# Andrew Moneypenny -- Individual Report

The most important contribution I made to the Group Project was the development and implementation of the Hospital location. The Hospital worked very similarly to the other 5 locations in that the player could collect items, use items for certain events, move around and, of course, give the player the opportunity to stay and prepare for the oncoming night. However, although I didn't add any major features to the Hospital, I did try to add some lore in order to try and add more depth to the events surrounding the game through a newspaper article and some entries in a set of documents. The documents also did utilise a small system that would allow the player to move between entries on the same Text Node as well in order to make it much more efficient.

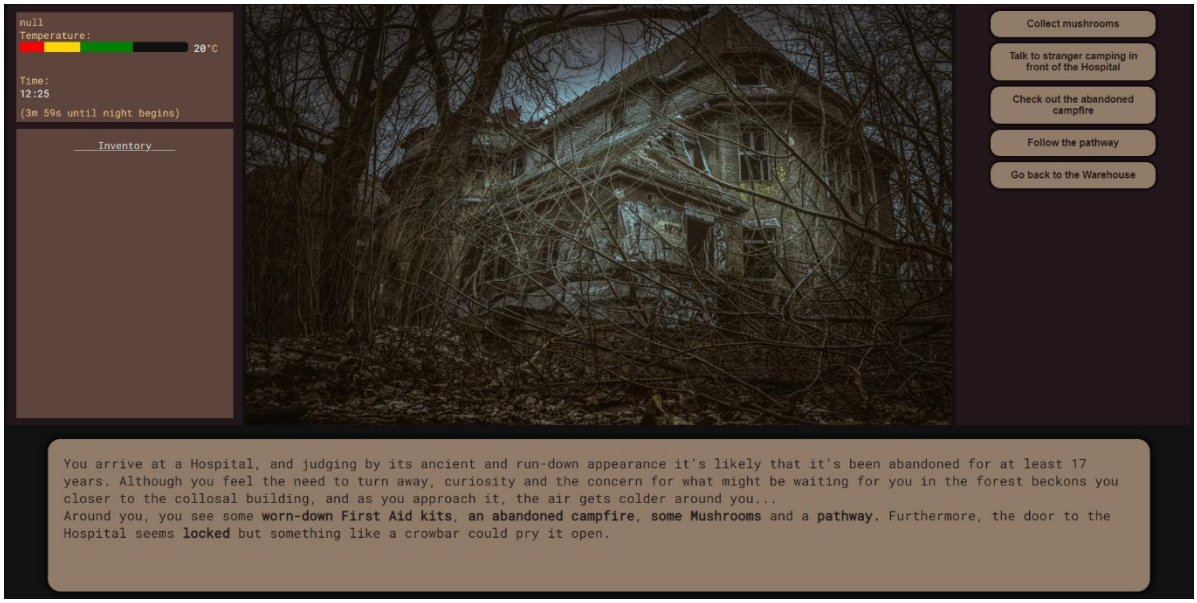


Figure 1: The Hospital Location upon visiting it for the first time

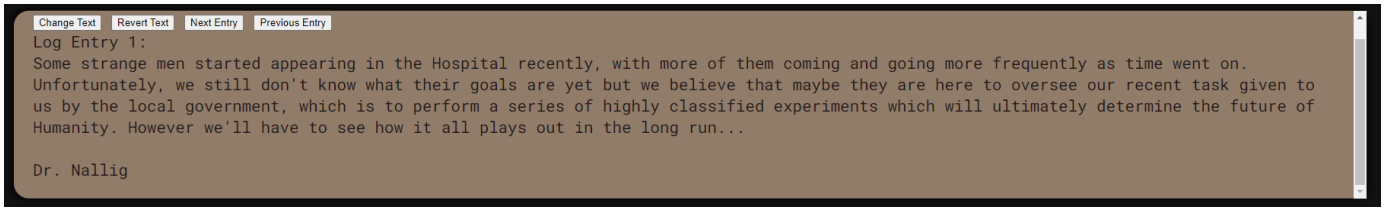


Figure 2: The Document's Text Entries

```
options: [
            {
                text: 'Collect mushrooms',
                requiredInventory: {Mushrooms: false, Crowbar: false},
                tempChange: 'decrease',
                nextText: 6
            },
            {
                text: 'Talk to stranger camping in front of the Hospital',
                tempChange: 'decrease',
                nextText: 5,
            },
            {
                text: 'Check out the First Aid kits scattered across the ground',
                requiredState: (currentState) => currentState.Doctor === true || currentState.WarVeteran === true,
                tempChange: 'decrease',
                nextText: 8
            },
            {
                text: 'Check out the abandoned campfire',
                tempChange: 'decrease',
                nextText: 10
            },
            {
                text: "Follow the pathway",
                tempChange: 'decrease',
                nextText: 4
            },
            {
                text: "Go back to the Warehouse",
                tempChange: "decrease",
                nextText: "Warehouse"
            }
        ]
    },
```

Figure 3: Code for the options on the Hospital's first Text Node

```
// Stores the index position for the array
var entry = 0;

// Allows the player to move onto the next Entry in the Hospital Documents
function NextText(){
    entry += 1
    /**
     * If the next entry to be displayed doesn't exist (is outside the array)
     * Then change the entry value back to the last entry in the entries array
     */
    if (entry >= entries.length) {
        entry = entries.length-1;
    }
    // Displays the appropriate text within the entries array
    document.getElementById("handwritten").innerHTML = entries[entry];
}

// Allows the player to move back to the previous Entry in the Hospital Documents
function PreviousText(){
    entry -= 1;
    /**
     * If the next entry to be displayed doesn't exist (is outside the array)
     * Then change the enry value back to the very first entry in the entries array
     */
    if (entry < 0) {
        entry = 0;
    }
    // Displays the appropriate text within the entries array
    document.getElementById("handwritten").innerHTML = entries[entry];
}
```

Figure 4: The code for the Text Entry System (which uses an Array)

Also, a few weeks into the project I tried to develop a prototype for a typewriter system using a version that I found online. The reason for implementing this was to add more depth to the game and allow the text to appear slowly over time rather than having blocks of text appear each time, because that could feel dull. However, the online version didn't suit our needs and was only very simple. Therefore, in order to have it better suit our needs, I altered it and added some extra functionality to it so that it would also work with HTML tags so that their effects would work properly, which was something that the basic version couldn't do. Unfortunately, this would've required a lot of extra methods, and so the typewriter system was later improved upon and better implemented by James.

```
+// Will display the textNode text by printing it in a typewriter liek fashion
+async function typeSentence(sentence, delay = 30) {
+
+    // Clears the HTML so that it doesn't keep adding on to it
+    document.getElementById('DialogueHospital').innerHTML = '';
+
+    // splits the sentence into individual characters
+    var letters = sentence.split("");
+
+    let i = 0;
+    while(i < letters.length) {
+
+        // the delay before each letter is printed
+        await waitForMs(delay);
+
+        // If the letter isn't an opening HTML tag character, this code runs
+        if (letters[i] != "<") {
+            document.getElementById('DialogueHospital').innerHTML += letters[i];
+            i++;
+        }
+        else {
+            var tag = letters[i];
+            i++;
+            while(letters[i] != ">"){
+                tag += letters[i];
+                i++;
+            }
+            tag += letters[i];
+            i++;
+            switch(tag){
+                case "<strong>": i=tagStrong(letters, i); break;
+                default: document.getElementById('DialogueHospital').innerHTML += tag;
+            }
+        }
+    }
+    return;
+}
```

```
+function tagStrong(letters, i){
+    var word = "";
+    while(letters[i] != "<"){
+        word += letters[i];
+        i++;
+    }
+    i += 9;
+    document.getElementById('DialogueHospital').innerHTML += "<strong>" + word;
+    return i;
+}
+
+
+
+
+// Creates a time delay before each letter is printed
+function waitForMs(ms) {
+    return new Promise(resolve => setTimeout(resolve, ms))
+}
```

Figures 5 & 6: Original code used for the prototype of the Typewriter System

Furthermore, I tried developing a simple Timer prototype system that would increment by 1 from 0 every second as soon as the game started. The main purpose of this timer was to allow the total time for a run of the game to be displayed after the player had either won or lost. However, in order to add more time-based events and functionality to the game, the Timer system was later built upon and implemented by James.


Figure 7: Original plan for the use of the Timer System

```
+var time = 0;
+var myTimer;
+
+function StartTimer() {
+    myTimer = setInterval(Timer, 1000);
+}
+
+function Timer() {
+    time++;
+    sessionStorage.setItem('Time', time);
+}
+
+function StopTimer() {
+    var times = sessionStorage.getItem('Time');
+    clearTimeout(myTimer);
+    document.getElementById('TheTime').innerHTML = times;
+}
```

Figure 8: Original code used for the prototype of the Timer

Additionally, we all agreed that the game needed to have some replay value, and so in order to make the player choose their decisions more carefully, I decided to try and develop a prototype for a temperature system. This would means that the player would start with a base temperature but every time the player selected a button/option, their temperature would decrease by a set amount (or increase by a set amount if they were by a fire for instance) and it took a while to get this to work with the Text Node system but eventually I did manage to get it working, but it was inefficient, so the Temperature system was also improved upon and better implemented later on by James.

```
+function decreaseTemp(value) {
+    var change = value;
+    if (change == "decrease") {
+        temperature -= tempDecrease;
+        if (temperature <= -21) {
+            sessionStorage.setItem("Temperature", temperature);
+            window.location.href = "EndStatistics.html";
+        }
     }
+    else if (change == "increase") {
+        temperature += tempIncrease;
+        if (temperature >= 80) {
+            sessionStorage.setItem("Temperature", temperature);
+            window.location.href = "EndStatistics.html";
+        }
+    }
```

Figure 9: Original code used for the prototype of the Temperature System


Figure 10: Original plan for the use of the Temperature System

However, I was able to develop and implement a fully functioning Cheat Code system that can alter things like the colour scheme of the game's main layout or affect in game systems, such as being able to disable the Typewriter system. The system wasn't too hard to develop as the text entered into the Cheat Code box is validated and if it matches the list of cheat codes then a change is made, otherwise nothing happens. Furthermore, the cheat codes entered do need to be case-specific as well.

```
// validates the entered Cheat Code
function validateCheat() {

    // Stores the Cheat Code entered
    var cheat = document.getElementById("Cheat").value;

    sessionStorage.setItem("ID", "Styling");
    sessionStorage.setItem("Attribute", "href");

    // switches the effects based on the Cheat Code's value
    switch(cheat){
        case "BlueScheme":
            sessionStorage.setItem("Value", "./assets/css/Cheat1.css");
            break;
        case "GreenScheme":
            sessionStorage.setItem("Value", "./assets/css/Cheat2.css");
            break;
        case "Monochrome":
            sessionStorage.setItem("Value", "./assets/css/Cheat3.css");
            break;
        case "betaDesign":
            sessionStorage.setItem("Value", "./assets/css/Cheat4.css");
            break;
        case "normalText":
            sessionStorage.setItem("Typewriter", true);
            break;
        default:
            sessionStorage.setItem("Value", "./assets/css/main.css");
    }
}
```

```
// Updates the HTML pages with the new Changes each time the page is opened
// Will be used in the body tag as an onLoad method
function updatePages() {
    // try-catch implemented to prevent unnecessary errors from showing up
    try {
        // Gets the ID of the Element being updated
        let ID = sessionStorage.getItem("ID");

        // Gets the attribute which will be changed
        let attribute = sessionStorage.getItem("Attribute");

        // Gets the value that will be replaced for the changed Attribute
        let value = sessionStorage.getItem("Value");

        // Carries out the change
        document.getElementById(ID).setAttribute(attribute, value);
    }
    catch {}
}
```

Figures 11 & 12: The code used to create the Cheat Code System

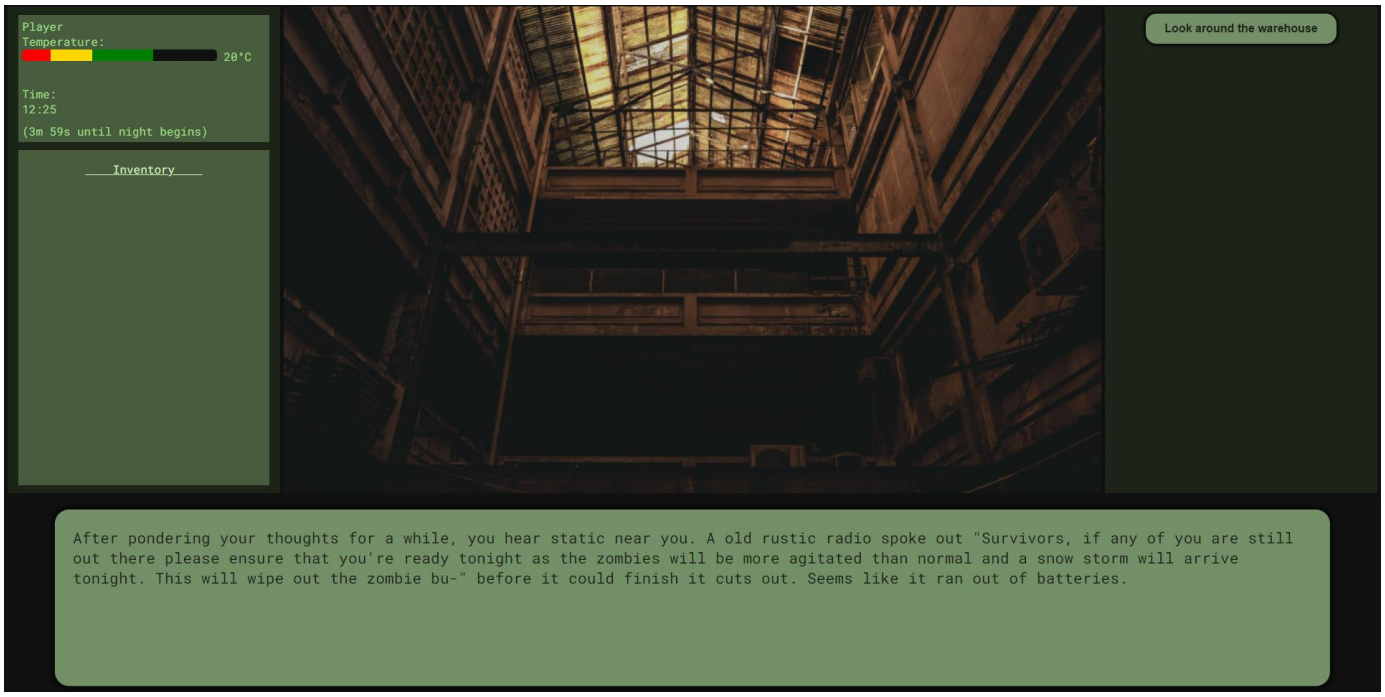


Figure 13: Example of the Cheat Code System when the code "GreenScheme" is used

Similarly, I decided to add some colour schemes in order to aid those who are colourblind. So, when the player loads up the game, they have the option in the form of radio buttons to change the colour scheme to a vibrant blue and orange colour scheme, a vibrant pink and yellow colour scheme or have the colour scheme be the main colour scheme (in case the player accidentally clicks on one of them/changes their mind)

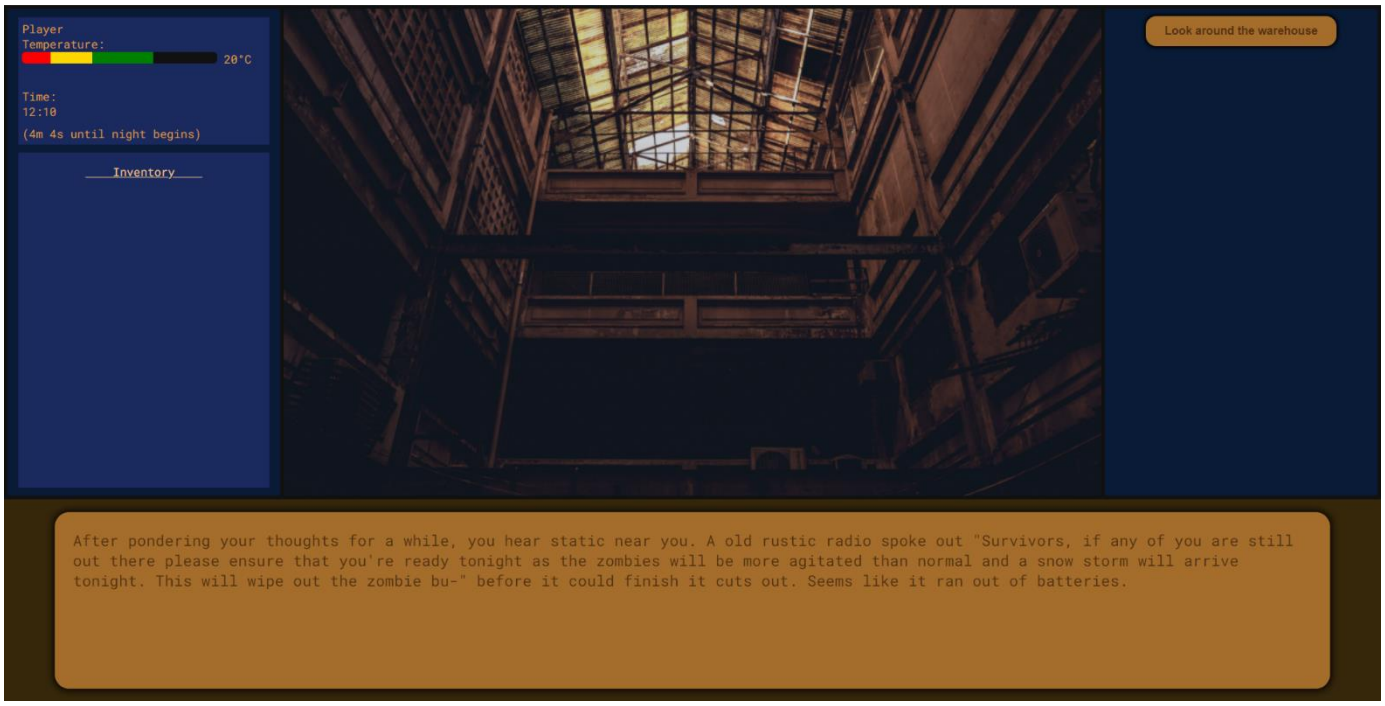


Figure 14: Example of one of the two Colourblind friendly colour schemes

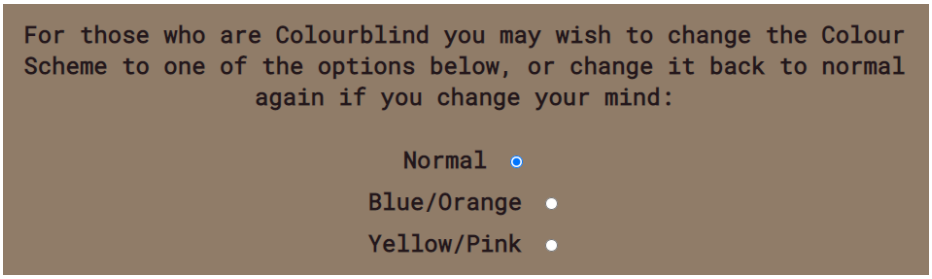


Figure 15: Player given the option to change the colour scheme to a colourblind friendly one

```
// validates the selected colour scheme
function validateOption() {

    // Stores a list of the radio buttons
    var optionList = document.getElementsByName('Scheme');
    // Loops through the list of radio buttons to check to see if one's checked.
    for (var i = 0; i < optionList.length; i++) {
        if (optionList[i].checked) {
            var option = optionList[i].value;
        }
    }

    sessionStorage.setItem("ID", "Styling");
    sessionStorage.setItem("Attribute", "href");

    // switches the effects based on the selected radio button's value
    switch(option){
        case "Blue":
            sessionStorage.setItem("Value", "./assets/css/ColourblindColourPalette.css"); break;
        case "Yellow":
            sessionStorage.setItem("Value", "./assets/css/ColourblindColourPalette2.css"); break;
    }
}
```

Figure 16: Code used to change the colour scheme to the appropriate colourblind friendly one

Lastly, although not major, I did make some changes to the Warehouse location after its initial structure and Text Nodes had been added, such as adding images, sounds and some endings to allow the player to stay and prepare for the night there as well as the other locations.