

Report – Tic Tac Toe

Von:

Alex Stephan – 22300959

Elias Eder – 22201619

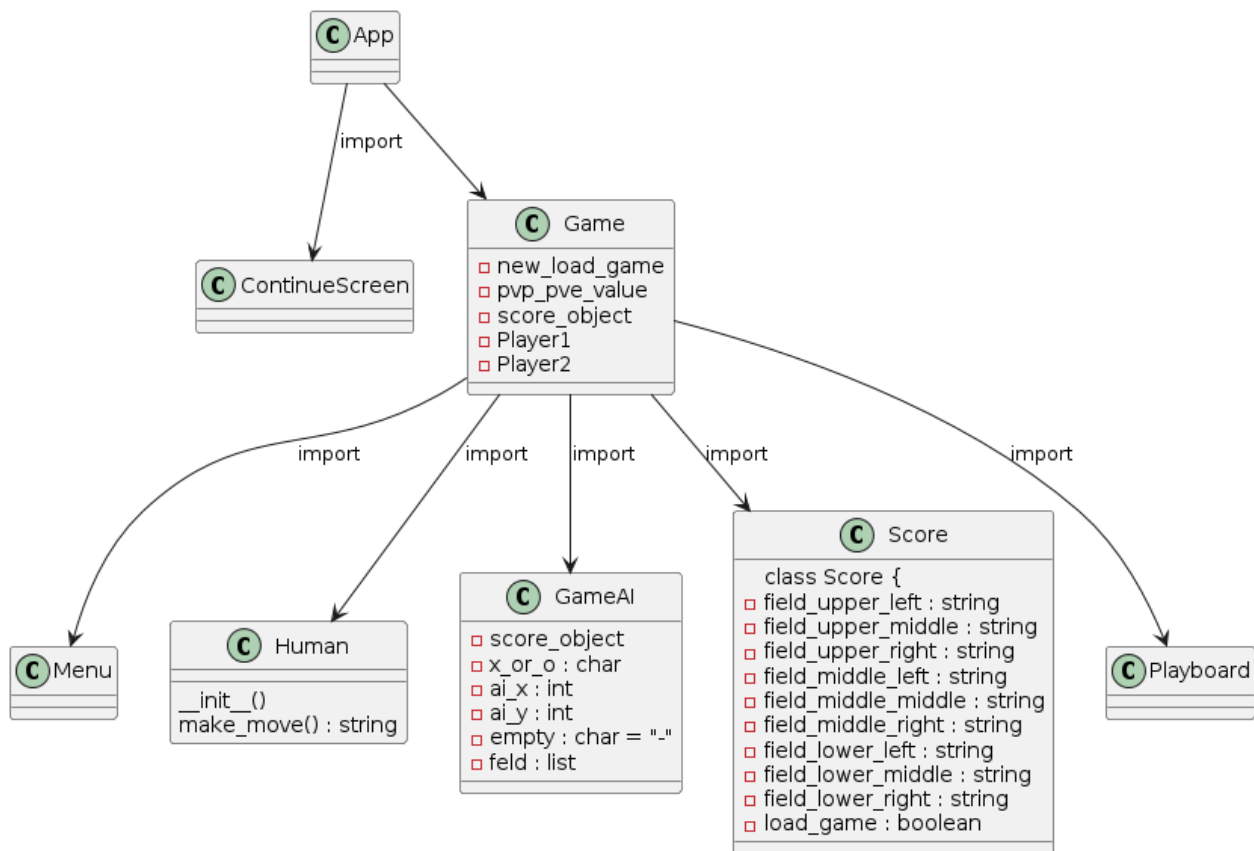
Jonas Gaßner – 22307744

1. Einleitung

Aufgabe war es ein Tic-Tac-ToeSpiel in Python zu implementieren mit einer integrierten KI als Gegenspieler. Zudem sollte eine Speicherfunktion für Spielstände hinzugefügt werden. Die Implementation sollte nach Model View Presenter bzw. Model View Controller erfolgen. Wir haben uns für ersteres entschieden. Zudem sollten noch Unit-Tests verfasst werden.

Im Folgenden erörtern wir unsere Vorgehensweise bei der Implementierung des Projektes.

2. Architektur



Zusammengesetzt ist das Projekt aus den Klassen Human, KI (welche die Spieler darstellen), Score (diese drei Klassen rechnet man zum Modell), dann gibt es die Presenter Klasse Game,

in welcher die Klassen GameAI, Human, Menu, Score und Playboard importiert werden, um dann das Spiel darauf aufzubauen. Zuletzt gibt es noch die Continue_Screen Klasse, welche zusammen mit Menu als View die graphische Oberfläche des MVP Entwurfsmusters bildet. Zur Ausführung des Ganzen, verwenden wir eine externe Datei (app.py), diese greift auf die Game und Continue_Screen Klassen zu. Dadurch kann das gesamte Programm vollständig ausgeführt werden.

Für die Speicherfunktion ist die Klasse Score von Bedeutung, welche mit einer JSON-Datei interagiert, um Spielstände aus- und ein-zu lesen.

Die View Klassen Menu und Continue_Screen sind die einzigen, welche Konsolenausgaben tätigen und Userinput entgegennehmen, um es dann an die Klasse Game weiterzuleiten.

3. Serialisierung und Deserialisierung

Serialisation wird im Programm über JSON realisiert. Es wird in der Klasse Game die Methode save_game (self, score_object) automatisch aufgerufen, wenn das Spiel zu Ende ist, bzw. wenn man das Spiel mit der Eingabe von „q“ quittiert hat. In dieser Methode ruft der Parameterscore_object, welcher ein Objekt der Klasse Score ist die Methode save_game von Score auf. Zunächst wird darin durch das time Modul ein timestamp ermittelt, um als Identifikator für Spiele zu dienen. Weiter werden dem Objekt seine Eigenschaften entnommen. Danach wird das Ganze, nachdem man ein id Dictionary erstellt hat zu diesem hinzugefügt. Wenn es keine JSON-Datei gibt, oder nur eine, die leer ist, wird ein leeres Dictionary erstellt, wenn es eine Datei mit Inhalt gibt, wird daraus der Datensatz geladen, welcher dann auch ein Dictionary ist. Daraufhin wird dem Dictionary, welches natürlich verschachtelt ist, ein neues Dictionary als Inhalt zugewiesen, der Key dazu ist der Timestamp und der Wert ist das Dictionary, welches wir aus dem Score Objekt entnommen haben. Jetzt wird nur noch die Datei entleert und dann das gesamte Dictionary, welches alle bisher nötigen Spielinformationen, aller Spiele enthält in die Datei geschrieben.

Deserialisierung erfolgt, indem man zunächst die zuvor im Menu abgefragte Variablen new_load_game auswertet. Falls sich im Menü für ein neues Spiel entschieden wurde, wird einfach ein neues Objekt der Klasse Score erstellt und zurückgegeben. Falls aber ein Spiel geladen werden soll, wird zunächst ein Menu Objekt erzeugt, welches dann aufgefordert wird, abzufragen welches Spiel man laden soll. Nachdem man entschieden hat, welches Spiel zu laden ist, wird ein neues Score Objekt erstellt. Bei falschen Eingaben wird das Score Objekt aufgefordert zu prüfen, ob es dieses Spiel gibt. Falls nicht, wird die Abfrage wiederholt. Falls das Spiel existiert, wird das Score Objekt aktualisiert, indem es angewiesen

wird, sich die Daten aus der JSON-Datei zu holen und dann seine Attribute (Feldattribute), mit den Werten zu belegen.

4. KI-Funktion

Die Umsetzung der KI-Funktion erfolgt durch Implementierung des Minimax-Algorithmus. Da in Game mit zwei Objekten Player1 und Player2 gearbeitet wird, welche von der Klasse Human oder GameAI stammen, kann, falls Human und GameAI eine gleichnamige Methode haben, diese einfach in Game aufgerufen werden, um für den jeweiligen Spielmodi (PvP, PvKI) die richtige Aktion auszuführen. Diese Methode heißt in beiden Klassen „make_move()“.

In der GameAI Klasse unter der Methode „make_move()“, wird zunächst ein leeres Feld angelegt. Dann wird das Score Objekt, welches als Attribut bei der Erstellung des GameAI mitgeliefert wird, in ein Array ausgelesen. Jetzt wird in diesem Array jedes „X“ mit einer 2, jedes „O“ mit einer 0 und jedes andersartige Zeichen mit „-“ ausgetauscht. Diesen Zeichenaustausch brauchen wir, da sich die anderen Methoden der Klasse GameAI daran orientieren. Dann wird dieses Array in ein zweidimensionales Array transformiert, und als Attribut für das GameAI Objekt zugewiesen. Mit diesem zweidimensionalen Array, wird ab sofort weitergearbeitet.

Als nächstes muss man natürlich wissen, ob man selbst Spieler1 oder Spieler2 ist, oder anders gesagt, ob man „X“ oder „O“ ist. Bei der Erstellung des Objekts wurde ein Parameter übergeben, welcher dies festhält. Man prüft nun also dieses Attribut. Wenn es „X“ ist, ruft das Objekt die Methode „get_max()“ auf. Ansonsten wird „get_min()“ aufgerufen. Durch diese zwei Methoden, wird schlussendlich berechnet, welcher Zug zu machen ist. Nach der Berechnung wird noch geprüft, welche Feldnummer (1 bis 9) den berechneten Koordinaten entspricht. Dies muss man machen, da dies das Score Objekt als Eingabe benötigt, um dieses Feld dementsprechend zu bearbeiten.

Da jetzt bekannt ist, wie der Ablauf grob von Statten geht, wird jetzt die Berechnung des Zuges aufgezeigt.

Zunächst muss die Methode „evaluation()“ besprochen werden, da diese für die weiteren Methoden grundlegend ist. Diese Methode prüft, ob ein Spieler gewonnen hat. Falls „X“ gewonnen hat, gibt sie 2 zurück, falls „O“ gewonnen hat 0. Vorgenommen wird dieser Test, indem man für 0 und 2, welche, wie oben bereits angeführt „X“ und „O“ repräsentieren, prüft, ob eine Reihe, eine Spalte, oder eine Diagonale nur 0 oder nur 2 enthält. Falls das der Fall ist, wird 0, bzw. 2 zurückgegeben. Es gibt aber noch die Möglichkeiten, dass das Spiel in ein Unentschieden geendet ist, und, dass das Spiel noch nicht zu Ende ist. Um zu prüfen, ob das Spiel schon zu Ende ist, durchläuft man einfach

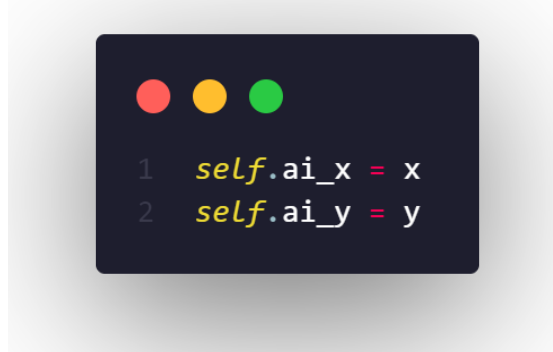
alle Felder und falls eines noch leer ist, gibt man -1 zurück. Da, falls die Funktion an diesem Punkt immer noch keinen Wert zurückgegeben hat, es ein Unentschieden sein muss, kann man schlussendlich noch „return 1“ einfügen. Die 1 repräsentiert also ein Unentschieden.

Es gibt noch 4 Methoden, welche jetzt erläutert werden.

Die 4 Methoden beginnen alle gleich. Sie starten mit der Prüfung, ob das Spiel schon zu Ende ist, also ob die Methode „evaluation()“ nicht -1 zurückgegeben hat. Falls es zu Ende ist, wird einfach der Wert aus „evaluation()“ zurückgegeben. Ansonsten erfolgt die weitere Prüfung.

Ich beginne nun mit der Methode „get_min()“, welche aufgerufen wird, falls die KI „O“ spielen soll. Da, falls „O“ gewinnt, 0 zurückgegeben wird (von „evaluation()“) und die anderen Werte der Spielausgänge höher sind als 0, versucht man also das minimale Ergebnis zu erzielen. In „get_min()“ werden nach dem im vorherigen Absatz geschilderten ersten Schritt, zwei for-Schleifen aufgemacht, um alle Elemente im zwei-Dimensionalen Array zu prüfen. Geprüft wird, ob ein Objekt an einer Stelle des Arrays noch leer ist. Falls dies der Fall ist, wird in dieses Feld 0 geschrieben. Dann wird für diese Methode „max()“ aufgerufen. Da die „max“ Funktionen den Zug von „X“ repräsentieren, sucht die „max()“ Methode also die nächste Option für den Spieler „X“. Da sich die Methoden „min()“ und „max()“ immer gegenseitig aufrufen, wird dadurch der weitere beste Spielverlauf zurückgegeben. Dann wird das alles noch für die anderen Felder geprüft. Die Variable, welche den besten Spielverlauf festhält, wird dabei geupdated, falls sie besser ist als einer der vorherigen (also kleiner). Bis zu diesem Punkt ist auch die „min()“ Funktion identisch. Der Unterschied ist jetzt jedoch, dass zwei Attribute des GameAI Objekts, welche die Koordinaten des nächsten Aufrufs speichern, jetzt gesetzt werden. Dann wird das Feld an dieser Stelle noch geleert. Zuletzt wird der Minimalwert zurückgegeben.

Die „get_max()“ Funktion unterscheidet sich zu „get_min()“ in seiner Absicht den optimalen Zug für „X“ zu finden. Um den Aufbau der Funktionen zu sehen ist hier der Code der „get_max()“ Funktion und der „get_min()“ Funktion. Die Methoden „min()“ bzw. „max()“ unterscheiden sich nur in zwei Zeilen von „get_min()“ und „get_max()“. Diese Zeilen sind jeweils folgende, welche in „min()“ bzw. „max()“ nicht vorhanden sind:



```
1 self.ai_x = x
2 self.ai_y = y
```

```

1  def get_min(self):
2      """Function contains the min-part of the algorithm"""
3      if self.evaluation() != -1:
4          return self.evaluation()
5      min_score = 999
6      for x in range(3):
7          for y in range(3):
8              if self.field[x][y] == self.empty:
9                  self.field[x][y] = 0
10                 value = self.max()
11                 if value < min_score:
12                     min_score = value
13                     self.ai_x = x
14                     self.ai_y = y
15                 self.field[x][y] = self.empty
16      return min_score
17
18  def get_max(self):
19      """Function contains the max-part of the algorithm"""
20      if self.evaluation() != -1:
21          return self.evaluation()
22      max_score = -999
23      for x in range(3):
24          for y in range(3):
25              if self.field[x][y] == self.empty:
26                  self.field[x][y] = 2
27                  value = self.min()
28                  if value > max_score:
29                      max_score = value
30                      self.ai_x = x
31                      self.ai_y = y
32                  self.field[x][y] = self.empty
33      return max_score

```

Man kann hier also im Code erkennen, wie sich die Funktionen gegenseitig aufrufen, um den optimalen Zug zu erarbeiten. Zu sagen ist abschließend noch, dass MinMax Algorithmus den besten Zug wählt, den er machen kann, indem er annimmt, dass der Gegner auch optimal spielt.

5. Tests:

Für das Unit Testing wurde das Framework unittest verwendet, sowie das Packet coverage, um die line coverage auszugeben. Für jede der Klassen beziehungsweise Python Datei wurde im Root Directory des Projekts eine test_class.py angelegt, welche automatisch als Testdatei erkannt wird und ausgeführt wird.

Herausfordernd waren vorallem die Unit Tests für die Klasse Game, da man hierbei mit der patch Funktion und der Mock Klasse Objekte simulieren musste.

Zum Durchlaufen der Tests wurde folgender Command benutzt:

```
coverage run -m unittest discover
```

Wodurch mit „coverage report -m“ folgende line coverage erreicht wurde:

Name	Stmts	Miss	Cover	Missing

model\ai.py	100	0	100%	
model\human.py	12	0	100%	
model\score.py	108	32	70%	28-48, 79-81, 143, 145, 149, 151, 153, 155, 157, 159, 163, 165, 167, 169
presenter\game.py	92	22	76%	38-40, 50, 83-84, 86-87, 94-96, 102-103, 107-115
test_ai.py	53	0	100%	
test_game.py	107	0	100%	
test_human.py	15	0	100%	
test_menu.py	39	0	100%	
test_playboard.py	31	0	100%	
test_score.py	66	1	98%	63
view\menu.py	40	0	100%	
view\playboard.py	17	0	100%	

TOTAL	680	55	92%	

6. Aufgabenverteilung

Die Aufgabenstellung wurde von ins in der Gruppe wie folgt, aufgeteilt:

Elias Eder:

- Implementierung des Spiels Tic-Tac-Toe, ohne Speicherfunktion und der KI Funktion

Jonas Gaßner:

- Implementierung der Speicherfunktion in das Spiel, sowie schreiben der Dokumentation

Alex Stephan:

- Implementierung der Unit-Tests, sowie PIP-8 Refactoring

7. Fazit und Aussicht

Abschließend lässt sich sagen, dass, obwohl das Spiel die vollständig geforderte Funktionalität besitzt, einiges natürlich weiterhin verbesserungswürdig ist. Bei der Speicherfunktionalität, könnte so z.B. noch das Menu angepasst werden, da, falls man ein Spiel laden will, es nicht nötig ist zu wissen, ob es ein PvP oder PvKI Spiel ist. Ferner könnte auch die Laufzeit verbessert werden. Außerdem gibt es bei den Unit-Tests noch viel Optimierungspotenzial.

Für die GameAI Klasse könnte man auch noch eine Bibliothek für die besten Züge, unter bestimmten Spielständen erstellen, damit man z.B. den besten Anfangszug nicht immer neu berechnen muss, und somit Laufzeit einspart. Ferner könnte man außerdem Alpha-Beta-Pruning implementieren, um die KI Funktion zu verbessern.

Da unser Spiel natürlich „nur“ Tic-Tac-Toe ist, ist es nicht wirklich ausschlaggebend Laufzeitorientierung vorzunehmen, da es simpel und klein ist. Zum Ende kann man also sagen, dass das Spiel natürlich noch verfeinert werden kann, es jedoch grundlegend die gewünschten Features beinhaltet.

Link zum Repository:

<https://mygit.th-deg.de/programmierung-2/tic-tac-toe>