DES310 Portfolio

Ollie Hall 1700066 - Programmer

'***Uncovered***' by Red Ink

Team 10 – Society of Antiquaries of Scotland

Uncovered was a fascinating project in collaboration with the Society of Antiquaries of Scotland. I worked on *Programming*, *Animation*, *UI Implementation* and contributed towards some *3D Modelling*. I also produced the Teaser Trailer.

This project enabled me to develop my experience in *C#* and *Unity* as well as learning new programming skills, working effectively within a team, helping to manage a project and all the complex elements it involved.

Contents

Research ... 4
Project Work ... 8
Concept Development ... 8
Prototyping ... 10
Player Movement Prototyping ... 10
Artifact Interaction System Prototyping ... 11
Development Work ... 13
Player Movement ... 13
Physics vs Translation ... 13
Dealing with Verticality ... 13
The CharacterController component ... 15
Dealing with gravity ... 16
Artifact Interaction ... 17
Controls ... 17
Overview of system ... 17
Finding an artifact ... 18
Entering Interaction ... 19
Rotating an artifact ... 20
Points of Interest ... 22
The Drone ... 23
Drone Movement ... 23
The Drone Controller ... 24
Drone Signal ... 25
Artifact Detection ... 26
Drone Audio ... 27
Animating the Tools ... 28
The Specialist Page ... 30
Changing the Design ... 30
Functionality ... 31
The Journal ... 32
Creating the Journal ... 32
The Tab System ... 32
The Inventory Page ... 33

The Information Page 34
The Site Page 35
Implementing settings into the Settings Page 36
Game Mode System 36
The Popup System 37
Types of Popups 37
The Queue System 40
Pausing the Game 41
Altering the Slab System 42
Improving the Top Layer 42
Highlighting Slabs 43
Digging Particles 43
Dig Progress Indicator 43
Blender Rocks 44
Main Menu 44
Improving the Map 45
Occlusion Culling and Lighting 46
Creation and Modification of terrain 46
AudioManager 47
The Trailer 47
References 48

Research

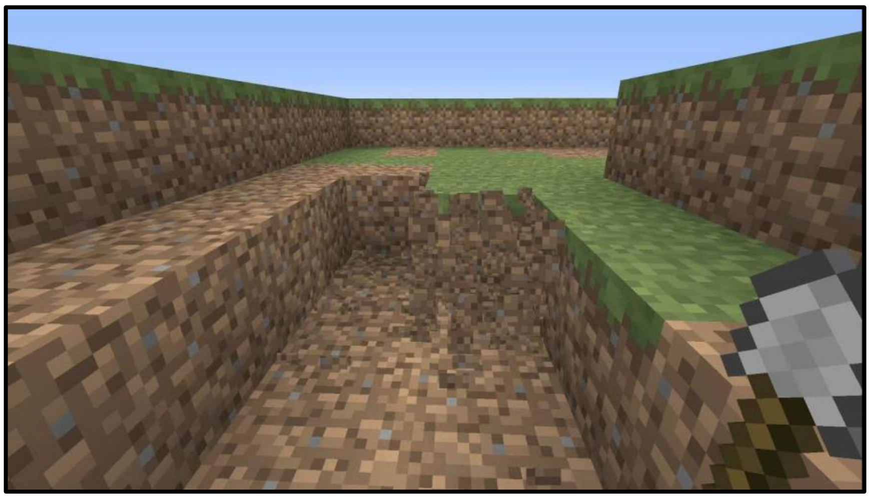
As our client has no experience in the gaming industry, we could not reference them for our work, but instead used our own inspiration combined with their education to create an archaeological experience which is both **educational** and **gamified**.

We kept in communication with the correspondent from the Society of Antiquaries of Scotland, Jeff Sanders, to ensure that our game was accurate enough to mirror real archaeology and that it would showcase the best of the sector. We were made aware of some of the stigmas attached to the archaeology community such as that of "*treasure hunting*" spawned form media such as Indiana Jones or Tomb Raider.

After discussing the platform and demographic for our game, the team concluded that we were going to make a **PC** game for students between the age of **12 and 20** who have an established interest in archaeology. We initially were open to providing controller support for the game so that the player could use a gamepad to play, but as the UI systems built up, we realised that it wouldn't be feasible for our scope of the game and the time we had.

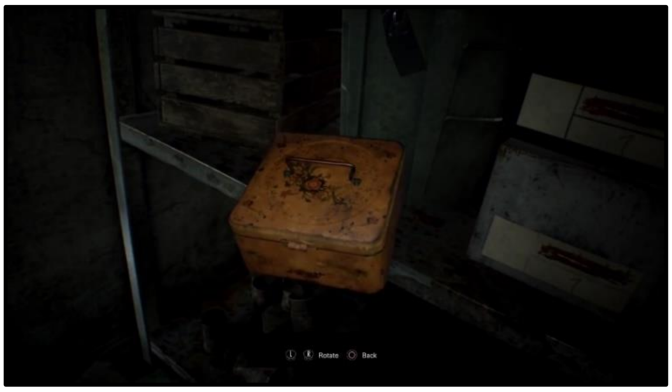
Although the idea for the game was rather unique, the individual mechanics that make it up were heavily inspired from other games:

Within our game, the player can use a trowel or a shovel to dig up dirt to uncover artifacts. To achieve this, Ben came up with the idea of a grid of "**slabs**" that could be generated, and the player could dig these slabs to deform the terrain and find objects underground. This was inspired from *Minecraft's* its iconic digging mechanic.

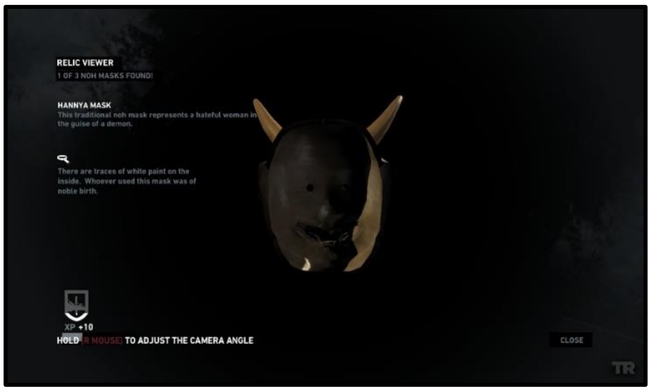

Minecraft (2011) - Digging dirt blocks with a shovel

Digging was always going to be at the centre of our game – the team took a trip to Edinburgh to speak to John Lawson, an Archaeology Officer, to discuss and learn the dig process, precautions and practices. When doing research for animating the tools, I looked up videos of archaeological development using trowels and shovels.

The "**Artifact Interaction Mode**" within our game is heavily inspired from other games – the player can look at an artifact and pick it up, rotating it to examine it and uncover details about the object. The visuals of this mode were based on that of *Resident Evil 7 (2017)*. The "Point of Interest" system where you uncover information is inspired from the "*Relic Viewer*" in *Tomb Raider (2013)*, where the player must rotate artifacts to uncover secrets.



Object inspection in Resident Evil 7 (2017)

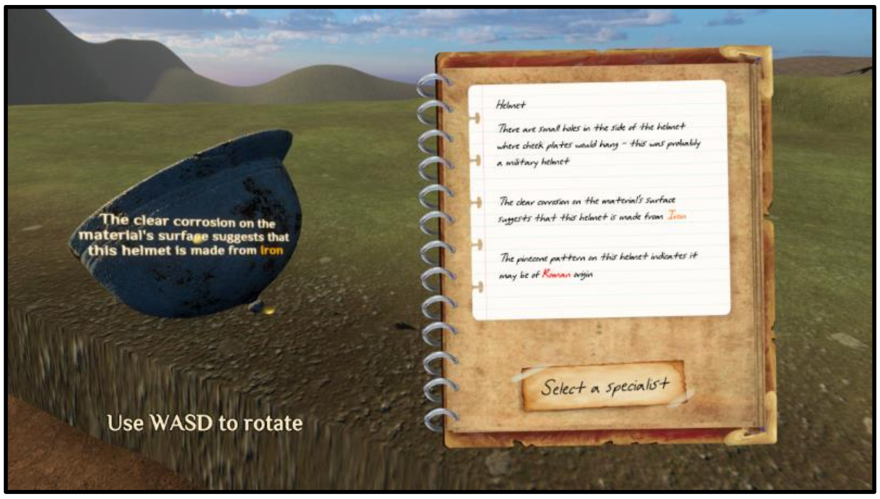

RELIC VIEWER
1 OF 3 NOH MASKS FOUND!

HANNYA MASK
This traditional noh mask represents a hateful woman in the guise of a demon.

There are traces of white paint on the inside. Whoever used this mask was of noble birth.

XP +10
HOLD [R MOUSE] TO ADJUST THE CAMERA ANGLE

The Relic Viewer in Tomb Raider (2013)


The clear corrosion on the material's surface suggests that this helmet is made from iron

Use WASD to rotate

Helmet

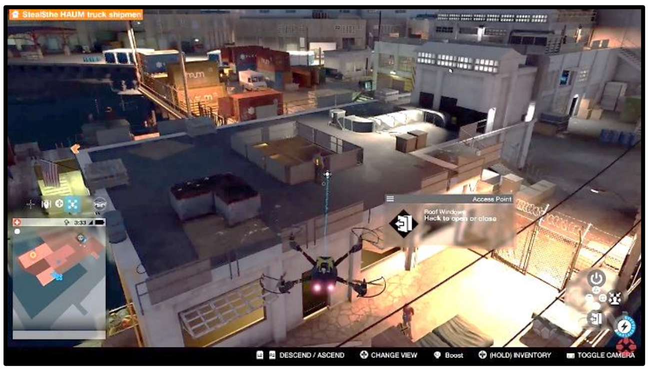
There are small holes in the side of the helmet where cheek plates would hang – this was probably a military helmet

The clear corrosion on the material's surface suggests that this helmet is made from Iron

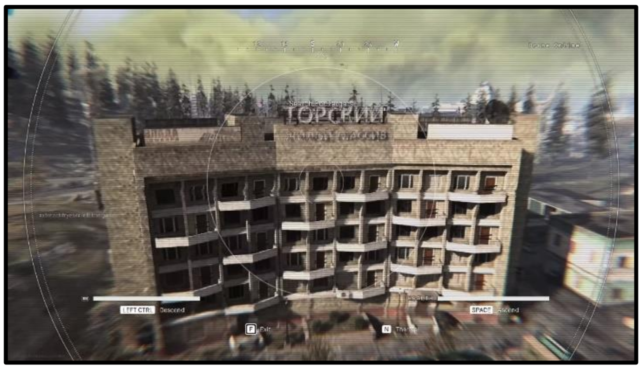
The pinecone pattern on this helmet indicates it may be of Roman origin

The Artifact Interaction System in the final build of Uncovered

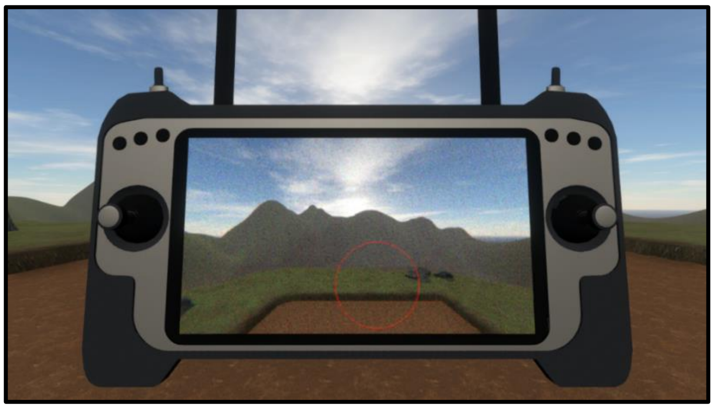
The **Drone** mechanic is an important part of our game, and tackling the idea of the drone's controls, movement and purpose were made easier by games that have already achieved great Drone systems, such as *Watch Dogs 2 (2016)* and *Call of Duty: Warzone (2020*) which both have the player control the drone to survey their surroundings. The controls for the drone were based on the latter game, as I found the movement to be satisfying and simple. Other aspects of the Drone, such as signal range were based on *Watch Dogs 2 (2016)*.



The Quadcopter in Watch Dogs 2 (2016)



The drone in Call of Duty: Warzone (2020)



The Drone Mechanic in the final build of Uncovered

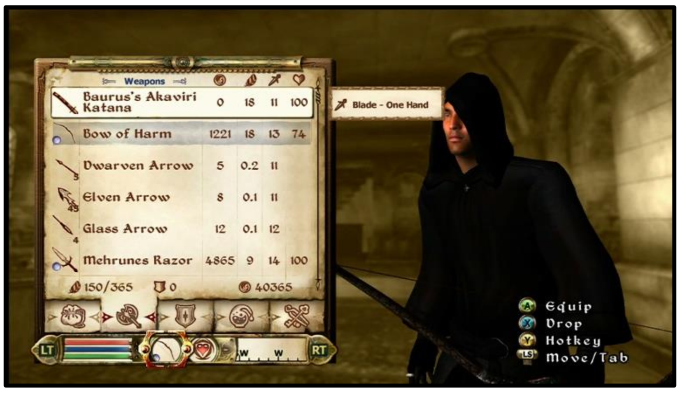
Creating the Player Movement system was one thing, but making it feel somewhat realistic and smooth was another challenge. To solve this, I added head bobbing to the movement, and used the animation of the tools to represent *movement* and *weight*.

The feeling of head bobbing was inspired by that of *Kingdom Come: Deliverance* (2018), as the head bobbing is mostly horizontal, emphasising the left and right foot stepping in alternation.

Overwatch (2016) was a great inspiration for the animation of tools as it conveys character, weight and speed entirely through the animation of the weapons as the character moves.

Overwatch (2016) conveys character through the animation cycles of the characters' weapons

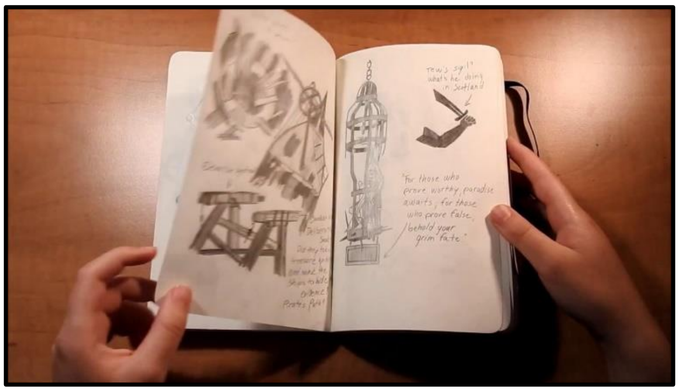
The team decided to make the majority of the game's User Interface diegetic – a Journal that the player writes notes to and can refer to. This adds to the immersive experience we set out to achieve. Notable inspirations for this idea come from *The Elder Scrolls IV: Oblivion (2006)* and *Uncharted 4: A Thief's End (2017)*.



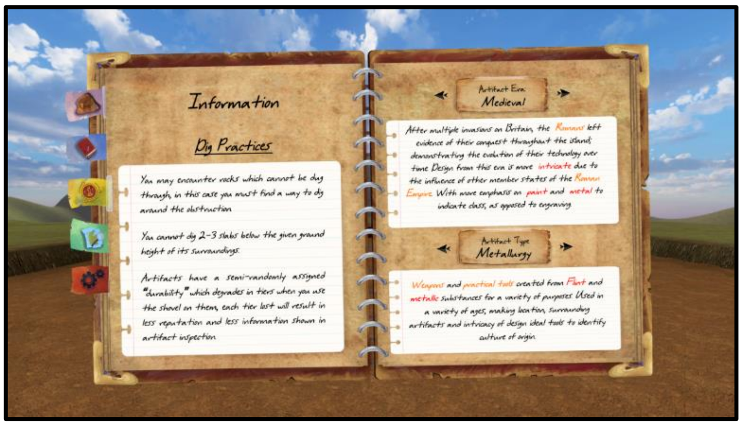
Weapons				
Baurus's Akaviri Katana	0	18	11	100
Bow of Harm	1221	18	13	74
Dwarven Arrow	5	0.2	11	
Elven Arrow	8	0.1	11	
Glass Arrow	12	0.1	12	
Mehrunes Razor	4865	9	14	100

150/365 0 40365

Inventory screen in The Elder Scrolls IV: Oblivion (2006)



The journal in Uncharted 4: A Thief's End



Information

Dig Practices

You may encounter rocks which cannot be dug through, in this case you must find a way to dig around the obstruction

You cannot dig 2-3 slabs below the given ground height of its surroundings.

Artifacts have a semi-randomly assigned "durability" which degrades in tiers when you use the shovel on them, each tier lost will result in less reputation and less information shown in artifact inspection

Artifact Era:
Medieval

After multiple invasions on Britain, the Romans left evidence of their conquest throughout the island; demonstrating the evolution of their technology over time Design from this era is more intricate due to the influence of other member states of the Roman Empire. With more emphasis on paint and metal to indicate class, as opposed to engraving.

Artifact Type
Metallurgy

Weapons and practical tools created from Flint and metallic substances for a variety of purposes. Used in a variety of ages, making location, surrounding artifacts and intricacy of design ideal tools to identify culture of origin.

The Information page of the Journal in Uncovered

I used the YouTube channels of *Jason Weinnman* and *Sebastian Lague* to both inspire me and help me learn better practices in Unity and C#.

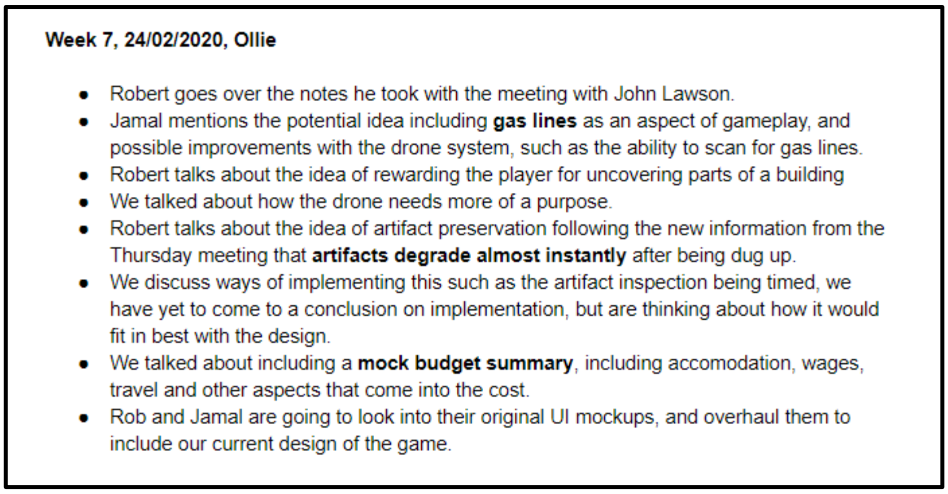
Project Work

Concept Development

In the first weeks as we were coming up with ideas for the game and prototyping them, I was communicating constantly with designers about their vision and ideas for the game to ensure I was on the same "mental wavelength" as them – I believed that as a programmer it was important that I do my best job to bring the team's vision into reality. This was also important as I had to plan the structure of the game's scripts and functions to suit the design of the game, and as programmers we had to ensure that the idea was feasible for us to work on at our current level of skill and experience.

We used methods to ensure our ideas were consolidated and unanimous within the team by we noted down our trains of thought for specific mechanics such as the *Dig Site Generation System* and even the entire game loop. After writing this down, we would discuss what we had written and point out any discrepancies. This method even resulted in improved designs from people mistaking parts of the mechanic, with the mistake being a better idea. This was overall a very cooperative and production method of ensuring the team were comfortable with the ideas and designs of the game we were working to create.

Every week a different member of the team wrote meeting minutes . We did this for the entire duration of development, meaning we have a clear document containing what each team member was working on at the time, and our discussions for changes, additions and improvements to the game.

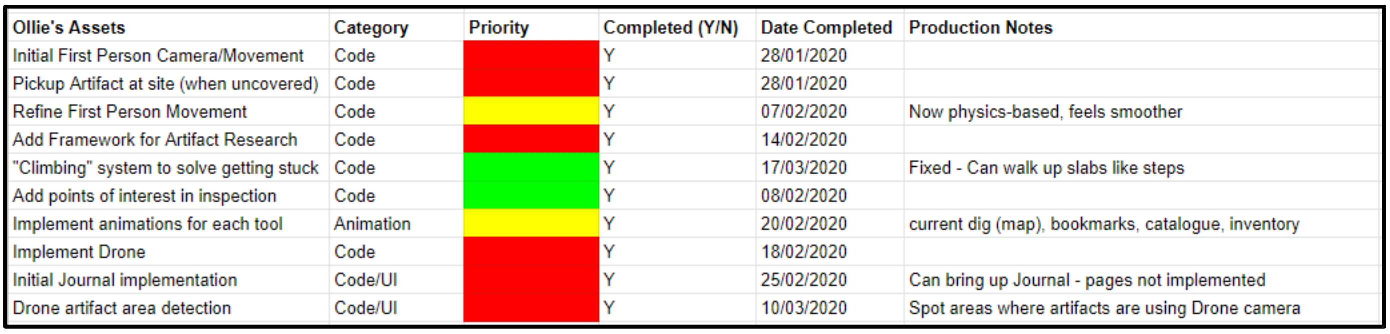


Week 7, 24/02/2020, Ollie

- Robert goes over the notes he took with the meeting with John Lawson.
- Jamal mentions the potential idea including **gas lines** as an aspect of gameplay, and possible improvements with the drone system, such as the ability to scan for gas lines.
- Robert talks about the idea of rewarding the player for uncovering parts of a building
- We talked about how the drone needs more of a purpose.
- Robert talks about the idea of artifact preservation following the new information from the Thursday meeting that **artifacts degrade almost instantly** after being dug up.
- We discuss ways of implementing this such as the artifact inspection being timed, we have yet to come to a conclusion on implementation, but are thinking about how it would fit in best with the design.
- We talked about including a **mock budget summary**, including accomodation, wages, travel and other aspects that come into the cost.
- Rob and Jamal are going to look into their original UI mockups, and overhaul them to include our current design of the game.

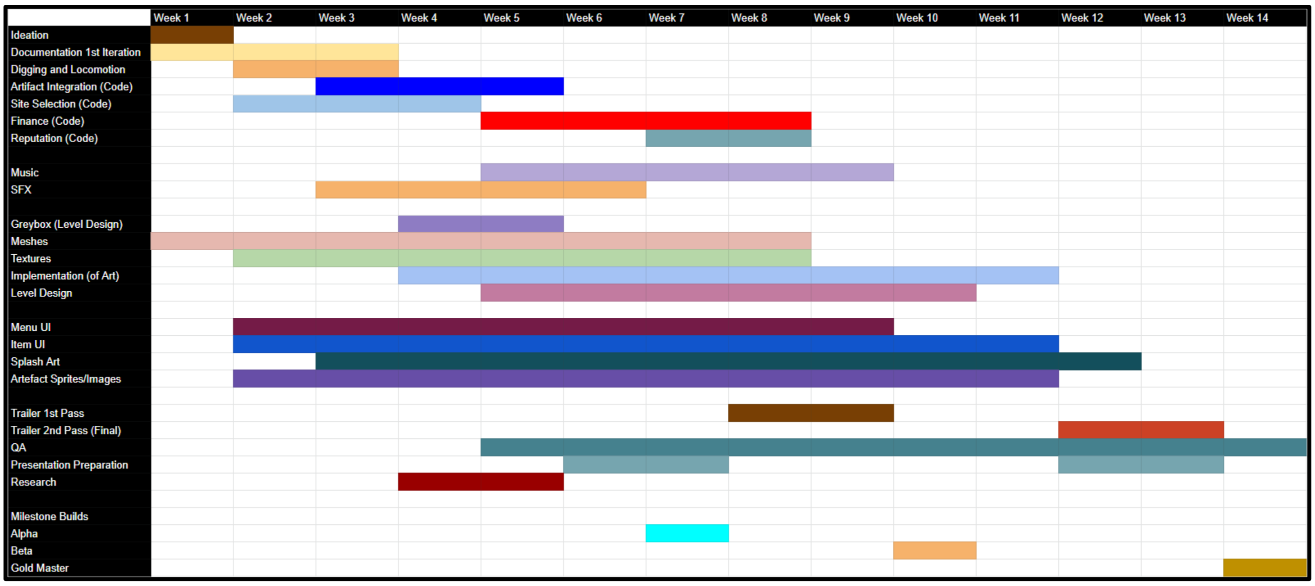
A snippet of one of the meeting minutes I wrote midway through development

On top of this we had an asset list that contained the assets we needed to make, their priority (required for Minimum Viable Product, Surplus to Minimum Viable Product, and Stretch Goals), the date of completion and production notes. This was important so that each member of the team could keep track of each other to ensure a smooth development process.



Ollie's Assets	Category	Priority	Completed (Y/N)	Date Completed	Production Notes
Initial First Person Camera/Movement	Code	🟥	Y	28/01/2020	
Pickup Artifact at site (when uncovered)	Code	🟥	Y	28/01/2020	
Refine First Person Movement	Code	🟨	Y	07/02/2020	Now physics-based, feels smoother
Add Framework for Artifact Research	Code	🟥	Y	14/02/2020	
"Climbing" system to solve getting stuck	Code	🟩	Y	17/03/2020	Fixed - Can walk up slabs like steps
Add points of interest in inspection	Code	🟩	Y	08/02/2020	
Implement animations for each tool	Animation	🟨	Y	20/02/2020	current dig (map), bookmarks, catalogue, inventory
Implement Drone	Code	🟥	Y	18/02/2020	
Initial Journal implementation	Code/UI	🟥	Y	25/02/2020	Can bring up Journal - pages not implemented
Drone artifact area detection	Code/UI	🟥	Y	10/03/2020	Spot areas where artifacts are using Drone camera

My section of the Asset List – Assets were added as we were developing the game.



I kept to a production schedule created by the designers to ensure development was smooth and constant

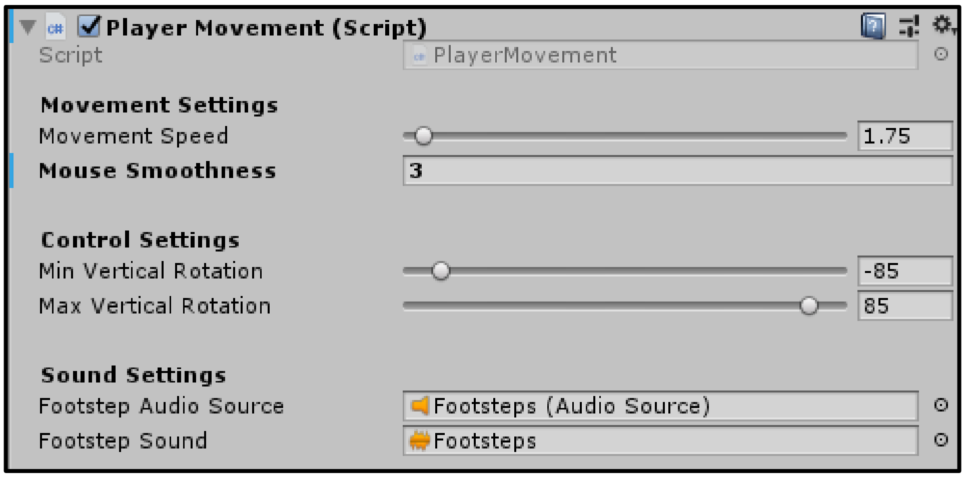
Prototyping

Player Movement Prototyping

I started my development on the game by prototyping the **Player Movement System** and **Artifact Interaction System**.

For the **Player Movement System**, I decided to avoid using Unity's pre-made assets and scripts from the "Standard Assets" package, as I wanted to make a custom First-Person Controller that I could fully understand and tweak to fit the game. Mechanics such as jumping, sprinting and crouching were never intended to be in the game, so a custom script could be as simplified as I needed. The system went through many iterations and changes which I will cover in development work, but it was important that for the basis of the game the movement felt freeing, smooth and controlled.

I designed the **Player Movement** script so that it could be easily accessed by the designers, and each relevant value could be tweaked within a clamped range:

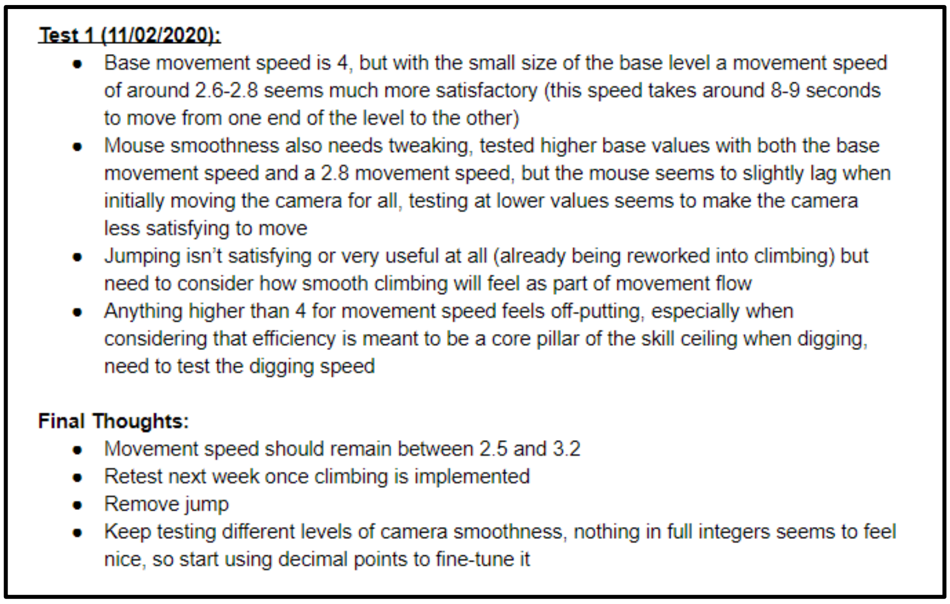


Player Movement (Script)

Script	PlayerMovement
Movement Settings	
Movement Speed	1.75
Mouse Smoothness	3
Control Settings	
Min Vertical Rotation	-85
Max Vertical Rotation	85
Sound Settings	
Footstep Audio Source	Footsteps (Audio Source)
Footstep Sound	Footsteps

The Player Movement script in the Unity Editor inspector tab.

The designers used this inspector to investigate the Player Movement to find the best values possible, and recorded them in a document:



Test 1 (11/02/2020):

- Base movement speed is 4, but with the small size of the base level a movement speed of around 2.6-2.8 seems much more satisfactory (this speed takes around 8-9 seconds to move from one end of the level to the other)
- Mouse smoothness also needs tweaking, tested higher base values with both the base movement speed and a 2.8 movement speed, but the mouse seems to slightly lag when initially moving the camera for all, testing at lower values seems to make the camera less satisfying to move
- Jumping isn't satisfying or very useful at all (already being reworked into climbing) but need to consider how smooth climbing will feel as part of movement flow
- Anything higher than 4 for movement speed feels off-putting, especially when considering that efficiency is meant to be a core pillar of the skill ceiling when digging, need to test the digging speed

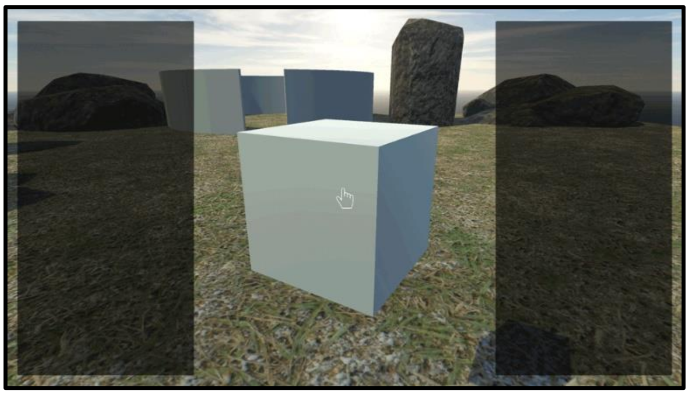
Final Thoughts:

- Movement speed should remain between 2.5 and 3.2
- Retest next week once climbing is implemented
- Remove jump
- Keep testing different levels of camera smoothness, nothing in full integers seems to feel nice, so start using decimal points to fine-tune it

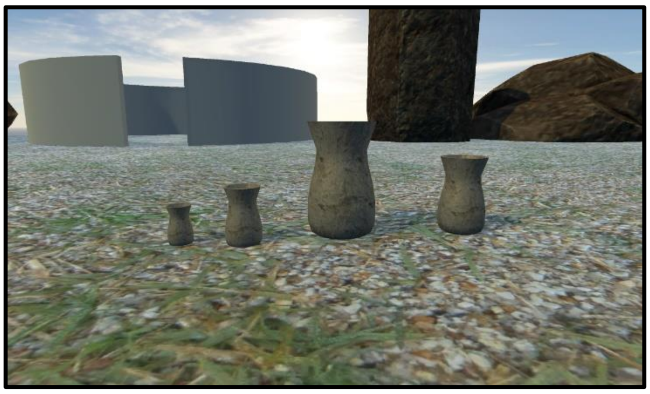
The designers' notes on the Player Movement System. I took this to heart and used it as important feedback for future iterations of the system.

Artifact Interaction System Prototyping

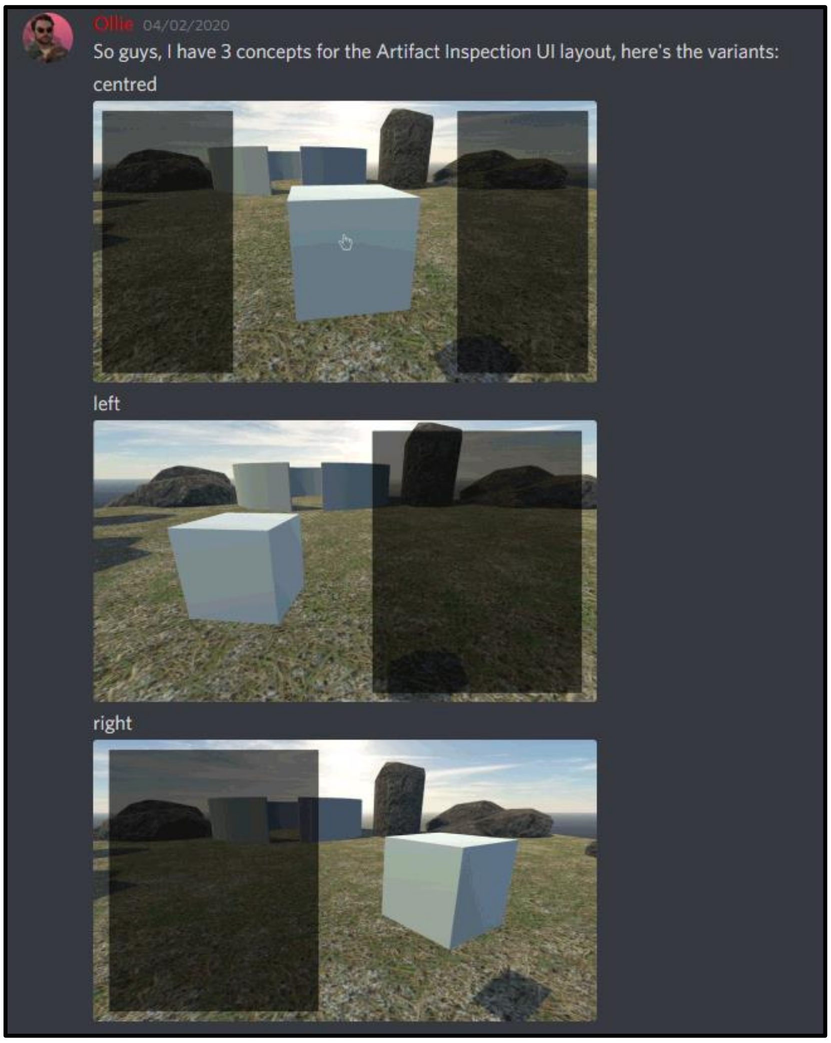
The **Artifact Interaction System** was designed for the player to have the option to view the artifact up-close when they have found it – this mechanic went through a few design changes that I suggested and pushed to be included, such as the **Point of Interest System**. It was very important that when the player finds an artifact, their hard work is paid off with a satisfying and in-depth look at the artifact which is both visually pleasing and educational.



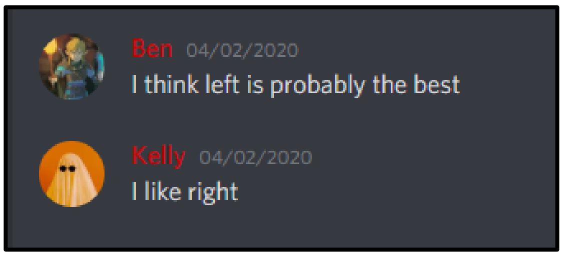
An early prototype of the Artifact Interaction System.



Testing multiple sizes of artifacts to work with the system



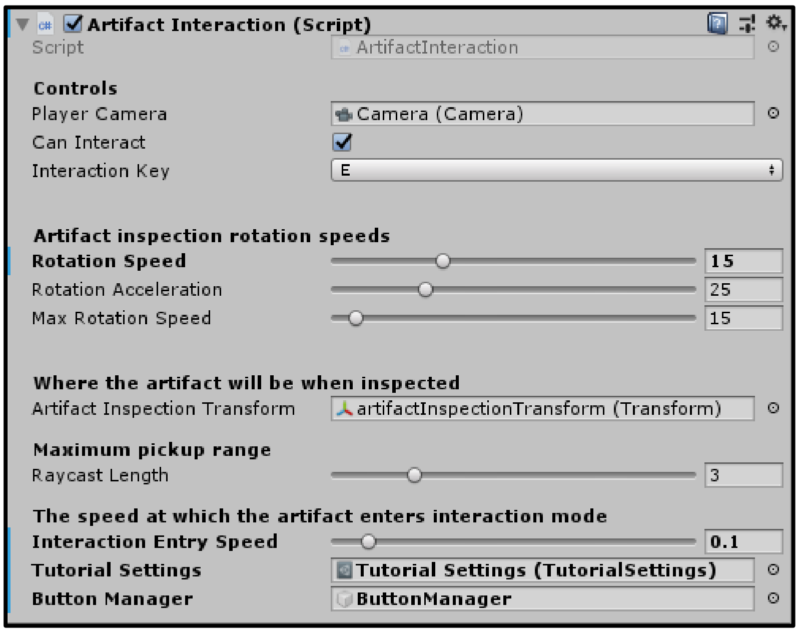
Ollie 04/02/2020
So guys, I have 3 concepts for the Artifact Inspection UI layout, here's the variants:
centred
left
right



Ben 04/02/2020
I think left is probably the best

Kelly 04/02/2020
I like right

Receiving feedback from members of the team on multiple variants of UI to determine which looks best



Artifact Interaction (Script)

Script	ArtifactInteraction
Controls	
Player Camera	Camera (Camera)
Can Interact	✓
Interaction Key	E
Artifact inspection rotation speeds	
Rotation Speed	**15**
Rotation Acceleration	25
Max Rotation Speed	15
Where the artifact will be when inspected	
Artifact Inspection Transform	artifactInspectionTransform (Transform)
Maximum pickup range	
Raycast Length	3
The speed at which the artifact enters interaction mode	
Interaction Entry Speed	**0.1**
Tutorial Settings	**Tutorial Settings (TutorialSettings)**
Button Manager	**ButtonManager**

The Artifact Interaction script in the Unity Editor. I ensured that designers could modify the appropriate values

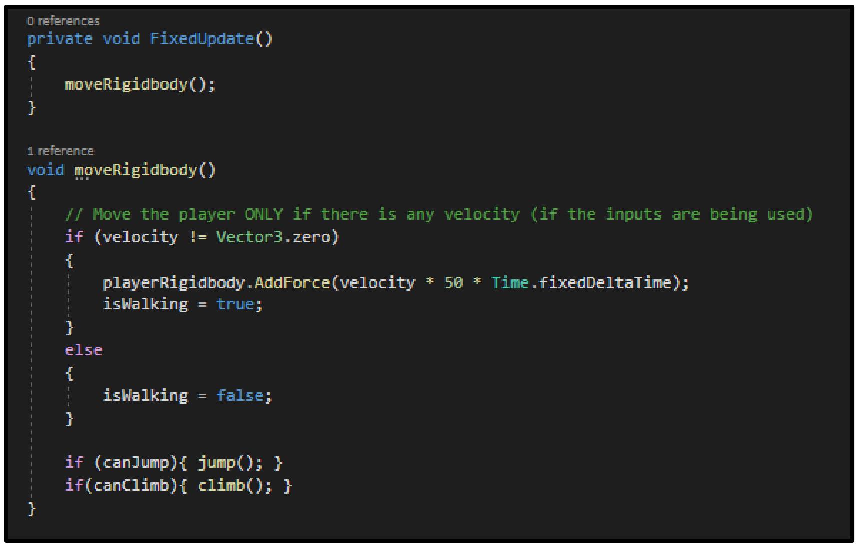
Development Work

Player Movement

The player can't achieve anything without being able to move within the environment, so the **Player Movement System** was the first mechanic I worked on. It was important for this mechanic to work as well as possible, as it is the foundation for the rest of the game.

Physics vs Translation

Initially I created the script to take in the W, A, S and D keys and depending on the key, a force would be applied to a Rigidbody component that is attached to the player. For a long time, this was used for the player movement, but using physics is both more *computationally expensive* and unreliable. When the forces clash with mathematical movement, the two movements clash and create a noticeable stutter when the player is moving. I believed that using physics to move the player would make the movement feel smoother, but due to these issues, it turned out to be the opposite.



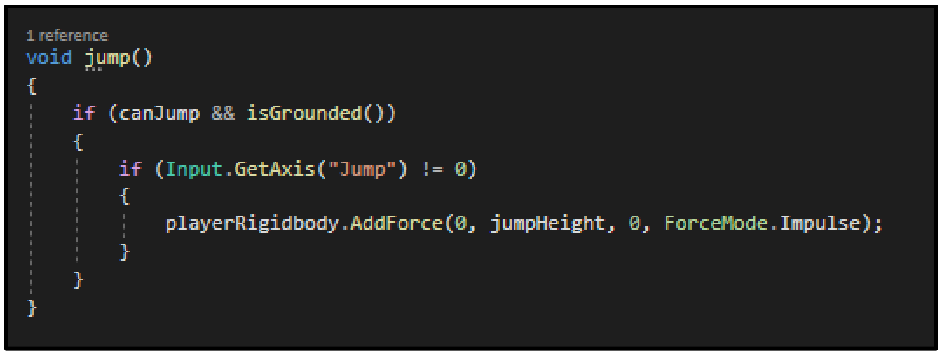
```
0 references
private void FixedUpdate()
{
    moveRigidbody();
}

1 reference
void moveRigidbody()
{
    // Move the player ONLY if there is any velocity (if the inputs are being used)
    if (velocity != Vector3.zero)
    {
        playerRigidbody.AddForce(velocity * 50 * Time.fixedDeltaTime);
        isWalking = true;
    }
    else
    {
        isWalking = false;
    }

    if (canJump){ jump(); }
    if(canClimb){ climb(); }
}
```

An early iteration of the PlayerMovement script. FixedUpdate() and physics forced were used.

Dealing with Verticality

One of the hardest challenges to deal with when developing the Player Movement System was how it dealt with the slab system and getting stuck in holes. Walking around the level was fine until the player began digging. If they walked into a hole, there would be no way to exit it. At this point the Player Movement System was not developed to deal with any form of verticality – an oversight on my behalf. There were multiple ways I investigated fixing this.

As a temporary workaround for being able to leave holes I added **jumping** to the Player Movement System. The player could press the *Spacebar* to add a force based impulse upwards. This was implemented while the system was still using physics-based movement.



```
1 reference
void jump()
{
    if (canJump && isGrounded())
    {
        if (Input.GetAxis("Jump") != 0)
        {
            playerRigidbody.AddForce(0, jumpHeight, 0, ForceMode.Impulse);
        }
    }
}
```

The code for jumping in the player movement script. This was eventually replaced

This method of escaping holes dug in the site was not at all satisfying and felt forced and out of place. It was useful for testing the game in development as there was no other way to escape holes.

My next experiment to solve the climbing issue was creating a function that detects a slab in front of the player's feet and uses the Unity Lerp function to transport the player to above that slab. The idea was that this would allow the player to "climb" slabs as if they were using a ladder – pressing the jump button would transport them to the ground level. This method quickly introduced many issues – it would not always correctly detect the top slab, resulting in the game trying to pull the player into the area where the slabs are.

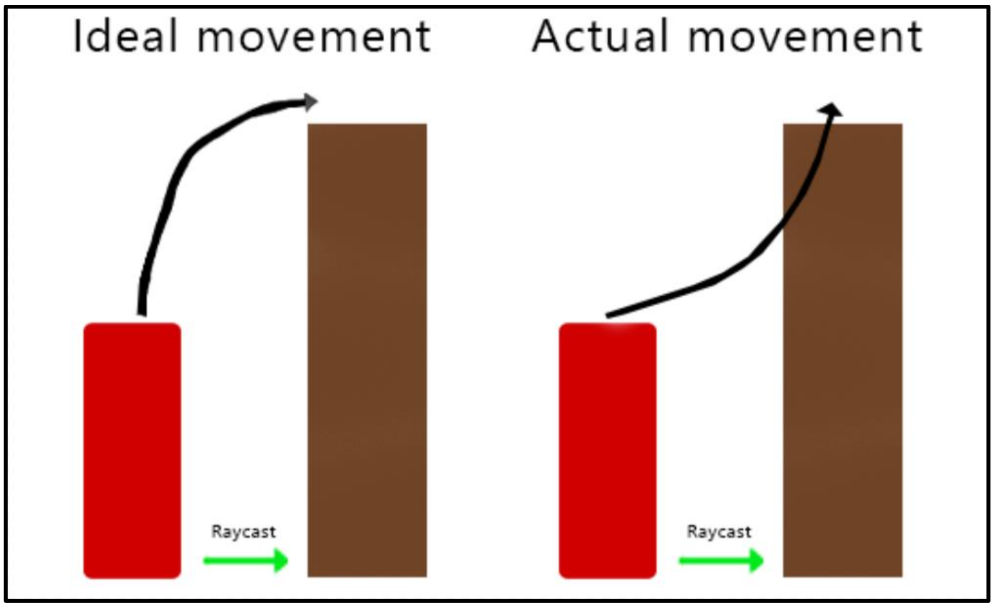
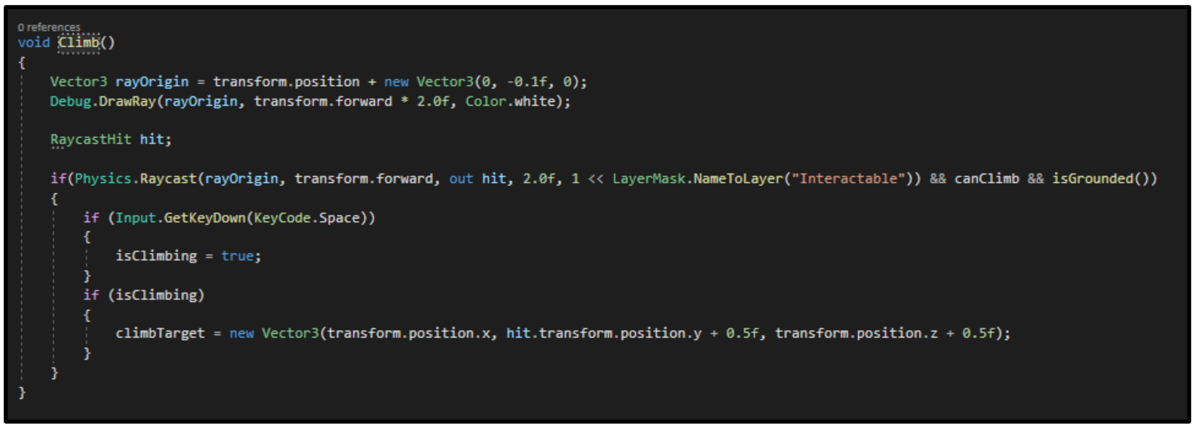


Diagram representing how the 'climbing' function should ideally perform vs how it's performed.



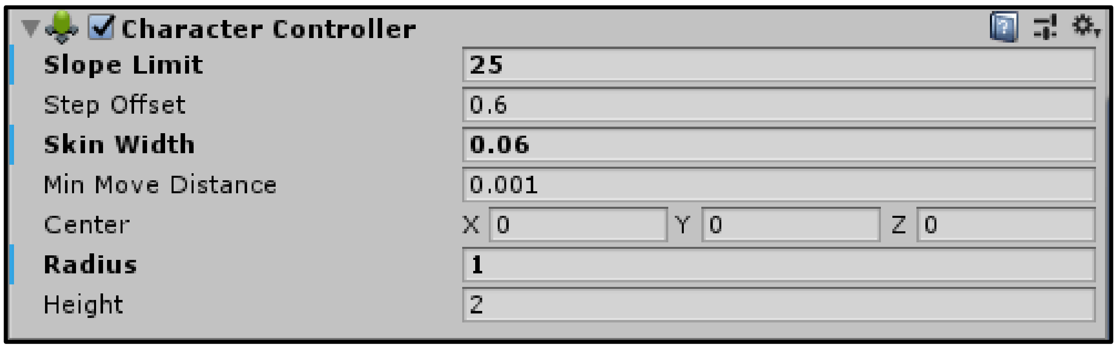
```
0 references
void Climb()
{
    Vector3 rayOrigin = transform.position + new Vector3(0, -0.1f, 0);
    Debug.DrawRay(rayOrigin, transform.forward * 2.0f, Color.white);

    RaycastHit hit;

    if(Physics.Raycast(rayOrigin, transform.forward, out hit, 2.0f, 1 << LayerMask.NameToLayer("Interactable")) && canClimb && isGrounded())
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            isClimbing = true;
        }
        if (isClimbing)
        {
            climbTarget = new Vector3(transform.position.x, hit.transform.position.y + 0.5f, transform.position.z + 0.5f);
        }
    }
}
```

The function that I implemented for "climbing" slabs.

Replacing the 'Lerp' with a simple translation in position technically worked but the player snapping to the correct position was disorienting and broke the immersive feeling of the game. This attempt at solving the issue created many more problems than it resolved so I decided the discontinue development and experimentation with this function.

The CharacterController component

I decided to test out the Unity inbuilt component, the **CharacterController**. I chose this option as a last resort as I would have much preferred to create my own climbing system that I could fully understand, tweak and experiment with, but in the end, it solved the climbing issue. I decided to use the CharacterController component as it supported smooth player movement but most importantly it had built in functionality for supporting slopes and verticality. After implementing the CharacterController and tweaking it appropriately, it worked fantastically with the slab system, with the player being able to walk up and down the slabs as if they were steps (Player can climb 5 slabs at a time). The CharacterController Move() function was now how the PlayerMovement script moved the player.



Character Controller

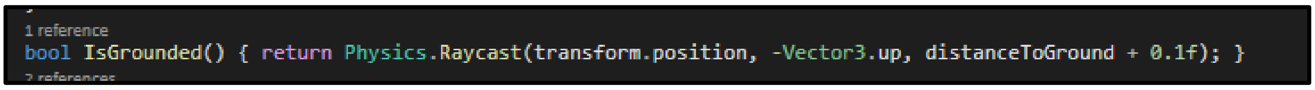
Slope Limit	**25**
Step Offset	0.6
Skin Width	**0.06**
Min Move Distance	0.001
Center	X 0 Y 0 Z 0
Radius	**1**
Height	2

The Character Controller component on the player.

Dealing with gravity

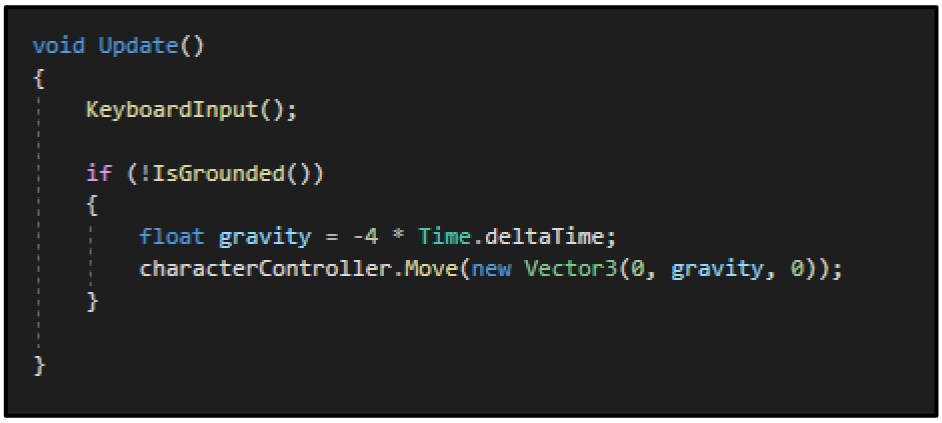
The fact that the player movement was not physics based meant that there was no gravity applied to the player. This meant that the player couldn't walk into the dug-up areas they had created. A function I created named **IsGrounded()** sends a raycast from where the player's foot would be, and this raycast detects if there is a collider directly below it. If a collider is detected, the function returns true, else false. Within the Update() function, if the IsGrounded() function returns false, a constant gravity is applied to the CharacterController in the y direction. There is no simulation of gravitational acceleration, but due to the game's level not being too vertical, it wasn't very noticeable.

*



```
1 reference
bool IsGrounded() { return Physics.Raycast(transform.position, -Vector3.up, distanceToGround + 0.1f); }
```

The IsGrounded() function.



```
void Update()
{
    KeyboardInput();

    if (!IsGrounded())
    {
        float gravity = -4 * Time.deltaTime;
        characterController.Move(new Vector3(0, gravity, 0));
    }

}
```

Gravity being applied to the player if they aren't grounded.

There were some issues with the player falling through the slabs and being stuck within them – this was due to the character controller searching for steps and the slab system confusing it. Tweaking the values seemed to iron this issue out for the most part but I believe it still occasionally happens when trying to be replicated.

Artifact Interaction

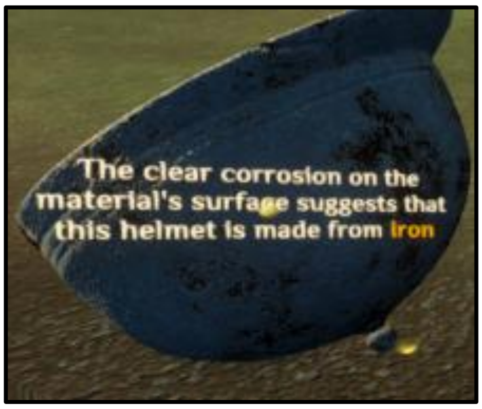
Resident Evil 7 (2017) and Tomb Raider (2013) were the main references I used when creating the **Artifact Interaction System**. The artifact had to come close enough to the screen for the player to be able to see it in full detail and appreciate the models created by our artists.

Controls

As a team we decided that this mechanic would use the WASD controls to rotate the artifact, rather than using the mouse. Being able to drag the artifact to rotate it with a mouse would be great, but for the time limit and scope of our game, we didn't see any point in putting valuable time into a small change that could be achieved with a different methods.

Overview of system

The **Artifact Interaction System** allows for the player to pick up the artifact, which brings the artifact model up in front of the player and displays a journal page which the **Points of Interests** are "written" to. **Points of Interest** are areas placed on the model of the artifact which contain certain information about the artifact in context of where it is placed. The player must rotate the artifact to find each individual point of interest, and they can proceed once they have uncovered every point of interest on the artifact.

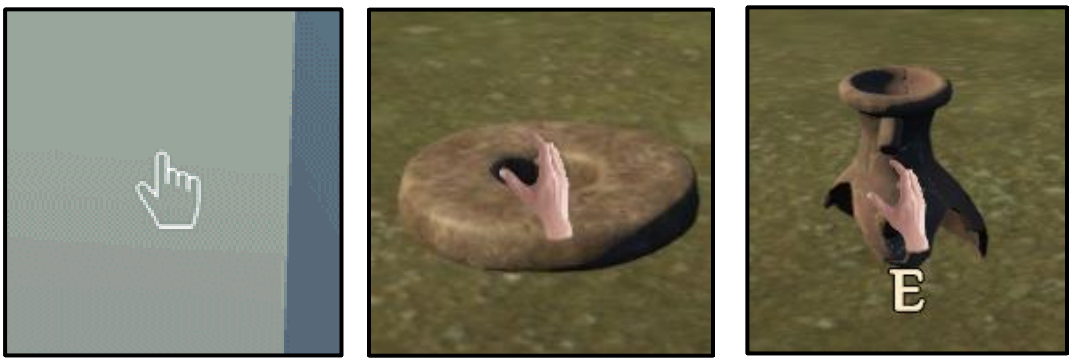

The clear corrosion on the material's surface suggests that this helmet is made from iron

An example of a Point of Interest on an artifact.

One of our programmers, Ben, created the initial system which allowed for the player to look at an artifact and commence the artifact interaction. I continued the system from then onwards.

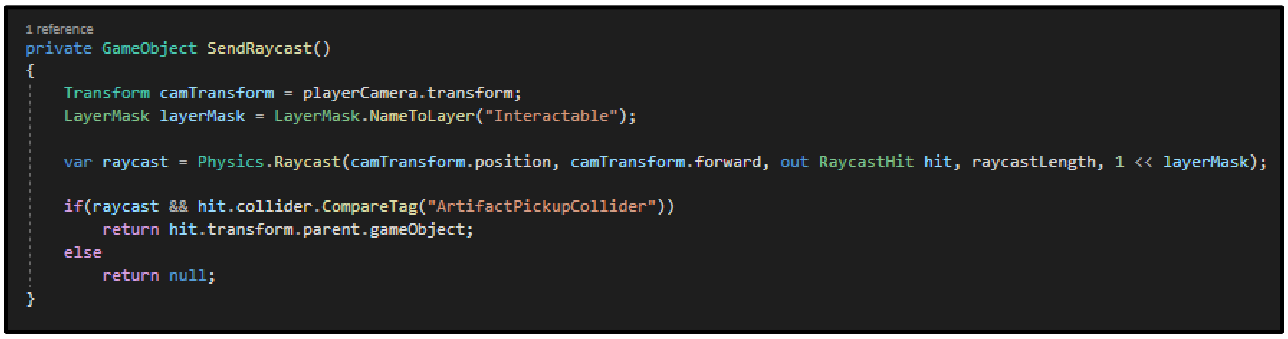
Finding an artifact

A raycast is fired directly from the centre of the player camera in the direction they are facing, to search for any objects in the centre of the screen – in this case the raycast is looking for artifacts. When the raycast finds an artifact within the specified maximum distance, an icon of a hand is displayed. This shows the player that they can interact with this artifact. This hand went through a few iterations, eventually including a small "E" letter to show the player which button they must press to commence Artifact Interaction. An animation was added so that the hand fades in – this was a small but important detail to give the game the feeling of *polish*.



Multiple iterations of the artifact interaction prompt.

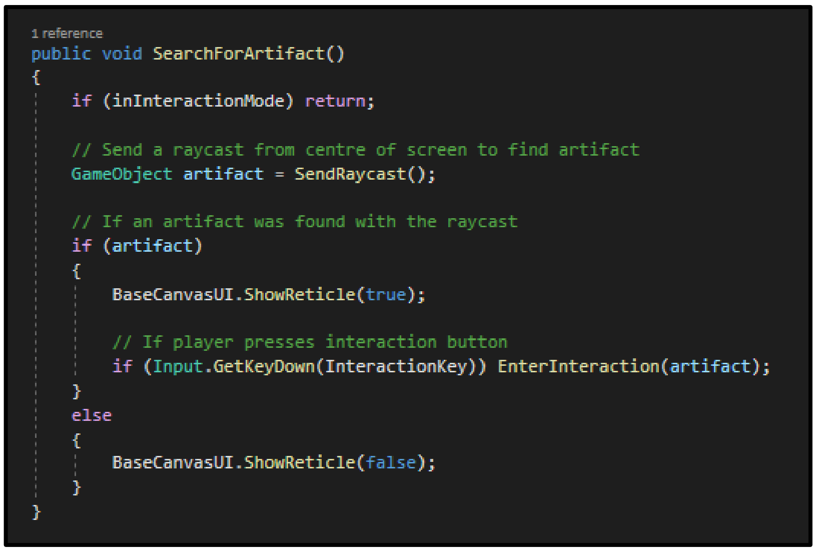
A function named **SendRaycast()** within the ArtifactInteraction script fires a raycast from the centre of the screen, and it searches for any GameObjects marked "Interactable". Every artifact within our game is in the "Interactable" layer within the Unity layer system – if an artifact is being looked at directly by the player, the function returns a reference to the found artifact, otherwise it returns null.



```
1 reference
private GameObject SendRaycast()
{
    Transform camTransform = playerCamera.transform;
    LayerMask layerMask = LayerMask.NameToLayer("Interactable");

    var raycast = Physics.Raycast(camTransform.position, camTransform.forward, out RaycastHit hit, raycastLength, 1 << layerMask);

    if(raycast && hit.collider.CompareTag("ArtifactPickupCollider"))
        return hit.transform.parent.gameObject;
    else
        return null;
}
```

The function SendRaycast() returns any artifacts found by a raycast from the player camera

The function **SearchForArtifact()** is called every update if an artifact is not currently being interacted with. It calls the function SendRaycast() which returns an artifact if it is seen by the raycast. If an artifact is found and the player presses the *Interaction Key* (bound to 'E' in the final game), the function **EnterInteraction()** is called.



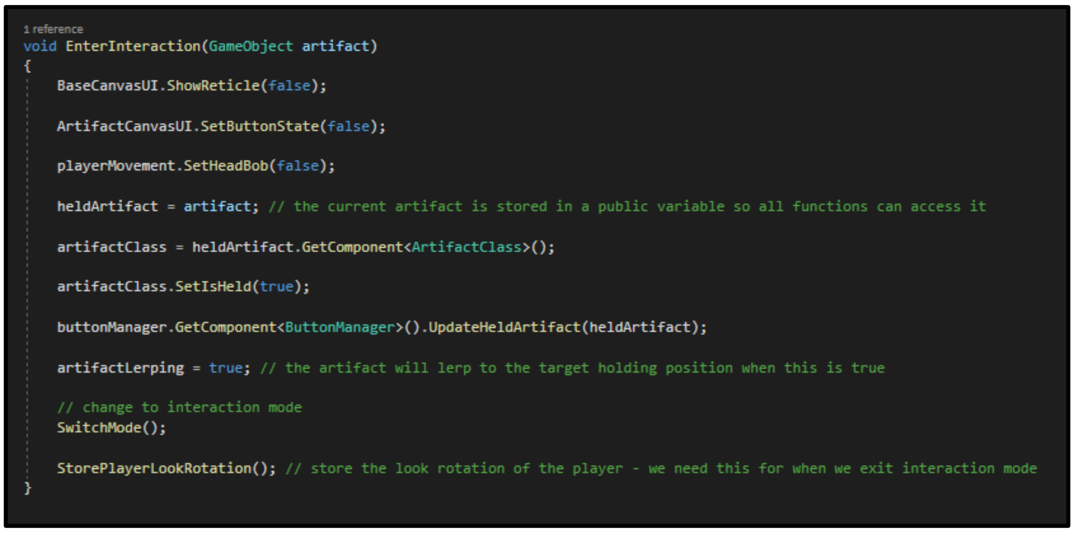
```
1 reference
public void SearchForArtifact()
{
    if (inInteractionMode) return;

    // Send a raycast from centre of screen to find artifact
    GameObject artifact = SendRaycast();

    // If an artifact was found with the raycast
    if (artifact)
    {
        BaseCanvasUI.ShowReticle(true);

        // If player presses interaction button
        if (Input.GetKeyDown(InteractionKey)) EnterInteraction(artifact);
    }
    else
    {
        BaseCanvasUI.ShowReticle(false);
    }
}
```

The SearchForArtifact() function determines if the requirements are met to enter the interaction mode

Entering Interaction

The **EnterInteraction()** function effectively ensures that the UI and player controls are changed so that artifact interaction works without flaws and is uninterrupted by other scripts. I created this function early in the development of the game and have learnt a lot since – I didn't revisit this function as it was too interconnected with other scripts midway through development.

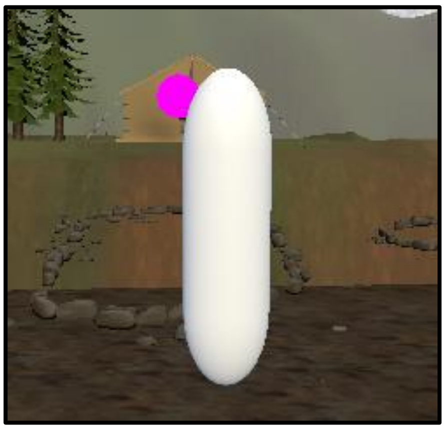
If I could revisit this script, I would introduce a game-wide event system and scriptable objects to hold certain values so that changing "modes" in the game is clear and simple, and the scripts don't depend on each other.



```
1 reference
void EnterInteraction(GameObject artifact)
{
    BaseCanvasUI.ShowReticle(false);

    ArtifactCanvasUI.SetButtonState(false);

    playerMovement.SetHeadBob(false);

    heldArtifact = artifact; // the current artifact is stored in a public variable so all functions can access it

    artifactClass = heldArtifact.GetComponent<ArtifactClass>();

    artifactClass.SetIsHeld(true);

    buttonManager.GetComponent<ButtonManager>().UpdateHeldArtifact(heldArtifact);

    artifactLerping = true; // the artifact will lerp to the target holding position when this is true

    // change to interaction mode
    SwitchMode();

    StorePlayerLookRotation(); // store the look rotation of the player - we need this for when we exit interaction mode
}
```

The SearchForArtifact() function determines if the requirements are met to enter the interaction mode

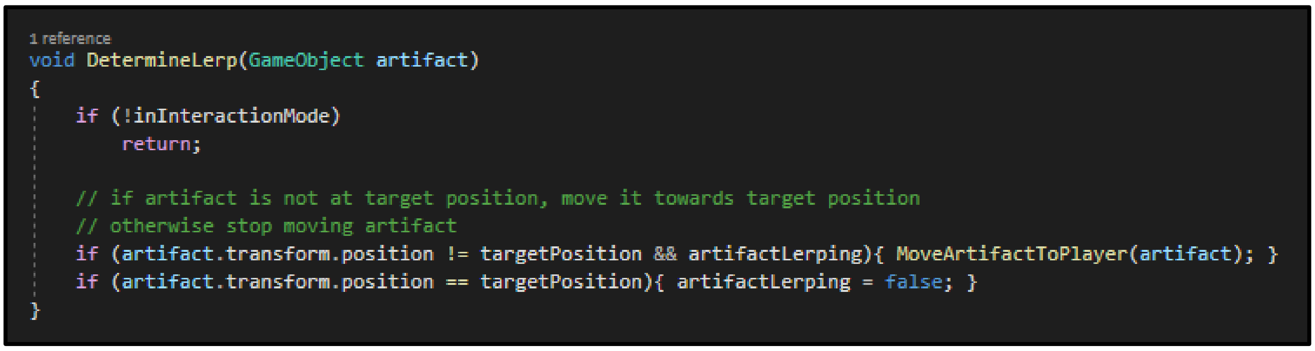
Once the artifact has been "interacted" with, it moves to a point in front of the player. There were many ways to create this movement – snapping, translating or "lerping".



The destination of the artifact when it moves towards a position in front of the player.

Behind the player (left) and in front (right)

I decided to use lerping (linear interpolation) over the other two methods – this is due to the smooth change in speed it provides, rather than a constant speed or sudden change in position. The artifact stops lerping as soon as it reaches its target destination.



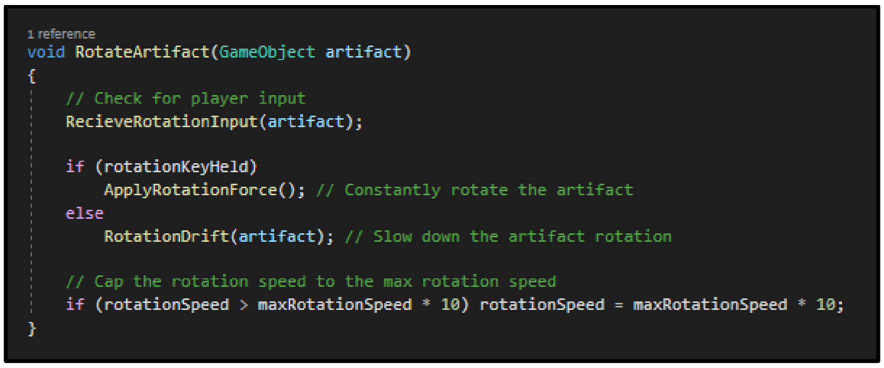
```
1 reference
void DetermineLerp(GameObject artifact)
{
    if (!inInteractionMode)
        return;

    // if artifact is not at target position, move it towards target position
    // otherwise stop moving artifact
    if (artifact.transform.position != targetPosition && artifactLerping){ MoveArtifactToPlayer(artifact); }
    if (artifact.transform.position == targetPosition){ artifactLerping = false; }
}
```

The DetermineLerp() function that determines if the artifact has reached its destination.

Rotating an artifact

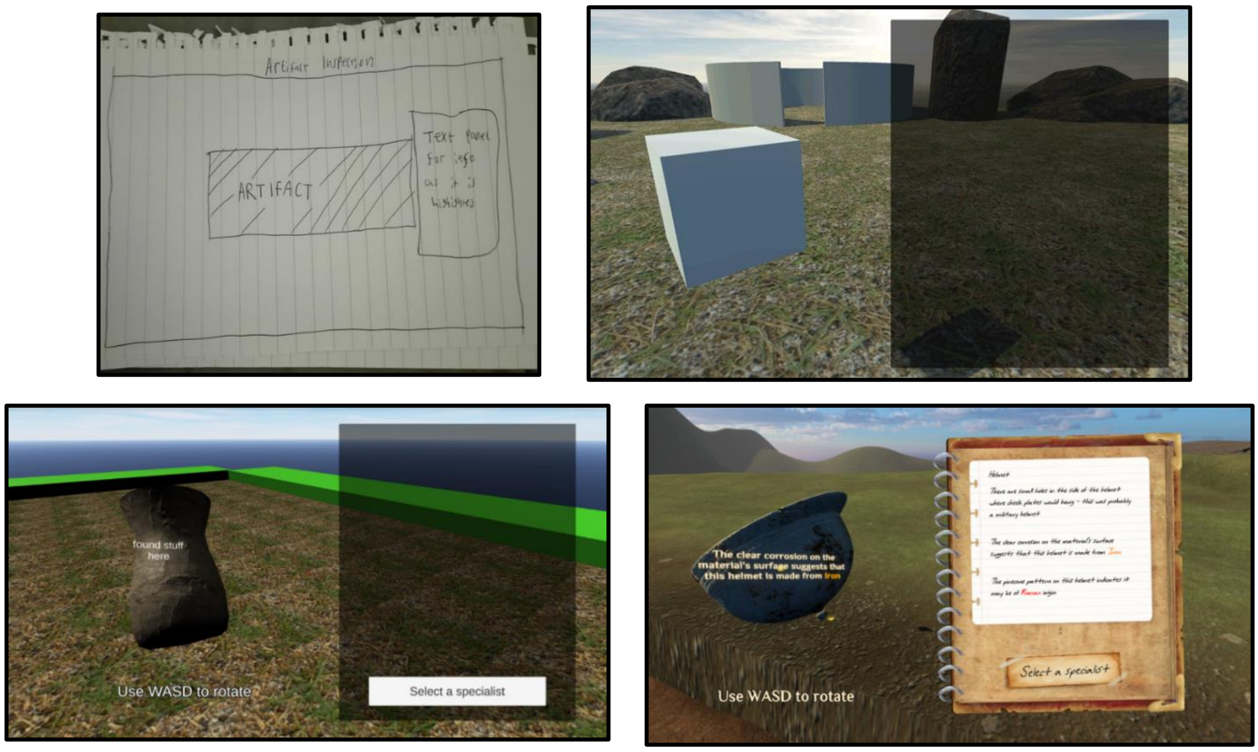
The most challenging part of making the Artifact Interaction System was the **rotation** of the artifact. I had dealt with Quaternions, Euler angles and general rotation qualms when I was doing the Green Game Jam in November. The rotation of the artifact has acceleration so that it feels smooth, but this time I faced the issue of *decelerating* the rotation *after* the player releases the key.

I had to keep track of whether the player is pressing a key to rotate the artifact, the rotation direction at the point that the player releases the key, and when the artifact has fully stopped rotating. It was a challenge I did not foresee but managed to overcome to make a very smooth, polished rotation system that I am proud of.



```
1 reference
void RotateArtifact(GameObject artifact)
{
    // Check for player input
    RecieveRotationInput(artifact);

    if (rotationKeyHeld)
        ApplyRotationForce(); // Constantly rotate the artifact
    else
        RotationDrift(artifact); // Slow down the artifact rotation

    // Cap the rotation speed to the max rotation speed
    if (rotationSpeed > maxRotationSpeed * 10) rotationSpeed = maxRotationSpeed * 10;
}
```

The RotateArtifact() script, different elements of the rotation had to be separated into multiple functions


Artifact Inspection
ARTIFACT
Text panel for info as it is highlighted
found stuff here
Use WASD to rotate
Select a specialist
The clear corrosion on the material's surfage suggests that this helmet is made from iron
Helmet
There are small holes in the side of the helmet where cheek plates would hang – this was probably a military helmet
The clear corrosion on the material's surface suggests that this helmet is made from Iron
The pinecone pattern on this helmet indicates it may be of Roman origin
Use WASD to rotate
Select a specialist

Development of the Artifact Interaction System over time, from wireframe to final look.

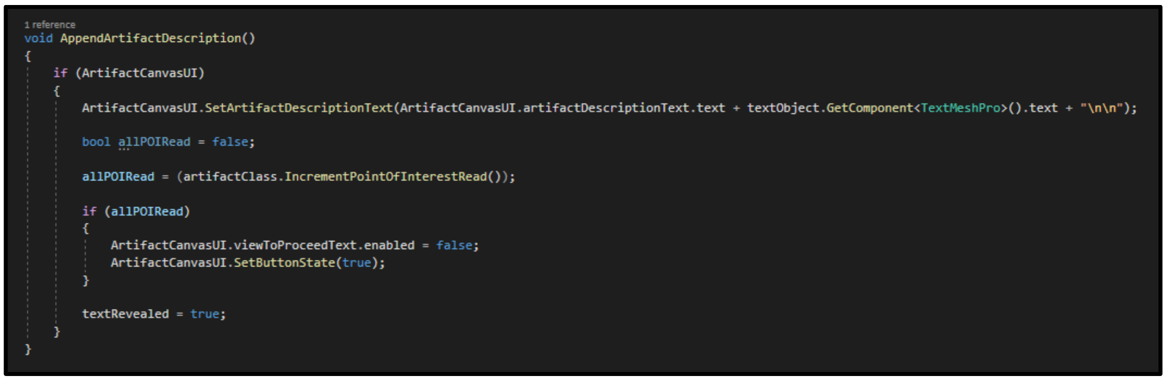
Points of Interest

The next step of the **Artifact Interaction System** was adding **Points of Interest.**

Each point of interest is a GameObject that is a child of an empty GameObject named "PointsOfInterest" which is a child of the artifact. These GameObjects have a Box Collider attached to them which fires a raycast in the direction the GameObject is facing in the Z axis. This is controlled by the script **PointOfInterest** which is attached to the GameObject. If the raycast finds the sphere collider of the GameObject "Head" that is a child of the player, it will display the Mesh Rendered text that is a child of the PointOfInterest object (this text is disabled by default). This results in text being revealed as the player rotates the artifact and finds new points of interest. The text always remains upright when the artifact is being rotated.

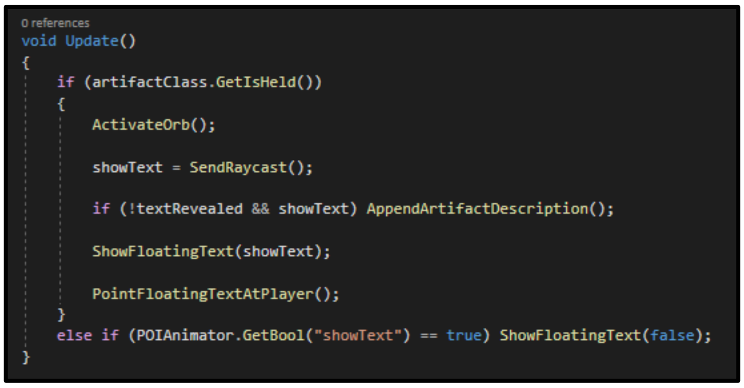
A small golden sphere was added to each point of interest. These spheres are visible as soon as the player picks up the artifact. This was a design decision from midway through development, which made it easier for the player to find points of interest without randomly rotating artifacts.

The panel to the right of the artifact is a notebook for the Point of Interest information to be written onto – the function **AppendArtifactDescription()** gets a reference of the ArtifactCanvas - the canvas which holds the panel, and adds the text held by the point of interest to it.



```
1 reference
void AppendArtifactDescription()
{
    if (ArtifactCanvasUI)
    {
        ArtifactCanvasUI.SetArtifactDescriptionText(ArtifactCanvasUI.artifactDescriptionText.text + textObject.GetComponent<TextMeshPro>().text + "\n\n");

        bool allPOIRead = false;

        allPOIRead = (artifactClass.IncrementPointOfInterestRead());

        if (allPOIRead)
        {
            ArtifactCanvasUI.viewToProceedText.enabled = false;
            ArtifactCanvasUI.SetButtonState(true);
        }

        textRevealed = true;
    }
}
```

The AppendArtifactDescription() function within the script PointOfInterest script



```
0 references
void Update()
{
    if (artifactClass.GetIsHeld())
    {
        ActivateOrb();

        showText = SendRaycast();

        if (!textRevealed && showText) AppendArtifactDescription();

        ShowFloatingText(showText);

        PointFloatingTextAtPlayer();
    }
    else if (POIAnimator.GetBool("showText") == true) ShowFloatingText(false);
}
```

The Update() function of the PointOfInterest script. This calls the SendRaycast() function to check if the player's head is seen.

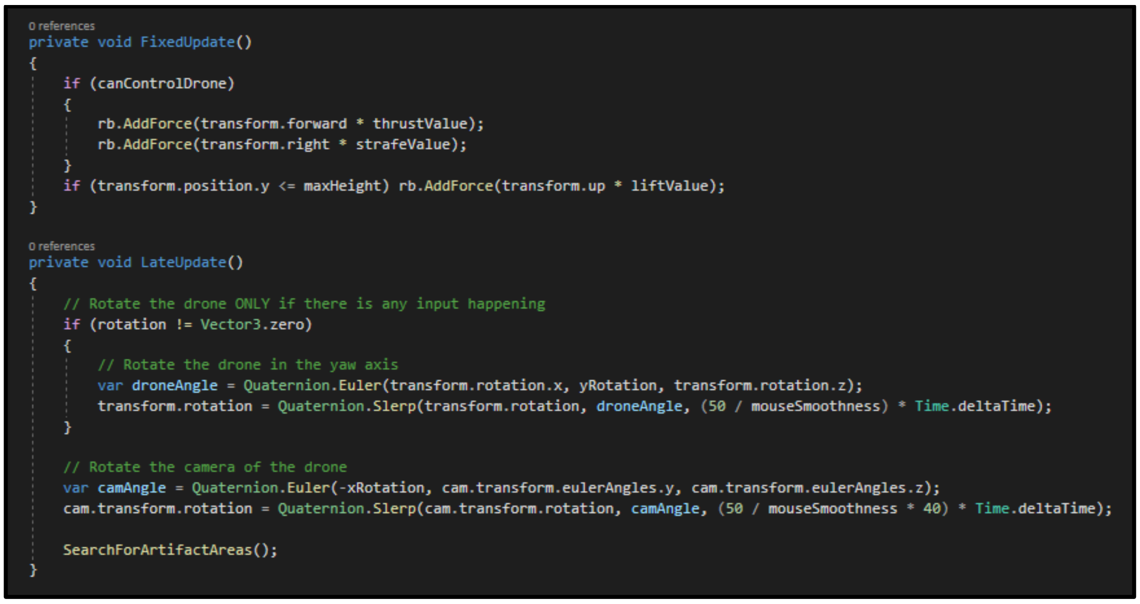
The Drone

When brainstorming ideas for the teaser trailer, I decided that for some of our shots it would be wise to create a custom "**Cinematic Camera**" that can fly around and pan across the scene smoothly. I created this mechanic and decided that it would be a perfect fit for one of our **Research Methods** - perks that can be invested in for equipment and assistance for the dig site. I brought it up to the designers and the rest of the team and they agreed that it would be a perfect fit for the game.

Drone Movement

The **Drone** moves using Unity's in-built physics and uses the WASD keys for moving in the X and Y directions. Space and Shift are used to move up and down respectively, and the Mouse is used to look around. These controls changed when we decided to remove the ability to change elevation. The decision to remove changing the altitude was made due to the player being able to crash the drone into the ground or trees, which resulted in messy physics.

The script **DroneCamera** has a FixedUpdate() function for manipulating physics and applies certain forces to the Drone's Rigidbody depending on player input. The movement of the mouse rotates the Drone's yaw, but the Drone's camera moves independently - the pitch axis changes if the player moves the mouse up or down. The camera rotation is clamped between a minimum and maximum value – this means that while controlling the Drone, the player can look straight down and straight up, but no angle beyond that, which would result in the camera rolling upside down. The Drone has a Rigidbody component which is unaffected by gravity, allowing the Drone to keep its height in the air, akin to that of a real drone.

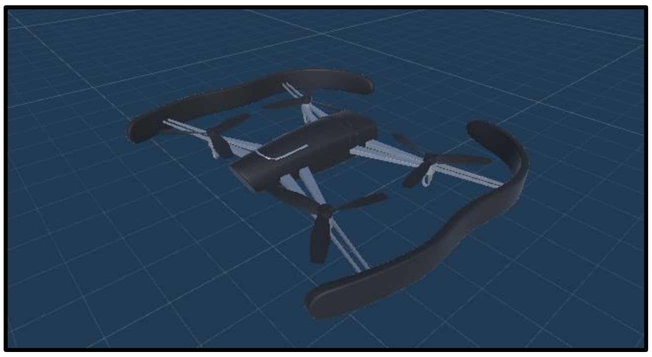


```
0 references
private void FixedUpdate()
{
    if (canControlDrone)
    {
        rb.AddForce(transform.forward * thrustValue);
        rb.AddForce(transform.right * strafeValue);
    }
    if (transform.position.y <= maxHeight) rb.AddForce(transform.up * liftValue);
}

0 references
private void LateUpdate()
{
    // Rotate the drone ONLY if there is any input happening
    if (rotation != Vector3.zero)
    {
        // Rotate the drone in the yaw axis
        var droneAngle = Quaternion.Euler(transform.rotation.x, yRotation, transform.rotation.z);
        transform.rotation = Quaternion.Slerp(transform.rotation, droneAngle, (50 / mouseSmoothness) * Time.deltaTime);
    }

    // Rotate the camera of the drone
    var camAngle = Quaternion.Euler(-xRotation, cam.transform.eulerAngles.y, cam.transform.eulerAngles.z);
    cam.transform.rotation = Quaternion.Slerp(cam.transform.rotation, camAngle, (50 / mouseSmoothness * 40) * Time.deltaTime);

    SearchForArtifactAreas();
}
```

The two update functions that determine the force applied to the Drone and its rotation based on player input.

I got a 3D model of a drone from the online website TurboSquid and animated the blades to spin. It is unlikely that the player will see the Drone but I thought it would be a nice little touch – I made the animation 4 frames long so that the cost is negligible in terms of processing power.

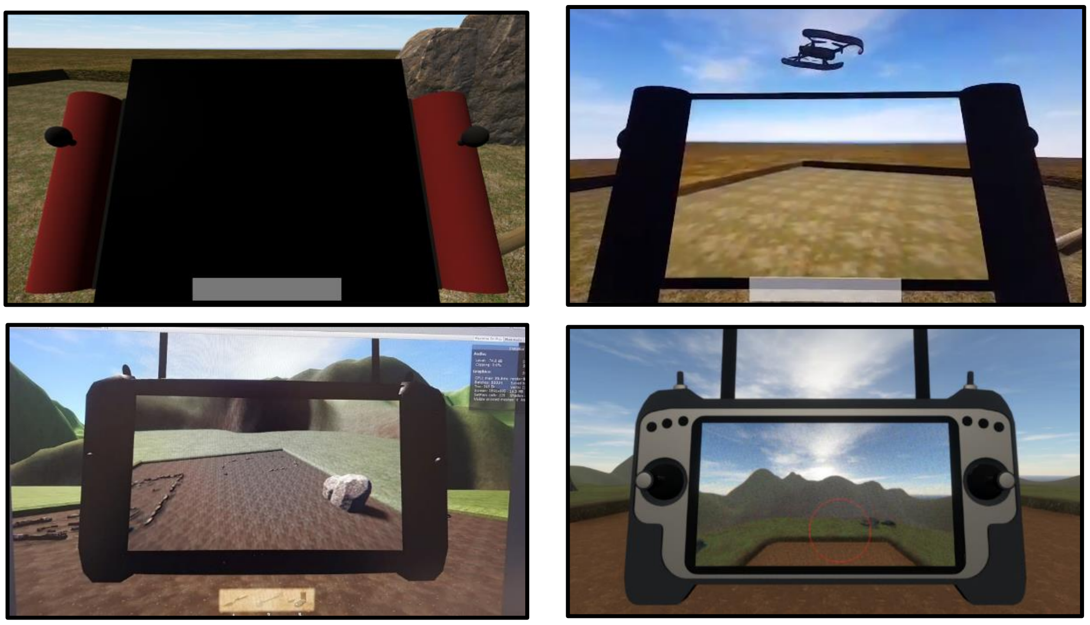
The Drone model in its untextured, initial form.


The Drone model after texturing and animation.

The Drone Controller

When the player presses the button to bring up the Drone (Initially 'L', now the number 3), a remote controller is raised from below the viewport, which displays a video feed of the Drone's camera. This is done by creating a Render Texture that the Drone's camera outputs directly to. The camera also has a couple of post-processing effects such as film grain, lens distortion and colour grading to make the Drone's camera feed appear to be a video stream.

I originally created the model of the Drone Controller using simple shapes, and over time it was developed until our prop artist Csenge provided a lovely new model for the final Drone Controller.

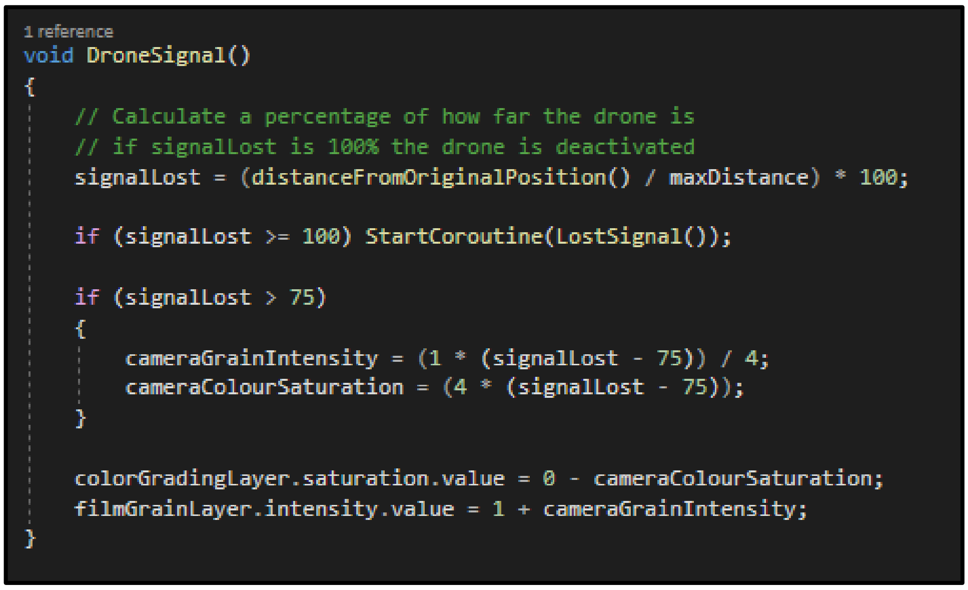

The Drone Controller iterating over time – the bottom right image is of Csenge's Drone Controller Model

When the Drone Controller is equipped, the Drone prefab is instantiated – this means that the Drone does not exist until the Drone Controller is equipped. When the Drone Controller is unequipped, the Drone is destroyed. This aids performance as it removes a second camera from the scene.

As the Drone was being developed, we discussed the idea of a "**battery**" system, where the Player had a limited amount of time to use the Drone within the level. I raised the point that this would be limiting in terms of the "fun factor" of the game and emphasised that these aspects of the game should not be time-limited, as the game is foremost an educational game where people can learn at their own pace.

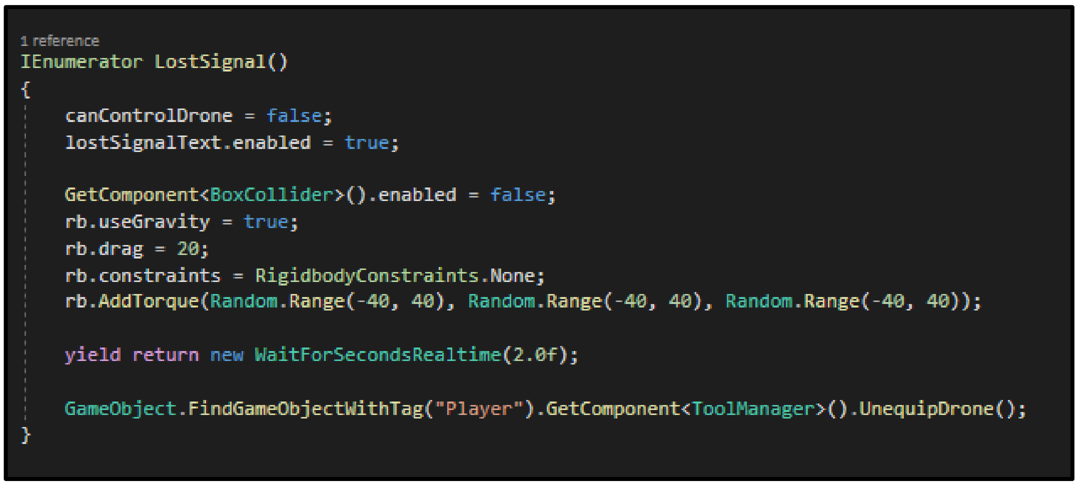
Drone Signal

As a method of limiting the drone from moving too far from the centre of the level, I added functionality to the **DroneCamera** script to check this distance. The distance the drone is from the maximum set distance is calculated as a percentage – for example if the Drone was 80% distance it is near the maximum distance from the centre of the level. Post-processing effects change and intensify depending on the percentage – the further the player gets, the grainier and more desaturated the screen becomes, representing signal loss. The idea of the Drone having a signal was inspired by other games such as Watch Dogs 2 (2016) and Call of Duty: Warzone (2020) – both games use drones that are limited by their signal distance.


```
1 reference
void DroneSignal()
{
    // Calculate a percentage of how far the drone is
    // if signalLost is 100% the drone is deactivated
    signalLost = (distanceFromOriginalPosition() / maxDistance) * 100;

    if (signalLost >= 100) StartCoroutine(LostSignal());

    if (signalLost > 75)
    {
        cameraGrainIntensity = (1 * (signalLost - 75)) / 4;
        cameraColourSaturation = (4 * (signalLost - 75));
    }

    colorGradingLayer.saturation.value = 0 - cameraColourSaturation;
    filmGrainLayer.intensity.value = 1 + cameraGrainIntensity;
}
```

The DroneSignal() function that determines the Drone's distance from the centre of the level

If the Drone reaches the maximum distance, the function **LostSignal()** is called. This function stops the player from being able to control the drone and alters the Rigidbody values of this instance of the Drone to make it plummet towards the ground. After a couple of seconds of crashing, the Drone is destroyed, and the Drone Controller is unequipped. If the player re-equips the Controller, the Drone is instantiated once again.

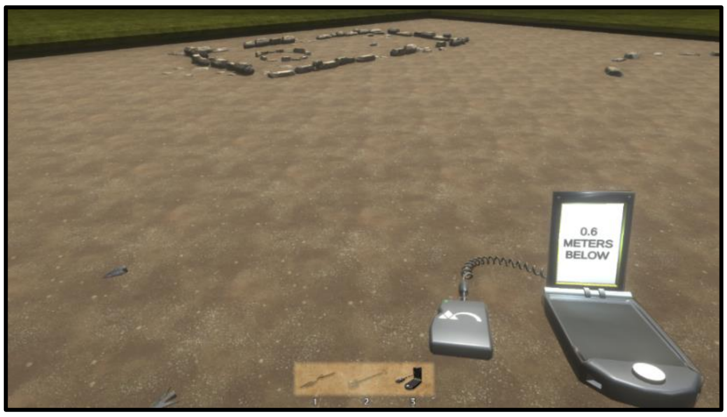


```
1 reference
IEnumerator LostSignal()
{
    canControlDrone = false;
    lostSignalText.enabled = true;

    GetComponent<BoxCollider>().enabled = false;
    rb.useGravity = true;
    rb.drag = 20;
    rb.constraints = RigidbodyConstraints.None;
    rb.AddTorque(Random.Range(-40, 40), Random.Range(-40, 40), Random.Range(-40, 40));

    yield return new WaitForSecondsRealtime(2.0f);

    GameObject.FindGameObjectWithTag("Player").GetComponent<ToolManager>().UnequipDrone();
}
```

The LostSignal() function that "crashes" the Drone and prevents the Player from continuing to fly out of the distance limit

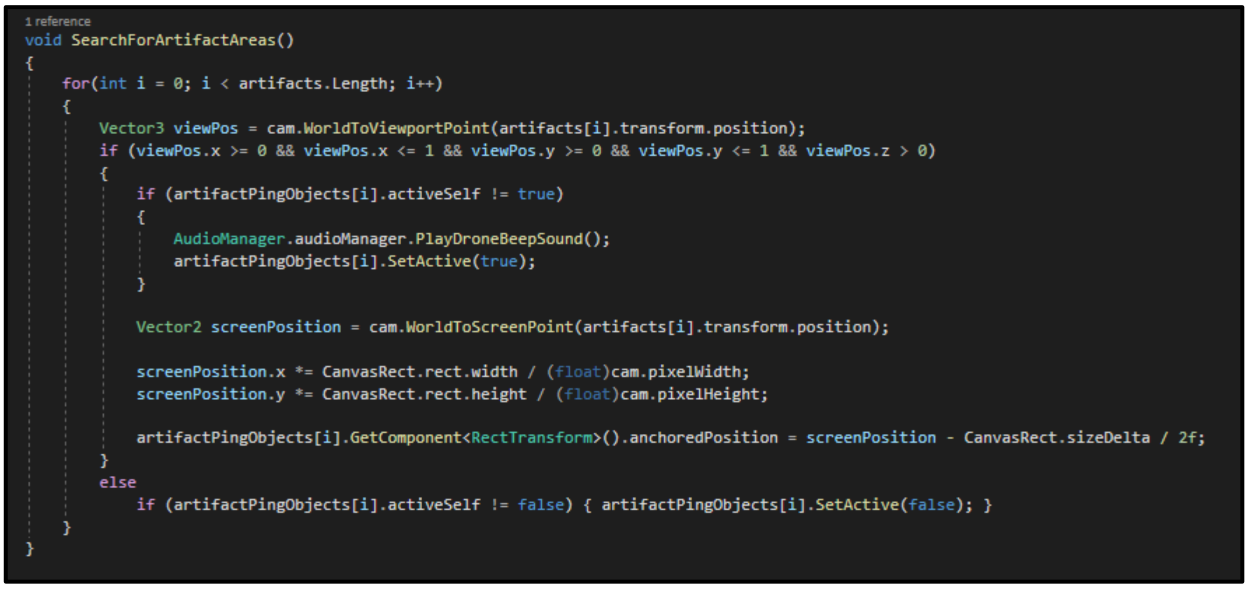
Artifact Detection

For the first couple of months of development, we were using a tool named the "**Artifact Detector**" created by our programmer, Ben. This tool would scan below the earth and find artifacts directly below the player, showing how deep the object is in metres on its small screen. The mechanic was very well made, but the team decided that it was too "powerful" a tool for the player to wield and would remove a lot of time and strategy from the game. At this time the Drone didn't have a purpose beyond being able to see everything from an advantage, so we saw a situation where we could provide vague locations of artifacts whilst giving the Drone a proper purpose.



The Artifact Detector that was once an important part of the game.

Each Artifact has a child object named "ArtifactArea" which holds a sphere collider component, offset by a random amount on the x and y axis so it isn't centred on the artifact. The function **SearchForArtifactAreas()** within **DroneCamera** searches through each found ArtifactArea and checks if it is on screen – if so, a sonar-like beeping sound is played, and an object is set to active which animates, representing an expanding circle denoting the rough location of an artifact. If many artifacts are seen on-screen, there are many of these circles displayed.



```
1 reference
void SearchForArtifactAreas()
{
    for(int i = 0; i < artifacts.Length; i++)
    {
        Vector3 viewPos = cam.WorldToViewportPoint(artifacts[i].transform.position);
        if (viewPos.x >= 0 && viewPos.x <= 1 && viewPos.y >= 0 && viewPos.y <= 1 && viewPos.z > 0)
        {
            if (artifactPingObjects[i].activeSelf != true)
            {
                AudioManager.audioManager.PlayDroneBeepSound();
                artifactPingObjects[i].SetActive(true);
            }

            Vector2 screenPosition = cam.WorldToScreenPoint(artifacts[i].transform.position);

            screenPosition.x *= CanvasRect.rect.width / (float)cam.pixelWidth;
            screenPosition.y *= CanvasRect.rect.height / (float)cam.pixelHeight;

            artifactPingObjects[i].GetComponent<RectTransform>().anchoredPosition = screenPosition - CanvasRect.sizeDelta / 2f;
        }
        else
            if (artifactPingObjects[i].activeSelf != false) { artifactPingObjects[i].SetActive(false); }
    }
}
```

The SearchForArtifactAreas() function which determines which artifacts are within the bounds of the screen.

Drone Audio

The Drone holds an AudioSource component that plays a looping sound when created. I altered the AudioSource component to present the sound in a 3D range – when the player deploys the Drone, they can hear it flying above them.

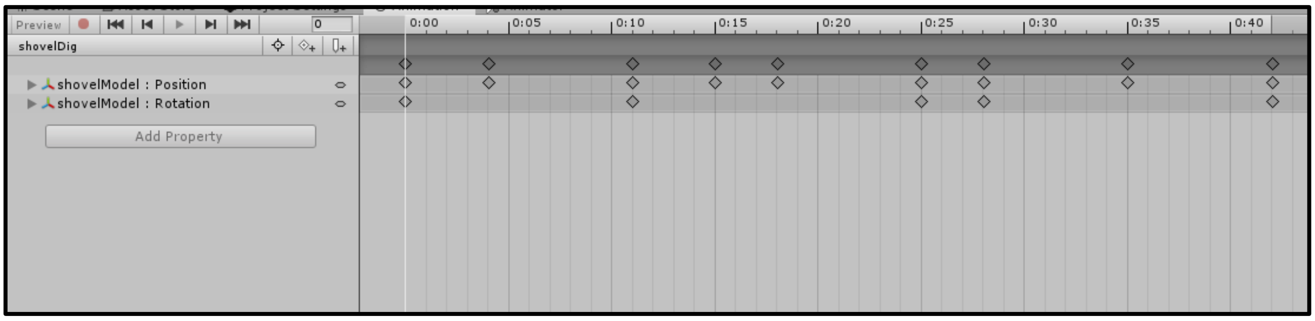
Near the end of development, a Cinematic Camera was created based on the drone. This allowed me to make landscape shots for the teaser trailer.

Animating the Tools

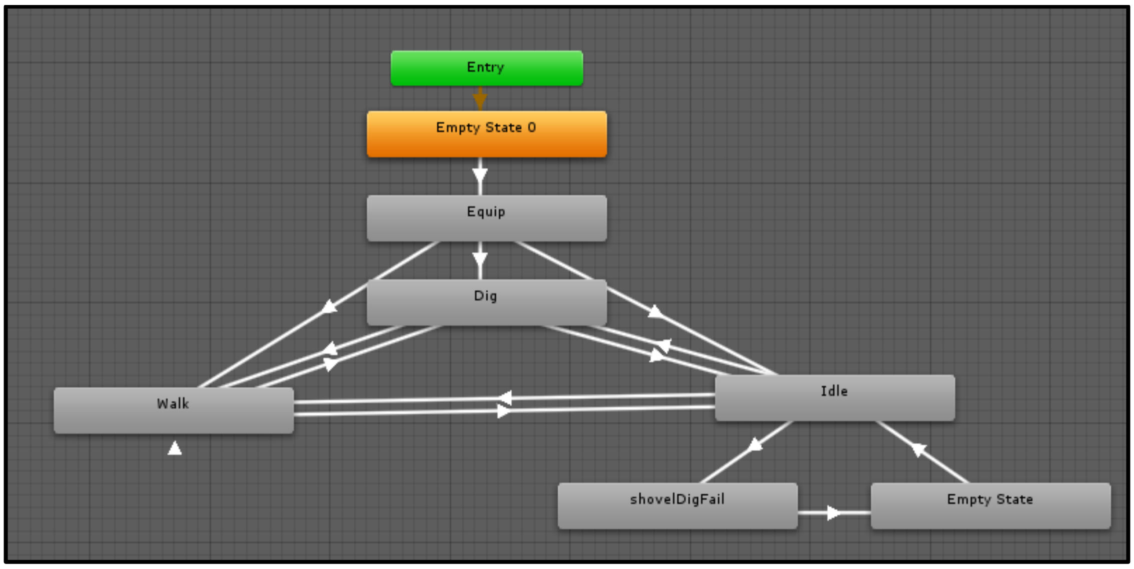
As our 3D artists specialise in environment and prop assets, they couldn't provide animations for the game. We designed the game to be animation-light to compensate for this. Due to my moderate experience in using the Unity in-engine animator, I decided to take on the role of animator for UI and tools.

The UI animations are simple – if the UI element is non-diegetic, it appears by fading and scaling in, as if from nowhere. If the UI element is diegetic like the Journal, it pans up from the bottom of the screen, as if the player has pulled the Journal out of their pocket. It was important to differentiate the diegetic and non-diegetic UI, so that the player would understand what belongs in the world of the game, and what is made simply to assist and guide the player through non-diegetic means.

When animating the tools, I had to consider creating animations for equipping the tools, unequipping them, the player being idle, walking, digging and failing to dig. The Unity animator helped a lot with blending these animations together.

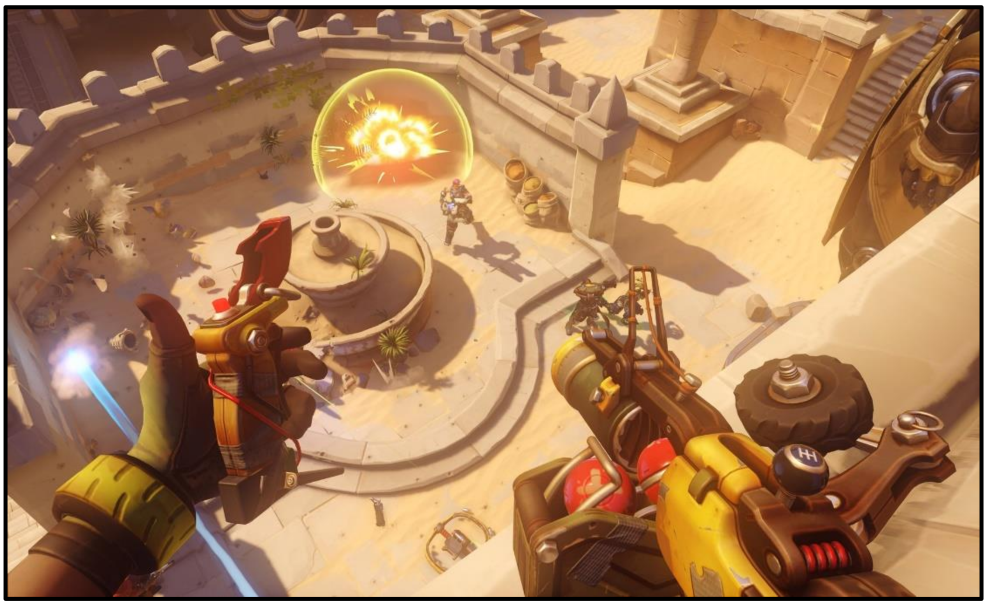


The keyframes for the Shovel digging animation – they manipulate only the position and rotation of the model



The animation tree for the Shovel

As someone who's relatively inexperienced in animation, I looked at the games Overwatch (2016) and Firewatch (2016) for inspiration. Due to the first-person perspective in both games, they use the animations of the characters' hands, tools or weapons to convey both character and motion.



The character "Junkrat" from Overwatch (2016) – their homemade bomb launcher almost falls apart with every step as individual elements of the weapon bounce and shake. The bob of the weapon reflects Junkrat's limp due to his wooden leg.

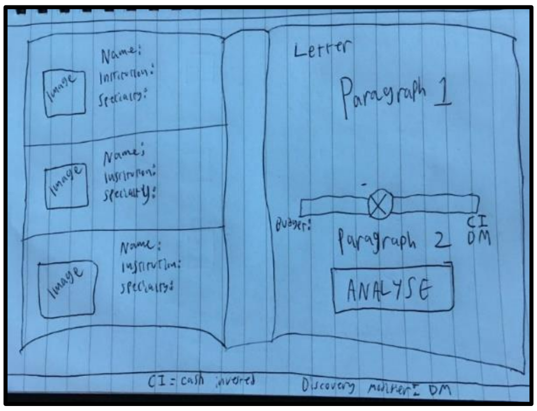
I based the digging animations of the Trowel and the Shovel on videos found on YouTube of archaeological excavation, to make it as realistic to the process as I can. I refined the animations and their transitions over time to be more accurate and responsive – I removed the "unequip" animation as a way of speeding up the changing of tools. I timed the animations so that they were synced with our sound effects. Working with Ben's **ToolManager** script, I added the animations and sounds into the appropriate functions.

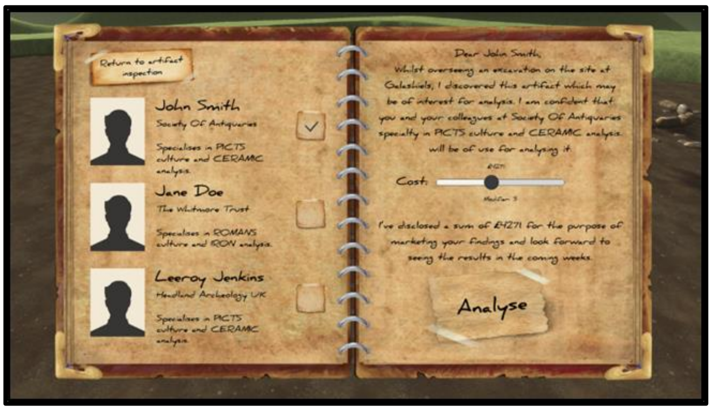
The Specialist Page

After the player has viewed every Point of Interest on an artifact, they can proceed to the **Specialist Page**. This page allows the player to select a certain "specialist" which the artifact is sent to for analysis. If their specialisation matches the type or era of the artifact (for example Iron Age or Pottery), you receive bonuses to your reputation – this means that it is important to try to send the artifact to the correct specialist.

The initial design for the Specialist Page contained only three specialists, but by the end we had four. I pushed for a change to the design to allow for a more modular way of selecting specialists whilst giving the UI more space to "breathe".

Changing the Design




Return to artifact inspection

John Smith
Society Of Antiquaries
Specialises in PICTS culture and CERAMIC analysis.

Jane Doe
The Whitmore Trust
Specialises in ROMANS culture and IRON analysis.

Leeroy Jenkins
Headland Archeology UK
Specialises in PICTS culture and CERAMIC analysis.

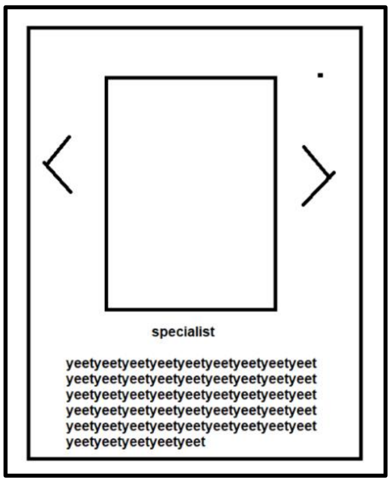
Dear John Smith,
Whilst overseeing an excavation on the site at Galashiels, I discovered this artifact which may be of interest for analysis. I am confident that you and your colleagues at Society Of Antiquaries specialty in PICTS culture and CERAMIC analysis will be of use for analysing it.

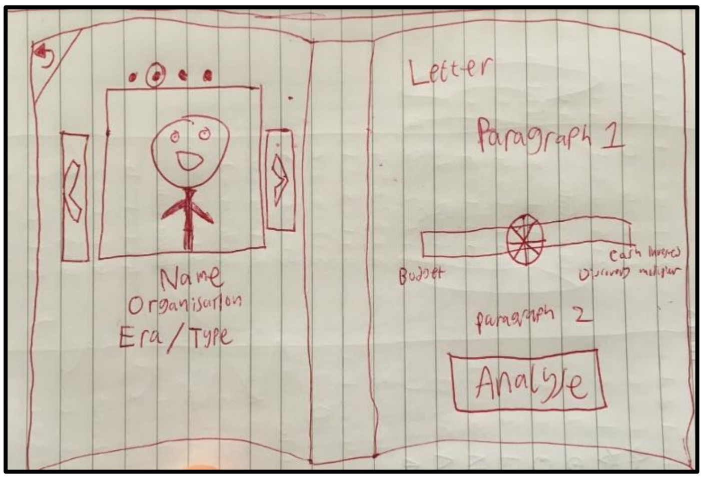
Cost:

I've disclosed a sum of £4271 for the purpose of marketing your findings and look forward to seeing the results in the coming weeks.

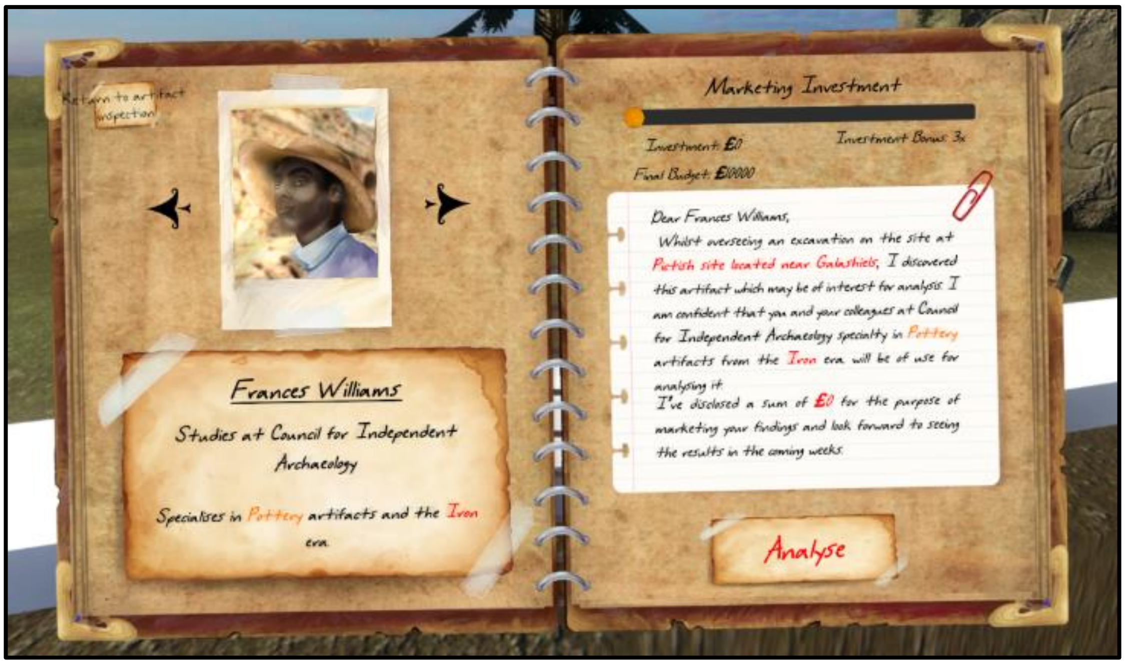
Analyse

The initial layout of the Specialist Page – wireframe (left), in game (right).


specialist
yeetyeetyeetyeetyeetyeetyeetyeetyeet
yeetyeetyeetyeetyeetyeetyeetyeetyeet
yeetyeetyeetyeetyeetyeetyeetyeetyeet
yeetyeetyeetyeetyeetyeetyeetyeetyeet
yeetyeetyeetyeetyeetyeetyeetyeetyeet
yeetyeetyeetyeetyeet



My rough design of the new left page (left) and the revised wireframe (right)



Frances Williams

Studies at Council for Independent Archaeology

Specialises in Pottery artifacts and the Iron era

Marketing Investment

Investment: £0 Investment Bonus: 3x

Final Budget: £10000

Dear Frances Williams,
Whilst overseeing an excavation on the site at Pictish site located near Galashiels, I discovered this artifact which may be of interest for analysis. I am confident that you and your colleagues at Council for Independent Archaeology specialty in Pottery artifacts from the Iron era will be of use for analysing it.
I've disclosed a sum of £0 for the purpose of marketing your findings and look forward to seeing the results in the coming weeks.

The new design in-game

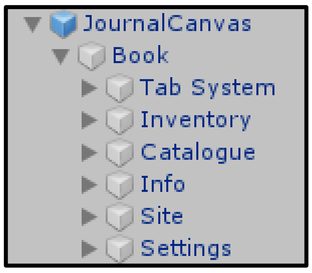
Functionality

Our programmer, James, worked with the currency aspect of this page. The left page is linked to scriptable objects that hold information on each specialist, making it easy to add as many specialists as needed. The text on the right page is hard coded apart from where it denotes the site name, the specialist's type and era, and the player's investment. I decided to make these "keywords" highlighted in red and orange. Red represents eras, the current site and finance. Orange represents the type of artifact. This text colouring is reflected throughout the level.

The Journal

Creating the Journal

I created the **Journal** by making a new Canvas element, which every element of the Journal would be stored under. There are 6 main elements of the Journal – the Tab System, the Inventory Page, the Catalogue Page, the Info Page and the Settings Page. Each element was self-contained and held different objects and scripts depending on the function of that page or system.

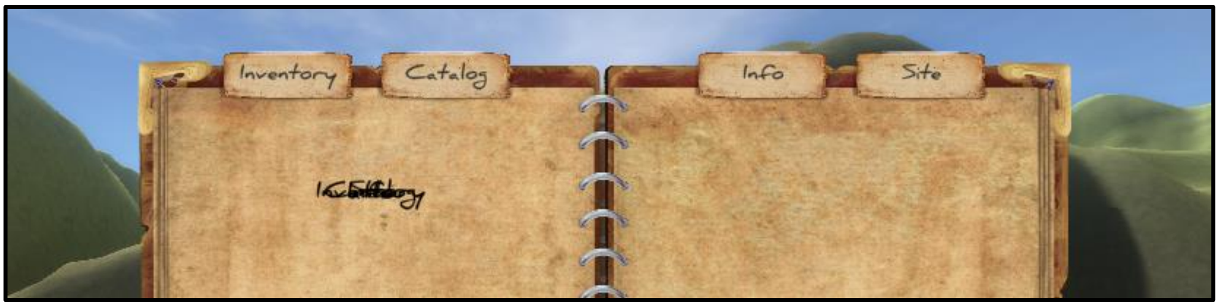


```
▼ JournalCanvas
  ▼ Book
    ▶ Tab System
    ▶ Inventory
    ▶ Catalogue
    ▶ Info
    ▶ Site
    ▶ Settings
```

The hierarchy of the Journal

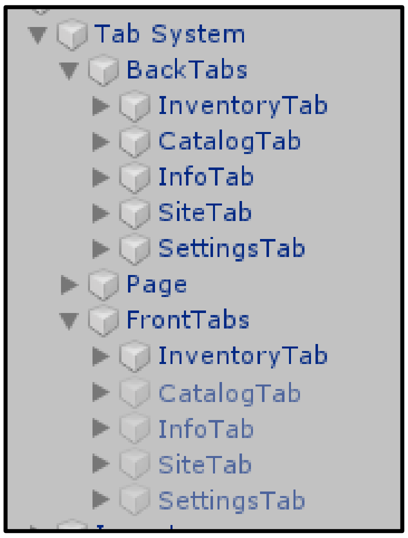
The Tab System

We needed to provide the player with a means of navigating the pages of the Journal, so I created the **Tab System.** The Tab System allows the player to click "tabs" which are like bookmarks - clicking these tabs will open the designated page. When initially creating the tabs, the tabs always appeared to look like they were on the current page:



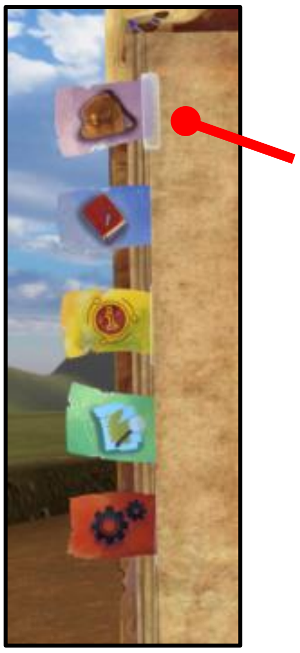
The original look of the tabs without the implementation of Front Tabs and Back Tabs

This looked somewhat strange and didn't make sense for the Journal, so I designed a system where all the tabs are shown, but behind the page sprite so they looked like they were on another page. If a tab is selected, their associated front tab is activated, and all other front tabs are deactivated. This gives the effect that only the tab of the current page is at the front.



```
▼ Tab System
  ▼ BackTabs
    ▶ InventoryTab
    ▶ CatalogTab
    ▶ InfoTab
    ▶ SiteTab
    ▶ SettingsTab
  ▶ Page
  ▼ FrontTabs
    ▶ InventoryTab
    ▶ CatalogTab
    ▶ InfoTab
    ▶ SiteTab
    ▶ SettingsTab
```

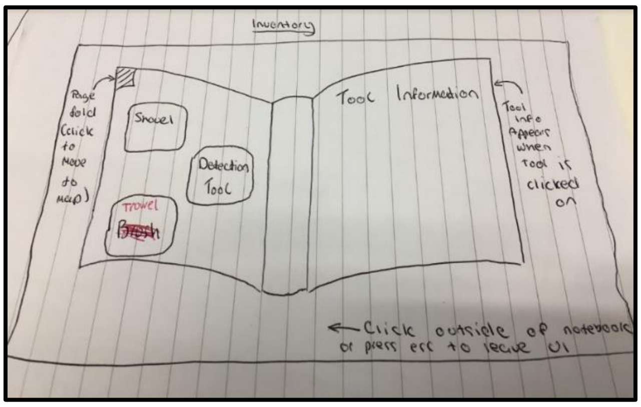
The hierarchy of the Tab System. The “Page” object in the middle holds the sprite of the page and it’s binding.

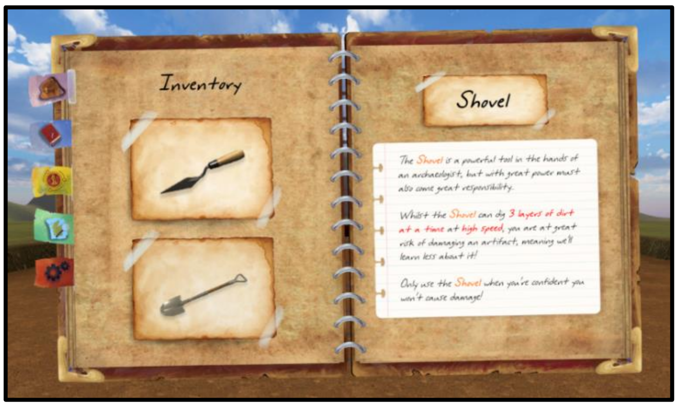


As the Inventory Page is being viewed, the Inventory Tab is represented as being at the front - you can see the tape on the current page.

The Inventory Page

The original design and wireframe of the **Inventory Page** was made to accommodate the Artifact Detector as well as the Trowel and Shovel, but as this mechanic was removed, the design had to change to remove this element.


Inventory
Page fold (click to move to map)
Shovel
Detection Tool
Trowel
~~Brush~~
Tool Information
Tool info appears when tool is clicked on
← Click outside of notebook or press esc to leave UI



The Shovel is a powerful tool in the hands of an archaeologist, but with great power must also come great responsibility.

Whilst the Shovel can dig 3 layers of dirt at a time at high speed, you are at great risk of damaging an artifact, meaning we'll learn less about it!

Only use the Shovel when you're confident you won't cause damage!

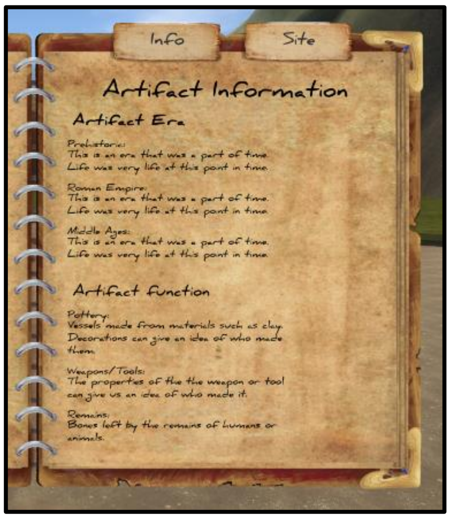
The original wireframe and final design – the Artifact Detector icon is gone and the Trowel and Shovel are no longer offset from each other

The two icons on the left page are buttons, and pressing these buttons set the text on the right page to the strings stored within two Scriptable Objects named “TrowelInfo” and “ShovelInfo”. If more tools were added to the game, this method means that it would be very simple to add more buttons and entries.

The Information Page

The **Information Page** holds all the educational info about archaeological dig practices, the eras that the artifacts could come from, and their type. The player can use this page as a reference for understanding where and when an artifact could come from.

The Dig Practices on the left page are entirely static as they don't change throughout the game. The right page contains different information depending on the selected artifact type or era. It was originally static too, and all the information was stored on the page, but it was far too cramped. Ben and I discussed ways of changing this:



Info Site

Artifact Information

Artifact Era

Prehistoric:
This is an era that was a part of time.
Life was very life at this point in time.

Roman Empire:
This is an era that was a part of time.
Life was very life at this point in time.

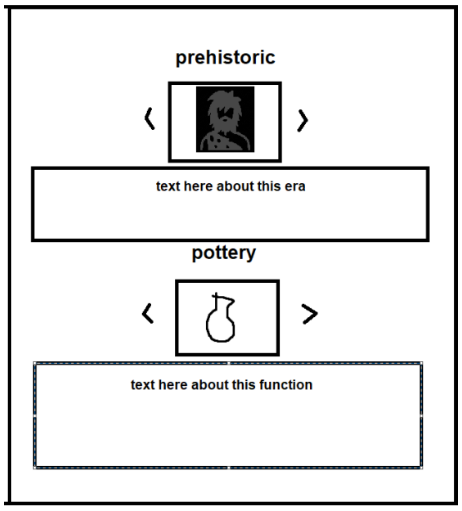
Middle Ages:
This is an era that was a part of time.
Life was very life at this point in time.

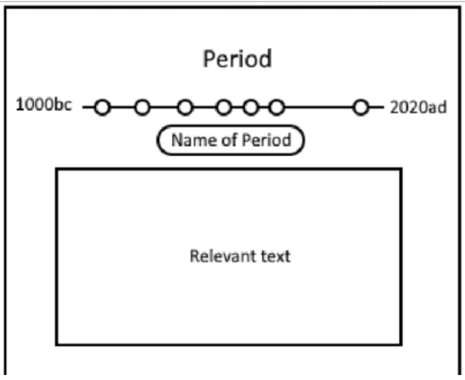
Artifact function

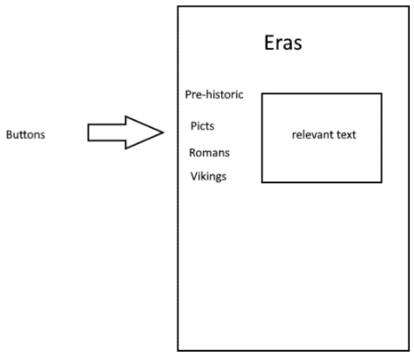
Pottery:
Vessels made from materials such as clay.
Decorations can give an idea of who made them.

Weapons/Tools:
The properties of the the weapon or tool can give us an idea of who made it.

Remains:
Bones left by the remains of humans or animals.

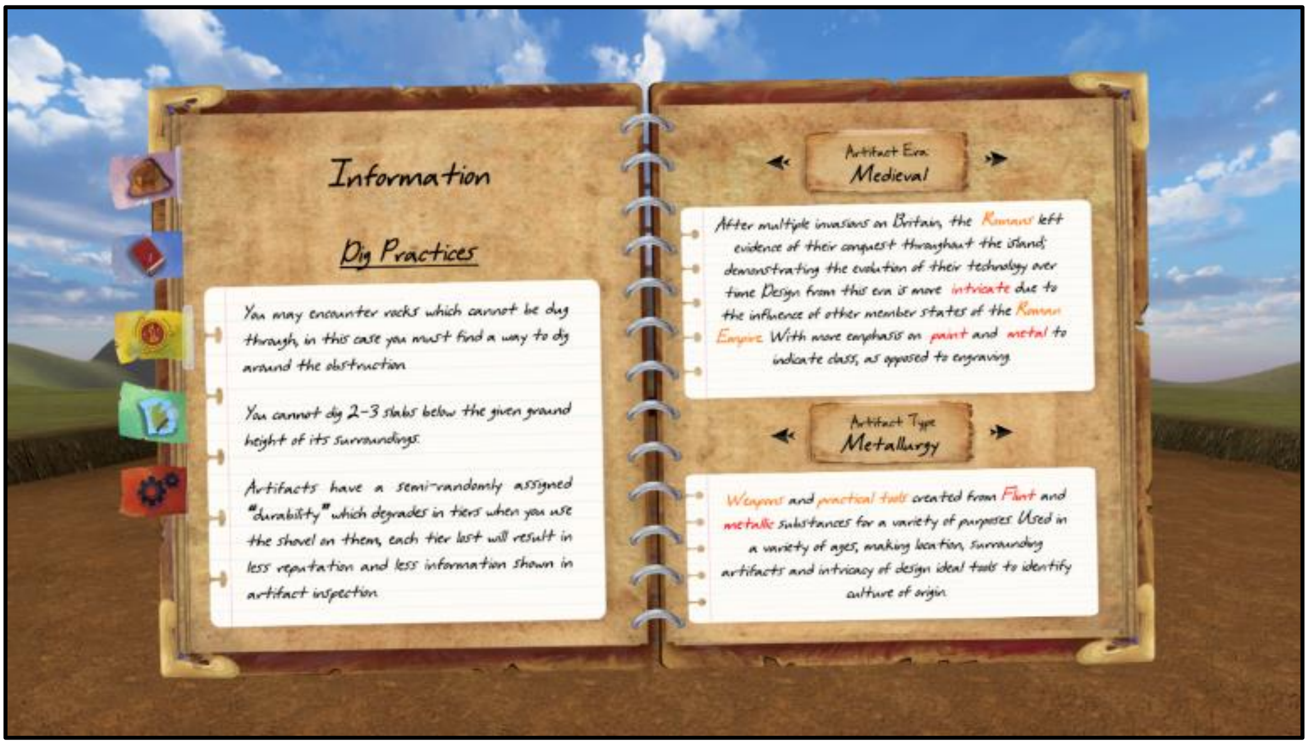






(top left) The right page of the Information Page – as you can see, this initially showed all the eras and artifact "functions" on the same page, resulting in a cramped area of text.

Ben and I made many designs to explore the best way of presenting the artifact information in a clean and interesting fashion



Information

Dig Practices

You may encounter rocks which cannot be dug through, in this case you must find a way to dig around the obstruction

You cannot dig 2-3 slabs below the given ground height of its surroundings.

Artifacts have a semi-randomly assigned "durability" which degrades in tiers when you use the shovel on them, each tier lost will result in less reputation and less information shown in artifact inspection

Artifact Era:
Medieval

After multiple invasions on Britain, the Romans left evidence of their conquest throughout the island; demonstrating the evolution of their technology over time Design from this era is more intricate due to the influence of other member states of the Roman Empire. With more emphasis on paint and metal to indicate class, as opposed to engraving

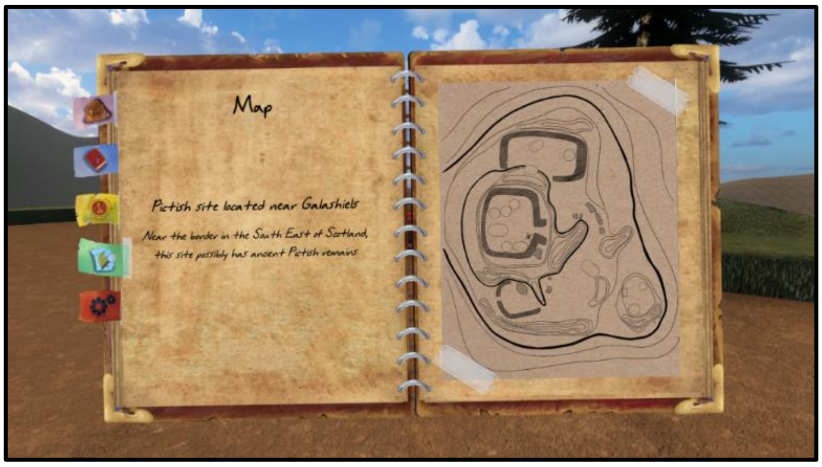
Artifact Type
Metallurgy

Weapons and practical tools created from Flint and metallic substances for a variety of purposes Used in a variety of ages, making location, surrounding artifacts and intricacy of design ideal tools to identify culture of origin

The final design of the information page – the player can switch through each era and type of artifact which reveals different information. The text is displayed over white lined paper, increasing legibility.

The Site Page

The **Site Page** is very simple – It displays the name and information of the site on the left page, and an image of the site map on the right page. This page is linked to a Scriptable Object named "SiteInfoSO", which contains all the information of the site, including its map image. If we made more levels, this page would change to fit the current dig site information.



Map

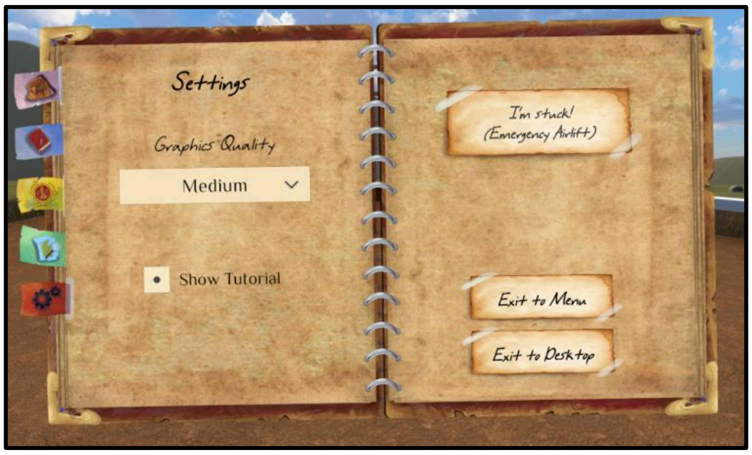
Pictish site located near Galashiels

Near the border in the South East of Scotland, this site possibly has ancient Pictish remains

The final version of the Map Page

Implementing settings into the Settings Page

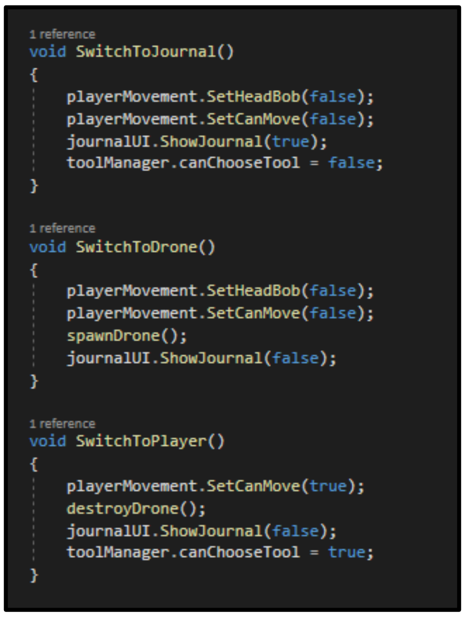
James created the initial **Settings Page** using my system and added two buttons for the player to exit to the main menu, or to exit to the desktop. I added the graphics settings, the "show tutorial" toggle and the "I'm stuck" button which is linked to Ben's **ResetPosition** script.



The final version of the Settings Page

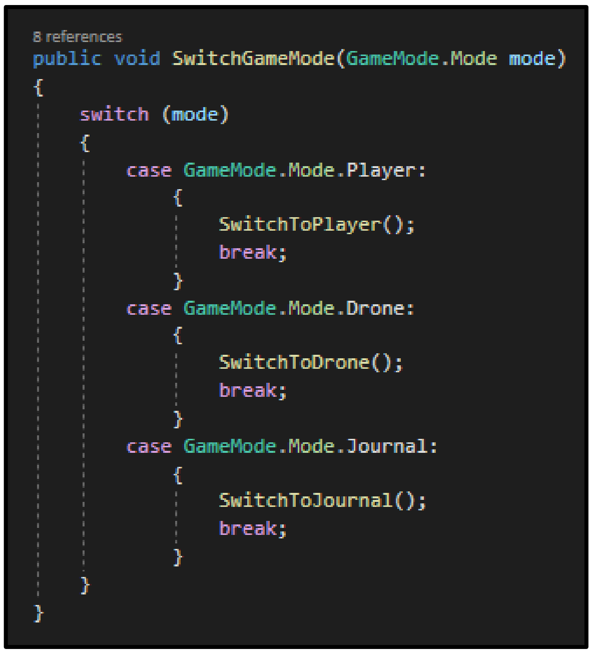
Game Mode System

Within the level of the game there are 3 "modes": The **Player** mode, the **Drone** mode and the **Journal** mode. These modes use different inputs and change many aspects of the game, so I needed to create a script that managed the different states and changes. I created the **ChangeMode** script to manage these states.



```
1 reference
void SwitchToJournal()
{
    playerMovement.SetHeadBob(false);
    playerMovement.SetCanMove(false);
    journalUI.ShowJournal(true);
    toolManager.canChooseTool = false;
}

1 reference
void SwitchToDrone()
{
    playerMovement.SetHeadBob(false);
    playerMovement.SetCanMove(false);
    spawnDrone();
    journalUI.ShowJournal(false);
}

1 reference
void SwitchToPlayer()
{
    playerMovement.SetCanMove(true);
    destroyDrone();
    journalUI.ShowJournal(false);
    toolManager.canChooseTool = true;
}
```

Functions within the ChangeMode script that change aspects of the game depending on the current "mode" that is selected



```
8 references
public void SwitchGameMode(GameMode.Mode mode)
{
    switch (mode)
    {
        case GameMode.Mode.Player:
            {
                SwitchToPlayer();
                break;
            }
        case GameMode.Mode.Drone:
            {
                SwitchToDrone();
                break;
            }
        case GameMode.Mode.Journal:
            {
                SwitchToJournal();
                break;
            }
    }
}
```

The function SwitchGameMode() changes the mode depending on the Enumerator "GameMode" which is changed from other scripts.

The Popup System

Types of Popups

We needed to guide the player through the level of the game with a tutorial, so I created a **Popup System** to be able to display information to the player in different ways.

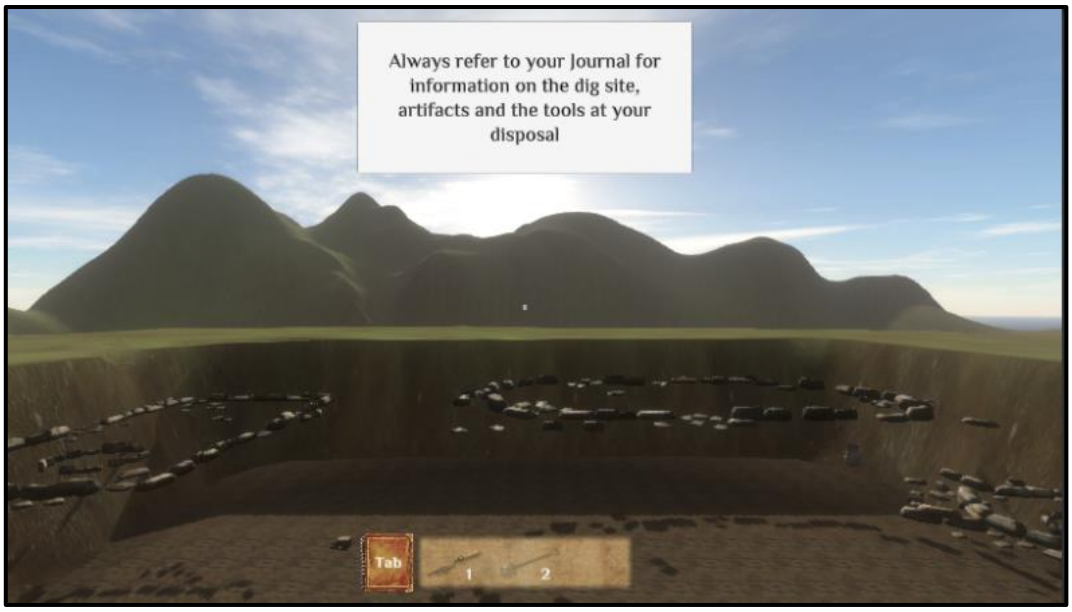
There are 4 different types of Popups:

Window Popups display in the centre of the screen and pause the game. A message is displayed within the window with an "OK" button that closes the Popups and resumes the game when pressed.

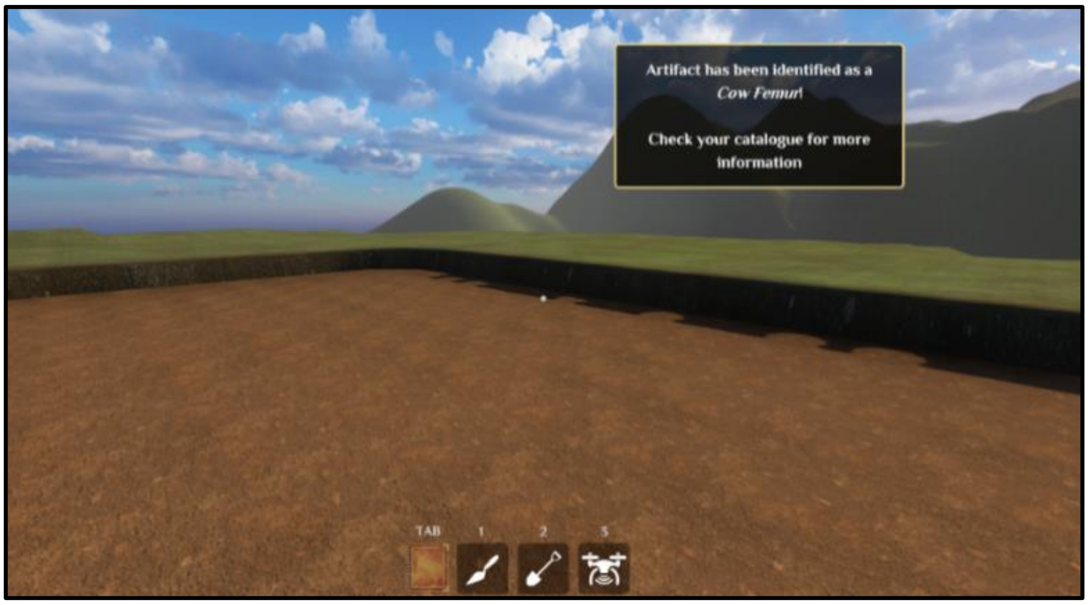
Passive Popups can be displayed anywhere on the screen for a specified amount of time. They don't pause the game.

Large Passive Popups work functionally the same as **Passive Popups but** are larger so that they can accommodate more information if needed.

Confirmation Popups are like **Window Popups** – they display in the centre of the screen and pause the game. The difference is that the player has two options: "Yes" and "No". Pressing "Yes" executes the function that is stored within the popups, and "No" simply closes the popup without calling any function. This means that if the player makes an important decision, the game asks them for confirmation.

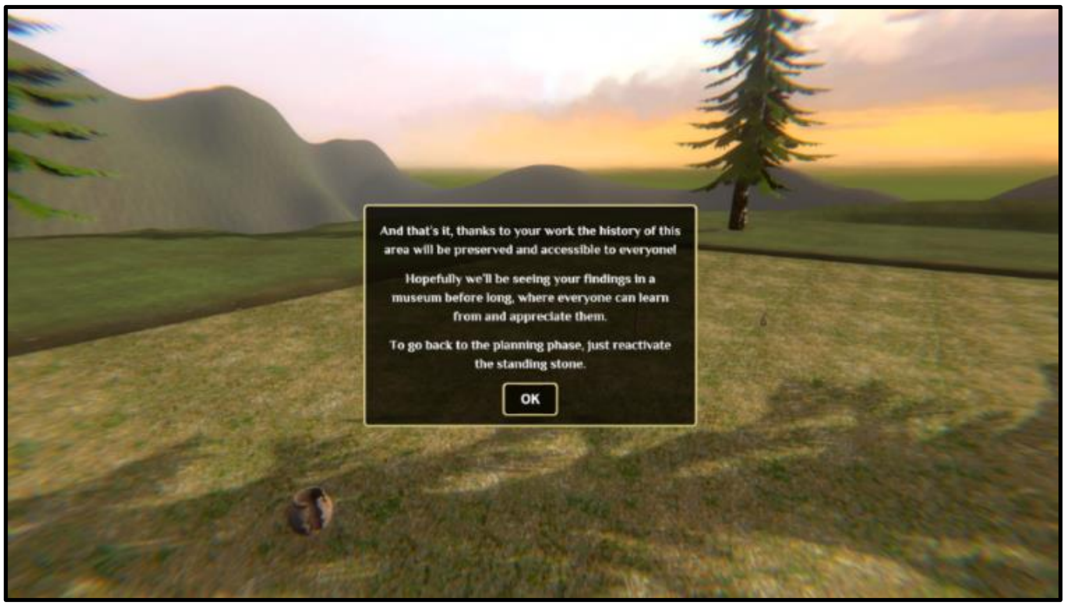

Always refer to your Journal for information on the dig site, artifacts and the tools at your disposal

A prototype of the Passive Popup


Artifact has been identified as a *Cow Femur*!

Check your catalogue for more information

An example of a Passive Popup that notifies the player that an artifact has been analysed without interrupting gameplay.

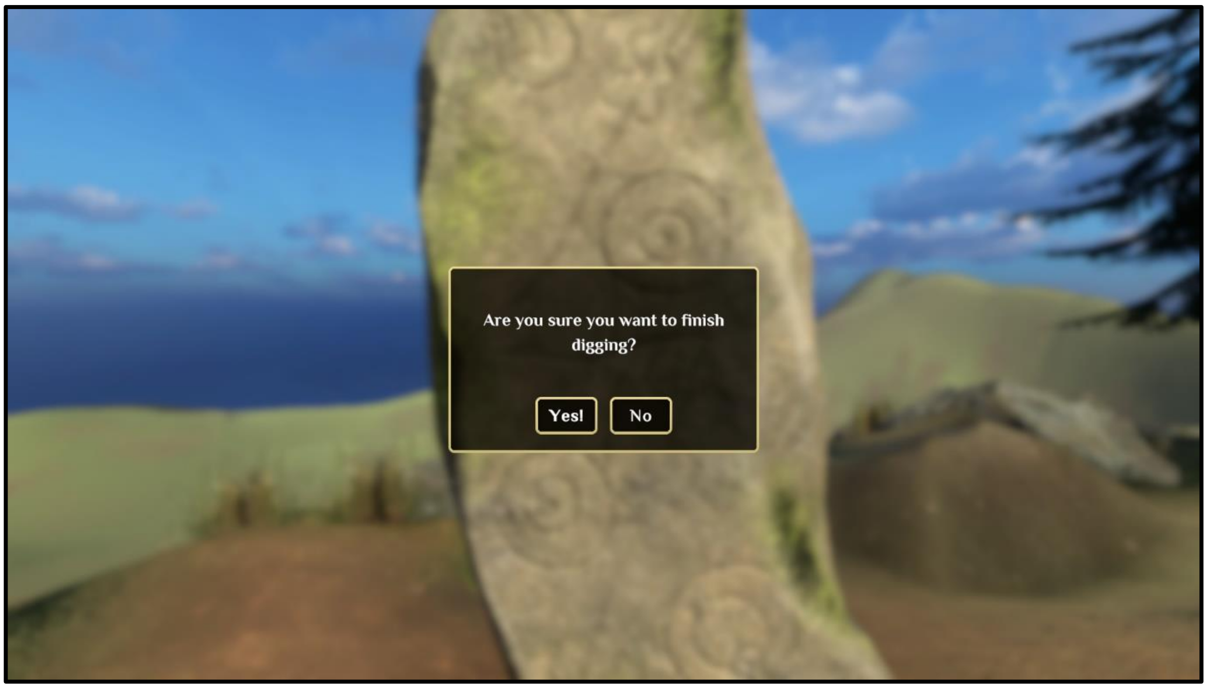


And that's it, thanks to your work the history of this area will be preserved and accessible to everyone!

Hopefully we'll be seeing your findings in a museum before long, where everyone can learn from and appreciate them.

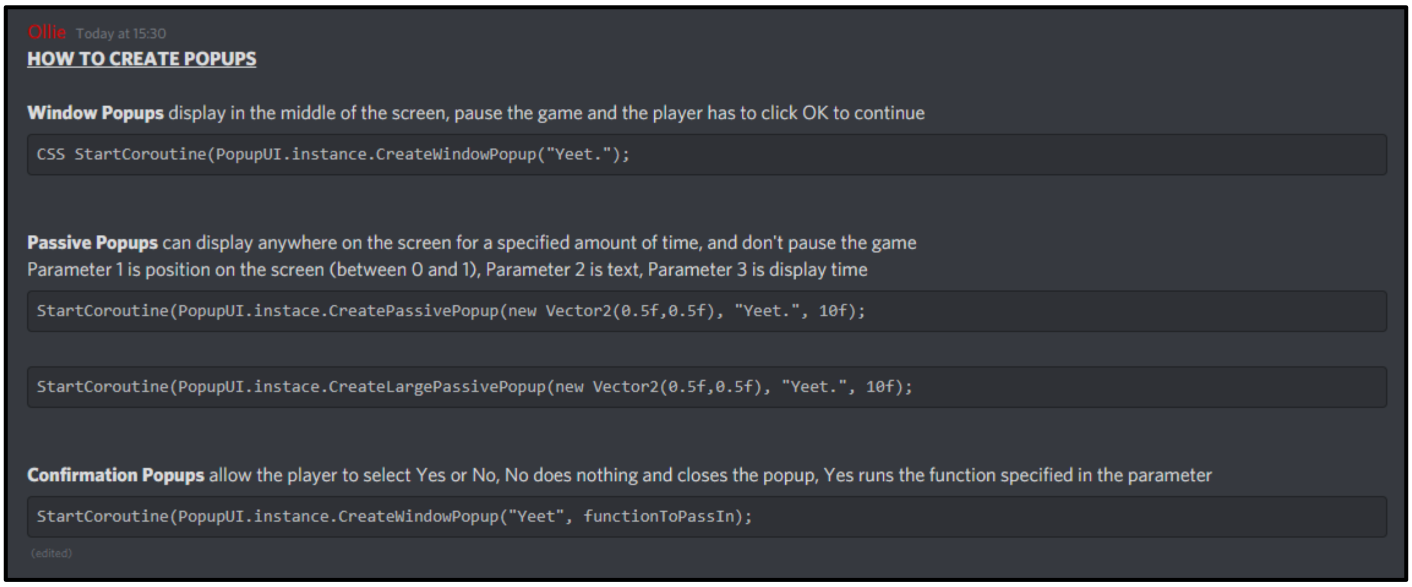
To go back to the planning phase, just reactivate the standing stone.

A Window Popup that displays information and pauses the game, resuming it once the player has pressed the "OK" button.



Are you sure you want to finish digging?

A Confirmation Popup that asks if the player is sure they want to end the level and visit the recreation of the dig site in the past.



Ollie Today at 15:30

HOW TO CREATE POPUPS

Window Popups display in the middle of the screen, pause the game and the player has to click OK to continue

```
CSS StartCoroutine(PopupUI.instance.CreateWindowPopup("Yeet.");
```

Passive Popups can display anywhere on the screen for a specified amount of time, and don't pause the game
Parameter 1 is position on the screen (between 0 and 1), Parameter 2 is text, Parameter 3 is display time

```
StartCoroutine(PopupUI.instace.CreatePassivePopup(new Vector2(0.5f,0.5f), "Yeet.", 10f);
```

```
StartCoroutine(PopupUI.instace.CreateLargePassivePopup(new Vector2(0.5f,0.5f), "Yeet.", 10f);
```

Confirmation Popups allow the player to select Yes or No, No does nothing and closes the popup, Yes runs the function specified in the parameter

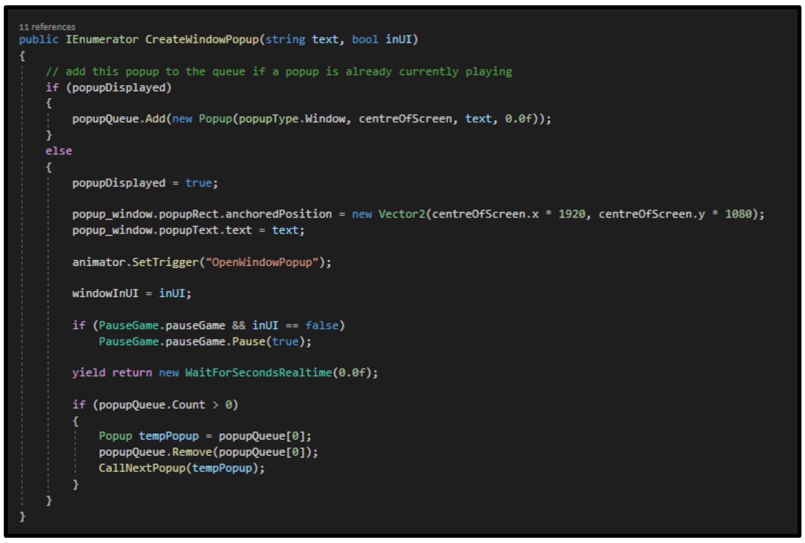
```
StartCoroutine(PopupUI.instance.CreateWindowPopup("Yeet", functionToPassIn);
```

(edited)

A guide I sent to the team's Discord chat showing them how to create any type of Popup from any script.

<u>The Queue System</u>

I created a queue system that stores pending Popups in a List data structure. This means that if many Popups are called at the same time, they display one after the other, rather than all at once. The Popup System is very flexible due to this and works very well with the rest of the scripts within the game.

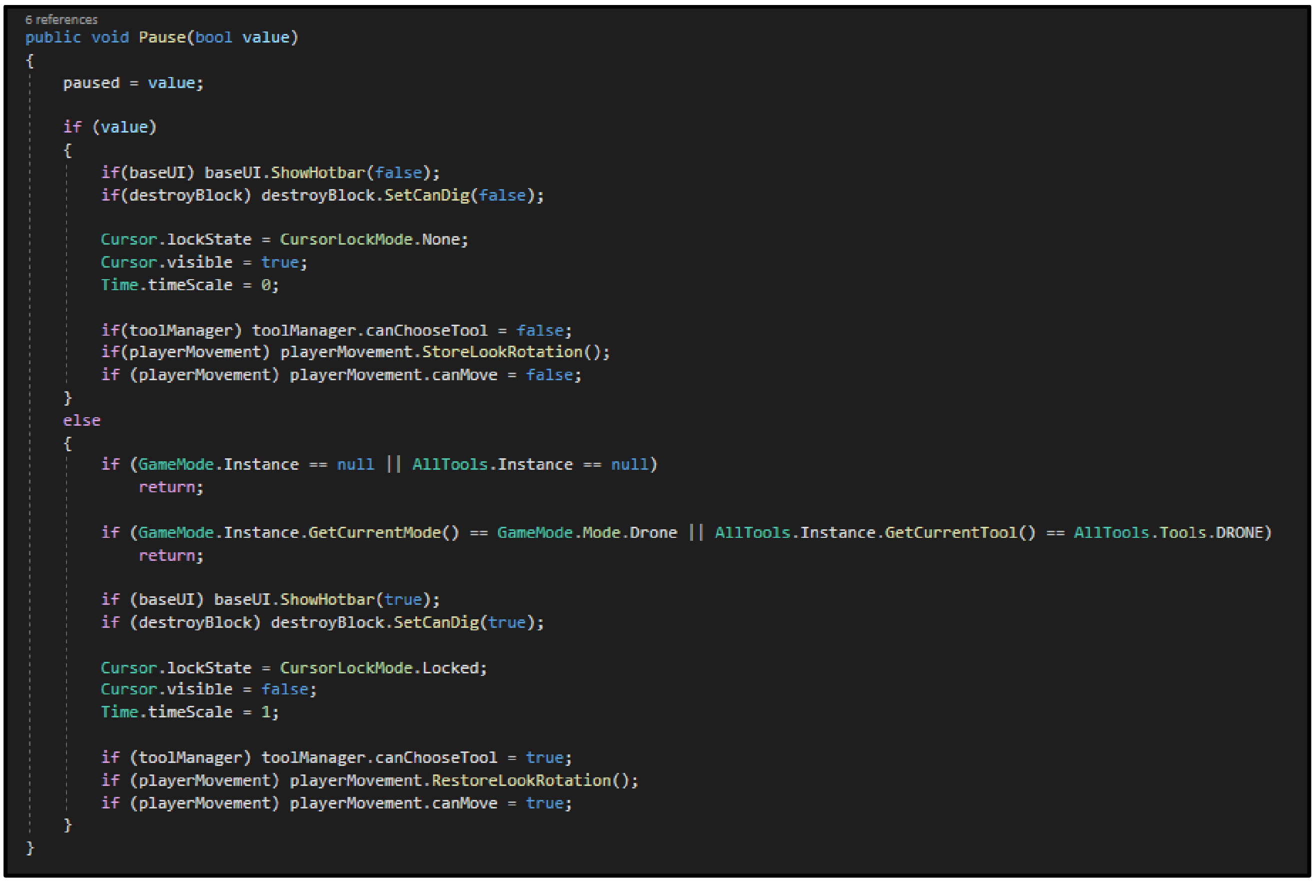


```
11 references
public IEnumerator CreateWindowPopup(string text, bool inUI)
{
    // add this popup to the queue if a popup is already currently playing
    if (popupDisplayed)
    {
        popupQueue.Add(new Popup(popupType.Window, centreOfScreen, text, 0.0f));
    }
    else
    {
        popupDisplayed = true;

        popup_window.popupRect.anchoredPosition = new Vector2(centreOfScreen.x * 1920, centreOfScreen.y * 1080);
        popup_window.popupText.text = text;

        animator.SetTrigger("OpenWindowPopup");

        windowInUI = inUI;

        if (PauseGame.pauseGame && inUI == false)
            PauseGame.pauseGame.Pause(true);

        yield return new WaitForSecondsRealtime(0.0f);

        if (popupQueue.Count > 0)
        {
            Popup tempPopup = popupQueue[0];
            popupQueue.Remove(popupQueue[0]);
            CallNextPopup(tempPopup);
        }
    }
}
```

The CreateWindowPopup() function in the script PopupUI. It adds the Popup to the queue if a Popup is currently being displayed, otherwise it pauses the game and places the Popup in the appropriate place and calls the next Popup in the queue if there is one.

Pausing the Game

I created a script called **PauseGame** which disables any action the player can take until the game is un-paused. It sets the game's timescale to 0, meaning that no time passes, and no code linked to Delta time is executed. This function can be called from other scripts and is used for elements of UI such as the Journal or Window Popups. The player camera's depth-of-field post-processing is also controlled in this script to blur the background.



```
6 references
public void Pause(bool value)
{
    paused = value;

    if (value)
    {
        if(baseUI) baseUI.ShowHotbar(false);
        if(destroyBlock) destroyBlock.SetCanDig(false);

        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        Time.timeScale = 0;

        if(toolManager) toolManager.canChooseTool = false;
        if(playerMovement) playerMovement.StoreLookRotation();
        if (playerMovement) playerMovement.canMove = false;
    }
    else
    {
        if (GameMode.Instance == null || AllTools.Instance == null)
            return;

        if (GameMode.Instance.GetCurrentMode() == GameMode.Mode.Drone || AllTools.Instance.GetCurrentTool() == AllTools.Tools.DRONE)
            return;

        if (baseUI) baseUI.ShowHotbar(true);
        if (destroyBlock) destroyBlock.SetCanDig(true);

        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        Time.timeScale = 1;

        if (toolManager) toolManager.canChooseTool = true;
        if (playerMovement) playerMovement.RestoreLookRotation();
        if (playerMovement) playerMovement.canMove = true;
    }
}
```

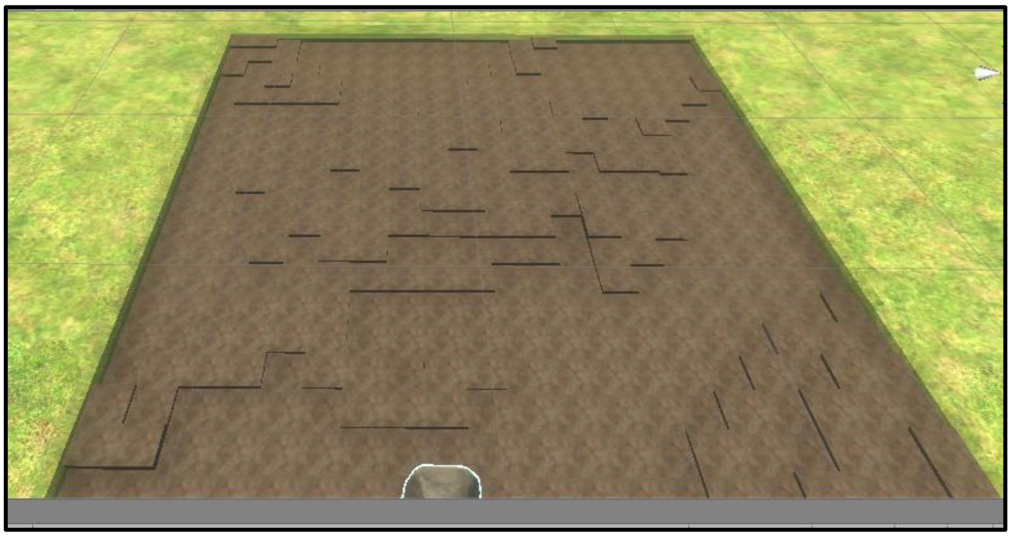
The Pause() function of the script PauseGame. This function is used by many other scripts.

Altering the Slab System

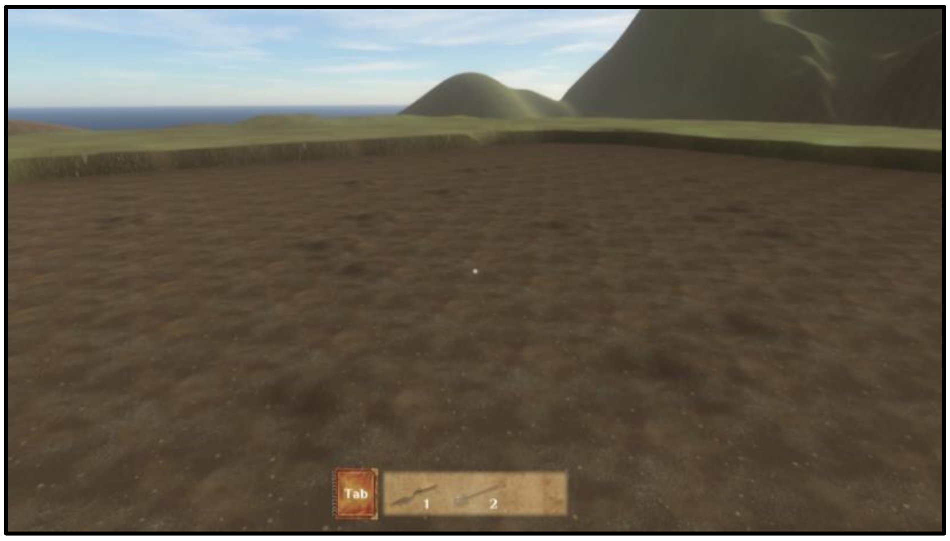
Improving the Top Layer

Our **Dig Site** and its generation system was for the most part created by our programmer, Ben. I made some additions to it to add some visual flourish to the game.

We felt that the top layer of dirt on the dig site seemed very unnatural and flat, so we investigated ways of adding "texture" to it.



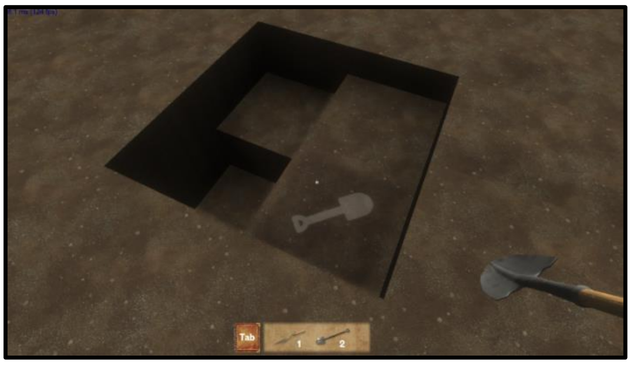
Ben investigated adding heightmaps into the Dig Site Generation System, providing some varied levels of slabs. I felt this wasn't too visually appealing but was a step in the right direction.

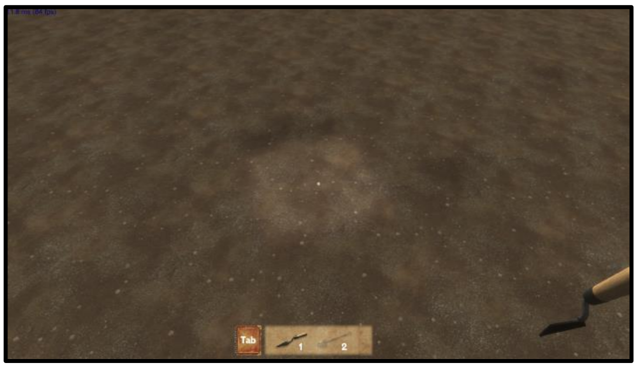


I altered the functionality of the Dig Site Generation System so that the top layer has a random chance of using "bumpy" models of the slab that I created with Blender. This added some subtle, organic texture to the dig site

Highlighting Slabs

I also altered Ben's **DestroyBlock** script so that if a slab can be dug, it is highlighted. This is a quality of life improvement to indicate to the player what they can and can't dig. I experimented with different visuals for the highlight, including showing the current tool within the highlight texture.





Different versions of the slabs being highlighted – after asking the team we decided to stick with the version on the right

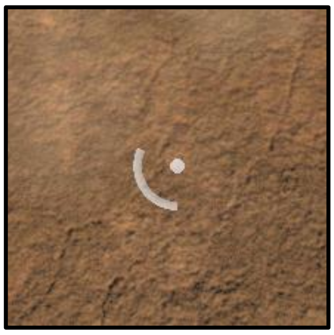
I also animated the slabs so that they "erode" as you dig them, levelling down until they disappear. This was part of our game for a very long time, but after revisiting it, we decided that it was somewhat unsatisfying and didn't work well when using the Shovel, as only the top slab would animate, yet the two slabs below would also disappear.

Digging Particles

I added a Particle System to the slab prefab which animated dust clouds as the player digs it. The particle object is set to disabled and is only enabled once the slab is being dug – this saved a lot of performance.

Dig Progress Indicator

Since there was a set duration that it takes to dig with the Trowel and Shovel, I decided to make a UI element that represents this duration in a simple and slick style, which would provide feedback for the player without ruining the immersion of digging. I created a 360-degree slider that wraps around the reticle in the centre of the screen. This circle completes when the slab has been dug.

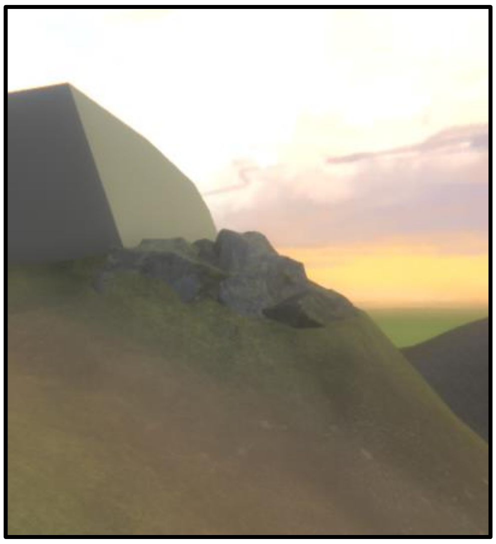


The dig progress indicator in the centre of the screen.

Blender Rocks

In my free time I decided to follow a tutorial to create some simple rocks in the 3D modelling software, Blender. As a placeholder, I put these rocks in our environment, not intending for them to be in the final game. When our environment artist, Kelly, was putting together her assets in-engine, she decided to use these rocks near the entrance to the settlement.

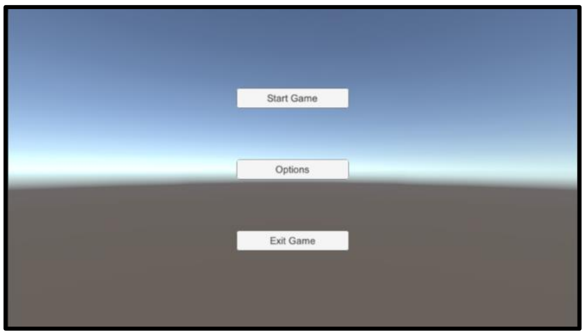
Some simple rocks made in Blender


The rocks in-game

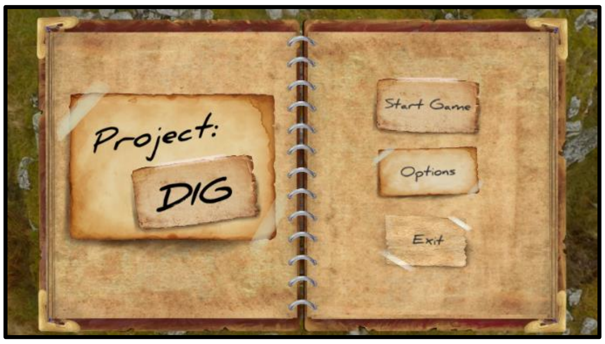
Main Menu

The **Main Menu** has three options, "Start Game", "Options" and "Exit". Pressing "Start Game" takes the player to the Map Screen. "Options" opens a panel where the player can select graphics settings and can choose to disable or enable the tutorials. "Exit" closes the application.

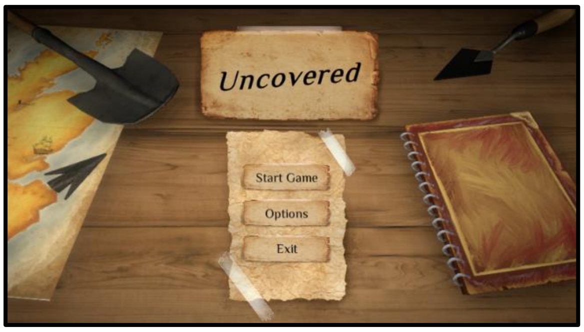
The initial visual of the Main Menu was very simple and empty, but over time I iterated it to carry the visual aesthetic of the rest of the game.



The Main Menu that James created was functional but lacked aesthetic.

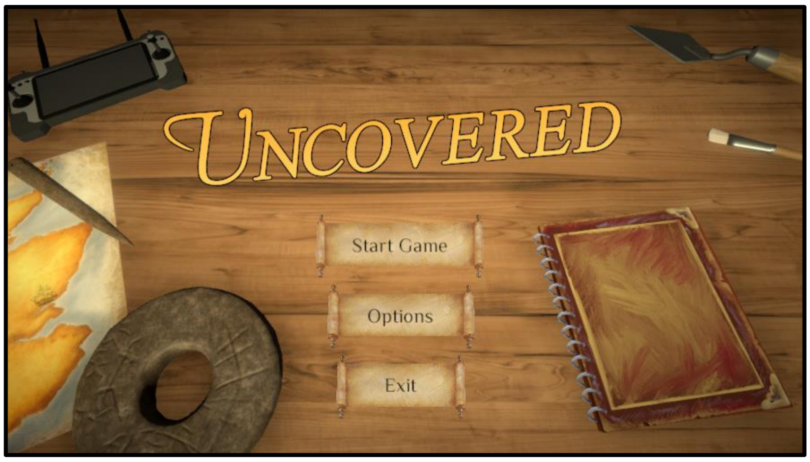


My first iteration of the Main Menu that used the Journal aesthetic.



An overhaul of the Main Menu – animations were included for visual flourish.

The camera pans onto the desk to reveal the title and buttons.



The final version of the Main Menu.

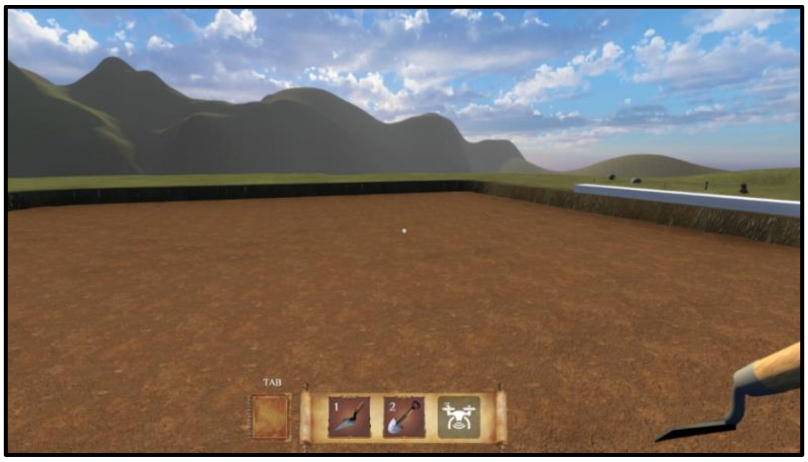
Improving the Map

I modified James' MapMovement script to have linear movement as opposed to constant. Previously the player would move their cursor to the edge of the screen and the Map would move suddenly in that direction. To give the player more control, I modified the code so that the speed was greater the closer the mouse is to the edge of the screen. This choice was inspired from many games with a similar panning system such as The *Sims 3* (2009) or *Sid Meier's Civilization V* (2010).

Occlusion Culling and Lighting

I did a lot of experimentation and tweaking with Unity’s Lighting and Post-Processing settings to improve the overall visual look. I added fog to give the game more atmosphere and experimented with particle systems to add dust particles to the air, although I decided to remove this to aid performance.

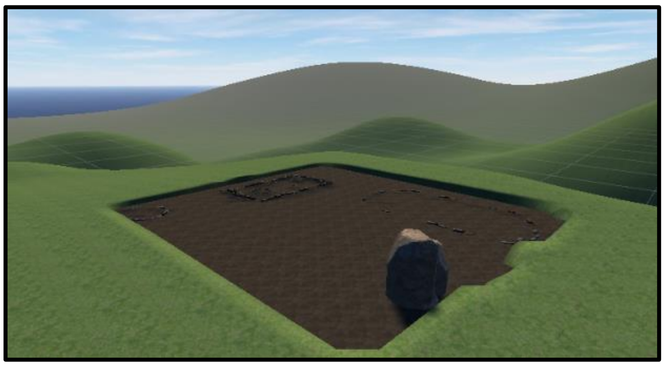
I set up Occlusion Culling so that the slabs within the dig site are only rendered if they are visible to the player. This greatly helped performance as it ensured that the game was only rendering what the player needed to see at the time.



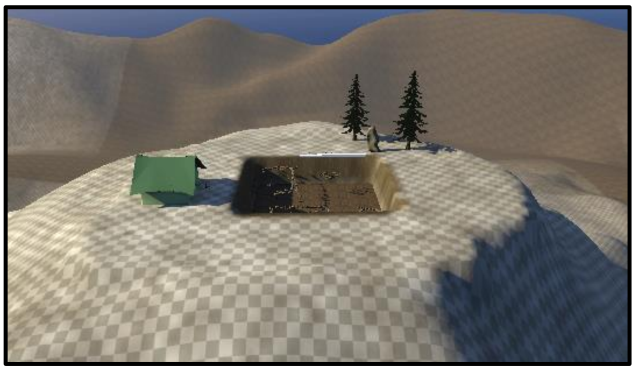
A screenshot of the dig site. You can see the fog on the hills.

Creation and Modification of Terrain

I created the initial terrain for the dig site on the hill as a placeholder, and our environment artist, Kelly, created a much better version of the terrain which I modified to accommodate the dig site.



The early terrain made by me.



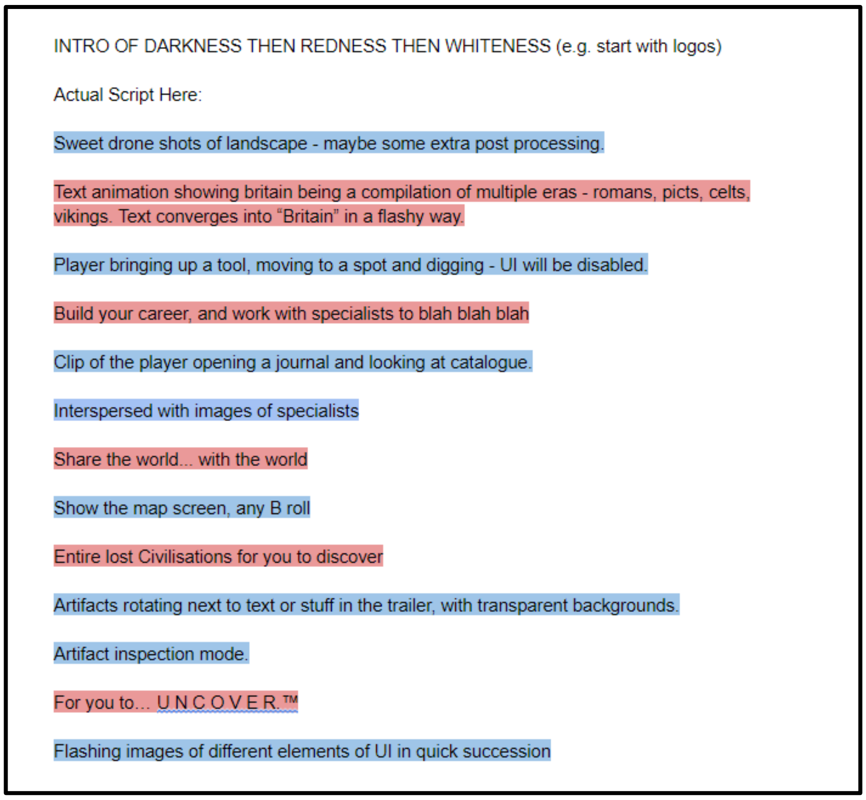
I had to create a square hole in the top of the hill to accommodate the dig site area, which would populate the hole with slabs.

AudioManager

I created a script named **AudioManager** that handles each audio clip within the level. It is a singleton script, meaning that there is a single instance of it that can be accessed easily by any other script. Another script can simply call a function within *this* script and the associated sound will play. This was very useful for providing a main hub that any script can refer to for playing sound effects.

The Trailer

I gathered "B-roll" footage for the trailer with hidden UI, footage from the cinematic camera in the present, and footage from the cinematic camera in the past. Adobe Premiere Pro was used to cut together the footage, and the shots were planned out by myself and Robert using a Google Doc to note down each sequence of the trailer. I waited for enough assets to be in the game to make the trailer, but this was very late, and I ended up having to edit it quickly. I would have preferred to have had more time to improve the trailer with better footage in order to effectively promote the game.



INTRO OF DARKNESS THEN REDNESS THEN WHITENESS (e.g. start with logos)

Actual Script Here:

Sweet drone shots of landscape - maybe some extra post processing.

Text animation showing britain being a compilation of multiple eras - romans, picts, celts, vikings. Text converges into "Britain" in a flashy way.

Player bringing up a tool, moving to a spot and digging - UI will be disabled.

Build your career, and work with specialists to blah blah blah

Clip of the player opening a journal and looking at catalogue.

Interspersed with images of specialists

Share the world... with the world

Show the map screen, any B roll

Entire lost Civilisations for you to discover

Artifacts rotating next to text or stuff in the trailer, with transparent backgrounds.

Artifact inspection mode.

For you to... U N C O V E R.™

Flashing images of different elements of UI in quick succession

The trailer script created by Robert and me.

Conclusion

Contributing to the development of 'Uncovered' for the Society of Antiquaries of Scotland has been an incredible learning experience. I feel that my skills in programming, design and working within a team have greatly improved since we began development in January. Overall, I am very happy with what we managed to achieve within a limited timescale. If asked to develop the game further, I would enjoy the challenge of fine tuning many of the features we created, and I look forward to other future collaborative projects.

References

- *Minecraft*. 2011. [disk]. Microsoft Windows, Mac OSX, Linux. Mojang.
- *Resident Evil 7: Biohazard*. 2017. [disk]. Microsoft Windows, PlayStation 4, Xbox One, Nintendo Switch. Capcom.
- *Tomb Raider*. 2013. [disk]. Microsoft Windows, PlayStation 3, Xbox 360. Crystal Dynamics.
- *Watch Dogs 2*. 2016. [disk]. Microsoft Windows, PlayStation 4, Xbox One.
- *Call of Duty: Warzone.* 2020. [disk]. Microsoft Windows, PlayStation 4, Xbox One. Infinity Ward.
- *Overwatch.* 2016. [disk]. Microsoft Windows, PlayStation 4, Xbox One. Blizzard Entertainment.
- *The Elder Scrolls IV: Oblivion.* 2006. [disk]. Microsoft Windows, PlayStation 3, Xbox 360. Bethesda Softworks.
- *Uncharted 4: A Thief's End.* 2017. [disk]. PlayStation 4. Naughty Dog.
- *Jason Weinmann*. [online]. Available from https://www.youtube.com/channel/UCX_b3NNQN5bzExm-22-NVVg.
- *Sebastian Lague*. [online]. Available from https://www.youtube.com/user/Cercopithecan.
- *Firewatch.* 2016. [disk]. Microsoft Windows, PlayStation 4, Xbox One, Nintendo Switch. Campo Santo.
- *The Sims 3*. 2009. [disk]. Microsoft Windows, PlayStation 3, Xbox 360. Maxis.
- *Sid Meier's Civilisation V*. 2010. [disk]. Microsoft Windows, OS X, Linux. Firaxis Games.