**## 1701QCA: Assessment 3 ##**

**# Title: ‘Two products of a non-agent’ #**

![Image](ad.jpg)

**## Index**: ##

**## BACKGROUND ##**

1. RESEARCH
   1. Readings
   2. Rationalisations (lizard)
   3. Research and influence
   4. Novelty discuss similar
2. Overview of design
   1. Interaction chart (if changed (safeguards))

**## THE BODY ##**

1. NECK
   1. Making the neck
   2. Does the neck actually shift weight to enable walking? yes
   3. Neck on final body
2. PULLEY AND LIMBS
   1. Just one
   2. Making all the final limbs, pulley and arrangement on body LONG BIT SEE MESSENGER
   3. All four
3. Finalising the body
   1. Microbit shield deliberatiins on how to minimise pins, wiring all to one, later changed for better control, how the servos are layed out (which side theyre on and such). Microbit shield is small enough to fit show how, also how microbit is secured
   2. Tail, battery arrangement, and how everything is powered.
   3. Wiring: what maps to what now, shortening of wires
   4. Grips
4. Does it walk all together?
   1. Can it get up
   2. Does it need a screw
   3. What walking pattern
   4. Code (minimum heart rate, maximum heart rate is irrelevant)

**## HEART BOX ##**

1. New Sensor
2. Heart Box (drawings)
   1. Research reference, branching, veins, outward
   2. Options considered (lizard, lizard mouth, heart), what a combination!
   3. Chosen design; why? Drawing connections, removing agency, minimising user input aside from passive, reading conceptualisation map, aesthetic wires, (drawings)
   4. Making of (wire threading, copper tape flop)
3. Code with Heartbox
   1. Code issues
   2. Demonstration with heart box
   3. Instructions???

**FINAL TOUCHES**

1. Presentation of lizard
   1. Keeping access plans (Velcro)
   2. Fur (length)
   3. Neck issue?
   4. Pulley cap
   5. Heartbox lizard

**FINAL OUTCOME**

**# THE BODY #**

### 2.1 Materials ###

The materials I ended up using

* Electrical tape
* a second microbit
* 5 small servo motors
* 1 large servo motor
* Pulse Sensor from Pulsesensor.com
* A pulley wheel
* Crimping tool
* Two 6v battery packs
* Two 3v battery packs
* Super glue
* Hook and Loop fasteners

Some things I already owned

* The first microbit
* Assorted screws, nuts and bolts from around the house
* Thermoplastic beads
* Binding wire and thread
* Nylon string
* Faux Fur
* Heat shrink tubing (could’ve been straws)

**## Early development ##**

### Connections ###

Most of my connects are solderless and use twist and tape method from the replication project. I used the soldering iron once for a particularly difficult connection.

### Servo Basics ###

I already knew how to operate servos from classwork. I used a simple bit of code with button a and b presses corresponding to different servo positions to test functionality of my connections.

First I connected a servo just to the microbit (Input line, Vin and Ground) and found out in [this forum](https://arduino.stackexchange.com/questions/40145/micro-servo-is-not-working-with-arduino-uno) that if the Micro:bit is also connected to a computer via USB, it makes the servo jitter because the voltage is too high. I keep this in mind for the rest of my experimentation, but I will be using an external battery pack for most of it.

First connecting the external power source I learned from this [video](<https://www.youtube.com/watch?v=9qgABVkvbYI>) that the battery pack ground also has to connect to Micro:bit ground.

![Image](test1.gif)

Adding more servos (four in total, they started jittering again. It appears jittering has everything to do with power. I needed to increase from my 3V to a 6V external power source and they worked again.

![Image](test2.gif)

In later experiments I kept finding that batteries were running out quite quickly and that was often the reason my servos stopped working suddenly.

### Walking plan ###

Without shifting weight it would definitely just shuffle because of how the legs would have to touch the ground when pushing off and also when bringing them back up.

![Image](test3.gif)

I understood that my designs involving shifting forward weight with the head were necessary to make the lizard walk. This is supported by the photovore insect project and Monster Chan from my research.

In early plans, the leg pairs were diagonal, that is the back left and the front right would move in tandem and the inverse is true. The neck (and then I thought maybe the tail) would shift the weight of the lizard onto the leg that was contacting the ground, and the one that is already in a back position would become airborne and rotate forward. In these sketches I also thought that two halves of the body rotated independently of one another.

![Image](diary1.jpg)

I ditched the tail as another shifting weight (tail 2) as the servo count was getting expensive. Instead the tail is a counterweight.

I considered the different angles the servos could sit on and how they would affect movement. I nonsensically thought that angling them a certain way would prevent contact on the legs movement back. Option 2 is influenced by the photovore project, but I decided against it because it requires constant contact to push off since its vertical position doesn’t change. I went with option 1.5 because it moves the body back and away so that it moves in a sort of zig zag motion, ideally.

![Image](diary2.jpg)

![Image](diary3.jpg)

Here was my final idea of how the Lizard was going to walk.

![Image](walk.jpg)

### First frame ###

I made the first frame out of thermoplastic with a slight upward angle in the sides for the servos to sit on. I held everything down temporarily with screws, and the legs were bound to the servo arm with wire running through drill holes. A large screw supported the body in the middle. The wires were fed through a hook.

![Image](body.jpg)

I couldn’t test out how well this body moved. In the process of connecting one of the servos with the leg I broke it. Another servo had a gear that was thicker than all the little plastic arms that it came with. I forced one of them on it, which widened the arm. I forgot this and later when the last arm I had left was too loose for the last servo and I used a screw to secure it. I guess I used the wrong screw, because it loosened the gears inside the servo and I couldn’t fix it. I bought another one later, but for now, there’s how the front legs move without a neck.

![Image](twoleg.gif)

![Image](servocode2.png)

## Neck ##

The neck saw some design iterations from the original. I realised the original bike chain design wouldn’t turn evenly through the spine, and that how well it turned greatly depended on the friction between the pieces. This was too much precision for me.

![Image](neck3.jpg)

![Image](neck.jpg)

I changed the design modelling it after my friend’s crocodile rattle toy like this one:

![Image](croc.jpg)

![Image](neck2.jpg)

I thought I could skimp out on work just by connecting the plates with thread, but there was too much rotational movement and I realised I would have to adhere to the original crocodile with a strip of canvas running through the middle as it only has one axis of movement.

![Image](neckfail.jpg)

![Image](neckfail.gif)

I used super glue to join the neck pieces to the canvas.

![Image](crocneck.jpg)

![Image](crocneck3.gif)

In my plans, the degrees for left and right were a bit extreme (45 and 135). After settled for an average between those and neutral (90) and it worked well.

![Image](neckcode.png)

Although I had to add a supporting wire to the neck. The neck tended to curve back so much that when the servo turned to the other side it didn’t follow. It also bowed too much.

![Image](neckbad.gif)

The supporting wire was quite a nice result. It has a good bobble to it.

![Image](bobble.jpg)

![Image](bobble.gif)

### Does the neck actually shift weight to enable walking? ###

Experimenting with just 2 legs as I still didn’t have the 4th leg servo. To my surprise, balancing on the screw the lizard was able to walk! Here it also becomes obvious that the walking servos can be written to one pin.

(walking video)

![Image](legneckcode.png)

I thought maybe I could make the walking more gradual as these motions are quite violent. I created degree increments and separated them with small pauses. However, this just creates stuttering and sounds awful.

(awful video)

![Image](legneckcode2.png)

### Neck on final body ###

![Image]()

## Pulley ##

### Pulley tension ###

The pulley servo tightens the limbs of the lizard enabling it to walk, and releases when it is “dead” to make the lizard drop. It works by threading nylon string through the joints and loosening or tightening it up with a stronger motor (6V) with a disc. When it’s tight, the joints come together rigid in a lock and key fashion. The string is guided through the body with eye screws. In my original sketches this lock and key was round.

![Image](pulleyplan.jpg)

The sphere was too slippery though and even while rigid the limb rotated.

![Image](sphere.gif)

Next I tried a cube lock and key, but it turned out too tight and would slot correctly without effort from me. It is a good lego-style slot in though, quite strong and may be useful in future projects.

![Image](square.gif)

Lastly I went for in-between: a pyramid lock and key (lock indented with edge of sharpener). To my surprise this worked. The leg is soft when the pulley is at the original position, and tight enough that it can bring the lizard up off the ground I believe when the pulley is at maximum (180) position.

![Image](triangle.gif)

![Image](puleycode1.png)

### Does the pulley servo conflict with the walking servo? ###

I connected this leg to the walking servo with two screws, but the servo arm I did not screw on.

The leg I’m using here is the same as in the previous pulley experiment.

The walking servo was held in place with the pulley servo with a rubber band.

This experiment did show: yes! the legs move back and forth like usual. There is minimal conflict between the walking and tension.

(video)

![Image](pulleylegcode.png)

\*\*\_Angle of the string\_\*\*

In this experiment I learned that the angle at which the string is pulled relative to the walking servo is very important. It needs to be parallel to the walking servo, otherwise it’ll exert most of the pressure behind the servo (and therefore stop it from functioning or break the servo when the servo arm is screwed into the servo). For this reason I also left behind the angled body design of the previous experiments for a flat body where the angle of the string is much easier to control. The limbs needs to be perpendicular to the walking servo so that the string pulls “into” the servo, not up.

\*\*\_Tightness\_\*\*

In the code above because of how I positioned the pulley, the tightest position is at 0, so all the numbers are inverse. Of course, if the pulley pulls in the other direction enough (180) it’ll tighten that way as well, but that is not how the string intends to be pulled. It is not a major difference.

The numbers here are ideals for this particular experiment in tightness discovered through trial and error: too tight and the leg cannot move when it is taught, too loose and when the leg is “dead” it can twist in its socket and shorten the string available to pull.

In the final body this ideal tightness should be set first for one leg, and then manually the strings should be tightened (with a screw) for all the other legs to maximum tightness at the same position as the first leg is most tight.

\*\*\_Notes for the future\_\*\*

The other viable design would be design 2, with the servos being on the underside of the body and rotating on the x rather than y axis. I’ve previously discussed why this isn’t my preferred design, but it would work with these servo requirements.

I also worry that the pulley can keep one leg tense, but 4, maybe not. If so I can fall back on my front leg only design that tightens the front two legs and the back legs are decorative.

### Making the final limbs ###

(MESSENGER)

### Can a pulley tighten all 4 legs? ###

I have bought the 4th servo and am able to test out how the pulley works with all 4 legs.

\*\*\_Power supply issue\_\*\*

When I tried to test walking, I ran into a power supply issue. A 6v battery definitely couldn’t sustain 4 small servos and 1 large servo as it cannot sustain 2 small servos and 1 large servo. From my previous experiments, 1 large servo and 1 small servo are just about the limit. I will need to purchase another 6v battery. My progress here was stalled.

For the mean time I just tested out whether the pulley could tighten all 4 legs. I also started using the second micro:bit as a wireless controller, to avoid interfering with the lizard and for convenience.

The results show that the pulley can tighten all four legs enough to stand on with some trail and error of manually tightening the screws. It cannot get itself up anymore as the experiment with just one leg did because the body cannot turn as freely to accommodate the lock and key, but with some wiggling of the walking servos combined with tension the locks can find the fit I’m sure of it.

The neck and LED are also in action here.

(first 4 pulley experiments)

Once again, the smaller the number the tighter in this code.

![Image](4pulleytight1.png)

![Image](4pulleytight2.png)

## Finalising the body ##

(RESULT overview)

### 2.6 LED ###

The 1cm LED that I used diffused quite well through the lizard’s skin. It is very bright.

I did my maths wrong calculating what resistor I’d need. The guy at the store who helped me find the 27 ohm resistor that I thought I needed according to my maths said that one should only need a 47 ohm resistor according to his book.

The maths I did was influenced by this [site]( <https://www.evilmadscientist.com/2012/resistors-for-leds/>).

There is a 2.4V drop according to the manufacturer

I’m bringing that down from 3V

3 – 2.4 = 0.6V

V = I\*R

0.6 / 0.025 (suggested value) = R

R = 24 ohms

Oh well! It definitely works with 47ohm.

I did a basic gradual loop:

![Image](LEDcode1.png)

This LED was places between the two front legs, neck servo and pulley servo, with the wires going directly through the body onto the other side with the microbit shield.

### Microbit on the lizard ###

I thought about how I might bring the number of necessary pins down to 3 in order to fit in just a microbit on the underside of the lizard. I knew that I could write all legs to one pin because of how I oriented them (positive degrees corresponds with movement forward on one side and movement back on the other)

I still, however, had 4 pins: legs, neck, pulley and LED.

I would really like the LED on there. I thought that maybe some of the servo writes could change the brightness of the LED, and they could, but it seems servos operate on such a small difference in current, it doesn’t have much of an effect.

I managed to fit the microbit shield with extra pins onto the underside.

(microbit shield)

I later secured the microbit in there with a rubber band that is stretched by two screws across the width of the body.

(rubber band)

### Tail and power supply ###

Early on I decided that the tail would not also swing from left to right, and instead house all the batteries. After I had to replace a 3V pack with a 6V pack, the body got considerably larger, but it worked. The 6v battery on the bottom is being held in place by a screw and the other two batteries are connected with Velcro.

(battery placement)

Shortened all the wires and mapped all the legs to their own pins since I thought I could get quite natural motions like this.

(cleaned up wires)

At this point the connections mapped like this:

P1: Back right

P2: Back left

P12: Front right

P16: Front left

P0: Pulley

P15: LED

P20: Neck

### Grips ###

I added some grips to the feet to assist the lizard in getting up on its own. I stretched white, self-adhering silicone tape over the feet.

## Does it walk all together? ##

### Power Supply Issues, again ###

Even with enough battery packs I ran into an issue I’ve never encountered before.

When I hooked everything up it started to shake.

(shaking)

This was clearly a power issue. After removing the code for some pins the issue stopped. Adding one servo at a time, it came back.

I had 8 pins connected when I first tried to test its walking ability: individual pins for the legs (4) as I thought it would move more organically with individual control, pulley, neck, and LED.

It appears as though the Micro:bit can only write 3 signals at a time, because when I added 1 servo Pin to the LED, Pulley and Neck command it started shaking where it didn’t before. After I replaced the LED pin with a servo pin it didn’t shake.

I knew I could write the legs to the same pin, but this meant I still had four pins.

![Image](servoalt.png)

I had to remove one and it was obviously the LED. It is a shame and I think it added to the strength of the connection between the lizard and the heart beat but nothing can be done on such short notice.

### Can it Get up? ###

Angles corresponding with motion on each leg:

right back 90+ = backward

Left back 90+ = forward

right front 90+ = backward

Left front 90+ = Forward

So lefts and rights move in tandem.

I thought it would be cute if the lizard could up with the front legs first like a real animal, but this was no longer feasible.

My theory earlier about it needing to move the legs a bit for the keys and locks to fit together and fully tense up the pulley was true. Once I combined the pulley tensing sequence with leg movement it got up with no problems in 5. When the leg moves, the keys have a chance to move closer to the lock and every time they do their position is further secured by the pulley tension.

(video)

This can be done by the movements of a regular walking cycle.

### Can it walk? ###

For these tests I added a screw in the chest of the lizard where the LED used to be. It is as long as the legs and touches the ground with them.

\*\* \_ Script\_ \*\*

This was the walking script in line with my previous ideas of how I was going to make it walk:

**\*\* Start (Button A in this test): \*\***

Head is on a severe tilt (P14:25)

Pulley is loose (P0:130)

Legs straight (P2:90)

**\*\* On input (Button B): \*\***

Head is straight (P14:90)

Pulley is taught (P0:50)

Head rocks left (P14:70): Lizard tilts left very slightly

---pause---

Left feet back, right feet forward (P2:65)

---pause---

Head rocks right (P14:110)

---pause---

Left feet back, right feet forward (P2:115)

---pause---

loop until…

**\*\*No input (represented by button A+B): \*\***

Head is on a severe tilt (P14:25)

Pulley is loose (P0:130)

Legs straight (P2:90)

![Image](servoalt2.png)

\*\* \_ Centre of Gravity\_ \*\*

The head wasn’t doing anything at all because the centre of gravity on this lizard is way back in the tail. Who would’ve thought that 10 batteries would weigh a lot. I tried to shift the pivot screw back. The microbit shield was previously in the way and now I didn’t need it anymore with only 3 pins used.

The microbit is held in place by bolts fitting into snuggly fitting drilled holes in a staircase piece to prevent the restart button from being pushed. The bolts are both keeping the microbit in place and holding down the wires. The 3V battery pack has had to move from the tail to the chest.

(photo of microbit holder)

A series of complications have occurred here. When I was doing my initial experiments with the walking, I did not account for the weight of the batteries and the corresponding centre of gravity. In those experiments, the battery was being dragged behind. There they are attached to the frame, bringing the centre of gravity backward, away from the head.

The second issue is that the pulley did compete with the legs after all. In my earlier experiments it didn’t appear so but with the whole weight of the lizard its clear more torque is needed to meaningfully propel it. The legs do move, just not enough.

My first solution was to prop the lizard up with a screw in the back instead of the front, delegating some of that battery weight to the screw which can be tilted on. I tried a variety of screw lengths. None of them facilitated walking because the neck was so angled downward that it was hitting the ground, and the back feet were not touching the ground. The lizard did not propel itself.

(video of screw)

My second solution was a screw, a shorter one, and also moving one of the large battery packs to the front as well.

This moved the centre of gravity, but I then realised the lizard was just too heavy to be affected by a plastic head.

I tried putting a large bolt in its head, totally ruining the lizard’s appearance. I experimented with various weights on the head but I ran into the same issue of the head hitting the ground. Without suspension, all the mass that’s meant to be tilting the lizard is just supported by the ground.

I tried to support the neck some more with a suspension wire underneath the neck as well.

This did nothing but limit the neck’s movement.

I also turned off the pulley and taped the legs together, so they were rigid on their own. I knew if this solved my issue I could just glue the lock and key together and leave the slumping behind. This certainly helped, but not to the extent that it was worth abandoning all the effort I put into slumping.

If I had more time I would experiment with better means to shift the centre of gravity. I would also angle the neck up to counteract the downward weight. I would change the material with less elasticity. The fabric is elastic enough that there is marginal y axis tilt that becomes obvious with increased weight. A hard plastic sheet would do a much better job, and would also likely not need suspension with a wire.

At this stage there is no way I can, with reasonable expectation of success, enable this lizard to walk by the due date. The other option I didn’t (and can no longer) explore here is the aid of the wheel. But in my decision framework I made it clear it is a high priority for the lizard to move rhythmically rather than continuously. I am falling back on the option of just having the lizard shuffle in place. It still retains the ability to slump over and pick itself up.

I removed the grips (they were preventing the lizard from flopping back down), removed the bolt, moved the battery back to the tail. The batteries in the tail are held in place with Velcro in the case that they need to be changed out. The battery on the underside of body is also being held in place by Velcro and also a rubber band that wrapped around two screws. The Velcro keeps it from sliding and the rubber band keeps the battery from coming off the Velcro as might happen being on the underside of the lizard.

Here is the result, it mostly shuffles:

(video)

![Image](servoalt3.png)

### Wiring changes ###

\*\* What I thought the wiring would look like: \*\*

Sender Micro:bit

![Image](diagram3.jpg)

Receiver Micro:bit

![Image](diagram1.jpg)

\*\* Changes made since then \*\*

Sender Micro:bit

![Image](page02.jpg)

Receiver Micro:bit

![Image](page01.jpg)

Here is what the final lizard is built like:

![Image](.izard.jpg)

**# HEART BOX #**

## Heart Module ##

### MAX30100 ###

There were a lot of issues with the heart rate monitor. The original one that I used was MAX30100.

It was made for Arduino and not for Micro:bit as it’s pins connect to the clock and data pins of the Arduino. Furthermore, the [library](<https://github.com/mfitzp/max30100>) I learned some Micropython for seems to be written in Python and needs to be adapted to Micropython to actually work, which is way beyond my capacity. This is according to this [forum](<https://forum.micropython.org/viewtopic.php?t=8101&p=46069>). I assume this is why it was giving me memory errors.

I did learn all the Micropython commands I knew in MakeCode, how to import a library and how to use the serial screen.

### Pulsesensor.com knock off ###

My backup analog heart module arrived in the mail just as I started feeling lost. It seemed quite similar to the PulseSensor.com sensor so I just followed the Micro:bit instructions on their website: Vin to 3v, G to ground, Input to any pin and just read analog pin for signal.

It worked well, with clear “beat” and “no beat”. Although I found that the pressure applied was very important. Too much and the signal was maximum (1032), too little and it stayed pretty low (<300) and didn’t fluctuate much in both instances. Also very strangely sometimes the sensor detected a pulse on my desk which I can only guess has something to do with my lamp.

![Image](heartrate.gif)

I needed to figure out how to detect a beat and make sure this sensor was accurate, but \*\*IT BROKE! The connections were soldered on and didn’t take much twisting. I ordered the actual pulsesensor.com one.\*\*

I thought about two methods for when the next one comes. Until then I will use the simulator, dragging the pin up and down in time with a metronome:

\*\*method 1\*\*

If PS is 0, and within X time it is >500, count that as a beat, otherwise the time expires and repeat. Make a counter for the beats and after 15 seconds display beats per minute.

\*\*method 2\*\*

Alternatively, if PS = 0 and PS > 500 in time X, mark upper or lower with timestamp, repeat for next beat and subtract the timestamps. Average out after 30 times gives average time in between beats (this is similar to my timing gate replication project).

### **Method 1 ###**

**\*\*Step 1: detecting a beat\*\***

**![Image](heartcode1.png)**

When signal is above 370, box 1 is checked

When signal is below 100, box 2 is checked

Both boxes checked, count as beat, increase ‘beat’ by 0.5 because the beat is counted twice, on the rise and fall.

Uncheck boxes.

**\*\*Step 2: Timer\*\***

Adding a clock that shows T for 10 seconds and displays BPM (number of beats X 6) afterwards, then resets beat counter.

**![Image](heartcode2.png)**

**\*\*Step 3: Timer linked to beats\*\***

**While timer box is unchecked measure for beats. When it is checked show BPM.**

**![Image](heartcode3.png)**

\*\*Step 4: Testing if this method gives accurate BPM\*\*

I have to test accuracy with manual inputs guided by a metronome, so there’s a degree of human error. A few trails gave these values:

**![Image](hearttrial.PNG)**

So accurate enough, at least the system. Might take adjustments with sensor when it arrives.   
I can also decrease the sample from 10 seconds to 5 if it feels too laggy on the user (probably will).

**Method 2**

I am dismissing this method because method 1 is effective enough. It also didn’t account for the return of the beat from 1032 back to 0 (shouldn’t be counted as a beat). This is dealt with in method 1 simply by halving it. The 10 second period is an average already. But the returning “beats” cannot be counted in method 2 because this produces in inconsistency in time (illustrated below) This could most likely be worked around but at this point it is needlessly more complicated.

**![Image](hearttrial2.PNG)**

## Knock off pulse-sensor code ##

This code was written before any of the components were put together, and their problems were not yet apparent. This code is the foundation for the final code however.

\*\***Sender:\*\***

**\_Heart code\_**

Identical to final heart code in section 2.7

![Image](heartcode3.png]

**\_Radio \_**

Send value when timer is up

![Image](radiocode1.png]

**\*\*Receiver:\*\***

**\_Radio \_**

When BPM is received convert to wait time (time in between motions).

If bpm is 70 beats per minute, how much time is that in between each beat?

Bpm/60 = 0.8 seconds.

0.8 x 1000 = 800 ms

![Image](radiocode2.png]

\_**LED**:\_

Taking code from LED script, the 30, 50, 100 and 1000 ms increment gradient should happen in the space of one ‘wait’. So each gradient should be a fraction of ‘wait’ proportional to the times I gave it in the LED code from before. I also need to make sure the LED is only one when the lizard is in “live” mode (wait is about 10ms).

Some maths converting the four current millisecond increments to fractions:

8 X 30 = 240

2 x 50 = 100

1 x 1000 = 1000

1 x 100 = 100

= 1540 ms total duration of cycle

**30ms/1540 = 0.019**

**50ms/1540 = 0.032**

**100ms/1540 = 0.065**

**1000ms/1540 = 0.649**

![Image](LEDcode2.PNG)

**\_Servos\_**

On start: dead mode. When rhythm detected, tense up and straighten neck. After that, loop walking cycle separated by “wait” time.

If wait stops being above 10ms, go back to dead mode.

![Image](servocode.PNG)

\_**Some touches\_**

On sender Micro:bit:

To ensure that only a finger will set off the timer, skin should close a circuit and press a pin (pin 0).

If it’s not pressed, then timer is set to 2 so it doesn’t affect anything and radio signal is 0.

If it is closed, draw heartrate bar graph on LED screen (feedback for user)

Indicate readiness but absence of input if pin 0 is not closed with LED dot.

![Image](finalcode.PNG)

## Official Pulse Sensor ##

The new sensor is very similar although somehow is less reliable than the knock-off. The finger needed to be held in a very specific position, midway between the flat and the tip of the finger, to get a good reading.

I spent a long time trying to trouble shoot this with various spacers thinking it was the pressure I was putting on it. Then I learned how the finger should be positioned through looking at some third party material like [this](<http://www.circuitstoday.com/pulse-sensor-arduino>)

Worth noting is that on the sensor’s website the sensor is shown on the flat side of the finger.

It is accurate. I compared readings with a heart rate phone app and they were nearly matched (sensor 72BPM from phone 70BPM) in the 5 trials I did holding my finger correctly.

### Levels ###

After some experimentation the thresholds I found that were most reliable were 450 (lower threshold) and 550 (upper threshold). A much smaller variation than before.

![Image](heartcodealt1.png)

For this reason I exaggerated the difference on the display screen with some math manipulation.

![Image](heartcodealt2.png)

### Timer iteration ###

When timer state is 1 (run out), show bpm. This is only for testing purposes and it will not show BPM in the final.

When timer state 0 (ongoing), show the graph plotting heartbeat for feedback to the user about whether their finger is positioned correctly and subsequently whether their measurements are being taken correctly. This will be in the final code.

When timer state 2, where nothing happens except the start screen.

Of note is that negative numbers are plotted positively, so if the input in the sensor is less than 200, every number below that is a negative plot. This is not a major issue since the sensor doesn’t fall below 300 even when finger isnt being read. Just notable.

I then added Pin 0 to the new code, the gate that allows measurements to be taken when a finger closes it and confirms contact. I did this (already in the previous code) because there’s really no way to easily turn off the microbit, and even if there was I wanted this to be a minimum touch control aside from sticking your finger in, without asking you for confirmation for whether or not you want to start sending this bpm over.

Without a BPM to send over, the lizard doesn’t move.

Closing Pin 0 during timer state 0 displays the graph and during state 1 displays BPM. I initially implemented this incorrectly. There was a conflicting command of “when timer is = 1 show BPM, otherwise show graph” I included earlier, and also “when timer is 2, show start screen”. The screen flashed both.

After that I added the condition of pin 0 being pressed and timer state being not 1 for the LED to display a graph.

![Image](heartcodealt3.png)

After that I found that when Pin 0 was released, bpm was still being displayed. I wanted all measuring functions to cease, so I added a ‘clear screen’ which made the start screen flash but I don’t mind it too much.

![Image](heartcodealt4.png)

### Timer-gate issue ###

My condition of ‘while pin 0 is pressed go through measuring sequence’ doesn’t immediately stop when pin 0 is not being pressed. It still displays BPM at the end of the timer and would therefore at this point radio a very wrong bpm to lizard.

My theory about why the BPM would still display is that pause is not a command that gets overridden by new input, so if pin 0 was let go of and not touched it should not display bpm like it would if the finger touched again.

This turned out to be wrong. It did display BPM anyway.

![Image](heartcodealt5.png)

My second theory is that the If pin 0 is pressed sequence cannot be interrupted by new input until its done. I even made pin 0 release as a separate input hoping it would override the sequence.

I looked at using the other similar command to logic, “while”.

![Image](heartcodealt6.png)

This also didn’t work. I thought maybe its also because pressing a pin is more like an event than a continuous input, like pressing a button, so I tried ensuring that it was a state by creating an explicit state when pin 0 is pressed. This also did nothing.

![Image](heartcodealt7.png)

I found a way to hide that the sequence was still going by clearing the screen when pin 0 isn’t being pressed with clear screen. This worked and also showed that pin press is definitely a condition not an event. I was determined to actually solve the issue though.

![Image](heartcodealt7.png)

There was extensive experimentation with averages. I divided the 10 second sequence into four parts and recorded each part as a separate variable that would get pushed down with every recording. I did this so that the sequence could end sooner and the new state of pin 0 not being pressed can take over.

(write bpmold -> bpmoldold,

write bpmnew -> bpmold)

This worked terribly, giving inaccurate values.

I decide this is not a major issue.

I guess its really not that big of a deal. The user has feedback for when the pin isn’t closed, and since they don’t see the radio signal being sent they’re not aware of the issue. If the user connects pin 0 again during the same sequence, their pulse will continue to be measured. With how It was before, after 3 sequences (the pulse sensor has some sort of internal adjustment period) my pulse stabilised at a realistic 75. So really this just represents a lag of a maximum of 10 seconds between the input of the user and a response from the lizard, which was already there and so far necessary when it came to updating the BPM.

Although I can shorten the timer and take more frequent measurements, I feel a longer timer is more accurate since the end of the timer interrupts the measuring process. I also feel the lizard will move to erratically with a smaller sequence, and itll be unclear whats going on.

I spent some time using the masking of the false bpm recording to tell the user very briefly that their sequence was recorded. I couldn’t make it short with the pause command, so I made two displays conflict and make them flash quickly.

I instantly decided against this. It’s a confusing signal.

Unlike the start screen that very much implies a static state with not much going on there’s very little I can fit on the screen to explicitly say that the sequence was captured (other than spelling it out, which would take way to long) that would not be a symbol the user would have to decode or be told about. The user doesn’t need to know the backend, they just need to know their pulse is being taken.

This is the code I ended up with.

![Image](heartcodealtlast.png)

## Building the Heart Box ##

### Research and Design ###

For the aesthetic design of the heart box I consulted my research. I found no useful similar projects using a finger sensor, not even children’s finger clamps.

Common imagery among my sources were vein like trees or other plant imagery like Jenny Kang’s ‘Pulse’ program. In Sean Montgomery’s ‘Emergence’ this is also true, with tree-like wires transmitting your pulse in a wave radiating from you. I was interested in the outward motion created by your pulse, as ‘sending out’ is exactly what the heart box does. I thought about using a branching graph on the LED screen.

In terms of similar microcontroller pulse sensor projects, they are all very practical and not concerned with the user experience.

The other designs I considered was the heart box being the head of the lizard or the body of the lizard to which you were giving life with your touch, but I decided this implied a lot of agency and made the user a Dr. Frankenstein.

I wanted the lizard and the user to be as equal as possible despite their relationship. I focussed on their commonality: the heart. Through putting their finger in this heart, it is somewhat ambiguous whether the heart is the user’s heart or the lizard’s heart and so it is shared. I planned out a box resembling a heart.

This is the design I resulted with after some experimentation. My original sketches were incoherent.

![Image](heart.jpg)

### Experimentation and Construction ###

![Image](progrss

I found that the LED screen emitted enough light to be seen through the thermoplastic and so I covered it with a thin cover that diffused this light very nicely. For this reason, I had to let go of the branching LED, but I retained the branching imagery of other projects through plastic arteries on the top left and through exposed wires of the pulse sensor which resemble smaller vessels. I made these decisions rapidly and intuitively while physically playing around with its design so I cannot speak for the exact influences.

The pulse sensor wires are thread through various drilled holes in the bottom and middle layer of the heart box.

The user sticks their finger through a guiding hole. I angled it so the finger, when at rest, would be in the correct position. I made the finger hole with my thumb to accommodate larger fingers, but this still turned out too small for large stepdad index fingers. His pinky still fits though.

I wanted this heart box to have one input and no buttons. I wanted the user to have the least agency in the experience possible, as really the input that controls the lizard is even out of the user’s direct control. For the reason the pin that needed to be closed to start the sequence needs to be triggered as part of the only action the user can do.

To strengthen the connection between lizard and heart box without creating a relationship of agency, the battery pack that powers the heart box is covered in fur like the lizard is. This suggests the relationship but doesn’t position the lizard as something being acted on. In fact, the lizard is providing as much power to this heart as the user does. When you disconnect the furry battery pack nothing works.

\*\* Gate \*\*

I stripped wires in the middle.

(picture)

I thread wires through drilled holes into the finger hole, into the wall that is closest to the user as that one most accurately positions the finger. The exposed wire is the only part showing.

(picture)

These wires do not leave the heart box as the pulse sensor wires do and connect to the microbit directly through the underside of the second layer.

I meant to use copper tape to cover the exposed wires, but I found that the copper tape wasn’t all that sensitive. Removing the copper tape either changed nothing or improved sensitivity to the pin being closed. The exposed wires did a good job most of the time, but after sitting dormant for some reason the gate would not close until some attempts. My guess is either after some struggle the moisture in the finger helps the conductivity, or the wires are accumulating current? I don’t know enough about physics to know for sure, but after a few strokes it starts to work.

### Demonstration with heart box ###

I’m super excited because that’s what the result does!! The diffusion of light eliminates a possible user tendency to read the bars too much (say minor fluctuations which to an inexperienced user may appear to be pulse and not just noise) and instead just see a more ambiguous beat that clearly slows down and speeds up, but indiscernible in the level that the sensor puts out which is entirely irrelevant to the user. In this video I haven’t yet reprogrammed the microbit to remove the bar indicating the sequence has been captured.

(VIDEO)

# FINAL TOUCHES #

## Combining Code ##

**### Integrating Lizard Microbit ###**

I changed the pauses in between steps to be ‘wait’, which is the received BPM converted to ms.

I also set conditions for when the lizard moves.

Minimum:

If the heart rate is too low (45BPM) it will not move. This way the lizard is not set off while the user is figuring out the sensor and while it’s stabilising. If the minimum rate is 45 BPM, the maximum wait time in ms is:

(60/35)\*1000 =

Max wait 1714 ms.

Maximum:

If the heart rate is too high (250 bpm) it will not move. This is a healthy max heart rate with some margin of error. This way the lizard does not break if some error occurs in the sensor.

Max wait = 240ms

This minimum wait also slumps the lizard over when the signal sent is 0, such as when the finger is removed.

![Image](servoaltlast.png)

No changes needed to be made to the heart code in order to integrate it.

### Testing ###

After testing I found that.

1. I found that the 10 second timer is too long so I halved it to 5 seconds.

I later forgot to increase the factor by which the beats were multiplied in response to this (from 6 to 12) .

1. The lizard reacts to the first heart beat too quickly. The first sequence that is recorded is faulty and erratic. To solve this I created another state, ‘gate’ that would be set to 1 if pin 0 was closed. This allowed me to put a 5 second delay between pin 0 and state change, discarding the first sequence.
2. The servo was set to tight and prevented the legs from moving after it was fully tightened. This is new. The code I used with this issue was the same as before and the legs were not too tight then. I then discovered I needed to change the batteries but lowered the tightness anyway as the batteries weren’t drained, they just weren’t fully charged. This is unreasonably tight.

**### Final Code ###**

**Sender:**

![Image](**senderbitcode**.png)

**Reciever:**

![Image](**recieverbitcode**.png)

## Lizard Exterior ##

The lizard is covered in fur.

Some progress shots of how I stitched it.

(photo)

(photo)

(photo)

A bin full of shaved fur:

(photo)

I kept access to the batteries, microbit and servos with Velcro strips.

(photo)

This is also true for the Heart box lizard

(photo)