ChatGPT/LLM Log - Document all prompts and output

Bryan Lim Li Cheng

Do your best to copy and paste all prompts and output into this document.

A white text on a white background

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

**Prompt:**

How would I do the behavior trees?

**Output:**

Great question. Since your FSM is already handling **how** the enemy performs behaviors (like patrolling or attacking), your **Behavior Tree (BT)** will control **what** the enemy wants to do at a higher level — it makes decisions based on priorities and conditions.

**What You’ll Build**

A **simple Behavior Tree system** with:

* **Nodes**: Base, Composite (Selector/Sequence), Condition, and Action nodes.
* A **BT Runner** on the enemy that evaluates the tree each frame.
* Integration between BT decisions and your FSM (EnemyState enum).

**Behavior Tree Structure (For Your AI)**

Here's what your BT might look like:

mathematica

CopyEdit

Root (Selector)

├── Sequence (Engage Player)

│ ├── Condition: Is Player In Detection Radius?

│ └── Selector (Attack Selector)

│ ├── Sequence (Melee)

│ │ ├── Condition: In Melee Range?

│ │ └── Action: Set FSM to MeleeAttack

│ └── Sequence (Ranged)

│ ├── Condition: In Ranged Range?

│ └── Action: Set FSM to RangeAttack

├── Sequence (Chase Player)

│ ├── Condition: Player in Detection Radius?

│ └── Action: Set FSM to Chase

└── Action: Set FSM to Patrol

**Step-by-Step Implementation**

**1. Define Base Node**

csharp

CopyEdit

public enum BTResult { Success, Failure, Running }

public abstract class BTNode

{

public abstract BTResult Tick();

}

**2. Composite Nodes (Selector & Sequence)**

csharp

CopyEdit

public class Selector : BTNode

{

private List<BTNode> children;

public Selector(List<BTNode> children) => this.children = children;

public override BTResult Tick()

{

foreach (var child in children)

{

var result = child.Tick();

if (result != BTResult.Failure)

return result;

}

return BTResult.Failure;

}

}

public class Sequence : BTNode

{

private List<BTNode> children;

public Sequence(List<BTNode> children) => this.children = children;

public override BTResult Tick()

{

foreach (var child in children)

{

var result = child.Tick();

if (result != BTResult.Success)

return result;

}

return BTResult.Success;

}

}

**3. Condition & Action Nodes**

csharp

CopyEdit

public class ConditionNode : BTNode

{

private System.Func<bool> condition;

public ConditionNode(System.Func<bool> condition) => this.condition = condition;

public override BTResult Tick() => condition() ? BTResult.Success : BTResult.Failure;

}

public class ActionNode : BTNode

{

private System.Func<BTResult> action;

public ActionNode(System.Func<BTResult> action) => this.action = action;

public override BTResult Tick() => action();

}

**4. Connect to Enemy FSM**

In your EnemyController.cs:

csharp

CopyEdit

private BTNode behaviorTree;

void Start()

{

// Your existing setup...

behaviorTree = CreateBehaviorTree();

}

void Update()

{

behaviorTree.Tick();

stateMachine.FSMUpdate(currentState);

}

private BTNode CreateBehaviorTree()

{

return new Selector(new List<BTNode>

{

new Sequence(new List<BTNode>

{

new ConditionNode(() => Vector2.Distance(transform.position, target.position) <= meleeRange),

new ActionNode(() => { currentState = EnemyState.MeleeAttack; return BTResult.Success; })

}),

new Sequence(new List<BTNode>

{

new ConditionNode(() => Vector2.Distance(transform.position, target.position) <= rangeAttackRange),

new ActionNode(() => { currentState = EnemyState.RangeAttack; return BTResult.Success; })

}),

new Sequence(new List<BTNode>

{

new ConditionNode(() => Vector2.Distance(transform.position, target.position) <= detectionRadius),

new ActionNode(() => { currentState = EnemyState.Chase; return BTResult.Success; })

}),

new ActionNode(() => { currentState = EnemyState.Patrol; return BTResult.Success; })

});

}