Skript zur Vorlesung
Managing and Mining Multiplayer Online Games
im Sommersemester 2012

Kapitel 2: Der Game Core

http://www.dbs.informatik.uni-muenchen.de/cms/VO_Managing_Massive_Multiplayer_Online_Games

Kapitelübersicht

- Generelle Modellierung eins Spielzustandes (Game State)
- Zeit-Modellierung (Zug und Tick-System)
- Aktionsverarbeitung
- Interaktion mit anderen Komponenten der Game Engine
- Räumliche Verwaltung und Aufteilung des Game States

Interne Darstellung von Spielen

Benutzersicht

ID	Type	PosX	PosY	Health	…
412	Knight	1023	2142	98	…
232	Soldier	1139	2035	20	…
245	Cleric	1200	2100	40	…
…					

Game State

Gutes Design: Strikte Trennung von Daten und Darstellung (Model-View-Controller Pattern)

- MMO-Server: Verwalten des Game State /keine Darstellung notwendig
- MMO-Client: Teile des Game States aber I/O und Darstellung Komponente
- Begünstigt die Implementierung unterschiedlicher Clients für dasselbe Spiel. (unterschiedliche Graphikqualitäten)

Game State

Gesamtheit aller Daten die den Spielzustand repräsentieren

- Modellierung mit ER-Modell oder UML möglich() (Objekte, Attribute, Beziehungen..)
- Modell aller veränderlichen Informationen
- Liste aller Game Entities
- Attribute der Game Entities
- Informationen über das gesamte Spiel

Informationen, die nicht im Game State stehen müssen:

- Statische Informationen
- Umgebungsmodelle/Karten
- Standard-Attribute von Einheiten

:

Game Entitäten

Entsprechen den Objekten

Beispiele für Game Entities:

- Einheiten in einem RTS-Spiel
- Felder in einem Brettspiel
- Charaktere in einem RPG
- Gegenstände
- Umgebungsobjekte
  (Truhen, Türen,..)

Attribute und Beziehungen

Regelrelevante Eigenschaften einer Game Entity

Entspricht Attributen und Beziehungen

Beispiele:

- Aktuelle HP (max. HP nur wenn veränderlich)
- Ausbaustufe von Einheiten in einem RTS
- Umgebungsobjekte: offene oder geschlossene Türen
- Beziehungen zwischen Objekten:
  - Charakter A hat Item X im Inventar (1:n)
  - A ist mit B in einer Gruppe (n:m)
  - A befindet sich im Kampf mit C (n:m)
  - A hält Waffe W in der rechten Hand (1:1)

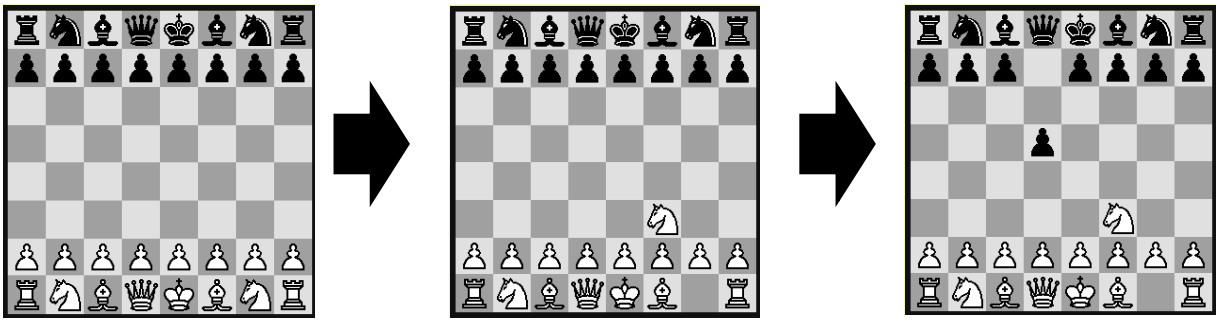
Informationen über das gesamte Spiel

Alle Informationen über den Spielzustand, die nicht über Entitäten erfassbar sind

- Ingame Tageszeiten
- Karten auf der gespielt wird
- Sichtbereich der Spieler in einem RTS Game
  (Falls für die Spieler kein abstraktes Entity erzeugt wird)
- Servertyp in einem MMORPG (PVP/PVE/RP)
- …

Achtung: Informationen können als Attribute des Game States oder auch als Entitäten modelliert werden.

Beispiel: Schach

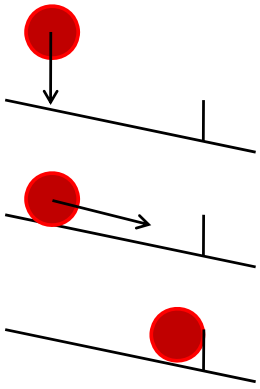


- Information über das Spiel:
  - Spieler und Zuordnung der Seite(schwarz oder weiß)
  - Spielmodus: mit „Schachuhr“ oder ohne
- Game State:
  - Positionen aller Figuren/ Belegung aller Spielfelder
    (Entitäten sind hier entweder Figuren oder Felder)
  - Spieler der gerade am Zug ist
  - verbleibende Zeit für beide Spiele (abhängig vom Spielmodus)

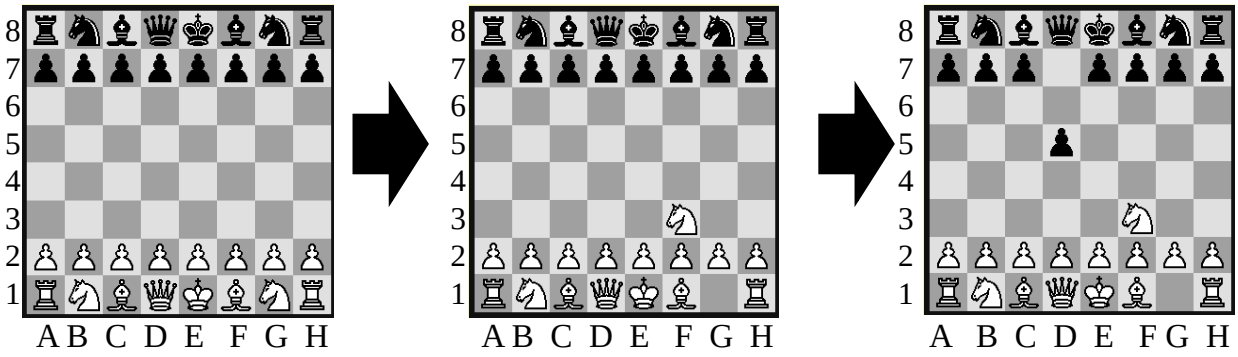
Aktionen

- Aktionen überführen einen gültigen Game State in einen anderen gültigen Game State
- Aktionen implementieren die Regeln des Systems
- Game Core organisiert die Entstehung durch:
  - Spieler (Benutzereingabe)
  - NPC-Steurung (AI Steuerung)
  - das Umgebungsmodell

    Beispiel:
    - Ball wird auf eine Schräge gelegt.
    - Umgebungsmodell berechnet mit Hilfe der Physics Engine das Ball rollt
    - Aktion die sowohl die Position als auch den Zustande des Ball(Beschleunigung) ändert wird ausgelöst



Beispiel: Schach



- Aktionen: Ändern Position der Figuren/Belegung der Felder
  - Pferd von G1 auf F3
  - Bauer von D7 auf D5
- Aktionen realisieren die Regeln:
  - schwarzer Bauer 2 Felder nach vorne falls er noch nicht bewegt wurde
  - Pferd darf 2 nach vorne und 1 nach links (unter anderem)

Zeitmodellierung der Aktionen

- Regelt den Ausführungszeitpunkt einer Aktion
- Verhältnis von Spielzeit (verarbeitete Aktionen/Zeiteinheit) zur Echtzeit (Wall-Clock Time)
- Synchronisation mit anderen Komponenten des Spiels:
  - Rendering (Graphik/Sound)
  - Abfrage von Benutzereingaben
  - Aufrufe der AI für NPCs
  - …
- Umgang mit Aktionen die noch nicht verarbeitet werden können
  - Löschen (2ter Zug in Folge von einem Spieler im Schach)
  - Verzögern (Ausführen der Aktion sobald sie erlaubt ist)

- Lösung hängt stark vom Spielprinzip ab

Aktionsverarbeitung bei Eventsteuerung

- Rundenbasierte Spiele: Art und Reihenfolge der Aktionen ist festgelegt und wird vom Game Core Verwaltet
- Game Core ruft Aktionserzeuger in fester Reihenfolge auf.
- Realisierung über Schleifen, Zustandsautomaten,..
- Keine Nebenläufigkeit möglich
- Beispiele:
  - Schach
  - Civilization
  - Siedler von Catan
  - Rundenbasierte RPGs

Nachteile:

- Anwesenheit der Spieler erforderlich
- Spielprinzip würde eventuell gleichzeitiges Ziehen mehrerer Spieler zulassen (weniger Wartezeit)

Aktionsverarbeitung bei Eventsteuerung

Realtime/Transaktionsystem

- Game verwendet keine feste Steuerung des Aktionserzeugung
- Spieler können Züge asynchron zueinander absetzen
- NPC/Umgebungsmodelle können in unabhängigen Threads agieren
- Nebenläufigkeit kann wie bei Transkationssystemen realisiert werden (Sperren, Rücksetzten, ..)
- Beispiel:
  - bestimmte Browser Games

Aktionsverarbeitung bei Eventsteuerung

Vorteile:

- Kann auf Standardlösungen Aussetzen (z.B. DBS)
- Vollständige Nebenläufigkeit:
  - wenig Wartezeit auf andere Spieler (Spiel kann Wartezeiten verlangen)
  - Verteilte Realisierung ist einfach

Nachteile:

- keine Synchronisation zwischen Spielzeit und Echtzeit
  => Spielzeit (Aktionen/Minute) kann stagnieren
- Keine Kontrolle über max. Handlungen pro Zeiteinheit
- Gleichzeitige Handlungen sind unmöglich (Serialisierbarkeit)

Realisierung der Zeitabhängigkeit

Tick-Systeme (Soft-Real-Time Simulation)

- Handlungen werden nur zu fest getakteten Zeitpunkten (Ticks) verarbeitet.
- Aktionen können zu beliebigen Zeitpunkten erzeugt werden
- Ein Tick hat ein Mindestdauer (z.B. 1/24 s)
  => feste Synchronization von Echtzeit und Spielzeit
- Alle Aktionen innerhalb eines Ticks gelten als gleichzeitig
  (keine Serialisierung)
- Der nächste Game State entsteht aus einer kumulierten Betrachtung aller Aktionen (keine Isolation)
- Verwendetes Modell im Rendering, da hier fest Frameraten und gleichzeitige Änderungen notwendig sind

Zeit Modellierung für Aktionen

Bewertung des Tick-Systems

Vorteile:

- Synchronisation zwischen Echt und Spielzeit
- Faire Regelung der Aktionen pro Zeiteinheit
- Gleichzeitigkeit

Nachteile :

- Lag-Behandlung
  (Server schafft Tick-Berechnung nicht rechtzeitig)
- Konfliktlösung bei gleichzeitigen und widersprüchlichen Aktionen
- Zeitliche Reihenfolge
  (alle Aktionen die in einem Tick ankommen werden als Gleichzeitig betrachtet)

Zeit Modellierung für Aktionen

weitere wichtige Aspekte im Tick-System:

- Berechnungszeit eines Ticks hängt von vielen Einflüssen ab:
  - Hardware
  - Größe des Game State
  - Aktionen
  - Komplexität der Aktionen
  - Durchführung der Synchronisation und der Aufgaben andere Subsysteme:
    - Verteilung eines Game States an das Persistenz-System

Aktionen und Transaktionen

Züge/Handlungen erinnern stark an Transaktionen in DBS

- Atomarität: Zug/Handlung wird ganz oder gar nicht durchgeführt

  Beispiel: Spieler A zieht, Schachuhr für A wird angehalten, Schachuhr für B läuft weiter
- Consistency: Überführen eines gültigen Game States in einen anderen gültigen Game State
- Dauerhaftigkeit: Ergebnisse von Transaktionen stehen fest im Game Sate und werden (zumindest partiell) an das Persistenzsystem übergeben.

Außerdem:

Übergänge müssen den Spielregeln entsprechen (Integritätserhalt)

- statisch: Game State ist regelkonform
- dynamisch: Übergang ist regelkonform

Unterschiede zu Transaktionen

Zeitliche Verarbeitung der Handlungen spielt wichtige Rolle

- Durchführung von Handlungen sollte möglichst fair sein
  - keine Verzögerung der Handlungen eines Spielers
  - Anzahl der max. Handlungen pro Zeit für alle Spieler gleich
- Gleichzeitiges Handeln (Simulation der Realität) sollte prinzipiell möglich sein
- max. Bearbeitungsdauer für flüssiges Spiel notwendig
  evtl. Wegfallen von Handlungen bei Zeit-Überschreitungen
- Zeitsynchronisation zwischen Spielzeit und Echtzeit sollte möglich sein

Unterschiede zu Transaktionen

kein zwingender logischer Einbenutzerbetrieb (Isolation)

- Bei gleichzeitigen Handlungen müssen Aktionen in Abhängigkeit berechnet werden. (keine Serialisierbarkeit)
- Beispiel:
  - Charakter A hat 100/100 HP (=Hit Points)
  - Zum Zeitpunkt t_j bekommt A 100 HP Schaden von Charakter B
  - Zum Zeitpunkt t_j bekommt A 100 HP Heilung von Charakter C

  Ergebnis unter Isolation: A stirbt,
  - erst Heilung (Überheilung) und dann Schaden=> A hat 0 HP und stirbt
  - erst 100 Schaden => A stirbt und Heilung ist nicht mehr möglich

  Ergebnis unter gleichzeitigen Handlungen:
  - A bekommt 100 Schaden und 100 Heilung: Verrechnet heben sich beide Effekte auf.

Umsetzung im Game Loop

- Endlosschleife in der Aktionen auf den aktuellen Game State angewendet werden und diesen so konsistent verändern (Handlungsverarbeitung)
- Zeitmodell ist für den Start der nächsten Iteration verantwortlich
- weitere Funktionalitäten die vom Game Loop abhängen
  - Lesen und Verarbeiten von Benutzereingaben (=> Benutzerhandlungen)
  - Aufruf der KI von NPCs (=> NPC Handlungen)
  - Aufruf Umgebungsmodell

  } erzeugen Handlungen
  - Graphik und Sound Rendering
  - Speichern bestimmter Spielinhalte auf dem Sekundärspeicher
  - Übermittlung von Daten an das Netzwerkm
  - Update unterstützender Datenstrukturen (räumliche Indexstrukturen, Graphik-Buffer,..)
  - …

Realisierung von Game Loops

- Ein Game Loop für alle Aufgaben:
  - kein Overhead durch Synchronisation => Effizient
  - schlechte Schichtung der Architektur: bei Änderung eines Aspektes muss auch der Game Core überarbeitet werden

- Unterschiedliche Game Loops für unterschiedliche Subsysteme (Bsp.: AI-Loop, Netzwerk-Loop, Rendering Loop, ..)
  - Gute Schichtung des Systems
  - Subsysteme können bei Client-Server Einteilung ausgeschaltet werden
    - Client braucht keine eigen NPC Steuerung
    - Sever braucht keine Rendering Loop
  - Synchronisation der Game Loops

Kommunikation mit dem Game Loop

- Game Loop ruft andere Module auf
  - Lösung für Systeme, die im Takt oder langsamer als Game Loop laufen
  - Schlecht geeignet für Multi-Threading
  - Beispiel: Persistenz-System, Netzwerk, Sound Rendering, …
- Game Loop schickt Messages an Subsystem
  - Erlaubt Multi-Threading
  - Aufrufhäufigkeit ist ein Vielfaches vom Takt der Game Loop
  - Beispiel: NPC-Steuerung, Synchronisation mit Clients, Sound-Rendering…
- Synchronisation über lesenden Zugriff auf Game State
  - Bei schneller getakteten Systemen benötigt Subsystem eine eigene Loop
  - Multi-Threading mit umfassendem Zugriff auf dem Game State
  - Gelesene Daten müssen konsistent sein (noch nicht geändert)

    z.B. Graphik-Rendering, Persistenz-System,…

Bearbeiten von Aktionen

- Aufgabe der Aktionsbearbeitung: Umsetzung von Spiel-Aktionen (laufen, schießen, springen,..) in Änderungen des Game State
- Aktionsverarbeitung ist dabei Umsetzung der Spielmechanik
- Berechnungsvorschriften für erlaubte Aktionen
- Leseoperation
- Schreibeoperationen
- Verwendung von Subsystemen möglich

  z.B. Spatial Management Module oder Physics Engine

Konsistenzerhalt bei der Aktionsverarbeitung

- Im Tick-System: gleichzeitige Aktionen möglich
- Reihenfolgeunabhängigkeit bzgl. der Aktionen in einem Tick
- Problem: Lesen von bereits geänderten Daten
- Lösungen:
  - Schattenspeicherkonzept:
    - Es gibt 2 Game States G1 und G2
    - G1 enthält den letzten konsistenten Stand (aktiv)
    - G2 wird in aktueller Iteration geändert (inaktiv)
    - Bei Abschluss des Ticks wird G2 auf aktiv und G1 auf inaktiv gesetzt
  - Feste Reihenfolge von Lese und Schreibeoperationen
    - Erfordert Zerlegen und Neuordnen der Aktionen
    - Alle Aktionen werden gleichzeitig bearbeitet

Konflikte bei Gleichzeitigkeit

- Bei Gleichzeitigkeit können Konflikte entstehen (z.B. Gleichzeitiges aufheben einer Goldmünze)
- Problem das Ergebnis der Handlung kann nicht in Isolation berechnet werden. (Wenn A die Münze bekommt kann B sie nicht auch bekommen)
- Konfliktbehandlung:
  - Streichen beider Handlungen (Undo both)
    => Konflikterkennung und evtl. Rücksetzten der Daten
  - Zufällige Auswahl einer der Aktionen und Löschen (random)
    => Konflikt muss erkannt und evtl. Rücksetzen der Daten
  - Erste Aktion bekommt recht (natural order)
    => Lösung nicht unbedingt fair
    (Ausführungsreihenfolge≠ Handlungsreihenfolge)
    => aber: Einteilung in Ticks kann die Handlungsreihenfolge ohnehin beeinflussen