

08 Game and Engine Architecture

Tvorba a dizajn počítačových hier
Návrh a vývoj počítačových hier

Game development is software development

- Use of well-defined design patterns
- Relatively strict structure of a game and game engine
- Not only graphics and physics
- Although lots of concepts are borrowed/required from these fields

Game architecture

- What are all the parts a game consists of?
- How do they fit together?
- Relying on established standards will make your life easier
- You will create maintainable and reusable software pieces
- Instead of doing something that immediately works, you need to think about the one constant in software development
- Going against the stream might pay off in terms of efficiency
- But the results might be harder to comprehend by others

**What's the only constant
thing in software
development?**

Change

Game architecture layers

- Architecture with lots of layers (think TCP/IP)
- Every subsystem can be put into one of these categories:
 - Application layer
 - Deals with the hardware and the operating system
 - This is usually handled by the game engine
 - Game logic layer
 - Manages your game state and how it changes over time
 - Game view layer
 - Presents the game state with graphics and sound
- Similar to the well-known design pattern Model-View-Controller
- Changes in hardware should not affect the game logic or game view layers
 - Just like MVC

Game logic layer

- This is your game – all its mechanics
 - Without input systems, rendering & audio playback...
- Contains subsystems that manage the game world state
- Communicating state changes to other systems
 - Such as playing a sound when you fire a gun
 - Or playing an animation for the gun as it fires
- Also systems that enforce rules of your game world
 - Physics system
 - ...

Game view layer

- Presenting the game state to the user
- Translating input into game commands
- As many game view implementations as needed per game
- Not only drawing the game state on the screen
- Other views include:
 - AI agents get a “view” of the game state
 - A remote player gets a view of the game state
 - The state observed from game logic is the same
 - But they do different things

An example – racing game

- How will the game logic and game views look?
- Game logic
 - Holds the data that describes cars and tracks
 - Car weight distribution, engine performance, tire performance, fuel efficiency, ...
 - Track shape, surface properties
 - Physics that controls what happens to cars in various states
 - Input is only regarding what the actual driver does
 - Steering, acceleration, braking
 - If we add guns, also some trigger to fire the guns
 - Nothing else is needed to move the cars
 - Output
 - State changes and events
 - Car and wheel positions and orientations, damage stats, how much ammo is left
 - Events could be car collisions, passing checkpoints, ...

An example – racing game (2)

- We provide two game views
- Human view
 - A lot of work to do to output video and audio
 - Draw the scene, spawn particles for particle effects, play audio, force feedback
 - Also reads the input devices
 - “Accelerator at 100%”
 - “Steer left”
 - Sends these commands back to the game logic
 - What happens when I press SPACE (emergency brake)
 - The view sends a message to game logic
 - Game logic sets the emergencyBrakeOn to true
 - Game logic notifies the view that state changed (for a racing game, it happens even without input if any of the cars are moving)
 - The view responds by playing a sound and spawning a dust particle effect on tires

An example – racing game (3)

- AI game view
 - Receives the same game state and events as the human view
 - Which track, weather, car positions and orientations
 - Can react in response to events (such as “Go!”) by sending info to the game logic
 - Set accelerator to 100%
 - Steer left at 50%
 - Commands remain the same!

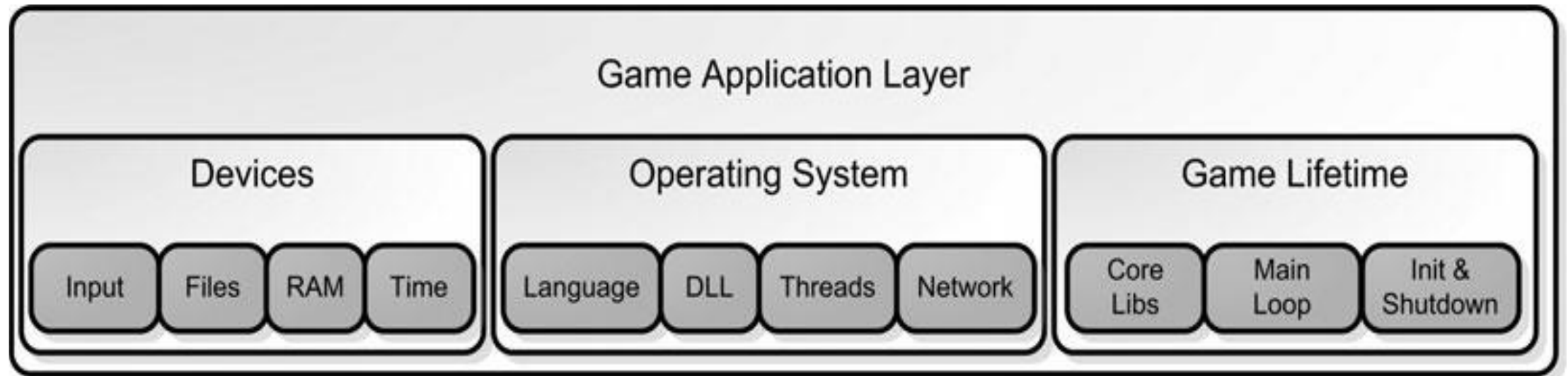
Game views

- Very flexible
 - You can have any number of human or AI views
 - Trivial to swap humans with AI and vice versa
- You could create a game view that just records game events into a buffer
 - You might replay them later
 - The game logic is kind of disabled during replay, since a view is sending all the events
 - With a little extra work, you can get a rewind feature working
 - Need to handle undoing events (continuous Memento pattern?)
- Or a special game view that forwards game status to a remote player
 - Handle network logic with regard to game events
 - Pack and send
 - Receive input, unpack, create events

Game views (2)

- Views can give advantages or disadvantages
 - 4:3 aspect ratio might give smaller field of view than 16:9 aspect ratio
 - AI might know more about game state than the player (see through walls)
- A little hard to get used to
- Unity does not offer a strict separation of views from logic implicitly
- Using it explicitly will make more maintainable code
- Even if you do not strictly separate, it's good practice to know which layers are communicating

Application layer



Reading input

- Need to provide a layer between the OS and the rest of the game application layer
- The state is translated into game commands
- Should be configurable
- Game state should never change directly from reading user input
 - Not flexible
 - Lots of changes when controls change
- Unity offers both ways
 - `Input.GetKey()` is the direct way
 - `Input.GetAxis()` / `Input.GetButton()` is indirect and configurable inside the editor or during runtime

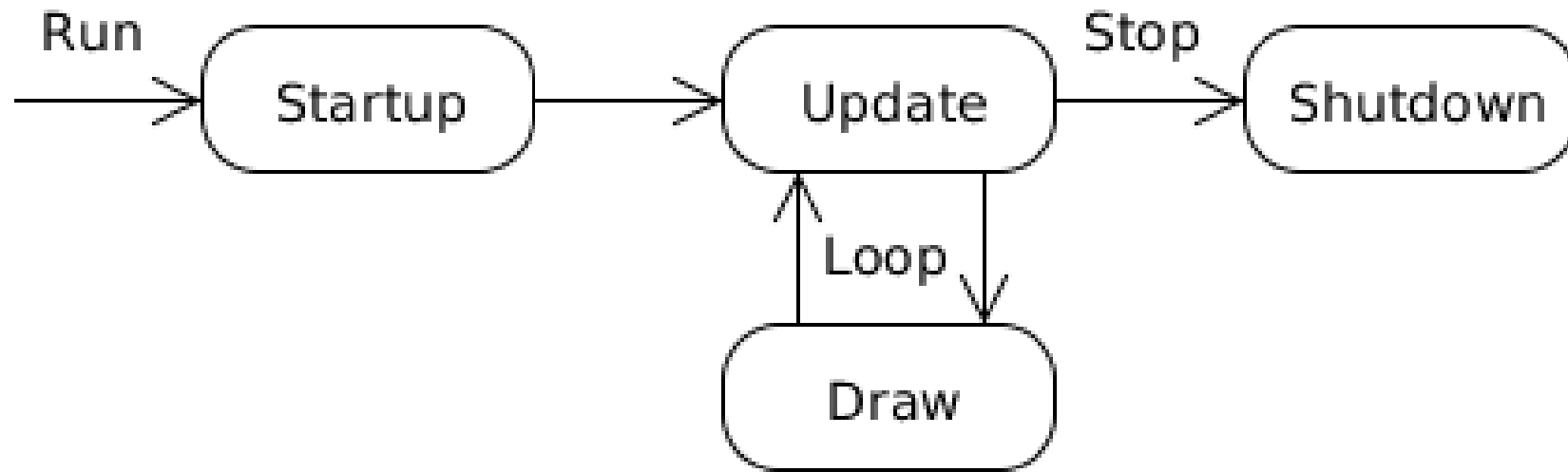
File System and Resource Caching

- Reading and writing from disk and other storage media
- Managing resource files can be complicated
- One of the hidden systems is the resource cache
 - Commonly used assets are always in memory
 - Rarely used assets are in memory only when needed (end-game video)
 - The resource cache tries to “fool” the game into thinking that all the assets are available in memory
 - If all goes well, the cache can load files **before** they are needed
 - `SceneManager.LoadSceneAsync()`
 - Cache misses might occur if it fails to load something in time
 - Solve by loading screens or small lags

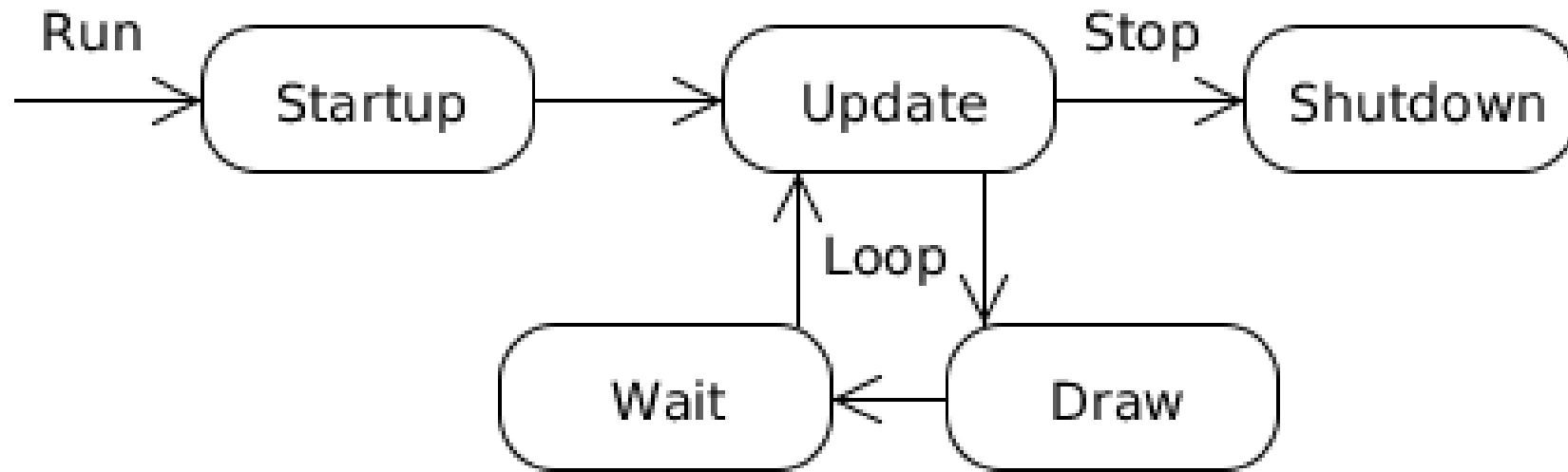
Initialization, Main Loop, and Shutdown

- Most software waits for user interaction, doing almost nothing
 - Can have lots of these running with minimal overhead
- Games are simulations that have a life of their own
 - Player input is not required for the game to continue simulation
- The system controlling the game simulation is the **main loop** or **game loop**
- Usually has three stages
 - Grabbing user input
 - Updating game logic
 - Presenting the state through all views
 - Rendering, playing sounds, sending state over the internet

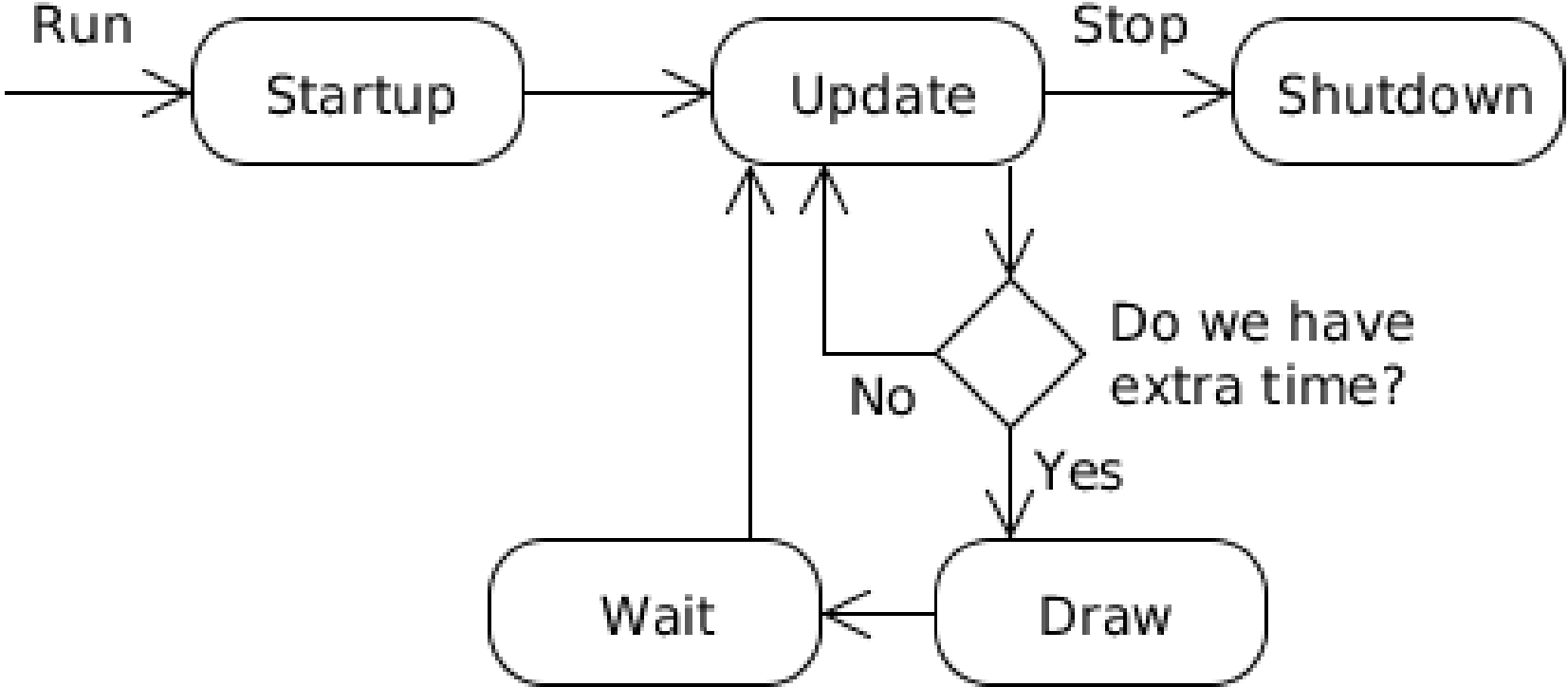
Simple game loop



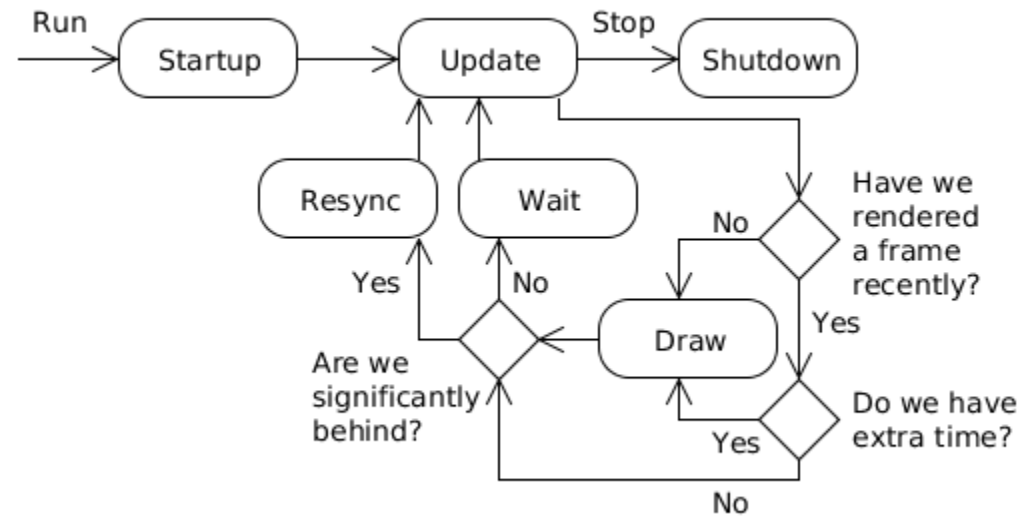
FPS limited game loop



Advanced game loop



Very advanced game loop



Game loop in Unity

- Game loop handled implicitly
- Querying user input is handled by the engine
- Game programmers are given two main callbacks
 - `Update()`
 - Updates as often as possible
 - Called once for each frame rendered
 - `FixedUpdate()`
 - Updates in fixed intervals
 - Called once for each physics engine step
- Rendering is done by the engine as well!
 - We only set what will be rendered
 - The actual rendering is executed somewhere inside Unity
 - Exception: Using low-level rendering access with the `GL` or `Graphics` class

How Unity game loop looks (probably)

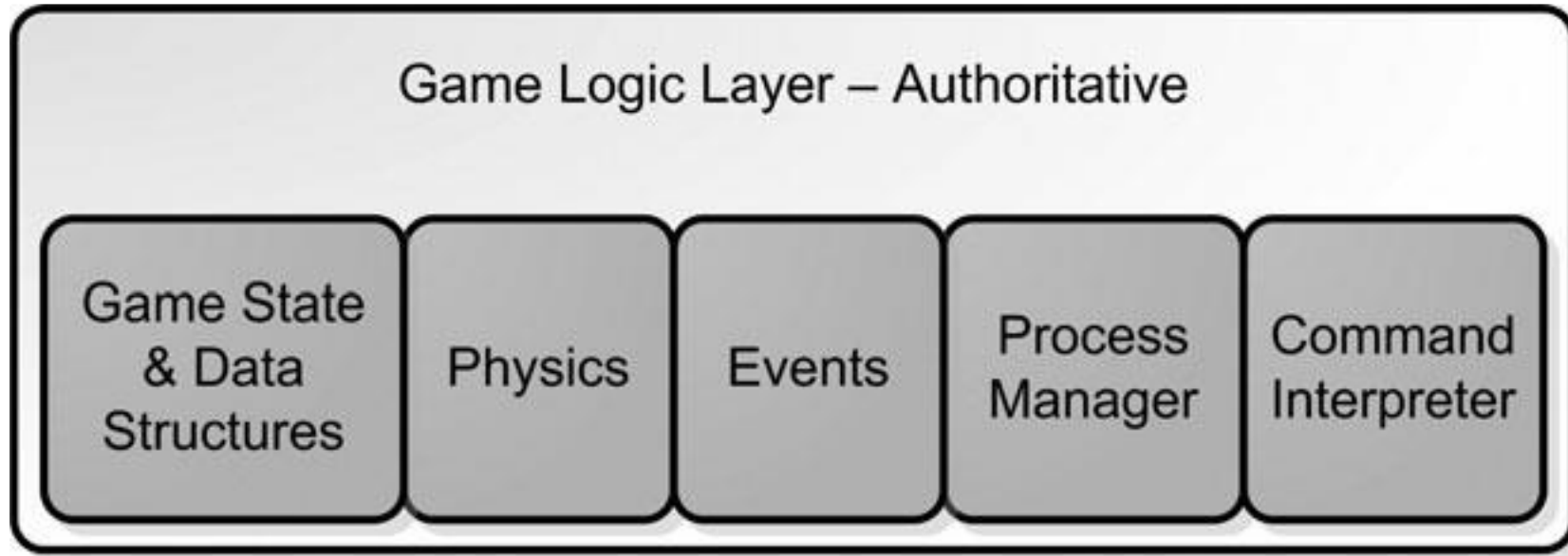
```
float timer = 0;
while (true)
{
    while (timer > fixedTimeStep)
    {
        FixedUpdate();
        timer -= fixedTimeStep;
    }
    Update();
    LateUpdate();
    Render();
    timer += deltaTime;
}
```

<http://docs.unity3d.com/Manual/ExecutionOrder.html>

Other Application Layer Code

- System clock
- String handling
- DLLs
- Threads and thread synchronization
- Network communication
- Initialization
- Shutdown
- (Scripting language)

Game logic



Game logic

- Defines the game universe
- What things (entities) are there
- How they interact
- Defines how game state can be changed by external stimulus

Game state and data structures

- A game needs to store its game objects in a container
 - Must be able to traverse quickly to change game state
 - Should be flexible as to what data it will hold
 - Special game data (hitpoints, inventory, ...) are stored in some custom data structure
- In Unity:
 - Upgraded scene graph hierarchy
 - Retrieve objects either by pointing directly to them, or special functions
 - Static function part of GameObject:
 - `GameObject.Find()`
 - `GameObject.FindWithTag()`
 - `GameObject.FindObjectsWithTag()`
 - Implicitly traverses a tree or a hash table to efficiently find the objects
 - Special game data stored as public variables of script components

Game state and data structures

- Easy to confuse game logic representation with the visual representation of objects
 - Amount of damage a weapon deals is stored in game logic
 - The weapon's model, textures, icons are only relevant to the game view
- Another example:
 - Skeletal mesh object that is used for skinning the character when rendering
 - It seems that it has something to do with the character's weight
 - Skeletal mesh – view, weight – logic (probably for physics calculations)

Entity-Component-System (ECS)

- Design pattern used to store and manipulate game state
- Favor composition over inheritance
- Entities consist of Components
- Each component has a single responsibility
- Systems run in background and handle component changes
- One system serves only one purpose
 - Physics System, Player Damage System
- Components register in one or more systems
- Entities are affected indirectly (through their components)

ECS in current Unity

- Entity = GameObject
- Component = Component
- System
 - Configurable but not directly visible
 - Can create own systems
 - Individual components register with their respective systems
 - **Split across multiple components**
- Examples
 - Collider and Rigidbody register with the Physics System
 - MeshRenderer, Camera, Light register with the Rendering System
 - Scripts register with the Scripting System and Event System
 - You create other systems through scripts – Player Damage System, Items & Inventory...

New ECS in Unity (aka. DOTS)

- Entity = just an ID (64-bit int)
- Component = component of an entity, holds only data
 - **Does not hold functionality**
- System = Programmable
 - **Has all the functionality**
 - Iterates over all components that it works with
- Hybrid => Each game object is also an entity
- Pure => Screw game objects, just use entities
- MVC again?
 - Entity+Components = Model
 - Visual/Audio Components + Rendering System = View
 - System = Controller
- This is all in alpha preview – DO NOT USE YET!

Physics and Collision

- Rules of your game universe
- No need for over-complicated physics to make a game fun
- If our game is completely abstract, unrealistic physics will not be disturbing the player that much
- If, however, we simulate the real world, small errors might cause players to be very disturbed by those errors
- Usually rigid body dynamics with dynamic objects being mostly convex
- Few exceptions:
 - Ragdoll physics
 - Fluid simulation
 - ...

Game Events

- When the game state changes, a lot of other systems need to react accordingly
- Game logic is responsible for generating these events and passing them further
- Subsystems register with the Event Manager to listen to events that they react to
- Clean and efficient separation of unrelated code with a simple abstraction and the use of a unified event interface
- The Observer pattern

Process Manager

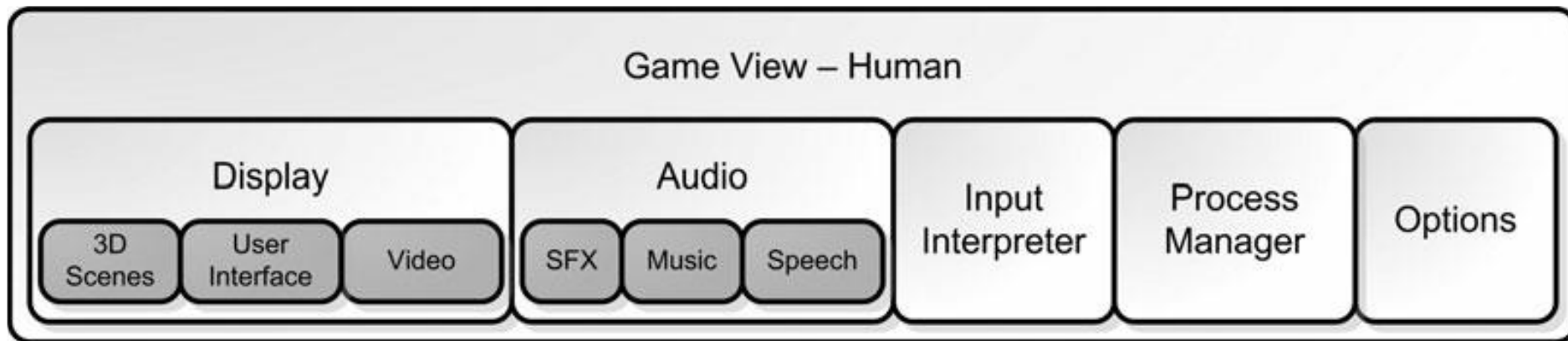
- The game logic is composed of small parts of code that need to be executed periodically for the game to work correctly
- Multiple processes that are completely independent (but we might provide dependencies)
- Mostly script executions
 - If we have a scripting language
 - Otherwise, it's just one class per operation to keep things simple
- Chaining processes together (throwing a grenade, explode on collision)
- Unity handles this by calling the Update() (or other event) functions on all script components attached to currently enabled game objects in the scene
 - You may sometimes come up with code that needs a certain order of execution, **AVOID IT as much as you can**

Command Interpreter

- Provides a good separation of the game view and game logic
- Need to interpret commands sent by AI or human players
- Unified interface, no mixing of unrelated code
- Provides more efficient debugging
 - You can send separate commands while debugging
 - This is usually what consoles are for
 - Not the hardware, the Counter-Strike console
 - The Elder Scrolls also had a console
 - As well as many others

Game view - Human

- Views are a collection of systems that communicate with game logic to present the game to an observer
- Observers can be human, AI,...
- The view responds to game events as well as controller input
- Works as a translator component that outputs game commands on one side and the presentation of the game on the other side



Graphics Display

- Renders the objects in the scene
- Renders the user interface (HUD)
- Must draw the scene as fast as possible
- Lots of problems
 - Which objects to draw
 - How to handle complex transforms
 - How to handle complex visual effects
 - Handles animation interpolation
 - Lighting conditions
 - Post-processing
 - Level-of-detail
 - ...

Graphics Display (2)

- For very complex 3D scenes, need a lot of pre-computation
 - Light maps
 - Potentially visible sets (PVS)
 - Light Probes
- **The artist must be aware of the game engine capabilities**
- Lots of constraints
- Unity tries to handle all of this
- You might reduce performance of the game without even knowing it

Audio

- Playing sounds
- Three main areas
 - Sound effects
 - Pretty simple, when an event occurs, play some audio
 - Music
 - A little harder when done right
 - Tone of music adjusting according to game situation (Elder Scrolls, Halo, ...)
 - Speech
 - Very tricky
 - Lip-sync and storage of lots of sounds are the main problems
- Take into account 3D audio
 - 3D positions of listener and sources

Audio in Unity

- AudioSource and AudioListener components
- AudioManager, groups, filters, effects
- Can use other audio engine – FMOD, Wwise...

User Interface Presentation

- Every UI must be very specific and adjusted to the game
- Re-using components is possible, but only to a certain extent
- Sometimes you need a completely new GUI component to fit the game
 - A compass, special inventory, ...

UI in Unity

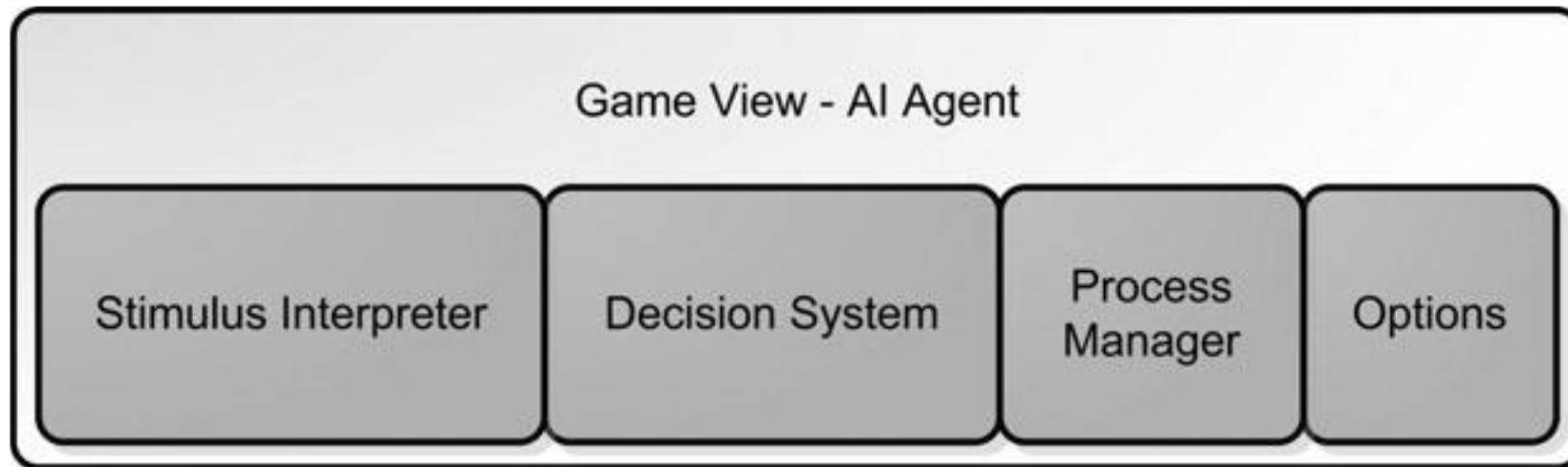
- Uses separate rendering from 2D and 3D
 - All components rendered on a Canvas
 - Objects use 2D sprites for rendering
- Has numerous tools for anchoring, scaling for different resolutions, events...
- Can do screen-space as well as world-space UI
 - Can even mix UI rendering and standard rendering

Options

- Configuring the view
- Resolution, aspect ratio
- Controls
- Sound effects
- Graphics quality and performance

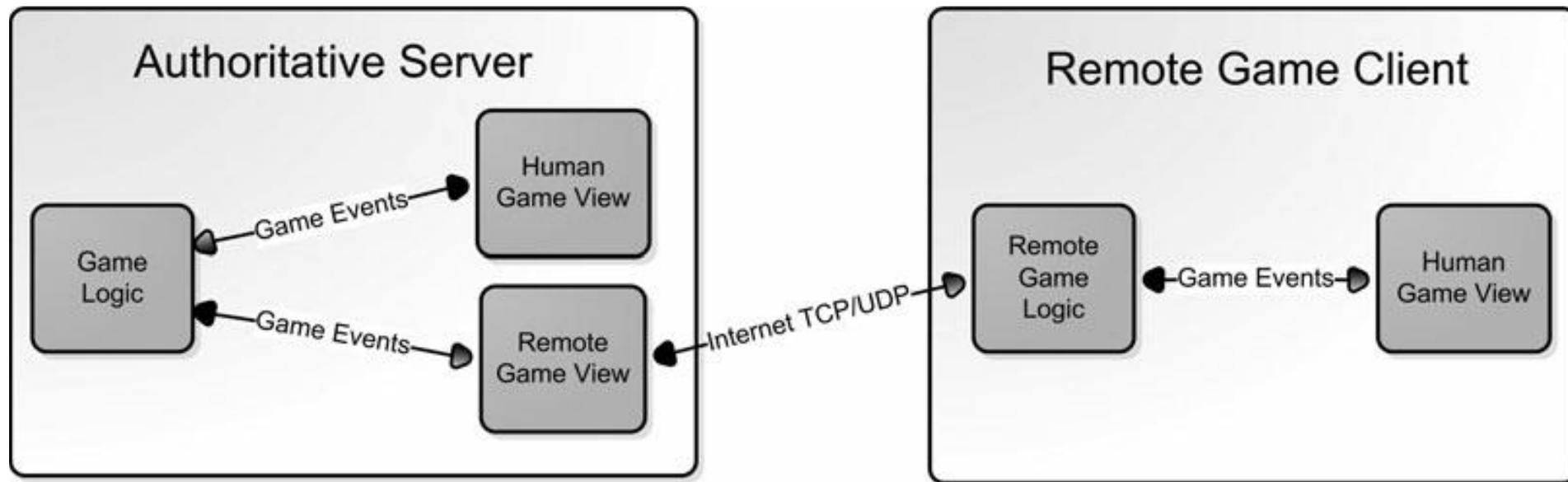
Game view - AI

- Stimulus interpreter receives all game events
- Up to the AI programmer to choose which events will be ignored, which it should react to etc.
- A blind AI ignores that the enemy is in front of him
 - Even though it will receive the event
- The decision system uses advanced AI techniques to determine what the AI will do next (might be a finite state machine)



Upgrading to a networked game

- We implement two additional classes to enable networking
 - Remote game view
 - Remote game logic
 - Proxy design pattern



Remote Game View

- On the server, a remote player is just like an AI agent
- What happens inside the game view is totally different
- Game events are packed up and sent via TCP or UDP to a client
 - Need to compress data, select only important game events
 - No need to send “object moved” if several such events occurred since the last packet was sent, we send only the most recent
- Receives game commands from the client
 - Should not trust these entirely
 - Need sanity checks to prevent hacking
 - After filtering impossible commands, they are passed on to the game logic

Remote Game Logic

- The game logic is an authoritative server, it represents the real game state
- Clients need a copy of the game state to be able to present the player the game correctly
 - The job of Remote Game Logic
- They also need to account for network delays and network errors
- It is similar to the server game logic
- There is a need to simulate the game without receiving events from the server
 - Reduce bandwidth
- Allow for “against the rules” corrections when the server sends the correct data

References

- McShaffry, M. & Graham, D. (2013). *Game coding complete*. Boston, Mass: Course Technology PTR. 4th ed.
 - Chapter 2 – What's in a game?
- <http://gameprogrammingpatterns.com/>
- <http://entropyinteractive.com/2011/02/game-engine-design-the-game-loop/>