John Romero Programming Proverbs

- 10. “Try to code transparently. Tell your lead and peers exactly how you are going to solve your current task and get feedback and advice. Do not treat game programming like each coder is a black box. The project could go off the rails and cause delays.”

- John Romero, “The Early Days of Id Software - John Romero @ WeAreDevelopers Conference 2017”

Recommended Reading

- Ian Millington, "Game Physics Engine Development", 2nd Edition, Morgan Kaufmann, 2010
- read chapter 1
- read chapter 3

What is a Game Physics?

- it could include the way light travels and bounces around a simulated world to make better looking graphics
- generally however it does not include the above, but refers to classical mechanics
- mass, intertia, elasticity, density, friction, velocity, acceleration of objects

What is a Game Physics?

- might include objects which can be stacked, destroyed, crumpled
  - ragdoll effects
  - fluid flow
  - flags in wind
  - rope
  - springs

What is a Physics Engine?

- in early games, physics was coded up ad-hoc, per game
  - usually consisting of crude hacks
  - memory was extremely tight, many machines did not have floating point units

- as machines become faster and had more memory developers started to use high level languages for games
  - thus the push for reusible code
  - reusible generic code
  - became natural to attempt and isolate game physics in a module or set of modules

Advantages of a Physics Engine

- development time saving
  - a couple of thousand lines of code should solve many of the common effects required
- quality of game physics
  - reliability of code behaviour
  - accuracy of physics
- separating the physics engine from the game should allow more realistic scenarios
  - as the developer does not need to enumerate each combination

Weakness of a physics engine

- speed
  - a physics engine can become a simulation
  - feature creep, more realism, more computation required
- may be fine on a desktop
  - but will be unusable on a mobile device
- development time, if you build a physics engine, be mindful that the game might be improved easily by devoting time elsewhere
- one size fits all, rarely

Approaches to Physics Engines

- first distinction is between "rigid body" and "mass aggregate" systems
- consider a crate

Rigid body systems

- our crate is a single object, simulated as a whole

Mass aggregate systems

- our crate could be considered as eight boxes
  - one per corner, connected by rods
- these systems are easier to program
  - for example rotation can be considered at the center of our crate

Contact resolution

- game engines treat contact differently
  - what happens when an object comes to rest touching another?

- consider a game engine simulating jenga

Contact resolution

- three kinds of contact resolution
  - iterative (fast, but one object might effect another)
  - jacobian based (physically realistic)
  - reduced coordinate (for a specific scenareo we are simulating)

Impulses and forces

- how does the physics engine resolve contacts?
- consider a box on a table
- a game engine using forces treats gravity as a force pulling the box down, equally the table pushing up by the same amount
- a game engine using impulses consider both table and box to be having many, many, micro collisions or impulses occurring
  - this is easier to code than the forces approach

Collision detection or prediction

- finally there different approaches on how collisions are caught
  - most common is frame based collision detection
  - after each frame each object is tested to see if it has intersected with another
  - in which case a collsion has occurred and a re-action is calculated
- alternatively the game engine can predict when a collision will occur
  - waits until this time and then calculates the re-action
- the code for reaction will be the same

Collision detection or prediction

- the implementation for frame based collision is much easier than collision prediction
- collision prediction game engines, might be more efficient, more pixel accurate
  - very complex, so much so that normally objects are very restricted
- collision prediction, works well for simulating bagatelle, or snooker
- frame based collision works well for Rage etc