AI Perception and Pursuit System

Introduction

Welcome to AI Perception and Pursuit System Tutorial

This tutorial teaches you how to build intelligent enemy AI in Unreal Engine using industry-standard techniques. By the end of this guide, you will have created a fully functional enemy that can detect, chase, and engage the player.

What You'll Learn

By completing this tutorial, you will:

- **Understand Behavior Trees** - Learn how modern games structure AI decision-making
- **Implement AI Perception** - Create realistic enemy senses that detect the player
- **Build a Blackboard System** - Store and manage AI memory data
- **Create Chase Behavior** - Implement distance-based pursuit logic
- **Use Services** - Monitor conditions in real-time and update AI behavior
- **Debug AI Systems** - Identify and fix common issues

Practical Skills:

- Set up AI Controllers and Character blueprints
- Configure PawnSensing components
- Build multi-state Behavior Trees
- Implement custom tasks and services
- Use NavMesh for pathfinding

- Create distance-based state transitions

Prerequisites

Required Knowledge:

- Basic Unreal Engine familiarity (editor navigation, Content Browser)
- Blueprint fundamentals (nodes, execution pins, basic logic)
- Basic 3D concepts (coordinates, distances, spatial relationships)

Software Requirements:

- Unreal Engine 5.3 or later (UE 4.27+ also compatible)
- 16GB RAM recommended
- Third Person Template or any project with a playable character

Helpful But Not Required:

- Game AI concepts
- C++ knowledge (this is 100% Blueprint-based)
- Prior Behavior Tree experience

Learning Objectives

By the end of this tutorial, you will be able to:

1. Explain the core components of modern game AI (Perception, Decision-Making, Action)
2. Describe how Behavior Trees differ from Finite State Machines
3. Create a functional enemy AI that detects and pursues the player
4. Implement distance-based logic for chase, attack, and disengage behaviors
5. Configure AI perception systems with realistic vision parameters
6. Debug common AI issues systematically
7. Extend the system with your own behaviors and enemy types

Target Audience

This tutorial is designed for:

- Beginner to Intermediate Unreal Engine users learning game AI
- Game design students studying AI behavior
- Indie developers building action or stealth games
- Hobbyists creating enemy characters
- Anyone curious about game AI systems

Time Investment: Approximately 2-3 hours

What You'll Build

An enemy AI character that:

- Patrols or stays idle by default
- Detects the player within vision range (600 units)
- Chases the player using NavMesh pathfinding
- Stops pursuing when the player escapes (700+ units)
- Attacks when close enough
- Returns to idle state when losing the target

Systems Implemented:

- AI Controller with Behavior Tree
- Blackboard for data storage
- PawnSensing for player detection
- Custom chase task
- Distance-monitoring service
- State-based behavior switching

How to Use This Tutorial

Step-by-Step Approach:

1. Read the Theoretical Foundations
2. Follow the Implementation Guide
3. Test as you go
4. Complete Practice Exercises
5. Use the Debugging Guide when needed

Tips for Success:

- Do not skip steps
- Name things exactly as shown (case-sensitive!)
- Test frequently
- Use Print Strings for debugging
- Reference the screenshots
- Check the Debugging Guide if stuck

Troubleshooting:

1. Check the Debugging Guide for your specific issue
2. Verify all names match exactly
3. Compile all blueprints and fix errors
4. Add Print Strings to trace execution
5. Compare your setup to screenshots

Project Files Included

- **Tutorial Documentation** - Complete guide (theory + implementation + exercises)
- **Complete Implementation** - Fully functional reference project
- **Starter Template** - Partially complete project for practice
- **Video Tutorial** - 10-15 minute demonstration
- **README** - Quick start guide and project overview

Let's Get Started!

You are about to learn AI techniques used in AAA games like The Last of Us, Halo, and Splinter Cell. These fundamental concepts will serve as the foundation for more complex AI behaviors.

Ready? Let's begin with the Theoretical Foundations...

Theoretical Foundations:

---

1. Game AI Fundamentals

Purpose: Create believable, entertaining AI rather than optimal or unbeatable opponents.

Key Principles:

- Real-time performance (decisions in milliseconds)
- Predictable yet interesting behavior
- Designer-friendly tools for iteration

---

2. Behavior Trees

What: Hierarchical decision-making structures that control AI actions.

Why Better Than Finite State Machines:

- No need to define every state transition
- Modular and reusable
- Re-evaluates from root each tick (reactive)

Core Node Types:

- **Selector** - Try children until one succeeds (OR logic)
- **Sequence** - Execute children until one fails (AND logic)
- **Decorator** - Add conditions to child execution
- **Task** - Perform actions (move, attack, wait)

Execution: Tree evaluates top-to-bottom each tick, enabling immediate response to world changes.

---

3. Blackboard System

What: Shared memory storage for AI data (key-value pairs).

Purpose:

- Stores perception data (detected targets, last known positions)

- Decouples AI subsystems (perception writes, behavior tree reads)
- Observable (changes trigger behavior tree updates)

Example Keys:

targetActor: Object (player reference)

seeingTarget?: Bool (is target visible)

lastKnownPosition: Vector

---

4. AI Perception

What: Simulates realistic senses (vision, hearing) rather than omniscient AI.

Vision Components:

- **Sight Radius:** Detection range (e.g., 600 units)
- **Peripheral Vision Angle:** Field of view (e.g., 90°)
- **Line-of-Sight:** Can't see through walls
- **Lose Sight Radius:** Prevents flickering detection

Why Important:

- Creates immersion (AI feels "alive")
- Enables stealth gameplay
- Players understand limitations

---

5. Chase/Pursuit Behavior

Distance-Based States:

0-200 units: Attack Range

200-600 units: Active Chase

600-2000 units: Pursuit Zone

2000+ units: Disengage

Hysteresis Principle:

- Start chasing at 600 units

- Stop chasing at 800 units
- Buffer prevents rapid state flickering

Navigation: Uses NavMesh pathfinding (A* algorithm) for obstacle avoidance.

---

6. Services

What: Background processes that run while a behavior tree node is active.

Purpose:

- Monitor conditions (check distance every 0.5 seconds)
- Update blackboard based on changes
- Optimize performance (don't check every frame)

Example: Distance monitoring service clears blackboard when player exceeds chase range.

---

7. System Integration

The Complete Loop:

1. Perception detects player

2. Updates Blackboard (targetActor, seeingTarget?)

3. Behavior Tree reads Blackboard

4. Decorator checks conditions

5. Executes Chase Sequence

6. Service monitors distance

7. Clears Blackboard if too far

8. Tree re-evaluates → Returns to Idle

Observer Pattern:

- Decorators watch Blackboard changes
- **Observer Aborts: Self** - Immediately stops current behavior when condition changes
- Enables reactive AI without delays

---

8. Real-World Applications

Industry Examples:

- *The Last of Us* - Advanced perception with hearing visualization
- *Halo* - Squad tactics using shared blackboards
- *Splinter Cell* - Stealth mechanics based on AI perception limits

---

9. Key Advantages

Behavior Trees: ✅ Visual, designer-friendly ✅ Modular and reusable ✅ Reactive to world changes

Blackboard: ✅ Centralized data storage ✅ Decouples subsystems ✅ Observable for triggers

AI Perception: ✅ Realistic, believable behavior ✅ Enables stealth gameplay ✅ Fair and understandable

---

10. Summary

This system demonstrates three pillars of modern game AI:

1. **Sensing** - AI Perception simulates vision/hearing
2. **Thinking** - Behavior Trees make decisions based on blackboard data
3. **Acting** - Chase tasks move AI toward target, Services monitor state

Together, they create enemies that detect players realistically, pursue intelligently, and disengage appropriately - the foundation of engaging AI in action, stealth, and adventure games.

Step by Step Guide

Pawn Sensing

1. Create a new folder in your content browser and name it **AI**

2. Create a new Blackboard file – name it **BD_AI**

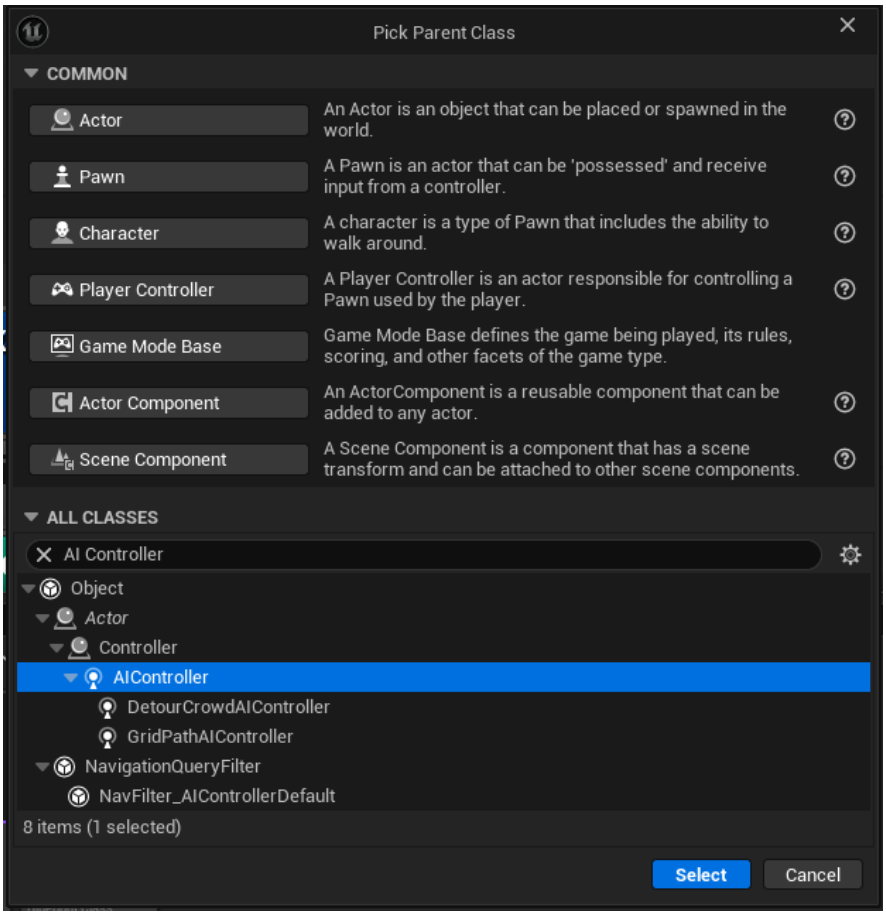
   You can find Blackboard in right-click in your **content browser -> Artificial Intelligence -> Blackboard**

3. Create a new Behavior Tree file in your AI folder and name it **BT_Enemy**

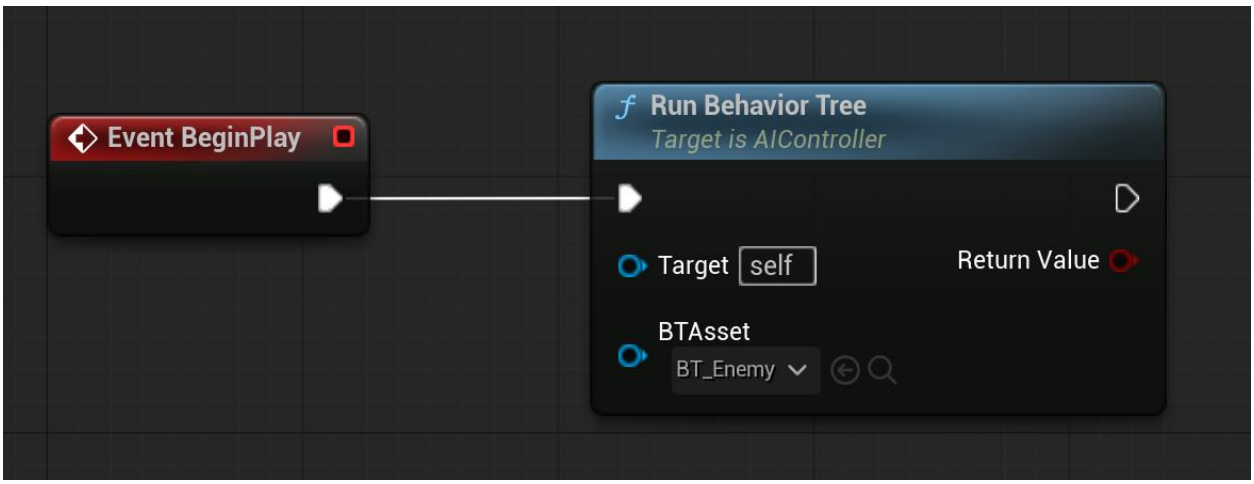
   content browser -> Artificial Intelligence -> Behavior Tree

4. Open the Behavior tree file you just created. In the details panel under Behavior Tree in Blackboard Asset **select your Blackboard BD_AI**

5. Create new Blueprint Class file in your AI folder, in All Classes search for **AI Controller** and click select. Name it **BP_AI_Enemy**



6. Open the Blueprint Class and go to EventGraph. And create the following Blueprint



7. Make sure to select your Behavior Tree in **BTAsset in Run Behavior Tree**

8. Create new Blueprint Class file in your AI folder, name it **BP_Enemy.** Open it and select Mesh under components -> and the in Details tab Select **Manny_Simple (Or any Skeleton Mesh you might have)** in Skeletal Mesh Asset.

9. After setting the Skeletal Mesh Asset, Open Class Default and then in Details tab Search for **AI Controller Class** under Pawn dropdown. And select the **BP_AI_Enemy** which we created earlier.

Also select **Placed in World or Spawn** under Auto Possess AI

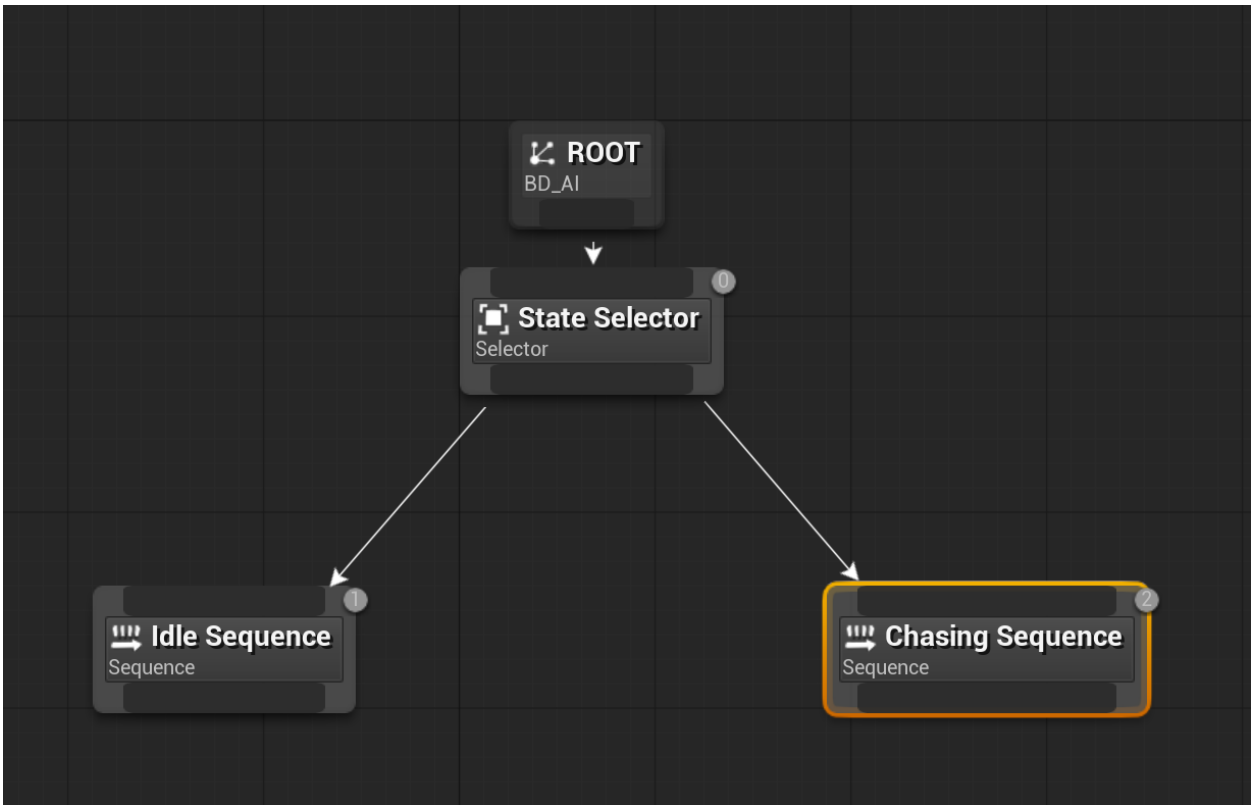
10. Search for Yaw in Details tab and **Disable Use Controller Rotation Yaw,** and in Components Tabs select Character Movement and then in Details tab, search for **Orient Rotation to Movement and Enable it.** After that Compile and Save it.

11. Open the Blackboard (BD_AI) and we want to create 2 Keys. Click on New Key and select Bool for the First Key -> name it **seeingTarget?**

For the second key, click on new key and select Obj and name it targetActor to create a new object key.

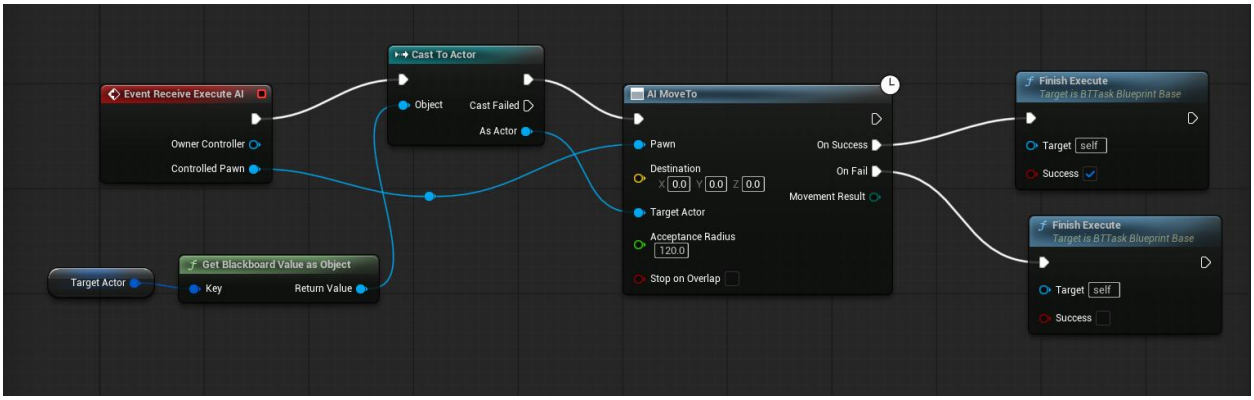
12. Open your Behavior Tree (BT_Enemy). There will be a root node, from the root note pull down an arrow which will open a tab to create a new node. Under Composite click on **Selector,** and in Details tab write the Node name to be **State Selector.**

After that create 2 Sequence nodes from the State Selector. **Idle and Chasing**

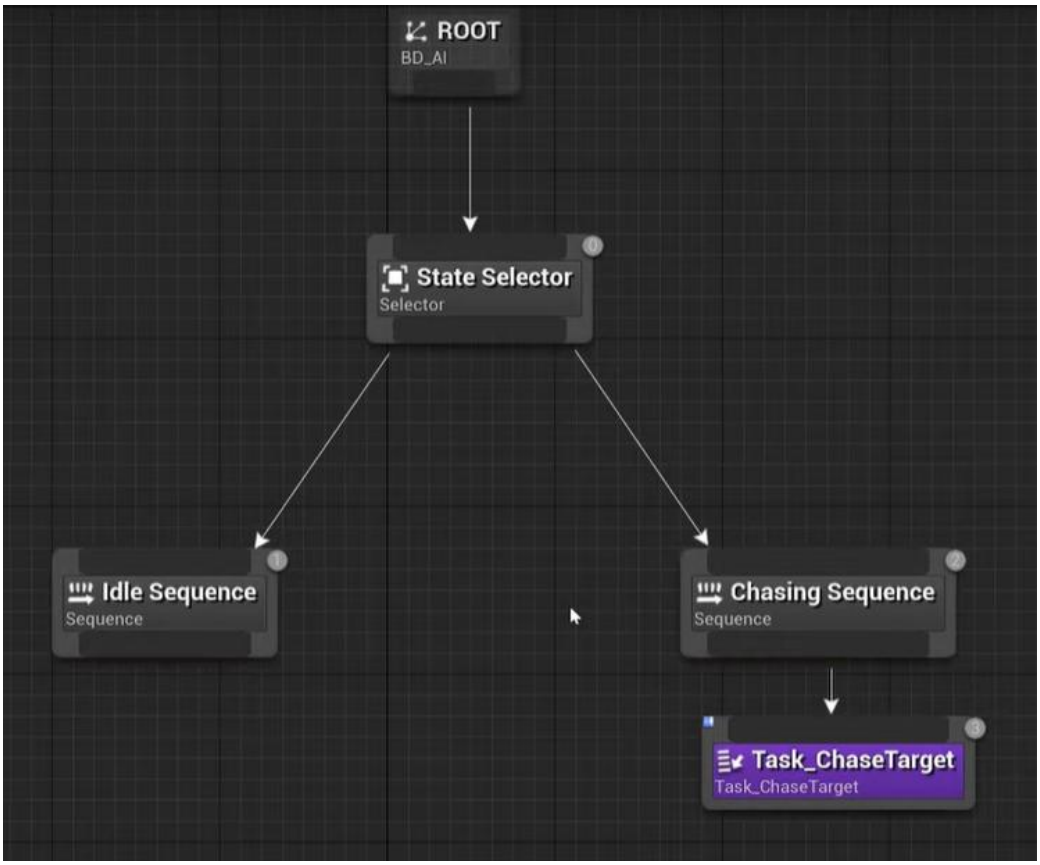


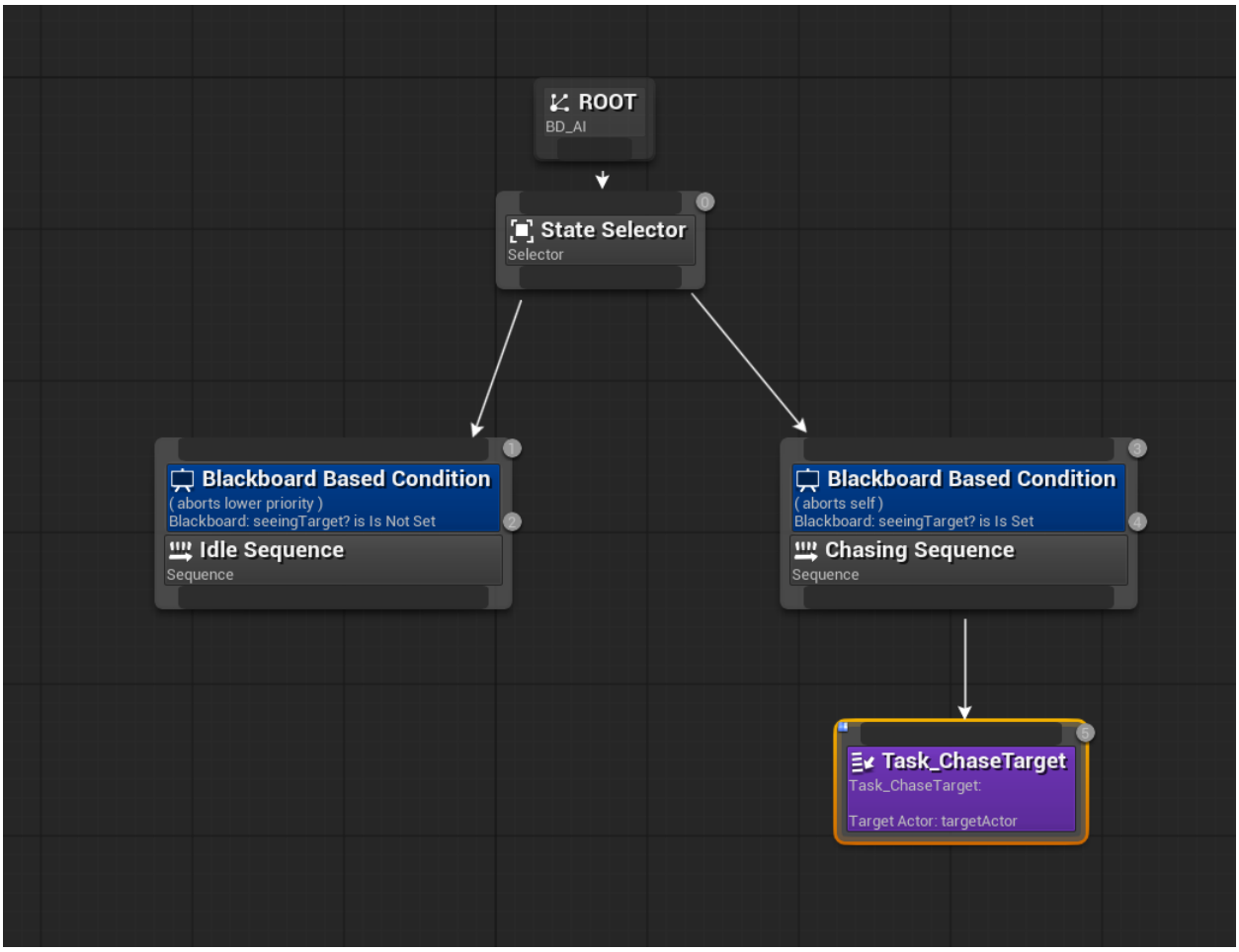
13. After Creating the Basic Behavior Tree with the Logic based on the State Selector and 2 sequences. We want to create a new Task for the Behavior Tree. Click on **New Task** in the top tool bar, and create a new folder names **Tasks** and give the name to the Task -> **Task_ChaseTarget**, and the save.

14. Open the created Task File and then in the left tab under Functions click on **Override dropdown** and the select **Receive Execute AI.** Create a new Variable named **targetActor of type Blackboard Key Selector** After that follow the below Blueprint and create it in the Task File, and the compile and save the file.



After that add the Task to the Chasing Sequence





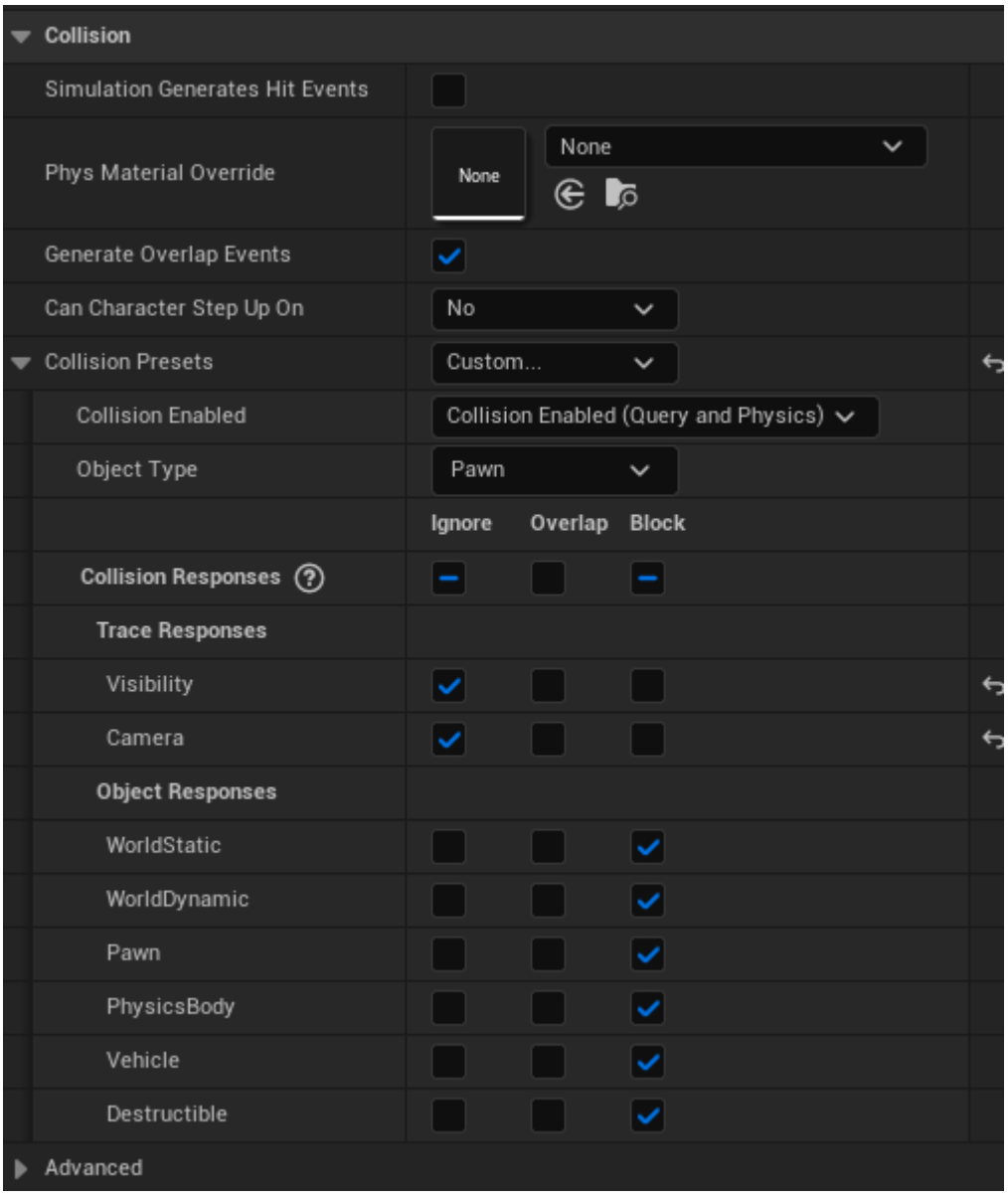
15. Once this is done, open your main level. Look for this Icon below in your Top tool Bar.

Under Place Actor -> Go to Volume -> **Select Nav mesh bounds volume.** Place it in the center of your level scene and scale it to cover the entire level. In this case, I did X,Y,Z = 20.0

(This will help the AI understand all the accessible areas for the AI, Unreal Engine Calculates all the possible paths that the AI can move around.) **Crucial Step!**

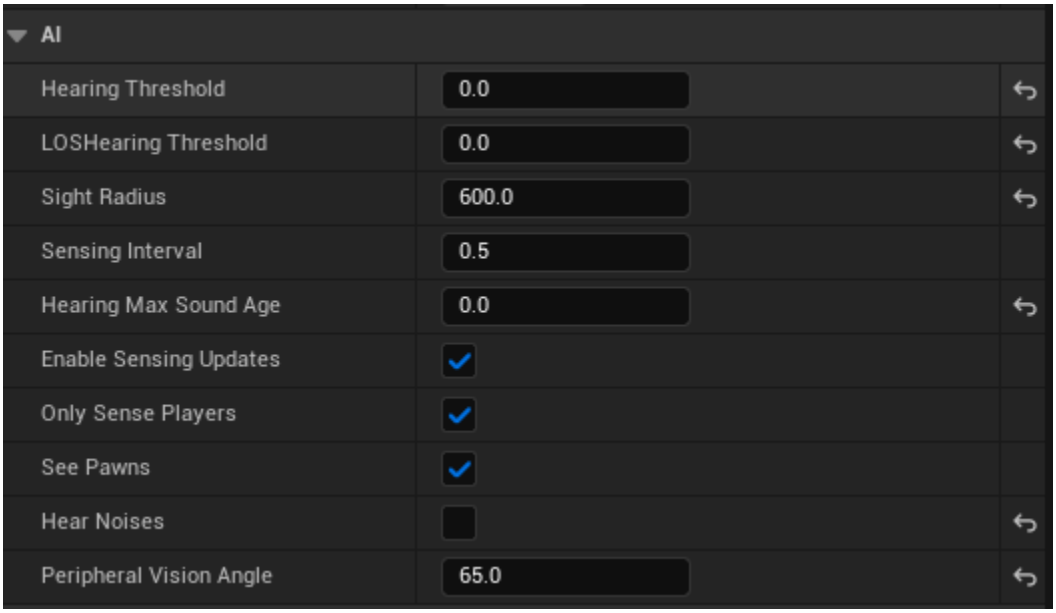
16. You can place the BP_Enemy in your level and Play the level in Editor. And see if the NPC follows you around. If the Enemy is moving too fast, go to BP_Enemy and in Character Movement component, set **Max Walk Speed to 350cm/s**

17. If the Camera collides with the Enemy NPC, go to Mesh in BP_Enemy. In Collision Presets under Collision. Select **Custom** and **check the Ignore Camera Box under Trace Responses.** Similarly Do it in Capsule Component (CollisionCylinder), In Collision Presets select Custom and check the Ignore Camera Box under Trace Responses.

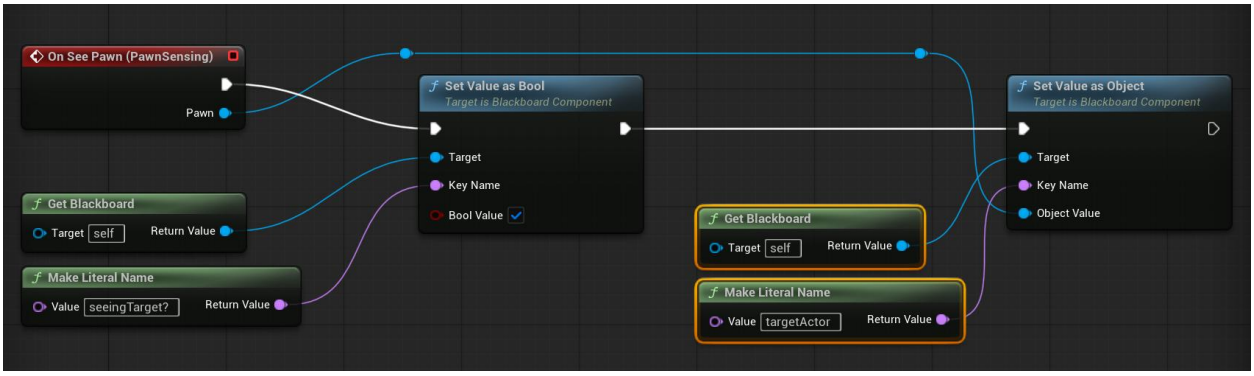


18. Now lets implement the logic of our Behavior Tree to the Enemy NPC model. It moves towards you when it sees you. So go to the BP_AI_Enemy file and in components tab Add a new component -> **Pawn Sensing**

We will be only implementing Vision in this PawnSensing for now, you can try out Hearing sense as an additional exercise.



19. Once this is set up, in the same file, under the Events tab in Pawn Sensing. Click on **On See Pawn,** this will create a new event in the event Graph



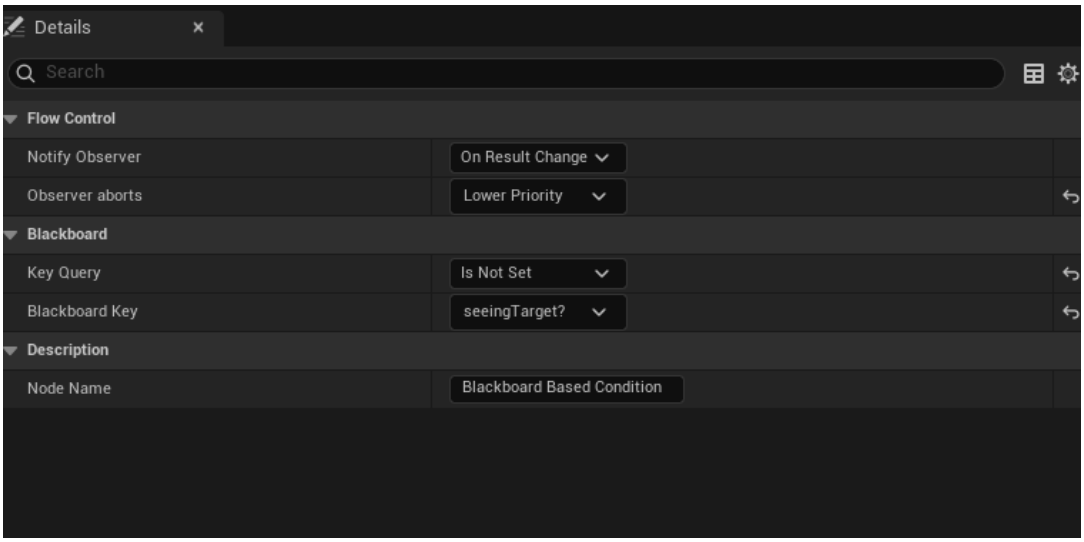
Implement the exact Blueprint as shown in the Image above. **Make sure the Key Names are exactly the same as created in the Blackboard.**

20. So in the Behavior Tree to specify different values that Sequences need to have, which are called decorators. So we need to add decorators to the Idle Sequence as well as the Chasing Sequence.

Right click on the **Idle Sequence -> Add Decorators -> Select Blackboard**

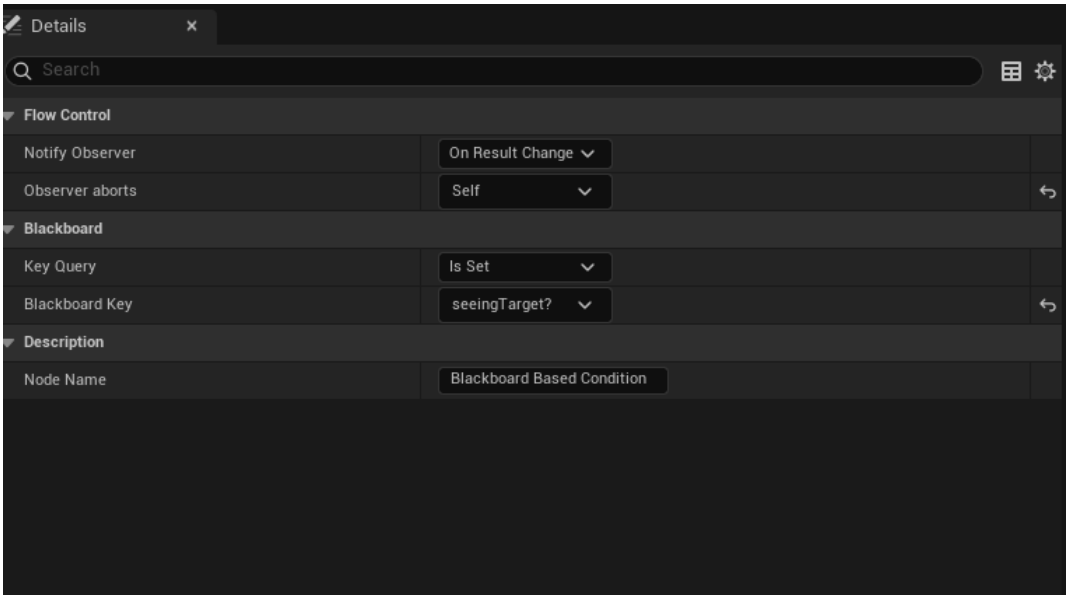
Click on the Blackboard Based condition Tab as setup the conditions in Details tabs as given below

Idle Sequence – Blackboard Based Condition



Similarly, Click on Add Decorator to the Chasing Sequence, Select the Blackboard Decorator and configure the Decorator like below

Chase Sequence – Blackboard Based Condition

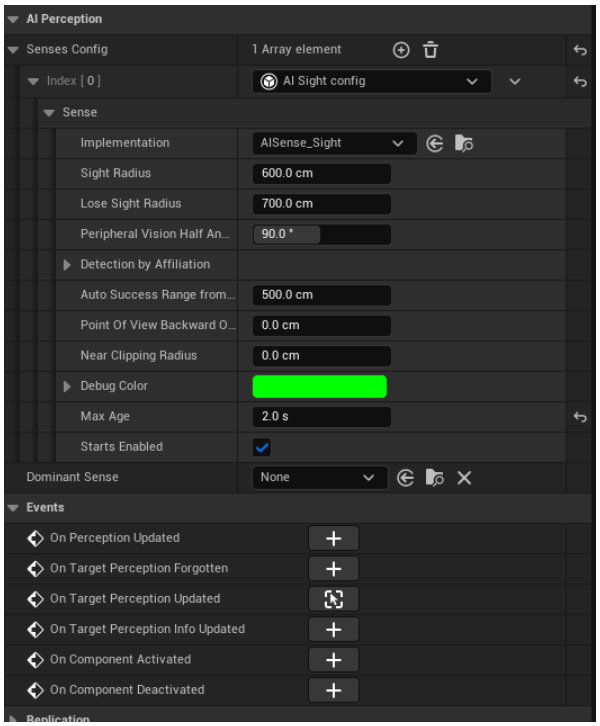


21. You can add Animation Montage or Anim Class to your Enemy NPC character to give certain movement characteristics to the NPC.

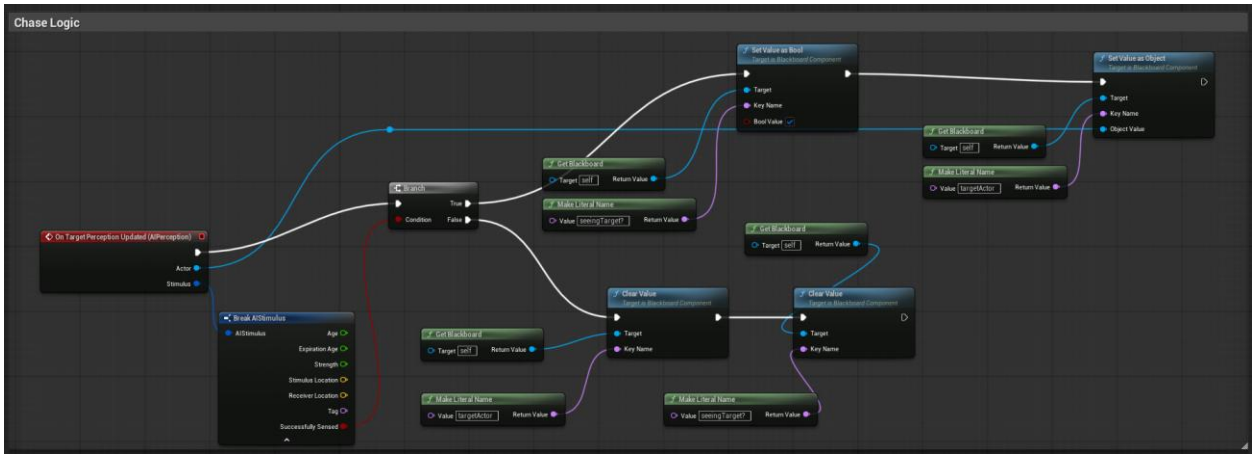
AI Perception

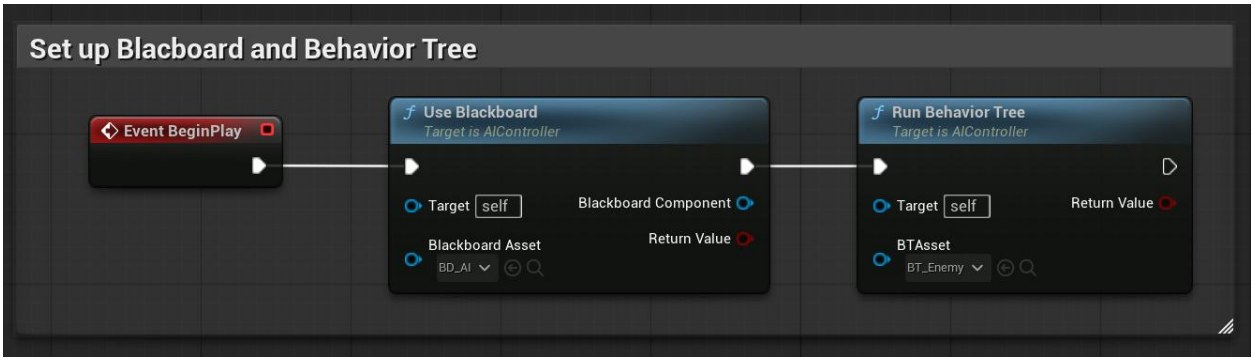
1. Create a New Blueprint Character Class named **BP_Enemy_2**
2. Create a new Blueprint AI Controller Class named **BP_AI_Enemy_2,** in the components Tab add **AIPerception.** Then in Details Tab under AI Perception, Add a new sense in **Senses config, in Index select AI Sight config.**

   Under Sense dropdown, you can set different variables like Sight Radius and Lose Sight Radius.

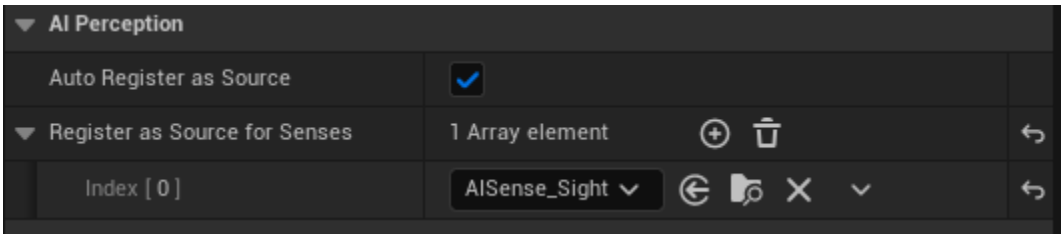


3. Once this is setup in Events tab click on On Target Perception Updated, Create these 2 different Blueprints to setup the AI Perception for the Enemy_2

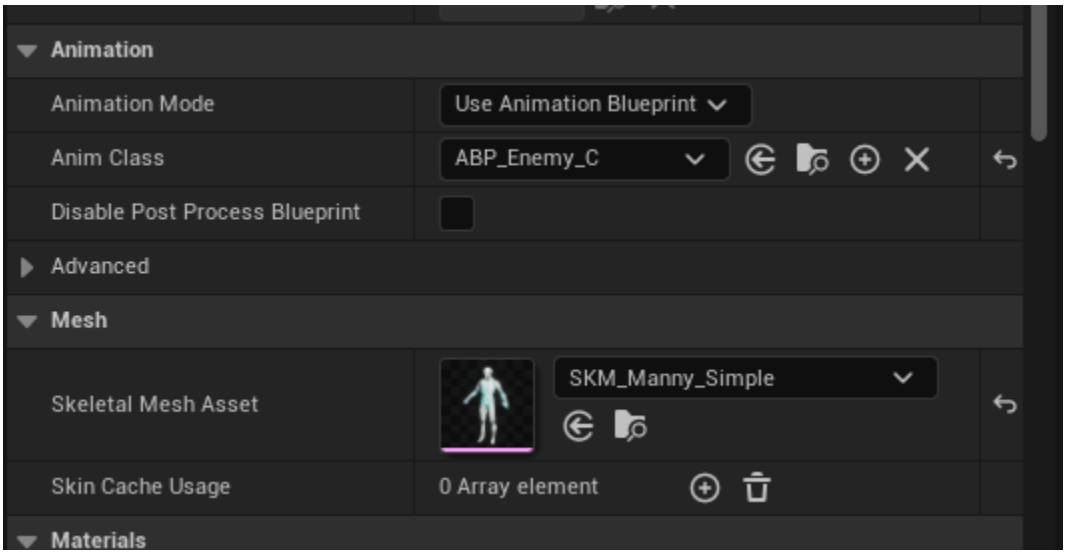




4. For the Enemy to sense the player character, you need to add **AIPerceptionStimuliSource in your BP_ThirdPersonCharacter and select the following setup.**



5. Also make sure to add the Skeletal Mesh Asset as Manny Simple or any of your desired mesh, and add Anim Class as we created for our previous enemy character.



6. Reduce the Walk Speed for the Enemy to 350cm/s.

Debug Guide

Problem 1: AI Won't Chase the Player

Quick Checks:

A. PawnSensing Settings

- Open **BP_AI_Enemy** → Select **PawnSensing** component
- Verify: ✅ See Pawns checked, ✅ Sight Radius = 600, ✅ Peripheral Vision Angle = 65-90

B. On See Pawn Event

- In **BP_AI_Enemy Event Graph**, verify **"On See Pawn"** event exists
- Must connect to: Set Value as Bool (seeingTarget?) → Set Value as Object (targetActor)

C. Blackboard Key Names

- Keys must match **exactly**: seeingTarget? (with ?) and targetActor (case-sensitive)
- Open **BD_AI** blackboard to verify exact spelling

D. Behavior Tree Decorator

- Click **Chasing Sequence** → Check decorator settings:
    - Key Query: **Is Set**

- Blackboard Key: **seeingTarget?**
- Observer Aborts: **Self** (NOT "None")

E. NavMesh Missing

- Press **P** in viewport - should see green overlay
- If no green: Place Actors → Volumes → **Nav Mesh Bounds Volume** → Scale to cover level (X=20, Y=20, Z=20)

F. AI Controller Not Assigned

- Open **BP_Enemy** → Class Defaults → AI Controller Class = **BP_AI_Enemy**
- Auto Possess AI = **Placed in World** or **Spawned**

G. Behavior Tree Not Running

- **BP_AI_Enemy** Event Graph: Event BeginPlay → Run Behavior Tree (BTAsset = **BT_Enemy**)

Quick Test:

Add Print String after "On See Pawn":

On See Pawn → Print String: "PLAYER DETECTED!"

- Prints = PawnSensing works
- No prints = PawnSensing issue

Problem 2: Blackboard Keys Not Found

Quick Checks:

A. Key Names Misspelled

- Blackboard keys are **case-sensitive**
- ❌ targetactor ≠ targetActor
- ❌ seeingTarget ≠ seeingTarget?
- Open **BD_AI** → Write down exact names → Update all blueprints to match

B. Keys Not Created

- Open **BD_AI** → Verify keys exist:
    - seeingTarget? (Bool type)
    - targetActor (Object type, Base Class = Actor)
- If missing: New Key → Select type → Name exactly

C. Wrong Blackboard Asset

- Open **BT_Enemy** → Details Panel → Blackboard Asset = **BD_AI**

D. Task Variable Wrong Type

- In **Task_ChaseTarget** → targetActor variable must be type: **Blackboard Key Selector**
- Instance Editable: ✅ Checked

Quick Test:

Get Blackboard → Get Value as Object (targetActor) → Print String

- Prints "None" = Key name wrong
- Errors = Key doesn't exist

Problem 3: Service Not Running

Quick Checks:

A. Service Not Attached (Most Common)

- **BT_Enemy** → Click **Chasing Sequence** → Details → Services
- Must show service name
- If missing: Click "+" → Select **BTS_CheckChaseDistance** → Save
- Look for small green icon on node

B. Service Interval = 0

- Open **BTS_CheckChaseDistance** → Click background → Details Panel
- Interval: Should be **0.5** (not 0)

C. Chasing Sequence Never Activates

- Service only runs when Chasing Sequence is active
- Verify AI enters chase mode (see Problem 1)

D. Service Has Errors

- Open **BTS_CheckChaseDistance** → Compile
- Fix any red error nodes
- Verify execution flow is complete

E. Wrong Parent Class

- Service must inherit from **BTService_BlueprintBase**
- Class Settings → Parent Class (check)

Quick Test:

Add at service start:

Event Receive Tick AI → Print String: "SERVICE TICK"

- Prints every 0.5 sec = Service running
- No prints = Not attached or not activating

Problem 4: NavMesh Not Working

Quick Checks:

A. No Nav Mesh Bounds Volume

- Press **P** in viewport → Should see **green overlay** on floors
- If no green:
    - Place Actors → Volumes → **Nav Mesh Bounds Volume**

    - Place in level → Scale (X=20, Y=20, Z=20)
    - Press P again to verify green appears

B. NavMesh Too Small

- Press **P** → Green doesn't cover all walkable areas
- Select Nav Mesh Bounds Volume → Increase Scale

C. Using Wrong Movement Node

- Use **AI MoveTo** (uses NavMesh) ✅
- NOT "Simple Move To" (ignores NavMesh) ❌

D. Acceptance Radius Too Small

- In **AI MoveTo** node → Acceptance Radius = **120-200** units
- Too small = AI gets stuck trying to reach exact point

E. NavMesh Not Rebuilt

- After changing level geometry: **Build** → **Build Paths**

F. CharacterMovement Missing

- **BP_Enemy** → Components → Verify **CharacterMovement** exists
- Max Walk Speed = 350-600

Quick Test:

- Press **P** in viewport

- Green = walkable ✅
- No color = not in NavMesh ❌
- Test in large open area first

📋 General Debugging Tips

1. Use Print Strings Everywhere

Print String with different colors:

- Green = Success/Entry point

- Yellow = Processing

- Red = Errors/Exits

- Cyan = Data values

2. Check Output Log

- Window → Developer Tools → **Output Log**
- Shows errors, warnings, and print messages

3. Compile All Blueprints

- Click **Compile** in each blueprint
- Yellow warnings = potential issues
- Red errors = must fix

4. Test Incrementally

- Test each system separately:
    1. Does PawnSensing detect? (Print after On See Pawn)
    2. Does Blackboard update? (Print blackboard values)
    3. Does BT switch states? (Print in each sequence)
    4. Does service run? (Print in service)
    5. Does AI move? (Print MoveTo success/fail)

5. Visual Debugging

- Press **P** = Show NavMesh
- Press **'** (apostrophe) = Show AI debugging info
- In Behavior Tree: Enable "Show Execution Flow" for visual feedback

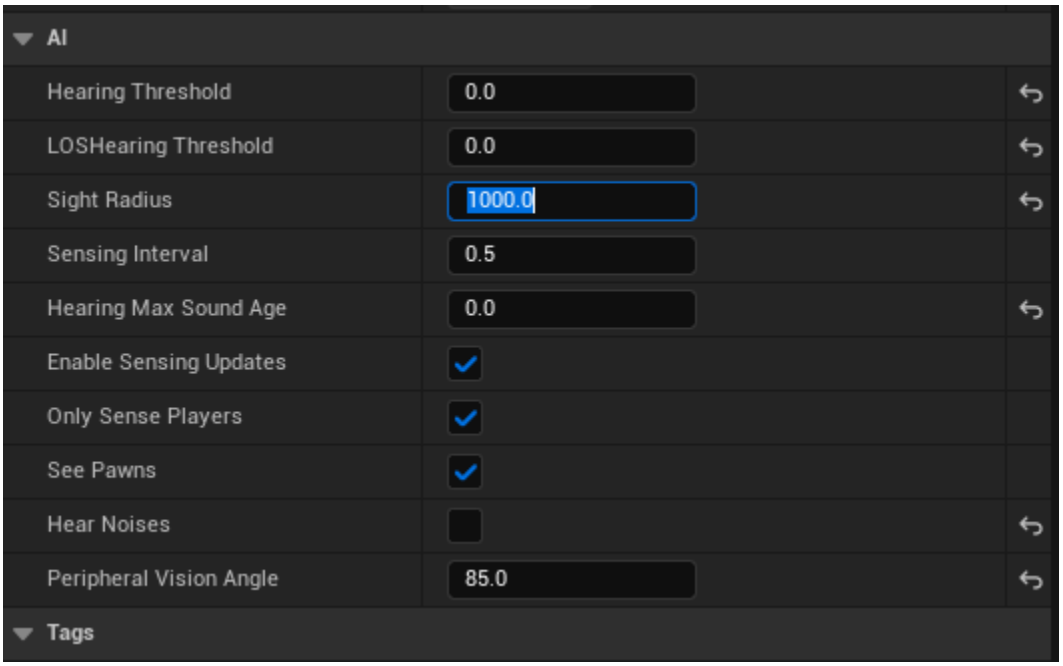
6. Common Mistake Patterns

- ❌ Observer Aborts = "None" (very common!)
- ❌ Blackboard keys misspelled (case-sensitive!)
- ❌ Service not attached to BT node
- ❌ NavMesh not covering area
- ❌ AI Controller not assigned to character

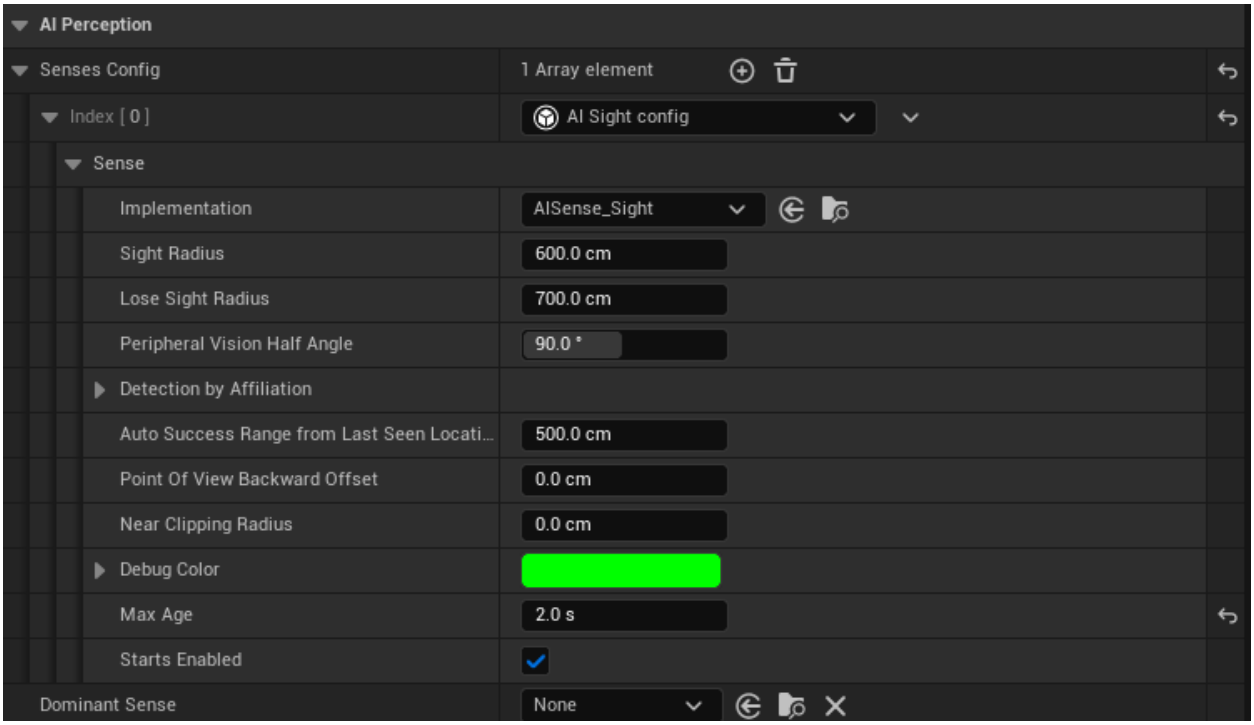
Practice Exercises

Exercise 1: Change the chase distance to 1000 units

Solution: Increase the Sight Radius so it has a longer sight radius.



Exercise 2: Create a second enemy type with different settings



Conclusion

Congratulations!

You have successfully built a complete AI Perception and Pursuit System! You now have a working enemy AI that can detect, chase, and engage the player using industry-standard techniques.

What You've Accomplished

Through this tutorial, you have learned and implemented:

- **Behavior Trees** - Hierarchical decision-making with Selector and Sequence nodes
- **Blackboard System** - Shared memory for AI data storage
- **AI Perception** - Realistic vision detection using PawnSensing
- **State Management** - Decorators that switch between Idle and Chase states
- **Chase Behavior** - Pathfinding-based pursuit using NavMesh
- **Services** - Background processes monitoring distance and state
- **Observer Pattern** - Reactive decorators responding to state changes

- **Debugging Skills** - Systematic troubleshooting approaches

You now understand the fundamental architecture of modern game AI and can apply these principles to any enemy type or behavior.

Key Takeaways

The Three Pillars of Game AI:

1. **SENSING** (AI Perception) - How AI knows about the world
2. **THINKING** (Behavior Trees + Blackboard) - How AI makes decisions
3. **ACTING** (Tasks + Services) - How AI executes behaviors

Core Principles:

- **Modularity** - Add or remove behaviors without breaking existing logic
- **Reactivity** - AI responds instantly to world changes
- **Decoupling** - Blackboards separate data from logic
- **Scalability** - Works for simple enemies and complex bosses

---

Next Steps: Expanding Your AI

Beginner Extensions:

- Add patrol behavior with waypoints
- Customize detection range and parameters

- Create multiple enemy types with different behaviors
- Add visual feedback showing detection range

Intermediate Challenges:

- Search behavior at last known position
- Multiple attack animations with distance-based selection
- Hearing sense for sound detection
- Team coordination between multiple enemies

Advanced Projects:

- Cover system when low on health
- Flanking behavior with multiple enemies
- Dynamic difficulty based on player performance
- Multi-phase boss fights

Real-World Applications

These techniques are used in:

Action Games: Devil May Cry, God of War, Bayonetta **Stealth Games:** Metal Gear Solid, Splinter Cell, Hitman **Horror Games:** Resident Evil, Dead Space, Alien: Isolation **Strategy Games:** XCOM, Civilization, StarCraft

---

Resources for Further Learning

Official Documentation:

- Unreal Engine AI Documentation
- Behavior Tree Quick Start Guide
- AI Perception System Guide

Books:

- "Behavioral Mathematics for Game AI" by Dave Mark & Kevin Dill
- "Game AI Pro" series by Steve Rabin
- "Programming Game AI by Example" by Mat Buckland

Online Communities:

- Unreal Slackers Discord
- r/unrealengine Reddit
- Unreal Engine Forums

Best Practices to Remember

- Start simple, add complexity incrementally
- Test frequently
- Use Services with intervals, not every frame
- Leverage Decorators for automatic state transitions

- Design for readability
- Balance challenge (AI should be fair and beatable)
- Profile performance in large scenes
- Iterate based on playtesting

Troubleshooting Reminder

If issues arise:

1. Reference the Debugging Guide
2. Check Blackboard key names (case-sensitive)
3. Verify Observer Aborts is set to "Self"
4. Add Print Strings for debugging
5. Compare to screenshots

Final Thoughts

Building believable AI is both an art and a science. The technical implementation is the foundation, but great AI comes from iteration, playtesting, and understanding player psychology.

The best AI is not the smartest, it is the one that creates the most engaging gameplay experience.

You now have the tools to create enemies that feel alive, reactive, and challenging. These fundamental techniques will serve you well in any game you build.

What's Next?

1. Complete the Practice Exercises
2. Experiment with different parameters
3. Apply these techniques to your own project
4. Share your work with the community
5. Explore advanced AI topics (GOAP, Utility AI, Machine Learning)

Thank You!

Thank you for following this tutorial. You have taken an important step in your game development journey. AI programming may seem intimidating at first, but breaking it into manageable components makes it accessible.

Remember: Every AAA game AI system started with the basics you have learned here. Keep building, keep learning, and keep creating!

Feedback & Support

- Star the GitHub repository if helpful

- Share your implementations in the community
- Report issues via GitHub Issues
- Suggest improvements for future learners

Questions? Check the Debugging Guide, search Unreal forums, or reach out to the community.

Now go build something amazing!

Tutorial Created By: Nachiket Sahare

Date: December 2024

Unreal Engine Version: 5.3+