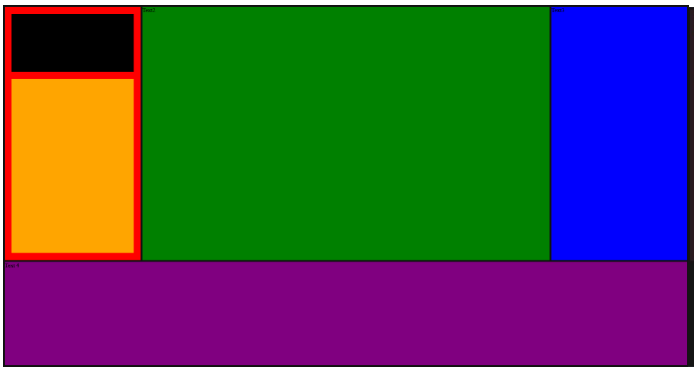
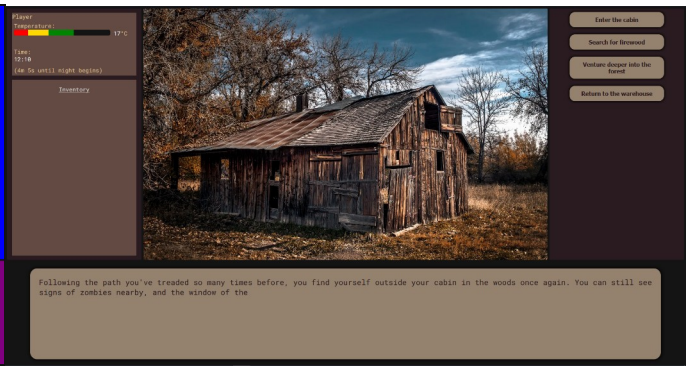
My contributions to the group project:
- Built the initial CSS layout, following that has since been contributed to by everybody else, with a map and inventory on the left, dialogue on the bottom and choices on the left, and iterated on this to improve it. The map was later on replaced with displaying the temperature and timer systems.

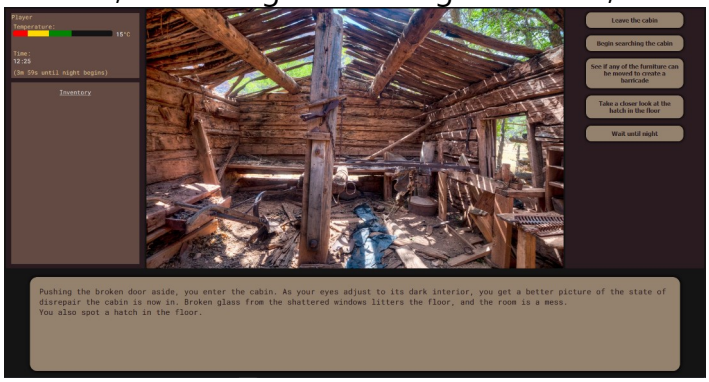


(Image showing initial testing of the layout CSS)

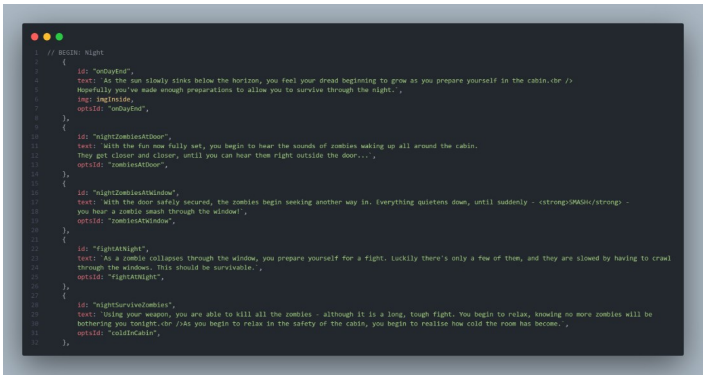


(Image of the final CSS, updated and properly themed)

- Created the cabin location, containing a wide range of events, items and endings



(A screenshot of part of the location)



```
// BEGIN: Night
    {
        id: "onDayEnd",
        text: `As the sun slowly sinks below the horizon, you feel your dread beginning to grow as you prepare yourself in the cabin.<br />
        Hopefully you've made enough preparations to allow you to survive through the night.`,
        img: imgInside,
        optsId: "onDayEnd",
    },
    {
        id: "nightZombiesAtDoor",
        text: `With the fun now fully set, you begin to hear the sounds of zombies waking up all around the cabin.
        They get closer and closer, until you can hear them right outside the door...`,
        optsId: "zombiesAtDoor",
    },
    {
        id: "nightZombiesAtWindow",
        text: `With the door safely secured, the zombies begin seeking another way in. Everything quietens down, until suddenly - <strong>SMASH</strong> -
        you hear a zombie smash through the window!`,
        optsId: "zombiesAtWindow",
    },
    {
        id: "fightAtNight",
        text: `As a zombie collapses through the window, you prepare yourself for a fight. Luckily there's only a few of them, and they are slowed by having to crawl
        through the windows. This should be survivable.`,
        optsId: "fightAtNight",
    },
    {
        id: "nightSurviveZombies",
        text: `Using your weapon, you are able to kill all the zombies - although it is a long, tough fight. You begin to relax, knowing no more zombies will be
        bothering you tonight.<br />As you begin to relax in the safety of the cabin, you begin to realise how cold the room has become.`,
        optsId: "coldInCabin",
    },
```

(An image of some of the data used to create the location – see cabin.js and cabinData.js for the full source)

- Took Andrew's prototype typewriter system and rewrote it to be significantly more robust (allowing any HTML element tags, not breaking when the user choices too quickly, etc.) while still remaining fully compatible with code that expected Andrew's version.

```
for (let i = 0; i < letters.length; i++) {
        // Wait before printing each letter
        // TODO: HTML tags are skipped over instantly (takes too long otherwise)
        if (!tag && delay) await waitForMs(delay);

        // If the user has gone on to the next event/location, stop displaying this one
        if (printCount != thisPrintCount && !isSlow) return;

        // If at the start of a tag
        if (letters[i] === "<") {
            tag += letters[i];
            isInTag = true;
        }
        // If in < these > and not also in " these ", then the next > marks the end of the tag
        else if (isInTag && !isInQuotes && letters[i] === ">") {
            tag += letters[i];
            hasReachedBackslash = true;
        }
        // Allow URLs etc containing `/`
        else if (isInTag && letters[i] === '"') {
            tag += letters[i];
            isInQuotes = !isInQuotes;
        }
        // If at the end of the tag
        else if (letters[i] === ">" && hasReachedBackslash) {
            tag += letters[i];

            // Add the tag (TODO: Currently added instantly)
            dialogueBox.innerHTML += tag;

            // Reset variables
            tag = "";
            isInTag = false;
            hasReachedBackslash = false;
        }
        // Any other `>` that is not the end of the tag
        else if (letters[i] === ">" && !hasReachedBackslash) {
            isInTag = false;
            tag += letters[i];
        }
        // Any other character while constructing the tag
        else if (tag) tag += letters[i];
        // Not constructing a tag
        else dialogueBox.innerHTML += letters[i];
    }
```

(A section of the typewriter's code – see Typewriter.js for the full source)

- Implemented a system for cross-fading audio, allowing a smooth transition between audio effects in different scenes, as I found the abrupt transition that we were initially using to be extremely jarring
- This audio system was also updated to support playing short, non-repeating sounds
- Later on, it was also updated to integrate with the volume slider Ryan had added

```
audioIn.volume = 0;
audioIn.src = newSource;
if (newSource) {
    audioIn.play();

    // If a new source is provided, fade into it on an interval
    let fadeIn = setInterval(function () {
        // Only fade if not at one already
        if (audioIn.volume < 0.999 * volumeMult - fadeInStep) {
            audioIn.volume += fadeInStep;
        } else {
            // Stop when volume is near-one
            clearInterval(fadeIn);
            audioIn.volume = volumeMult;
        }
    }, fadeInTime * fadeInStep);
}
```

(A section of the code that controls cross-fading audio – see crossfade.js for the full source)

- Created a temperature system, compatible with Andrew's initial prototype, that also allows each location to be more flexible with increasing and decreasing the temperature, made sure the temperature was saved to session storage so it would carry over between locations, allowing specific events to be called to end the game if the player gets too hot/cold, and displayed the temperature to the player both as a number and as a colour-coded bar that can be read easily at a glance

```
/**
 * Called when the temperature is changed so that the new value can be saved,
 * and the game ended if required.
 */
function tempUpdated(displayUpdate = true) {
    // Enforce absolute zero
    if (temperature < -273) temperature = -273;

    // Update the value shown to the user
    if (displayUpdate) meterSetup(temperature);
    try {
        temperatureSpan.innerHTML = temperature;
    } catch {}

    // Save the temperature to sessionStorage
    sessionStorage.setItem("Temperature", temperature);

    // End the game if the temperature becomes too high/too low
    if ((temperature <= minTemp || temperature >= maxTemp) && !isDead) {
        // Do whatever stuff to show the game is over
        isDead = true;
        if (temperature <= minTemp) showTextNodeFunction(tempTooLowId);
        else if (temperature >= maxTemp) showTextNodeFunction(tempTooHighId);
    } else isDead = false;
}
```

(A small sample of the code used to create the temperature system, see Temperature.js for the full source)

(The visual output of the temperature system)

- Improved Andrew's initial version of the timer by getting it to display the time remaining as part of the UI, and added functionality to run specific text nodes whenever the night started/ended, to make sure the time limit of 5 minutes was actually enforced

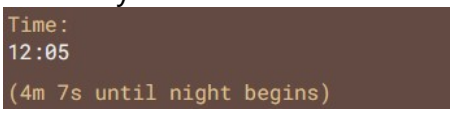
```
function StartTimer() {
    myTimer = setInterval(incrementTime, 1000);
    displayTime();
}

function incrementTime() {
    time++;
    timePlayed++;
    saveTime();

    displayTime();

    if (time >= dayLength && !hasNightStarted) {
        hasNightStarted = true;
        try {
            vaultClear();
        } catch {}
        showTextNodeFunctionTimer(onDayEndId);
    } else if (time >= dayLength + nightLength) {
        stopTimer();
        try {
            vaultClear();
        } catch {}
        showTextNodeFunctionTimer(onNightEndId);
    }
}
```

(A section of the timer's code, see Timer.js for the full source)

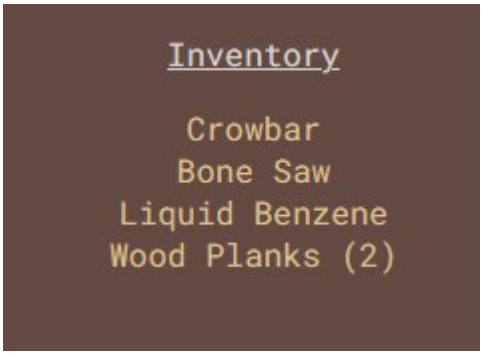


(The output of the timer system)

- Improved Ryan's inventory system by moving it to its own file to allow everyone to integrate it more easily (same as all the other systems), and allowed it to save the state of the inventory to session storage to allow it to be carried between locations, and finally allowed the player to pick up multiple items of the same type (e.g. have 2 sets of wood planks)

```
/**
 * Check if all the given requirements are met
 * @todo Any way of doing (A OR B)?
 * @param {JSON} reqs The requirements to check
 * @returns True if all requirements are met, false if not
 */
function meetsInventoryRequirements(reqs) {
    // Assume all requirements are met by default
    let allMet = true;
    // Iterate through the requirements
    for (let key in reqs) {
        // If all previous requirements were met, and
        // (the current requirement is explicitly met OR )
        allMet =
            allMet && // All previous conditions met
            (reqs[key] == inventory[key] || // Exact match
                (!reqs[key] && !inventory[key]) || // Or the value is supposed to be false and has not yet been defined in inventory
                (reqs[key] && inventory[key] > reqs[key])); // Or 1+ items are required and the player has all the required items
    }
    return allMet;
}
```

(Part of the inventory's code, see inventory.js for the full source)



(An example of the inventory's UI)