Overview

CIVIL-AI-SYSTEM is designed to make adding complex behavior for AI entities as frictionless as possible allowing for quick iteration and development speed. This system was designed for RPG and city building settings but could be used in a host of other settings. With an easy to learn but in depth **Job System**, any user will be able to develop complex behaviors quickly and breath life into your scenes.

Getting Started

A quick run through of how to implement it in your own projects can be found here. If you want to get a better understanding it is recommended to start with **Civil System** and **Job System** as this will cover the core parts of the tool.

Where to buy

This product can be brought on the Unity Asset Store - link...

Where to report bugs & Ask for new Features

Bugs and Features can be requested at the following link

Roadmap

Coming Soon

How to get into the Project

1. Bring up the #Package Manager' in Unity (Window -> Package Manager - Found along thew top left top bar in Unity)
2. Search for **CIVIL-AI-SYSTEM** and select it from the list
3. Click 'Import' (Bottom right of the Package Manager window)

How to setup

Setup of the system is straight forward, using the **Manager**.

1. Within 'Import Job System Data' section have all tick boxes ticked and click 'Create and Add Assets to Scene'. This will add all the objects required in the scene to be added.
2. Then add in a **Region** using the 'Region Management' section (located a bit further down from the setup section), the text box to the left of 'Create Region' is for the name and is required, 'Create Region' will add it to the scene.
3. Add in your NPC character Game Object and click 'Setup object as NPC' and save this as a prefab and remove from scene.
4. Add the NPC prefab to the local **Region** Character Pool, if **Civil Jobs** is set to 'NONE' then it can be used for all **Jobs**
5. That's everything within the Scene setup which is unique to this product and ready to receive your defined behaviors

Simple Setup of Nav Mesh in Unity

1. Set objects which the AI should be able to walk on as 'static' (top right box in the 'Inspector')
2. Bring up the Nav Mesh window (Window -> AI -> CIVIL-AI-SYSTEM -> Navigation)
3. Select the 'Bake' tab
4. Press 'Bake' (This can take some time)
5. When on the Nav Mesh window tab you should see blue areas showing the areas where the **NPC's** can move

Get Started making AI behavior

Let's create our first **Job**, in this example we will make a Woodcutter **Job** (video:). This assumes you have Setup the system for the Scene.

1. Bring up the 'Flowchart Editor' (Found Window -> AI -> CIVIL-AI-SYSTEM -> Flowchart Editor)
2. Within this screen it allows us to design the behavior of our **NPC's**
3. Right click on the window space and select '**Job**', this will create a Job Node which has a number of characteristics
4. Change the 'Name'of the **Job** Node to 'Woodcutter'
5. Then create a second node called 'Collect Wood'
6. Connect these nodes using the 'link button' on the nodes (this can be done)
7. Create a **Task** called 'Gather from Tree' and link to the **Duty** 'Collect Wood'
8. Now we can create a **Method** and we will call it 'Find Tree to fall'
9. The next step is to use an **Action** to find the tree, create a new node **Action**, within action characteristic select 'LOCATE' and change the 'Item Needed' to 'TREE', tick 'Set Item in Use' along with ticking 'No Reset of Item in Use' and connect it to the 'Find a Tree to fall' Method
10. Create another action but this time create it with the action characteristic being 'IDLE' and the 'Set Item in Use' ticked
11. Now we need to add in a wait time and an animation for the felling of the Tree using the Requirement System, this is done by clicking on the link between the 'Find a Tree to fall' and the section Action 'Idle' there will be a circle, click on that and you'll get a new popup providing three options, 'Delete', 'Animation' and 'Wait Time', we want to add an animation in and give a wait time so we will update these fields
12. Now that this is all done Export the AI Flow and call it 'Woodcutter'
13. Go back to the **Manager** and change the 'Import Job System Data' selection to 'Woodcutter'
14. Run the scene and you'll see a character spawn and perform the actions that have been setup

Overview

CIVIL-AI-SYSTEM is designed to make adding complex behavior for AI entities as frictionless as possible allowing for quick iteration and development speed. This system was designed for RPG and city building settings but could be used in a host of other settings. With an easy to learn but in depth **Job System**, any user will be able to develop complex behaviors quickly and breath life into your scenes.

Getting Started

A quick run through of how to implement it in your own projects can be found here. If you want to get a better understanding it is recommended to start with **Civil System** and **Job System** as this will cover the core parts of the tool.

Where to buy

This product can be brought on the Unity Asset Store - link...

Where to report bugs & Ask for new Features

Bugs and Features can be requested at the following link

Roadmap

Coming Soon

Overview

The **Civil System** covers the day-to-day life of **NPC's** and population of civil areas. **NPC's** that are managed by the **Civil System** have their own **Work Controller** that takes input from the Civil **Job System** to assign **Jobs**, **Houses** and **Workplaces** to each **NPC**

spawned in by the local **Civil Manager**.

How it works

It works by importing a set of Civil Job System parameters (creation: Flowchart, Setup: Manager) via the **Global Civil Manager** using the **CivilJobData** class, this is a static class so only one set of Civil Job parameters can be loaded at a time.

Regions are then used for population of the sections of the world when enabled. The local **Civil Manager** if enabled will then find **Houses** and **Workplaces** within it's section and figure out the max population and spawn in that amount of NPC's.

How to setup

To setup the **Civil System** the world needs a **Region** added in with at least one **Workplace** and **House** (See Manager to import templates to your scene).

Objects

Houses

The house component is used to provide a home to an NPC, the maxResidents can be set in the editor view but the default is five. Entrance can be set up having a child GameObject called ‘entrance’ or manually typed in.

Workplaces

These places provide work for NPC's. Their Entrances work in the same way as House Entrances and are the first place the NPC will attempt to get to when work begins. Job Data is an array of JobData, each entity of the JobData has a job type(name of job in the flowchart), how many people can do this job for this workplace and how many people are doing the job. Assets are specific to the workplace and can be used when performing actions.

Workers are the NPC's who have been assigned to the workplace, this is used for requirements and to help with debugging.

Zones

These are a set size of '25' (can be changed in **Actions**) from the coordinates provided (can be changed in **Actions**)

Route

These are set node by node and the order is important.

Regions

These are used to define sections within your scenes for the AI System. These regions will be used to populate the location they are based in and all NPC's will be a child of it's Region. This allows for easy enabling and disabling of sections of your scenes which can increase performance. The region checks for Houses and Workplaces within it's area (defined by size) and uses the 'Civil Mananger' component to handle the spawning and management of the this area. Each Region can also have it's own character pool which allow for different models to be used for the characters.

Items

This is used to define what a **Resource** is used in finding objects within the world. It uses the enum **ITEMS** as well as a boolean to whether the item is **InUse** which is used to prevent NPC's from using the same objects.

CivilJobData

Used to import the data used in the Civil System on load of the scene (called from the **Global Civil System**), this is a singleton class (Meaning there is only ever one instance of it) which imports the **Job System** which contains the definitions of the **Jobs** and is used by the **Work Controller**

Animation Controller

The animation controller is used to animate the NPC within the scene.

To Replace

Required Parameters

- Moving : Boolean
- Dynamic : Boolean

Required Animation

- Dynamic

References

- AIController - animator.SetBool("Moving", moving);
- AIController - animator.SetBool("Dynamic", dynamic);

Game Manager / Timing

The Game Manager is a simple service which is used for the time of day and the passage of time. The time is 0-2400 and has to stay within this range to keep it in keeping with the requirements set in the Flowchart Editor. It updates by 'Time.deltaTime' each update call

To Replace

Required Functions

- getTime() : return float;

References

- AIDataboard has a static reference to the GameManager - Set by search in Awake()
- Civil Manager has a Reference which is used on AIDataBoard.setupCharacter() function Editor
- AIManager loads one from Resources when using Auto Setup scene (System/Objects/Game Manager)
- TimeWindow has a static reference to the gameManager

Item System

The Item system is used to allow the project to identify objects within the scene. This uses currently ITEMS (An enum for names) as well as the Item class

To Replace

Item

Required Functions

- IsInUse : Boolean
- SetInUse(bool update) : Boolean

References

Action Node

- ItemNeeded
- ItemOutput

AI Databoard

- currentGoalObj
- getGoalItem()
- setGoalItem()

Actions

- Locate()
- LocateOwned()
- LocateRandomOwned()
- LocateItemInZone()
- PickUp()
- Mount()

Looking Mutations

- GetWorkplaceObjects()
- GetItems()

Flowchart Connection

- RequirementGeneral

Requirement Widget (Editor)

- RequirementGeneral

Action Node (Editor)

- Action()
- Item Needed
- Item Output

Logging

This is logging that is done within the Unity Console

Start Up

When a scene starts the system will log how many of each type of the Job System was loaded into the Civil Manager

Failed Action or Method

When an action fails a Warning console log will be created, providing the Guid reference of the Action as well as the name and the item needed. This will then make the NPC try another method so a log will be made stating the Method failed providing the id and name if available.

Gizmos Colour Code

Gizmos are only shown on selection of the object and provide some useful information.

Houses

Desc	Colour
Entrance	Black

Workplaces

Desc	Colour
Entrance	Black

NPC

Desc	Colour
Home Entrance	Green
Workplace Entrance	Blue
Look At Location	Magenta
Way point path	Red
Method from Job System	N/A
Action from Job System	N/A

Frequently Known Issues

The Region can't find a Workplace nor House

A Workplace or House must be on layer 'Civil' and have a collider for it to be found by the AI system, this was implemented to increase performance by removing the need to check each GameObject to see if it has the component.

The Region has a House and Workplace but no NPC is created

A **Workplace** requires **Job** data to be added, this is done by selecting the **Workplace** and adding to the JobData field, this section has a Job type (This is the name of the **Job**) and can have more than one position. The **Houses** has a default of five max residents but this could have been changed so is worth checking.

The Region has a House with more than zero max residents and a Workplace with Jobs setup but no NPC is created

This is down to the local **Region** character pool not having a character which fits the requirements. This could be either from the character pool being empty or the job needs a unique one (e.g. it doesn't use the shared pool and need a specific one defined).

A Item exists within the scene but the NPC can't find it?

An **Item** must be on layer 'Resource' and have a collider for it to be found by the AI system, this was implemented to increase performance by removing the need to check each GameObject to see if it has the component. If it is a check for 'owned' **Items** then the workplace of the NPC needs a reference to it in **Assets**.

My NPC get's stuck trying to reach a location

A common cause of this is the node height being too far from the ground, this results in the NPC never getting close enough to trigger the action as completed. Reasoning Y-axis is taken into account is to allow for 3D movement.

What happens if a stage fails?

A breakdown of what happens is explained here

Basic Movement & Input

The flowchart editor can be controlled by keyboard and mouse.

Name	Mouse	Positive	Negative
Adding Node	Right Click		
Panning Vertical	Scroll Wheel	Down Arrow	Top Arrow
Panning Horizontal	N/A	Right Arrow	Left Arrow
Zooming	N/A	'+'	'-'

Toolbar

The toolbar is the top section of the flowchart editor (shown in red) which is always present. Going from left to right down. The first section is where the camera is located on the canvas of the flowchart graph, the Slider handles the zoom level of the canvas with a reset to the left. Then there is export and import used for saving and loading. Clear is used to reset the Flowchart editor.

Node Creation

The flowchart editor uses the Job System. These nodes can be added into the flowchart editor by right clicking within the canvas. This creates a context menu which provides a list of the nodes presented in the table above.

Node Interaction

The nodes can be moved by clicking and holding the tab with the node type name above the interaction buttons. * Links are explained in more detail in Links and Ordering. * Focus is explained in Focus System.
* Clear links can be used to remove all child connections to the node * Delete is used to delete the current node.

Links and Ordering

Links of nodes (shown in Green) can only be done from one level down from the selected node (E.g., Job can connect to Duty but not Task, Method can connect to Action but not Task). Links are connected by selecting the 'link' button on the node which you wish to connect (parent node) and clicking the 'link' button on the node which you wish to make a child of the parent node. The order of the child nodes has an impact on the order the children are executed in zero being first and one being next. Links can be deleted using the flowchart connection widget by selecting the row and then clicking the '-' icon (bottom section of node, does disappear when zoomed out). Order can also be rearranged drag and dropping.

Requirement System

To add a requirement (shown in yellow), you can click on the 'circle' on the connection line, if the circle is filled in then a requirement is in place, if the circle is not filled then there are no requirements. Deleting is done via the widget. The widget can be collapsed and folded out by clicking on the circle. The requirement system allows for criteria to be required before a node can be executed, the four top level nodes (Job – Method) all have the same requirements selection which can be found below.

Name	Impact	Default
Required time	If enabled this node will only be started within this time slot	Disabled – 0-2400 (full day)
Item	What Item is required to perform the node	NULL
Num of wrks doing	Check how many workers are doing the node and if met	Disabled – 0 > None (last box is the name of the nodes which are children to the parent node)

Actions requirements work slightly differently, these are more requirements by the system for the action, this being wait time and animation reference. Items needed and outputs are defined on the action node.

Focus System

The focus system (shown in blue) is used to allow the user to focus in on specific paths in the **flowchart editor**. This is accessed via the 'focus' button on the Node. When this is selected, all connected lower nodes are only shown. The focus groups are shown below the toolbar (shown in blue) and is referenced by the unique identifier of the top node. To show all nodes, select 'Main'. Focus groups can be deleted by selecting the 'x' to the right of the group reference. If a new node is added then it will show within the focused group. If focus group is changed the new unconnected node will not be shown except in the 'Main' focus or if it gets connected to the parent node of the focus group or lower.

How to get into the Project

1. Bring up the #Package Manager' in Unity (Window -> Package Manager - Found along thew top left top bar in Unity)
2. Search for **CIVIL-AI-SYSTEM** and select it from the list
3. Click 'Import' (Bottom right of the Package Manager window)

How to setup

Setup of the system is straight forward, using the **Manager**.

1. Within 'Import Job System Data' section have all tick boxes ticked and click 'Create and Add Assets to Scene'. This will add all the objects required in the scene to be added.
2. Then add in a **Region** using the 'Region Management' section (located a bit further down from the setup section), the text box to the left of 'Create Region' is for the name and is required, 'Create Region' will add it to the scene.
3. Add in your NPC character Game Object and click 'Setup object as NPC' and save this as a prefab and remove from scene.
4. Add the NPC prefab to the local **Region** Character Pool, if **Civil Jobs** is set to 'NONE' then it can be used for all **Jobs**
5. That's everything within the Scene setup which is unique to this product and ready to receive your defined behaviors

Simple Setup of Nav Mesh in Unity

1. Set objects which the AI should be able to walk on as 'static' (top right box in the 'Inspector')
2. Bring up the Nav Mesh window (Window -> AI -> CIVIL-AI-SYSTEM -> Navigation)
3. Select the 'Bake' tab
4. Press 'Bake' (This can take some time)
5. When on the Nav Mesh window tab you should see blue areas showing the areas where the **NPC's** can move

Get Started making AI behavior

Let's create our first **Job**, in this example we will make a Woodcutter **Job** (video:). This assumes you have Setup the system for the Scene.

1. Bring up the 'Flowchart Editor' (Found Window -> AI -> CIVIL-AI-SYSTEM -> Flowchart Editor)
2. Within this screen it allows us to design the behavior of our **NPC's**
3. Right click on the window space and select '**Job**', this will create a Job Node which has a number of characteristics
4. Change the 'Name'of the **Job** Node to 'Woodcutter'
5. Then create a second node called 'Collect Wood'
6. Connect these nodes using the 'link button' on the nodes (this can be done)
7. Create a **Task** called 'Gather from Tree' and link to the **Duty** 'Collect Wood'
8. Now we can create a **Method** and we will call it 'Find Tree to fall'
9. The next step is to use an **Action** to find the tree, create a new node **Action**, within action characteristic select 'LOCATE' and change the 'Item Needed' to 'TREE', tick 'Set Item in Use' along with ticking 'No Reset of Item in Use' and connect it to the 'Find a Tree to fall' Method
10. Create another action but this time create it with the action characteristic being 'IDLE' and the 'Set Item in Use' ticked
11. Now we need to add in a wait time and an animation for the felling of the Tree using the Requirement System, this is done by clicking on the link between the 'Find a Tree to fall' and the section Action 'Idle' there will be a circle, click on that and you'll get a new popup providing three options, 'Delete', 'Animation' and 'Wait Time', we want to add an animation in and give a wait time so we will update these fields
12. Now that this is all done Export the AI Flow and call it 'Woodcutter'
13. Go back to the **Manager** and change the 'Import Job System Data' selection to 'Woodcutter'
14. Run the scene and you'll see a character spawn and perform the actions that have been setup

Overview

The Job System is an adaptable system which allows for visual scripting of AI behavior tailored towards presenting real world day-to-day tasks simply.

How to Create

Creation is done using the Flowchart Editor which is a visual node based system of creating the data required. This can then be imported in by using the Manager.

Objects

Job

The overall name of the occupation the NPC can do. It contains the start and end time of the **Job** as well as whether the NPC character uses the **global pool**

Duty

A high-level concept of what a **Job** requires the NPC to do made up of **Tasks**

Task

A goal oriented part of a **Duty** which is used to define more complex behavior

Method

A way of achieving a goal, having more than one of these allow for different ways of achieving the same results. This level allows for added robustness in the system.

Action

A predefined explicit set of functions that make the NPC perform **actions** in the scene (e.g., find something, go somewhere, pick up **Item** etc)

How these are applied

The **Job** is applied on creation of the NPC and this does not change.

Then a **Duty** which requirements are met (each is checked from 0 up) is then set. After that the same approach is used for setting the **Task**, **Method** and **Action**. When an **Action** is successful then the **Action Requirement** is applied (wait time and animation). After the **Action Requirement** is done the next **Action** is selected and this continues like so until all are completed and then the next **Task** is started (**Method** is skipped as each **Method** for a **Task** should fulfill the same goal) with the **action** to perform being reset to 0 . The same takes place with **Duties**. If all **Duties** are done then the process starts again from 0.

However on a **Action** failing this will cause the **Method** to fail too. When this happens the next **Method** will be attempted and the NPC will keep trying until either a **Method** is successful or all **Methods** fail. When all **Methods** then the remaining **Tasks** for this **Duty** will be skipped and the next **Duty** will be attempted.

When are Requirements Applied

Requirements for the middle three layers (**Duty**, **Task** and **Method**) are only applied on entering of that node. This means if a **Duty** can only be performed before '1700' and the **Duty** was started before that time but still has **Tasks** left after that time then the remaining **Tasks** will be completed.

With **Action** the Requirements are performed after the **Action** has been successful and holds the return to the **Work Controller** off from being successful until it has been completed.

Predefined Actions

Overview

Predefined actions also refereed to as **actions** are functions which define how the interaction takes place within the Scene. They return a nullable boolean value to show the three possible states they can be in; successfully completed, still taking place and failed. This is then used by the **Work Controller** to select the next behavior.

List of Predefined Actions

- IDLE
- MOUNT
- DISMOUNT
- AWAIT FOR MOUNT FILLED
- LOCATE
- LOCATE OWNED
- LOCATE RANDOM OWNED
- PICK UP
- DROP OFF
- FOLLOW ROUTE
- FOLLOW ROUTE REVERSE
- LOCATE IN ZONE 1
- LOCATE IN ZONE 2
- RANDOM LOCATION IN ZONE 1
- RANDOM LOCATION IN ZONE 2

Locating logic

The location logic is mostly the same for all three functions but has some differences. The main method in which they work is they try to find **Item**'s within '25' (default, can be changed in **Actions**, if owned it only uses the list of the workplaces and doesn't care about distance) of the NPC location of the specific type. The list is then used to figure out which one the NPC should head to dependent on the function, if it is not specified in the name then it'll be the closest that is used. With Zones it is the closest to the AI which is within the Zone.

Pick Up & Drop Off

This allows for the NPC to pick up, carry objects objects around and place down elsewhere. This works by looking for the closest not **InUse** item which is within '5' (can be changed in **actions**). It is then made a child of the **GameObject** which is either the object which has been assigned to the right or left hand (Set in the **AI Databoard**) and gravity is turned off. The small distance means it is recommend to use a 'LOCATE' action before doing calling the 'PICK UP'. On 'Drop off' the item parent is set to null and gravity is turned on.

Mount Actions

The mount actions allow for the NPC's to be able to use different means of transportation. It works by using the generic **Mount** class which defines where the NPC's use the **Mount** from. The first one added is the one in charge of controlling the **Mount**. It also contains how many are using it currently and the maximum amount.

Mount works by checking if the NPC is within '5' and if so it'll make sure the Mount is not full. If it isn't full then the NPC is teleported to an unused control point. The Mount boolean for being controlled is set to true and a reference is added to the mount for the current NPC

Waiting works by not completing the **Action** until the mount return true from the 'IsFull' check

Dismount works by cleaning up the Mount (remove the user reference) and cleaning the NPC databoard up by setting back the local settings mounted and inControlOfMount

What happens on failures

Actions

If an action fails then the rest of the actions belonging to the method will be skipped and the method will automatically be failed.

Methods

If a method fails (caused by an action within it failing) and there is another method belonging to the current task then the next method will be tried until successful or all methods have failed.

Tasks

A task will only fail when all children methods of this task fails. This results in the duty failing

Duties

If a duty fails then the system will move onto the next duty, when all duties are done the process is started again from the start.

What is this for?

The Manager allows for quick setup of the Civil System within a scene, along with management of regions. It also handles the importing of the Job System

Actions

Import Node Group

This allows for the importing of Job System data into the correct file to allow for the Civil System to pick up at run time. The list provides all the made groups and applies across all projects (note: change this before switching projects if they require different Job System data).

When a new Job System is imported it moves the collections across into the root Resource file in the AI folder and creates and replace the **CIVIL_JOBS** enum (Found 'AI/Script/Enums').

Setup AI in Scene

The setup tool is used to setup a full fresh scene with the system. There are a few options which can be used to customize the import. The main thing it does is import the **Global Civil Manager** and can provide templates for setting up NPC's, Houses and Workplaces. This does still require at least Region and at least one NPC (Setup: *NPC*) added to that Region character pool.

NPC Template

Used to give a base template for setting up a NPC, simple setup should only require changing the me

Region Management

This is used to create local Region management object. Some small changes can be done from here, such as location and size of the area. **Focus** can be used to find the object within the scene and selects the object.

Overview

NPC's are the agents that move around the world. NPC standing for Non Playable Character.

Setup

A NPC when being created needs to have an **Animator**, **AI Controller**, **AI Databoard** and **Nav Mesh Agent**. This can simply be

setup by selecting the option 'Setup object as NPC' in Manager and the **GameObject** you wish to setup and all components will be added. Then save as a 'prefab' and add to the character pool in Region

Required Components

Animator

The animator is required for animations. The main part this system is interested in is the 'animator controller' as it requires a 'Moving' and 'Dynamic' boolean to be present as well as a 'Motion' called 'Dynamic' to be present and played when the 'Dynamic' parameter is true. A basic 'animator controller' can be found within the 'Animation' folder

AI Controller

The AI controller is the low level control of the NPC. From within here the 'animator controller' settings are set, destinations and pointing out is also controlled. It also handles the interface between the Mount interface.

AI Databoard

This section provides the core data of the NPC and provide a common interface between all systems, this includes states, timers, specific information for this NPC (Where is home, current goal, mounted, if at work etc.) along with access to the inventory.

Nav Mesh Agent

This is the built in Unity feature which provides pathfinding and is used by default with this project. For quick setup see Getting Started

Work Controller (Added via Civil Manager)

The work controller is an optional component which is added at run time if this NPC has a Job. This interfaces with the Civil Job System to provide the character with Job information (What they do, where it is) and holds the logic on how the system progresses as well as updating work states (e.g. setting when they are 'at work')

Overview

The Job System is an adaptable system which allows for visual scripting of AI behavior tailored towards presenting real world day-to-day tasks simply.

How to Create

Creation is done using the Flowchart Editor which is a visual node based system of creating the data required. This can then be imported in by using the Manager.

Objects

Job

The overall name of the occupation the NPC can do. It contains the start and end time of the **Job** as well as whether the NPC character uses the **global pool**

Duty

A high-level concept of what a **Job** requires the NPC to do made up of **Tasks**

Task

A goal oriented part of a **Duty** which is used to define more complex behavior

Method

A way of achieving a goal, having more than one of these allow for different ways of achieving the same results. This level allows for added robustness in the system.

Action

A predefined explicit set of functions that make the NPC perform **actions** in the scene (e.g., find something, go somewhere, pick up **Item** etc)

How these are applied

The **Job** is applied on creation of the NPC and this does not change.

Then a **Duty** which requirements are met (each is checked from 0 up) is then set. After that the same approach is used for setting the **Task**, **Method** and **Action**. When an **Action** is successful then the **Action Requirement** is applied (wait time and animation). After the **Action Requirement** is done the next **Action** is selected and this continues like so until all are completed and then the next **Task** is started (**Method** is skipped as each **Method** for a **Task** should fulfill the same goal) with the **action** to perform being reset to 0 . The same takes place with **Duties**. If all **Duties** are done then the process starts again from 0.

However on a **Action** failing this will cause the **Method** to fail too. When this happens the next **Method** will be attempted and the NPC will keep trying until either a **Method** is successful or all **Methods** fail. When all **Methods** then the remaining **Tasks** for this **Duty** will be skipped and the next **Duty** will be attempted.

When are Requirements Applied

Requirements for the middle three layers (**Duty**, **Task** and **Method**) are only applied on entering of that node. This means if a **Duty** can only be performed before '1700' and the **Duty** was started before that time but still has **Tasks** left after that time then the remaining **Tasks** will be completed.

With **Action** the Requirements are performed after the **Action** has been successful and holds the return to the **Work Controller** off from being successful until it has been completed.

Predefined Actions

Overview

Predefined actions also refereed to as **actions** are functions which define how the interaction takes place within the Scene. They return a nullable boolean value to show the three possible states they can be in; successfully completed, still taking place and failed. This is then used by the **Work Controller** to select the next behavior.

List of Predefined Actions

- IDLE
- MOUNT
- DISMOUNT

- AWAIT FOR MOUNT FILLED
- LOCATE
- LOCATE OWNED
- LOCATE RANDOM OWNED
- PICK UP
- DROP OFF
- FOLLOW ROUTE
- FOLLOW ROUTE REVERSE
- LOCATE IN ZONE 1
- LOCATE IN ZONE 2
- RANDOM LOCATION IN ZONE 1
- RANDOM LOCATION IN ZONE 2

Locating logic

The location logic is mostly the same for all three functions but has some differences. The main method in which they work is they try to find **Item**'s within '25' (default, can be changed in **Actions**, if owned it only uses the list of the workplaces and doesn't care about distance) of the NPC location of the specific type. The list is then used to figure out which one the NPC should head to dependent on the function, if it is not specified in the name then it'll be the closest that is used. With Zones it is the closest to the AI which is within the Zone.

Pick Up & Drop Off

This allows for the NPC to pick up, carry objects objects around and place down elsewhere. This works by looking for the closest not **InUse** item which is within '5' (can be changed in **actions**). It is then made a child of the **GameObject** which is either the object which has been assigned to the right or left hand (Set in the **AI Databoard**) and gravity is turned off. The small distance means it is recommend to use a 'LOCATE' action before doing calling the 'PICK UP'. On 'Drop off' the item parent is set to null and gravity is turned on.

Mount Actions

The mount actions allow for the NPC's to be able to use different means of transportation. It works by using the generic **Mount** class which defines where the NPC's use the **Mount** from. The first one added is the one in charge of controlling the **Mount**. It also contains how many are using it currently and the maximum amount.

Mount works by checking if the NPC is within '5' and if so it'll make sure the Mount is not full. If it isn't full then the NPC is teleported to an unused control point. The Mount boolean for being controlled is set to true and a reference is added to the mount for the current NPC

Waiting works by not completing the **Action** until the mount return true from the 'IsFull' check

Dismount works by cleaning up the Mount (remove the user reference) and cleaning the NPC databoard up by setting back the local settings mounted and inControlOfMount

What happens on failures

Actions

If an action fails then the rest of the actions belonging to the method will be skipped and the method will automatically be failed.

Methods

If a method fails (caused by an action within it failing) and there is another method belonging to the current task then the next method will be tried until successful or all methods have failed.

Tasks

A task will only fail when all children methods of this task fails. This results in the duty failing

Duties

If a duty fails then the system will move onto the next duty, when all duties are done the process is started again from the start.

Basic Movement & Input

The flowchart editor can be controlled by keyboard and mouse.

Name	Mouse	Positive	Negative
Adding Node	Right Click		
Panning Vertical	Scroll Wheel	Down Arrow	Top Arrow
Panning Horizontal	N/A	Right Arrow	Left Arrow
Zooming	N/A	'+'	'-'

Toolbar

The toolbar is the top section of the flowchart editor (shown in red) which is always present. Going from left to right down. The first section is where the camera is located on the canvas of the flowchart graph, the Slider handles the zoom level of the canvas with a reset to the left. Then there is export and import used for saving and loading. Clear is used to reset the Flowchart editor.

Node Creation

The flowchart editor uses the Job System. These nodes can be added into the flowchart editor by right clicking within the canvas. This creates a context menu which provides a list of the nodes presented in the table above.

Node Interaction

The nodes can be moved by clicking and holding the tab with the node type name above the interaction buttons. * Links are explained in more detail in Links and Ordering. * Focus is explained in Focus System.
* Clear links can be used to remove all child connections to the node * Delete is used to delete the current node.

Links and Ordering

Links of nodes (shown in Green) can only be done from one level down from the selected node (E.g., Job can connect to Duty but not Task, Method can connect to Action but not Task). Links are connected by selecting the 'link' button on the node which you wish to connect (parent node) and clicking the 'link' button on the node which you wish to make a child of the parent node. The order of the child nodes has an impact on the order the children are executed in zero being first and one being next. Links can be deleted using the flowchart connection widget by selecting the row and then clicking the '-' icon (bottom section of node, does disappear when zoomed out). Order can also be rearranged drag and dropping.

Requirement System

To add a requirement (shown in yellow), you can click on the 'circle' on the connection line, if the circle is filled in then a requirement is in place, if the circle is not filled then there are no requirements. Deleting is done via the widget. The widget can be collapsed and folded out by clicking on the circle. The requirement system allows for criteria to be required before a node can be

executed, the four top level nodes (Job – Method) all have the same requirements selection which can be found below.

Name	Impact	Default
Required time	If enabled this node will only be started within this time slot	Disabled – 0-2400 (full day)
Item	What Item is required to perform the node	NULL
Num of wrks doing	Check how many workers are doing the node and if met	Disabled – 0 > None (last box is the name of the nodes which are children to the parent node)

Actions requirements work slightly differently, these are more requirements by the system for the action, this being wait time and animation reference. Items needed and outputs are defined on the action node.

Focus System

The focus system (shown in blue) is used to allow the user to focus in on specific paths in the **flowchart editor**. This is accessed via the 'focus' button on the Node. When this is selected, all connected lower nodes are only shown. The focus groups are shown below the toolbar (shown in blue) and is referenced by the unique identifier of the top node. To show all nodes, select 'Main'. Focus groups can be deleted by selecting the 'x' to the right of the group reference. If a new node is added then it will show within the focused group. If focus group is changed the new unconnected node will not be shown except in the 'Main' focus or if it gets connected to the parent node of the focus group or lower.

What is this for?

The Manager allows for quick setup of the Civil System within a scene, along with management of regions. It also handles the importing of the Job System

Actions

Import Node Group

This allows for the importing of Job System data into the correct file to allow for the Civil System to pick up at run time. The list provides all the made groups and applies across all projects (note: change this before switching projects if they require different Job System data).

When a new Job System is imported it moves the collections across into the root Resource file in the AI folder and creates and replace the **CIVIL_JOBS** enum (Found 'AI/Script/Enums').

Setup AI in Scene

The setup tool is used to setup a full fresh scene with the system. There are a few options which can be used to customize the import. The main thing it does is import the **Global Civil Manager** and can provide templates for setting up NPC's, Houses and Workplaces. This does still require at least Region and at least one NPC (Setup: *NPC*) added to that Region character pool.

NPC Template

Used to give a base template for setting up a NPC, simple setup should only require changing the me

Region Management

This is used to create local Region management object. Some small changes can be done from here, such as location and size of the area. **Focus** can be used to find the object within the scene and selects the object.

Overview

NPC's are the agents that move around the world. NPC standing for Non Playable Character.

Setup

A NPC when being created needs to have an **Animator**, **AI Controller**, **AI Databoard** and **Nav Mesh Agent**. This can simply be setup by selecting the option 'Setup object as NPC' in Manager and the **GameObject** you wish to setup and all components will be added. Then save as a 'prefab' and add to the character pool in Region

Required Components

Animator

The animator is required for animations. The main part this system is interested in is the 'animator controller' as it requires a 'Moving' and 'Dynamic' boolean to be present as well as a 'Motion' called 'Dynamic' to be present and played when the 'Dynamic' parameter is true. A basic 'animator controller' can be found within the 'Animation' folder

AI Controller

The AI controller is the low level control of the NPC. From within here the 'animator controller' settings are set, destinations and pointing out is also controlled. It also handles the interface between the Mount interface.

AI Databoard

This section provides the core data of the NPC and provide a common interface between all systems, this includes states, timers, specific information for this NPC (Where is home, current goal, mounted, if at work etc.) along with access to the inventory.

Nav Mesh Agent

This is the built in Unity feature which provides pathfinding and is used by default with this project. For quick setup see Getting Started

Work Controller (Added via Civil Manager)

The work controller is an optional component which is added at run time if this NPC has a Job. This interfaces with the Civil Job System to provide the character with Job information (What they do, where it is) and holds the logic on how the system progresses as well as updating work states (e.g. setting when they are 'at work')

Logging

This is logging that is done within the Unity Console

Start Up

When a scene starts the system will log how many of each type of the Job System was loaded into the Civil Manager

Failed Action or Method

When an action fails a Warning console log will be created, providing the Guid reference of the Action as well as the name and the item needed. This will then make the NPC try another method so a log will be made stating the Method failed providing the id and name if available.

Gizmos Colour Code

Gizmos are only shown on selection of the object and provide some useful information.

Houses

Desc	Colour
Entrance	Black

Workplaces

Desc	Colour
Entrance	Black

NPC

Desc	Colour
Home Entrance	Green
Workplace Entrance	Blue
Look At Location	Magenta
Way point path	Red
Method from Job System	N/A
Action from Job System	N/A

Frequently Known Issues

The Region can't find a Workplace nor House

A Workplace or House must be on layer 'Civil' and have a collider for it to be found by the AI system, this was implemented to increase performance by removing the need to check each GameObject to see if it has the component.

The Region has a House and Workplace but no NPC is created

A **Workplace** requires **Job** data to be added, this is done by selecting the **Workplace** and adding to the JobData field, this section has a Job type (This is the name of the **Job**) and can have more than one position. The **Houses** has a default of five max residents but this could have been changed so is worth checking.

The Region has a House with more than zero max residents and a Workplace with Jobs setup but no NPC is created

This is down to the local **Region** character pool not having a character which fits the requirements. This could be either from the character pool being empty or the job needs a unique one (e.g. it doesn't use the shared pool and need a specific one defined).

A Item exists within the scene but the NPC can't find it?

An **Item** must be on layer 'Resource' and have a collider for it to be found by the AI system, this was implemented to increase performance by removing the need to check each GameObject to see if it has the component. If it is a check for 'owned' **Items** then the workplace of the NPC needs a reference to it in **Assets**.

My NPC get's stuck trying to reach a location

A common cause of this is the node height being too far from the ground, this results in the NPC never getting close enough to trigger the action as completed. Reasoning Y-axis is taken into account is to allow for 3D movement.

What happens if a stage fails?

A breakdown of what happens is explained here

Animation Controller

The animation controller is used to animate the NPC within the scene.

To Replace

Required Parameters

- Moving : Boolean
- Dynamic : Boolean

Required Animation

- Dynamic

References

- AIController - animator.SetBool("Moving", moving);
- AIController - animator.SetBool("Dynamic", dynamic);

Game Manager / Timing

The Game Manager is a simple service which is used for the time of day and the passage of time. The time is 0-2400 and has to stay within this range to keep it in keeping with the requirements set in the Flowchart Editor. It updates by 'Time.deltaTime' each

update call

To Replace

Required Functions

- getTime() : return float;

References

- AIDataboard has a static reference to the GameManager - Set by search in Awake()
- Civil Manager has a Reference which is used on AIDataBoard.setupCharacter() function Editor
- AIManager loads one from Resources when using Auto Setup scene (System/Objects/Game Manager)
- TimeWindow has a static reference to the gameManager

Item System

The Item system is used to allow the project to identify objects within the scene. This uses currently ITEMS (An enum for names) as well as the Item class

To Replace

Item

Required Functions

- IsInUse : Boolean
- SetInUse(bool update) : Boolean

References

Action Node

- ItemNeeded
- ItemOutput

AI Databoard

- currentGoalObj
- getGoalItem()
- setGoalItem()

Actions

- Locate()
- LocateOwned()
- LocateRandomOwned()
- LocateItemInZone()
- PickUp()
- Mount()

Looking Mutations

- GetWorkplaceObjects()
- GetItems()

Flowchart Connection

- RequirementGeneral

Requirement Widget (Editor)

- RequirementGeneral

Action Node (Editor)

- Action()
- Item Needed
- Item Output