Our screenshot:

Part 1:

CSIT128: Assignment 1			Group G1T16
Student Number / Name / Email	7026304	Andre	7026304@uow.edu.au
	7772427	Kenneth	7772427@uow.edu.au

Our code:

```
<!-- Part 1 -->
<div>
    <h1><u>Part 1:</u></h1>
</div>
<table border="1" cellpadding="10" cellspacing="2">
    <tr>
        <th colspan="3" bgcolor="lightblue"><b>CSIT128: Assignment 1</b></th>
        <th bgcolor="lightblue">Group G1T16</th>
    </tr>
    <tr>
        <td rowspan="2">Student Number /<br> Name / Email</td>
        <td>7026304</td>
        <td>Andre</td>
        <td>7026304@uow.edu.au</td>
    </tr>
    <tr>
        <td>7772427</td>
        <td>Kenneth</td>
        <td>7772427@uow.edu.au</td>
    </tr>
</table>
```

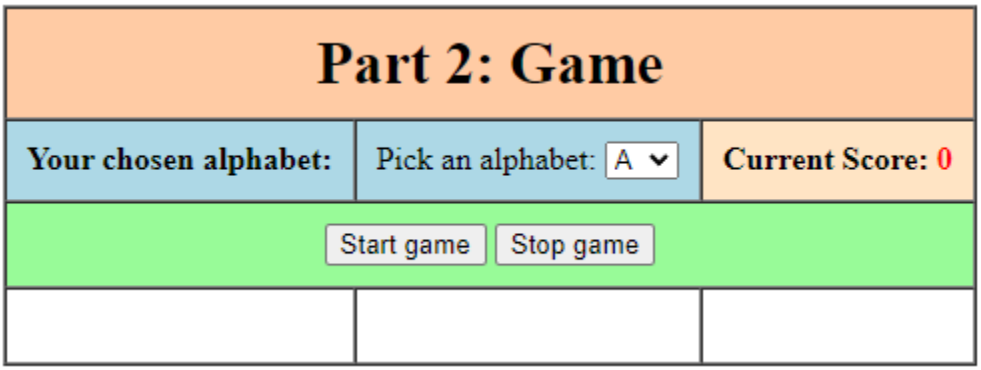
Explanation:

The background colour of light blue for the table header is set using bgcolor=”lightblue”. The table is styled with borders and padding. As seen, the table header occupies 3 columns of space, and the table data cell occupies 2 rows of space.

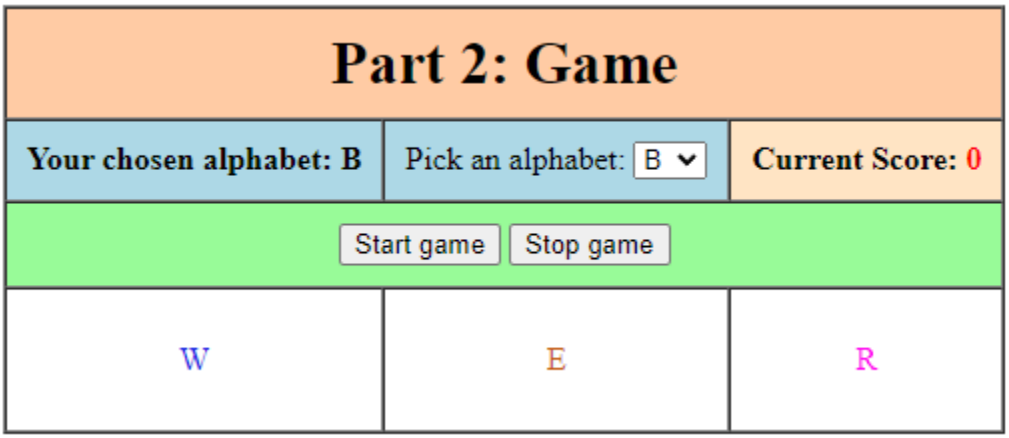
Part 2:

Our screenshot:

Game Default:



Game Start:



Wrong letter selected: (0 to -2)

Correct letter selected: (-2 to 2)

Our code:

```
<script>
    var score = 0; // set default score to 0
    var timedelay;
    var gameStart = false; // set default game start to false
    var chosenLetter = ''; // store chosen letter once game start

    // Return alphabet selected and shown
    function getSelected() {
        if (!gameStart) {
            chosenLetter = document.getElementById("alphabetListBox").value;
            document.getElementById("LetterShown").textContent = chosenLetter;
        }
    }

    // Start the game
    function start(action) {
        var startButton = document.getElementById("start");
        var alphabetListbox = document.getElementById("alphabetListBox");

        if (action == 0) {
            if (!gameStart) {
                resetGame();
                chosenLetter = document.getElementById("alphabetListBox").value; // Lock in chosen alphabet
                document.getElementById("LetterShown").textContent = chosenLetter;
                gameStart = true;
                alphabetListbox.disabled = true; // Disable changing the alphabet
                startButton.disabled = true; // Disable the start button
                timedelay = setInterval(generateRandomLetters, 1000); // 1s delay
            }
        }
        else if (action == 1) {
            gameStart = false;
            alphabetListbox.disabled = false; // Enable changing the alphabet
            startButton.disabled = false; // Enable the start button
            clearInterval(timedelay);
        }
    }
```

```
// Reset the game
function resetGame() {
    score = 0;
    document.getElementById("current").textContent = 0;
}

// Generate random alphabets
function generateRandomLetters() {
    const alphabets = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    var alpha1, alpha2, alpha3;

    do {
        alpha1 = alphabets.charAt(Math.floor(Math.random() * alphabets.length));
        alpha2 = alphabets.charAt(Math.floor(Math.random() * alphabets.length));
        alpha3 = alphabets.charAt(Math.floor(Math.random() * alphabets.length));
    } while (alpha1 == alpha2 || alpha1 == alpha3 || alpha2 == alpha3);

    document.getElementById("firstAlp").textContent = alpha1;
    document.getElementById("secondAlp").textContent = alpha2;
    document.getElementById("thirdAlp").textContent = alpha3;
}
```

```
    function handleAlphabetClicked(alphabetClick) {
        if (gameStart) {
            var temp1;
            if (alphabetClick == 1) {
                temp1 = document.getElementById("firstAlp").textContent;
            } else if (alphabetClick == 2) {
                temp1 = document.getElementById("secondAlp").textContent;
            } else if (alphabetClick == 3) {
                temp1 = document.getElementById("thirdAlp").textContent;
            }

            if (temp1 === chosenLetter) {
                score += 4;
            } else {
                score -= 2;
            }

            document.getElementById("current").textContent = score;
            generateRandomLetters();
        }
    }
</script>
```

```
<!-- Part 2 -->
<table border="1" cellpadding="10" cellspacing="0">
    <tr bgcolor="#ffcba4" style="font-size: 30px;">
        <th colspan="3">Part 2: Game</th>
    </tr>
    <tr bgcolor="lightblue">
        <td><b>Your chosen alphabet: <span id="LetterShown"></span></b></td>
        <td style="text-align: center">
            Pick an alphabet:
            <select id="alphabetListBox" size="1" onchange="getSelected()">
                <option value="A">A</option>
                <option value="B">B</option>
                <option value="C">C</option>
                <option value="D">D</option>
                <option value="E">E</option>
                <option value="F">F</option>
                <option value="G">G</option>
                <option value="H">H</option>
                <option value="I">I</option>
                <option value="J">J</option>
                <option value="K">K</option>
                <option value="L">L</option>
                <option value="M">M</option>
                <option value="N">N</option>
                <option value="O">O</option>
                <option value="P">P</option>
                <option value="Q">Q</option>
                <option value="R">R</option>
                <option value="S">S</option>
                <option value="T">T</option>
                <option value="U">U</option>
                <option value="V">V</option>
                <option value="W">W</option>
                <option value="X">X</option>
                <option value="Y">Y</option>
                <option value="Z">Z</option>
            </select>
        </td>
        <td bgcolor="#FFE4C4" style="text-align: center; font-weight: bold;">
            <b>Current Score: <span id="current" style="color: red">0</span></b>
        </td>
```

```
        </tr>
        <tr>
            <td colspan="3" bgcolor="#98FB98" align="center">
                <button id="start" onclick="start(0)">Start game</button>
                <button onclick="start(1)">Stop game</button>
            </td>
        </tr>
        <tr style="text-align: center">
            <td onclick="handleAlphabetClicked(1)" class="clickable">
                <p><span id="firstAlp" style="color: #1829E3"></span></p>
            </td>
            <td onclick="handleAlphabetClicked(2)" class="clickable">
                <p><span id="secondAlp" style="color: #c45911"></span></p>
            </td>
            <td onclick="handleAlphabetClicked(3)" class="clickable">
                <p><span id="thirdAlp" style="color: #ff00f0"></span></p>
            </td>
        </tr>
    </table><br/>
```

Explanation:

The game interface is organised into a table format consisting of several rows that display the chosen letter, game controls (start/stop buttons), the current score, and clickable letters for gameplay. All of which is defined using CSS properties such as table-border, cellpadding, cellspacing, and bgcolor.

A dropdown menu (<select> tag) allows the user to choose an alphabet letter from A to Z. This selection triggers a display update for the chosen letter through an onchange event linked to the JavaScript function getSelected(). The current score is styled using CSS properties is also set and displayed at “0”.

Once the user selects an alphabet, getSelected() is triggered, where this function updates the display with the currently selected letter from the dropdown menu. The “onchange” trigger is called whenever the player changes their selection in the dropdown. The function retrieves the selected letter from the dropdown menu, updates the variable, and displays this letter.

When “Start game” is clicked, the function start(action), sets the action to equals 0. Both the dropdown menu for selecting an alphabet and start button are disabled upon starting the game to prevent changes during the gameplay. This sets ‘gameStart’ as true, the chosen alphabet is locked. A ‘setInterval’ will call generateRandomLetters() every second, continuously generating and displaying new alphabets. When the game is stopped, the action equals 1, which makes the

“gameStart” set to false, and the dropdown and start button are re-enabled, allowing for new selections. The interval in “setInterval” is cleared, stopping the generation of new alphabets.

The function generateRandomLetters() contains a string of the alphabets and selects three alphabets from the string at random. The alphabets are then displayed in “firstAlp, "secondAlp," and “thirdAlp” in separate table cells.

The function handleAlphabetClicked(alphabetClick) manages the score of the game. The parameter of alphabetClick indicates which of the three displayed alphabets was clicked by the user. The function compares the parameter with chosenLetter. If the parameter matches, the current score adds 4 points, otherwise it will deduct 2 points.

When “Stop game” is clicked, the “start(1)” function is called. gameStart is set to false, enables the dropdown menu and the "Start game" button by setting disabled to false, allow the user to choose a new alphabet and start a new game. It stops the generation of alphabets by clearing the interval timer in “timeInterval”.

The function resetGame() sets the score to zero and updates the score display on the webpage to reflect this reset. This ensures that each new game starts with a clean slate. The function is triggered when “Stop game” is clicked, a new alphabet is selected and “Start game” is clicked again.

In overall, creating the tables and displaying the selection in HTML was straight forward. However, creating a javascript to write the logic for the game to run according to the A2 specification was a challenge. There were many things that we had to understand before we proceeded to write the codes and also, we had to breakdown each of the logic part by part such that the game could run as intended. There was alot of debugging involved as certain parts of the javascript was working but not to what we had expected.

Part 3

Our screenshot:

In .html:

```
179     <!-- Part 3 -->
180     <iframe src="/A2.xml" width="850" height="1000"></iframe>
```

In the .html file, line 180 was added with <iframe src="/A2.xml" width="850" height="1000"></iframe> to call the .xml file and also to create a frame with a width of 850 and height of 1000 to neatly place the table within.

In .xml:

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <?xml-stylesheet  type="text/xsl" href="S2.xsl"?>
3   <forecast queryTime="28/04/2024 10:45 AM" queryLocation="Singapore">
```

In the .xml file, line 2 <?xml-stylesheet type="text/xsl" href="S2.xsl"?> was added in order to specify the stylesheet to be used for transforming the xml document. This stylesheet is referenced to the file name “S2.xsl” accordingly by using the href=”S2.xsl”.

In .xsl:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="html" indent="yes" version="1.0" encoding="UTF-8"/>
    <xsl:template match="/forecast">
        <html>
            <body>
                <h1>Part 3</h1>
                <h2>
                    <!-- Display queryLocation and queryTime here -->
                    <xsl:value-of select="@queryLocation" />
                    [<xsl:value-of select="@queryTime" />]
                </h2>
                <table border="1" style="text-align:center">
                    <tr bgcolor="green" >
                        <th width="60px" style="color: black;"><b>Date</b></th>
                        <th style="color: black;"><b>Mon</b></th>
                        <th style="color: black;"><b>Tue</b></th>
                        <th style="color: black;"><b>Wed</b></th>
                        <th style="color: black;"><b>Thu</b></th>
                        <th style="color: black;"><b>Fri</b></th>
                        <th style="color: black;"><b>Sat</b></th>
                        <th style="color: black;"><b>Sun</b></th>
                    </tr>

                    <xsl:for-each select="//weather">
                        <!-- sort by month, then date -->
                        <xsl:sort select="month" data-type="number"
                            order="descending" />
                        <xsl:sort select="day" data-type="number"
                            order="descending" />

                        <tr>
                            <td bgcolor="green" style="color: black;">
                                <xsl:value-of select="format-number(date, '00')" />
                                <xsl:text> </xsl:text>
                                <xsl:choose>
                                    <xsl:when test="number(month) = 1">Jan</xsl:when>
                                    <xsl:when test="number(month) = 2">Feb</xsl:when>
                                    <xsl:when test="number(month) = 3">Mar</xsl:when>
                                    <xsl:when test="number(month) = 4">Apr</xsl:when>
                                    <xsl:when test="number(month) = 5">May</xsl:when>
                                    <xsl:when test="number(month) = 6">Jun</xsl:when>
                                    <xsl:when test="number(month) = 7">Jul</xsl:when>
                                    <xsl:when test="number(month) = 8">Aug</xsl:when>
                                    <xsl:when test="number(month) = 9">Sep</xsl:when>
                                    <xsl:when test="number(month) = 10">Oct</xsl:when>
```

```
                                    <xsl:otherwise>Dec</xsl:otherwise>
                                </xsl:choose>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Mon'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Tue'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Wed'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Thu'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Fri'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Sat'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                            <td width="100px">
                                <xsl:if test="dayOfWeek='Sun'">
                                    <xsl:call-template name="datediv" />
                                </xsl:if>
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
```

```
94    <xsl:template name="datediv">
95      <div>
96        <xsl:variable name="highest_temp" select="highest" />
97        <xsl:variable name="lowest_temp" select="lowest" />
98        <xsl:for-each select=".">
99          <xsl:sort select="number($highest_temp)" data-type="number" order="descending" />
100         <xsl:value-of select="$lowest_temp"/>°-<xsl:value-of select="$highest_temp"/>°
101       </xsl:for-each>
102     </div>
103
104     <div>
105     <img>
106       <xsl:attribute name="src">
107         <xsl:value-of select="overallCode" />
108         <xsl:text>.png</xsl:text>
109       </xsl:attribute>
110     </img>
111     </div>
112
113     <div>
114       <xsl:choose>
115         <xsl:when test="overallCode = 'cloudy' or (overallCode = 'rain')" >
116           <span style="color: blue"><xsl:value-of select="overall" /></span>
117         </xsl:when>
118         <xsl:when test="(overallCode = 'thunderstorm')">
119           <span style="color: green"><xsl:value-of select="overall" /></span>
120         </xsl:when>
121         <xsl:when test="(overallCode = 'sunny')">
122           <span style="color: red"><xsl:value-of select="overall" /></span>
123         </xsl:when>
124         <xsl:when test="(overallCode = 'partlySunny')">
125           <span style="color: orange"><xsl:value-of select="overall" /></span>
126           </xsl:when>
127       </xsl:choose>
128     </div>
129   </xsl:template>
130 </xsl:stylesheet>
```

In line 11 and 12, <xsl:value-of select="@queryLocation" /> and [<xsl:value-of select="@queryTime" />] was added such that it retrieves and would display the query location and timewithin '<h2>'.

In the table, the specifications for A2 assignment was to set the the background colour for the first row and column as green. Therefore, <tr bgcolor="green" > was added in line 15 and <td bgcolor="green" style="color: black;"> for line 34. These sets of lines would set the colour of the background for the required row and column in the table to be green and the other rows and columns would not be affected by the colour change .

In order for the forecast to display in descending order from latest to earliest date and temperature to be from highest to lowest, the settings were added from line 96 to 100.

The images from the A2 resources folder were utilized by adding code from line 105 to 110.

Initially, there was confusion regarding whether to use "overall" or "overallCode" as it was confusing whether one or the other was the correct word to use. However, through a bit of trial and error, we managed to resolve the issue and successfully call the images.

The colours for each weather were selected according to the A2 specifications by implementing codes from line 114 to 127.

In .xsd:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/A2"
    xmlns:tns="http://www.example.org/A2" elementFormDefault="unqualified">
    <element name="forecast" type="tns:forecast"/>

    <complexType name="forecast">
        <sequence>
            <element name="weather" type="tns:weather" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="queryTime" type="string" use="required"/>
        <attribute name="queryLocation" type="string" use="required"/>
    </complexType>

    <complexType name="weather">
        <sequence>
            <element name="year" type="tns:year"/>
            <element name="month" type="tns:month"/>
            <element name="date" type="tns:date"/>
            <element name="dayOfWeek" type="tns:dayOfWeek"/>
            <element name="overall" type="string"/>
            <element name="overallCode" type="tns:overallCode"/>
            <element name="highest" type="tns:temp"/>
            <element name="lowest" type="tns:temp"/>
        </sequence>
        <attribute name="yyyymmdd" type="tns:ymd"/>
    </complexType>

    <simpleType name="year">
        <restriction base="int">
            <pattern value="[12][0-9]{3}"/>
        </restriction>
    </simpleType>

    <simpleType name="month">
        <restriction base="int">
            <minInclusive value="1"/>
            <maxInclusive value="12"/>
        </restriction>
    </simpleType>

    <simpleType name="date">
        <restriction base="int">
            <pattern value="[0-3]?[0-9]"/>
        </restriction>
    </simpleType>

    <simpleType name="dayOfWeek">
        <restriction base="string">
```

```
48              <enumeration value="Mon"/>
49              <enumeration value="Tue"/>
50              <enumeration value="Wed"/>
51              <enumeration value="Thu"/>
52              <enumeration value="Fri"/>
53              <enumeration value="Sat"/>
54              <enumeration value="Sun"/>
55          </restriction>
56      </simpleType>
57
58      <simpleType name="overallCode">
59          <restriction base="string">
60              <enumeration value="cloudy"/>
61              <enumeration value="partlySunny"/>
62              <enumeration value="rain"/>
63              <enumeration value="sunny"/>
64              <enumeration value="thunderstorm"/>
65          </restriction>
66      </simpleType>
67
68      <simpleType name="temp">
69          <restriction base="int">
70              <minInclusive value="-100"/>
71              <maxInclusive value="70"/>
72          </restriction>
73      </simpleType>
74
75      <simpleType name="ymd">
76          <restriction base="int">
77              <pattern value="[12][0-9]{3}[01][0-9][0-3][0-9]"/>
78          </restriction>
79      </simpleType>
80  </schema>
```

Explanation:

In line 3 element declaration was added by using <element name="forecast" type="tns:forecast"/>. This declares an element which is called “forecast” with the type “forecast” and these are defined in the target namespace.

There are a couple of complex type element. The ‘forecast’ describes the struce of forecast element whereas ‘weather’ describes the structure of a weather element. Both of these complex types includes a sequence of elements and attributes which defines the structure of forecast and weather data.

‘Year’, ‘month’, ‘date’ and ‘dayOfWeek’ element is of simple type. These elements represents their sets of date information. Each of these simple types definitions that are in the schema provides constraints and restrictions on the structure and content of the XML data which follows this schema. Therefore, this ensures consistency and validity.