Technische Universität Darmstadt
Institut für Informatik
Sommersemester 2006

Alex J. Champandard:
AI Game Development, Chapters 13-18
Shooting and Aimining

Veranstaltung: KE und Lernen in Spielen
Leiter: Prof. Dr. Johannes Fürnkranz
Referent: Michael Burkhardt

Einleitung

- Nach dem die Bots sich bewegen können lernen sie jetzt schießen
- Was ist schießen?
  - Zielauswahl
  - Zielen
  - Waffe abfeuern
  - Später vielleicht auch erst noch Waffe auswählen
- Waffenauswahl => Typisierung der Waffen
  - Reichweite
  - Flugverhalten der Projektile
  - Wirkung

Gliederung

- Umweltanalyse
- Wie schießt der Mensch?
- Wie schießt der Computer?
  - Physik-Modelle für bewegte Ziele
  - Neutrale Netze und schießen -> später mehr wenn es um Multilayer Netze geht
- Verhindern/ aus bügeln von Fehlern beim Zielen

Umweltanalyse

- Kampf Mann gegen Man (Nahkampf)
  - Waffentypen
  - Waffen und die Rolle der Umwelt
- Fernkampf
  - Waffentypen
  - Waffen und die Rolle der Umwelt

Nahkampf

- Waffen sind Handwaffen
- Axt
- Schwert
- Einfach nach zu bauen ein Objekt bewegt sich und trifft ein anderes
- Dann wird ein bestimmter Schaden von den Lebenspunkten abgezogen

Fernkampf

- Waffen für den Fernkampf sind:
  - Raketen
  - Pfeile und Bogen
- Wenn man die physikalischen Eigenschaften weg lässt ist dies genauso einfach wie der Nahkampf
- Er wird erst durch die Umwelt komplex:
  - Angreifer nützt Deckung aus (Scharfschütze)
  - Verteidiger nutzt Deckung um Projektilen auszuweichen
  - Physik der Projektile selbst

Wie kämpft ein Mensch?

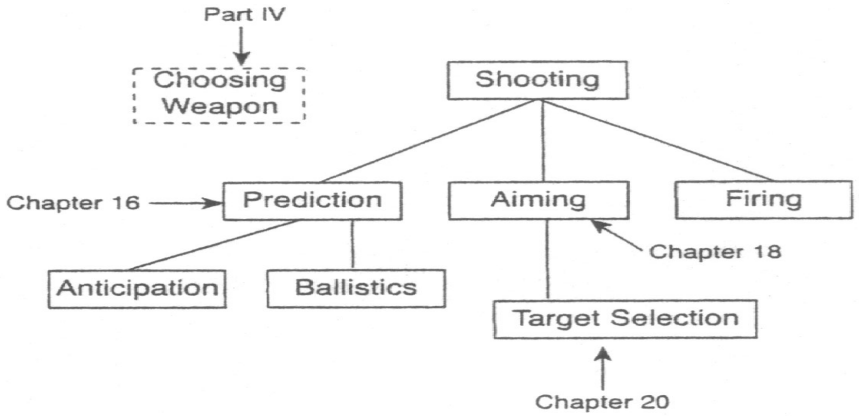


Figure 14.1 The hierarchy of skills required to successfully shoot down a moving enemy with distance weapons.

- Er benutzt seine Fähigkeiten
  - Kraft
  - Geschick
  - Strategie

Nahkampf

- Nahkampf
  - Lernen der Kombinationen
  - Einteilen der Kraft des Avatar
  - Solange üben bis es in Fleisch und Blut über geht

Fernkampf

- Fernkampf
  - Abschätzen der Entfernung
  - Ungenauigkeiten der Maussteuerung ausgleichen
  - Ruhe und Genauigkeit
  - Schätzen was macht der Gegner:
  - Man sucht sich ein leichtes Ziel
  - Zielen
  - Bewegung erahnen
  - Ballistik der Waffe bedenken
  - Eventuelle vorhalten
  - Abdrücken

Rollenverhalten

- Rollenverhalten
  - Verteidiger legen mehr Wert auf Sperrfeuer als Deckung
  - Angreifer wollen Druck machen mit gezielten Treffern

Wie soll sich jetzt ein NPC verhalten?

Figure 14.2 Two scenarios showing prediction abilities. On the left, a player uses a rocket to intercept the target. On the right, the plan is to aim for the floor where the enemy is predicted.

- Er soll sicher scheißen. -> Sich nicht selbst treffen
- Möglichst natürlich handeln
- Aber auch möglichst effizient sein -> eben wie ein Spieler mit Training

Waffenmodel

- In Ego-Shootern genaues Zielen unwichtig
- Die Masse bringt die nötigen Treffer
- Waffen haben oft nur einfaches Physik-Modell
  - Nur Gravitation wirkt auf die Geschosse

Schießen

- Zum schießen braucht ein NPC vier Eingaben
  - Wissen über seine Waffe(n)
  - Wo und wer ist sein Ziel
  - Die Beschaffenheit der Umgebung
  - Die Möglichkeit die Waffe physisch zu benutzen
    - Er braucht z.B. Eine Möglichkeit seinen Körper zu bewegen

Implementieren eines Schützen

- Zwei Arten von Schützen:
  - Scharfschütze
  - „normaler“ Schütze
- Scharfschütze:
  - Schießt nur auf unbewegliche Ziele
  - Muss sich sicher sein das sich das Ziel nicht bewegt
  - Schützt sich auch damit, wenn er nur die sicheren Ziele auswählt
  - Ums seine Schußrate zu erhöhen muss man ihm vielleicht ein weniger strenges Regelwerk geben (Perceptron)

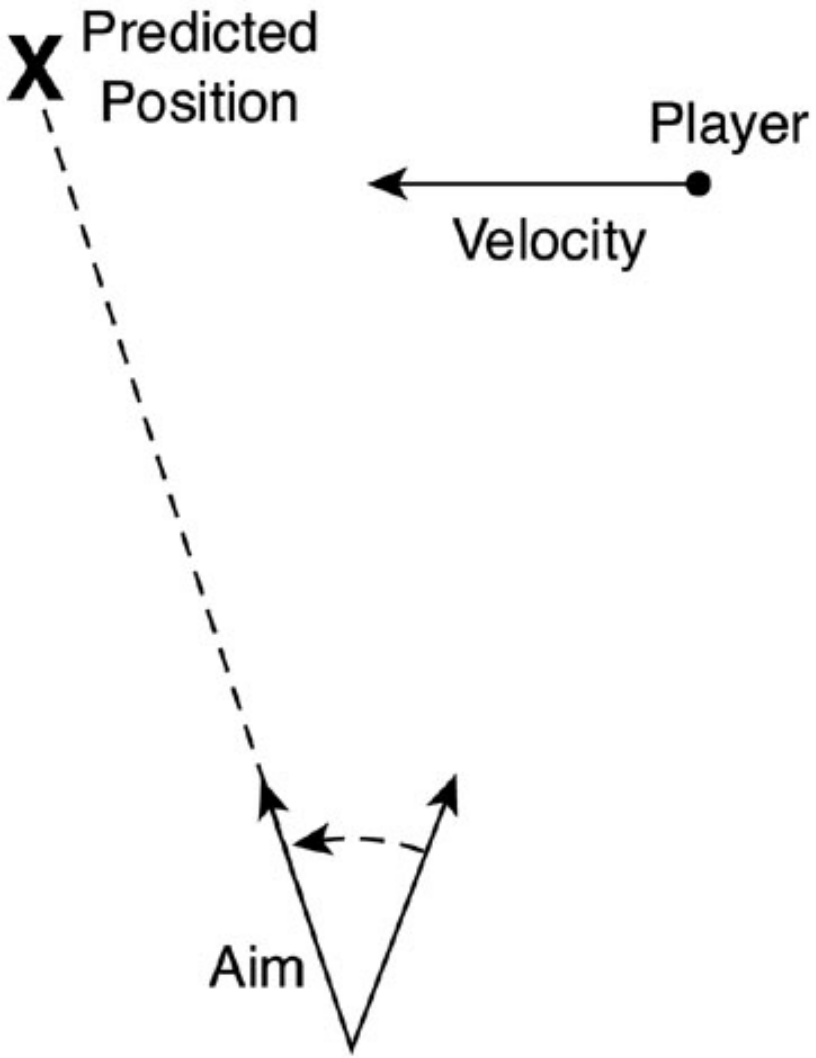
Codebeispiel: Scharfschütze

```
Global variables:
Timestill          time since the enemy
 began standing still
StandingStill   1 if standing still, 0
 otherwise


When it begins standing still
     StandingStill=1
     Timestill=now If

StandingStill and more than X seconds
 have elapsed since
Timestill
     Shoot
```

Bewegte Ziele



- Scharfschützen AI wird erweitert
- Es bekommt Tracker-AI hinzu
- Faktoren:
  - Geschwindigkeit der Kugel eine Rolle
  - Entfernung Schütze Ziel
  - Bewegungsrichtung Ziel

Codebeispiel Tracker

```
float d=distance (sniper, target)
float time=d/bulletspeed
point pos=predictposition(target,time)
if aiming at pos shoot()
else target at pos;
```

Waffen komplexer

- Maschinengewehre:
  - Anzahl der Kugel berücksichtigen
  - Ladezeiten
- Raketen:
  - Ballistik
  - Lenksysteme:
  - keines -> Ballistische Rakete
  - IR oder Radar -> Rakete ist nachdem Abschuss ein neuer Akteur gesteuert mit Chasing Algorithmen

Das Perceptron

- Nachempfunden nach den Neuronen des Gehirns
- Kann trainiert werden damit bestimmte Ergebnisse erzielt werden
- Es gibt einfach und multilayer Ausführungen

Entstehungsgeschichte

- Frank Rosenblatt erfindet es 1957 am Cornell Aeronautical Labotory
- Es sollte helfen menschliches Gedächtnis und Lernen zu verstehen
- Er baut auf den Studien von Neurobiologen auf.

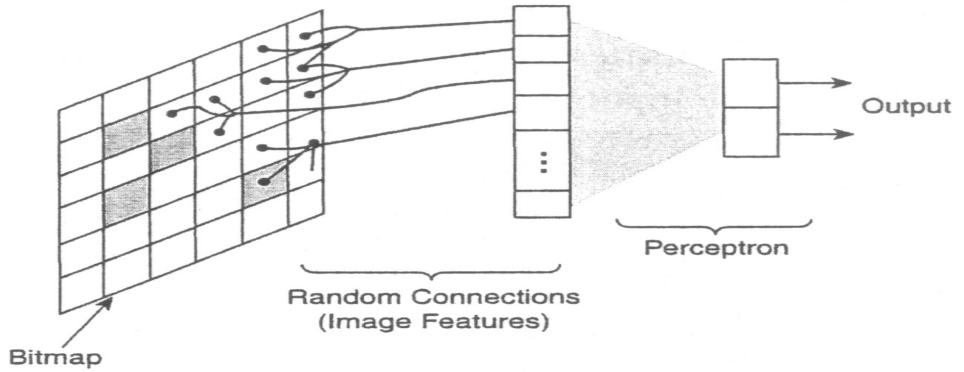


Figure 17.1 Rosenblatt's perceptron connected to a bitmap image, capable of recognizing some of its features.

Erklärung des einfachen Perceptrons

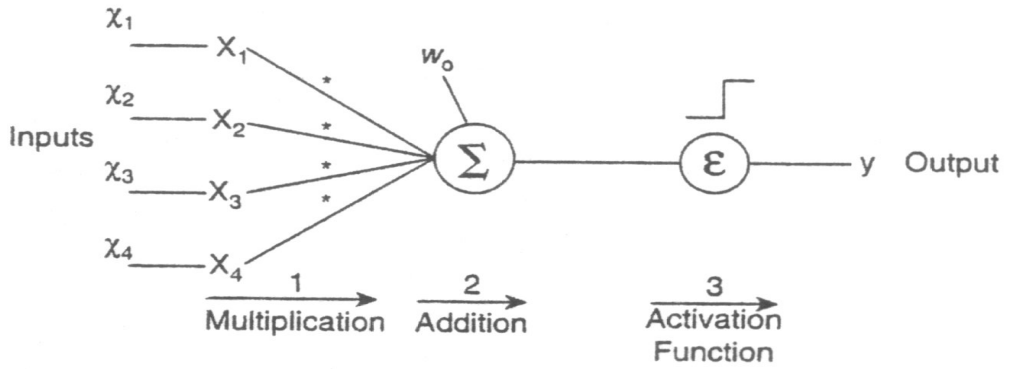


Figure 17.3 Outline of the operations used for computing the output of a perceptron, based on the input pattern.

- Output ist eine gewichtete Summe der Inputs

$$\zeta = \sum_{i=0}^{n} w_i x_i$$

$$= w_0 + \sum_{i=0}^{n} w_i x_i$$

$$y = \sigma(\zeta)$$

$$\sigma(x) = 1 \; wenn \; x > 0$$

$$oder$$

$$\sigma(x) = 0 \; wenn \; x \leq 0$$

Codebeispiel

```
net_sum  = 0
for all i
net_sum += input[i] * weight[i]
end for
output = activation( net_sum )


// input array
// weight array
// beide sind vordefiniert
// activation Funktion die σ(x)
   simmuliert
```

Verbesserung des Output

- Man kann den Output verbessern in dem man die die Gewichte verbessert.
- Dies geschieht durch Training
- Oft verwendet man hierfür die Delta Rule

Delta Rule

- Jedes Gewicht trägt eigenständig zum Output bei
- Der Betrag hängt vom Input ab
- Fehler des Perceptrons sind die Schuld der Gewichte
- Fehlerberechnung

$$E = \frac{1}{2}(t - y)^2$$

y ist der Output des Perceptrons

t ist der gewünschte Output

Fehlerkorrektur

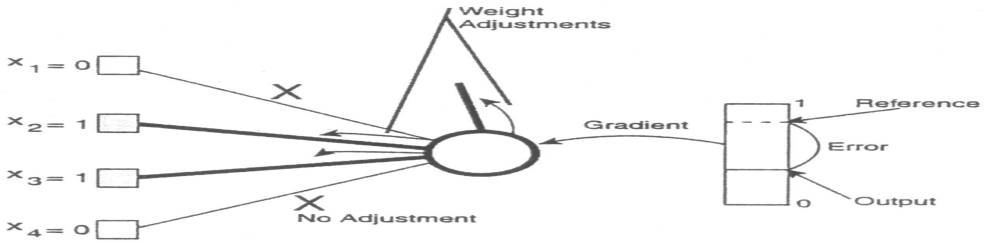


Figure 17.11 Correcting the weights of the perceptron based on the output error, proportionally to the input values.

$$E = \frac{1}{2}(t-y)^2 \quad E' = (t-y)^2$$

$$\frac{\delta E}{\delta w_i} = x_i(t-y)$$ | Ableitung

$$\Delta w_i = \eta x_i(t-y)$$ | η Konstante (Lernrate). Sie wir vom Programmierer festgelegt.

Anwendungsarten

- Delta Rule kann jetzt auf zwei Arten angewendet werden:
  - Einzel auf jedes Trainingsset (perceptron training algorithm)
  - Auf alle, wenn alle benutzt wurden (batched delta rule)
- Wenn η klein genug finden beide gute Lösungen
- BDR bringt besser Lösung ->global
- PTA kann auch nur lokal beste Lösung liefern

Aiming mit Perceptron

- Simuliert die Art zu zielen mit der Maus
- Dazu werden verschiedene Winkelkombinationen berechnet und mit der BDR gelernt
- Der bot dreht sich jetzt gleichmäßiger und schießt mehr wie wie ein Mensch

Fazit

- NPC kann gute Schießleistungen bringen, wenn er mit Perceptoren ausgestattet ist.
- Sie können eingesetzt werden für;
  - Freund- Feind-Erkennung
  - Zielen
  - Schießen
  - Verhindern reinen Nahkampf
  - Brauchen aber viel Zeit wenn alle Schritte einzeln gemacht werden d.h. ein Perceptron pro Aufgabe

Quellen

- Grundlage
  - Alex J. Champandard: AI Game Development, New Riders Publishing, 2003, Chapters 13-18.
- AI in Games
  - http://www.ai-depot.com/
  - http://www.aigamedev.com/
  - http://www.cgf-ai.com/todays_tfps_ai.html
  - http://www.aigamedev.com/sclrb/

Quellen

- Shooting
- http://www.peachpit.com/articles/article.asp?p=1
- Perceptron
  - http://wwwuser.gwdg.de/~mherrma/v0/node5.html
  - http://ei.cs.vt.edu/~history/Perceptrons.Estebon.html
  - http://en.wikipedia.org/wiki/Perceptron
- Delta Rule
  - http://uhaweb.hartford.edu/compsci/neural-networks-
  - http://en.wikipedia.org/wiki/Delta_rule