
gcap
games
connect
asia pacific
EMPOWER

Thinking In Data: Leveraging Unity's Data-Oriented Tech Stack for AI

By Morgan Wall, Lead Engineer at Gameloft

linkedin.com/in/morgan-wall/

github.com/morgan-wall/

Shared Resources

Git Hub

Contents

Prerequisites

- Experience with:
  - Unity
  - C# or C/C++
- An understanding of:
  - Object-Oriented Programming (OOP)
  - Basic CPU features (caches, cores, etc)

About Me

About Me

- Lead engineer at Gameloft
- Gameplay and AI specialist
- Blizzard
  - *Hearthstone*
- Defiant Development
  - *Hand of Fate 2*
  - *The World in My Attic*
- Playcorp Studios
  - *Beyond Contact*

DOTS Experience

- Shipped *Beyond Contact*
- Multiplayer survival crafting
- Used jobs, burst, and ECS
  - Decision making
  - Action execution
  - Perception and stimuli
  - Relationship management

Object-Oriented Programming

Object-Oriented Programming (OOP)

- A human-centric approach to programming at scale
- Chiefly defined by four key principles
  - Abstraction, encapsulation, inheritance, polymorphism
- It is a go-to methodology for game development
  - Unity, Unreal, Godot, CryEngine, Amazon Lumberyard, etc
- It is supported throughout many languages
  - C++, C#, Java, Python, JavaScript (via prototypes), etc
- Despite this, it has seen a great deal of criticism for decades

Issues with OOP

- We're going to focus on the issues with human-centricity
  - We are incorrectly focused on abstractions and concepts
    - These ideas abstract away the hardware
  - We ignore how computers manage memory
    - Data is fragmented across classes and objects
    - Data is laid out according to an inheritance hierarchy
  - We ignore how computers manage instructions
    - Functions operate on individual object instances
      - Especially via polymorphism

Data-Oriented Programming

Aside: Computer Memory Refresher

- Memory exists in a hierarchy of access speed
  - Tertiary (removable media / mediums, USBs, CDs, etc)
  - Secondary (HDDs, SSDs, etc)
  - Primary (RAM, CPU caches (L1, L2, etc), registers, etc)
- Used to store instructions and data to operate upon
- Latency is reduced by accessing memory sequentially
  - E.g., linked lists vs arrays, cache misses, etc
- Computers employ tricks to enhance performance
  - E.g., data / instruction prefetching, branch prediction, etc

Aside: Computer Memory Refresher

Computer Architecture Operation	Duration (in nanoseconds)
L1 cache reference	0.5
Branch mispredict	5
L2 cache reference	7
Mutex lock / unlock	100
Main memory reference	100
Send 2 KB over 1 Gbps network	20,000
Read 1 MB sequentially from memory	250,000
Disk seek	10,000,000

Helpful Resource: Jeff Dean's 12 Numbers

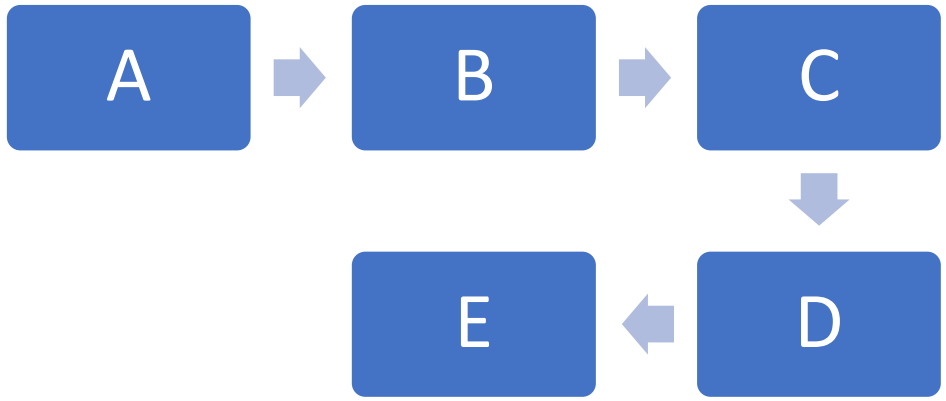
Data-Oriented Programming (DOP)

- A computer-centric approach to programming at scale
- It shifts the programmer from objects to the actual data
  - The type of the data
    - Mutable or immutable
    - Managed or unmanaged
  - The layout of the data in memory
    - Contiguous rather than fragmented
  - How it will be read and processed by the CPU
    - Locality, caching, and parallelisation (incl. instructions)

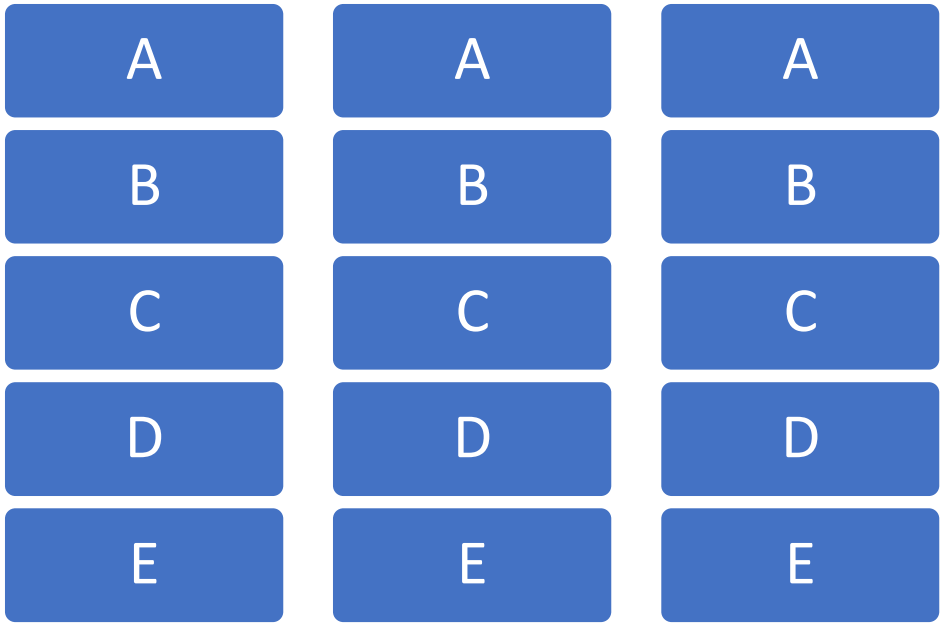
Data-Oriented Programming (DOP)

- The execution of this paradigm has a shared set of principles
  - Like data is stored in contiguous chunks for caching
  - Like data is loaded and processed in batches
  - Data is flagged as im/mutable for safety and efficiency
    - It's explicit when data can be shared across threads

Figure A: Call Sequence Comparison



Call sequence: A, B, C, D, E, A, B, C, D, E …



Call sequence: A, A, A, B, B, B, C, C, C, D, D, D, E, E, E

The Data-Oriented Tech Stack

The Core Tech

- C# job system
- Burst compiler
- Entity component system

The C# Job System

- Empowers you to write safe multithreaded code
  - Runs your software on available CPU cores
  - Unity typically executes your code on a single thread
    - Known as the main thread
  - Jobs are run on multiple worker threads instead

Helpful Resource: [The Official Job System Documentation]

Figure B: A Function Evaluator Job

```
public struct QuadraticFunctionEvaluatorJob : IJobParallelFor {
    [ReadOnly] public float A;
    [ReadOnly] public float B;
    [ReadOnly] public float C;
    [ReadOnly] public NativeArray<float> X;
    [WriteOnly] public NativeArray<float> Y;

    public void Execute(int index) {
        // Equation: y = ax^2 + bx + c
        Y[index] = A * math.pow(X[index], y: 2) + B * X[index] + C;
    }
}
```

The Burst Compiler

- Translates IL/.NET bytecode into optimised native code
- Primarily used for optimising jobs for the C# job system
- Only compatible with a subset of C#
  - The High Performance C# (HPC#) feature set
  - Reference types and classes are not supported
- Managed via attributes

Helpful Resource: The Official Burst Documentation

Figure C: A Burst Compiled Job

```
[BurstCompile]
public struct QuadraticFunctionEvaluatorJob : IJobParallelFor {
    [ReadOnly] public float A;
    [ReadOnly] public float B;
    [ReadOnly] public float C;
    [ReadOnly] public NativeArray<float> X;
    [WriteOnly] public NativeArray<float> Y;

    public void Execute(int index) {
        // Equation: y = ax^2 + bx + c
        Y[index] = A * math.pow(X[index], y: 2) + B * X[index] + C;
    }
}
```

The Entity Component System (ECS)

- A specific implementation of generalised ECS architecture
  - An entity identifies an instanced set of components
  - A component is an individual package of data
  - A system runs over all entities with certain component(s)
- Entities and components are members of HPC#
- Systems can be defined using burst compiled jobs
  - This is the ideal case which we should be striving for

Helpful Resource: The Official ECS Documentation

Applying DOTS to AI

Let's Focus On Decision Making

- It's well-trodden ground within the games industry
  - Not including neural network models and academics
- It has several well-defined archetypes that we can analyse
  - Finite State Machines (FSM)
  - Behaviour trees
  - Goal-Oriented Action Planning (GOAP)
  - Utility systems
- The architectures are fairly human-oriented
  - In part thanks to decades of OOP

Let's Focus On Utility Systems

- Using mathematical operations to evaluate and rank choices
- It's ultimately the most compatible with DOP principles
  - It is relatively stateless in it's execution
    - FSMs, behaviour trees, and GOAP are all very stateful
  - It is easily broken down into entities and components
  - It is easily broken down into small systems / jobs
    - No costly operations like traversal and searching
  - It is easily broken down into basic mathematical ops
    - We're in the best position to leverage burst

How Do Utility Systems Work?

- The fundamental algorithm is consistent across incarnations
  - For each possible option, calculate it's value / utility
  - Select a single option with the most "appealing" value
    - E.g., the option with the highest value / utility
    - E.g., any option with non-zero value from a subset
- We will apply the Infinite Axis Utility System (IAUS)
  - Designed and developed by David Mark

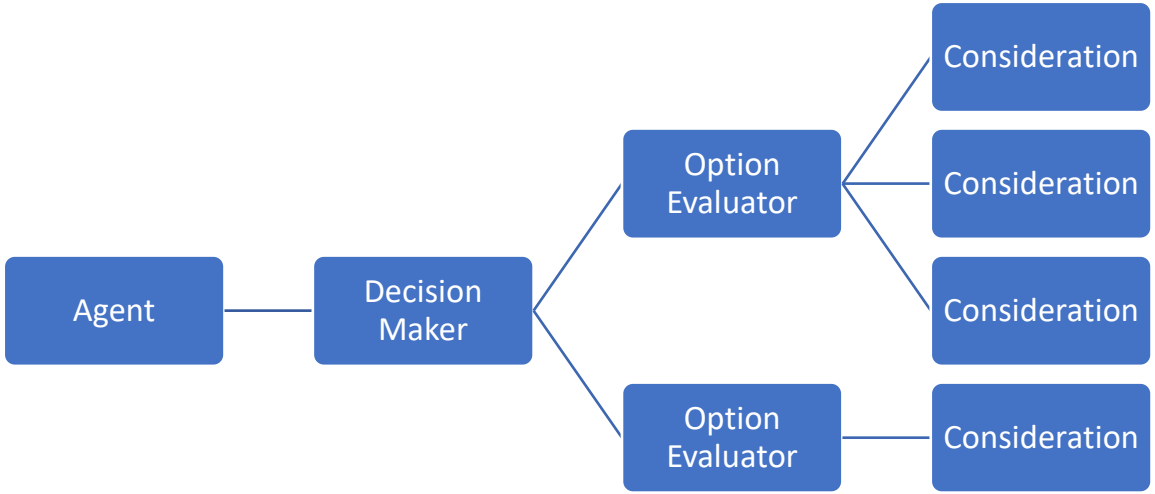
Helpful Resource: An Introduction to Utility Theory

Examples of Utility Systems

- Most commonly used in data-heavy and choice-rich settings
  - E.g., *The Sims*, *Chess*, worker units in RTSs

The IAUS In A Nutshell

- We deal with three core concepts:
  1. Agents: a decision maker (individuals, squads, directors)
  2. Options: a thing actualised by the agent (reload, attack)
  3. Considerations: a factor in determining an action's utility



Consideration Breakdown

- Considerations take an input, normalise it, and transform it

Input
- Any value from the agent or the game state
- E.g., health remaining, ammo in clip, is knocked down, etc

Normalisation
- Normalise the input to be between 0 and 1
- E.g., bounded between a min and max value, etc

Transformation
- Pass the normalised value through a response curve / function
- E.g., linear, polynomials, trigonometric, exponential, etc

Considerations in ECS

- The entity has an input, normaliser, and a response curve

```
public struct Consideration : IComponentData {
    public float RawInput;
    public float NormalizedInput;
    public float ResponseCurveOutput;
}
```

```
public struct BoundedNormalizer : IComponentData {
    public float MinValue;
    public float MaxValue;
}
```

```
public struct ExponentialResponseCurve : IComponentData {
    // Function: y = m^(k * (x - c)) + b
    public float K;
    public float M;
    public float B;
    public float C;
}
```

Figure D: A Response Curve System

```
[BurstCompile]
[UpdateInGroup(typeof(ConsiderationResponseCurveSystemGroup))]
public partial struct ExponentialResponseCurveSystem : ISystem {
    public void OnCreate(ref SystemState state) {}
    public void OnDestroy(ref SystemState state) {}
    public void OnUpdate(ref SystemState state) {
        new ExponentialResponseCurveJob().ScheduleParallel();
    }
}

[BurstCompile]
public partial struct ExponentialResponseCurveJob : IJobEntity {
    private const float MinOutput = 0.0f;
    private const float MaxOutput = 1.0f;
    private void Execute(in ExponentialResponseCurve responseCurve, ref Consideration consideration) {
        // Function: y = m^(k * (x - c)) + b
        consideration.ResponseCurveOutput = math.pow( x: responseCurve.M,  y: responseCurve.K * (consideration.NormalizedInput - responseCurve.C)) + responseCurve.B;
        consideration.ResponseCurveOutput = math.clamp( x: consideration.ResponseCurveOutput,  a: MinOutput,  b: MaxOutput);
    }
}
```

An Example Consideration

- Our input is our remaining health
  - 10 points of health remaining
- Our input is bounded between our min and max health
  - Our min health is 0 and our max health is 100
  - Our normalised input is 10 / 100 = 0.1
- Our response curve is exponential decay
  - y = (1 / 2)^x, if x = 0.1, y = 0.93

Option Evaluator Breakdown

- Option evaluators aggregate all processed considerations

Consideration Processing

- Wait for all considerations to be processed
- Incl. input, normalisation, and transformation

Aggregation

- Aggregate all consideration inputs into a single utility
- E.g., multiplication, taking the mean, taking the max / min

Option Evaluators in ECS

- The entity has considerations, an aggregator, and a tracker

```
public struct ConsiderationHandleElement : IBufferElementData {
    public Entity Entity;
}
```

```
public struct MultiplicationAggregator : IComponentData {
}
```

```
public struct OptionEvaluator : IComponentData {
    public int OptionKey;
    public float Utility;
}
```

Figure E: An Aggregator System

```
[BurstCompile]
public partial struct MultiplicationAggregatorJob : IJobEntity {
    private const float MinUtility = 0.0f;
    private const float MaxUtility = 1.0f;
    [ReadOnly] public ComponentLookup<Consideration> ConsiderationLookup;
    private void Execute(in MultiplicationAggregator aggregator,
        in DynamicBuffer<ConsiderationHandleElement> considerationHandleBuffer, ref OptionEvaluator optionEvaluator) {
        optionEvaluator.Utility = MinUtility;
        if (considerationHandleBuffer.IsEmpty) {
            return;
        }

        optionEvaluator.Utility = MaxUtility;
        foreach (ConsiderationHandleElement considerationHandleElement in considerationHandleBuffer) {
            if (!ConsiderationLookup.HasComponent(considerationHandleElement.Entity)) {
                continue;
            }

            RefRO<Consideration> consideration = ConsiderationLookup.GetRefRO(considerationHandleElement.Entity);
            optionEvaluator.Utility *= consideration.ValueRO.ResponseCurveOutput;
        }
    }
}
```

Decision Maker Breakdown

- Decision makers select one action from all the evaluators

Option Evaluator Processing

- Wait for all option evaluators to be processed
- Incl. aggregation

Option Selection

- Select an option given the utility of all option evaluators
- E.g., with the highest utility, from a bucket, etc

Decision Makers in ECS

- The entity has option evaluators, a selector, and a decision

```
public struct OptionEvaluatorHandleElement : IBufferElementData {
    public Entity Entity;
}
```

```
public struct HighestUtilitySelector : IComponentData {
}
```

```
public struct Decision : IComponentData {
    public int SelectedOptionKey;
    public float UtilityOfSelection;
}
```

Figure F: An Option Selector System

```
[BurstCompile]
public partial struct HighestUtilitySelectorJob : IJobEntity
{
    [ReadOnly] public ComponentLookup<OptionEvaluator> OptionEvaluatorLookup;
    private void Execute(in HighestUtilitySelector selector,
        in DynamicBuffer<OptionEvaluatorHandleElement> optionEvaluatorHandleBuffer, ref Decision decision) {
        var isFirstOptionEvaluator = true;
        foreach (var optionEvaluatorHandleElement in optionEvaluatorHandleBuffer) {
            if (!OptionEvaluatorLookup.HasComponent(optionEvaluatorHandleElement.Entity)) {
                continue;
            }

            RefRO<OptionEvaluator> optionEvaluator = OptionEvaluatorLookup.GetRefRO(optionEvaluatorHandleElement.Entity);
            if (isFirstOptionEvaluator
                || decision.UtilityOfSelection < optionEvaluator.ValueRO.Utility) {
                decision.SelectedOptionKey = optionEvaluator.ValueRO.OptionKey;
                decision.UtilityOfSelection = optionEvaluator.ValueRO.Utility;
            }

            isFirstOptionEvaluator = false;
        }
    }
}
```

Wrapping Up: System Groups

- Using system groups for ordering the execution of jobs

```
[UpdateInGroup(typeof(SimulationSystemGroup))]
public partial class UtilitySystemSystemGroup : ComponentSystemGroup {
}
```

```
[UpdateInGroup(typeof(UtilitySystemSystemGroup))]
[UpdateBefore(typeof(OptionSystemGroup))]
public partial class ConsiderationSystemGroup : ComponentSystemGroup {
}
```

```
[UpdateInGroup(typeof(ConsiderationSystemGroup))]
[UpdateAfter(typeof(ConsiderationInputSystemGroup))]
[UpdateBefore(typeof(ConsiderationResponseCurveSystemGroup))]
public partial class ConsiderationNormalizationSystemGroup : ComponentSystemGroup {
}
```

```
[BurstCompile]
[UpdateInGroup(typeof(ConsiderationNormalizationSystemGroup))]
public partial struct BoundedNormalizerSystem : ISystem
```

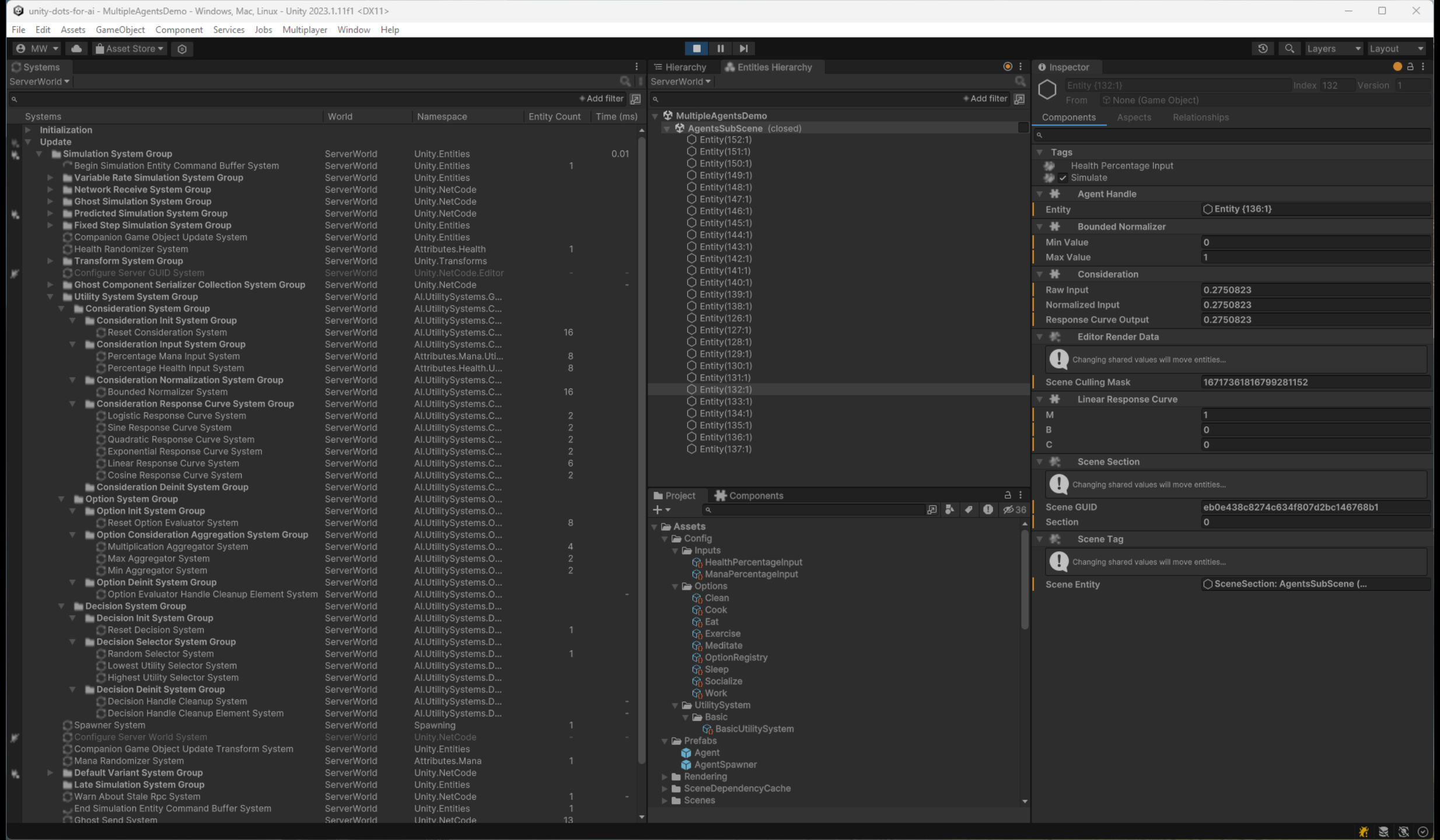
Wrapping Up: Cleanup Components

- Using Cleanup components to propagate agent deletion

```
[BurstCompile]
[WithNone( params types: typeof(OptionEvaluatorHandleElement))]
public partial struct OptionEvaluatorHandleCleanupElementJob : IJobEntity {
    public EntityCommandBuffer.ParallelWriter ParallelWriter;

    private void Execute(Entity entity, [ChunkIndexInQuery] int sortKey,
        in DynamicBuffer<OptionEvaluatorHandleCleanupElement> optionEvaluatorHandleCleanupBuffer) {
        foreach (OptionEvaluatorHandleCleanupElement optionEvaluatorHandleCleanupElement in optionEvaluatorHandleCleanupBuffer) {
            ParallelWriter.DestroyEntity(sortKey, optionEvaluatorHandleCleanupElement.Entity);
        }

        ParallelWriter.RemoveComponent<OptionEvaluatorHandleCleanupElement>(sortKey, entity);
    }
}
```


unity-dots-for-ai - MultipleAgentsDemo - Windows, Mac, Linux - Unity 2023.1.11f1 <DX11>
File Edit Assets GameObject Component Services Jobs Multiplayer Window Help
MW
Asset Store
Layers
Layout
Systems
ServerWorld
Add filter
Systems World Namespace Entity Count Time (ms)
Initialization
Update
Simulation System Group ServerWorld Unity.Entities 0.01
Begin Simulation Entity Command Buffer System ServerWorld Unity.Entities 1
Variable Rate Simulation System Group ServerWorld Unity.Entities
Network Receive System Group ServerWorld Unity.NetCode
Ghost Simulation System Group ServerWorld Unity.NetCode
Predicted Simulation System Group ServerWorld Unity.NetCode
Fixed Step Simulation System Group ServerWorld Unity.Entities
Companion Game Object Update System ServerWorld Unity.Entities
Health Randomizer System ServerWorld Attributes.Health 1
Transform System Group ServerWorld Unity.Transforms
Configure Server GUID System ServerWorld Unity.NetCode.Editor - -
Ghost Component Serializer Collection System Group ServerWorld Unity.NetCode -
Utility System System Group ServerWorld AI.UtilitySystems.G...
Consideration System Group ServerWorld AI.UtilitySystems.C...
Consideration Init System Group ServerWorld AI.UtilitySystems.C...
Reset Consideration System ServerWorld AI.UtilitySystems.C... 16
Consideration Input System Group ServerWorld AI.UtilitySystems.C...
Percentage Mana Input System ServerWorld Attributes.Mana.Uti... 8
Percentage Health Input System ServerWorld Attributes.Health.U... 8
Consideration Normalization System Group ServerWorld AI.UtilitySystems.C...
Bounded Normalizer System ServerWorld AI.UtilitySystems.C... 16
Consideration Response Curve System Group ServerWorld AI.UtilitySystems.C...
Logistic Response Curve System ServerWorld AI.UtilitySystems.C... 2
Sine Response Curve System ServerWorld AI.UtilitySystems.C... 2
Quadratic Response Curve System ServerWorld AI.UtilitySystems.C... 2
Exponential Response Curve System ServerWorld AI.UtilitySystems.C... 2
Linear Response Curve System ServerWorld AI.UtilitySystems.C... 6
Cosine Response Curve System ServerWorld AI.UtilitySystems.C... 2
Consideration Deinit System Group ServerWorld AI.UtilitySystems.C...
Option System Group ServerWorld AI.UtilitySystems.O...
Option Init System Group ServerWorld AI.UtilitySystems.O...
Reset Option Evaluator System ServerWorld AI.UtilitySystems.O... 8
Option Consideration Aggregation System Group ServerWorld AI.UtilitySystems.O...
Multiplication Aggregator System ServerWorld AI.UtilitySystems.O... 4
Max Aggregator System ServerWorld AI.UtilitySystems.O... 2
Min Aggregator System ServerWorld AI.UtilitySystems.O... 2
Option Deinit System Group ServerWorld AI.UtilitySystems.O...
Option Evaluator Handle Cleanup Element System ServerWorld AI.UtilitySystems.O... -
Decision System Group ServerWorld AI.UtilitySystems.D...
Decision Init System Group ServerWorld AI.UtilitySystems.D...
Reset Decision System ServerWorld AI.UtilitySystems.D... 1
Decision Selector System Group ServerWorld AI.UtilitySystems.D...
Random Selector System ServerWorld AI.UtilitySystems.D... 1
Lowest Utility Selector System ServerWorld AI.UtilitySystems.D...
Highest Utility Selector System ServerWorld AI.UtilitySystems.D...
Decision Deinit System Group ServerWorld AI.UtilitySystems.D...
Decision Handle Cleanup System ServerWorld AI.UtilitySystems.D... -
Decision Handle Cleanup Element System ServerWorld AI.UtilitySystems.D... -
Spawner System ServerWorld Spawning 1
Configure Server World System ServerWorld Unity.NetCode - -
Companion Game Object Update Transform System ServerWorld Unity.Entities
Mana Randomizer System ServerWorld Attributes.Mana 1
Default Variant System Group ServerWorld Unity.NetCode
Late Simulation System Group ServerWorld Unity.Entities
Warn About Stale Rpc System ServerWorld Unity.NetCode 1 -
End Simulation Entity Command Buffer System ServerWorld Unity.Entities 1
Ghost Send System ServerWorld Unity.NetCode 13
Hierarchy
Entities Hierarchy
ServerWorld
Add filter
MultipleAgentsDemo
AgentsSubScene (closed)
Entity(152:1)
Entity(151:1)
Entity(150:1)
Entity(149:1)
Entity(148:1)
Entity(147:1)
Entity(146:1)
Entity(145:1)
Entity(144:1)
Entity(143:1)
Entity(142:1)
Entity(141:1)
Entity(140:1)
Entity(139:1)
Entity(138:1)
Entity(126:1)
Entity(127:1)
Entity(128:1)
Entity(129:1)
Entity(130:1)
Entity(131:1)
Entity(132:1)
Entity(133:1)
Entity(134:1)
Entity(135:1)
Entity(136:1)
Entity(137:1)
Project
Components
Assets
Config
Inputs
HealthPercentageInput
ManaPercentageInput
Options
Clean
Cook
Eat
Exercise
Meditate
OptionRegistry
Sleep
Socialize
Work
UtilitySystem
Basic
BasicUtilitySystem
Prefabs
Agent
AgentSpawner
Rendering
SceneDependencyCache
Scenes
Inspector
Entity {132:1}
Index 132
Version 1
From None (Game Object)
Components
Aspects
Relationships
Tags
Health Percentage Input
Simulate
Agent Handle
Entity
Entity {136:1}
Bounded Normalizer
Min Value 0
Max Value 1
Consideration
Raw Input 0.2750823
Normalized Input 0.2750823
Response Curve Output 0.2750823
Editor Render Data
Changing shared values will move entities...
Scene Culling Mask 16717361816799281152
Linear Response Curve
M 1
B 0
C 0
Scene Section
Changing shared values will move entities...
Scene GUID eb0e438c8274c634f807d2bc146768b1
Section 0
Scene Tag
Changing shared values will move entities...
Scene Entity
SceneSection: AgentsSubScene (...

Extending The Decision maker

- Adding more inputs, normalisers, and response curves
- Adding more aggregators and selectors
- Adding subjects to option evaluators and considerations
  - E.g., attack enemy X, health ally Y, consume potion Z
- Adding dual utility reasoning in the form of buckets
  - Selecting an option with non-zero utility before others
- Reducing the number of entities per decision maker

Alternative Decision Makers

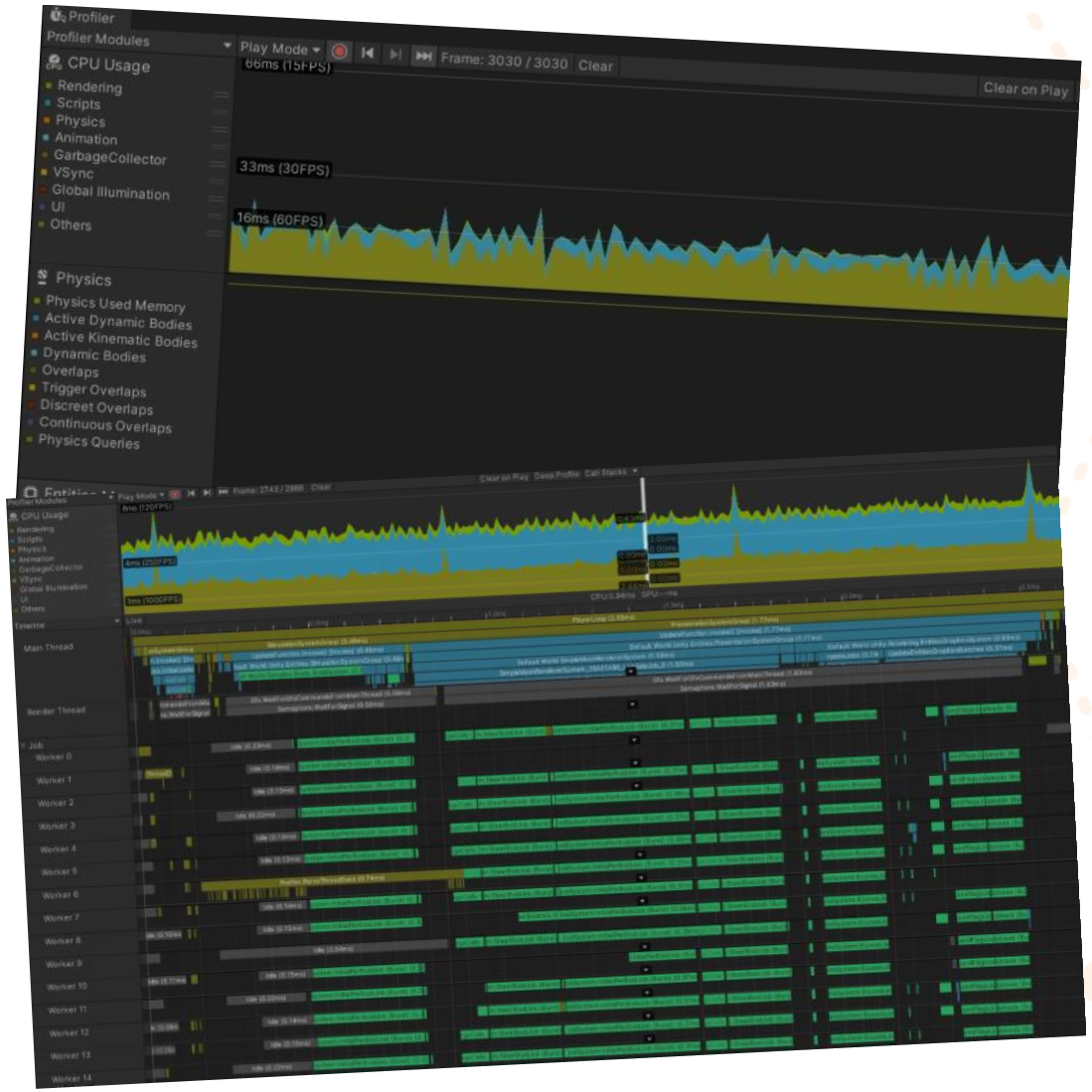
- It's technically possible to implement all architectures in ECS
- Ensure that your jobs aren't long running
  - Jobs don't yield execution, unlike threads
- Ensure you minimise dependencies where possible
  - Minimise changes to entities and components
  - This could be problematic for FSMs and behaviour trees
- Ensure your components aren't too large

Helpful Resources: An Introduction to FSMs, The Behaviour Tree Starter Kit, A Review of GOAP

Evaluating DOTS

Strengths

- Performance via tweaking
  - But not by default
- Highly scalable
  - But hardware dependent
- Encourages good practices
  - Planning and research
  - Separation of concerns
  - Reusable components

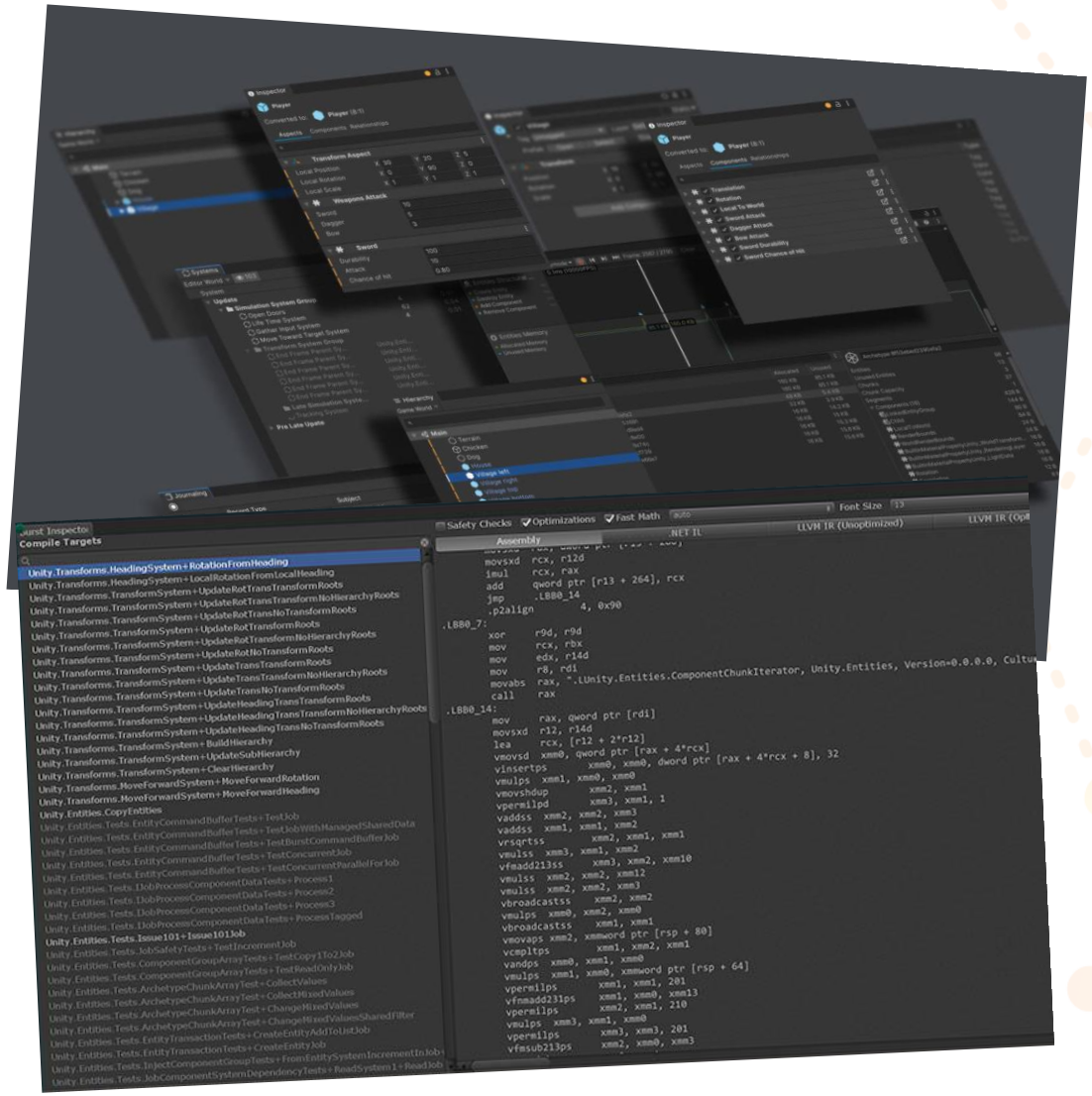


Strengths

- Future proofing
  - Future hardware
- Upskilling and education
  - Hardware considerations
  - Un/managed memory
- Other potential benefits
  - Extended battery life
  - Tighter thermals

Weaknesses

- Suitable for specific problems
- No one size fits all approach
  - Multiple vs single entity
- Bigger upfront planning costs
  - Steep learning curve
  - Limited C# features
  - Limited engine features
- Limited runtime changes
- Debuggability

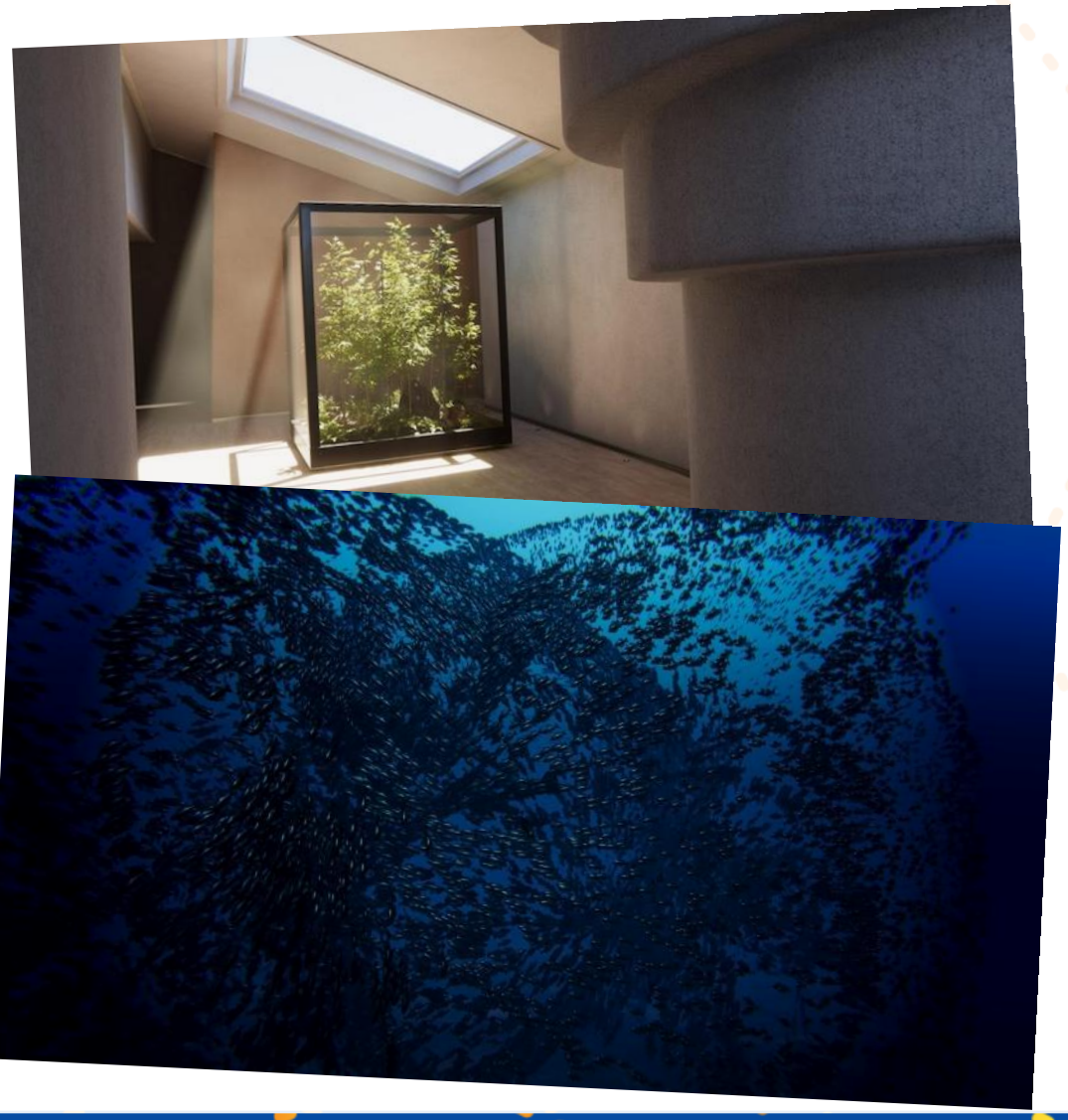


Weaknesses

- Easy to inject issues
  - Distributed execution
  - System execution order
- Death by a thousand cuts
  - Too many sync points
  - Too much data per entity
  - Too many entities
  - Not enough entities

Recommendations

- Understand your needs
  - Art, design, and tech
  - Know what is supported
  - Know what isn't supported
- Prototype and research
- Communicate differences
  - Across all workflows
  - Across the entire pipeline



Recommendations

- Monitor performance deltas
  - Integrate tests into CI
  - Respond to changes early
- Plan out supplementary work
  - Custom editors (Odin)
  - Addressing weaknesses
    - E.g., runtime changes

Conclusions

The Key Takeaways

- When used correctly, Unity's DOTS is a game changer
- It's a substantial change to the Unity development experience
- It's compatibility with existing architectures is inconsistent
- Any transition to DOTS requires a holistic evaluation and execution strategy

Shared Resources

Git Hub

Questions

References

Champandard, A. J., & Dunstan, P. (2013). The Behaviour Tree Starter Kit. In Rabin, S. (Ed.) *Game AI Pro: Collected Wisdom of Game AI Professionals* (pp. 73-91). CRC Press. https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter06_The_Behavior_Tree_Starter_Kit.pdf

Conway, C., Higley, P., & Jacopin E. [GDC]. (2017, October 9). *Goal-Oriented Action Planning: Ten Years of AI Programming* [Video]. YouTube. https://www.youtube.com/watch?v=gm7K68663rA

Dicken, L., Dini, D., & Mark, D. [GDC]. (2013). *Architecture Tricks: Managing Behaviors in Time, Space, and Depth* [Video]. GDC Vault. https://www.gdcvault.com/play/1018040/

Dill, K. (2015). Dual-Utility Reasoning. In Rabin, S. (Ed.) *Game AI Pro 2: Collected Wisdom of Game AI Professionals* (pp. 23-26). CRC Press. http://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter03_Dual-Utility_Reasoning.pdf

Graham, D. (2013). An Introduction to Utility Theory. In Rabin, S. (Ed.), *Game AI Pro: Collected Wisdom of Game AI Professionals* (pp. 113-126). CRC Press. http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf

Mines, J. (2018, March 21). *Data-Oriented vs Object-Oriented Design*. Retrieved September 12, 2023, from https://medium.com/@jonathanmines/data-oriented-vs-object-oriented-design-50ef35a99056

Patterson, J., & Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc.

Robson, D. (1981, August). Object-Oriented Software Systems. *Byte Magazine*. https://archive.org/details/byte-magazine-1981-08/page/n75/mode/2up

Unity Technologies. (2023, August 9). *About Burst*. Retrieved September 20, 2023, from https://docs.unity3d.com/Packages/com.unity.burst@1.8/manual/index.html

Suzdalnitskiy, I. (2019, July 11). *Object-Oriented Programming - The Trillion Dollar Disaster*. Retrieved September 12, 2023, from https://betterprogramming.pub/object-oriented-programming-the-trillion-dollar-disaster-92a4b666c7c7

References

Unity Technologies. (2023, September 13). *Entities Overview*. Retrieved September 20, 2023, from https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/index.html

Unity Technologies. (2023, September 15). *Job System*. Retrieved September 20, 2023, from https://docs.unity3d.com/Manual/JobSystem.html

Wikipedia. (2023, August 18). *Branch predictor*. Retrieved September 20, 2023 from https://en.wikipedia.org/w/index.php?title=Branch_predictor&action=history

Wikipedia. (2023, September 12). *Cache prefetching*. Retrieved September 20, 2023 from https://en.wikipedia.org/wiki/Cache_prefetching

Wikipedia. (2023, September 9). *Central processing unit*. Retrieved September 12, 2023 from https://en.wikipedia.org/wiki/Central_processing_unit#Operation

Wikipedia. (2023, August 3). *Computer architecture*. Retrieved September 12, 2023 from https://en.wikipedia.org/wiki/Computer_architecture

Wikipedia. (2023, September 6). *Computer memory*. Retrieved September 12, 2023 from https://en.wikipedia.org/wiki/Computer_memory

Wikipedia. (2023, September 5). *Data-oriented design*. Retrieved September 12, 2023 from https://en.wikipedia.org/wiki/Data-oriented_design

Wikipedia. (2023, September 10). *Object-oriented programming*. Retrieved September 12, 2023 from https://en.wikipedia.org/wiki/Object-oriented_programming