TECHNISCHE UNIVERSITÄT DRESDEN

Streichholz - KI

– Implementierung der KI–

Professur für Didaktik der Informatik

Im Folgenden wollen wir unsere Streichholzschachtel-KI selbst umsetzen. Öffne dazu den Webeditor playcode.io. Wenn du dich noch nicht so sicher fühlst, kannst du auch diese Vorgabe benutzen, in der die Struktur bereits vorgegeben ist. Wir werden nur in **ki.js** arbeiten. Du kannst das Projekt direkt bearbeiten. Wenn du speichern möchtest, kannst du dich entweder mit deinem eigenen Google-Account anmelden. Dann kannst du an deinem Projekt auch zuhause noch arbeiten. Alternativ musst du deinen Code in einem Textdokument speichern.

HINWEIS

In *ki.js* ist unsere KI. Diese KI muss natürlich mit Figuren und einem Brett arbeiten. Dazu benutzen wir zwei Bibliotheken. Für das Ermitteln von möglichen Zügen und Prüfen der Regeln benutzen wir **chess.js**. Um das Schachbrett auch darzustellen verwenden wir außerdem die Bibliothek **chessboard.js**. Mach dir keine Sorgen, wir führen die nötigen Funktionen Schritt für Schritt ein. Für jede Programmaufgabe gibt es außerdem Hilfen, mit denen es einfacher wird.

AUFBAU DER KI

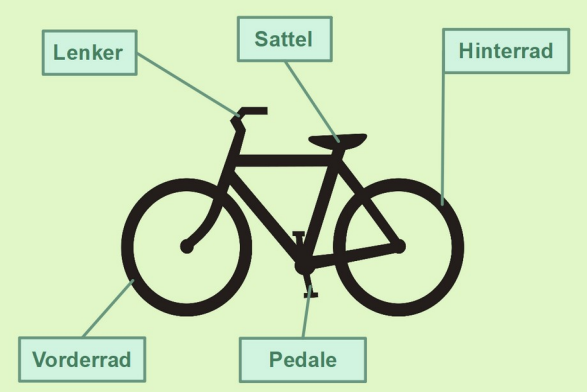
Bevor wir anfangen, schauen wir uns noch einmal den Aufbau der KI an. Wir werden in diesem Projekt **objektorientiert** arbeiten. Falls du dich damit bereits schon auskennst, kannst du die folgende Box überspringen.

OBJEKTORIENTIERTE PROGRAMMIERUNG

Die *objektorientierte Programmierung* (OOP) ist ein wenig anders als die klassische **imperative** Programmierung die man meist als Erstes lernt. Tatsächlich ist sie aber wichtigste **Programmierparadigma** (Leitbild) in der Anwendungsentwicklung.

Die Grundidee lautet, dass wir versuchen die Realwelt mittels Objekten abzubilden. Vielleicht sagen dir **Objekte** und **Klassen** schon etwas aus dem Informatikunterricht. Alles was du in deiner Umgebung siehst, sind Objekte. Ein Fahrrad, ein Computer oder auch ein Blumentopf. Diese Objekte übertragen wir jetzt in unsere Programmiersprache.

Bleiben wir mal beim Fahrrad und schauen uns das genauer an.



So ein Fahrrad hat verschiedene Eigenschaften. Zum Beispiel besitzt es Räder, einen Lenker oder hat eine Farbe. Diese Eigenschaften nennen wir **Attribute**. Das Fahrrad kann aber auch Dinge tun. Das Fahrrad kann fahren, lenken oder umlackiert werden. Diese Fähigkeiten nennen wir **Methoden**.

Jedes Objekt hat also sogenannte Attribute und Methoden. Diese benutzen wir in der OOP. Wir erstellen verschiedenste Objekte, mit verschiedenen Attributen und Methoden und lassen diese miteinander interagieren.

Schneiden wir noch schnell den Begriff der **Klasse** an. Klassen kann man sich als Baupläne für Objekte vorstellen. Es gibt beispielsweise viele verschiedene Fahrräder. Manche haben große Reifen, andere kleine. Manche sind schwarz, andere sind rot. Die Klasse der der Fahrräder macht also die Vorgaben: „Ein Fahrrad hat das Attribut Vorderrad. Ein Fahrrad hat eine Farbe. Mit einem Fahrrad muss man fahren können.“ Diese Anforderungen sind die *Klasse*, ein konkretes Fahrrad ist ein *Objekt* dieser Klasse.

Klassen (und Objekte) können mithilfe eines **UML – Diagramm** abgebildet werden. Dabei werden untereinander Attribute und Methoden dargestellt. Wichtig ist die im Beispiel fettgedruckte Methode. Diese ist der sogenannte **Constructor**. Mit dieser Methode wird ein Objekt der Klasse **erstellt**.

FAHRRAD
- Rahmenfarbe - Vorderrad - Hinterrad
+ **Fahrrad()** + fahre() + lackieren(Farbe)

Uml – Diagramm der Klasse Fahrrad

Jetzt reicht es aber mit der trockenen Theorie. Wie ist denn nun unsere KI aufgebaut? Sie besteht aus 2 Klassen. Untenstehend siehst du die UML – Diagramme der Klassen **Ki** und **Schachtel**. Du kannst hier bereits alle Attribute und Methoden sehen, die wir um folgenden benutzen oder selbst definieren werden.

Ki
- alleSchachteln:Schachtel[ ]
+ Ki() + sucheSchachtel() + lerne()

Schachtel
- stellung: String - perlen: String[ ]
+ Schachtel(stellung: String) + macheZug() + entfernePerle(move: String)

AUFGABE 3

Im OPAL – Kurs findest du unter **Aufgabe 3** eine kleine Übung. Ordne darin die Elemente von Ki und Schachtel ihrer Echtweltentsprechung zu.

Eine gute Variablen und Methodenbenennung ist entscheidend für größere Programmierprojekte, damit Außenstehende sich leicht einarbeiten können.

AUFGABE 4

Erschließe dir die Funktion der Attribute und Methoden der beiden Klassen. Worin unterscheiden sie sich zu ihrem Gegenstück in der echten Welt?

DIE KI MACHT EINEN ZUG

Jetzt wollen wir aber anfangen. Dazu sorgen wir erst einmal dafür, dass die KI einen Zug machen kann. Wir müssen dazu als aller Erstes dafür sorgen, dass die Methode *sucheSchachteln()* entweder:

1. eine bestehende Schachtel aus der Liste `alleSchachteln` heraussucht

oder

2. eine neue Schachtel erstellt, wenn die KI die Stellung noch nie zuvor gesehen hat.

Wie speichern wir jetzt verschiedene Stellungen? Dazu nutzen wir den sogenannten **FEN**-String. Dieser ist genau für Notation von Schachstellungen entstanden. Wenn du dich mehr zu dem Aufbau von diesen FEN – Strings interessierst, schau mal auf den Spickzettel. Für die Bearbeitung der Station ist dies aber nicht nötig

```
let stellung = game.fen()
let board
```

Den FEN-String erhalten wir, indem wir das Spiel mit der Funktion `fen()` nach dem String fragen.

ACHTUNG

Alle für diese Station nötigen Funktionen befinden sich auf dem **Spickzettel** im OPAL – Kurs. Wenn du also ohne Hinweise programmieren möchtest, findest du dort alles was du brauchst.
Ansonsten gibt es für jede Aufgabe **Hinweise** im OPAL – Kurs. Wenn du nicht weiter weißt, sieh dir den nächsten Hinweis an.

Wir programmieren nun den ersten Teil von `sucheSchachteln()`. Als Erstes wollen wir einrichten, dass für jede Stellung eine neue `Schachtel` der Liste `alleSchachteln` hinzugefügt wird.

HINWEIS

In unserem Realweltbeispiel haben wir auf die Symmetrie geachtet und Schachteln eingespart. Darauf müssen wir nun nicht achten, der Rechner hat dafür genügend Rechenpower. Bei einer richtigen Schach – KI wird das dann aber wieder sehr wichtig, da die Anzahl aller möglichen Stellungen, dann doch sehr hoch ist.

AUFGABE 5

Ergänze den nötigen Code in *`sucheSchachteln()`* an Stelle 1.1.

HINWEIS

Falls dein Wissen über objektorientierte Programmierung etwas eingerostet ist, kannst du dir auch direkt die kommentierte Lösung anschauen. Dann kannst du in der nächsten Aufgabe wieder voll durchstarten.

Als nächstes sorgen wir dafür, dass die KI einen zufälligen Zug macht, man also gegen sie spielen kann. Dazu soll in `Schachtel.`*`macheZug()`* ein zufälliger Zug aus gewählt werden. Diese Funktion muss dann in *`sucheSchachteln()`* aufgerufen werden.

AUFGABE 6

Ergänze den Code für *`macheZug()`*. und füge den Aufruf der Funktion in *`sucheSchachteln()`* ein.

ACHTUNG

Achte darauf, dass der Zug im `game`-Objekt auch gespielt und im `chessboard`-Objekt visualisiert wird.

Die KI kann jetzt zufällige Züge spielen. Sie erkennt jedoch noch keine Stellungen und lernt auch nicht. Trotzdem haben wir nun unsere erste simple Schach-KI gebaut.

DIE KI ERKENNT STELLUNGEN

Bis jetzt legt die KI jedes mal ein neues Objekt Schachtel an, wenn sie an der Reihe ist. Sie sucht also nicht aus der Liste `alleSchachteln` heraus, ob sie die Stellung bereits kennt. Dies wollen wir nun ändern.

AUFGABE 7

Erweitere den Code in *sucheSchachtel()* an Stelle 1.2. **Bevor** eine neue Schachtel angelegt wird, soll nun durchsucht werden, ob die Stellung bereits bekannt ist. Wenn ja, wählen wir die Schachtel aus und lassen sie einen Zug ausführen.

ACHTUNG

Wenn die Schachtel bereits bekannt ist, soll natürlich keine neue angelegt werden. Achte darauf, dass die Funktion davor abgebrochen wird.