Criterion C: Development

Table of Contents

1. Array

```
1035    //Creating array for planets using the data in the code above
1036    arr_p = [];
```

```
1065        arr_p.push(p);
```

The above images are snippets from the program. There is an array_p = [] array, which is for storing the planets. Every time planet is created, It is added to these arrays.

The use of array benefits the program as the data can be reused in different parts of the program, which saves time.

2. Simple Selection

```
1492    let val = this.value;
1493    //timer 10мс s     m    h   d    week  month  70d    1/2year  1y      2y      5y      10y
1494    let arr_range = [0.01, 0.6, 36, 864, 6048, 25920, 60480, 157248, 315360, 630720, 1576800, 3153600];
1495    document.querySelector('.step').value = arr_range[val];
1496    //"not accurate" Message
1497    if(val >= 7){
1498      no_right = 1;
1499    } else {
1500      no_right = 0;
1501    }
1502    time_compare();
1503  });
```

This program contains multiple if, else statements throughout the program. One of the examples is shown above. The above if, else statement is used to show or hide the warning message: “Low Accuracy”. If the user slides the slider over the 7th array content, In this case its over 60480, then the program will display the “Low accuracy” message. Else, the program will not display the message.

3. Complex Selection

```
1032        //If the button says start
1033        if(start.innerText == 'start' || (start.innerText == 'start' && no_timer == 1)) {
1034          ctx.clearRect(0, 0, start.x*2, start.y*2);
1035          //Creating array for planets using the data in the code above
1036          arr_p = [];
1037          //Adding sun
1038          let s = {};
```

These images illustrate the application of complex selection (nested if) in my program. The above image is outer if, where if the condition of: button of start or stop the simulation says “start” is met, then the program will execute the action and stop the timer.

```
1093          if(no_timer != 1) {
1094            //starting timer
1095            timer = setInterval(timerFunc, step_t);
1096            //changing the "start" to "stop"
1097            start.innerText = 'stop';
1098            //Changing the button color
1099            start.parentElement.parentElement.querySelector('.line_anim').classList.remove('line_green');
1100            start.parentElement.parentElement.querySelector('.line_anim').classList.add('line_red');
1101          }
1102        } else if(start.innerText == 'stop' && no_timer != 1){
1103          //Stopping timer
1104          clearTimeout(timer);
1105          //Changing the "stop" to "start"
1106          start.innerText = 'start';
1107          //Changing the color of the button
1108          start.parentElement.parentElement.querySelector('.line_anim').classList.remove('line_red');
1109          start.parentElement.parentElement.querySelector('.line_anim').classList.add('line_green');
1110        }
1111      }
```

Within the if statement in the first image, there is another if statement, where if the condition of: no_timer != 1 (which means the timer is running, planets are in action), the statement is executed and is used to change the text in the button from “start” to “stop”.

4. Loops

```
967     for(let i = mapping.length - 1; i >= 0 ; i--){
968       if(r >= mapping[i].r){
969         r_arc = mapping[i].p;
970         break;
971       }
```

The above image demonstrates the use of loops in my program. This is part of program used for the zooming in and out functionality. The for loop starts a counter at mapping.length - 1, then it iterates until it is 0. This section is part of the magnifying functionality of the program, provides the user with the ability to scroll in and out of the solar system and magnify the scale of the simulation.

5. Nested Loops

```
861         if(f == 0) {
862           document.querySelectorAll('.format').forEach(function(elem, i){
863               let val = Number(elem.value.replace( /\s/g, ""));
864               if(elem.classList.contains('au')){
865                   val = elem.dataset.au;
866                   elem.classList.remove('au');
867               };
868               //trasnlating into normal format
869               let temp = val;
870               let n = 0;
871               let ost= 0;
872               let result = '';
873               //counting the decimal places
874               for(var i = 0; temp > 1; i++) {
875                 temp /= 10;
876               }
877               temp = val;
878               let zi = i - z;
879
880               while(temp >= 1){
881                 ost = temp % 10;
882                 temp = Math.trunc((temp / 10));
883                 result = (n >= zi ? ost : 0) + result;
884                 if(n % 3 == 2){
885                   result = ' ' + result;
886                 }
887                 n++;
888               }
889
890               elem.value = result.trim();
891           });
```

Both images demonstrate the use of nested loops (for and while) in the program. The first screenshot demonstrates a while loop inside of a for loop, this allows the program to do execute 2 conditions within 1 block of code.

```
999         //Vector, speed and x, y calculations
1000        for(let i = 0; i < arr_p.length; i++){
1001          for(let j = 0; j < arr_p.length; j++){
1002            if(i == j) {
1003              continue;
1004            }
1005            r.x = arr_p[j].x - arr_p[i].x;//x projectile
1006            r.y = arr_p[j].y - arr_p[i].y;
1007            r.md = Math.round(Math.sqrt(r.x*r.x + r.y*r.y));
1008            //In a situation where the distance between the bodies is 0
1009            if(r.md == 0){
1010              r.md = 0.0000000001;
1011            }
1012            let G = 0.0000000000667; //G constant H*m^2/kg^2
1013            //let backG = 14992503748;
1014            //Devivding by 1000000000 for transfering into km/s^2
1015            let a = G*Number(arr_p[j].m)/(r.md*r.md);
1016            arr_p[i].ax = a * r.x / (r.md * 1000000000);
1017            arr_p[i].ay = a * r.y / (r.md * 1000000000);
1018            arr_p[i].vx = arr_p[i].vx + arr_p[i].ax * dt;
1019            arr_p[i].vy = arr_p[i].vy + arr_p[i].ay * dt;
1020          }
1021        }
1022        //Mapping the planets (Location)
1023        for(let i = 0; i < arr_p.length; i++){
1024          arr_p[i].x = arr_p[i].x + arr_p[i].vx * dt;
1025          arr_p[i].y = arr_p[i].y + arr_p[i].vy * dt;
1026        }
1027        draw();
1028      }
```

The second image has 2 nested for loops.

6. OOP

```
1053          let p = {};
1054          p.name = elem.querySelector('.planet_name').value;
1055          p.m = Number(elem.querySelector('.planet_mass').value.replace( /\s/g, ""));
1056          p.r = Number(!(elem.querySelector('.planet_rad').classList.contains('au'))
1057          p.v = Number(elem.querySelector('.planet_v').value.replace( /\s/g, ""));
1058          p.color = elem.querySelector('.planet_color').value;
1059          p.x = 0-p.r;
1060          p.y = 0;
1061          p.vx = 0;
1062          p.vy = p.v;
1063          p.ax = 0;
```

Above image is an example of objects. Some of the benefits of creating objects are that planets data is stored separately, each planet can be retrieved and handled separately, therefore making it easy to delete one or add one.

7. Graphical User Interface

```
/*Button animation*/
.btn_cont_anim {
  width: 150px;
  height: 40px;
  margin-left: 5px;
  margin-right: 5px;
  margin-top: 0;
  position: relative;
  display: inline-block;
  border-radius: 3px;
}
.line_anim {
  stroke: #009FFD;
  stroke-dasharray: 85 400;
  stroke-dashoffset: -223;
  stroke-width: 6px;
  fill: transparent;
  transition: 1s all ease;
}
.line_anim.line_red {
  stroke: #cc2222;
}
.line_anim.line_green {
  stroke: #207519;
}
.btn_cont {
  margin: -40px 0 0 0;
}
.btn_cont button {
    width: 150px;
    background: none;
    color: white;
    font-weight: 100;
    font-size: 15px;
    text-decoration: none;
    border: none;
    cursor: pointer;
    height: 40px;
    margin: 0;
}
.btn_cont_anim:hover .line_anim {
  stroke: #06D6A0;
  stroke-dasharray: 50 0;
  stroke-width: 3px;
  stroke-dashoffset: 0;
}
.btn_cont_anim:hover .line_anim.line_red {
  stroke: #65271b;
}
.btn_cont_anim:hover .line_anim.line_green {
  stroke: #14a001;
}

```

This program has CSS code integrated with the HTML and JavaScript code, making a clean GUI for the user as required by my client. An example, as shown in the picture, can be the button to start and stop the simulation.

References

"Planetary Fact Sheet". *Nssdc.Gsfc.Nasa.Gov*, 2021, https://nssdc.gsfc.nasa.gov/planetary/factsheet/.

"171 CSS Buttons". *Free Frontend*, 2022, https://freefrontend.com/css-buttons/.

"Functions - Javascript | MDN". *Developer.Mozilla.Org*, 2022, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions.

"Newton'S Law Of Universal Gravitation | Boundless Physics". *Courses.Lumenlearning.Com*, https://courses.lumenlearning.com/boundless-physics/chapter/newtons-law-of-universal-gravitation/.

"International Astronomical Union | IAU". *Iau.Org*, https://www.iau.org/public/themes/measuring/.

Word Count: 465