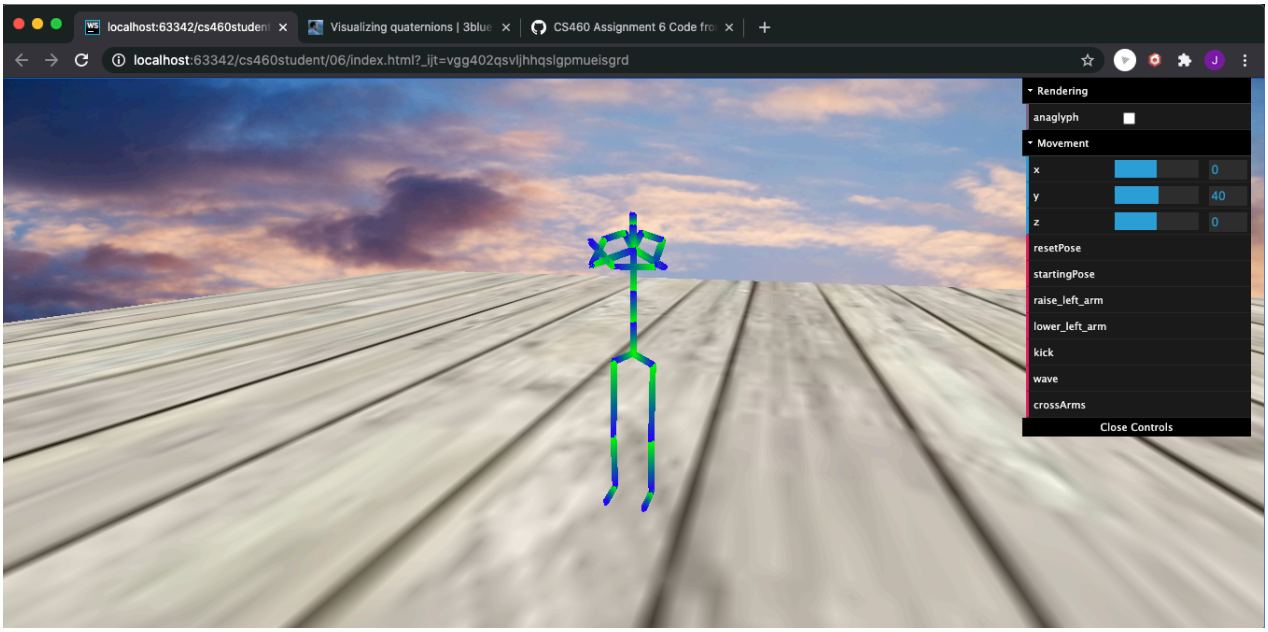
CS460 Fall 2020
Github Username: JamesEdMichaud
Due Date: 11/02/2020

Assignment 6: The Virtual Robot!

We will create a virtual robot that uses a human-inspired skeleton.
Hi Loraine! I couldn't get line widths to change on my computer, so I implemented a loop that creates a cluster of robots in a small area, giving the appearance of a thick line. This is DISABLED in my submitted code, but can be turned on by changing the value of N on line 66 of index.html to 3 or 5. You may need to change the line width on line 548 of robot.js if you do and it looks bulky.
Also, I apologize for the very lengthy code. I didn't have time to refactor it as much as I usually do, so there are a lot of copy/paste functions with slight differences for different bones.

In class, we started using the `THREE.Bone` class to assemble a skeleton of a virtual robot. Here, we will extend this skeleton and add code to animate certain robot movements. From a software engineering perspective, we will be writing **object-oriented code**. This allows us to create multiple robots within the same scene later on.



Starter code for assignment 6. In class, we pulled from the upstream repository and received the `06/index.html` file. This is the official starter code. However, we already worked on it during class so your local version might include some of this assignment.

Part 1 (2 points): Create an empty `robot.js` file and include it in the `index.html` file using the `<script>`-tag..

Part 2 (2 points): Modify `robot.js` to include an empty constructor for a Robot object. The constructor should take `x`, `y`, and `z` parameters to configure the position of the robot.
I also included a size parameter for the constructor. The y position is automatically set to the size. I could have just used y as the height, but other modifications require separate values.

Part 3 (2 points): Add properties to the Robot class. Each property is pre-fixed with `this.` which refers to the current robot if we created multiple robots. Please add the following properties: `this.head`, `this.neck`, `this.torso`.
I created a number of additional joints... Torso is broken into pelvis, lowerBack, midBack, and upperBack.

Part 4 (2 points): Initialize `THREE.Bone` objects for `this.head`, `this.neck`, and `this.torso`.

Part 5 (1 points): Create the **scene graph** hierarchy with `this.head` as root.

```
this.head.add(this.neck);
this.neck.add(this.torso);
```

I used the pelvis as the root. I hope that's acceptable.

Part 6 (10 points): Now we create the limbs of the robot. Let's start with the left and right arms. Each arm has three elements: `upper_arm`, `lower_arm`, and `hand`. Please create the hierarchy of the left and right arms. The parent node of the arms is `this.neck`. And, please do not forget to create `THREE.Bone` objects for all elements.
I added a number of additional joints here too. Instead of upper and lower arm, I used shoulder and elbow. Some rotations were giving me a really hard time (like if you spin your forearm), so I added extra joints for bicep and forearm to handle that type of rotation.

Part 7 (10 points): What about the legs? Please create the left and right legs. Each leg consists of a hierarchy of three elements: `upper_leg`, `lower_leg`, and `foot`. The parent node of the legs is `this.torso`. Also, please use `THREE.Bone` objects for all elements.
I added a few extras here too. Hip, hipRotator (like the forearm rotation above), knee, ankle, foot.

Part 8 (15 points): Write a `Robot.prototype.show(scene)` method that displays the robot by using the `THREE.SkeletonHelper` object. Then, change `index.html` code to create the robot and call the `show` method.

In `robot.js` add

```
Robot.prototype.show = function(scene) {

  var rGroup = new THREE.Group();
  rGroup.add( this.head );

  var helper = new THREE.SkeletonHelper( rGroup );
  helper.material.linewidth = 3; // make the skeleton thick

  scene.add(rGroup);
  scene.add(helper);

};
```

And in `index.html` add

```
// ...
var r = new Robot(0, -55, 0); // any position works
r.show(scene);

animate();
// ...
```

I couldn't get the skeleton helper lines to change thickness, so I instead created an NxNxN array of robots clumped together. I know this won't work once we skin the robot, but for this assignment it helped me to view the skeleton.
But nothing shows up! We need to adjust positions of the bones in part 9.

Part 9 (15 points): Please adjust the positions of all properties. All our properties are now THREE.Bone objects which have a position associated. Please modify the position to layout the robot's skeleton as we did in class. And remember, in this scene graph hierarchy, all positions are relative to the parent element.

Example:

```
Robot = function(x, y, z) {

  this.head = new THREE.Bone();
  this.head.position.set( x, y, z );

  // ...

  this.neck.position.y = -10;
  this.torso.position.y = -30; // the torso is 3x as long as the neck
```

The robot should now appear when reloading the local website!

Part 10 (20 points): A static robot is no fun! Let's add some movement. Let's first include the dat.GUI library and add sliders to change the x, y, and z position of the robot.

```
<script src="https://threejs.org/examples/js/libs/dat.gui.min.js" type="text/javascript"></script>
```

And configure the menu (and at the same time add the anaglyph rendering):

```
var controller = {
  anaglyph: false
}

var gui = new dat.GUI();
var rendering = gui.addFolder( "Rendering" );
rendering.add( controller, 'anaglyph' );
rendering.open();
```

For anaglyph rendering, replace the `render()` call in `animate()` with `effect.render()` depending on the controller.anaglyph flag. Note: use the `renderer.setClearAlpha(1)` to deactivate the sky background if anaglyph is active.

Then, add sliders for movement in x,y,z:

```
var moving = gui.addFolder( "Movement" );
moving.add( r.head.position, "x", -1000, 1000 );
// ... add y and z
moving.open();
```

Done! This was a bit tricky to do with the group of robots. I needed to save the initial robot x, y, z offsets as instance variables for each robot.
There is slider bug in dat.GUI and we need to set up the trackball controls only on the domElement:

```
controls = new THREE.TrackballControls( camera, renderer.domElement );
```

Part 11 (20 points): Finally, let's add some animations! We can use slerp interpolation between quaternions to move the robots' limbs. We will need a `Robot.prototype.onAnimate` function that is called from the `animate()` loop.
Then, please add the following functionality

```
Robot.prototype.raise_left_arm = function() {
  this.movement = 'raise left arm';
};

Robot.prototype.lower_left_arm = function() {
  this.movement = 'lower left arm';
};
```

```
Robot.prototype.kick = function() {
  this.movement = 'kick';
};

Robot.prototype.onAnimate = function() {
  if (this.movement == 'raise left arm') {
    // ... TODO slerping
  } else if (this.movement == 'lower left arm') {
    // ... TODO slerping
  } else if (this.movement == 'kick') {
  // ... TODO slerping and check once it is done for a backwards slerp
  // you can use the identity quaternion for a backwards slerp
  }
};
```

And then, please connect dat.GUI to these actions!

Part 12 (1 points): Please update the screenshot above with your own and then post the github pages url here:

https://jamesedmichaud.github.io

Bonus (33 points):

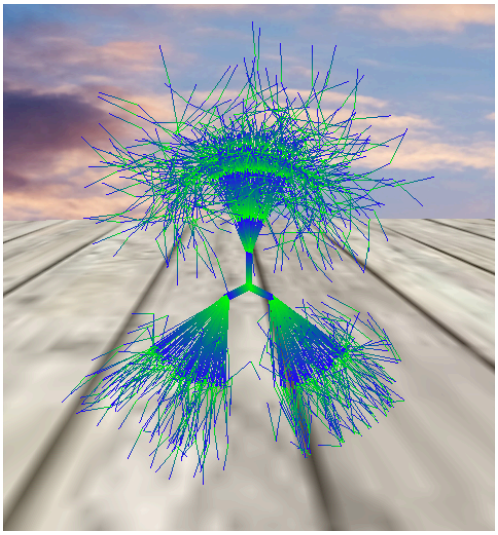
Part 1 (11 points): Use the window.onclick callback and a THREE.RayCaster to place to robot on the floor plane if the user clicks while holding shift. Don't forget to update the dat.GUI sliders with the new position (look at the `.listen()` function of dat.GUI).
Done. (see next page for Part 2)

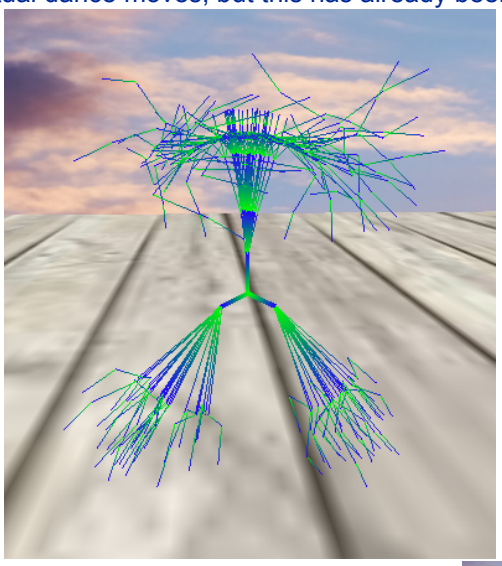
Part 2 (22 points): Add a `Robot.prototype.dance` function that makes the robot go crazy (a lot of movements). Please add some screenshots.

Now this was interesting! My robot class builds an array of functions that can be called to slerp any joint (Bone) that I wrote actions for. Each function picks a random side (left/right, if applicable) and a random angle within the range of motion for that joint, then slerp's the quaternion.

Since the numbers are generated randomly by the robot, wacky things happen with my approach of using multiple robot instances to simulate a thick line.

With more time I might have tried to make actual dance moves, but this has already been a relatively large time sink...

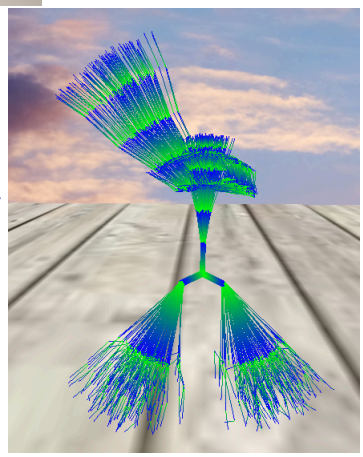


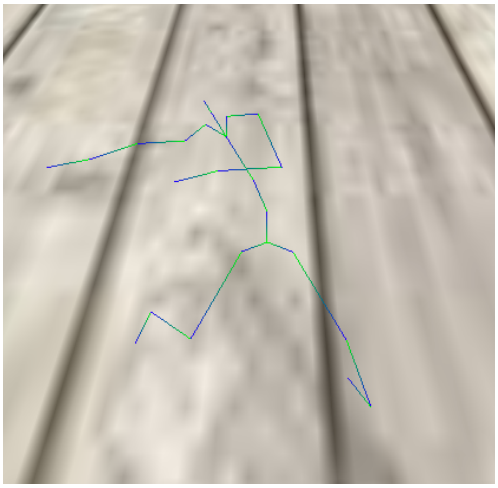


On the left it's 5x5x5 (125 robots), and on the right it's 3x3x3 (27 robots). This happens because each of the robots in the cluster chooses its own random Bone, angle, and interpolation value.

Left: Some pretty cool trail-like effects can be seen if you make the robot kick mid-dance.

Right: Waving is just kind of weird looking though.





And here are two screenshots of a solo dancer.