http://www.fuzzybinary.com/articles/IntroToArchitecture.ppt#256,1,Intro to Game Architecture

http://www.youtube.com/watch?v=aTUe8eGzow8&feature=related

Game Architecture

http://www.daftcartoons.co.uk/Cartoon%20Library/About/Architects.jpg

3D Game Engines: Architecture

D.H. Eberly: 3D Game Engine Architecture
J. Gregory: Game Engine Architecture

- Book presents the basics for developing an architecture
- Software snippets to quickly realize something

Software Architecture

http://msdn.microsoft.com/de-de/library/ee658098.aspx

Bass, Clements, and Kazman *Software Architecture in Practice (2nd edition)*

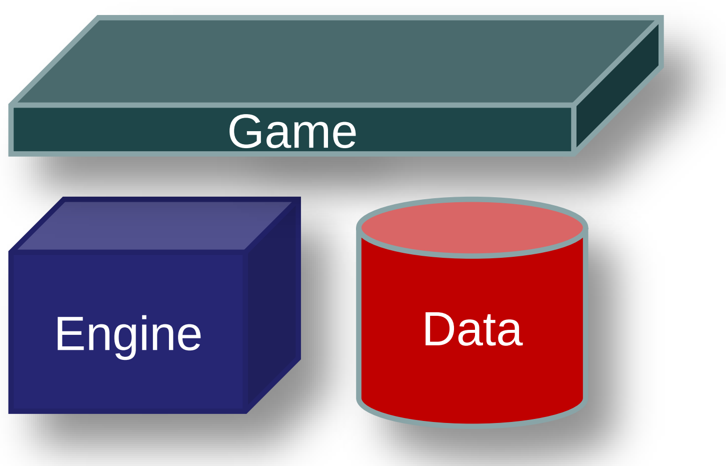
“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural.”

Architecture =
1. externally visible properties and
2. relationships between

elements of the software system

Motivation: Why Game Engines?

- Game Engines allow **simplifying** the development of games
  - **Typical routines and algorithms** are already available as library
  - Ideal case:
    - Include only data to finalize a game
    - Not achievable since often engines used for multiple genres and only routines that are shared by them are implemented
      - Each genre requires it‘s own supplement of routines



What is a Game Engine

http://en.wikipedia.org/wiki/List_of_game_engines

- Important issue for game development: attachment of
  - Development tools (Editors etc.)
  - Reusable software components („middleware“)
    - Data driven development – change only data, not code
- Platform independent
- Component based architecture
  - E.g. physics engines can be replaced by others (Havok vs. physX)

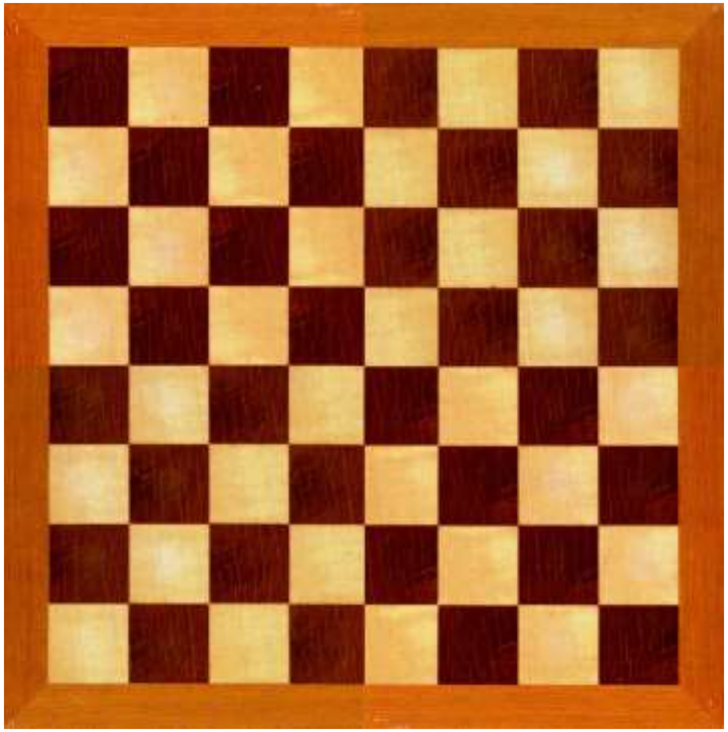
Game Engines

- Open Source
  - OGRE/Yake, Crystal Space, Irrlicht, The Nebula Device2 , Panda 3D, NeoAxis, Torque 3D
    - not always complete
  - Doom, Doom2, Quake, Quake2
    - older, complete
- Commercially
  - Doom 3, Quake 3, Half Life, Half Life 2, Unreal Tournament, Unreal Tournament 2004, Unreal 3.
    - contains all necessary features
    - Quake 3: source code open

Example: Chess

- Let us assume that we want to build a chess game
- What do we need?

Graphical Elements



http://www.sjgames.com/proteus/img/chessboard.jpg

http://www.chessncrafts.com/brass-chess-pieces/images/3466s1.jpg

Physics: Pieces Move and Collide

http://elder-geek.com/wp-content/uploads/2010/08/battle-vs-chess.jpg

AI: Select Best Next Move

http://thumbs.dreamstime.com/z/3d-chess-strategy-horse-12269770.jpg

How to Save Time

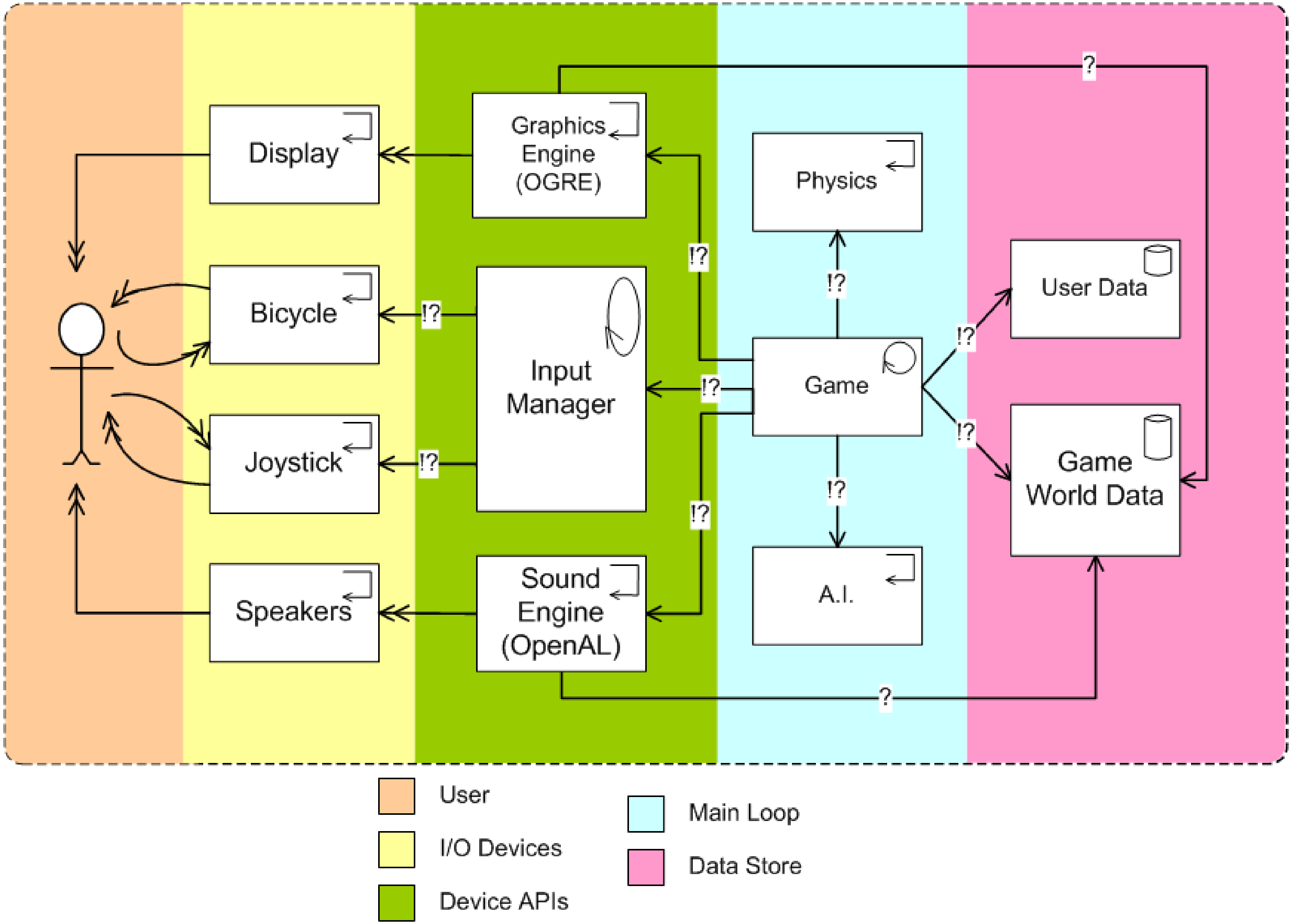
- Use available elements to build game
  - this is a game engine...
- .... and just program what is not yet there...

Elements

- what is behind….

http://1.bp.blogspot.com/-qvm_rlF45SI/TsMXq-6EjLI/AAAAAAAAAGw/1Jd9zRPf7Fg/s1600/behind-the-curtain.jpg http://stl.cs.queensu.ca/wiki/index.php/CAX_Game_Architecture

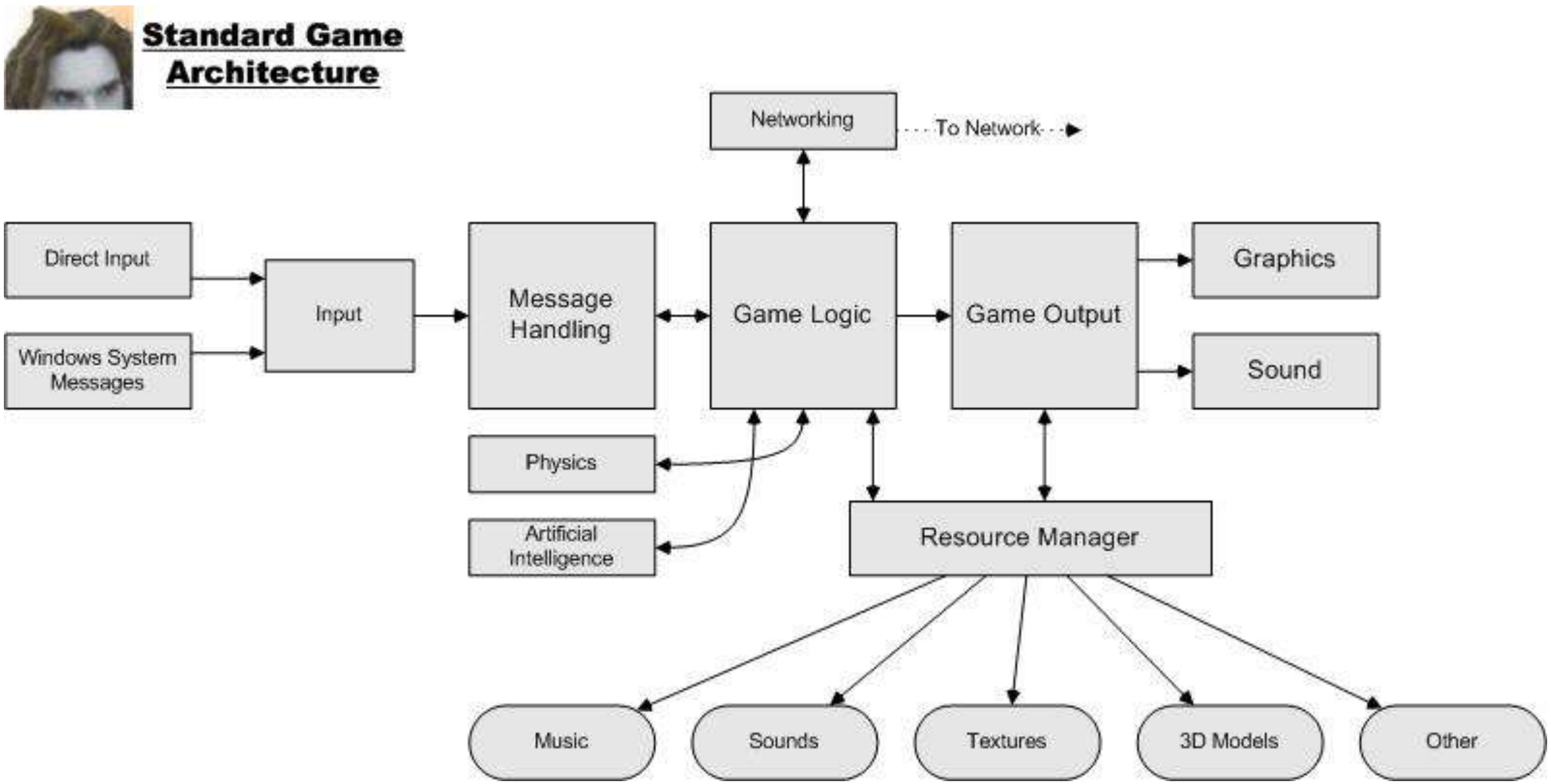
Example



Discussion

- What do we learn about this architecture?
  - Layered: access only to layers above and below
    - However, OGRE and OpenAL do not follow this line
    - Game: complex communication within the same layer
    - Advantage:
      - clear interfaces and responsibilities
      - reduced complexity
      - modular

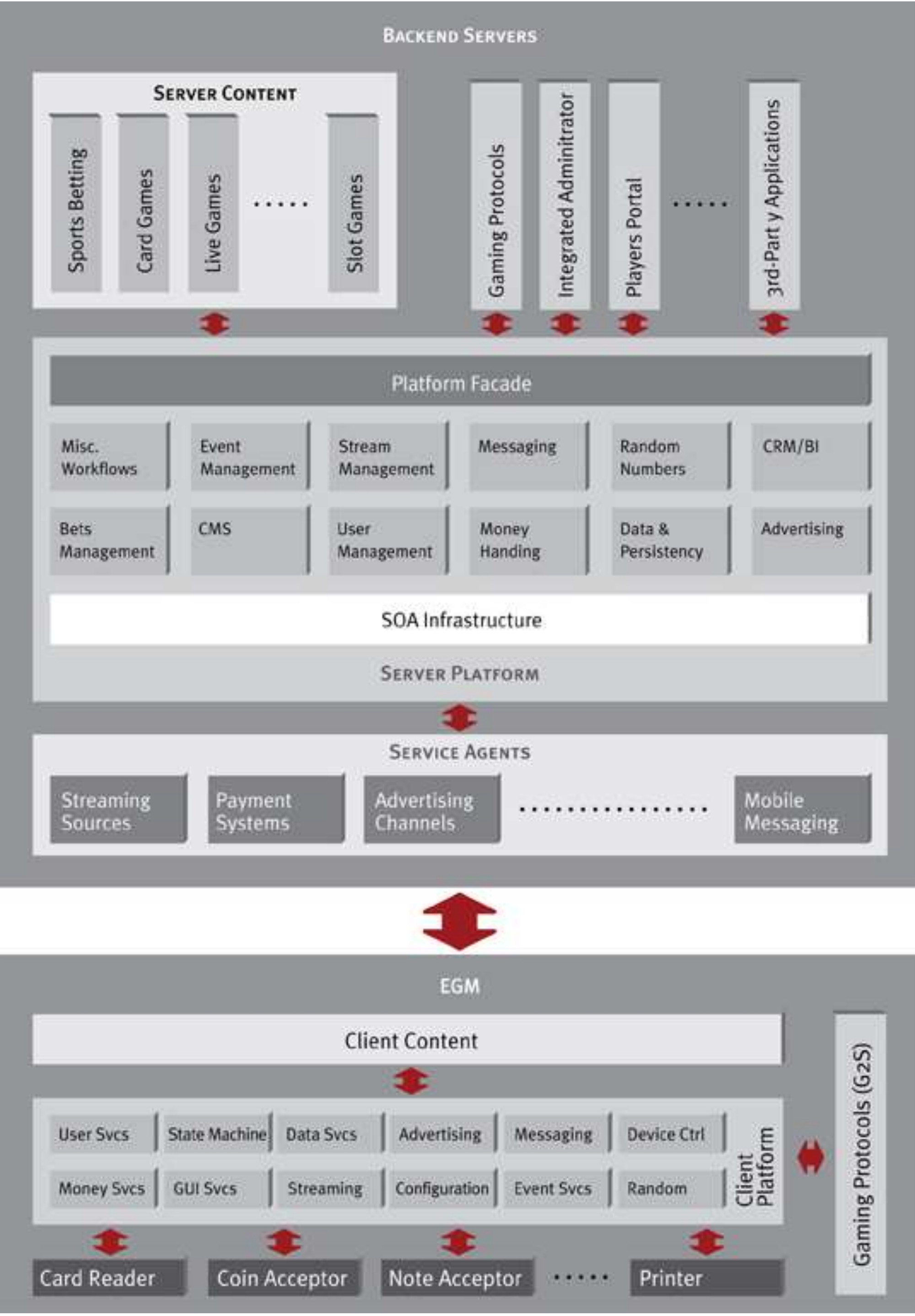
Example



http://www2.fiu.edu/~jmarr002/html/images/Diagrams/BasicGameArchitecture.jpg

Discussion

- Separation between input and output
  - different devices
- Game logic is now central node



http://gaming.comtrade.com/images/Gaming_Platform_Architecture.jpg

Example

more than a game, beyond our scope....

Discussion

- good Game Engines have the following properties
  - extremely modular and extendable
  - low complexity to that it can learned easily
  - contains all relevant functionality that is needed in a game (no extra algorithms are required)

Discussion: Flexibility

1. Game Engines are consisting of sub-engines that can be exchanged
   - Graphics
   - Physics
   - etc.

2. Game Engines use special classes that make
   - communication
   - resource control
   - configuration

   easier

3. Game Engines provide ways so that non-programmers can easily work with them
   - configuration files
   - scripting

Discussion: Simplicity

- Game Engines try to reduce the code that has to be provided for a new game to a minimum so that all essential functions are available
- Game Engines have a class structure that allows easy configuration of components
  - Discussion s. later

Internal Components

- Graphics
- Collision/Physics
- Animation/AI
    - Comprising Path Planning
- Audio/Video

- Additional elements:
  - Interfacing to input devices
  - Networking
  - Scripting

Game Engines

- Develop engines for as many games as possible but
  - Mostly restricted to a single genre – see first lecture where this was discussed
  - See list of games and engines
- Genre requirements
  - First person shooters
    - Fast rendering, physics based animation, AI
  - Platformers
    - Dynamic world, good animation, camera views
  - Fighting games
    - Animation database, accurate user input, character animation
  - Racing games
    - Level of detail rendering, rigid body physics (e.g. cars), evtl. deformations
  - Real-time strategy
    - Crowd simulation, evolving environment, AI

Game Engines can be characterized by views

- Architectural or structural view
- Functional view
- Implementation view
- User view

Architectural View

Functional/ View

- Functional view
  - Game Engine
    - s.a.
  - Game Logic
    - Scripts, Byte-Code, DLL
    - Controlling game
    - Content
    - Users can modify game on this level
  - Game Art

Implementation View

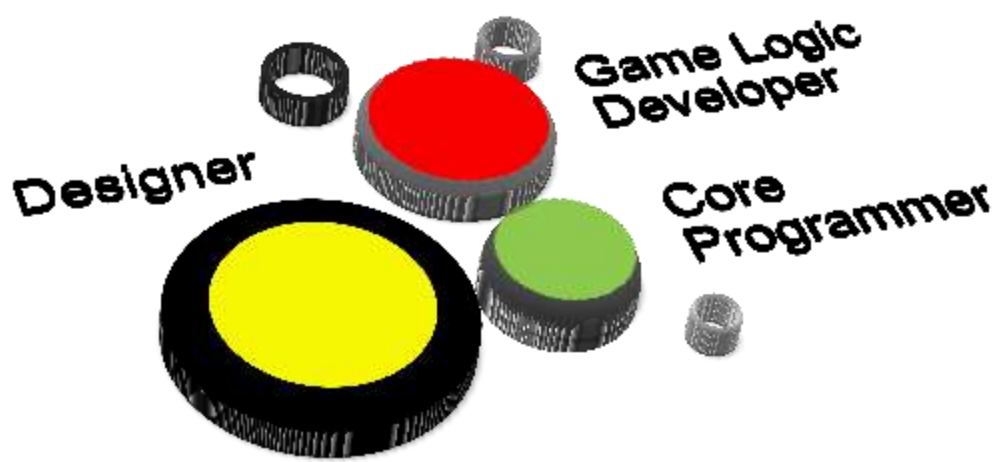
- Implementation View
- (http://www.cs.auckland.ac.nz/~burkhard/Reports/2005_S1_AndrewGits.pdf)
  - Game Engine (.exe)
  - Game Code (.dll) or script (.xx)
  - Game Content (media)

User View

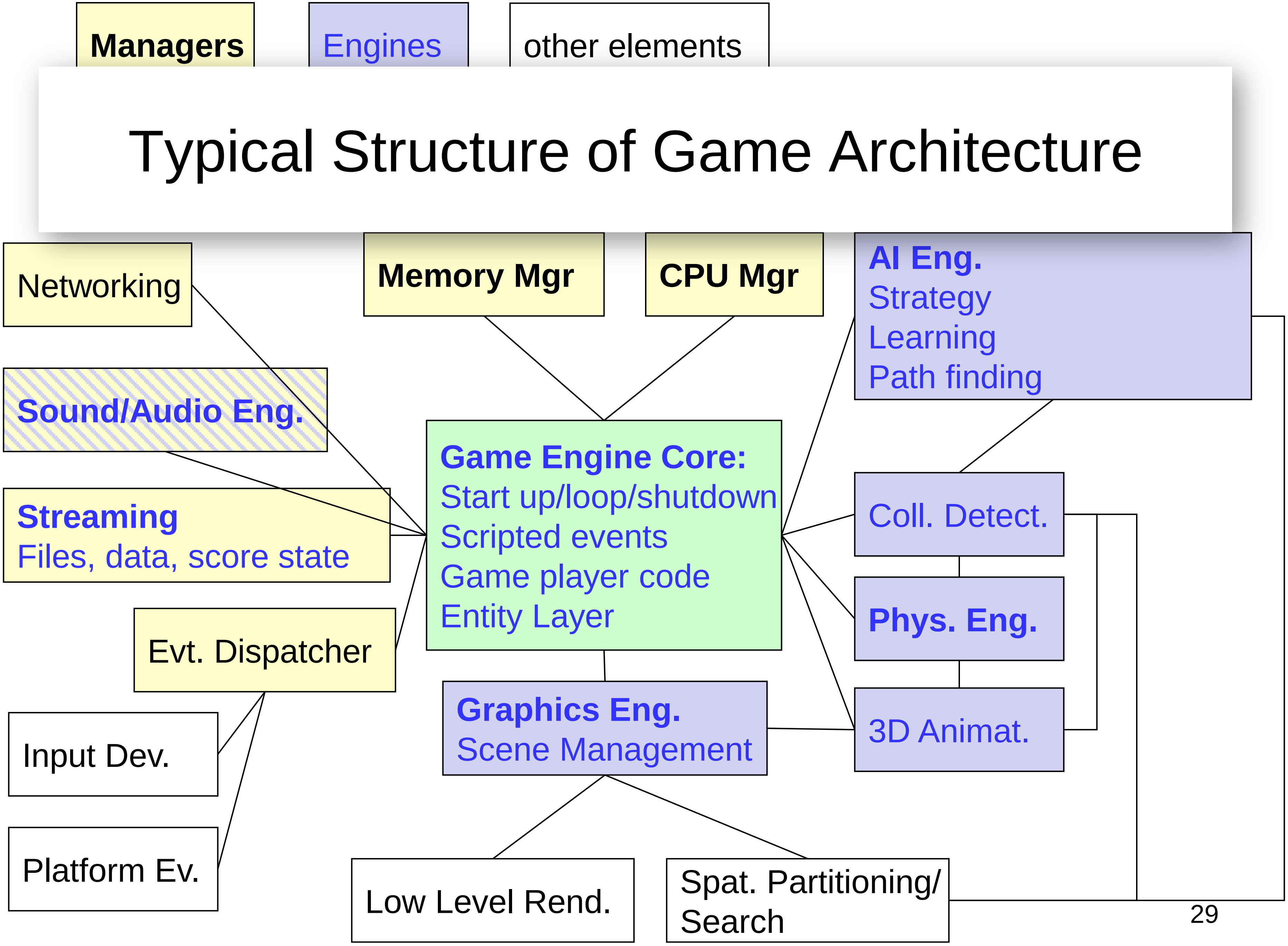
- Game Engine development
  - Engineers
- Game designers (small programming, scripting, script)
  - Game flow etc.
- Editors, graphics/animation etc.
  - Artists
- … producers, publishers and studios, other staff

Why is a computer game a good example of realizing complex software with little resources?

- Separation between
  - core code: mostly given by libraries (game engines)
  - scripted code: faster to develop than compiled code
  - data driven: can be done by users/designers



Typical Structure of Game Architecture

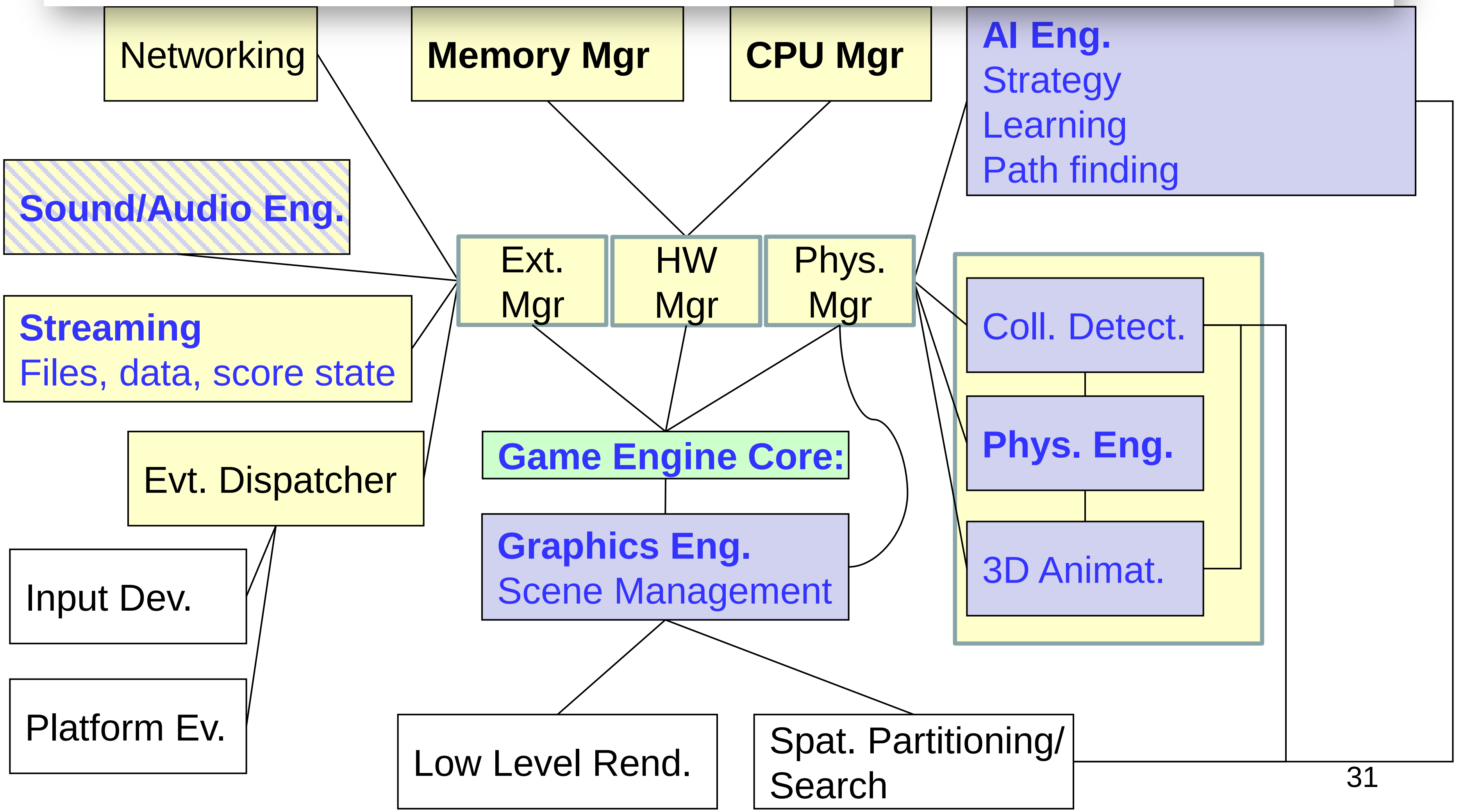


Discussing Architectures

- Software architecture describes
  - elements of the software system
  - how they are integrated
  - how they fulfill the requirements
- In computer games, we have the problem of increasing complexity
  - reducing complexity is one of the main goals
  - this can be achieved by reducing the number of dependencies between the elements

Managers
Engines
other elements

Improved


Networking
Memory Mgr
CPU Mgr
AI Eng.
Strategy
Learning
Path finding
Sound/Audio Eng.
Ext. Mgr
HW Mgr
Phys. Mgr
Streaming
Files, data, score state
Coll. Detect.
Phys. Eng.
3D Animat.
Game Engine Core:
Evt. Dispatcher
Graphics Eng.
Scene Management
Input Dev.
Platform Ev.
Low Level Rend.
Spat. Partitioning/
Search

Core Elements

I. Support Systems

II. Gameplay System

III. Runtime Gameplay Foundation Systems

IV. Architecture of Runtime Object Model

I. Support Systems

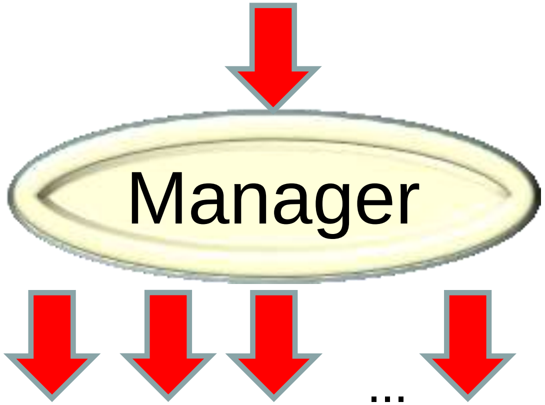
- Support systems allow to simplify routine tasks
- Allow to maximize quality by using structured elements that have been proven useful in other games

Support Systems

- Support Systems

  1. Manager Classes
  2. Controller Classes
  3. Message Passing System
  4. State Machines
  5. Mathematics System
  6. Scripting Engine
  7. Interfacing

1. Managers



- Role
  - coordinate parts and resources of game engine
- Pro
  - Hides complexity
    - multiple, different objects to be accessed but of no interest from game logic
    - complex handling of resources, internal part is highly specific and of no interest for the game logic
  - keeps flexibility
    - one location to change code if new objects are included or new management algorithms are implemented
- Con
  - additional indirection – costs a little performance
- Where:
  - coordinate resources and coordinate I/O elements

Manager-Class

Manager

…manage different objects and resources

- access over manager simplifies the structure of the game engine significantly!
- have typical singleton structure
  - exists only once in the program: centralized control
  - this guarantees that there is no conflict or misuse of managers
- for each type of resource in each case, a manager is responsible
  - in the general construction of a manager, the implementation of individual managers (derived) concentrate purely on their own issues.

Singleton Design Pattern

from: http://www.oodesign.com/singleton-pattern.html

- Ensure that only one instance of a class is created.
- Provide a global point of access to the object.

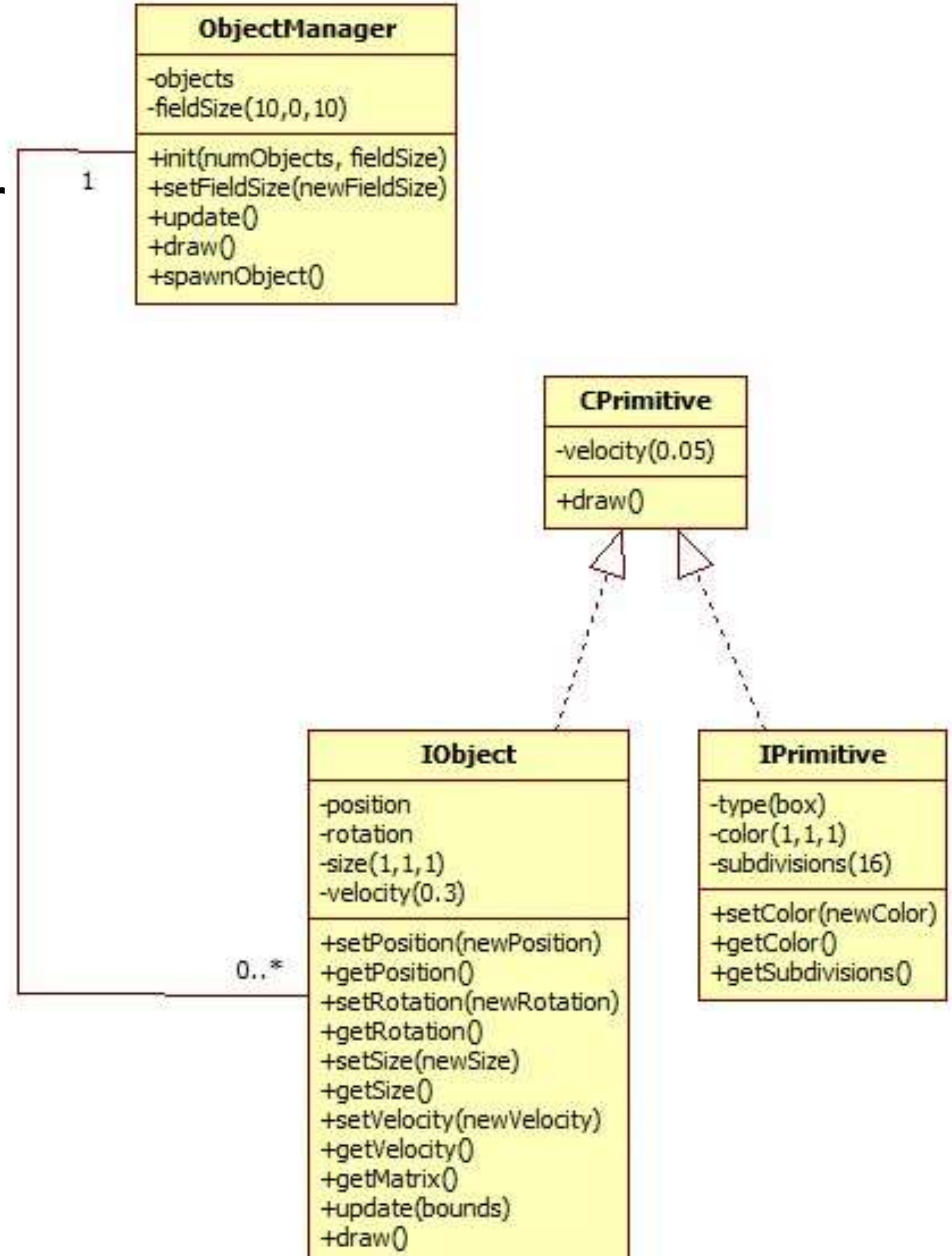
```
class Singleton
{
        private static Singleton instance;
        private Singleton()
        {
                ...
        }

        public static synchronized Singleton getInstance()
        {
                if (instance == null)
                        instance = new Singleton();

                return instance;
        }
        ...
        public void doSomething()
        {
                ...
        }
}
```

Example: Object Manager

one `ObjectManager` manages multiple `IObjects`

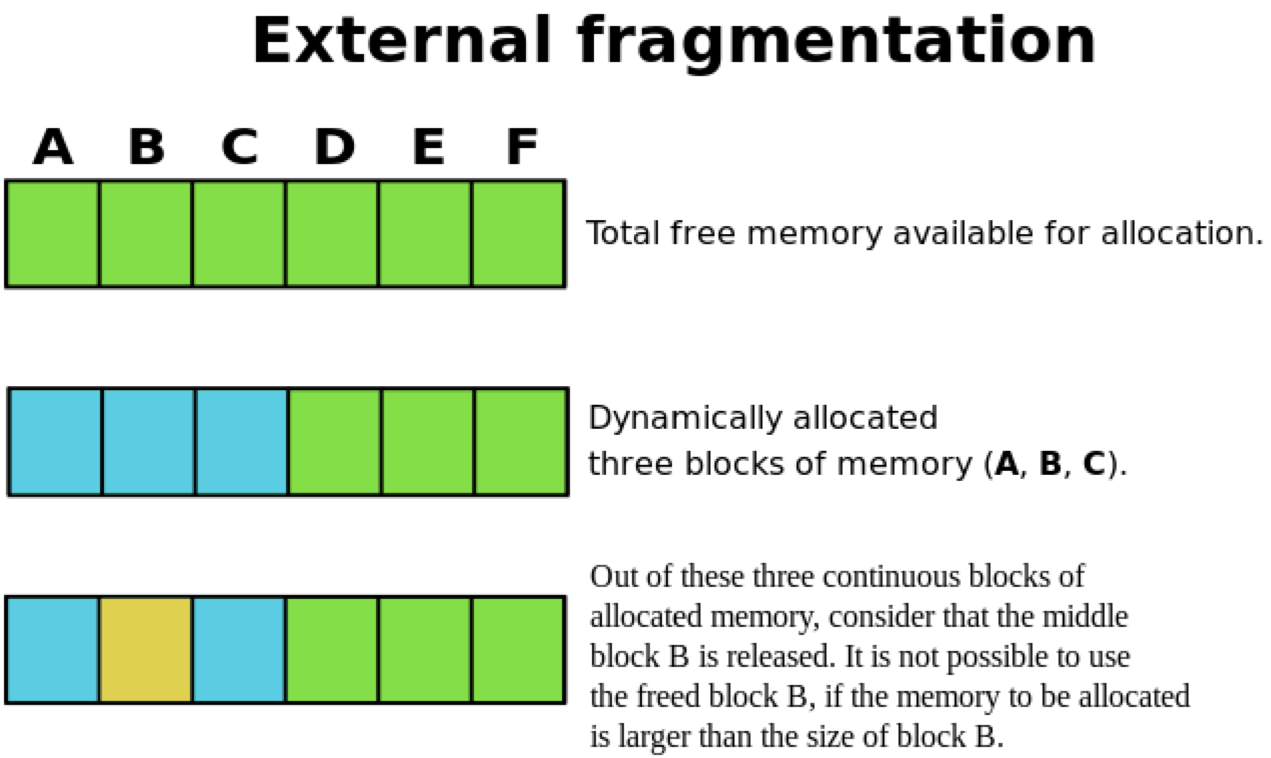


http://support.quest3d.com/index.php?title=Tutorials/Object_Oriented_Programming/Manager_Pattern

Example: Memory Manager

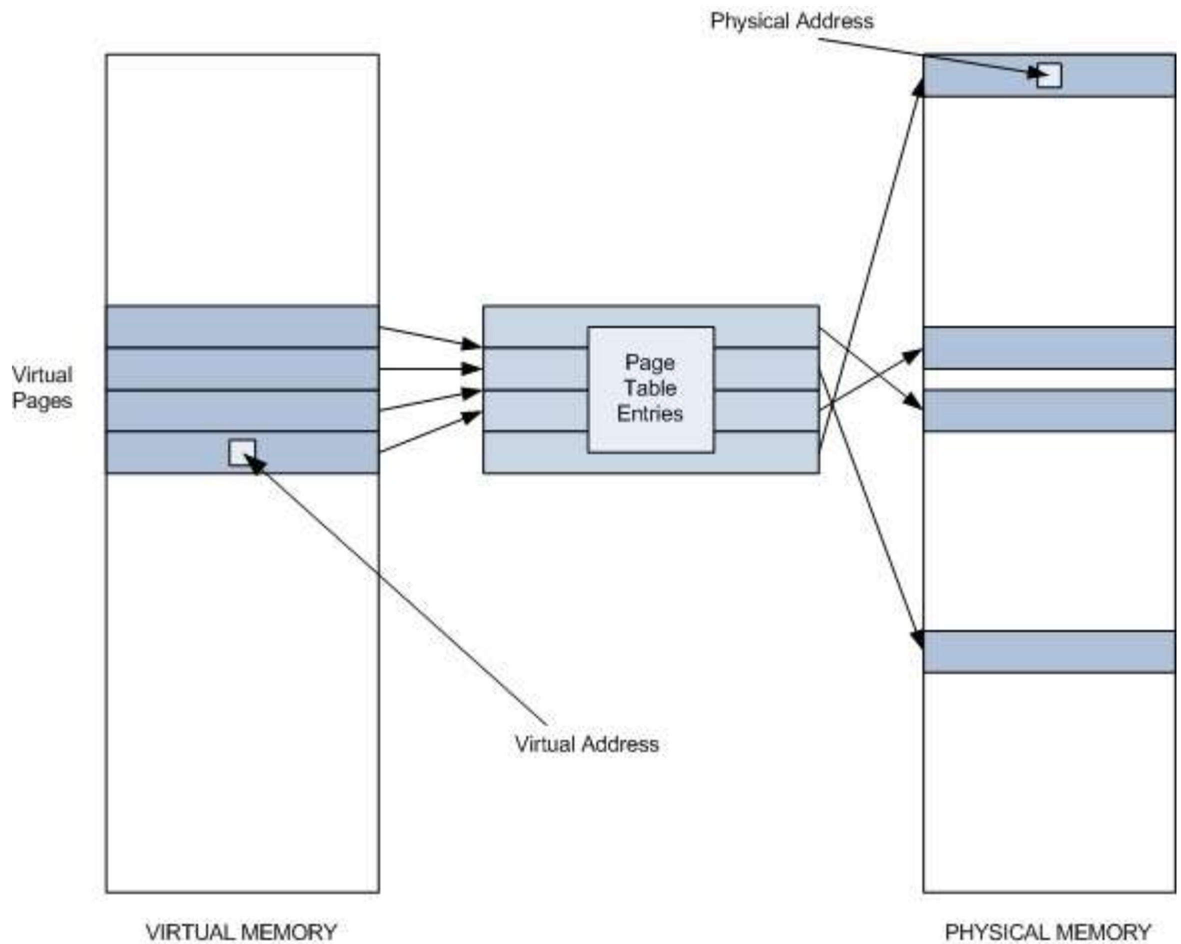
s.a. http://gamasutra.com/blogs/StewartLynch/20130520/192576/Anatomy_of_a_Memory_Manager.php

- Problem 1:
  - Request to allocate large block in memory
  - Problem: memory fragmented by allocation/free



Example: Memory Manager

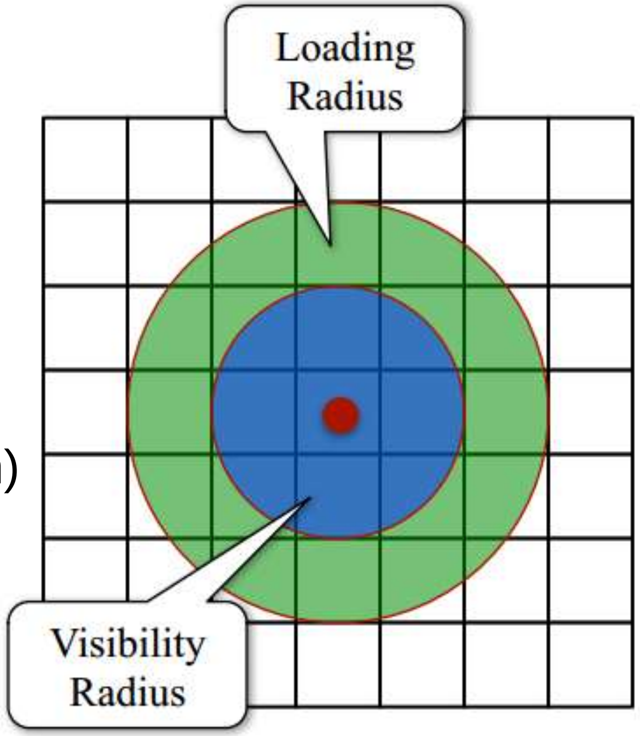
- Solution: use virtual instead of physical memory and reallocate entries



Example: Memory Manager

http://www.cs.cornell.edu/courses/CS4152/2013sp/sessions/09-MemoryManagement.pdf

- Problem 2:
  - not enough free space for allocation
- Solution:
  - dynamic loading
    - keep only necessary data on main memory and push rest on hard disk
      - on demand
      - invisible for the request (allocation)
    - games:
      - often spatial coherency used
      - depends on game objects (game-specific)



2. Controllers

- are discussed when handling animation tasks..
  - they are responsible to handle animations
    - different animation techniques might be used
    - the user is selecting a certain type of animation and all non-necessary complexity behind the animation technique is hidden to simplify the interface to the user
  - animation techniques can thus be easily replaced by other, more realistic ones without code change

Physics/Collision/AI

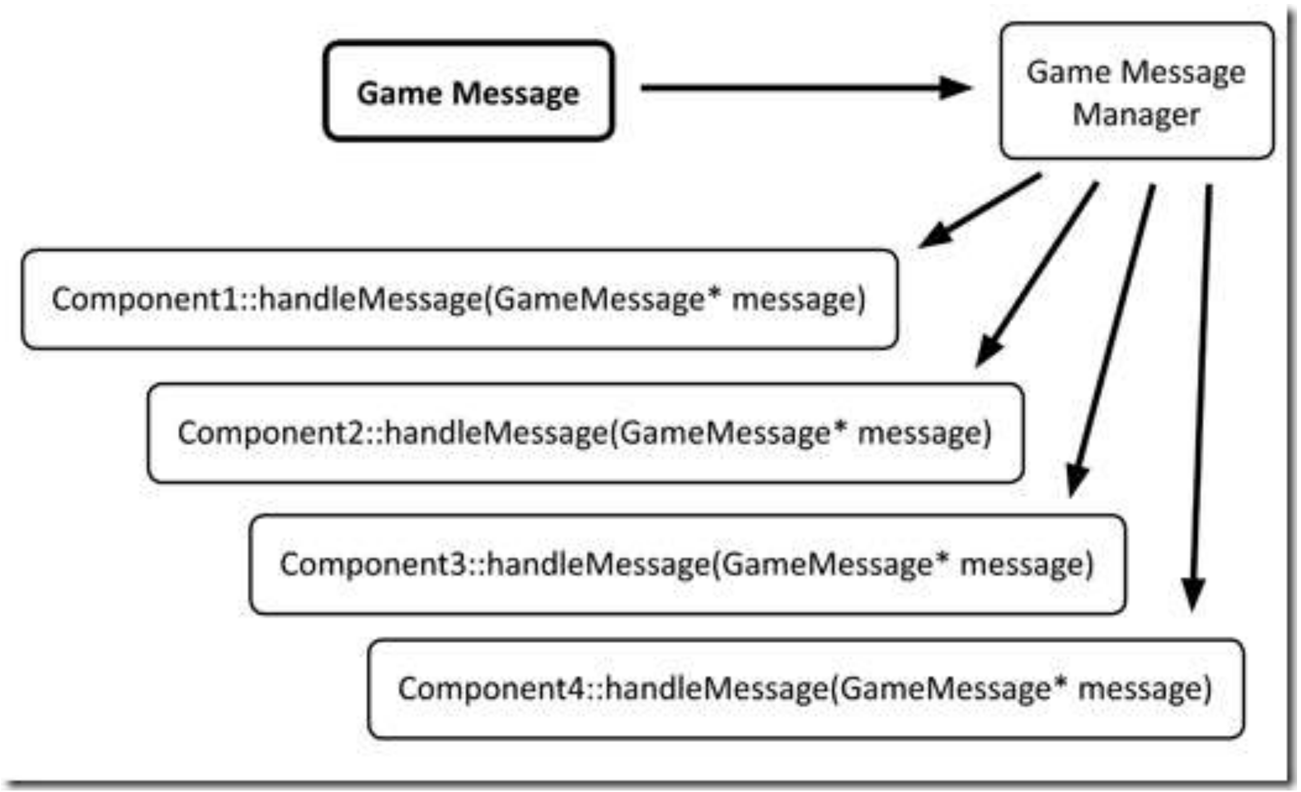
- managed by manager classes
  - used different algorithms depending on the situation
    - the manager has the task of selecting the appropriate algorithms
  - AI: different behaviors
    - be implemented by controllers
      - Replacing the controller enables a variety of behaviors
      - Controllers can be reused (details later)
        - » Factory pattern

Example

- Physics simulation
  - simple parametric models
  - physical simulation with simple solvers
  - physical simulation with complex solvers
- A manager manages the controller, via keywords, you can select the various controllers and assign a default controller
- All controllers have the same interface to the outside so that they are interchangeable

3. Messaging System

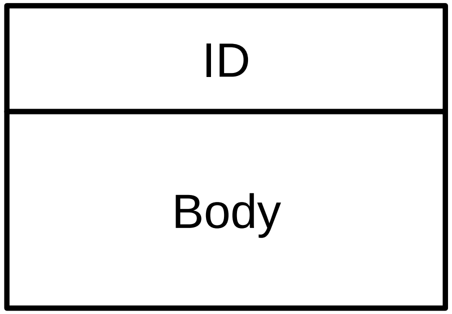
- Idea
  - coordinate communication between game objects



http://allenwp.com/blog/2012/09/22/two-game-messaging-systems-using-observer-and-visitor-patterns/

Messaging System

- Message



- functions to be implemented:
  - sendMsg() – direct communication
  - a message board where to store messages and where to pick up messages
  - receiveMsg() – polling for new messages
  - objects are to be included into a listener queue or removed from this queue
    - can listen specific objects for broadcasts

Example

http://www.randygaul.net/2013/09/02/powerful-c-messaging/

- receivers = player->nearbyObject();
- receivers->sendMsg(“Action“);

- person::receiveMsg(str) { if str = “Action“ do_anything(); }

Example: Message Listeners without polling

- objectA->subscribe(objectB, “MsgType“, functionPtr);
- objectB->postMsg(“MsgType“);
- ListenerList l = objectB->getListeners();

```
for(auto *begin = l; l=l->next)
        if l->ID = “MsgType“  l->functionPtr();
```

Messaging System

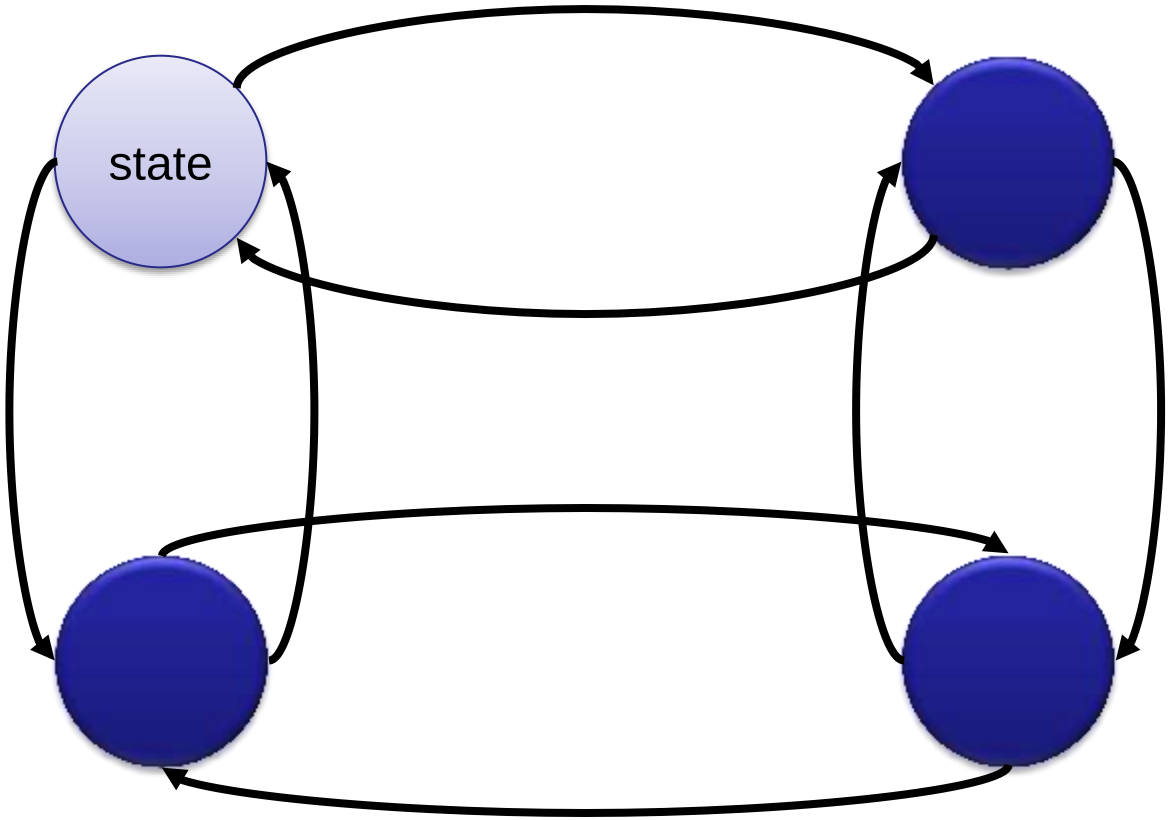
- objects can communicate in order to exchange their states by sending messages
- with N objects and full communication, we would have NxN different communication channels that are hard to control
  - better: have a centralized instance that is handling the message exchange
    - easy control who is sending what
    - easy modification of messages
    - manages the case where recipient is not available or not existing
    - blocks unnecessary messages if system is busy
- has the role a post office has as well

4. State Machines

- State machines are tools that allow to give characters a set of properties that might change over environmental variations
- The states are well defined and one can easily describe and characterize different objects by using only one code and different data for these state machines
- **State machines are often the nucleus of AI in games.**

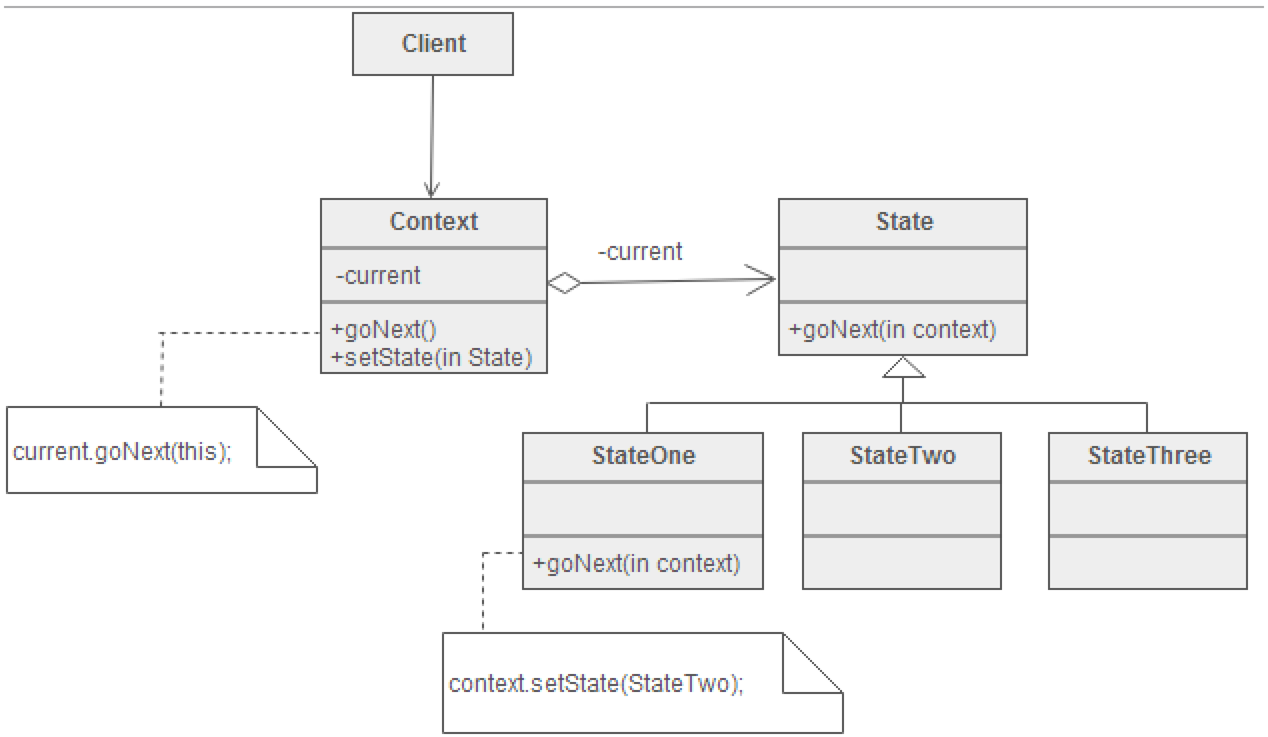
State Machines

transition – depends on event



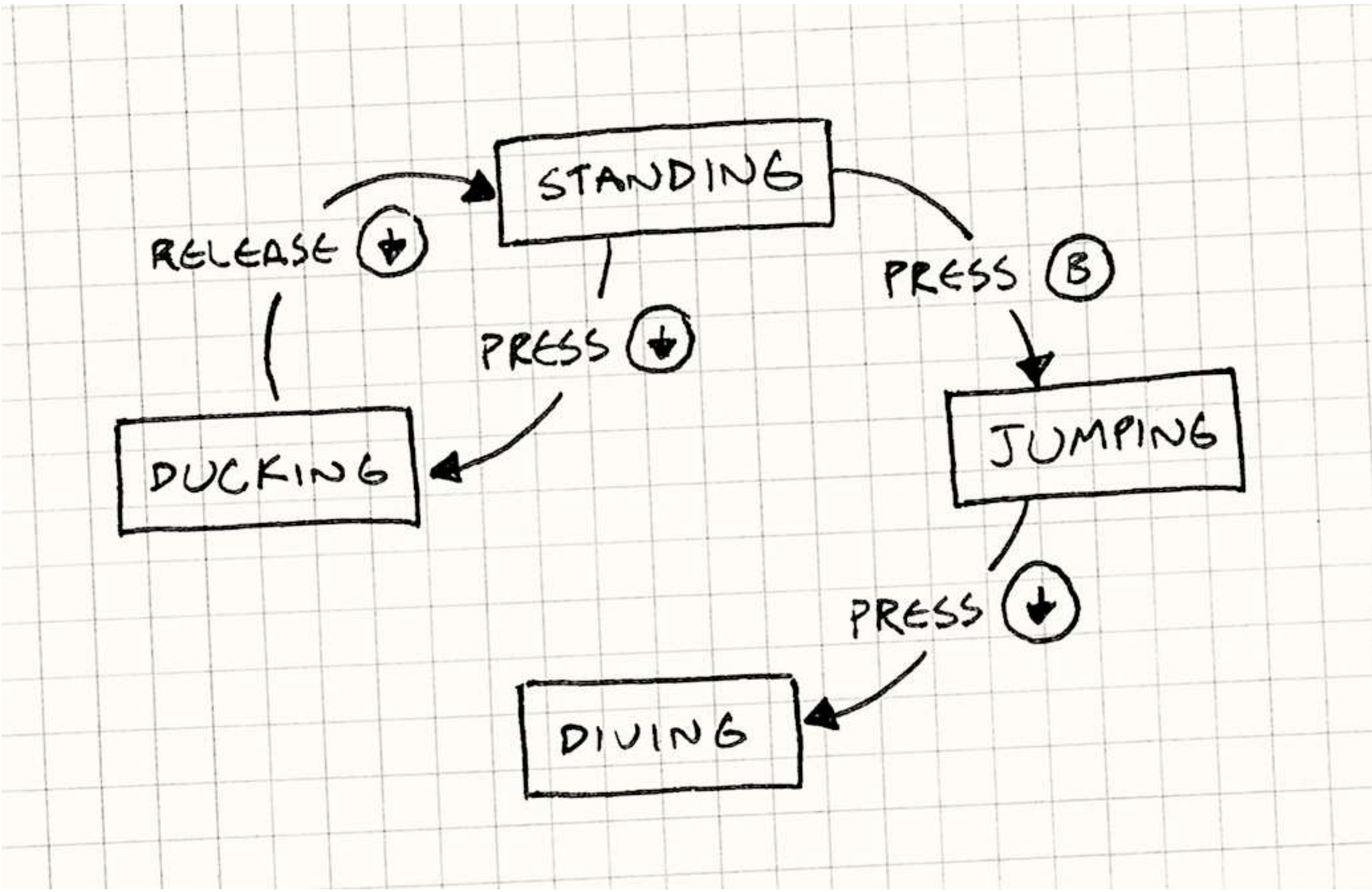
Design of State Machines

http://sourcemaking.com/files/v2/content/patterns/State1.svg



Example

http://gameprogrammingpatterns.com/state.html



5. Mathematics System

- Role
  - provide level of functions that are fast for the given hardware
    - partially parallel code
    - efficient code
    - shortcuts for higher speed at cost of accuracy

Mathematics-System

- Contains constants and optimized functions
- Examples
  - ACos(Real)
  - ATan2(Real)
  - Cos(Real)
  - InvSqrt(Real)
  - ...
- Constants
  - Let the work be done by the compiler
    - PI = (float) (4.0 * atan(1.0));

Mathematics-System

- Fast functions
  - $x \in [0,\pi/2]$: $\mathrm{Sin}(x) \approx x - 0.16605\, x^3 + 0.00761\, x^5$; error: $< 1.7\ 10^{-4}$
  - fast sine: $\mathrm{fSin}(x) \approx x - 0.166666664\, x^3 + 0.0083333315\, x^5$

    $- 0.0001984090\, x^7 + 0.00000027526\, x^9 - 0.0000000239\, x^{11}$; error: $< 1.7\ 10^{-8}$
- Where to get them?
  - Newton iteration formula (additive)
- Alternatives
  - Look-Up-Tables (LUTs)
    - Only reasonable if functions are complex
    - Interpolation between LUT-entries

6. Scripting Engine

- Scripting is a very essential part of game engines
  - Scripts are realized in interpreted languages
    - Python, Lua, QtScript etc.
  - Interpreted languages are quicker to write than compiled languages like C++
    - they are ideally suited for game specific modifications like controlling animation, physics, collisions, messages etc.
    - all modern games use scripts
    - this is found outside the field of games as well in an increasing amount of cases since it speeds up configuration of software to demands of different customers

Engine+Script

Engines:

- Graphics: rendering, ...
- Physics: dynamics, collision, ...
- AI: pathfinding algorithm, behavior pattern, ...

Script:

- Graphics: add/remove light, load...
- Physics: assign objects mass, collisions, ...
- AI: select computed paths, perform decision making, ...

7. Interfacing

- Interfaces to standard development tools
  - Modeling
    - 3DS Max
    - Maya
    - Blender
    - Softimage
  - Audio
    - Sound Forge, Audacity
  - World editor
    - Radiant, Hammer, UnrealEd

http://ui15.gamespot.com/2254/speed2007072011385278_2.jpg

II. Gameplay Systems

s.a. J. Gregory: Game engine architecture. A.K. Peters, Natrick, MA

Up to now: tools and techniques that are basic elements of a game

Now: the mechanics that are specific for games

- Vary much between different types of games

World Elements

- Virtual Game World
  - Static elements
    - Terrain
    - Buildings
    - Roads
    - Bridges
    - …static structures

http://www.sandagames.co.uk/images/war-game-terrain.jpg

http://redjak.com/Terrain/MVC-011F.jpg

World Elements

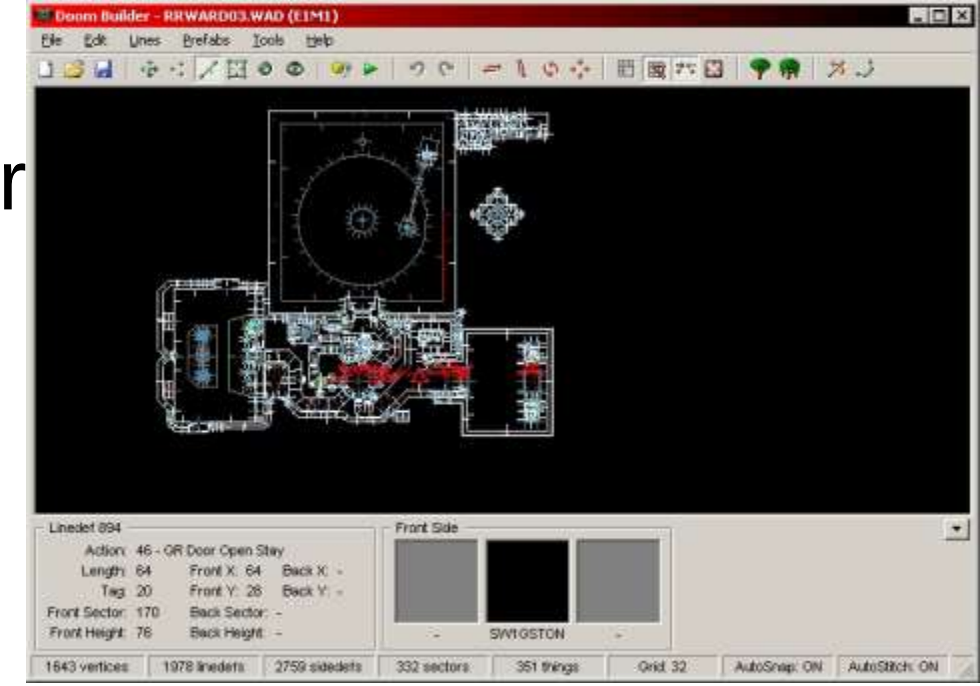
- Dynamic elements
  - Characters
  - Vehicles
  - Weaponry
  - Power-ups and health packs
  - Collectible objects
  - Particle emitters, dynamic lights
  - Invisible regions or splines for paths

http://www.gameon.co.uk/files/images/games/w/WET/arena_01.jpg

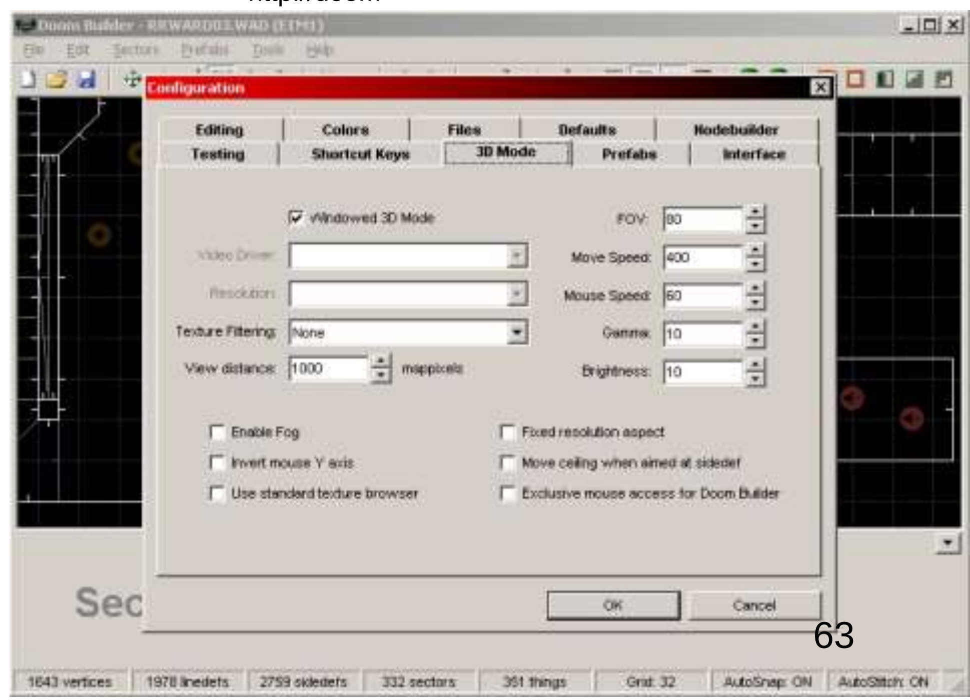
http://media.photobucket.com/image/dynamic%20elements%20game/Acuteboy/screen00981.jpg

Game World Editor

- Data driven Models for asset-creation
  - Maya, Photoshop, Havok content tools
  - Generate individual assets
- Analog: game world editor
  - Permits game world chunks to be defined and enriched with dynamic elements
    - Radiant (Quake, Doom)
    - Hammer (Valve‘s source)
  - Permits definition of initial states of game objects (values of their attributes)
  - Control on behaviors of the dynamic objects: configuration parameters

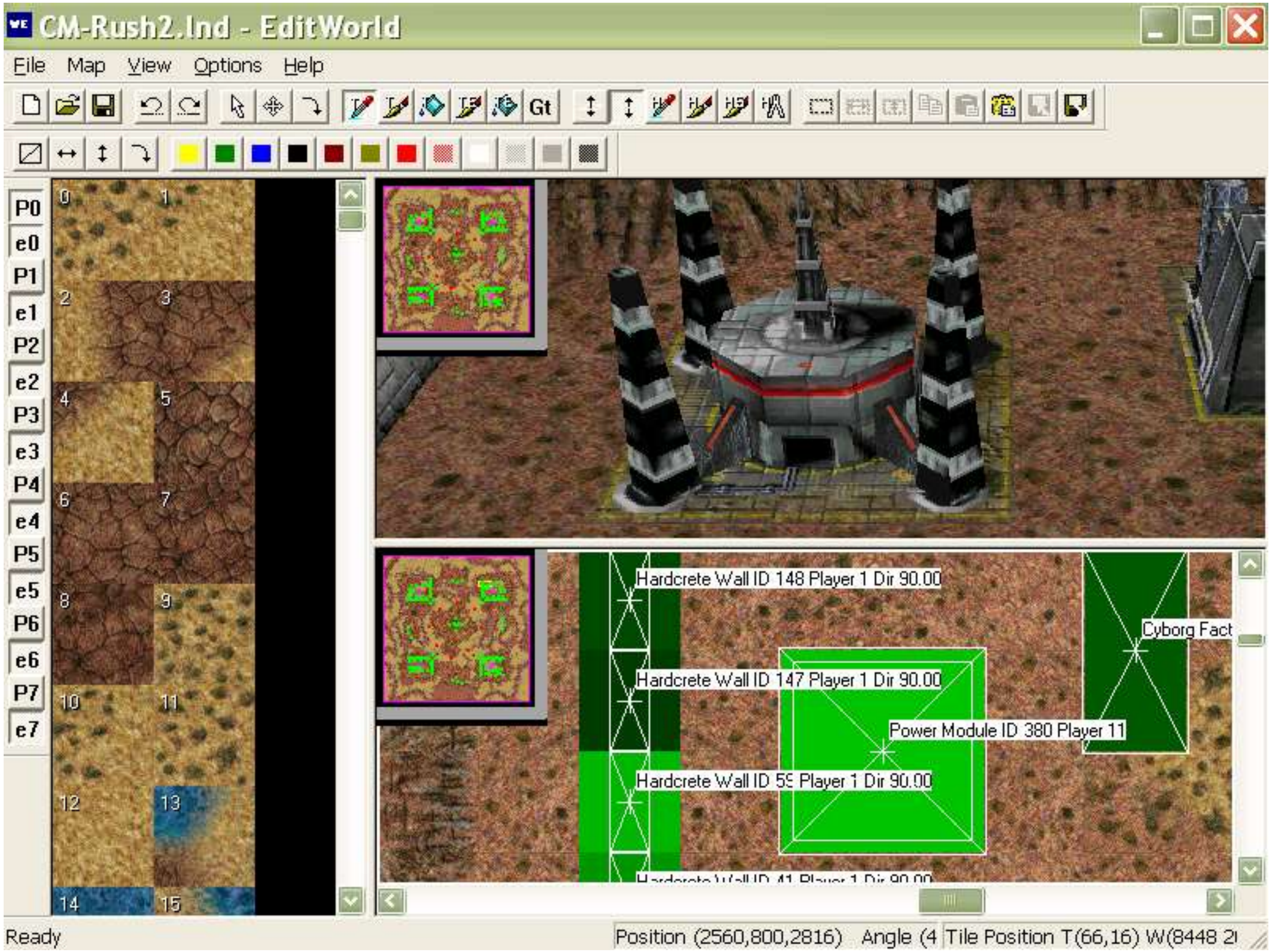


http://doom-



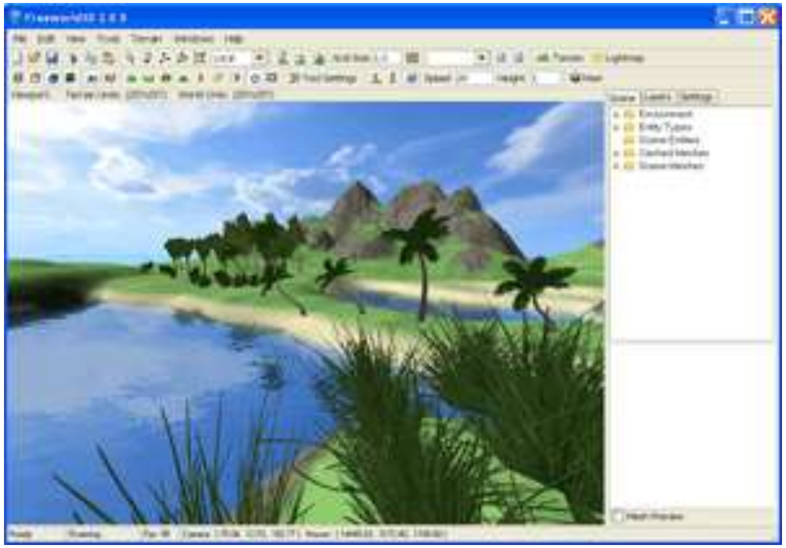
http://doom-builder.software.informer.com/screenshot/86436/

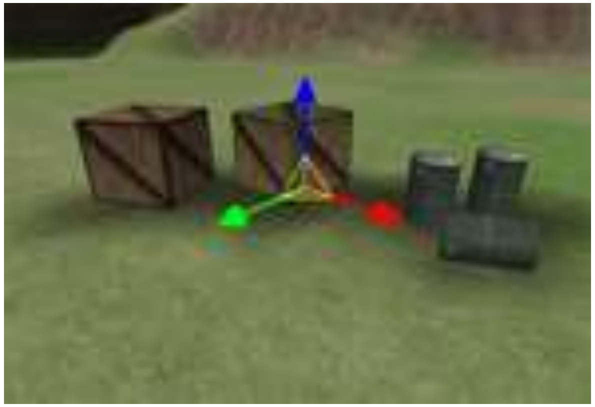
Example



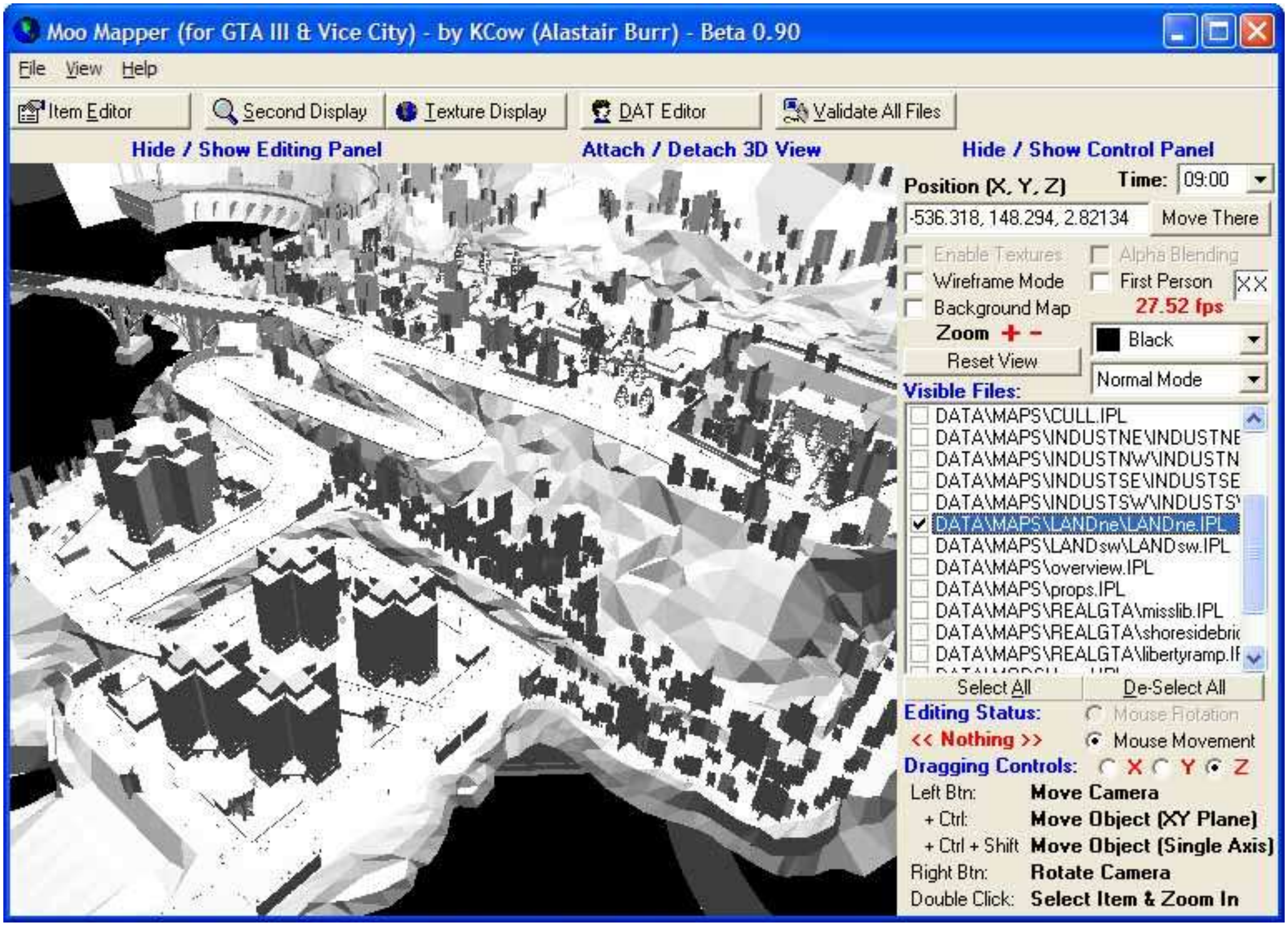
http://de.wikipedia.org/wiki/Karteneditor

Example http://freeworld3d.org/



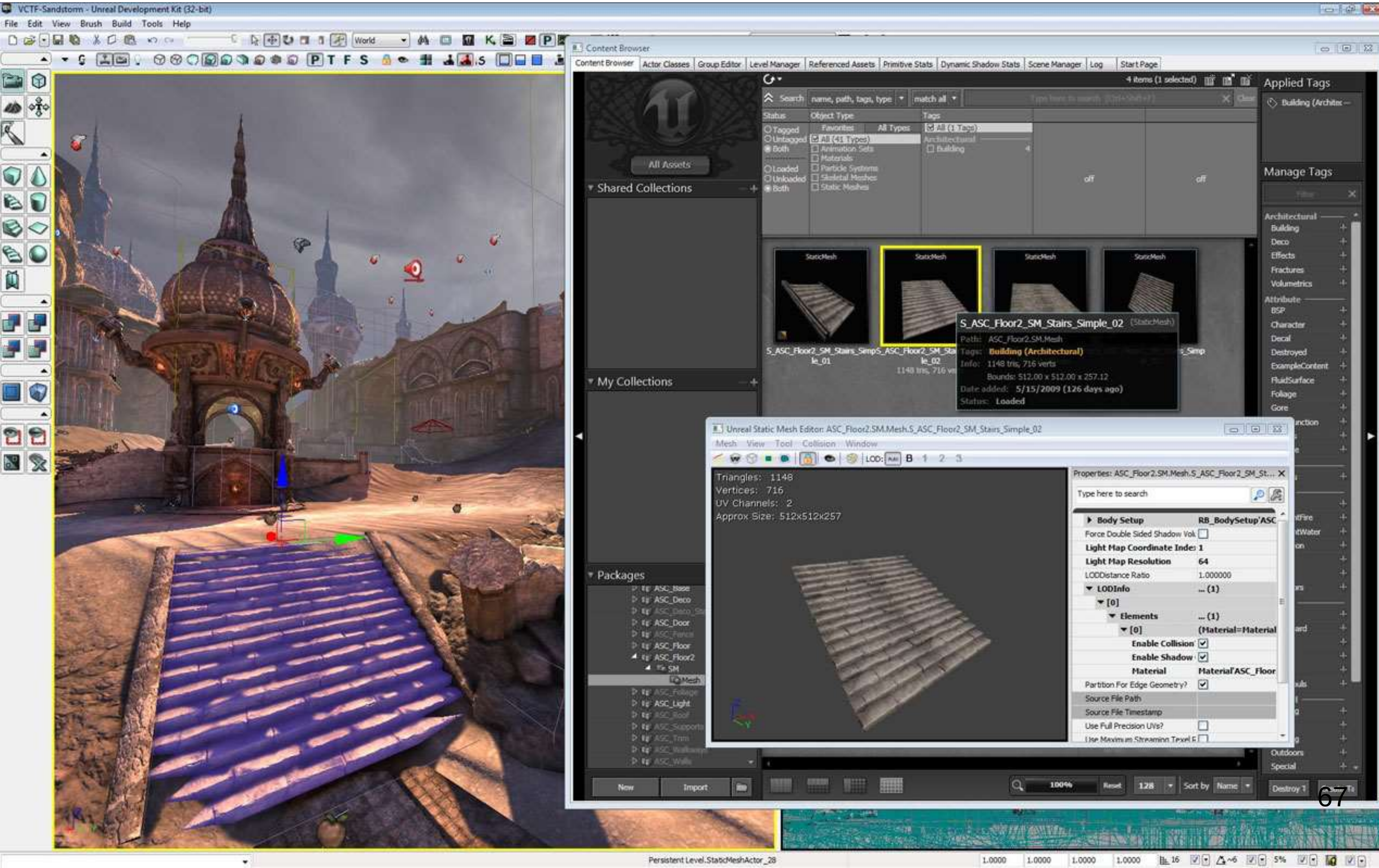


Example


Moo Mapper (for GTA III & Vice City) - by KCow (Alastair Burr) - Beta 0.90
File
View
Help
Item Editor
Second Display
Texture Display
DAT Editor
Validate All Files
Hide / Show Editing Panel
Attach / Detach 3D View
Hide / Show Control Panel
Position (X, Y, Z)
Time: 09:00
-536.318, 148.294, 2.82134
Move There
Enable Textures
Alpha Blending
Wireframe Mode
First Person
Background Map
27.52 fps
Zoom + -
Black
Reset View
Normal Mode
Visible Files:
DATA\MAPS\CULL.IPL
DATA\MAPS\LANDne\LANDne.IPL
DATA\MAPS\LANDsw\LANDsw.IPL
DATA\MAPS\overview.IPL
DATA\MAPS\props.IPL
DATA\MAPS\REALGTA\misslib.IPL
Select All
De-Select All
Editing Status:
Mouse Rotation
<< Nothing >>
Mouse Movement
Dragging Controls: X Y Z
Left Btn: Move Camera
+ Ctrl: Move Object (XY Plane)
+ Ctrl + Shift Move Object (Single Axis)
Right Btn: Rotate Camera
Double Click: Select Item & Zoom In

http://www.unrealed.info/

Example



III. Runtime Gameplay Foundation Systems

- Runtime software with game engine
  - Level management: load and unload contents (streaming)
  - Real-time model update
  - Messaging and event handling
  - Scripting
  - Objectives and game flow management
    - Organization of all game flow objects
      - Current status of player
      - Updates etc.
  - Runtime object model
    - Spawning/destroying game objects dynamically
    - Link to low-level engine and simulation
    - Define new game object types: by XML files
    - Unique object ids (handle objects over ids), handle queries and references
    - Support finite state machine
    - Saving/loading game

General Comment on Pointers in Configuration Files

- Do not use pointers but id‘s
  - Use a table to convert id‘s to pointers
- Pro
  - Pointers may be changed during a game to tidy up the heap (make space for new objects)
    - Classical solution: handles (pointers to pointers)
    - Games: id‘s: if pointers are changed, just the table is updated, not each object pointer!
  - Id‘s are short (less memory, lightweight)
- Con
  - Extra indirect step

Id	pointer
0	403840
1	5958751
2	710727
3	1794240
4	5796120
5	2936571
6	621904
7	9325604

IV. Architecture of Runtime Object Model

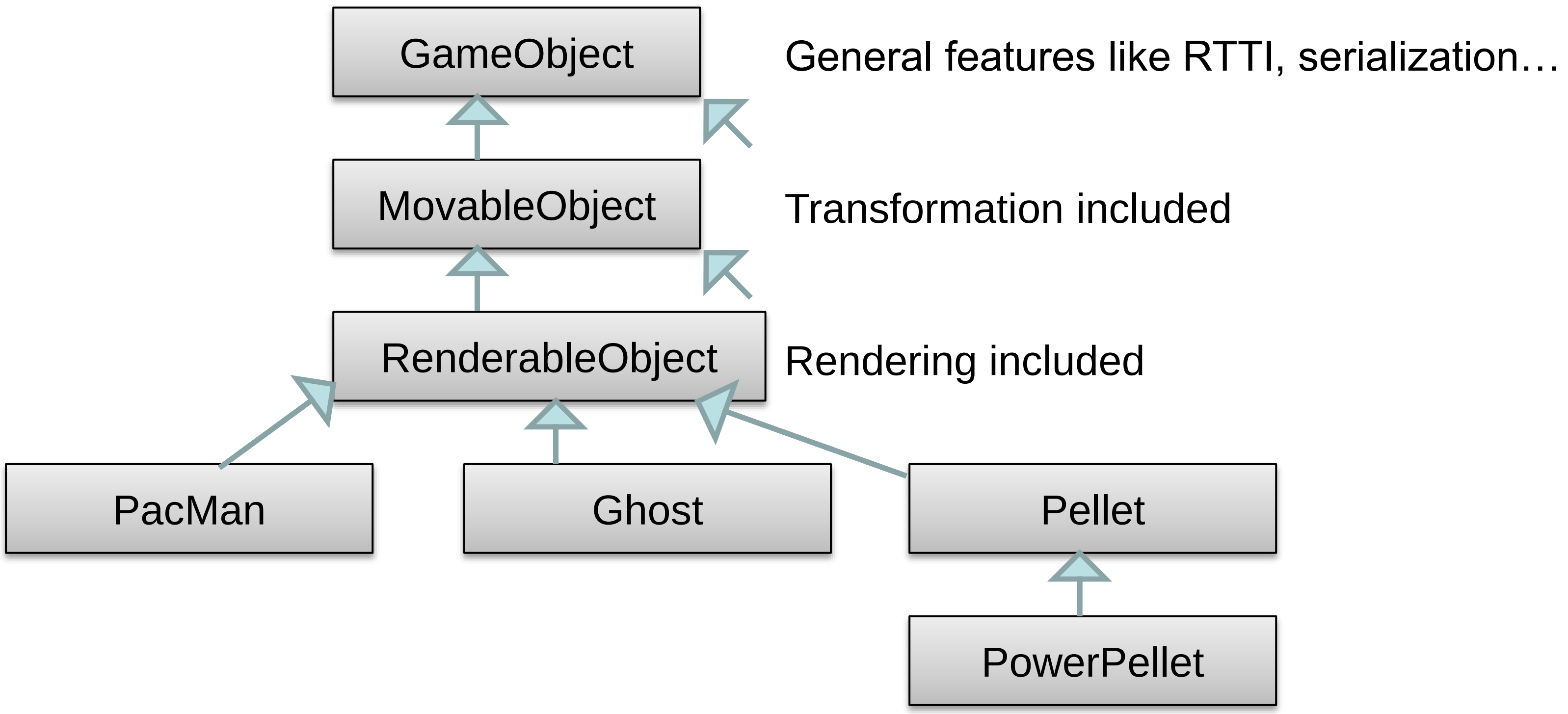
- Object centric:
  - represented as single class instance with attributes and behaviors:

    Class Pointer
- Property centric:
  - represented by ID

    properties are in data tables

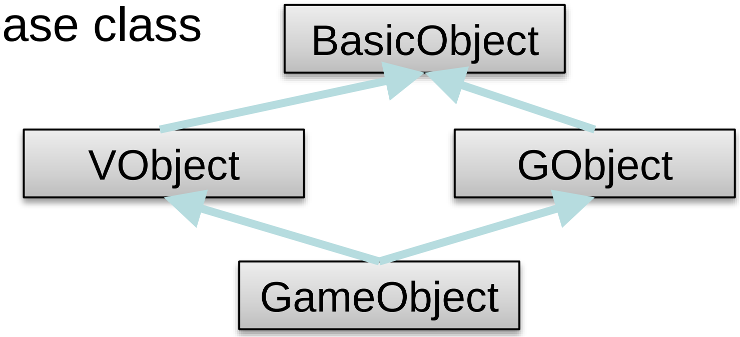
Object Centric Architecture

- Monolithic Class Hierarchies: PacMan



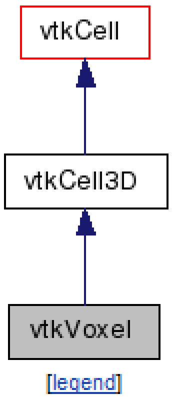
Monolithic Architecture - Discussion

- can be difficult for many classes (deep and wide hierarchies)
  - Deep hierarchies: different taxonomies can be used for subdivision, not always clear and natural/different ways are possible
  - Multiple inheritance: generates loops in class hierarchies („diamond of death“)
    - Contains multiple copies of base class

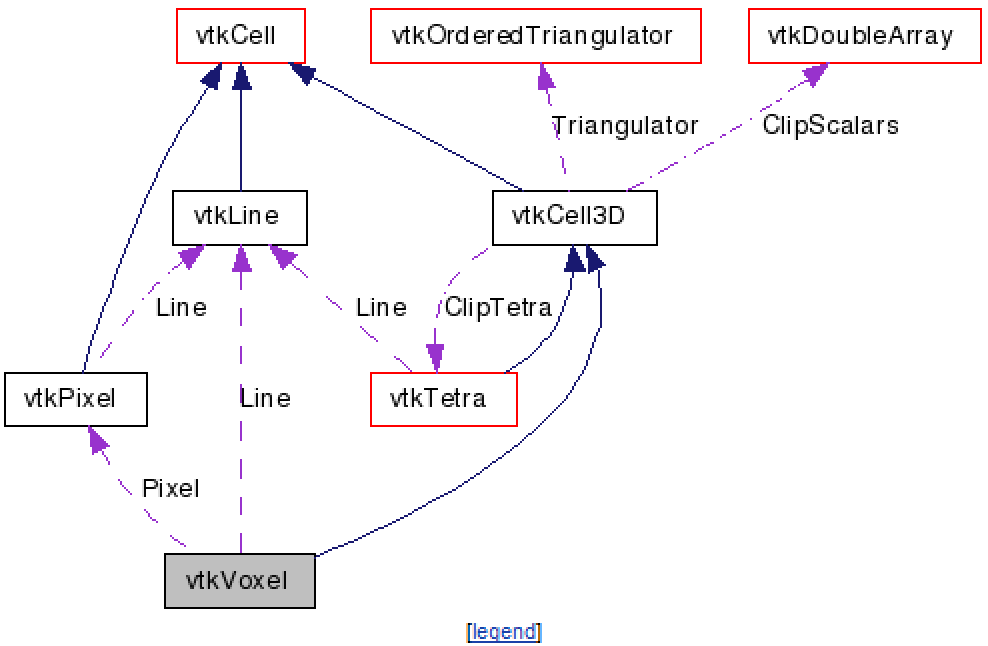


  - Bubble-up-effect: share properties from classes that have otherwise nothing to do with the own one
- Solutions
  - Is-a-composition leads often to monolithic architectures: is-a-relationship as design principle
  - Alternative: has-a-composition .... see next

Example vtk



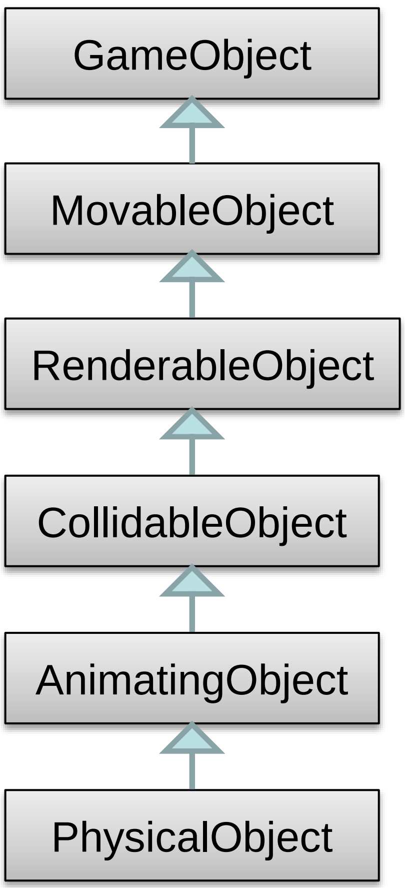
vtkCell
vtkCell3D
vtkVoxel
[legend]



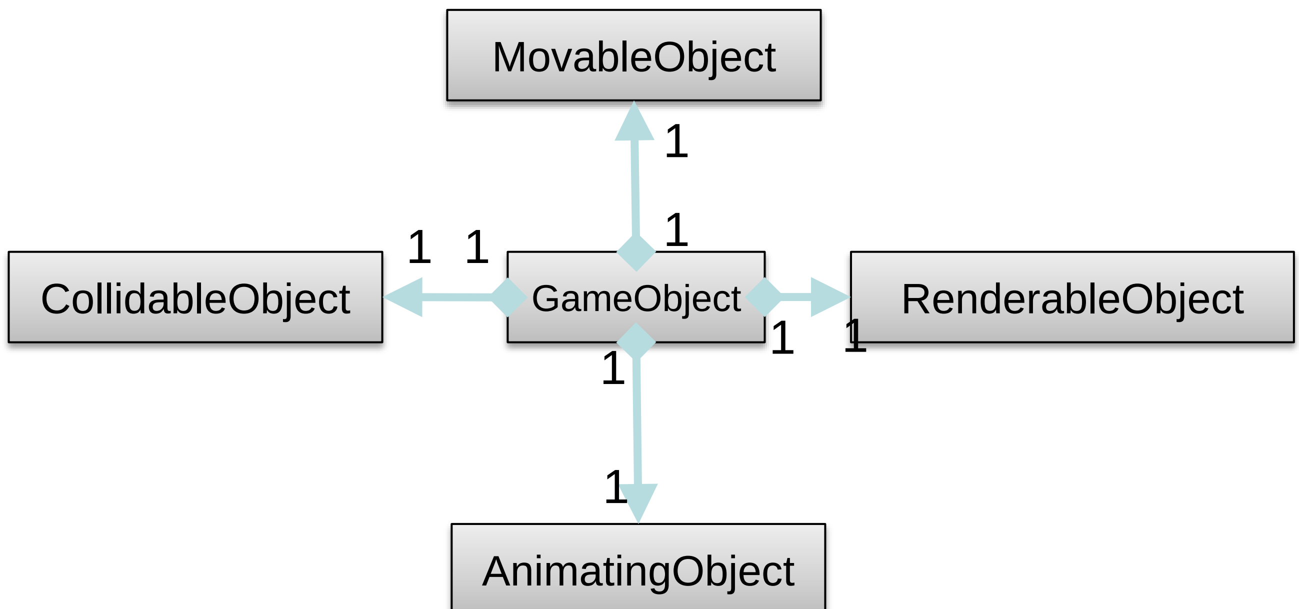
vtkCell
vtkOrderedTriangulator
vtkDoubleArray
Triangulator
ClipScalars
vtkLine
vtkCell3D
Line
Line
ClipTetra
vtkPixel
Line
vtkTetra
Pixel
vtkVoxel
[legend]

Example

- Instead of



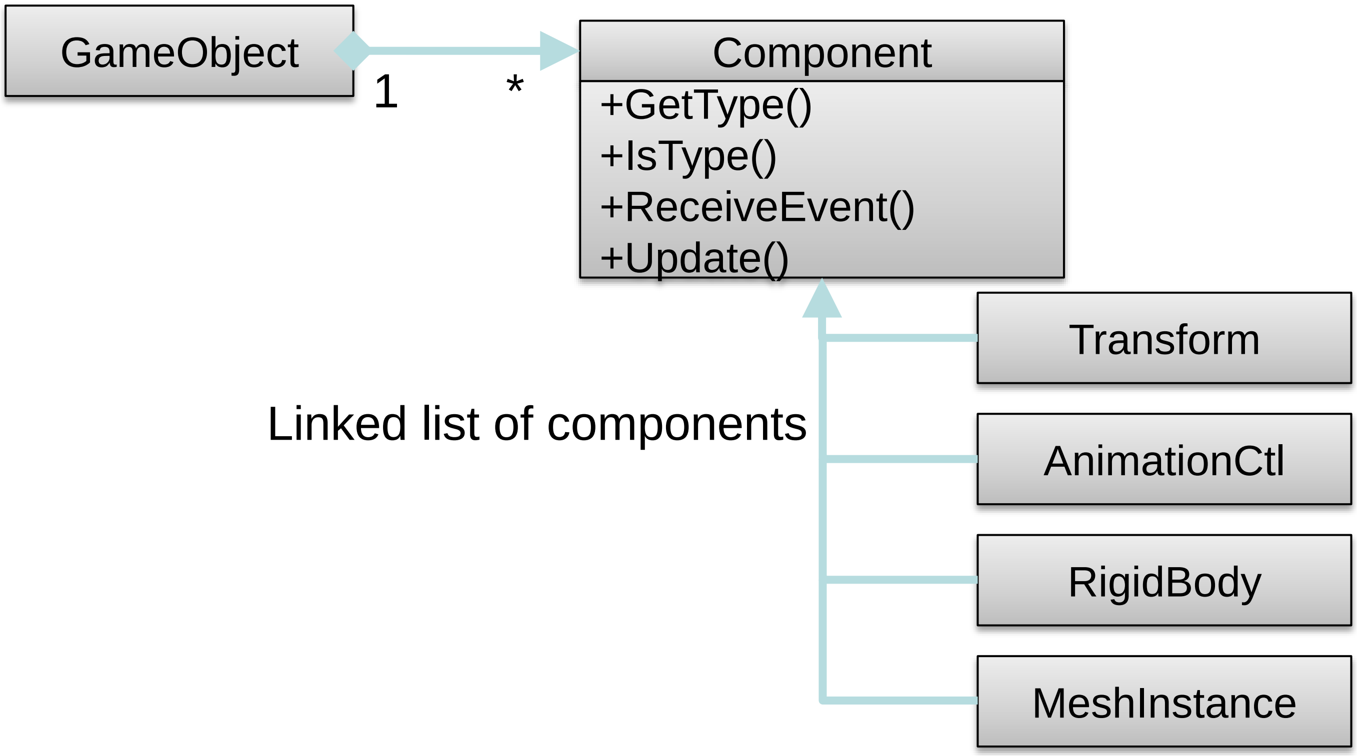
- Isolate features in individual classes (service objects): has-a-relationship



What is the advantage and disadvantage?

- Pro
  - attach on need
  - isolated elements
- Con
  - fixed number of components foreseen
  - difficult to extend: internal code

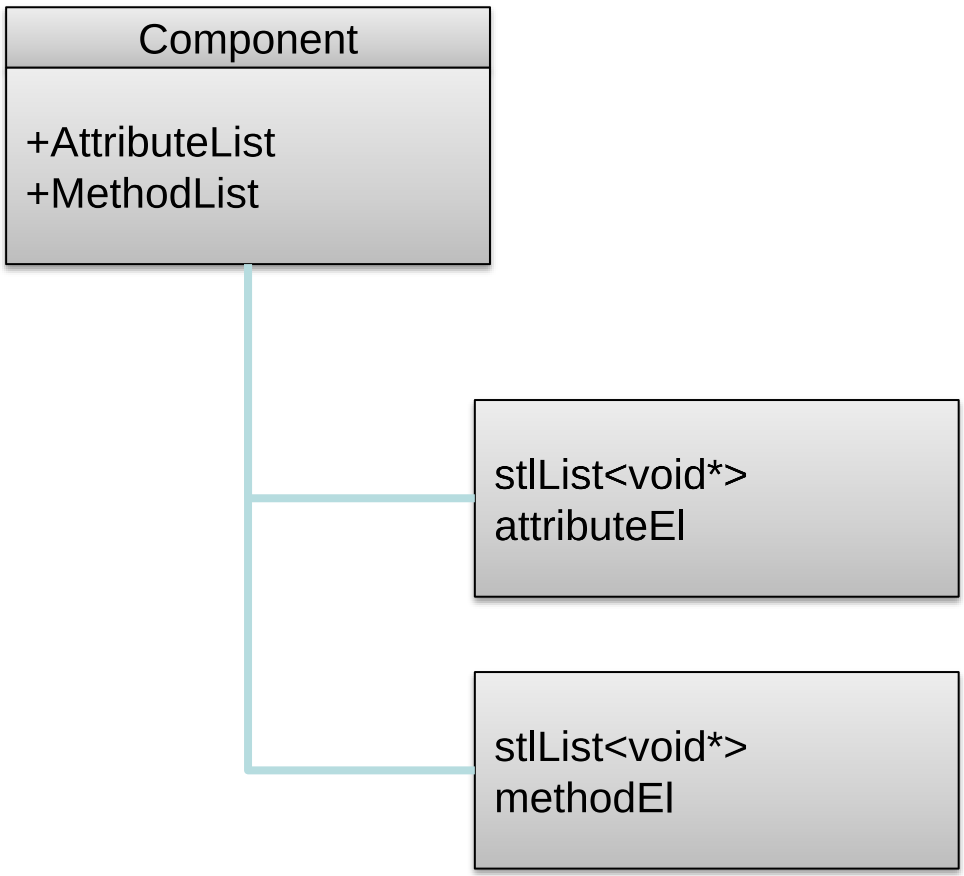
Better Solution: Generic Components



Pro/Con

- Pro
  - as before
  - but easier to enlarge
- Con
  - still fixed list

Generic Generic



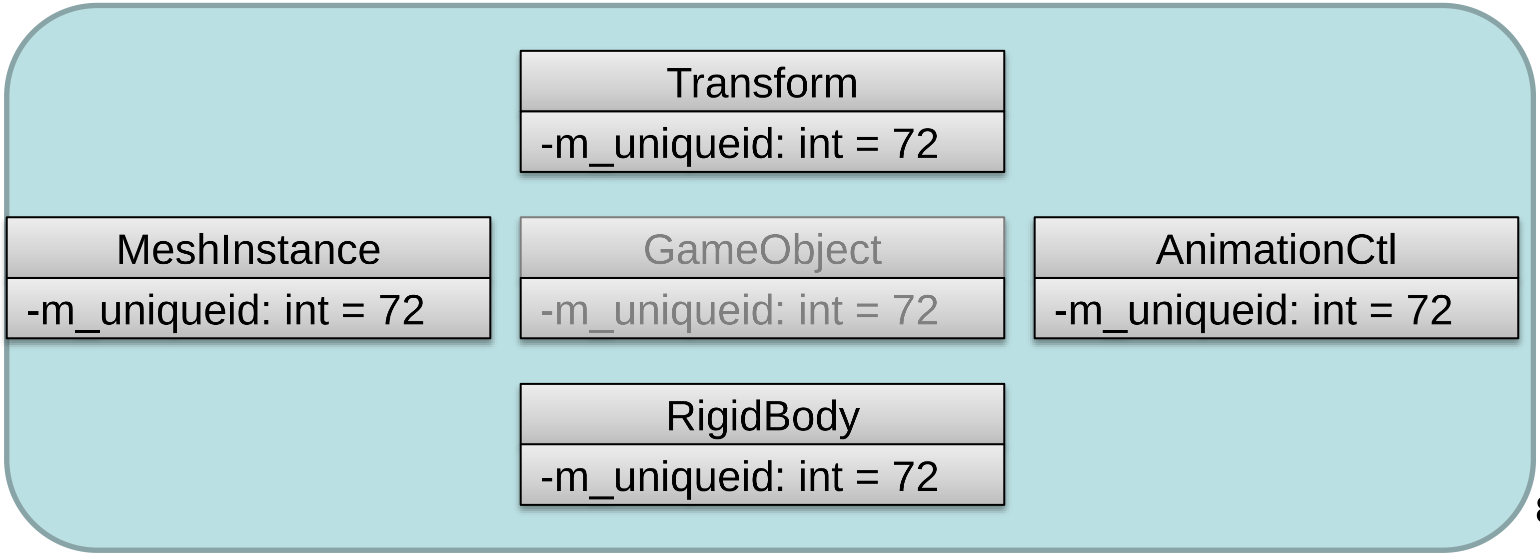
In program: convert void* into actual type

Pro/Con

- Pro
  - as before
  - but now list can be update on the fly
- Con
  - <void*> pointer does not allow to be completely generic
    - solution: C++14 auto keyword

Pure Component Models

- Leave out any logic in GameObject, only components in container
  - Linked together in logic id
    - Look up components which are used by id, pointers are in tables (id is address, pointers are contents)
      - Initialization and communication could be a problem since there is not THE corresponding GameObject to be called



Property-Centric Architectures

- …think about objects having attributes and behaviors: property centric view

Object1
- Position = (0,3,15)
- Orientation = (0, 43, 0)

Object2
- Position = (-12,0,8)
- Health = 15

or

Position
- Object1= (0,3,15)
- Object2 = (-12,0,8)

Orientation
- …

- Very successful architecture: mimics relational data base with game object‘s id as primary key
- Question: How do we implement behavior?

Pro/Con

- different view: from the properties and not from objects
- Pro
  - each property knows which object requires this property
  - easy to attach new object
  - property on demand
- Con
  - object is not aware of its properties
    - would require a property manager that is caring about that

Implementing Behavior

- Via Property Classes
  - Each property can be implemented as property class
    - Bool, float, renderable triangle mesh, AI
    - Behavior over class internal methods
    - Full behavior over aggregation of such classes
- Via Script
  - Put property values in table and use scripts to for implementation
- Versus Components
  - Property object: use multiple sub-objects for each game object
    - Each sub-object defines an attribute

Games by Properties

- We first define the individual functionalities like
  - movability
  - rotatability
  - physics
  - collidable
  - renderable
  - animatable
- Then we generate a set of objects with IDs
- Finally, we provide object lists for each functionality
  - The object interfaces for each functionality should always be the same, so we can handle any sort of incoming object.

Overview of the next sessions...

- Graphics
- Collision
- Physics/Animation/AI
- Path Planning

Graphics Engine

- Built on top of low-level interfaces like OpenGL or DirectX
- High-level interface to reduce complexity and having tuned functions
  - Sprites
    - Background image that is displayed instead of rendered
- High-level modeling
- Handles complicated display aspects
  - Special effects
  - Overlays
  - Multiple views
- Reduces graphics to model generation
  - All rendering aspects are encapsulated like
    - Scene graph exploration
    - Visual effects like particle mapping, shadows, textures, graphical user interfaces, menus, …

Collision Engine

- Determines collisions between objects of parts of objects in environment
  - Needs just a flag for allowing for collisions, no knowledge about algorithms required
  - Can handle different degrees of realism
- Often part of physics engine

Physics Engine

- Computes the physics simulation
  - No need to understand the physics, just tick to switch on gravity, soft objects, collisions etc.
  - Handles complex scenes like explosions
  - Handles massive number of objects like in crowd simulation
- Typical engines are
  - Havok
  - physX
  - Open DynamicsEngine
  - Bullet

Animation Engine

- Often part of physics engine
- Can handle offline things like
  - Motion capture
  - Motion editing and annotation
- Handles online
  - Sprites/texture animation
  - Vertex animation by skinning
  - Rigid body and skeletal motion
- Examples
  - Granny
  - Havok
  - Edge
  - Endorphin/Euphoria

AI (often combined with Path Planning)

- Models behavior and interaction like dialogue (scripted or generated)
- Motion (path planning, obstacle avoidance)
- Strategies (hiding, attack)
- Decision making
- Crowds
- Often: agent-based with the entities
  - Perception: sense input
  - Decision: AI core, rules/state machine
  - Action: sequence of operations
- Examples
  - Preagis
  - Kynapse
  - DirectIA
  - SimBionic

Input Devices

- Consoles
- Joystick
- Special input devices like
  - Wiimode
  - PS move controller
  - Kinect

Core Engine

- Implements
  - Event system for communication between objects
  - Scripting for game logic
- Gameplay in native language or script
  - World actions and rules, character abilities, objectives of game
- Scripting languages
  - Easy control, quick to learn and use, data driven
  - Low performance
  - Examples
    - Python
    - Lua
    - GameMonkey
    - AngelScript

Management Functions in Game Engines

- Mathematics System
  - Accelerates frequently used functions
    - Low resolution results
- Manager Classes
  - Manage resources and objects
- State Machines
  - Allow to simplify the dynamic behavior of the Game Logic
    - Different states in game which are well defined
    - Clear state transition due to external signals
    - Clear behavior in each state resp. state transition
  - Allows to implement arbitrary complex rule systems for AI

Summary

- Games exist of three parts
  - Game Engine
  - Game Logic
  - Game Art
- Central
  - Important parts: Physics, Collision, AI, Graphics
  - Manager for Audio, Graphics/Material, Files, Memory, CPU