dem Schachbrett ziehen
- Erlaubte Züge berechnen
- Auf Schach/Schachmatt prüfen
- Testen, ob Rochade, En Passant als erlaubte Züge gelistet werden
- *(Optional)* Schachbrett als SVG in JupyterNotebook ausgeben
- *(Optional)* Möglichkeiten evaluieren, Schachbrett je nach Positionierungen eindeutig identifizierbar in CSV zu schreiben

Des Weiteren werden folgende Aktionen/Berechnungen auf deren Umsetzbarkeit getestet. Dabei wird herausgefunden werden was nötig ist, um die aufgelisteten Aktionen mit der Bibliothek durchzuführen.

- Berechnen & Ausgeben erlaubter Züge für eine spezielle Figur
- Abwechselnder User & KI Input auf demselben Board
- Berechnen des Wertes des Schachbretts und der attackierten Figuren
- Speichern von Zügen/Entwicklung eines Schachbretts inklusive Sieg/Niederlagen als CSV-Datei

1.1 Vorraussetzungen

Folgend werden die allgemeinen Vorraussetzungen für die Verwendung und Installation der "python-chess-library" erläutert.

- Python 3 Da die zu verwendende Bibliothek auf Python 3 basiert muss diese Version auf dem auszuführenden Computer vorhanden sein.
  - macOS
  - Linux
  - Win
- Jupyter Notebooks Damit die aufgeführten Scripts direkt im Browser ausgeführt werden können haben sich die Autoren dieser Arbeit darauf verständigt Jupyter Notebook zu nutzen. Dabei bietet diese Applikation den Vorteil, dass Code und Dokumente live geteilt werden können und der entwickelte Code sofort ausgeführt werden kann. Ebenfalls bietet Jupyter Notebook die Möglichkeit unter Anderem Daten zu visualisieren.
  - Installationsanleitung
- "python-chess-library"
  - `pip install python-chess` Im weiteren Verlauf dieses Notebooks werden zusätzliche Module, wie beispielsweise Pandas, genutzt, die ebenfalls auf dem auszuführenden Rechner installiert sein müssen.

1.2 Erstellen eines Schachbretts und ausgeben als ASCII

Zu Beginn muss die Python-Chess-Library eingebunden werden, die für den weiteren Verlauf der Evaluierung und Implementierung benötigt wird.

```
In [3]: import chess
```

Das unten einzusehende Code Snippet zeigt, wie ein neues Schachbrett mit standardmäßigen Positionierungen der Figuren erstellt werden kann. Dies ist umzusetzen durch den Aufruf der `chess.Board()` Funktion und das Speicherns der Rückgabe dieser Funktion in einer `board` Variable. Dieses `board` beinhaltet die Positionierungen aller Figuren und kann mittels der pythoneigenen `print()` Funktion als ASCII Code ausgegeben werden, wie unten einzusehen ist. Zuletzt wird noch die Möglichkeit aufgezeigt, die auf einem spezifizierten Feld befindliche Figur auszulesen. Dieses Feld wird dabei über das Datenfeld `chess.B1` aufgerufen.

```
In [4]: board = chess.Board()

        print ("\nBoard:")
        print (board)

        print ("\nPiece at B1:")
        print (board.piece_at(chess.B1))
```

```
Board:
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R

Piece at B1:
N
```

Dabei ist einzusehen, dass die Figuren durch einen Buchstaben abgekürzt werden. Figuren des ersten Spielers werden als Kleinbuchstaben dargestellt, Figuren des zweiten Spielers als äquivalente Großbuchstaben. Leere Felder werden mit einem Punkt dargestellt. Aus folgender Liste kann die Zurodnung der Buchstaben zu den Schachfiguren entnommen werden:

- p/P: Pawn / Bauer
- r/R: Rook / Turm
- n/N: Knight / Springer
- b/B: Bishop / Läufer
- q/Q: Queen / Dame
- k/K: King / König

Die Zuordnung der Felder findet mittels einer Kombination aus einem Buchstaben und einer Zahl statt. Die Buchstaben A-H geben dabei die horizontale Reihe an, die Zahlen 1-8 die vertikalen Reihen. Jedem Feld wird eine namensäquivalente Variable in der statischen Klasse "chess" zugeordnet, wie beispielhaft in Zeile 16 im oberen Code Snippet erkannt werden kann.

1.3 Alternierende Eingaben von Nutzer & KI

Damit ein Schachspiel zustande kommen kann, ist es zwingend erforderlich, dass Nutzer und KI abwechselnd Züge auswählen und zum Board "pushen" können. Dafür muss zunächst überprüft werden, welcher Spieler an der Reihe ist und dieser muss einen Zug auswählen können, der daraufhin von dem Schachbrett übernommen wird. Auf diesem Schachbrett kann dann der nächste Spieler seinen Zug auswählen.

Bei der *"python-chess"* Bibliothek kann der zu agierende Spieler über die `board.turn` Variable ermittelt werden. Diese steht auf `True` wenn Spieler 1 an der Reihe ist und auf `False`, wenn Spieler

2 den nächsten Zug machen muss. Beim Ausführen einer `push` Operation auf dem `board`, wechselt die Variable automatisch ihren Wert.

Damit das Spiel erkennt, wann dieses vorbei ist, bietet die Bibliothek die `board.is_game_over()` Funktion. Diese gibt den Wert `True` zurück, falls das Spiel auf Grund eines Schachmatts oder anderer spielbeendender Umstände vorbei ist.

Um alle möglichen Züge ausgeben zu können, bietet die verwendete Bibliothek die Funktion `board.legal_moves`. Diese gibt alle gültigen Züge nach *chess960* Standard aus. Dabei werden nur die Züge ausgegeben, die zum einen vom aktuellen Schachbrett aus durchführbar sind und zum anderen nicht zu einer unmittelbaren Niederlage führen. Das bedeutet, dass bei den legalen Zügen keine Züge ausgegeben werden, bei der sich der Spieler beispielsweise selbst ins "Schach" stellt oder ein durch den Gegner verursachtes "Schach" ignoriert. Mittels der `board.uci()` Funktion können diese in die, aus 4 Zeichen bestehende, leserlichere Form gebracht werden. Die ersten beiden Zeichen stellen dabei das startende Feld dar, während die zweiten zwei Zeichen das Feld darstellen, auf das sich die Figur vom Startfeld bewegt.

Diese Funktion kann beim Zug des Nutzers verwendet werden, um diesem eine einfachere Übersicht über seine Möglichkeiten zu geben und den Zug in einer ihm verständlichen Form einzulesen.

Mittels der `list()"` Funktion können die legalen Züge zu einer Liste zusammengefasst werden.

Im folgenden Beispiel sind alle notwendigen Schritte für ein Schachspiel zwischen Nutzer und KI erkennbar. Die KI ermittelt dabei ihren Zug durch eine zufällige Auswahl aus der Liste aller legalen Züge. Dabei wurden die einzelnen Methoden, wie oben beschrieben, implementiert und genutzt.

Zusätzlich müssen einigen Funktionen der Python-Chess-Library importiert werden, die für die Visualisierung eines Schachbretts als SVG benötigt werden. Ebenfalls wird das Modul `random` eingebunden, da es für die Berechnung zufälliger Züge benötigt wird.

```
In [5]: import chess.svg
        import random
        board = chess.Board()
```

`get_random_move` wählt zufällig einen Zug aus den erlaubten Zügen aus und gibt diese als `CHESS.Move` zurück. Hingegen wird die Funktion `get_legal_moves_uci` genutzt um die erlaubten Züge berechnen zu lassen.

```
In [6]: def get_random_move():
            return random.choice(list(board.legal_moves))

        def get_legal_moves_uci():
            return list(map(board.uci, board.legal_moves))
```

Die Funktion `get_user_move` wird verwendet um den Nutzer die möglichen, bzw. erlaubten, Züge auszugeben. Dabei wird die zuvor eingeführte Funktion zur Berechnung von legalen Zügen verwendet.

```
In [7]: def get_user_move():
            print("Possible Moves: ")
            print(get_legal_moves_uci())
```

```
            print("Enter your move:")
            move = input()

            return chess.Move.from_uci(move)
```

Der folgende Codeausschnitt wird genutzt um einen Nutzer gegen den Computer spielen zu lassen, der jeweils einen zufälligen Zug verwendet. Hierbei wird für das Jupyter Notebook ein Counter eingeführt, damit das Spiel bereits nach zwei Zügen pro Spieler beendet ist. Alternativ dazu wird das Spiel beendet, sobald ein Sieger oder ein Patt feststeht.

Ebenfalls wird als Unterstützung für den Spieler pro Zug das Schachbrett und die legalen Züge ausgegeben.

```
In [8]: counter = 0
        while (not board.is_game_over() and counter < 4):
            print("---------------")
            print(board)
            print("---------------")
            print()

            if board.turn:
                board.push(get_user_move())
                print("Your Move: ")
            else:
                board.push(get_random_move())
                print("AIs Move:")

            counter+=1

        print(board)
        print("[...]")
```

```
---------------
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
---------------

Possible Moves:
['g1h3', 'g1f3', 'b1c3', 'b1a3', 'h2h3', 'g2g3', 'f2f3', 'e2e3', 'd2d3', 'c2c3', 'b2b3', 'a2a3',
Enter your move:
c2c3
Your Move:
---------------
```

```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . P . . . . .
P P . P P P P P
R N B Q K B N R
---------------

AIs Move:
---------------
r . b q k b n r
p p p p p p p p
. . n . . . . .
. . . . . . . .
. . . . . . . .
. . P . . . . .
P P . P P P P P
R N B Q K B N R
---------------

Possible Moves:
['g1h3', 'g1f3', 'd1a4', 'd1b3', 'd1c2', 'b1a3', 'c3c4', 'h2h3', 'g2g3', 'f2f3', 'e2e3', 'd2d3',
Enter your move:
d1a4
Your Move:
---------------
r . b q k b n r
p p p p p p p p
. . n . . . . .
. . . . . . . .
Q . . . . . . .
. . P . . . . .
P P . P P P P P
R N B . K B N R
---------------

AIs Move:
r . b q k b n r
. p p p p p p p
p . n . . . . .
. . . . . . . .
Q . . . . . . .
. . P . . . . .
P P . P P P P P
R N B . K B N R
[...]
```

1.4 Ausgeben des Boards als SVG

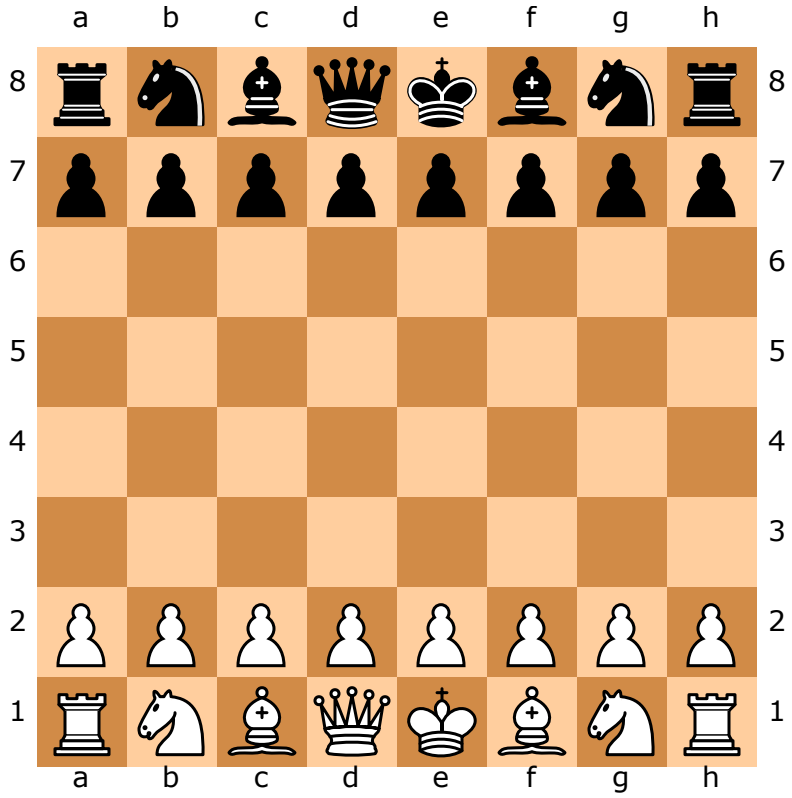
Um eine bessere Darstellung und Erklärung der Implementation in der theoretischen Ausarbeitung zu ermöglichen, wäre eine visuelle Darstellung des Schachbretts wünschenswert. Die "python-chess-library" ermöglicht dies durch ein Konvertieren des Boards zu einer SVG-Datei. Diese kann dann mittels der python-eigenen `SVG()` Funktion im Python-Notebook angezeigt werden.

Um dies zu ermöglichen, muss zunächst die SVG library aus *"IPython"* importiert werden. Anschließend kann das Board über die `chess.svg.board()` Funktion konvertiert und anschließend als SVG ausgegeben werden.

```
In [9]: from IPython.display import SVG

        board = chess.Board()
        SVG(chess.svg.board(board=board))
```

Out[9]:

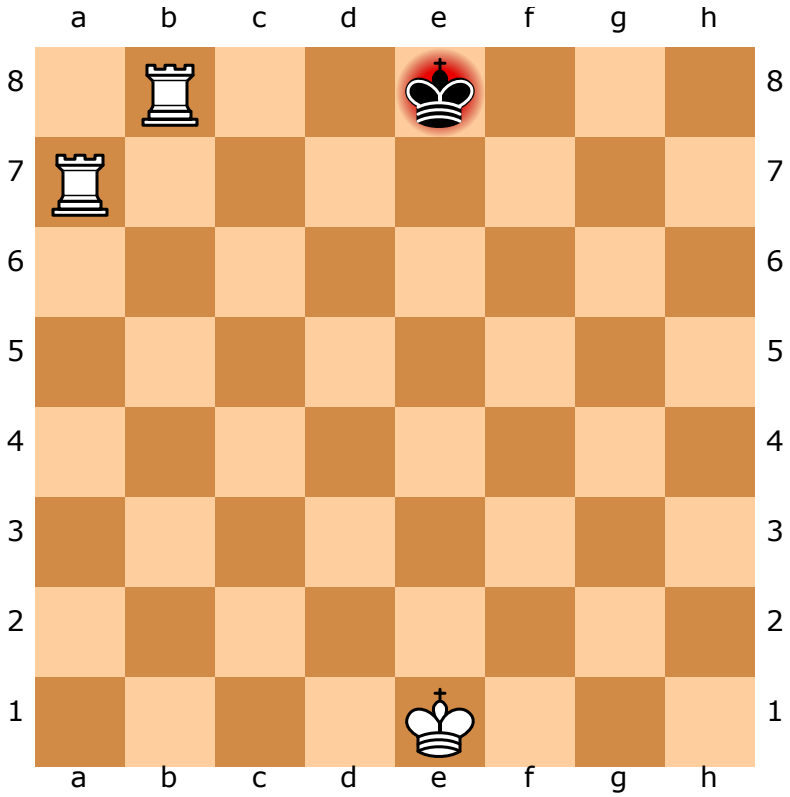


Die verwendete Bibliothek bringt den Vorteil mit sich, dass einige wichtige Eigenschaften während eines Spiels visualisiert werden können. Ein Beispiel für diese Visualisierung ist das Schachmatt, bei dem der betroffende König rot hervorgehoben wird. Ein weiteres Beispiel zum Thema Visualisierung wird im folgenden Kapitel gegeben.

Um diesen Fall simulieren zu können, wird in diesem Beispiel eine bestimmte Situation während eines Schachspiels importiert, in der ein Schachmatt vorliegt. Das Schachmatt auf der Position `E8` wird in diesem Codeausschnitt manuell, durch den Übergabeparamter `chess.E8`, hervorgehoben.

```
In [10]: board = chess.Board("1R2k3/R7/8/8/8/8/8/4K3 b KQkq - 0 1")
         SVG(chess.svg.board(board, check=chess.E8))
```

Out[10]:



1.5 Berechnen & Ausgeben legaler Züge eines speziellen Felds

Für eine bessere, visuelle Darstellung in der theoretischen Ausarbeitung wurde die Möglichkeit geprüft spezielle Felder auf dem ausgegebenen SVG markieren zu können. Solche Felder können beispielsweise die errechneten erlaubten Züge einer speziellen Figur / eines speziellen Felds darstellen, um anzuzeigen, welche Möglichkeiten eine Figur in der aktuellen Situation besitzt.

Dies kann realisiert werden, indem auf die legalen Züge zurückgegriffen wird und diese gefiltert werden. Bei dem verwendeten Filter müssen die Ausgangspositionen, die mit `move.from_square` ausgelesen werden können, mit dem eingegebenen Feld übereinstimmen.

```
In [11]: board = chess.Board()

         print("Please enter field:")
```

```
        field = input()

        moves_from_spec_field = list(filter(lambda move: move.from_square is chess.SQUARE_NAMES
```

```
Please enter field:
a2
```

Diese herausgefilterten Züge werden dann zu deren Zielfelder zugeordnet. Diese können mit der Funktion `move.to_square` ausgelsen werden.

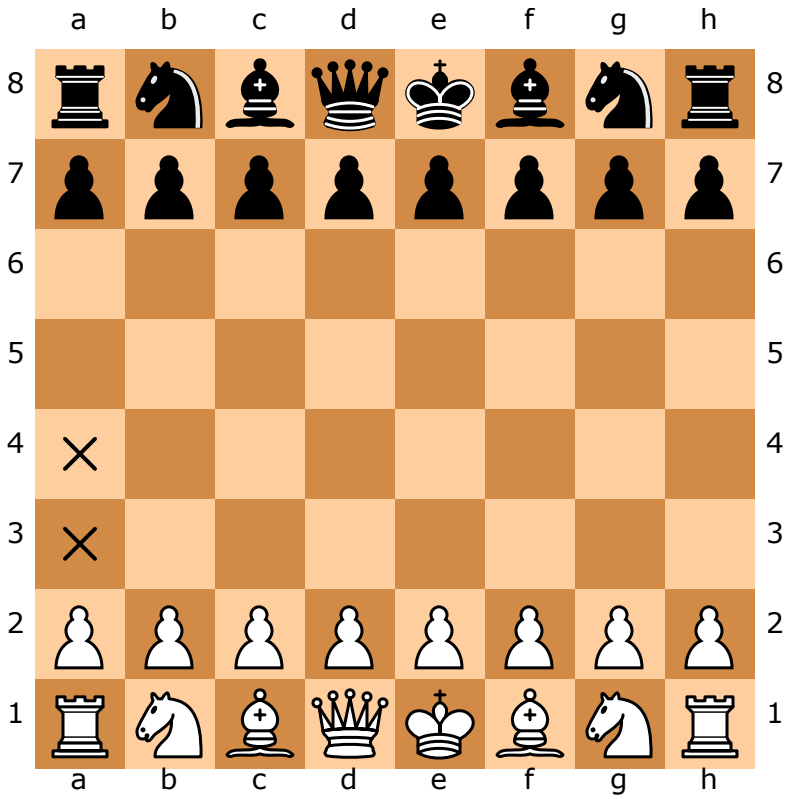
```
In [12]: square_nums = list(map(lambda move: move.to_square, moves_from_spec_field))
```

Nun kann ein SquareSet erstellt und alle gefilterten Felder diesem hinzugefügt werden. Das SquareSet kann dann beim Erstellen des Board angegeben werden. Dies veranlasst das Zeichnen von Kreuzen auf den berechneten Feldern, die von dem angegebenen Feld aus erreicht werden können.

```
In [13]: squares = chess.SquareSet()
         for square_num in square_nums : squares.add(square_num)

         SVG(chess.svg.board(board=board, squares=squares))
```

Out[13]:



1.6 Speicherung der einzelnen Spielzüge in history.csv

Um die KI Entscheidungen um ihre Spielzüge auch von vorherigen Spielen und desse Ausgängen abhängig zu machen, muss eine Historie angelegt werden, die die Spielbretter und den Ausgang des Spiels im Nachgang beinhaltet. Anhand dieser Historie kann die KI in zukünfigen Spielen dann den Wert des Spielbrettes abschätzen und das Spielbrett evaluieren, das die höchste Chance auf einen Sieg bietet.

Zur Erweiterung der Liste wird zuerst das Modul Pandas genutzt, das es ermöglicht Informationen als Datenbank in einer Variable zu speichern. Ebenso wird angegeben unter welchem Pfad sich die Zughistorie befinden soll.

```
In [14]: import pandas as pd
         HISTORY_FILE_LOC = "res/history.csv"
```

Dazu kann testweise ein Spiel durchgespielt werden, wobei jeder Zug zufällig aus der Liste der legalen Züge gewählt wird.

```
In [15]: def get_random_move(board):
             return random.choice(list(board.legal_moves))
```

Nach jedem Zug wird das Spielbrett in der "FEN" Darstellung einer Liste hinzugefügt. Die "FEN" Darstellung ist eine gekürzte Form der Darstellung des Spielbretts, wobei das Spielbrett dennoch eindeutig identifizierbar bleibt. Dabei wird neben dem aktuellen Spielbrett auch die Farbe der nächste zu spielenden Figur, die Anzahl der Züge beider Seiten und weitere Informationen angegeben. Um diese aus der Darstellung zu kürzen, wird die auf die "FEN" Darstellung des Boards die `split` Funktion angewandt und nur das erste Element aus der daraus entstehenden Liste gespeichert, da die einzelnen Merkmale in dieser Darstellung per Leerzeichen getrennt werden.

```
In [16]: def play_random_chess_game(board):
             turn_list = list()
             while not board.is_game_over():
                 turn_list.append(board.fen().split(" ")[0])
                 board.push(get_random_move(board))
             return turn_list
```

Nach Ende des Spiels wird über der zu spielenden Farbe ermittelt, wer der Sieger des Spiels ist und dementsprechend der Wert "+1" oder "-1" zurückgegeben. Mit diesem wird dann eine Key-Value Liste erstellt, die zu jedem Spielbrett des vergangenen Spiels den Wert zuweist, der Aussage über den Sieger gibt.

Diese Liste wird dann zusammen gefügt mit der bisher vorhandenen Historie. Falls bereits ein Eintrag für ein gleichartiges Schachbrett existiert, wird der Siegwert zu dem aktuellen Wert aus der Historie dazu addiert, andernfalls wird ein neuer Eintrag mit dem Siegwert angelegt.

```
In [17]: def store_game(turn_list, victory_status):
             new_turn_dict = dict.fromkeys(turn_list, victory_status)
             # get existing board history
             history = pd.read_csv(HISTORY_FILE_LOC)
             history_dict = dict(zip(list(history.board), list(history.value)))
```

```
            # merge existing history with new boards and sum the victory states
            merged_history_dict = { k: new_turn_dict.get(k, 0) + history_dict.get(k, 0) for
            merged_history = pd.DataFrame(list(merged_history_dict.items()), columns=['boar
            # overwrite history csv with new, merged history
            merged_history.to_csv(HISTORY_FILE_LOC)
```

Der Siegwert nach dem Schachbrett gibt dann, je nach Höhe, Auskunft darüber, wie wahrscheinlich es ist mit einem solchen Schachbrett zu gewinnen. Umso höher der Wert in den positiven Bereich fällt, desto wahrscheinlicher ist ein Sieg für Spieler 1. Umso höher der Wert in den negative Bereich fällt, desto besser sind die Aussichten für Spieler 2.

Diese Erkenntnis kann dann von der KI genutzt werde, um beim "iterative Deepening" Prozess alle möglichen Schachbretter zu evaluieren und sich so für den besten Zug zu entscheiden.

Der folgende Code koordiniert die Abläufe der Funktionen um die Datei `history.csv` zu erweitern.

```
In [ ]: for i in range(0, 10):
            i += 1
            board = chess.Board()
            turn_list = play_random_chess_game(board)

            victory_status = 1 if board.turn else -1
            store_game(turn_list, victory_status)
        print("Games finished")
```

Ein Ausschnitt aus der history.csv ist im folgenden Snippet zu sehen.

```
In [ ]: df = pd.read_csv(HISTORY_FILE_LOC)
        df.head()
```

Damit eingeschätzt werden kann, wie aussichtsreich ein bestimmter Boardzustand ist, kann ein gespeicherter Spielverlauf zur Hilfe genommen werden. Mit Hilfe der Funktion `compare_board_history` kann ein Erwartungswert berechnet werden. Umso höher der zurückgegebene Wert der Funktion, desto wahrscheinlicher ist ein Sieg für Weiß. Hingegen ist ein Sieg für Schwarz umso wahrscheinlicher, sobald der Wert deutlicher in den negative Bereich fällt.

Der Vorgang zur Berechnung dieses Wertes beginnt mit dem Einlesen der `history.csv` in ein Pandas Dataframe. Daraufhin wird durch die Reihe des Dataframes iteriert, die die Schachbretter in FEN-Schreibweise enthält. Sobald die aktuelle Spielsituation in der Historie gefunden wurde, wird deren Wert ausgelesen und zurückgegeben. Falls die aktuelle Spielsituation nicht vorhanden ist, wird der Wert `0` zurückgegeben.

```
In [ ]: def compare_board_history(board):
                df = pd.read_csv(HISTORY_FILE_LOC)
                row = df.loc[df['board'] == board.fen().split(" ")[0]]
                value = row['value'].item() if len(row['value']) == 1 else 0
                return value
```

1.7 Überprüfung auf ein Schach oder Spielende

Damit der Spielverlauf korrekt nach den Regeln des Spiels verlaufen kann, muss während jedes Zuges darauf geprüft werden, ob ein Schach, Schachmatt oder Patt vorliegt. Schach ist hierbei

eine Stellung während des Spiels, bei dem der König in Bedrängnis geraten ist. Ein Schach kann im weiteren Verlauf zu einem Schachmatt führen. Hierbei liegt der Unterschied darin, dass der Spieler, der sich im Schachmatt befindet mit keinem regelkonformen Zug sich aus dem Schachmatt befreien kann und somit das Spiel verloren hat. Steht der König Schach kann er mit einem gezielten Zug sich aus dieser Lage befreien. Ein Patt ist eine Endposition beim Schach, die die Eigenschaft hat, dass keiner der beiden Spieler ein Schachmatt erreichen kann.

Da ein Schachmatt oder Patt das Schachspiel beendet ist diese Prüfung ein essenzieller Aspekt der zu entwickelnden KI. Um diese Prüfung durchzuführen bietet die verwendete Bibliothek bereits einige Funktionen. Ebenso ist es wichtig zu überprüfen, ob ein Schach vorliegt, da der Spieler zuerst dies lösen muss, bevor er weiterspielen kann. Um diese Prüfung durchzuführen steht jeweils für die Überprüfung eines Schachs oder Spielendes eine Funktion zur Verfügung, die das aktuelle Spiel auf diesen Aspekt überprüft. Um herauszufinden, ob es sich um ein Schach handelt, kann die folgende Funktion verwendet werden. * `is_check()` * Überprüft die aktuellen Begebenheiten auf ein mögliches Schach

Hingegen gibt es die Funktion `is_game_over`, die die gespielten Züge auf jegliche Arten einer benötigten Beendigung des Spiels überprüft. Hierbei beinhaltet `is_game_over` einige mögliche Überprüfungen auf Schachmatt oder Patt. * `is_game_over()` * Stellt sicher, ob das Spiel auf Grund eines Schachmatts oder anderer spielbeendender Umstände vorbei ist * Hierbei wird überprüft, ob ein Schachmatt (`is_checkmate()`), ein Patt (`is_stalemate()`), eine Tote Stellung (`is_insufficient_material()`), die 75-Züge-Regel (`is_seventyfive_moves()`), eine Figur fünfmal auf der gleichen Position sich befindet (`fivefold_repetition`), oder eine spezielle Endbedingung vorliegt.

Um herauszufinden, ob ein Spieler tatsächlich gewonnen hat kann eine Teilfunktion der zuvor genannten genutzt werden. Diese Funktion überprüft lediglich, ob ein Schachmatt vorliegt. * `is_checkmate()` * Stellt fest, ob es sich nach den bereits gespielten Zügen ein Schachmatt vorliegt

```
In [ ]: # import situation where checkmate is True
        board = chess.Board("r1bqkb1r/pppp1Qpp/2n2n2/4p3/2B1P3/8/PPPP1PPP/RNB1K1NR b KQkq - 0 4"

        print("Check: ", board.is_check())
        print("Checkmate: ", board.is_checkmate())
        print("Game is over: ", board.is_game_over())
```

Im folgenden wird ein Beispiel einer Spielsituation gezeigt, in der ein Schachmatt vorliegt und die auf Schach, Schachmatt und Spielende überprüft wurde.

```
In [ ]: SVG(chess.svg.board(board=board))
```

1.8 Überprüfung, ob "en passant" und "Rochaden" unterstützt werden

Im Schach gibt es einige spezielle Züge, die es ermöglichen eine Figur eine Aktion durchzuführen zu lassen, die normalerweise laut der grundlegenden Definition dieser nicht möglich ist.

Die Schachfigur eines Bauers darf normalerweise nur dann eine andere Figur schlagen, wenn sich diese direkt in dem diagonal vor dem Bauern angrenzenden Feld befindet. Durch die sogenannte Regel "en passant", im deutschen "im Vorbeigehen", wird diese Regel erweitert. Eingesetzt werden kann "en passant", wenn auf einen Bauern die Sonderregel des Doppelschritts aus der Grundstellung angewendet wird. Steht in diesem Fall der mit Doppelschritt herausgerückte Bauern neben einem des Gegners, dann kann dieser den neu herausgerückten Bauern durch "en

passant" schlagen. Hierbei springt der angreifende Bauer des Gegners direkt hinter den herausgerückten und schlägt ihn somit.

Damit die Schach-KI alle möglichen Züge des Gegners bedenken und ebenso alle Züge ausführen soll, muss überprüft werden, ob die verwendete Library die beiden Sonderzüge unterstützt, oder ob diese Unterstützung manuell entwickelt werden muss.

```
In [ ]: # import situation where en passant is possible
        board = chess.Board("rnbqkbnr/1pp1pppp/p7/3pP3/8/8/PPPP1PPP/RNBQKBNR w KQkq d6 0 3")

        # the function board.has_legal_en_passant() could check if a en passant is possible, but
        print("Library is supporting en passant: ",chess.Move.from_uci("e5d6") in board.legal_mo
```

Die Chess-Core Bibliothek unterstützt somit den Sonderzug en passent.

Im Folgenden wird ein Schachbrett angezeigt, bei dem es dem weißen Bauer (`e5`) möglich ist im Vorübergehen den schwarzen Bauer (`d5`) zu schlagen.

```
In [ ]: squares = chess.SquareSet([chess.D6])
        chess.svg.board(board=board, squares=squares)
```

Neben dem en passant gibt es einen weiteren bekannten Sonderzug, die sogenannte Rochade. Bei der Rochade lassen sich die Positionen eines Turms und des Königs tauschen, wobei für diesen Vorgang nur ein Zug benötigt wird. Dabei ist zu beachten, dass die Voraussetzung für diesen Zug ist, dass sowohl der zu verwendende Turm, als auch der König im Verlauf des Spiels nicht genutzt wurden. Ebenfalls dürfen die Felder zwischen König und Turm nicht belegt sein und keines der Felder, durch die der König ziehen muss, darf durch eine gegnerische Figur bedroht sein, sowie der König vor und nach der Rochade nicht im Schach stehen.

Für jeden Spieler gibt es zwei verschiedene Möglichkeiten der Rochade, einerseits die kurze, als auch die lange Rochade. Ein Beispiel für eine lange Rochade der weißen Figuren ist, dass der Turm (`a1`) und der König (`e1`) ihre Positionen tauschen und somit der Turm sich auf dem Feld `d1` und der König auf `c1` befindet.

```
In [ ]: # import situation where castling is possible
        board = chess.Board("8/8/8/8/8/8/8/R3K3 w KQkq - 0 1")
        # shortcut for the castling-move with the queenside rook
        castling = "O-O-O"

        # check if castling is in the legal moves and print the result
        if castling in str(board.legal_moves):
            print("Library is supporting castling: True")
            board.push_san(castling)
        else:
            print("Library is supporting castling: False")

        SVG(chess.svg.board(board=board))
```

Eine durchgeführte kurze weiße Rochade, bei der `a1` und `e1` die Positionen getauscht haben, sodass der König sich nun auf `c1` und der Turm auf `d1` befindet. Die Abkürzung für eine kurze Rochade ist *O-O*, das für den Turm auf der Seite des Königs steht und *O-O-O* für eine lange Rochade, wobei die Abkürzung für den Turm auf der Seite der Dame steht.

1.9 Einbinden eines Opening-Books in die Chess AI

Polyglot ist ein Open-Source Format, in dem sogenannte Opening-Books erstellt werden können. Opening-Books sind Ansammlungen von Spielzügen, die im Schach zur Eröffnung genutzt werden können. Hierbei wird zum Einstieg des Spiels nicht auf eine Künstliche Intelligenz zurückgegriffen, sondern auf ein Verzeichnis von bereits bestehenden Zügen, die sich innerhalb dieses Opening-Books befinden. Der Vorteil eines Opening-Books liegt darin, dass bereits qualitativ hochwertige Strategien verfügbar sind und somit ein anspruchsvolles Spiel ermöglichen.

Sobald das Opening-Book, im Verlauf des Spiels, keinen passenden Zug als Antwort bereitstellen kann, übernimmt die Künstliche Intelligenz.

Für das Einbinden eines Opening-Books werden einige Funktionen aus der Core Chess Library genutzt und müssen somit eingebunden werden.

```
In [ ]: import chess.polyglot
```

Als nächstes wird ein neues Spiel/Board erzeugt, indem die Figuren standardmäßig angeordnet werden. Ebenfalls wird ein Opening-Book in eine Variable geladen, damit aus dieser der bestmöglichen Zug ausgewählt werden kann. Dies geschieht, indem das Opening-Book das aktuelle Board übergeben bekommt und anhand der Positionen der Figuren einen passenden Zug auswählt.

Der ausgewählte Zug wird als nächstes auf dem Schachbrett ausgeführt und somit wird in diesem Beispiel der weiße Bauer von `e2` nach `e4` gezogen.

```
In [ ]: board = chess.Board()
        book = chess.polyglot.open_reader("res/polyglot/Performance.bin")

        def get_move(board):
            # find the move with the highest weight for the current board
            try:
                main_entry = book.find(board)
                all_entries = book.find_all(board)

                move = main_entry.move()
                print("Selected move with the highest weight: ", move)
                print("All available moves for this situation: ", ", ".join([str(entry.move()) f

                return move
            except IndexError:
                print("The opening book cannot find an appropriate move!")

        for i in range (1,3):
            next_move = get_move(board)
            board.push(next_move)
            print("Selected move: {}\n".format(next_move))
        book.close()

        SVG(chess.svg.board(board=board))
```