Object Systems

Methods for attaching data to objects, and connecting behaviors

by Doug Church

Outline of Talk

- Quick definitions
- Progression of object data complexity
- The problem in the abstract
- Current approaches to this problem
- Small Case Study
- Problems, lessons, and tradeoffs

Why an ‘object system’?

- Games have more and more objects to track
- Compared to years ago, these objects are
  - more diverse (different types of data)
  - more dynamic (data and fields change often)
  - much larger
  - support more emergent behaviors
- Game object data has become a major part of our games, worth real analysis

A brief history of Objects

The goal of this section is to make clear the complexity of objects in some modern games.

We will examine a sequence of object models, from simplest up, looking at how they handle the data issues.

Hard-coded Constants

eg: Space Invaders

- Objects simply have type and position
- All behaviors are implied from this
- A big static array of objects
- No save/load or versioning

Simple object structures

eg: basic RPG circa 1990

- ~5 custom structure types (npc, weapon)
- Each has a common beginning (location, id)
- Each has custom fields (hp, damage, speed)
- Often pre-sized static arrays for each type
- Simple save/load without versioning

C++ hierarchy

- Just use C++ inheritance as is
- Objects are simply class objects
- Derived classes add new fields
- Binary rep not trivial to effectively version
  - Though C++ makes serialize functions easier
- This initially was just data, not code

Example: Space Invaders

- All objects have same data
- Type data used to control behavior
- Your Ship, Enemy Ships, Bullets, UFO
- Global state (remaining enemies, time) used to control enemy speed, animation
  - Probably position too, really

Object
- Type
- Position

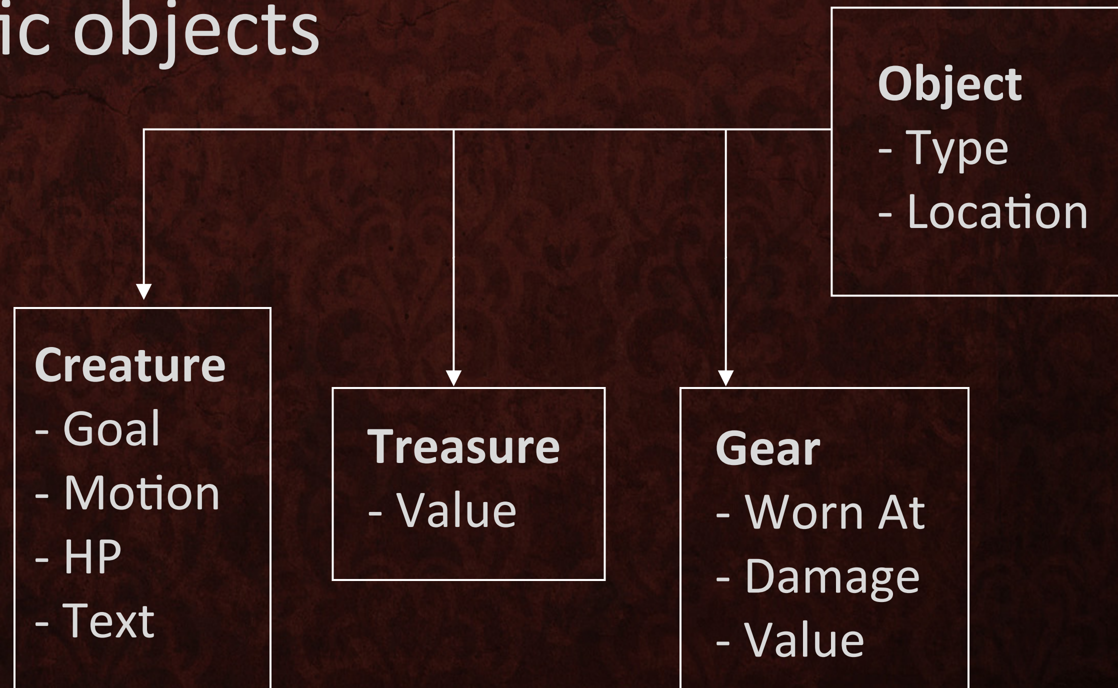
Example: Tempest

- More types of enemy ship, with more individualized/asynchronous behavior
- More data to store per object

Object
- Type
- Position
- Speed
- Animation

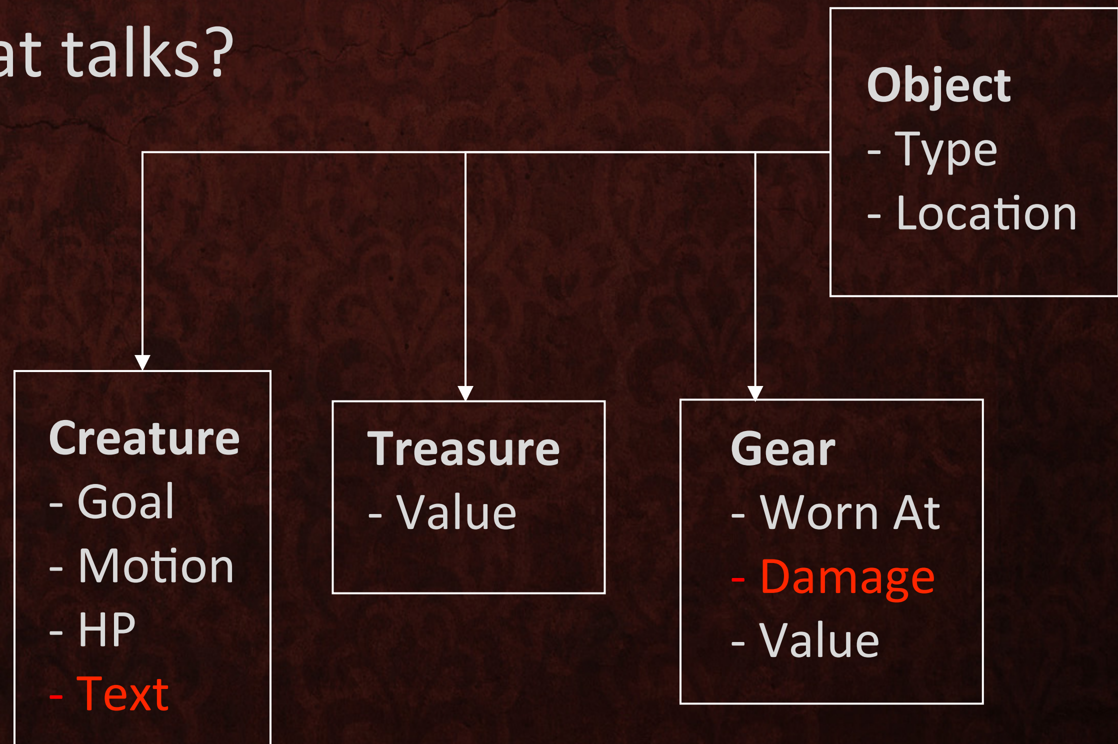
Example: Old School Tile RPG

- Several game object types, some common data, but also type specific information
- Many specific objects



Example: More complex RPG

- But how do I make
  - a spiky tree that does damage?
  - a sword that talks?



Example: More dynamic RPG

- What if I want some objects to be able to catch on fire, be poisoned, or make a sound when hit, or…
- Do we just start moving all possibly necessary data up into the basic object?
- This may sound like a few simple things, but a modern simulation has hundreds, if not more, potential data fields to track

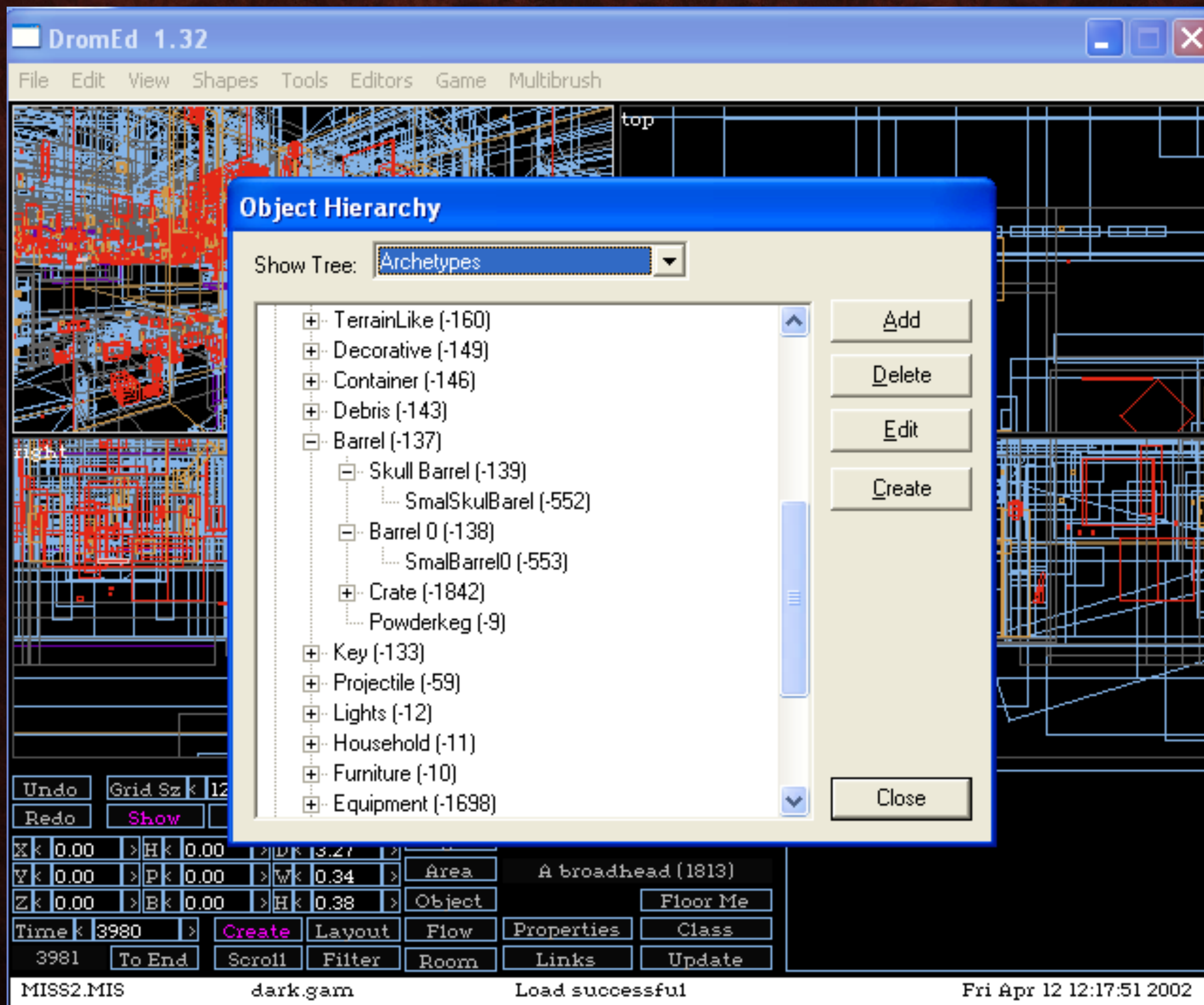
If you don’t believe it can really get extremely complicated

Here are some example screenshots

(from the Thief Game Editor)…

- Object Browser
- Specific Object
- Property List
- Property Editing on an Object

Object Browser



Specific Object

DromEd 1.32

File Edit View Shapes Tools Editors Game Multibrush

A PosterScroll (1018)

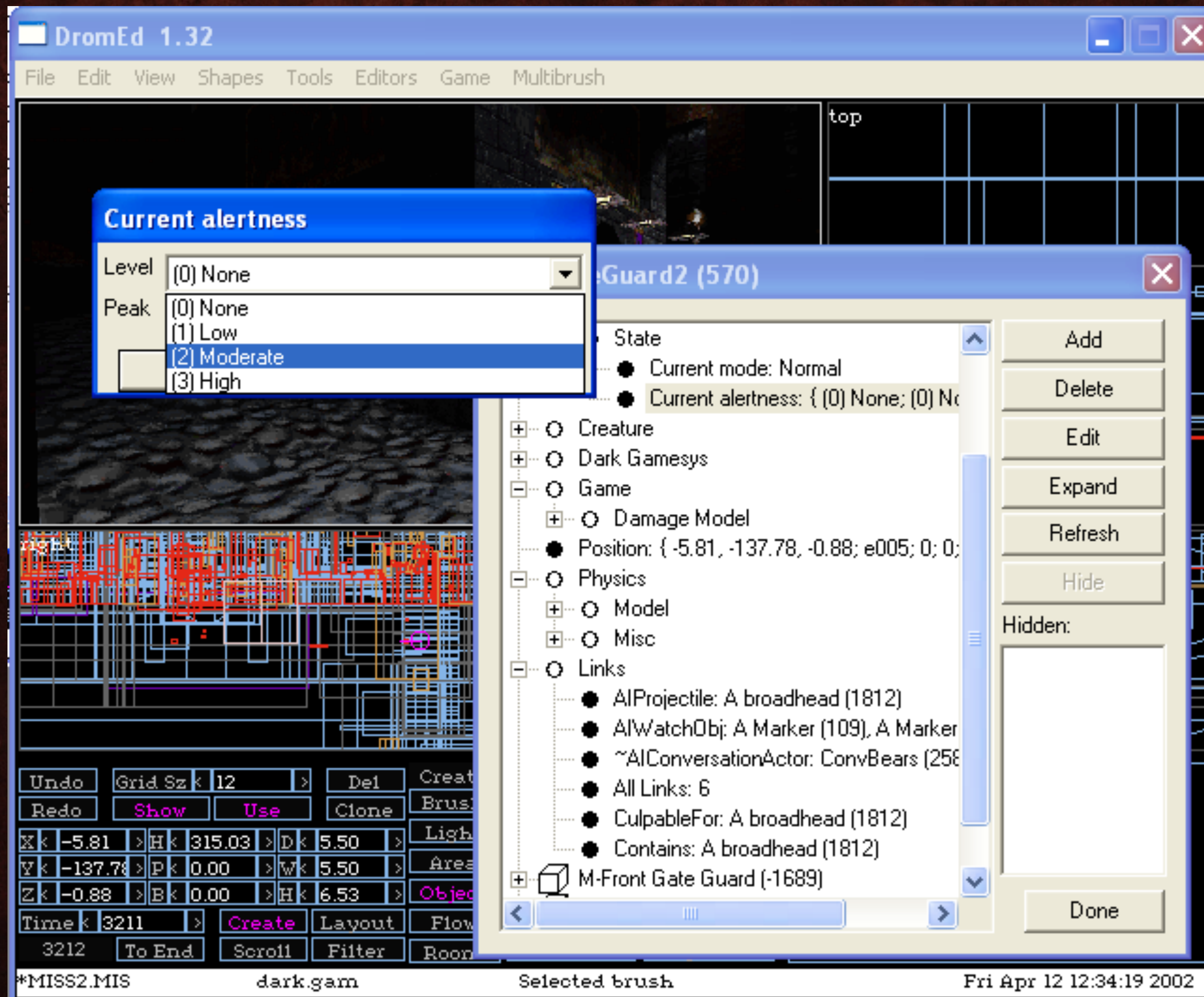
- A PosterScroll (1018)
  - Position: { -12.07, -178.02, -4.50; 0; 0; 0; -1; -1
  - Physics
    - Misc
      - Collision Type: Bounce
  - PosterScroll (-2489)
    - Shape
      - Model Name: scrollop
    - Plaque (-2487)
      - Engine Features
        - FrobInfo: { Script; [None]; [None]}
      - Scripts: { StdBook; ; ; ; FALSE}
      - Decorative (-149)
        - physical (-7)
          - Game
            - Bash Params: { 200.00; 0.
          - Inventory
          - Physics
          - Object (-1)

Add | Delete | Edit | Expand | Refresh | Hide | Hidden: | Done

*MISS2.MIS dark.gam Selected brush Fri Apr 12 12:28:07 2002

Property List

Property Editing on an Object



Lesson

As we get more complex, simple approaches break, data becomes more sparse and pushing it up the tree is very inefficient in terms of space, and it is hard to manage the data effectively.

Picture trying to put all that data in the basic object… or dynamically resizing and repacking objects with in-lined fields… or?

So what do we do about it?

Various cool approaches from real software:

- View it all as a relational data base
- Build in introspection like capabilities
- Attach code to objects for flexibility
- Support varieties of storage forms
- Inheritance and instantiation of aspects

What sorts of choices do we have?

- Is the data stored with objects or properties?
- Major speed/space tradeoffs for retrieval
- How does inheritance and overriding work
- Where are the behaviors associated with the property? In scripts, in game code, in both?
- How different are abstract and concrete objects?
- How dynamic is the system? Can you create types at run-time? Change properties on abstracts?

Some approaches

Particular approaches from recently shipped games

Halo

- Object-centric view
  - Fixed size block (stored in an array)
    - Position, velocity, and a pointer to the variable block
  - Variable sized block (stored in a heap)
    - Subsystem dependent properties (AI, physics models, effects)
- Class hierarchy of types
  - Shipped with 11 leaf classes
    - "things would have been much easier if we'd specialized more"
- Function evaluation system
  - Jeep velocity → jeep brake lights to get "backup lights"
  - Plasma rifle heat value → heat display meter

Thief

- Property-centric view of the world
  - Objects are just ID's, and can have any property
  - Of course, many have implicit rules of use and relationships
- Programmers code property behaviors
  - Publish property fields, value range, help message to edit tool
  - Flexible implementations, allowing a speed/space choice
- Scripting works with property system
  - Scripts can be attached and inherited like other properties
  - Script system can easily read/write property values
- Links between objects
  - Avoid out-of-date object reference bugs
  - Ability to attach data to connections (timeouts, etc)
  - Allow multiplicity… an object can care about lots of others at once
- 4000+ abstract objects, several thousand concrete objects

Dungeon Siege

- Property-centric view
    - Property hierarchy with single inheritance
    - Components are compositions of properties
- Highly integrated with scripting language
    - Properties can be written in C++ or script language
    - Properties can have scripts attached
- Components in hierarchy
    - Discouraged designers from dynamic component creation
        - "I don't want our designers mixing and matching types"
    - Designers wanted lots of prefab types for authoring speed
        - tree_waving, tree_bushy, etc.
    - 7500 leaf classes when shipped

Thief Property System Implementation Details

A Quick Case Study

Major goals

- Allow systems to quickly iterate over the data and objects they care about, don't force indirections.
- Programmer chooses what cases to optimize for, but system automatically supports all cases.
- Maximize designer flexibility and control.
- Allow scripting, but don't depend on it.
- A property should be implemented with as little impact on other systems as possible.
- Most properties are sparse, so creating and adding properties must be a light weight decision.

Coding with the system

- Objects are only an ID.
- One references a property by GUID (which you can get by asking the Property Manager for a GUID for a given name).
  - IProperty *pProp = GetProperty("HitPoints")
- Given a property, ID gets the data.
  - int hp = pProp->GetIntProperty(ID)
- Cannot change the data in place, need to call Set.
- For Structures, pass reference to pointer you want.

Property Implementation

- A collection of data storage models are pre-built, instantiate the template for your type…
  - Arrays, individual hash, shared hash, bitfields.
- Define the fields of the property by name.
- Can deal with properties in two ways:
  - Given an object, query about the property.
  - Iterate over all objects with the property.
- Which data store you use depends on your primary usage case, and how often you reference it.
- All of this property code is very localized.

Example: Physics

- Physics needs fast random data access about all physics objects while in physics loop.
- Physics keeps a linear array of data.
  - Array has all physics information + Object ID.
  - Supplementary hash connects ID -> array.
- Physics loop, therefore, just operates on its own array.
  - No indirections or overhead, just use the array information.
  - Can use the ObjID to get the values of any other properties.
- Other systems which need physics data go through GetProperty, which internally calls the physics storage model, hashes the ID, and returns array data for that object.

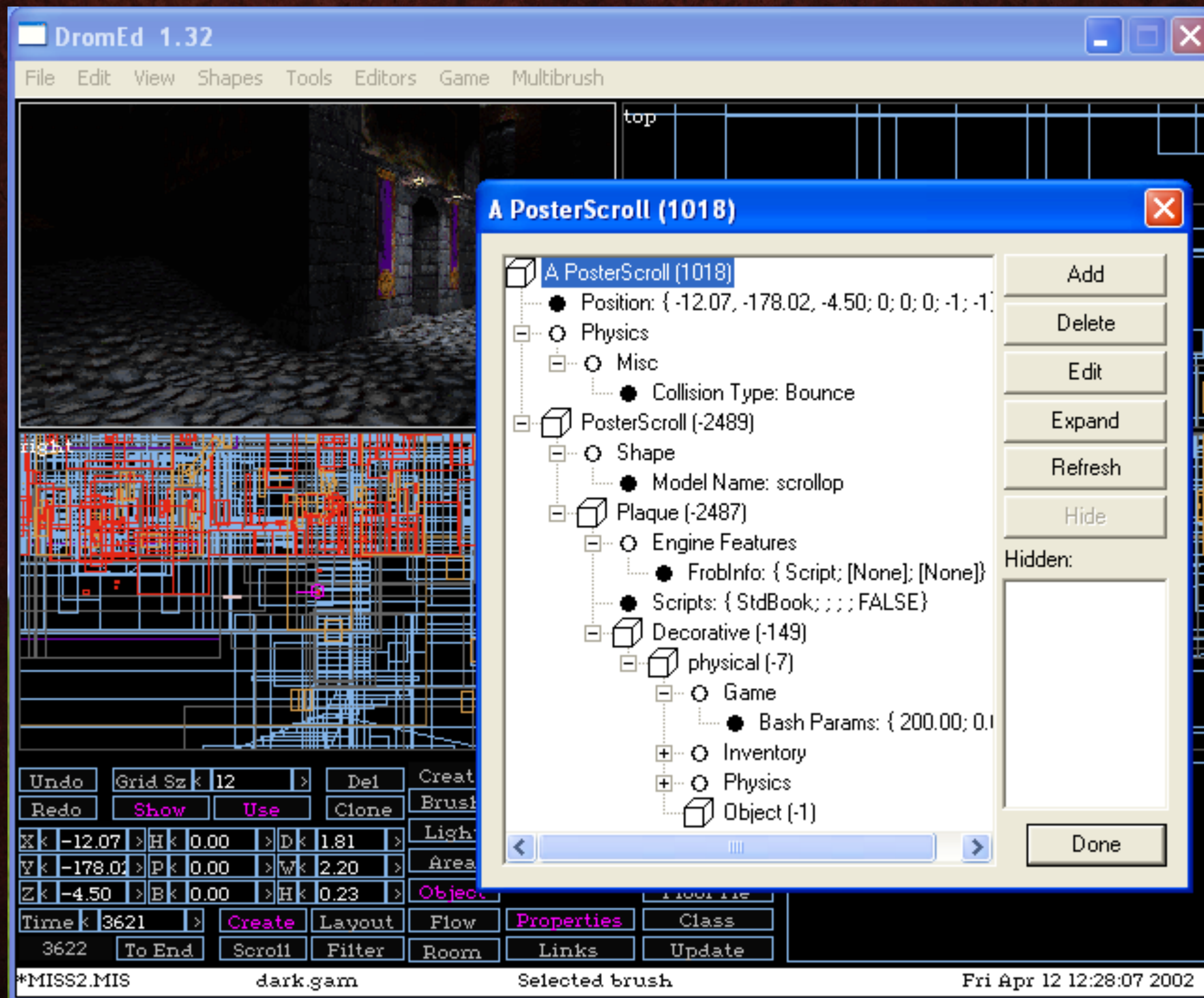
Sadly

- The object tree is big, so object based property lookups end up being relatively slow.
- This is because many property retrievals require a traversal of the entire tree.
- So we need to reduce the tree searches.
- When we find a property value, cache it at the lowest abstract level of the tree.
- Future lookups will find the cached version.
- Note: Need careful coding to make sure that cache invalidation works right when tree changes.

Designer Usage

- Programmers expose a name and valid data ranges for each field of each property.
- Designers can place properties on (and links between) objects, and change attached data.
- Designers create and maintain the object hierarchy, decide how to define the world.
- Can create simple (int) properties by name.

Specific Object



Major Issues

- Object creation time too long.
  - Have to ask each property what it wants to do
  - (machine gun bullets in Combat Flight Sim)
- Programmer complexity… to meaningfully use the system, steep learning curve.
- Tools were not robust until the middle of the development.
  - So we missed chances to do things right earlier.
  - Lots of code from earlier ‘unfinished’ period shipped.
- Property systems underlie the game, so changes to them impact large sections of the code.
  - The system allows easy change to property code.
  - But changes to the system itself are not so easy.

Problem Example #1

- Hierarchy, inherited value, local override
  - What happens when global value changed?
  - Several subtle issues, for instance:
    - If a local property has same value as global, do you save it out, or do you auto-delete it as it is redundant?
    - Of course, if you remove it, and the global changes, then you lose the original set value.
    - Probably want the local to remain itself even when shadowed by the same global setting.

Problem Example #2

- Failed queries can take a lot of time.
  - Determining a property is missing requires a full search up the tree.
  - Successful lookups are stored in cache.
  - To deal with failure problem, they must be too.
- This fixes the speed problem.
  - But requires extra cache space and complexity.

Typical Object System Design Frustrations

- Object creation time can get very large.
- Need tools which show why a given property value is on a given object
  - Especially when debugging the system itself.
- Your designers will use the system… be prepared for very “creative” usage.
- Educating the team is vital! Designers and programmers both must “get it”.