Basic Movements Chasing and Evading

AI for Game Developers – Kapitel 2

Vortag von Alexander Skrinjar

Inhalt:

1) Aufgaben eines KI-Systems
2) Predator/Prey behaviour
    1) Notwendigkeit
    2) Mögliche Verhaltensweisen
    3) Ablauf und Ziele
3) Anwendung von Predator/Prey in Spielen
4) Exkurs: Grundlagen tile-basierte Welten / kontinuierliche Welten
5) Anwendung von Line-of-Sight Algorithmen
    1) LoS in tile-basierten Welten
    2) LoS in kontinuierlichen Welten
    3) Abfangen / Ausweichen
6) Anwendung von LoS Algorithmen in Spielen
7) Ausblick
8) Demos/Anwendungen/Videos
9) Fragen

Aufgaben einer künstlichen Intelligenz:

- simulieren von intelligentem Verhalten bzw. intelligentes Verhalten

- simulieren von instinktivem Verhaltens bzw. instinktives Verhalten

Grundlegende, angeborene Triebe:

Jagd- und Fluchttrieb

Verhaltensweisen:

suchen / verfolgen
fliehen / ausweichen

Notwendigkeit von „Predator/Prey Behaviour“:

- Allgegenwärtiges Prinzip
- In vielen „Kinderspielen“ aufgenommen
- Im Naturreich wichtigster Trieb

Notwendigkeit der Umsetzung in KI-Systemen

grundlegender Bestandteil fast jedes Computerspiels

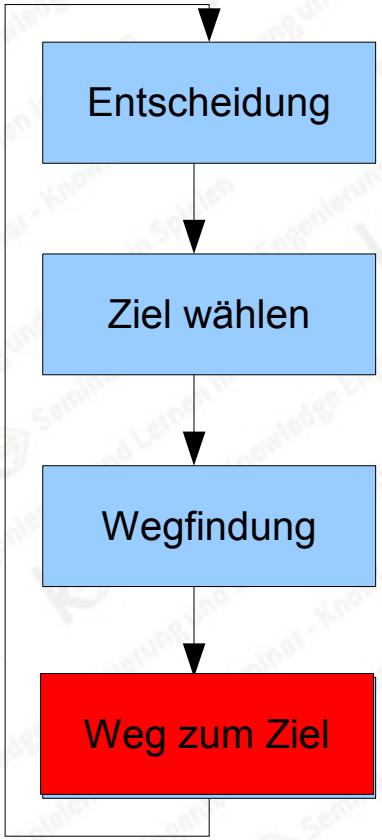
Spielziele ohne Predator/Prey behaviour:

Tetris, Karten/Brettspiele, WiSi, Adventure

Was haben diese Spiele gemeinsam?

In der Regel keine Simulation einer KI-Systems
(im Sinne eines emotionalen Lebewesens), sondern mathematischen Modelle oder Regelmodelle.

Aufbau des Predator/Prey Modells:



KI – Mögliche Verhaltensweisen:

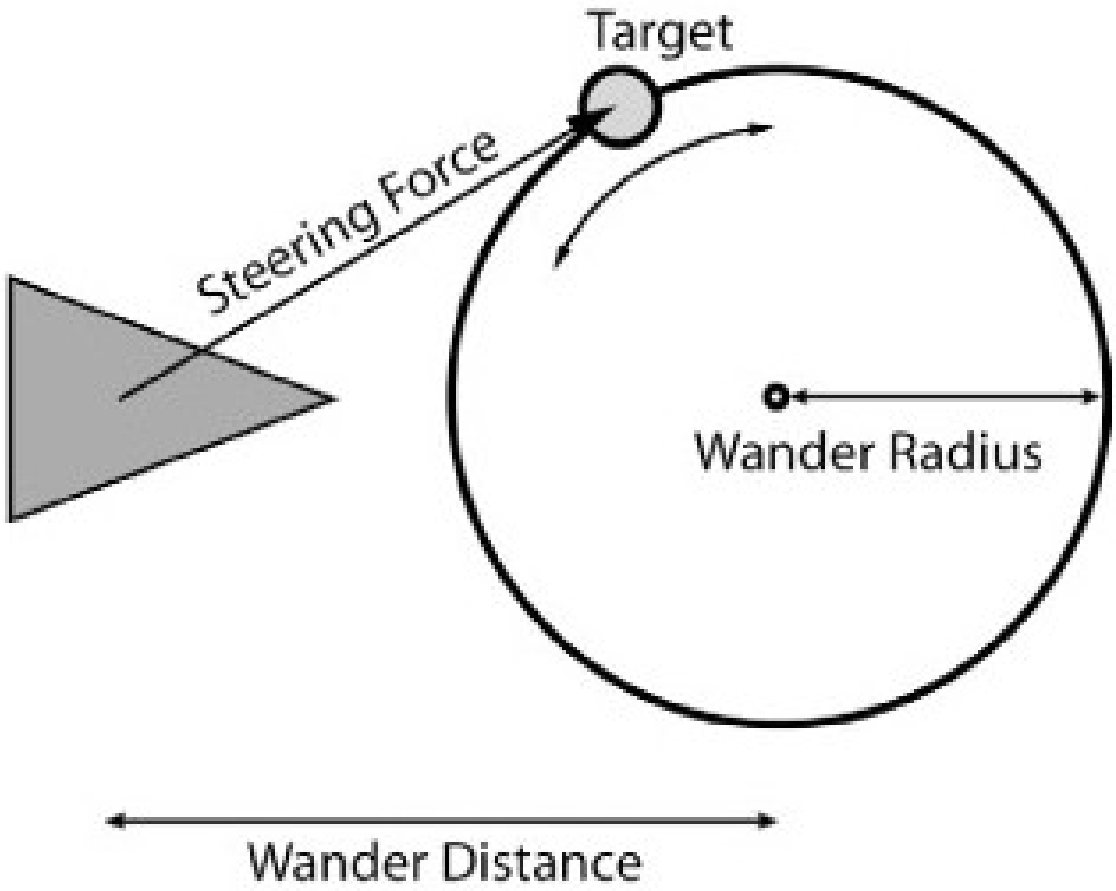
- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide

KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide

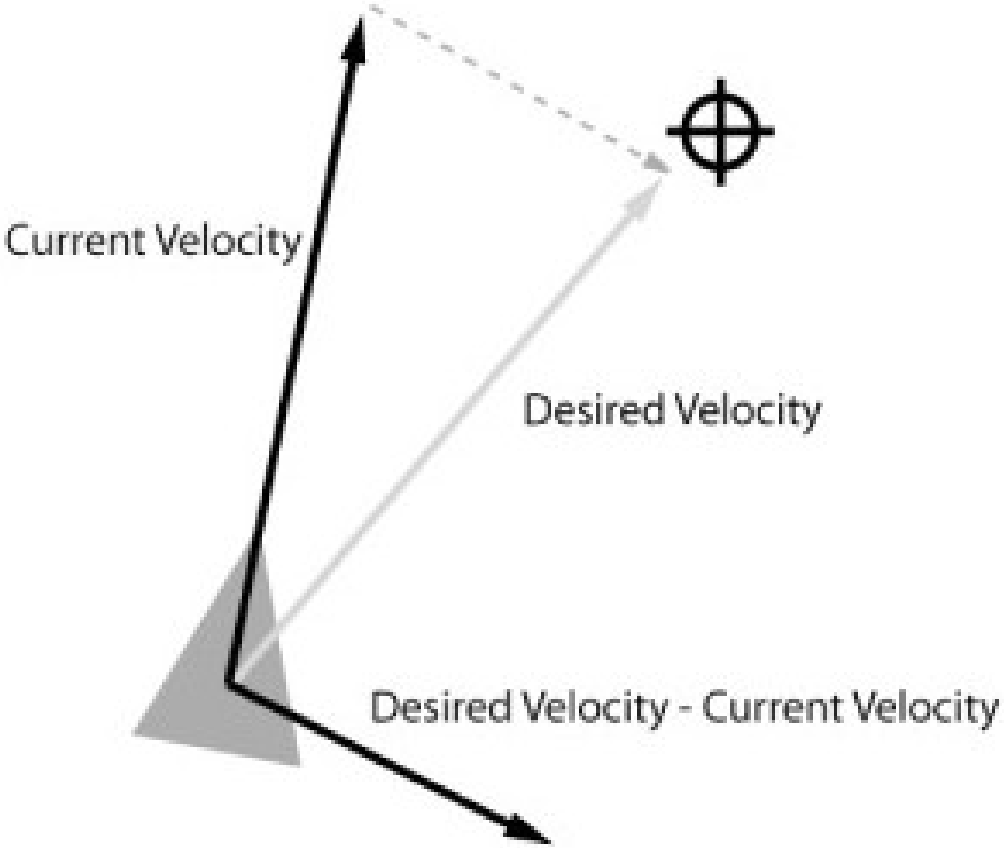
KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide



KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide

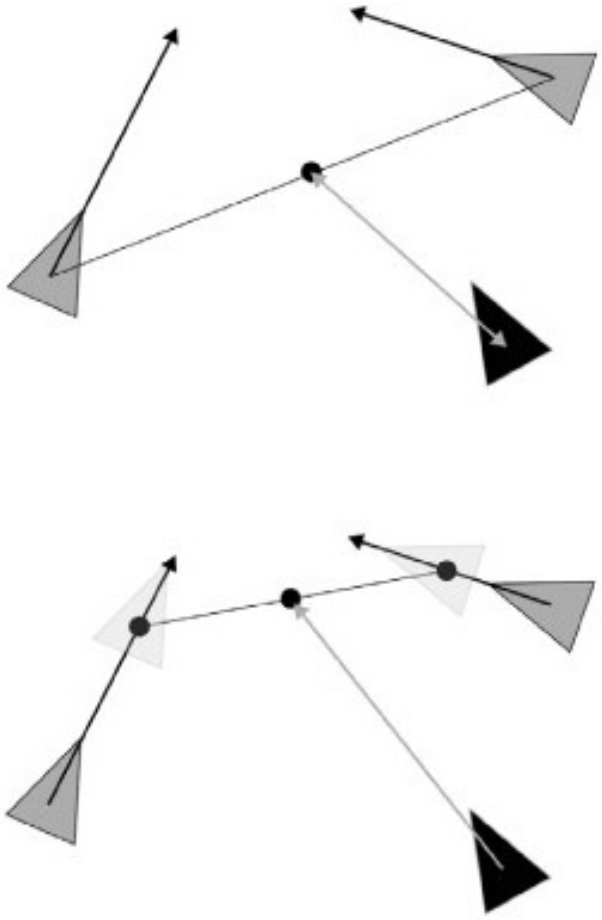


KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide

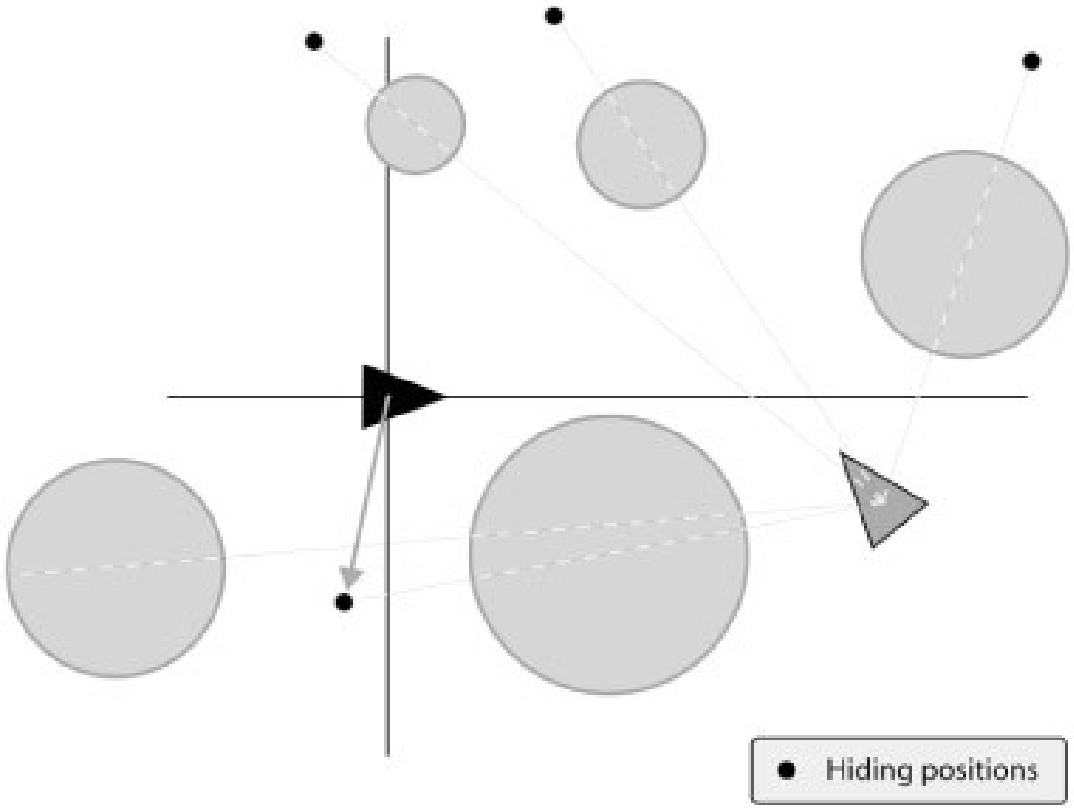
KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide



KI – Mögliche Verhaltensweisen:

- Idle
- Wandering
- Seek
- Flee
- Pursuit
- Evasion
- Interposing
- Hide



Ziele des Jägers:

Distanz zur Beute minimieren

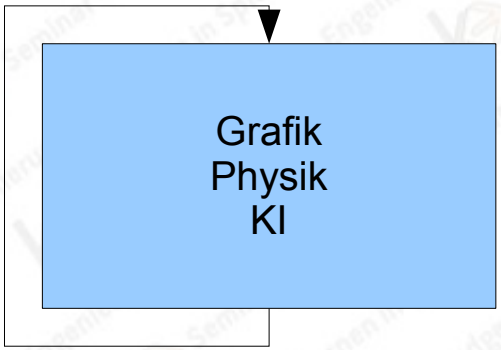
Seek / Intercept

Ziele der Beute:

Distanz zum Jäger maximieren

Flee / Evade

Gameloop

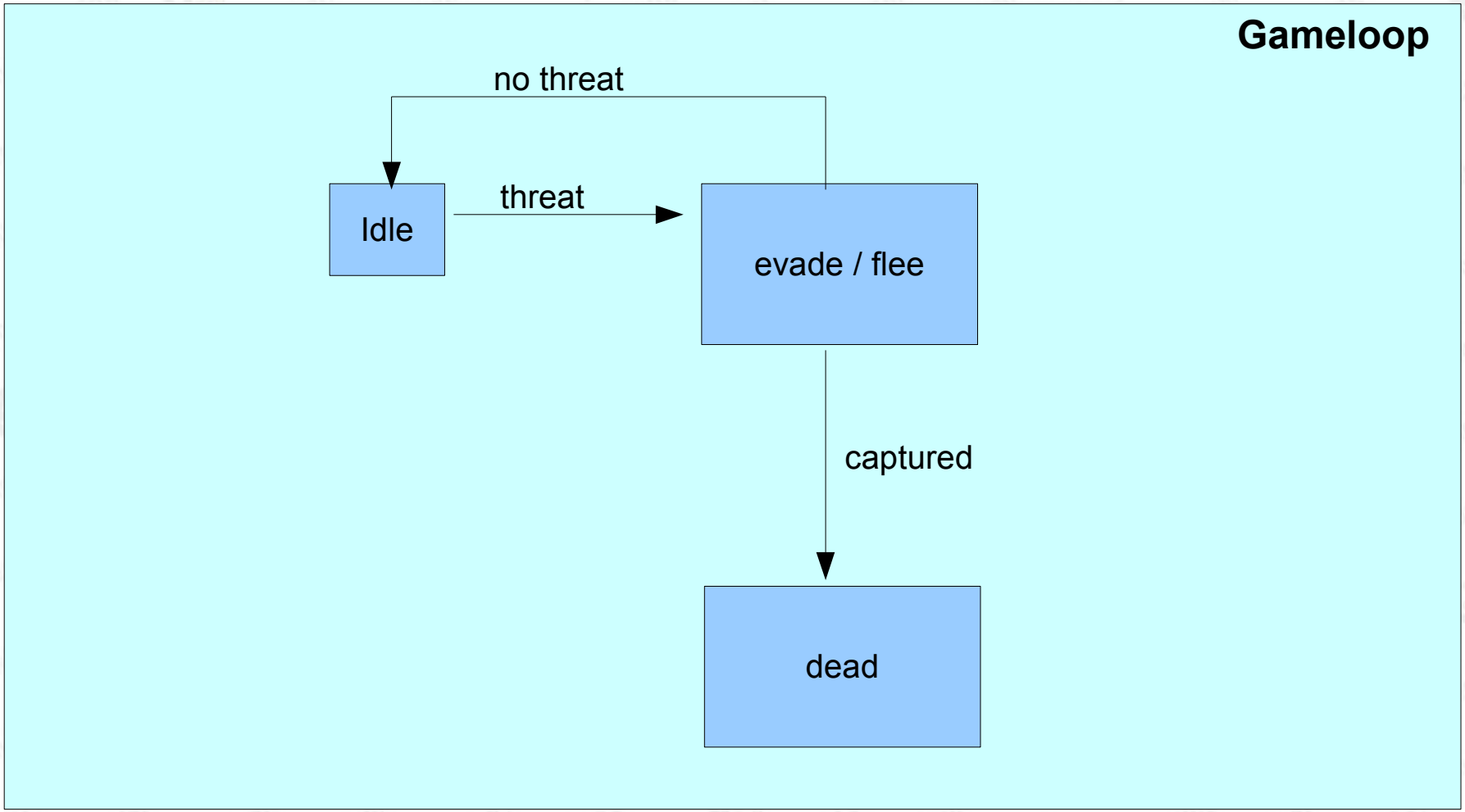


Orts-Parameter anpassen

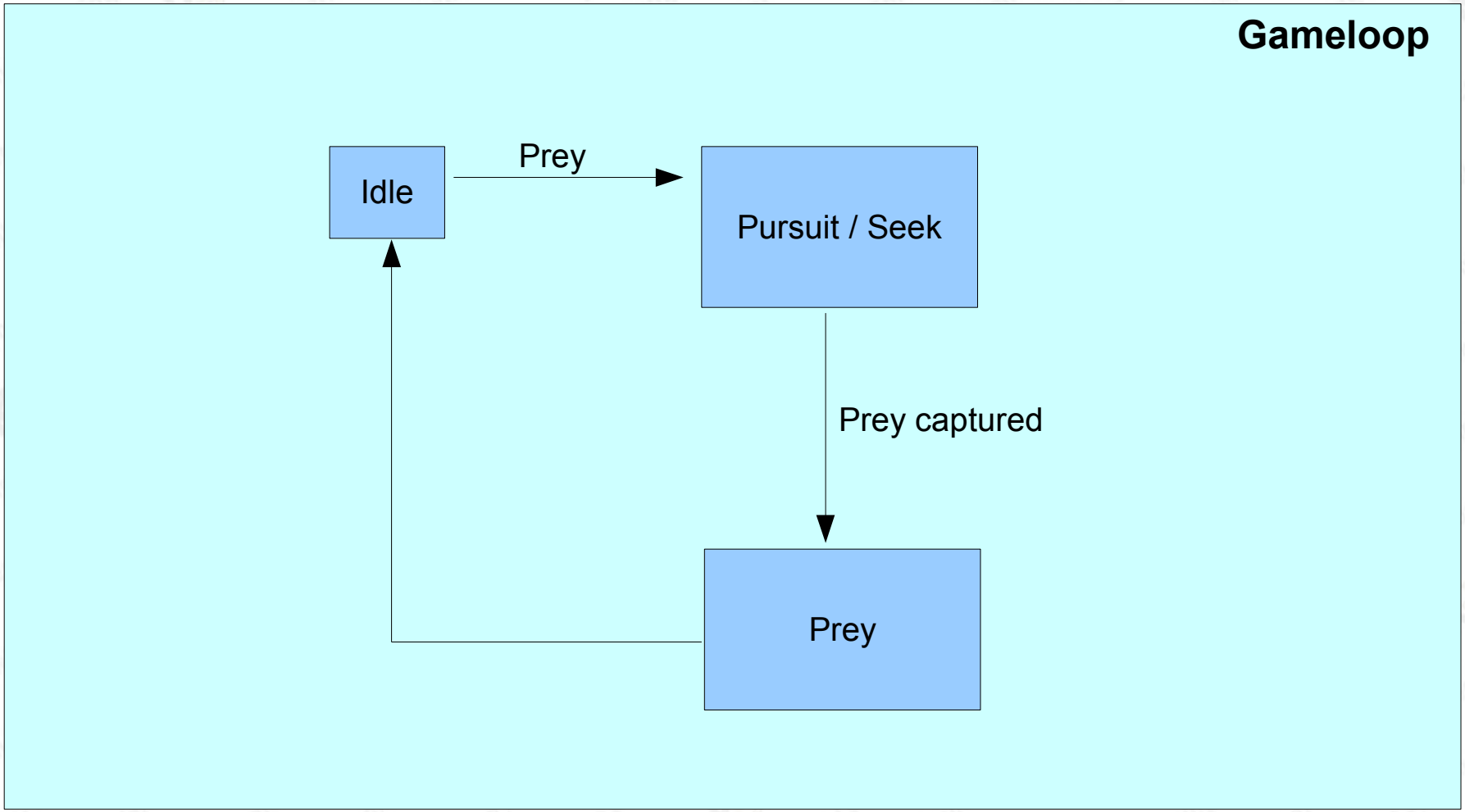
Besser:

- physikalische Parameter
- Umgebung
- Hindernisse
- „potential functions“

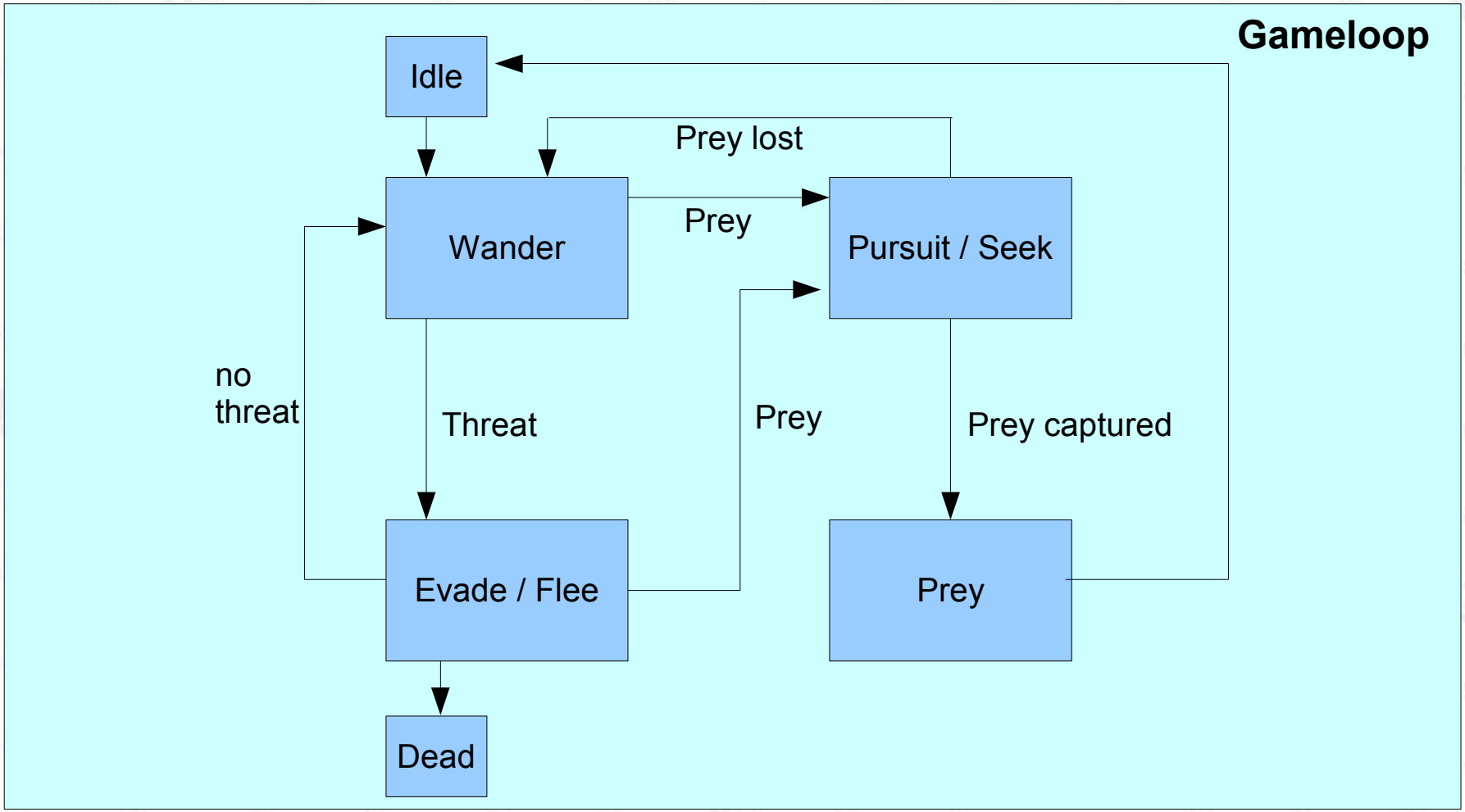
prey behaviour



predator behaviour



predator behaviour – complex version



Erweiterungen:

- Arrive/Depart
- Abbruchbedingungen
- Koordinaten in Erfahrung bringen
- Synchronisierung in Gruppen
- Scattering/Separation
- Bedrohung neutralisieren
- Alignment
- Cohesion
- Smoothing

Anwendung in Spielen:

- Sportsimulation: Autorennen

Anwendung in Spielen:

- Real-Time Strategy

Anwendung in Spielen:

- 1st Person Shooter

Anwendung in Spielen:

- Role-Play Games

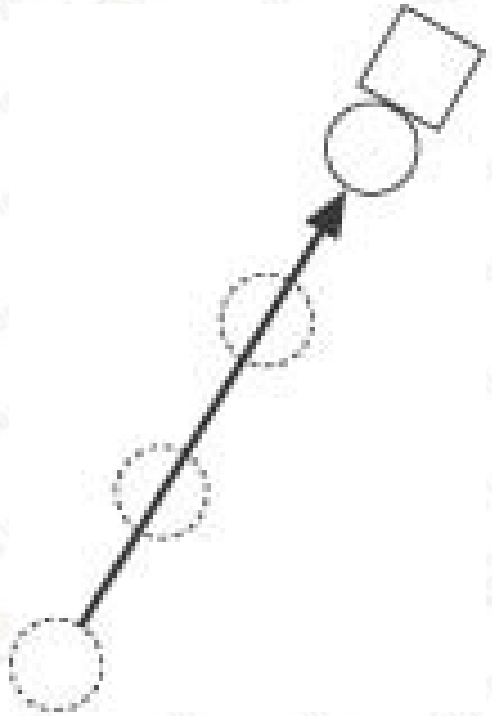
Line-of-sight:

Annahmen:

- Sichtkontakt zwischen Predator/Prey
- Keine Hindernisse zwischen Ihnen
- Umgebung nimmt keinen Einfluß

Aufgabe:

- Direkten Weg zum Ziel verfolgen

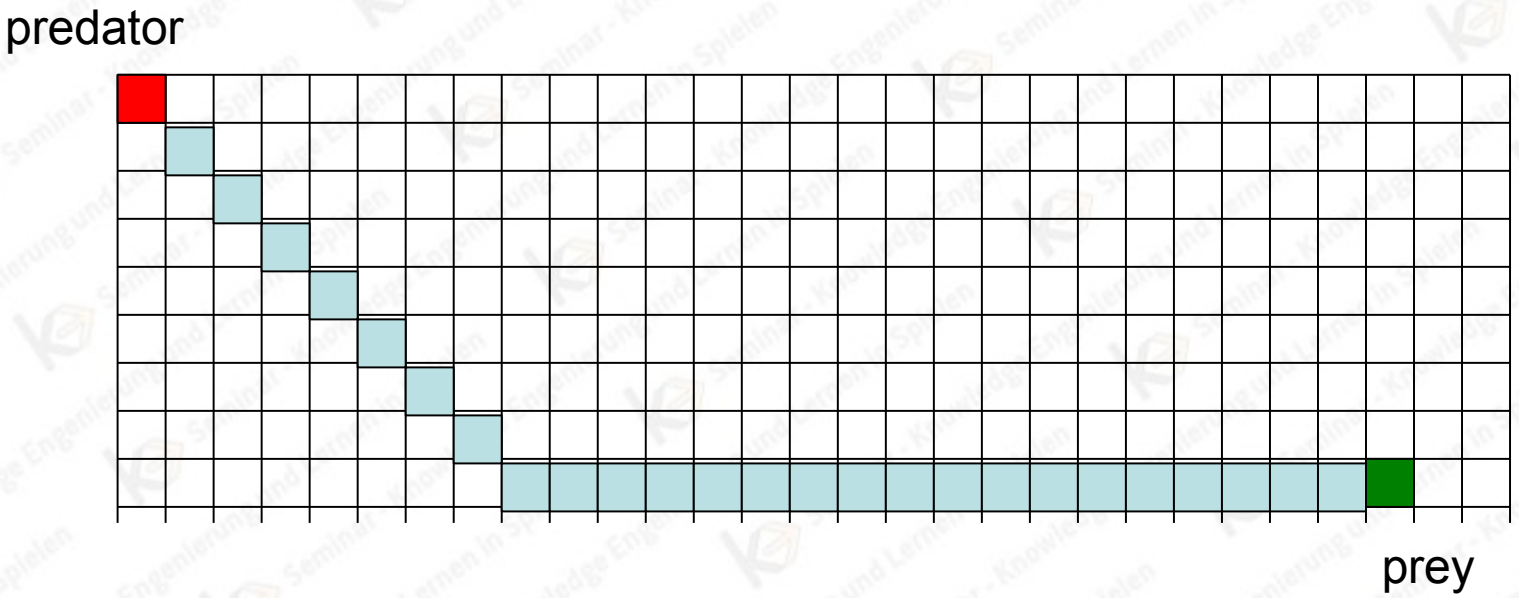


Grundlagen:

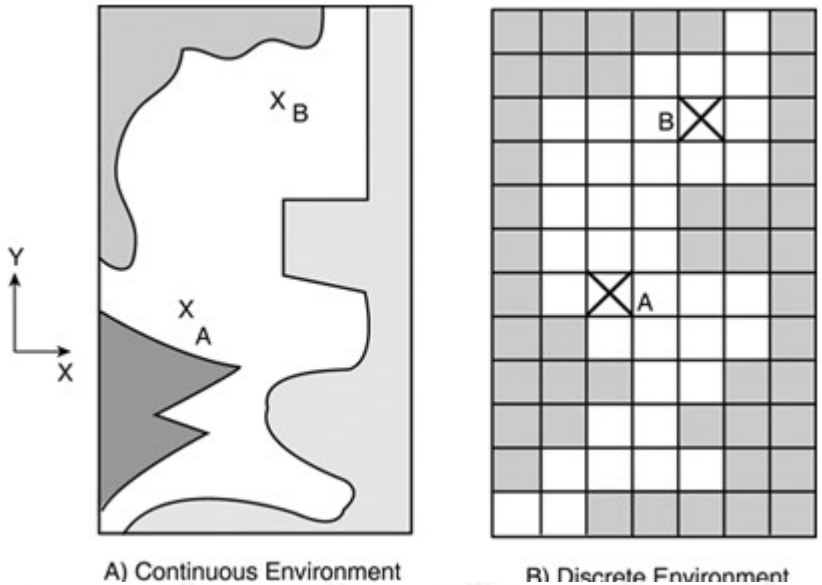
wenn jäger.x < beute.x => jäger.x++
wenn jäger.x > beute.x => jäger.x--

wenn jäger.y < beute.y => jäger.y++
wenn jäger.y > beute.y => jäger.y--

Wie wird ++ bzw. -- umgesetzt?



Exkurs: tile-basierende / kontinuierliche Welt



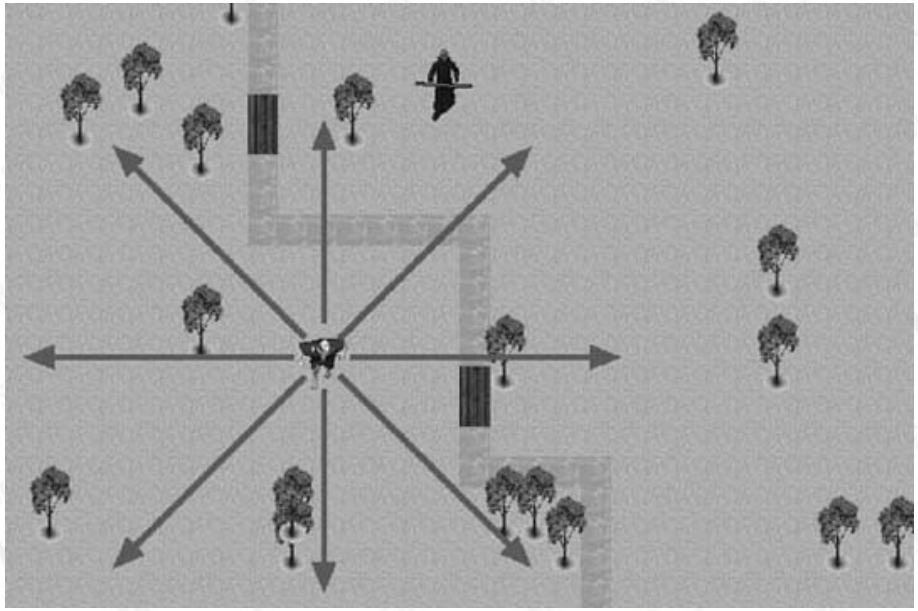
Kontinuierliche Welten:

- „Analoge“ Welt
- FP-Berechnungen
- Datenstruktur: Vektoren / Matrizen
- Hoher Freiheitsgrad (Pixelgenau)

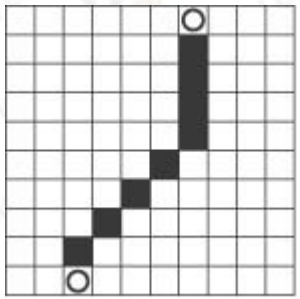
Tile-basierende Welten:

- Diskrete Welt, unterteilt in Kacheln
- Basierend auf Integerberechnung (effizient)
- Datenstruktur: Grid (z.B. 2-D Array)
- Beschränkt auf Kacheln

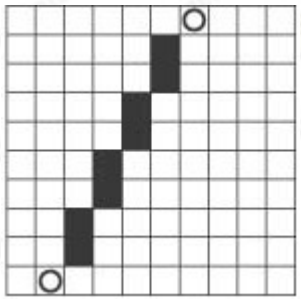
Line-of-sight – tile-based Games:



- Kachelstuktur gibt Richtungen vor
- Standard: Quadrat => 8 Orientierungen
  (reicht i.d.R. für Darstellung)

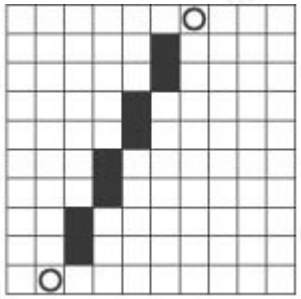


- Standard Algorithmus:
  - Kürzester Weg
  - unnatürlich

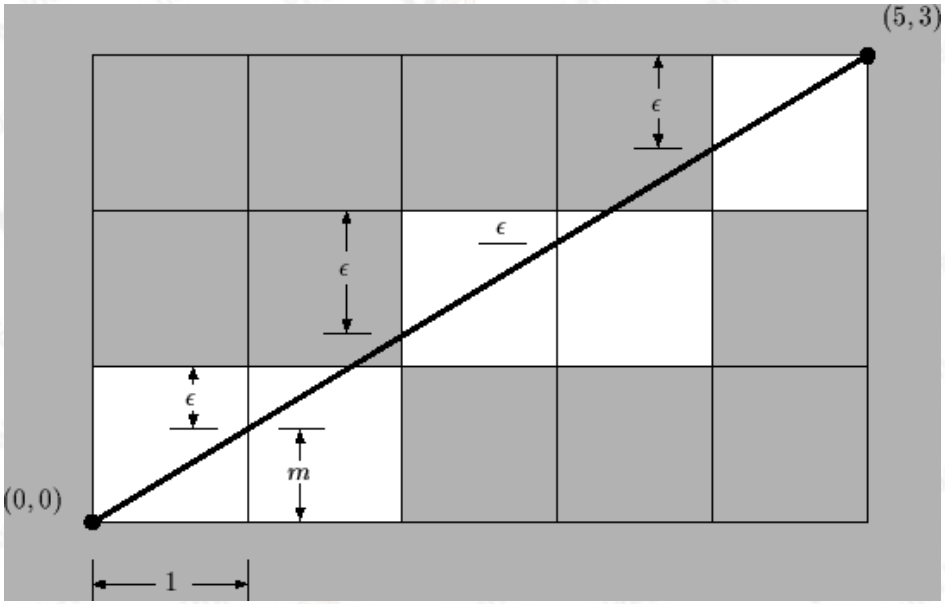


- Scan-Line Algorithmus:
  - Kürzester Weg
  - natürlicher

Line-of-sight – tile-based Games:

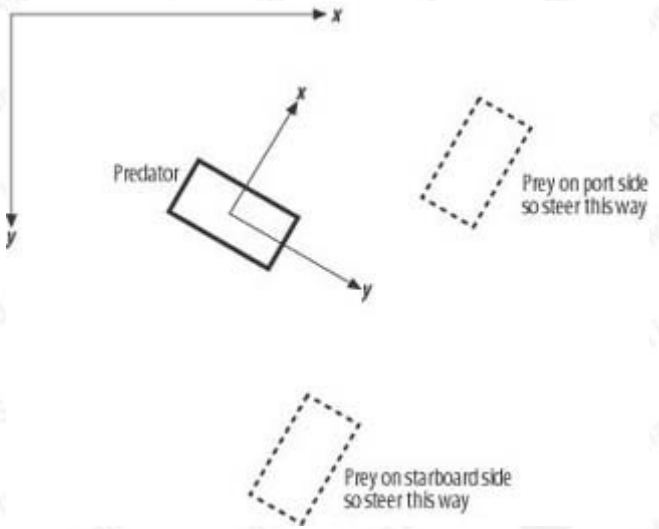


- Scan-Line Algorithmus (Bresenham):
  - Kürzester Weg
  - Natürliche Bewegung
  - Direkte Verbindung

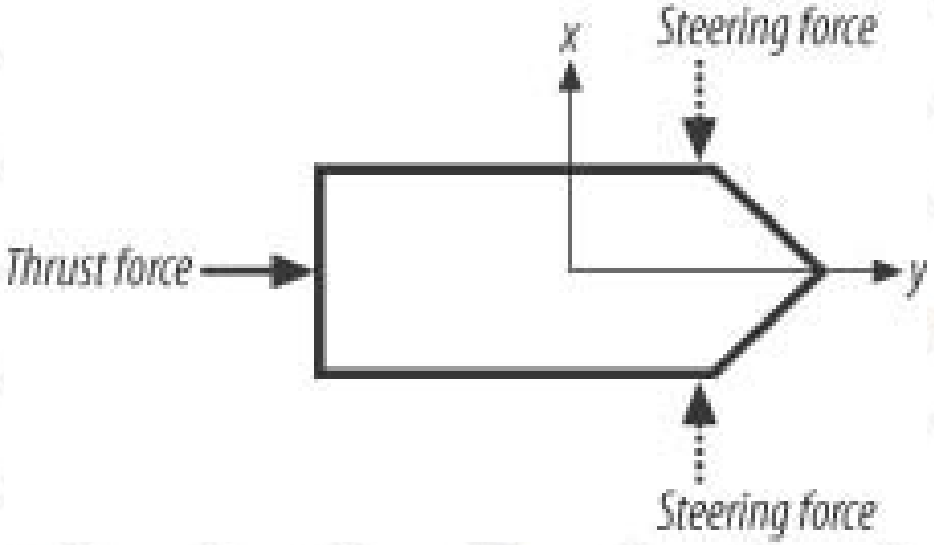


Applet

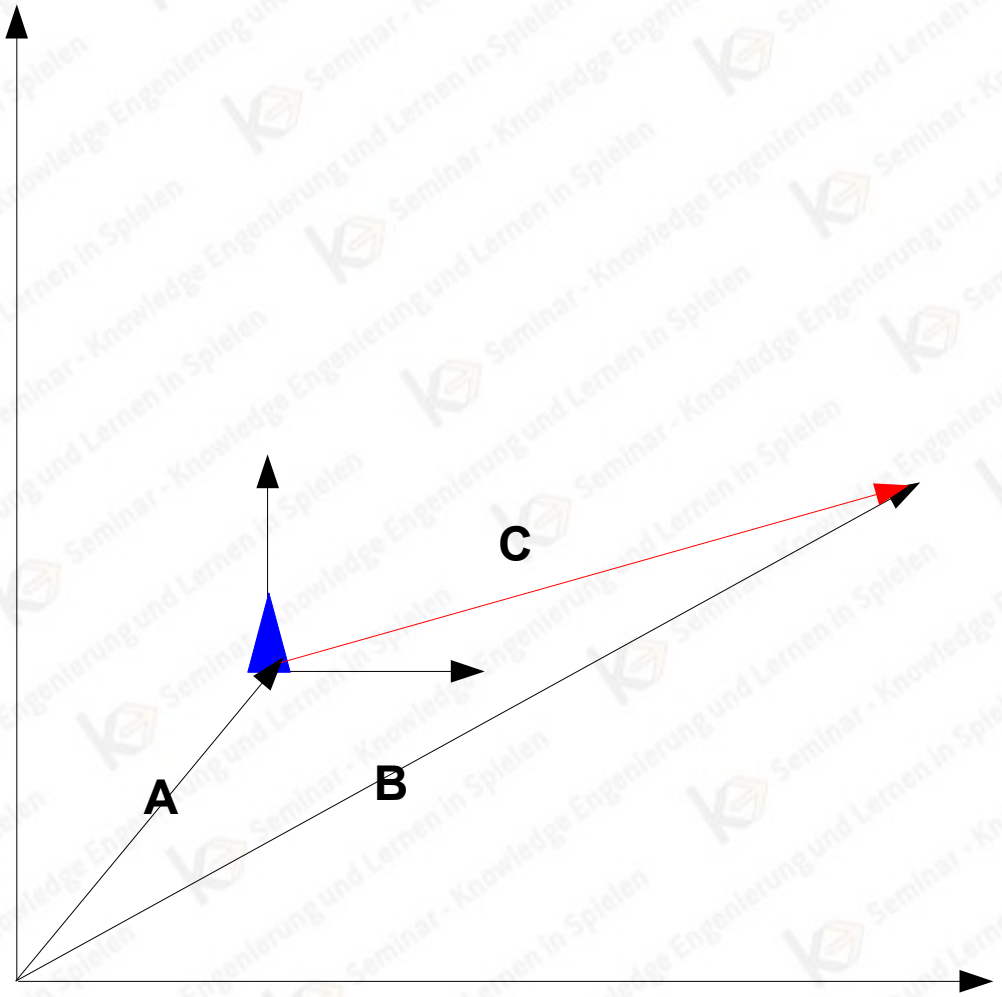
Line-of-sight – kontinuierliche Welten:



- Vektoren geben Orientierung an
- Vektor zum Ziel berechnen
- Bewegung entlang diesen Vektors



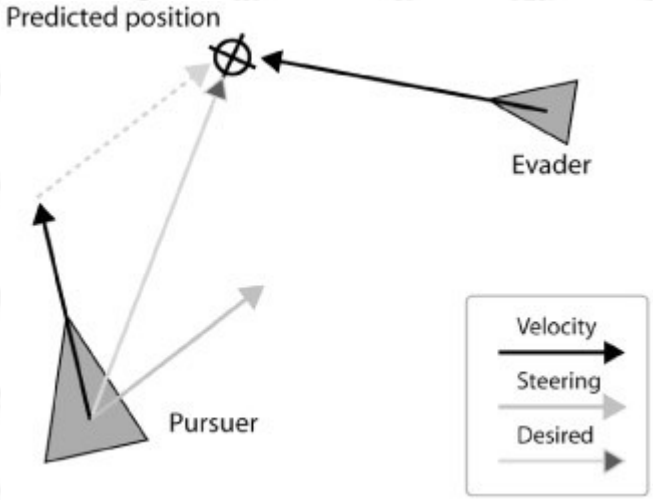
Line-of-sight – kontinuierliche Welten:



C = B - A

$$\cos\lambda = \frac{\vec{a} * \vec{b}}{\left|\vec{a}\right| * \left|\vec{b}\right|}$$

Intercepting:



- Vektoren geben Orientierung an
- Vektor zum Abfangen berechnen
  - Zeit bis Abfangen berechnen
  - Vektoren zur Zeit berechnen
- Bewegung entlang diesen Vektors

$$V_r = V_{prey} - V_{predator}$$

$$S_r = S_{prey} - S_{predator}$$

$$t_c = |S_r| / |V_r|$$

$$S_r = S_{prey} + (V_{prey} \times t_c)$$

Anwendung in Spielen:

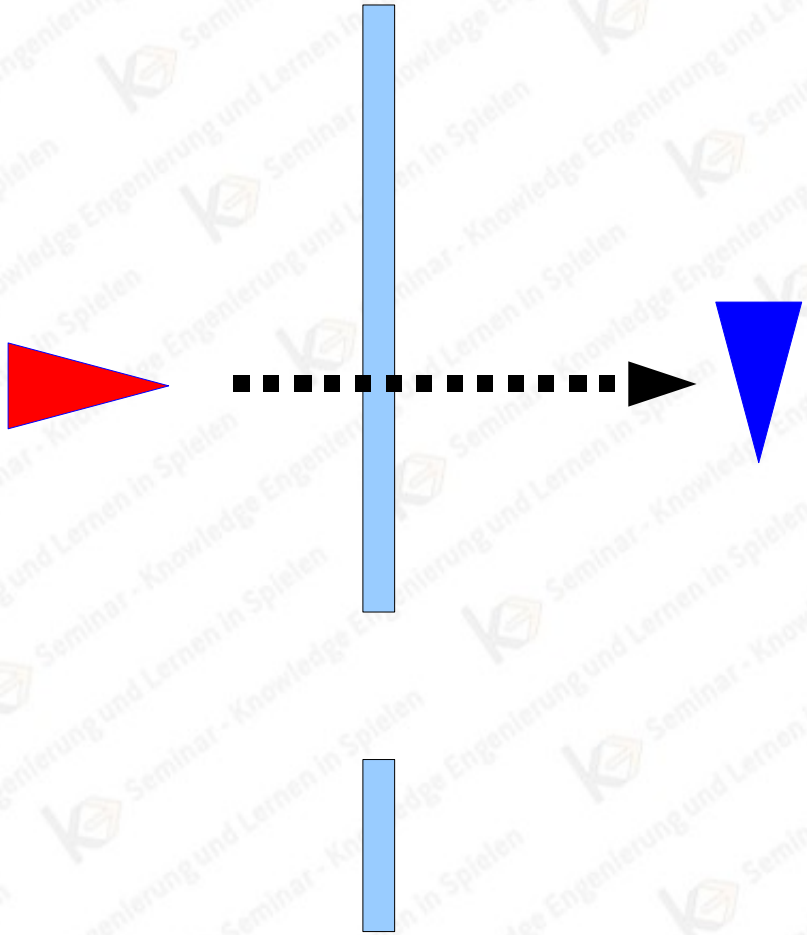
Als Orientierungshilfe:

- Mini-Maps
- Kompass

LoS keine stand-alone Lösung, daher:

- Grundkomponente von vielen fortgeschritten Lösungsansätzen

Mögliche Probleme:



Fehlverhalten:

- Erweiterte Konzepte erforderlich
  - Pathfinding
  - Obstacle Avoidance
- Nicht ausreichen zur Simulation von intelligentem Verhalten

Ausblick:

- Entscheidung treffen (FSM, neuronale Netze)
- Pathfinding (A*, Waypoint Systeme, Obstacle Avoidance)
- Gruppendynamik (Flocking)
- Fortgeschrittene Konzepte (Pattern Movement)
- etc.

Anwendungen/Demos

Noch Fragen?