Contents

1.0 Introduction

1.1. Setting

The scene will start with the player outside in a forest in a perpetual foggy night cycle with a spooky house in front of them creating a suspenseful atmosphere. They will then approach the house triggering a floodlight. Upon entering the house, the front door will lock, and they will have to complete the following steps, with various elements increasing the intensity of the atmosphere, to escape the house and trigger a daytime cycle, completely changing the mood of the scene

1.2. Scene flow

1. Approach house triggering a floodlight.
2. Enter house and doors lock (exterior sounds get quieter and interior sounds volume increases) – loud clock ticking adding suspense.
3. The cuckoo clock goes off when nearby surprising the player and then dropping it's cuckoo.
4. Carry the cuckoo upstairs.
5. Enter bedroom, bedroom door locks adding suspense.
6. Throw the cuckoo in the bedroom fire increasing intensity – A vase appears.
7. Pick up the bathroom key from the bedside table.
8. Exit bedroom via the en-suite bathroom.
9. Throw the vase down the stairs, smashing it to reveal the lounge key.
10. Enter the lounge – the piano starts playing a spooky music loop to add uncertainty to the atmosphere.
11. Pick up the fruit bowl.
12. Approach the lounge fire – Tv comes on adding to the unsure atmosphere and making the player uneasy.
13. Throw the fruit bowl in the lounge fire, increasing intensity and melting the candle. – Kitchen key appears.
14. Enter the kitchen – serving hatch doors start clattering startling the player.
15. Exit the kitchen to the conservatory.
16. Collect the conservatory key from the pool table.
17. Exit conservatory triggering daytime the mood of the scene changes to reflect the escape.
18. Display Message – "You escaped!"

2.0 Project plan

2.1. Initial Project Plan

Below is my initial project plan (Figure 2.1) on Trello (Atlassian, 2021) which is available at the following link - https://trello.com/b/x2qX9eYN/p3d-asignment.

All expandable cards are also shown below (Figures 2.2 – 2.7)

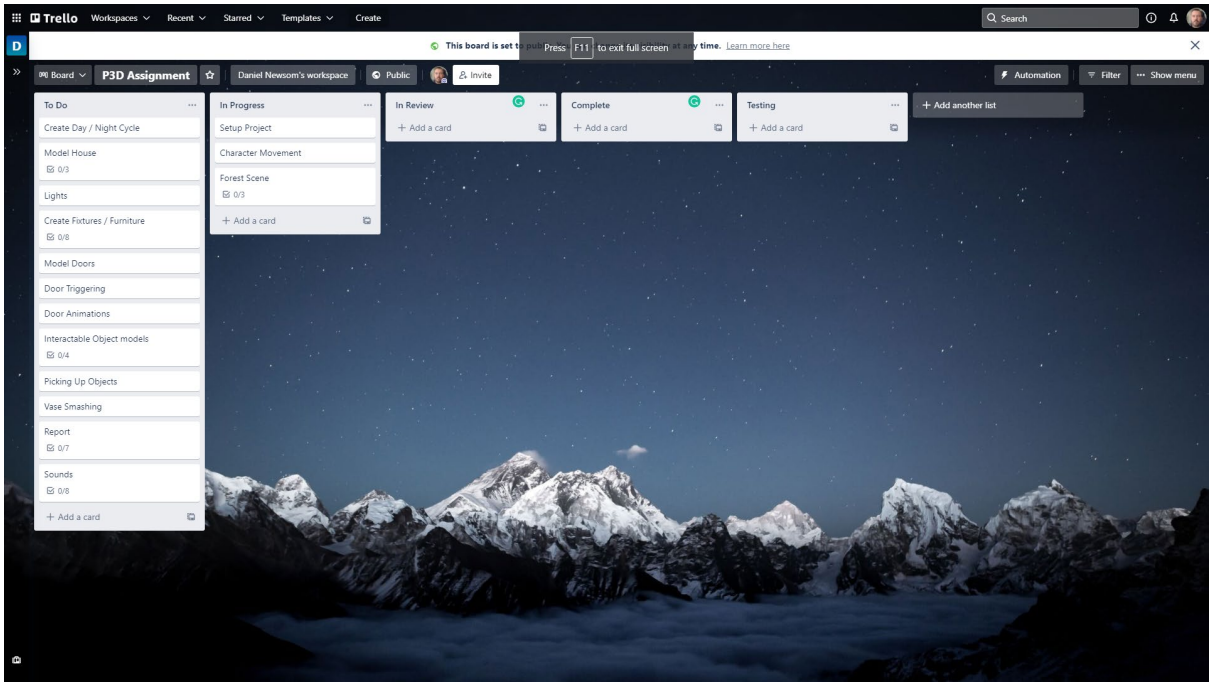

Figure 2.1 – Initial project plan on Trello (Atlassian, 2021)

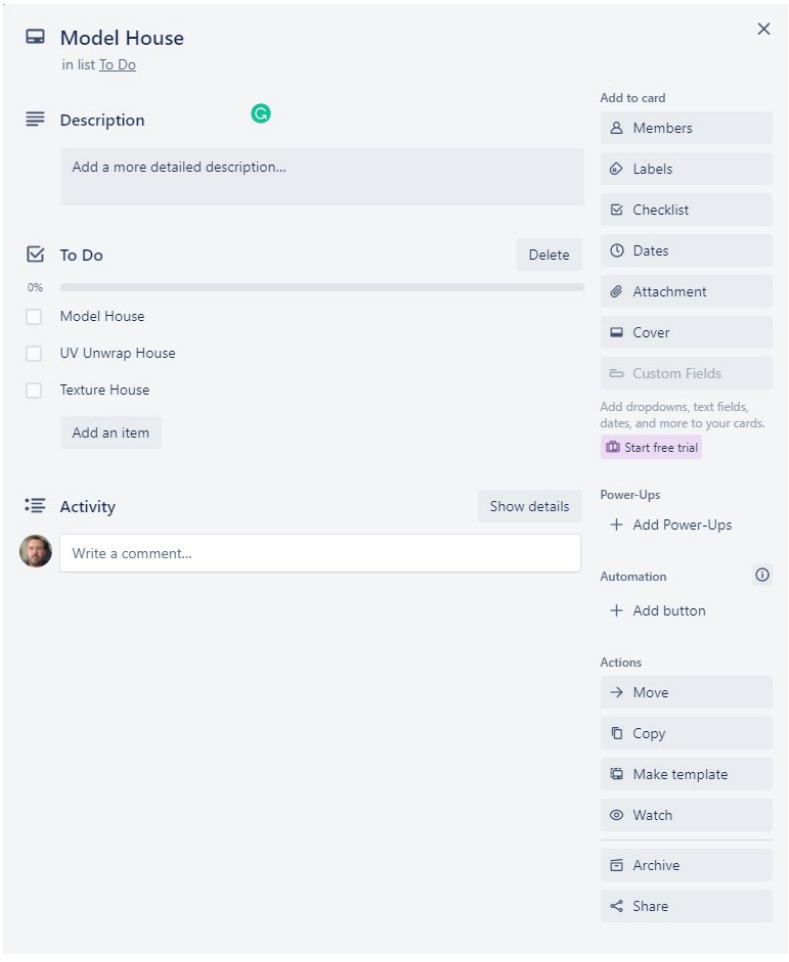

Figure 2.2 – Model house card

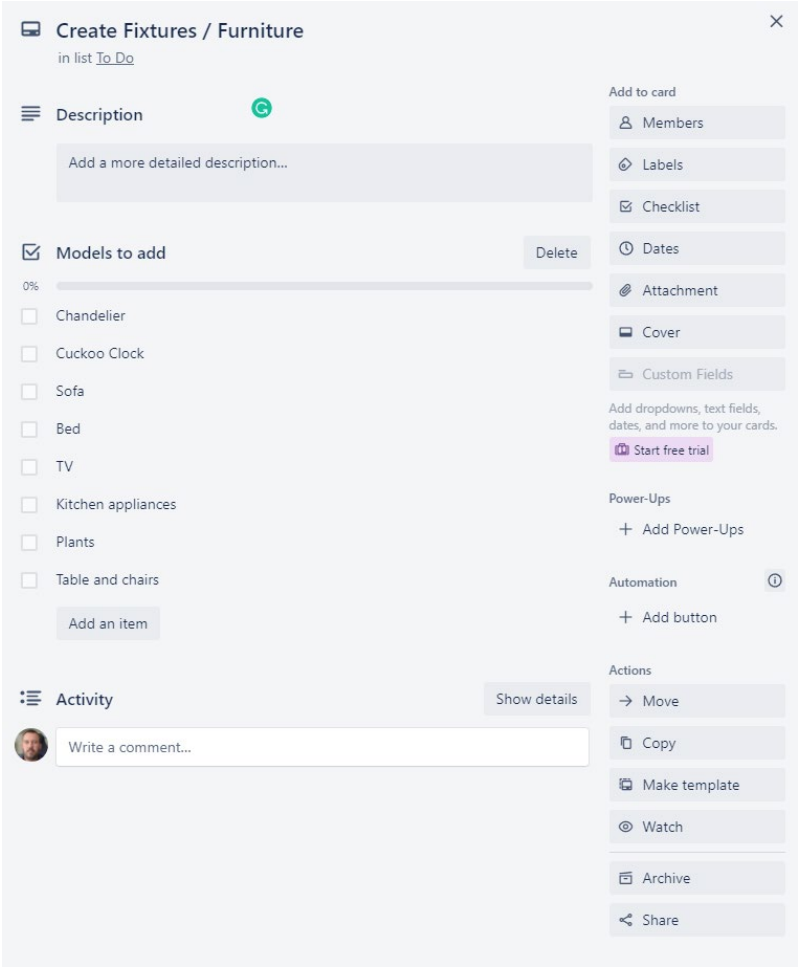

Figure 2.3 – Create Fixtures / Furniture card

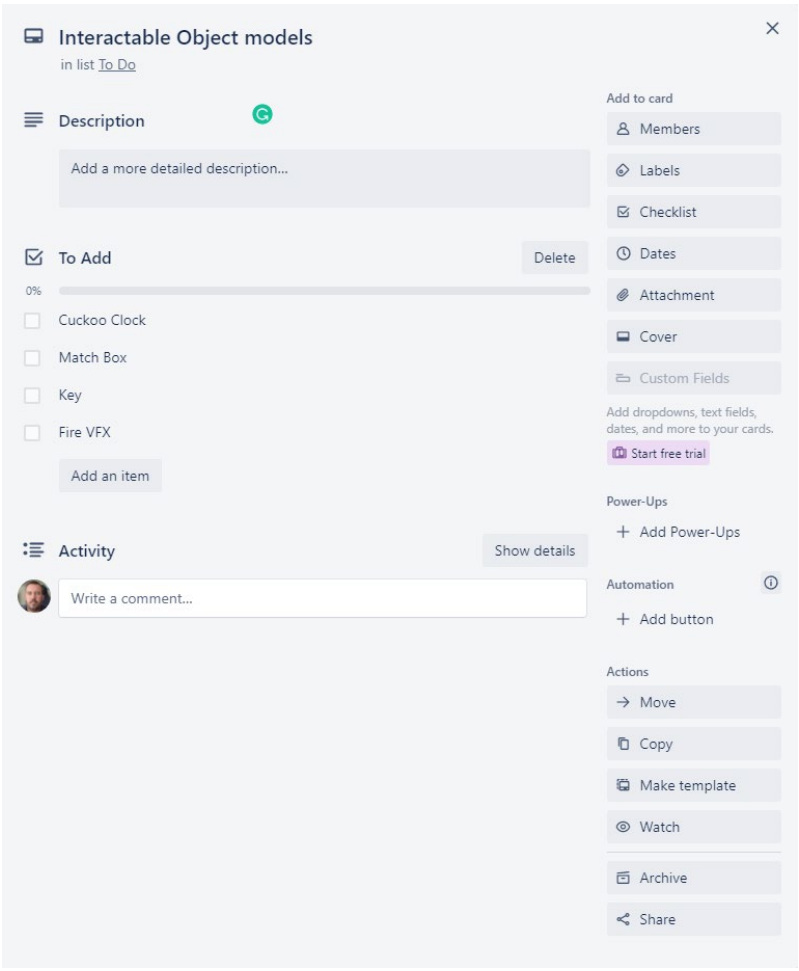

Figure 2.4 – Interactable Object Models card

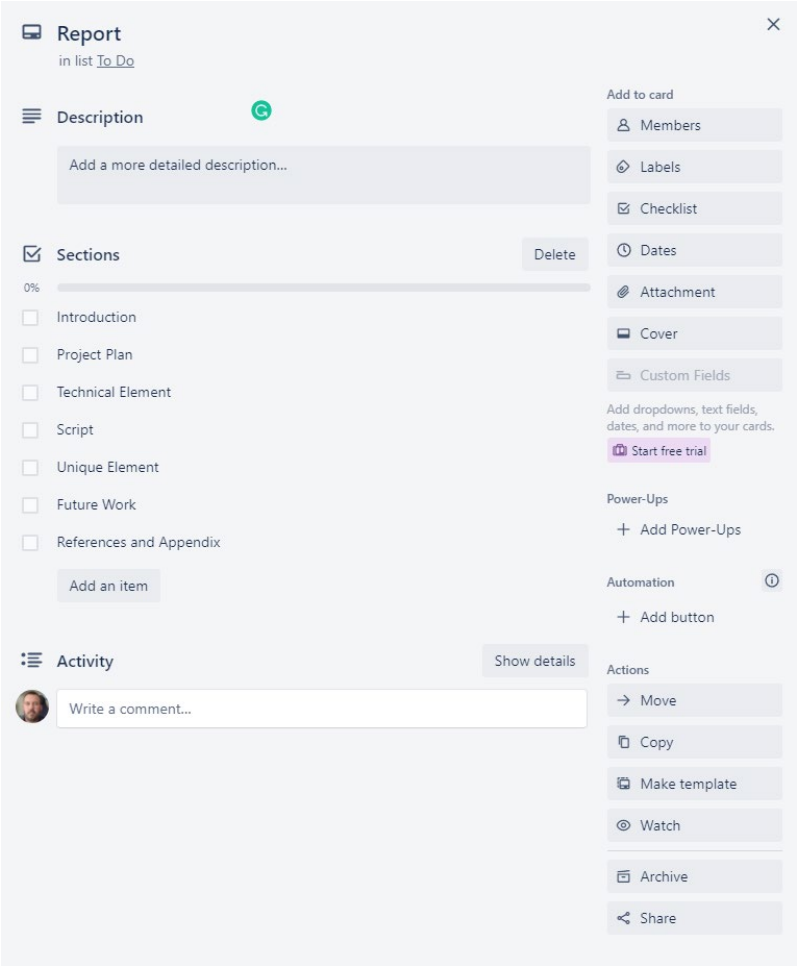

Figure 2.5 – Report card

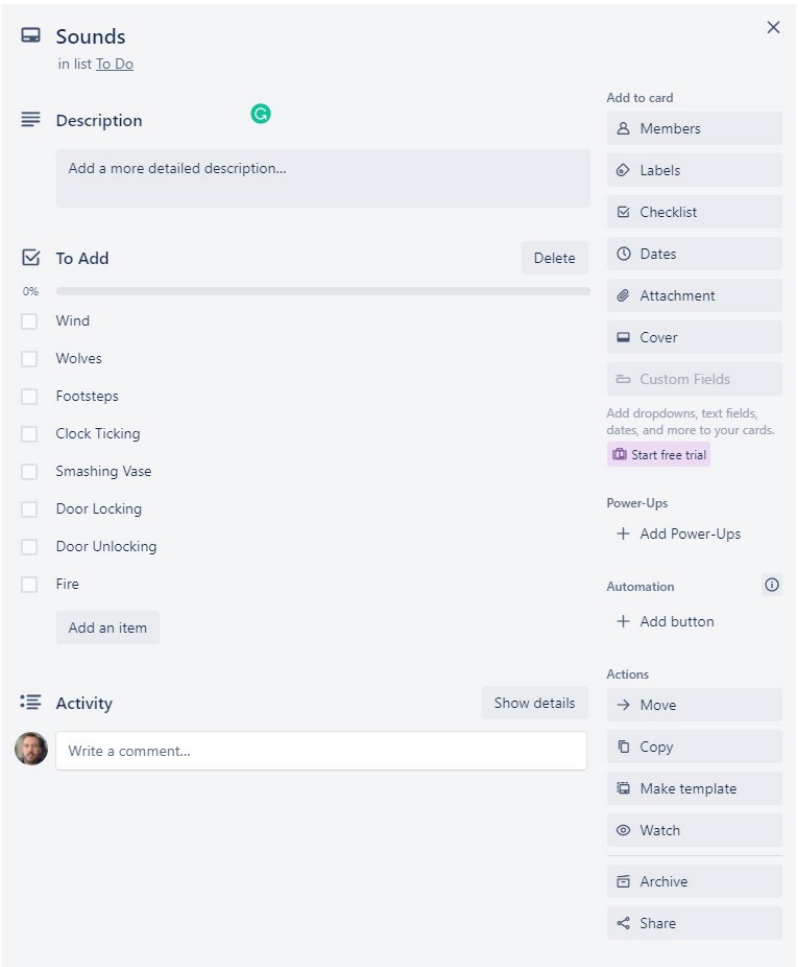

Figure 2.6 – Sounds card

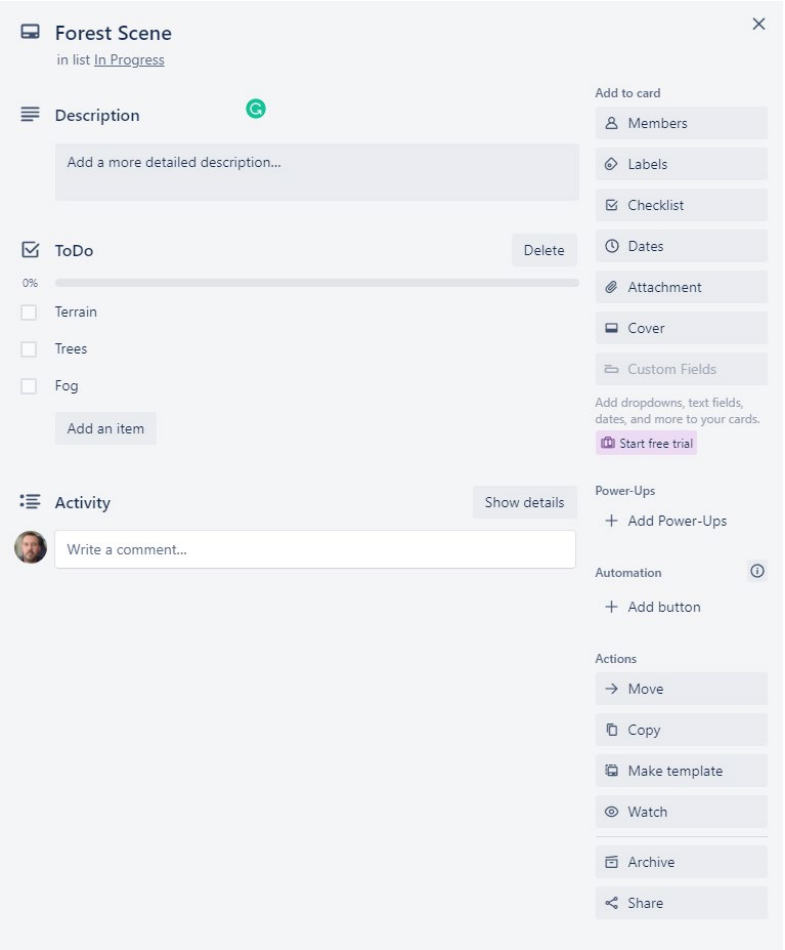

Figure 2.7 – Forest Scene card

2.2. Week 1

The plan for week 1 is as follows

- Setup the Unity (Unity, 2021) project using the high definition render pipeline (HDRP).
- Create the main scene and add a Terrain.
- Paint textures on the terrain from the Forest Environment - Dynamic Nature (NatureManufacture_Forest_Environment, 2021) asset pack.
- Add Trees on the terrain using the Mountain Trees - Dynamic Nature (NatureManufacture_Mountain_Trees, 2021) asset pack.
- Implement the character's movement using the Starter Assets - First Person Character Controller (Unity_Starter_Assets, 2021) as a base to build upon.
- Start Modelling house using Blender (Blender, 2021)

The updated Trello (Atlassian, 2021) board at the end of week 1 is shown below (Figure 2.8) as well as the relevant expanded forest scene card (Figures 2.9) and the report card (Figure 2.10).

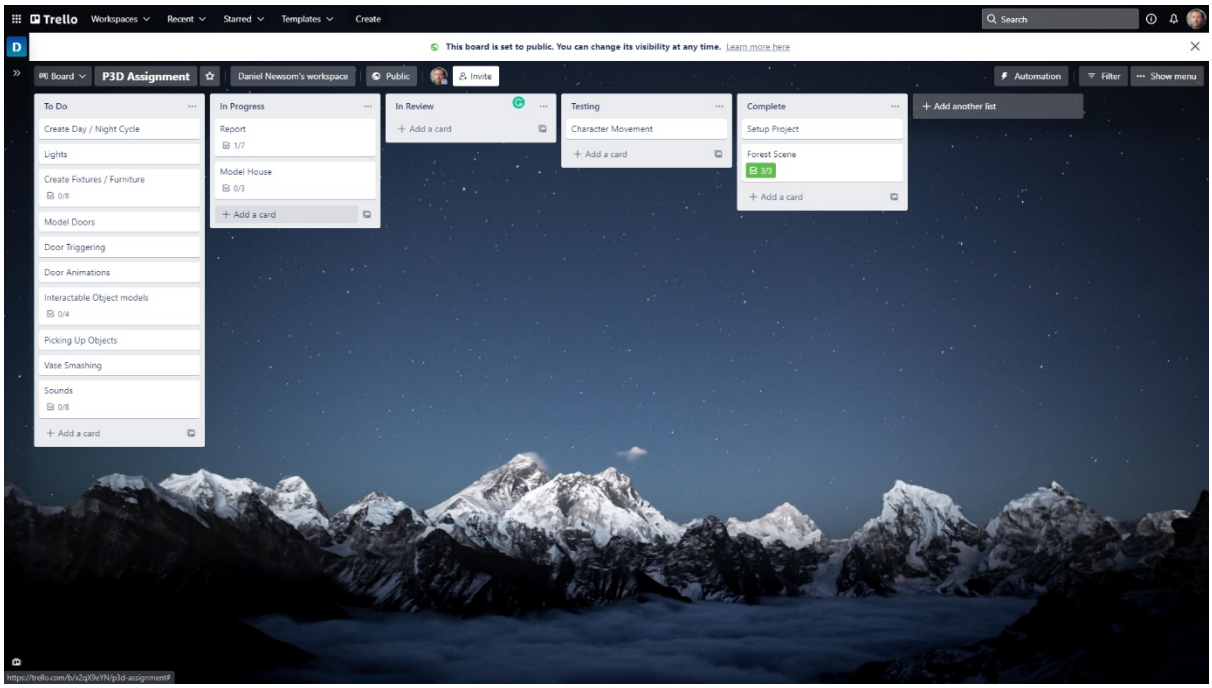

Figure 2.8 – Trello (Atlassian, 2021) board status at the end of week 1

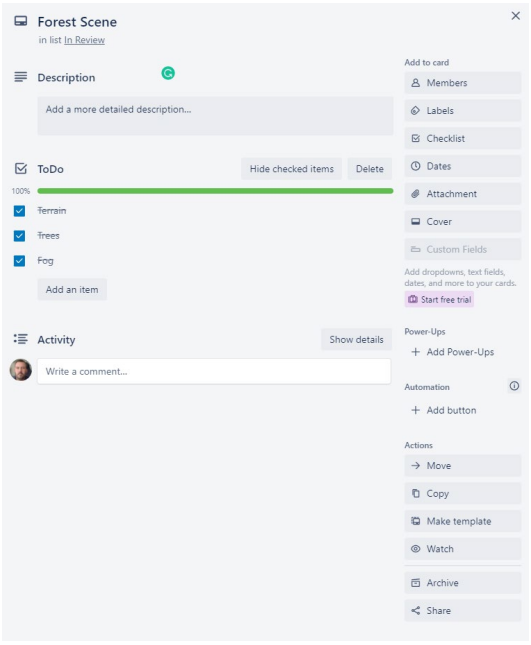

Figure 2.9 – Forest scene card updated

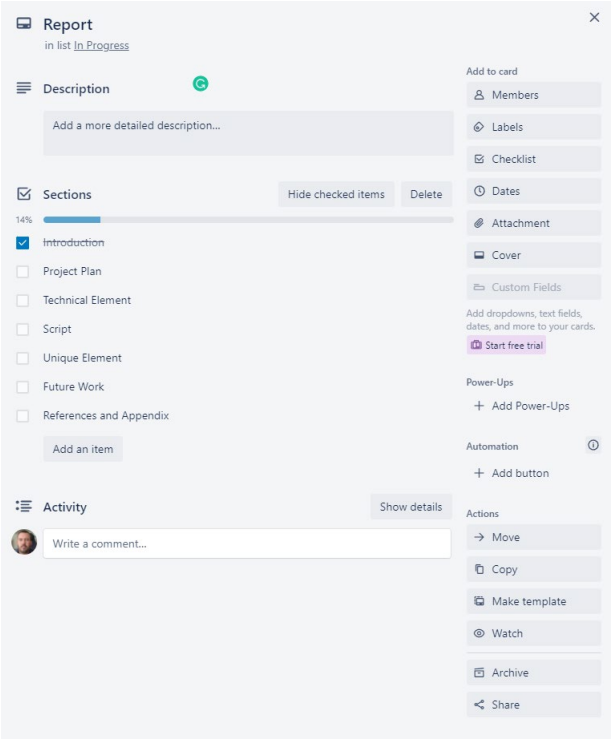

Figure 2.10 – Report card updated

2.3. Week 2

The plan for week 2 is as below

- Finish modelling the house
- UV unwrap the house
- Add textures and materials to the house model
- Model the doors
- Animate the doors
- Add trigger scripts to the doors

The updated Trello (Atlassian, 2021) board at the end of week 2 is shown below (Figure 2.11) as well as the relevant expanded model house card (Figures 2.12)

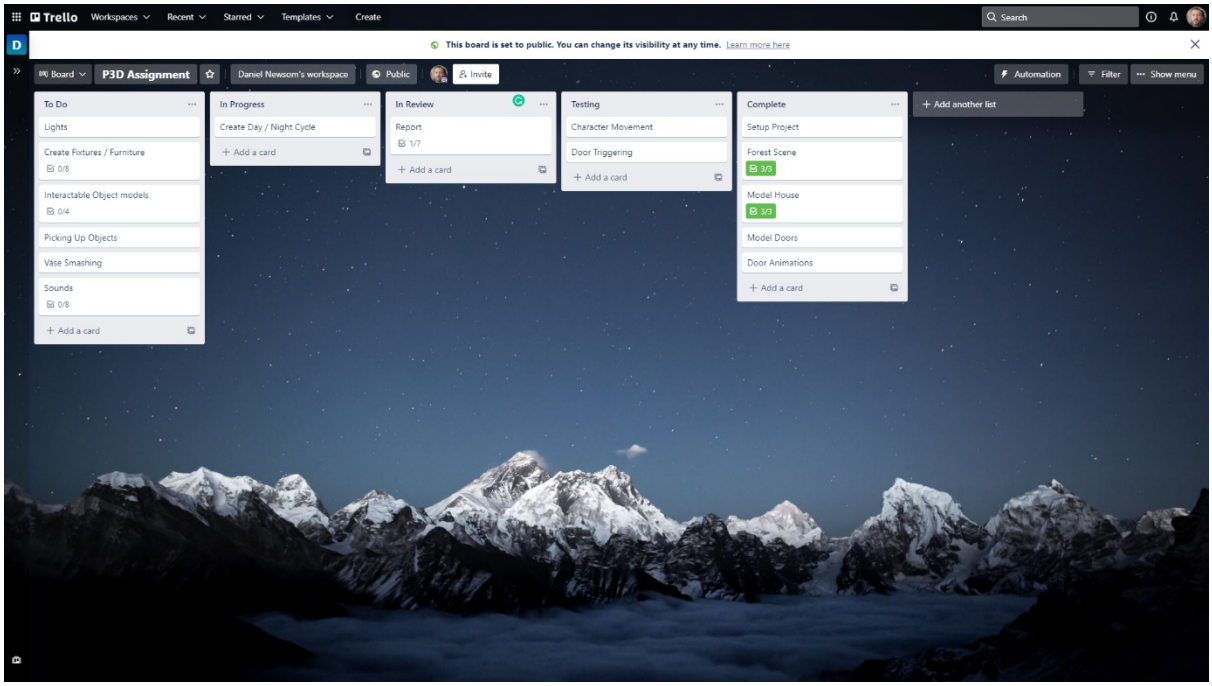

Figure 2.11 - Trello (Atlassian, 2021) board status at the end of week 2

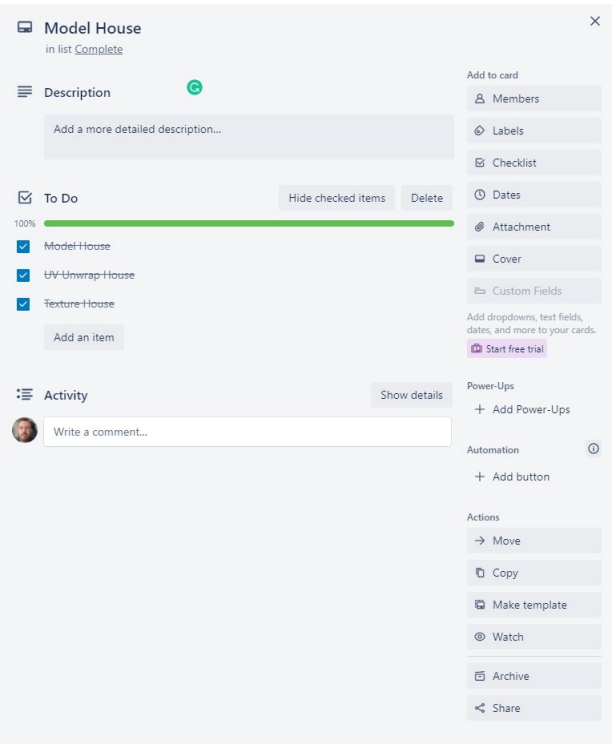

Figure 2.12 – Model House card expanded

2.4. Week 3

The plan for week 3 is as follows

- Re-write the introduction section of the report
- Add chandelier model
- Animate chandelier model to sway
- Add light models to all other rooms
- Add/adjust lights to light models within Unity (Unity, 2021)
- Refactor Character controller script
- Start Day / Night Cycle animation and trigger.

The updated Trello (Atlassian, 2021) board at the end of week 3 is shown below (Figure 2.13) as well as the relevant expanded create fixtures/furniture card (Figures 2.14).

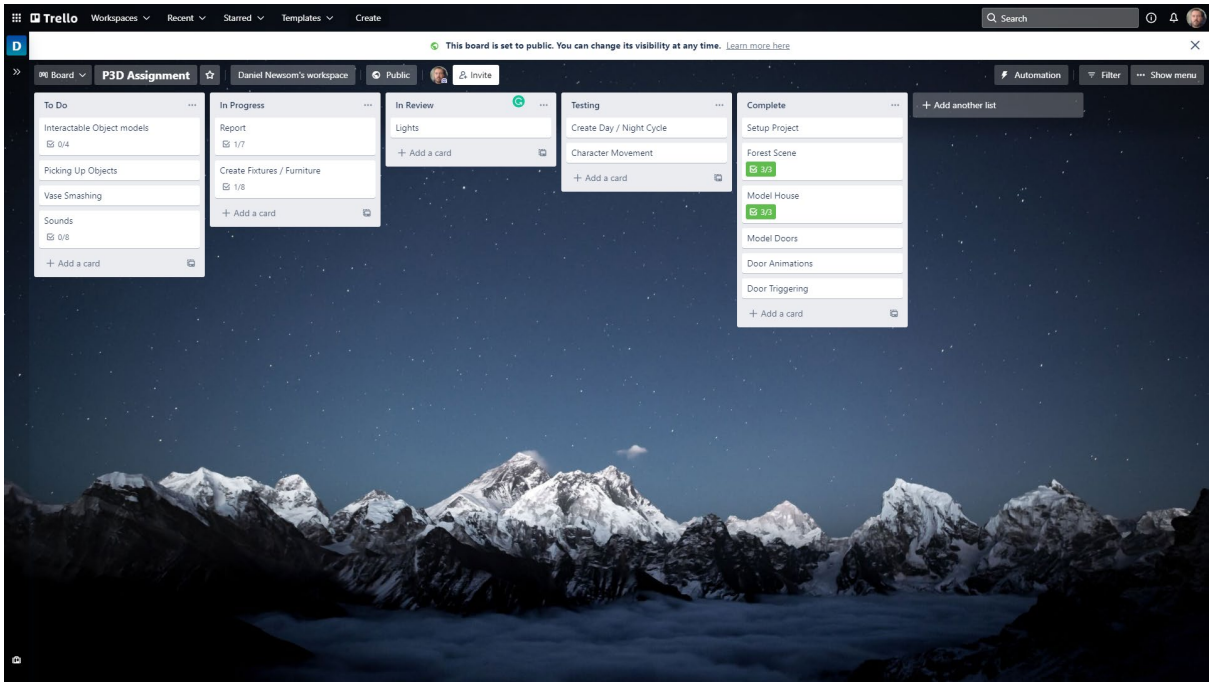

Figure 2.13 - Trello (Atlassian, 2021) board status at the end of week 3

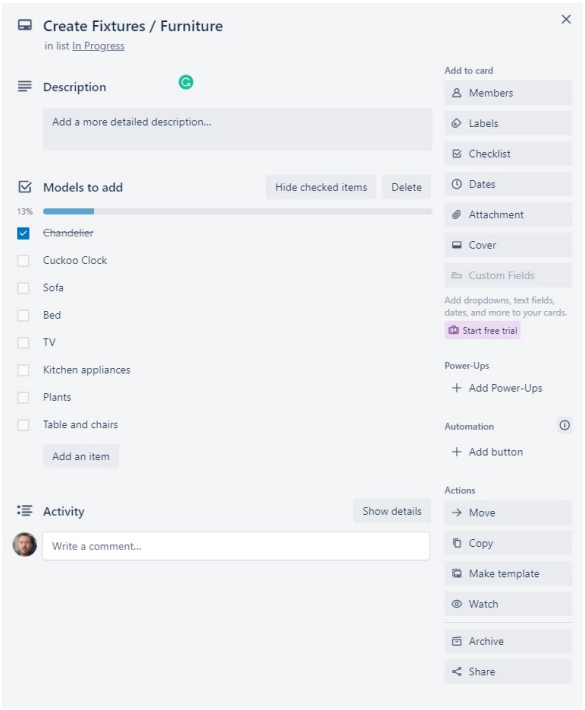

Figure 2.14 – Create Fixtures / Furniture card expanded

2.5. Week 4

The plan for week 4 is as follows

- Model / add Interactable objects
- Start adding furniture (Priority Cuckoo clock)
- Implement picking up and using/throwing objects
- Add VFX fires and triggers
- Start adding sounds.

The updated Trello (Atlassian, 2021) board at the end of week 4 is shown below (Figure 2.15) as well as the relevant expanded Interactable Object models (figure 2.16), create fixtures/furniture card (figures 2.17) and Sounds card (figure 2.18).

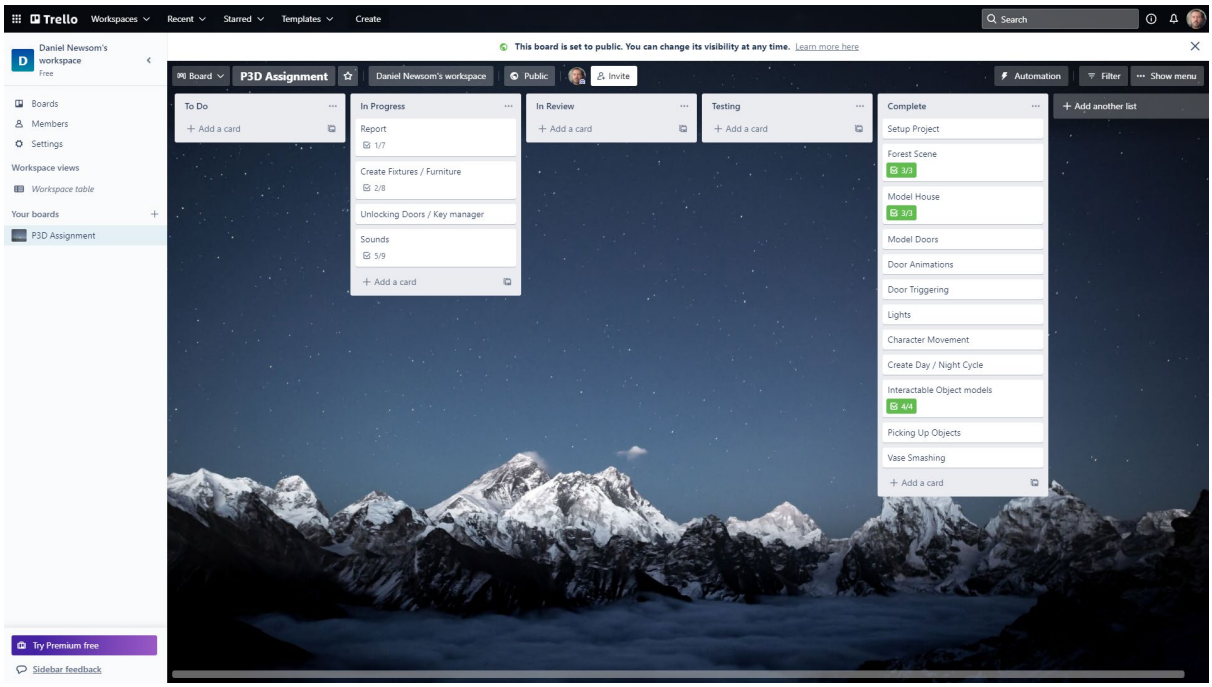

Figure 2.15 - Trello (Atlassian, 2021) board status at the end of week 4

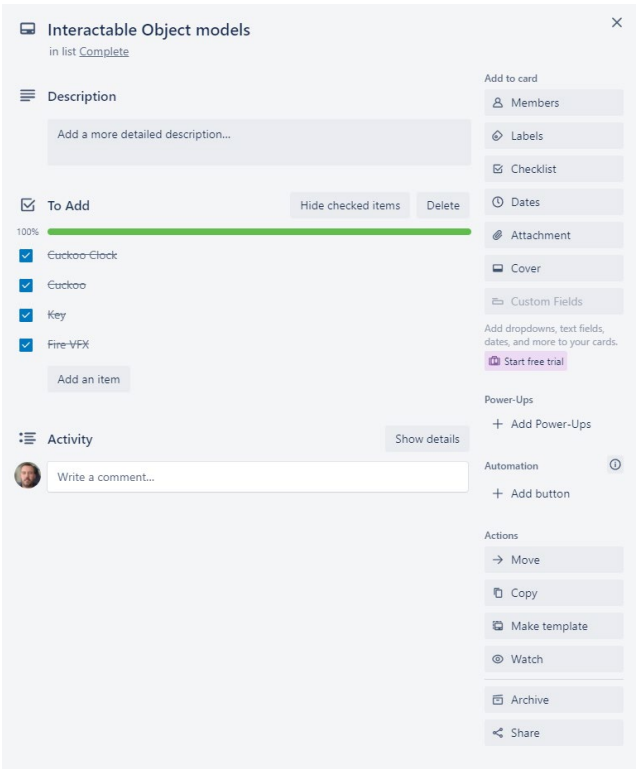

Figure 2.16 – Expanded Interactable object models card.

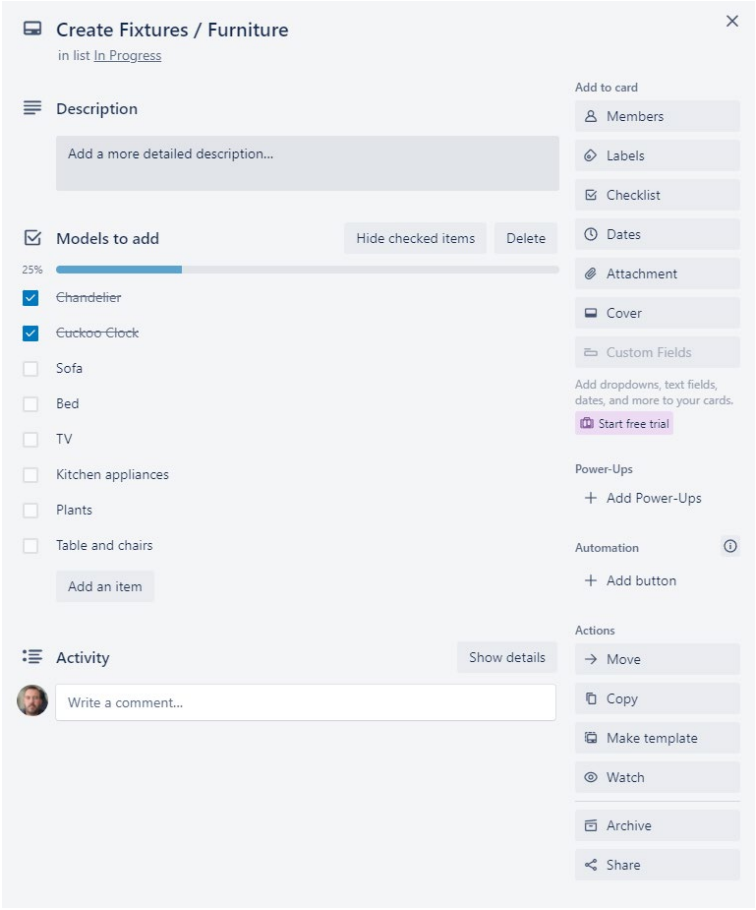

Figure 2.17 – Expanded Create Fixtures / Furniture card.

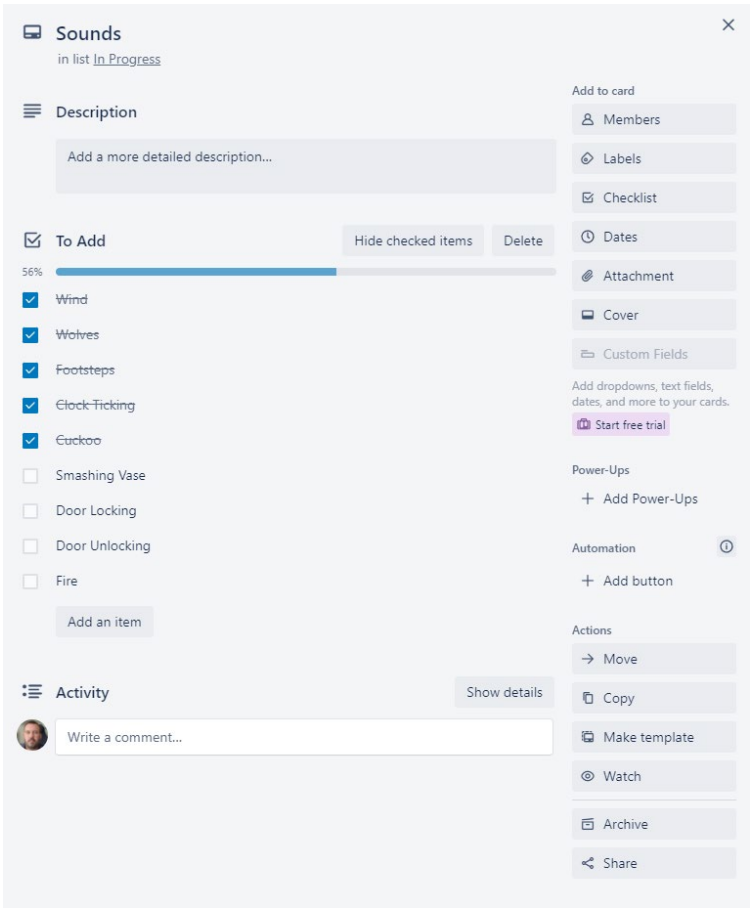

Figure 2.18 – Expanded Sounds card.

2.6. Week 5

The plan for week 5 is as follows

- Implement collecting and using keys
- Add remaining fixtures and fittings
- Add remaining sounds
- Record first demo video for feedback

The updated Trello (Atlassian, 2021) board at the end of week 5 is shown below (Figure 2.19) as well as the relevant expanded Create fixtures/furniture card (figures 2.20) and Sounds card (figure 2.21).

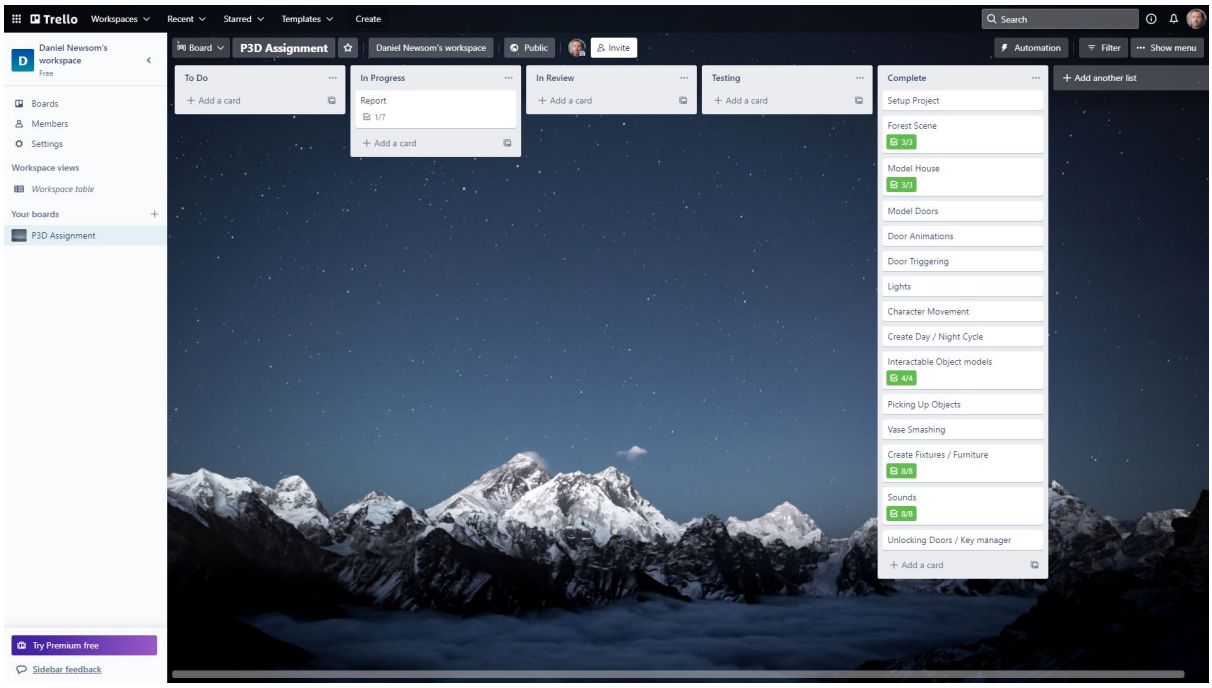

Figure 2.19 - Trello (Atlassian, 2021) board status at the end of week 5

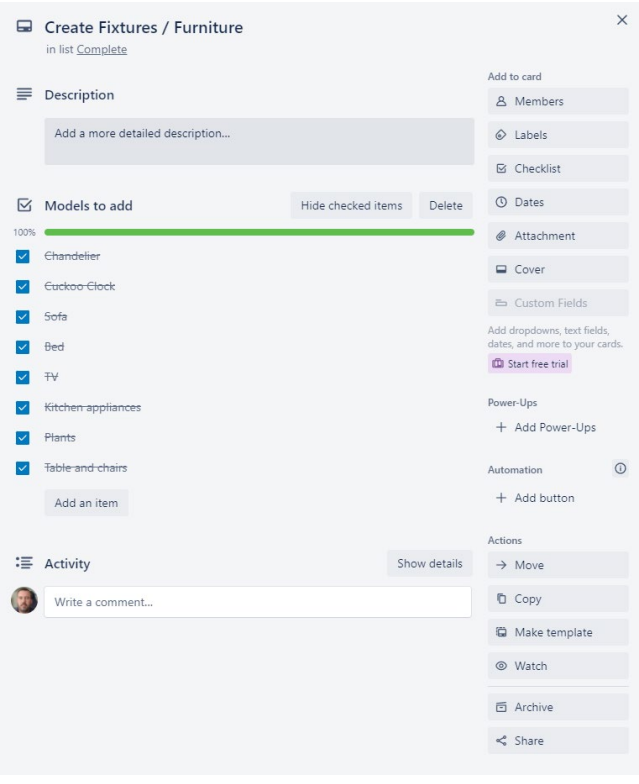

Figure 2.20 – Expanded Create Fixtures / Furniture card.

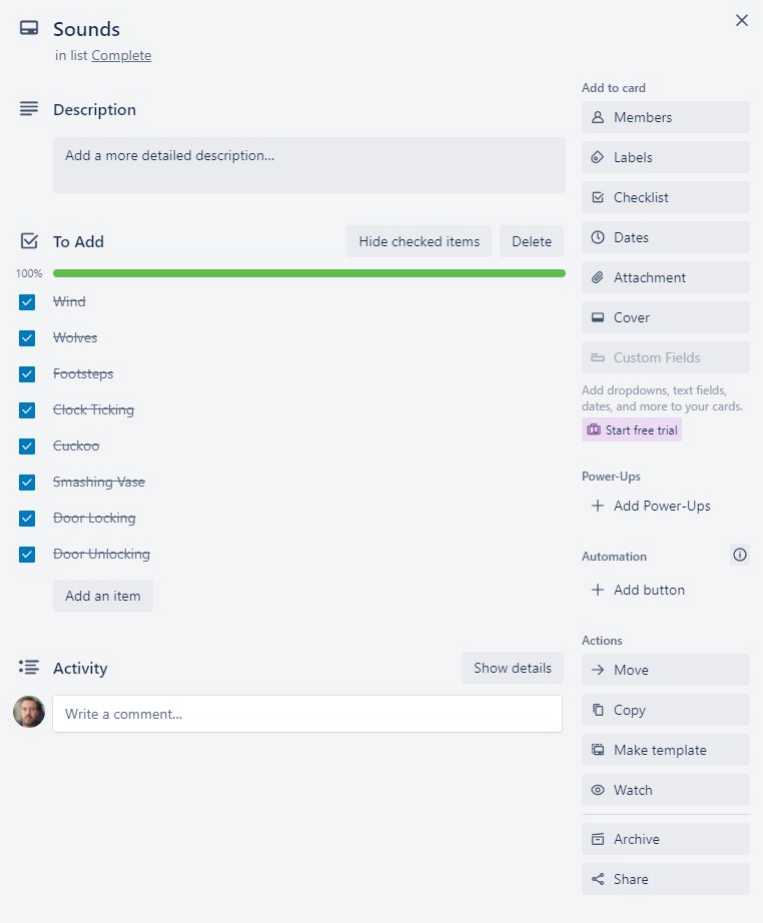

Figure 2.21 – Expanded Sounds card.

2.7. Week 6

The plan for week 6 is as follows

- Add Asset references to the report
- Refactor and tidy up all scripts
- Add all scripts to the appendix
- Complete the remainder of the report

The updated Trello (Atlassian, 2021) board at the end of week 6 is shown below (Figure 2.22) as well as the relevant expanded Report Card (figures 2.23)

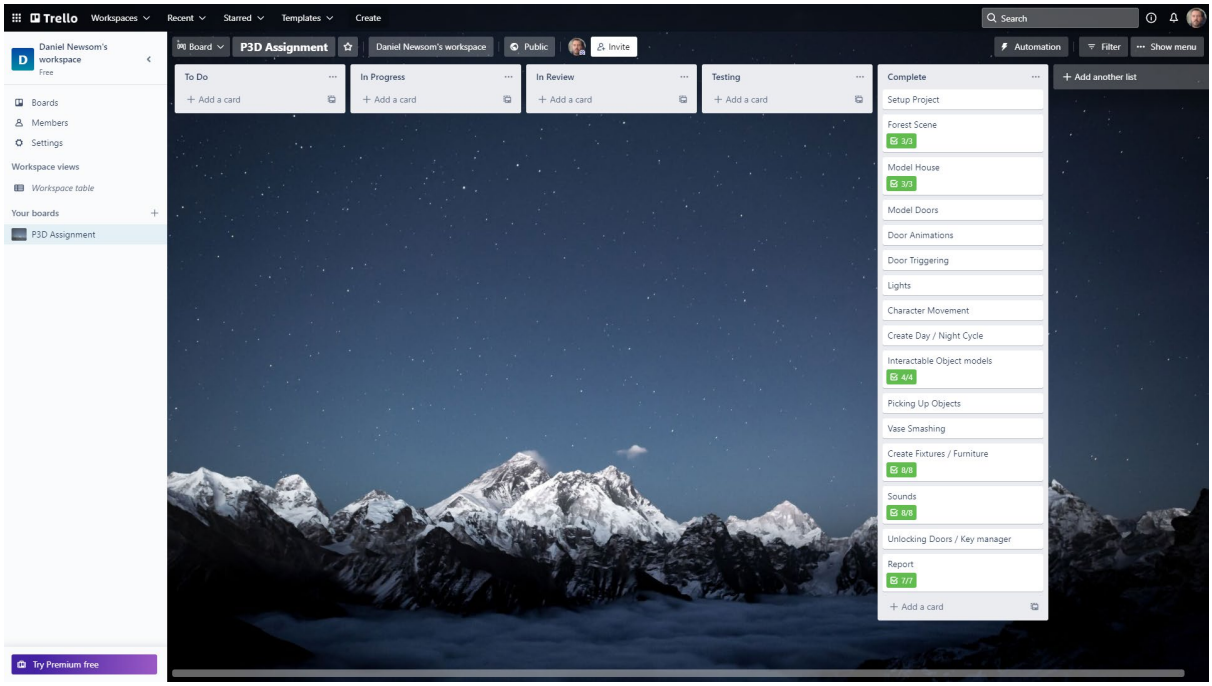

Figure 2.22 - Trello (Atlassian, 2021) board status at the end of week 6

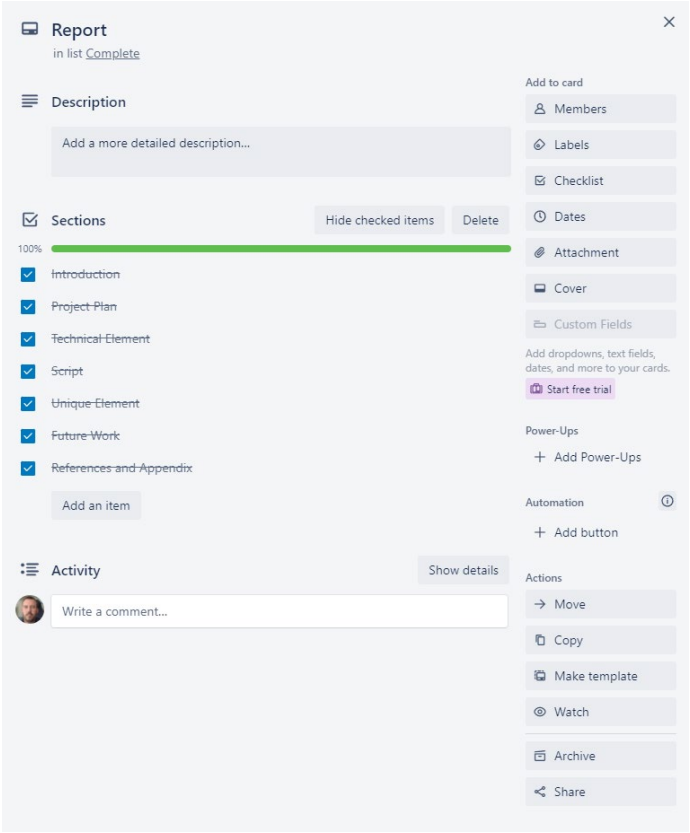

Figure 2.23 – Expanded Report Card

3.0 Technical Element - Vectors

3.1. Introduction

A vector is described by Marc Beaujean (Beaujean, 2020) as “a mathematical unit that can consist of more than one value.” He also states that “It's important to distinguish between vectors in traditional mathematics and in game engines”.

A vector in the context of game development is used to store an object’s various elements current state. They consist mainly of Vector3 with x,y and z values used for objects in 3D space or Vector2 with just the x and y value for objects in 2D space.

3.2. Uses in Video Games

3.2.1. Main uses

The main uses of vectors are as follows.

- Store the position of an object, this is either relative to the scene as a whole (Global) or the object's parent as an offset (local).
- Store the rotation of an object around each of the vector’s axes. Again this can be Local or Global
- Store an object’s current velocity and direction.
  I used this approach in my project when detecting the vases collision with another object in the Vase.cs script (Appendix 9.1.26). If the objects current velocity in either the x,y or z axis was above the given threshold on collision, then the `SmashVase()` method was called to destroy the object.

```
private void OnCollisionEnter(Collision other)
{
    if (other.gameObject.CompareTag("Player")) return;
    if (other.relativeVelocity.x > smashForce
        || other.relativeVelocity.y > smashForce
        || other.relativeVelocity.z > smashForce)
    {
        SmashVase();
    }
}

private void SmashVase()
{
    GetComponent<AudioSource>().Play();
    vaseModel.SetActive(false);
    GetComponent<Collider>().enabled = false;
    brokenVaseModel.SetActive(true);
    keyCollectible.SetActive(true);
    Invoke(nameof(RemoveVelocity),0.2f);
}
```

3.2.2. Movement

Using vectors we can move an object by adding a movement vector to the objects current position to calculate the objects new position, as shown below (figure 3.1) in the diagram from R Nave (Nave, 2021)

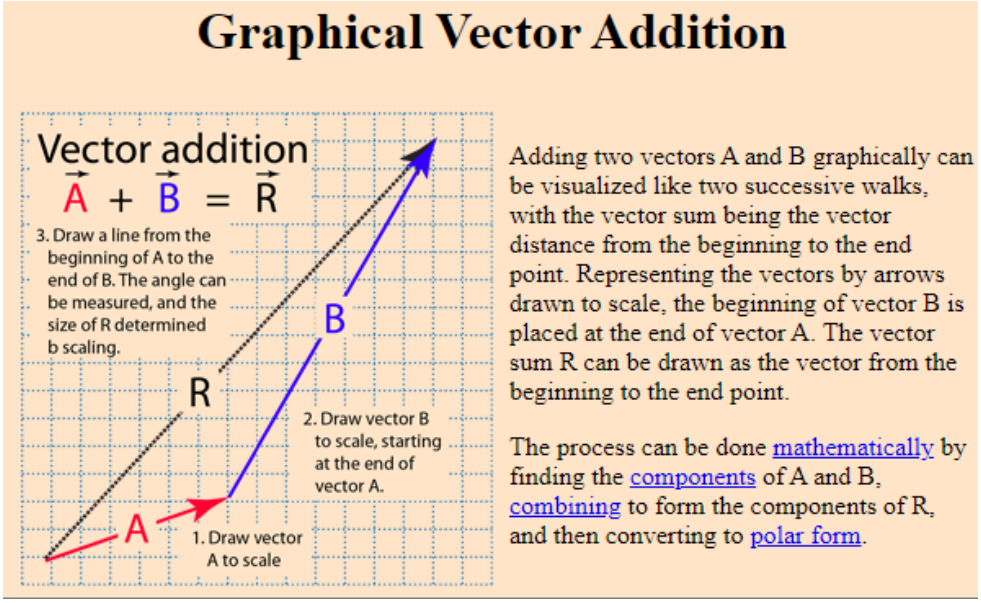

Figure 3.1 – Vector addition diagram (Nave, 2021)

We can move towards another object by getting the direction between them, by subtracting the target objects position from the current object's position and normalizing this value to remove the distance. We can then add this value, multiplied by a distance value to the current object's position vector to move towards that object by the given distance value.

This is demonstrated in the code snippet below from Saad Khan (Khan, 2017)

```
// Calculate direction vector
Vector3 dir = obj1.transform.position - ob2.transform.position;
// Normalize resultant vector to unit Vector
dir = dir.normalized;
// Move in the direction of the direction vector every frame
obj1.transform.position += dir * Time.deltaTime * speed;
```

This process can be simplified by using the built-in `MoveToward()` function from the Vector classes.

3.2.3. Distance

Vectors can be used to calculate distances between two objects using the `Distance(Vector3, Vector3)` method from the Vector Classes which returns the Euclidean distance between the two vectors.

The formula used for this calculation is as described by Rayman May (May, 2014)

Given two points in space $point\ a = (x_0, y_0, z_0),\ point\ b(x_1, y_1, z_1)$

$$distance = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

3.2.4. Other Uses

Vectors within Unity can also be used to store various other variables requiring multiple values for example a range between two floats by storing the min and max values in the x and y values respectively.

I used a Vector2 in my LightBulb.cs script (Appendix 9.1.17) to set a min and max wait time for the bulb to wait before flickering off and on again and the time for the bulb to turn off for.

```
[SerializeField] private Vector2 flickerDelay = new Vector2(0.1f, 5f);
[SerializeField] private Vector2 flickerOffTime = new Vector2(0.01f, 0.1f);
```

I then selected a random value between these to use as the countdown until the next flicker and the delay to turn the light back on.

```
private void FixedUpdate()
{
    if (flickering)
    {
        _flickerDelayTimer -= Time.deltaTime;
        if (_flickerDelayTimer <= 0)
        {
            SwitchLightsOff();
            _flickerDelayTimer = Random.Range(flickerDelay.x, flickerDelay.y);
            Invoke(nameof(SwitchLightsOn),Random.Range(flickerOffTime.x,flickerOffTime.y));
        }
    }
}
```

4.0 Script – ItemGrabber.cs

The script I have chosen to highlight is the ItemGrabber.cs (Appendix 9.1.13.) which is used to pick up objects and keys, throw held objects as well as update the UI when the player is looking at an interactable item.

The beginning of the script is used in the usual way by declaring system packages, variables, and a start method to set any required private member variables.

```
using System.Collections.Generic;
using UnityEngine;

public class ItemGrabber : MonoBehaviour
{
    [SerializeField] private LayerMask pickupLayerMask;
    [SerializeField] private float raycastDistance = 5f;
    [SerializeField] private Transform cameraTransform;
    [SerializeField] [Range(0f,20f)] private float throwForce = 10f;

    private readonly List<GameObject> _heldItems = new List<GameObject>();
    private KeyManager _keyManager;
    private PlayerInputHandler _playerInputHandler;
    private UIController _uiController;
    private GameObject _currentItem;
    private int _currentItemIndex;
    private bool _inCooldown;

    private void Start()
    {
        _playerInputHandler = FindObjectOfType<PlayerInputHandler>();
        _keyManager = FindObjectOfType<KeyManager>();
        _uiController = FindObjectOfType<UIController>();
    }
```

This is followed by the Update method which first performs a Raycast forward from the centre of the camera using a layer mask to detect only key and pickup objects. This contains an out Raycast hit variable that is used in the remainder of the method. If this Raycast is successful, then the player is looking at an interactable object and the UI Controller is called to update the display to which interactable item is being looked at by the player. If it is not then the UI display is cleared.

```
private void Update()
{
    if (Physics.Raycast(cameraTransform.position, cameraTransform.forward, out RaycastHit hitInfo,
        raycastDistance,
        pickupLayerMask, QueryTriggerInteraction.Collide))
    {
        if (hitInfo.transform.TryGetComponent<Key>(out Key key))
        {
            _uiController.SetInfoDisplay($"Pick up {KeyManager.GetKeyName(key.DoorToOpen)} key");
        }
        else if (hitInfo.transform.TryGetComponent<Pickup>(out Pickup pickup))
        {
            _uiController.SetInfoDisplay($"Pick up {pickup.Type}");
        }
    }
    else
    {
        _uiController.SetInfoDisplay("");
    }
```

The script then returns from the update method if the in cooldown boolean variable is true. This stops multiple keypresses from being detected in each update cycle.

```
if(_inCooldown){return;}
```

If the script is not in a cooldown, it then checks if the pickup/throw button has been pressed on the Player Input Handler this cycle and then checks if the hit info variable is valid (i.e not null).

```
if (_playerInputHandler.Throw)
{
    if (hitInfo.transform)
    {
```

If the hit info is valid, then the player is either looking at a key or a pickup item and so the script first checks if the hit info object has a key script and if it does, collects the key and adds it to the Key Manager. The script then sets the cooldown boolean to true and invokes a method with a delay to reset the cooldown boolean and returns from the update method.

```
if (hitInfo.transform.CompareTag("Key"))
{
    if (hitInfo.transform.TryGetComponent<Key>(out Key key))
    {
        _keyManager.AddKey(key.DoorToOpen);
        Destroy(hitInfo.transform.gameObject);
    }
    _inCooldown = true;
    Invoke(nameof(ResetCooldown),0.2f);
    return;
}
```

If the hit info object is not a pickup or the player already has that object, then the method returns from the update method.

```
if (!hitInfo.transform.CompareTag("Pickup") ||
    _heldItems.Contains(hitInfo.transform.gameObject)) return;
```

If the hit info object is a pickup, then the script adds it to the held items list, sets it as the current item, sets its index in the held Items List as the current item index and sets the object to active using the `SetActiveObject()` method (see below). This enables cycling through multiple held objects.

```
_currentItem = hitInfo.transform.gameObject;
_heldItems.Add(_currentItem);
_currentItemIndex = _heldItems.IndexOf(_currentItem);
SetActiveObject();
```

The script then disables this new current item object RigidBody and Collider so that it does not interfere with the player's movement and then sets its parent, position, and rotation to the item grabbers transform. Finally, the scale of the object is set to 0.25 on all axes and as before the cooldown is initiated.

```
_currentItem.GetComponent<Rigidbody>().isKinematic = true;
_currentItem.GetComponent<Collider>().enabled = false;
Transform grabberTransform = transform;
_currentItem.transform.parent = grabberTransform;
_currentItem.transform.position = grabberTransform.position;
_currentItem.transform.rotation = grabberTransform.rotation;
_currentItem.transform.localScale = new Vector3(0.25f,0.25f,0.25f);
_inCooldown = true;
Invoke(nameof(ResetCooldown),0.2f);
```

If the player is not looking at an interactable object (but has pressed the pickup/throw key), then it is assumed that the player wants to throw the current object.

The script first checks that there is a valid current item and returns from the update method if not.

```
if (!_currentItem) return;
```

if the current item is valid then it is removed from the held items list and stored in a temporary variable.

```
GameObject itemToThrow = _currentItem;
_heldItems.Remove(itemToThrow);
```

If the player has other items in the held items list, then the item at the end of the list is set as the current item as previously. The current item is set to null if the list length is 0.

```
if (_heldItems.Count > 0)
{
    _currentItem = _heldItems[_heldItems.Count - 1];
    _currentItemIndex = _heldItems.IndexOf(_currentItem);
    SetActiveObject();
}
else
{
    _currentItem = null;
}
```

The temporary item to throw variable then has its parent removed, it Rigidbody and Collider re-enabled and scale reset to 1. It also has a force applied to its Rigidbody to propel it forward using the throw force variable. Again, the cooldown process is then triggered.

```
itemToThrow.transform.parent = null;
        itemToThrow.GetComponent<Collider>().enabled = true;
        itemToThrow.TryGetComponent<Rigidbody>(out Rigidbody itemRb);
        itemRb.isKinematic = false;
        itemRb.AddForce(cameraTransform.forward * (throwForce * 100f), ForceMode.Acceleration);
        itemToThrow.transform.localScale = new Vector3(1, 1, 1);
        _inCooldown = true;
        Invoke(nameof(ResetCooldown),0.2f);
    }
}
```

If the Pickup/throw button was not pressed this cycle, then finally the script checks both the next and previous item buttons on the Player input and calls each one's corresponding method if it has been pressed this cycle.

```
else
{
    if (_playerInputHandler.NextItem)
    {
        NextItem();
    }
    else if (_playerInputHandler.PreviousItem)
    {
        PreviousItem();
    }
}
```

The `NextItem()` method is used to cycle to the next item held by the player.

It first gets the number of items held and returns from the `NextItem()` method if it is 1 or less.

```
private void NextItem()
{
    int numberOfItems = _heldItems.Count;
    if(numberOfItems <= 1){return;}
```

If not then if the current item index is the last item in the list, it is set to the first. If it is not the last item then it is simply incremented.

```
if (_currentItemIndex == numberOfItems-1)
{
    _currentItemIndex = 0;
}
else
{
    _currentItemIndex++;
}
```

The current item is then set to the item in the held items list that corresponds to this new index and is set as the active object using the `SetActiveObject()`method (see below).

```
_currentItem = _heldItems[_currentItemIndex];
SetActiveObject();
```

The cooldown process is then called as before.

```
_inCooldown = true;
    Invoke(nameof(ResetCooldown),0.2f);
}
```

The `PreviousItem()` method is identical to the `NextItem()` method except that if the item is the first item in the held items list it is set to the last, and if not it is simply decremented.

```
private void PreviousItem()
{
    int numberOfItems = _heldItems.Count;
    if(numberOfItems <= 1){return;}

    if (_currentItemIndex == 0)
    {
        _currentItemIndex = numberOfItems - 1;
    }
    else
    {
        _currentItemIndex--;
    }
    _currentItem = _heldItems[_currentItemIndex];
    SetActiveObject();
    _inCooldown = true;
    Invoke(nameof(ResetCooldown),0.2f);
}
```

The `SetActiveObject()` method returns if there are no items in the held items list.

If there are items held then it will cycle through each, in turn, using a for loop and set the object to active if the current item index equals the current for loop index or set inactive if it does not.

```
private void SetActiveObject()
{
    if (_heldItems.Count <= 0) return;

    for (int i = 0; i < _heldItems.Count; i++)
    {
        if (i == _currentItemIndex)
        {
            _heldItems[i].SetActive(true);
        }
        else
        {
            _heldItems[i].SetActive(false);
        }
    }
}
```

The ResetCooldown method simply sets the in cooldown boolean to false and is called using the `Invoke(string methodName, float time)` method to add a delay to the call.

```
private void ResetCooldown()
{
    _inCooldown = false;
}
```

5.0 Unique Element – Day / Night Cycle

I used the Unity (Unity, 2021) HDRP for my project and as such created a Physically based sky for the scene. I added two versions of this for both Daytime and Night-time each with different settings for the colour, fog and post-processing, and then animated the sun and moon objects to rotate across the sky.

This animation approach also allowed me to add animation events to switch out the volume profiles to adjust each one's fog and post-processing separately as well as change the ambient sounds for day or night and switch on and off the house lights at dusk or dawn during the daytime cycle.

I also used this animation to animate the hands of the cuckoo clock to match the time of day or night within the scene.

The scene starts by looping perpetually through the night-time animation until the player escapes the house, the animation is then fast-forwarded to the daytime animation using triggers on the animator object to speed up the animation until the daytime cycle starts when the initial speed is reset.

Switching to this daytime cylcle with less fog and brighter colour tones completely changes the atmosphere within the scene.

6.0 Summary

The scene that I created consisted of a forest area containing a spooky house, stuck in a perpetual night-time cycle. The player must then enter the house, complete the tasks highlighted within the introduction section picking up and using objects and keys to unlock the conservatory door and escape. This then triggers the daytime cycle and displays a "You escaped message to the player.

The player controller used was an edited version of the Starter Assets - First Person Character Controller (Unity_Starter_Assets, 2021)

The exterior of the scene was put together using terrain textures from Forest Environment - Dynamic Nature (NatureManufacture_Forest_Environment, 2021) and trees from Mountain Trees - Dynamic Nature (NatureManufacture_Mountain_Trees, 2021)

The House and many of the fixtures and fittings were modelled, UV unwrapped and textured by me. Some of the furniture was re-used from a previous Clue (Newsom (Clue), 2021) coursework project that were initially all modelled by me. Others were used from various other sources including my portfolio website (Newsom, 2021) as detailed in section 8.2 Assets.

7.0 Future Work

I believe that there is scope for various other puzzles and objects within the house to prolong the playtime and make the whole process trickier. These could include

- Playing a set of notes on the piano.
- Switching the channel on the TV to show some hidden item via a security camera.
- Get something from the fridge and put it in the oven.
- Adding a secret room behind one of the bookcases.
- Potting a ball on the pool table to eject a key or item.
- Watering a plant in the house to produce an item.
- I could also add a timer and high scores to try and beat the time to escape the house.

8.0 References

8.1. Github

Main Github page - https://github.com/dgnewsom

Project Repository - https://github.com/dgnewsom/215851_P3D

8.2. Assets

8.2.1. Models

- **Apple Model** - (Baria3DAsset(Apple), 2020)
- **Banana Model** - (Baria3DAsset(Banana), 2020)
- **Bath** - (diger-mpt, 2019)
- **Bath Tap** - (Renderscope, 2015)
- **Bookcases** - (Newsom (Clue), 2021)
- **Desk** - (Newsom (Clue), 2021)
- **Dining Set**
  - **Table** - (Newsom (Clue), 2021)
  - **Chair** - (Newsom (Chair), 2021)
- **Fridge** - (Newsom (Clue), 2021)
- **Hat Stand**
  - **Gangster Hat** - (MichalCavrnoch, 2020)
  - **Cowboy Hat** - (vanmourik-elise, 2016)
- **Kitchen Cupboards** - (Newsom (Clue), 2021)
- **Orange Model** - (mikailtasim, 2019)
- **Oven** - (Newsom (Clue), 2021)
- **Piano** - (choughry, 2021)
- **Plant** - (dominiklesniak, 2018)
- **Pool Table** - (Newsom (Pool Table), 2021)
- **Shower** - (claudiodubas, 2015)
- **Shower Cubicle** - (sweethome3d, 2013)
- **Sink** - (BSW2142, 2018)
- **Sofa** - (Newsom (Sofa), 2021)
- **Telephone Table**
  - **Table** - (Newsom (Clue), 2021)
  - **Old Phone** - (Polymake, 2020)
  - **NoteBook** - (Hard, 2021)
- **Television and Stand** - (Newsom (Clue), 2021)
- **Toilet Model** - (MarkStead, 2021)
- **Toilet Roll Models** - (dandcowan, 2021)

8.2.2. Audio

- **Bird Song** - (Imjeax, 2018)
- **Clock Tick** - (straget, 2019)
- **Cuckoo** - (InspectorJ, 2017)
- **Door Close** - (rivernile7, 2014)
- **Door Lock / Unlock** - (Fabrizio84, 2019)
- **Door Open** - (pagancow, 2006)
- **Footstep Sounds** - (Glowkeeper_(Footsteps_Audio), 2021)
- **Piano** - (deadrobotmusic, 2021)
- **Serving Hatch** - (craigsmith, 2018)

- **Vase Smash** - (kingsrow, 2013)
- **Wind** - (nsstudios, 2019)
- **Wolf Sounds**
    1. (y89312, 2011)
    2. (killyourpepe, 2017)
    3. (cazadordoblekatana, 2018)
    4. (newagesoup, 2016)
    5. (NaturesTemper, 2017)

8.2.3. Fonts

- **Title Font** - (PutraCetol, 2021)
- **UI Font** - (Art_Power, 2017)

8.2.4. Sprites

- **Smoke Sprite Sheet** - (Beast, 2013)

8.2.5. Scripts

- **Character Controller** - (Unity_Starter_Assets, 2021)
- **Footsteps code** - (Glowkeeper_(Footsteps_Code), 2021)

8.2.6. Full References

Art_Power, 2017. *DaFont - Stranger back in the Night.* [Online]
Available at: https://www.dafont.com/stranger-back-in-the-night.font
[Accessed 9 December 2021].

Atlassian, 2021. *Trello.* [Online]
Available at: https://trello.com/
[Accessed 5 November 2021].

Baria3DAsset(Apple), 2020. *CGTrader - Apple Fruit Free low-poly 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/food/fruit/apple-fruit-afa5e664-8ff8-4598-9568-a21c35520ca3
[Accessed 7 December 2021].

Baria3DAsset(Banana), 2020. *CGTrader - Banana Free low-poly 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/food/fruit/banana-a86fda85-aa6c-4261-8d87-da597aa55f28
[Accessed 7 December 2021].

Beast, 2013. *OpenGameArt.Org - Swirling Prerendered Smoke Animation with seamless looping..* [Online]
Available at: https://opengameart.org/content/smoke-aura
[Accessed 3 December 2021].

Beaujean, M., 2020. *Vector Maths for Game Dev Beginners.* [Online]
Available at: https://www.gamedeveloper.com/disciplines/vector-maths-for-game-dev-beginners
[Accessed 10 December 2021].

Blender, 2021. *Blender.* [Online]
Available at: https://www.blender.org/
[Accessed 5 November 2021].

BSW2142, 2018. *CGTrader - Marble Bathroom Sink Free low-poly 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/furniture/other/bathroom-sink-07b46f36-c6ba-41d5-ad60-9475c96988fc
[Accessed 8 December 2021].

cazadordoblekatana, 2018. *Freesound - 15-WolfCrying.wav.* [Online]
Available at: https://freesound.org/people/cazadordoblekatana/sounds/429109/
[Accessed 29 November 2021].

choughry, 2021. *CGTrader - Piano Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/interior/house/piano-7956bf16-98c3-4bd3-8171-8df382638c08
[Accessed March 2021].

claudiodubas, 2015. *CGTrader - Shower 4 Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/interior/bathroom/shower-4--2
[Accessed 8 December 2021].

craigsmith, 2018. *Freesound - G31-04-Small Wooden Door.wav.* [Online]
Available at: https://freesound.org/people/craigsmith/sounds/438442/
[Accessed 9 December 2021].

dandcowan, 2021. *CGTrader - Toilet Rolls Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/household/other/toilet-rolls
[Accessed 8 December 2021].

deadrobotmusic, 2021. *Freesound - SOS Dark Piano Loop [148bpm] [C Minor].* [Online]
Available at: https://freesound.org/people/deadrobotmusic/sounds/574346/
[Accessed 8 December 2021].

diger-mpt, 2019. *CGTrader - Asymmetric bathtub size 1600x1100 Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/interior/bathroom/asymmetric-bathtub-size-1600x1100
[Accessed 8 December 2021].

dominiklesniak, 2018. *CGTrader - monstera-deliciosa.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/plant/pot-plant/monstera-deliciosa
[Accessed March 2021].

Fabrizio84, 2019. *Freesound - Locking Door.* [Online]
Available at: https://freesound.org/people/Fabrizio84/sounds/458013/
[Accessed 6 December 2021].

Glowkeeper_(Footsteps_Audio), 2021. *Github - P3D/assets/audio/.* [Online]
Available at: https://github.com/glowkeeper/P3D/tree/master/assets/audio
[Accessed 29 November 2021].

Glowkeeper_(Footsteps_Code), 2021. *Github - P3D Lab for Week 4, Session 2 - Audio in Unity.* [Online]
Available at: https://github.com/glowkeeper/P3D/blob/master/docs/labs/week4Session2.md
[Accessed 29 November 2021].

Hard, A., 2021. *CGTrader - A4 Notebook 3d model nots student mockup school Free low-poly 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/household/other/a4-notebook-3d-model-nots-student-mockup-school
[Accessed March 2021].

Imjeax, 2018. *Freesound - Forest Ambient LOOP.* [Online]
Available at: https://freesound.org/people/Imjeax/sounds/427400/
[Accessed 29 November 2021].

InspectorJ, 2017. *Freesound - Cuckoo Clock, Single, A.wav.* [Online]
Available at: https://freesound.org/people/InspectorJ/sounds/398194/
[Accessed 29 November 2021].

Khan, S., 2017. *Unity - moving one gameobject towards another.* [Online]
Available at: https://answers.unity.com/questions/1303472/moving-one-gameobject-towards-another.html
[Accessed 10 December 2021].

killyourpepe, 2017. *Freesound - DuskWolf.wav.* [Online]
Available at: 1. https://freesound.org/people/killyourpepe/sounds/395192/
[Accessed 29 November 2021].

kingsrow, 2013. *Freesound - BreakingVase02.wav.* [Online]
Available at: https://freesound.org/people/kingsrow/sounds/194684/
[Accessed 9 December 2021].

MarkStead, 2021. *CGTrader - Toilet Suite Kohler Reach 5233A-0 Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/interior/bathroom/toilet-suite-kohler-reach-5233a-0
[Accessed 8 December 2021].

May, R., 2014. *Calculate distance in 3D space.* [Online]
Available at: https://math.stackexchange.com/q/1069627
[Accessed 10 December 10].

MichalCavrnoch, 2020. *CGTrader - Gangster hats Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/character/clothing/gangster-hats
[Accessed March 2021].

mikailtasim, 2019. *CGTrader - Orange Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/food/fruit/orange-f548261d-6a73-4f1f-a318-0236d1c2b22b
[Accessed 7 December 2021].

NatureManufacture_Forest_Environment, 2021. *Forest Environment - Dynamic Nature.* [Online]
Available at: https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668
[Accessed 5 November 2021].

NatureManufacture_Mountain_Trees, 2021. *Mountain Trees - Dynamic Nature.* [Online]
Available at: https://assetstore.unity.com/packages/3d/vegetation/trees/mountain-trees-dynamic-nature-107004
[Accessed 5 November 2021].

NaturesTemper, 2017. *Freesound - Wolf howl.* [Online]
Available at: https://freesound.org/people/NaturesTemper/sounds/398430/
[Accessed 29 November 2021].

Nave, R., 2021. *Basic Vector Operations.* [Online]
Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/vect.html
[Accessed 10 December 2021].

newagesoup, 2016. *Freesound - wolf-growl.wav.* [Online]
Available at: https://freesound.org/people/newagesoup/sounds/338674/
[Accessed 29 November 2021].

Newsom (Chair), D., 2021. *Daniel Newsom - Photorealistic Chair.* [Online]
Available at: https://danielnewsom.co.uk/modelling/chair.php
[Accessed 7 December 2021].

Newsom (Clue), D., 2021. *Daniel Newsom - Clue.* [Online]
Available at: https://danielnewsom.co.uk/gamepages/clue.php
[Accessed 7 December 2021].

Newsom (Pool Table), D., 2021. *Daniel Newsom - Pool Table.* [Online]
[Accessed 7 December 2021].

Newsom (Sofa), D., 2021. *Daniel Newsom - Sofa.* [Online]
[Accessed 7 December 2021].

Newsom, 2021. *Daniel Newsom - Portfolio website.* [Online]
Available at: https://danielnewsom.co.uk
[Accessed 8 December 2021].

nsstudios, 2019. *FreeSound - wind blowing loop 1.* [Online]
Available at: https://freesound.org/people/nsstudios/sounds/479192/
[Accessed 29 November 2021].

pagancow, 2006. *Freesound - dorm door opening.wav.* [Online]
Available at: https://freesound.org/people/pagancow/sounds/15419/
[Accessed 6 December 2021].

Polymake, 2020. *CGTrader - Old Phone Low-poly 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/electronics/phone/old-phone-7bab903e-bdab-45b3-9a29-7077ea99f37c
[Accessed March 2021].

PutraCetol, S., 2021. *DaFont - The Night Lamp.* [Online]
Available at: https://www.dafont.com/the-night-lamp.font
[Accessed 9 December 2021].

Renderscope, 2015. *CGTrader - Bathroom faucet Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/interior/bathroom/bathroom-faucet-2--2
[Accessed 8 December 2021].

rivernile7, 2014. *Freesound - Door Open And Close.* [Online]
Available at: https://freesound.org/people/rivernile7/sounds/234244/
[Accessed 6 December 2021].

straget, 2019. *Freesound - Antique wall clock.wav.* [Online]
Available at: https://freesound.org/people/straget/sounds/456236/
[Accessed 29 November 2021].

sweethome3d, 2013. *CGtrader - Shower cabin Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/household/other/shower-cabin
[Accessed 8 December 2021].

Unity_Starter_Assets, 2021. *Starter Assets - First Person Character Controller.* [Online]
Available at: https://assetstore.unity.com/packages/essentials/starter-assets-first-person-character-controller-196525
[Accessed 5 November 2021].

Unity, 2021. *Unity.* [Online]
Available at: https://unity.com/
[Accessed 25 November 2021].

vanmourik-elise, 2016. *CGTrader - Cowboy Hat Free 3D model.* [Online]
Available at: https://www.cgtrader.com/free-3d-models/character/clothing/cowboy-hat-d21039a1-a146-4e56-8d8f-060629853f81
[Accessed March 2021].

y89312, 2011. *Freesound - 41.wav.* [Online]
Available at: https://freesound.org/people/y89312/sounds/139924/
[Accessed 29 November 2021].

9.0 Appendix

9.1. Scripts

9.1.1. Candle.cs

```
using UnityEngine;

public class Candle : MonoBehaviour
{
    [SerializeField] private GameObject keyObjectToAppear;

    private readonly Vector3 _targetScale = new Vector3(0.75f,0.75f,0.1f);
    private bool _burningDown;

    private void Update()
    {
        if (_burningDown)
        {
            transform.localScale = Vector3.Lerp(transform.localScale, _targetScale, Time.deltaTime);
            if (transform.localScale.z <= 0.15f)
            {
                keyObjectToAppear.SetActive(true);
                _burningDown = false;
            }
        }
    }

    public void StartBurning()
    {
        _burningDown = true;
    }
}
```

9.1.2. CuckooClock.cs

```
using UnityEngine;

public class CuckooClock : MonoBehaviour
{
    [SerializeField] private AudioClip cuckooSound;
    [SerializeField] private GameObject cuckooToDrop;

    private AudioSource _cuckooClockAudioSource;
    private bool _hasPlayed;
    private static readonly int Cuckoo = Animator.StringToHash("Cuckoo");

    private void Start()
    {
        _cuckooClockAudioSource = transform.parent.GetComponent<AudioSource>();
        cuckooToDrop.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if(_hasPlayed){return;}

        if (other.CompareTag("Player"))
        {
            GetComponent<Animator>().SetBool(Cuckoo,true);
            _hasPlayed = true;
        }
    }

    public void PlayCuckooSound()
    {
        _cuckooClockAudioSource.PlayOneShot(cuckooSound);
    }

    public void DropCuckoo()
    {
        cuckooToDrop.SetActive(true);
    }
}
```

9.1.3. DayNightController.cs

```
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.Rendering;

public class DayNightController : MonoBehaviour
{
    [Header("Volume Profiles")]
    [SerializeField] private VolumeProfile dayProfile;
    [SerializeField] private VolumeProfile nightProfile;
    [Header("Audio")]
    [SerializeField] private AudioClip nightAmbientSound;
    [SerializeField] private AudioClip dayAmbientSound;
    [Header("Event to trigger on switch to daytime")]
    [SerializeField] private UnityEvent endingEvent;

    private LightBulb[] _lights;
    private bool _isDaytime;
    private AudioSource _outsideAudioSource;
    private Volume _volume;
    private Animator _animator;
    private float _animationSpeed;
    private static readonly int Speed = Animator.StringToHash("Speed");
    private static readonly int Daytime = Animator.StringToHash("IsDaytime");

    public bool IsDaytime => _isDaytime;

    private void Start()
    {
        _animator = GetComponent<Animator>();
        _volume = GetComponentInChildren<Volume>();
        _lights = FindObjectsOfType<LightBulb>();
        _animationSpeed = _animator.GetFloat(Speed);
        _outsideAudioSource = GetComponent<AudioSource>();
        SetNighttime();
    }

    private void UpdateAnimator()
    {
        _animator.SetBool(Daytime,_isDaytime);
    }

    private void SetDaytime()
    {
        _isDaytime = true;
        _outsideAudioSource.clip = dayAmbientSound;
        _outsideAudioSource.Play();
        _volume.profile = dayProfile;
        UpdateAnimator();
        Invoke(nameof(EndingEvent),1f);
    }

    private void SetNighttime()
    {
        _isDaytime = false;
        _outsideAudioSource.clip = nightAmbientSound;
        _outsideAudioSource.Play();
        _volume.profile = nightProfile;
        UpdateAnimator();
    }

    [ContextMenu("Toggle Day/Night")]
    public void ToggleDayNight()
    {
        _isDaytime = !_isDaytime;
        if (_isDaytime)
        {
            SetDaytime();
        }
        else
        {
            SetNighttime();
```

```
        }
    }

    [ContextMenu("Skip To Day")]
    public void SkipToDaytime()
    {
        SetDaytime();
        _animator.SetFloat(Speed,5f);
    }

    [ContextMenu("Skip To Night")]
    public void SkipToNighttime()
    {
        SetNighttime();
        _animator.SetFloat(Speed,5f);
    }

    public void EnableHouseLights()
    {
        foreach (LightBulb lightBulb in _lights)
        {
            lightBulb.enabled = true;
        }
    }

    public void DisableHouseLights()
    {
        foreach (LightBulb lightBulb in _lights)
        {
            lightBulb.enabled = false;
        }

    }

    public void ResetSpeed()
    {
        _animator.SetFloat(Speed,_animationSpeed);
    }

    private void EndingEvent()
    {
        endingEvent.Invoke();
    }
}
```

9.1.4. Door.cs

```
using System.Collections;
using UnityEngine;

//Door states (indexes match animator int values)
public enum DoorState
{
    OpenIn,
    Closed,
    OpenOut
}
public class Door : MonoBehaviour
{
    [Header("Lock settings")]
    [SerializeField] private bool isLocked;
    [Header("Trigger daytime / ending")]
    [SerializeField] private bool triggerDaytime;
    [Header("KeyType required to unlock")]
    [SerializeField] private KeyType keyType;

    private DoorState _currentState = DoorState.Closed;
    private Animator _animator;
    private readonly float coolDownDelay = 0.5f;
    private float _cooldownTimer;
    private KeyManager _keyManager;
    private DoorTexts _doorTexts;
    private DoorSounds _doorSounds;
    private static readonly int OpenState = Animator.StringToHash("OpenState");

    public DoorState CurrentState => _currentState;

    public bool IsLocked => isLocked;

    public KeyType Type => keyType;

    private void Start()
    {
        _animator = GetComponentInChildren<Animator>();
        _keyManager = FindObjectOfType<KeyManager>();
        _doorTexts = GetComponentInChildren<DoorTexts>();
        _doorSounds = GetComponentInChildren<DoorSounds>();
    }

    private void FixedUpdate()
    {
        if (_cooldownTimer == 0f){return;}

        _cooldownTimer -= Time.deltaTime;

        if (_cooldownTimer <= 0f)
        {
            _cooldownTimer = 0f;
            SetDoorState(_currentState);
        }
    }

    internal void SetDoorState(DoorState newState, float autoCloseDelay = 0f)
    {
        if(_cooldownTimer > 0){return;}

        if (isLocked)
        {
            if (_keyManager.CheckIfKeyHeld(keyType) && newState != DoorState.Closed)
            {
                isLocked = false;
                _doorSounds.PlayDoorUnlockSound();
                _doorTexts.SetDoorText();
            }
        }
        if (!isLocked && _currentState != newState)
        {
            _doorTexts.ClearDoorText();
```

```
            _cooldownTimer = coolDownDelay;
            StartCoroutine(DoorStateDelay(autoCloseDelay,newState));
            if (triggerDaytime)
            {
                DayNightController dayNightController = FindObjectOfType<DayNightController>();
                if (!dayNightController.IsDaytime)
                {
                    dayNightController.SkipToDaytime();
                }
            }
        }
    }

    private IEnumerator DoorStateDelay(float delay, DoorState newState)
    {
        yield return new WaitForSeconds(delay);
        _currentState = newState;
        _animator.SetInteger(OpenState,(int)_currentState);
    }

    public void LockDoor()
    {
        if (isLocked) return;
        isLocked = true;
        StartCoroutine(DoorStateDelay(0,DoorState.Closed));
    }
}
```

9.1.5. DoorLockTrigger.cs

```
using UnityEngine;

public class DoorLockTrigger : MonoBehaviour
{
    private Door _door;

    private void Start()
    {
        _door = transform.parent.GetComponent<Door>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _door.LockDoor();
        }
    }
}
```

9.1.6. DoorSounds.cs

```
using UnityEngine;

public class DoorSounds : MonoBehaviour
{
    [SerializeField] private AudioClip doorCloseSound;
    [SerializeField] private AudioClip doorOpenSound;
    [SerializeField] private AudioClip doorLockSound;
    [SerializeField] private AudioClip doorUnlockSound;

    private Door _door;
    private AudioSource _audioSource;

    void Start()
    {
        _audioSource = GetComponent<AudioSource>();
        _door = transform.parent.GetComponent<Door>();
    }

    public void PlayDoorCloseSound()
    {
        _audioSource.PlayOneShot(doorCloseSound);
        if (_door.IsLocked)
        {
            Invoke(nameof(PlayDoorLockSound),0.25f);
        }
    }

    public void PlayDoorOpenSound()
    {
        _audioSource.PlayOneShot(doorOpenSound);
    }

    public void PlayDoorLockSound()
    {
        _audioSource.PlayOneShot(doorLockSound);
    }

    public void PlayDoorUnlockSound()
    {
        _audioSource.PlayOneShot(doorUnlockSound);
    }
}
```

9.1.7. DoorTexts.cs

```
using TMPro;
using UnityEngine;

public class DoorTexts : MonoBehaviour
{
    [SerializeField] private GameObject[] textPanels;

    private Door _door;
    private KeyManager _keyManager;
    private Transform _player;
    private Animator _animator;
    private static readonly int OpenState = Animator.StringToHash("OpenState");

    private void Start()
    {
        _door = GetComponentInParent<Door>();
        _keyManager = FindObjectOfType<KeyManager>();
        _player = GameObject.FindWithTag("Player").transform;
        _animator = transform.parent.GetComponentInChildren<Animator>();
        ClearDoorText();
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            SetDoorText();
        }
    }

    private void OnTriggerExit(Collider other)
    {
        Invoke(nameof(ClearDoorText),1f);
    }

    public void SetDoorText()
    {
        ClearDoorText();
        GameObject nearestTextPanel = GetNearestTextPanel();
        if(nearestTextPanel == null){return;}
        nearestTextPanel.SetActive(true);
        if (_door.IsLocked)
        {
            if (_keyManager.CheckIfKeyHeld(_door.Type))
            {
                UpdateDoorText(nearestTextPanel.GetComponentInChildren<TMP_Text>(),$"Unlock");
            }
            else
            {
               UpdateDoorText(nearestTextPanel.GetComponentInChildren<TMP_Text>(),
                              $"{KeyManager.GetKeyName(_door.Type)}\nkey\nRequired");
            }
        }
        else
        {
            if(_animator.GetInteger(OpenState) == 1)
            {
                UpdateDoorText(nearestTextPanel.GetComponentInChildren<TMP_Text>(),$"Open");
            }
            else
            {
                ClearDoorText();
            }
        }
    }

    private GameObject GetNearestTextPanel()
    {
        GameObject nearestTextPanel = null;
        float distance = float.MaxValue;
        foreach (GameObject textPanel in textPanels)
```

```
        {
            float doorTextDistance = Vector3.Distance(textPanel.transform.position, _player.position);
            if (doorTextDistance < distance)
            {
                distance = doorTextDistance;
                nearestTextPanel = textPanel;
            }
        }
        return nearestTextPanel;
    }

    public void ClearDoorText()
    {
        foreach (GameObject textPanel in textPanels)
        {
            textPanel.SetActive(false);
        }
    }

    private void UpdateDoorText(TMP_Text nearestDoorText, string newDoorText)
    {
        nearestDoorText.text = newDoorText;
    }
}
```

9.1.8. DoorTrigger.cs

```
using UnityEngine;

public class DoorTrigger : MonoBehaviour
{
    [SerializeField] private DoorState doorState;

    private Door _door;
    private PlayerInputHandler _inputHandler;

    void Start()
    {
        _door = GetComponentInParent<Door>();
        _inputHandler = FindObjectOfType<PlayerInputHandler>();
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (_inputHandler.Interact)
            {
                if (_door.CurrentState.Equals(DoorState.Closed))
                {
                    _door.SetDoorState(doorState);
                }
                else if (_door.CurrentState.Equals(doorState))
                {
                    _door.SetDoorState(DoorState.Closed);
                }
                else
                {
                    _door.SetDoorState(DoorState.Closed);
                }

            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        _door.SetDoorState(DoorState.Closed, 3f);
    }
}
```

9.1.9. EndingTrigger.cs

```
using UnityEngine;
using UnityEngine.Events;

public class EndingTrigger : MonoBehaviour
{
    [SerializeField] private UnityEvent endingEvent;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            endingEvent.Invoke();
        }
    }
}
```

9.1.10. FireBasket.cs

```
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.VFX;
using Random = UnityEngine.Random;

public class FireBasket : MonoBehaviour
{
    [SerializeField] private VisualEffect fireEffect;
    [SerializeField] private PickupType triggerObject;
    [SerializeField] private UnityEvent triggerEvent;

    private Light _fireGlowLight;
    private Vector2 _fireGlowIntensityRange = new Vector2(40,80);

    private void Start()
    {
        _fireGlowLight = GetComponentInChildren<Light>();
    }

    private void FixedUpdate()
    {
        _fireGlowLight.intensity = Mathf.Lerp(_fireGlowLight.intensity,
                                              Random.Range(_fireGlowIntensityRange.x,
                                                           _fireGlowIntensityRange.y),
                                              Time.deltaTime * 10f);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (!other.gameObject.TryGetComponent<Pickup>(out Pickup pickup)) return;
        if (pickup.Type.Equals(triggerObject))
        {
            TriggerAction();
            Destroy(other.gameObject);
        }
    }

    private void TriggerAction()
    {
        _fireGlowIntensityRange *= 2;
        fireEffect.SetFloat("SpawnRate",500f);
        fireEffect.SetVector3("MinVelocity",new Vector3(0.5f,1.5f,0.25f));
        fireEffect.SetVector3("MaxVelocity",new Vector3(0.5f,2f,0.25f));
        fireEffect.SetFloat("BaseSize",0.75f);
        Invoke(nameof(EnableItem),0.5f);
    }

    private void EnableItem()
    {
        triggerEvent.Invoke();
    }
}
```

9.1.11. FirstPersonController.cs

```
using UnityEngine;
#if ENABLE_INPUT_SYSTEM && STARTER_ASSETS_PACKAGES_CHECKED
using UnityEngine.InputSystem;
#endif

/* Note: animations are called via the controller for both the character and capsule using animator
   null checks
 */

namespace StarterAssets
{
   [RequireComponent(typeof(CharacterController))]
#if ENABLE_INPUT_SYSTEM && STARTER_ASSETS_PACKAGES_CHECKED
   [RequireComponent(typeof(PlayerInputHandler))]
#endif
   public class FirstPersonController : MonoBehaviour
   {
      [Header("Player")]
      [Tooltip("Move speed of the character in m/s")]
      [SerializeField] private float MoveSpeed = 4.0f;
      [Tooltip("Sprint speed of the character in m/s")]
      [SerializeField] private float SprintSpeed = 6.0f;
      [Tooltip("Rotation speed of the character")]
      [SerializeField] private float RotationSpeed = 1.0f;
      [Tooltip("Acceleration and deceleration")]
      [SerializeField] private float SpeedChangeRate = 10.0f;

      [Space(10)]
      [Tooltip("The height the player can jump")]
      [SerializeField] private float JumpHeight = 1.2f;
      [Tooltip("The character uses its own gravity value. The engine default is -9.81f")]
      [SerializeField] private float Gravity = -15.0f;

      [Space(10)]
      [Tooltip("Time required to pass before being able to jump again. Set to 0f to instantly jump
                again")]
      [SerializeField] private float JumpTimeout = 0.1f;
      [Tooltip("Time required to pass before entering the fall state. Useful for walking down stairs")]
      [SerializeField] private float FallTimeout = 0.15f;

      [Header("Player Grounded")]
      [Tooltip("If the character is grounded or not. Not part of the CharacterController built in
                grounded check")]
      [SerializeField] private bool Grounded = true;
      [Tooltip("Useful for rough ground")]
      [SerializeField] private float GroundedOffset = -0.14f;
      [Tooltip("The radius of the grounded check. Should match the radius of the CharacterController")]
      [SerializeField] private float GroundedRadius = 0.5f;
      [Tooltip("What layers the character uses as ground")]
      [SerializeField] private LayerMask GroundLayers;

      [Header("Cinemachine")]
      [Tooltip("The follow target set in the Cinemachine Virtual Camera that the camera will follow")]
      [SerializeField] private GameObject CinemachineCameraTarget;
      [Tooltip("How far in degrees can you move the camera up")]
      [SerializeField] private float TopClamp = 90.0f;
      [Tooltip("How far in degrees can you move the camera down")]
      [SerializeField] private float BottomClamp = -90.0f;

      // cinemachine
      private float _cinemachineTargetPitch;

      // player
      private float _speed;
      private float _rotationVelocity;
      private float _verticalVelocity;
      private float _terminalVelocity = 53.0f;

      // timeout deltatime
      private float _jumpTimeoutDelta;
      private float _fallTimeoutDelta;
```

```
    private CharacterController _controller;
    private PlayerInputHandler inputHandler;
    private GameObject _mainCamera;

    private const float _threshold = 0.01f;

    //Footsteps sounds
    // Exposed audio variables
    [Header("Audio")]
    [Tooltip("An array of footstep sounds. One gets randonly selected to play")]
    [SerializeField] private AudioClip[] footstepSounds;
    [Tooltip("Effects the gap between footstep sounds. Smaller number = smaller gap")]
    [Min(1.0f)] [SerializeField] private float stepRate = 1.0f;

    // Private audio variables
    private float nextStep = 0.0f;
    private AudioSource audioSource;

    private void Awake()
    {
        // get a reference to our main camera
        if (_mainCamera == null)
        {
            _mainCamera = GameObject.FindGameObjectWithTag("MainCamera");
        }
    }

    private void Start()
    {
        _controller = GetComponent<CharacterController>();
        inputHandler = GetComponent<PlayerInputHandler>();

        // reset our timeouts on start
        _jumpTimeoutDelta = JumpTimeout;
        _fallTimeoutDelta = FallTimeout;

        audioSource = GetComponent<AudioSource>();
    }

    private void Update()
    {
        JumpAndGravity();
        GroundedCheck();
        Move();
    }

    private void LateUpdate()
    {
        CameraRotation();
    }

    private void GroundedCheck()
    {
        // set sphere position, with offset
        Vector3 spherePosition = new Vector3(transform.position.x,
                                             transform.position.y - GroundedOffset,
                                             transform.position.z);
        Grounded = Physics.CheckSphere(spherePosition,
                                       GroundedRadius,
                                       GroundLayers,
                                       QueryTriggerInteraction.Ignore);
    }

    private void CameraRotation()
    {
        // if there is an input
        if (inputHandler.Look.sqrMagnitude >= _threshold)
        {
            _cinemachineTargetPitch += inputHandler.Look.y * RotationSpeed * Time.deltaTime;
            _rotationVelocity = inputHandler.Look.x * RotationSpeed * Time.deltaTime;

            // clamp our pitch rotation
```

```
            _cinemachineTargetPitch = ClampAngle(_cinemachineTargetPitch, BottomClamp, TopClamp);

            // Update Cinemachine camera target pitch
            CinemachineCameraTarget.transform.localRotation = Quaternion.Euler(_cinemachineTargetPitch,
                                                                               0.0f,
                                                                               0.0f);
            // rotate the player left and right
            transform.Rotate(Vector3.up * _rotationVelocity);
        }
    }

    private void Move()
    {
        // set target speed based on move speed, sprint speed and if sprint is pressed
        float targetSpeed = inputHandler.Sprint ? SprintSpeed : MoveSpeed;

        // a simplistic acceleration and deceleration designed to be easy to remove, replace, or
           iterate upon

        // note: Vector2's == operator uses approximation so is not floating point error prone, and is
           cheaper than magnitude
        // if there is no input, set the target speed to 0
        if (inputHandler.Move == Vector2.zero) targetSpeed = 0.0f;

        // a reference to the players current horizontal velocity
        float currentHorizontalSpeed = new Vector3(_controller.velocity.x,
                                                   0.0f,
                                                   _controller.velocity.z).magnitude;

        float speedOffset = 0.1f;
        float inputMagnitude = inputHandler.AnalogMovement ? inputHandler.Move.magnitude : 1f;

        // accelerate or decelerate to target speed
        if (currentHorizontalSpeed < targetSpeed - speedOffset
            || currentHorizontalSpeed > targetSpeed + speedOffset)
        {
            // creates curved result rather than a linear one giving a more organic speed change
            // note T in Lerp is clamped, so we don't need to clamp our speed
            _speed = Mathf.Lerp(currentHorizontalSpeed,
                                targetSpeed * inputMagnitude,
                                Time.deltaTime * SpeedChangeRate);

            // round speed to 3 decimal places
            _speed = Mathf.Round(_speed * 1000f) / 1000f;
        }
        else
        {
            _speed = targetSpeed;
        }

        // normalise input direction
        Vector3 inputDirection = new Vector3(inputHandler.Move.x,
                                             0.0f,
                                             inputHandler.Move.y).normalized;

        // note: Vector2's != operator uses approximation so is not floating point error prone, and is
           cheaper than magnitude
        // if there is a move input rotate player when the player is moving
        if (inputHandler.Move != Vector2.zero)
        {
            // move
            inputDirection = transform.right * inputHandler.Move.x
                           + transform.forward * inputHandler.Move.y;
        }

        // move the player
        _controller.Move(inputDirection.normalized * (_speed * Time.deltaTime)
                         + new Vector3(0.0f, _verticalVelocity, 0.0f) * Time.deltaTime);

        PlayFootStepAudio();

    }
```

```
        private void JumpAndGravity()
        {
            if (Grounded)
            {
                // reset the fall timeout timer
                _fallTimeoutDelta = FallTimeout;

                // stop our velocity dropping infinitely when grounded
                if (_verticalVelocity < 0.0f)
                {
                    _verticalVelocity = -2f;
                }

                // Jump
                if (inputHandler.Jump && _jumpTimeoutDelta <= 0.0f)
                {
                    // the square root of H * -2 * G = how much velocity needed to reach desired height
                    _verticalVelocity = Mathf.Sqrt(JumpHeight * -2f * Gravity);
                }

                // jump timeout
                if (_jumpTimeoutDelta >= 0.0f)
                {
                    _jumpTimeoutDelta -= Time.deltaTime;
                }
            }
            else
            {
                // reset the jump timeout timer
                _jumpTimeoutDelta = JumpTimeout;

                // fall timeout
                if (_fallTimeoutDelta >= 0.0f)
                {
                    _fallTimeoutDelta -= Time.deltaTime;
                }

                // if we are not grounded, do not jump
                inputHandler.Jump = false;
            }

            // apply gravity over time if under terminal (multiply by delta time twice to linearly speed
                up over time)
            if (_verticalVelocity < _terminalVelocity)
            {
                _verticalVelocity += Gravity * Time.deltaTime;
            }
        }

        private static float ClampAngle(float lfAngle, float lfMin, float lfMax)
        {
            if (lfAngle < -360f) lfAngle += 360f;
            if (lfAngle > 360f) lfAngle -= 360f;
            return Mathf.Clamp(lfAngle, lfMin, lfMax);
        }

        private void OnDrawGizmosSelected()
        {
            Color transparentGreen = new Color(0.0f, 1.0f, 0.0f, 0.35f);
            Color transparentRed = new Color(1.0f, 0.0f, 0.0f, 0.35f);

            if (Grounded) Gizmos.color = transparentGreen;
            else Gizmos.color = transparentRed;

            // when selected, draw a gizmo in the position of, and matching radius of, the grounded
                collider
            Gizmos.DrawSphere(new Vector3(transform.position.x,
                                          transform.position.y - GroundedOffset,
                                          transform.position.z),
                              GroundedRadius);
        }

        private void PlayFootStepAudio()
```

```
        {
            // Debug.Log("Next " + nextStep);
            if (Grounded && _speed > 0.0f && Time.time > nextStep)
            {
                // Debug.Log("Time " + Time.time);
                float offset = _speed;
                if ( _speed >= stepRate ) {
                    offset = (stepRate / _speed);
                }
                nextStep = Time.time + offset;
                // pick & play a random footstep sound from the array,
                // excluding sound at index 0
                int n = Random.Range(1, footstepSounds.Length);
                audioSource.clip = footstepSounds[n];
                audioSource.PlayOneShot(audioSource.clip);
                // move picked sound to index 0 so it's not picked next time
                footstepSounds[n] = footstepSounds[0];
                footstepSounds[0] = audioSource.clip;
            }
        }
    }
}
```

9.1.12. InsideHouseController.cs

```
using UnityEngine;

public class InsideHouseController : MonoBehaviour
{
    [SerializeField] private AudioSource[] outsideAudioSource;
    [SerializeField] private AudioSource[] insideAudioSource;

    private bool _isInside = false;

    public bool IsInside => _isInside;

    private void Start()
    {
        SetInside(_isInside);
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (!_isInside)
            {
                _isInside = true;
                SetInside(_isInside);
            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _isInside = false;
            SetInside(_isInside);
        }
    }

    public void SetInside(bool isInside)
    {
        if (isInside)
        {
            foreach (AudioSource audioSource in outsideAudioSource)
            {
                audioSource.volume = 0.2f;
            }

            foreach (AudioSource audioSource in insideAudioSource)
            {
                audioSource.volume = 1f;
            }
        }
        else
        {
            foreach (AudioSource audioSource in outsideAudioSource)
            {
                audioSource.volume = 1f;
            }

            foreach (AudioSource audioSource in insideAudioSource)
            {
                audioSource.volume = 0f;
            }
        }
    }
}
```

9.1.13. ItemGrabber.cs

```
using System.Collections.Generic;
using UnityEngine;

public class ItemGrabber : MonoBehaviour
{
    [SerializeField] private LayerMask pickupLayerMask;
    [SerializeField] private float raycastDistance = 5f;
    [SerializeField] private Transform cameraTransform;
    [SerializeField] [Range(0f,20f)] private float throwForce = 10f;

    private readonly List<GameObject> _heldItems = new List<GameObject>();
    private KeyManager _keyManager;
    private PlayerInputHandler _playerInputHandler;
    private UIController _uiController;
    private GameObject _currentItem;
    private int _currentItemIndex;
    private bool _inCooldown;

    private void Start()
    {
        _playerInputHandler = FindObjectOfType<PlayerInputHandler>();
        _keyManager = FindObjectOfType<KeyManager>();
        _uiController = FindObjectOfType<UIController>();
    }

    private void Update()
    {
        if (Physics.Raycast(cameraTransform.position,
                            cameraTransform.forward,
                            out RaycastHit hitInfo,
                            raycastDistance,
                            pickupLayerMask,
                            QueryTriggerInteraction.Collide))
        {
            if (hitInfo.transform.TryGetComponent<Key>(out Key key))
            {
                _uiController.SetInfoDisplay($"Pick up {KeyManager.GetKeyName(key.DoorToOpen)} key");
            }
            else if (hitInfo.transform.TryGetComponent<Pickup>(out Pickup pickup))
            {
                _uiController.SetInfoDisplay($"Pick up {pickup.Type}");
            }
        }
        else
        {
            _uiController.SetInfoDisplay("");
        }

        if(_inCooldown){return;}

        if (_playerInputHandler.Throw)
        {
            if (hitInfo.transform)
            {

                if (hitInfo.transform.CompareTag("Key"))
                {
                    if (hitInfo.transform.TryGetComponent<Key>(out Key key))
                    {
                        _keyManager.AddKey(key.DoorToOpen);
                        Destroy(hitInfo.transform.gameObject);
                    }
                    _inCooldown = true;
                    Invoke(nameof(ResetCooldown),0.2f);
                    return;
                }

                if (!hitInfo.transform.CompareTag("Pickup") ||
                    _heldItems.Contains(hitInfo.transform.gameObject)) return;
                _currentItem = hitInfo.transform.gameObject;
                _heldItems.Add(_currentItem);
```

```
                _currentItemIndex = _heldItems.IndexOf(_currentItem);
                SetActiveObject();
                _currentItem.GetComponent<Rigidbody>().isKinematic = true;
                _currentItem.GetComponent<Collider>().enabled = false;
                Transform grabberTransform = transform;
                _currentItem.transform.parent = grabberTransform;
                _currentItem.transform.position = grabberTransform.position;
                _currentItem.transform.rotation = grabberTransform.rotation;
                _currentItem.transform.localScale = new Vector3(0.25f,0.25f,0.25f);
                _inCooldown = true;
                Invoke(nameof(ResetCooldown),0.2f);
            }
            else
            {
                if (!_currentItem) return;
                GameObject itemToThrow = _currentItem;
                _heldItems.Remove(itemToThrow);
                if (_heldItems.Count > 0)
                {
                    _currentItem = _heldItems[_heldItems.Count - 1];
                    _currentItemIndex = _heldItems.IndexOf(_currentItem);
                    SetActiveObject();
                }
                else
                {
                    _currentItem = null;
                }
                itemToThrow.transform.parent = null;
                itemToThrow.GetComponent<Collider>().enabled = true;
                itemToThrow.TryGetComponent<Rigidbody>(out Rigidbody itemRb);
                itemRb.isKinematic = false;
                itemRb.AddForce(cameraTransform.forward * (throwForce * 100f), ForceMode.Acceleration);
                itemToThrow.transform.localScale = new Vector3(1, 1, 1);
                _inCooldown = true;
                Invoke(nameof(ResetCooldown),0.2f);
            }
        }
        else
        {
            if (_playerInputHandler.NextItem)
            {
                NextItem();
            }
            else if (_playerInputHandler.PreviousItem)
            {
                PreviousItem();
            }
        }
    }

    private void NextItem()
    {
        int numberOfItems = _heldItems.Count;
        if(numberOfItems <= 1){return;}
        if (_currentItemIndex == numberOfItems-1)
        {
            _currentItemIndex = 0;
        }
        else
        {
            _currentItemIndex++;
        }
        _currentItem = _heldItems[_currentItemIndex];
        SetActiveObject();
        _inCooldown = true;
        Invoke(nameof(ResetCooldown),0.2f);
    }

    private void PreviousItem()
    {
        int numberOfItems = _heldItems.Count;
        if(numberOfItems <= 1){return;}
```

```
        if (_currentItemIndex == 0)
        {
            _currentItemIndex = numberOfItems - 1;
        }
        else
        {
            _currentItemIndex--;
        }
        _currentItem = _heldItems[_currentItemIndex];
        SetActiveObject();
        _inCooldown = true;
        Invoke(nameof(ResetCooldown),0.2f);
    }

    private void SetActiveObject()
    {
        if (_heldItems.Count <= 0) return;

        for (int i = 0; i < _heldItems.Count; i++)
        {
            if (i == _currentItemIndex)
            {
                _heldItems[i].SetActive(true);
            }
            else
            {
                _heldItems[i].SetActive(false);
            }
        }
    }

    private void ResetCooldown()
    {
        _inCooldown = false;
    }
}
```

9.1.14. Key.cs

```
using UnityEngine;


public class Key : MonoBehaviour
{
    [SerializeField] private KeyType doorToOpen;

    public KeyType DoorToOpen => doorToOpen;
}
```

9.1.15. KeyDisplay.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class KeyDisplay : MonoBehaviour
{
    [SerializeField] private TMP_Text keyDisplayText;
    [SerializeField] private Image crossImage;

    private KeyType _keyType;

    public KeyType Type => _keyType;

    public void SetKeyType(KeyType keyType)
    {
        _keyType = keyType;
        keyDisplayText.text = $"{KeyManager.GetKeyName(keyType)}";
    }

    public void SetCollected(bool collected)
    {
        crossImage.enabled = !collected;
    }
}
```

9.1.16. KeyManager.cs

```
using System.Collections.Generic;
using UnityEngine;

public enum KeyType
{
    BackDoor,
    Bathroom,
    Bedroom,
    Conservatory,
    FrontDoor,
    Kitchen,
    Lounge
}

public class KeyManager : MonoBehaviour
{
    private List<KeyType> keysFound = new List<KeyType>();
    private UIController _uiController;

    private void Start()
    {
        _uiController = FindObjectOfType<UIController>();
    }

    public void AddKey(KeyType keyToAdd)
    {
        if (!keysFound.Contains(keyToAdd))
        {
            keysFound.Add(keyToAdd);
            _uiController.UpdateKeys(keysFound);
        }
    }

    public bool CheckIfKeyHeld(KeyType keyToCheck)
    {
        foreach (KeyType key in keysFound)
        {
            if (key.Equals(keyToCheck))
            {
                return true;
            }
        }
        return false;
    }

    public static string GetKeyName(KeyType keyType)
    {
        switch (keyType)
        {
            case KeyType.BackDoor:
                return "Back Door";
            case KeyType.FrontDoor:
                return "Front Door";
            default:
                return keyType.ToString();
        }
    }
}
```

9.1.17. LightBulb.cs

```
using UnityEngine;
using Random = UnityEngine.Random;

public class LightBulb : MonoBehaviour
{
    [SerializeField] private bool flickering;
    [SerializeField] private Vector2 flickerDelay = new Vector2(0.1f, 5f);
    [SerializeField] private Vector2 flickerOffTime = new Vector2(0.01f, 0.1f);

    private float _flickerDelayTimer;
    private Light[] _bulbs;

    private void Awake()
    {
        _bulbs = GetComponentsInChildren<Light>();
    }

    private void FixedUpdate()
    {
        if (flickering)
        {
            _flickerDelayTimer -= Time.deltaTime;
            if (_flickerDelayTimer <= 0)
            {
                SwitchLightsOff();
                _flickerDelayTimer = Random.Range(flickerDelay.x, flickerDelay.y);
                Invoke(nameof(SwitchLightsOn),Random.Range(flickerOffTime.x,flickerOffTime.y));
            }
        }
    }

    private void OnEnable()
    {
        SwitchLightsOn();
    }

    private void SwitchLightsOn()
    {
        foreach (Light bulb in _bulbs)
        {
            bulb.enabled = true;
        }
    }

    private void OnDisable()
    {
        SwitchLightsOff();
    }

    private void SwitchLightsOff()
    {
        foreach (Light bulb in _bulbs)
        {
            bulb.enabled = false;
        }
    }
}
```

9.1.18. Pendulum.cs

```
using UnityEngine;

public class Pendulum : MonoBehaviour
{
    [SerializeField] private AudioClip tickSound;
    [SerializeField] private AudioClip tockSound;

    private AudioSource _audioSource;
    private bool _isTickNext;

    private void Start()
    {
        _audioSource = GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Pendulum"))
        {
            if (_isTickNext)
            {
                _audioSource.PlayOneShot(tickSound);
            }
            else
            {
                _audioSource.PlayOneShot(tockSound);
            }
            _isTickNext = !_isTickNext;
        }
    }
}
```

9.1.19. PianoTrigger.cs

```
using UnityEngine;

public class PianoTrigger : MonoBehaviour
{
    private AudioSource _audioSource;

    private bool _isPlaying;
    private void Start()
    {
        _audioSource = transform.parent.GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if(_isPlaying){return;}
        if (other.CompareTag("Player"))
        {
            _audioSource.Play();
            _isPlaying = true;
        }
    }
}
```

9.1.20. Pickup.cs

```
using UnityEngine;

public enum PickupType
{
    Cuckoo,
    Vase,
    FruitBowl
}

public class Pickup : MonoBehaviour
{
    [SerializeField] private PickupType pickupType;

    public PickupType Type => pickupType;

    public static string GetPickupName(PickupType pickupType)
    {
        switch (pickupType)
        {
            case PickupType.FruitBowl:
                return "Fruit Bowl";
            default:
                return pickupType.ToString();
        }
    }
}
```

9.1.21. PlayerInputHandler.cs

```
using UnityEngine;
#if ENABLE_INPUT_SYSTEM && STARTER_ASSETS_PACKAGES_CHECKED
using UnityEngine.InputSystem;
#endif

public class PlayerInputHandler : MonoBehaviour
{
    [Header("Movement Settings")]
    [SerializeField] private bool analogMovement;

    [Header("Mouse Cursor Settings")]
    [SerializeField] private bool lockCursor;

    private PlayerInput _input;

    private Vector2 _move;
    private Vector2 _look;
    private bool _jump;
    private bool _sprint;
    private bool _interact;
    private bool _throw;
    private bool _nextItem;
    private bool _previousItem;

    public Vector2 Move => _move;
    public Vector2 Look => _look;

    public bool Jump
    {
        get => _jump;
        set => _jump = value;
    }

    public bool Sprint => _sprint;

    public bool Interact => _interact;

    public bool Throw => _throw;

    public bool AnalogMovement => analogMovement;

    public bool NextItem => _nextItem;

    public bool PreviousItem => _previousItem;

    private void Start()
    {
        _input = GetComponent<PlayerInput>();
        if (lockCursor)
        {
            Cursor.lockState = CursorLockMode.Locked;
        }
    }

    public void OnControlsChanged()
    {
        print(_input.currentControlScheme);
    }

   public void OnMove(InputValue value)
   {
      _move = value.Get<Vector2>();
   }

   public void OnLook(InputValue value)
   {
      _look = value.Get<Vector2>();
   }

   public void OnJump(InputValue value)
   {
```

```
        _jump = value.isPressed;
    }

    public void OnSprint(InputValue value)
    {
        _sprint = value.isPressed;
    }

     public void OnInteract(InputValue value)
     {
        _interact = value.isPressed;
     }

     public void OnThrow(InputValue value)
     {
        _throw = value.isPressed;
     }

     public void OnNextItem(InputValue value)
     {
        _nextItem = value.isPressed;
     }

     public void OnPreviousItem(InputValue value)
     {
        _previousItem = value.isPressed;
     }
}
```

9.1.22. ServingHatchTrigger.cs

```
using UnityEngine;

public class ServingHatchTrigger : MonoBehaviour
{
    private Animator _animator;
    private AudioSource _audioSource;
    private bool _isPlaying;
    private static readonly int IsActive = Animator.StringToHash("isActive");

    // Start is called before the first frame update
    void Start()
    {
        _animator = transform.parent.GetComponentInChildren<Animator>();
        _audioSource = GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if(_isPlaying){return;}

        if (other.CompareTag("Player"))
        {
            _isPlaying = true;
            _audioSource.Play();
            _animator.SetBool(IsActive,true);
        }
    }
}
```

9.1.23. SpotlightTrigger.cs

```
using UnityEngine;

public class SpotlightTrigger : MonoBehaviour
{
    [SerializeField] private float lightTimeout = 5f;

    private LightBulb _light;
    private bool _playerIsInTrigger;
    private float _lightTimer;

    private void Start()
    {
        _light = transform.parent.GetComponentInChildren<LightBulb>();
        _light.enabled = false;
        _lightTimer = lightTimeout;
    }

    private void Update()
    {
        if (_playerIsInTrigger)
        {
            _lightTimer = lightTimeout;
        }
        else
        {
            _lightTimer -= Time.deltaTime;
        }

        if (!(_lightTimer < 0f)) return;
        _light.enabled = false;
        _lightTimer = lightTimeout;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        _playerIsInTrigger = true;
        _light.enabled = true;
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerIsInTrigger = false;
        }
    }
}
```

9.1.24. TVTrigger.cs

```
using UnityEngine;

public class TVTrigger : MonoBehaviour
{
    [SerializeField] private GameObject screen;

    private void OnTriggerEnter(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        screen.SetActive(true);
    }
}
```

9.1.25. UIController.cs

```
using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class UIController : MonoBehaviour
{
    [SerializeField] private Transform keysPanel;
    [SerializeField] private GameObject keyDisplayPrefab;
    [SerializeField] private TMP_Text infoDisplay;
    [SerializeField] private GameObject startText;

    private void Start()
    {
        foreach (KeyType keyType in Enum.GetValues(typeof(KeyType)))
        {
            GameObject keyDisplay = Instantiate(keyDisplayPrefab, keysPanel);
            keyDisplay.GetComponent<KeyDisplay>().SetKeyType(keyType);
        }
        keysPanel.parent.gameObject.SetActive(false);
        Invoke(nameof(RemoveStartText),5f);
    }

    public void UpdateKeys(List<KeyType> keys)
    {
        foreach (Transform keyItem in keysPanel)
        {
            if (keyItem.TryGetComponent<KeyDisplay>(out KeyDisplay keyDisplay))
            {
                if (keys.Contains(keyDisplay.Type))
                {
                    keyDisplay.SetCollected(true);
                }
                else
                {
                    keyDisplay.SetCollected(false);
                }
            }
        }
    }

    public void SetInfoDisplay(string textToDisplay)
    {
        infoDisplay.text = textToDisplay;
    }

    private void RemoveStartText()
    {
        startText.SetActive(false);
        keysPanel.parent.gameObject.SetActive(true);
    }
}
```

9.1.26. Vase.cs

```
using UnityEngine;

public class Vase : MonoBehaviour
{
    [SerializeField] private GameObject vaseModel;
    [SerializeField] private GameObject brokenVaseModel;
    [SerializeField] private GameObject keyCollectible;

    private Material _vaseMaterial;
    private float _shaderProgress = 1;
    private bool _isAppearing;
    private readonly float smashForce = 9f;
    private bool _alreadyAppeared;
    private static readonly int Progress = Shader.PropertyToID("_Progress");

    private void Update()
    {
        if (_isAppearing)
        {
            _shaderProgress = Mathf.Clamp01(_shaderProgress - Time.deltaTime * 0.5f);
            if (_shaderProgress <= 0)
            {
                _isAppearing = false;
            }
            _vaseMaterial.SetFloat(Progress,_shaderProgress);
        }
    }

    private void OnEnable()
    {
        if (!_alreadyAppeared)
        {
            _vaseMaterial = vaseModel.GetComponentInChildren<Renderer>().material;
            _shaderProgress = 1;
            _vaseMaterial.SetFloat(Progress,_shaderProgress);
            _isAppearing = true;
            _alreadyAppeared = true;
        }
    }

    private void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.CompareTag("Player")) return;
        if (other.relativeVelocity.x > smashForce || other.relativeVelocity.y > smashForce ||
other.relativeVelocity.z > smashForce)
        {
            SmashVase();
        }
    }

    private void SmashVase()
    {
        GetComponent<AudioSource>().Play();
        vaseModel.SetActive(false);
        GetComponent<Collider>().enabled = false;
        brokenVaseModel.SetActive(true);
        keyCollectible.SetActive(true);
        Invoke(nameof(RemoveVelocity),0.2f);
    }

    private void RemoveVelocity()
    {
        foreach (Rigidbody rb in GetComponents<Rigidbody>())
        {
            rb.isKinematic = true;
            rb.velocity = Vector3.zero;
        }
    }

}
```

9.1.27. Wolves.cs

```
using UnityEngine;
using Random = UnityEngine.Random;

public class Wolves : MonoBehaviour
{
    [SerializeField] private AudioClip[] wolfSounds;
    [SerializeField] private Vector2 delayMinMax = new Vector2(10f, 20f);

    private DayNightController _dayNightController;
    private AudioSource _audioSource;
    private float _delayTimer;

    private void Start()
    {
        _audioSource = GetComponent<AudioSource>();
        _dayNightController = GetComponent<DayNightController>();
        ResetDelayTimer();
        PlayRandomWolfSound();
    }

    private void FixedUpdate()
    {
        if(_dayNightController.IsDaytime){return;}
        _delayTimer -= Time.deltaTime;
        if (!(_delayTimer <= 0f)) return;
        PlayRandomWolfSound();
        ResetDelayTimer();
    }

    private void ResetDelayTimer()
    {
        _delayTimer = Random.Range(delayMinMax.x, delayMinMax.y);
    }

    private void PlayRandomWolfSound()
    {
        int indexToPlay = Random.Range(1, wolfSounds.Length);
        AudioClip soundToPlay = wolfSounds[indexToPlay];
        _audioSource.PlayOneShot(soundToPlay);
        (wolfSounds[0], wolfSounds[indexToPlay]) = (wolfSounds[indexToPlay], wolfSounds[0]);
    }
}
```