Entity Compon ms

Object-oriented games

- Everything (player, enemy, tank, bullet, light, sound effect, etc) defined by its own class
- Often great dea e classes: inheritance (eg, e from Vehicle class)
- Intuitive and relatively effecti
- Can add Truck by inheriting from Vehicle
- Vehicle might inherit from PhysicalObject, as can Projectile

Object-oriented games

- Design requires detailed design of class hierarchy
- Hierarchy of increasingly abstract classes
- If hierarchy pl plementation, can build large, co heritance
- But, the deeper the class hierar re brittle and fragile it becomes:
  - Requirements change after implementation begins
  - Need to add/remove functionality to some abstract classes
  - Changes will affect all subclasses (even if they don't actually need changes)
  - End up with messy code additions pushed up towards root

Entity component games

- Core principles of good software design is modularity
- Many benefits of modularity, including flexibility
- Can replace o g to change everything, es talks to it the same way (i.e., new replacement p rms to same interface)

Entity component games

- Entity represents concrete “thing” (e.g., Tank)
  - has no Tank-specific logic
  - actually barely any logic at all (really just an ID)
  - Real magic: the
- Component is a attribute?)
  - things that Entities have (e.g., Positi enderableMesh, PhysicalBody, etc.)
- Entity little more than bag of Components
- Entity has no explicit knowledge of what parts it contains
- => All entities can be treated the same way by the rest of the game

Entity component games

- Possible because components take care of themselves, regardless of which entity they belong to
- Example:
  - RenderableMesh o render a 3D model
  - Model assigned t
  - Component assigned to entity
  - Entity calls generic Draw() function on nents, without needing to know what it does
  - RenderableMesh draws itself to display
- All the entity needs to do is call some generic update function on each of its components each frame, and each component will do its own thing

Entity component games - Pros

- Components are generic, perform single role, same way, regardless of parent entity
  - RenderableMesh of a Tank object would draw itself the same way as that of a Car ob
  - Only difference: ach component
- Different types of entities can b ured easily by plugging different reusable components into empty entity

Entity component games - Pros

- Great for maintaining flexibility during and after development
  - Changes to entities typically involve changing 1 or 2 components in isolation
  - No need to ch s or pollute other entities
  - New functionality can be added w ent addition of new components

Entity component games - Cons

- There are inherent relationships between components which requires them to be coupled in some way or other
- E.g., Renderabl nder, but not where without consulting Position c

Entity component games - Cons

- Velocity component not much good without being able to update Position component
  - Possible solution: push 'shared data' up into the Entity itself
  - all components variable
  - leads to analog itional OO hierarchy: slippery slope
  - movement also needs to know h lth entity has
  - either communicate with Health component or push health up into entity
  - and so on….

Entity component games - Cons

- Another solution: allow components to hold references to each other and communicate directly
- Multiple draw
  - couples components very tightly
  - references need to be assigned with
  - could implement some elaborate system of runtime inter-component dependency resolution and injection: extremely complex

Entity component games - Cons

- Alternative: attach message dispatching system to each entity
- Allows compon something interesting happens, and han vents of interest fired by other components
- Decouples components (nice), but comes at cost of increased complexity and persistent performance penalty

Entity component games - Cons

- Even if all components hooked up and working together, completely encapsulating logic related to each component fraught with danger of components becoming bloated with functionality
- Theoretically, Ph andle all physical interaction between its parent ent of the world, but does the knowledge of the rest of t ly belong in a component?
- Maybe split out physics calculations into a centralized physics manager: which components have logic and which don't, and how much?