Classifier Systems und Defensive Strategies

Seminar Lernen in Spielen

13.06.2006

Inhalt

- Learning Classifier Systems
  - Definition und Repräsentation
  - Architektur
  - Vor- und Nachteile
- Verteidigungsstrategien mit genetischen Algorithmen
  - Repräsentation der Aktionsparameter bei Rocket Jumping und Ausweichen von Raketen
  - Genetische Operatoren
  - Evolution und Anwendung
  - Beispielimplementierung in Quake II

Learning Classifier Systems (LCS)

- LCS kombinieren drei Techniken der künstlichen Intelligenz
  - Genetische Algorithmen
  - Regelbasierte Systeme
  - Reinforcement Lernen
- Sie können die beste Aktion in der gegeben Situation lernen

  ➔ Das Klassifizierungsproblem lösen

Repräsentation von LCS

- Ein Satz von Regeln, die „Klassifizierer“ genannt werden
- Jeder Klassifizierer besteht aus zwei Teilen
  - Head (die Eingabedaten verwerten)
  - Body (eine geeignete Reaktion finden)
- Zusätzliche Informationen
  - von den Regeln gespeichert
  - Nutzen abschätzen
  - Fehlerwahrscheinlichkeit voraussagen

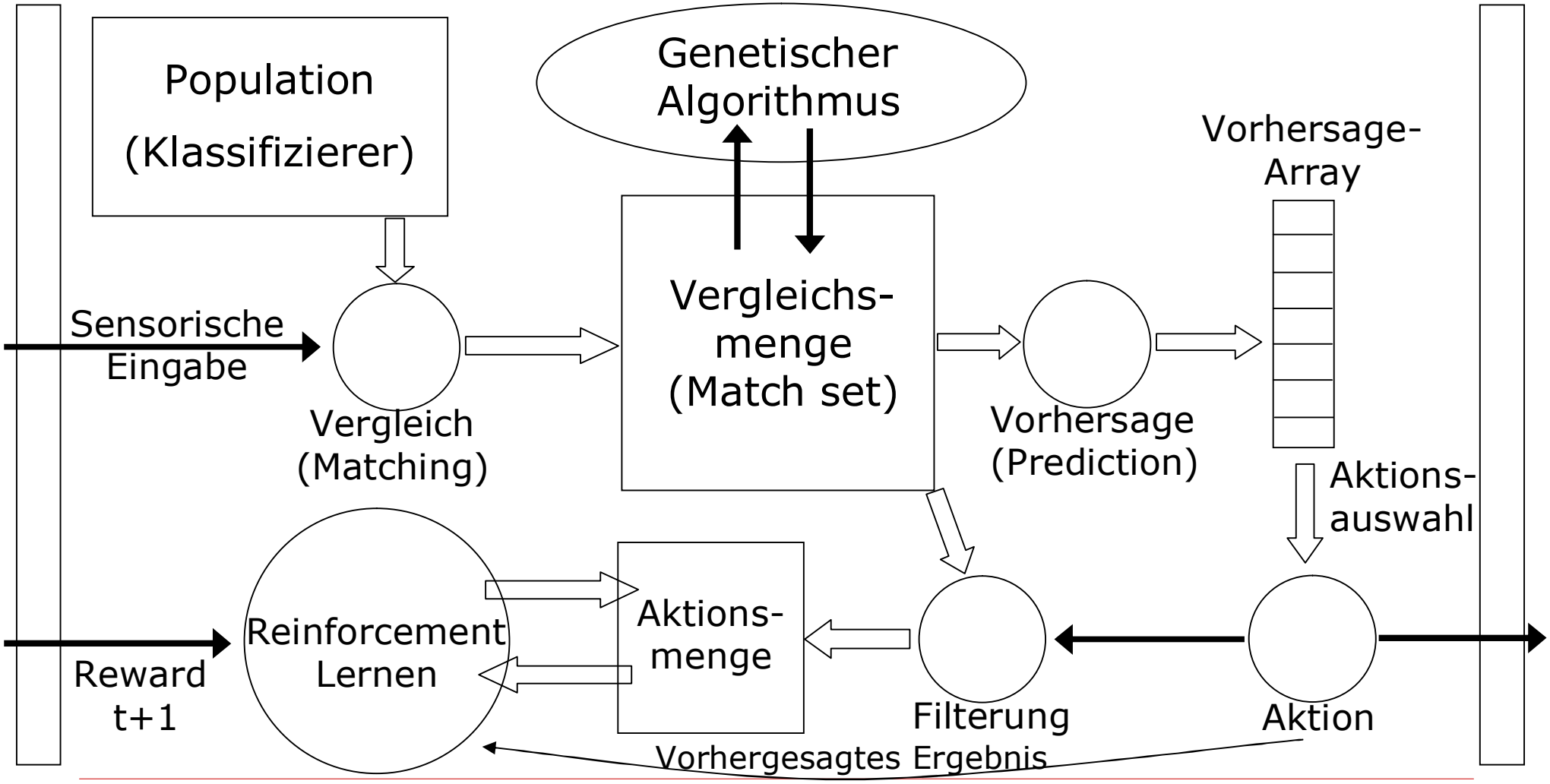
Head und Body

- Verarbeitung der Eingabe (*Head*)
  - Modellierung des Klassifizierers mit 3 Werten: „0“, „1“ und „#“ (don‘t care)
  - Damit wird eine Generalisierung ermöglicht
  - Ein „Match“ liegt vor, sobald ein Korrespondierendes Bit 1 oder # ist.
- Aktionen des Klassifizierers (*Body*)
  - Keine Generalisierung notwendig ➔ 0,1

Weitere Attribute des LCS

- Vorhersage (*prediction*) korrespondiert mit
- erwartetem Ertrag (*return*)
  - Über die Zeit aufsummierter Lohn (*reward*)
  - Güte des Klassifizierers auf lange Sicht
- Geschätzte Genauigkeit (*accuracy*)
  - Konsistenz des Klassifizierers wird bestimmt
  - Niedrige accuracy → Schlecht bei Vorhersagen
  - Hohe accuracy → Gutes Verständnis der „Welt"
- Ein Faktor für die *fitness* wird gesucht
  - Früher: Vorhersage des return
    - → Das System wird daran gemessen, wie gut es **denkt**, dass es ist. Führt nicht zu optimalem Ergebnis. Warum?
  - Heute: accuracy

Architektur des LCS



Vor- und Nachteile

- LCS gut für die Schätzung des „benifits" von Aktionen
  - Die besten Aktionen werden verwertet
  - Wissen über die „Welt" ist i.d.R gut ➔ funktionierendes Lernv.
  - Wissen des Entwicklers kann einfließen
- Binäre Repräsentation ist komplex
  - Schlecht zu lesen
  - Input häufig nicht binär (Fließkommazahlen, Arrays)
    ➔ müssen (umständlich) konvertiert werden
  - Erweiterung: Symbolmenge statt Binärwerte
- Vielfältige Einsatzmöglichkeiten in Spielen
  - Anpassung in Online-Anwendungen
  - Kontrollprobleme häufigste Anwendungsform
    ➔ Das können Regel-Lerner allerdings auch gut

Fazit: Fast überall einsetzbar, aber sehr hoher Ressourcenverbrauch
Anwendung: Obstacle Avoidance, Waffenauswahl
In der Praxis sind andere Techniken meist eher angebracht

Beispiel in dem LCS „fast“ vorkommen: Creatures 1-3

- Classifier System vorhanden
- allerdings „hard coded“
- Klassifizierung aller Kreaturen
- Genetische Algorithmen im Spiel umgesetzt
  - Zufallsgenerierung
  - Mutation
  - Phänotyp/Genotyp
  - zusätzlich: „Gehirn“ etc.

Bsp. „Classifier System“

family	genus	species	name	url
1	1	1	Rock near Norn pool	Masha
1	1	2	Norn door cutaway	Masha
1	1	3	Norn hump cutaway	Masha
1	1	4	Norn entrance to burrow	Masha
…				
3	10	55100	tables	chani
3	10	55101	grabber	chani
3	10	55102	chest	chani
4	1	1	male norn	Masha
4	1	2	female norn	Masha
4	1	1000	pending	serstel

Creatures Screenshot

Inhalt

- Learning Classifier Systems
  - Definition und Repräsentation
  - Architektur
  - Vor- und Nachteile
- Verteidigungsstrategien mit genetischen Algorithmen
  - Repräsentation der Aktionsparameter bei Rocket Jumping und Ausweichen von Raketen
  - Genetische Operatoren
  - Evolution und Anwendung
  - Beispielimplementierung in Quake II

Adaptive Verteidigung mit genetischen Algorithmen

- Verteidigungsstrategien neben Angriffsstrategien auch wichtig für realistisches Spielen
  - Raketen ausweichen (Laufen oder Springen)
  - Rocket Jumping

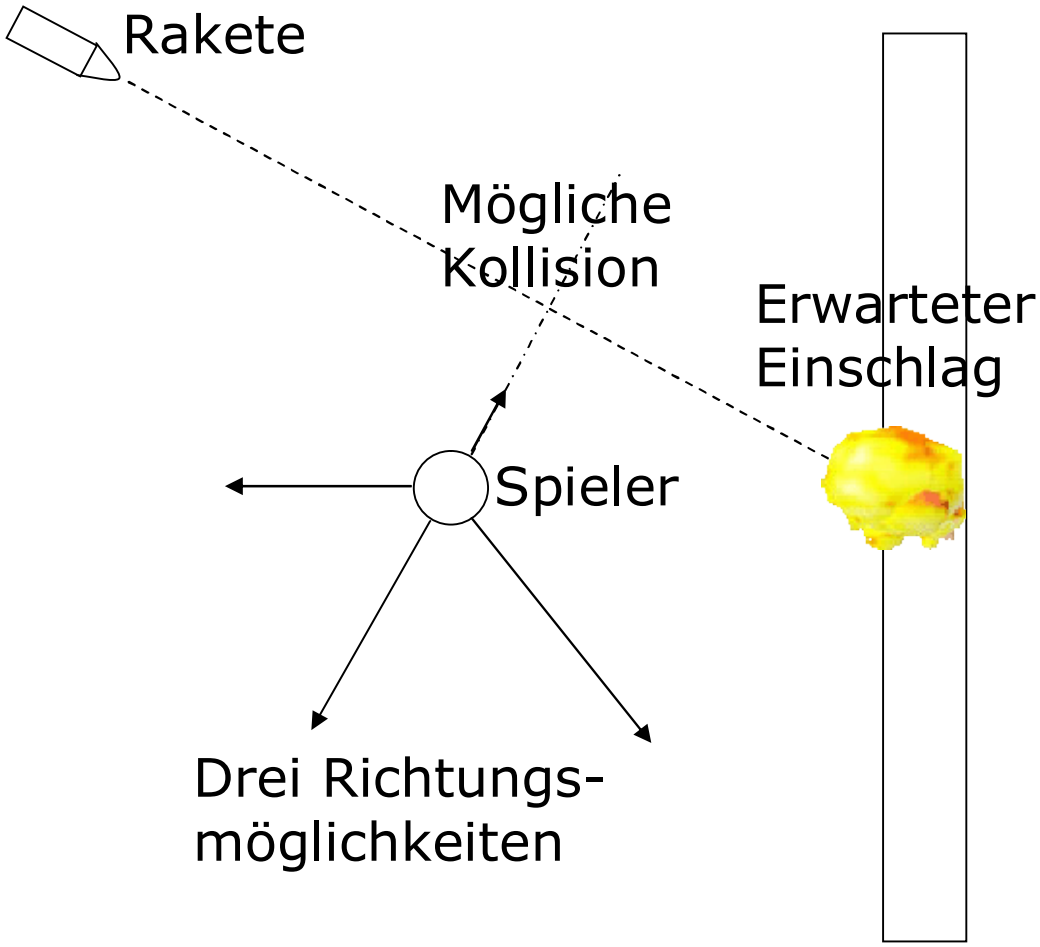
- Genetische Algorithmen zur Manipulation von Aktionssequenzen

Aktionen und Parameter

- Eine Teilmenge aller möglichen Aktionen als Aktionsmenge (z.B. nur grobe Richtung)
  - Einfacher zu entwickeln
  - Macht das Lernen schneller
- Manche Aktionen sind gegebene Parameter
  - Nur wenige parameterlose Aktionen
  - Das „was“ und das „wie“ werden getrennt
- Kombination der Parameter als *Sequenzen* von Aktionen

Bsp. Aktionsmenge

Aktion	Parameter
Look	Richtung
Move	Gewichte
Fire	-
Jump	-



- Timing
  - Relativ ➔ Offset zur vorherigen Aktion
  - Absolut ➔ Unabhängig von anderen Aktionen

Genetische Operatoren

- Die Basis-Operatoren für genetische Algorithmen werden eingesetzt
- Zufällige Generation
  - Time offset → Fließkommazahl mit MAX (z.B. 2 Sek.)
  - Aktionstyp → Zufällig generiertes Symbol (z.B. Move)
  - Parameter → Abhängig von Aktionen (z.B. weg von Rakete)
- Kreuzen (Crossover)
  - Ein-Punkt-Kreuzung → Zufälliger Split zwischen 2 Eltern und Zusammensetzen von 2 Kindern (in Bsp. verwendet)
  - Große Wahrscheinlichkeit für gute Sequenzen nach der Kreuzung zusammenzubleiben
- Mutation
  - 2 Arten von Mutation
  - Die Länge einer *Sequenz* durch Hinzufügen oder Wegnehmen von Aktionen verändern
  - Die *Aktionen* selbst werden durch Verändern der Werte mutiert

Evolutionäre Anpassung

- Konstante Größe der Population
  ➔ Speichereffizienz
- zu Beginn: Population zufällig generiert
- Evolutionsschritte auf Anfrage
  - Jedes Individuum ohne Fitness-Wert wird aus der Population gezogen und evolviert
  - Später: 2 Eltern werden mit der Wahrscheinlichkeit ihrer Fitness gekreuzt und ggf. mutiert
    ➔ Entstehende ‚Kinder' werden evolviert
- Entfernen von Individuen
  - Mit Wahrscheinlichkeit von 1-(relative Fitness)
  - Ähnlichkeit mit anderen Sequenzen verstärkt Wahrscheinlichkeit entfernt zu werden
    ➔ Abwägen zwischen Elite und Unterschiedlichkeit

Die Fitness berechnen

- Rocket Jumping
  - Ziel ist es, besonders hoch zu springen
    ➔ Reward steigt quadratisch mit der Höhe
- Raketen ausweichen
  - Distanz zwischen Spieler und Explosionspunkt maximieren
  - Stehenbleiben (in gewisser Distanz) sollte vermieden werden
    ➔ *Unterschied* in der Distanz wird maximiert
  - Schaden wird von der Fitness abgezogen

Variablen zur Ergebnisoptimierung

```
// ------------------------------------------------------------------------
// Constant Declarations
// ------------------------------------------------------------------------
const int k_PopulationSize = 8; ///< how many behaviors are stored?
const int k_MaxActionsPerSequence = 8; ///< limit of actions in a behavior
const float k_MaxSequenceLength = 2.0f; ///< maximal number of seconds in a sequence
const int k_DodgesPerTrial = 3; ///< number of dodges used to evaluate performance
```

Anwendung

- Die vom genetischen Algorithmus gegebenen Kandidaten-Sequenzen müssen vom „Animat“ mit konkretem Verhalten getestet werden
- Rocket Jumping und Raketen ausweichen separat lernen, da diese sowieso unabhängig sind. 2 Alternativen
  - Eine Fitness Funktion, verschiedene Phasen
  - Zwei Fitness Funktionen, die gleichzeitig gelernt werden
- Ergebnis: Zum Ausweichen vor Raketen wird ein Rocket Jump ausgeführt

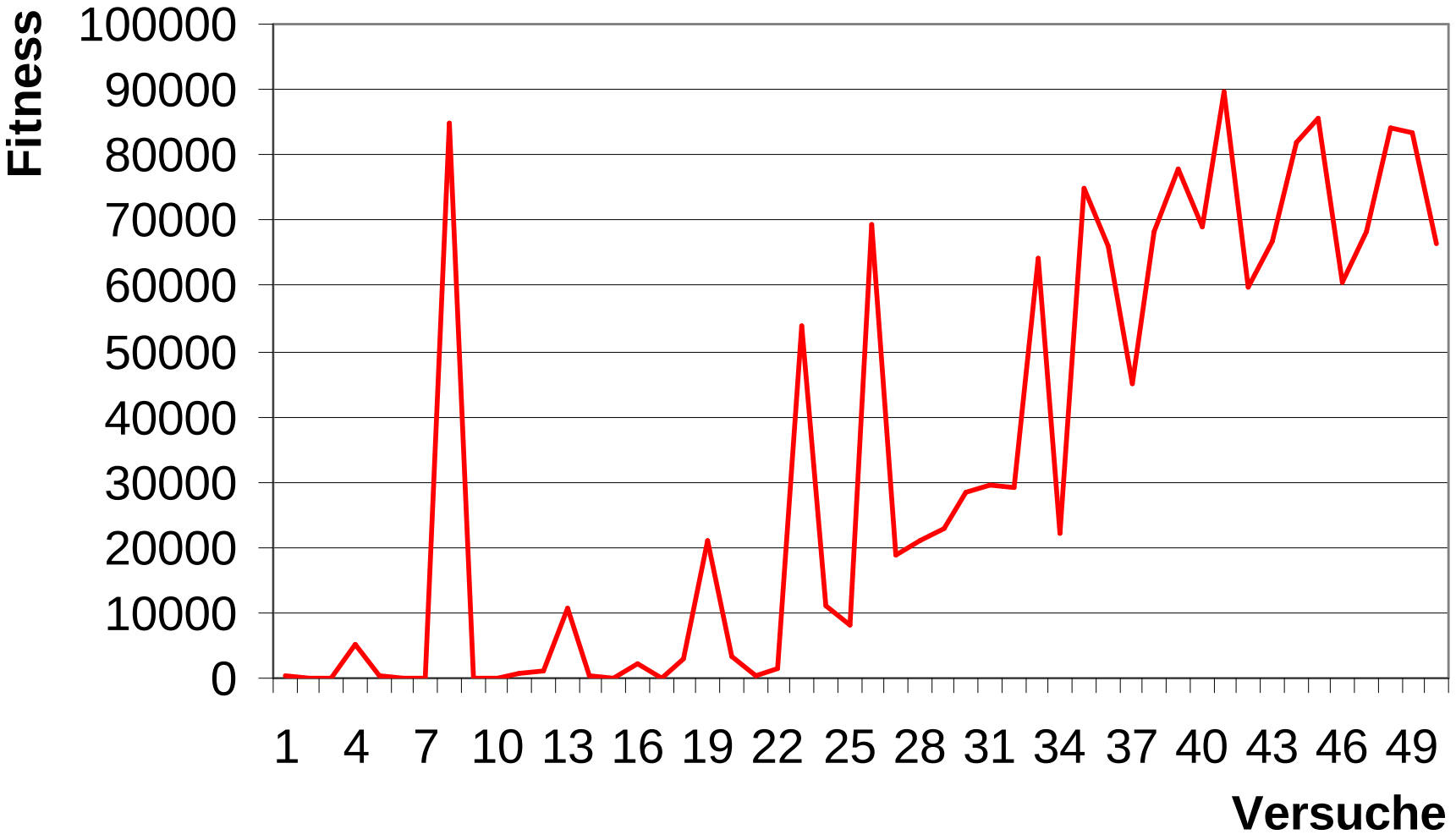
Evaluation

- Rocket Jumping wird schnell gelernt, sobald der Animat gelernt hat, dass Raketen den Sprung verbessern
- Ausweichen wird auch recht schnell gelernt, da die Repräsentation einfach gehalten wurde (allerdings unvollständig)
- Fitness als Fließkomma-Wert zu implementieren ist von großem Nutzen
  ➔ Besser als boolsche Werte
- LCS wären nicht gut für dieses Problem geeignet. Warum?

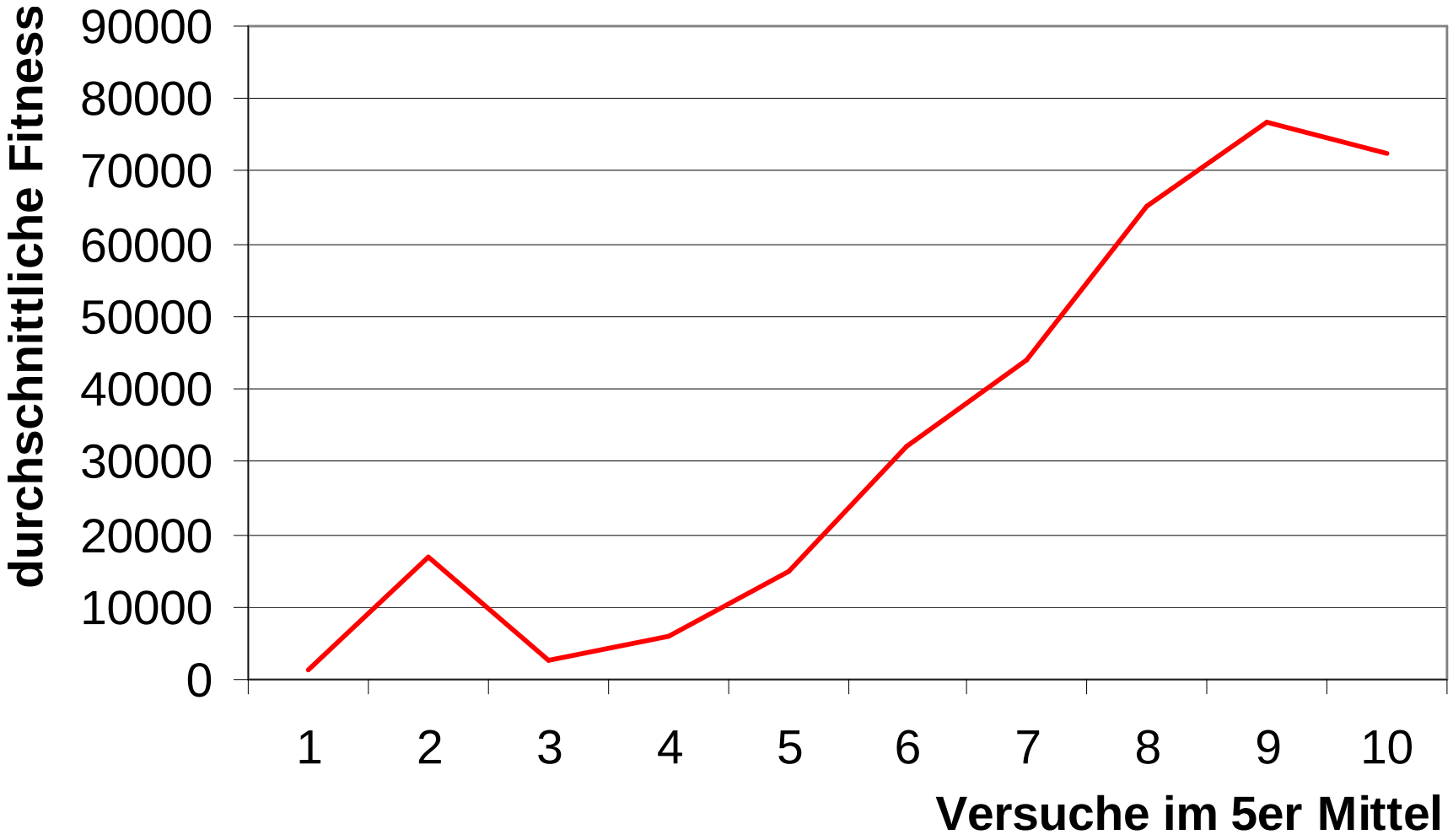
Umsetzung in Quake II

- Animat, der Rocket Jumping und Ausweichen lernt („Kanga“)
- Schnelle Umsetzung
  - ➔ nach etwa 50 Sprüngen hohe Fitness für Rocket Jump
- Repräsentation mit 2 Fitness Funktionen
- Praxis etwas schwieriger als Theorie
  - Rocket Jumping verursacht Schaden → Lernen dauert etwas länger
  - Raketenwerfer wird benötigt für Rocket Jump
  - Für Ausweichen wird schießender Spieler benötigt

Rocket Jumping Fitness



Rocket Jumping Average



Quellen

- Alex J. Champandard: AI Game Development, New Riders Publishing, 2003, Chapters 32 (Genetic Algortihms), 33 (Learning classifier systems, 34 (Adaptive defensive strategies)
- http://aigamedev.com/ und http://aigamedev.com/Forum/ (Implementierung der Quake II Bsp. aus dem Buch)
- http://fear.sourceforge.net/ Foundations for Genuine Game AI (Implementierung Quake II teilweise aktueller)
- http://en.wikipedia.org/wiki/Learning_classifier_system Definition LCS
- http://lcsweb.cs.bath.ac.uk/papers/Reveley2002a LCS-Implementierung für Poker
- http://www.cems.uwe.ac.uk/lcsg/ Learning Classifier Systems Group (Anwendungs-Bsp.)
- Seredynski, F., Cichosz, P., & Klebus, G. P. (1995). Learning classifier systems in multi-agent environments (http://www.ise.pw.edu.pl/~cichosz/pubs/) eher Spieltheorie
- http://creatures.wikia.com/wiki/Creatures_Wiki_Homepage Informationen über Creatures