Engineering at a Games Company: What do we do?

Dan White
CTO
Pipeworks
October 2022

My Goal Today

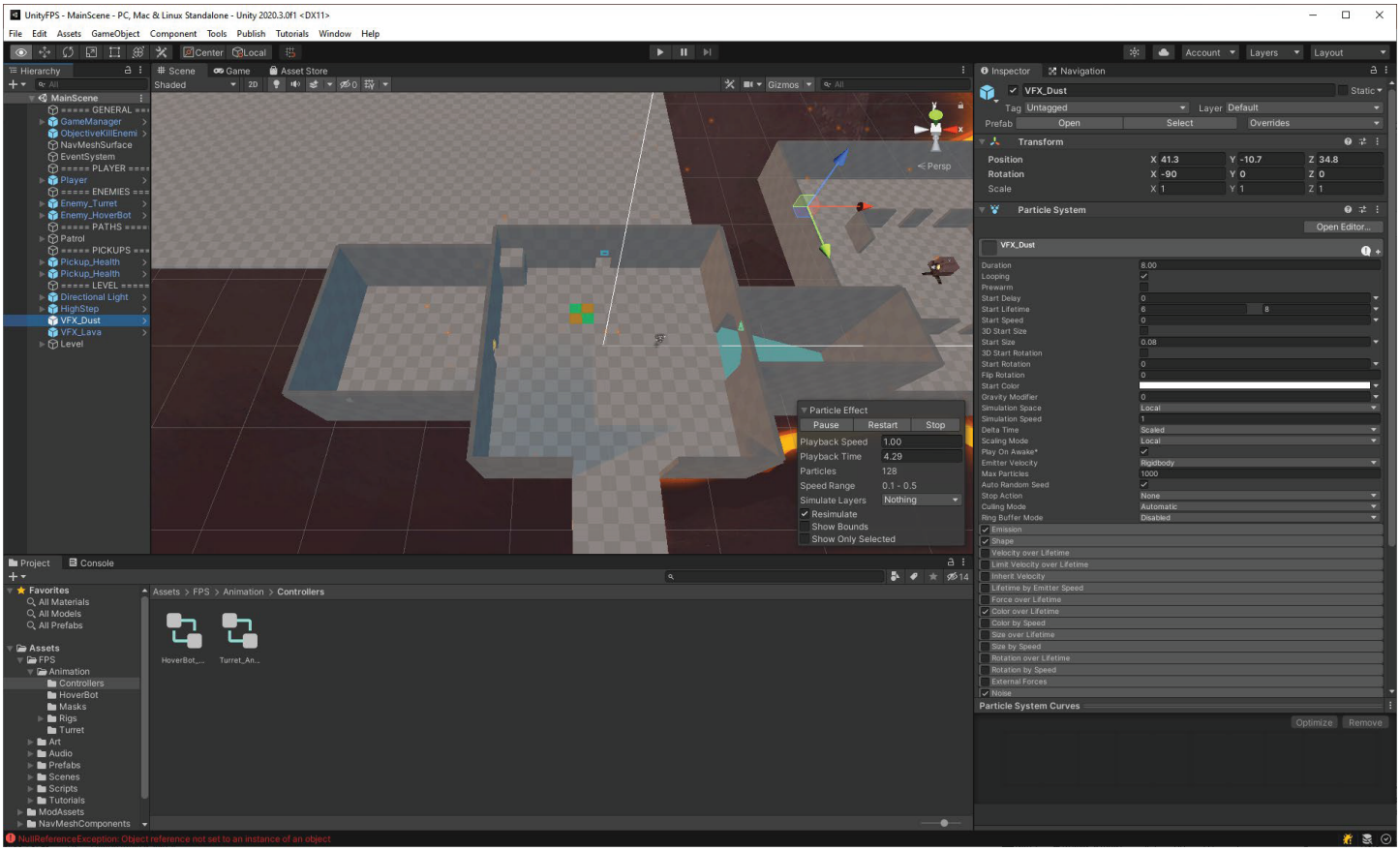
- Give you some perspective about what engineers at a commercial games company do
  - Hopefully gives you some motivation for this class
- Hobby and indie developers are different
  - But very relevant!
    - For example: Terraria, Vampire Survivors

Game Team Organization

- Three major disciplines in game companies:
  - Design
    - Figure out what the game is
    - Provide specs for key systems
    - Create content (dialog, levels, questions)
    - Balance and polish
  - Art
    - Figure out how the game looks look
    - Make assets (textures, models, animations, shaders)
    - Artists are amazing. Your exposure at OSU will be sadly limited.
  - Engineering
    - Support artists and game designers in realizing their visions
    - Make tools and systems for designers and artists to use
    - Build the things they cannot build themselves
    - Put it all together
- Other disciplines: Production, QA, Audio

What tools do teams use to make games?

- When Pipeworks started in 1999:
  - Blank hard drives
  - Visual Studio
  - 3ds Max SDK
- Now: **Game Engines**
  - Unity
  - Unreal
    - A few custom engines survive…
- Art is made with DCC tools
  - Maya, Blender, 3DS Max
  - Photoshop



Eventual Goal

- Eventually: Artists and designers will be able to use engines to make games without engineers
  - This is how it show be: You can write a novel w/o engineering support!
  - This is decades off
  - Some designers program…the line is blurry
- Right now: The game engine will provide 95+% of the code needed to make the game
  - Again, typical: 95% of the code to display a web page is provided to you
- Our job is to provide what the game engine does not and fix when it falls over

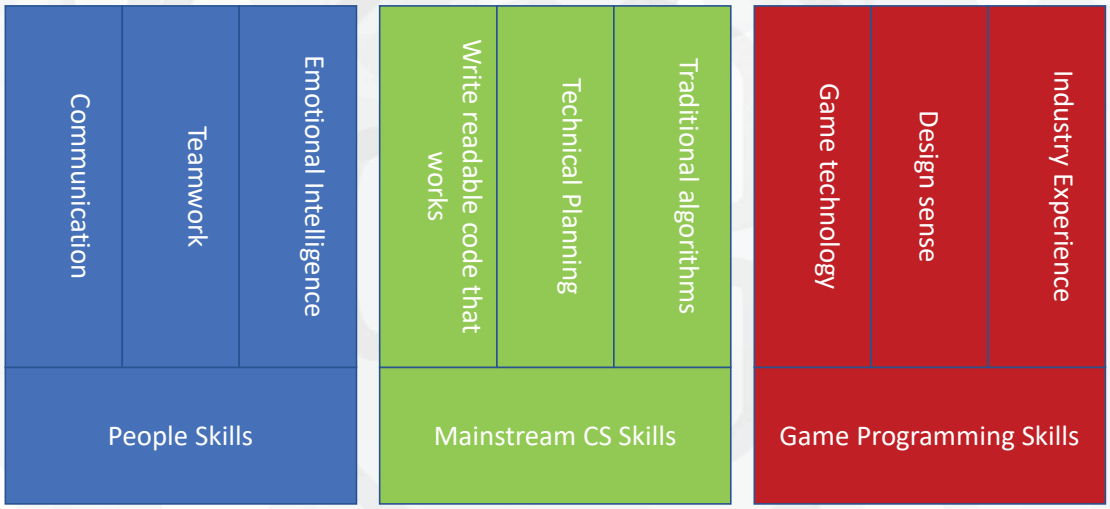
Is game programming still creative?

- Yes!
- Even more so than in the past
  - No longer making triangle rasterizers and whatnot
- The bit the game engine doesn't provide is often what makes a game unique
- Designers provide only high-level specs of features

What skill do you need?

- Work in a team with different sorts of people
  - Creative work can be difficult *because engineers don't like failure*
- Typical CS Stuff – how to write large programs
  - Software development (source control, integrations)
  - Memory management, languages
  - Algorithms, Data Structures
  - User Interface (important)
  - Discrete Math
- Other Stuff
  - Graphics
    - But not as much as you think
  - 3D Math
  - Simulation & Physics
  - Real-time networking
- I hope this class helps with the Other Stuff!
  - Useful not only for games, but machine vision, robotics, and so on

What makes a great games programmer?



What do we actually do?

- A sample of what our engineers have been working on:
  - Fix performance problems
  - Simplified physics
  - Special Graphical Techniques
  - UI
  - AI
  - Procedural Content
  - Networking
  - Back-End
  - DevOps
  - Game Code

Performance

- Engineering is not required to make high visual quality
- Dev model: Artists add stuff until there is a problem then engineers figure out why
- Better hardware doesn't help because we make more detailed content
- The goal is a consistent framerate (30hz or 60hz)
  - Stuttering can be very noticeable
  - Amortized speed doesn't count
- Rarely are perf problems fixed with just code changes
  - No more rewriting stuff in assembler
  - Shaders are an exception
- Most important thing is to understand the rendering & update pipeline to find bottlenecks
  - Solutions are often content changes, pre-calculation and so forth
  - GPU's hate state change
  - Threading when possible
- Memory bandwidth problems can dominate

Culling and Streaming

- The best way to improve performance is to load & draw less stuff
- Strategy for culling and streaming is genre dependent:
  - RTS - large number of objects, but camera typically points down
  - Interior shooter: Portals
  - Exterior open work game: zone and imposters
  - Stadium sports: doesn't matter
- This is where general purpose engines struggle the most
- Engineering defines & implements the culling strategy

Simplified Physics

- Gameplay is hard to design and time consuming to make
- Physics is gameplay for “free!”
  - Angry Birds is a demo for Box2D
- Free until it’s not – gameplay has to be predictable, performant and understandable
- Many game engines have very sophisticated physics systems
  - The math is crazy
  - Check out Bullet Physics
- Even with physics systems, engineering needed for:
  - Optimizations
  - Fractures
  - Tires/Cloth/Soft bodies
- Physics based games often do better without a complex physics simulation
  - Again, predictable/controllable behavior is the goal
  - E.g. Roller Coasters, Pinball, Driving

Networking

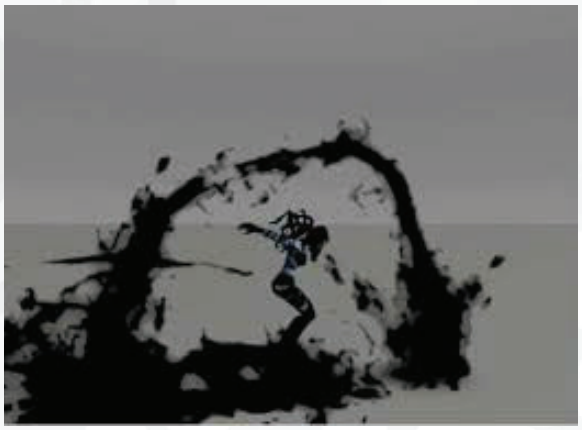
- Why multiplayer in everything?
  - Sartre: “~~Hell~~ Free content is other people”
  - One view: World of Warcraft is a themed chat room
- Synchronizing, managing & debugging a distributed simulation…hard
  - Built in engine support is often not sufficient
- Error handling
  - Most CS problems, errors are unusual
  - Everything that can go wrong, will…a lot…and users will make it worse
- TCP and typical web API’s not well-suited to games
  - Typically use UDP with some sort of reliability layer – check out Enet
- Strategy is to ration bandwidth and prioritize updates
- Always a tradeoff between latency, and accuracy
  - This will vary per-genre and per-game
  - Will use various blending strategies to smooth out updates

Back End

- Most of our games are connected to a server of some sort
- Profile storage
  - Instead of saving on disc
  - Typically, a SQL database with web front end. Access via HTTPS and REST.
- Matchmaking
  - This is a hard problem because people like to smurf
- Microtransactions
- Analytics
  - Most of the work is in the call sites, not in the analytics engine
  - Gets a bad rap, but is very positive for games overall
- Historical note:
  - Back-end connections saved the PC from Piracy

Special Graphical Techniques

- Often games have a graphical effect linked to gameplay
  - E.g. Brutal Legend
- Most shaders can be made by artists
  - DCC tools make graphics easy
  - Writing shaders is now a technical art position
  - Fixing shader performance is a graphics engineer problem
- Particle systems: Yes!



Curved World in Animal Crossing

This is done with a vertex shader-the world is flat!

Other special techniques…

Borderlands 3 – Cel Shading

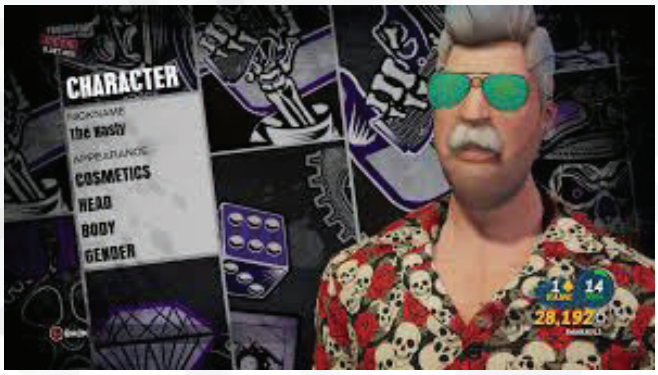
Monument Valley –
perspective rendering tricks

AI

- A famously vague term
  - Not at all what CS people mean
- For games we usually want:
  - Pathfinding
  - Satisfying opponents
  - Believable NPC's
  - Optimality not required (or even desirable)
- Usually bespoke and rule-based
  - Harder than you might think
  - Need to know rules in detail
  - Check out Steering Behaviors For Autonomous Characters
  - We have been trying to make autonomous vehicles long before it was fashionable. Good luck!
- Design needs a lot of help with AI
  - Design was control but also emergent behavior which are at odds
- A lot of interest in reinforcement learning techniques.

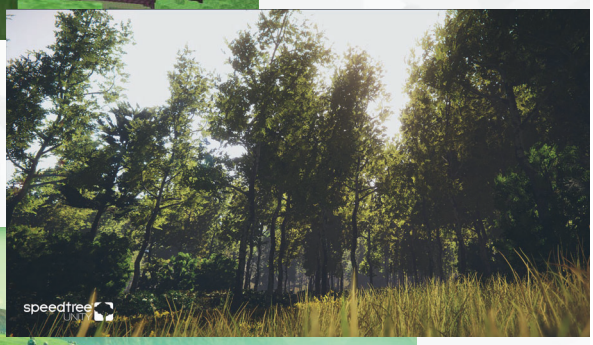
UI

- User interface is important
- Often mixes with 3d in the world
- Rendering is done by the 3d pipeline
  - Using 3d is faster than raster methods
  - Flash no more!
  - Engines have their own UI systems
- Typical Pipeline:
  - Screen mock-ups made by designers
  - Pretty is added by artists
  - Functionality is from engineering
- Lots of color, and animation and VFX
- Madden: 500 screens



Procedural Content

- Stuff that artists and designers don't make
- Allows replayability at low-cost
  - Once again: Want content for free!
- Avatar systems
  - E.g Character Creation
- User created structures
  - E.g. buildings in Fortnite
- Foliage
- Crowd and background characters
- Terrain
  - The world in Minecraft or Terraria
- Very game-specific
  - Always made by engineers



DEVOPs

- Old days: 2-year dev cycle leading to a gold master disc
  - Sooo much stress!
- Now: 18 month dev cycle, early access launch, periodic updates
- Engineering owns the build/deploy tool chain
  - Jenkins/CI etc.
  - Source control, which is notably difficult for Games
    - Git model does not work as well (but LFS helps)
- Satisfy Console and Platform requirements
  - Far more rigorous than the App Store

Game Code

- Code that implements the core play mechanic of your game
- New mechanics are very rare
  - E.g.: FPS, RTS, Racing, Fighting
  - Most are already implemented in engines or available as samples or in asset stores
    - In this case, you read and modify the existing code to fit the design
- Typically include:
  - Camera
  - Control
  - Character animation state machine
  - Game rules

Underlying Skills diff vs. Typical CS

- 3d Math
- Matrices
- Simple physics
- Blending
  - Nature is smooth
- Mesh Manipulation
- Robustness

Robustness: Floating Point is the Devil

- Traditional scientific programming tends to underplay robustness issues
- What does this return?
  - Does it even return?

```
float add_forever()
{
    float t = 0;

    while (1)
    {
        float next = t + 1.f / 30;
        if (next == t)
                break;
        t = next;
    }

    return t;
}
```

Answer

1048576.00 = 2^20 = 2^25 / 2^5

- If you update your simulation time this way, time stops after ~12 days
- Most games & graphics software runs on 32-bit float
- A big issues for flight sims and large worlds
- Safety in double is illusory

Thank You