Brief – Networked Resource Management

What it is:

It's a modular networked resource management system that will be easy to implement by a developer, the system will be split into multiple subsystems in the form of components and scripts that the developer can pick and choose to use from, all the developer has to do is add the subsystems to the game object they want to have that functionality and set up the variables in the inspector.

This can be used for developing online and offline City Builders and Colony Sims, like SimCity, Cities: Skylines, Banished, and Patron. The package will contain pre-made subsystems including a Resource Attribute and Management System, a Resource Node System, and Networking systems the developer can use to speed up their workflow by removing the need to spend time developing these systems themselves, this will allow the developer to integrate their unique systems and the other game mechanics into their game, fast tracking their development cycle.

Third party libraries:

Mirror v66.0.9:

License: Extension Asset

Mirror is a high level networking API for Unity, supporting different low level Transports. It's included in the system to take care of the majority of the networking side of the system.

What will be contained in the Resource Management part:

- Food:
  - Random spawning vegetation
  - Farming food
  - Whether or not it respawns over time

- ○ Variables:
  - Rate of growth (float)
  - Food resource produced (enum)
  - Amount of resource obtained (float)
  - How much Hunger and Thirst it restores (float)
  - Time it takes to harvest (float)
  - What tool is used to harvest (enum)

- Water:
  - Well – A well will be a placeable building that can be used to collect drinking water. Space is pre-defined.

  - Body of water – A definable area of water that can represent lakes and rivers, these can be used to collect drinking water. Space is definable.

- Variables:
  - Is it clean/safe (bool)
  - Is it depletable (bool)
  - How much water is left (float)
  - Does it refill (bool)
  - Time it takes to refill (float)
  - How much thirst does it restore (float)
  - Time it takes to harvest (float)
  - Resource produced (enum)
  - What tool is used to harvest (enum)

- Customisable resource nodes:
  - Does it spawn is clusters or by itself (bool)
  - How spread apart the nodes are (float)
  - How rare it is (float)
  - Is it farmable (bool)
  - Is it plantable (bool)
  - Rate it grows at (if viable) (float)
  - Time it takes to harvest (float)
  - What resource it is / what resource it drops (enum)
  - How much of that resource it drops (float)
  - How depleted it is (if viable) (float)
  - What tool is used to harvest (enum)
  - If it auto respawns (bool)
  - If desirable, can inherit functionality from food and/or water. For the functionality that overlaps, the version of the variable in the Custom Node will be used, the version in the duplicated functionality will be discarded.

- Storage
  - What resources can it store
  - Are the resources visualised on/in it
  - The model/s the visualised resources will use
  - How many resources it can hold

Mathematical equations:

A combination of basic addition, subtraction, division, and multiplication will be used in most functions.

The constant rate of decay will just use a lerp function **(a + (b – a) * t)** and **t** will be equal to the **current accumulated time** divided by the **maximum amount of time (t = time / max time)**.

When adding resources to the storage, there will be an equation to increase the time of decay **((number of resources + amount to add) / max amount of resources * max time)** since time controls how many resources there are.

Nodes that grow will use the same lerp function **(a + (b – a) * t)** as the constant rate of decay, using **t** as a percentage of its life cycle to control the max yield that resource can produce using logic to determine when during that life cycle the resources able to be gained will drop off.

How the systems will be modular:

Networking:

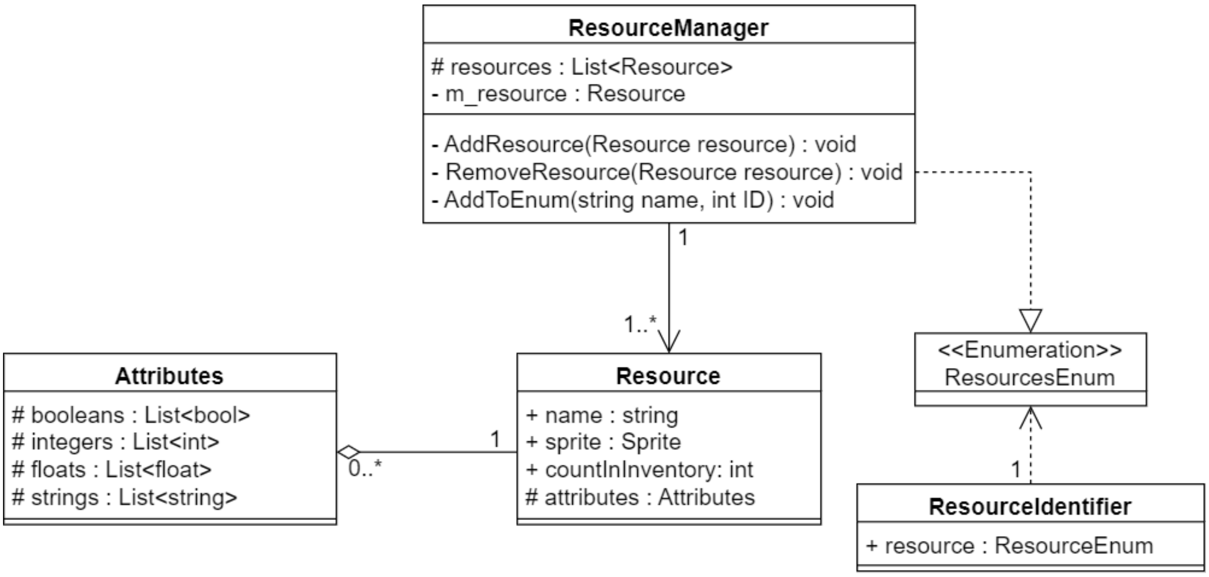
A script that is added to a game object will allow the developer to specify what public variables in the other scripts are to be sent to and from the server and what connection type to use. The system will have default variables that are sent to and from the server for the resource management system that can be overridden by the developer.

The variables the developer chooses to send to and from the server will be split into two lists representing two types of communication protocols, TCP/IP and UDP. The developer will be able to attach different scripts as well that control what to do when there's no response from the server or if there's an unexpected response. There will be some default options for what to do when there is no response from the server.

Resource Attribute System:

A scriptable object will be used to create the resources and their attributes as two separate list.

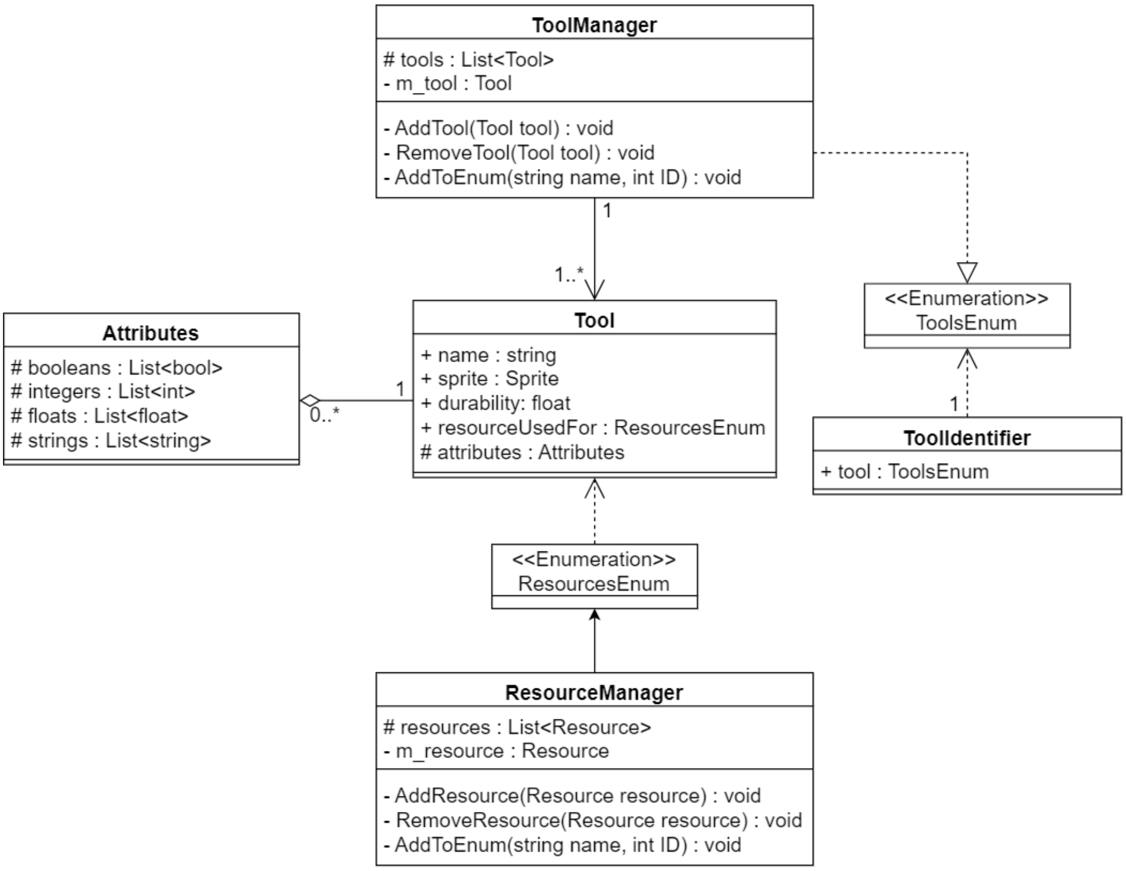
The resources and attributes will be accessible from other components and scripts.



Tool Attribute System:

A similar system to the Resource Attribute System will be implemented for creating tools using a scriptable object with the tool and the attributes being split into two separate lists.

The tools and attributes will be accessible from other components and scripts.



Resource Nodes:

The resource node will be a component that can be added to a game object and will hold information that will control how that resource node interacts with the world and the player, the information it will hold is listed as follows:

- Does it spawn is clusters or by itself (bool)
- How spread apart the nodes are (float)
- How rare it is (float)
- Is it farmable (bool)
- Is it plantable (bool)
- Rate it grows at (if viable) (float)
- Time it takes to harvest (float)
- What resource it is / what resource it drops (enum)
- How much of that resource it drops (float)
- How depleted it is (if viable) (float)
- What tool is used to harvest (enum)
- If it auto respawns (bool)

There will also be an option inherit variables from Food and Water resources and to add custom variables to the resource node, the developer will have to script in what the custom variables do.

Storage Nodes:

The Storage Node will contain a list of the resources and tools with the amount of each resource and tool stored within that Storage Node. Storage Nodes won't have any default way of visualising in game what is in them or how many of each item is contained, this is something the developer will have to implement themselves through scripts. The Storage Nodes will contain functions to add and remove a specific amount of a resource or tool. The information and functions in the Storage Node will be accessible from scripts.

How it will be integrated:

The system will be a unity package downloadable from the asset store, when added to the project, the developer will import the package.

To define resources and tools and the attributes for both, the developer will be able to create two separate scriptable objects, a Resource Manager and a Tool Manager. Both the resources and the tools will be accessible from components, the resources and their attributes will be accessible from the Resource Node component to define the physical resource in the world. Both the resources and the tools will be accessible from the Storage Node component, so if the developer wants to, both resources and tools and be put in storage. Both resources and tools will also be accessible from scripts allowing the developer to create custom usages for the resources and tools if desired, an example of a custom usage for resources and tools would be being displayed in and selectable in a hotbar/inventory system.

A Resource Node component will be able to be added to game objects for the Resource Management System to recognise that object as a node for a specific resource. Resource Nodes contain information on how the world and the player can interact with the Resource Node and what is obtained from it once harvested. The information in Resource Nodes will be accessible from scripts allowing the developer to implement custom functionality if desired.

A Storage Node component will be able to be added to a game object for the Resource Management system to recognise that object as a node to store resources and tools in. Storage Nodes will contain information of how much of each resource and tool is currently held in a Storage Node. The information of how much of each resource and tool will be accessible from scripts allowing the developer to create custom usages for the Storage Node if desired, an example of a custom usage would be an inventory system for the Storage Node showing what is stored in there and how much.

Networking for the Resource Management System will use a component that can be added to a game object that will contain two lists, one holding names of scripts to be used and another holding names of variables in those scripts to be used. The developer will fill the lists with the names of the scripts and variables in the parent game object, the variables will be sent between the client and server using the KCP UDP communication protocol.