MANIC AI

Features

1. Time & Scheduling

- **Clock & Timetable**
    - **UManicClockPluginBPLibrary**: Set or advance in-game time (SetInGameClockTime, TickManicClockAndGet).
    - **FManicTime**: Defines day/hour/minute/second slots.
    - **UManicTimetableNodeBPLibrary**: Finds the active timetable slot for a given time.
    - **UManicServiceSetTimeKey** (BT Service): Updates Blackboard keys with the current time or an active slot name.
- **Day of the Week**
    - **UManicDayOfWeekBPLibrary**: Converts day numbers into weekday strings (e.g. 1 → “Monday”).
- **Time Events**
    - **UManicTimeEventBPLibrary**: Detect if current time has reached a target (daily or once-only triggers).

MANIC AI

2. AI Behaviour Tasks

- **Find & Move**
  - **UManicFindActorInRegion**: Searches a radius for an actor of a specific class, checking a boolean property (e.g. bIsFree).
  - **UManicTaskFindSpotInArea**: Randomly picks a reachable point within a radius.
  - **UManicTaskGoToLocation**: Finds a random spot and moves there, re-trying if it times out.
  - **UManicTaskMoveToTarget**: Moves AI to a Blackboard actor or location, with obstacle checks.
- **Obstacle Avoidance & Stuck Detection**
  - **UManicObsAvoidance**: Offsets the AI's goal if blocked, detects if it's stuck, and fails if movement is impossible.

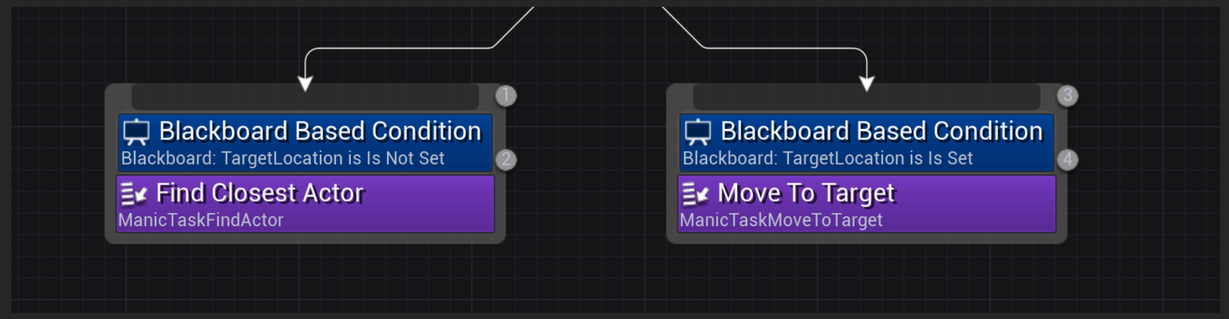
- **Actor Finding Variants**
  - **UManicTaskFindActor**: Finds the closest actor of a given class.
  - **UManicTaskFindFreeActor**: Similar to above, but ensures no other AI is targeting that actor.
- **Interaction**
  - **UManicInteract**: Executes a custom “PerformGenericAction” on the AI Controller and optionally waits.

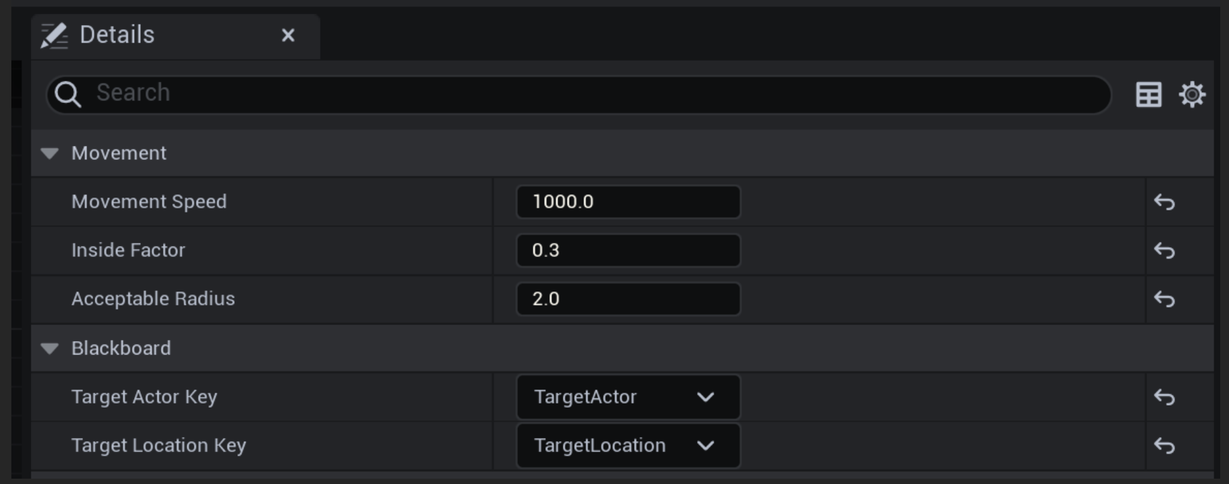
3. Dynamic Needs System

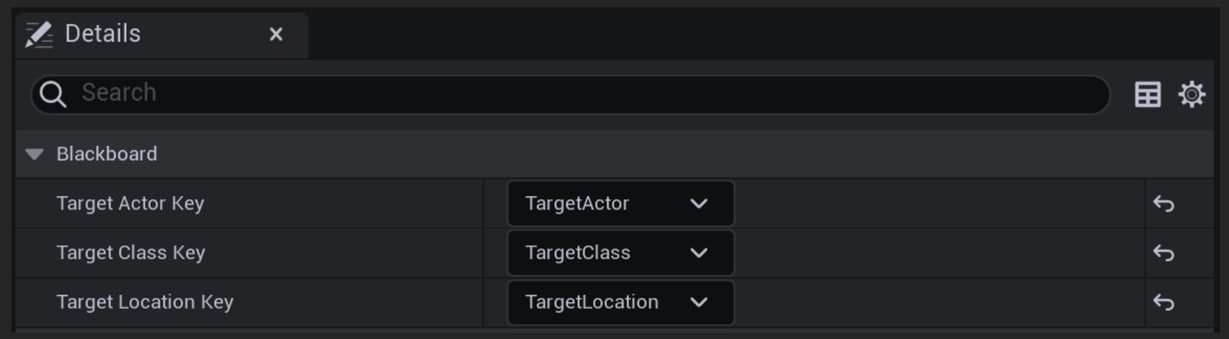
- **UDynamicManicNeedsComponent**
    - Tracks multiple “needs” (e.g. hunger), each ticking up or down at a chosen rate.
    - Fires OnAnyNeedThresholdCrossed when a threshold is reached and OnAllNeedsUpdated every tick call.
    - Add/remove or adjust needs at runtime (AddNeed, RemoveNeedAtIndex, etc.).

4. AI Controller

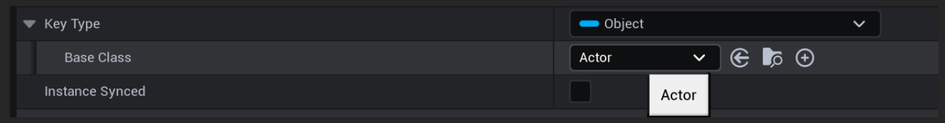
- **AManicAiController**
  - Holds a ControllerTimetable array (alternative to the service's internal timetable).
  - ActivityMap (string → soft object pointers).
  - **SetActivityKeyOnBlackboard**: Loads an actor from the map and sets it in a Blackboard key.
  - **PerformGenericAction**: A Blueprint event for AI-specific actions.





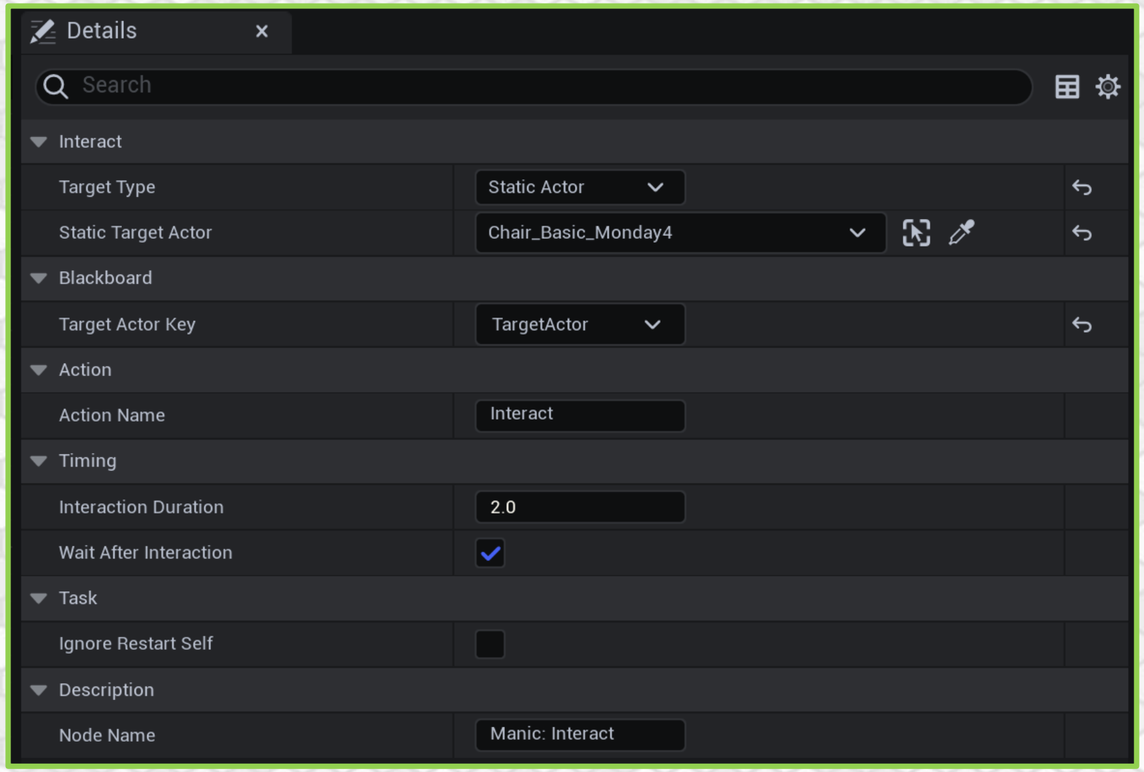


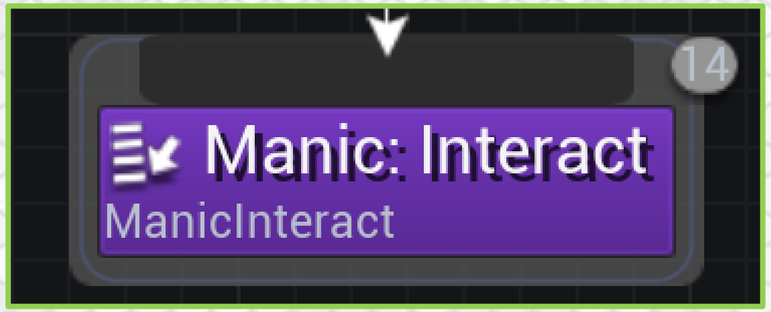
This task is designed for use in Unreal Engine's Behaviour Trees. Essentially, it commands an NPC to move towards a target (which can be either an actor or a location) and then stops it at a specifically computed point. When an actor is the target, the code calculates a stop location by taking the actor's centre and offsetting it inward by a fraction of its collision radius (controlled by the InsideFactor) plus any additional adjustments you set (via AcceptableRadius, ExtraStopOffset, or SurfaceOffset). The task then sends a move request to the AI, and its TickTask function continually checks whether the NPC has reached this computed stop location. Once the NPC is within a defined threshold of that point, the task stops the movement and completes, ensuring the NPC stops exactly where you need it—even inside the actor's collision bounds if required.

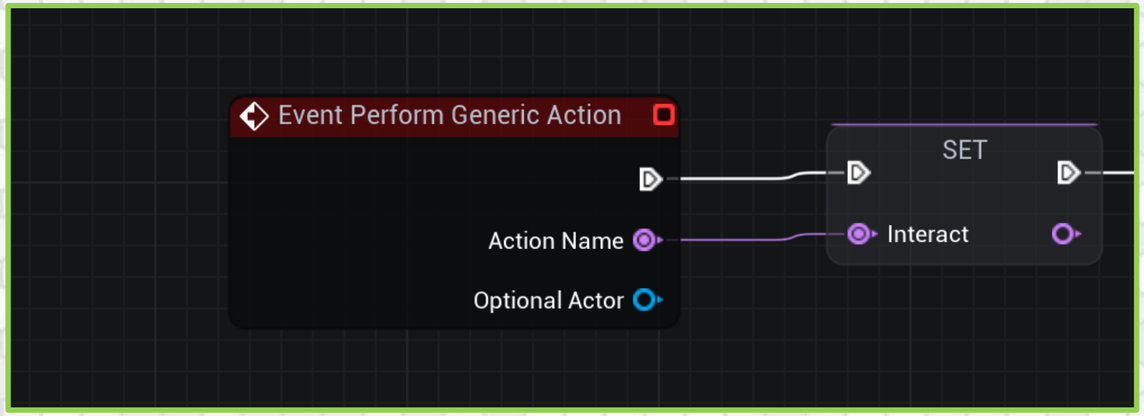


Manic Find Actor

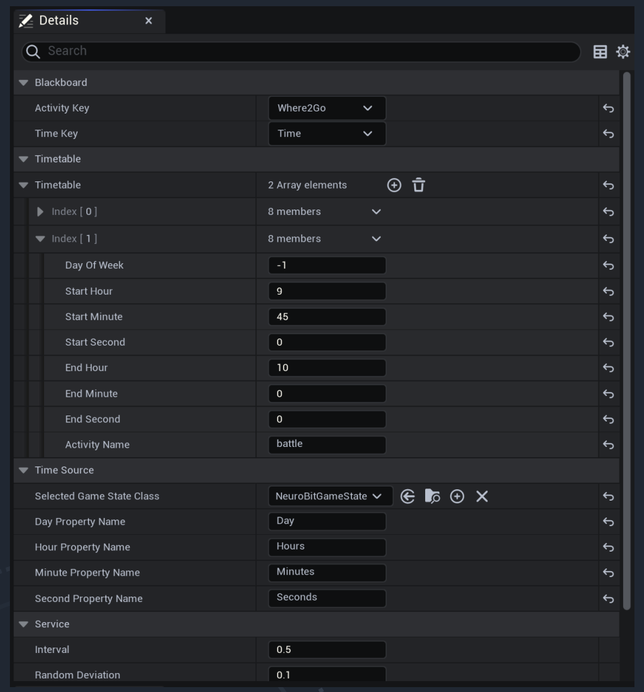
MANIC AI

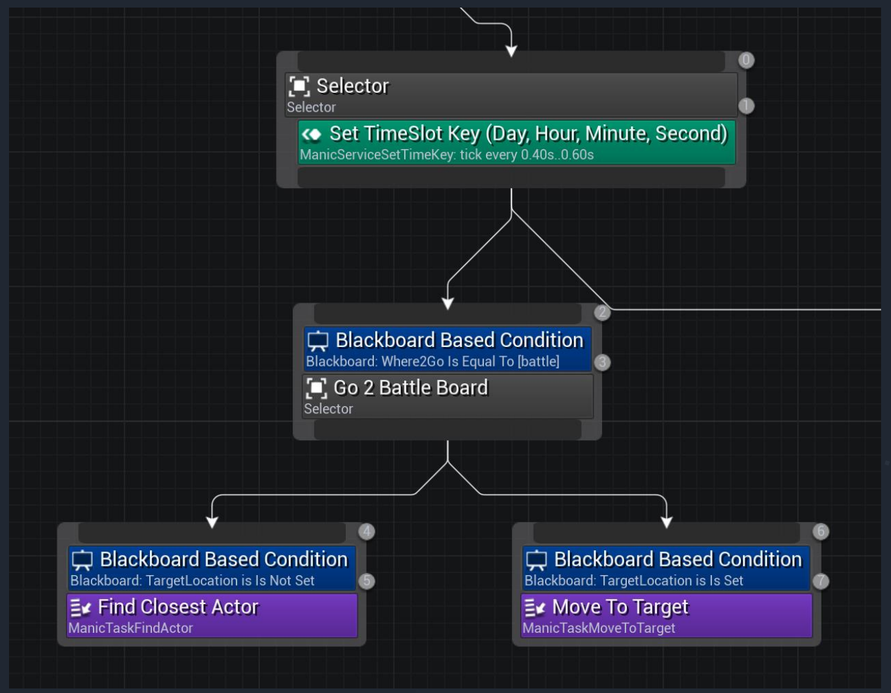






The system consists of two parts: a custom AI controller and a Behaviour Tree task. The custom AI controller (ManicAiController) defines a generic interaction function—PerformGenericAction—which you override in Blueprint to specify exactly what should happen when an NPC interacts (for example, sitting, opening a door, or triggering an event). The Behaviour Tree task (ManicInteract) retrieves an interactable actor (either from a blackboard key or a static reference) and calls PerformGenericAction on the AI controller, passing along an action name and the target actor. This setup keeps your interaction logic generic and flexible, so you can easily swap out or update behaviors without reworking your whole system.

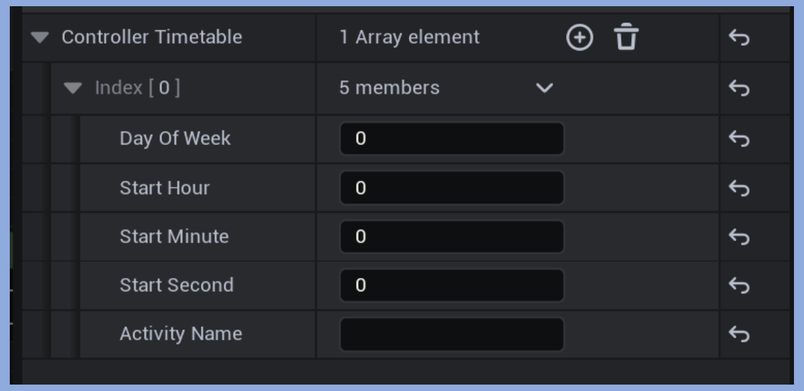


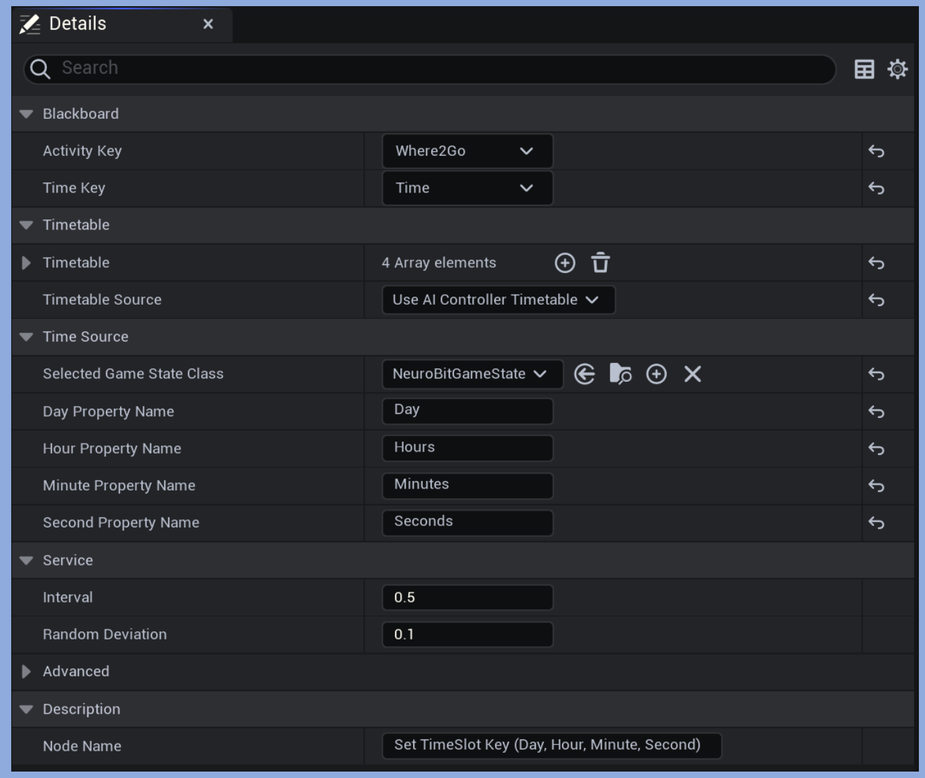


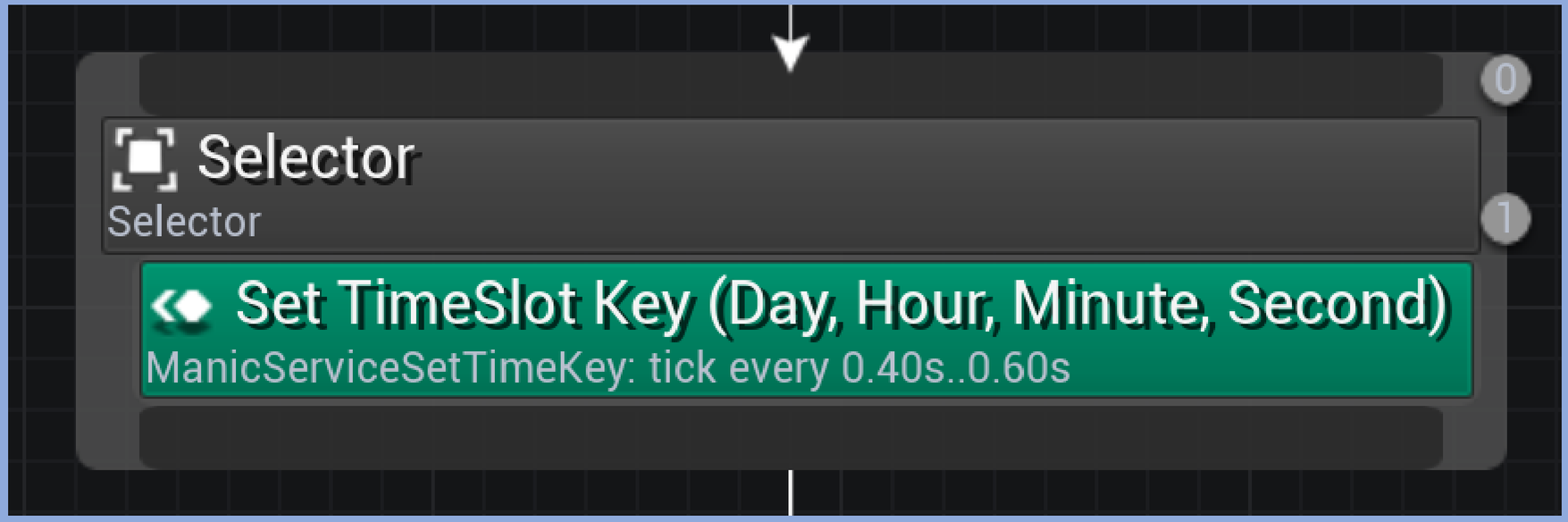
MANIC AI

Manic Service Set Time Key

To use **Manic Service Set Time Key**, simply add it as a Service node in your Behaviour Tree, then create two Blackboard string keys—for example, “Where2Go” for the slot outcome and “Time” for the current time. In the Service’s Details panel, assign those keys under ActivityKey and TimeKey, and fill in the Timetable array for any day(s) (with DayOfWeek = –1) or specific days (0 = Sunday, 1 = Monday, etc.), including start hour, minute, second, and an activity name. You can also select the new TimetableSource to either use the node’s own timetable data or the AI Controller’s “ControllerTimetable.” If you have a custom GameState that stores the current day/time, assign it in SelectedGameStateClass and specify the property names for day, hour, minute, and second; otherwise, the node falls back to internal defaults. When the Service runs, it updates your Blackboard with the appropriate timetable slot name (e.g. “Where2Go”) and a derived time string (e.g. “Time 09:51:07 Day 0”).

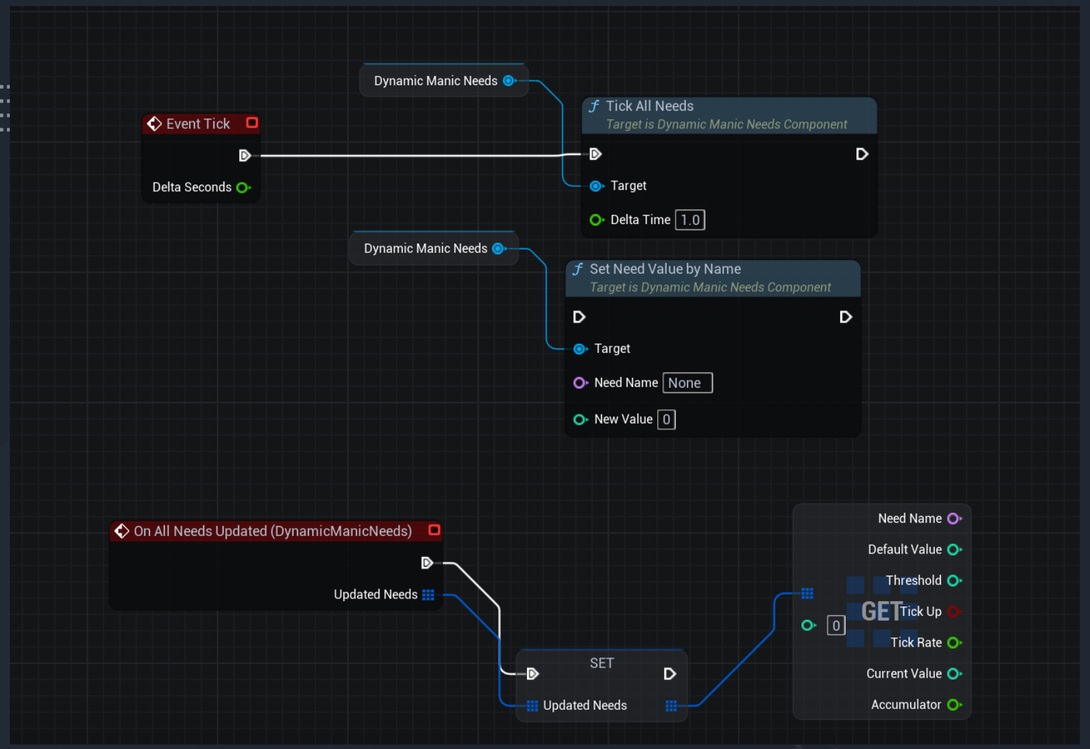


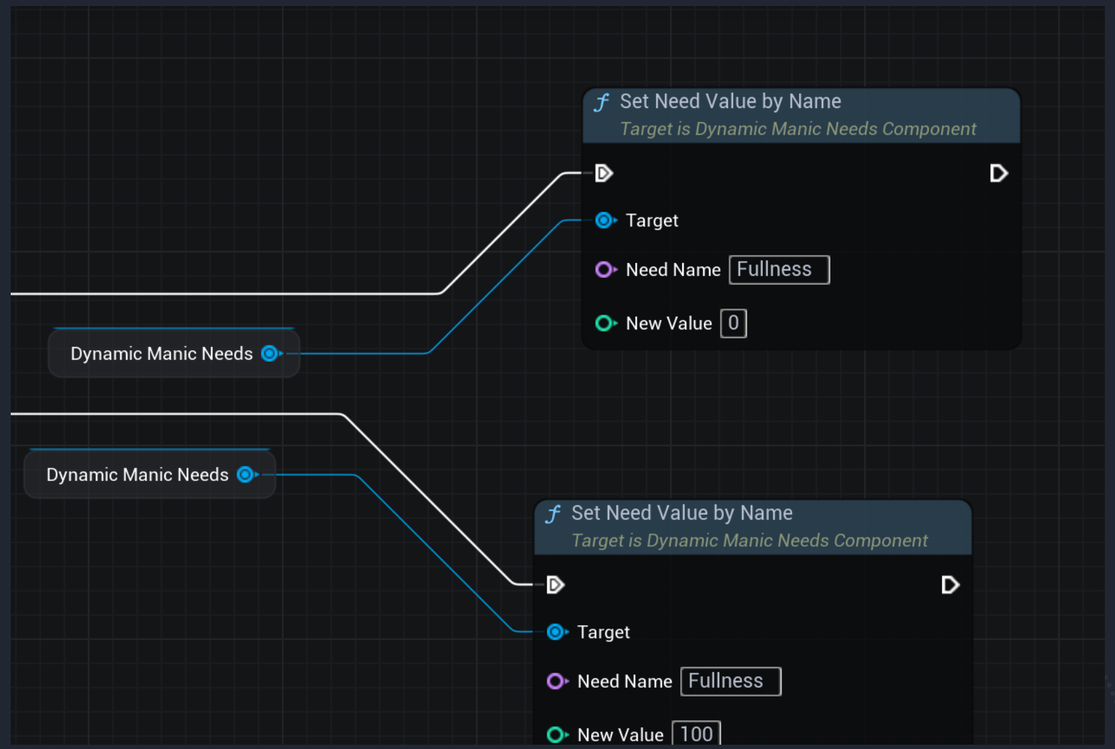




MANIC AI

The ManicServiceSetTimeKey node checks what day and time it is, decides which activity slot is active, and writes that activity's name to the Blackboard. By default, it uses an internal list of time slots (the "Node Timetable"). However, if you select "Use AI Controller Timetable" on the node, it will instead look at an array stored on the AI Controller, allowing each AI to have its own personalised schedule without needing separate Behaviour Trees or character changes.





Dynamic Manic Needs

MANIC AI

The Dynamic Needs system, provided by UDynamicManicNeedsComponent, tracks multiple “needs” (like hunger, thirst, or mood) that automatically change over time. Each need has a name, a default value, a threshold, and a rate at which it increases or decreases. When you call TickAllNeeds—typically once per frame—the component updates each need’s current value based on its rate, fires an event if a threshold is crossed, and optionally clamps the need value at that threshold. This makes it simple to trigger specific AI behaviour once a need reaches a certain level (e.g. “Hunger = 100”), and you can add, remove, or modify these needs at runtime to adapt AI behaviour dynamically.

MANIC AI

13

Obstacle-Aware: Move To Target

ManicObsAvoidance

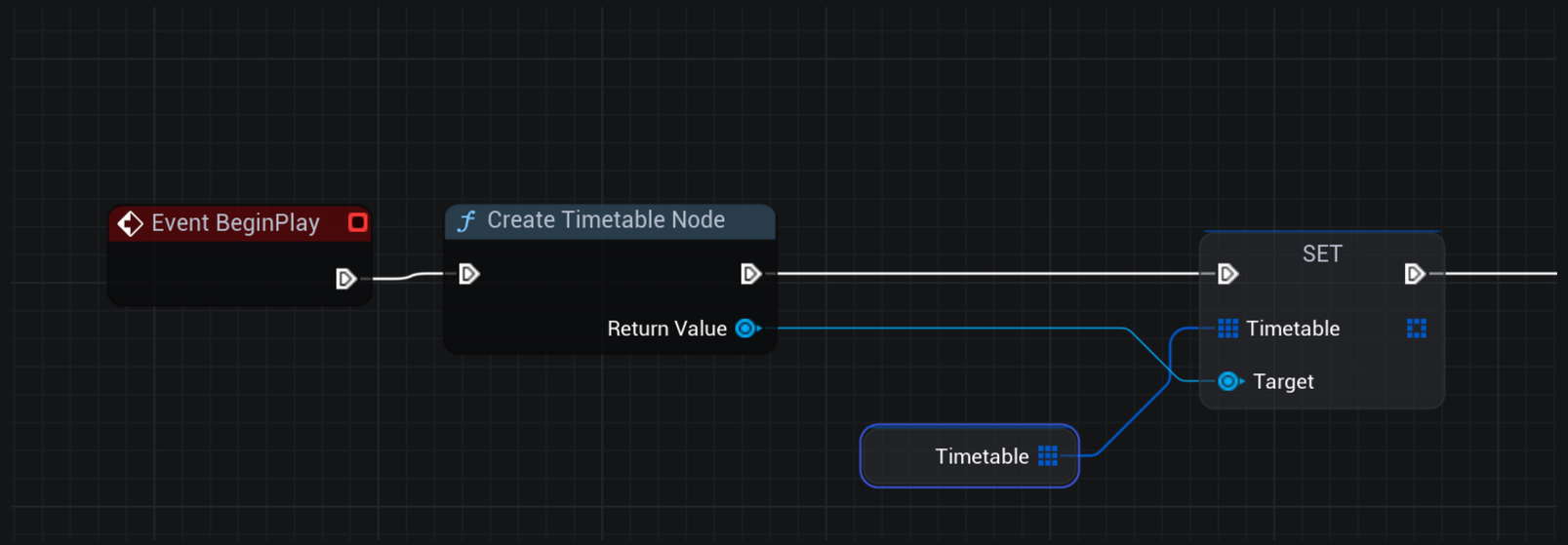
Details

Search

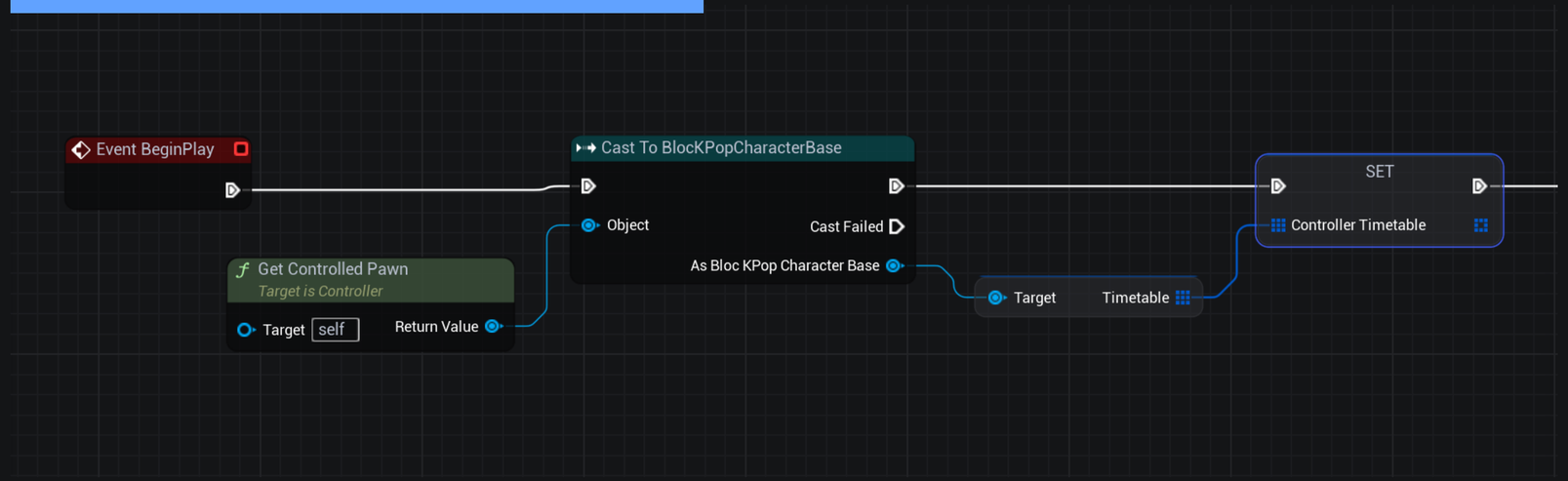
- Targeting
  - Target Type: Static Actor
  - Static Target Actor: Chair_Basic_Monday4
  - Acceptable Radius: 10.0
  - Inside Factor: 0.8
- Blackboard
  - Target Location Key: TargetLocation
  - Target Actor Key: TargetLocation
- Movement
  - Movement Speed: 600.0
- Avoidance
  - Obstacle Check Radius: 50.0
  - Minimum Clearance: 100.0
  - Max Adjustment Iterations: 5
- Debug
  - Draw Debug: ☑
  - Debug Sphere Radius: 10.0
  - Debug Draw Colour
  - Disable Obstacle Avoidance: ☐
- Stuck Detection
  - Min Move Distance: 10.0
  - Stuck Time Threshold: 1.0
  - Repel Offset: 30.0
- Task

This node intelligently moves an NPC to a chosen target (either a specific location, a target actor from the blackboard, or a static actor picked directly) by calculating an optimal destination that avoids obstacles along the way. It uses navmesh-based pathfinding with advanced repathing and even includes a “stuck” detection system that nudges the NPC if it gets trapped in tight corners or crowded situations. Additionally, it provides visual debug information so you can see the computed path and adjustments in real time, ensuring robust and smooth navigation in complex environments.

On the character (or anywhere)

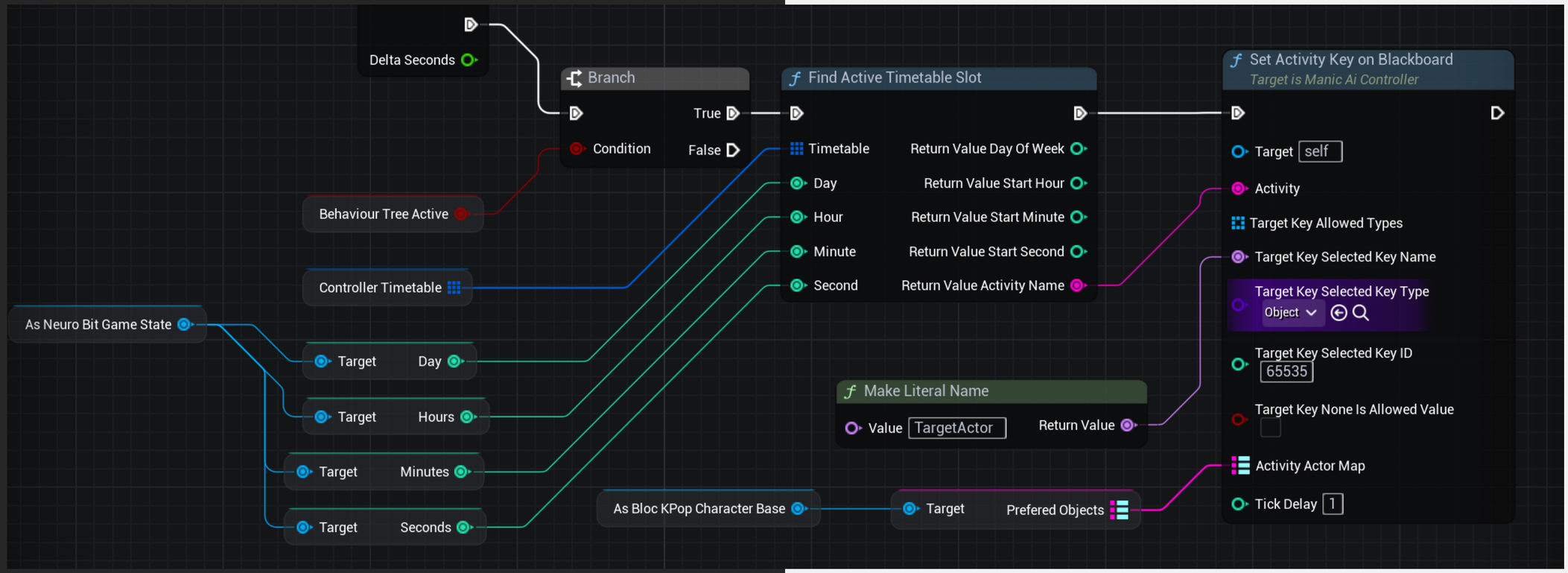


On the Ai Controler



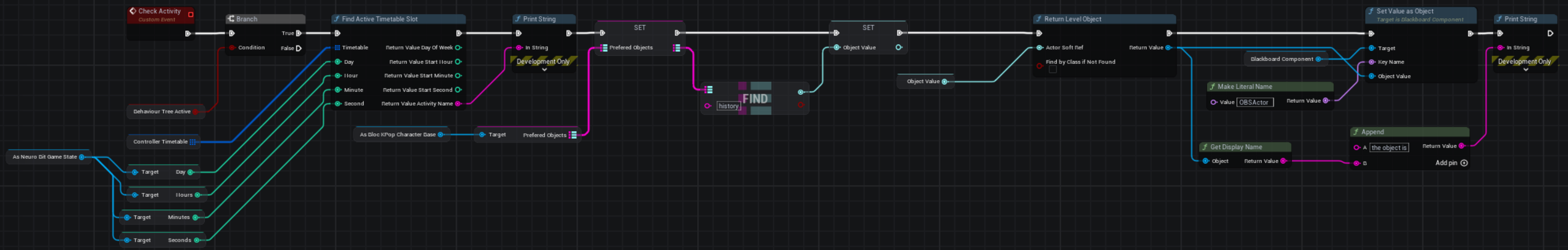
The **Create Timetable Node** function creates a new object that holds an editable list of time slots (FManicTime), which you can fill in Blueprint to define an AI character's schedule. Once created, you use the Get Timetable output from that node to set the Controller Timetable array on your AI Controller. When your Behaviour Tree runs, the ManicServiceSetTimeKey service checks which timetable source to use — if set to "Use AI Controller Timetable", it will read directly from this array and update the Blackboard with the correct activity based on in-game time.

MANIC AI



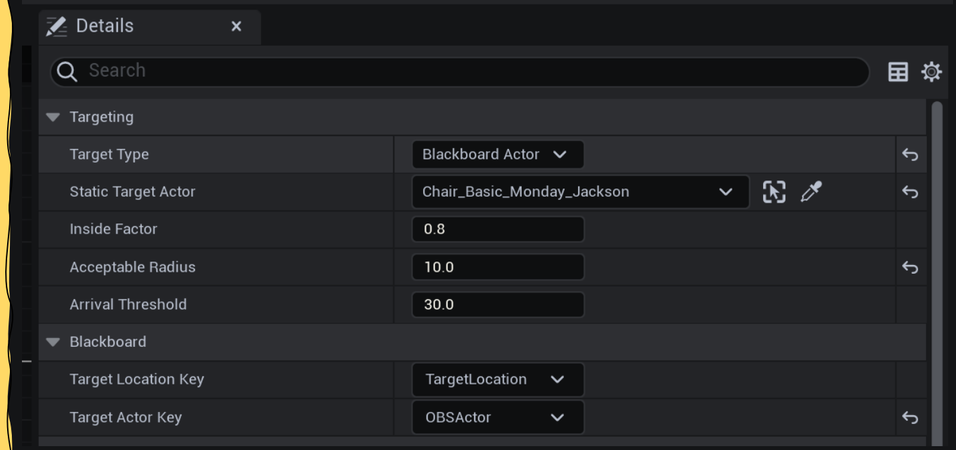
MANIC AI

In this Blueprint, the Event Tick retrieves the current time (day/hour/minute/second) from a custom Neuro Bit Game State and checks whether the AI's Behaviour Tree is active. If it is, the Blueprint calls Find Active Timetable Slot, passing in the AI's timetable and the current time. That function returns information about which activity slot is appropriate for that moment (including the activity's name, start hour, etc.). Next, the returned Activity Name is fed into Set Activity Key on Blackboard, a function on the AI Controller that looks up the corresponding actor (from the Activity Actor Map) and sets it in the Blackboard under a chosen key (here, something like TargetActor). The optional Tick Delay ensures you do not update the Blackboard every single tick, but rather once every N calls if needed. Essentially, this setup keeps the AI's Blackboard in sync with a real-time or accelerated schedule—letting the AI know which activity to pursue based on the in-game time.

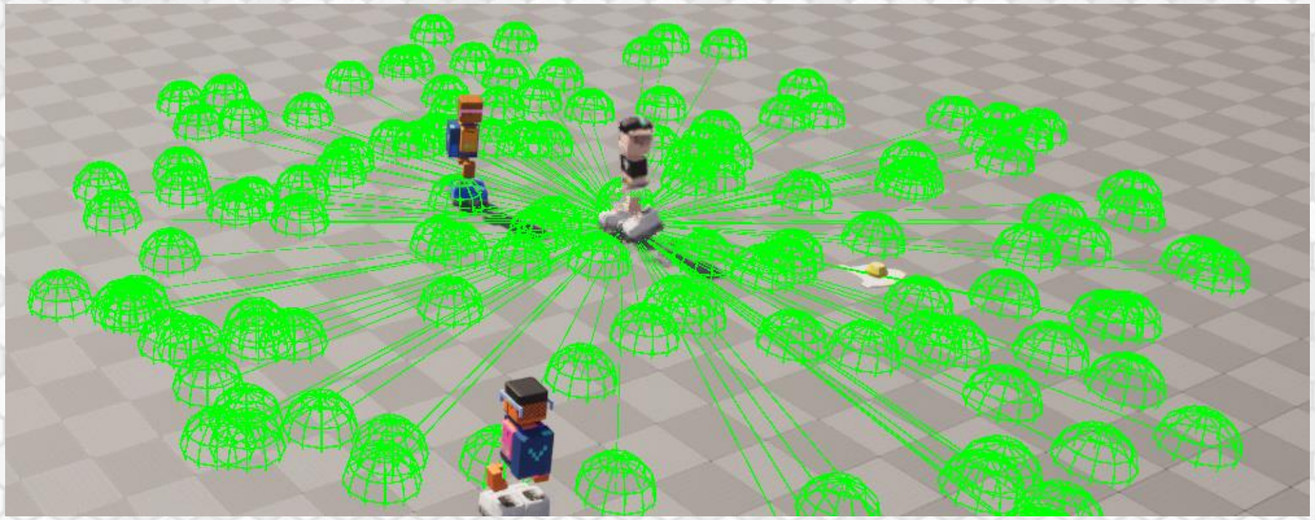


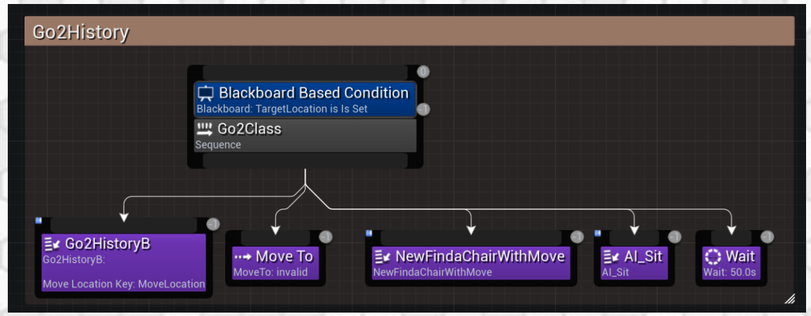
MANIC AI

You can store a map of string keys to soft actor references in your AI Character Blueprint (e.g. "History" -> TSoftObjectPtr of a specific chair). Then, at runtime, you retrieve one of those entries, feed it into the **Return Level Object** node (optionally enabling "Find By Class if Not Found"), and the node attempts to either load that soft-actor reference directly if it already exists in the level or, if that fails and the class fallback is allowed, it searches the world for any actor of that class and returns the first valid one. This way, you can easily define custom “preferred” objects (like special chairs, desks, etc.) for each AI or scenario without having to hard-reference them, and your AI can dynamically pick or move to those objects at runtime.



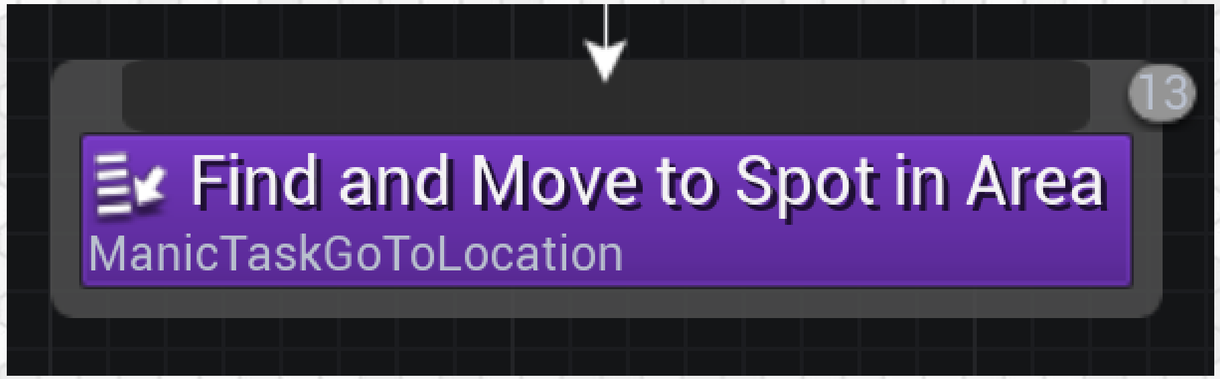
This node, "ManicTaskGoToLocation," enables NPCs to autonomously determine and navigate to a valid location within a specified area. Designers set the centre of the area, search radius, and any desired positional offset directly on the node. The node then finds a reachable point near that centre (or projects the centre onto the navmesh as a fallback) and commands the NPC to move there, monitoring its progress and reattempting if the destination isn’t reached in time. Optional debug visualisations assist in fine-tuning the area, ensuring reliable and flexible movement behaviour.

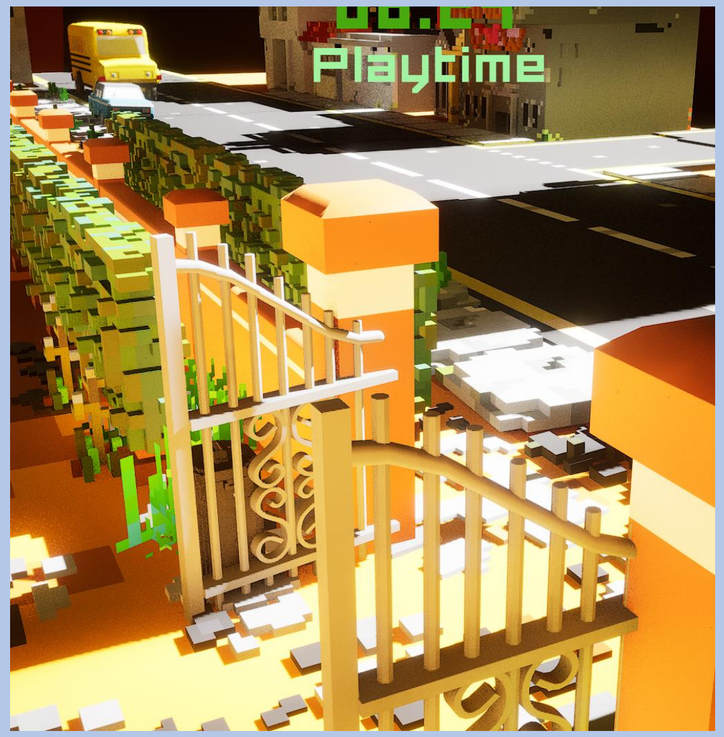


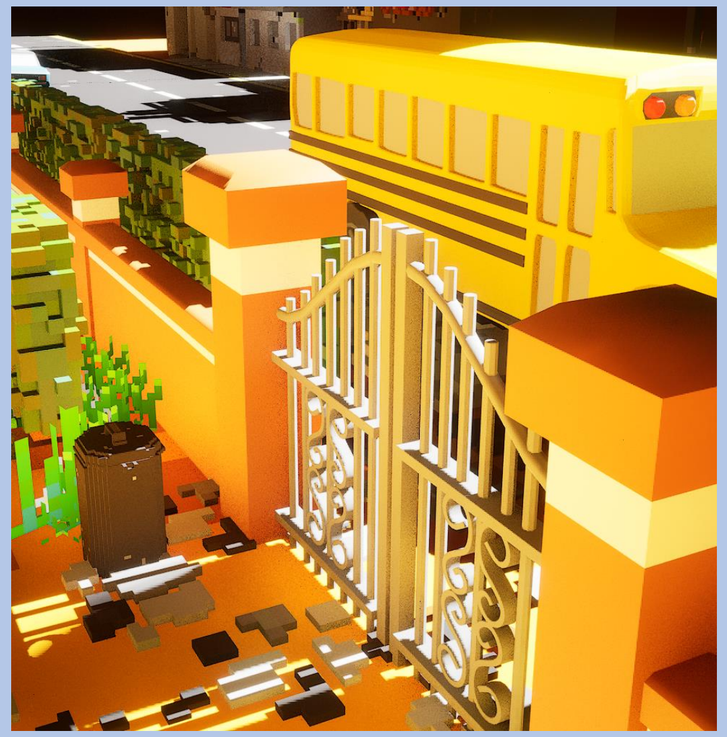


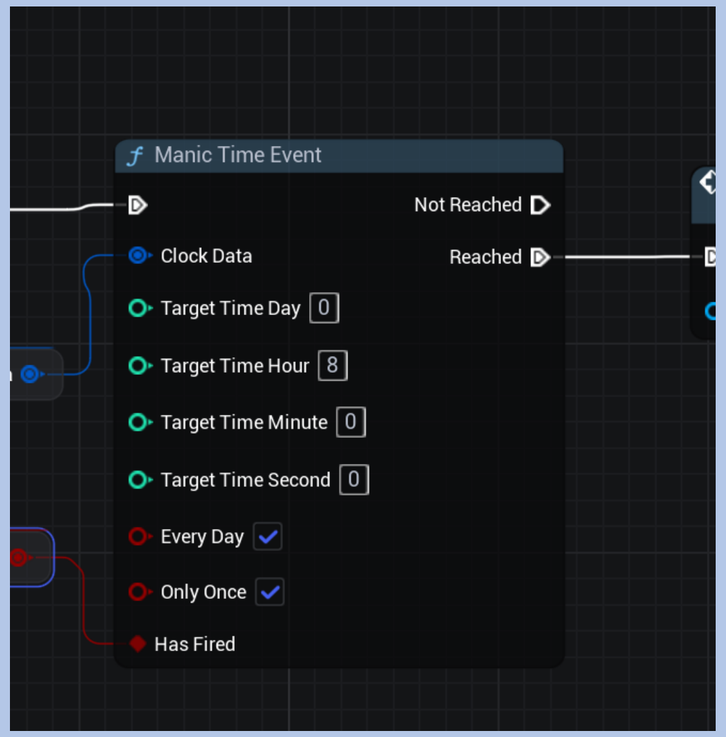

```
    if (bDrawDebugSphere)
    {
        DrawDebugSphere(World, FinalLocation, DebugSphereRadius,  Segments: 12, FColor::Green,  bPersistentLines: false,  LifeTime: 5.0f);
        DrawDebugLine(World,  LineStart: Center,  LineEnd: FinalLocation, FColor::Green,  bPersistentLines: false,  LifeTime: 5.0f,  DepthPriority: 0,  Thickness: 1.5f);
    }

    UE_LOG(LogTemp, Log, TEXT( InFormat: "ManicTaskGoToLocation: Found location %s within radius %.1f"), *FinalLocation.ToString(), SearchRadius);
    return EBTNodeResult::InProgress;
}
```









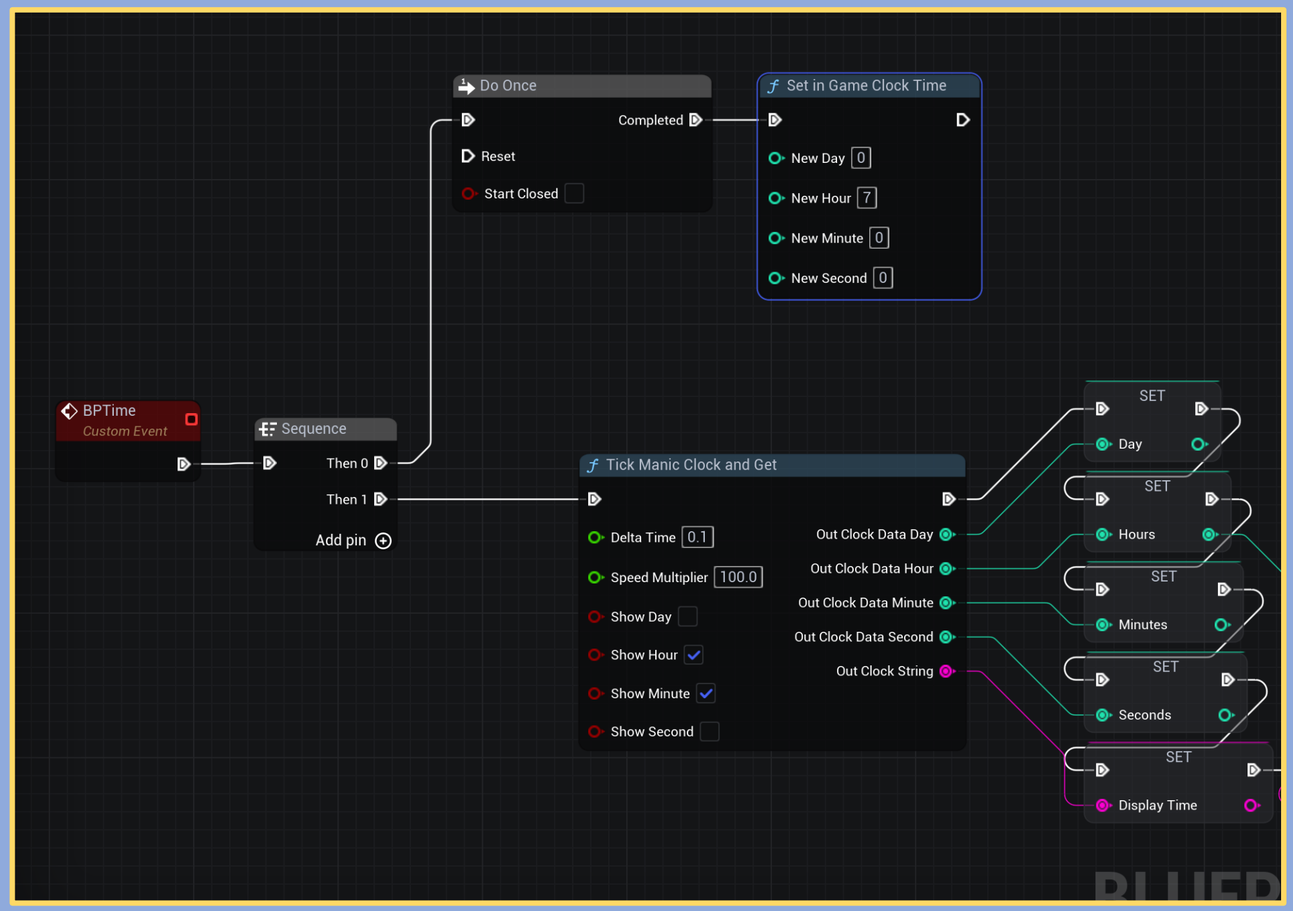
MANIC AI

Manic Time Event

Manic Time Event is a Blueprint node that checks if your current in-game clock (day/hour/minute/second) has reached or exceeded a specified target time. If it has, the node fires a **Reached** execution pin so you can run any event logic in the same Blueprint (e.g., open a gate). If it hasn’t, the **Not Reached** pin executes instead. You can also set it to only fire once until you reset a boolean variable, letting you control whether the event triggers repeatedly or just one time.

Manic Clock

Call **Tick Manic Clock And Get** every frame (for example, via **Event Tick**) to advance the clock by (**DeltaTime** × **SpeedMultiplier**). Optionally use **Set In Game Clock Time** beforehand to initialise the Day, Hour, Minute, and Second. In **Tick Manic Clock And Get**, link **Delta Seconds** into **DeltaTime** and choose your **Speed Multiplier** (e.g., 1.0 for normal speed or 60 for fast-forward). Toggle **Show Day**, **Show Hour**, **Show Minute**, or **Show Second** to decide which parts of the clock appear in the final text. The node outputs **Out Clock Data** (a struct with numerical values) plus **Out Clock String**, which you can display in your user interface.



MANIC AI

The **Find Active Timetable Slot** node checks the current in-game time against a list of time slots (a Timetable array made up of FManicTime structs) and returns the one that's currently active. It works just like the system used by your AI services: it looks at the current day, hour, minute, and second, and finds the most recent slot that has started (but not necessarily ended) and matches either today’s day or every day (DayOfWeek = -1). You can use this to show what activity is currently scheduled (e.g. “Sleeping”, “In Class”, “Lunch”) in a UI or to drive logic outside of Behaviour Trees.