Portfolio Part One (UFCFJL-30-1)

Student Name: Jamie Serlin

Student Number: 24030960

Glossary

Normal	The direction that a face is facing on a mesh, perpendicular to the tangent
Subsurface scattering	A graphical technique that approximates light passing through a translucent surface such as hands or leaves
UV Unwrapping	The process of folding out a 3D model to allow textures to be applied
PBR	Physically Based Rendering is a rendering technique that uses additional textures for effects such as roughness, metalness, and extra normal detail with normal maps
High Concept	A short summary of the main design features and ideas of a game.
Vertex	A single point in the world, with x,y,z coordinates (and extra data for UV mapping). It is used in 3D modelling as they can be connected with edges and faces to create a mesh.

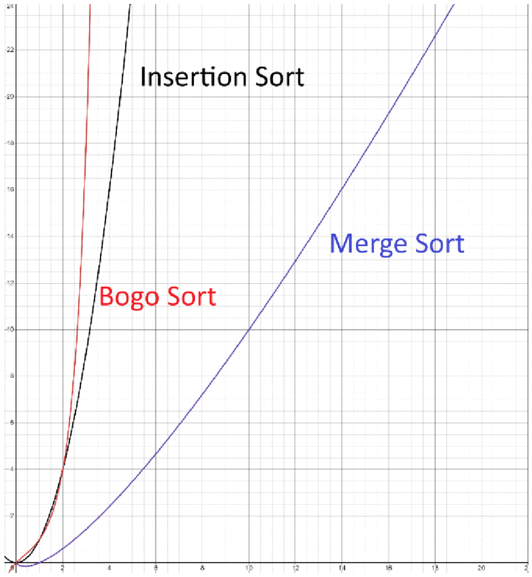
Linked List	A data structure in which multiple elements contain a reference to the next item of the list. This makes it easy to insert and delete elements.
Big O Notation	A mathematical concept to describe how fast different sorting algorithms work depending on the size of the set.

Task 1: Tools and Techniques

Game Engine	Pros	Cons	Appropriate Usage
Unity	-Easy to understand the interface and get started -Many packages and in-built components -Probuilder tool lets you create map blockouts in engine very quickly -You can make your own in-editor tools using C# and Unity's API -Works very well in both 2D and 3D	-Can have a lot of slowdown when projects get large -Not many high-quality assets on the asset store	-Great for indie games. -However, it is not well suited to large or AAA games as it lacks robustness.
Unreal Engine 5	-Extremely powerful graphics engine, with tools such as nanite and lumen for real-time global illumination. -Blueprint nodes let you script gameplay without needing to	- Quite demanding on performance, meaning you could be cutting out a large part of your playerbase -Not very good at making 2D games as it was designed primarily for 3D.	Unreal can be used for just about anything, with many large AAA titles being on this engine (Fortnite, Silent Hill 2), as well as smaller indie games (Grey Zone Warfare, Hi-Fi Rush).

	learn a language such as C++ - Quixel Megascans provides a huge library of high quality photo-scanned assets to use for free in the engine.		-Can also be used for films, architecture visualization and much more.
GameMaker 2	- Specifically tailored to 2D games, so has lots of support for things like tile-maps and sprite-sheets. - Very fast opening and build times, meaning it is quick to iterate on code. - Simple and intuitive interface	- Custom scripting language means that there are not a lot of outside libraries, and you will need to write a lot of your own methods. - No native video-playing support - £300 to buy, whereas other engines are free to use.	- Great for any 2D games - Good for beginner devs, especially those new to coding

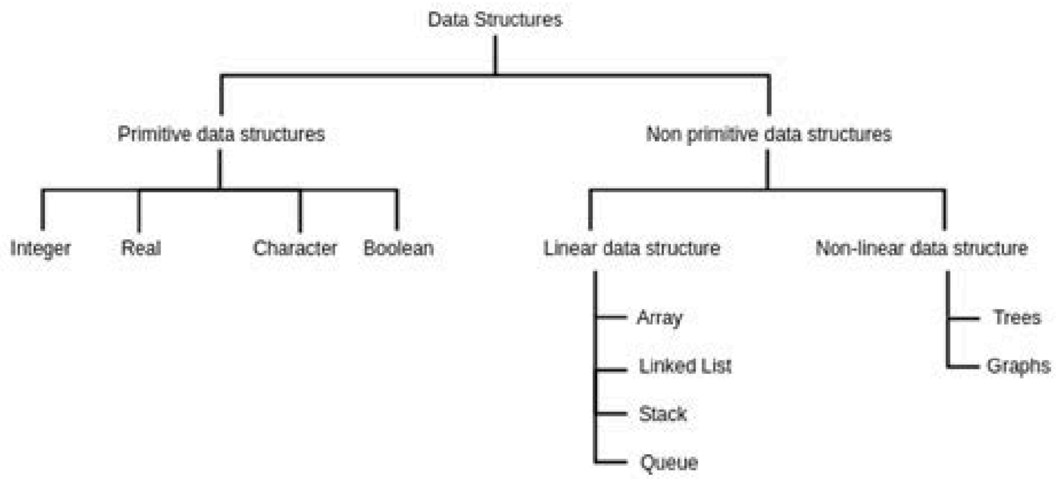
Task 2: Algorithms



Sorting Algorithm	Best Case Complexity	Worst Case Complexity
Insertion Sort	O(n)	O(n^2)

Merge Sort	O(n log(n))	O(n log(n))
Bogo Sort	O(n)	O(1)

Task 3: Data Structures



Integer

An Integer is a whole number, meaning it cannot have a decimal place. It has one bit reserved for the sign (if the number is positive or negative), and the rest store the value. Typically, an integer in C++ is 32 bit, meaning it can count from -2147483648 to 2147483647.

It is used in many different circumstances, for example counting how many times a player has died, or an ammo counter for a gun, as you cannot have half of a bullet.

Character

A Character is essentially an int, however each value is tied to an ASCII character such as ‘G’ or ‘m.’ Making an array of characters allows you to store entire words or sentences (usually called a string).

They are very useful for things like storing chat messages online, subtitles and any text in a game such as the main menu screen.

Array

An array contains multiple values of data of a specific type in a single variable. It has a fixed size, meaning you cannot add or remove an element from the array (without creating a new one). However, you can modify the variables inside the array by indexing it with an integer. The first element is 0, the next element is 1 and so on.

An example of a use could be a hotbar feature similar to Minecraft or a leaderboard in a competitive game.

Linked List

A linked list is similar to an array, however it gives you much more control over the order and size of the structure. It is made up of a HEAD and multiple Nodes that store data as well as the pointer to the next Node of the list. The final Node points to NULL. This is useful for something like a list of enemies that are alive, that has to be dynamically changed in size and content.

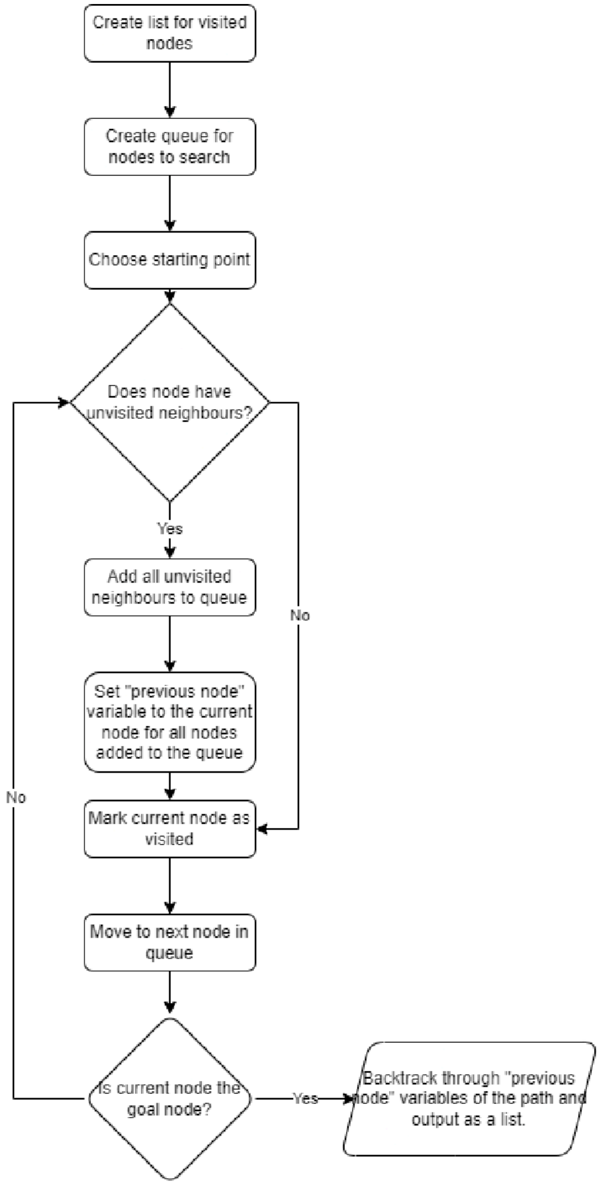
Tree

A binary tree is a hierarchical data structure where each Node can have up to two connecting nodes. It starts at the Root, and the ends of the tree are called Leaves. Each Node contains the data that it stores, as well as two pointers, one for the left branch and one for the right branch. The left or right can be set to nullptr, which means there is no branch coming from that side.

They are a hierarchical structure, so are very efficient for searching for data (e.g. "find all users that signed up between 2020 and 2021").

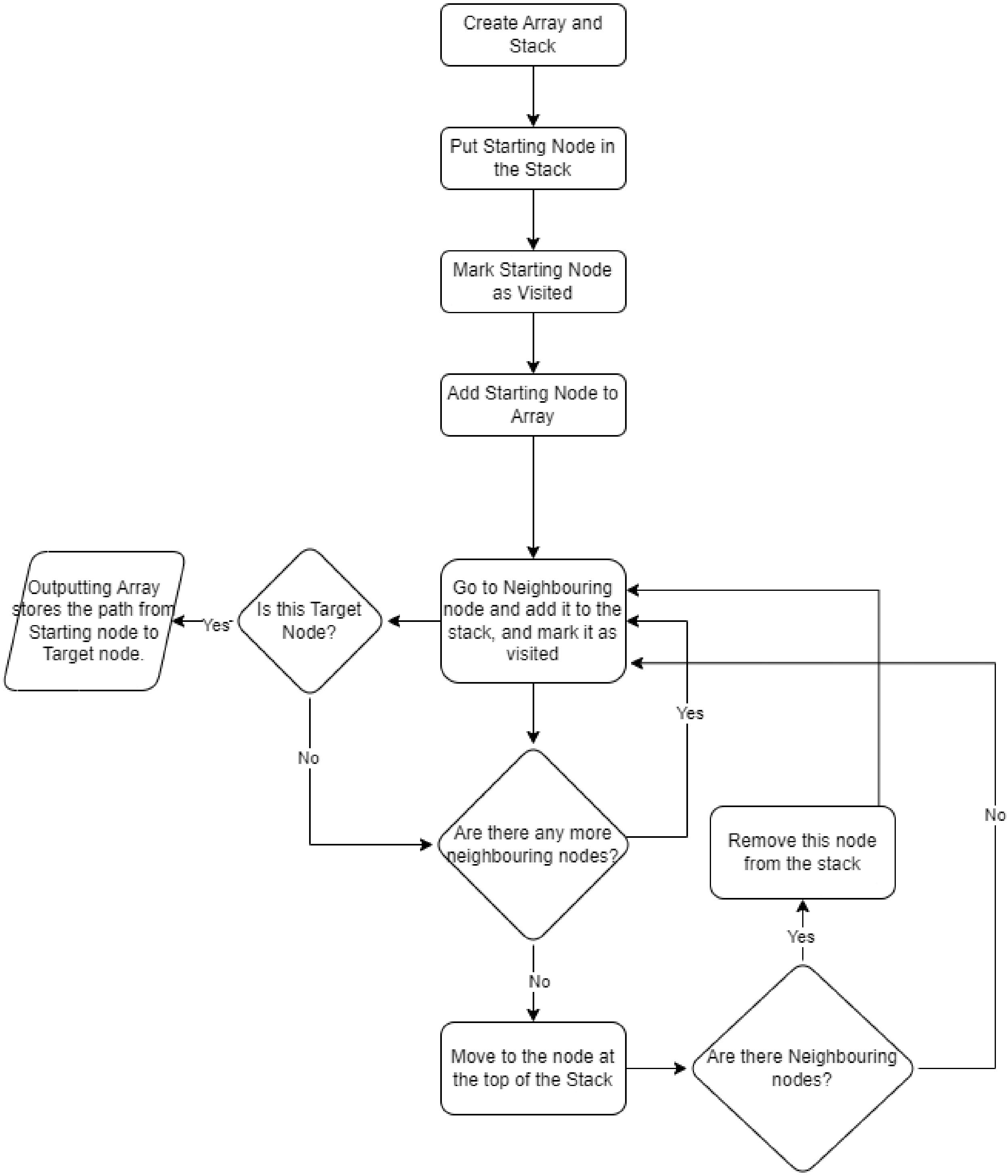
Task 4: AI in Games

Breadth-First Search

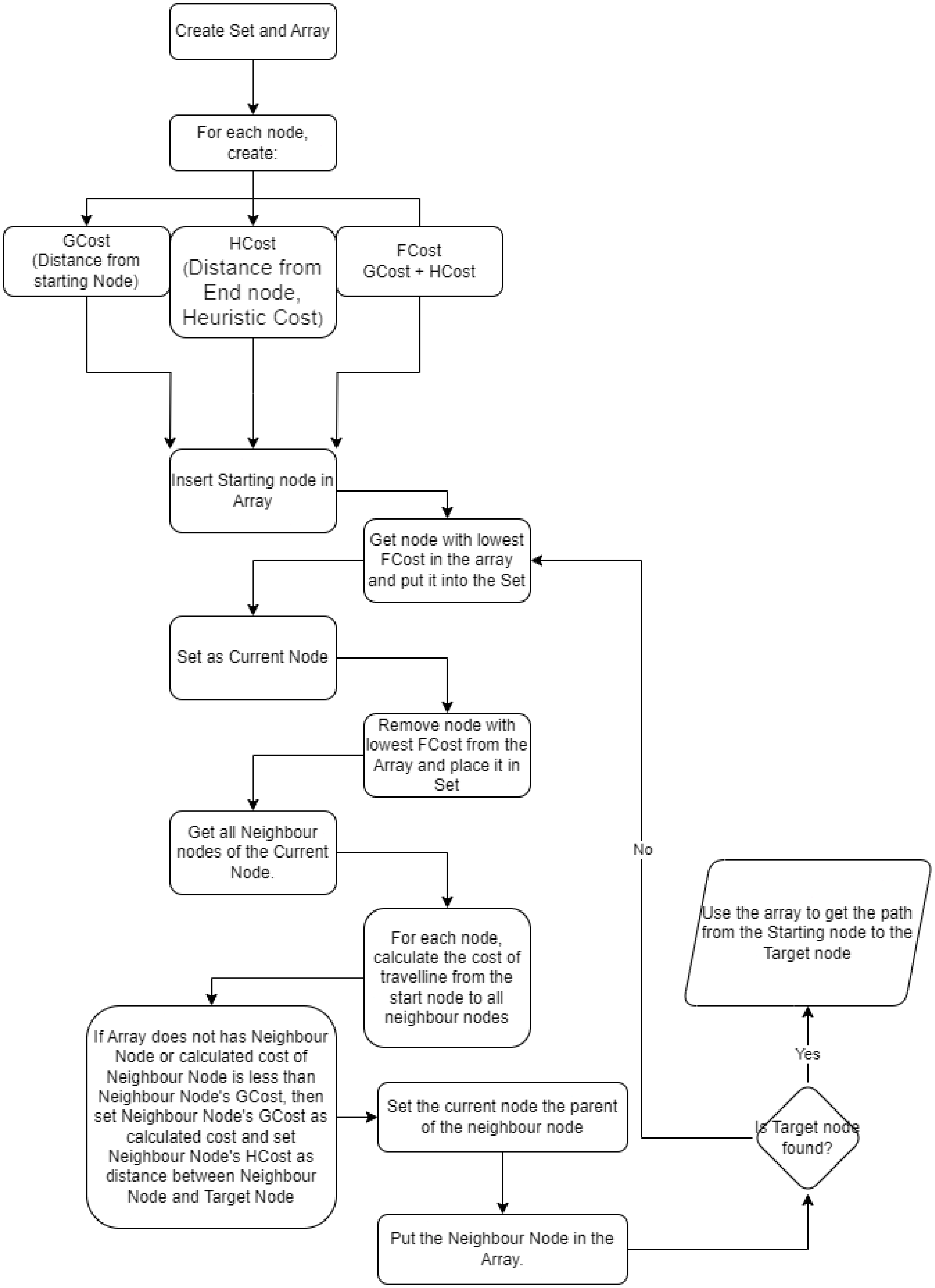


Create list for visited nodes
Create queue for nodes to search
Choose starting point
Does node have unvisited neighbours?
Yes
No
Add all unvisited neighbours to queue
Set "previous node" variable to the current node for all nodes added to the queue
Mark current node as visited
Move to next node in queue
Is current node the goal node?
No
Yes
Backtrack through "previous node" variables of the path and output as a list.

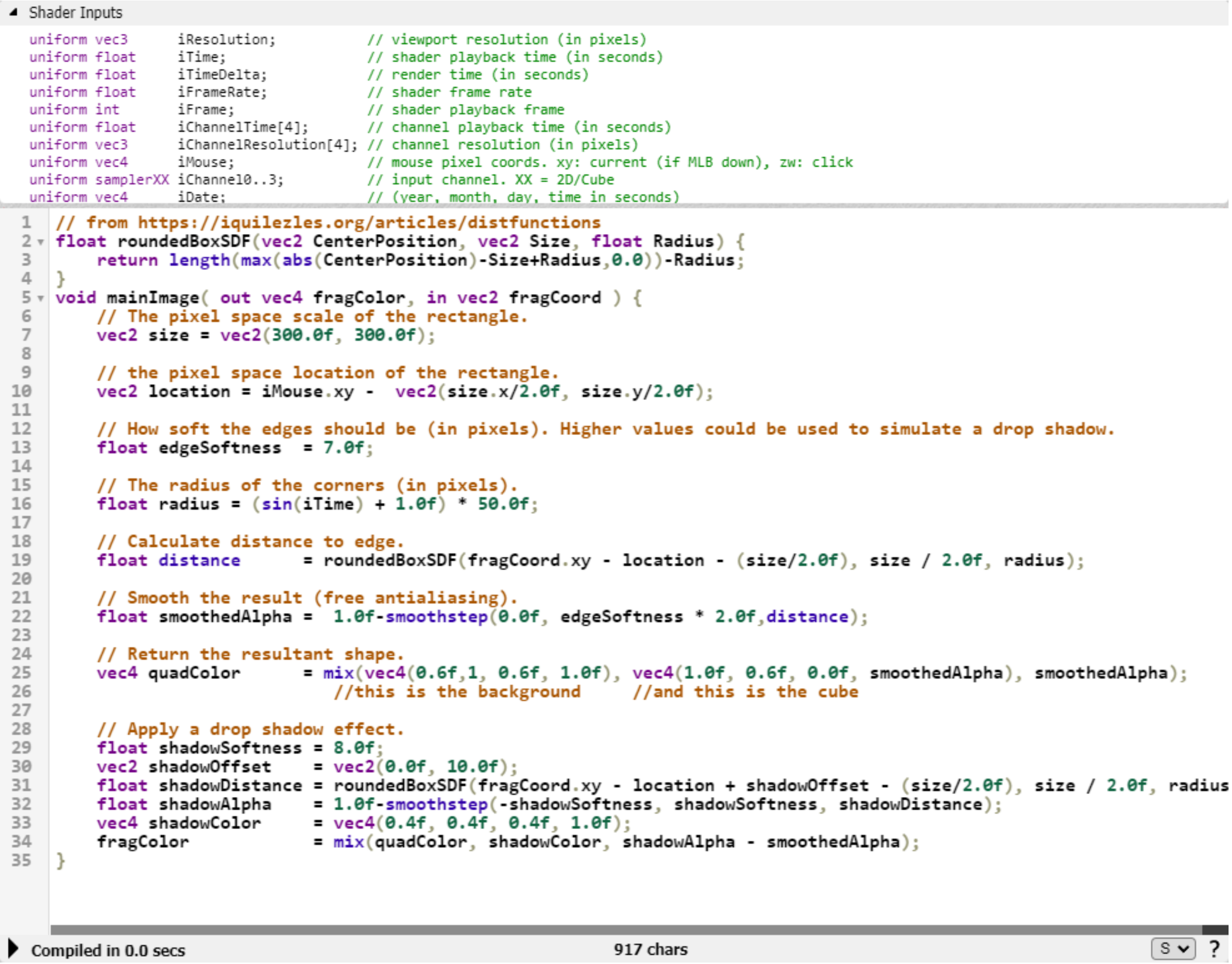
Depth-First Search


Create Array and Stack
Put Starting Node in the Stack
Mark Starting Node as Visited
Add Starting Node to Array
Go to Neighbouring node and add it to the stack, and mark it as visited
Is this Target Node?
Yes
Outputting Array stores the path from Starting node to Target node.
No
Are there any more neighbouring nodes?
Yes
No
Move to the node at the top of the Stack
Are there Neighbouring nodes?
Yes
Remove this node from the stack
No

A* Pathfinding


Create Set and Array
For each node, create:
GCost (Distance from starting Node)
HCost (Distance from End node, Heuristic Cost)
FCost GCost + HCost
Insert Starting node in Array
Get node with lowest FCost in the array and put it into the Set
Set as Current Node
Remove node with lowest FCost from the Array and place it in Set
Get all Neighbour nodes of the Current Node.
For each node, calculate the cost of travelline from the start node to all neighbour nodes
If Array does not has Neighbour Node or calculated cost of Neighbour Node is less than Neighbour Node's GCost, then set Neighbour Node's GCost as calculated cost and set Neighbour Node's HCost as distance between Neighbour Node and Target Node
Set the current node the parent of the neighbour node
Put the Neighbour Node in the Array.
Is Target node found?
Yes
No
Use the array to get the path from the Starting node to the Target node

Task 5: Rendering

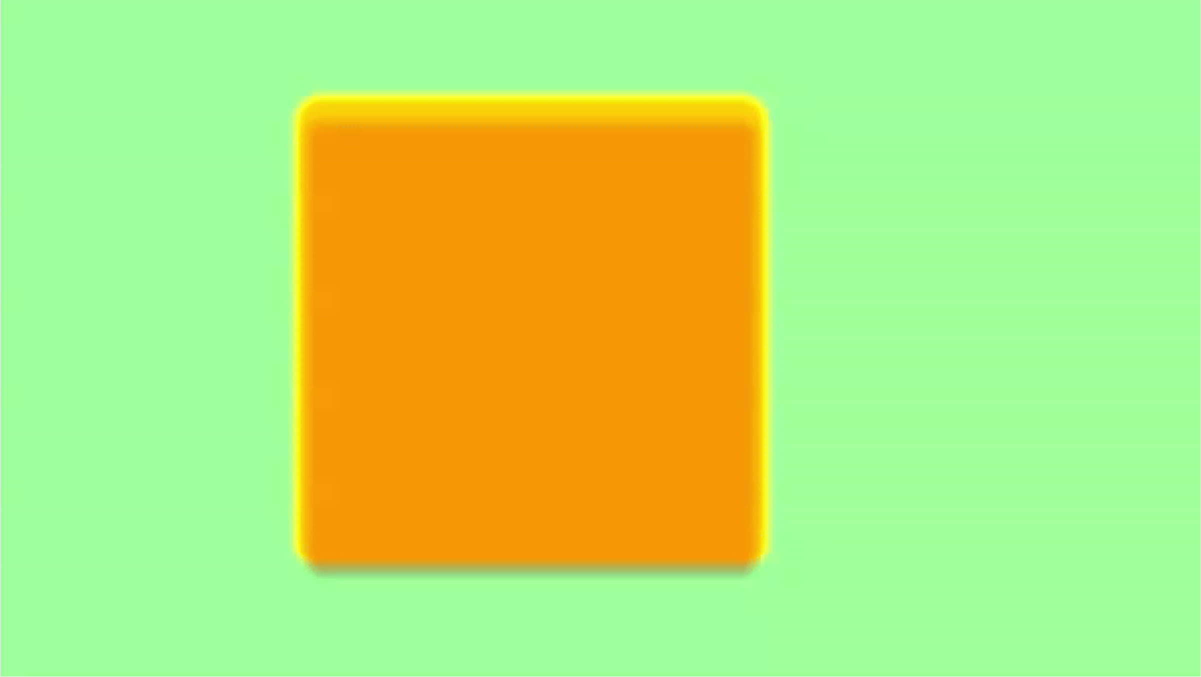


```
uniform vec3      iResolution;           // viewport resolution (in pixels)
uniform float     iTime;                 // shader playback time (in seconds)
uniform float     iTimeDelta;            // render time (in seconds)
uniform float     iFrameRate;            // shader frame rate
uniform int       iFrame;                // shader playback frame
uniform float     iChannelTime[4];       // channel playback time (in seconds)
uniform vec3      iChannelResolution[4]; // channel resolution (in pixels)
uniform vec4      iMouse;                // mouse pixel coords. xy: current (if MLB down), zw: click
uniform samplerXX iChannel0..3;          // input channel. XX = 2D/Cube
uniform vec4      iDate;                 // (year, month, day, time in seconds)
```

```
// from https://iquilezles.org/articles/distfunctions
float roundedBoxSDF(vec2 CenterPosition, vec2 Size, float Radius) {
    return length(max(abs(CenterPosition)-Size+Radius,0.0))-Radius;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    // The pixel space scale of the rectangle.
    vec2 size = vec2(300.0f, 300.0f);

    // the pixel space location of the rectangle.
    vec2 location = iMouse.xy -  vec2(size.x/2.0f, size.y/2.0f);

    // How soft the edges should be (in pixels). Higher values could be used to simulate a drop shadow.
    float edgeSoftness  = 7.0f;

    // The radius of the corners (in pixels).
    float radius = (sin(iTime) + 1.0f) * 50.0f;

    // Calculate distance to edge.
    float distance       = roundedBoxSDF(fragCoord.xy - location - (size/2.0f), size / 2.0f, radius);

    // Smooth the result (free antialiasing).
    float smoothedAlpha =  1.0f-smoothstep(0.0f, edgeSoftness * 2.0f,distance);

    // Return the resultant shape.
    vec4 quadColor       = mix(vec4(0.6f,1, 0.6f, 1.0f), vec4(1.0f, 0.6f, 0.0f, smoothedAlpha), smoothedAlpha);
                          //this is the background     //and this is the cube

    // Apply a drop shadow effect.
    float shadowSoftness = 8.0f;
    vec2 shadowOffset    = vec2(0.0f, 10.0f);
    float shadowDistance = roundedBoxSDF(fragCoord.xy - location + shadowOffset - (size/2.0f), size / 2.0f, radius
    float shadowAlpha    = 1.0f-smoothstep(-shadowSoftness, shadowSoftness, shadowDistance);
    vec4 shadowColor     = vec4(0.4f, 0.4f, 0.4f, 1.0f);
    fragColor            = mix(quadColor, shadowColor, shadowAlpha - smoothedAlpha);
}
```

Compiled in 0.0 secs

917 chars



```
vec2 location = iMouse.xy -  vec2(size.x/2.0f, size.y/2.0f);
```

I used the size variable to calculate the center of the square using the width and height, and offset the position by that. This means that the center of the square follows the mouse, instead of the bottom left corner.

```
   float edgeSoftness  = 7.0f;
```

increasing the edge softness made the sides of the cube much more blurred. This is likely to be a sort of antialiasing, as if you set it to 0, the edges look very jagged.

```
   float radius = (sin(iTime) + 1.0f) * 50.0f;
```

The radius of the corners change over time, and this is what makes that happen. I increased the multiplier to 90, and the square became much more round. If you increase this multiplier too high, the cube starts bending inwards.

```
   vec4 quadColor                = mix(vec4(0.6f,1, 0.6f, 1.0f), vec4(1.0f, 0.6f, 0.0f, 
smoothedAlpha), smoothedAlpha);
                    //this is the background     //and this is the cube
```

I switched around the colours of the background and the cube, and I think the mix of the drop shadow effect and the edge softness made it have a yellow border which makes it look almost like an app icon.