**Implementation Write up.**

Graphical user interface, text, application, email

Description automatically generated

The image above illustrates the HTML code that was prepared in preparation for the website's development. As shown in the code above, "link rel>" was utilised to link the HTML file to numerous CSS files, as well as to implement the custom font "Aboreto" so that it may be utilised across the website. Multiple divs were created inside the body section, which was modified and customised using CSS. Finally, "a href" and "button" was used, which makes a clickable button that links to a virtual reality portal located in the "portal.html" file.

Text

Description automatically generated

The image above displays the contents of the "portal.html" file. This contains a link to the "portal.css" file, which allows any class specified to be customised. Furthermore, "<script>" is utilised, allowing javascript code to be embedded in the HTML document. In this scenario, the javascript code supplied creates a timer using the "setTimeout()" function that waits for 2 seconds (2000 milliseconds) before redirecting the current website to the URL indicated in the window.location property, which is /VR/Art Gallery/art\_gallery.html in this case. The CSS for the HTML code above was not provided as it only consists of customisable properties and not the main functionalities for the project itself.

Text

Description automatically generated with medium confidence

The beginning of this code imports the aframe.js JavaScript library into an HTML document, enabling the A-Frame framework to be used on the website. This enables the creation of a virtual reality experience that can be utilised with VR headsets. Additionally, this code defines an a-scene element in A-Frame, which serves as a container for all 3D material and virtual reality experiences created. The cursor element provides the type of cursor used in the scene, while the rayOrigin attribute describes where the ray begins, which is a mouse in our case. The raycaster allows users to interact with only objects in the VR scene that contain the class name “raycastable”, which in our scenario would be the entire a-scene. An audio element and image assets were also created so that they could be used within the VR scene. Finally, the addEventListener() method was executed to play the fire video when the user clicks anywhere in the scene.

Graphical user interface, text

Description automatically generated

In the image above, a bench model was also implemented within the VR scene. This was done using the “gltf” file, which consists of the model and textures. Furthermore, multiple “a-mixin” elements were defined to be reusable and applied to any other a-frame entities using the id specified in the mixin.

A picture containing calendar

Description automatically generated

A picture containing text

Description automatically generatedThe image above consists of multiple a-frame entities that were used to construct the art gallery building. Each entity includes its own corresponding properties, for example the a-plane has a “repeat” property which allows for the texture specified to be repeated depending on the values given to it.

The entrance sign that was declared under “a-assets” was utilised inside the “a-box” entity, allowing for the image to cover the box in the VR scene. The doors have a "onclick" feature that allows the user to walk in when clicked.

Text

Description automatically generated

The image above highlights the javascript code associated with the "toggleDoor()" function. The door entity and current location of the door were obtained using "var" and either "querySelector" or "getAttribute" inside this method. In addition, the original and new positions of the door were established, and an if statement was utilised to determine if the present position of the door was in its original or new position. This means that when the user presses the button, the door will move to the new position specified, and when the button is pressed to shut, the door will return to its original position.

Text

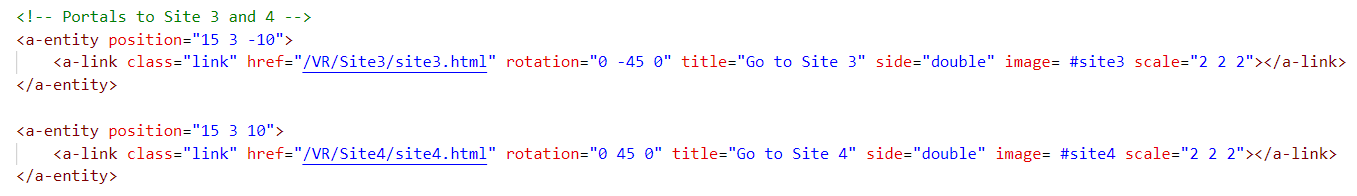
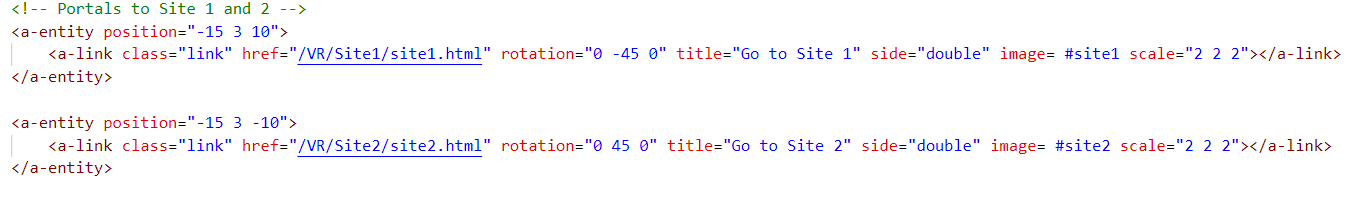
Description automatically generated

The image above is an a-entity which creates a point light source in the VR scene, which will radiate from a single point in all directions. Additional properties were also assigned to the light, such as intensity, distance, colour, and position, so the light could work well with the VR scene. Finally, the reusable properties in the mixin “abslight” was also used, meaning that only position and rotation were required to be assigned to the light entity.

A picture containing text

Description automatically generated

Within the “a-box” entity for the fireplace, “a-video” was indented inside and the “src” property was used to access the fire video. Additionally, an orange light was set just below the fireplace so that can represent a reflection of the fire in the VR scene. Decorations were added to the room such as benches, paintings and more.



The "a-link" element was used to establish a clickable link, such as "VR/Site1/site1.html." Moreover, each "a-link" was given an additional set of properties, such as an image that was assigned to each portal based on the site. This will generate a portal-like structure in the scene, enabling the user to click on it and proceed to the right site.

A picture containing graphical user interface

Description automatically generated

The image above shows the music asset using the “src” property. Two additional properties were set to the music, one of which was autoplay, which was set to true so that the music would play as soon as the user enters the VR scene. Moreover, the loop was also set to “true” so that once the music finishes, it will keep repeating. The “<a-camera>” element creates a camera entity with the look-controls property, which allows the user to move the camera. The wasd-controls property allows the camera to be moved with an acceleration of 80 by using the WASD keys. A green cursor is also created to be seen through the camera. Finally, two hand controllers were created in a 3D scene with laser beams that can interact with objects "raycaster" that have the "raycastable" class and a white line that indicates the direction of the laser.

As site 1 contained multiple videos, a new system was implemented to accommodate this, which included the separation of images and videos. This is demonstrated by the two a-scenes produced below, one of which displays the images and the other the videos:

A picture containing text

Description automatically generated

The objects that have been included in the a-scene are depicted in the image above. Here, each asset has been assigned to the "a-sky" element and its visibility has been set to either true or false. Additionally, the buttons are created using “a-plane” which include their corresponding properties as well as their own source texture.

Text

Description automatically generated with medium confidence

Text

Description automatically generatedThis creates a “home “ button in the VR scene which awaits for the user to click on this element, which in turn calls the listener function and redirects the user back to the art gallery scene so they can view the remaining sites. Similarly, to the home button the next code redirects the user to the videos in site 1 using the same functionality.

The image above highlights the functionalities for both the next and back button. Initially the variable “image Index” is set to 0, this will keeps track of which image is being accessed based on which index is being accessed. Then, "querySelectorAll" is used to select all "a-sky" elements. When the next button is clicked, the current image is concealed and the index is incremented by 1 to point to the next asset, which is then displayed.Once the final asset has been accessed, the next if statement resets the index to 0 so that the assets can be repeatedly cycled. The functionality of the back button remains the same; however, the "imageIndex" is decremented to display the preceding asset.

A picture containing text

Description automatically generated

The  image above depicts the declaration of the two videos contained within "a-videosphere." Each video has its own play and pause button, as well as a unique id, so that they can be identified  and used in the code below.

Text

Description automatically generated

The image above includes an event listener for each button. This assures that the value of the "a-text" variable changes based on whether the video is being played or paused and prevents text from overlapping. For instance, if the first video is playing, the button's value will change to "pause" and vice versa.

Text

Description automatically generated with low confidence

A variable named "isPlaying" was declared and initialised to false; it is then used to determine whether the video is currently playing or paused. In addition, four event listeners have been set up for each video's play and pause functions, which will set the variables "isPlaying" and "isPlaying1" to true or false depending on the video's state. The "repaint" value is set to 200 milliseconds so that when the button is clicked, the value changes instantaneously from "play" to "pause".

Finally, the “addEventListener()” for the playButton checks the “isPlaying”, “isPlaying1” variables after clicking the button. If the value is true, then the "pause()" method is invoked to pause the video and vice versa. Consequently, the user can control video playback using the play/pause button.

Text

Description automatically generated

The functionality of this code is identical to that described previously; the only difference is the addition of an event listener to the image button which when pressed it will redirect the user to “/VR/site1/site1.html” where they can access and view the images of that current site.

Text

Description automatically generated

This section controls the functionality of the videos in site 1. Through the "document.getElementById" command, the element for each videosphere is obtained. This code functions similarly to the one discussed previously, but loops through two videos. This is where when the “videoIndex” is 0 meaning the first video is accessed then videosphere visibility is set to true and videosphere1 is set to false. In addition, the "playButton" is concealed or displayed based on what video is being accessed. This is the same as below, Text, timeline

Description automatically generatedexcept videosphere1 is displayed and videosphere is concealed.

Similar to the code mentioned previously, this enables the user to access the preceding asset when they click the back button. If the “videoIndex” is less than 0 then it is set to 1 allowing for a continuous loop to be performed with the videos.

The following sites 2, 3 and 4 are vastly similar and therefore only one site was discussed below. Each site consists of their corresponding images and videos.

Text

Description automatically generated

The preceding image depicts the various a-frame entities utilised for site 2. This incorporates "a-sky" for the 360o images and "a-videosphere" for the 360o video. A camera with look controls was integrated so that the user could observe images and videos and wasd controls were not implemented to restrict the user’s movement. A white cursor is displayed, and the fuse property is set to false, requiring the user to click on the cursor to interact with an object. Additionally, the "fuseTiemout" property was set to "0," indicating no latency between the user clicking the cursor and the interaction being registered. This was essential because we determined that the user should receive immediate feedback when interacting with scene objects.

Text

Description automatically generated

The code specified above regulates the functionality of the play/pause button, enabling the video to be played or paused. The “addEventListener()” functionality makes sure that when the button is clicked, it checks its value property for "> Play". If so, it sets the text element's content to "II Pause" and the button's value property to "II Pause". If it doesn't mention "> Play," it updates the text and value property to "> Play".

The next code is similar to the previous code explained above as the functionalities are the same. However, the only difference is the functionalities is set to another a-scene which in this case site 2,3,4 will have the same code.

Text

Description automatically generated

Site 2, 3, 4 consists of the same code shown above, which redirects the user to the “art gallery” scene.

Text

Description automatically generated

The code above manages the "next" button functionality, which allows the user to see the next assets. To begin, the current asset was set to the first image stated previously. An event listener consisting of an if statement inside that function, was added to the "next" button. When the user presses the "next" button, the function is called which hides and checks if the current asset is a video. If this is the case, the video will be hidden by changing its visibility to true. Following this, the else statement checks whether the current asset is an image; if so, it hides the "a-sky". After this, the current asset index is increased by one, enabling the following asset to be displayed. Once all assets have been cycled through, it will reset to the first image and the next Text

Description automatically generatedasset can be shown. If the next asset is an image, the "a-sky" entity will be shown.

Similarly, to the code previously discussed, the “back” button functionality works the same where it allows the user to cycle through the previous assets whether it is an image or video.