WHAT IS GAME TECHNOLOGY?

- Technology that drives games
  - Graphics
  - Physics
  - GUI
  - Networking
  - AI
  - Sound

- Game engine connects these aspects in a coherent, organized manner
WHAT THIS COURSE IS NOT

▸ **Not** a game design course!

▸ **Not** a game development class (that’s kinda CS354p)

▸ **Not** an introduction to the basics of C++ (that’s CS354p)

▸ Thus making a game with cool systems is secondary to creating the engine that drives them

▸ But game features and systems are of course closely connected to engine implementation...
WHAT THIS COURSE IS

▸ A way to interact with a large-scale software system (specifically a game engine)

▸ An opportunity to build out common game engine features on top of an existing system

▸ An exploration of game engine features, their function, and ways to build them

▸ An environment to master team-based development and clear communication
GAME DEVELOPMENT TEAM

- System designers decide on game format and behavior
- Artists create models, textures, and animations
- Level designers create the game spaces and interactions
- Audio designers handle sounds
- Programmers write code to put everything together and create tools to make everyone else’s job easier
- And others: production, management, marketing, quality assurance
COMMUNICATIONS

- We’ll be using Discord for questions and answers to specific problems, and Discord for class communication/in-class discussion
- Please join the server so you are able to keep up on issues and ask questions
- Students should work together to solve problems before asking for teacher or TA involvement
- Grades and assignments will be done via Canvas
BOOKS AND RESOURCES

‣ Recommended “textbook”: “Game Engine Architecture” Jason Gregory
  ‣ Good exposition of many engine technology and design
  ‣ Not required but useful

‣ Other useful books:
  ‣ “Game Programming Gems 1-8”
  ‣ “3D Game Engine Design” David Eberly (lots of equations, less exposition, good math background and computer graphics)

‣ Website: www.gamasutra.com
  ‣ Game developer technical and trade news (articles may be hit or miss)

‣ GDC Vault and Siggraph archives
CLASS EXPECTATIONS

- During class we’ll explore key concepts and provide basic background info for projects.
- Ideally a time for discussion and group exploration.
  - Participation and questions highly encouraged!
- Outside of class you’ll implement this functionality in your game engine.
  - Note that this is a **programming-heavy** course!
  - Has a moderately heavy workload according to student reviews.
- Throughout the course you will encounter new technology and ideas that I won’t teach directly.
- This is a “finishing” class to help build job skills in addition to teaching core CS concepts.
THINGS I WILL SKIM OVER

- Things I will (mostly) assume you know:
  - 3D graphics concepts and programming
  - Vectors, matrices, geometric reasoning
  - C++ programming

- Things that are nice to know:
  - UI toolkits (FLTK, Glut, Qt, Interface Builder, etc)
  - 3D modeling and texturing (Maya, Substance, ZBrush, etc)
  - Scripting languages (Lua, Python, etc)
GRADING

- Projects and reports (no tests)
- 6 major projects
  - Groups of 3 assigned by the TA for projects 2-4
  - Self-forming groups allowed for the final project
- Potential periodic quizzes to check comprehension and if attendance drops
  - I will not force anyone to come to class, but I will get grumpy if no one shows up :)

Groups will be graded as one, but adjustments will be made based on individual performance

- Each group will be evaluated both on the project submissions and in-between milestones submitted via git

- We will use commits to assess how much each group member contributed to the project if there are group conflicts

For the final project, you will set your own milestones and goals

- You will be graded based on how well you achieve your goals factoring in degree of difficulty

- Each milestone will involve turning in a report
WORKING IN GROUPS

- Working in groups is an acquired skill and the most important thing you’ll learn in here!
  - For some information on group functioning, read [http://www-honors.ucdavis.edu/vohs/index.html](http://www-honors.ucdavis.edu/vohs/index.html)
- We assign teams – like in industry
- Group evaluation exercises throughout the semester will ensure an even distribution of work (and grades)
- You must evaluate teammates (even if only to say nice things about them!)
- Low performance and poor team evaluations can result in failing the class
PROJECT FORMAT

▸ To help with TA grading, your projects should run on the 3rd floor lab machines
  ▸ You **MUST** include screen capture of your program in action, a short report documenting key features and where they are implemented in the code base, and screenshots of your key code to simplify checking over the project should the build fail
  ▸ Godot projects are annoying to download via Canvas, so you **MUST** use version control for submitting your projects
    ▸ We’ll use GitLab ([www.gitlab.com](http://www.gitlab.com)), so make your repos private
    ▸ You’ll branch a “code-freeze” version for each project/milestone and submit repo information via Canvas. **Any modifications to the code-freeze branch after the project deadline will deduct from your late slips**
    ▸ Please include clear documentation on how to build your system even if it’s the default instructions (learning to write good documentation is also a skill)
THE ENGINE

- We will be using Godot for our engine development [https://godotengine.org/](https://godotengine.org/)
  - Open source under MIT license
- Godot will be built from source (rather than downloading the binary)
  - We are going to use Godot in the “engine building” way rather than the “game developer” way
  - You will be quite familiar with build systems by the end of this class
- Even if you develop on a personal machine, make sure your project and binaries runs on the lab machines
PROJECT TOOLS

- Source code control systems are essential for team projects
  - Games are asset intensive, so please use git-lfs for handling binary data
- Large software systems inevitably require using libraries and build systems
  - Cmake is very common, but Godot uses SCons
  - SCons uses Python 3 so you may need to adjust your environment variables
- If you have concerns about your code building correctly for the TA, please check before the submission deadline
TOOLS FOR CONTENT CREATION

- Models and art are the biggest expense in real games.
- This course doesn’t require outside art assets, but:
  - You can use Blender in the lab or other programs on your own machines.
  - Acknowledge any assets you download/purchase.
- Assets must be usable in the Linux environment but you can develop in non-Linux environments.
- May need to write format converters if you have a good tool that produces output that you can’t input.
- This is a big deal in the real world!
QUESTIONS ABOUT THE CLASS POLICIES OR ASSIGNMENTS?
INTERACTIVE PROGRAMMING

- A game is a user-controlled program
  - Responsive to user input in real time
  - Provides constant feedback about its state to help users understand what is happening
- Effective interaction is critical for player immersion
- How do we build software to achieve this?
EVENT-DRIVEN PROGRAMMING

› Everything happens in response to events

› Events occur asynchronously with respect to the execution of the program reacting to the event

› Events can come from users or system components

› Generated signals or messages sent to a system component

› Events, signals, and messages solve similar problems
SYSTEM-GENERATED EVENTS

- Consider: Timer events
  - Application calls a function requesting an event at a future time (e.g. next time a frame should be drawn)
  - System provides this event at the requested time
  - Application checks for and responds to the event (e.g. drawing the next frame)
USER-GENERATED EVENTS

‣ Consider: Button pressed
  
 ‣ Controller hardware sends a signal to the computer (called an interrupt)
  
 ‣ The OS responds to the interrupt by converting it to an item in an “event queue” for the windowing system
  
 ‣ Events can be kept in priority order, temporal order, etc
  
 ‣ API elements of UI toolkits check and respond to events

‣ What does it mean to check for events?
POLLING VS. WAITING

- Polling provides a call that returns immediately (non-blocking) to check if an event is pending
  - Happens whether or not there is an event
  - What do you do if there’s not one? Loop to keep checking? Go off and do something else?

- Blocking event functions wait (block) until an event has arrived
  - Only returns after the event is processed
  - What happens while your program waits? Does any work get done? Does the screen freeze up?
CALLBACKS

‣ Tell system what to do when a particular type of event arrives
  ‣ Code is executed automatically when this happens

‣ Most GUI systems operate this way

‣ Application makes a call to the GUI telling it what function to execute when the event arrives
  ‣ When a timer event arrives, the system calls a draw function
  ‣ When the left mouse button is clicked, the system calls the mouse event function
EVENT-RESPONSE CLASSES

- Two fundamental kinds of event responses:
  - Mode change events (cause the system to shift to a different mode of operation)
  - Task events (cause the system to perform a specific task within a mode of operation)

- Game software structure reflects this
  - e.g. menu system is separate from game runtime
REAL-TIME EVENT LOOPS

- Games and similar interactive systems look like an big infinite loop:
  ```
  while (1) {
      process events
      update state
      render
  }
  ```

- The number of times this loop executes per second is the **frame rate** (since each render operation creates a new frame)

- Measured in frames per second (fps)
Latency and Lag

- Latency is the time it takes from starting to do something to finishing it.
- Lag in user interaction is the latency from when a user provides input to the time they see the response.
- Controlling lag is extremely important for playability.
  - Distorts causality.
  - Causes motion sickness.
  - Makes it hard to track or target objects.
  - Makes interaction difficult.
EXAMPLE: INPUT LAG

Frame time

- process input
- update state
- render
- process input
- update state
- render
- process input
- update state
- render
BRUTE FORCING LAG

1. Pick a frame rate = 1/frame time
2. Do as much as you can in a frame time

- Faster algorithms and hardware means more can get done!
- Budgeted resources - graphics, AI, sound, physics, networking, etc - must now be done in the frame time
- Is this necessary for all resources?
Prioritizing Resources

- Priority is to reduce lag between user input and its direct consequences
  - Lag between input and other consequences may matter less
- Update different parts of the game at different rates
  - Achieve this by decoupling separable parts of the game
- This is where good software engineering practices come in!
  - Efficient software design, implementation, and algorithms allow for more content/better graphics/deeper immersion, etc!