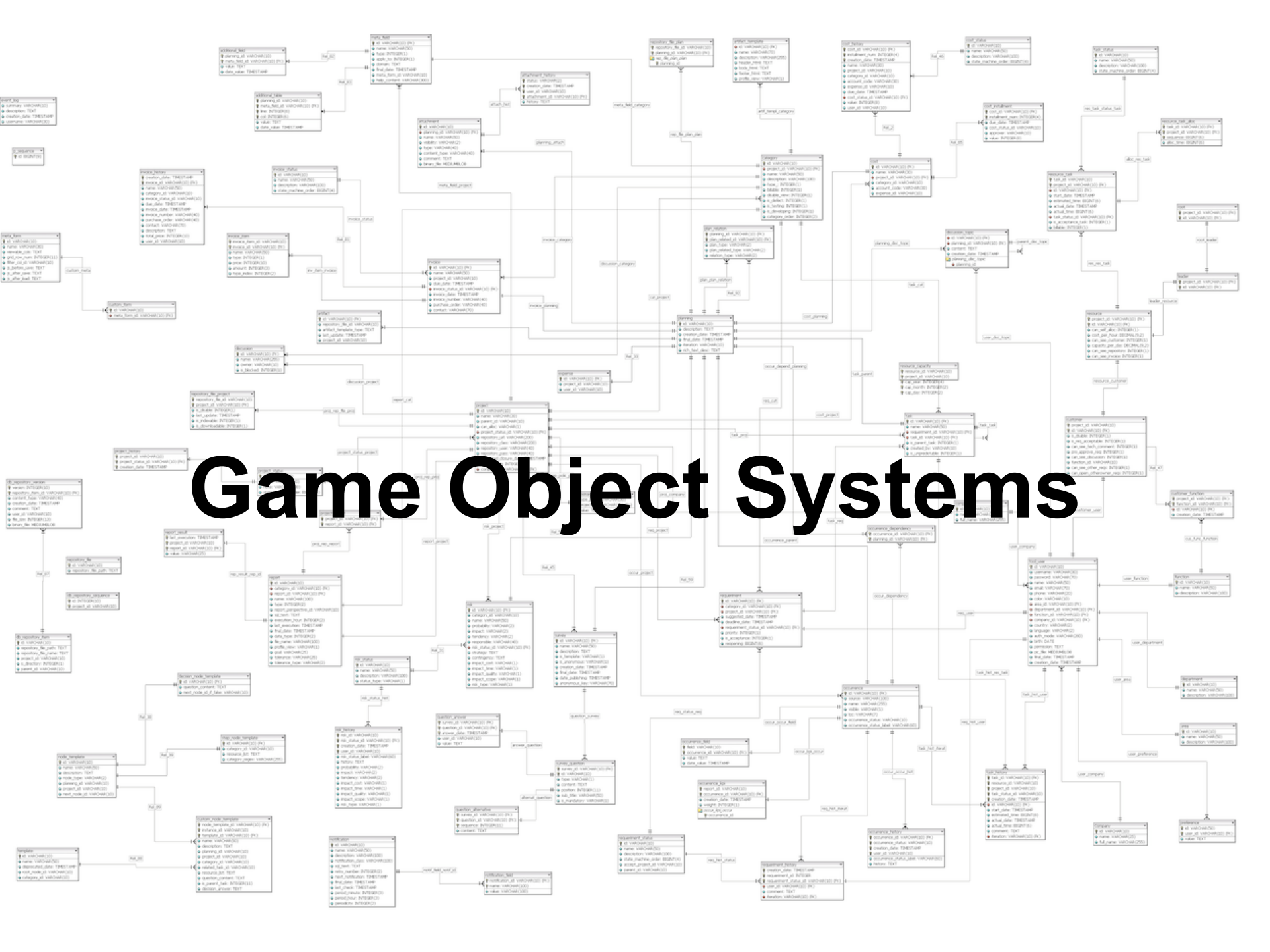

Game Object Systems

Introduction

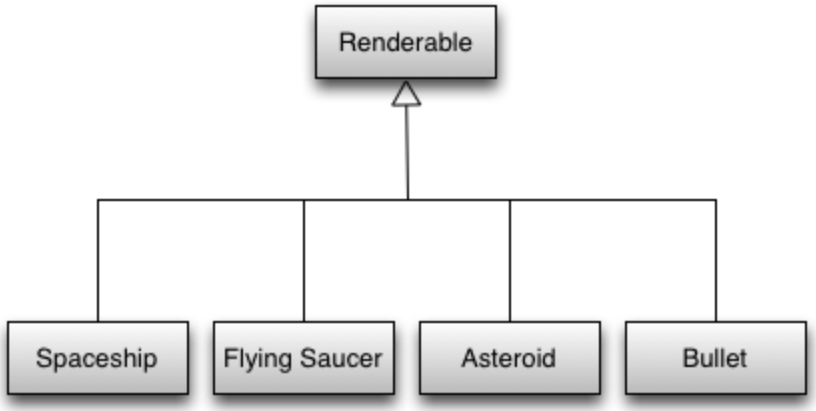
- So far we've explored many subsystems
  - Collision detection/response, AI, networking, etc.
- Today we'll explore game objects
  - Central to the architecture of a game engine
  - Difficult to implement in a large system
  - Lots of case studies from real games

Game Objects

- Piece of logical interactive content
  - Monster, tree, level, door, item, trigger, camera sequence, etc.
- Diverse set of behaviors
  - Pathfinding, networking, animating, rendering, triggering, persisting
- What are goals of a game object system?
  - Flexibility
  - Reusability
  - Ease of use (even for non-programmers)
  - Modularity
- Is a class hierarchy the best technique?

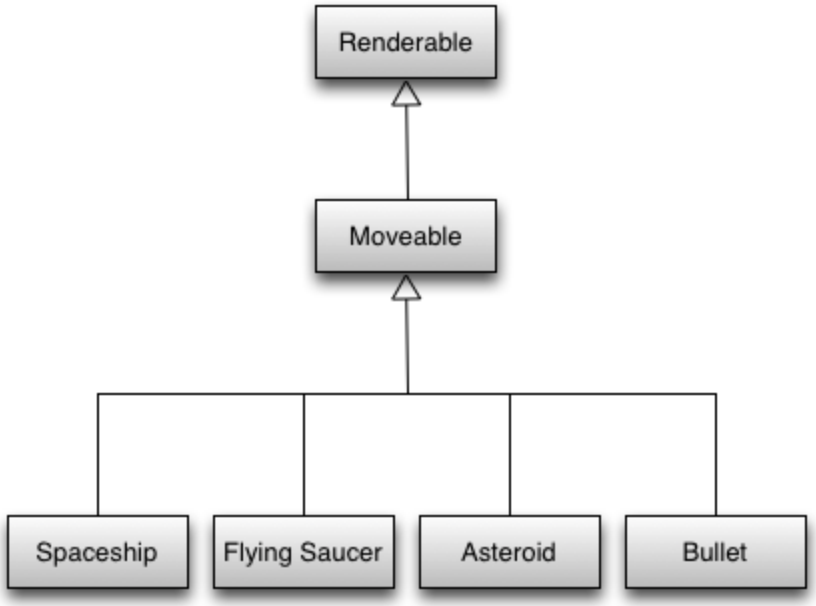
The Problem with Class Hierarchies

- Renderable class has a position and sprite



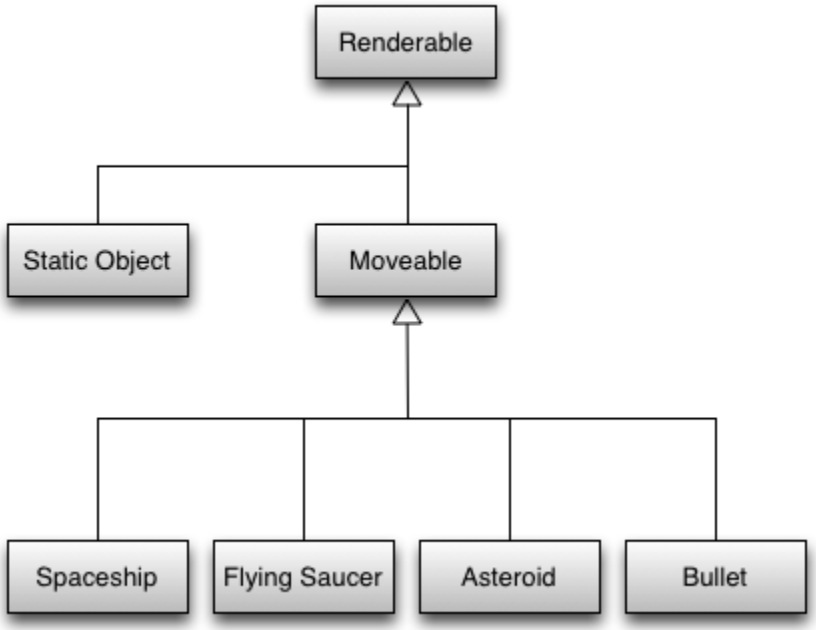
The Problem with Class Hierarchies

- Moveable class has position and velocity



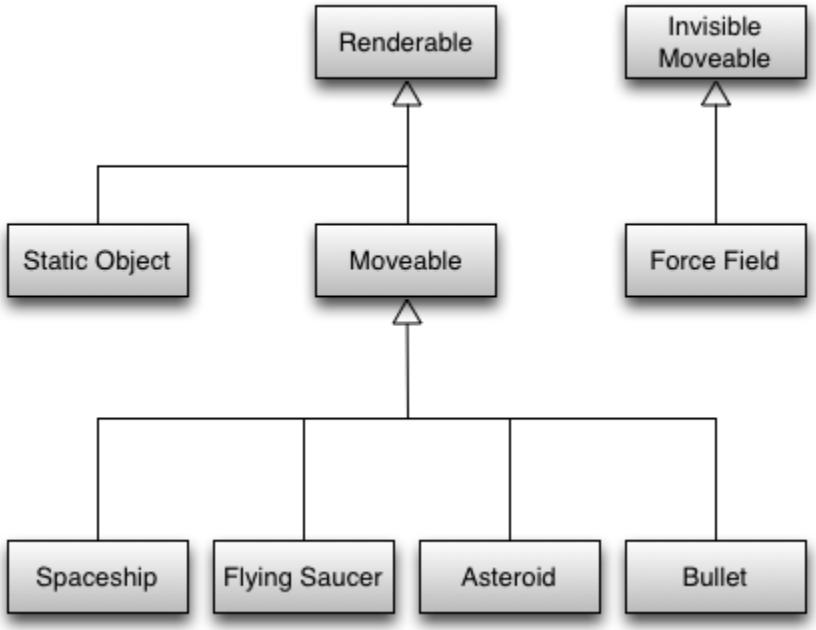
The Problem with Class Hierarchies

- Works for non-moveable renderables



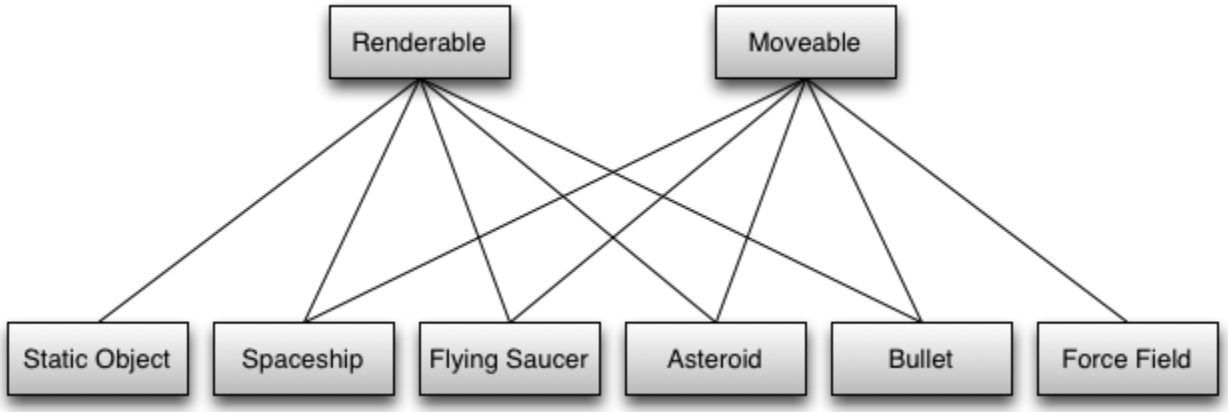
The Problem with Class Hierarchies

- What about non-renderable moveables?



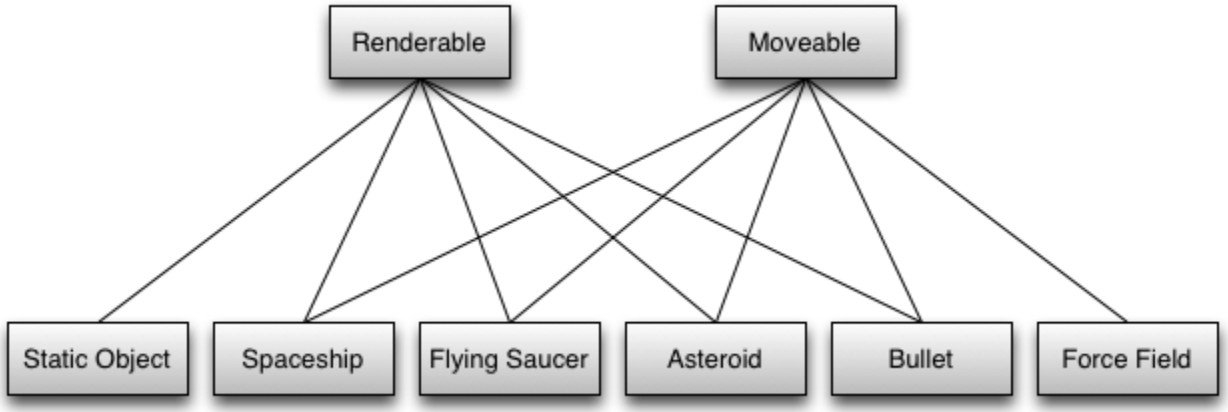
The Problem with Class Hierarchies

- Try composition over inheritance
  - Entity has renderable and moveable as members
  - Now there are two copies of position



The Problem with Class Hierarchies

- Try multiple inheritance
  - Virtual inheritance solves duplication (one position)
  - C# and Java don't have it (not portable)
  - Entities are still defined at compile-time, artists can't change entity definitions



The Problem with Class Hierarchies

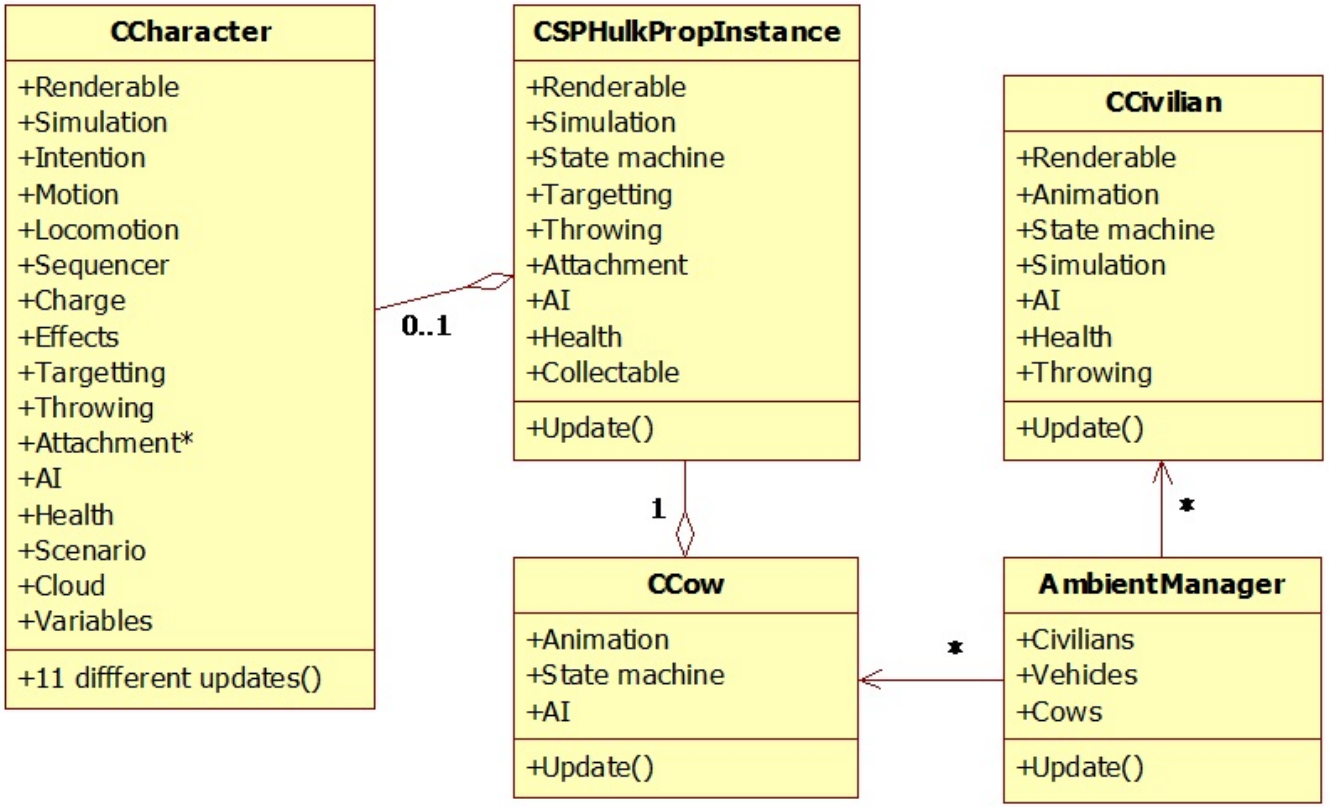
- Single inheritance hierarchies don't scale
  - Functionality drifts upwards
  - Hierarchy stops making sense
  - System "hardens", refactoring becomes expensive
  - A DAG can't describe every set of relationships

Case Study: *Hulk: Ultimate Destruction* (2005)

- Single inheritance game object system
- Monolithic CCharacter class
  - Common behaviors all boiled up to this superclass
  - 11,000 lines of code
  - 20k memory footprint
- CCivilian implemented from scratch
  - Too memory intensive to use CCharacter
- Other overarching problems
  - Have to decide how to categorize each new entity
  - Difficult to add new behaviors
  - Difficult for designer to tweak system

Case Study: *Hulk: Ultimate Destruction* (2005)

- Everything migrated up into CCharacter:



Component-Based Development

- Alternative to class hierarchies
- One entity class
  - List of behaviors (logic)
  - List of attributes (shared data)
- Separation of concerns
  - Components each handle a small, specific thing
  - Promotes modularity and cohesiveness
- Substitutable
  - Swap in one component for another

Component-Based Development

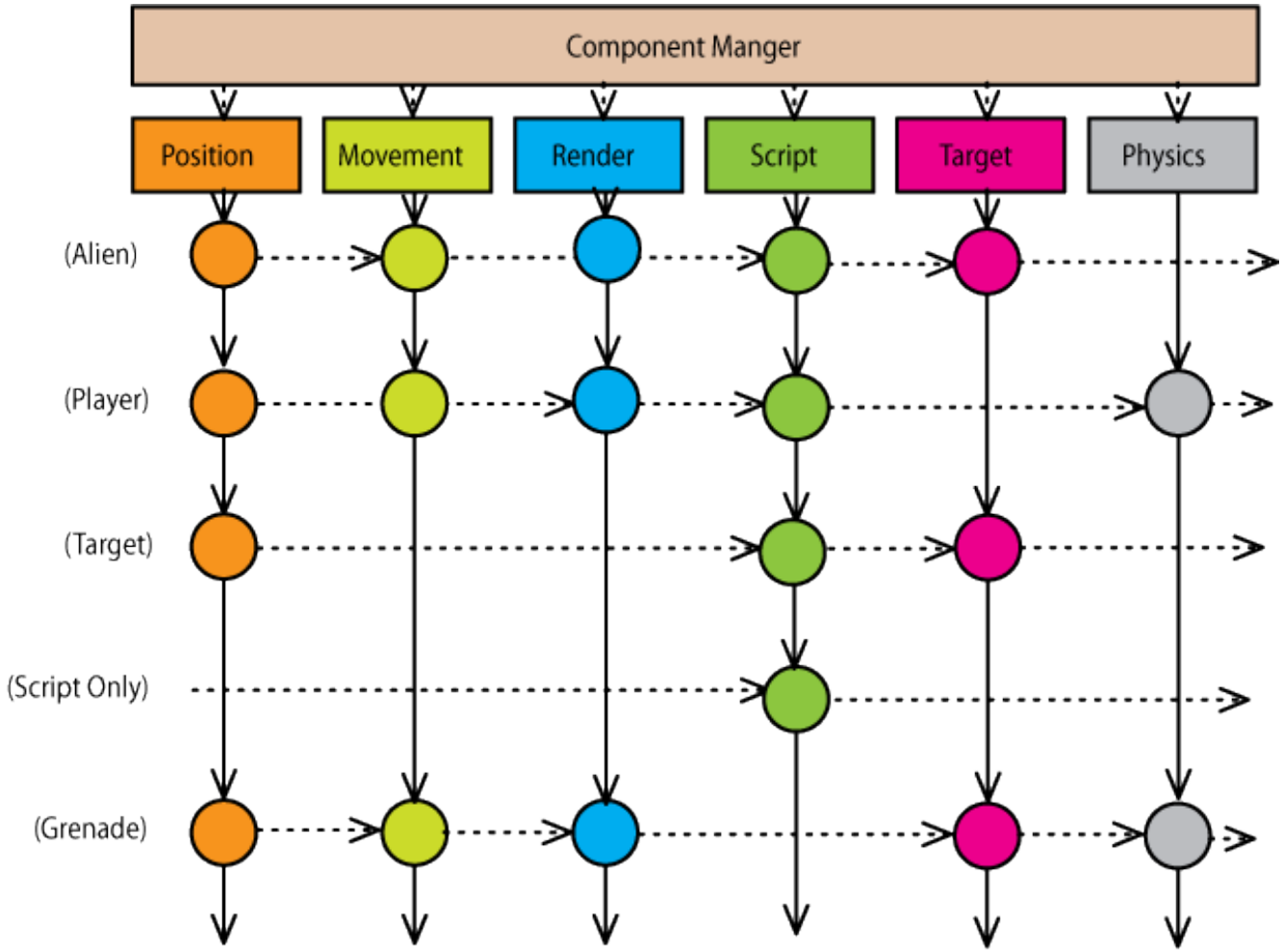


Figure 2 Object composition using components, viewed as a grid.

Case Study: *Prototype* (2009)

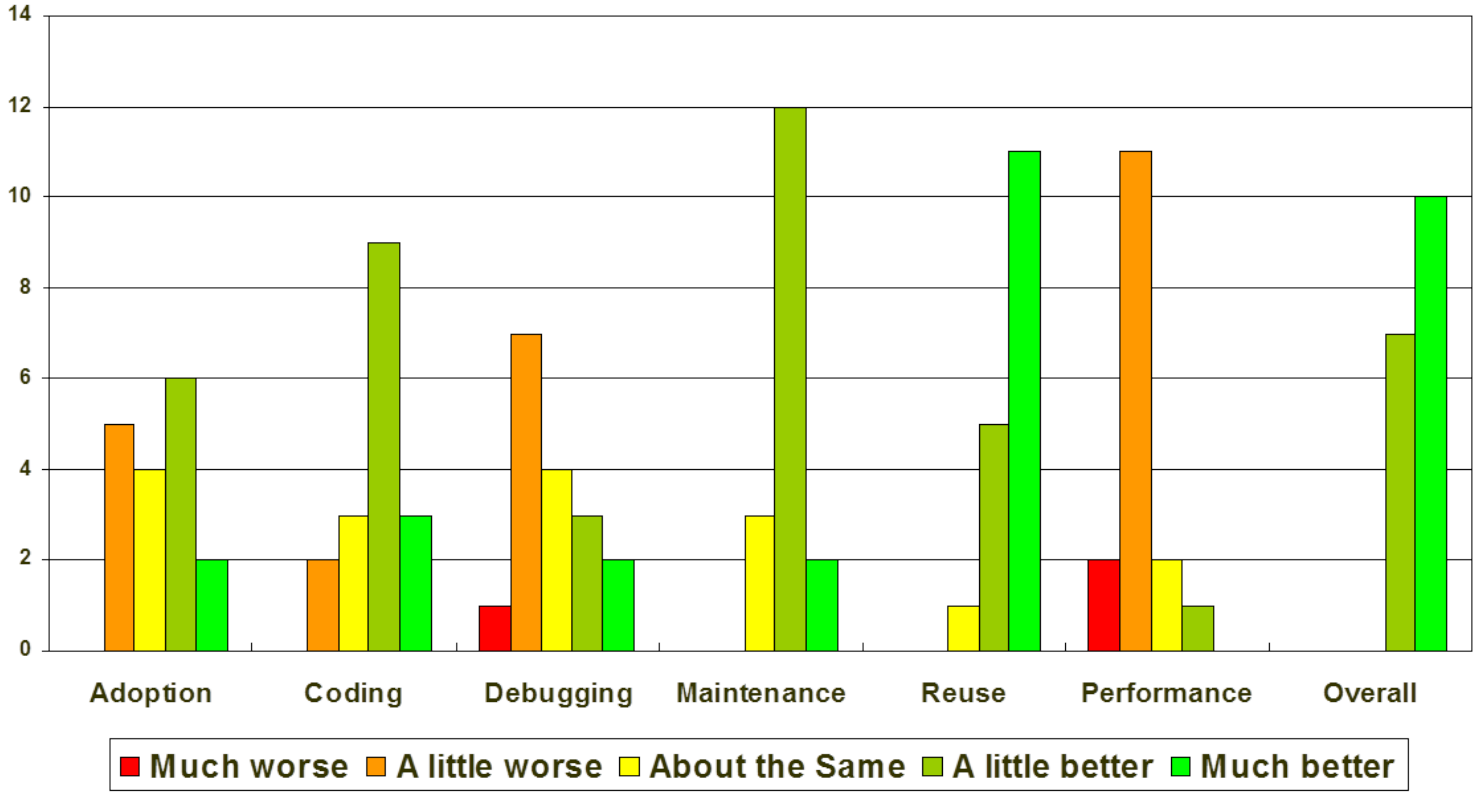
- First component-based system for Radical Entertainment
- *Lots* of specialized components
- Entities only hold relevant components
  - Avoids monolithic superclass problem from *Hulk*
  - Much more memory efficient
  - Easy to create new behaviors

Case Study: *Prototype* (2009)

Alex	Helicopter	Pedestrian(HLOD)	Pedestrian(LLOD)
PhysicsBehaviour	PhysicsBehaviour	PhysicsBehaviour	
TouchBehaviour	TouchBehaviour		
CharacterIntentionBehaviour	CharacterIntentionBehaviour	CharacterIntentionBehaviour	
MotionTreeBehaviour	MotionTreeBehaviour	MotionTreeBehaviour	
CollisionActionBehaviour	CollisionActionBehaviour		
PuppetBehaviour	PuppetBehaviour	PuppetBehaviour	
CharacterMotionBehaviour			
MotionStateBehaviour			
RagdollBehaviour			
CharacterSolverBehaviour	CharacterSolverBehaviour		
HealthBehaviour	HealthBehaviour	HealthBehaviour	
RenderBehaviour	RenderBehaviour	RenderBehaviour	
SensesInfoBehaviour			SensesInfoBehaviour
HitReactionBehaviour	HitReactionBehaviour		
GrabSelectionBehaviour			
GrabbableBehaviour	GrabbableBehaviour	GrabbableBehaviour	
	GrabBehavior	GrabBehaviour	
TargetableBehaviour	TargetableBehaviour	TargetableBehaviour	TargetableBehaviour
AudioEmitterBehaviour	AudioEmitterBehaviour	AudioEmitterBehaviour	
FightVariablesBehaviour	FightVariablesBehaviour		
	EmotionalStateBehaviour	EmotionalStateBehaviour	
ThreatReceiverBehaviour	ThreatReceiverBehaviour		
	FEDisplayBehaviour	FEDisplayBehaviour	
		CharacterPedBehaviour	PedBehaviour

Case Study: *Prototype* (2009)

- Survey givens to engineers on *Prototype*
  - Compares component system with old class system



Implementing Components

- Entities: attributes and behaviors
- Concerns
  - What if processing order is important?
  - What if components need to share state?
  - What if components need to communicate directly?

Components: Messages

- Broadcast information between behaviors
  - Sent to GameObject, relayed to behaviors
  - Only notifies behaviors that are interested in the sent message type
- Used for unscheduled processing
  - Anything that doesn't happen every frame
  - e.g. collisions, state transitions, event handling
- Slower than regular function calls

Components: Example Behavior

```
void HealthBehavior::onMessage(GameObject* obj, Message* m) {
  switch (m->type) {
    case APPLY_DAMAGE:
      obj->attr<float>(HEALTH_KEY)->value -= m->arg<float>(0);
      obj->send(new Message(ATTR_CHANGED, HEALTH_KEY));
      break;

    case ATTR_CHANGED:
      if (m->arg<int>(0) == HEALTH_KEY) {
        if (obj->attr<float>(HEALTH_KEY)->value <= 0) {
          obj->attr<bool>(DEAD_KEY)->value = true;
          obj->send(new Message(ATTR_CHANGED, DEAD_KEY));
        }
      }
      break;
  }
}
```

Pure Component Model

- All logic split amongst components
  - Don't even need game object classes!
  - Represent game objects by integer IDs
- Not without its problems...
  - Harder to instantiate correct collection of components
  - No logical grouping of behaviors
  - Performance

Case Study: *Thief* (1998)

- Goals
  - Maximize designer flexibility and control
  - All scripting
  - Allow systems to iterate over relevant data without indirection
- Game objects are only an ID
- Properties contain data for all game objects
  - Attributes vs. properties: Array-of-structs vs. struct-of-arrays
  - `int health = GetProperty(HEALTH)->GetIntValue(objId);`

Thief Example Property: Physics

- Goal: fast random access for all physics objects while in physics loop
- Physics behaviors stored in array
  - Owned by physics subsystem
  - Supplementary hash map from object ID into array
- Physics loop operates entirely on its internal array, no extra indirection
- Other systems can get physics data using the supplementary hash map
  - Hash object ID and return physics data
  - Still O(1), but extra level of indirection

Thief: Problems

- Object creation is slow
  - Each property needs to create data for each new ID
- Property lookups are relatively slow
  - Compared to direct variable accesses
- Programmer complexity
- Need debugging tools
  - Show why a property value is on a game object

Aside: Components with Multiple Inheritance

```
// Layer 1: Base
struct Entity { virtual void draw(); virtual void update(float seconds); };

// Layer 2: Attributes
struct Position : virtual Entity { Vector3 position; };
struct Velocity : virtual Entity { Vector3 velocity; };
struct Health : virtual Entity { int health; };
struct Target : virtual Entity { Position *target; };

// Layer 3: Behaviors
struct Renders : virtual Position { void draw(); };
struct Moves : virtual Position, virtual Velocity { void update(float); };
struct Seeks : virtual Moves, virtual Target { void update(float); };

// Layer 4: Definitions
struct Rocket : virtual Health, virtual Renders, virtual Seeks {};
```

Components: Conclusion

- Pros
  - Maintainability
  - Easy reuse
  - Usability for non-programmers
  - Easy to integrate with editor / tools
  - Scriptability
- Cons
  - Extra glue code to set up
  - Debugging
  - Annoying to manipulate without editor / tools
  - Performance

C++ Tip of the Week

- Rvalue references
  - Part of C++11
  - Simple uniform way to avoid copying temporaries
  - lvalue = rvalue

```
// error, can't bind non-const lvalue reference to rvalue
S& ref = S();

// works, S() is an rvalue (a temporary)
S&& rref = S();
```

C++ Tip of the Week

- Rvalue references

```
// Old C++
template <class T>
void std::swap(T& a, T& b) {
  T temp(a); // now we have two copies of a
  a = b;     // now we have two copies of b
  b = temp;  // now we have two copies of temp
}

struct S {
  S(const S& c); // copy constructor
  S& operator = (const S& c); // copy assignment operator
};
```

C++ Tip of the Week

- Rvalue references

```
// C++11
template <class T>
void std::swap(T& a, T& b) {
  T temp(std::move(a)); // no copies
  a = std::move(b);     // no copies
  b = std::move(temp);  // no copies
}

struct S {
  S(S&& c); // move constructor, destroys c
  S& operator = (S&& c); // move assignment, destroys c
};
```

C++ Tip of the Week

- Rvalue references

```
// std::move(x) is just a shorter static_cast<X&&>(x)
template <class T>
typename std::remove_reference<T>::type&&
std::move(T&& t) {
  return static_cast<
    typename std::remove_reference<T>::type&&>(t);
}
```

Remaining Final Project Deadlines

- Public playtesting notes due next week
  - Include notes in platformer_week4 handin
- Final handin due by the end of the 11th
- Final showcase on May 14th, exact time TBD
  - Fill out the when2meet ASAP!

Thanks!

- You're all awesome for giving cs195u a chance
  - Really awesome
- We had a blast creating cs195u
  - Even with all the late nights spent on lectures
- We hope you all enjoyed the result!
  - Even with all the late nights spent on projects :)
- Many thanks to Chad, made cs195u possible

Playtesting

References

www.gdcvault.com/play/1911/

http://scottbilas.com/files/2002/gdc_san_jose/game_objects_slides.pdf

www.gamasutra.com/gdcarchive/2003/Duran_Alex.ppt

www.jeffongames.com/2011/02/data-driven-programming-talk

http://pages.cpsc.ucalgary.ca/~bdstephe/585_W11/d103_game_architecture.pdf

http://www.artima.com/cppsource/rvalue.html

http://www.richardlord.net/blog/what-is-an-entity-framework