PlaneShift

October, 2008

What is PlaneShift

- MMORPG
  - Like EverQuest or World of Warcraft
  - Extremely ambitious project
- Free to Play
- All code under GPL
- All content under PlaneShift license

Made by Amateurs

- Project started in 2000, by Luca Pancallo in Italy
- Volunteer project with many contributors
  - 62 coders have directly committed code in the project (22 have over 100 commits)
  - Dozens have submitted patches without direct write access as well.
  - Dozens more have worked on other subteams
- Relatively large project
  - About 400K lines of C++ in the core applications, in about 1200 classes
  - Uses Crystal Space openGL library, which is >1 million lines of code itself
  - Uses mysql database. >70 tables and >5GB of data
- 560,000 registered players have tried it, with essentially zero PR. But average concurrently online is only 100-200 24x7.

MMO’s are complex

- “Normal” open source team structure
  - Coding team
  - QA team if you’re really organized

MMO's are complex

- PS team structure
  - 2d (concept art, textures)
  - 3d (world/terrain, items, monsters, characters, animation)
  - Sound (effects, background music)
  - Rules (item stats, item definitions, player stats, balancing, progression)
  - Settings (names, npc definitions, quests, histories, books, spells)
  - Web development (game content management)
  - Engine (coding) + QA team

- Key learning point for me on the project:
  - The content is much more difficult, time-consuming and costly than the engine.

Server Unique Design Constraints

- No money = donated hardware
  - Everything runs on 1 box right now
  - If we move to multiple boxes, they will almost certainly not be colocated = High latency
- No money = volunteer development team
  - High turnover on a large project
  - Learning curve is a major issue
  - Modularity of design is critical

So let’s see it

- http://www.youtube.com/watch?v=MZx-ZwzZwCY

Server Design

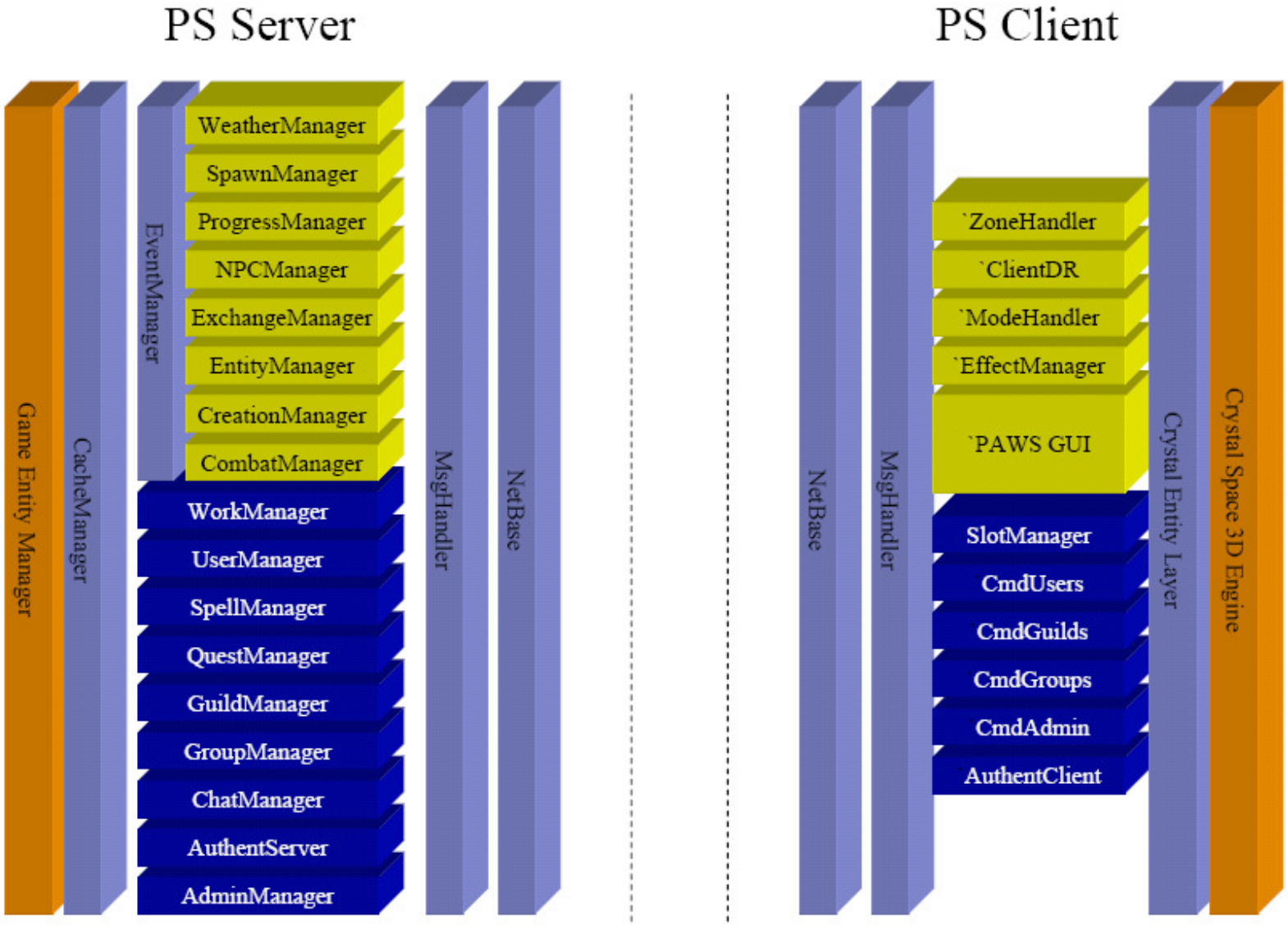
Overview

- Core of server is an event dispatcher
- Events can be:
  - Network messages
  - Time triggered messages
  - Intra-server messages
  - No distinction is made internally between these other than queue priority
- Basic Producer/Consumer threading model for networking vs worker threads.
- Publish/Subscribe Architecture allows for “managers” to act like plugins, each specializing in a certain class of events.

Benefits of Manager model

- Classes defined along functional game lines
  - Combat, Spells, Chat, Login all in distinct sources
  - Easy to find code that handles a feature of interest
- Lower learning curve
  - Get comfortable with a single source file before moving on to others
  - Some never move on, but become the deep maintainers of a certain manager
- Easy to bolt in new functionality
  - One more manager has no effect on performance or on other managers

“Manager” model in PS



Networking / Entity Management

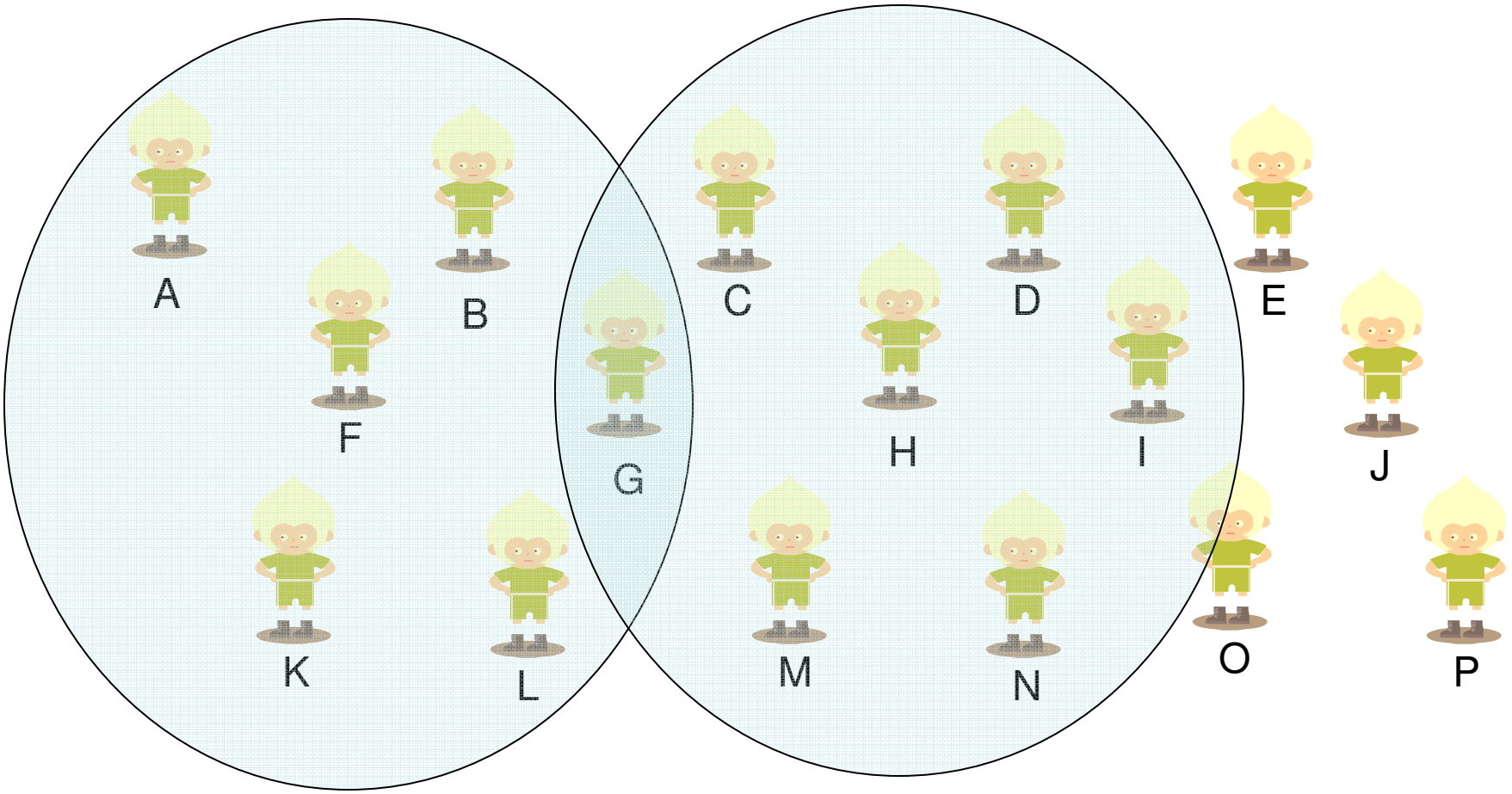
Base networking

- UDP based
  - TCP can result in higher latency due to guarantees on ordering of packets
- Two priority levels of messages
  - Guaranteed delivery (High)
  - Non-guaranteed delivery (Low)
- We need a third priority
  - Guaranteed order (Highest)
  - On the to-do list but not done yet
- Everything is packed pretty tightly byte by byte, and endian correct
- #1 limiting factor on server capacity is bandwidth consumption
  - Not CPU
  - Not Memory

Entity Management

- Server maintains all entities in-game
  - An “entity” is any object added after the fact to the maps
  - Characters, monsters, npcs, items dropped by players
- Each client gets a small subset of these entities
  - As a result, entities are constantly being created and destroyed on each client
  - The server maintains a “Proximity List” for each client, which holds all relevant entities for that client
  - Prox Lists determine what is visible to each client, which means it determines what updates are relevant to each client also

Proximity Lists Diagram



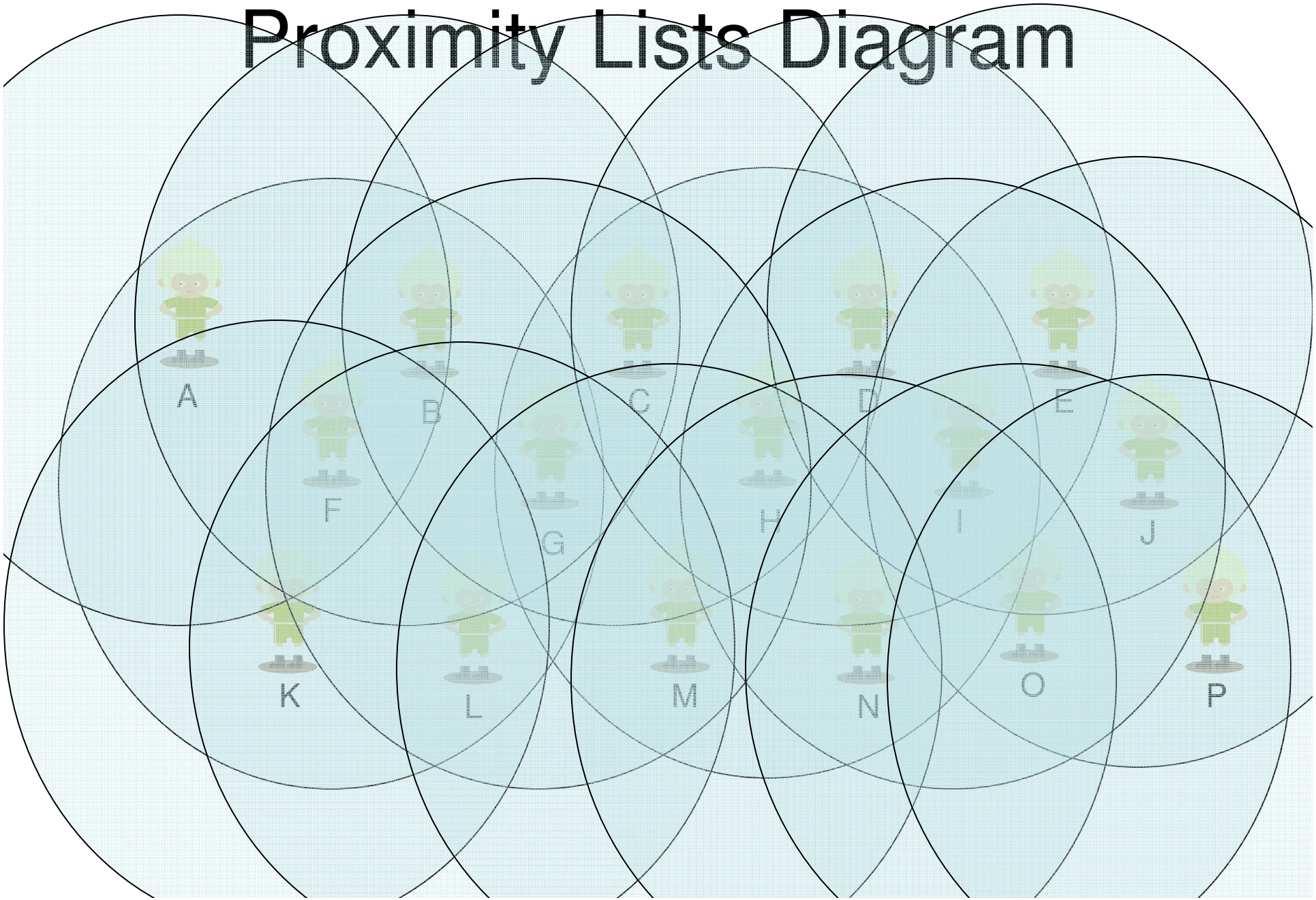
Player F can see himself and 5 other characters

Player H can see himself and 6 other characters

Proximity Lists

- Relevance is determined by:
  - Within a threshold distance to the player
  - “visible” to the player (complex)
  - Within the same instance (or compatible instance) to the player
- Benefits
  - Better for security (see future slides)
  - Minimizes bandwidth to relevant bandwidth
  - Very easy construct for Managers to use with broadcasting (multicasting)
- Costs
  - Server spends a lot of time maintaining and updating prox lists

Proximity Lists Diagram



Scripting in the Server

- The game server uses several types of home-made scripting to implement gameplay rather than C++ code.

- Benefits
  - Possible for non-programmers to produce (Rules, Settings, 2d)
  - Proprietary “content” not subject to GPL (i.e. secret)
  - Engine team concentrates on extending the scripting languages rather than coding new features directly
    - Meta programming is much more efficient in developer time

- Costs
  - Complexity of meta code increases learning curve for programmers

Math Scripts

"Calculate Repair Time"

```
Difficulty = Object:SalePrice/100;
Factor     = Worker:getSkillValue(Object:RequiredRepairSkill) / (Object:SalePrice/20);
Result     = Difficulty / Factor;
```

"Calculate Repair Result"

```
Factor = Worker:getSkillValue(Object:RequiredRepairSkill) / (Object:SalePrice/20);
Result = ((Object:SalePrice/25) * Factor) * (rnd(1)+0.5);
```

Highlights

- Precompiled to p-code on server startup
- Dynamic binding to objects like “Object” and “Worker” above
- Any C++ class can be bound if it implements iScriptable

Progression Scripts

"DamageHP"

```
<evt>
   <hp aim="target"  value="-Param0" />
   <msg aim="target" text="You lost $Param0 hitpoints."/>
   <msg aim="actor"  text="You hit $Target for $Param0 hitpoints."/>
</evt>
```

Highlights

- Loaded and compiled to virtual function calls (each operation is a mini-class)
- Scriptable by Rules team
- Value attribute can be any single line mathscript.

Quest Scripts

Quest: "Male Enki Alina Quest"

P: Greetings

Marth: Hail! Would you like to earn a little money?

P: No P: Yes

M: Well ok then. Have a good day.

M: My daughter Alina ran off with the smith and I fear they are up to no good. If you can find out if he really loves her, I'll pay you. Can you do this for me?

P: No P: Yes

M: Fair enough. I guess I'll just try to find someone else.

M: Last time I caught them upstairs above the blacksmith shop. Please hurry
up and ask her!

Assign Quest.

<snip>

P: Greetings

Marth: Oh I'm so glad you are back. Did you find him? Does he love her?{him:Smith,her:Alina}

P: * P: Smith says Smith does. Smith love Alina.

M: That isn't what Thalia told me 5 minutes ago. You're just faking to get
the money! Go away!

M: Ah! Thank heavens. Perhaps now they will get married! Thanks for your
help!

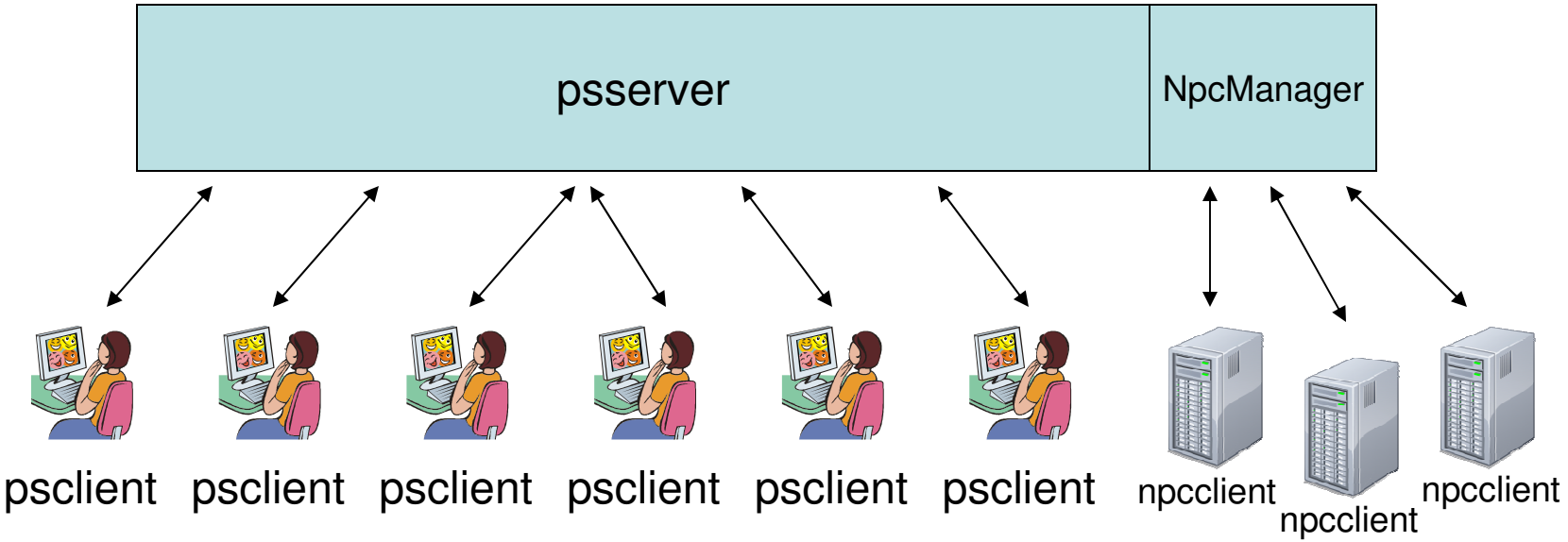
Give 25 tria.

NPC Management

NPC AI is hard to scale

- We have a monolithic server today
  - 100 concurrent players takes <5% CPU
  - We can scale up with players fairly far before machine capacity of the core server is an issue.
- AI is computationally expensive
  - The smarter the monsters, the more expensive
  - AI techniques evolve constantly (and get more expensive)
  - Deep experts exist who may contribute to the game someday
- AI doesn't belong in the same process, or on the same machine as the server

Superclient Architecture



Superclient Concepts

- Separate process
- Connects to server over the network
  - Uses same protocols as client for some things
  - Uses additional special messaging for “bulk” information transfer
- NpcManager class on server
  - Handles these special messages, and unrolls them for the rest of the server, republishing them.

Superclient Benefits

- Simplicity in server/client
  - NpcManager emulation layer means there is no distinction between human and automated players in almost the entire server.
  - There is no distinction on a client machine between an NPC and a remote player.
- Scalability
  - Net connected separate process means the process could be on its own machine.
  - Connecting as a client means that multiple superclients can be connected concurrently.
- Future expansion
  - No limitations on AI techniques used as long as net protocol is followed by each superclient.

How we do NPC AI

- “Artificial Stupidity”
  - Monsters don’t need much intelligence to be challenging to players (Pac-Man)
- Emphasis on CPU efficiency and flexibility
- 1 Superclient controls approximately 300 mobile npcs with <10% cpu today

AI Scripting Model - NPCTypes

- NPCs divided into “types” or classes
  - All scripting is done at the type level
- Types consist of
  - Set of Behaviors
  - Set of Reactions to Perceptions
    - Perceptions sent by the game server
    - Perceptions generated internally by the npcclient
- Inheritance structure for npc types
  - Behavior/Reaction overloading
- Loose structure enables very flexible scripting and potentially emergent behavior
  - Very hard to debug and maintain

AI Scripting Model – NPC's

- Each NPC in the game maintains
  - Priority queue of behaviors in its NPCType, prioritized by "need" to perform the behavior
  - Hate list of known players who have hurt or helped the NPC
- NPC runs the script for the behavior with highest need
- Reactions to perceptions only affect needs and hate.
  - If the need gets high enough, the current behavior is interrupted and replaced with a new behavior.
  - Hate affects targeting in attacks, mostly.

AI Scripting Example

```
<npctype name="Fighter">

   <behavior name="do nothing">
       <wait duration="1" anim="stand" />
   </behavior>

   <behavior name="turn" completion_decay="-1">
       <rotate type="random" min="90" max="270" anim="walk" ang_vel="30" />
       <move vel="2" anim="walk" duration="1.0"/>
   </behavior>

   <behavior name="fight">
       <locate obj="target"  range="50"/>
       <rotate type="locatedest" anim="walk" ang_vel="120" />
       <melee seek_range="50" melee_range="3" />
   </behavior>

   <behavior name="chase" completion_decay="100">
       <chase type="target" anim="run" vel="5" />
   </behavior>

   <react event="collision"           behavior="turn" />
   <react event="out of bounds"       behavior="turn" />
   <react event="attack"              behavior="fight" />
   <react event="damage"              behavior="fight" delta="20" weight="1" />
   <react event="target out of range" behavior="chase" />
   <react event="death"               behavior="fight" delta="0" />

</npctype>
```

Game Security in Open Source

Security Problems with Open Source

- Hacked clients are trivial
  - Speed/Teleport/Anti-gravity mods
  - Visibility mods
  - Bots and Macroing
  - Collision detection hacks
  - Map hacks
  - Item art hacks
  - Texture mods

How to combat these?

- No defense, low impact
  - Texture mods: They are fun to look at but only visible to the player with the mods
  - Item art hacks: Tight prox list radius on critical small items minimizes impact
- High impact, Easy defense
  - Visibility mods: Use of prox lists gives us tight control over what is available for the cheater to see
  - Speed/Teleport mods: Server can compare before and after positions and kick out cheaters

High impact, Hard defense

- Bots and Macroing
  - Extremely hard to detect automatically
  - PS is focusing on making repetitive activities less valuable to the player, so diminishing returns makes automation less valuable
- Collision detection & Map hacks
  - Very expensive in CPU and RAM to monitor on the server centrally.
  - Roadmap is to use a “superclient” type approach and have a separate process connect to the server to watch for cheaters, focusing on individuals at random intervals.

Follow up

Main game website
www.planeshift.it

Project Director is Luca Pancallo
info@planeshift.it

Keith Fulton
keith.fulton@gmail.com

Come help the team if you like free games
and you have a lot of free time!