CS354P

DR SARAH ABRAHAM

COMPILATION AND BUILD SYSTEMS

USING CONTINUOUS INTEGRATION

- Continuous integration just means we automate building the shared version rather than running the build locally/manually
    - Compilation and build stages still the same

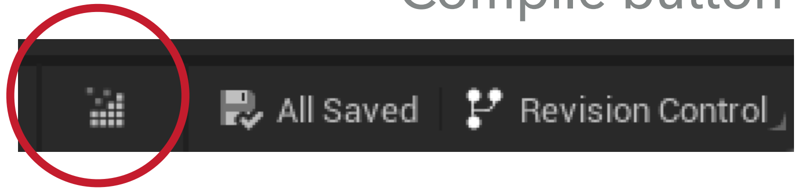
COMPILING UNREAL

- UE5 uses multiple batch files for building
    - We are going to assume .bat files for Windows but concepts should apply to OSX and Linux scripts
- These files can be run from the graphical interface or via command-line
    - Only command-line will work with containers, but we'll discuss the GUI systems first

UNREAL COMPILING AND BUILDING

- UE5 provides a GUI interface for compiling and building
  - Works for most local workflows but will not work for remote builds
- Compile button will compile all C++ files
  - Compile blueprints individually
- Build button will create desired build
  - Many options depending on what needs to be built

Compile button

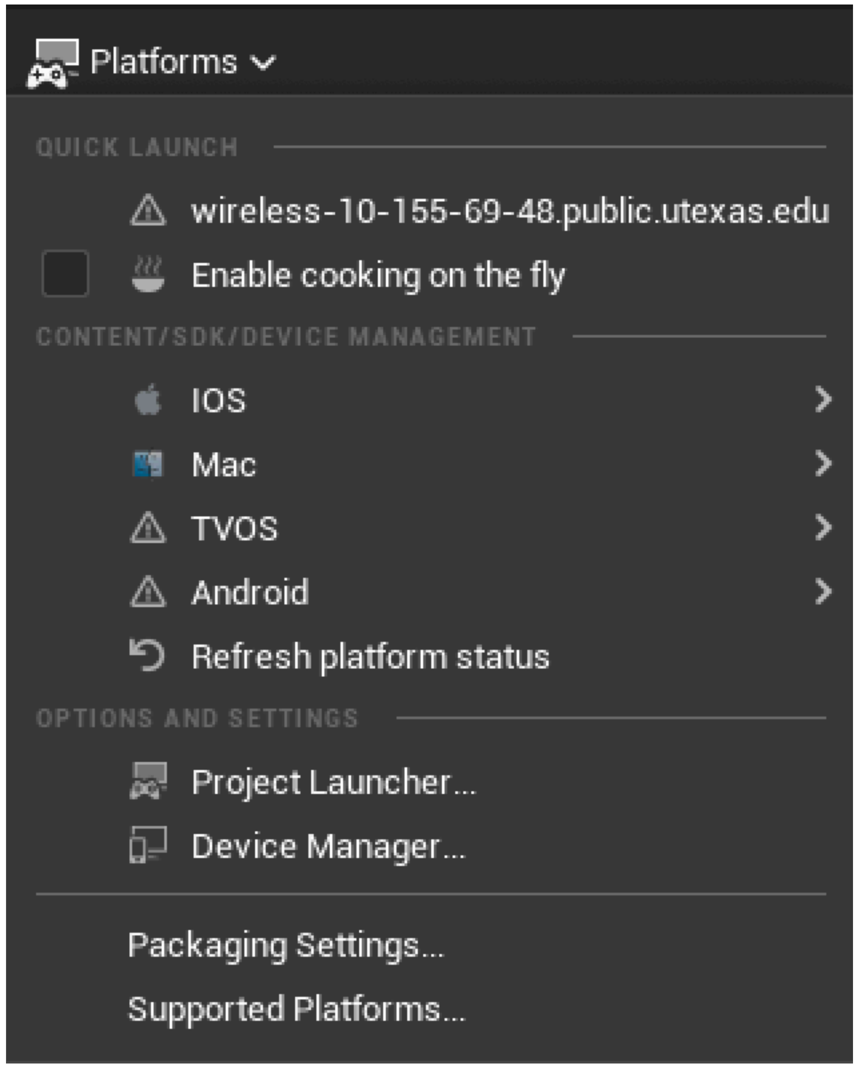


BUILD OPTIONS

- Options for building include:
  - Built Lighting Only
  - Build Geometry
  - Build Paths
  - Build LODs
  - Build Texture Streaming
- All of these are expensive graphical operations and don't need to be rerun every time!

PACKAGE PROJECT

- Project packaging is under Platforms drop down
  - Can select target platform, build configurations, and settings
- Note that just because your project compiles and runs successfully in the editor (PIE), **it does not mean it will successfully build the stand alone binary!**
  - Must use the Output Logs for debugging
  - Leave *plenty* of time for the build (it will take a long time and it may not succeed the first few tries)



COMMAND-LINE BUILDS

- Unreal Automation Tool (UAT) handles building and packaging projects and plugins
  - BuildCookRun used for building and packaging projects
  - BuildPlugin for building and packaging plugins
- Located under Engine/Build/BatchFiles within the UE5 *engine* installation
  - Note: important to keep track of where both UE5 and your projects are located on the file system

BUILD COOK RUN

- BuildCookRun script “cooks” content for a platform, packages it into native distribution format, and deploys (and possibly runs) automatically on device
  - UAT not required but very useful
- `Build` compiles executables for selected platform
- `Cook` converts assets into readable formats for the target platform
- `Stage` copies executables and content to a separate staging area
- `Package` packs project into the platform’s native distribution format
- `Deploy` builds to the target device
- `Run` starts the packaged project running on the target platform if necessary

BUILDING PLUGINS

- Same idea as building a project but a slightly different pipeline
- Plugins are collections of code that can be enabled and disabled within the Editor per-project
  - Can add runtime functionality
  - Can modify Engine features
  - Can extend Editor UI and modes

BUILDING A PIPELINE FOR AUTOMATION

- Automation is quite a bit of upfront work
  - Must create a system and pipeline to support all developers' workflow
- Smaller projects may have more ad hoc approaches but for larger projects, such pipelines become essential
  - Third-party developers are common in game dev
  - Changes in game direction and features are common
  - Employee turnover also really common :(

AUTOMATION AND CONTAINERS

- Build system must run within multiple computer environments to successfully automate
- A "container" includes code, runtime, system tools, system libraries and settings etc
  - Docker Engine is an example of this
- Containers help to isolate software from its environment, making both portability and deployment easier
  - Not always necessary but extremely useful for large, complex systems

SOFTWARE ENVIRONMENTS

- Different environments are often used for different types of builds
- Common environments:
    - Local
    - Development
    - QA
    - Staging
    - Production

LOCAL ENVIRONMENT

- Also called the Sandbox Environment
- Local workspace for an individual developer
  - May be configured to match shared environments
- Developer can experiment and implement without impacting other teammates
- Branches often used to allow for work on multiple tickets/features in entirely separate ways
- What is the advantage of separating all bug fixes and feature implementations?

DEVELOPMENT ENVIRONMENT

- Shared environment for all project contributors
  - Local environment generally matches this environment
- Place that local code is integrated into
  - Unit tests help ensure code builds correctly for all other developers
- Various types of branching/streaming schema used to integrate developer's local changes
- How would you use branching in the development environment to integrate developer changes?

UNIT TESTS

- Simplest form of testing to ensure code stability
  - Tests basic inputs and outputs of individual functions
- Automatically run every time code is integrated into the Development environment
- Try to have as much “coverage” as possible (i.e. test as many cases as possible)
- A good start but no guarantees and certainly not sufficient
- What are things you can unit test in a game?

QA ENVIRONMENT

- Also called Testing Environment
- May be closer to the Production Environment (e.g. build is for a console developer’s kit)
- Allows automated and manual tests on the product
  - Bugs and other unexpected behaviors
  - Initial stress and network testing

STAGING ENVIRONMENT

- Matches production environment to allow better integration and testing with final services
  - Connected to a live backend database
  - Running on actual servers
  - Builds run on the final platform
- Ensures all deployment configurations are correct
- Allows for more extensive load and network testing

PRODUCTION ENVIRONMENT

- The “live” environment
- In the case of backend servers, it is the code currently running on all machines
- In the case of applications, it is the fully vetted code that is ready for the final build
- This code should *never* be modified without extensive testing first…unless…
    - “Hot fixes” are changes made directly to production code and are only done in emergency situations

UE5 AUTOMATION TESTING

- UE5 does not support any automation testing within UObjects
  - Neither visible to Blueprints or the Reflection System
  - Run from the console command line in Editor
- Automation tests derive from `FAutomationTestBase`
  - Two basic types: simple and complex
- Create tests by using the appropriate macro and overriding virtual functions
- We will discuss these at greater length later in the semester…