

# Sound and AI Examples

Jim Bruce

15-493 Spring 2004

Carnegie Mellon University

March 4, 2004

# A Quick Introduction to Sound

---



# What is Sound

---

- To a human, sound is what we hear
- Physically, sound is compression waves in air
- For speakers, a sound signal is a time varying voltage
- For computers, sound is discrete voltage level for a sequence of small evenly spaced units of time



# Sound Parameters

---

Digital sound has several parameters:

- Sample format
- Sampling rate
- Channels



# Sound Parameters

---

## Sample Format:

- type of integer used for voltage level
- unsigned 8 bit (common old format)
- signed 16 bit (common format now)
- 24 or 32 bit (ultra high-fi)
- above 8 bits, endianness matters
- sample files come in all formats



# Sound Parameters

---

Sampling rate:

- number of distinct voltage levels specified each second
- measured in cycles per second (Hz)
- 8000 Hz - phone quality
- 11025 Hz - old games (Quake 1)
- 22050 Hz - modern-ish games (Quake 3)
- 44100 Hz - CD quality (new games, what you should use)



# Sound Parameters

---

Number of channels:

- each channel normally corresponds to a speaker
- Mono - one channel
- Stereo - two channels
- Surround Sound - 5 or 6 channels



# Sound File Formats

---

- AIFF - uncompressed audio
- WAV - normally uncompressed, bastardized format
- MP3 - compressed format, low cpu usage, ubiquitous
- OGG - better compressed format, low-ish cpu usage
- WMA - another 2nd generation format, more cpu usage however
- use a libraries to load these files or you will be sorry
- mp3 note: non-free products now require a small royalty



# Types of Sound in Games

---

## Music / Cut scenes

- usually global, does not depend on location
- usually in stereo
- often depends on position in storyline
- too large to buffer decoded version in memory
- usually compressed (MP3/Ogg)



# Types of Sound in Games

---

Speech: like music, but sometimes small enough to buffer

Sound Effects

- usually small WAV samples
- can be stereo but often just mono (why?)



# Sound Libraries

---

- Lots of existing libraries to make sound programming easier
- FMod - free for noncommercial use, high level, ultra portable
- SDL - free, mid level
- DirectSound - free, Windows-only, mid level
- OpenAL - new spec, high level, portable



# Simple SDL Sound Wrapper

---

I've written a wrapper for SDL to make it very easy to use. Doesn't support advanced options, look elsewhere or add them yourself.

Library initialization, shutdown

- `Sound::open()`
- `Sound::close()`



# WAV Sample IO

---

Reading and deleting samples:

- `bool load(Sample &smp,const char *filename)`
  - return value: true iff sound loaded
  - `smp`: returned sample
  - `filename`: complete path to wav file
- `Sample::reset()`



# Playing a Sample

---

Playing a sample:

- `spid play(const Sample &smp,int loop,float voll,float volr)`
  - return value: identifier for playing sound
  - `smp`: the sample to play
  - `loop`: how many times to repeat the sample
  - `voll,volr`: right and left stereo volume, respectively



# Operations on Playing Samples

---

- `bool volume(spId id, float voll, float volr)`
- changes the stereo volume on an existing sound
- `bool stop(spId id)`
- stops a playing sound
- `bool busy(spId id)`
- tests if a sound is still playing



# Music

---

- `bool playMusic(const char *oggfilename,int loop,float vol)`
- `void setMusicVolume(float vol)`
- `void stopMusic()`
- `void fadeMusic(float fadetime)`
- `bool musicBusy()`



# Advanced AI

---



# Policies

---

A policy is a function from world state to action for an agent. In the same world state, it always does the same thing.

A generalized policy can include additional parameters. Parameters describe goals or intent from a higher layer.

Reactive AI systems are single policies.



# Beware of Buzzwords

---

The following methods are often poorly misunderstood

- Neural Networks
- Genetic Algorithms
- Fuzzy Logic



# Neural Networks

---

## Strengths:

- can deal with input noise
- don't need to know how to solve problem

## Weaknesses:

- need to know how to encode problem
- often need to tweak network structure
- very slow to learn
- solution is unintelligible
- not very fast to execute



# Genetic Algorithms

---

## Strengths:

- can learn almost anything
- don't need to know how to or represent problem

## Weaknesses:

- incredibly slow to learn
- often just learn how to “cheat”
- solution is unintelligible



# Fuzzy Logic

---

Good idea: Policies should not be binary functions

Bad: Just about every field already knew this

Fuzzy Logic is basically a synonym for the following:

- probabilistic algorithms
- real-valued control theory



# More Useful for Games

---

- Hand coded Policies
- FSMs
- Decision Trees
- Weighted Experts Learning



# Hand Coded Policies

---

- easy to underestimate the power of hacking
- in games, we know how the world works



# Examples: Lab3 Flaming Death

---



# Examples: XKobo

---



# Examples: SnakeBall

---



# Heavyweight FSMs

---

- If we have few agents, can specify FSM+policies in code
- Also useful to decouple external time with FSM steps
- Common method in real-time Robotics



# Agile FSMs

---

- Each state decides whether to run
- If it runs, executes a single policy
- If it does not run, chooses next state
- Easy to code
  - Several tests to switch to other states
  - Followed by code or function calls to return an action instead



# Building Big AIs

---

- can be viewed as a ladder of layers
- upper layers are slower, smarter, more abstract
- sensors go up, become more abstract
- actions go down, become more concrete
- can incorporate multiple agents
  - lower layers separate for each agent
  - at some height layers merge to form a team manager



# Examples: CMU RoboCup F180 Team

---



# Influence Maps

---

- entity histogram for localized metrics of map “control”
- Q: how can you use this for smarter path planning?
- Q: how can you use this for gaining “easy” new territory?



# Other Maps

---

- influence maps can count other things too
- map A: entity counts
- map B: entity hitpoints
- map C: entity attack damage

How to combine them to find the following:

- concentration of most vulnerable enemies
- least risky place to cause high damage

