Game Engine

1. Examples
2. Data Driven, Event Driven
3. Typical Architecture

Game Development Practicum

- This course is prerequisite.
- Counts as Capstone.
- Projects started here can be continued in that class

Game Engine

- Data Driven: The program is designed to parse data (files) to determine its core behaviour.
- Event Driven: The program is designed to process events (key board, mouse, etc.) to determine its actions.
- Event Driven $\subseteq$ Data Driven

Game Engine Examples

Traditional game engines

1. ioquake3

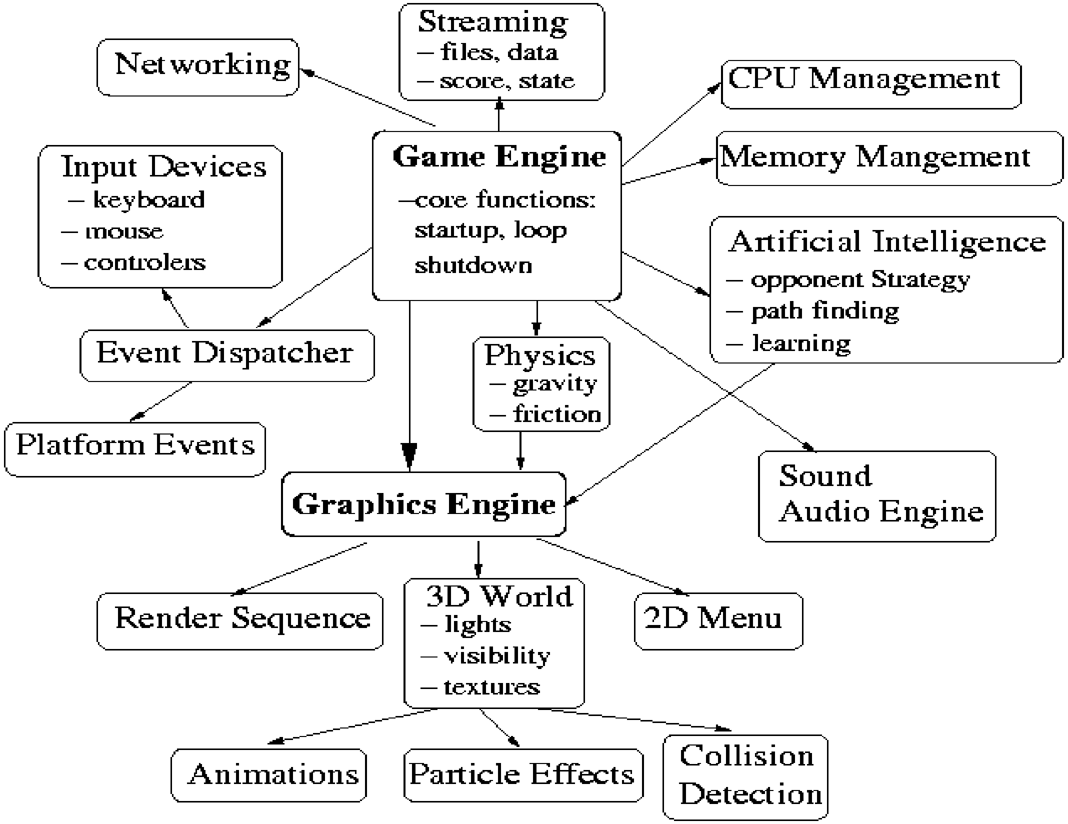
Commerical engines

2. TGE and TGEA
3. Unity

Open Source Partial Game Engines

4. opensim
5. ogre

Game Engine Components



Game Engine

- Game Engine: A Data Driven, Agent-Based, Graphical Computer Simulation.
  - Data Driven: The program is designed to parse data (files) to determine its core behaviour.
  - Event Driven: The program is designed to process events (key board, mouse, etc.) to determine its actions.) $\subseteq$ Data Driven
  - An agent is a system situated in an environment, which senses that environment, and acts on it, over time, in pursuit of its own agenda.

Data Driven Programming

- Files describe the features of the world
  - terrain, sky, buildings, water
  - different formats used for different types of objects
- Scripting used to handle interaction of objects in the world
  - written in a high level language designed to be executed by the game engine
  - usually compiled into byte code for faster execution
  - Think of byte code as an index into an array of pointers to functions.
- Mouse, key board, collision events.
- Agents: Bots and other AI.

Game Engine Programming

- Typically a large number of poorly documented classes. Abstractions frequently not well explained. You must learn to read the code.
- Very large software engineering project. To build a game you must apply software engineering principles,
- Must use an IDE (Integrated Development Environment—e.g. VC++ or Eclipse.

IDE Issues

- You must learn how to build a project in the IDE.
- IDE's have to be told where #include files and libraries reside. You must learn how to put that data in the IDE instance.
- Large libraries frequently reference files that are either missing or in unexpected locations. You need to have tools to search for such files on your machine and the skill to find missing files on the internet. When the missing or misplace file is a library and the only reference to that library is an undefined symbol, this can be difficult.

Compiling and Running Code

- You must learn to distinguish between compiler, linker, and runtime linker/loader errors. The first step in this process is to learn to read the error messages. Don't change code according to what you guess the error message means. Figure out what the error is before you change your code. Make a copy of your old code before you start changing it, so you can return to it later if necessary.
- You must know how to give dynamic linker/loader the location of your runtime libraries.
- See the Tale of J. Random Newbie, for a cautionary tale on what can go wrong.

Your Computer Environment

- You must be apply to download and install (sometimes compile) the class library at home or be willing to spend at least several hours a week in the Departmental labs doing your programming assignments.

Assignment 0

- Install an ogre sdk on a computer. See http://www.ogre3d.org/tikiwiki/Installing+the+Ogre+SDK

  The install can be on either a Window, Unix, or Mac, but note that wiki only hints on how to build the sdk on Unix or Mac machines. If you only have a unix or mac machine, see me.
- Install both, OgreSDK1.8 and OgreSDK1.9 in different directories.
- Run each of the sample programs provide with the package.
- If you have problems, bring your error messages to next class.