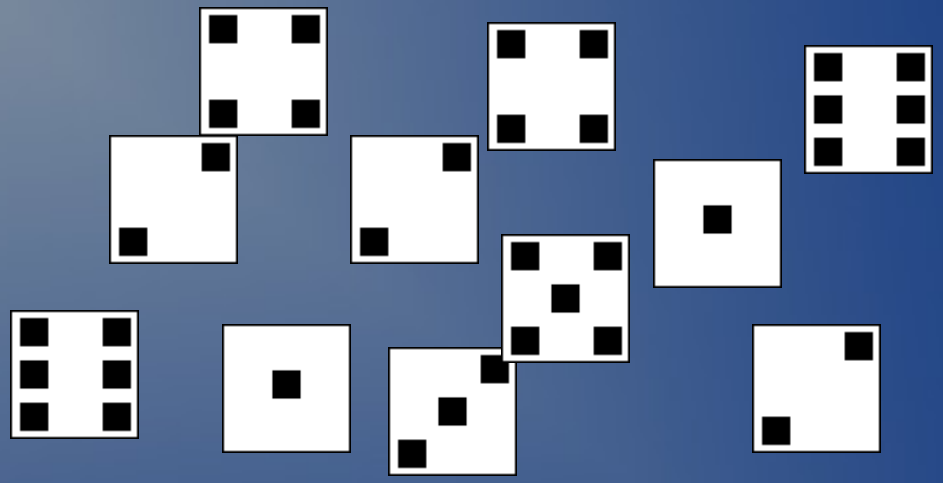
# Kniffel-Agenten



Von Alexander Holtkamp

# Übersicht

- Grundregeln
- Vorteil der „Monte Carlo“-Methode
- Gliederung des Projekts
- Aufbau „State“ - „Action“
- Kodierung von „State“ - „Action“
- Optimierung
- Aussicht

# Grundregeln von Kniffel

- Ziel des Spiels ist es, so viele Punkte wie möglich auf dem Spielblock zu erreichen
- Der Spielblock besteht aus 13 Feldern und 1 Bonusfeld
- Die Punkte in einem Feld entstehen durch bestimmte Würfelkombinationen
- Es darf insgesamt 3x mit allen Würfeln im Würfelbecher gewürfelt werden
- Nach dem 1. Wurf dürfen beliebig viele Würfel zur Seite gelegt werden, um den Würfelwert zu behalten
- Einmal zur Seite gelegte Würfel dürfen nicht erneut in den Würfelbecher gelegt werden, der Wert ist also fest
- Nach maximal 3x Neuwürfeln wird die Runde beendet, indem ein Feld auf dem Spielblock gewählt werden muss.

# Vorteil der „Monte Carlo“-Methode (1)

- Kniffel hat die Besonderheit, dass die letzten Zustände nicht entscheidend für die Bewertung des Spiels sind, sondern alle Aktionen, die man ausführt
- Q-Learning bewertet nur die unmittelbar vor dem Ausschütten der Belohnung verwendete Status-Action-Kombination
- Ein Lernverfahren ist notwendig, dass alle Schritte, die zur Belohnung geführt haben, bewertet werden

# Vorteil der „Monte Carlo“-Methode (2)

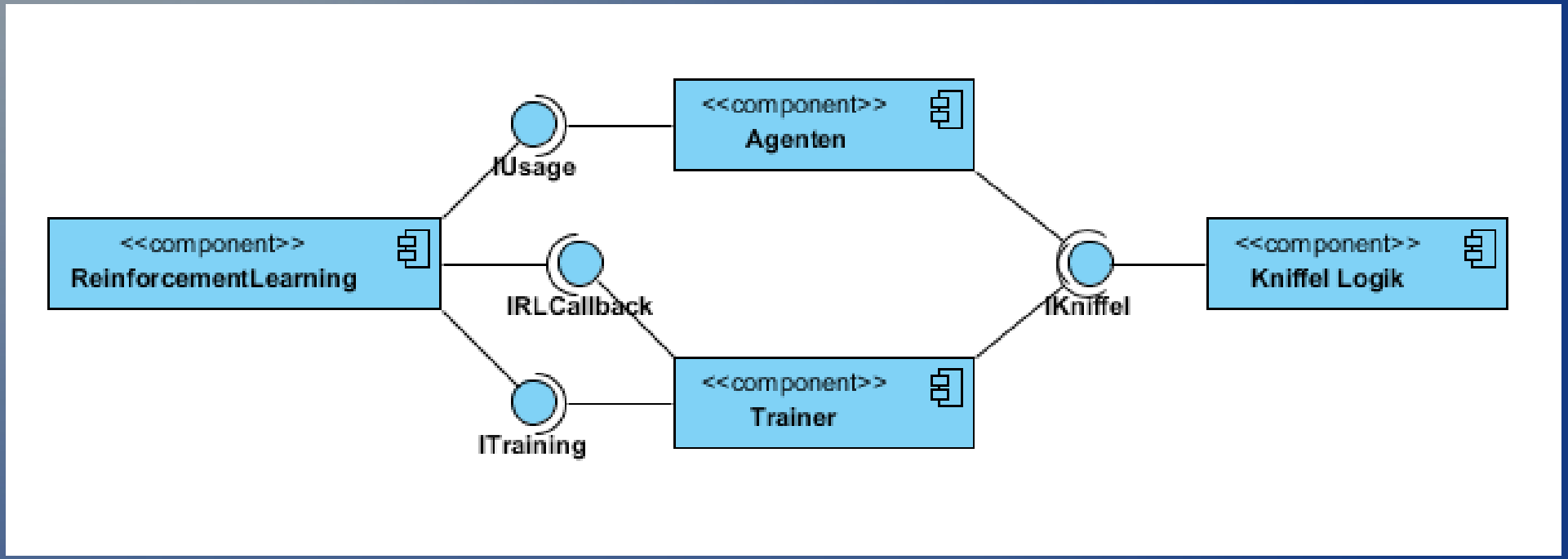
- Monte-Carlo: Speichert alle Status-Kombinationen, die durchlaufen werden. Sobald ein Status eine Belohnung gibt, wird diese Belohnung auf alle gesammelten Status-Actions-Kombinationen übertragen und die Liste danach geleert.
- Die Übertragung ist hierbei nicht 1:1 sondern bildet den Durchschnitt über alle Q-Werte, die bereits vorher ermittelt wurden.
- Dies macht die Speicherung der Anzahl, die angibt, wie oft diese Kombination schon belohnt wurde, notwendig, also bildet sich ein State-Action-Paar auf einem Wert-Anzahl-Paar ab, z.B::

  … (s1,a1) = (10,1), (s1,a2) = (0,4), (s2,a3) = (0,0)

  (s1,a1), (s1,a2) und (s2,a3) werden mit 20 belohnt.

  … (s1,a1) = (15,2), (s1,a2) = (4,5), (s2,a3) = (20,1)

# Gliederung des Projekts




IUsage wendet nur die ermittelten Q-Werte an und lernt nichts dazu. Es wird lediglich die Funktion getNextAction angeboten.

ITraining wendet in 90% der Fälle das Gelernte an und führt zu 10% zufällig gewählte Aktionen aus, um die Umgebung kennen zu lernen. IRLCallback stellt Interaktion mit der Umgebung dar, damit nicht alle Kombinationen bereits beim Aufruf übergeben werden müssen.

# Aufbau: „State“ - „Action“ (1)

State bildet den aktuellen Zustand des Spiels eindeutig ab.

Hierbei ist zu unterscheiden, welche Werte wichtige Informationen enthalten und für die Entscheidung, welche Aktion ausgeführt werden soll, entscheidend ist.

Es folgt eine Liste von möglichen Informationen, die abgebildet werden können, unabhängig von deren Wichtigkeit:

# Aufbau „State“ - „Action“ (2)

- ✓ Wie oft darf noch neu gewürfelt werden?
- ✓ Welche Werte haben die Würfel, die gerade angezeigt werden sollen
- ✓ Welche Würfel können neu gewürfelt werden?
- ✗ In welcher Runde befinde ich mich?
- ✗ Wie viele Runden gibt es?
- ✗ Der wievielte Spieler bin ich?
- ✗ Wie viele Spieler gibt es?
- ✓ Welche Felder auf dem Spielblock kann ich noch verwenden?
- ✗ Wie viele Punkte habe ich schon, und wo?
- ✗ Was ist in den Runden davor passiert?

# Aufbau „State“ - „Action“ (3)

Welche Aktionen sind theoretisch möglich?

- Würfeln
- Zur Seite legen von Würfel X
- Eintragen in Feld X

Bei 5 Würfeln und 13 Feldern gibt es also 19 verschiedene mögliche Aktionen.

# Kodierung von „State“ - „Action“ (1)

- Anzahl Rest-Würfe

  Typ: Ganzzahl-Wert, Werte: 0, 1, 2, 3

  4 unterschiedliche Werte, also **2 Bit benötigt**
- Würfel-Werte

  Typ: Ganzzahl-Wert, Werte: 1, 2, 3, 4, 5, 6

  6 unterschiedliche Werte sind 3 Bit / Würfel, also **15 Bit benötigt**
- „Halte“-Werte

  Typ: Wahrheits-Wert, Werte: wahr, falsch

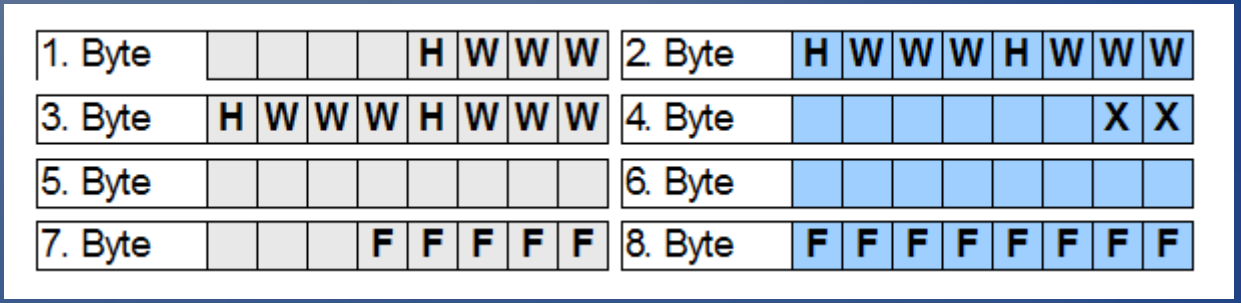
  2 unterschiedliche Werte sind 1 Bit / Würfel, also **5 Bit benötigt**
- Freie Felder

  Typ: Ja/Nein-Wert, Werte: wahr, falsch

  2 unterschiedliche Werte sind 1 Bit / Feld, also **13 Bit benötigt**

# Kodierung von „State“ - „Action“ (2)

- Insgesamt also 35 Bit, zum Speichern ist also ein Wert vom Typ: Long (64 Bit) erforderlich
- 35 Bit macht 34359738368 theoretische Kombinationen
- Der Long ist folgendermaßen belegt:




- Nicht alle Kombinationen genutzt, darum Optimierungsmöglichkeit bei den Würfelwerten

# Kodierung von „State“ - „Action“ (3)

Die Kodierung der Aktionen gestaltet sich recht einfach

- Aktion

  Typ: Wahrheits-Wert, Werte: wahr, falsch

  2 unterschiedliche Werte sind 1 Bit / Aktion, also 19 **Bit benötigt**

- Insgesamt also 19 Bit, zum Speichern ist also ein Wert vom Typ Integer erforderlich.
- 19 Bit macht 524287 theoretische Kombinationen

# Optimierung (1)

- Die Speicherung der State-Action Kombinationen und den dazugehörenden Q-Werten bzw. die Kombination aus Q-Wert und Auftreten hat einen enormen Speicher-Verbrauch zur Folge sobald man das Training beginnt.
- Bei Q-Learning (maximal):

  64 Bit (State) + 32 Bit (Action) + 32 Bit (Q-Wert) = 128 Bit = 16 Byte

  16 Byte * 2 ^ 35 (State-Kombinationen) * 2 ^ 19 (Action-Kombinationen)

  = 2 ^ 58 Byte = 288230376151711744 Byte = 262144 Tera Byte
- Bei Monte Carlo (maximal):

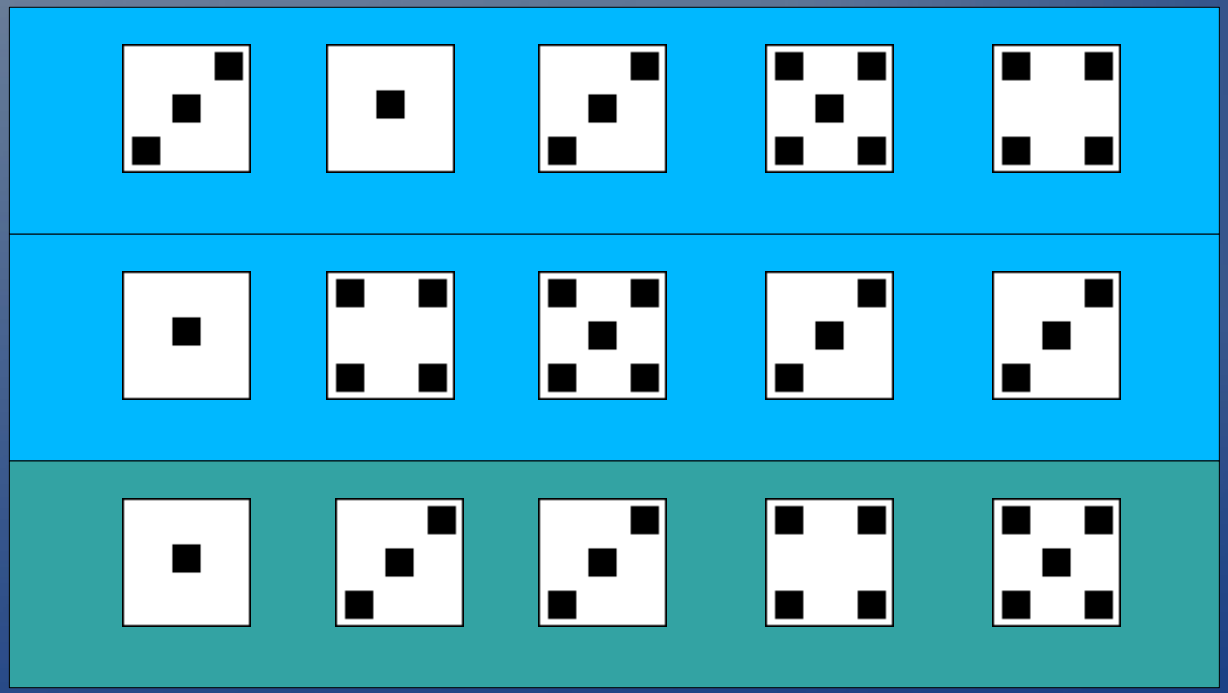
  64 Bit (State) + 32 Bit (Action) + 32 Bit (Q-Wert) + 32 Bit (Auftreten) = 20 Byte

  20 Byte * … = 327680 Tera Byte

- Ziel: Reduzierung von State und Action-Kombinationen

# Optimierung (2)

Sortierung der Würfel nach Augenwert sorgt dafür, dass nicht alle Kombinationen möglich sind, Beispiel:



**Kombination 1**

**Kombination 2**

**Vereinheitlicht**

# Optimierung (3)

- Sortierung der Würfel bereits implementiert
- Dies senkt die möglichen Zustandskombinationen schon mal enorm (weniger Status-Kombinationen = weniger Status-Action-Kombinationen = weniger Speicherverbrauch)

- Um den benötigten Datentypen für den KniffelState Integer (32 Bit) herunterzubringen, Funktionalität erstellen, die die möglichen sortierten Würfelwerte auf maximal 12 Bit abbildet
- Vorteil: Speicherverbrauch pro State-Action Kombination um 32 Bit verringert (weil man einen Integer statt eines Long-Typen benutzen kann) => gut für die Weitergabe der Daten
- Nachteil: Ausführungszeit erhöht, bei linearer Verbesserung des Speicherplatzverbrauchs => schlecht fürs Training
- noch nicht implementiert...

# Optimierung (4)

- Bei Aktionen ist nicht viel Spielraum
- Option: Entfernen von Einzelaktionen fürs Eintragen und nur eine Aktion „Eintragen“ anbieten, durch die dann automatisch die Eintrag-Aktion mit der höchsten Punktzahl gewählt wird.
- Risiko, dass etwas verloren geht, da nicht immer die höchste Punktzahl, die optimale Entscheidung ist.
- Beispiel: 0 Würfe übrig, Kleine Straße, Große Straße und Chance bereits belegt

- Ein menschlicher Spieler würde 1 wählen, da hier der geringste Verlust ist, die Implementation würde 5 eintragen, weil dies die höchste Punktzahl erzielt
- Man könnte natürlich solche Fälle einprogrammieren, allerdings würde man dann ein Strategiemuster verwenden, was nicht hier gemacht werden soll

# Aussicht

- Letzte Option mittlerweile implementiert, hierdurch kann man erste Erfolge sehen, auch wenn hierdurch bereits benannte Strategie verloren geht
- So implementiert, dass nur die beste Aktion für den Agenten angeboten wird, hierdurch kann er weiterhin das richtige Spiel spielen ohne, dass man die Aktionen generell entfernen muss
- Wegen bei steigender Datengröße immer langsameren Ausführung vermutlich Anbindung an neuronales Netz