# Plant spread prototype

## Introduction:

The plant spread algorithm uses two scripts to simulate the seeds of a plant spreading into the environment. The limiting factors of this spread are:

1. The plant can only spread if the checked location(s) are unobstructed
2. The location decided upon must be navigable from the navmesh

Following these two principles, the algorithm guarantees that the plants will always spawn somewhere accessible to the herbivores in the simulation.

## Code:

To ensure that the principles are followed, two scripts have been written. One facilitates the random nature of the seeds spreading, by calculating an array of points around the plant, using predefined parameters. The other script then randomly picks one of these points, checks for obstructions and verifies accessibility to the location via the navmesh:

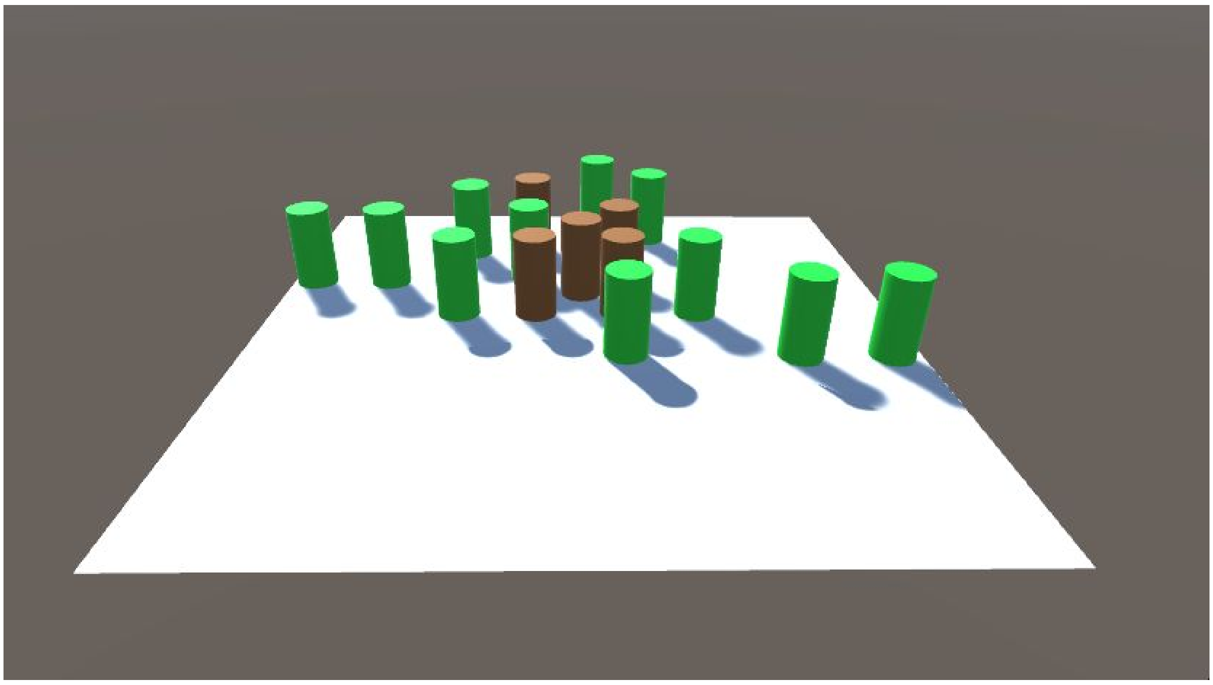
1. **CastPointScript**

*Generates an array of points around an object. The points are evenly spaced across the angles, but appear at random radii between a minimum and maximum distance value.*

2. **PlantScript**

*Randomly selects one of the points generated by the* ***CastPointScript****, then checks if the point is navigable from the navmesh. If it is, the script will then check to see if the area is clear, within a specified radius. This final check ensures that the plants do not appear too close to anything else that can be collided with, including each other. Each plant only has a number of attempts to spread. Both a successful and failed attempt will decrease this number. When the number reaches zero, the plant will be modified to reflect that it can no longer spread (turning brown in this prototype).*

**N.B:** *Whenever the values need to be tweaked, make sure to do so in the prefab instance of the plant, not in the plant which sits in the scene. This way the changes are reflected inside all future instances of the plant object as it spreads.*



# 1. CastpointScript:

## 1.0 Fields

Most of the fields in this script are initialized by the **PlantScript** class. The values which they are initialized with here, were simply for testing the CastPointScript class individually.

1. Public GameObject[ ] castPoints

*An array of references to the points generated around the plant.*

2. Public float rangeMin, rangeMax

*The minimum and maximum radius values by which the points will appear around the plant.*

3. Public int numberOfPoints

*How many points should be generated around the plant*

## 1.1 PopulatePoints()

This function generates the points around the plant, using the fields specified above. It is the only function in this class, containing a lot of functionality compressed into a single call. Here are the individual steps taken by the function, in their correct order:

1. Check if there are any objects currently in the array. If yes, clear them out. *This step might actually be redundant, as the array is re-initialized next. But you never know.*
2. Initialize the array.

3. Populate the array with with empty gameobjects
4. For each point in the array, find the angle (in radians) for the current point and a random radius between *rangeMin* and *rangeMax*. Apply the angle and range to calculate the correct coordinate, then update the position of the current object.

# 2. PlantScript:

## 2.0 Fields:

The fields found here partially reflect those found in the **CastPointScript** class, as some values are passed on to that class from here, to promote centralization:

1. GameObject[ ] castPoints

*A reference to the points found by the **CastPointScript** class.*

2. CastpointScript castpointScript

*A reference to the **CastPointScript** itself.*

3. Public float waitMin

*The minimum amount of time that can pass before the plant attempts to spread again.*

4. Public float waitMax

*The maximum amount of time that can pass before the plant attempts to spread again.*

5. Public float checkRadius

*The distance by which a spherical area must be cleared of non-ground colliders, before the plant is allowed to spread (eg. how close the plant is allowed to grow to other objects).*

6. Public float attempts

*How many attempts the plant has at spreading itself, before being rendered inert*

7. Public float rangeMin

*The range minimum passed to the **CastPointScript** class, used for random radius*

8. Public float rangeMax

*The range maximum passed to the **CastPointScript** class, used for random radius*

9. Public int numberOfPoints

*The value for the number of points passed to the **CastPointScript**, used for the resolution of the circle generated by the algorithm*

## 2.1 Start():

The Start function looks inside of the plant object's children, to find the "*CastPoints"* object and fetch a reference to the **CastPointScript** located on it. Once this has been established, a call to the *SpreadPlant* coroutine is made.

## 2.2 bool CastFromPoint(Transform **point**, out Vector3 **pointFound**):

This function will return true if it is possible to path from the given **point** parameter, to the plant itself. This is done with a downwards raycast. If the raycast returns nothing (eg. it is probably underground), a subsequent upwards raycast will be performed. Additionally, the out-parameter **pointFound** will be modified, should a new point have been found.

## 2.3 IENumerator SpreadPlant():

This coroutine waits for a random amount of time, between the *waitMin* and *waitMax* values (interpreted as seconds). Then it will pass its current values for *rangeMin*, *rangeMax* and *numberOfPoints* to the corresponding variables in the **CastPointScript** reference. Via this reference, the script will then be ordered to populate its array of points, the result of which is then stored in the *castPoints* variable within this class.
A call to *SpawnAtValidPoint* is then made, to randomly choose one of these points and attempt to create a new plant at that location. Then the value of the *attempts* variable is decremented by 1.
Finally, a check to see if there are any attempts left is made. If there are more attempts than 0, a call to the *SpreadPlant* coroutine is made (effectively looping the coroutine). If no more attempts are available, the colour of the plant is modified to reflect that it may no longer attempt to spread itself.

## 2.4 bool VerifyPath(NavMeshPath **path**):

If the status of the passed **path** parameter is fully navigable (eg. the status is "complete"), return true. However, if the path is invalid or even just partial, instead return false to indicate that the path does not satisfy the constraints.

## 2.5 bool IfClear(Vector3 **locationToCheck**):

This function indicates if a spherical area around the **locationToCheck** point is clear of obstacles. This is done by creating a an OverlapSphere - a component of the physics system - to return an array of all colliders intersecting with the sphere. The array is then iterated through, checking if any of the colliders contain a tag that isn't "Ground." Should this occur, the function will return false, as the plant will be too close to another object. Otherwise return true to indicate that only the ground has been detected.

## 2.6 void SpawnAtValidPoint():

This function handles the random selection of the points generated by the **CastPointScript** class. It will randomly pick an index value in the array, then with a call to *CastFromPoint* verify if the point is accessible, and subsequently a call to *IfClear* is made to see if the point is far enough from everything else to be instantiated. If both of these constraints are satisfied, a new instance of the “Plant” prefab will be created at the given location.

## 2.7 void Refresh():

A function meant to be called externally, this will check if more attempts are available, and restart the *SpreadPlant* loop if true.