Ludwig Maximilians Universität München
Department „Institut für Informatik“
Lehr- und Forschungseinheit für Datenbanksysteme

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

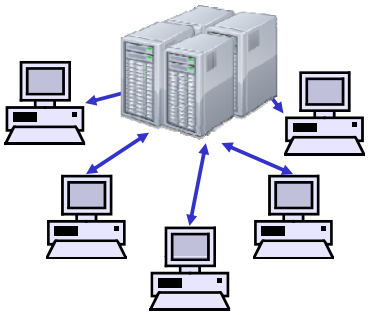
Kapitel 3: Verteilte Spielarchitekturen

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

Kapitelübersicht

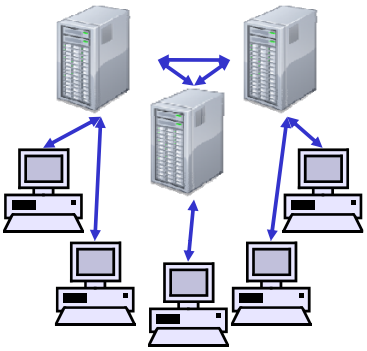
- Architekturmodelle für verteilte Spiele
- Aufteilung der Aktionverarbeitung
  - Fat-Client vs. Thin-Client
  - Probleme bei zentraler und Dezentraler Berechnung
  - Probleme bei lokalen Zeitstempeln
- Räumliche Bewegung und Dead Reckoning
  - Update-Strategien
  - Bewegungsmodelle
  - Fehlerkorrektur
- Netzwerkprotokolle und Spiele
  - typiche Netzlast durch Spiele
  - TCP und Spiele
  - UDP und Spiele

MMOG Architekturen



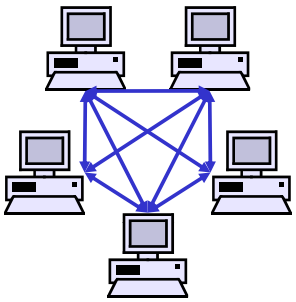
Client-Server:

- Betreiber hostet das Spiel in einem Rechenzentrum
- Spielclient und Server in unterschiedlicher Software
- zentrale Lösungen für:
  - Accountverwaltung
  - Aufteilung der Spielwelt
  - Monitoring
  - Persistenz



Multi-Server:

- mehrere Serverlokalitäten
- Redundante Datenhaltung
- Netzwerkdistanz zwischen Client-Server idR kürzer
- dynamische Lösungen:
  - Replikation
  - Proxy-Server

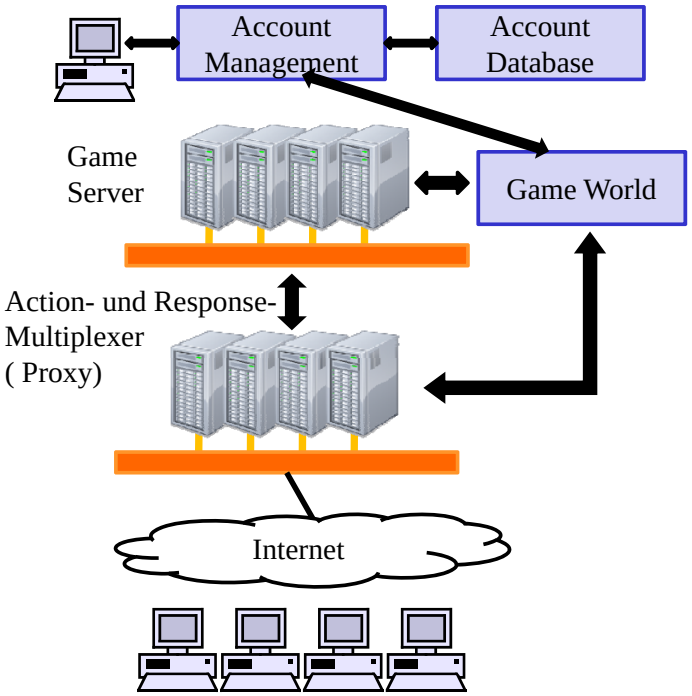


Peer-to-Peer:

- keine expliziten Server
- Datenaustausch zwischen benachbarten Tiers
- jedes Peer hostet einen Teil der Spielwelt
- Aufteilung der Spielwelt ist dynamisch organisiert

Client-Server-Architektur in Detail

- Hosting in einem Rechenzentrum
- mehrere Game Server teilen sich den Game State eines Realms
  - Zonen.Shards/Realms, Instanzen
  - strikte Trennung der Zonen
  - Seamless Aufteilung (Kommunikation zwischen den einzelnen Servern)
- Es gibt eigene Dienste zur Authentifizierung/ Accountverwaltung
- Action- und Reponse-Multiplexer (Proxy) können die Game Server entlasten indem Sie bestimmte Aufgaben des Servers übernehmen



Aufteilung des Game Cores in verteilten Systemen

Design-Entscheidungen:

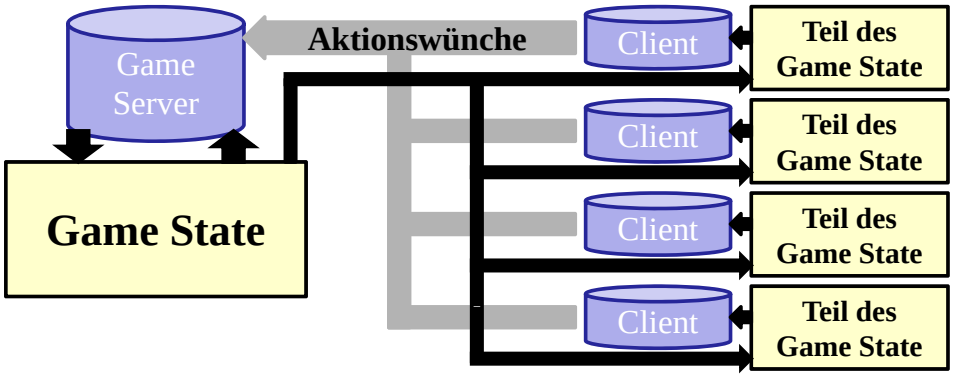
- Welche Arten von Teilnehmern (Peers) gibt ?
- Was tauschen die einzelnen Peers miteinander aus?
  (Aktionen, Objektzustände, Benutzereingaben,..)
- Wer darf welchen Teil lesen und wer darf auch schreiben?
- Wie wird die Last über die vorhandenen Peers umverteilt?
- Wie synchronisiert man die Zeit zwischen den einzelnen Peers?

Protokollinhalte

- Objektattribute: (action result protocol)
  - Protokoll setzt aktuelle Parameterwerte einer Game Entity
    (Setze HP von Spieler „ Facemeltor“ auf 96)
  - Protokoll versendet relative Änderungen
    (Reduziere HP von Spieler „Facemeltor “ um 100)
- Aktionen: (action requests protocol)
  - Enthält nur Spielereingabe aber keine direkte Auswirkung
    auf den Game State
  - Protokoll übermittelt nur Benutzereingaben
    => Ergebnisse müssen am Server berechnet werden
    (Versuche Spieler „Facemeltor “ mit „Aufwärtshacken“ zu schlagen)

Reine Thin-Client Lösung

- Server hält gesamten Game State und darf als einziger GEs ändern
- Clients bekommen einen Teil des Game States beim Einloggen übermittelt
- Der Server schickt Änderungen der GE an die Clients weiter
- Der Client schickt dem Server Aktionen, die er gerne Ausführen möchte. (Action requests)
- Der Server sammelt alle eintreffende Aktionen der Clients
- Die Aktionen werden in der Reihenfolge Ihres Eintreffens verarbeitet und der Server schickt das Ergebnis an die betroffenen Clients



Reine Thin-Client Lösung

Vorteile:

- Game State wird zentral verwaltet
  - Konsistenter Game State zur Berechnung der Aktionsergebnisse
  - keine Konflikte durch mehrere widersprüchliche Game States
  - Persistenz-System kann konsistente Spielstände sichern
- Geringes Cheat-Potential/ Aktionsverarbeitung nur auf dem Server

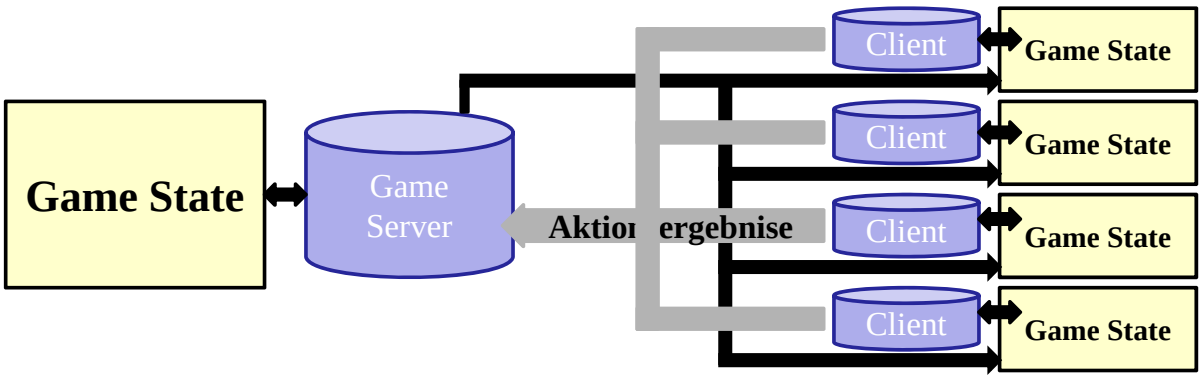
Nachteile:

- max. Serverlast, da komplette Aktionsverarbeitung server-seitig ist
- Potential für hohe Latenz-Zeiten (Handlungen müssen zum Server und zurück bevor Sie Wirkung zeigen können)
- Rechenleistung auf dem Client bleibt weitgehend ungenutzt

  (Client stellt nur das Abbild des Game States dar und leitet die Benutzereingaben an den Server weiter)

Reine Fat-Client Lösung

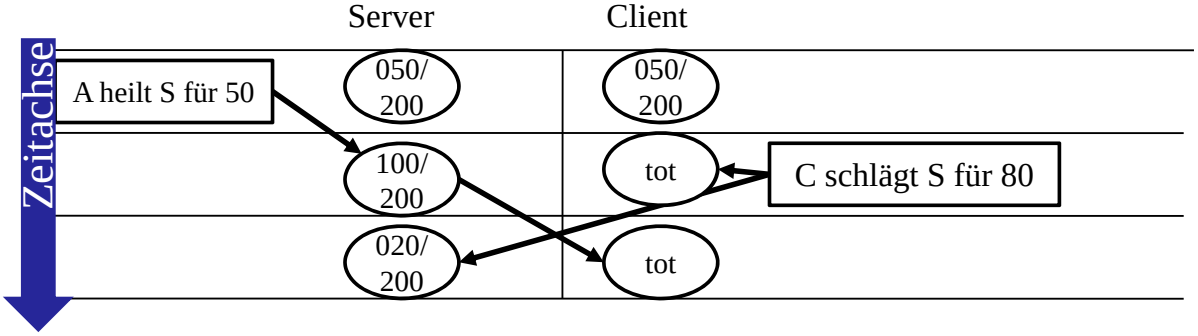
- Jeder Client hat eigene Objekte, die nur er ändern darf
- Server regelt die zeitliche Reihenfolge über Timestamps und verschickt Änderungen an die anderen Clients
- lokale Game States können durch Übertragungsverzögerungen variieren
- zeitliche Reihenfolge kann inkonsistent sein da lokale Änderungen vor globalen mit niedrigerem Zeitstempel verarbeitet werden können



Konflikte bei dezentraler Berechnung

- Lokale Änderung brauchen Zeit bis sie im Netzwerk verteilt werden
- Aktionen werden auf lokalen Spielständen berechnet und dort ausgeführt

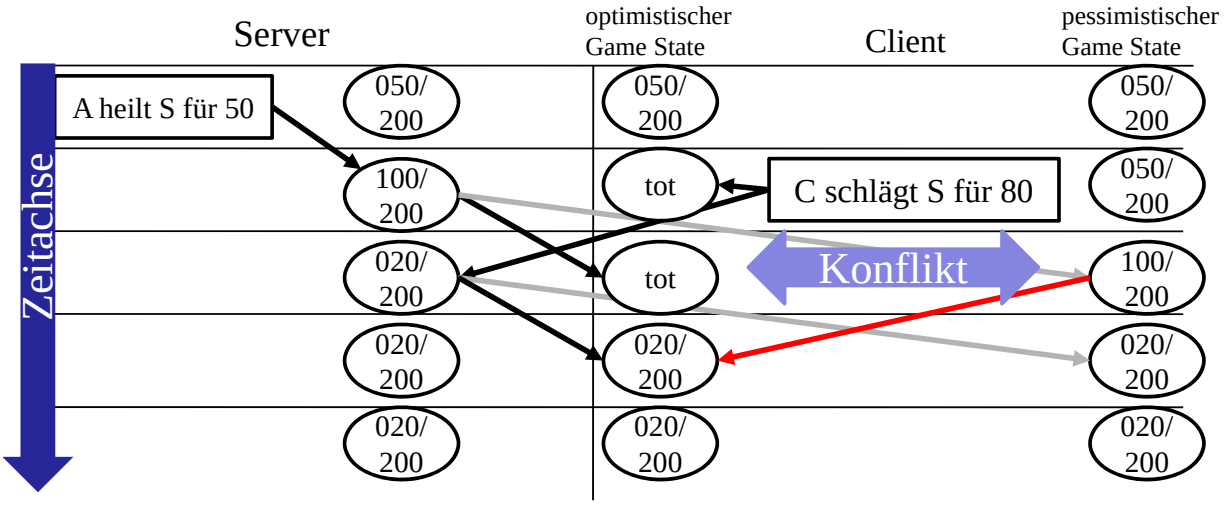
  => Änderungen die vor der eigenen Aktion passiert sind werden evtl. nicht berücksichtigt
- Einfache Lösungen:
  - Client darf nicht lokal ändern, lokale Änderungen werden erst nach Rücksendung des Servers ausgeführt.
  - Bei Objektprotokollen kann der Server ein Update des aktuellen Standes der Entities verschicken.



Lösungsansatz

Rücksetzen der lokalen Aktion

- Client besitzt 2 Game States:
  - optimistischer GS (enthält lokale Änderung)
  - pessimistischer GS (enthält die vom Server geschickten Aktionen)
- Bei Unterschied: Rücksetzen des optimistischen GS auf den Stand des pessimistischen GS



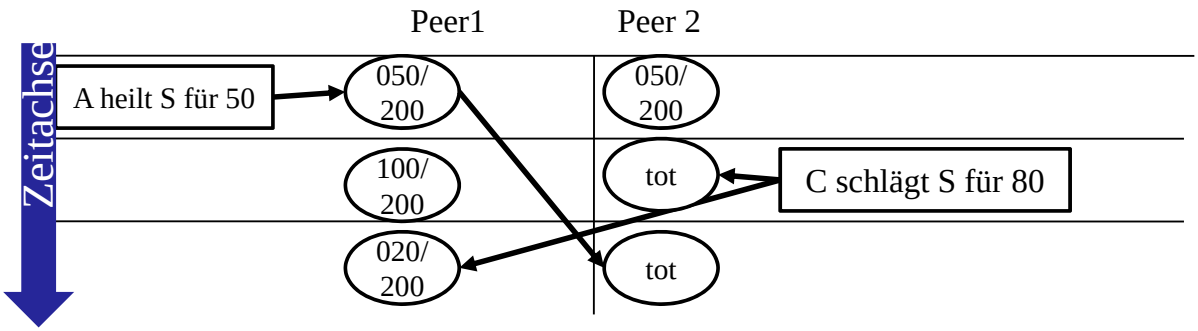
Lokale Zeit

Bisher: 1 Server regelt die Verarbeitungsreihenfolge

- nicht möglich in P2P Spielen und bei mehreren Servern

  => Ordnung nach Eintreffen beim Server nicht mehr eindeutig

  => Ordnung nach lokalen Zeitstempeln bei Erzeugung
- bei Verarbeitung können nicht nur eigene sondern auch fremde Änderungen in der falschen Reihenfolge auftreten
- Bei Inkonsistenzen können Game Entities synchronisiert werden

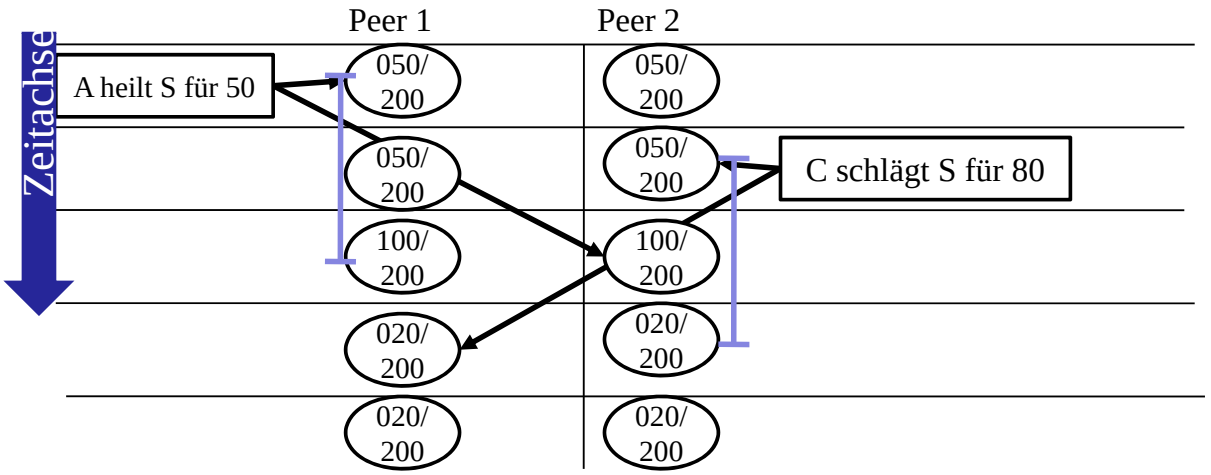


Lösung mit Lag-Mechanismus

Problem entsteht durch Unkenntnis aller vorheriger Aktionen

Lösung: Lag-Mechanismus

- Änderungen werden verzögert ausgeführt um anderen Aktionen Zeit zu geben noch einzutreffen
- Falls dieses Zeitfenster überschritten wird ist Konflikterkennung und Rücksetzen notwendig



Anwendungen in Spielen

Spiele können mehrere Lösungsansätze kombinieren indem Sie Aktionen unterschiedlich verarbeiten lassen.

Serverseitige Verarbeitung	**Clientseitige Verarbeitung**
• hohe Korrektheit ist entscheidend • Anwortzeit weniger wichtig • zeitliche Abfolge ist wichtig	• Antwortzeit ist entscheidend • Synchronisation und Reihenfolge sind weniger wichtig
• Schaden und Heilung • Aufheben von Gegenständen	• Bewegungs- und Positionsdaten • Animationen und andere Darstellungseffekte

Fazit:

- Generell existiert ein Trade-Off zwischen Latenz-Zeit (hier Antwortzeit des Spiels) und Konsistenz der Spielwelt.
- ein weiterer Aspekt ist die Reduktion von Änderungsübertragungen zur Reduktion der benutzten Bandbreite.

Bewegungsinformationen

Bewegungs-Updates haben eine spezielle Rolle in verteilten virtuellen Umgebungen

- Flüssige Bewegung
  - ⇒ Position kann sich bis zu mehrere male pro Sekunde ändern (24-60 FPS)
  - ⇒ Berechnung sollte eng mit dem Rendering verknüpft sein
  - ⇒ bei Gleichbehandlung von Bewegung und anderen Aktionen würde die Animation stark leiden
- Exakte Positionen sind zum Großteil nicht relevant für das Spielgeschehen:
  - Durch die hohe Updaterate ist der Verlust mehrerer Positionsupdates meist marginal

Folgen:

- Bewegungen in Real-Time Spielen werden überwiegend lokal auf dem Client berechnet
- Folgen von exakten Positionen werden nicht an weitere Peers übertragen, um Bandbreite zu sparen
- Bewegungen werden lokal extrapoliert und zu bestimmten Zeit werden die Positionen synchronisiert

=> **Dead Reckoning**: Simulation der Bewegung zwischen zwei Updates, um flüssige Bewegungen unter begrenzter Bandbreite zu ermöglichen.

Dead Reckoning

Komponenten zur Umsetzung:

- Update-Strategie auf dem Besitzer der GE:
  Wann werden Positionsinformationen gesendet und wie häufig?
  (Beeinflusst Bandbreite und Fehlerrate auf dem Client)

- Bewegungsmodell auf dem Remote Peer:
  Wie wird die Bewegung zwischen 2 Updates extrapoliert ?
  (beeinflusst Fehlerrate und Wahrnehmung der Bewegung auf dem Client)

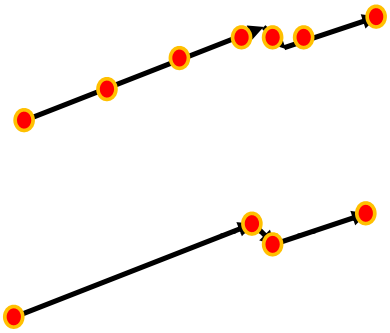
- Fehlerkorrektur auf dem Remote Peer:
  Wie werden die geschätzte und die übermittelte Position wieder zusammengeführt?
  (beeinflusst Wahrnehmung auf dem Client)

=> Es existiert ein Trade-Off zwischen :

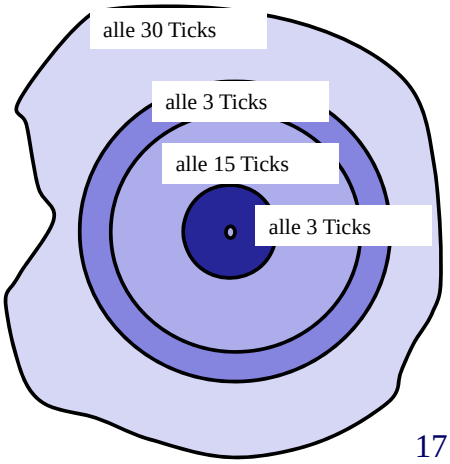
- Bandbreite und Fehlerrate
- Wahrnehmung und Rechenzeit

Update-Strategien für Dead Reckoning

- Regelmäßige Updates:
  - Sende Updates in regelmäßigen Abständen
- Event-basierte Updates:
  - Sende Update bei Änderungen der Bewegungsrichtung oder Art



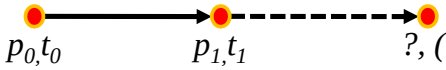
- Distanzbasierte-Updates:
  - exakte Positionen sind wichtiger je näher ein Objekt ist
  - je näher das Objekt an einer kritischen Reichweite ist
  - übermitteln regelmäßiger Updates, aber mit unterschiedlichen Raten je nach Abstand.



Bewegungsmodelle für Dead Reckoning

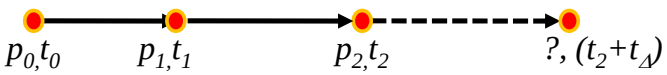
Zeitpunkt: t_i *Position:* $p(t_i)=(x_i,y_i)$ *Durchschnittsgeschwindigkeit:* $v(t_i)$ *Beschleunigung:* $a(t_i)$

Lineare Bewegung mit konstanter Geschwindigkeit:



$$p(t_1+t_\Delta) = p_1 + \underbrace{\frac{p(t_1)-p(t_0)}{\|p(t_1)-p(t_0)\|}}_{\text{Richtung}} \cdot t_\Delta \cdot \underbrace{\frac{\|p(t_1)-p(t_0)\|}{(t_1-t_0)}}_{\text{Geschwindigkeit}} = \boxed{p_1 + t_\Delta \cdot \frac{p(t_1)-p(t_0)}{(t_1-t_0)}}$$

Linear Bewegung mit konstanter Beschleunigung:



$$v(t_2+t_\Delta) = v(t_2) + t_\Delta \cdot \frac{v(t_2)-v(t_1)}{(t_2-t_1)} = v(t_2) + t_\Delta \cdot \frac{\frac{\|p(t_2)-p(t_1)\|}{(t_2-t_1)} - \frac{\|p(t_1)-p(t_0)\|}{(t_1-t_0)}}{(t_2-t_1)}$$

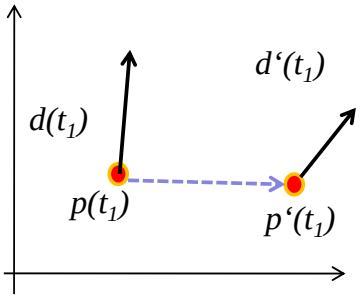
$$p(t_2+t_\Delta) = p(t_2) + t_\Delta \cdot v(t_2+t_\Delta) \cdot \frac{p(t_1)-p(t_0)}{\|p(t_1)-p(t_0)\|}$$

Fehlerkorrektur für Dead Reckoning

Problem Vorhersage und Update stimmen nicht überein.

- Objekt auf dem Remote-Peer wird durch Update überschrieben

  Bei hoher Fehlerrate:
  - Objekte springen
  - Objekte verschwinden und tauchen woanders wieder auf

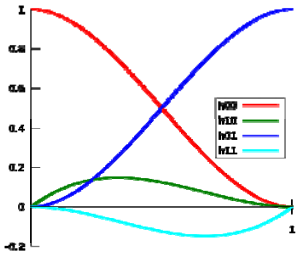


- Beide Positionen werden mit einer beschleunigten Bewegung flüssig zusammengeführt:
  - z.B. kubische Polynome: Bezier, B-Splines, Hermite
  - hierbei muss eine gewisse Korrekturzeit Δt eingeplant werden

Hermite Kurven zur polynomiellen Glättung

Vier Basispolynome:

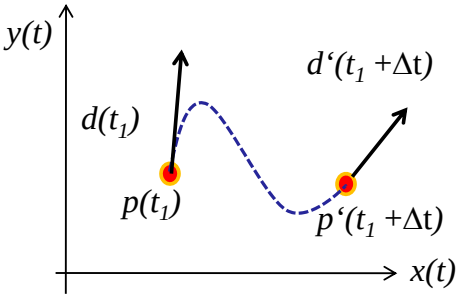
- $h1(x) = 2x^3 - 3x^2 + 1$
- $h2(s) = -2x^3 + 3x^2$
- $h3(x) = x^3 - 2x^2 + x$
- $h4(x) = x^3 - x^2$



Verbindung der Punkte p und p’ über folgende Linearkombination

$p(\Delta t) = p(t_1)\, h1(x) + p'(t_1 + \Delta t) h2(x) + d(t_1)\, h3(x) + d'(t+\Delta t)\, h4(x)$

- Position: $p(t_1)$ durch Dead Reckoning
- Bewegungsvektor: $d(t_1)$ durch Dead Reckoning
- Ziel: $p'(t_1 + \Delta t)$ durch Server-Update
- Endbewegungsrichtung: $d'(t+\Delta t)$ durch Server-Update
- Δt Zeit für die Korrektur (Kompensation durch höhere Geschwindigkeit)



Überlegungen zur Client-Server Kommunikation

Wichtige Einflussfaktoren

- **Latenzzeit**: Zeit bis das System reagiert
  - Round Trip Time (RTT)
  - Paketgröße
  - Systemlast außerhalb des Netzwerks
- **Bandbreite**: Wie hoch ist das Übertragungsvolumen?
- **Burstiness**: Wie verteilt sich das Datenvolumen über die Zeit?
- **Verbindungsorientierung/Packetorientierte Protokolle**
  - Verbindungsorientiert: Wegewahl findet 1 mal statt
  - Packetorierntiert: Routing findet pro Paket statt
- **Sicherheit**: Ist Datenverlust möglich?

Anforderungen von Computerspielen

application/platform	payload size (bytes) avg.	min	max	avg. bandwidth requirement pps	bps
Anarchy Online(PC)‡	98	8	1333	1.582	2168
World of Warcraft (PC)	26	6	1228	3.185	2046
Counter Strike (PC)	36	25	1342	8.064	19604
Halo 3	247	32	1264	27.778	60223
Gears of War (XBOX 360)	66	32	705	2.188	10264
Tony Hawk's Project 8 (XBOX 360)	90	32	576	3.247	5812
Test Unlimited (XBOX 360)	80	34	104	25	22912

aus: Harcsik, Petlund, Griwodz,Halvorsen: Latency Evaluation of Networking Mechanisms for Game Traffic, NetGames'07, 2007

- geringe Packetgrößen
- wenig Bandbreite ist erforderlich
- Latenzzeiten für Genres:
  - RTS-Spiele: <1000 ms
  - RPG: < 500 ms
  - FPS: < 100 ms

  (Geschätzte Latenz ab der eine Beeinträchtigung des Spielerlebnisses beobachtet wird)

Protokolle und Kommunikationslösungen

TCP/IP:

- sicheres Protokoll: über Neuübertragung
- Flusskontrolle und Congestion Control
- optimiert auf gute Ausnutzung der Bandbreite und Übertragung von Dateien (senden großer Pakete zur Reduktion der übertragenen TCP-Header)

Nachteile:

- Pakete können stark verspätet eintreffen (Neuübertragung)
  - => Latenzzeit leidet
  - => Packet wird teils nicht mehr benötigt, da neuere Informationen bereits übermittelt wurden
- Mechanismen zur Optimierung der Bandbreite erhöhen die Latenz künstlich
  - Warten auf Payload bei Paketunterfüllung
  - Bestätigungspakete bestätigen gleich mehrere Sendungen oder werden in dem Rückverkehr eingebettet

Optimierung durch Ausschalten von Features und Tuning

Protokolle und Kommunikationslösungen

UDP

- minimaler Datagramm Dienst
- keine explizite Verbindung zur Gegenstelle
- unsichere Übertragung, keine Reihenfolgegarantie
- keine Congestion Control Mechanismen

Vorteile:

- keine Neuübertragung bei Paketverlust
  => kein Nachsenden veralteter Informationen
- wenig Headeroverhead

Anwendung:

- dient als Grundlage von Middleware-Lösungen die fehlende Service-Merkmale auf der nächsten Protokollschicht implementieren:
  - Reihenfolgeerhalt
  - Sicherheit für bestimmte Meldungen (z.B. Inbesitzname von Gegenständen..)

Fazit Netzwerkprotokolle

- TCP/IP bleibt das am meisten verwendete Protokoll, da Infrastruktur und Router gut damit zurechtkommen
- UDP bietet eine kosteneffiziente Lösung für just-in-time Dienste (Sprache, Bewegungsdaten,…)
- Sichere Dienste sind aber für die meisten Spielen unabdingbar und müssen dann auf der Anwendungsschicht des Protokolls umgesetzt werden
- Es gibt Untersuchungen zu anderen Protokollen (z.B. SCTP), die keine entscheidenden Performanzgewinne nachweisen konnten.
- MMORPGs (z.B. World of Warcraft) verwenden TCP für ihre Kommunikation

Lernziele

- Client-Server und P2P Architekturen für Spiele
- Aufteilung der Aktionsverwaltung:
  - globale Verarbeitung
  - lokale Verarbeitung mit zentraler zeitliche Ordnung
  - lokale Verarbeitung mit lokaler zeitliche Ordnung
- Dead Reckoning
  - Update-Strategien
  - Bewegungsmodelle
  - Fehlerkorrektur
- Anforderungen von Spielen an Transportprotokolle
  - TCP und Spiele
  - UDP, Middleware und Spiele

Literatur

- N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, W. White
  Scalability for virtual worlds
  In Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, 2009.

- Jens Müller, Andreas Gössling, Sergei Gorlatch
  On correctness of scalable multi-server state replication in online games
  In Network and System Support for Games (NETGAMES'06), 2006.

- Jouni Smed, Timo Kaukoranta, Harri Hakonen
  A Review on Networking and Multiplayer Computer Games
  In IN MULTIPLAYER COMPUTER GAMES, PROC. INT. CONF. ON APPLICATION AND DEVELOPMENT OF COMPUTER GAMES IN THE 21ST CENTURY, 2002.

- S. Harcsik, A. Petlund, C. Griwodz, P. Halvorsen
  Latency evaluation of networking mechanisms for game traffic
  In Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, 2007.