PROTOTYPING MOVEMENT

with C++ and Arduino

MATTERLAB 2020 - RACHAEL SHIELDS

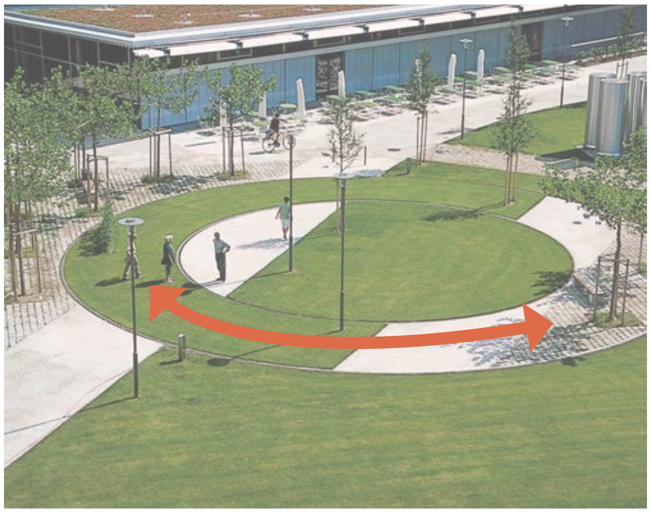
TABLE OF CONTENTS

“ kinetics is finding ways to continually move the built environment ”

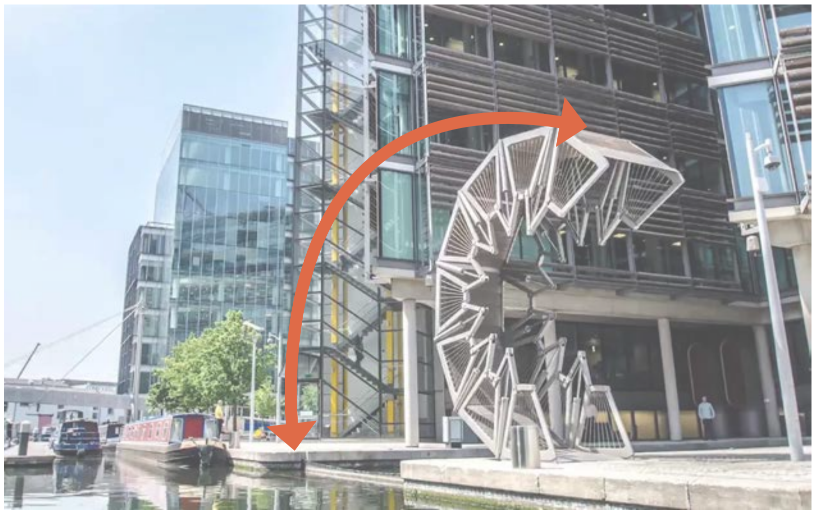
KINETICS

Kinetics in the built environment is movement that can range from adjustable seating, to movable trees, to an urban plaza that robotically transforms shape based on need. Currently, kinetics tends to stay within the realm of moving sculptures or stadium roofs, but the future holds an abundance of opportunities for kinetics in the landscape. I argue the built environment should have the flexibility to adapt to change, whether that is a changing society, environment, or user type without the need for continual demolition. Serious global problems like urban space constraints, stormwater management, and energy production might find their solutions with kinetic environments. Furthermore, kinetics can provide a powerful new addition to participatory design, putting design into the hands of the user with the ability to customize a kinetic environment to their particular needs or desires. The inspiring possibilities are seemingly endless.

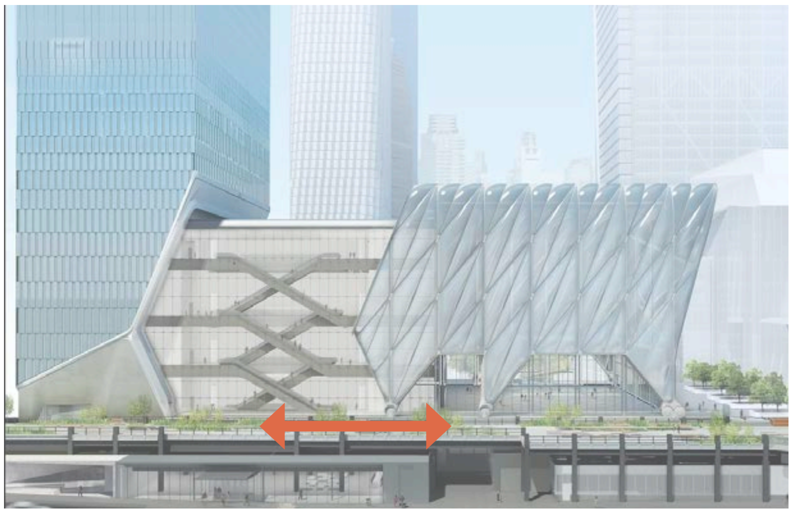
landscape architecture



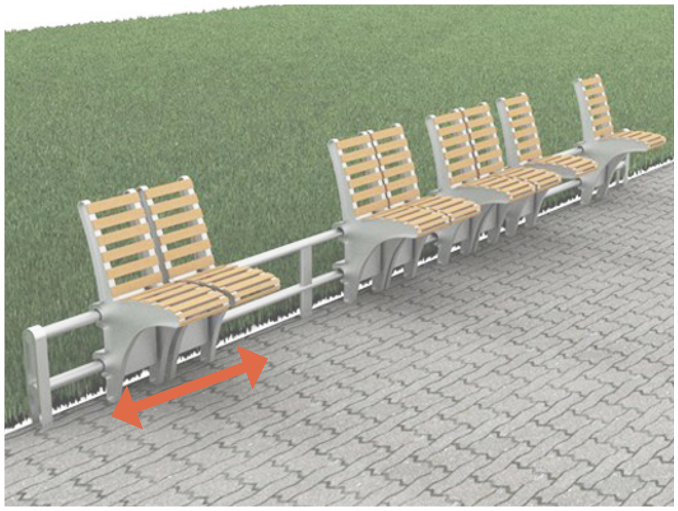
bridges



architecture



furniture



MATTERLAB
INTRO TO THE PROJECT

RESEARCH OBJECTIVES

1. create prototype iterations of kinetic topography.
2. create a guide to replicate the prototypes with the dual purpose of teaching how they work.

WHY PROTOTYPE MOVEMENT?

1. Motion adds a more complicated dimension to design. A 3D model is not enough to design moving parts.
2. Digital models make it difficult to take into account environmental or user inputs. Arduino offers this.
3. Animated digital models lack realistic movement, materials, and fabrication methods needed to accurately engineer such ideas.
4. Errors can be minimized before it becomes a financial or safety concern.
5. Prototyping is key to test out parameters like speed, actuator type, or any sensors involved.
6. A physical prototype strengthens the validity of complex, seemingly science fiction concepts like a kinetic environment for the client/user.
7. The act of prototyping can be used as a method of design.
8. Prototyping puts design into the landscape architect's hands instead of a mechanical engineer's.

HOW DO WE PROTOTYPE MOVEMENT?

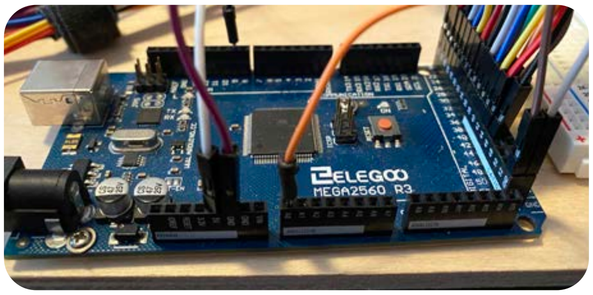
as a solution, I propose using Arduino and C++ code to program a movable and adjustable prototype. In essence, Arduino is a mini-computer that takes the hardware wired to it and C++ code to do a task. In this case the tasks will be moving servo motors, reading a water level sensor, and reading signals via bluetooth.

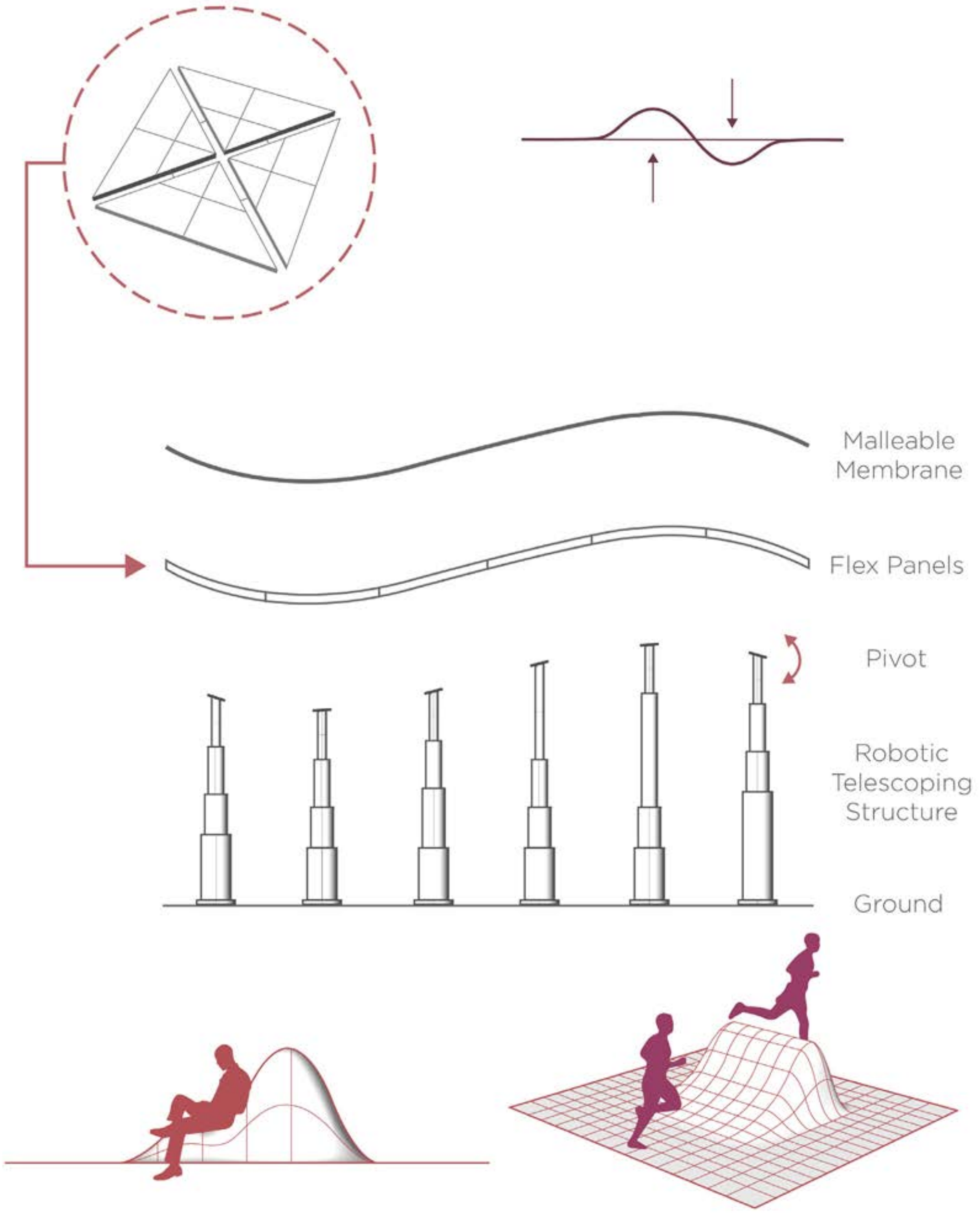
C++ CODE

```
void readWaterSensor() {
  digitalWrite(WATER_SENSOR_POWER_PIN, HIGH);
  delay(10);
  water_level_raw = analogRead(WATER_SENSOR_DATA_PIN);
  digitalWrite(WATER_SENSOR_POWER_PIN, LOW);
}
```

+

ARDUINO





WHAT WILL IT LOOK LIKE?

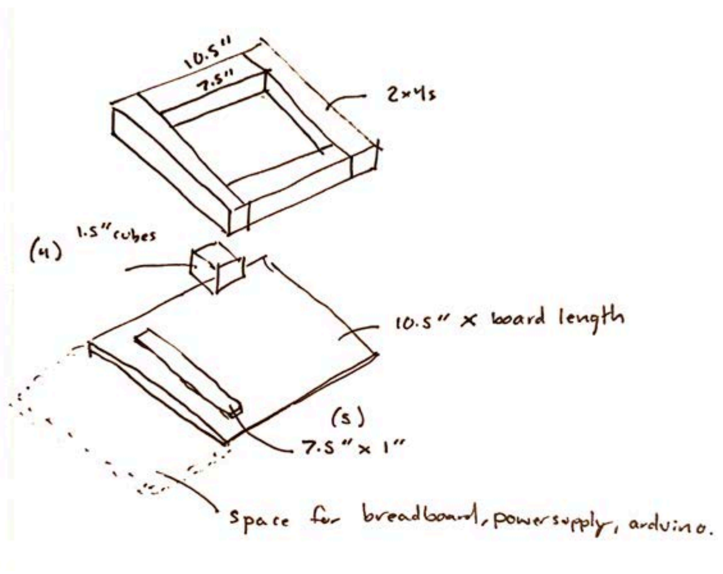
the goal is to create a malleable surface like this that can handle significant grade changes and hold the weight of a crowd of people.

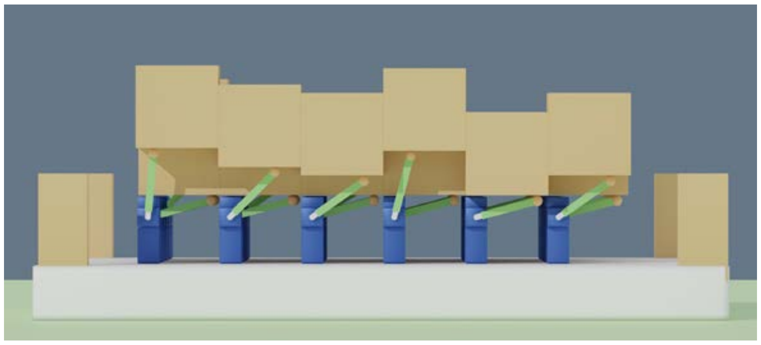
OVERVIEW
PROTOTYPE 1

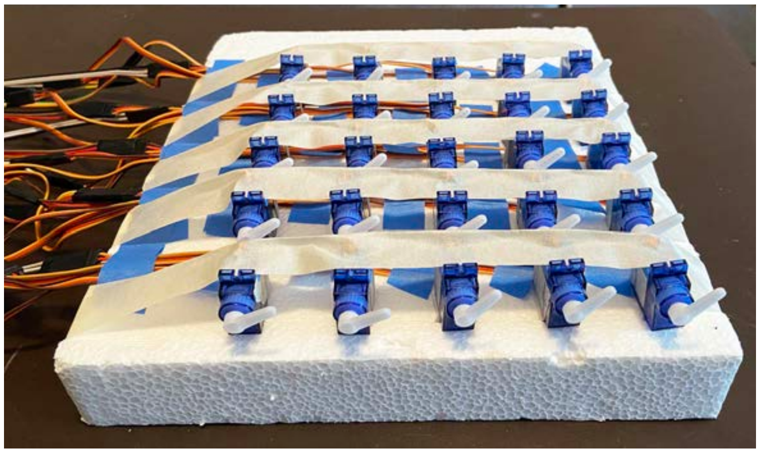
VIDEO LINK: https://youtu.be/daMVpNQ7z8o

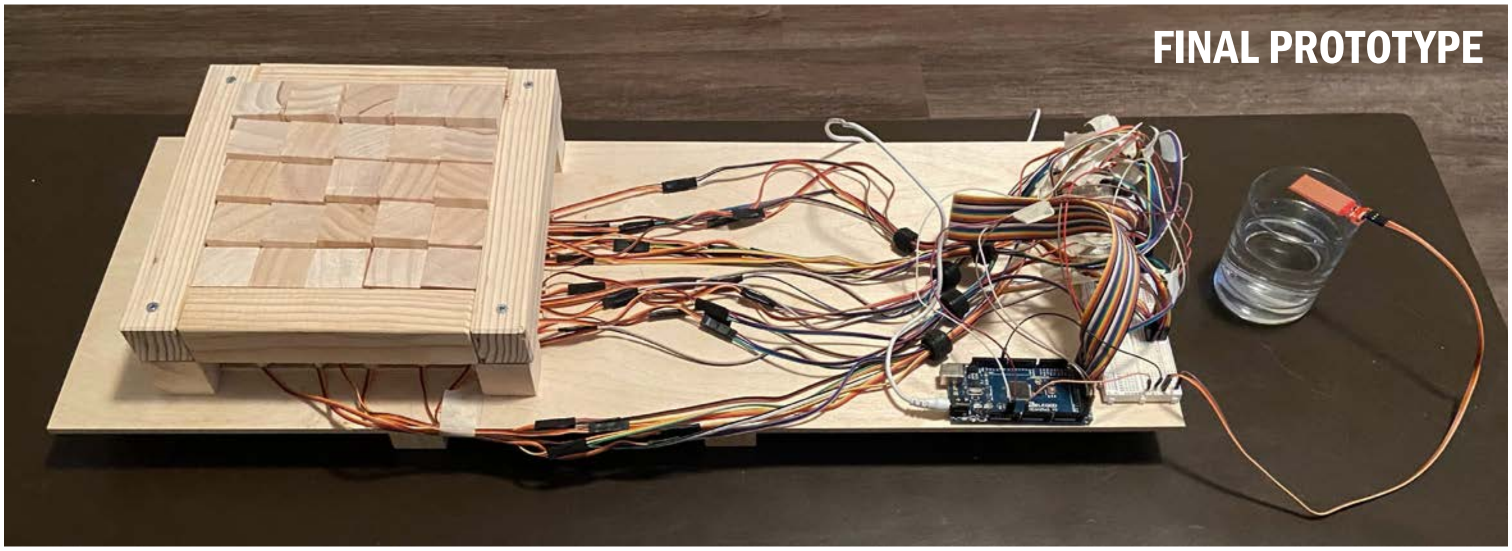
VISION
imagine this as a water-front plaza. maybe during a normal day it has a few blocks up for casual seating while viewing the coast. maybe during an event, a stage is created and tiered seating is pushed up. and maybe during a storm the blocks automatically react and rise up to create a flood barrier.

PROCESS
the process started with some hand sketches and progressed to 3D modeling. physical prototyping started with several small cardboard models, this styrofoam model, and then the final wood version.



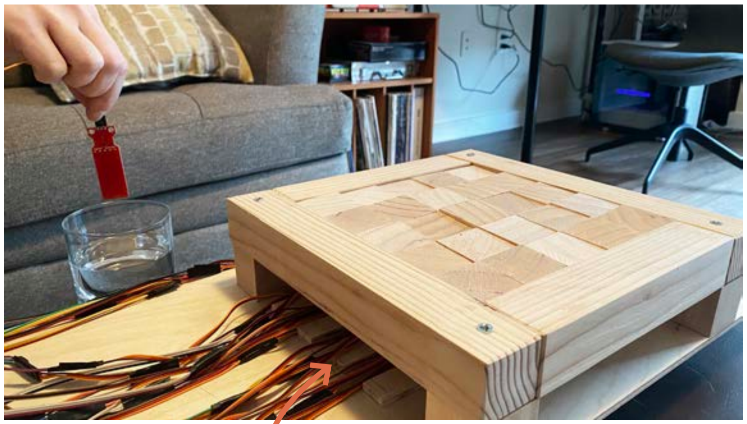


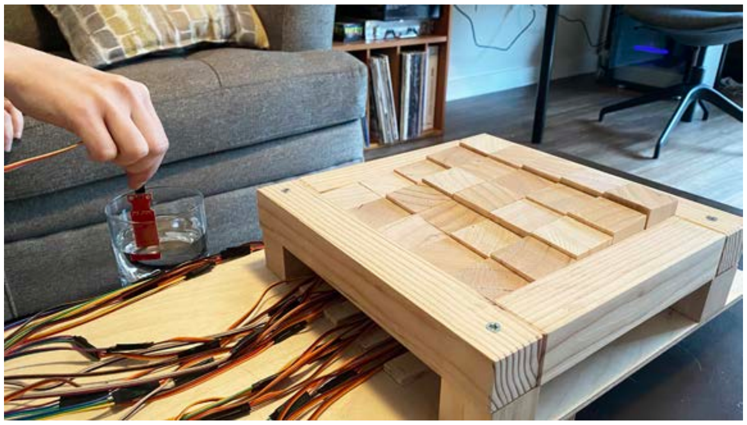


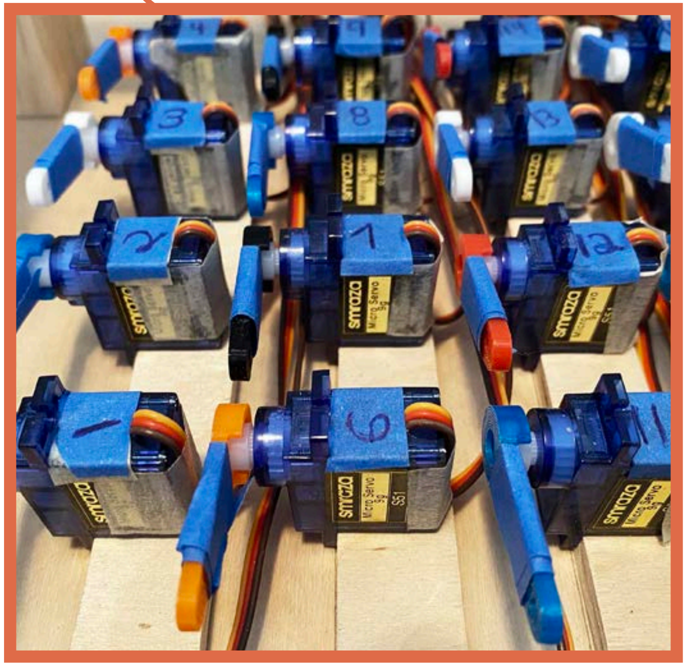


HOW IT WORKS

a water level sensor is inserted in water which causes 25 servo motors to rotate and lift up 25 wood blocks to different height

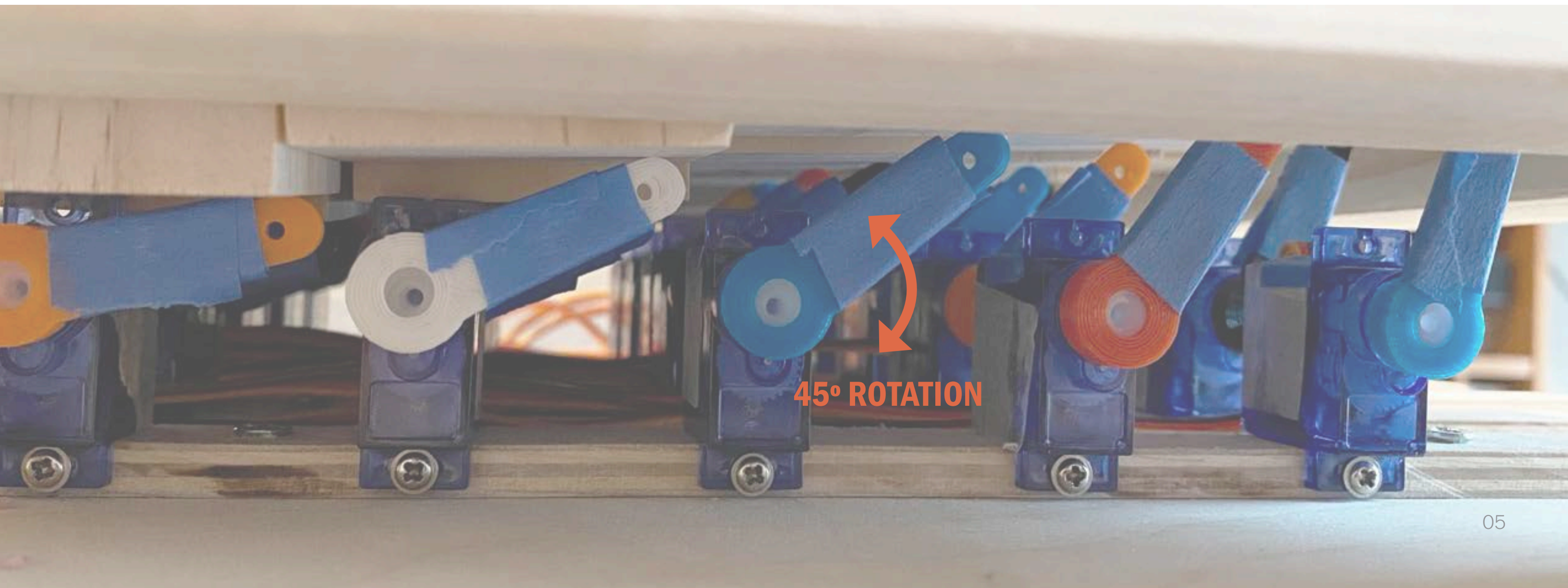






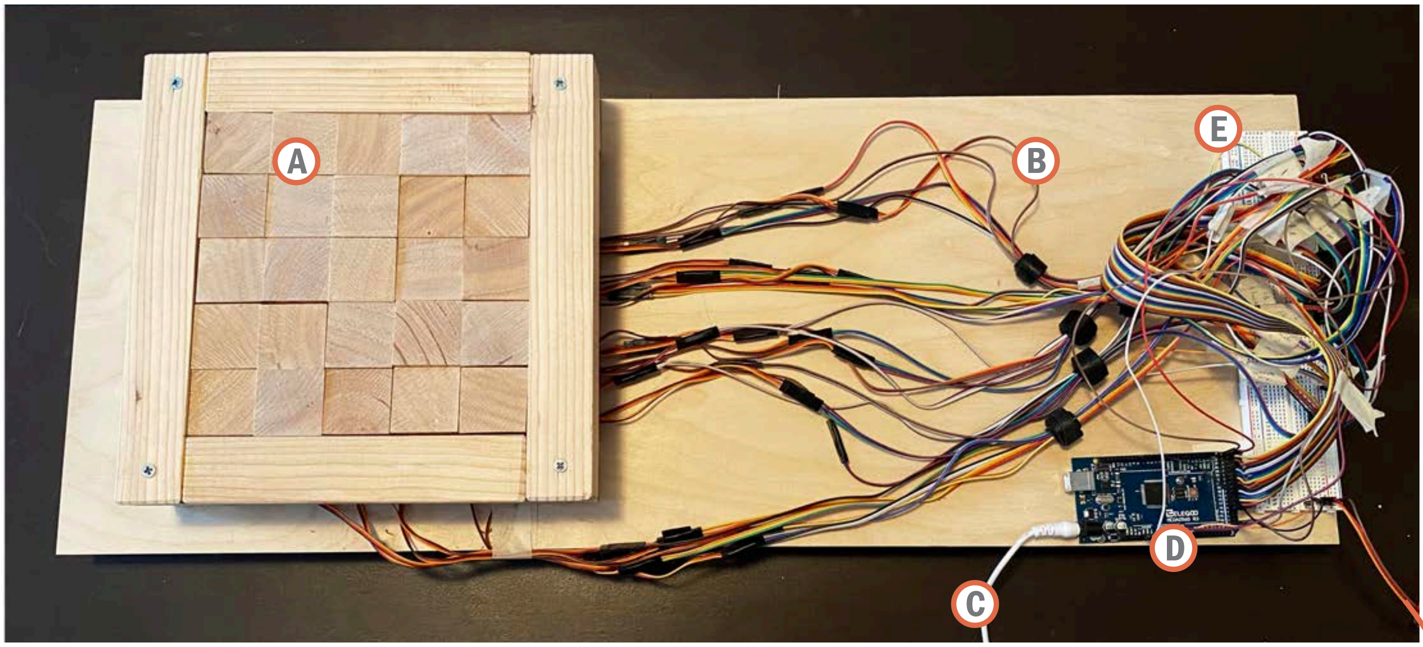
the servos are lined up in a grid under the wood blocks. they are numbered to help with setting up the code and wiring

the size of the motors directly created the size of the prototype. they couldn't be placed closer together and so smaller blocks couldn't easily be used in this configuration

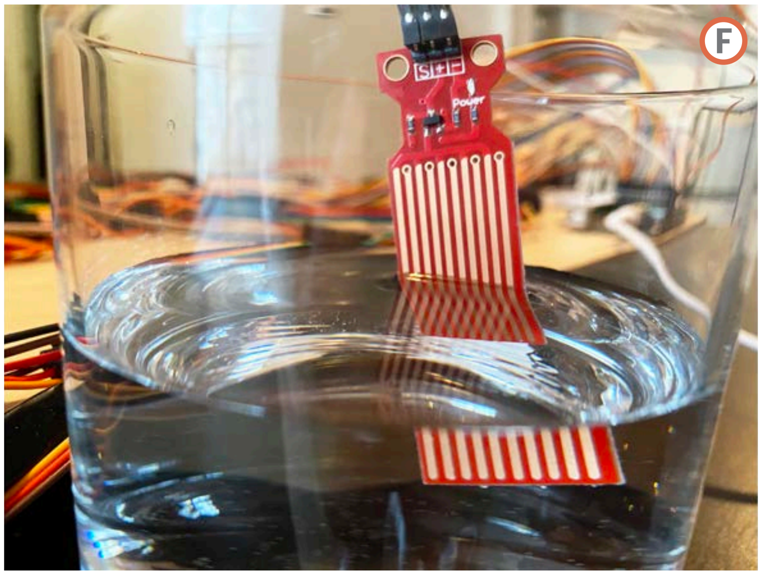


COMPONENTS

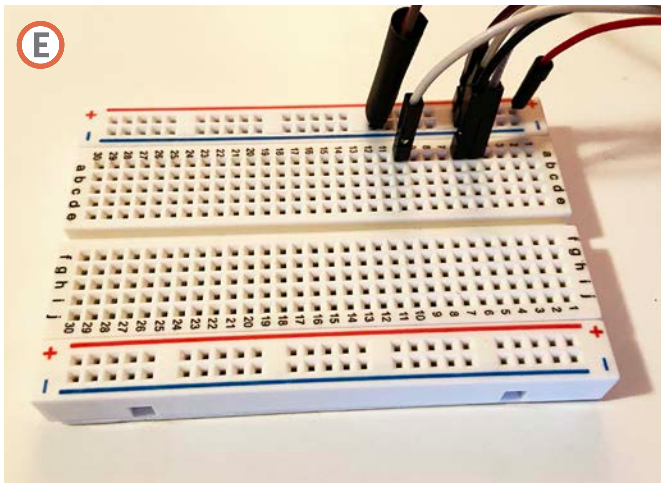
* arduino layout explained in detail in schematic section of booklet



A	WOOD BLOCKS
B	JUMPER WIRES
C	ARDUINO POWER
D	ARDUINO
E	BREADBOARD
F	WATER LEVEL SENSOR
G	WOOD FRAME WITH LEGS
H	SERVO MOTORS
I	3D PRINTED ARM EXTENSIONS
J	WOOD SERVO SUPPORTS

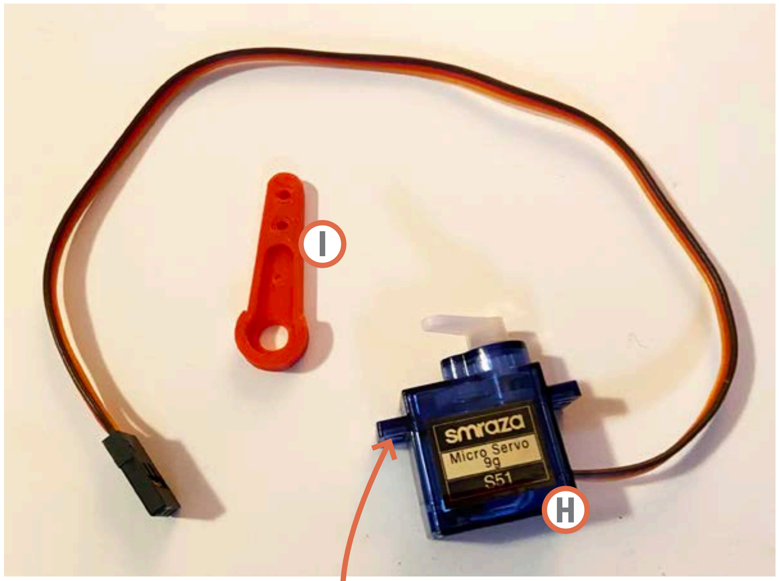


other sensors like a motion sensor could be attached, but a water level sensor was chosen to show how landscapes could react to climate change.



a breadboard allows us to quickly connect and remove all the wires to the arduino without soldering.

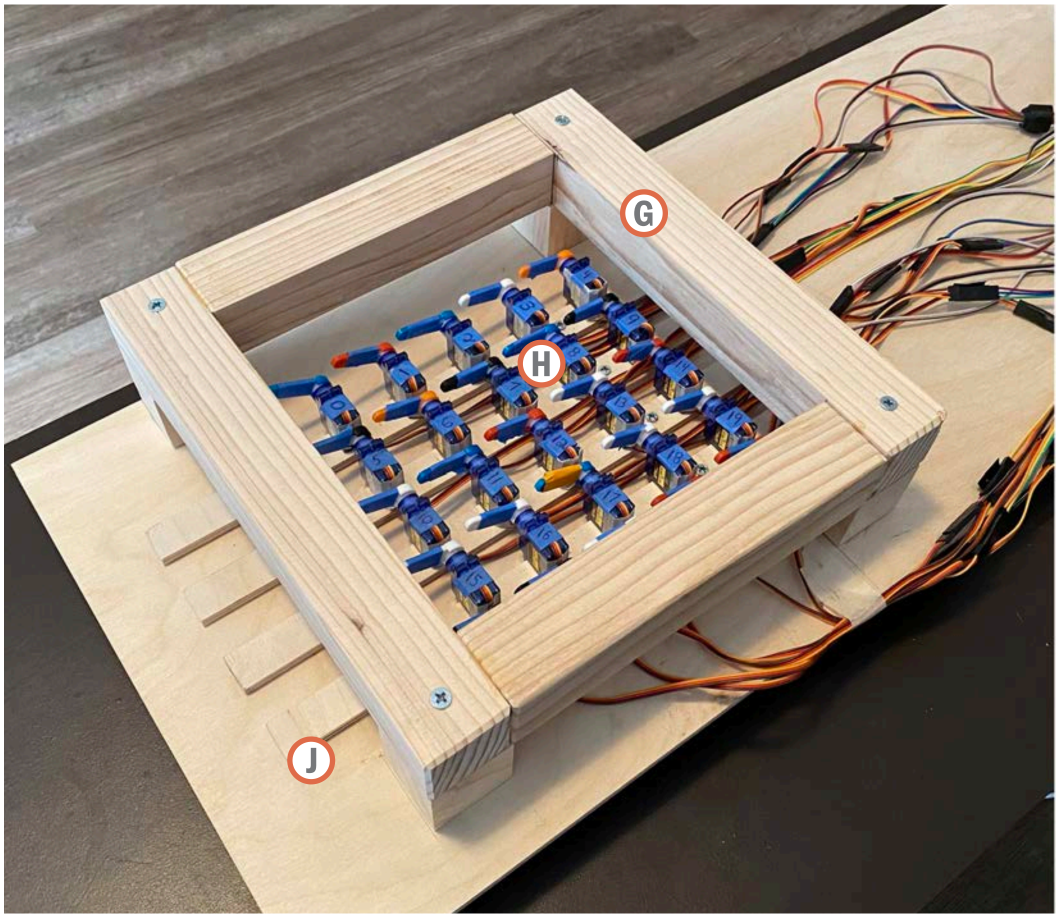
servo motors were used over other options like a linear actuator due to their small scale and affordability

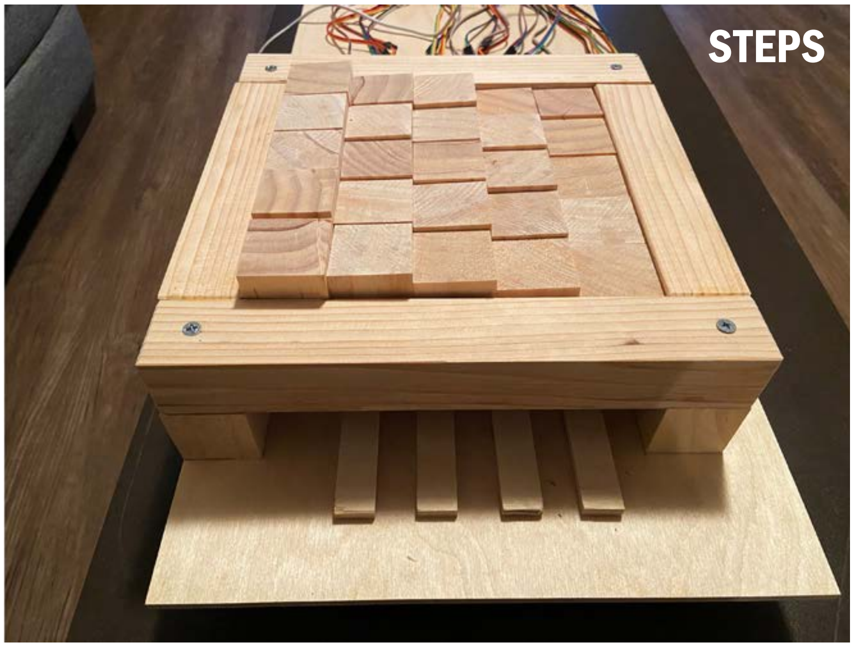


I

arm extensions increased range of motion from 1/8" to 1/2"

wood servo supports are need due to the screw hole placement





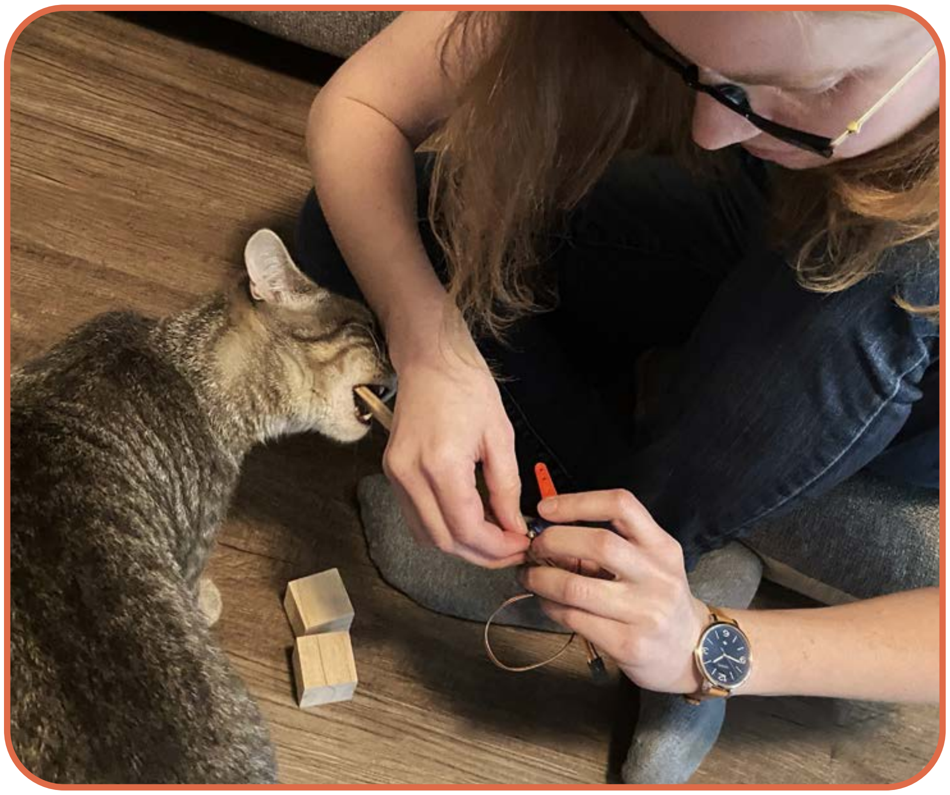
CONFIGURATIONS

to pair with the water level sensor I chose the stair stepping configuration. however, many others can be programmed. the 5 x 5 grid is the limiting factor here.

ANALYSIS

	PROBLEMS	SOLUTIONS
1.	the wood blocks did not move perfectly straight up and down. you can see the blocks not laying flat or going down smoothly. WHY: • i thought purchasing pre-cut blocks would help solve this. however, the blocks i purchased were not perfectly sanded so when I sanded them to reduce friction they were then not exactly square. • the servo arms were not aligned precisely to center.	current: a piece of code was added so that after each configuration the arms would shake the blocks to help get them back down into place. future: if i were to do another iteration i would put the blocks on rods to help with straight up and down movement. or try an iteration where the arms are attached to the blocks. future: i would laser cut more pieces to help with precision.
2.	the blocks only moved 1/2” WHY: • small servo motors were chosen because of their affordability and that they would reduce the overall size of the model. however, this came at the cost of a small amount of movement.	current: extension arms were added to get the 1/2”. you can see in the process image that i started without the arms and that only got 1/8” of movement. future: use a different method of movement. the better method would have been to use linear actuators (piston arms that move straight up and down). however, the project would have cost up to 15x more due also to the size increase.
3.	the resolution of the configurations is poor WHY: • in order to cut down on cost and make something that could be finished within the Matterlab timeframe it was important that the model be fairly small. the size of the motors and arms directly dictated the size of the blocks. the amount of servos/blocks was chosen out of concern for additional power requirements.	current: i deemed it more important to show the concept rather than invest extra resources and time. however, additional power could be added and the grid could be extended on this same concept. future: change the way that the arms function and interact with the blocks. consider moving the servos to the side and having long arms.
4.	it is pre-loaded configurations without user interaction. WHY: • user interaction adds an additional level of complexity. it was important to get to this level of prototype first before adding in anything else.	current: it doesn’t have user interaction, but it has environmental interaction. future: the sensor could easily be switched out to something like a motion sensor for user interaction,

SUMMARY



TAKEAWAYS FOR PROTOTYPE 2

- precision - nothing should be cut by hand if possible.
- provide user interaction
- try continuous topo
- add more servos to increase the resolution
- scale up the model

this prototype was disassembled to reuse the materials for prototype 2

this prototype took three months to build due to learning how to code and work with arduino. so there is indeed a large learning curve. i heavily underestimated the time it would take to complete the first prototype and that didn't leave much time for prototype 2. however, i was pretty proud of getting this far. it has always been important to me to find ways to use kinetics to provide more benefit than just a play surface. mostly because in order to sell people on an expensive concept like a moving landscape you need as many benefits as you can get. i didn't go into making this model with the intention of creating a space to adjust to flooding, it was a happy accident.

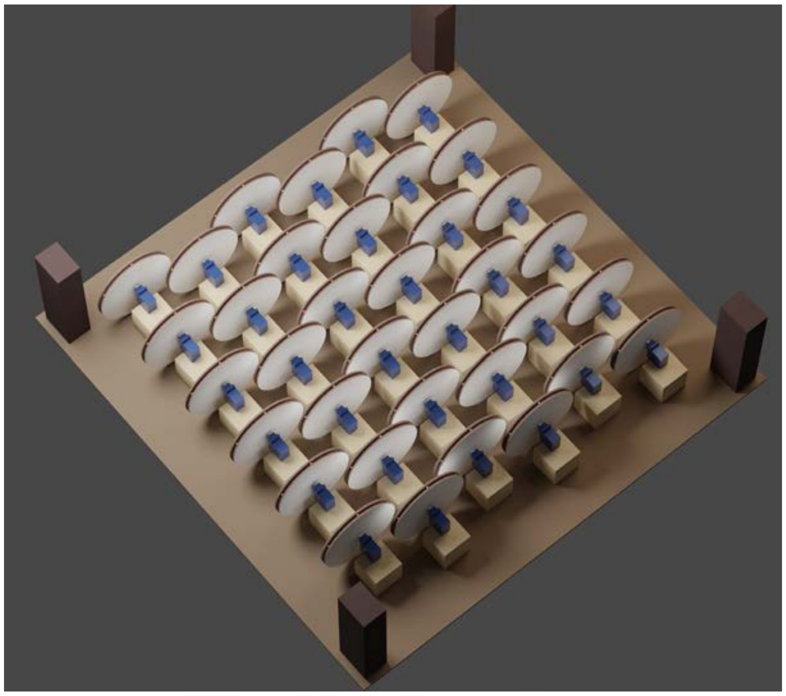
the original concept was to have a grid of pistons moving up and down with a flexible surface over top. that idea was quickly squashed when I realized the cost involved and the intensity of the engineering required. i needed something simpler to start with and so the idea for the wood blocks was born.

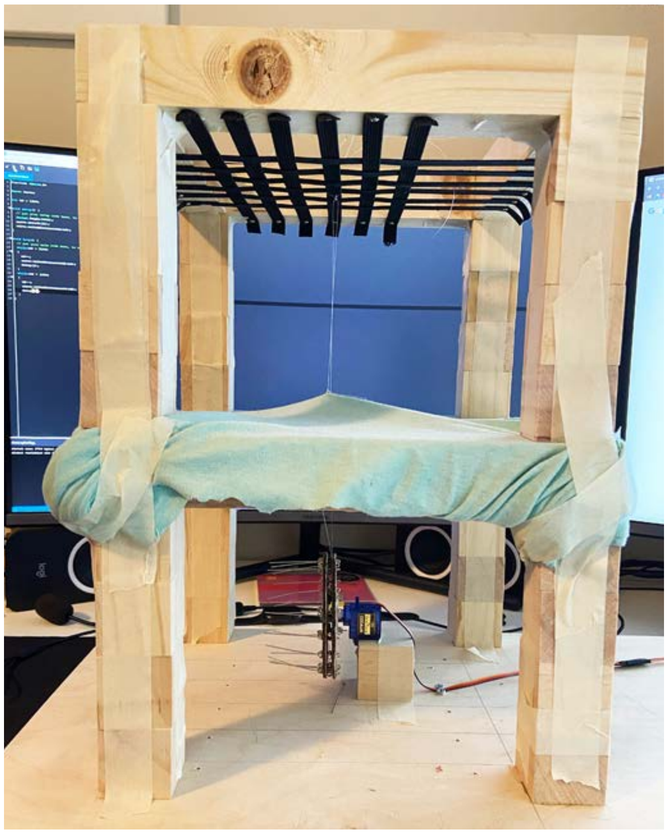
precision turned out to be extremely important to this prototype . it is small scale and imperfections easily multiplied. the servo motor really dictated the model and it was built around trying to work with its faults. several servos also died during the process. if I were to redo the block method model I would CNC or laser cut as much as possible and use a different method of movement.

OVERVIEW
PROTOTYPE 2

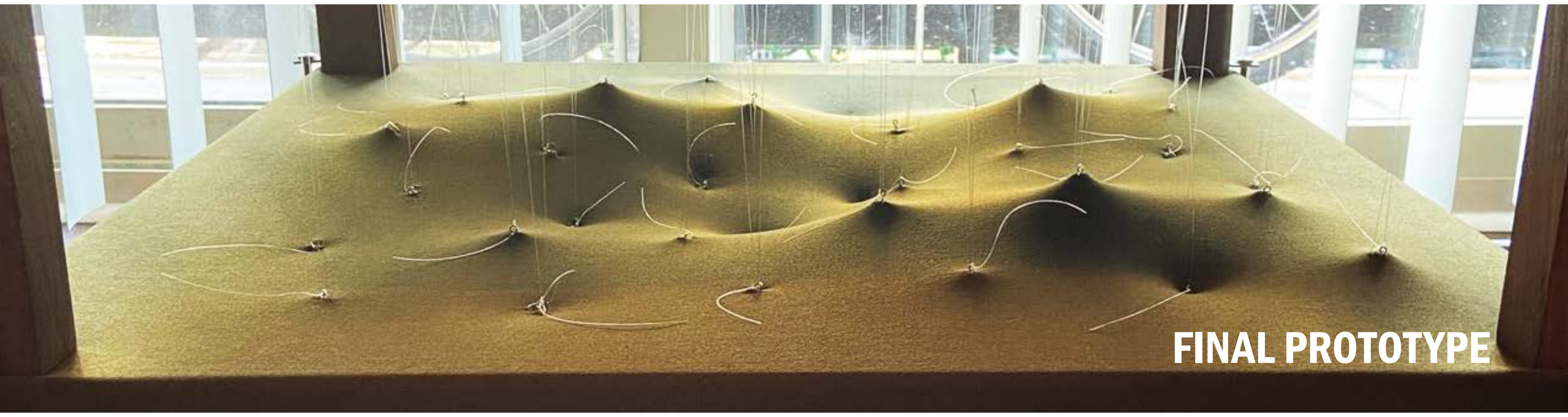
VISION
imagine this as a active play surface. a park that is modifiable to desires. a new form of interactive play/exercise. imagine this as a one-hole golf course with infinite configurations. or maybe futuristically a way to blend architecture and landscape architecture into one adjustable surface.

VIDEO LINK: https://youtu.be/6nVXO4o4Iwk





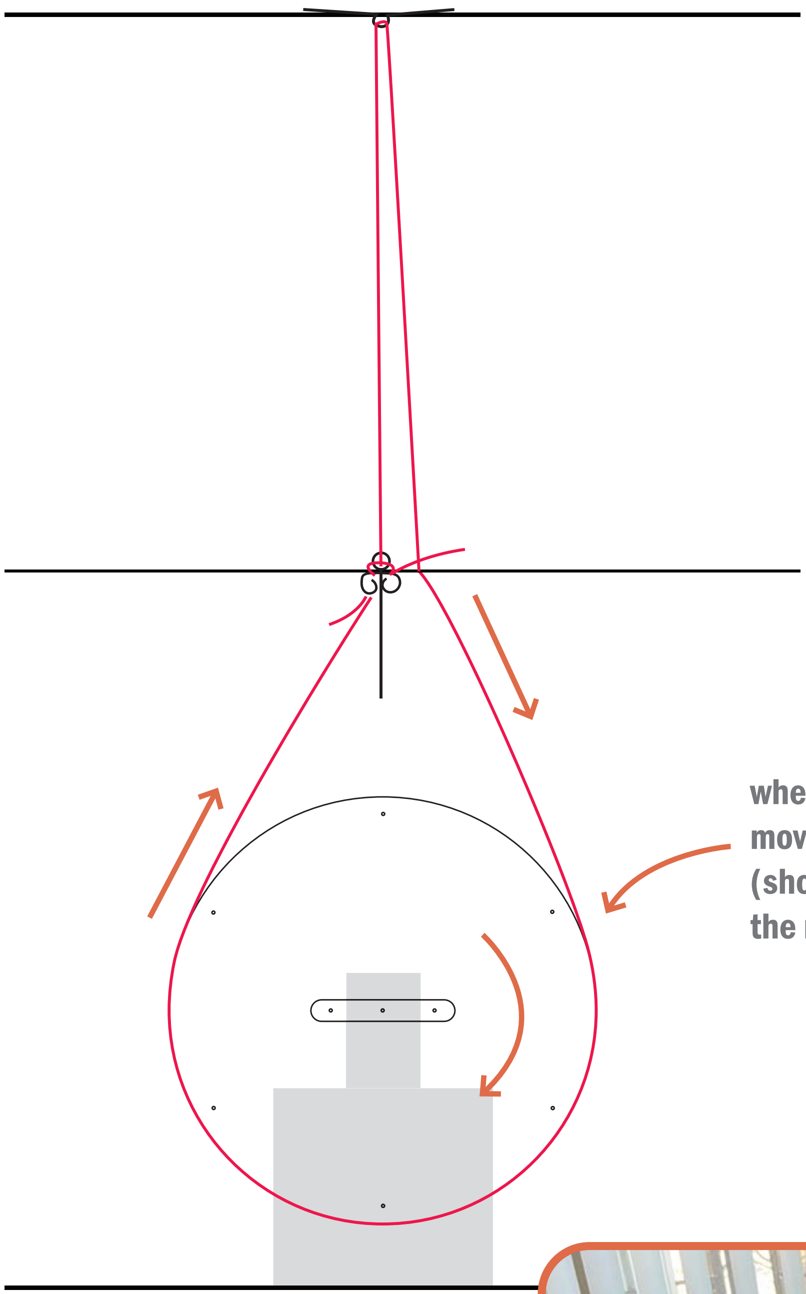
PROCESS
materials were recycled from the first prototype which created a process where a lot less sketching and 3D modeling was done in exchange for hands on tinkering.



modifiable topographic surface

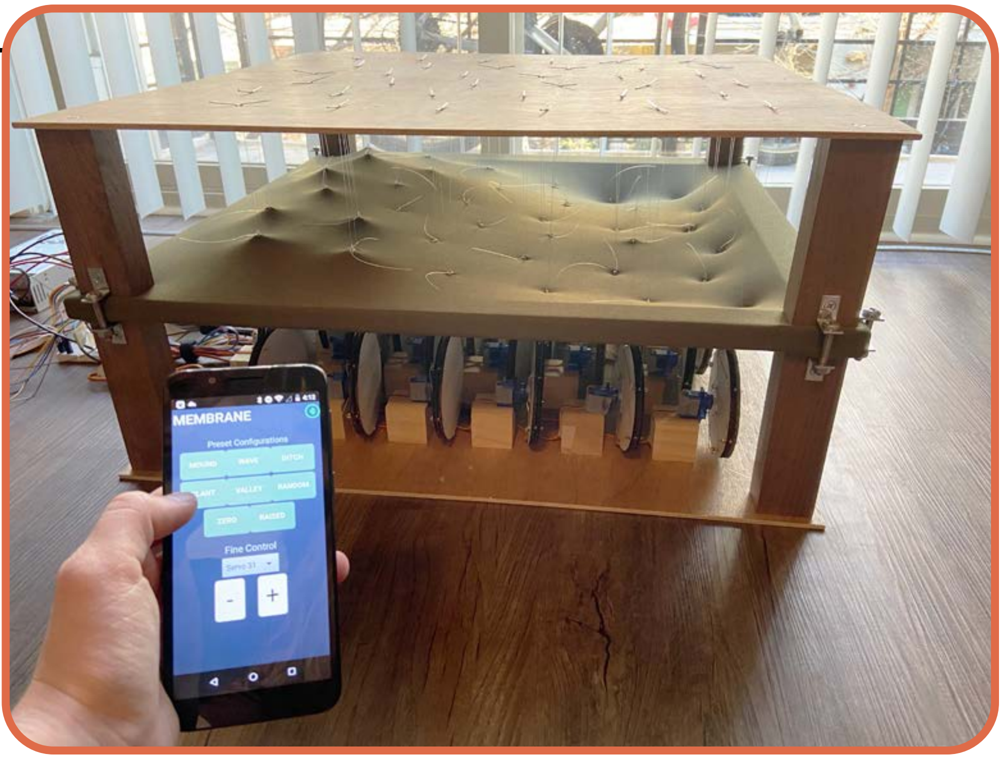
HOW IT WORKS

lesson learned from prototype 1 is that the servo arms didn’t move the blocks enough. an alternative solution to the arm extensions is to attach a wheel. with a large circumference we can get a lot more movement.



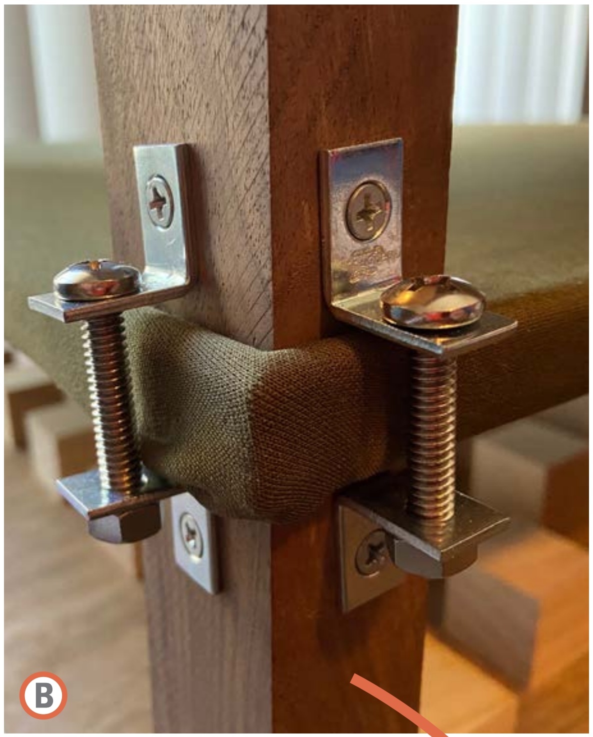
when the servo arm and wheel attachment move to the left the surface gets pulled down (shown in diagram) and when they move to the right the surface gets pulled up.

the membrane is operated with an app that has preset configurations or individual servo control.

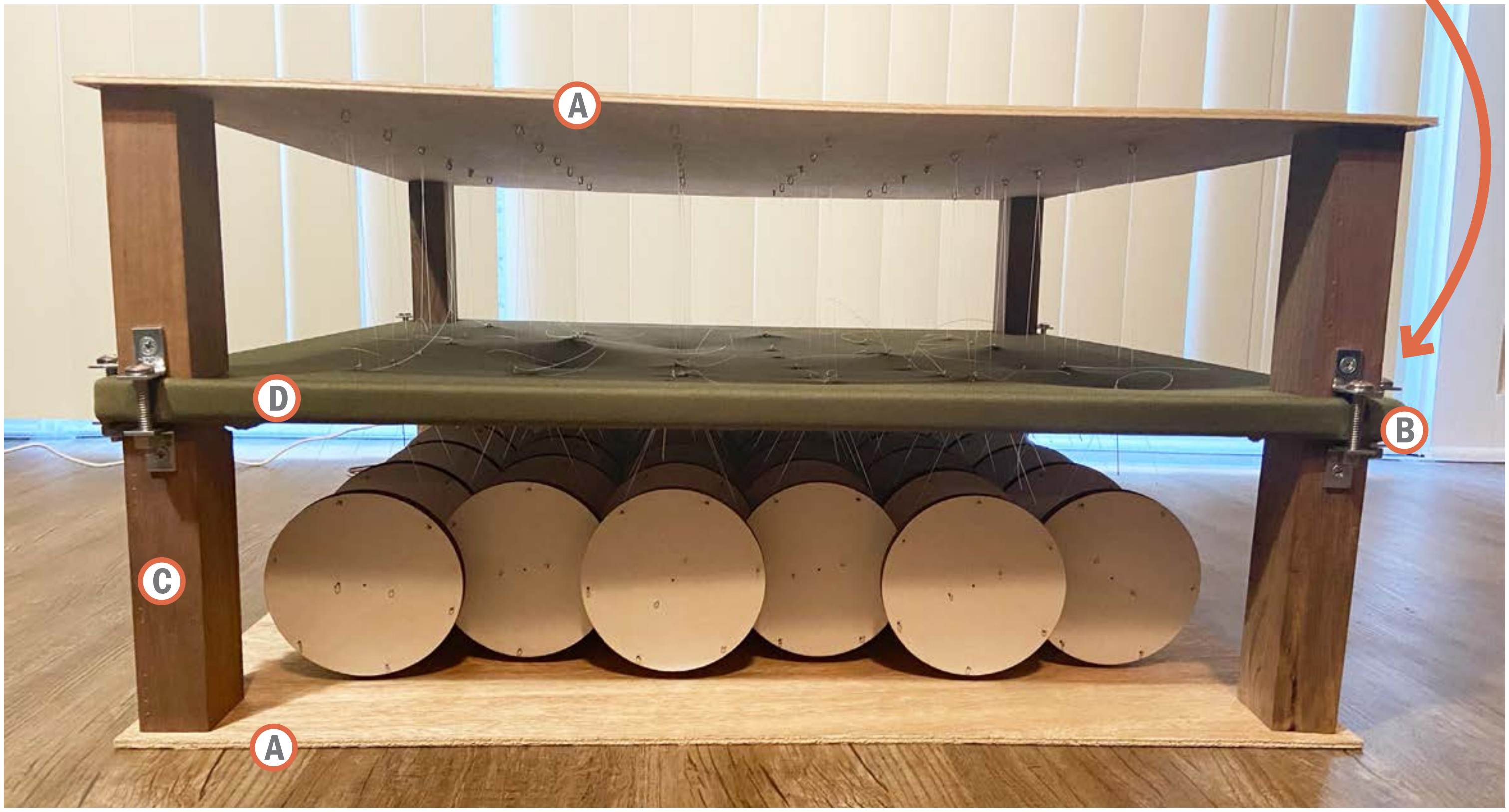


COMPONENTS

bolts and L brackets were a solution to having to attach the wood legs to fabric. this allowed the fabric to stretch and also did not create any cuts that would later rip.

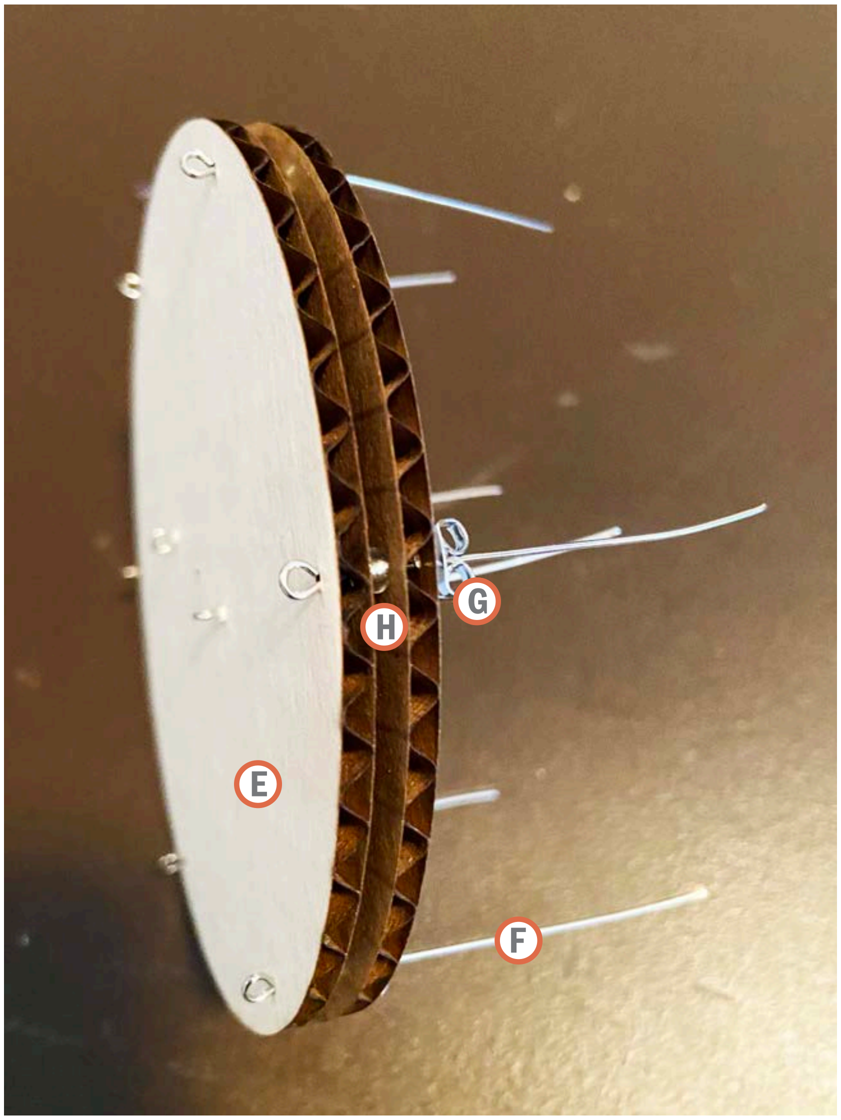


A	2' x 2' WOOD BASE AND TOP
B	BOLTS AND L BRACKETS
C	6" AND 5" WOOD LEGS
D	STRETCHABLE FABRIC STAPLED TO WOODEN FRAME

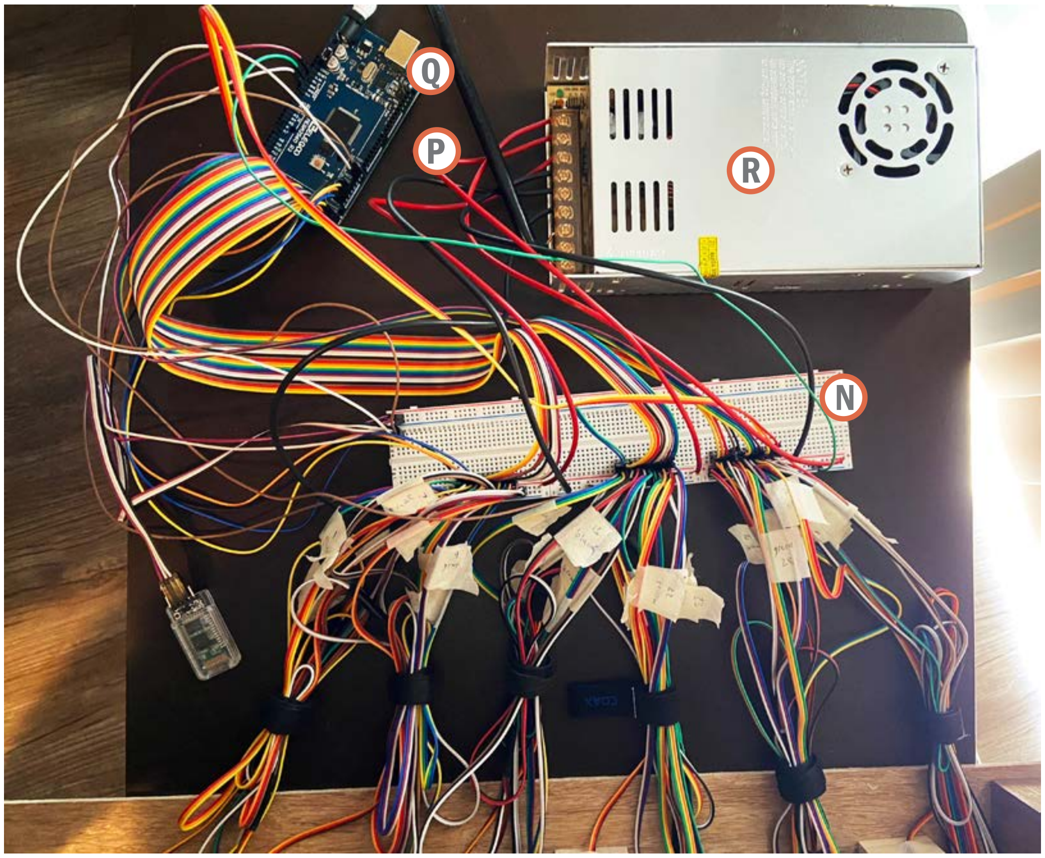


the wood base and top both have 36 holes drilled in them to screw in the wood blocks and attach cotter pins. these were hand grided out and drilled with a drill press. precision for this was less important, but I should have laser cut the holes instead.

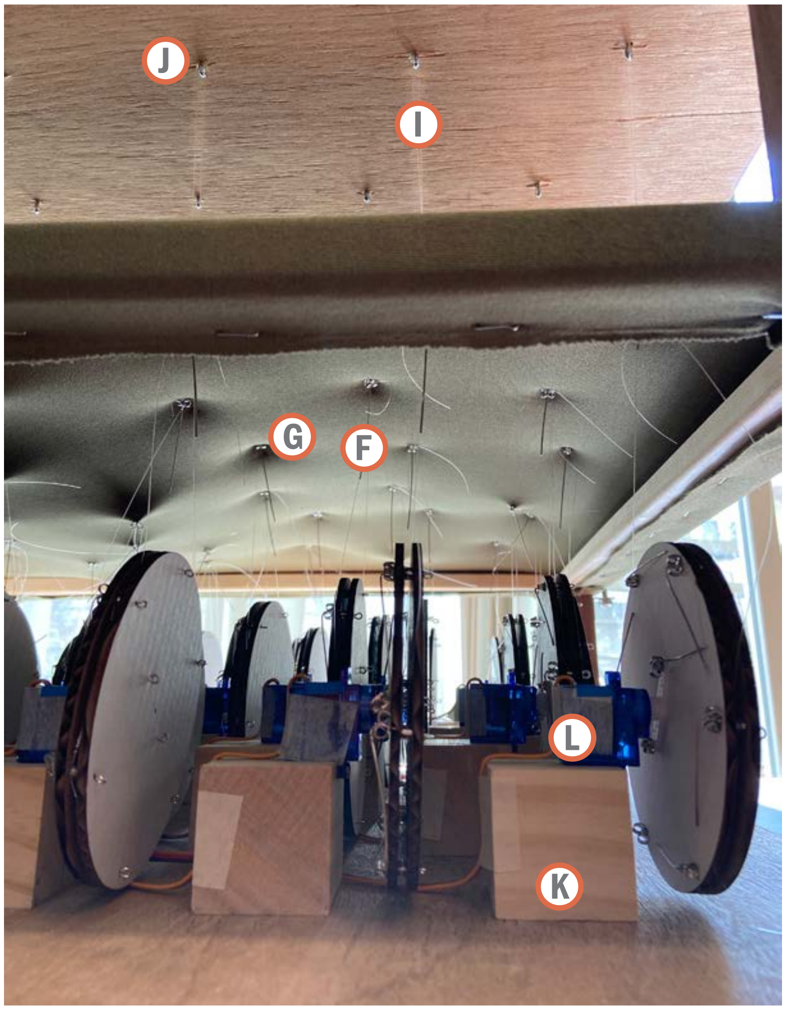
E	LASER CUT 4" CARDBOARD CIRCLES
F	EARRING RODS
G	EARRING BACKS
H	SPACER BEAD



each wheel attachments comprised of 6 earring rods and backs to clamp the two cardboard circles together. there is a spacer bead in between where the elastic will run around the perimeter. there are also two earring rods and backs to clamp on the servo in the middle.

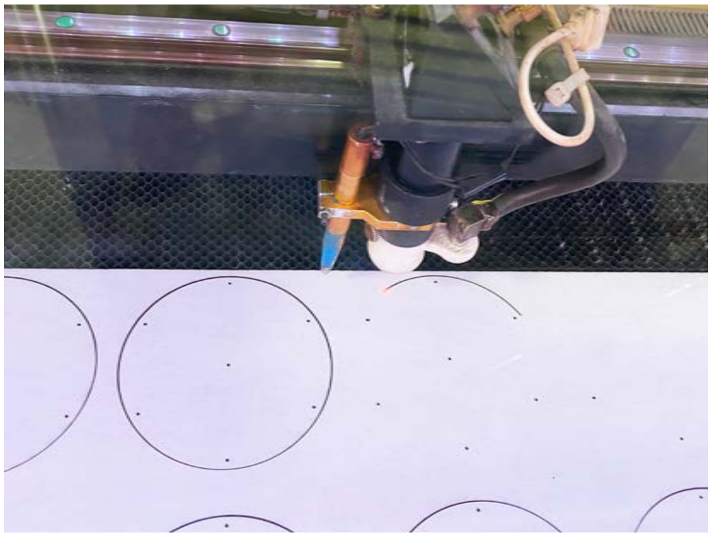


arduino wiring schematics are shown in detail on pages 26-31. the power supply had to be introduced to go from 25 servos to 36. the bluetooth module allows the app to control the membrane.

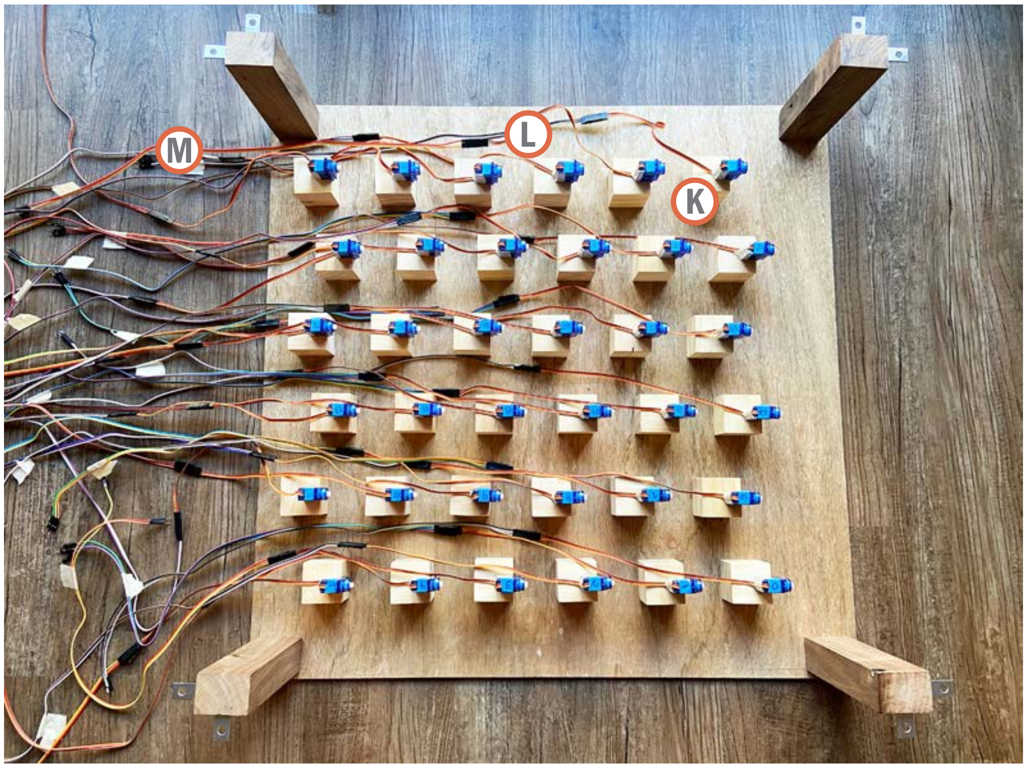


the earring rods and backs and sewn into the fabric. elastic string is then looped around the cardboard circles and the up to the cotter pins. in a real landscape the pulley system can be covered and be a design element.

the circles were laser cut due to the lesson learned in prototype 1 about precision.

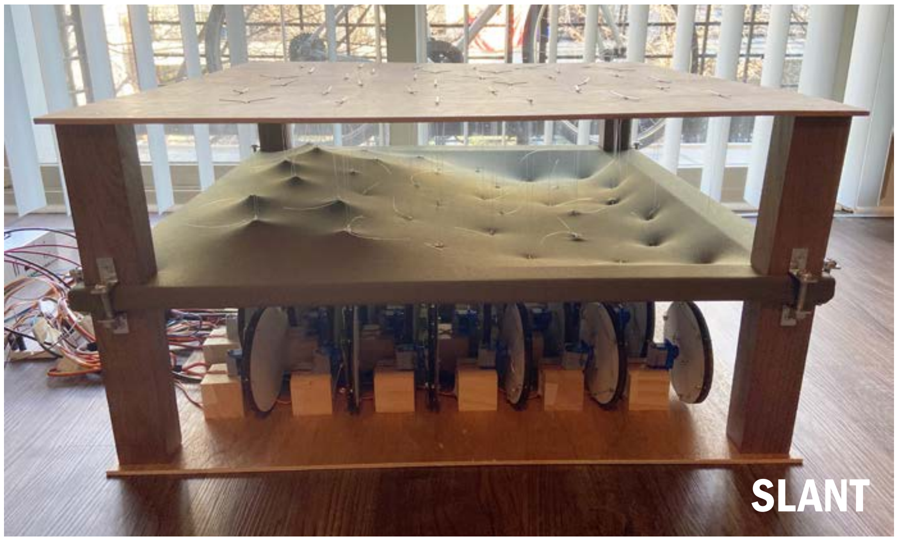


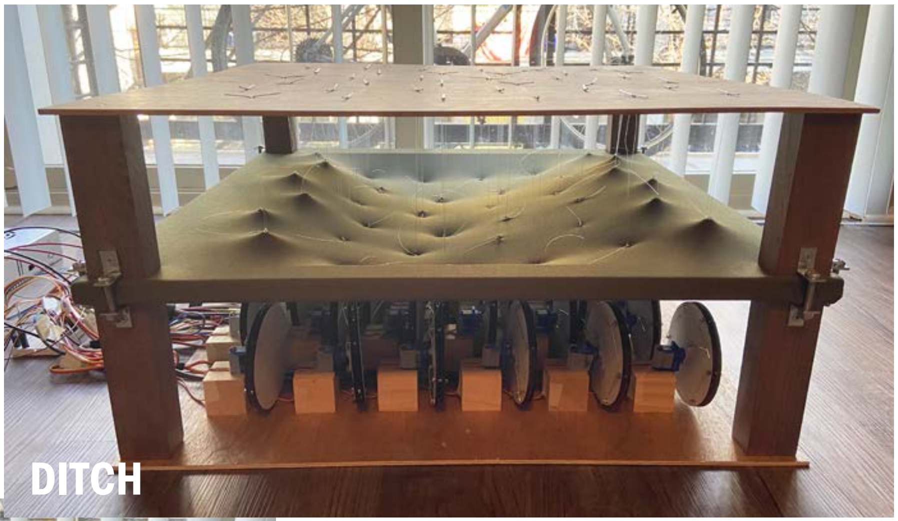
I	ELASTIC STRING
J	COTTER PINS
K	WOOD BLOCKS
L	SERVOS
M	JUMPER WIRES
N	BREADBOARD
O	BLUETOOTH MODULE
P	POWER SUPPLY WIRES
Q	ARDUINO BOARD
R	POWER SUPPLY

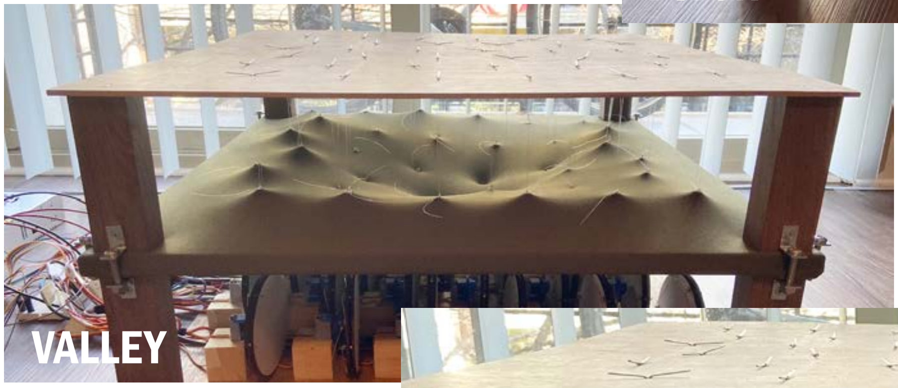


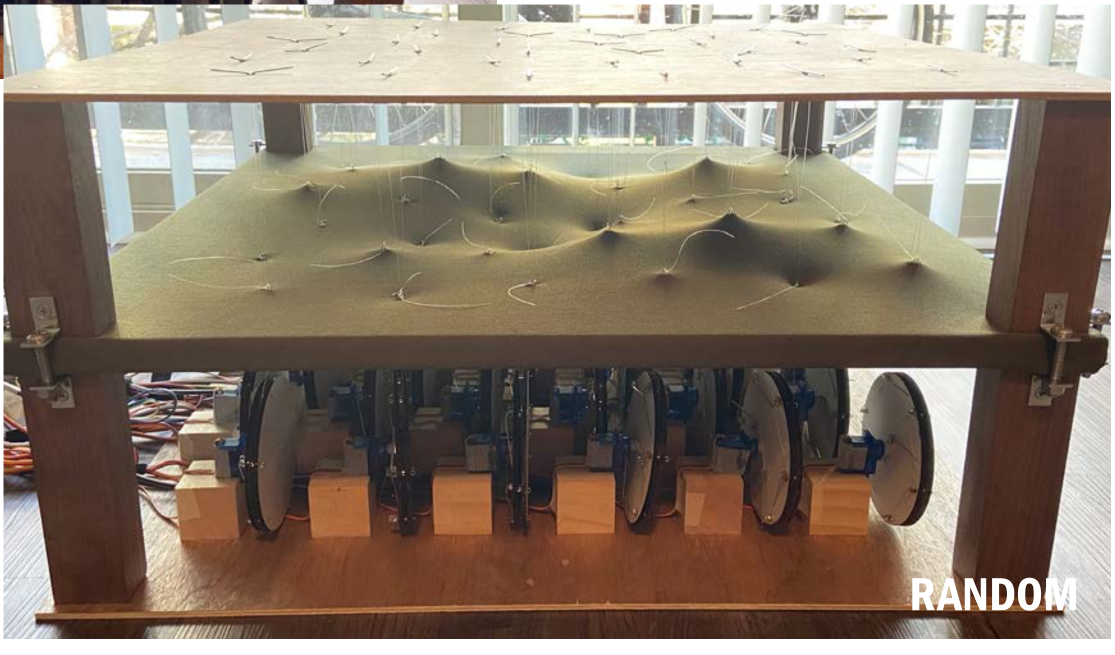
CONFIGURATIONS

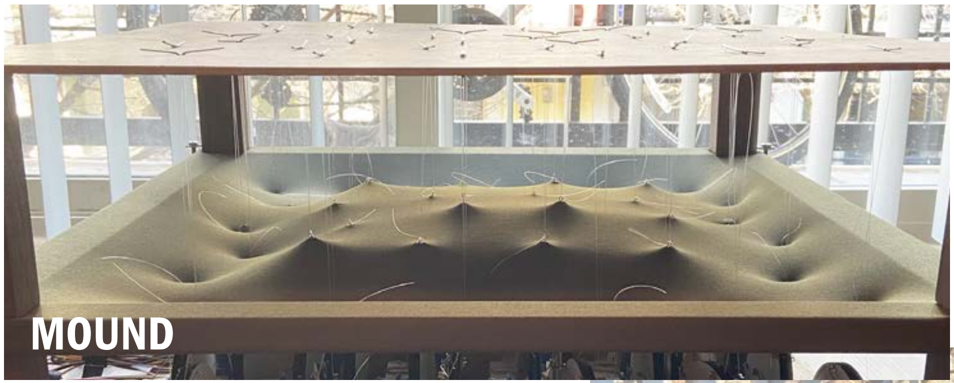
these are the preset configurations. the wave is a multi-step configuration. more moving configurations like the wave should be explored. I see potential with flood control using a moving wave configuration that reduces incoming wave force.

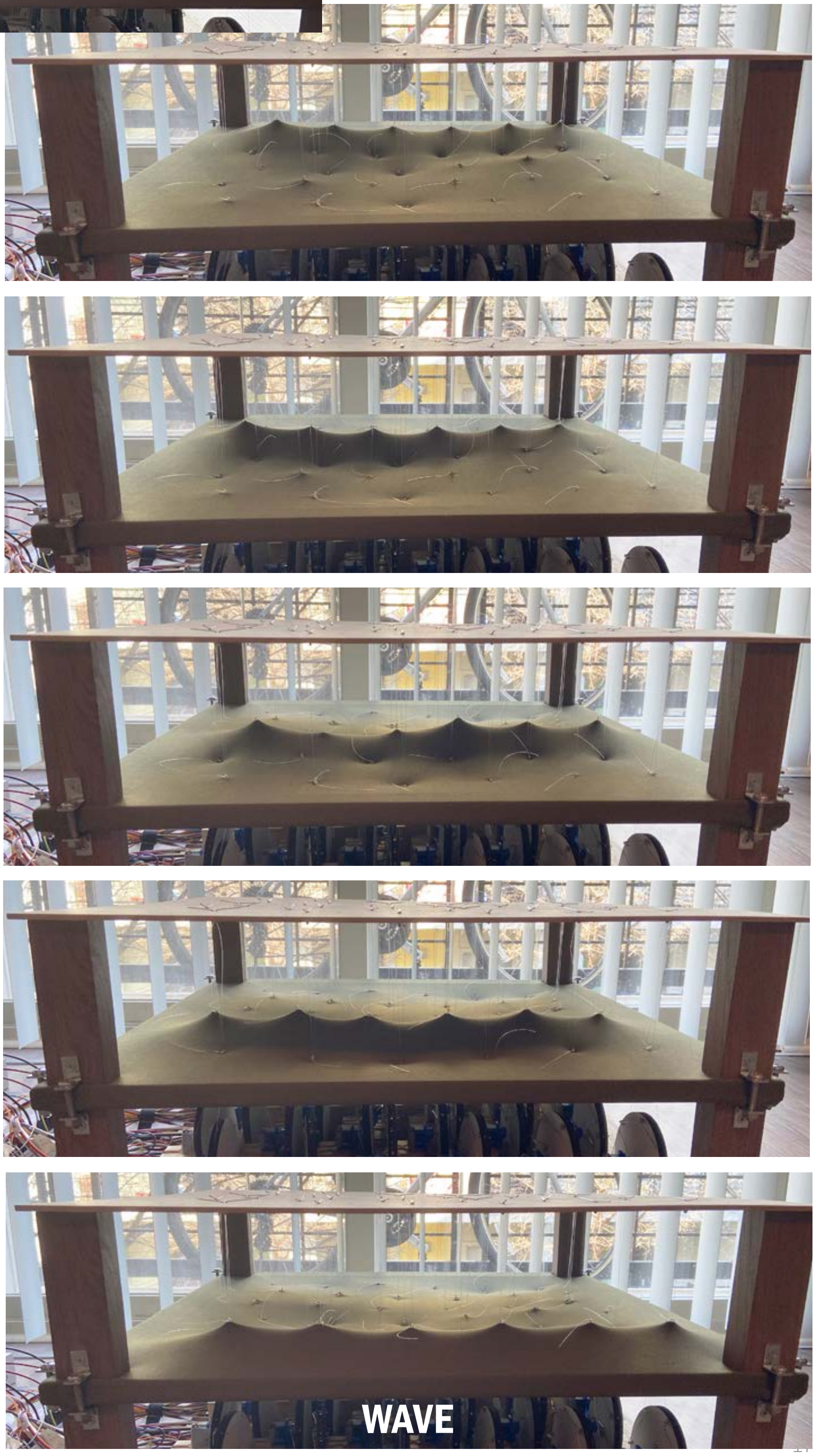

SLANT


DITCH


VALLEY

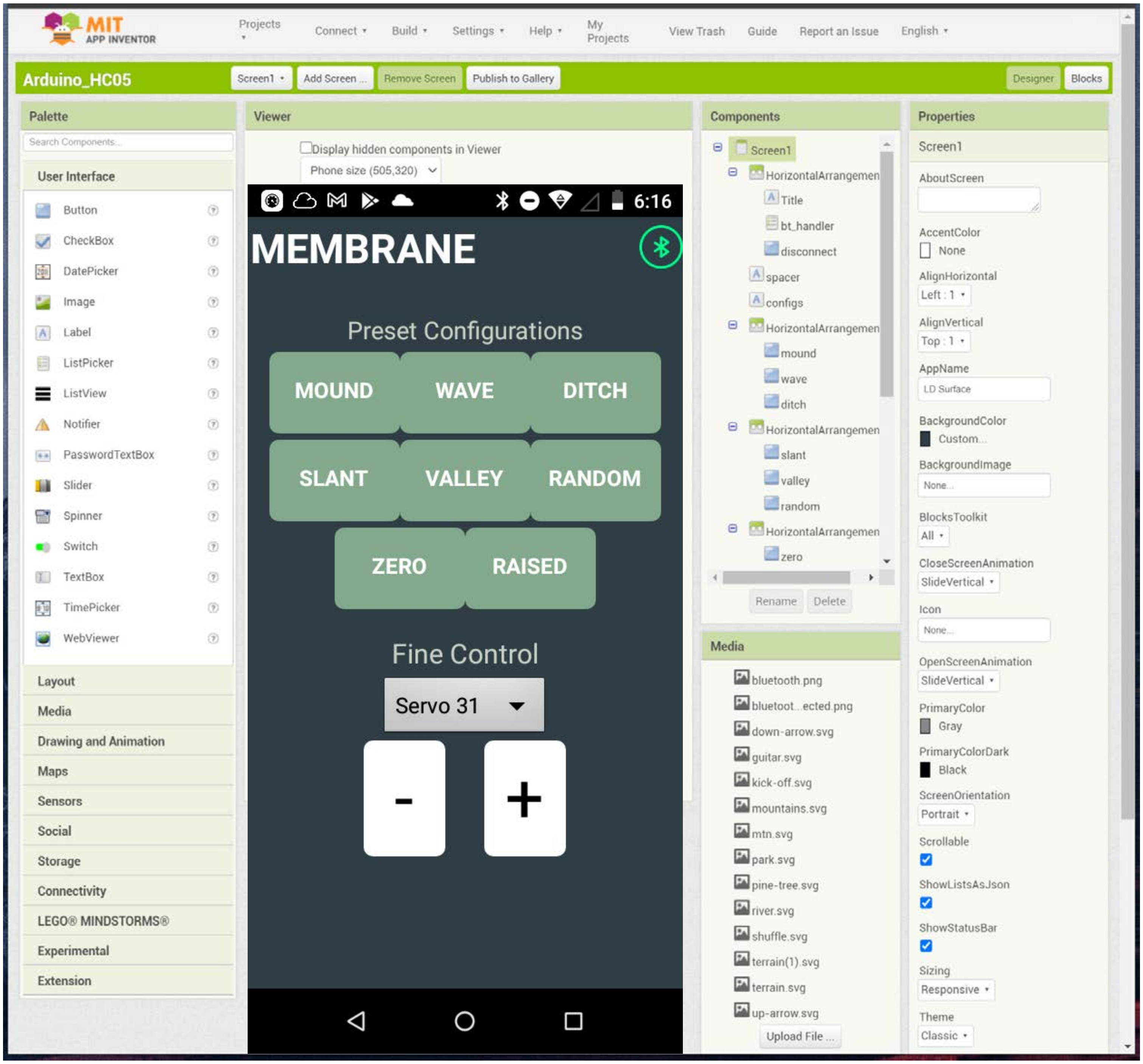

RANDOM


MOUND


WAVE

THE APP

MIT App Inventor was used to create the membrane app. it was the easiest and quickest way i could find. the technology isn't great since it's from around 2006, but it got the job done.



AboutScreen

AccentColor: None

AlignHorizontal: Left : 1

AlignVertical: Top : 1

AppName: LD Surface

BackgroundColor: Custom...

BackgroundImage: None...

BlocksToolkit: All

CloseScreenAnimation: SlideVertical

Icon: None...

OpenScreenAnimation: SlideVertical

PrimaryColor: Gray

PrimaryColorDark: Black

ScreenOrientation: Portrait

Scrollable: ✓

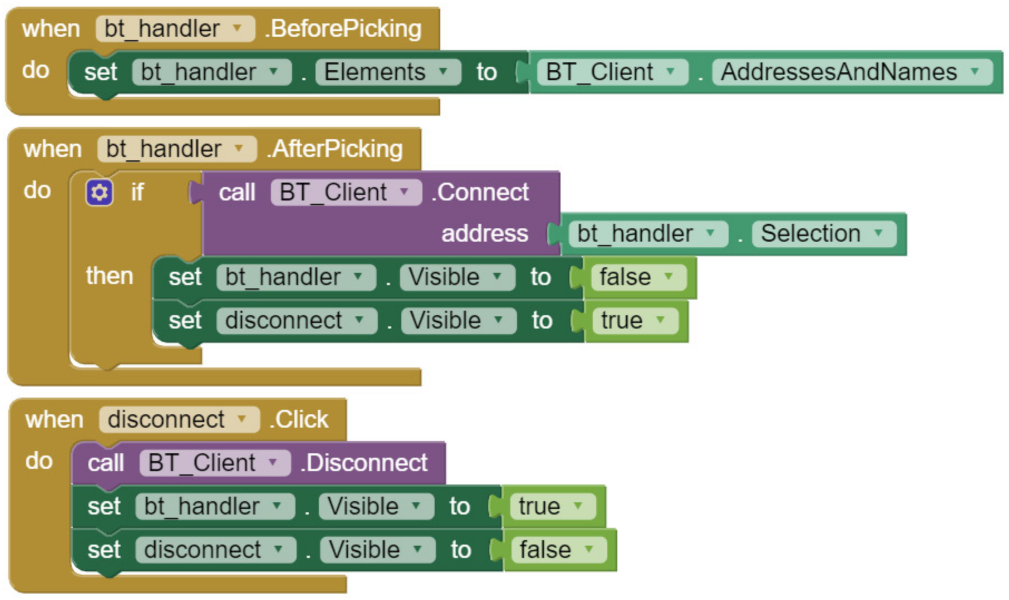
ShowListsAsJson: ✓

ShowStatusBar: ✓

Sizing: Responsive

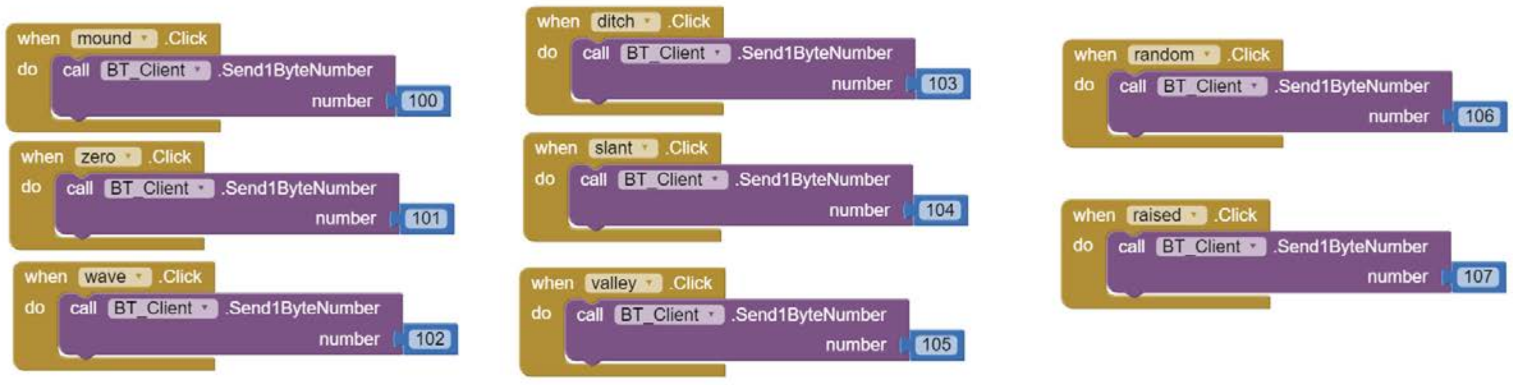
Theme: Classic

there are two ways to use the app. the preset configurations are a quick way to get a design in place. the fine control portion allows each servo to be controlled individually. you select the servo number and can push the plus or minus buttons to move the surface up or down.

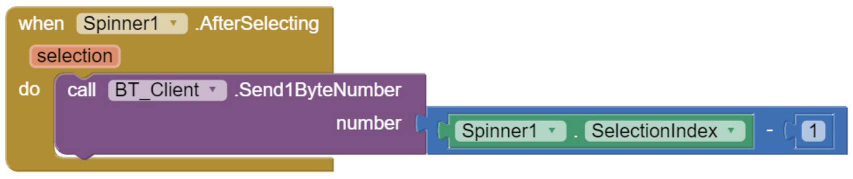


```
when bt_handler .BeforePicking
do  set bt_handler . Elements to BT_Client . AddressesAndNames

when bt_handler .AfterPicking
do  if   call BT_Client .Connect
              address  bt_handler . Selection
    then set bt_handler . Visible to false
         set disconnect . Visible to true

when disconnect .Click
do  call BT_Client .Disconnect
    set bt_handler . Visible to true
    set disconnect . Visible to false
```

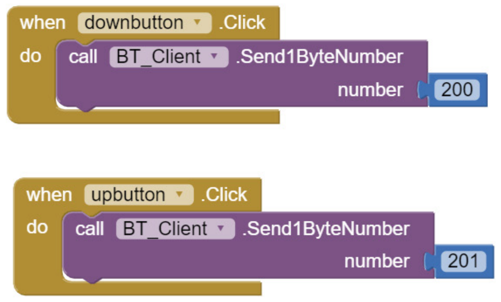
summary:
block 1: gets all of the available bluetooth connections and adds them to the list you can pick from.
block 2: establishes the connection between the bluetooth signal and the phone.
block 3: handles bluetooth disconnection.



```
when mound .Click
do  call BT_Client .Send1ByteNumber
                   number 100

when zero .Click
do  call BT_Client .Send1ByteNumber
                   number 101

when wave .Click
do  call BT_Client .Send1ByteNumber
                   number 102

when ditch .Click
do  call BT_Client .Send1ByteNumber
                   number 103

when slant .Click
do  call BT_Client .Send1ByteNumber
                   number 104

when valley .Click
do  call BT_Client .Send1ByteNumber
                   number 105

when random .Click
do  call BT_Client .Send1ByteNumber
                   number 106

when raised .Click
do  call BT_Client .Send1ByteNumber
                   number 107
```



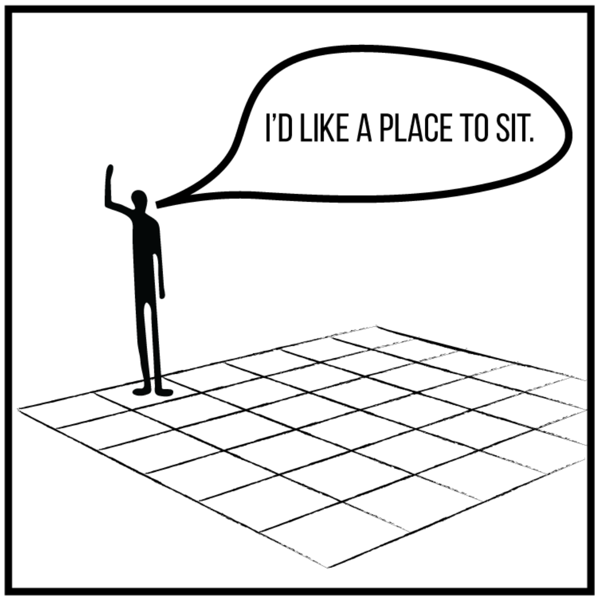
```
when Spinner1 .AfterSelecting
 selection
do  call BT_Client .Send1ByteNumber
                   number  Spinner1 . SelectionIndex - 1
```

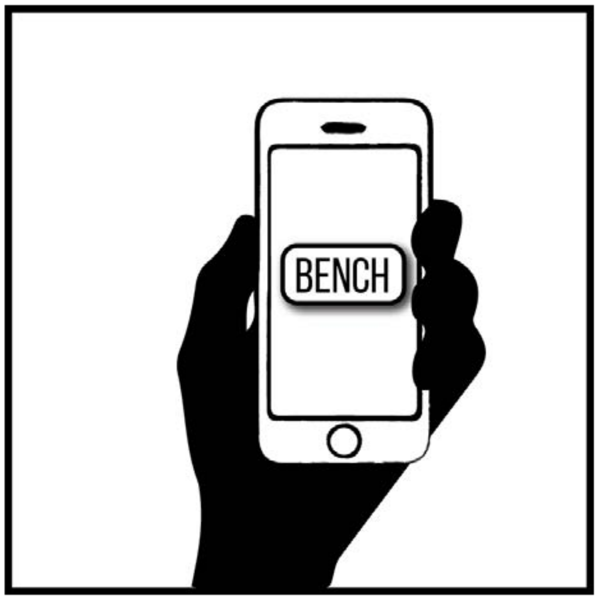


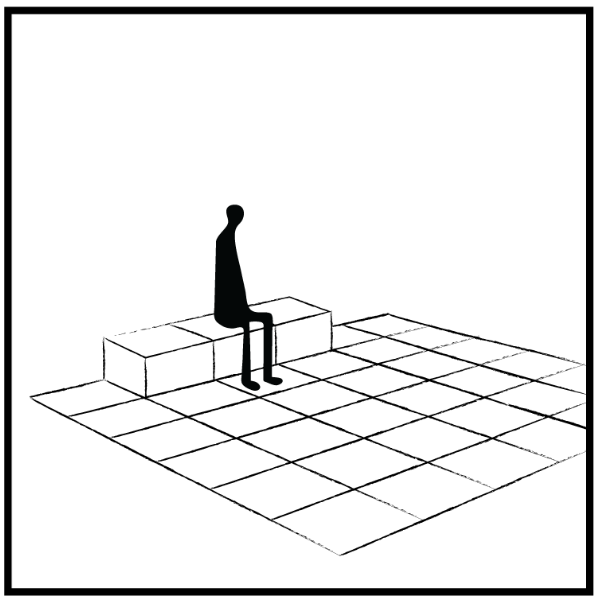
```
when downbutton .Click
do  call BT_Client .Send1ByteNumber
                   number 200

when upbutton .Click
do  call BT_Client .Send1ByteNumber
                   number 201
```

summary:
block set 1: when you click on each configuration it sends the corresponding number through the bluetooth to the arduino
block 2: the spinner is the drop down menu. when you select a servo to modify it sends that selection to the arduino.
block 3: when you click the down button it sends the 200 code to the arduino. to move the selected servo down.
block 4: when you click the up button it sends the 201 code to the arduino. to move the selected servo up.

how the app can also work with prototype 1







SUMMARY

prototype 2 was directly created from the in depth analysis of prototype 1. I tried to take the issues with prototype 1 and solve them with the second. for example, the range of motion was small in prototype 1 so the entire purpose of using the wheel attachments in prototype 2 is to get about a 4" range of movement.

the part I am most happy with on prototype 2 is being able to use an app to modify it. I think people can more easily understand the possibilities with the surface because they can relate to how apps work. I am also happy with the size of the model and that is overall more cleanly built than prototype 1. on this note though, the use of the earring rods and backs is probably the worst part about the project. they seemed to hold tight when static, but when enough force is applied they don't perform well. bending the rods back on each circle was also a poor decision because the backs don't work well with bent rods. the securing of the servo arms with the earrings rods and backs is loose and this is visible with crooked circles. the earring rods and backs were used because something needed to piece cardboard and fit through the servo arm holes. the wheels needed to be cardboard because they needed to be lightweight. next time an entirely different connector should be used. using the elastic thread and non-securely held together wheels creates a fairly delicate prototype. it is much more fragile than prototype 1 was.

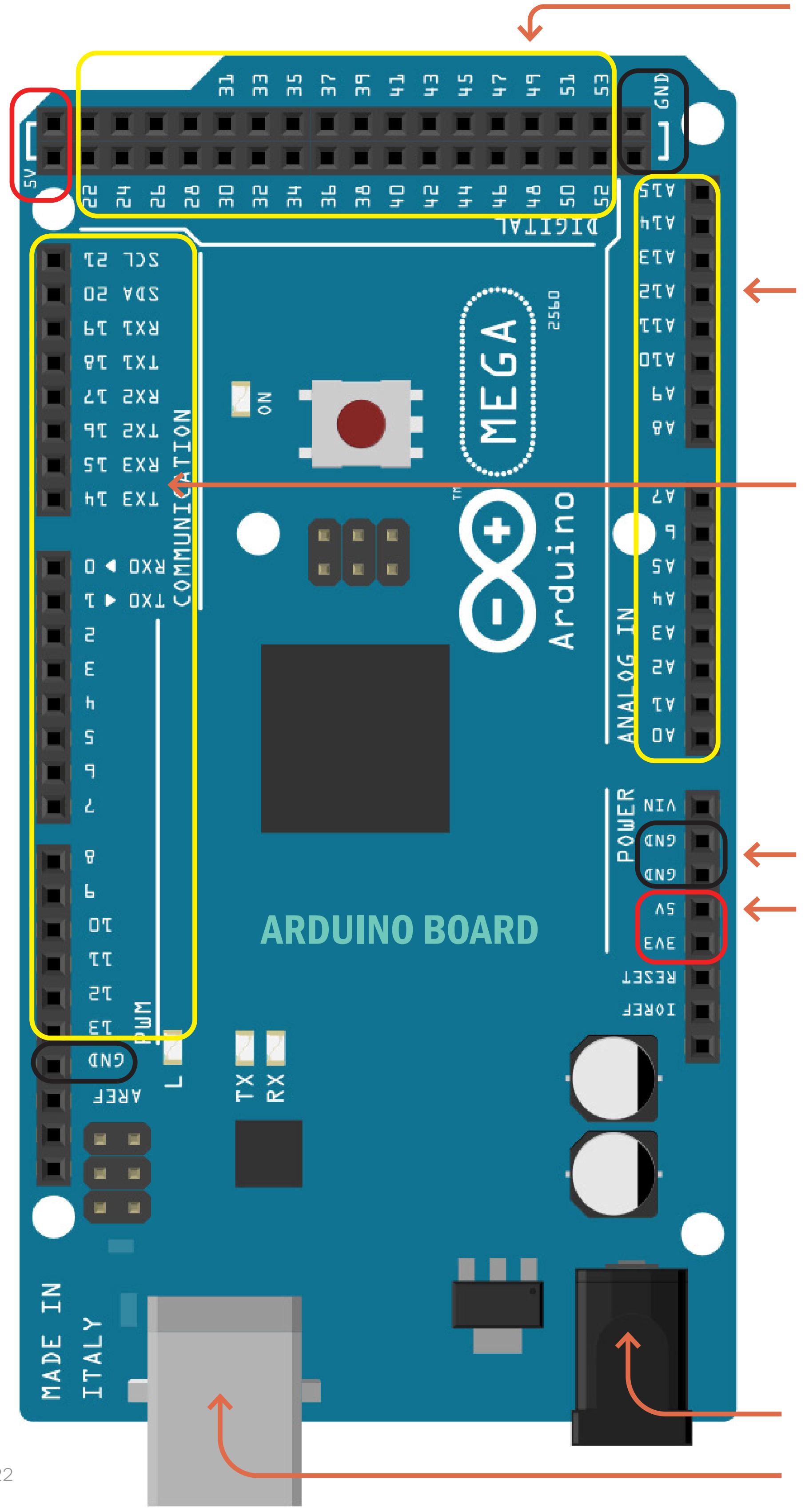
the most challenging part of making prototype 2 was figuring out how to power it. I didn't know increasing from 25 servos to 36 would require extra power otherwise I wouldn't have done it. however, during the making of prototype 1 I went down a month-long rabbit hole of learning how to add additional power because I thought it would need it. turned out prototype 1 didn't need extra power, but that knowledge and hardware came in handy when it was Thursday, the day before the project deadline and the surface wasn't cooperating. actually, the surface went rogue and was doing it's own version of the configurations because of the lack of power. hilariously it is probably the exact nightmare people have about this type of technology. I admit it was slightly frightening in the moment when I didn't know what was happening.

IMPROVEMENTS FROM PROTOTYPE 1

- laser cutter was used for more precision over 3D printing and hand work.
- more range of movement was introduced.
- the model is overall more cleanly built.
- the model is scaled up to a larger size
- more servos were included for better resolution.
- user interaction was added.
- the continuous topo method was included

ARDUINO

arduino: an open-source hardware and software company. arduino makes several different microcontrollers. a microcontroller is essentially a mini-computer that allows us to control the prototype. shown here is the Arduino Mega, but in the physical prototype I use an off-brand board.



digital numbered pins - these are for input wires. there are 53 digital pins on this board. we will be using 25 or 36 of these for the servos and 1 for controlling the water level sensor power.

analog numbered pins - these are for input wires. there are 15 analog pins on this board. we will be using 1 to provide input to the water level sensor.

TX and RX pins are for receiving and transmitting data.

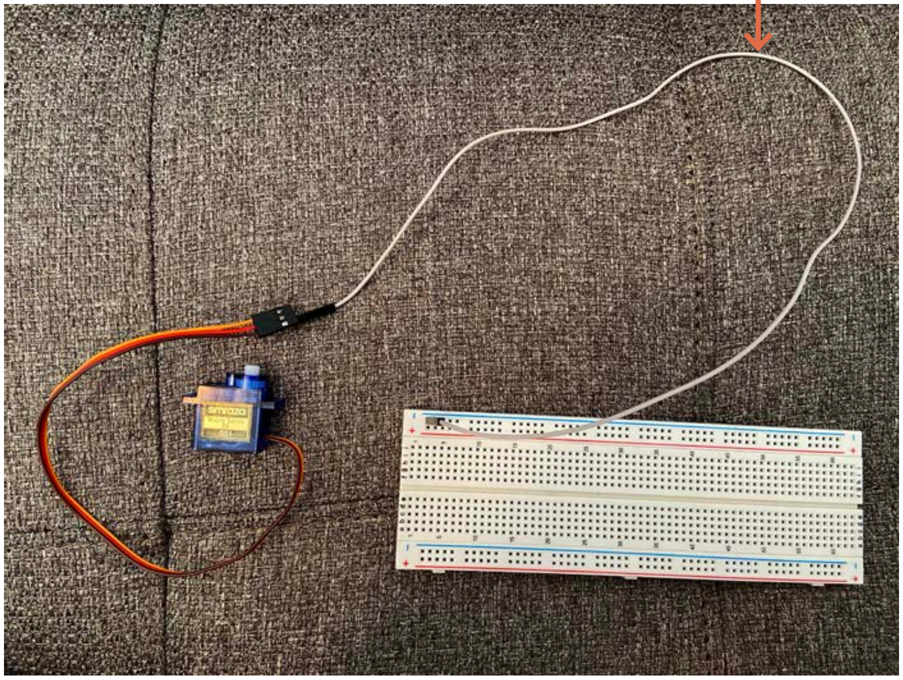
anywhere on the board that says GND is where we can connect the ground wires. this board has 5 pins.

anywhere on the board that says 5V or 3V3 is where we can connect power wires. this board has 4 pins. 3V3 = 3.3 volts.

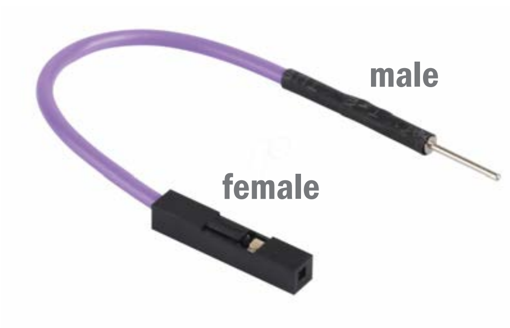
connection to wall outlet

connection to computer

in the schematic diagrams on the following pages servos are shown connected directly to the breadboard but I added additional male to male jumper wires as extension cords in the physical prototype.



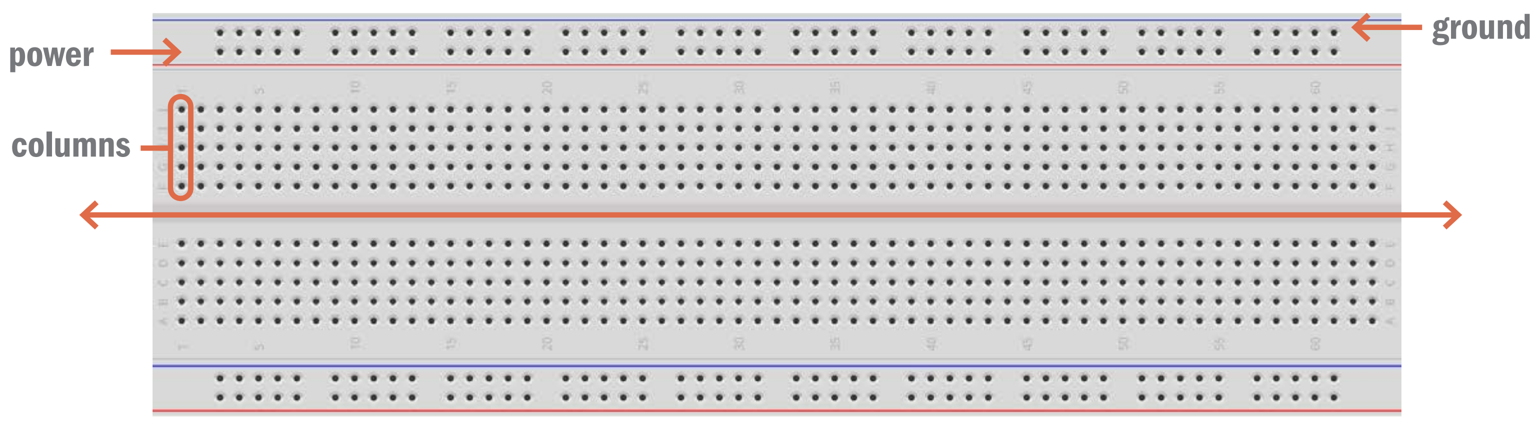
JUMPER WIRES



jumper wires connect everything to the breadboard and the bread board to the arduino.

BREADBOARD

a breadboard acts like a power strip does. it allows us to plug more into the arduino board.



🟦	ground (-)
🟥	power (+)

- all of the in holes in each column are connected, but not across the middle divider. this board has 2 sets of 63 columns. all of the holes in the power row are connected and all of the holes in the ground row are connected.
- the breadboard is split across the middle. one side does not connect to the other without additional components not used in this prototype.

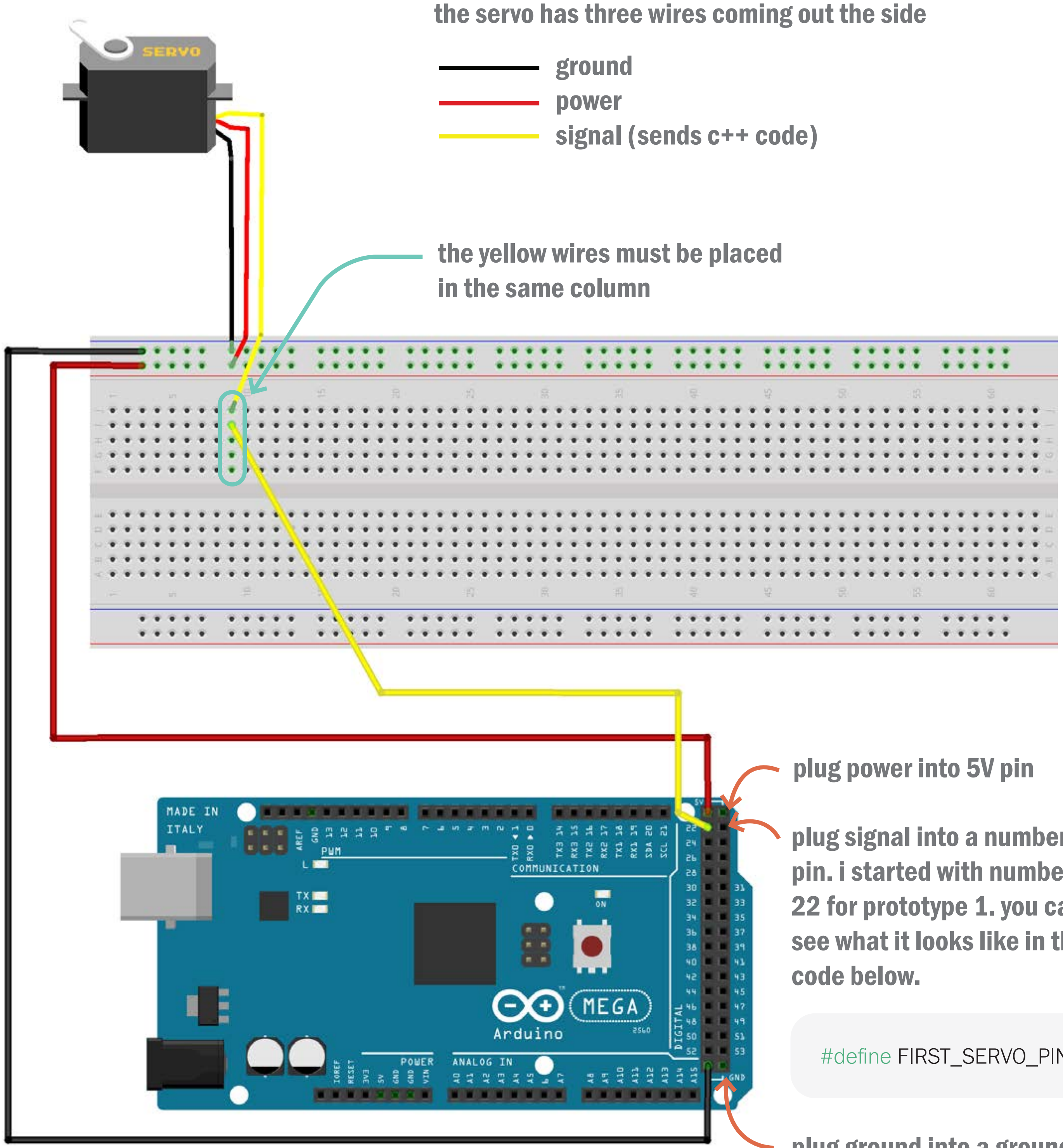
SCHEMATICS

STEP 1: CONNECTING A SERVO MOTOR

connect each servo wire to the breadboard and then connect those breadboard slots to the arduino.

the servo has three wires coming out the side

— ground
— power
— signal (sends c++ code)



the yellow wires must be placed in the same column

plug power into 5V pin

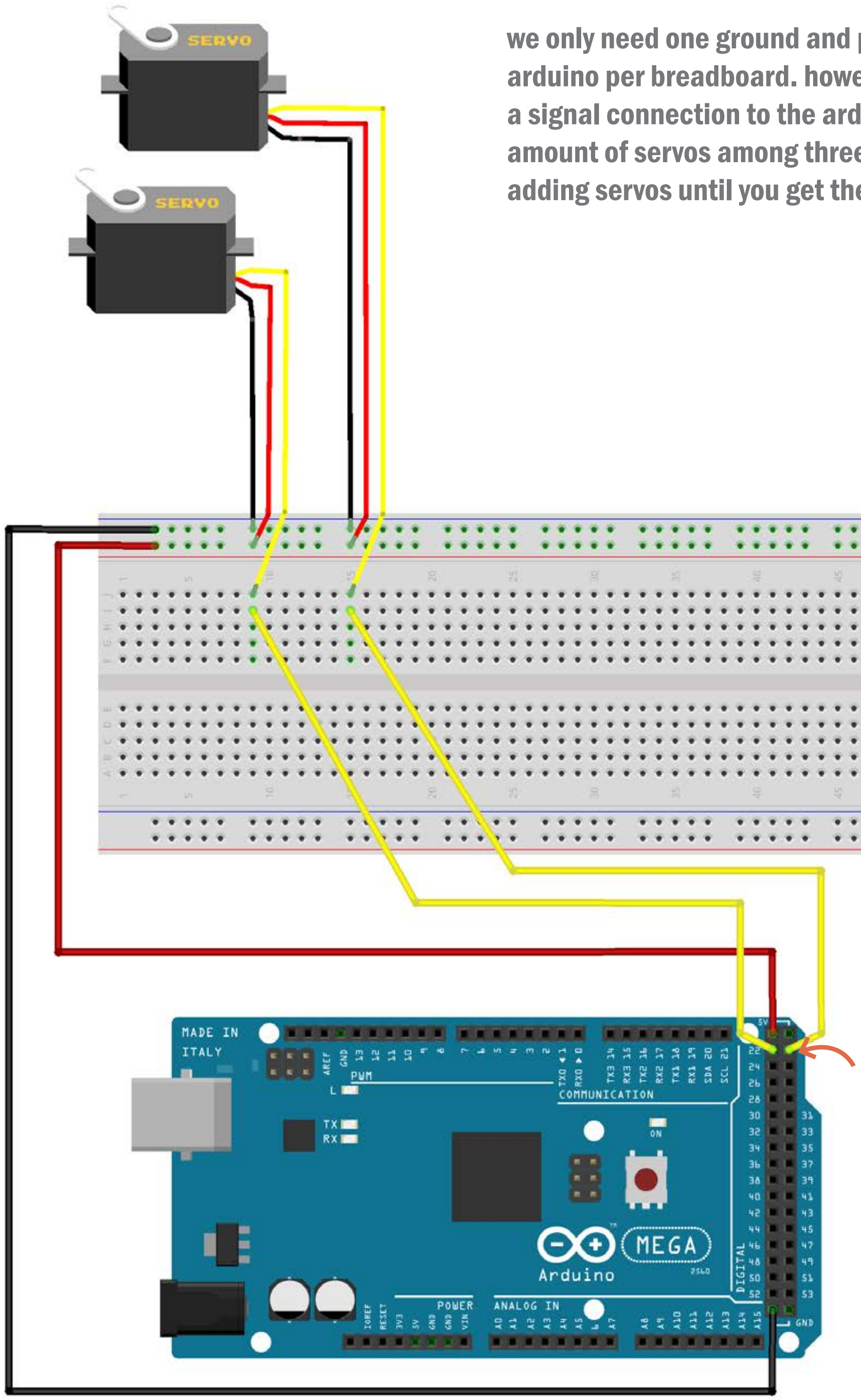
plug signal into a numbered pin. i started with number 22 for prototype 1. you can see what it looks like in the code below.

```
#define FIRST_SERVO_PIN 22
```

plug ground into a ground pin

STEP 2: CONNECTING MORE SERVO MOTORS

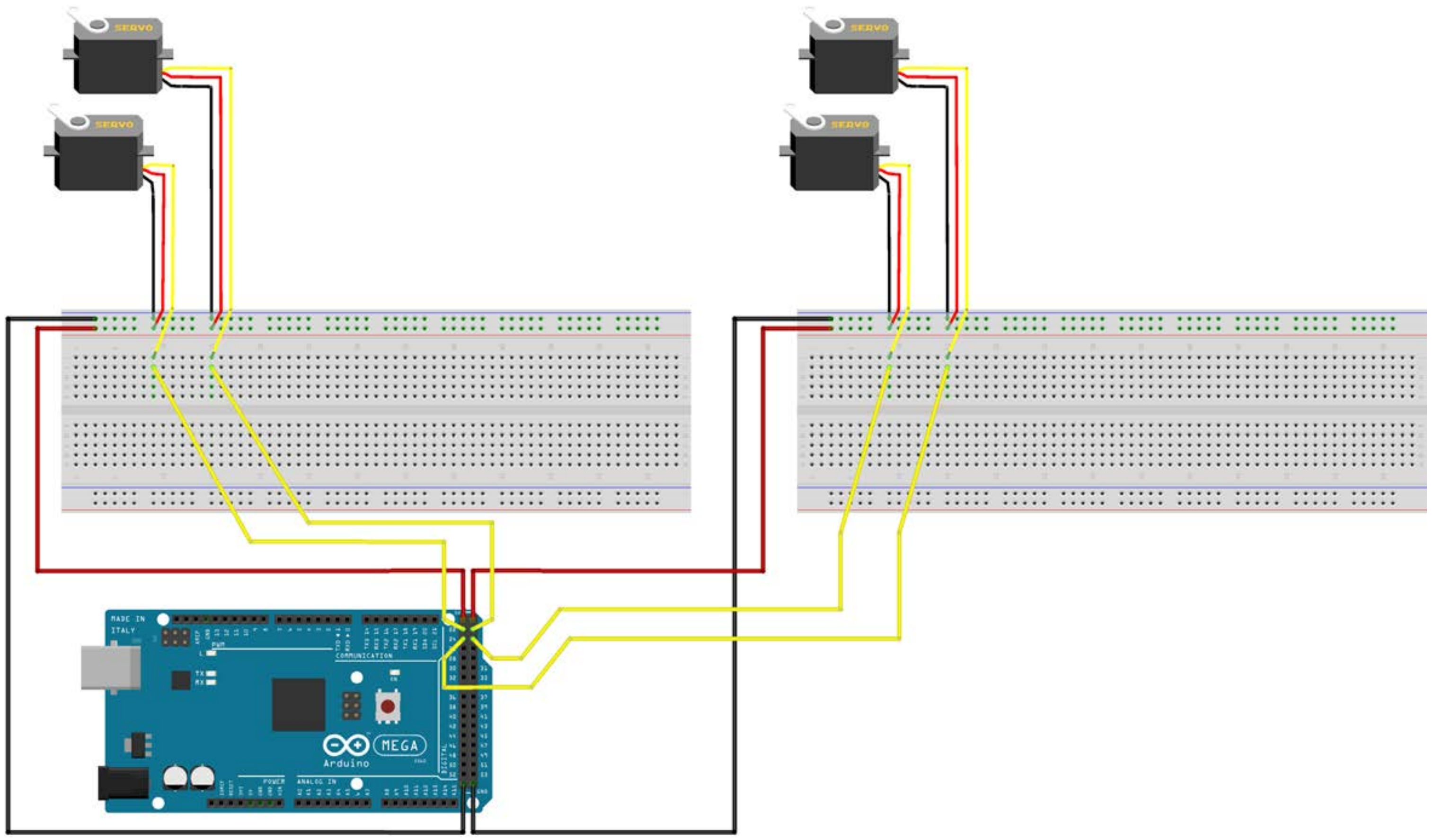
we only need one ground and power connection to the arduino per breadboard. however, each servo still needs a signal connection to the arduino. i divided the total amount of servos among three breadboards. so keep adding servos until you get there.



plug the 2nd signal wire into the next numbered pin. in this case it is pin 23, which isn't numbered on this board.

STEP 3: CONNECTING MORE BREADBOARDS

to add a second breadboard we need to add another ground and power connection to the arduino.

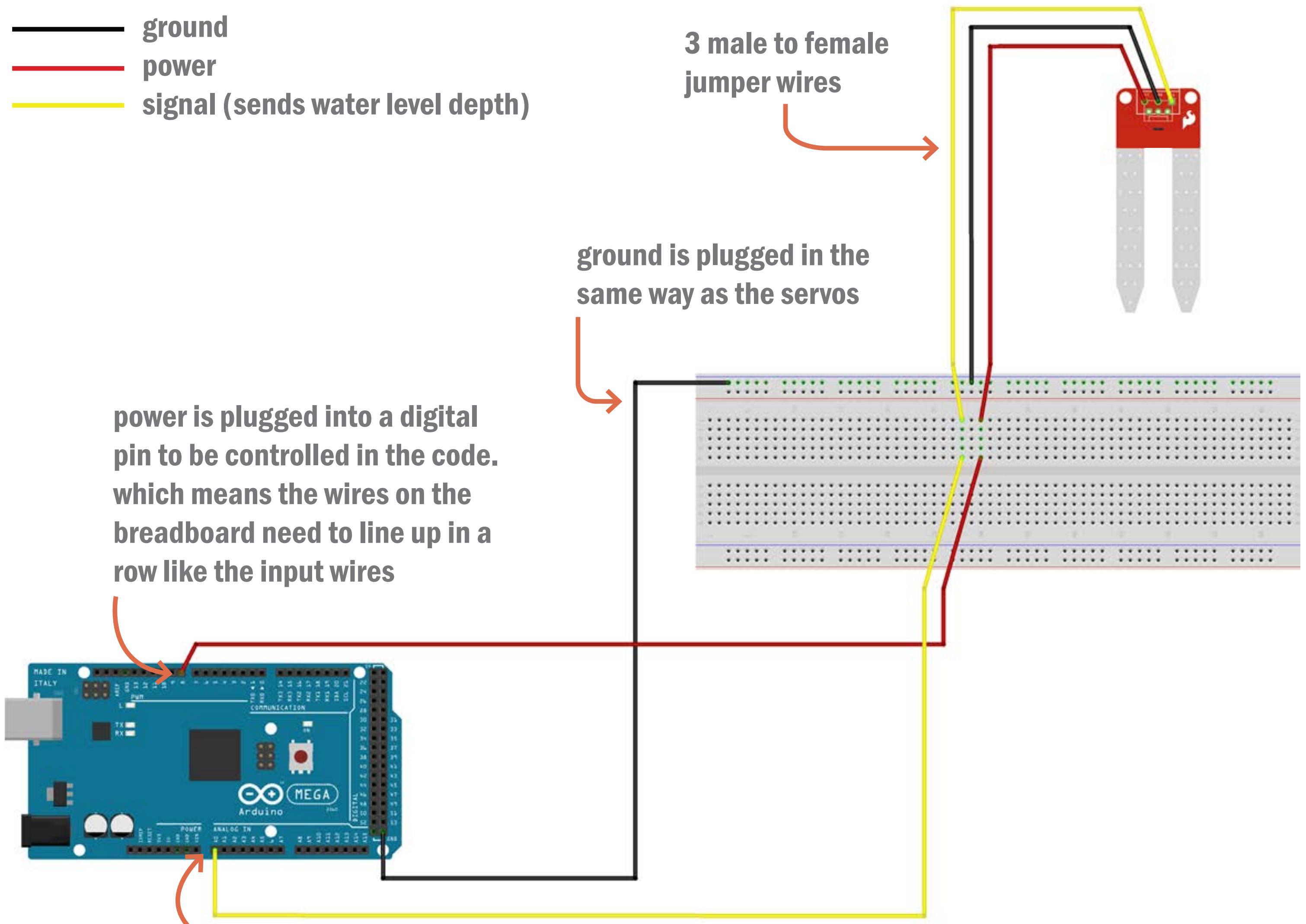


ground
power
signal (sends water level depth)

3 male to female jumper wires

ground is plugged in the same way as the servos

power is plugged into a digital pin to be controlled in the code. which means the wires on the breadboard need to line up in a row like the input wires



signal is plugged into an analog pin because it provides a range of depth levels.

PROTOTYPE 1: CONNECTING THE WATER LEVEL SENSOR

after placing 25 servos and 3 breadboards we can add the water level sensor. it also has power, ground, and signal wires, but they don't come with the sensor. so male to female jumper wires are used to connect the sensor to the breadboard.

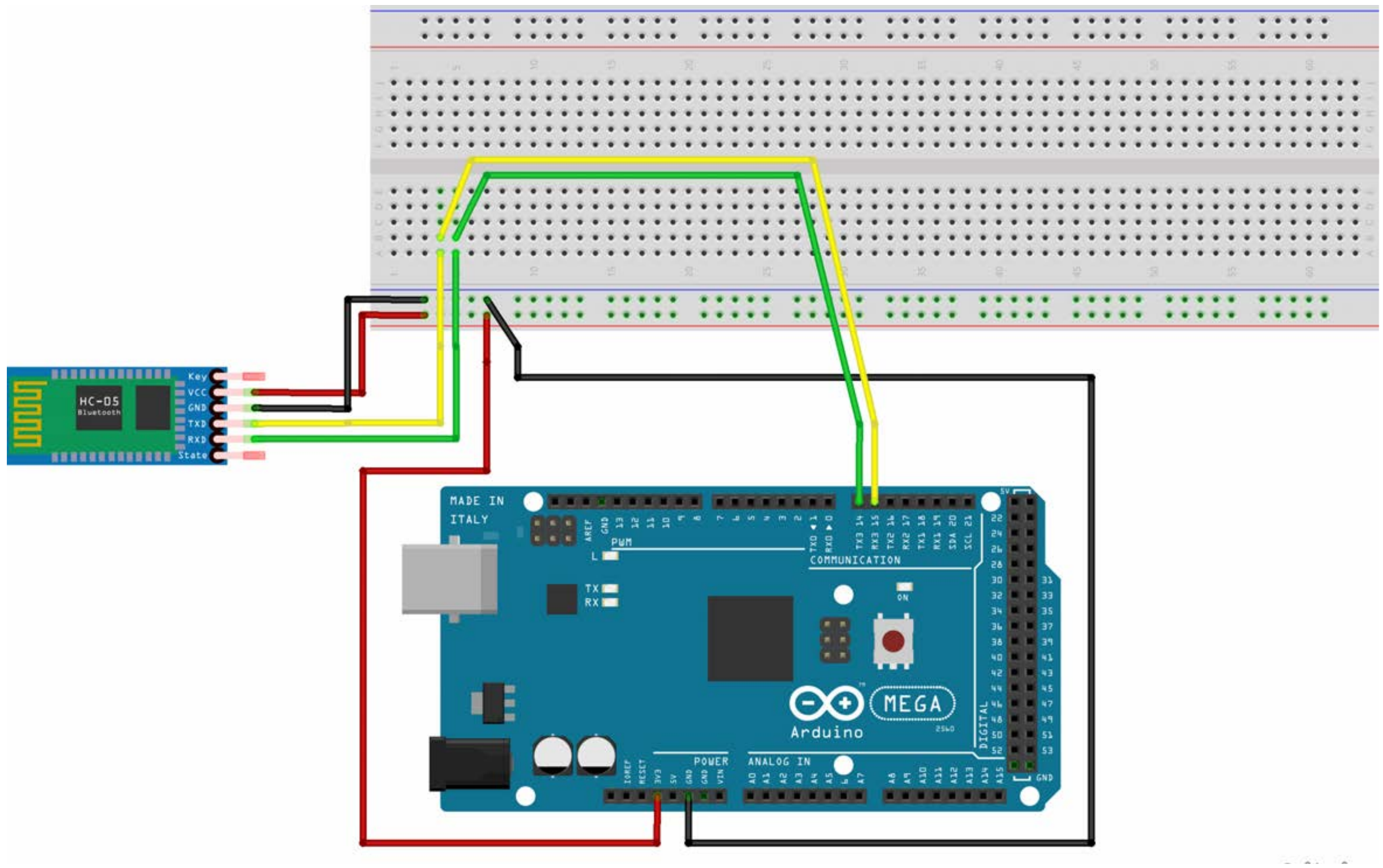
digital pins: only have two values (HIGH and LOW). the water level sensor power is plugged into a digital pin because we don't want it to have constant power. plugging it into a 5V pin would leave it constantly powered but plugging it into a digital pin allows us to turn the power on and off when needed because we can send 5V to it with the code.

analog pins: can take on any number of values. the water level sensor needs to have its signal wire attached to an analog pin because it provides a range of depth levels.

PROTOTYPE 2: CONNECTING THE BLUETOOTH MODULE

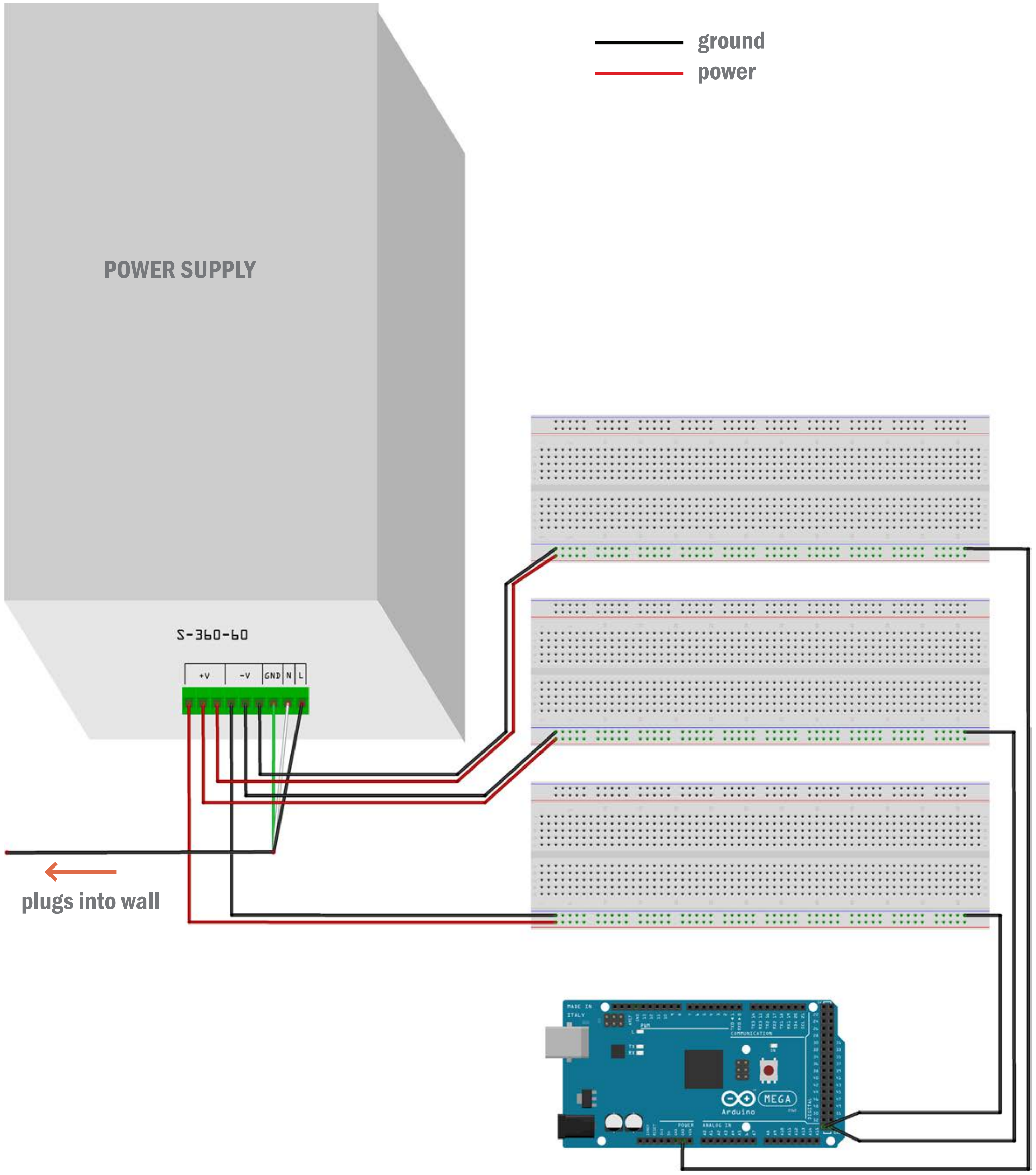
4 male to female jumper wires will be used to connect the bluetooth module. ground gets plugged in like usual. power (VCC on the bluetooth module) gets plugged into the 3V3 pin this time. TXD stands for transmit and RXD stands for receive. transmit on the bluetooth module needs to connect to a receive pin on the arduino and receive needs to connect to the transmit on the arduino. this is to send and receive via bluetooth.

⬛ ground
🟥 power
🟨 signal (transmits)
🟩 signal (receives)



PROTOTYPE 2: CONNECTING THE POWER SUPPLY

36 servos need a lot of power to operate so an additional power supply was required. the power supply gets plugged into the wall. ground and power from the power supply connects to each breadboard. and a ground from each breadboard connects to the arduino if this hasn't already been done in another step. connecting the grounds is the most important step. the power supply has enough current to kill you.



CODE
PROTOTYPE 1

i broke the code down into 8 sections. there is a summary with each section to help explain what is happening.

## [1] - includes

```
#include <Servo.h>
```

summary: the first line of code is telling the program to use functions included in the Arduino Servo Library. this includes all the functions like .write or .attached that we will need to control the servo motors.

## [2] - constants

```
#define WATER_SENSOR_POWER_PIN 9
#define WATER_SENSOR_DATA_PIN A0

#define DIM_X 5
#define DIM_Y 5

const unsigned short TOTAL_SERVOS_ATTACHED = DIM_X * DIM_Y;

#define FIRST_SERVO_PIN 22

#define ROT_INIT 0

#define ROT_DELTA 45

#define WATER_LEVEL_THRESHOLD 25
```

summary: constant values (const) and definitions (#define) don’t change. once they are defined here they are set permanently and the compiler or arduino cannot change them. for example, the 5x5 wood block grid does not change.

the first two lines indicate which pins on the arduino the water level sensor is plugged into. digital pin 9 for its power and analog pin A0 for its signal.

the next three lines defines the servo grid and gives us the total number of servos attached (5x5 =25).

we then define that we are plugging in servo 0 to pin 22 on the arduino, servo 1 will go in pin 23, servo 2 in pin 24, etc.

next we define the initial rotation which is the starting point for all of the servo arms. delta refers to the change in movement. here meaning the servo arm can move 45 degrees.

lastly we define that at a read out of 25 on the water level sensor, movement is triggered.

[3] - configurations

```
const float CONFIG_NULL[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0
};

const float CONFIG_MAX[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0
};

const float CONFIG_SHAKE[TOTAL_SERVOS_ATTACHED] = {
  0.1, 0.1, 0.1, 0.1, 0.1,
  0.1, 0.1, 0.1, 0.1, 0.1,
  0.1, 0.1, 0.1, 0.1, 0.1,
  0.1, 0.1, 0.1, 0.1, 0.1,
  0.1, 0.1, 0.1, 0.1, 0.1
};

const float CONFIG_MOUNTAIN[TOTAL_SERVOS_ATTACHED] = {
  1.0, 0.8, 0.6, 0.4, 0.2,
  0.8, 0.7, 0.5, 0.3, 0.1,
  0.6, 0.5, 0.4, 0.3, 0.1,
  0.4, 0.3, 0.3, 0.2, 0.0,
  0.2, 0.1, 0.1, 0.0, 0.0
};

const float CONFIG_DIAMOND[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.3, 0.0, 0.0,
  0.0, 0.3, 0.6, 0.3, 0.0,
  0.3, 0.6, 1.0, 0.6, 0.3,
  0.0, 0.3, 0.6, 0.3, 0.0,
  0.0, 0.0, 0.3, 0.0, 0.0
};

const float CONFIG_STEPS[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0,
  0.7, 0.7, 0.7, 0.7, 0.7,
  0.5, 0.5, 0.5, 0.5, 0.5,
  0.2, 0.2, 0.2, 0.2, 0.2,
  0.0, 0.0, 0.0, 0.0, 0.0
};
```

summary: these are the different height configurations of the wood blocks. there are 25 numbers, one for each servo. the numbers are shown as parts of the max rotation of the servo arms. for example, 0.1 = 10% of the 45 degree rotation, 1.0 = 100% of the 45 degree rotation.

the grid setup of the servos runs left to right like this:

```
00--01--02--03--04
05--06--07--08--09
10--11--12--13--14
15--16--17--18--19
20--21--22--23--24
```

the first configuration shows that all blocks are flat.

the second configuration shows all the blocks at max height.

the third configuration works in a series to help shake the blocks down into place. this was added after realizing the blocks were not moving up and down perfectly.

the fourth configuration slopes down from the top left corner.

the fifth configuration creates a diamond effect.

the last configuration on this page looks like stairs and is what will be used when the water level sensor is triggered.

## [3] - configurations (continued)

```
const float CONFIG_INVERT_MOUNTAIN_1[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 0.0, 0.0, 0.0, 1.0,
  1.0, 0.0, 0.0, 0.0, 1.0,
  1.0, 0.0, 0.0, 0.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0
};

const float CONFIG_INVERT_MOUNTAIN_2[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 0.0, 0.0, 0.0, 1.0,
  1.0, 0.0, 1.0, 0.0, 1.0,
  1.0, 0.0, 0.0, 0.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0
};

const float CONFIG_INVERT_MOUNTAIN_3[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 0.5, 0.5, 0.5, 1.0,
  1.0, 0.5, 0.0, 0.5, 1.0,
  1.0, 0.5, 0.5, 0.5, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0
};

const float CONFIG_X[TOTAL_SERVOS_ATTACHED] = {
  1.0, 0.0, 0.0, 0.0, 1.0,
  0.0, 1.0, 0.0, 1.0, 0.0,
  0.0, 0.0, 1.0, 0.0, 0.0,
  0.0, 1.0, 0.0, 1.0, 0.0,
  1.0, 0.0, 0.0, 0.0, 1.0
};
```

summary:
the first three configurations work together in a series to create an inverted mountain.

the fourth configuration is set up in an x formation.

## [4] - variables

```
int water_level_raw = 0;

Servo servos[TOTAL_SERVOS_ATTACHED];
```

summary:

the first variable is named water level raw. here we are creating a variable that will be referenced and changed to a value once the sensor is dipped in water. the second line of code means we are creating an array. the length of the array is the number of servos attached which is 25. an array makes it so we don’t have to have 25 lines of code to change all of the servos.

## [5] - initial functions

```
void InitServos()
{

 for (int i = 0; i < TOTAL_SERVOS_ATTACHED; i++)
 {
   servos[i].attach( i + FIRST_SERVO_PIN );
   servos[i].write( ROT_INIT );
  }
 }

void InitSensors()
{
 pinMode(WATER_SENSOR_POWER_PIN, OUTPUT);
 digitalWrite(WATER_SENSOR_POWER_PIN, LOW);
}
```

summary:
initial functions only run once at the beginning setup.

the first function will loop through all 25 servos, attaching them in consecutive order based on the first pin being 22 (which we defined in section 2 of the code). it will then set the arms to 0.

InitSensors is a function that sets up the arduino to power the water level sensor.

## [6] - runtime functions

```
void SetToConfig(const float * new_config)
{
   for (int i = 0; i < TOTAL_SERVOS_ATTACHED; i++)
   {
     unsigned short new_angle = ROT_INIT + (floor(new_config[i] * ROT_DELTA));
     servos[i].write( new_angle );
     delay(100);
   }
 }

void ShakeServos()
{
  SetToConfig(CONFIG_NULL);
  SetToConfig(CONFIG_SHAKE);
  SetToConfig(CONFIG_NULL);
}

void readWaterSensor() {
  digitalWrite(WATER_SENSOR_POWER_PIN, HIGH);
  delay(10);
  water_level_raw = analogRead(WATER_SENSOR_DATA_PIN);
  digitalWrite(WATER_SENSOR_POWER_PIN, LOW);
}
```

summary:
runtime functions will continually run throughout the process.

the first function is for setting the servos to the configurations. it will loop through 25 times because it is a for loop which runs a set number of times. it takes the starting position of the servo arm and calculates the rotation defined in the configuration. floor just means it rounds the number down to whole number. then it sets each actuator to the new angle. a delay is added between rotating each servo to help with power concerns. the delay is in milliseconds. 100 = 1/10 of a second.

next we create the configuration series to shake the servos. it makes all the blocks flat, all to 0.1, and then all flat again.

the last section turns on the water sensor, reads and saves the data, turns off the sensor, and then sends the reading back.

## [7] - arduino setup

```
void setup() {
  Serial.begin(9600);
  InitSensros();
  InitServos();
 }
```

summary:

serial.begin starts up the arduino. it then runs the initial functions in section 5

## [8] - arduino loop

```
void loop() {
  readWaterSensor();

  if (water_level_raw > WATER_LEVEL_THRESHOLD)
  {
    SetToConfig(CONFIG_STEPS);
    ShakeServos();
  }
  else
  {
    SetToConfig(CONFIG_X);
    delay(2000);
    ShakeServos();
    SetToConfig(CONFIG_INVERT_MOUNTAIN_1);
    delay(2000);
    ShakeServos();
    SetToConfig(CONFIG_INVERT_MOUNTAIN_2);
    delay(2000);
    ShakeServos();
    SetToConfig(CONFIG_INVERT_MOUNTAIN_3);
    delay(2000);
    ShakeServos();
    SetToConfig(CONFIG_STEPS);
    delay(2000);
    ShakeServos();
  }
}
```

summary:
this loop will keep running the entire time arduino is running.

first it checks the water level sensor. if the level is greater than the defined threshold then the stair stepping configuration is activated. followed by the servo shake to settle the blocks back down into place.

if the water level is not above the set threshold it cycles through some other configurations. this would not happen in reality. it is merely a way to show how other configurations would work and how it would transition between them. not all the configurations listed in section 3 of the code are demonstrated here in the final prototype. these are the configurations that worked the best.

## [1] - includes

```
#include <Servo.h>
```

summary: the first line of code is telling the program to use functions included in the Arduino Servo Library. this includes all the functions like .write or .attached that we will need to control the servo motors.

## [2] - constants

```
#define DIM_X 6
#define DIM_Y 6

const unsigned short TOTAL_SERVOS_ATTACHED = DIM_X * DIM_Y;

#define FIRST_SERVO_PIN 18

#define ROT_MIDDLE 1500

#define ROT_DELTA -500
```

summary: constant values (const) and definitions (#define) don't change. once they are defined here they are set permanently and the compiler or arduino cannot change them. for example, the 6x6 servo grid does not change.

the first three lines define the servo grid and gives us the total number of servos attached (6x6 =36).

we then define that we are plugging in servo 0 to pin 18 on the arduino, servo 1 will go in pin 19, servo 2 in pin 20, etc.

next we define the middle of the servo rotation. this number is in microseconds. delta refers to the change in movement. here meaning the servo arm can move 500 microseconds in either direction with a full range of 100 microseconds to 2000 microseconds. Microseconds are more reliable than putting in degrees.

[3] - configurations

```
const float CONFIG_FLAT[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};

const float CONFIG_MAX[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0
};

const float CONFIG_MIN[TOTAL_SERVOS_ATTACHED] = {
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
  -1.0, -1.0, -1.0, -1.0, -1.0, -1.0
};

const float CONFIG_MOUND[TOTAL_SERVOS_ATTACHED] = {
  -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
  -0.8, 0.2, 0.4, 0.4, 0.4, -0.8,
  -0.8, 0.1, 0.5, 1.0, 0.3, -0.8,
  -0.8, 0.2, 0.3, 0.3, 0.1, -0.8,
  -0.8, 0.2, 0.3, 0.2, 0.2, -0.8,
  -0.8, -0.8, -0.8, -0.8, -0.8, -0.8
};

const float CONFIG_SLOPE[TOTAL_SERVOS_ATTACHED] = {
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
  -1.0, -0.5, -0.2, 0.2, 0.6, 1.0,
};
```

summary:
these are the different height configurations of the membrane. there are 36 numbers, one for each servo. 1.0 is the max height, 0.0 is flat, and -1.0 is the minimum height.

the grid setup of the servos runs left to right like this:

```
00--01--02--03--04-05
06--07--08--09--10-11
12--13--14--15--16-17
18--19--20--21--22-23
24--25--26--27--28-29
30--31--32--33--34--35
```

the configurations are as follows:
1. flat membrane
2. every node at max height
3. every node at min height
4. mound
5. slope from max height on one side to min height on other.

## [3] - configurations (continued)

```
const float CONFIG_DITCH[TOTAL_SERVOS_ATTACHED] = {
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
  1.0, -0.1, -0.4, -1.0, -0.5, 0.5,
};

const float CONFIG_RANDOM[TOTAL_SERVOS_ATTACHED] = {
  0.5, -0.1, 0.4, 0.3, 0.0, 0.0,
  0.6, 1.0, 0.6, 0.2, 0.1, 0.0,
  0.0, 0.1, 0.2, -0.3, -0.1, 0.1,
  0.1, 0.3, -0.5, -0.2, 0.1, 0.3,
  0.3, 0.8, 0.2, 0.6, 1.0, 0.2,
  0.4, 0.2, -0.2, 0.3, 0.1, 0.0
};

const float CONFIG_VALLEY[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.7, 0.1, 0.2, 0.0, 0.0, 1.0,
  0.5, 0.3, -1.0, -1.0, 0.0, 1.0,
  0.9, -0.3, -1.0, -0.3, 0.2, 1.0,
  1.0, 0.1, 0.2, 0.4, 0.5, 1.0,
  1.0, 0.9, 0.8, 1.0, 1.0, 1.0
};
```

summary:
the configurations are as follows:
1. ditch.
2. random
3. valley at center

[3] - configurations (continued)

```
const float CONFIG_WAVE_1[TOTAL_SERVOS_ATTACHED] = {
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
const float CONFIG_WAVE_2[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
const float CONFIG_WAVE_3[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
const float CONFIG_WAVE_4[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
const float CONFIG_WAVE_5[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
const float CONFIG_WAVE_6[TOTAL_SERVOS_ATTACHED] = {
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  1.0, 1.0, 1.0, 1.0, 1.0, 1.0
};
```

summary:
this is one multi-part configuration called the wave, where each row goes up and then down before going to the next row.

## [4] - variables

```
int incoming_command;

int selected_servo;

Servo servos[TOTAL_SERVOS_ATTACHED];
```

summary:

all the commands that come in from the bluetooth module come in the form of numbers (0-255). the incoming_command is where commands are saved from the app/phone.

selected_servo is correlated with the drop down menu in the app where you can choose an individual servo to modify.

the third line of code means we are creating an array. the length of the array is the number of servos attached which is 36. an array makes it so we don’t have to have 36 lines of code to change all of the servos.

## [5] - initial functions

```
void InitServos()
{

 for (int i = 0; i < TOTAL_SERVOS_ATTACHED; i++)
 {
  byte attachment_pin = i + FIRST_SERVO_PIN;
  servos[i].attach( attachment_pin, 1000, 2000 );
  servos[i].writeMicroseconds( ROT_MIDDLE );

  delay(50);
 }
}

void InitBluetooth()
{
 Serial3.begin(9600);
}
```

summary:
initial functions only run once at the beginning setup.

the first function will loop through all 36 servos, attaching them in consecutive order based on the first pin being 18 (which we defined in section 2 of the code). 1000 and 2000 are then set as the range of movement on each servo.

next the arms are set to middle rotation. there will be a delay of 50 microseconds between each servo attachment.

InitBluetooth is a function that initializes the bluetooth communication. Serial 3 is the TX and RX pins it is plugged into on the arduino. 9600 is the rate (9600 bits/second) the data is exchanged.

## [6] - runtime functions

```
void SetToConfig(const float * new_config)
{
  for (int i = 0; i < TOTAL_SERVOS_ATTACHED; i++)
  {
  unsigned short new_angle = ROT_MIDDLE + (floor(new_config[i] * ROT_DELTA));
  servos[i].writeMicroseconds( new_angle );
  delay(100);
  }
 }

void SetSpecifiedServo(const char servo_index, const float direction)
{
 if (10 < servos[servo_index].read() && servos[servo_index].read() < 170)
 {
  unsigned short new_angle = servos[servo_index].read() + (10 * direction);
  servos[servo_index].write( new_angle );
  delay(100);
 }
}
```

summary:
runtime functions will continually run throughout the process.

the first function is for setting the servos to the configurations. it will loop through 36 times because it is a for loop which runs a set number of times. it takes the starting position of the servo arm and calculates the rotation defined in the configuration. floor just means it rounds the number down to whole number. then it sets each actuator to the new angle. a delay is added between rotating each servo to help with power concerns. the delay is in microseconds.

the second function is for controlling the individual servos. 10 is the increment of degrees it goes up or down.

## [7] - arduino setup

```
void setup() {
 Serial.begin(9600);
 InitBluetooth();
 InitServos();
}
```

summary:

serial.begin starts up the arduino. it then runs the initial functions in section 5

[8] - arduino loop

```
void loop() {
 if (Serial3.available())
 {
  incoming_command = Serial3.read();
  if (incoming_command >= 36)
  {
   switch(incoming_command)
   {
    case 200:
     SetSpecifiedServo(selected_servo, 1);
     break;
    case 201:
     SetSpecifiedServo(selected_servo, -1);
     break;
    case 100: // mound
     SetToConfig(CONFIG_MOUND);
     break;
    case 101: // flat
     SetToConfig(CONFIG_FLAT);
     break;
    case 102: // wave
     SetToConfig(CONFIG_WAVE_1);
     SetToConfig(CONFIG_WAVE_2);
     SetToConfig(CONFIG_WAVE_3);
     SetToConfig(CONFIG_WAVE_4);
     SetToConfig(CONFIG_WAVE_5);
     SetToConfig(CONFIG_WAVE_6);
     SetToConfig(CONFIG_FLAT);
     break;
    case 103: // ditch
     SetToConfig(CONFIG_DITCH);
     break;
    case 104: // slope
     SetToConfig(CONFIG_SLOPE);
     break;
    case 105: // valley
     SetToConfig(CONFIG_VALLEY);
     break;
    case 106: // random
     SetToConfig(CONFIG_RANDOM);
     break;
    case 107: // all up
     SetToConfig(CONFIG_MAX);
     break;
   }
  }
  else if (0 <= incoming_command && incoming_command < 36)
  {
   selected_servo = incoming_command;
  }
 }
}
```

summary:
this loop will keep running the entire time arduino is running.

it looks at the bluetooth connection and sees if there is any new data to read. if there is data it gets saved to the incoming_command. there are 256 slots to save data to.

values 0-35 are for selecting individual servos for control.

values 200 and 201 are for moving the selected servo up and down.

values 100-107 are for preset configurations.

CONCLUSION

ANALYSIS

- The biggest impact to the design of the two prototypes was using the 180 degree servo motors. They were chosen because they are the cheapest thing I could find by a long shot. Using any other motor would have blown the budget. Every other part of the prototypes were chosen specifically to work with the servos. Going into the project I didn't realize how much of an impact that would have on construction time and ease of use. Perhaps paying the premium price is worth it.
- I had originally intended to use the prototypes to be able to recommend materials for actual construction of a kinetic landscape. However, I now realize a material mockup prototype is several prototypes away. I probably even started with too advanced of a prototype. I should have started with small, simple movement models that test for the best method of movement rather than spending the time and resources to construct a full model.
- Working with Arduino is very complex, more than the average landscape architecture firm would have the capability to manage. Before continuing with this method of prototyping I recommend deep discussions with architects, artists, and mechanical engineers that have produced kinetic structures to see their design procedures.

FUTURE RESEARCH

1. prototypes:
   - hydroponic model to show that trees and plant material can be moved.
   - a hybrid model to combine the structural features of prototype 1 and the fluidity of prototype 2.
   - movement models
2. interviewing:
   - talk with architects, artists, and mechanical engineers that have built kinetic structures to see how to make it more of a reality for a landscape.

COST

OVERALL COSTS - $902 (i.e. makerspace membership, classes, arduino kit)
PROTOTYPE 1 - $87 (i.e. wood and fasteners)
PROTOTYPE 2 - $373 (i.e. power supply, app consultant)
TOTAL - $1,362

SUMMARY

In summary, my entire project concept was pretty ambitious. The learning curve with the set time frame was extremely intense. Yet, I think the push generated great results and in the process I picked up a lot of useful skills. I became more skilled in 3D printing, laser cutting, wood working, app making, coding, and working with arduino. The project was a success in that it confirms that designing by prototype is indeed a great method for communicating the idea of kinetic landscapes and generating concepts. Everyone I showed the models to had ideas about what they could be and what kind of kinetic space they would want. When just discussing the topic of kinetics or showing pictures of other kinetic structures this doesn’t happen. A prototype moving in front of you really gets the idea across.