Modellierung von Aktionen

Wie werden die erlaubten Aktionen eines Spiels umgesetzt?

- Direkte Implementierung in der Host-Sprache
  - Vorteil: hohe Effizienz
  - Nachteile:
    - Doppelte Implementierung der gleichen Effekte
    - Redundanter Code
    - Inkonsistenzen sind schwer überprüfbar
- Verwendung von Packages und Subsystemen die bestimmte teile der Aktionsverarbeitung kapseln:
  - Physics Engine (Kollisionstests, Beschleunigung, Abprallen,…)
  - Spatial Management Module (nächste Nachbarn, Sichtbereiche,
  - AI Engine (Wegewahl, Schwarmverhalten, …)

Modellierung von Aktionen

- Scripting Engine
  - Stellt eigene Programmiersprache zur Verfügung
  - Befehle können direkten Zugriff auf Game State verhindern (Unterstützt Konsistenzerhalt)
  - Entitäten und Ihr Verhalten werden einheitlich modelliert
  - Nachteil: nicht alle Design Optionen sind ohne Änderungen der Skriptsprache umsetzbar
  - Beispiel: LUA (http://http://lua-users.org/wiki/ClassesAndMethodsExample)

```
require("INC_Class.lua")
--=========================
 cAnimal=setclass("Animal")

function cAnimal.
       methods:init(action, cutename)
       self.superaction = action
       self.supercutename = cutename
end
```

```
--=========================
cTiger=setclass("Tiger", cAnimal)

function cTiger.methods:init(cutename)
        self:init
       super("HUNT (Tiger)", "Zoo Animal (Tiger)")
       self.action = "ROAR FOR ME!!"
       self.cutename = cutename
 end
```

Physics Engines

- Umsetzung von Festkörperphysik/klassische Mechanik
- Game Entität muss alle benötigten Parameter abbilden
  - Räumliche Ausdehnung (Polygonmesh, Vereinfachungen: Zylinder, MUR)
  - Geschwindigkeitsvektor
  - Masse
  - …
- Umsetzung meist über Differentialgleichungssysteme
- Für realistische Effekte hohe Tick-Raten und detailierte Modellierung
- hoher Berechnungsaufwand macht effiziente numerische Näherungsalgorithmen notwendig

Physics Engines und MMO-Server

- Viele der Ergebnisse einer klassischen Physics Engine werden nur zur realistischeren Darstellung benötigt

  z.B. Partikelfilter, Rag-Doll Animation,..
- Gekoppelte Berechnung mit Darstellungs-Layer ist häufig sinnvoll, da Ergebnisse für Änderung der Darstellung und des Game States notwendig sind (Bewegung)
- Hohe Tick-Raten

⇒ Client-seitige Verwendung

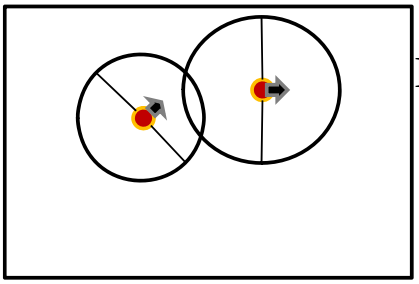
⇒ Auf Serverseite meist zu hoher Aufwand

⇒ Für Umsetzung des Designs reichen häufig grobe Vereinfachungen

⇒ Verwendung von Physics Engine kann auf zur Parameterbestimmung von direkten Implementierungen verwendet werden

Spatial Management in Game Servern

- Großteil der Spiele besitzt eine räumliche Komponente z.B. 2D/3D Karten, Spielwelt…
- Aktionsverarbeitung, NPC-Steuerung und Netzwerk-Layer beinhalten räumliche Anfragen:
  - Welche anderen Game Entitäten kann ich beeinflussen, können mich beeinflussen oder kann ich überhaupt sehen? (AoI = Area of Interest)
  - Unterstützung von Kollisionsanfragen (vgl. Physics Engine) und Bereichsschnittmengen
  - Welcher Spieler steht mir am nächsten?
  - Betritt ein Spieler die Angriffsumgebung (Aggro-Range) eines NPCs?

Attack-Range  Hit-Box/Range

Spatial Management in Game Servern

- Bei kleinen Spielwelten wenige räumliche Objekte (Game State als Liste organisiert)
- Anfragebearbeitung über sequentielle Suche
- Bei widerholter Anfragebearbeitung und großer Anzahl beweglicher Entitäten entsteht erheblicher Suchaufwand

  Beispeil: 1000 Game Entities in einer Zone, 24 Ticks/s

  naive AoI Berechnung: 24 mio. Distanzvergleiche pro Sekunde

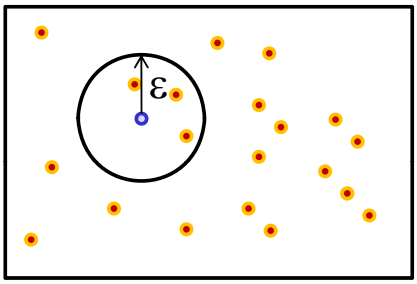
- **Folge**: bei wachsenden Game States fällt sehr viel Berechnungsaufwand für räumliche Anfragen an.

räumliche Anfragen (1)

Räumliche Anfragen(hier mit euklidischer Distanz im IR^2)

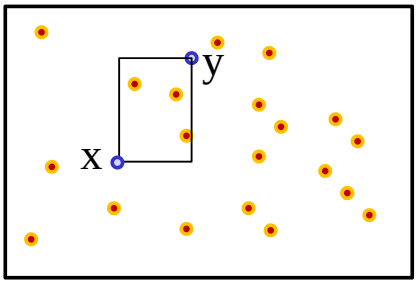
- Range-Query

$$RQ(q,\varepsilon) = \{v \in GS \mid \sqrt{(q_1 - v_1)^2 + (q_2 - v_2)^2} \leq \varepsilon\}$$



- Box-Query

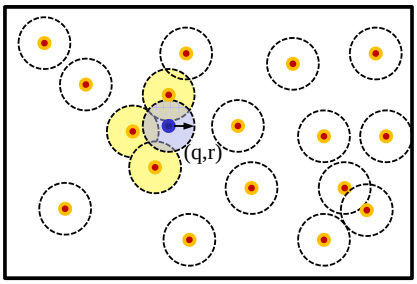
$$BQ(x,y) = \{v \in GS \mid x_1 \leq v_1 \leq y_1 \wedge x_2 \leq v_2 \leq y_2\}$$



räumliche Anfragen (2)

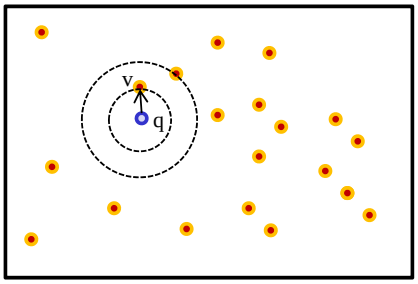
- Intersection Query

$$SIQ(q,r) = \{(v,s) \in GS \times IR \mid \sqrt{(q_1 - v_1)^2 + (q_2 - v_2)^2} \leq r + s\}$$



- NN-Query

$$NN(q) = \{ v \in GS \mid \forall x \in GS : \sqrt{(q_1 - v_1)^2 + (q_2 - v_2)^2} \leq \sqrt{(q_1 - x_1)^2 + (q_2 - x_2)^2} \}$$



Effizienzsteigerung für räumlich Anfragen

- Methoden zur Reduktion der betrachteten Objekte (Pruning)
  - Aufteilung der Spielwelt (Zoning, Instanziierung, Sharding..)
  - Index-Strukturen (BSP-Tree, KD-Tree, R-Tree, Ball-Tree)
- Reduktion der Aufrufe von räumlichen Anfragen
  - Reduktion der Anfrage-Ticks
  - Spatial-Publish-Subscribe
- Effiziente Anfragealgorithmen
  - Nearest-Neighbor Anfragen
  - ε-Range Join (gleichzeitiges Bestimmen aller Areas of Interest)

Sharding und Instanziierung

- Kopieren einer räumlichen Region für eine bestimmte Gruppe
- Dabei existiert die gleiche Region der Karte beliebig häufig
- Instanzen und Shards wurden primär zur Umsetzung des Game Designs geschaffen.
  (Begrenzung der Spieler zum Lösen einer Aufgabe)
- Aber: Je mehr Spieler in einer Instanz, desto weniger Performanzprobleme in der offenen Welt

Problem:

- Löst das Problem nicht (keine zusammenhängende MMO Welt)
- Speicherung des lokalen Game State auch wenn kein Spieler mehr in der Instanz ist.

=> Management der Instanzen kann Aufwand verursachen
(Worst Case: 1000 parallele Game States für 1000 Spieler)

Zoning

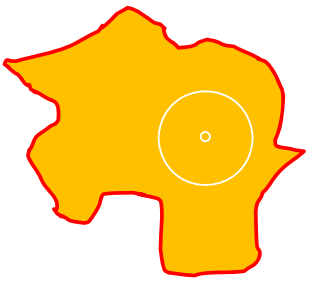
- Aufteilen der offenen Spielwelt in mehrere feste Teillandschaften
- Anfragen müssen nur Objekte in der aktuellen Zone berücksichtigen
- Teilt nicht nur den Raum sondern auch den Game State auf
- Erleichtert Verteilung der Spielwelt auf mehrere Rechner

Probleme:

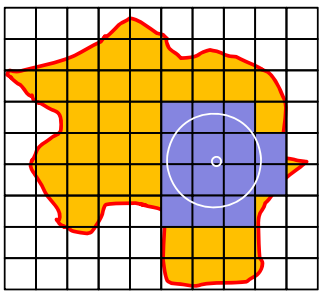
- Bei Randbereichen müssen evtl. Objekte in mehreren Zonen berücksichtigt werden
- Ungleichmäßige Verteilung der Spieler

Micro-Zoning

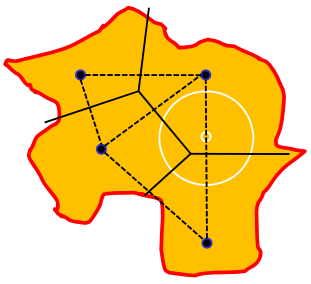
- Spielwelt wird in viele kleine Zonen (Micro-Zonen) aufgeteilt
- Game Entities werden in ihrer aktuellen Micro-Zone verwaltet
- Nur Zonen, die die AoI schneiden sind relevant
- Sequentielle Suchen innerhalb der Region
- Zonen können durch verschiedene Methoden erzeugt werden (Grids, Voronoizellen, ..)



Zoning



Micro-Zoning
(Grid-basiert)



Micro-Zoning
(Voronoi-basiert)

Spatial Publish-Subscribe

- Micro-Zoning in Kombination mit Subscriber-System
- Game Entities werden in ihre aktuelle Micro-Zone eingetragen (publish)
- Game Entities abonnieren die Informationen aller Micro-Zonen, die ihre AoI schneiden (subscribe)
- Liste aller Game Entities in AoI über Vereinigung der Einträge der abonnierten Micro-Zonen

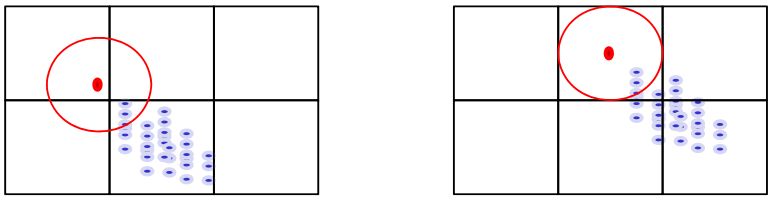
Vorteile:

- Effiziente Bestimmung nahe gelegener Objekte
- bei Änderung können neue Informationen an Subscriber durchgereicht werden (keine regelmäßige Anfrage notwendig)

Micro Zoning und Spatial Publish-Subscribe

Nachteile:

- Auch Micro-Zonen können überfüllt sein
  => je kleiner die Fläche desto weniger wahrscheinlich
- Bei zu kleinen Zonen steigt der Overhead für einen Zonenwechsel
  => je kleiner die Zonen je häufiger der Wechsel
- Lage des Zonengrenzen kann zu starken Schwankungen bei den betrachteten Objekten führen.
- Bei sehr hohen Änderungsraten steigt der Overhead von Zonen-Wechsel extrem an.
  => sehr viele Subscribe- und Unsubscribe-Vorgänge bremsen das System aus



Klassische Indexstrukturen

- Verwaltung von räumlichen Objekten kann auch über räumliche Suchbäume erfolgen
- Suchbäume passen ihre Seitenregionen (Zonen) der Datenverteilung an
  - ⇒ Garantie über maximale Füllung einer Seitenregion/Zone
  - ⇒ Bessere Suchperformanz durch Reduktion der betrachteten Objekte
  - ⇒ Es entsteht Aufwand bei der Anpassung des Suchbaumes
- Anpassung durch rekursive Aufteilung des Raumes (Quad-Tree, BSP-Trees)
- Anpassung durch Aufteilen der Daten auf minimale umgebende Seitenregionen