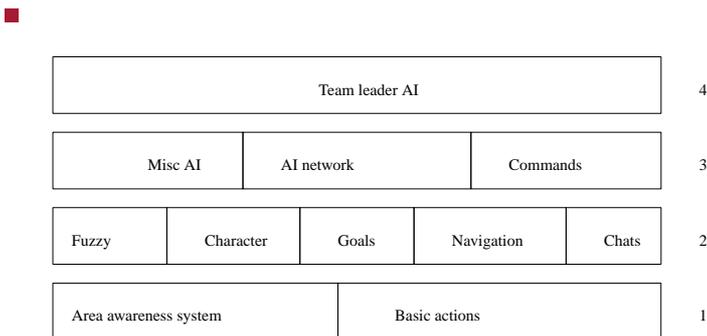


AI in ioquake

- Quake 3 arena box AI is layered
- much like a network protocol stack
- decisions made at higher layers are executed through lower layers

slide 3
gaius

AI in ioquake



slide 4
gaius

Layer one

- the input and output layer for the bot
 - area awareness system is the system which provides the bot with all information about the world
 - much of the information has already been preprocessed when building the maps
 - some of it will be dynamic
 - access to it is fast
 - everything the bot senses goes through the Area Awareness System AAS
- basic actions are the output of the bot
 - outputs are presented in a way which is very similar to keyboard/mouse input of a human player

Layer two

- provides the intelligence that is often subconscious to a skilled human player
 - includes AI to select goals using fuzzy logic
 - AI to navigate towards a goal
 - AI to interpret chats
 - AI to construct chats
- functionality to store and retrieve characteristics of bots, for example
 - aim skill
 - aim accuracy
 - aggression
 - weapon jumping

Layer three

- a mixture of production rules
 - and an AI network with special nodes for different situations and states of mind
 - very similar to a state machine
 - many `if then else` rules
- all higher level thinking and reasoning takes place in this layer
- it also contains the command module
 - which allows the bot to understand orders and commands
 - from other players or a team leader
- misc AI module supports AI for fighting, navigating around obstacles and solving puzzles

Layer four

- one of the bots is designated as team leader which enables this layer
 - all other bots disable this layer
- typically the leader might request other bot to follow me
 - track enemy, attack enemy etc

Tutorial work

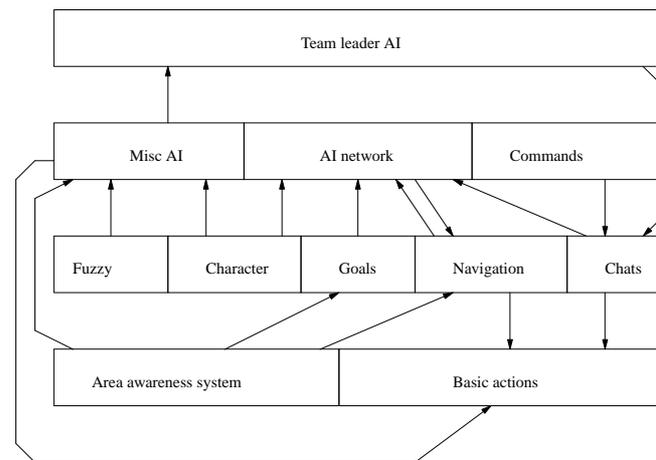
- for each `.C` file in `Sandpit/ioquake-latest/ioquake3/code/botlib`
- write a one line summary of what it does and determine to which level it belongs

AI system

- AI system runs in small time frames
 - referred to as a time-based simulation
- system runs at 10Hz
- this matches the game physics, although they are not locked in step
- each frame the bot checks upon its health and situation and decides the best action to be taken

Information flow between the layers

■



Information flow between the layers

- upward arrows represent information about the bots environment and its status
 - used to keep up to date with the world
 - used in the decision making to achieve certain goals
- data moving up the layered architecture represents more and more abstract concepts
- all information flowing downwards is from the result of a decision taken by the bot
 - will normally result in a basic action being taken
 - jump, move forward etc

Information flow between the layers

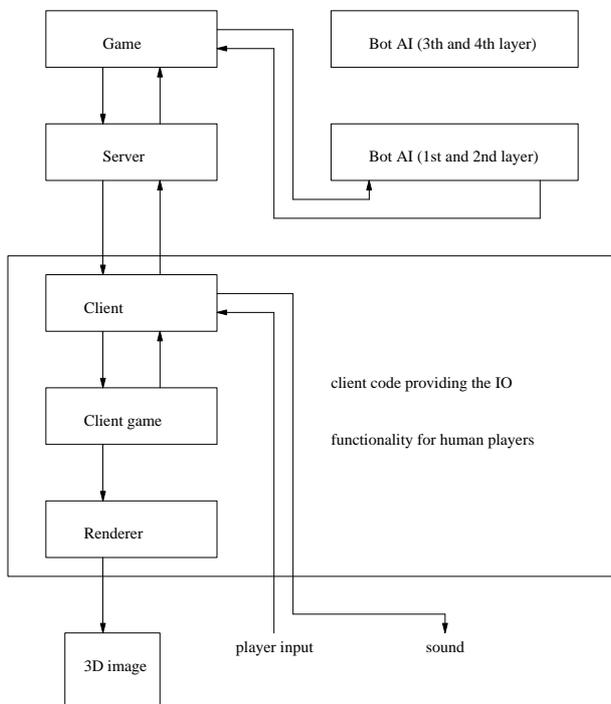
- limited inter layer communication
- in the third layer
 - AI network retrieves information from both the Misc AI and Command components
- in the second layer
 - Goals retrieves information from Fuzzy

Information flow between the layers

- code at higher layers requests information from lower layers
- information readily available is passed back immediately
 - other calculations might need to be made, however such calculations must not take more than a couple of milliseconds
 - otherwise the game will hitch
- at 10Hz a delay during a calculation can become noticeable

Structure of the ioquake game engine

■



Structure of the ioquake game engine

- the game module sets the rules and dictates how the game works
 - capture the flag or death match or team arena
- server provides functionality to allow players to connect to the game
- client, client game, and renderer modules together provide the input/output functionality for the human player

Structure of the ioquake game engine

- `client` module passes input events to the server
- `client` also forwards information from the server to the `client game`
 - for example what is visible

Structure of the ioquake game engine

- `client game` interprets this information and passes necessary data to the `renderer`
- `client game` sends information about which sounds can be heard to the `client`

Structure of the ioquake game engine

- the `game` module provides the Area Awareness System with all the necessary information about the state of the game world
- information consists mainly of entity data
 - position, type, appearance of entities are communicated to the Area Awareness System
 - gibs, player models, skins etc
- bot input, or basic actions are sent to the `game` module

Remote procedure call

- RPC marshalling of arguments
 - make sure you have read Tanenbaum's description of how RPC's operate
 - and how marshalling is employed and why it is necessary
- Andrew Tanenbaum, Modern Operating Systems, Prentice-Hall International
 - sections 10.3, 10.3.1, 10.3.2

md3 bot files

- a md3 file package is just a zip archive
- inside a md3 archive is a collection of files

Some of Alien .pk3 files

```
alien/models/players/alien/head_1.md3
alien/models/players/alien/lower_default.skin
alien/models/players/alien/head_default.skin
alien/models/players/alien/animation.cfg
alien/models/players/alien/head.tga
alien/models/players/alien/lower_1.md3
alien/scripts/Alien.bot
alien/botfiles/bots/alien_i.c
alien/botfiles/bots/alien_c.c
alien/botfiles/bots/alien_w.c
alien/botfiles/bots/alien_t.c
alien/botfiles/bots/alien.bot
```

Some of Alien .pk3 files

- some of the files have been left out of the list above
 - left out different colour alien
- notice that filenames are case insensitive
 - take care under GNU/Linux which by default differentiates between case

contents of alien/botfiles/bots/alien.bot

```
{
  name           Alien
  model          Alien
  funname        Alien
  color          4
  aifile         bots/alien_c.c
}
```

Contents of bots/alien_c.c

```
#include "chars.h"

skill 1
{
    CHARACTERISTIC_NAME      "alien"
    CHARACTERISTIC_GENDER    "it"
    CHARACTERISTIC_ATTACK_SKILL 0.75
    CHARACTERISTIC_WEAPONWEIGHTS "bots/alien_w.c"
    CHARACTERISTIC_AIM_SKILL 0.5
    CHARACTERISTIC_CHAT_FILE  "bots/alien_t.c"
    CHARACTERISTIC_CHAT_NAME  "alien"

    ...

    CHARACTERISTIC_ITEMWEIGHTS "bots/alien_i.c"
    CHARACTERISTIC_AGGRESSION 0.95
    CHARACTERISTIC_SELFPRESERVATION 0.25
    CHARACTERISTIC_VENGEFULNESS 0.95
    CHARACTERISTIC_CAMPER 0.5
    CHARACTERISTIC_EASY_FRAGGER 0.1
    CHARACTERISTIC_ALERTNESS 0.5
}
```

alien/botfiles/bots/alien_t.c

```
chat "alien"
{
#include "teampplay.h"

type "game_enter"
{
    "You cannot hide in ", 4, ". This we like.";
    "We hunt. We kill. We feed. Queen happy.";
    "We see ", 1, "! Good sacrifice for Queen. Much joy.
    "There will be much meat in ", 4, ". You will all di

type "game_exit"
{
    "I go find Ripley.";
    "I go find some space marines to play with now.";
    4, " not good.";
    1, " is like newborn from pod.";
}
...
}
```

extending .pk3 archive

new files:

<name>/models/players/<name>/<name>.body

<name>/bot-<name>/botfiles/bots/<scriptname>

<name>/models/players/<name>/gibs/

- abdomen.md3 arm.md3 brain.md3
- fist.md3 foot.md3 forearm.md3
- gibs.jpg intestine.md3 leg.md3
- skull.md3

for example in alien we have:

alien/models/players/alien/alien.body

```
//
// extra body features
//
{
    BODY_NO_OF_SKULLS      3
    BODY_NO_OF_BRAINS      3
    BODY_NO_OF_ABDOMENS    16
    BODY_NO_OF_ARMS        4
    BODY_NO_OF_FISTS       4
    BODY_NO_OF_FEET        4
    BODY_NO_OF_FOREARMS    4
    BODY_NO_OF_INTESTINES  32
    BODY_NO_OF_LEGS        8
}
```

Tutorial

- make the model vince into a bot and give him:

BODY_NO_OF_INTESTINES	32
BODY_NO_OF_ABDOMENS	32

- and the appropriate number of other entities

Tutorial

- make a new legoman model copy (lego-red)
- which only has one colour and remove the blood from the lego gibs
 - restore the gibs to lego colours

Extending bot scripts to use Python

- for example examine bot-
legoman/botfiles/bots/legoman.py

bot-legoman/botfiles/bots/legoman.py

- ```
import botlib, time

id = botlib.bot("localhost", 7000)
print "hello world, python is alive in Quake 3"
id.defaults()
print "bot is now active!"
id.disableAI()
while True:
 print "trying to crouch"
 id.crouch()
 print "in crouch position"
 time.sleep(1)
 id.jump()
 print "in jump position"
 time.sleep(1)
 id.fire()
 print "fire"
 time.sleep(1)
```

## Python botlib

- examine the botlib code and see how the parameters are marshalled `python-bot/bot-legoman/botfiles/bots/botlib.py`