The narrative I chose for my game was one of a small person living in the human world. The idea is that you must go and acquire items which can be used back home. The list of items is given to the player in the form of a quest. The advanced techniques I chose to implement are loading levels and level-design, object instantiation and destruction and lighting.

I started by creating the questing system for my game which consists of two main classes – Goal and Quest – which others inherit from. Goal outlines how to check if a quest is complete and Quest outlines the general content of a quest. I only have one type of goal implemented – GatherGoal – but due to the structure different goals can easily be created.

Quest is designed so that all quests can inherit from it, all it requires is a list of goals. This list links to the type of goal you added, in my case they are all gather goals, so they all make a reference to that script. It is here that the check is run to see if the item is needed and the appropriate action is done[1]. This is run for each goal for each is a new instantiation of the script GatherGoal[2].

In order to assign the quests to the player I created a questGiver script. Initially this just attached the script which held that quest to an empty game object, which in turn would update the UI with the quest requirements. It also toggled the quest canvas on, giving the player a visual representation of receiving the quest (this is then disabled by “accepting” the quest). However, this proved problematic with multiple quests for it would assign the quest even if the old one was not yet complete, something not part of the design. To change this, I created a reference to the previous quest. The script checks if this is complete before assigning the new quest[3][4]. However, this created the need for a “dummy quest” so the first quest can be assigned[5]. This structure can be used on a wider scale for quest chains.

I was originally going to have an inventory system, however this felt redundant for the player can’t interact with the items beyond picking them up. For this reason, I decided not to include the system in my game. The design of this system was set up to retrieve a database of items via JSON and store them in a list so they could be accessed by other components. If the player picked up an item the UI inventory would be updated with its details and, upon use, would remove it from the inventory.

I used the player model from the Standard Assets pack; however, I created my own movement and animation for it. The player movement is controlled via a joystick. This works by getting the position of the touch within the defined space and translating it to a vector[6], which is used to determine what force is added to the player[7]. The original implementation resulted in force being applied relative to the world, not the player. To change this, I adjusted it using the cameras direction[8][9]. Transitioning between animations is also controlled by this vector.

Level design largely involves the way I implemented my questing system. Its design allows the flexibility to fit any size game, allowing as many quests to be created as desired. My game loads different rooms triggered by the player colliding with a door in the scene. Scenes are loaded asynchronously (the one being loaded is done so in the background) and upon switch sets the previous scene to inactive. I also make use of SceneManager.MoveGameObjectToScene() in order to transition the player, main canvas, and other needed objects from scene to scene[10].

A couple of my quest items – pins and batteries – spawn using an object pool[11]. This is also randomised meaning that the spawn locations move each time the scene is loaded[12][13]. This is done by deciding how many objects should be spawned (a random number between a defined minimum and the number of spawns available) and then picking a random spawn point that many times. A check is done when deciding which points to use so there are no repeats[14]. Originally, they didn’t spawn when transitioning scenes while playing; this was fixed by creating them as a child of a game object. Item.setActive(false) is used upon pickup so the item is never destroyed and can be reactivated upon reload.

Considering most of the objects never move, a lot of lighting in my scenes is baked. For the small amount that do – the player and the quest items – I set up multiple light probes within scenes[15]. This method reduces the power needed to generate the lighting within a scene for most of it is done already.

Alongside my lighting I also altered other settings in order to optimise for mobile. Due to the player being the only object that needs to collide, all other collisions between layers where turned off. I also decreased the quality of the shadows. All assets I made myself are low poly in order to minimise rendering time. Audio is compressed and loaded in a way designed to reduce the resources needed to play it; the way this is done varying depending on what the sound is used for.

Overall, the game turned out as expected. However, there was a few features I wanted to implement which I didn’t due to several reasons. The inventory being one, which later felt redundant. The other was the idea of a cat that would patrol one of the rooms. If you were spotted, it would chase you and when caught it would be game over. One of the reasons I didn’t implement this was the model I had was incredibly high poly and new animations would have needed to be created.

There’re more ways in which I could have optimised the game. However, due to its small size minute adjustments made little to no difference compared to the impact they would have on a larger game.

Resources:

Character and animation – Standard Assets pack Unity Store

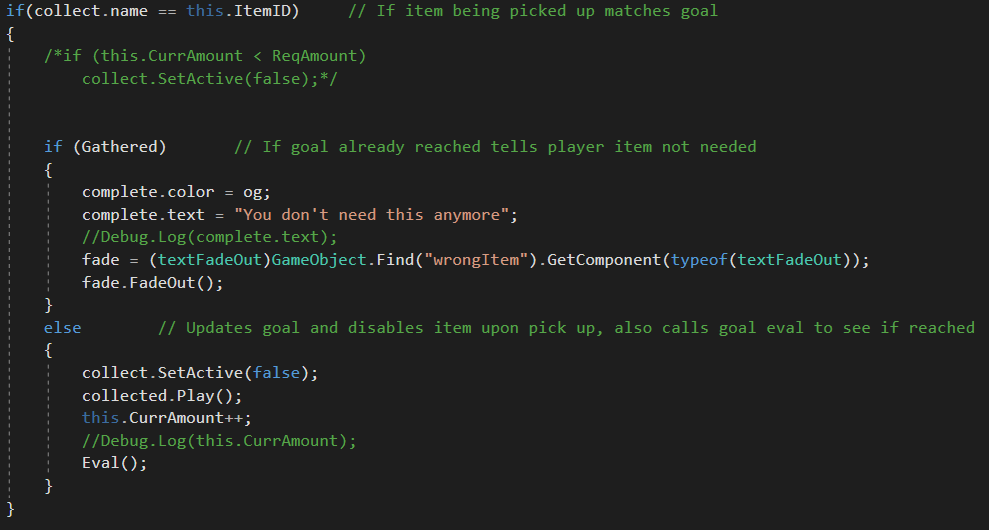
Furniture – Toon Furniture and Big Furniture pack Unity Store

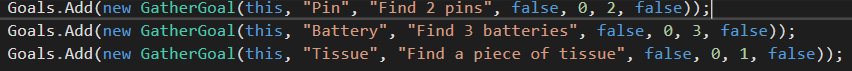
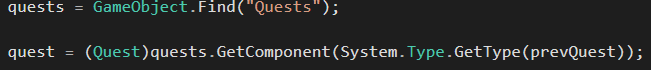
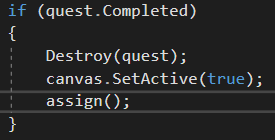
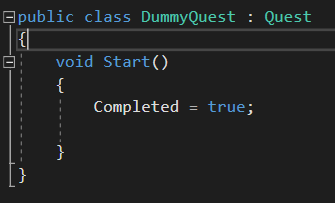
Joystick images – Joystick pack Unity Store

Models made – All items (pin, battery, tissue box and sugar bowl), TV, fireplace and notice board

Item collected sound – zapsplat.net

Quest complete and background sound – freesound.org





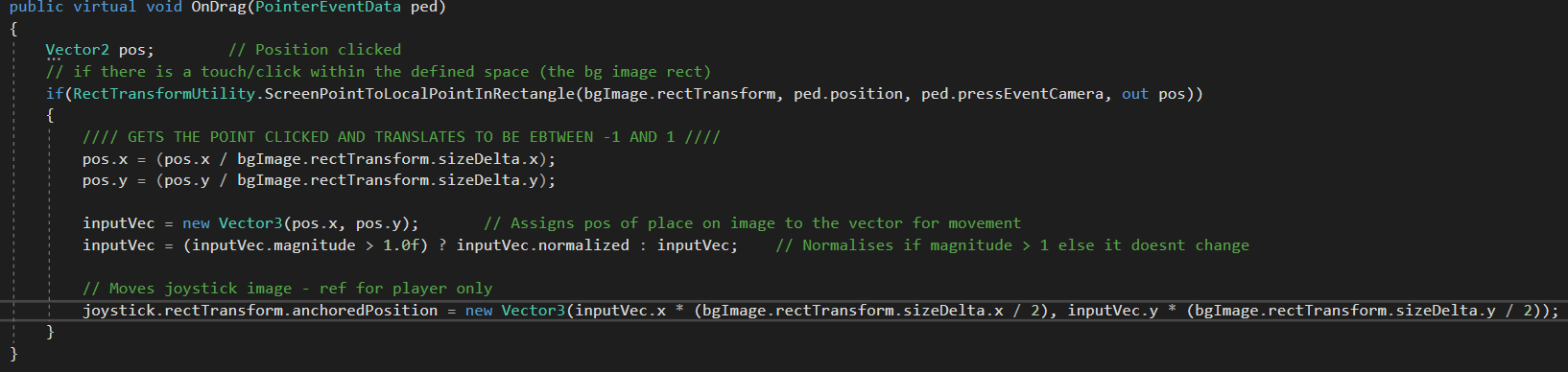
[5] Dummy quest which is attached in inspector so is there on game start, used to give a reference for the first quests prevQuest field, not an actual quest

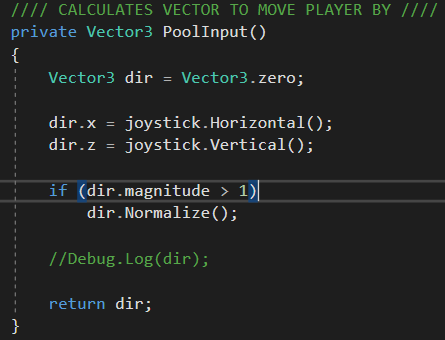
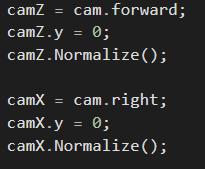
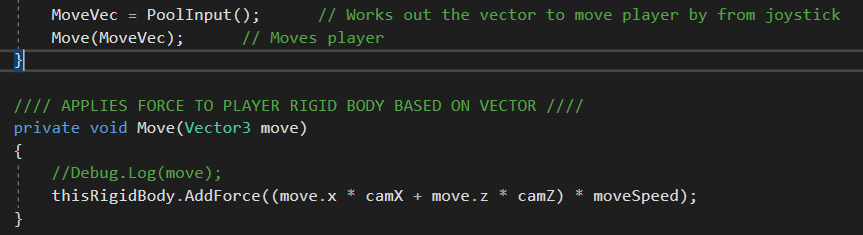
[3] (Above) Finding the previous quest and setting it to a variable so its components can be accessed

[4] (Left) If previous is complete, removes it and assigns the new quest

[2] List of goals for a quest – each a new instantiation of a goal

[1] If the item is the one the quest needs it checks if the goal has already been met (Gathered) and does the required action



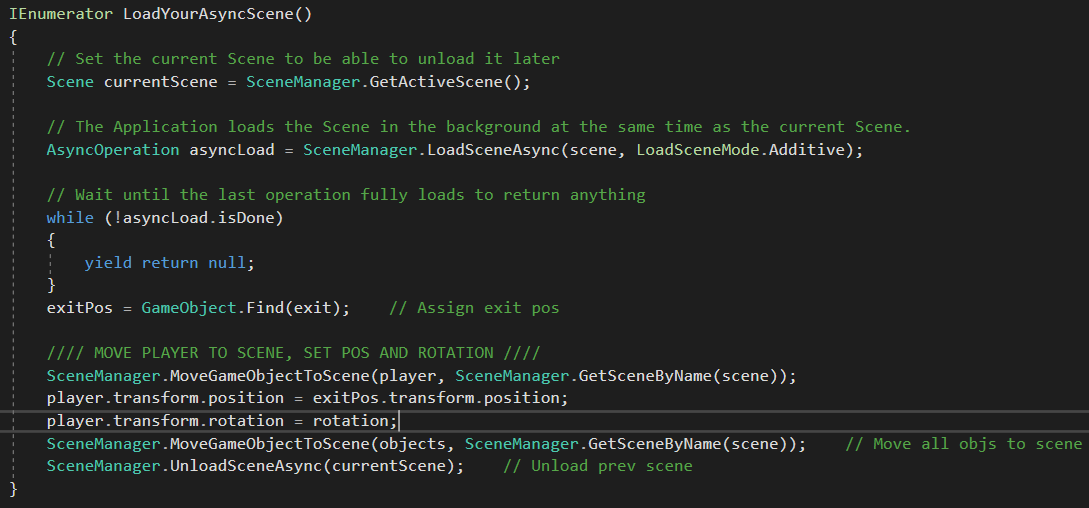
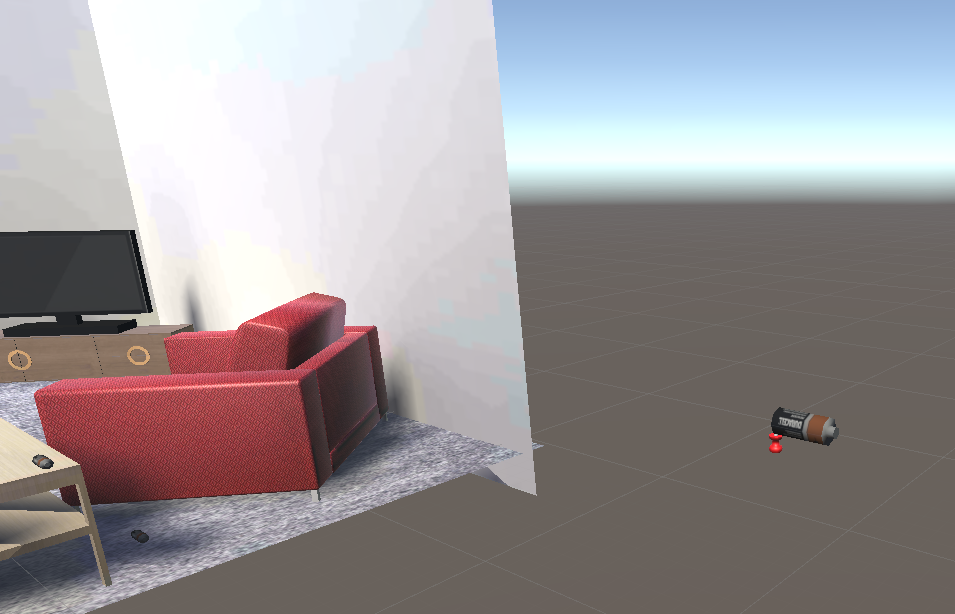


[6] Converts the input data from the joystick to a usable vector – translating it to be between -1 and 1 and normalising it, also moves the visual joystick which only serves as feedback to the player, has no bearing on actual movement

[8] (Left) Gets the forward and right vectors of the camera in Update(), the Y is reset to remove rotation and they are normalised

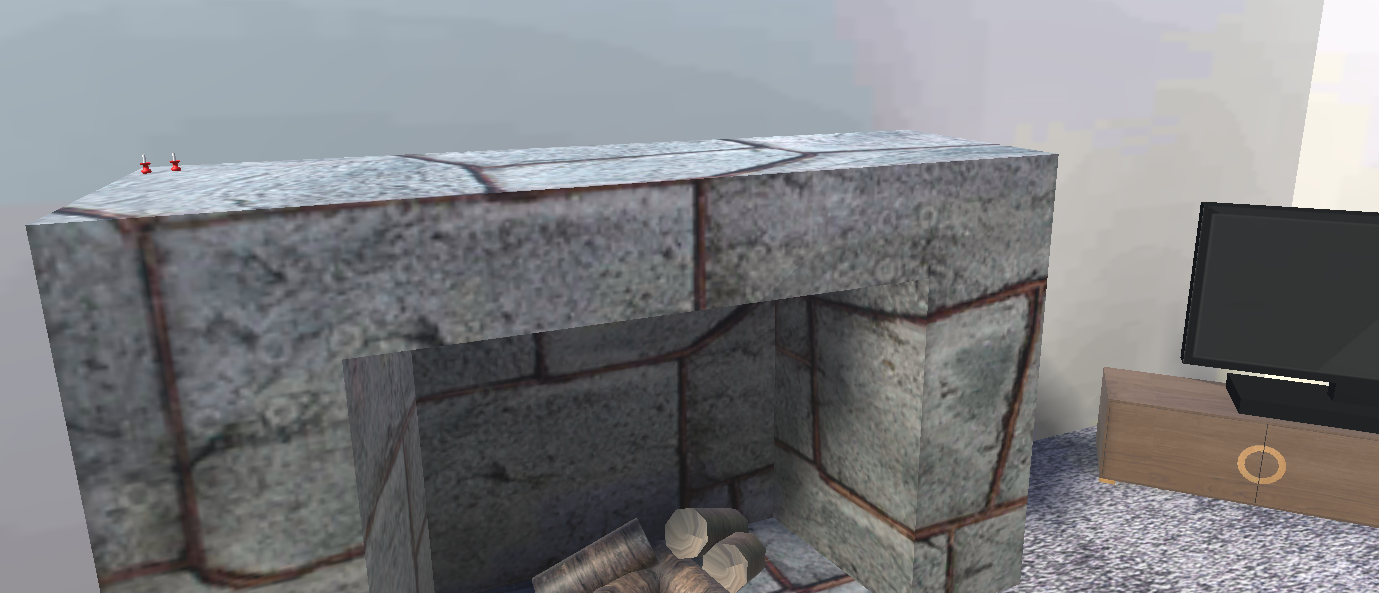
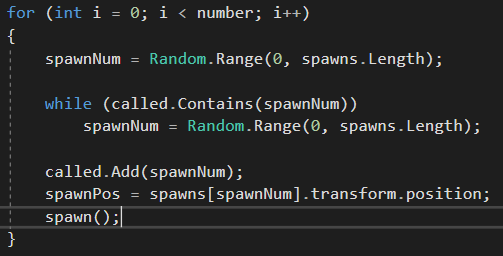
[9] (Below) Multiplies the vector from the joystick with that of the camera so force is now relative to player, moveSpeed variable used to increase force

[7] Translates the vector from the joystick to one that can be used for movement – the Y component from the joystick influences the Z movement



[11] Objects spawned off the side of the scene so they cannot be seen by player – they remain here until they are needed in the scene itself

[10] Loads the target scene in the background and waits till complete, moves the player to new scene and sets its position and rotation as well as a set of objects before unloading the previous scene

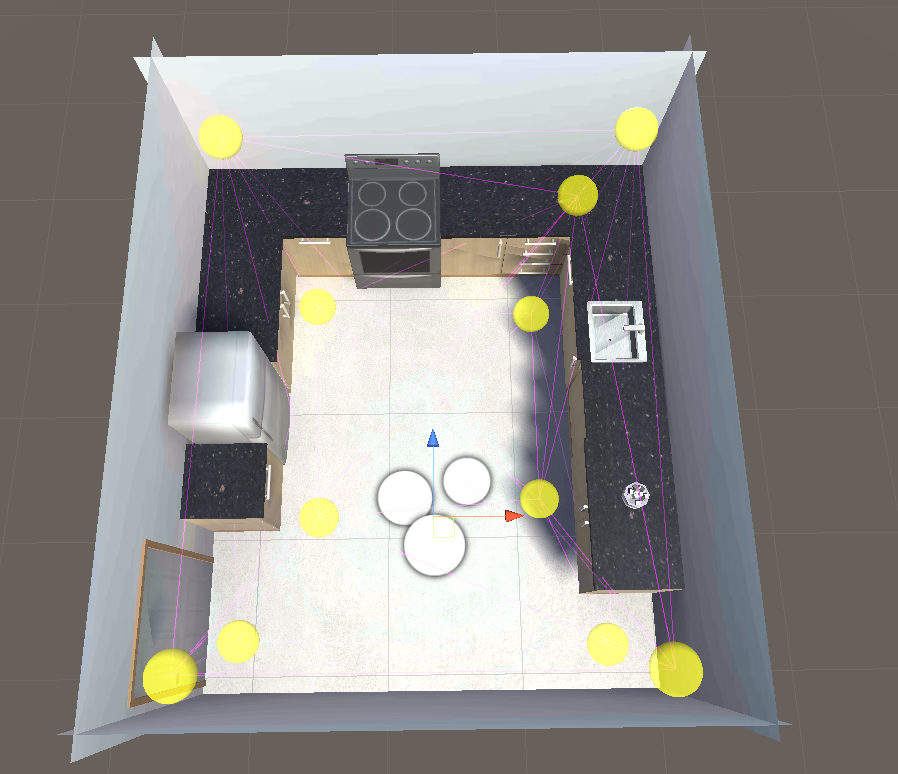


[12] (Top) One instance of scene

[13] (Bottom) Second instance of scene

Shows spawn points moving with each instance of the scene

[14] Code which decides which spawn points to use – random spawn point chosen and is compared against a list of already called points, while it matches calls a new random point, otherwise moves an object to the chosen point



[15] Light probes used in one of the scenes – allows shadows to be cast on moving objects