Literature Review and Annotated Bibliography

Search Term: Behavior Tree Search Engine: Google Chrome

Found Sources:

1. Behavior trees for AI: How they work:

https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

Last accessed: 13/10/24

Type: Article Online

Author: Chris Simpson

Summary: “a behaviour tree is a tree of hierarchical nodes that control the flow of decision making of an AI entity. At the extents of the tree, the leaves, are the actual commands that control the AI entity, and forming the branches are various types of utility nodes that control the AI’s walk down the trees to reach the sequences of commands best suited to the situation.”(Simpson 2014)

The article also goes deeper into how each node works.

2. Introduction to Behavior trees:

https://robohub.org/introduction-to-behavior-trees/

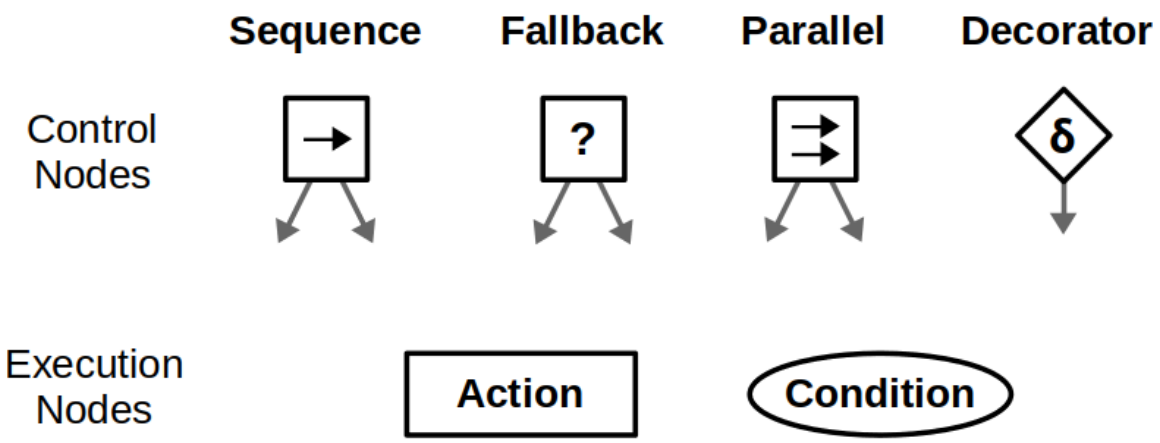
Last accessed: 13/10/24

Type: Article Online

Author: Sebastian Castro

Summary:

Behavior Tree terminology



(Castro 2021)

Specific to BTs vs. FSMs, there is a tradeoff between **modularity** and **reactivity**. Generally, BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors.(Castro 2021)

The article gives an examples of behavior trees and hierarchical state machines also compares them with each other and links to the following resources.

This is our first resources we added and went over.

- https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work
- https://web.stanford.edu/class/cs123/lectures/CS123_lec08_HFSM_BT.pdf

3. 2.4 Hierarchical Finite State Machine (HFSM) & Behavior Tree (BT)

https://web.stanford.edu/class/cs123/lectures/CS123_lec08_HFSM_BT.pdf

Last accessed: 13/10/24

Found: the PDF was linked in this article https://robohub.org/introduction-to-behavior-trees/

Type: Online PDF

Author: Kyong-Sok (KC) Chang & David Zhu

Summary:

Four types of nodes:

• Root node – no parent, one child (ticks)

• Composite node (“Control flow ”) – one parent, and one or more children

• Leaf node (“Execution”) – one parent, no child (Leaves)

• Decorator node (“Operator”) – one parent, one child

(Chang & Zhu, z.d.)

Tree Traversal Issue

• Always start from root node

• This isn’t a very efficient way to do things, especially when the behavior tree gets deeper as its developed and expanded during development.

• Store any currently processing nodes so they can be ticked directly within the behavior tree engine rather than per tick traversal of the entire tree

(Chang & Zhu, z.d.)

Benefits of BT

• Maintainability: transitions in BT are defined by the structure, not by conditions inside the states. Because of this, nodes can be designed independent from each other, thus, when adding or removing new nodes (or even subtrees) in a small part of the tree, it is not necessary to change other parts of the model.

• Scalability: when a BT have many nodes, it can be decomposed into small sub-trees saving the readability of the graphical model.

• Reusability: due to the independence of nodes in BT, the subtrees are also independent. This allows the reuse of nodes or subtrees among other trees or projects.

(Chang & Zhu, z.d.)

Search Term: finite state machines Search Engine: Google Chrome

4. 2.3 Finite State Machine (FSM) Concept and Implementation

https://web.stanford.edu/class/cs123/lectures/CS123_lec07_Finite_State_Machine.pdf

Last accessed 13/10/24

Type: online PDF

Author: Kyong-Sok (KC) Chang & David Zhu

Summary:

What Is A Finite State Machine?

• A reactive system whose response to a particular stimulus (a signal, or a piece of input) is not the same on every occasion, depending on its current "state". (Chang & Zhu, z.d.)

How To Implement an FSM

• The Finite State Machine class keeps track of the current state, and the list of valid state transitions.

• You define each transition by specifying :

• FromState - the starting state for this transition

• ToState - the end state for this transition

• condition - a callable which when it returns True means this transition is valid

• callback - an optional callable function which is invoked when this transition is executed.

(Chang & Zhu, z.d.)

FSM Examples in Daily Live

• Vending Machines

• Traffic Lights

• Elevators

• Alarm Clock

• Microwave

• Cash Registers

Each of these devices can be thought of as a reactive system - that

is because each of them work by reacting to signals or inputs from the external world.

(Chang & Zhu, z.d.)

Search Term: Game programming patterns state Search Engine: Google Chrome

5. State

https://gameprogrammingpatterns.com/state.html

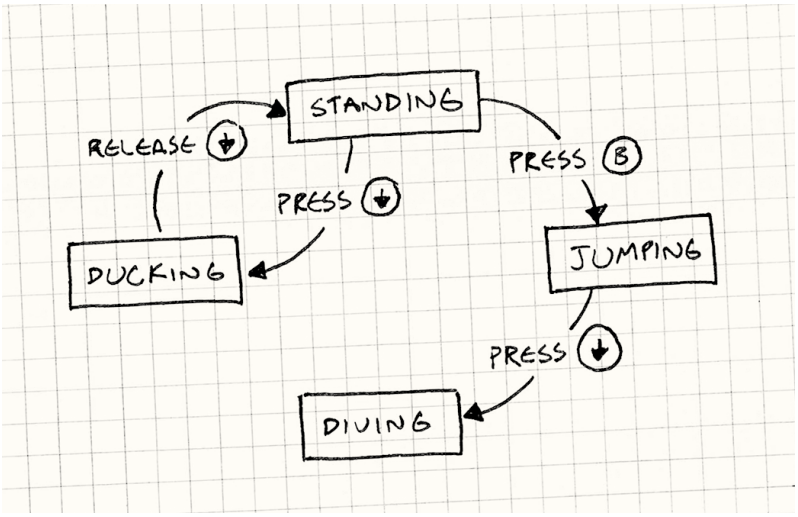
Last accessed 14/10/24

Type: online book

Author: Bob Nystrom

Summary:

The chapter of the book explains how to make state machines and why you should use them.



- You have a fixed *set of states* that the machine can be in. For our example, that's standing, jumping, ducking, and diving.
- The machine can only be in *one* state at a time. Our heroine can't be jumping and standing simultaneously. In fact, preventing that is one reason we're going to use an FSM.
- A sequence of *inputs* or *events* is sent to the machine. In our example, that's the raw button presses and releases.
- Each state has a *set of transitions*, each associated with an input and pointing to a state. When an input comes in, if it matches a transition for the current state, the machine changes to the state that transition points to.

(Nystrom 2014)

Finite state machines are useful when:

- You have an entity whose behavior changes based on some internal state.
- That state can be rigidly divided into one of a relatively small number of distinct options.
- The entity responds to a series of inputs or events over time.

(Nystrom 2014)

Search Term: Finite state machine in games Search Engine: Google Chrome

6. Finite State Machines in Game Development

https://medium.com/@parameesupipiinduwari/finite-state-machines-in-game-development-3c12dc4d667a

Last accessed 14/10/24

Type: Article online

Author: Paramee Supipi

Summary: The article is a step by step guide on how to make a finite state machine.

What is a Finite State Machine?

A Finite State Machine is a computational model used to design algorithms. It consists of a finite number of states, transitions between those states, and actions that occur due to those transitions. FSMs are used to manage the states of objects, characters, or systems within a game, ensuring that each entity behaves predictably.

(Supipi 2024)

FSMs are widely used in game development for:

- Character Behavior: Managing AI states like patrolling, chasing, and attacking.
- Game States: Handling different phases of a game such as menus, gameplay, and game over screens.
- Animations: Controlling animation sequences based on character states.
- Event Systems: Triggering events based on game states.

(Supipi 2024)

Conclusion

Finite State Machines are essential for managing complex behaviors in games. They help in organizing code, making systems more predictable, and simplifying debugging. By following the steps outlined above, you can implement FSMs in Unity to control character behaviors, game states, and more. This structured approach will enhance your game's maintainability and scalability, providing a better experience for players and developers alike.

(Supipi 2024)

7. Goal Oriented Action Planning for a Smarter AI

https://gamedevelopment.tutsplus.com/goal-oriented-action-planning-for-a-smarter-ai--cms-20793t

Last accessed 14/10/24

Type: Article online

Author: Brent Owens

Summary:

What is GOAP?

Goal oriented action planning is an artificial intelligence system for agents that allows them to plan a sequence of actions to satisfy a particular goal. The particular sequence of actions depends not only on the goal but also on the current state of the world and the agent. This means that if the same goal is supplied for different agents or world states, you can get a completely different sequence of actions., which makes the AI more dynamic and realistic.

(Owens 2014)

The GOAP Planner

The GOAP planner is a piece of code that looks at actions' preconditions and effects, and creates queues of actions that will fulfill a goal. That goal is supplied by the agent, along with a world state, and a list of actions the agent can perform. With this information the GOAP planner can order the actions, see which can run and which can't, and then decide which actions are the best to perform. (Owens 2014)

Conclusion

With GOAP, you can create a large series of actions without the headache of interconnected states that often comes with a Finite State Machine. Actions can be added and removed from an agent to produce dynamic results, as well as to keep you sane when maintaining the code. You will end up with a flexible, smart, and dynamic AI. (Owens 2014)

8. Better AI in Unity - GOAP (Goal Oriented Action Planning)

https://www.youtube.com/watch?v=T_sBYgP7_2k

Last accessed 18/10/24

Type: youtube video

Author: git-amend

Summary:

Goap is an artificial intelligence system for autonomous agents that allow them to dynamically plan a sequence of actions to satisfy a set goal.(git-amend 2024)

The video goes over what goap is and builds a system. I can use this guide to maybe make a goap system in unity.

Search: found in the description of nr 8 video Platform: Youtube

9. Applying Goal-Oriented Action Planning to Games

https://web.archive.org/web/20230912173044/https://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf

Last accessed 18/10/24

Type: Online pdf on the waybackmachine.

Author: Jeff Orkin

Summary:

A number of games have implemented characters with goal directed decision-making capabilities. A goal-directed character displays some measure of intelligence by autonomously deciding to activate the behavior that will satisfy the most relevant goal at any instance. Goal-Oriented Action Planning (GOAP) is a decision-making architecture that takes the next step, and allows characters to decide not only what to do, but how to do it. But why would we want to empower our characters with so much freedom?
A character that formulates his own plan to satisfy his goals exhibits less repetitive, predictable behavior, and can adapt his actions to custom fit his current situation. In addition, the structured nature of a GOAP architecture facilitates authoring, maintaining, and re-using behaviors (Orkin 2003)

The Need for GOAP:

With each new game released, the bar is set higher for AI behavior. As expectations for the complexity of character behavior rises, we need to look toward more structured, formalized solutions to creating scalable, maintainable, and re-usable decision-making systems. Goal-oriented action planning is one such solution. By letting go of the reins and allowing games to formulate plans at runtime, we are handing over key decisions to those who are in the best position to make them; the characters themselves. (Orkin 2003)

Search: Utility ai in games search engine: google chrome

10. An introduction to Utility AI

https://shaggydev.com/2023/04/19/utility-ai/

Last accessed 19/10/24

Type: online article

Author: Jason McCollum

Summary:

Utility AI tries to answer to the question “What’s the best action I can take right now?” by allowing for objective comparison between actions that can be as similar or different as the game demands. This objectivity comes from assigning a normalized score in the range 0-1 to each possible action a entity can take in that moment and then selecting the action with the highest score for execution. This score is typically referred to as the action’s “utility”, hence the name for this technique.(McCollum 2023)

It’s also worth noting that Utility AI is not concerned with how to take a given action, only what action it should take, meaning that the concepts behind it are fully compatible with whatever technique you’re using to execute actions in your game, whether that be via state machine or some other mechanism. The AI will simply say “this is the action to take” and then your game code is responsible for parsing that into actionable data however you see fit, making it a highly flexible system.(McCollum 2023)

Scoring is where all the magic happens in a utility system, and how you decide to convert the varied data of your game state into a normalized number is everything to this process. Unit positions, stat values, items in your vicinity - it can all play a part, and the one of the first steps in building a utility system is to figure out how you’ll access the current game state, typically referred to as the “context”, and turn the relevant data into numbers. This process may vary depending on what the game looks like, but as a general solution, Dave Mark, one of *the* experts on Utility AI systems design, has discussed the idea of a “clearing house” - a place where you can ask for data from anywhere in the game and receive it back in a normalized state. (McCollum 2023)

Found: was a reference in nr 10 An introduction to Utility AI
https://shaggydev.com/2023/04/19/utility-ai/

11. An Introduction to Utility Theory

https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf

Last accessed 19/10/24

Type: online pdf

Author: David “Rez” Graham

Summary:

Introduction

Decision making forms the core of any AI system. There are many different approaches to decision making, several of which are discussed in other chapters in this book. One of the most robust and powerful systems we’ve encountered is a utility-based system. The general concept of a utility-based system is that every possible action is scored at once and one of the top scoring actions is chosen. By itself, this is a very simple and straightforward approach. In this article, we’ll talk about common techniques, best practices, pitfalls to avoid, and how you can best apply utility theory to your AI. (Graham, z.d.)

Conclusion

Utility theory is a very powerful way to get rich, life-like behavior out of your AI and has been used in countless games of nearly every genre. It can be extremely fast, especially if you choose simple utility functions, and it scales very well. One of the great appeals of this system is the amount of emergent behavior you can get with just a few simple values and a handful of weights to add some personality. By combining decision factors in meaningful ways, you can build up decisions from these atomic components to provide very deep AI. In this article, we had a whirlwind tour of utility theory and how it can be applied to games. We showed you some basic principles of decision making and dug into the math behind it. With the tools in this article and a bit of work, you can build a powerful, emergent AI system.(Graham, z.d.)

12. AI Made Easy with Utility AI

https://medium.com/@morganwalkupdev/ai-made-easy-with-utility-ai-fef94cd36161

Last accessed 19/10/24

Type: online Article

Author: Morgan Walkup

Summary:

Let's say we want to build an AI that can win Pokemon battles by selecting the best attack every turn.

First, what are Bulbasaur's possible actions?

1. Tackle (a damaging, normal-type move)2. Growl (a non-damaging, normal-type move)3. Leech Seed (a status-inducing, grass-type move)4. Vine Whip (a damaging, grass-type move)

Next, let's gather some data about the current situation.

enemy type: fire;enemy health: low;enemy status: seeded;

Finally, we'll use our data to score each action. How you choose to score your actions is totally up to you. It all depends on how you want your AI to feel. For this example, I'll use a scale of 1–10, and I want my AI to be on the aggressive side.

Tackle (Score: 7)
- Enemy's health is low, so this damaging move might finish them off
- Enemy's type is fire, so this normal move will deal full damage**Growl (Score: 1)**
- Enemy's health is low, so we want to finish them off. Growl deals no damage.
- Growl will still lower the enemy's attack, so it could be useful for our next pokemon. It's a valid option, but not a very good one.**Leech Seed (Score: 0)**
- Leech seed has no effect if the enemy's already seeded. We won't choose this option.**Vine Whip (Score: 5)**
- Enemy's health is low, so this damaging move might finish them off
- Enemy's type is fire, so this grass move will deal half damage

Now that we have our scores, we can clearly see that Tackle is the best option.

(Walkup 2021)

13. How Utility AI Helps NPCs Decide What To Do Next | AI 101

https://www.youtube.com/watch?v=p3Jbp2cZg3Q

Last accessed 20/10/24

Type: youtube video

Author: AI and games

Summary:

The video goes over:

- Why do we need utility AI
- Calculating utility
- Utility ai in AAA games
    - Used in the sims
    - Dragon Age: Inquisition
    - Total War: Three Kingdoms
    - F.E.A.R **goap is basically a utility system.**
    - Alien isolation.

14. Behaviour Trees: The Cornerstone of Modern Game AI | AI 101

https://www.youtube.com/watch?v=6VBCXvfNlCM

Last accessed 20/10/24

Type: youtube video

Author: AI and games

Summary:

The video goes over:

- What is a nehavior tree and how does it work.
- Explains the nodes.
- Explains blackboards(toolchain in unreal engine).
    - Finite state machine becomes a nightmare to follow the bigger it gets.
    - Designer friendly.
    - Finite state machines code intensive.
- The logic is reusable.
    - Behavior trees have become more optimized over time.

Got this source during the meeting from another student who already researched it during gameplay programming.

15. Goal Oriented Action Planning - Finian Horrie, 2GD10

https://github.com/FHorrie/2DAE_GOAP_Research

Last accessed 20/10/24

Type: Github repository

Author: Finian Horrie

Summary:

What is goap:

It is an alternative to a Finite State Machine for game AI, and is an attempt to manage complexity in AI behaviors. Unlike Finite State Machines and Behavior Trees, GOAP decouples actions and goals so they can be used separately in the working memory. GOAP makes use of planning to allow complex behavior. This makes GOAP slower than equivalent Finite State Machines and Behavior Trees, but more flexible when it comes to adding or removing functionality. (Horrie 2024)

We can feed our 3 sources of information to the planner in order to figure out a plan:

1.Actions 2. Goals 3. Information on the World State

The planner can then formulate a plan to attempt to achieve our goal.

- Plans may only be executed if every precondition is met.
    - Example: AI can only shoot the target if TargetVisible == true , WeaponLoaded == true and IsAlive == true.
- When a goal is met, the world has to change in some way.
    - Example: AI opens a closed door → Worldstate.door1 == closed → Worldstate.door1 == open

A plan will always be validated going from back to front to see if it is achievable.

- If an action in the plan gets invalidated due to external input, the plan is discarded and another plan is formulated.

What if we have 2 valid plans for the same goal?

- We add a cost per action to find the optimal plan.

The planner can then make use of any node based search algorithm (preferably A*) to find the plan with the lowest cost. A plan execution is done with a very simple Finite State Machine. (”Move” and “Do action” nodes)In my implementation, almost all actions work with the "Move" state.(Horrie 2024)

Conclusion: Goal Oriented Action Planning presents itself as a great method for creating game based AI. It amazes me how flexible it is, but also saddens me that it's so scarcely used in favor of behavior trees or finite state machines. Planners can be very heavy on performance when using many entities, which isn't optimal for AAA games. (Horrie 2024)

16. Three States and a Plan: The A.I. of F.E.A.R.

This is pdf was a reference from nr 15 Goal Oriented Action Planning - Finian Horrie, 2GD10

https://www.gamedevs.org/uploads/three-states-plan-ai-of-fear.pdf

Last accessed 20/10/24

Type: online pdf

Author: Jeff Orkin

Summary:

FSMs vs Planning

Let's compare FSMs to planning. An FSM tells an A.I. exactly how to behave in every situation. A planning system tells the A.I. what his goals and actions are, and lets the A.I. decide how to sequence actions to satisfy goals. FSMs are procedural, while planning is declarative. Later we will see how we can use these two types of systems together to complement one another. The motivation to explore planning came from the fact that we had only one A.I. programmer, but there are lots of A.I. characters. The thought was that if we can delegate some of the workload to these A.I. guys, we'd be in good shape. If we want squad behavior in addition to individual unit behavior, it's going to take more man-hours to develop. If the A.I. are really so smart, and they can figure out some things on their own, then we'll be all set! Before we continue, we should nail down exactly what we mean by the term planning. Planning is a formalized process of searching for sequence of actions to satisfy a goal. The planning process is called plan formulation. The planning system that we implemented for F.E.A.R. most closely resembles the STRIPS planning system from academia. (Orkin 2006)

Three Benefits of Planning:

The previous case study illustrates the first of three benefits of a planning system. The first benefit is the ability to decouple goals and actions, to allow different types of characters to satisfy goals in different ways. The second benefit of a planning system is facilitation of layering simple behaviors to produce complex observable behavior. The third benefit is empowering characters with dynamic problem solving abilities. (Orkin 2006)

- Decoupling goals and actions
- Layering behaviors
- Dynamic problem solving

Conclusion:

Real-time planning empowers A.I. characters with the ability to reason. Think of the difference between predefining state transitions and planning in real-time as analogous to the difference between pre-rendered and real-time rendered graphics. If we render in real-time, we can simulate the affects of lighting and perspective, and bring the gaming experience that much closer to reality. The same can be said of planning. By planning in real-time, we can simulate the affect of various factors on reasoning, and adapt behavior to correspond. With F.E.A.R., we demonstrated that planning in real-time is a practical solution for the current generation of games. Moving forward, planning looks to be a promising solution for modeling group and social behaviors, increasing characters' command of language, and designing cooperative characters.

(Orkin 2006)

Search: Utility ai in unity Search engine: google chrome

17. Goal Driven Behaviour

https://learn.unity.com/project/goal-driven-behaviour

Last accessed 20/10/24

Type: Unity learn / a tutorial

Author: Penny de Byl

Summary:

In this project you will learn about the Goal-Orientated Action Planning (GOAP) architecture used to create intelligent agents that can set goals and plan at achieving them. Unlike Finite State Machines, actions and states are uncoupled, making for a very flexible system. You will build a GOAP system from the ground up and implement it in a simple hospital simulation.

Project Objective

- Explain the GOAP architecture and discuss its advantages and uses.
- Develop a reusable GOAP library with C#.
- Implement GOAP in creating a simulated hospital environment.
- Understand the differences between goals, states, plans and actions.
- Develop an inventory system to assign and deassign game resources to agents for use in performing actions."

(de Byl, z.d.)

Search: Utility ai in unity Platform: Youtube

18. Utility AI For unity

https://www.youtube.com/watch?v=ejKrvhusU1I&list=PLDpv2FF85TOp2KpIGcrxXY1POzfYL-GWIb

Last accessed 20/10/24

Type: youtube playlist

Author: TooLoo

Summary:

A playlist on how to make utility ai from scratch for unity. Could come in handy for the experiment.

Search: use goap or behavior trees Search engine: google chrome

19. Choosing between Behavior Tree and GOAP (Planning)

https://www.davideaversa.it/blog/choosing-behavior-tree-goap-planning/

Last accessed 20/10/24

Type: Online article

Author: Davide Aversa

Summary:

Behavior trees vs planning:

1. **Much higher implementation complexity.** BTs are quite easy to understand and to implement. Moreover, BTs are already built-in into a lot of game engines! GOAP, on the other hand, is not as simple. It harder to implement, and it is harder to debug.
2. In general, **plan-based techniques are computationally more expensive than BTs**. Implementing GOAP in a way that is good enough for real-time games requires fine-tuning and a good design for the "state representation" and the set of possible actions (and we came back to point 1).
3. **By not writing the rules of AI by ourselves, we lose control of the AI**. If we say that the character's goal is to kill the player, we may have some situation in which the solution to this goal is too effective or, in general, not fun. Because fun is the goal, we need to change this, but we can only act on the goal, not the way the character reaches the goal. For another example, if the character starts doing something strange, it will be harder to understand *"why"*. In BTs, we can follow the tree and find the problem. In GOAP, this is much harder.

(Aversa 2018)

When to choose what?

1. **The number of possible action sequences, and the number of rules governing them, is too high**. For instance, I can have too many different possible actions (or better, too many ways to combine these actions), and encoding all of them becomes a nightmare.
2. **You do not know at "design phase" the possible ways a character can act.** For instance, your game may involve a lot of user-created content, and it may be impossible to predict how the player would interact with your characters during development.

(Aversa 2018)

Conclusion

This is a very high-level view of the problem, and it is nowhere complete. Things are more complicated and nuanced than that. For instance, there are many ways to make huge BTs more manageable. This reduces the importance of motivation number (1). The number (2), instead, remains still a hot motivation for planning. We will talk more about this in the future if you like.

(Aversa 2018)

20. Common issues with Behavior Trees and things you should competely avoid.

https://dev.epicgames.com/community/learning/tutorials/L9vK/unreal-engine-common-issues-with-behavior-trees-and-things-you-should-competely-avoid

Last accessed 20/10/24

Type: Online unreal Community post

Author: IntaxQ

Summary:

Do not try to implement a State Machine into Behavior Trees.

This is the *most* common problem around the community from what I can see. Behavior Trees invented after State Machines, to make developers life easy by getting rid of the complexity of transitions and unreadable sphagetti graphs. But now people started to implement states to Behavior Trees by decorators. **Behavior Trees prioritizate tasks**, and unlike State Machines, **they snap to most prior task when selected.** So they don't provide the efficiency of transitions and you have almost zero control over execution flow. If you try to implement a state pattern by selector & decorators combo, you'll end up doing weird hacks to control the execution flow and it'll get very complex and difficult manage on the long run. **Behavior Trees are not built with the mindset of jumping from a node to another by user's direct control.** You are going to prepare and control values in BTServices, then design decorators and selectors intelligently to create your execution flow. **If you are not able to do this, that means Behavior Trees are not answer to your problem.** If it suits your taste, either try StateTree, which is a mix of BT's selectors and state machines, or implement your own desired tool on your own. **There are many other solutions for AI like GOAP, Finite State Machines, Utility Editors etc.**(IntaxQ 2022)

Utility Systems saves the day most of the time.

Utility Systems are an abstract idea of selecting something by scoring multiple elements. It's something every game developer must get experienced with since it's a very helpful pattern/method and makes things easier to use in many game mechanics. Environment Query System is basically an utility scoring system that selects the item that has more score based on the filters you designed in the graph. Same method can be implemented to Behavior Trees. Though, there aren't much out of the box implementations for this that you can find for Behavior Trees.(IntaxQ 2022)

21. Comparing Behaviour Trees with Finite State Machin

https://www.zotero.org/groups/5200590/howest_dae_grad_work_vault/collections/8A9HILUY/search/bram%20vernimmen/titleCreatorYear/items/L2NUHHT5/attachment/ZREQXA9K/reader

This is a previous work from a student comparing finite state machines and behavior trees

Last accessed 20/10/24

Type: Research

Author: Bram Vernimmen

Conclusion:

So, how does a Behaviour Tree compare with a Finite State Machine when simulating animal behaviours using these structures? There is no such thing as the perfect structure so you mostly have to weigh your options first before picking one to use. As proven, the Finite State Machine is more performant than the Behaviour Tree, but is also more tightly coupled and makes the code less readable and maintainable. However it also lacks the tools to debug, making it harder to maintain once more behaviours have been added. If you are looking for a very lightweight structure that only needs to support a couple behaviours and you are confident in what you are doing. The Finite State Machine is a good fit. If you want to create an AI that has a lot of behaviours, should easily be expandable and you like the option to visually debug everything very easily. The Behaviour Tree is a good fit. A big helper will be the visual debugging aspect of the Behaviour Tree. If you are fairly new and want to make sure everything you do makes sense and works perfectly, this will be the best option to pick. For a more advanced approach using a game as example, it would be possible to use a Finite State Machine to simulate basic behaviours if an animal is really far away from a player. This will be more performant and the basic behaviours still give some sort of activity to the animal, making it look more dynamic. If the player goes closer, a Behaviour Tree can be activated, which a lot more behaviours. These behaviours could give a more in depth feel and activity to this animal. It will be less performant, but it will look a lot better. (Vernimmen, z.d.)

Future Work:

As this paper delved into the comparison of only 2 behavioural structures, it has the potential to test out many more. The reason why Behaviour Trees and Finite State Machines got picked is because it seemed most logical for basic behaviours. In that sense, it would be a lot better to also explore the more in depth feel of detailed behaviours. As the current simulation went, it was just a herd of zebras trying to survive. A simple addition to make the behaviours a lot more complex would be to add in natural predators like lions or hyenas and simulate their behaviours. If the route of adding predators gets explored, it would be nice to test different behaviour structures between predator and prey, and see what the effects are. We also saw the problem of a bottleneck with the Behaviour Tree, so it would be a good thing to address this in later research. Trying to optimise the structure and perhaps make it nearly as performant as a Finite State Machine. In the case of Finite State Machines, adding visual debugging tools and conditionals for transitions would be an amazing step in the right direction.(Vernimmen, z.d.)