INTRODUCTION TO UNITY

CS378

DR SARAH ABRAHAM
HISTORY OF UNITY

- First launched in 2005 as an OSX exclusive game engine but quickly supported multi-platform development
  - Intended to make game development more accessible to general developers
- Widely used for iOS development in 2008
- Unity 5 was released in 2015 and introduced more “advanced” features such as global illumination
  - Further pushes to take “Triple A” market in addition to mobile, indie, and hobbist spaces
- Moved to year versions as of 2017
WHAT MAKES UNITY DIFFERENT?
A VERY BRIEF HISTORY OF GAME ENGINES..

- Aughths were a very different time in game development space...
  - Transitioning out of 90s-style “group of friends” development but not quite “modern”

- Numerous, specialty engines/frameworks available for non-Triple A space:
  - GameMaker
  - RPGMaker
  - RenPy
  - XNA

- Triple A engines such as Unreal 3, Frostbite, CryEngine available for large (paying) studios
  - But many game studios working on top of “in-house” engines
ENGINES VERSUS FRAMEWORKS

- Game engines refer to a holistic, collection of functionality for subtasks
  - Subtasks: Level editors, asset management, physics, graphics, AI, GUI, etc...

- Game frameworks refer to libraries that provide underlying functionality but do not unify this functionality
  - Examples: Monogame/XNA, Cocos2d, Phaser, etc...

- Game engines provide more structure, game frameworks provide more flexibility
  - Early development scope much smaller making both feasible
ENTER UNITY...

- Unity niche is as a “general purpose” game engine
  - Easier to work with than lower-level frameworks
  - More flexible than “speciality” engines like GameMaker and RPGMaker
  - Many of the features of Triple A game engines without the complex software architecture
Should look familiar...
UNITY ENGINE

- Written in C++ (but underlying systems hidden from users unless a certain tier of paying client)
  - Less boilerplate to setup
  - Less system knowledge required to get started
- Main interface is level editor (no programming) and C# scripting (programming)
  - Easier to rapidly prototype and develop
C# LANGUAGE

- C# is a strongly-typed object-oriented language created as part of Microsoft’s .NET framework
- Compiled by .NET’s Common Language Runtime (CLR) VM
  - First compiled to Intermediate Language (IL) then just-in-time (JIT) compiled into machine code at runtime
UNITY GAME OBJECTS

- GameObject is base class for all entities in Unity scenes
- Component is base class for all objects attached to GameObjects
- Behaviours are Components that can be enabled or disabled
- MonoBehaviour derives from Behaviour and is the base class for Unity scripts
GAMEOBJECTS AND COMPONENTS

- Unity uses a component-based paradigm
- GameObject provides basic memory management and object tracking
- Functionality created by attaching Components
  - Components can only be called from a GameObject
  - ScriptableObject allows the execution of scripts without a GameObject
- Scene hierarchy built out of nested GameObjects
**PREFABS**

- GameObjects can be stored as prefabs
  - Contain all the components, property values and child objects
- Prefab instances can be placed into scenes
- Prefabs allow for the reuse of assets
  - Modifications to a prefab update associate instances
  - Possible to override instances of a prefab or create variant groups
- Prefabs can be instantiated at runtime using `Instantiate()`
MONOBEHAVIOUR

- MonoBehaviour is Unity’s base class for scripting system
- Provides basic functionality for game logic
  - e.g. Start(), Update(), Awake(), OnDestroy() etc..
- Only MonoBehaviour objects can be displayed in Unity Editor
- Possible to combine MonoBehaviour and standard C#
  - C# necessary for multi-threading and encouraged for event-handling
COROUTINES AND INVOKE

- Unity emulates concurrency with Coroutines and Invokes
- Coroutines “yield” after running once per Update
  - Possible to create interpolations by creating a Coroutine that runs a certain number of times
- Invokes are similar to timers but not performant because not event-based
- C# has both Timers and Events but Unity does not use them
  - Presumably related to their thread management system
- For frequent calls, putting code in Update may be more performant than either
  - Coroutines are there for code readability, I guess?
UNITY EVENTS

- UnityEvents aren’t...actually events
  - Called using Invoke, which is neither performant nor event-based
- Take 2.5x to 40x longer than native C# Events
- Useful if you need to have access to “events” in the Editor
  - More designer-friendly
- Also uses weak references for better GC allocation
  - Safer since you don’t have to manage delegates directly
- Less footprint with multiple listeners
PERFORMANCE BENCHMARK ON DELEGATES VS INVOKE

https://www.reddit.com/r/Unity3D/comments/35sa5h/unityevent_vs_delegate_event_benchmark_for_those
PERFORMANCE ANALYSIS BASED ON NUMBER OF LISTENERS

Event Performance (1 arg)

Event Performance (0 args)

Event Performance (2 args)
NAVIGATING THE SCENE HIERARCHY

- Possible to navigate both a GameObject hierarchy and access arbitrary game objects within a scene via the Tag system.

- `GetComponent<
ComponentType>()` provides access to the first Component of that type found within a GameObject.

  - `GetComponents<
ComponentType>()` accesses all the Components of that type.

- `FindWithTag("tag")` returns one object tagged with "tag".

  - `FindGameObjectsWithTag("tag")` returns an array of objects.

  - Preferred to `Find()` which does not require a tag for performance reasons.
WHAT ABOUT STORING PERSISTENT DATA?
SINGLETONS REVISITED

- No built-in equivalent to UE4’s GameInstance class
- Standard solution is use of singleton Manager classes
  - Convenient way to store and access data between scenes
- Side not:
  - Possible to save GameObjects between scenes using `DontDestroyOnLoad()`
  - Can also store per-session data in `ScriptableObjects`, which live in Assets directory
WORKING WITH SINGLETONS IN UNITY

- Singletons have the same problems in Unity as they do anywhere else!
  - Global state leads to state management issues
- Possible to create a Manager of Managers to mitigate state issues much like UE4’s GameInstance
- Also possible to use Service Locators
Service Locator Pattern allows the manager (service locator) to determine the requested service and return necessary information.
IMPLEMENTING SERVICE LOCATORS

- Service Locator stores list of all possible services.
- Requests to process a service sent to the Service Locator via that service’s interface.
  - Hides details of service implementation from user.
- Service Locator finds associated service in its list of services and updates the necessary state.
MOVING TO A DATA-DRIVEN MODEL...

- Unity is currently rebuilding engine from ground up to be data-driven rather than object-oriented
  - Data-oriented Technology Stack (DOTS)
- Intended to be more performant with better multi-threading models
- Move from GameObjects to entities and move to full Entity-Component System
- Currently very much in alpha...
  - It’s looking like 2022 at earliest?
REFERENCES

- Event Performance: C# vs UnityEvent <https://www.jacksondunstan.com/articles/3335>
- Unity DOTS <https://unity.com/dots>