Wizards Code Character System

The Wizards Code character system provides an ambient AI system for your game characters. Using this system your characters will come to life. This document will get you going quickly in the Getting Started section. It also has sections that dive deeper into the internals of the system.

For more practical guidance on how to develop AI see the demo scenes in `Characters/Scenes`. Each subfolder within this section is designed to demonstrate a new topic. Within each subfolder there are a set of number scenes. Each one building on the previous, so start at the 101 scene and work through. When run each scene will display a summary of the new concepts introduced in that scene.

Contents

[Getting Started 3](#_Toc106567731)

[Core Concepts 5](#_Toc106567732)

[Behaviours 6](#_Toc106567733)

[Adding a Wander Behaviour 6](#_Toc106567734)

[Deciding which Behaviour To Enact 6](#_Toc106567735)

[Desired States 9](#_Toc106567736)

[Getting Tired and Resting 11](#_Toc106567737)

[Debugging UI 13](#_Toc106567738)

[Debugging the Rest Behaviour 13](#_Toc106567739)

[Interactables 15](#_Toc106567740)

[Actor Cues 16](#_Toc106567741)

[Anatomy of a Cue 16](#_Toc106567742)

[Memory 17](#_Toc106567743)

[Example: Shooter Combat 18](#_Toc106567744)

[Configure the Scene 19](#_Toc106567745)

[Create the Actor Cues 20](#_Toc106567746)

[Configure the AI 21](#_Toc106567747)

[Example: Populating a World 22](#_Toc106567748)

[Create the World 23](#_Toc106567749)

[Create Our First Character 24](#_Toc106567750)

[Harvesting Crops 25](#_Toc106567751)

[Work and Rest Cycle 28](#_Toc106567752)

[Desired State – Sufficiently Rested 28](#_Toc106567753)

[Behaviour – Sit and Rest 28](#_Toc106567754)

[Behaviour - Resting when Idle 29](#_Toc106567755)

[Gathering Edible Food 31](#_Toc106567756)

[Desired State – Carrying Enough Food 31](#_Toc106567757)

[Behaviour – Harvest Edible Food 31](#_Toc106567758)

[Hunger and Eating 33](#_Toc106567759)

[Desired State 33](#_Toc106567760)

[Behaviour 33](#_Toc106567761)

[Directing the Character 34](#_Toc106567762)

[Driving AI Through Story Narrative 35](#_Toc106567763)

[Creating Custom Behaviours 36](#_Toc106567764)

[Design Your Behaviour 37](#_Toc106567765)

[Coding New Behaviours 38](#_Toc106567766)

# Getting Started

In this section we will set up a character that has two behaviours, wander and sleep. They will wander until they are tired and then they will sleep. We will also introduce the debugging tools available.

The goal of this section is not to present an in-depth analysis of how the AI system works. This is a very large and complex system. Instead the focus of this section is to get you up and running with a basic character quickly. We will introduce the core concepts needed to build more complex characters. In subsequent sections we will go deeper into crafting more complex AI.

To create a basic character, follow these steps:

1. Create a new Scene
2. Add a Plane, call it `Ground`
3. Set the plane to static
4. Make the plane size 3, 1, 3
5. Create an empty game object and name it for your character, e.g. “AI 1”
   1. Put it onto the `AI` layer
6. Add the characters model under this and name it “Model” (you can find a simple humanoid model at `Assets/Wizards Code/Character AI/Character/Models/Character Model.fbx` which is ideal for testing)
7. Ensure there is an Animator on the root of your character
   1. Turn off root motion on the animator
   2. Add ` Assets/Wizards Code/Character AI/Animation/Animations/Controllers/Humanoid Controller (Override This).controller` the animation controller for this animator (you will normally override this controller, or make your own, but this is a quick start 😉)
8. Add `Animator Actor Controller` to the root object
   1. Set the `Head` and `Animator` values in `Animator Actor Controller`
9. Drag the `Assets/Wizards Code/Character AI/Character/Prefabs/AI/Brain.prefab` prefab onto the character root
10. Add a `Rigidbody` to the root
11. Add an appropriate `Capsule Collider` and adjust to cover the character  
      
    A picture containing text

    Description automatically generatedA screenshot of a computer

    Description automatically generated with medium confidence
12. Add an `Audio Source` to the root
13. Bake the `NavMesh` in your scene
14. Add `Assets/Wizards Code/Character AI/Character/Prefabs/Essential Managers.prefab` into your scene
15. Hit play - your character should idle

For more information see the demo scene in `Characters/Scenes/Behaviours/101 Minimal Character` and `Behaviours/101 Minimal Character.md`.

# Core Concepts

If you worked through the getting started section you have some basic AI characters, but you don’t really know how they work yet. In this section we’ll go over the core concepts within the AI system. The concepts introduced are:

* Behaviours – the actions the character can choose to take
* Statistics – that define the sate of a character
* Desired States – that define how the character makes decisions about what to do next
* Interactables – which are items the character can interact with in the world
* Actor Cues – that tell the character how to act in given situations
* Memory – the characters ability to remember things in the world and act accordingly

## Behaviours

Your character will either be controlled by the AI brain or by directions sent from a director such as a player or an AI Director.

It’s important to give a character a set of behaviours they will perform when no other suitable behaviour is available. When a character has no behaviour they can perform they will simply idle, as is the case in the previous section. However, most characters should do something rather than just stand there. With that in mind let’s add a wander behaviour to this character.

### Adding a Wander Behaviour

This section will add a basic wander AI behaviour to your character. It's not a very interesting behaviour, but it is a start. This section won’t explore how the behaviour works, that will come later, for now we’ll just add the behaviour to the character so we can see them doing something.

1. Create a child of your characters Brain called `Idle Behaviours`
2. Add the `Wander` or `Wander with Intent` component to this object. the difference is only in the way the character wanders. Wander is entirely random, wheras Wander with Intent will continue in roughly the same direction until they run out of space, then they will pick a new direction.
3. The character will be moving around so let’s add a Cinemachine camera to make it easier to follow them ` Cinemachine -> Create Virtual Camera`
4. Setup the `CM VCam` to follow and look at your character. Set `Follow` to be the root transform of your character and `Look At` to something like the neck.
5. Make the follow offset something like 0, 4, 0
6. Set Yaw damping to something like 10
7. Hit play - your character should wander, although they will run everywhere since the limited animations provided only include a run. You can improve the animations by adding your own into the override controller.

### Deciding which Behaviour To Enact

Graphical user interface, text, application

Description automatically generatedCharacters will automatically decide which behaviour they should enact at any given time. They do this by having a Brain which periodically considers all possible behaviours and picks the one that will bring them nearest to their desired state. How the brain decides is quite involved. It all hinges on the current state of the character, defined by a set of statistics, and the desired state for those statistics.

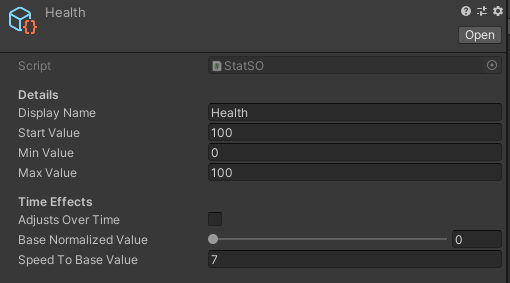
The above wander behaviour has no required statistics to enable it to be performed. Therefore, it can be performed at any time. Since the character has no other behaviours right now this means the character will always choose to wander.

This character is not very interesting yet, so let’s add another behaviour. Before we add this behaviour we have to understand the concept of Statistics and Desired State.

#### Statistics (Stats)

Each character has several stats that are used to track the status of the character and to help them decide what behaviours they should enact.

Stats are defined as scriptable objects. They have a Display Name that will be used in the UI, a start value used to initialize the stat upon creation and a minimum and maximum value.

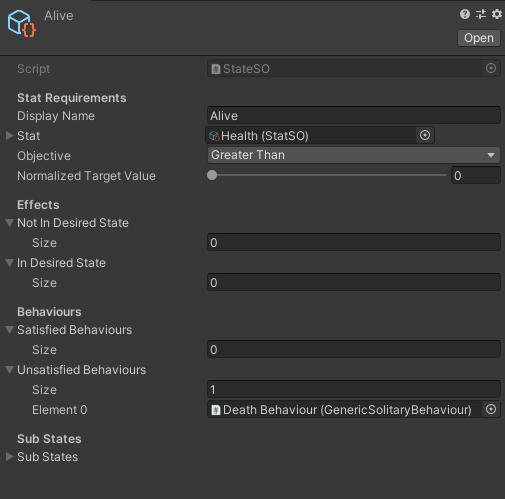
They can also have a time-based modifier that will cause the stat to tend towards a particular value over time. For now let’s ignore the time based stats and focus on as simple stat like Health. Here, health is defined as having a min value of 0 and a max of 100. Any character that has this stat will start out with a health of 100.

Characters are assigned stats in the Brain component. If the brain attempts to read a value for a stat and it does not exist then it will be created and tracked from that moment onwards. This means that Characters will automatically track any stats that are important to them in your game world. This, of course, begs the question “how does the brain know to access a stat value”. There are a number of ways this can happen, for example a behaviour may require a specific stat level, e.g. buying food will require the character to have some money. Another way of introducing a stat to a brain is through Desired States. The next section will discuss this state.

## Desired States

Desired States define the goals the character has. A desired state describes a stat and a target value. The character will prefer to take actions that drive their stats towards desired states.

If a character is not in a desired state zero or more stat effects will be applied. For example, if a player is hungry their health can gradually fall. It is also possible to force specific behaviours when in or not in the desired state.

Perhaps the simplest example is the “Alive” desired state. This tells a character that they should seek to have a health value greater than 0. There are no effectors defined in this desired state, however, if the character ever moves out of this state, i.e. normalized health falls to 0, then the Death Behaviour will be triggered.

As the value of Health tends towards zero this character will place a greater emphasis on performing actions that will increase their Health stat. Stats can therefore be used to drive base behaviours, like staying alive, but they can also be used to give characters apparent personalities.

For example, by giving a character a Happiness stat and a desired state of a high value for Happiness they will prioritize activities they enjoy over ones that benefit in other ways, such as work to increase money. Such a character would appear to be more hedonistic than another character whose desired states preference money.

Graphical user interface, application

Description automatically generatedSince the Alive desired state refers to the Health stat attaching this desired state to a characters brain will result in the Health stat being tracked. This state is attached by default to the brain if you used the provided brain prefab as described above.

If you hit play with and move the slider for the Normalized Target Value to 1 then the character will no longer be in this desired state since normalized health can never be greater than 1). This will cause the “Unsatisfied Behaviours” to trigger, which in this case is the Death Behaviour.

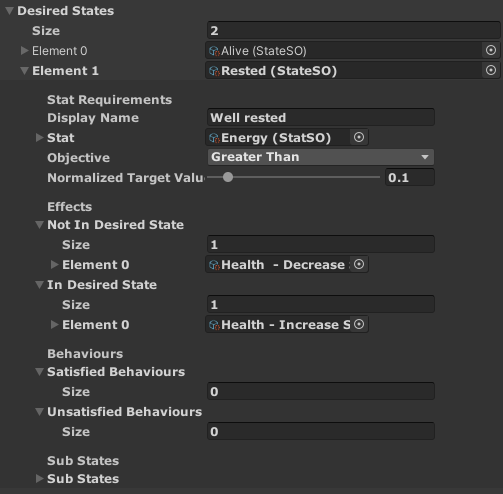
Try it. Your character will fall to the floor, dead. This is an artificial example. Normally you would not kill a character because they have maximum health, but the experiment serves a purpose.

Let’s now add a desired state that will encourage the character to choose a behaviour other than the default wander behaviour above.

### Getting Tired and Resting

Let’s add a new desired state, called Rested, that causes the character to have a certain level of energy. When they drop below that level of energy they will rest.

As before, we’ll work with prefabs for now. The goal is to introduce the various concepts of the system gradually. However, if you are eager to explore you will find that all parameters exposed in the inspector have tooltips.

1. Select your characters Brain and increase the size of the Desired States array to 2.
2. Click on the selection target for the new element and select the Rested desired state from the available states list.

Expanding out the new element you will see that the goal of this character is to have a normalized Energy level above 0.1. This desired state introduces us to the effectors that states can have.

If the character is not in the desired state the health of the character will decrease, on the other hand if they are in the desired state health will increase. Let’s skip over how this happens for now, just trust in the name of the effectors applied. But as noted above, if you want to dig in then expand out the elements and see if you can figure out what is happening from the tooltips.

Graphical user interface, application

Description automatically generatedWhat will cause energy to go down? As is the case with many parts of this system many things may cause the energy to go down. For example, we might have a “Not Hungry” desired state with an effector that reduces energy if they are hungry. We could also have behaviours that use up energy. However, in this case we don’t need to do anything as the system is set up to automatically reduce energy levels over time. You can see this by expanding out the Energy stat.

Unlike the health stat that we looked at above this stat does have an “Adjust Over Time” setting. We can see that the Enegy stat will tend towards 0 over time.

This is great, but at the moment the character has no means of recovering energy. This means that if left long enough the character will simply die. What will happen is energy levels will decrease until they fall below acceptable levels, then health will start to decrease. Eventually, health will hit 0 and the character will enact the Death behaviour as they will no longer be in the desired state for health. We must therefore add a behaviour that allow the character to rest.

#### Adding a Rest Behaviour

We don’t want our character to rest unless they are tired. So the rest behaviour will e the first one I in which we look at the anatomy of a Behaviour. First off let’s make it work.

1. Add the ` Packages/wizardscode.character/Character/Prefabs/AI/Behaviour Sets/Behaviour Demo Scenes/Rest Behaviour Set.prefab` to your characters Brain by simply dragging it as a child onto the brain – note that you can add additional layers of game objects in order to group behaviours together if that is useful, that is, behaviours do not need to be a direct child of the brain.

Graphical user interface, application

Description automatically generatedBehaviours can have required stats before they will be chosen by the brain. For example, we don’t want our character to sleep all the time, we only want them to do so when they are tired. So the sleep behaviour requires that energy be below 30 before it will be available for selection.

If you hit play now you will need to wait for quite some time for the energy levels to drop below the threshold level for resting. This isn’t ideal for testing, so lets add a UI to help with testing.

## Debugging UI

In order to understand what a character is doing, and why, we need some feedback from them.

1. Drag the ` Packages/wizardscode.character/Character/Prefabs/UI/Dev UI Canvas.prefab` into your scene.
2. Graphical user interface, text, application

   Description automatically generatedAdd an `Event System` (Game Object -> UI -> Event System).
3. Hit play.

You now have a simple UI giving you information about what is happening in the world. Click on your character and a panel in the bottom lefts will show the stats the characters Brain is currently tracking. It also provides a slider to allow you to change the current value of those stats.

Drag the slider for Energy down to below 30 (remember 30 is the desired value and thus the not in-state effectors will kick in. If you now watch Health you will see it will be dropping as a result of the low energy levels of the character. However, the character is not resting. Lets use the Debug output to figure out why.

### Debugging the Rest Behaviour

We have already established thorough the stats panel that the desired state is working as expected because health decreases when energy levels are too low. We therefore need to look into the decision making process of the brain.

If you consult the Console you will see a log of the brains “thinking”. As with many AI systems it can sometimes be difficult to ascertain exactly what is happening, but we try to log as much as possible here. So lets take a look at a decision when the energy level is below 30:

Text

Description automatically generated

We can see in this log that the character ultimately decided to “Wander with Intent”, well we know that, we could see it in the debug UI. However, they next set of lines are new information. We can see that the character did evaluate whether they should sleep, but that the behaviour was not available. The reasoning for this is that while energy is below the target of 70 the character “Couldn’t see or recall a suitable interactable nearby” and thus they “will have to find one first.”

Graphical user interface, application

Description automatically generatedWhat this means is that the character won’t choose to sleep just anywhere. They must find a suitable place to sleep first – an interactable. Interactables are things in the world that, if the character interacts with them, will influence them. If we examine the sleep behaviour, we can see what kind of interactable the character is looking for.

Expanding out the “Desired State Impacts” we see that we need to find an interactable that will cause the stat “Energy” to be greater than the value it was at the start of the interaction. OK, that makes sense.

## Interactables

How do we add an interactable into the world? We’ll go into details later but for now lets just use a prefab. Drag a ` Packages/wizardscode.character/Character/Prefabs/Interactables/Buildings/Bed.prefab` into the scene, hit play, select the character in the game view and drag the energy slider down to under 30.

The character will now go to the bed and lie down and start snoring. They will recover energy and eventually get up and start wandering again. How do they know to lie down and snore? We will look at that next. First, there are a couple of things to note about this process.

Firstly, the character will only see a bed that is within a distance defined in the behaviour. By default, this distance is quite small. The larger you make the distance the less performance the system will be. This can make the AI look quite dumb as they will only “see” the interactable when they are close to it, even if they have walked right by it before. There is a memory system which can improve this significantly without a major impact on performance. However, before we get to the memory system, we have other things to explore.

The second thing to be aware of is that many objects have a cooldown period that prevents them from being reused for a period. Therefore, sometimes the character will appear to be ignoring the bed, but it may simply be that the bed is not available to them. In normal gameplay this wouldn’t be a problem as the character will never want to sleep in quick succession, but in testing you may artificially reduce energy causing the character to need to sleep very quickly. A way to avoid this in testing is to add another interactable. You might test this by adding a second bed, or maybe a ` Packages/wizardscode.character/Character/Prefabs/Interactables/Buildings/Home.prefab` which also allows characters to sleep.

## Actor Cues

AI Characters are considered to be actors in your game. We can give the actors cues to tell them how to behave in each situation. This will include things like making sounds or playing animations.

Graphical user interface, text, application

Description automatically generatedWe saw such a cue when the character lay down on the bed. Select the `Rest Behaviour Set` and open take a look at the Sleep behaviour. Notice that there is a section that defines actor cues.

There are several cues we can define, each being used at a different stage of enacting the behaviour, however, all the cues have the same general structure. The stages of enactment are as follows:

* On Start – this cue is prompted as soon as the character starts to engage in this behaviour. It is usually used to signal the character is about to enact this behaviour. For example, the character may pull out an item they will use in the interaction.
* On Prepare – this cue is prompted as the character starts the interaction. It will usually be used to perform any kind of “warm up” activity prior to the interaction. For example, they may position themselves perfectly to carry out the interaction.
* On Perform – unlike the other cues the On Perform cue can take multiple cues. One of these cues will be selected to be prompted as the character performs the interaction. In this cue we will typically perform the actual interaction.
* On Finalize – the Finalize cue is used to finish the interaction. This is where you will perform any recovery actions.
* On End – The End cue is used to signal that the character has completed the interaction. For example, they may put away the item used.

In the above pictured sleep behaviour, we have a defined a preparation (set sleeping animation parameter to true), Perform (play sleeping sound) and Finalize (set sleeping animation parameter to false) cue. We could add an undress or yawn cue for start and a dress or stretch cue for the end if we wanted.

### Anatomy of a Cue

Cues can instruct the actor to do several things, such as those listed below. In the case of the sleep animation, we simply set a Boolean parameter to true or false depending on whether we are preparing or finalizing the sleep behaviour.

* Move to a given Mark
* Play a sound
* Change the weight of an animation layer
* Change animation parameters
* Play animation clips

## Memory

By default, our characters have no memory. If they can’t see what they need to perform a behaviour that behaviour will not be chosen. However, in many cases you want your characters to remember things that they have used in the past. This is easily achieved by dropping a memory component onto your character. It can go anywhere as long as it is a child of the character root.

There is no configuration needed for the memory component. If it is present, it will be automatically used.

Memories have both long and short-term stores. Memories go into short-term first. They will remain there until either driven out by a more recent memory or moved into long term memory. The conditions under which they are moved into long term memory as set by the `Long Term Memory Threshold`.

# Example: Shooter Combat

In this section we’ll describe how to set up a simple shooter combat between Ais.

## Configure the Scene

1. Create a basic scene as per the Getting started guide
2. Duplicate the AI character and name it something like “AI 2”
3. Move the two AI characters so they are some distance apart, say 20m
4. Position the camera so you can see both AI
5. Give one of them a red skin color by dragging ` Assets/Wizards Code/Character AI/Character/Materials/Character Prototype Red.mat` onto the renderer for your character  
   Graphical user interface

   Description automatically generated

## Create the Actor Cues

When the AI are given the direction to perform a particular behaviour they will need a set of cues to define how they should behave. Think of the cues as phases of the behaviour, see Actor Cues for more information.

For the Shooter behaviour we will need:

1. “Shoot Pistol – Prepare” cue that will Move the AI into an aiming stance. See ` Assets/Wizards Code/Character AI/Character/Resources/Actor Cues/Shoot Pistol - Perform.asset`
2. “Shoot Pistol – Perform” cue that will fire the shooting animation. See ` Assets/Wizards Code/Character AI/Character/Resources/Actor Cues/Shoot Pistol - Perform.asset`

## Configure the First Shooting AI

In this section we will configure the Red AI to shoot the blue AI.

1. Add an empty game object as a child of the `Brain` and call it `Enemy Senses`
   1. Add a `SightSense` component to `Enemy Senses` object and name it `Spot Enemy`
   2. Configure the `Layer Mask` to the “AI” layer
   3. Add the `Alive` state to the `Required Satisfied States` otherwise the AI will continue to take notice of the enemy even when dead.
2. Add an empty Game Object as a child of the `Brain` and call it ` Behaviours - Enemy Spotted`
   1. Add a `Attack Behaviour`
   2. Give it a name and a description. For example “Shoot at Enemy with Pistol” and “If the AI spots an enemy attempt to move within shooting range and shoot them with a pistol.”
   3. Reduce the `Retry frequency` to 1 second as we want the AI to react quickly if this is a valid behaviour
   4. Mark is as `Is Interuptable` so that other behaviours we might add later, such sa run away if injured, can interrupt this behaviour
   5. Change the `Maximum Execution Time` to 5 seconds as it shouldn’t take long for the AI to complete the shoot operation and, if in combat, we don’t want them getting stuck.
   6. Setup the Actor Cues as follows:
      1. The prepare cue should be ‘Assets/Wizards Code/Character AI/Character/Resources/Actor Cues/Shoot Pistol - Prepare.asset’
      2. The perform cue should be ‘Assets/Wizards Code/Character AI/Character/Resources/Actor Cues/Shoot Pistol - Perform.asset’
   7. Add the `Enemy Senses` sense as a required sense  
      Graphical user interface, application, website

      Description automatically generated
   8. Set `Require Consent` to false since it is rare to require that your enemy agrees to being shot
   9. Reduce the `Cooldown Time` to 1 second so that the AI can shoot frequently
   10. Configure the hit influence as follows:  
       A screenshot of a computer

       Description automatically generated with medium confidence
   11. Hit play and let the Red AI shoot the Blue AI a bunch of times, eventually the Blue AI will keel over and die.

## Configure the Second Shooter AI

The blue AI, as they are currently configured, will just stand there and take damage until they keel over and die. Let’s fix that, we can make them shoot back.

1. Copy the `Enemy Senses` and the `Behaviours – Enemy Spotted` objects from the red AI.
2. Paste them as children of the blue AI’s `Brain`
3. Hit Play

# Example: Populating a World

In this section we will create an entire world, populated with characters and animals that go about their daily life. We are going to use the following assets, but you can replace these assets with any that you have. The AI code should remain the same.

* Wizards Code [Rolling Hills Terrain, Brushes and Stamps](https://bit.ly/UnityRollingHills) (Free)
* Synty Studios [Farm Pack](https://bit.ly/PolygonFarm)

## Create the World

* Copy Synty Farm Pack demo Scene into `\_PopulateTheWorld` folder   
  Graphical user interface, text

  Description automatically generated

## Create Our First Character

* Setup a Farmer character as per Getting Started
* Add Cinemachine Virtual Camera to the scene
  + Set to follow and look at Farmer
* Hit Play – Farmer should wander

## Harvesting Crops

* Add a Harvest Crops behaviour
  + Graphical user interface, application

    Description automatically generated  
    Graphical user interface

    Description automatically generated
* Add a desired state for Raw Food
  + Graphical user interface

    Description automatically generated
* Make a harvestable potato prefab
  + Graphical user interface, application

    Description automatically generated
  + Graphical user interface

    Description automatically generated
* Replace all large potatoes in the scene with the harvestable potato
* Hit play
  + Since the raw food desired state is not satisfied the harvest crops behaviour is added
  + Character should harvest the potatoes
  + Once enough crops have been harvested the harvest crops behaviour should be removes

## Work and Rest Cycle

### Desired State – Sufficiently Rested

The desired state is to have Energy levels above 10% of their maximum. When not in this sate they will lose health and the rest behaviour will be added to the character.

Graphical user interface, text

Description automatically generated

### Behaviour – Sit and Rest

The rest behaviour will have the player sit down and do nothing for a while. This will regenerate a little energy. They will be willing to rest whenever their energy levels are below 30. The with of this behaviour is quite high as having too little energy is damaging to health.

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

### Behaviour - Resting when Idle

The character should rest, when not busy, even when not unhealthily tired. The goal is to never get that tired. To achieve this we also add a rest behaviour under the Idle Behaviours that are permanently on active on the character:

Graphical user interface, text, application

Description automatically generated

This resting behaviour has a higher minimum energy value before it will be chosen but a lower weight. This results in the behaviour being selected when there is nothing else that needs doing.

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

## Gathering Edible Food

### Desired State – Carrying Enough Food

Our characters will want easy access to enough food for them to survive. We therefore add a Desired state that requires them to be requiring around 25% of their maximum food value. When they have less than this then a Harvest Edible Food behaviour is added.

Graphical user interface, text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

### Behaviour – Harvest Edible Food

This behaviour will seek out edible food sources and, if available, will gather food from them.

Graphical user interface, application

Description automatically generated Graphical user interface

Description automatically generated

## Hunger and Eating

Naturally our characters will need to eat when hungry.

### Desired State

### Behaviour

# Directing the Character

Sometimes you will want to instruct the character to perform specific actions. This can be done via the API in response to things like player input, an AI director, or a story. If left alone the AI will take whatever action it can to satisfy its needs. This will be discussed in the next section, but first let’s see how an AI can be given directions within a story.

## Driving AI Through Story Narrative

The character system extends Ink to allow the character to be given instructions. This is great for storytelling games but is also very useful for testing new characters. The resulting Ink script will still run in standard Ink interpreters and so you can easily split story telling from AI directions.

1. Create an object called `Ink Manager` and add the `Ink Manager` components to it.
2. Drag the `Ink/Prefabs/Ink Manager` into the scene
3. Create a new Ink story using `Create -> Ink`
4. Add the content below to the story file
5. Drag the compiled JSON file (stored in the same folder as you Ink file) into the `Ink Manager` parameter `Ink JSON`
6. Add an Actor to the list of Actors in the `Ink Manager` and drag your Character into the slot.
7. Somewhere in your scene add a cube called `Mark1`, it must be on the NavMesh, but other than that it can be anywhere. Remove the collider.
8. Hit play, your character will wander as before. However, you will now have a narrative dialog.
9. Click the "Go to Mark 1" button, the character will navigate to the mark you set.
10. If left to their own devices the character will start to wander again.

-> CharacterDevScript

= CharacterDevScript

{stopping:

- Hi, I'm your new character.

- What shall I do now?

}

+ [Go to Mark 1] -> Mark1

\* [Goodbye] -> END

= Mark1

>>> MoveTo: Goat Herder, Mark1

Oh, I love Mark 1.

-> CharacterDevScript

# Creating Custom Behaviours

In this section we will cover the basic process of creating custom behaviours.

## Design Your Behaviour

There are a number of types of behaviour. The first decision you need to make is what type is the behaviour you are trying to make. The available types are:

* Solitary Behaviour – a behaviour that can be carried out at any time without any kind of external object or character to interact with.
* Interaction Behaviour – a behaviour that requires some kind of interactable object to be available.
* Actor Interaction Behaviour – a behaviour that requires another character to be available.
* Building Behaviour – a behaviour that uses resources to create something in the world.

In many cases you won’t need to create new code to create a new behaviour. You will be able to take one of the generic implementations of these behaviours and configure it in the editor to create the effect you desire.

See `GenericSolitaryBehaviour`, `GenericInteractionBehaviour`, `GenericActorInteractionbehaviour` and `GenericBuildingBehaviour` components and their use in the examples provided in `Assets/Wizards Code/Character AI/Scenes`.

## Coding New Behaviours

Sometimes the generic behaviours do not provide the ability to build the characters you need. In these circumstances you will need to create your own. This requires some coding and as such is an advanced topic. In this section we will examine the creation of the `ShootBehaviour`. A more advanced version of this behaviour is available as part of the asset, see ` Assets/Wizards Code/Character AI/Character/Scripts/Runtime/Actor/Behaviours/ShootertBehaviour.cs`, and can be seen in the demo at ` Assets/Wizards Code/Character AI/Scenes/05 Combat/201 Ranged Combat.unity`.

Since this behaviour will see the AI shooting at another character this will be an Actor Interaction Behaviour. So we start out by creating a class called `ShootBehaviour` which extends `GenericActorInteractionBehaviour`.

The Ai will first need to move to a suitable position to attack the other character, this is handled in the ` UpdateInteractionPosition(bool requireNavMesh)`. For our simple shooter we will have an optimal range from which to attack and we’ll tell the AI to move to any point it wants within that range.