AT3 Major Project

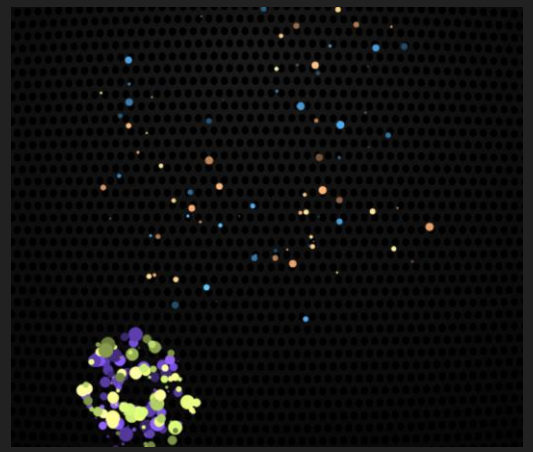
Jai Matthews | Computer Science 2023

Goals & The Stakeholder

AT3 Major Project

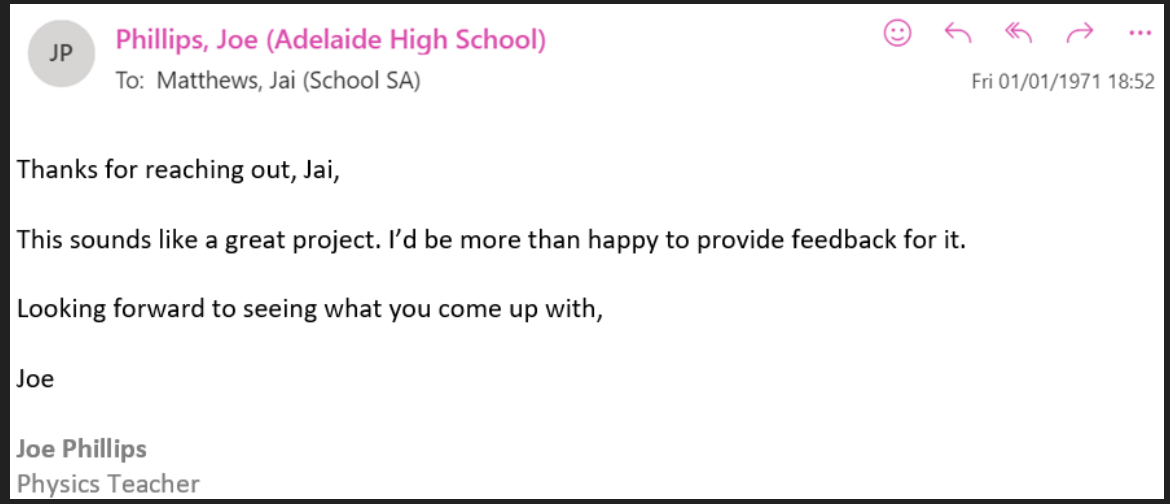
Project Scope

- The original idea was to create something visual.
- Animation could be a part of it. Particularly if that animation is generated using code.
- This code could be procedural, random, physics-based etc. to create this style
- This could partially be inspired by a web project called ‘Balls’. Screenshot below.



Stakeholder

- I decided upon a physics-based software. Akin to a physics simulation.
- I reached out to a physics teacher to be the stakeholder for this task.
- Joe Phillips has been a physics teacher for 17 years and was keen to see a general-purpose physics simulation.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

Thanks for reaching out, Jai,

This sounds like a great project. I'd be more than happy to provide feedback for it.

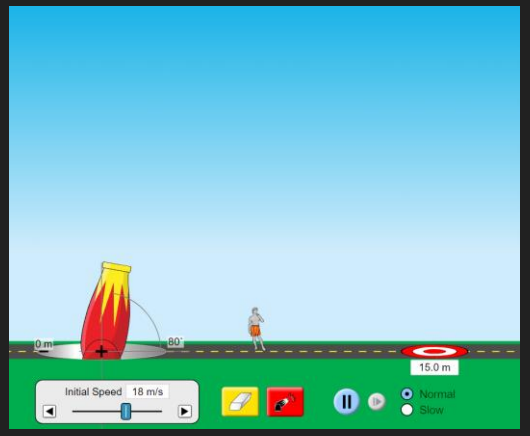
Looking forward to seeing what you come up with,

Joe

Joe Phillips
Physics Teacher

Problem Statement

- The purpose is to provide teachers with a physics tool to communicate motion topics to students.
- Many services available are limited to only performing specific tasks. Mine would be general-purpose and can be used for a variety of different reasons.
- This may mean, for example, that students will need to know how to set up a digital experiment and thus show an extra level of understanding.



An example of an existing physics sim (that I myself have been subjected to) that can *only* handle projectile motion and nothing else.

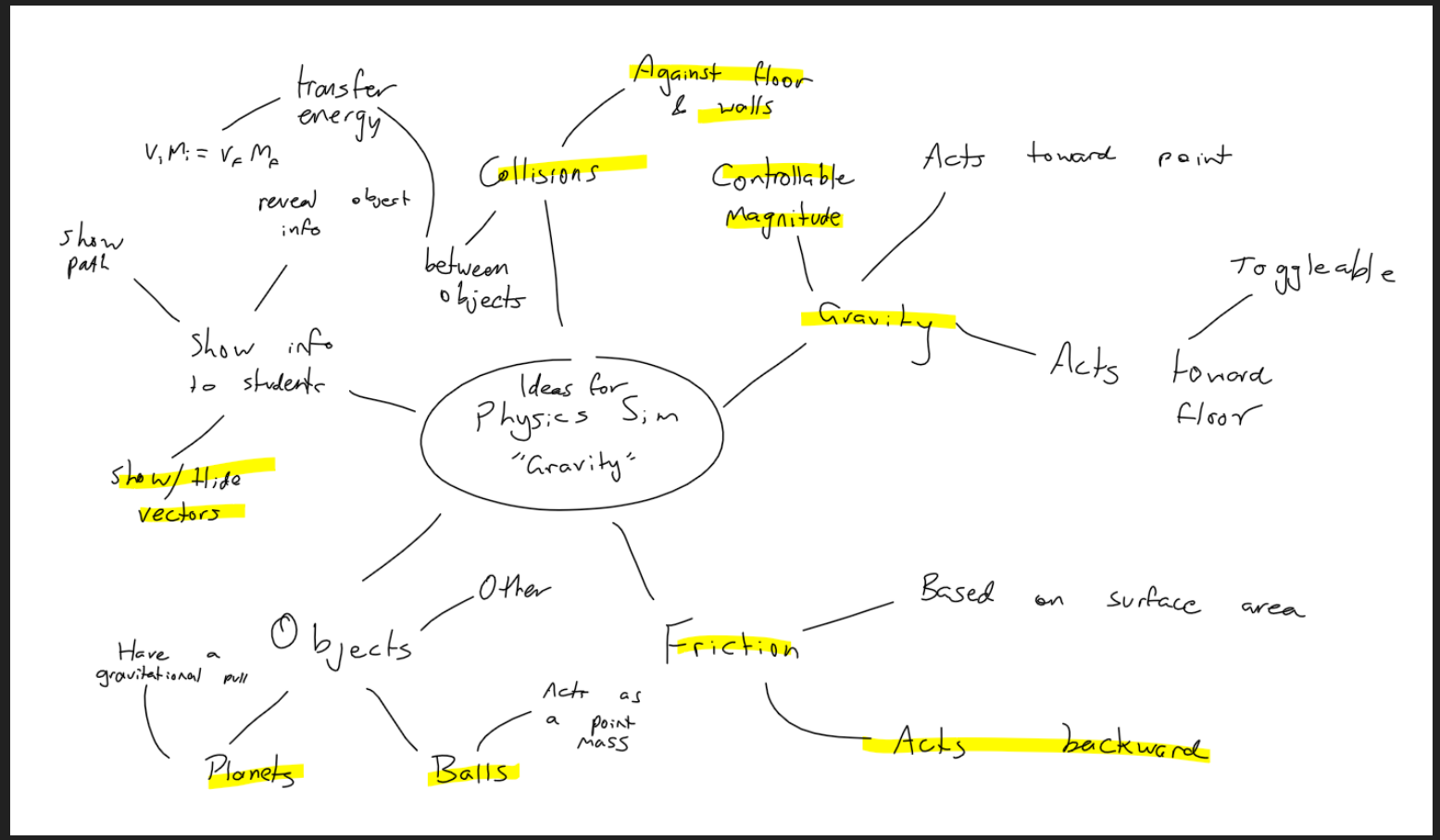
Goals

- Create a general-purpose physics simulation.
- Allow teachers to send setups to students that they can watch.
- Make it easy to use so students can set up their own settings.
- Include elements that allow for a greater understanding of the underlying physics.
- Make objects behave in it as they would in the real world.
- Include options for how the world works (e.g., controls for acceleration due to gravity)
- Make it enjoyable to use to spark students' interest.

Project Planning

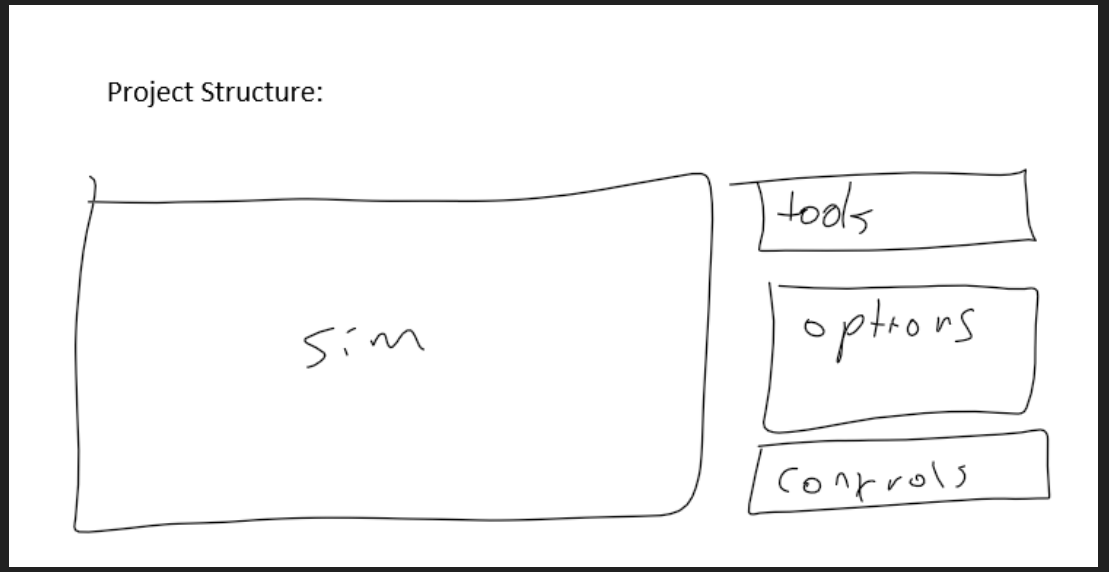
AT3 Major Project

Mindmap of ideas



Project Structure

- The structure of the project was planned and those plans are shown on right.
- The project is to be a simulation of physics with tools including ‘balls’ and ‘planets’ and options for how the sim works.
- You can play and pause the sim using the ‘controls’.



Types of Code

- The assignment is to be done in pure JS, to make it as accessible as possible to students.
- I will need a central array of every item on the screen; each one saved as an object.
- I will need a constructor function to generate each object.
- Given users will turn on & off settings, I will need if statements & other conditions to determine whether these settings are on or not.
- To animate every frame, I would need to use an interval function.
- Overall, I want an elegant and innovate way of making this simulation work. I believe the strategy I have chosen constitutes this.

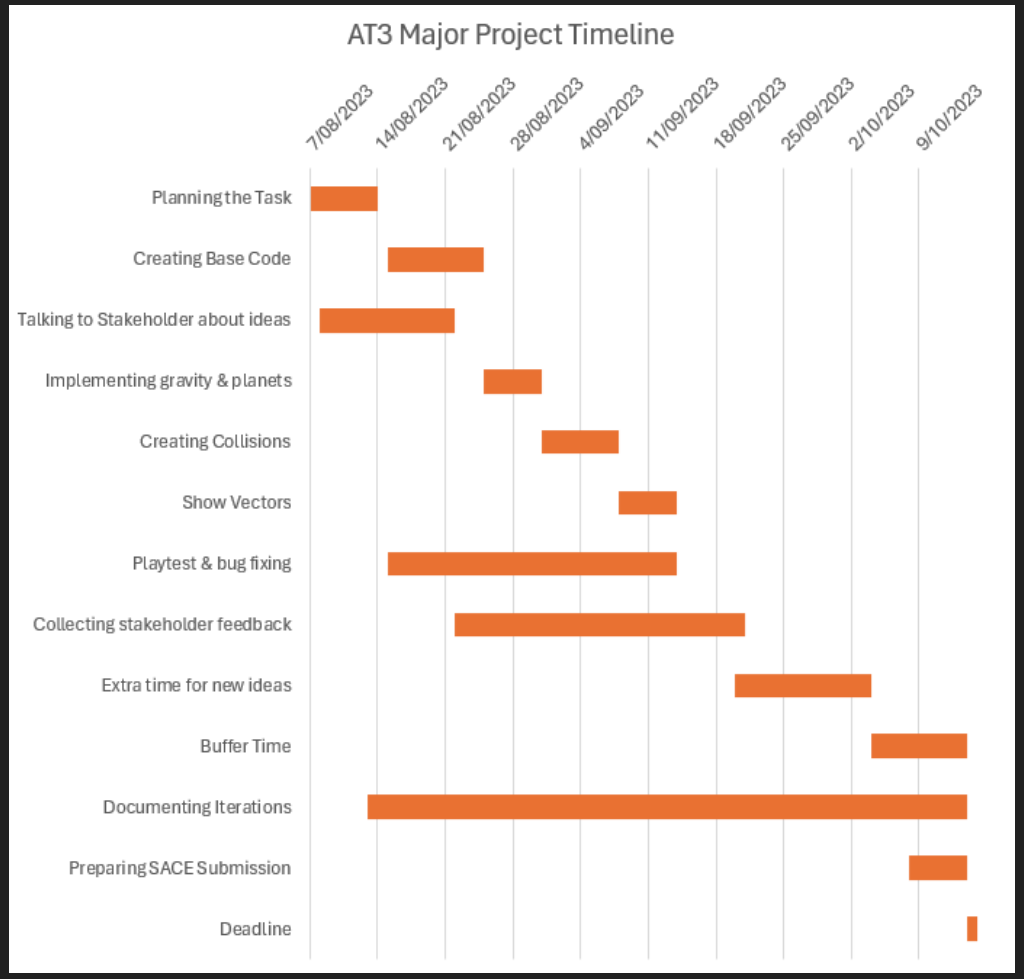
Sections

The Sections that need to be coded are listed below:

- Base code (including kinematics & animation)
- Gravity & Planets with gravitational pull
- Collisions between objects and against the wall.
- The appearance of vectors to show what forces are on an object.
- And more to come from feedback!

Timeline

- This Gantt chart helped me keep track of what needed doing and when.
- It gave me strict deadlines for key tasks.
- I also programmed in sections for new ideas; I hoped that testing & feedback would give rise to features I hadn't thought of yet in planning.

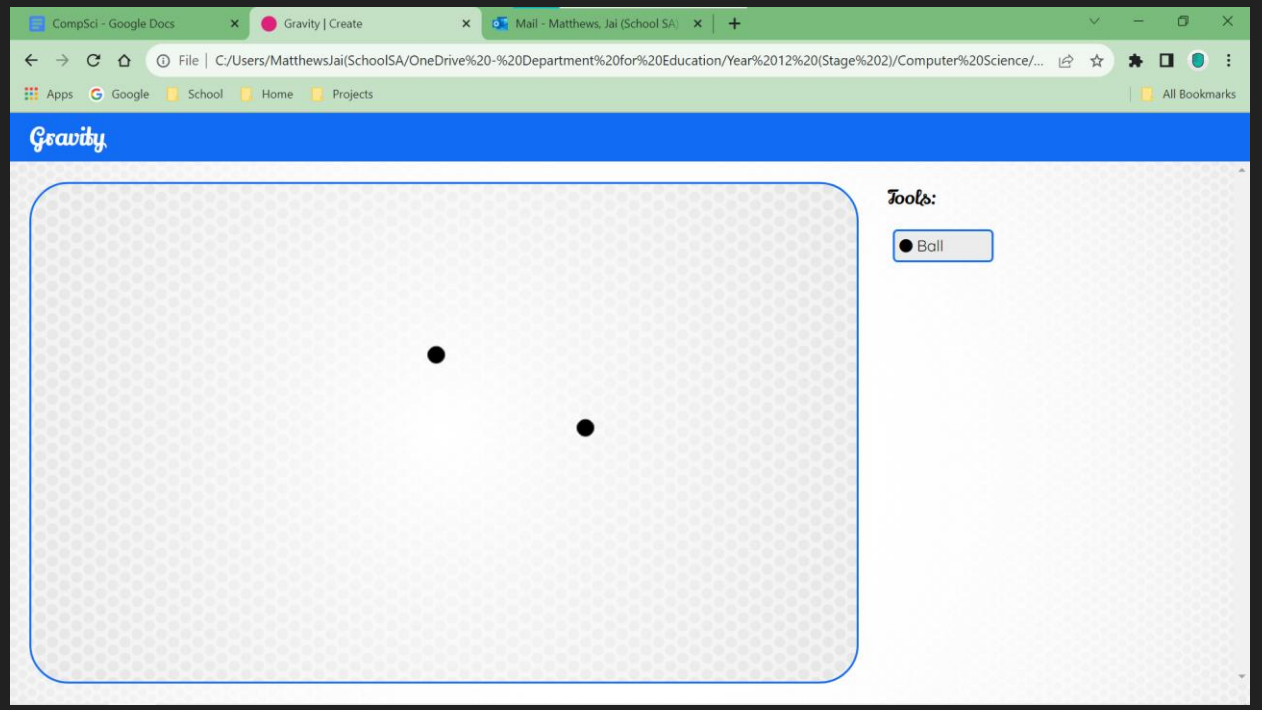


Iteration 1

AT3 Major Project

Overview

- After the first iteration, I had created an animation pane & the ability to click to add balls to it.



Constructor Function

- To allow you to click to add balls, I defined a constructor function that handled this.

```
function Ball(x, y, vx, vy, radius) {
    this.type = 'Ball'
    this.x = pxTOm(x)
    this.y = pxTOm(y)
    this.vx = vx
    this.vy = vy

    this.radius = radius

    this.draw = function() {


        //draw ball!
        c.beginPath()
        c.strokeStyle = 'black'
        c.fillStyle = 'black'
        c.lineWidth = 1
        c.arc(mTOpx(this.x), mTOpx(this.y), mTOpx(this.radius), 0, Math.PI * 2, false)
        c.stroke()
        c.fill()

    }

    this.update = function() {



        //calculate velocity & position
        this.vx += this.ax/fps
        this.vy += this.ay/fps

        this.x += this.vx/fps
        this.y -= this.vy/fps

        //done!
        this.draw()

    }
}
```

define properties

draw ball on canvas

deal with kinematics

call draw()

Kinematics Pseudocode

- The pseudocode shown on right handles the movement of objects.
- I divide the motion by fps to ensure that for a total of 1 second, the object moves the correct amount of distance

```
define fps (for example 24)

Every Frame: (fps times per second)

// velocity
vertical velocity += vertical acceleration / fps
horizontal velocity += horizontal acceleration / fps

// displacement (position
Vertical position += vertical velocity / fps
Horizontal position += horizontal velocity / fps

//draw
Draw ball at (horizontal position, vertical position)
```

Coding Animation

- For each frame, update every object using the function shown on slide previous.

```
function run() {
    //start running animation
    running = true

    //create interval
    animation = setInterval(function () {

        //clear screen
        c.clearRect(0, 0, preview.scrollWidth, preview.scrollHeight)

        //update each object. This also draws it
        for (i = 0; i < objects.length; i++) {
            objects[i].update()
        }

        //find number of objects
        document.getElementById('oCount').innerText = objects.length

    }, 1000/fps);
}
```

Converting Units

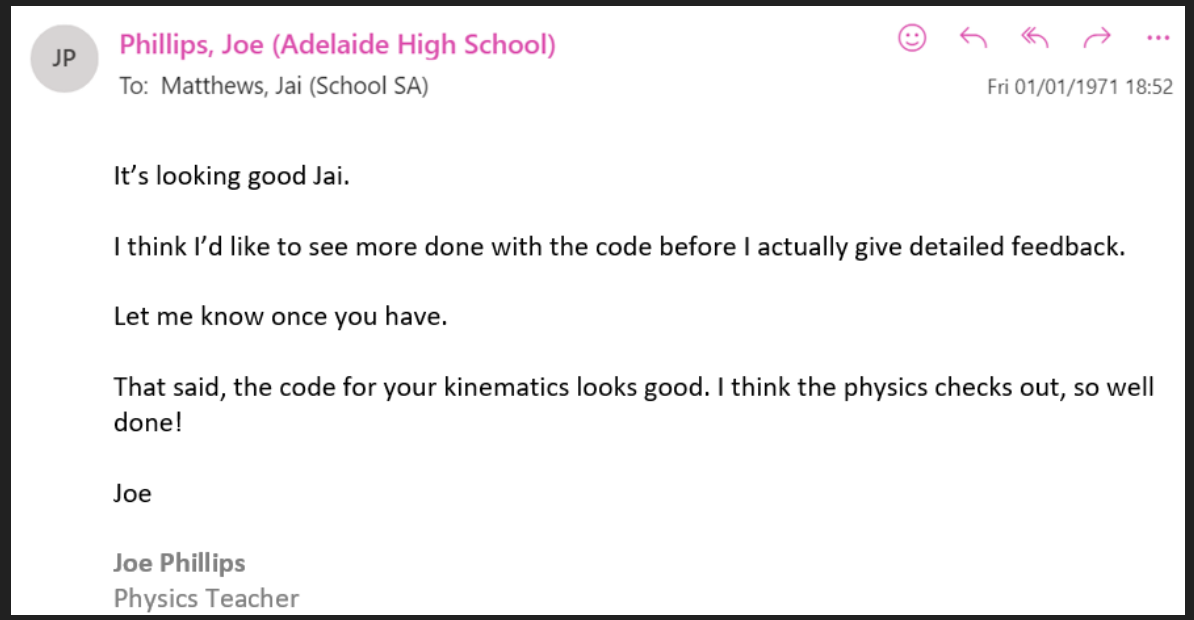
- I would like the code to handle everything using metres to allow for accurate calculations using SI units.
- BUT the HTML canvas handles things in pixels.
- For this reason, a simple conversion function is used that uses the ratio between the pixels and the known screen size.
- This function is called whenever things are displayed on screen.

```
// functions that convert between metres and pixels
function mTOpx(x) {
    return x * preview.scrollHeight / 6
}

function pxTOm(x) {
    return x * 6 / preview.scrollHeight
}
```

```
//draw ball!
c.beginPath()

//set up styling
c.strokeStyle = 'black'
c.fillStyle = 'black'
c.lineWidth = 1

//position is converted x and converted y using mTOpx()
c.arc(mTOpx(this.x), mTOpx(this.y), mTOpx(this.radius), 0, Math.PI * 2, false)

//put on canvas
c.stroke()
c.fill()
```

Feedback

- Feedback suggested that more needed to be done before much can be said.
- I was told that the physics looks good and it checks out.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

It's looking good Jai.

I think I'd like to see more done with the code before I actually give detailed feedback.

Let me know once you have.

That said, the code for your kinematics looks good. I think the physics checks out, so well done!

Joe

Joe Phillips
Physics Teacher

Evaluation

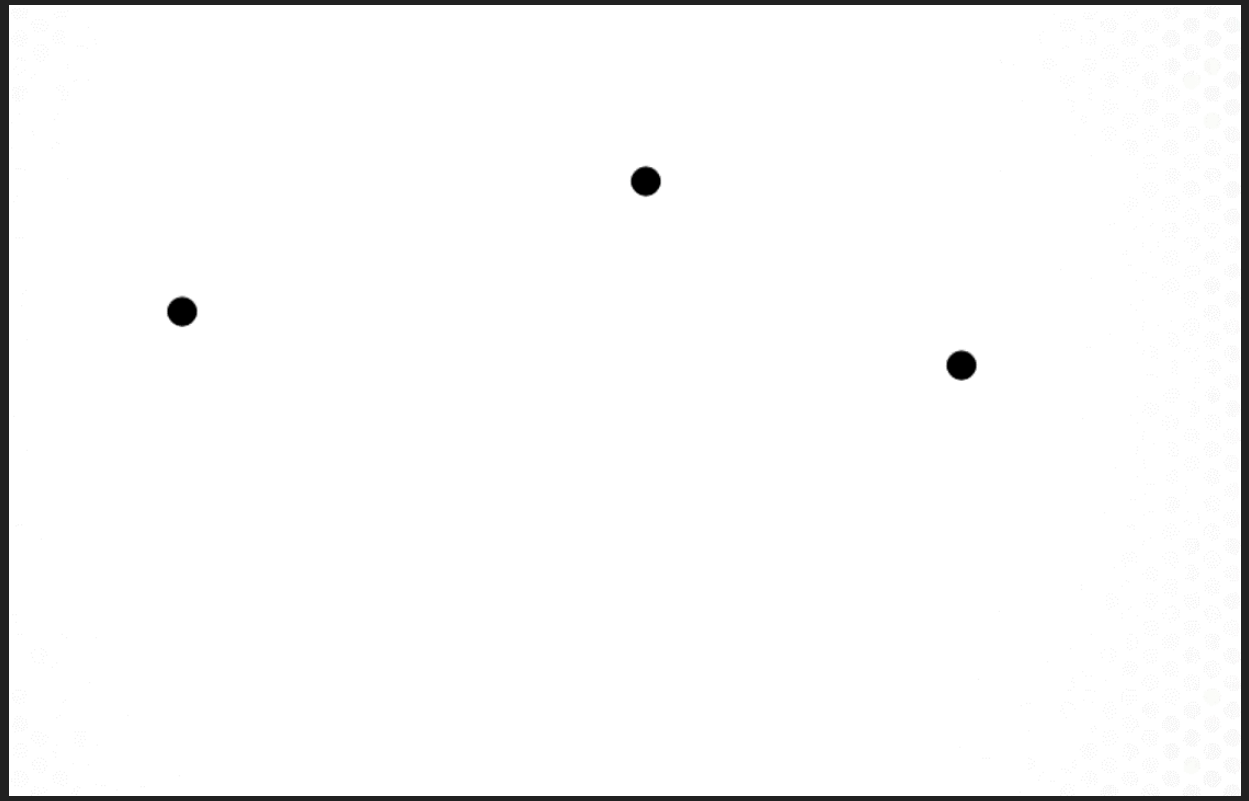
- Mostly, I think it works really well.
- That said, it isn't particularly good at handling different page zooms. If a page zooms out partway through the pixels of the sim window will increase and the ratio will become wrong.
- Other than this, I'm looking forward to working on more of the project!

Iteration 2

AT3 Major Project

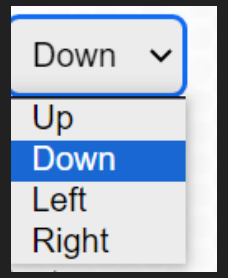
Overview

- I added gravity & air resistance to the code. Now, objects have forces acted upon them. This worked perfectly with the kinematic code I set up in the previous iteration.



Convert direction to ax & ay

- I allowed users to pick a gravitational direction from a drop down.
- This meant I needed code to turn this into ax & ay (which are the horizontal and vertical acceleration values)
- Instead of using a lookup table (the boring option) I used a arrays where every direction has the presence or absence of acceleration in a certain direction

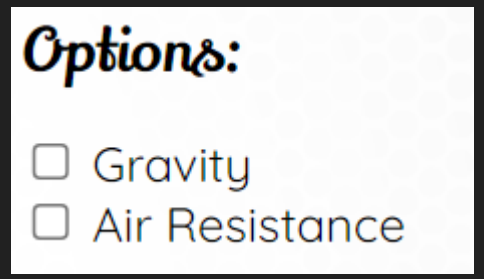


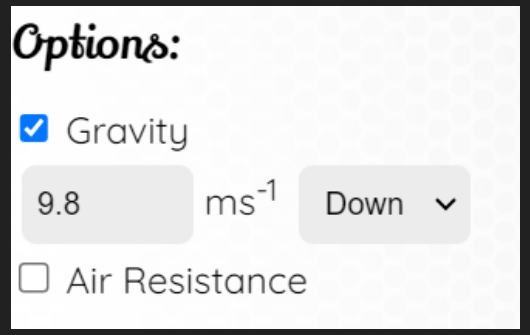

```
<select>
    <option value=[0,1]>Up</option>
    <option value=[0,-1] selected>Down</option>
    <option value=[-1,0]>Left</option>
    <option value=[1,0]>Right</option>
</select>
```

```
//get array value of selected direction
direction = JSON.parse(document.getElementById('gravityOp').querySelector('select').value)

//get the acceleration due to gravity magnitude
magnitude = document.getElementById('gravityOp').querySelector('input').value

//determine the x & y gravitational acceleration using these two values
xgrav = magnitude * direction[0]
ygrav = magnitude * direction[1]
```

Show/ Hide settings

- Because there were now settings for individual options within the editor, I needed code to show/hide them based upon whether they were checked.
- This was very simple. Each checkbox would pass itself through a function when onChange and then the options div would become visible.






```
function toggleOp(me) {
    document.getElementById(me.id + 'Op').classList.toggle('visible')
}
```

Air Resistance & Friction Formula

- Another hard challenge during this iteration was programming air resistance.
- Air resistance is a form of friction and thus obeys this law in physics:
- The force acts against the direction of velocity. I was unable to perfectly replicate this, so I instead reduced acceleration by a factor of the velocity and the cross section area (radius).
- Whilst this doesn't perfectly match the formula, it produces an equivalent result.

$$F_D = \frac{1}{2}\rho v^2 C_D A$$

F_D = drag

ρ = density of fluid

v = speed of the object relative to the fluid

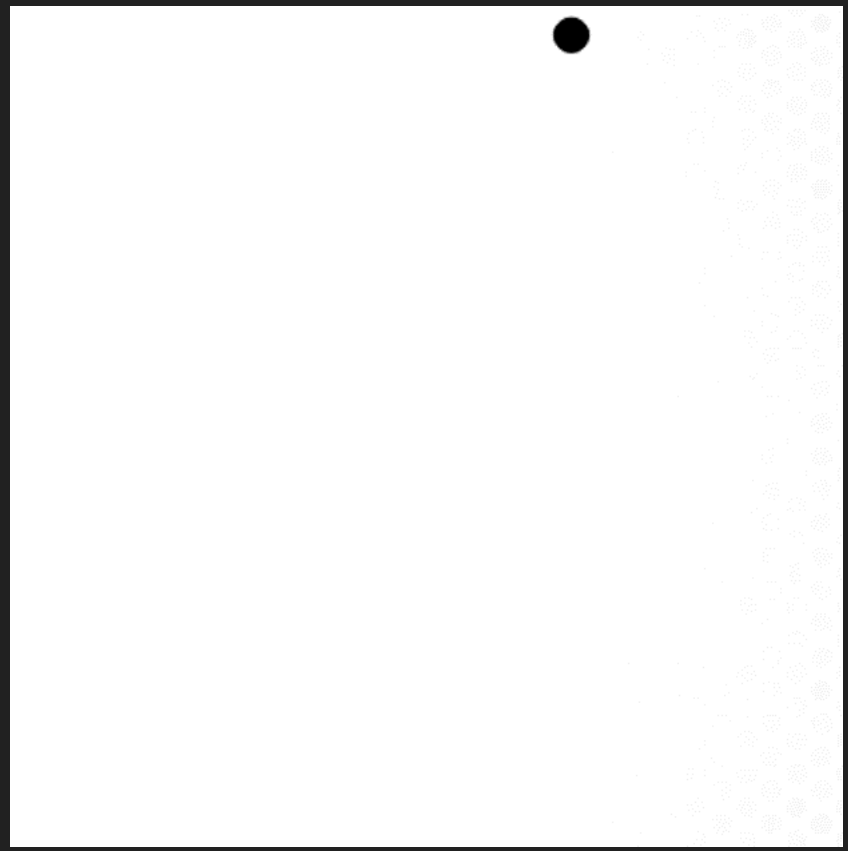
C_D = drag coefficient

A = cross sectional area

```
//lower acceleration due to air resistance
this.ax = this.ax - (friction * this.vx * this.radius)
this.ay = this.ay - (friction * this.vy * this.radius)
```

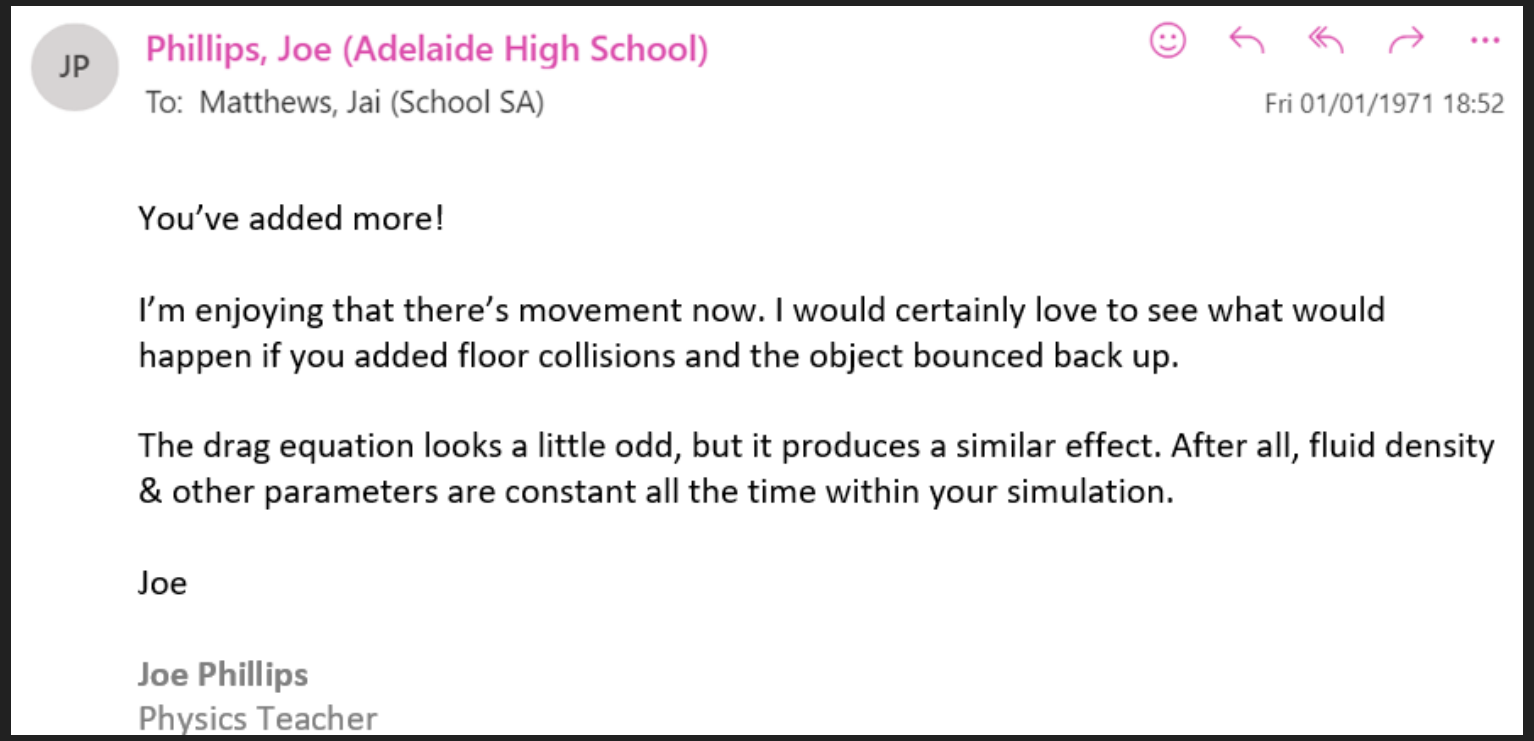
Consequence of code: Terminal Velocity

- Air resistance is the reason behind why objects have a terminal velocity & this effect can be seen in my simulation as a result of the implementation of drag.
- It's a bit difficult to see but look for the point where it stops accelerating.



Feedback

- The feedback for this iteration was mostly positive.
- It requested the inclusion of floor collisions which I then scheduled in for completion in iteration 4.
- It also confirmed my suspicion that the frictional equations weren't 100% accurate.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

You've added more!

I'm enjoying that there's movement now. I would certainly love to see what would happen if you added floor collisions and the object bounced back up.

The drag equation looks a little odd, but it produces a similar effect. After all, fluid density & other parameters are constant all the time within your simulation.

Joe

Joe Phillips
Physics Teacher

Evaluation

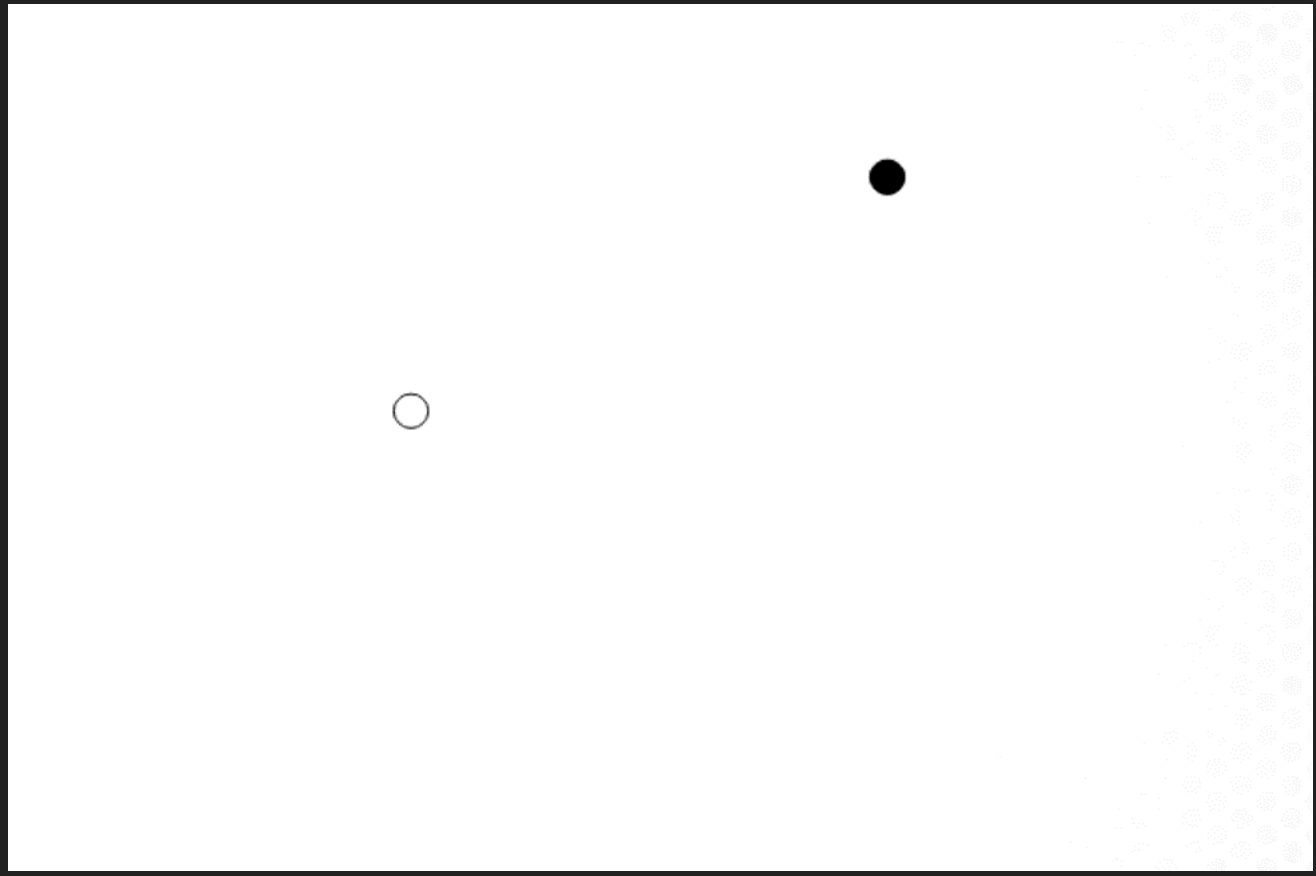
- I was quite happy after this iteration to see things moving about and functioning.
- I was still a bit concerned about the drag equation, but it looked mostly good to me.
- And I believe that the ability for users to change constants like acceleration due to gravity & coefficient of friction mean that the program is conforming to my desired general-purpose simulation that can be used for anything.

Iteration 3

AT3 Major Project

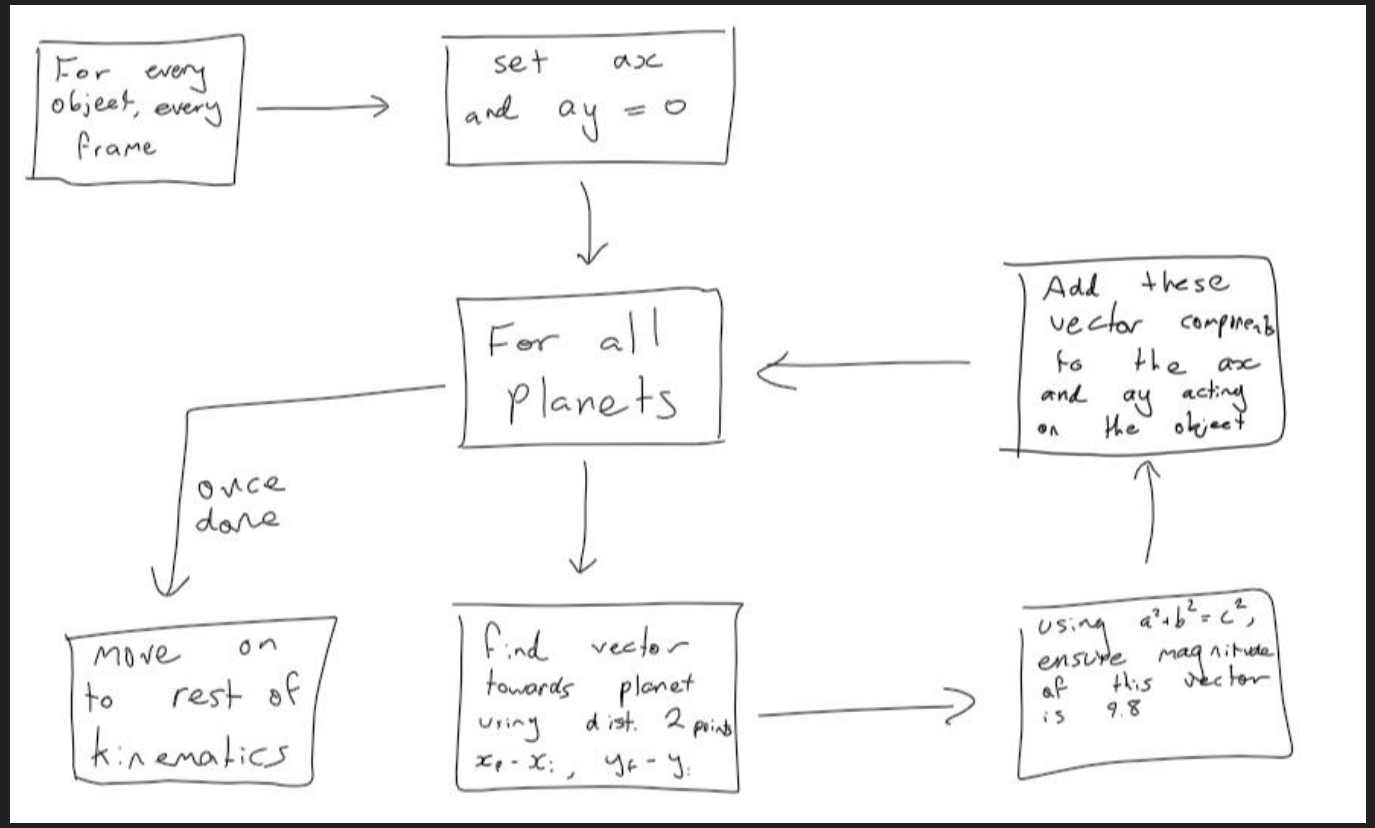
Overview

- For this iteration, I added planets to the simulation. As planned, this would essentially be a point that drew objects toward it as a centre of gravity.
- The planet is shown as a hollow ring (to differentiate it from a ball object).



Vector Addition Flowchart

- My next task was to plan how an object would detect every planet and go towards it.



Coding a planet list

- The previous flowchart mentioned a list of planets. To achieve this, every frame a list of planets would be written with each planets x & y positions. This code was done as part of the animation function.

```
//every frame:
//find all gravitational sources
gravSources = []

//go through each object,
for (i = 0; i < objects.length; i++) {
    //if the object is a planet,
    if (objects[i].type == 'Planet') {
        //add its x & y position to the afforementioned list
        gravSources.push([objects[i].x, objects[i].y])
    }
}
```

Coding vector addition

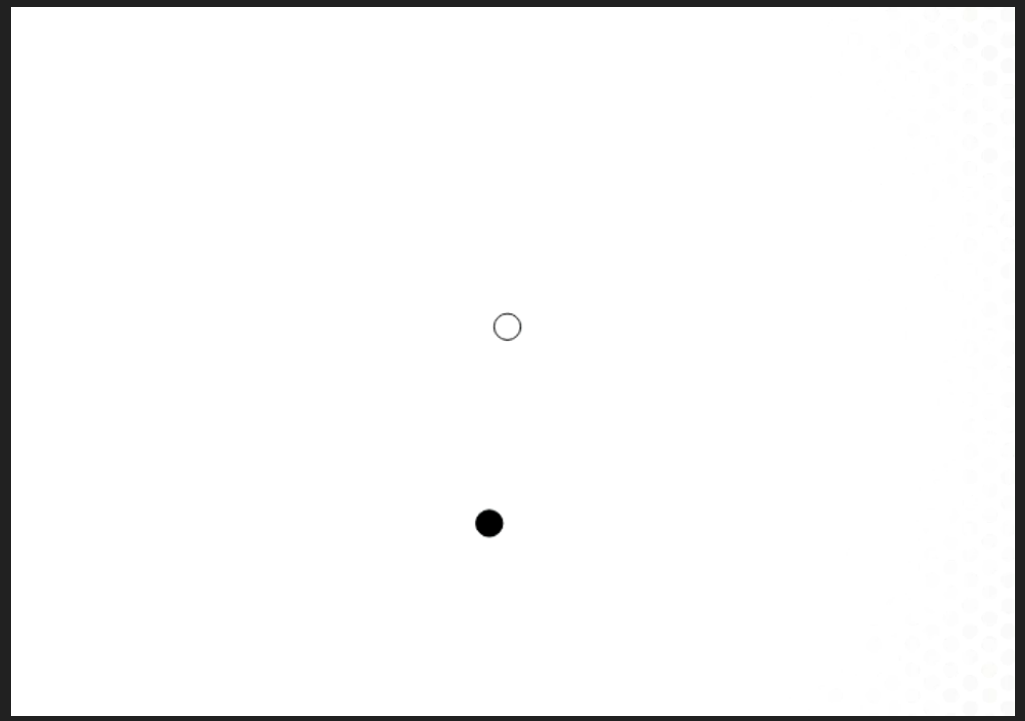
- The process shown in the flowchart essentially depicts vector addition; done by summing the components. The flowchart directly corresponds to this piece of code which is reliant on the list of gravitational sources shown on slide previous.

```
//find acceleration due to gravitational bodies
//for every gravitational source
for (this.k = 0; this.k < gravSources.length; this.k++) {
    //find the magnitude of the distance between the grav source and the object
    this.xmag = (gravSources[this.k][0] - this.x)
    this.ymag = (gravSources[this.k][1] - this.y)

    //alter the acceleration vector components of the object by this number.
    //multiply it by 9.8 and divide by the vector magnitude to ensure the vector's magnitude becomes 9.8
    this.ax += (this.xmag * 9.8) / pyth(this.xmag, this.ymag)
    this.ay -= (this.ymag * 9.8) / pyth(this.xmag, this.ymag)
}
```

```
//pythagorean theorem used for some vector calculations:
function pyth(a, b) {
    return Math.sqrt(a**2 + b**2)
}
```

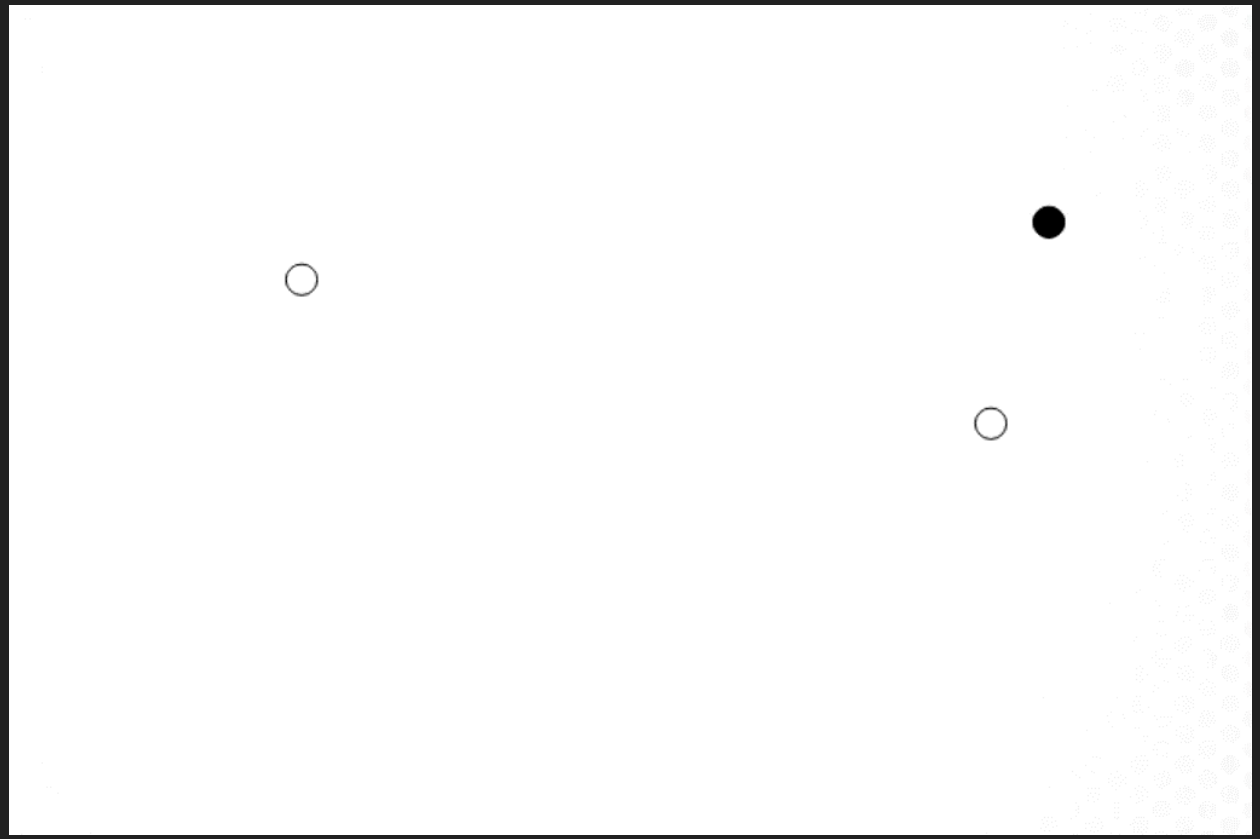
Consequence of code: Orbits

- One expected consequence of the code is that it could allow for uniform circular motion. In other words; an orbit. This uses the fact that the acceleration is roughly perpendicular to the velocity of an object.
- As expected, any object with a velocity in a different direction to its acceleration will move elliptically around a singular gravitational body.



Consequence of code: Rest

- Another consequence of this code that was not expected is that an object will come to a rest state between two planets. This is particularly the case when there is drag.
- Whilst it was unexpected, it makes sense as this represents the presence of a balanced force.



Evaluation

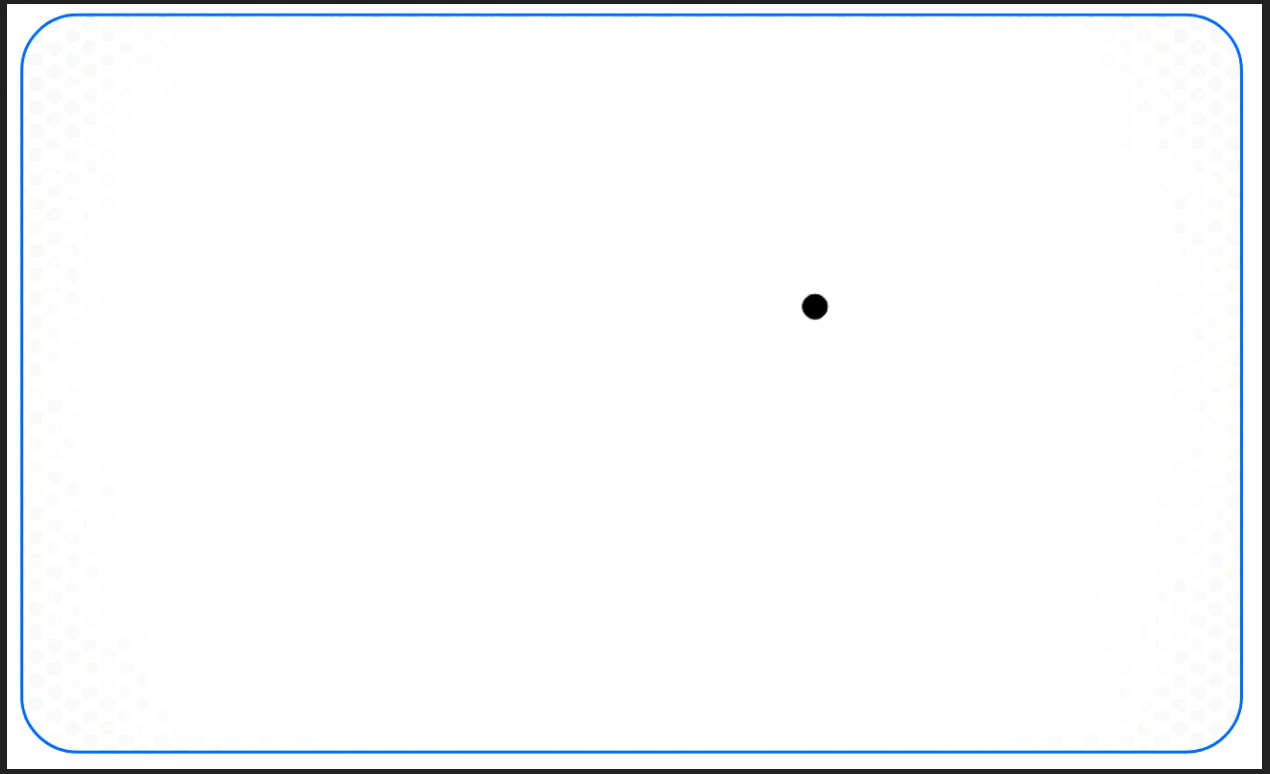
- It really works, but it has flaws.
- The primary flaw is that acceleration due to gravity doesn't increase with proximity. This was necessary to mitigate the following issue which was…
- Using both planetary gravity and a downward gravity is a juxtaposition that wouldn't really exist in the real world. It doesn't make too much sense to use both at the same time. For this reason, teachers will probably have to tell their students which of these two features to use; not both.
- Whilst the above doesn't make sense, the fact that both can be used in the same software constitutes an innovative feature that I'm glad this simulation has.

Iteration 4

AT3 Major Project

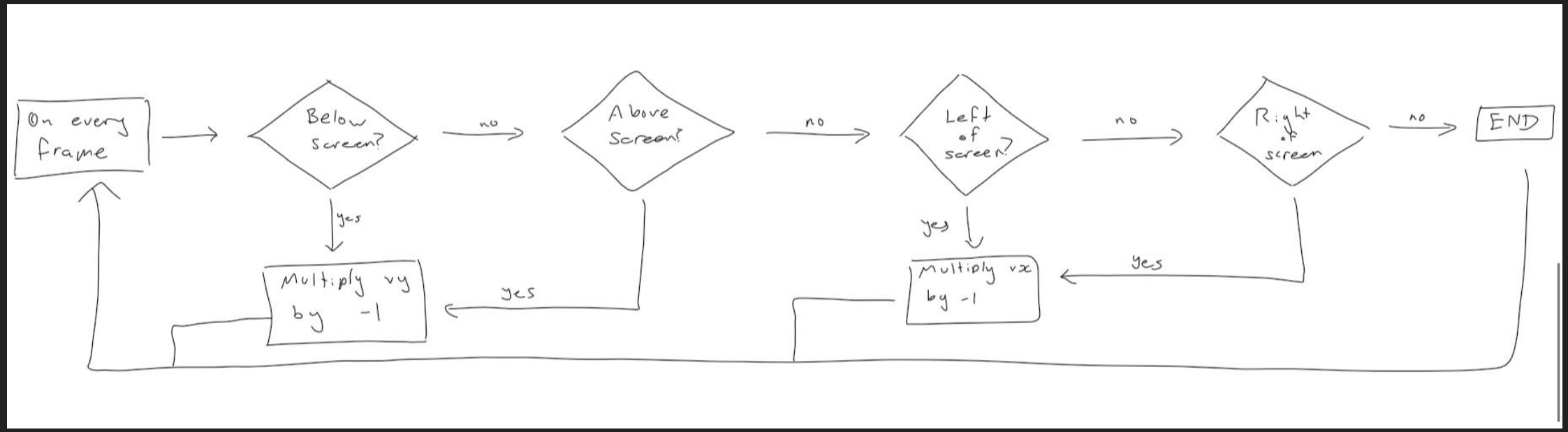
Overview

- For this iteration, I added wall collisions to the simulation. Now, objects can repel against the wall should you turn on wall collisions.
- I chose to do this because feedback I had received from my stakeholder indicated this would be a good idea.



Collision Flowchart

Below is a flowchart for code that handles collisions.



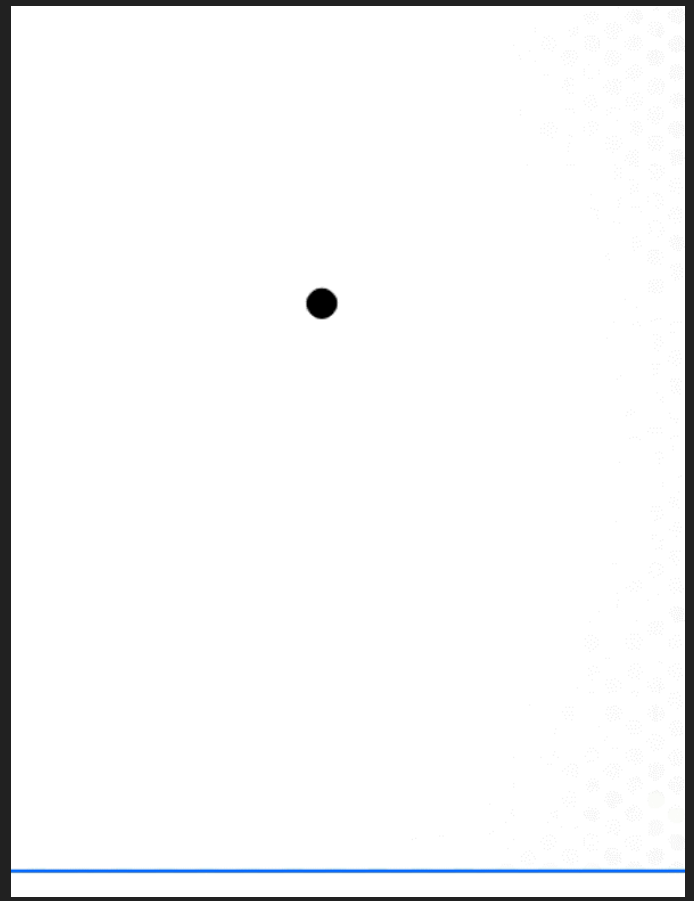
Collision Code

- I altered the code from the flowchart to handle each conditional separately rather than in a chain for the code.
- This would allow for the possibility of an object moving into the corner.

```
//bounce off walls
if (wallc) { //check if wall collisions are enabled
    //check for bottom of screen
    //radius is added as I want the edge of the ball to be affected, not the centre of the ball.
    if (this.y + this.radius > pxTOm(canvas.height)) {
        this.vy = -1 * this.vy
    }
    //bottom of screen
    if (this.y - this.radius < 0) {
        this.vy = -1 * this.vy
    }
    //screen right
    if (this.x + this.radius > pxTOm(canvas.width)) {
        this.vx = -1 * this.vx
    }
    //screen left
    if (this.x - this.radius < 0) {
        this.vx = -1 * this.vx
    }
}
```

Bug: Glitching through floor

- One pretty big issue is that, well… the code didn't work.
- This was because it detected the ball being below the floor multiple times and would reverse the velocity multiple times and on every frame.
- This was a pretty big issue.



Solution to bug

- The solution was to teleport the object to just above the floor when it went below the ground.
- This fixed it :)

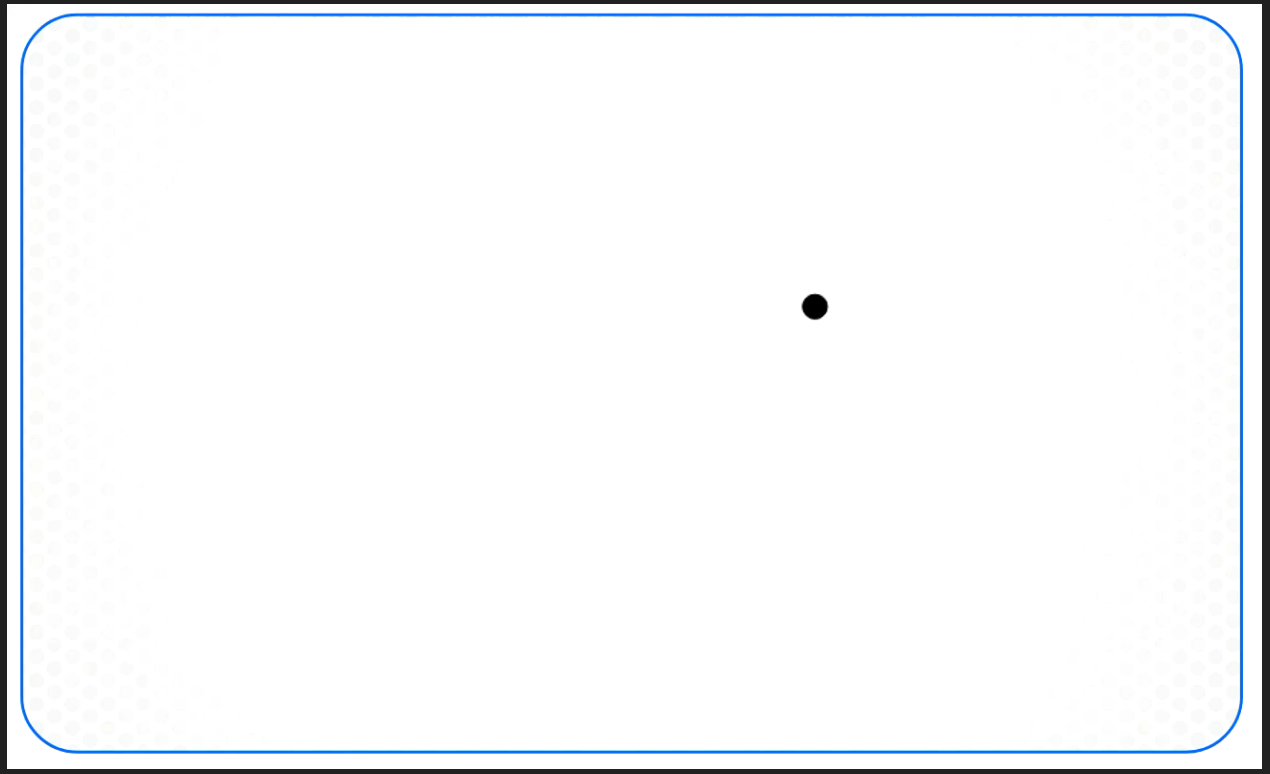
```
//bounce off walls
if (wallc) { //check if wall collisions are enabled
    //check for bottom of screen
    //radius is added as I want the edge of the ball to be affected, not the centre of the ball.
    if (this.y + this.radius > pxTOm(canvas.height)) {
        //telport to correct position to prevent glitching
        this.y = pxTOm(canvas.height) - this.radius
        //reverse vertical velocity
        this.vy = -1 * this.vy
    }
    //bottom of screen
    if (this.y - this.radius < 0) {
        //telport
        this.y = this.radius
        //reverse
        this.vy = -1 * this.vy
    }
    //screen right
    if (this.x + this.radius > pxTOm(canvas.width)) {
        //teleport
        this.x = pxTOm(canvas.width) - this.radius
        //reverse
        this.vx = -1 * this.vx
    }
    //screen left
    if (this.x - this.radius < 0) {
        //teleport
        this.x = this.radius
        //reverse
        this.vx = -1 * this.vx
    }
}
```

Does a rest state need to be coded?

- I wanted objects to rest on the ground once their kinetic energy has been worn out. This would be the case if there was air resistance.
- I wondered if I would need to detect the object having a small enough velocity for me to set it as ‘at rest’ and not move.
- However, I discovered this proved to not be necessary. The recording at the start showed that the rules I’ve already stated were sufficient in creating a rest state without extra code.

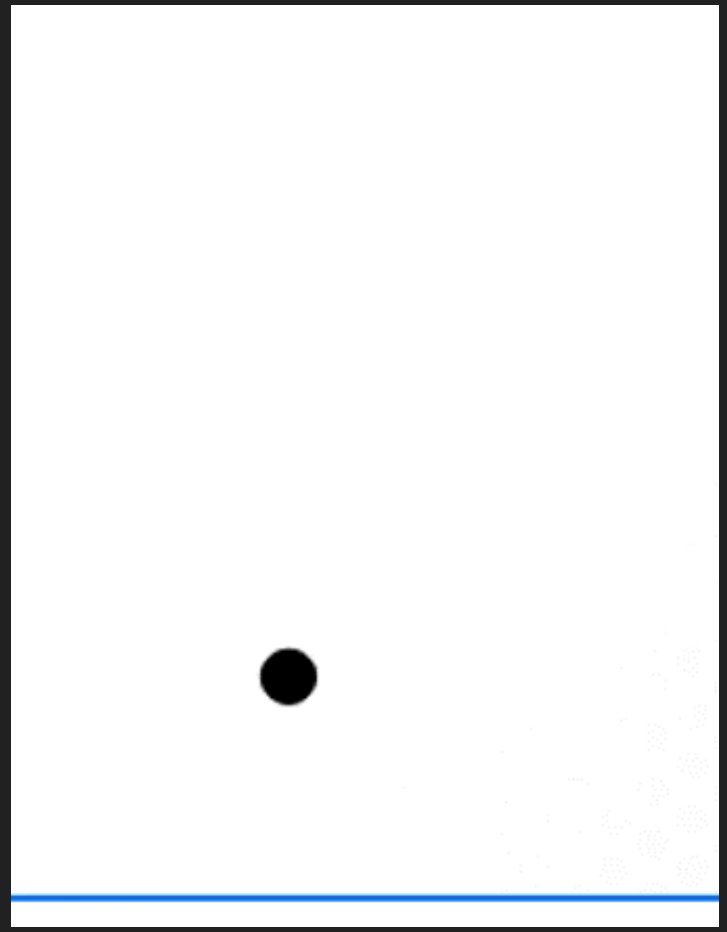
Consequence of code: Parabolic motion

- One nice consequence of the code (that I expected) is the presence of clear parabolic motion in the movement of the balls.
- This makes sense as a parabola is defined as a quadratic y component and a linear x component which is what is achieved by my code.



Bug: Floating upwards when gravity off

- Another bug is that when I turn off gravity, an object at rest begins to move upward arbitrarily.
- This one really puzzled me.

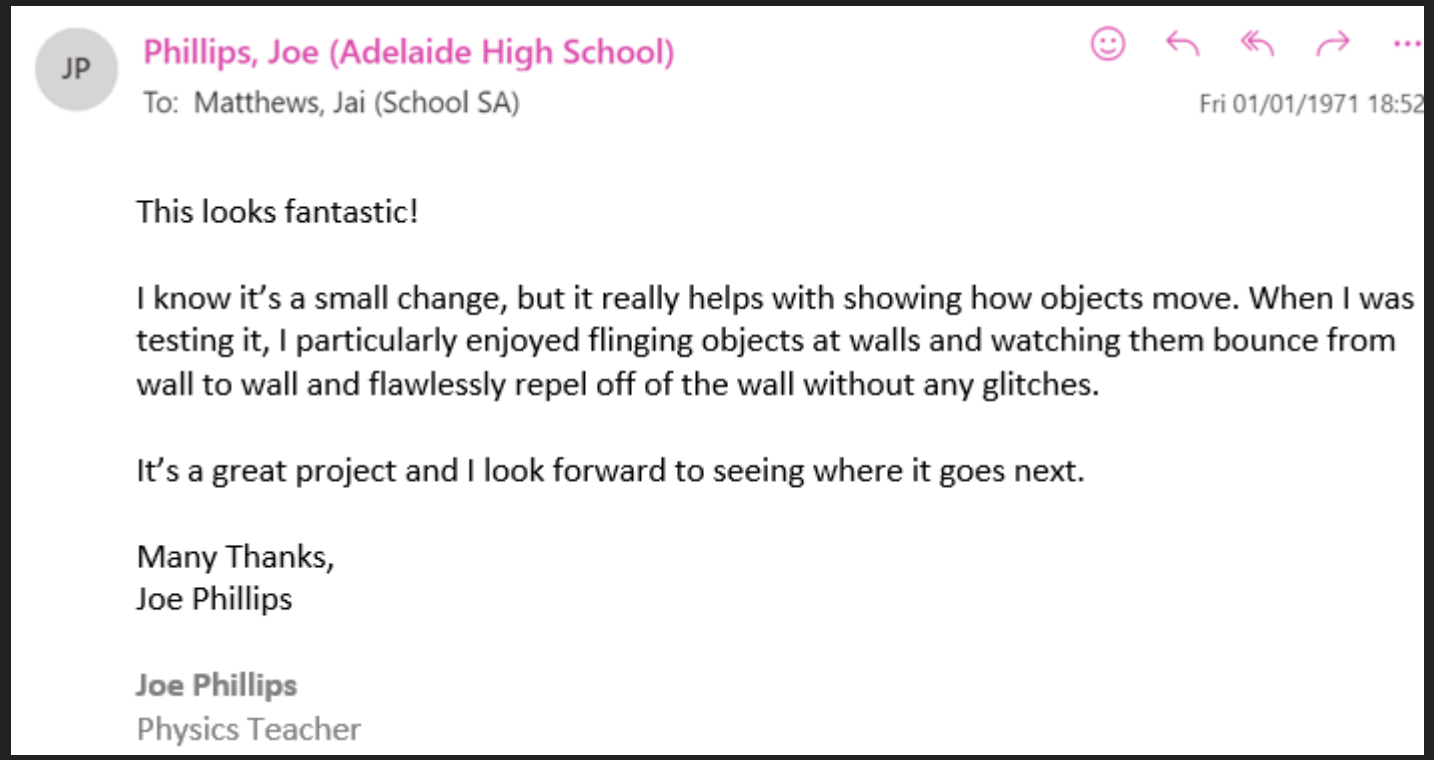


Solution: wait it's fine nvm

- Eventually I concluded that if gravity really were to turn off, objects would drift upward slightly due to discrepancies in their shape, small air currents, or the lack of a downward force.
- For this reason, I left this feature in the code.

Feedback

- The feedback I received for this iteration was very positive and enjoyed the fact that you could now perform far more interesting and engaging simulations.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

This looks fantastic!

I know it's a small change, but it really helps with showing how objects move. When I was testing it, I particularly enjoyed flinging objects at walls and watching them bounce from wall to wall and flawlessly repel off of the wall without any glitches.

It's a great project and I look forward to seeing where it goes next.

Many Thanks,
Joe Phillips

Joe Phillips
Physics Teacher

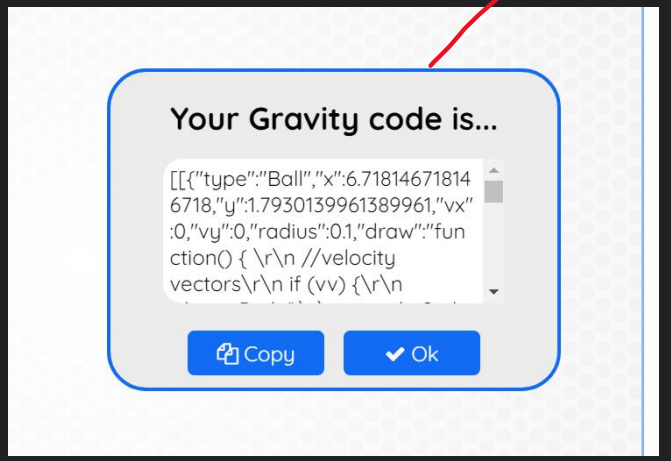
Iteration 5

AT3 Major Project

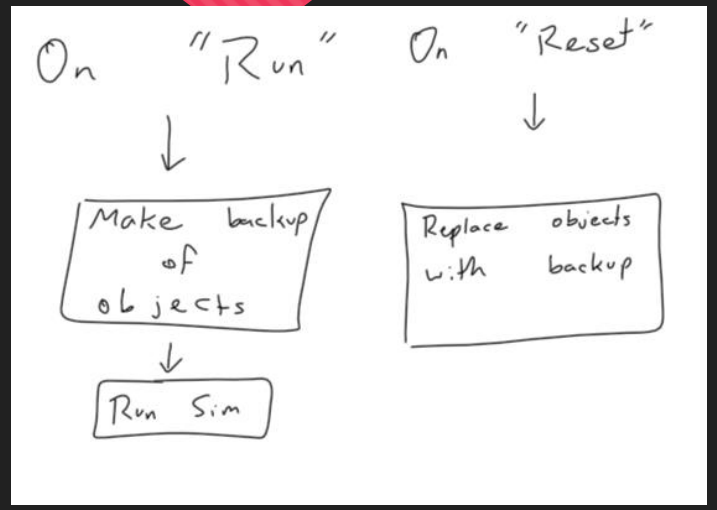
Overview

In this iteration, I added the ability some helpful saving abilities:

- Save & load the state of the simulation
- Completely clear the simulation
- Reset the simulation to before you ran it.



Resetting flowchart.




```
function run() {
    //stringify all the objects when running the function.
    //I made sure this only works EITHER when you're in the editor or its the first backup
    //this is because I will soon add a viewer when I only want it to reset to factory settings, not when it was last run
    if (objectsbackup.length == 0 || editing) {
        objectsbackup = JSON.parse(JSON.stringify(objects));
    }
```

create backup

```
function reset() {
    //the code below SHOULD work
    temp = JSON.parse(JSON.stringify(objectsbackup))

    //but it also needs this.
    //that's because the stringified/parsed object isn't identical... somehow
    //my solution is to manually look for the corresponding parameters as below
    //for every object
    for (i = 0; i < objects.length; i++) {
        //and every parameter of that object
        for (let j in temp[i]) {
            //replace the main object list with that of the backup one
            objects[i][j] = temp[i][j]
        }
    }
```

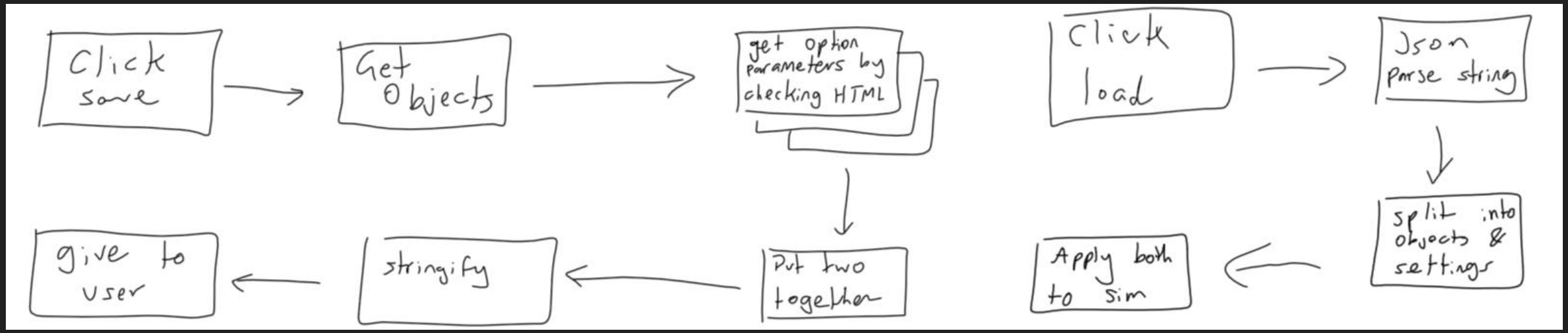
get backup

fix issues

- I thought, as many would, that coding this would be simple. However, I found that when the sim updated the objects, it would also update the backup. I had created the backup with reference.
- Many solutions documented online wouldn't work for the nested nature of my objects. This was a conundrum.
- Eventually I had to stringify and then parse the function to make it work.

Saving & Loading flowchart

- Flowchart for saving & loading the simulation.



Saving code

- I had a slight issue with the saving code. I wanted to stringify the array of simulation objects, but this was not working.
- I looked into it and it turned out that the issue was that my object included functions; which could not be stringified.
- Instead, I had to convert all functions into strings first.
- I looked into using APIs that could handle this, but converting it to a string was less work and was fairly innovative code.
- The load code more or less just does the inverse of this.

```
function save() {
    //get settings from the simulation. This is done by looking at the checkboxes
    settings = [
        [
            document.getElementById('gravity').checked,
            document.getElementById('gravityOp').querySelector('input').value,
            document.getElementById('gravityOp').querySelector('select').value
        ],
        [
            document.getElementById('drag').checked,
            document.getElementById('dragOp').querySelector('input').value
        ],
        document.getElementById('wall').checked,
        document.getElementById('collide').checked,
        document.getElementById('transfer').checked,
        [
            document.getElementById('vvector').checked,
            document.getElementById('vvectorOp').querySelector('input').value,
        ],
        [
            document.getElementById('avector').checked,
            document.getElementById('avectorOp').querySelector('input').value,
        ],
    ]

    //create a JSON string for the objects.
    jsonString = JSON.stringify([objects, settings], (key, value) => {
        //if a function is encountered, convert it to a string so it can be converted to JSON
        if (typeof value === 'function') {
            return value.toString();
        }
        //if not, continue as usual
        return value;
    });

    //show the saving window
    document.getElementById('savePopup').style.display = 'block'
    document.getElementById('main').classList.toggle('disabled') //this just disables the rest of the page

    //and display the newly generated code on the webpage
    document.getElementById('saveBox').innerText = jsonString
}
```

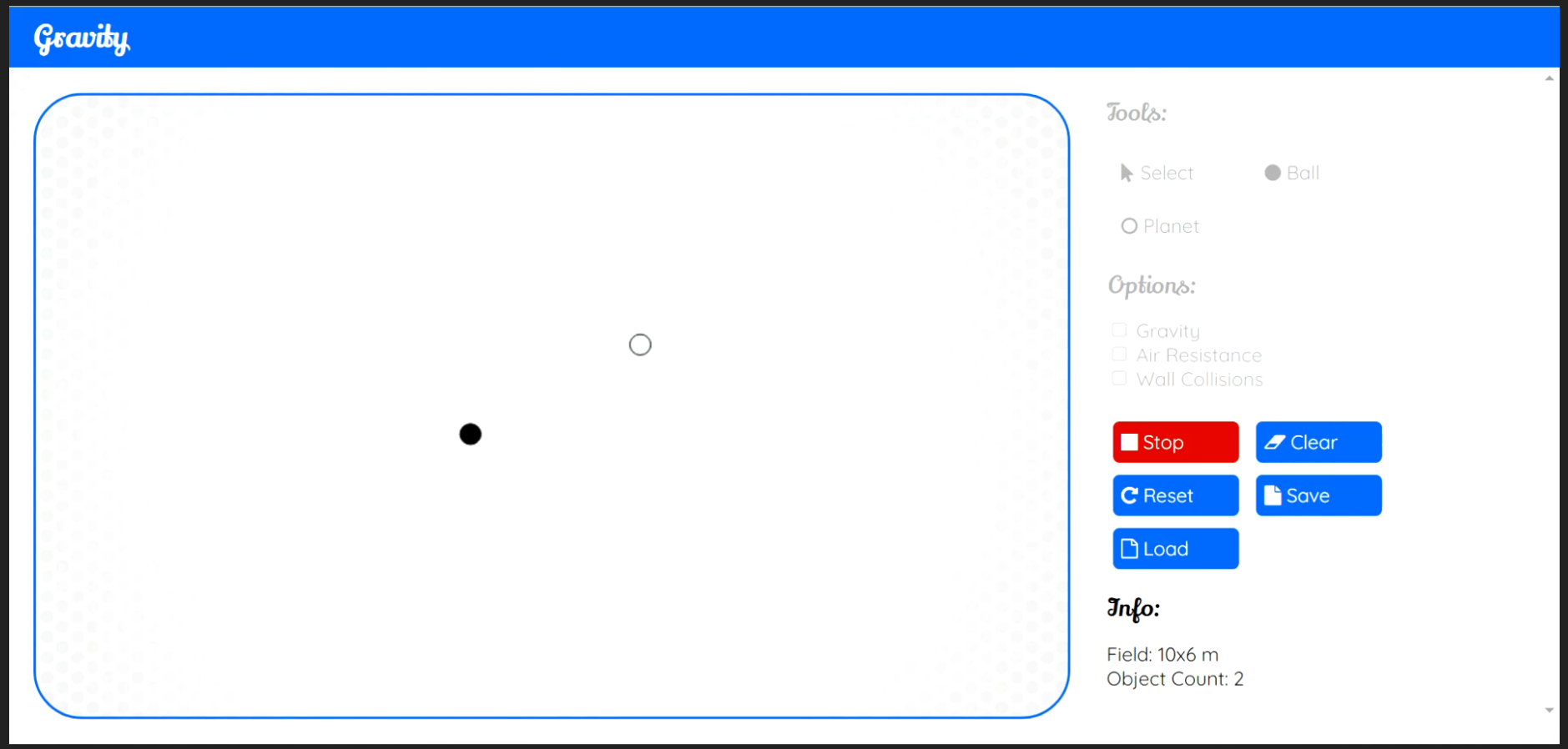
determining settings

sorting out functions

stringifying

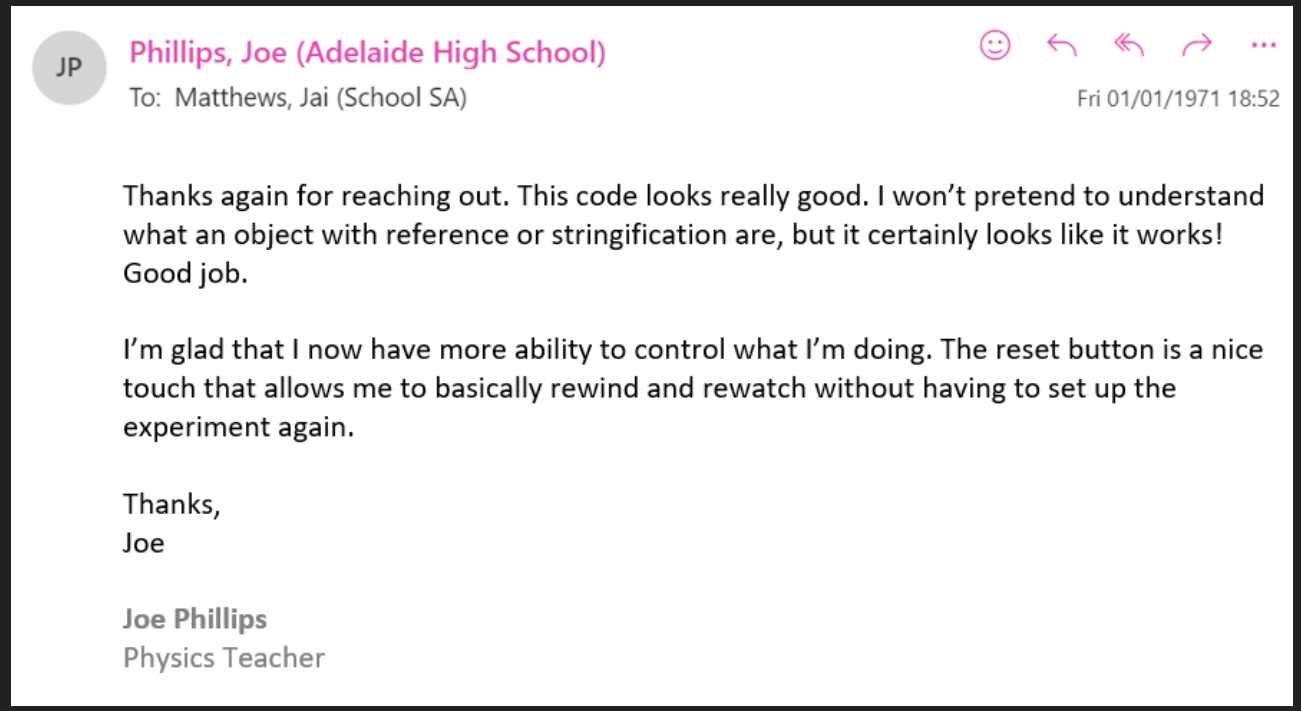
display stuff

Save & Load demo



Feedback

- Feedback for this iteration was wholly positive!
- My stakeholder particularly enjoyed the wealth of options & control this gives the user.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

Thanks again for reaching out. This code looks really good. I won't pretend to understand what an object with reference or stringification are, but it certainly looks like it works! Good job.

I'm glad that I now have more ability to control what I'm doing. The reset button is a nice touch that allows me to basically rewind and rewatch without having to set up the experiment again.

Thanks,
Joe

Joe Phillips
Physics Teacher

Evaluation

- Whilst I mostly agree with Joe that this is a positive iteration, there is one flaw I am quite aware of.
- I do not like that the save codes are SO long. Like, the screenshot on the right is literally just the 2-body orbit from before.
- If you think about it, for every object it's saving the function that controls its movement. This really doesn't seem necessary because when read back in, the program will already know how to control an objects' movement.
- It's something that can be worked on in future, if need be, but for now it is sufficient. As it works.

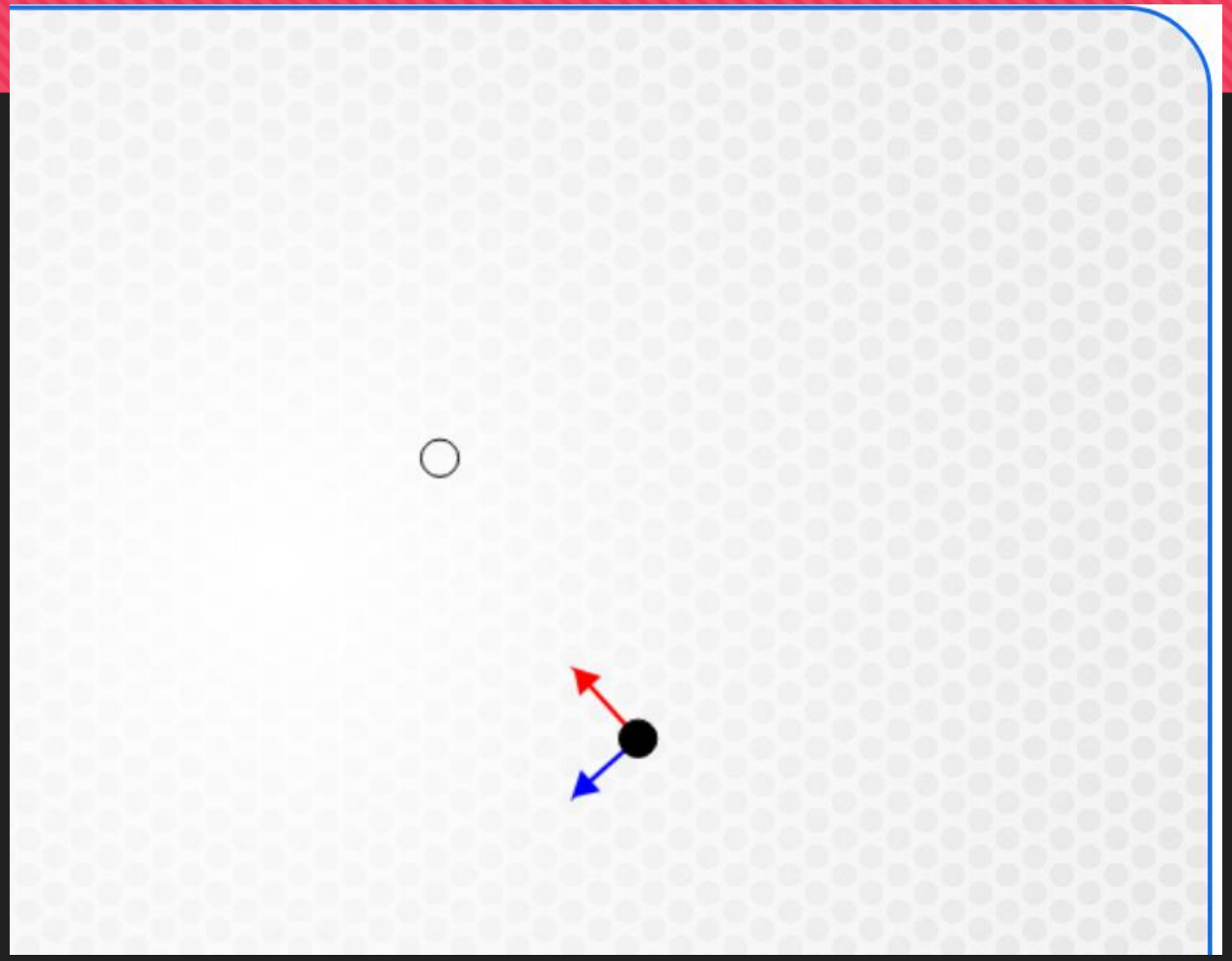
[[{"type":"Ball","x":7.153339210474805,"y":3.5923210195704174,"vx":-2.647443738395918,"vy":-3.7029953312826094,"radius":0.1,"draw":"function() { \r\n //velocity vectors\r\n if (vv) {\r\n c.beginPath()\r\n c.strokeStyle = \"blue\"\r\n c.lineWidth = 2\r\n c.moveTo(mTOpx(this.x),mTOpx(this.y))\r\n c.lineTo(mTOpx(this.x + (this.vx *(1/vvs))),mTOpx(this.y - (this.vy *(1/vvs))))\r\n\r\n this.dx = mTOpx(this.x + (this.vx *(1/vvs))) - mTOpx(this.x);\r\n this.dy = mTOpx(this.y - (this.vy *(1/vvs))) - mTOpx(this.y);\r\n this.angle = Math.atan2(this.dy, this.dx);\r\n c.lineTo(mTOpx(this.x + (this.vx *(1/vvs))), mTOpx(this.y - (this.vy *(1/vvs))));\r\n c.lineTo(mTOpx(this.x + (this.vx *(1/vvs))) - headlen * Math.cos(this.angle - Math.PI / 6), mTOpx(this.y - (this.vy *(1/vvs))) - headlen * Math.sin(this.angle - Math.PI / 6));\r\n c.lineTo(mTOpx(this.x + (this.vx *(1/vvs))) - headlen * Math.cos(this.angle + Math.PI / 6), mTOpx(this.y - (this.vy *(1/vvs))) - headlen * Math.sin(this.angle + Math.PI / 6));\r\n c.lineTo(mTOpx(this.x + (this.vx *(1/vvs))), mTOpx(this.y - (this.vy *(1/vvs))));\r\n\r\n\r\n c.fillStyle = 'blue'\r\n c.fill()\r\n c.stroke()\r\n }\r\n\r\n\r\n //acceleration vectors\r\n if (av) {\r\n c.beginPath()\r\n c.strokeStyle = \"red\"\r\n c.lineWidth = 2\r\n c.moveTo(mTOpx(this.x),mTOpx(this.y))\r\n c.lineTo(mTOpx(this.x + (this.ax *(1/avs))),mTOpx(this.y - (this.ay *(1/avs))))\r\n\r\n\r\n this.dx = mTOpx(this.x + (this.ax *(1/avs))) - mTOpx(this.x);\r\n this.dy = mTOpx(this.y - (this.ay *(1/avs))) - mTOpx(this.y);\r\n this.angle = Math.atan2(this.dy, this.dx);\r\n c.lineTo(mTOpx(this.x + (this.ax *(1/avs))), mTOpx(this.y - (this.ay *(1/avs))));\r\n c.lineTo(mTOpx(this.x + (this.ax *(1/avs))) - headlen * Math.cos(this.angle - Math.PI / 6), mTOpx(this.y - (this.ay *(1/avs))) - headlen * Math.sin(this.angle - Math.PI / 6));\r\n c.lineTo(mTOpx(this.x + (this.ax *(1/avs))) - headlen * Math.cos(this.angle + Math.PI / 6), mTOpx(this.y - (this.ay *(1/avs))) - headlen * Math.sin(this.angle + Math.PI / 6));\r\n c.lineTo(mTOpx(this.x + (this.ax *(1/avs))), mTOpx(this.y - (this.ay *(1/avs))));\r\n\r\n\r\n c.fillStyle = 'red'\r\n c.fill()\r\n c.stroke()\r\n }\r\n \r\n //draw ball!\r\n c.beginPath()\r\n\r\n //set up styling\r\n c.strokeStyle = 'black'\r\n c.fillStyle = 'black'\r\n c.lineWidth = 1\r\n\r\n //position is converted x and converted y using mTOpx()\r\n c.arc(mTOpx(this.x), mTOpx(this.y), mTOpx(this.radius), 0, Math.PI * 2, false)\r\n\r\n //put on canvas\r\n c.stroke()\r\n c.fill()\r\n \r\n }","update":"function() {\r\n //get acceleration from gravity\r\n this.ax = xgrav\r\n this.ay = ygrav\r\n \r\n //find acceleration due to gravitational bodies\r\n //for every gravitational source\r\n for (this.k = 0; this.k < gravSources.length; this.k++) {\r\n //find the magnitude of the distance between the grav source and the object\r\n this.xmag = (gravSources[this.k][0] - this.x)\r\n this.ymag = (gravSources[this.k][1] - this.y)\r\n\r\n //alter the acceleration vector components of the object by this number.\r\n //multiply it by 9.8 and divide by the vector magnitude to ensure the vector's magnitude becomes 9.8\r\n this.ax += (this.xmag * 9.8) / pyth(this.xmag, this.ymag)\r\n this.ay -= (this.ymag * 9.8) / pyth(this.xmag, this.ymag)\r\n }\r\n\r\n\r\n //lower acceleration due to air resistance\r\n this.ax = this.ax - (friction * this.vx)\r\n this.ay = this.ay - (friction * this.vy)\r\n\r\n\r\n\r\n //calculate velocity & position\r\n this.vx += this.ax/fps\r\n this.vy += this.ay/fps\r\n\r\n\r\n this.x += this.vx/fps\r\n this.y -= this.vy/fps\r\n\r\n //bounce off walls\r\n if (wallc) { //check if wall collisions are enabled\r\n //check for bottom of screen\r\n //radius is added as I want the edge of the ball to be affected, not the centre of the ball.\r\n if (this.y + this.radius > pxTOm(canvas.height)) {\r\n //telport to correct position to prevent glitching\r\n this.y = pxTOm(canvas.height) - this.radius\r\n //reverse vertical velocity\r\n this.vy = -1 * this.vy\r\n }\r\n //bottom of screen\r\n if (this.y - this.radius < 0) {\r\n //telport\r\n this.y = this.radius\r\n //reverse\r\n this.vy = -1 * this.vy\r\n }\r\n //screen right\r\n if (this.x + this.radius > pxTOm(canvas.width)) {\r\n //teleport\r\n this.x = pxTOm(canvas.width) - this.radius\r\n //reverse\r\n this.vx = -1 * this.vx\r\n }\r\n //screen left\r\n if (this.x - this.radius < 0) {\r\n //teleport\r\n this.x = this.radius\r\n //reverse\r\n this.vx = -1 * this.vx\r\n }\r\n }\r\n\r\n\r\n //done!\r\n this.draw()\r\n \r\n }","ax":-8.163092707670161,"ay":5.422537915587336,"k":1,"xmag":-1.5057886174718211,"ymag":-1.0002576429676768},{"type":"Planet","x":5.77992277992278,"y":2.40691361003861,"vx":0,"vy":0,"mass":9.8,"fixed":true,"radius":0.1,"draw":"function() {\r\n c.beginPath()\r\n c.arc(mTOpx(this.x), mTOpx(this.y), mTOpx(this.radius), 0, Math.PI * 2, false)\r\n c.strokeStyle = 'black'\r\n c.lineWidth = 1\r\n c.stroke()\r\n }","update":"function() {\r\n if (!this.fixed) {\r\n this.ax = xgrav\r\n this.ay = ygrav\r\n\r\n this.vx += this.ax/fps\r\n this.vy += this.ay/fps\r\n\r\n this.x += this.vx/fps\r\n this.y -= this.vy/fps\r\n }\r\n\r\n\r\n this.draw()\r\n \r\n }"}],[[false,"9.8","[0,-1]"],[false,"0.5"],false,false,false,[false,"10"],[false,"20"]]]

Iteration 6

AT3 Major Project

Overview

- For this iteration, I added vectors to the simulation.
- The red and blue represent any object's acceleration and velocity, respectively.



Vector Pseudocode

- Drawing the lines that were the vectors would be aided by my existing kinematic code.
- I began with some pseudocode to show my plans for how to do this.

```
Every Frame, for every object:

    //for the velocity vector, draw a line between an objects' position and its position plus its velocity
    Place line cursor at (this.x, this.y)
    Start drawing line
    Move line cursor to (this.x + this.vx, this.y + this.vy)
    Add arrowhead
    End drawing line

    //do the same thing for acceleration
    Place line cursor at (this.x, this.y)
    Start drawing line
    Move line cursor to (this.x + this.ax, this.y + this.ay)
    Add arrowhead
    End drawing line

    Move on to the next object
```

Vector Code

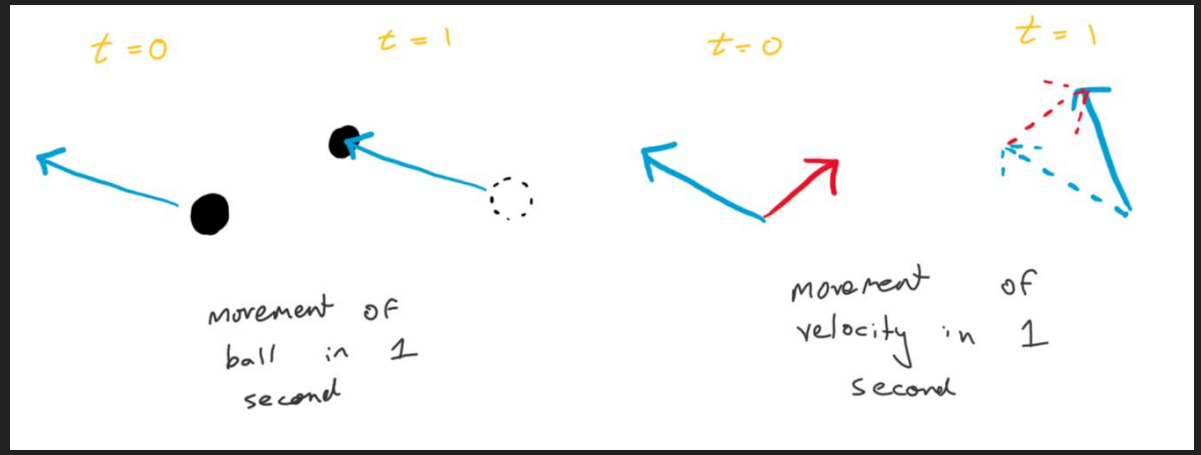
- The exact code is shown that achieves this (left).
- However this code doesn't do arrowheads.
- For that, I had to appropriate others' code that essentially does some complex trigonometry to determine the points where the arrows should be. (right)

```
this.draw = function() {
    //velocity vectors
    if (vv) {
        c.beginPath()
        c.strokeStyle = "blue"
        c.lineWidth = 2
        c.moveTo(mTOpx(this.x),mTOpx(this.y))
        c.lineTo(mTOpx(this.x + (this.vx)),mTOpx(this.y - (this.vy)))

        c.fillStyle = 'blue'
        c.fill()
        c.stroke()
    }

    //acceleration vectors
    if (av) {
        c.beginPath()
        c.strokeStyle = "red"
        c.lineWidth = 2
        c.moveTo(mTOpx(this.x),mTOpx(this.y))
        c.lineTo(mTOpx(this.x + (this.ax)),mTOpx(this.y - (this.ay)))

        c.fillStyle = 'red'
        c.fill()
        c.stroke()
    }
```

```
//velocity vectors
if (vv) {
    //my code is below
    c.beginPath()
    c.strokeStyle = "blue"
    c.lineWidth = 2
    c.moveTo(mTOpx(this.x),mTOpx(this.y))
    c.lineTo(mTOpx(this.x + (this.vx)),mTOpx(this.y - (this.vy)))

    //a version of this code was initially shared on stack overflow.
    //I altered it to make the arrow filled in, differently sized, and work within an object
    //basically all it does is some complex trigonometry to determine the arrow position
    this.dx = mTOpx(this.x + (this.vx)) - mTOpx(this.x);
    this.dy = mTOpx(this.y - (this.vy)) - mTOpx(this.y);
    this.angle = Math.atan2(this.dy, this.dx);
    c.lineTo(mTOpx(this.x + (this.vx)), mTOpx(this.y - (this.vy)));
    c.lineTo(mTOpx(this.x + (this.vx)) - headlen * Math.cos(this.angle - Math.PI / 6), mTOpx(this.y - (this.vy)) - he
    c.lineTo(mTOpx(this.x + (this.vx)) - headlen * Math.cos(this.angle + Math.PI / 6), mTOpx(this.y - (this.vy)) - he
    c.lineTo(mTOpx(this.x + (this.vx)), mTOpx(this.y - (this.vy)));

    //and back to the code that deals with the rendering
    c.fillStyle = 'blue'
    c.fill()
    c.stroke()
}
```

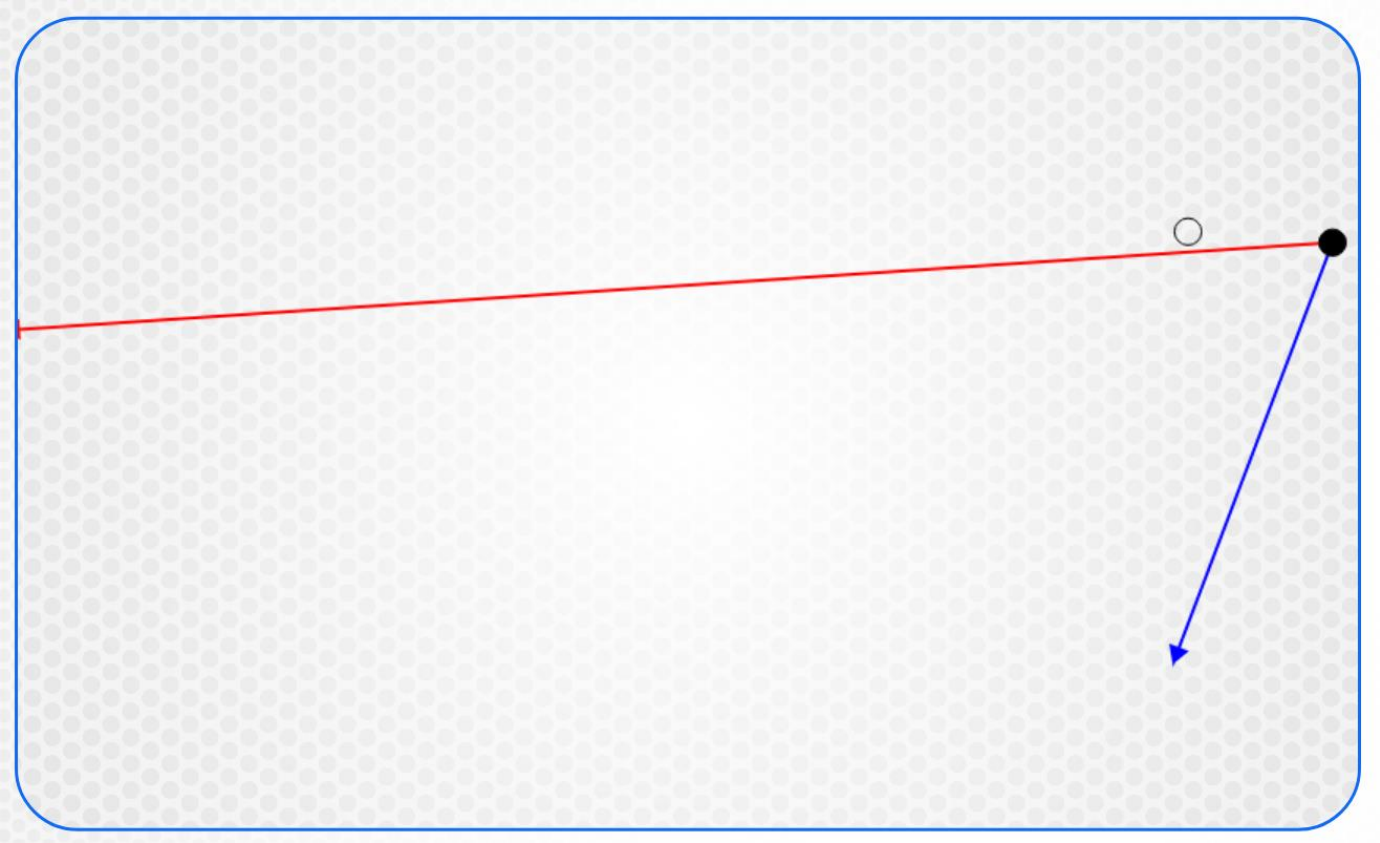
Acceleration Vectors

- It's clearer to see the relationship between a velocity vector and an object. An object will travel the length of a velocity vector in 1 second.
- The acceleration vector works similarly. In one second the velocity vector will change by the magnitude and direction of the acceleration vector. It's calculus!
- For this reason, the same code can be used for both, but with the vx & vy swapped out for ax & ay; the acceleration vector components.



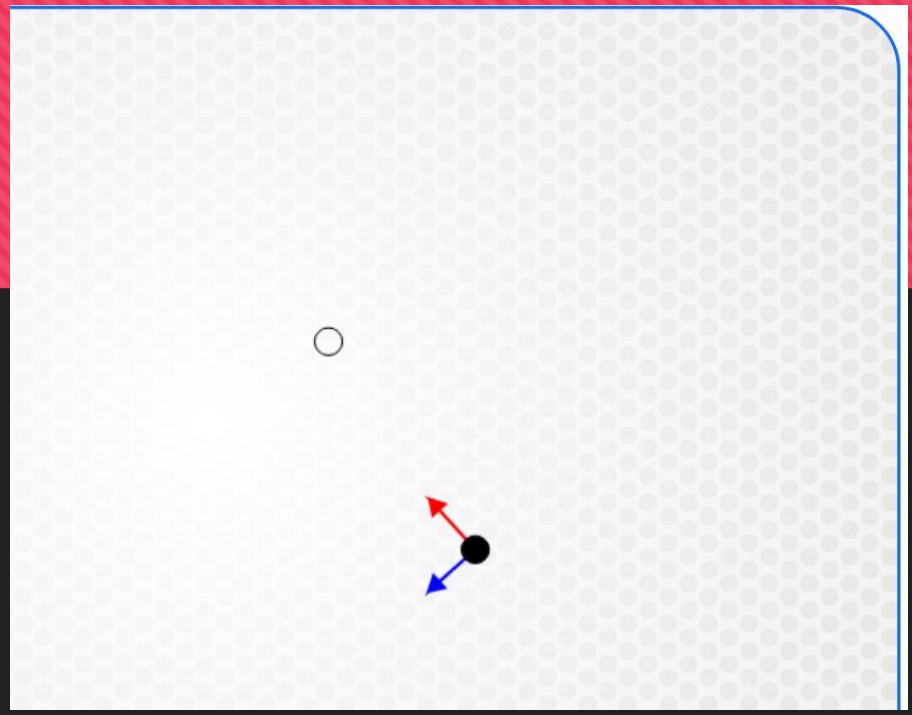
Bug: Vectors too big

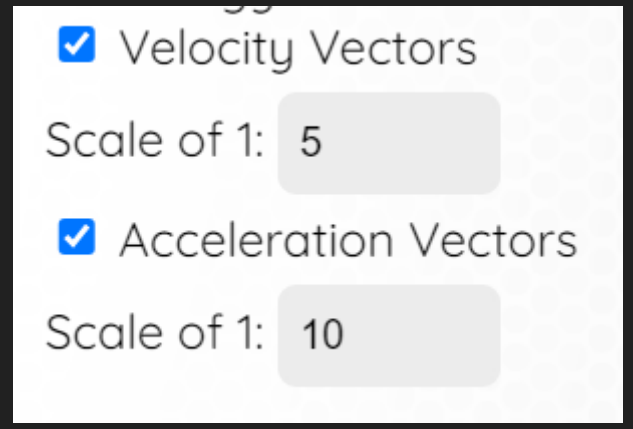
- A pretty big bug I noticed was that the vectors appeared far too large.
- I was somewhat stumped. This did not make sense.



Solution: Scale vectors

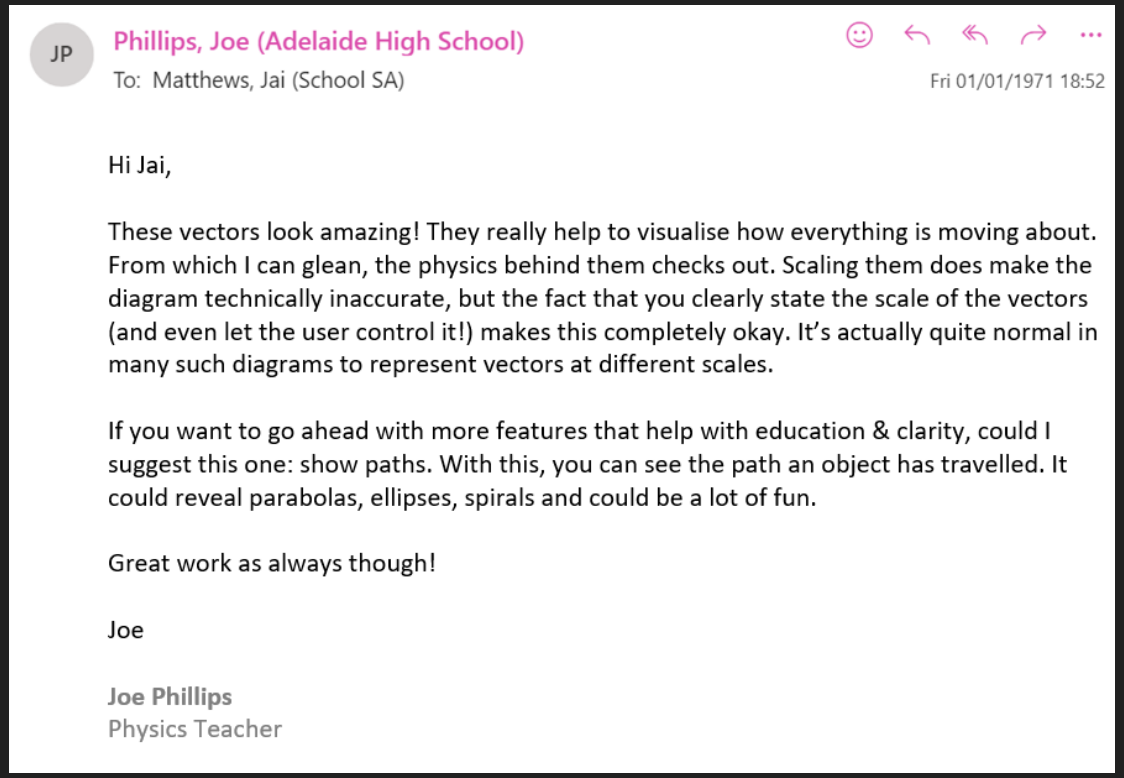
- But then, the more I thought about it, the more it actually did make sense.
- Acceleration due to gravity is 9.8m/s/s, and the screen is only 10m wide. So of course the vectors are as large as the screen.
- On one hand the current model was accurate, but on the other it looked bad and unhelpful.
- So I chose to have an option to scale the vectors down by an optional factor.
- This innovative feature allowed for greater user experience without compromising my goas.





Feedback

- The feedback for this iteration was largely about how it helped with visualising the way in which objects move.
- Joe said I could potentially take it even further if I made it possible to view the path an object had travelled.
- This was helpful feedback and I added it to the list of things I wanted to do.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

Hi Jai,

These vectors look amazing! They really help to visualise how everything is moving about. From which I can glean, the physics behind them checks out. Scaling them does make the diagram technically inaccurate, but the fact that you clearly state the scale of the vectors (and even let the user control it!) makes this completely okay. It's actually quite normal in many such diagrams to represent vectors at different scales.

If you want to go ahead with more features that help with education & clarity, could I suggest this one: show paths. With this, you can see the path an object has travelled. It could reveal parabolas, ellipses, spirals and could be a lot of fun.

Great work as always though!

Joe

Joe Phillips
Physics Teacher

Evaluation

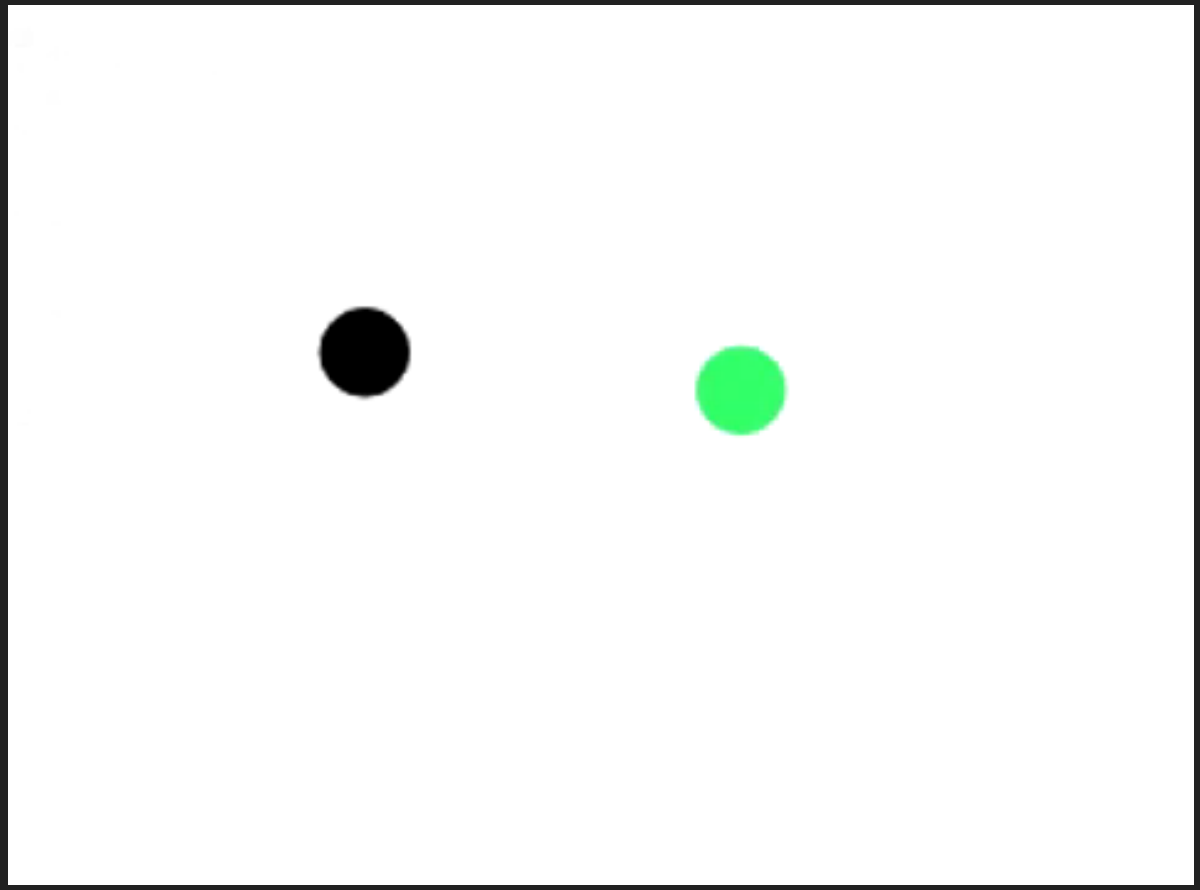
- I agreed with Joe that new features such as path visualisation could be great.
- But in terms of the features I did add, I was shocked by how effective they are at clearing up what was going on.
- I am extraordinarily pleased with this iteration.

Iteration 7

AT3 Major Project

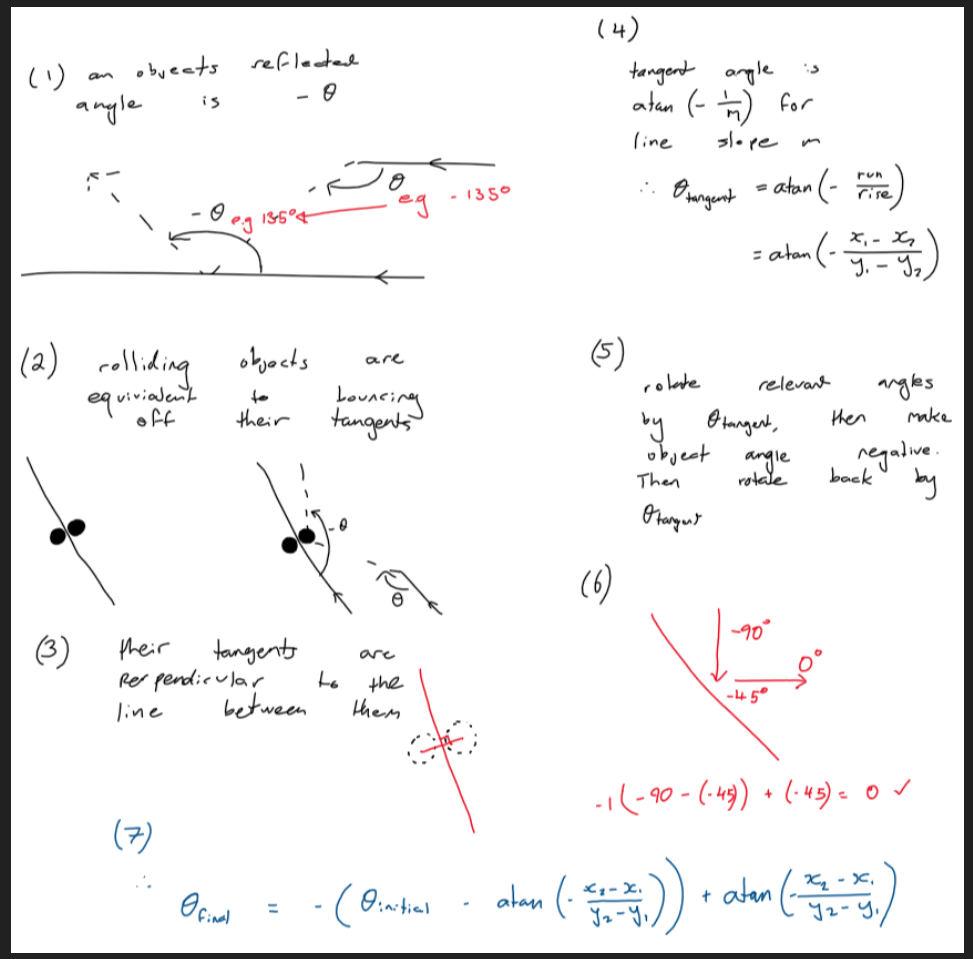
Overview

- In this iteration, I added object collisions.
- Now, when two objects collide, they bounce off one another.
- To properly illustrate this, I've manually made one ball green to see this effect.



Mathematical Derivation

- The first step in making these collisions happen is to figure out the math behind it.
- This used a lot of trigonometry & everything was done using angles rather than cartesian points like much of the other code.
- I began by considering the behaviour in a horizontal setting, then considered how to translate any given setting into a horizontal one.
- I made use of properties like line perpendicularity & tangents of circles.
- The working out and explanation is on the right.



Collision Code

- This code does exactly the math described in the previous slide.
- It detects a ball being nearby and does the correct trigonometry.
- It also has a feature at the bottom to prevent multiple firings during the one collision; it waits until post-collision to grant permission to run the code again.

```
//object collisions
if (collide) {
    //go through each object
    for (this.k = 0; this.k < objects.length; this.k++) {
        //check that the distance is less than the sum of the two objects' radii. In other words, they're touching
        //Also make sure that we're not checking an object is touching itself. After all, it always will be.
        if (pyth(this.x - objects[this.k].x, this.y - objects[this.k].y) <= this.radius + objects[this.k].radius && this != objects[this.k] && this.collide) {
            console.log('collision!')
            //and now math! calculate this particular objects' direction & speed
            this.speed = pyth(this.vx,this.vy)
            this.angle = angle(this.vx,this.vy)

            //find the angle of the line that is tangent to both balls at the point of contact
            this.lineAngle = Math.atan(-1 * ((this.x - objects[this.k].x) / (this.y - objects[this.k].y)))

            //and adjust the object's angle using this line angle
            this.angle = -1 * (this.angle + this.lineAngle) - this.lineAngle

            //convert the speed & angle back into vx & vy (cartesian velocity)
            this.vx = this.speed * Math.cos(this.angle)
            this.vy = this.speed * Math.sin(this.angle)

            //cancel this object's ability to collide (temporarily)
            //this will prevent it from running the code twice within the same collision
            this.collide = false
        }
    }

    this.touchCount = 0
    //another for loop to check if we're still touching eny objects
    for (this.k = 0; this.k < objects.length; this.k++) {
        //check for objects touching
        if (pyth(this.x - objects[this.k].x, this.y - objects[this.k].y) <= this.radius + objects[this.k].radius && this != objects[this.k]) {
            //if they're touching raise a number
            this.touchCount += 1
        }
    }
    //if the number is still 0, no objects are touching
    if (this.touchCount == 0) {
        //and thus collisions are allowed again.
        this.collide = true
    }
}
```

go through objects check for touching

convert cartesian to angles & back

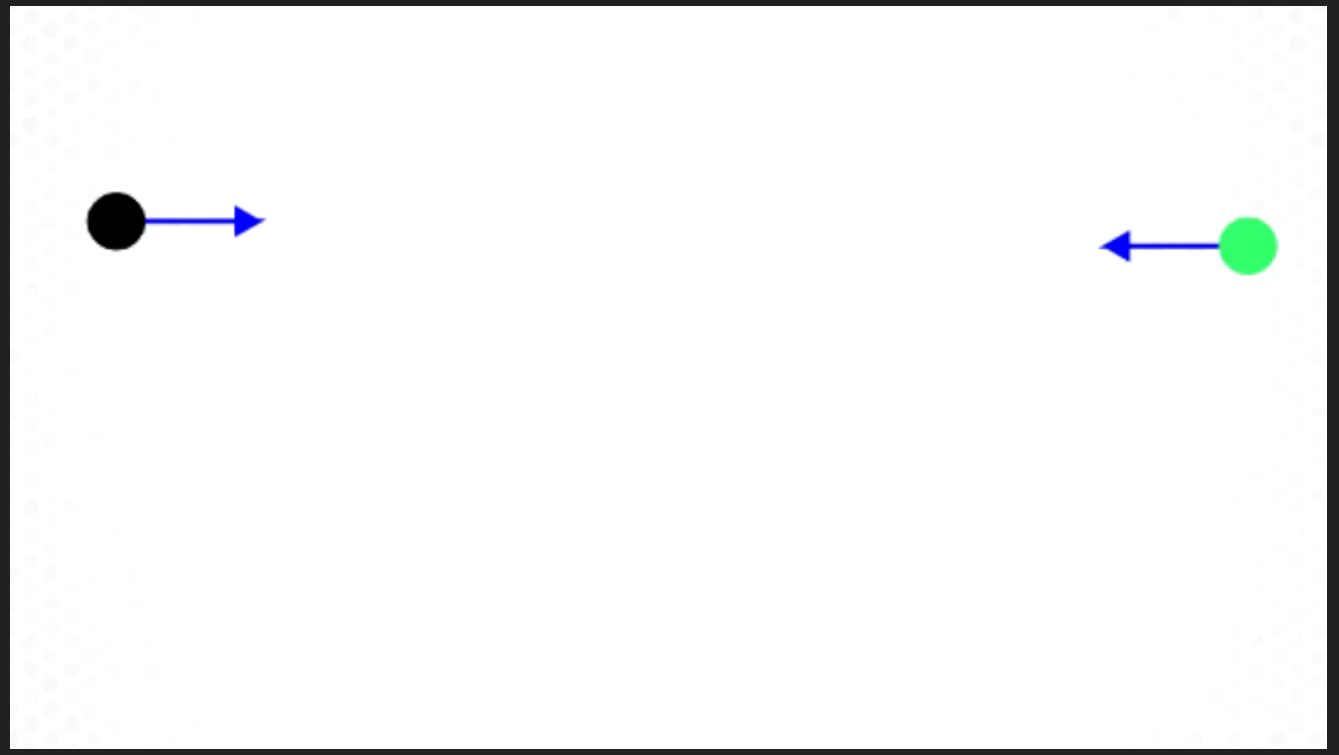
Math from other slide

prevent activating from again

check when allowed to use collision code again

Bug: Objects pass through one another

- A major bug I experienced was that objects at speed would often pass through one another.
- There is a lot of high speed movement in my sim, so this happens a lot.



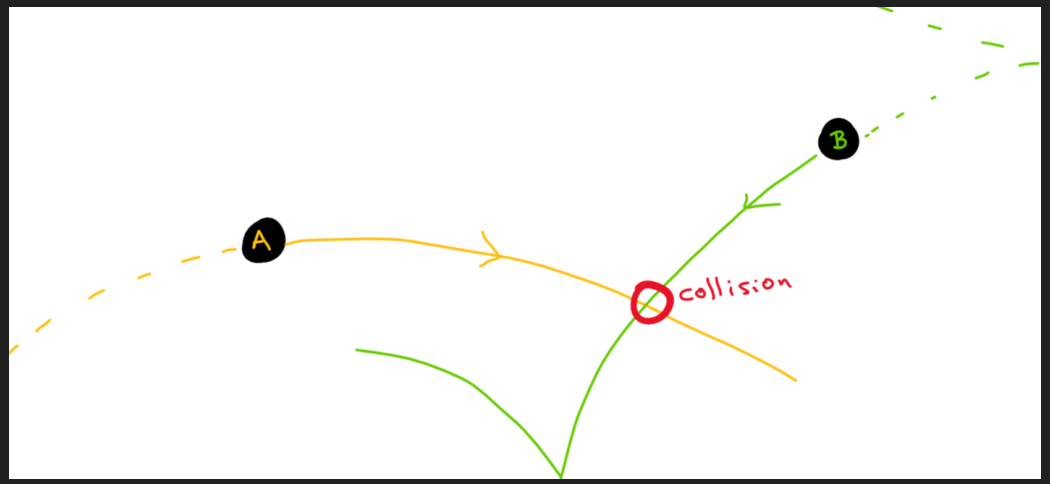
Investigating bug

- I then began thinking about what could have caused this.
- In the wall collision section, the issue was often that individual frames wouldn't perfectly time themselves to when there should be a collision.
- This is likely what's going on here too. Objects at speed move past each other such that there is no individual frame where they're close enough for the code to recognise a collision.

- I *could* increase the framerate so that it's more likely there will be such a frame, but this comes with several issues:
  - My laptop likely won't be able to handle particularly high framerates
  - JavaScript is pretty bad at accurately counting extremely low time intervals
  - Regardless, there'll still be a threshold speed where the issue remains.

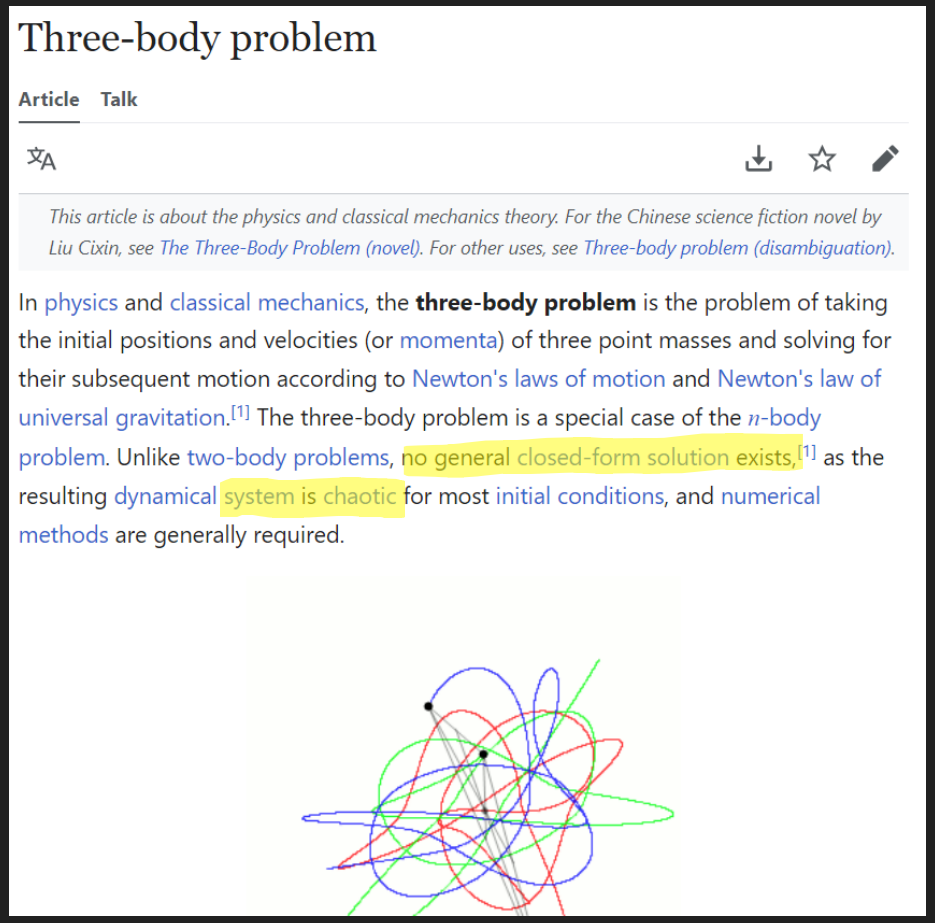
Theoretical Solution

- I began thinking of other, more creative solutions.
- The only thing I could think of was if I could somehow calculate the expected path of an object as a continuous function.
- Then I would do this for every object and look for intersections. This would require creation of graphs, knowledge of parametric equations, and some other complicated math.



Evaluating Solution

- But even this solution has major problems.
- The paths calculated could not be based on the current game state as that state would change in time as other influential objects moved. It would therefore be extraordinarily challenging to determine an accurate path.
- This links to things like the (currently unsolved) 3-Body problem which describes a struggle to simulate physics situations like that which I am experiencing.
- For this reason, I am deeming this debugging not necessarily impossible, but beyond the scope of this project.



Three-body problem

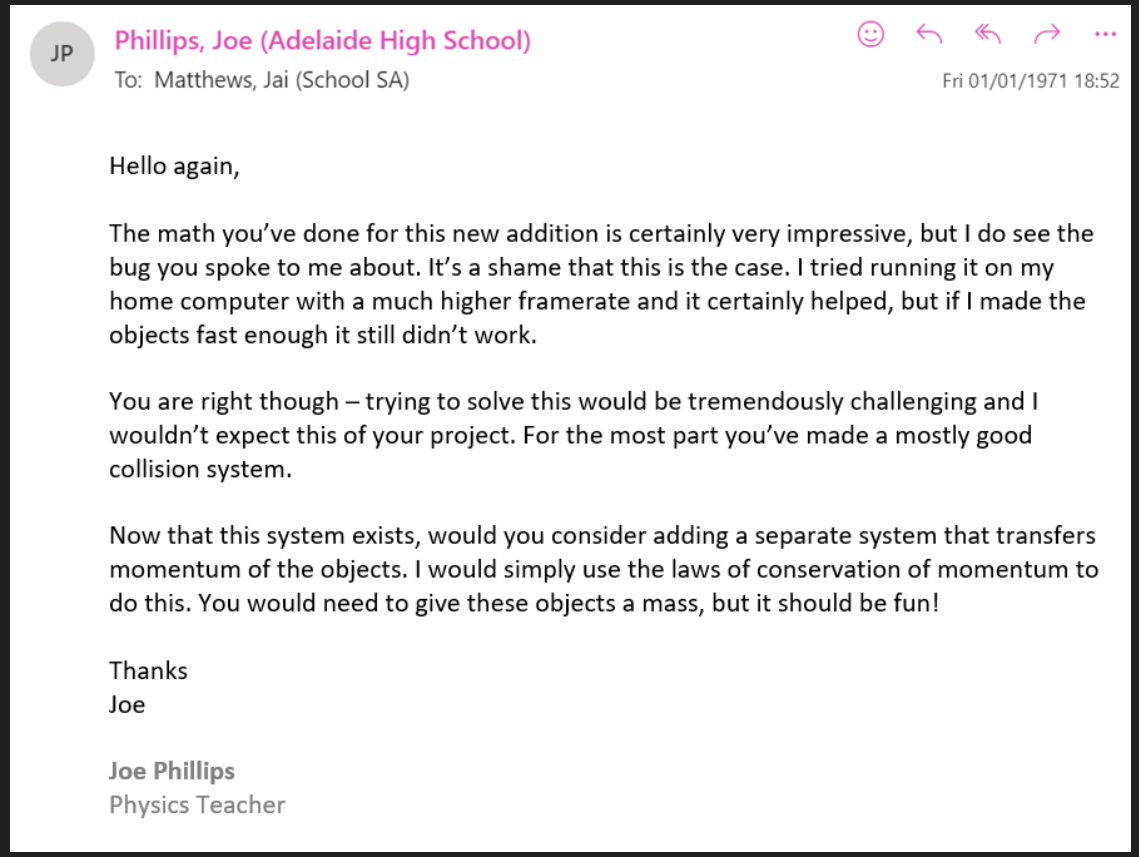
Article Talk

This article is about the physics and classical mechanics theory. For the Chinese science fiction novel by Liu Cixin, see The Three-Body Problem (novel). For other uses, see Three-body problem (disambiguation).

In physics and classical mechanics, the **three-body problem** is the problem of taking the initial positions and velocities (or momenta) of three point masses and solving for their subsequent motion according to Newton's laws of motion and Newton's law of universal gravitation.[1] The three-body problem is a special case of the n-body problem. Unlike two-body problems, no general closed-form solution exists,[1] as the resulting dynamical system is chaotic for most initial conditions, and numerical methods are generally required.

Feedback

- My stakeholder completely understood the flaws I discussed throughout this section and agreed that trying to fix it was not worth the time.
- Interestingly, he had the ability to run the code at a higher framerate and confirmed that it certainly mitigated the problem.
- Other than that, he really enjoyed the inclusion of a collision system.
- On a final note, he suggested adding momentum transfer.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

Hello again,

The math you've done for this new addition is certainly very impressive, but I do see the bug you spoke to me about. It's a shame that this is the case. I tried running it on my home computer with a much higher framerate and it certainly helped, but if I made the objects fast enough it still didn't work.

You are right though – trying to solve this would be tremendously challenging and I wouldn't expect this of your project. For the most part you've made a mostly good collision system.

Now that this system exists, would you consider adding a separate system that transfers momentum of the objects. I would simply use the laws of conservation of momentum to do this. You would need to give these objects a mass, but it should be fun!

Thanks
Joe

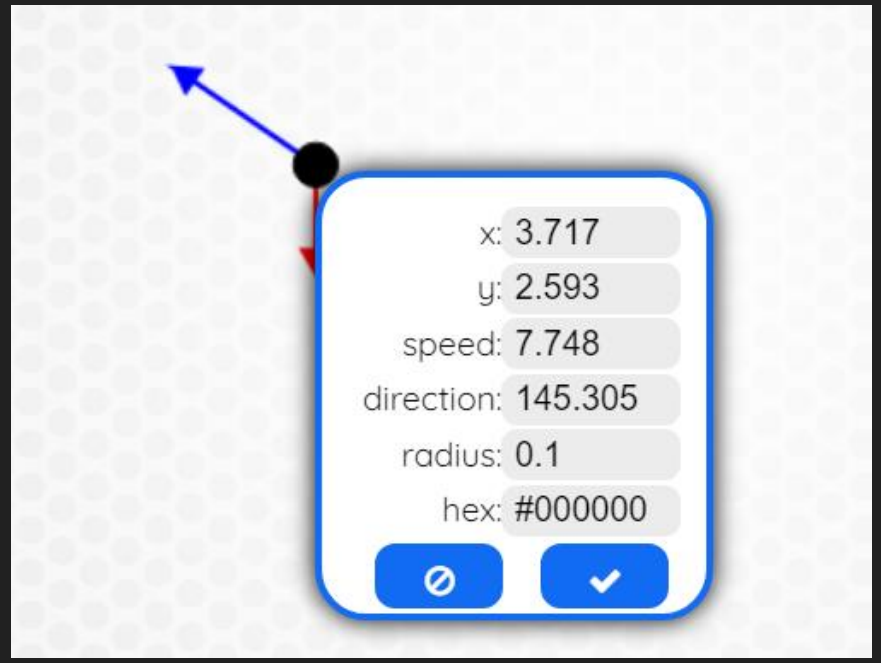
Joe Phillips
Physics Teacher

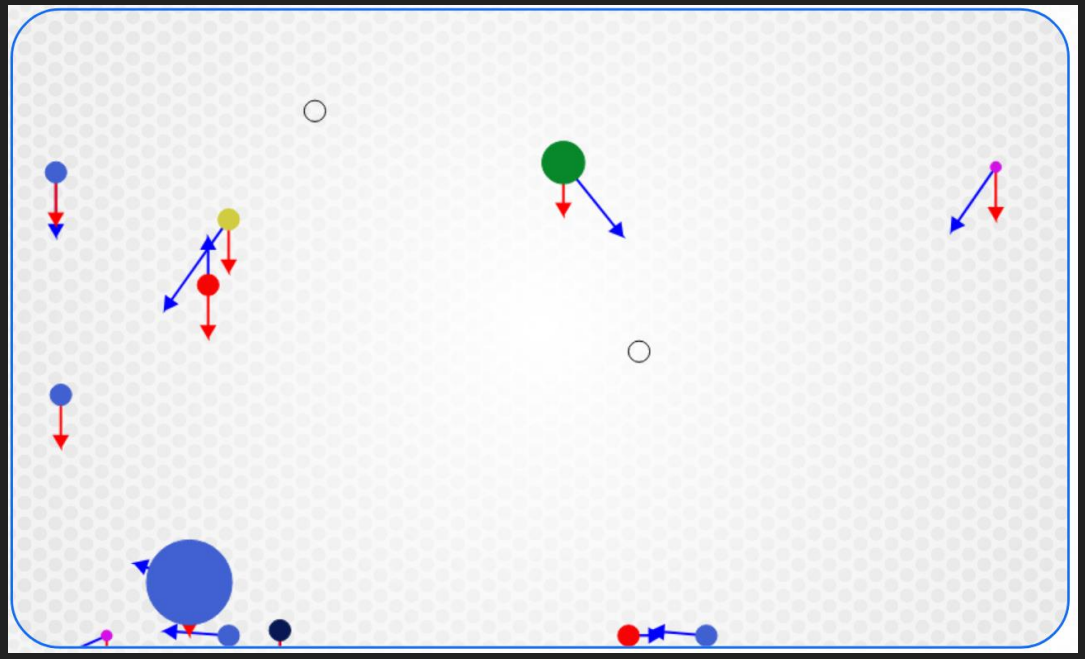
Iteration 8

AT3 Major Project

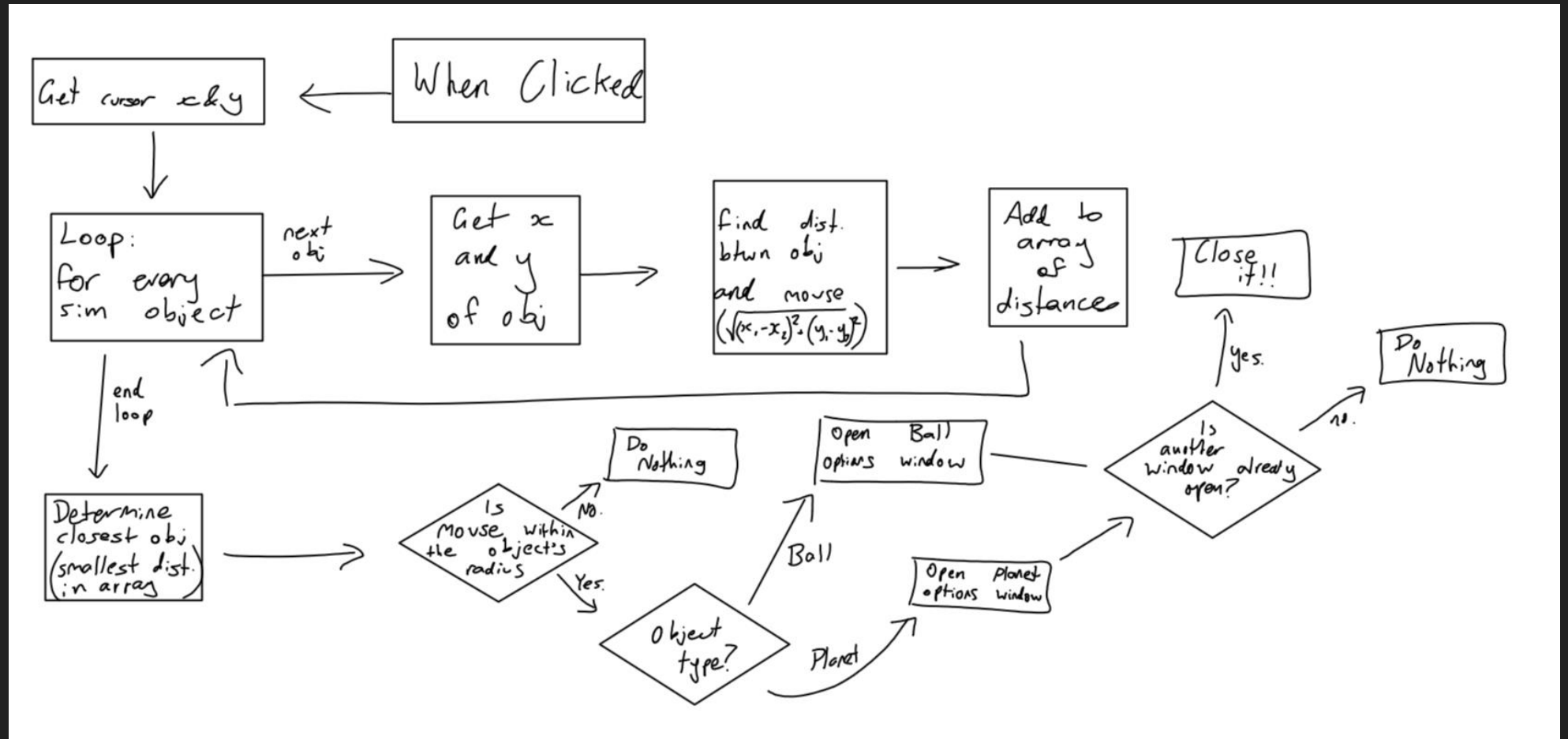
Overview

- In this iteration, I added the ability to select an object and change its properties with a little popup window.
- The rest of the simulation also had to be made to handle non-regular objects.





Select Flowchart



Select Code

When you use the select tool, it does the following as per the flowchart:

- List the distances between every object and the cursor.
- Finds the closest and check the cursor is over the object
- Show the popup next to the object
- Put the object properties into the popup, making sure to convert vx & vy into speed and direction as it's more user-friendly & easy to understand.

```
case 'select':
    //get canvas info
    rect = canvas.getBoundingClientRect()

    //start recording distances between cursor and objects
    distances = []

    //go through each object and determine the distance between it and the cursor
    for (i = 0; i < objects.length; i++) {
        if (objects[i].type == 'Ball') {
            //object pos is stored in m. we need to convert this.
            xdistance = (mTOpx(objects[i].x) + rect.left) - event.clientX
            ydistance = (mTOpx(objects[i].y) + rect.top) - event.clientY
            //use pythagorean theorem to determine distances
            d = pyth(xdistance, ydistance)
            //add this to an array
            distances.push(d)
        } else {
            distances.push(10000000000)
        }
    }

    //find the shortest distance and record this object
    selectedObjectID = distances.indexOf(Math.min(...distances))

    //if the shortest distance is less than the radius, the cursor is over it!
    //I add 10 just to give the user some slight leeway
    if (mTOpx(objects[selectedObjectID].radius) + 10 < distances[selectedObjectID]) {
        break
    }

    //show the popup with all the object parameters
    popupWindow = document.getElementById('selectPopup')
    popupWindow.classList.remove('visible')

    //place this popup in the correct spot
    //add the position of the object in the canvas to the position of the canvas to get the position on the page
    popupWindow.style.top = mTOpx(objects[selectedObjectID].y) + rect.top + 'px'
    popupWindow.style.left = mTOpx(objects[selectedObjectID].x) + rect.left + 'px'

    //and now make the values of all the inputs correct
    //using just some trig i've converted the vx & vy into a direction & speed which is more user friendly
    popupWindow.querySelector('#x').value = roundTo(objects[selectedObjectID].x, 3)
    popupWindow.querySelector('#y').value = roundTo(objects[selectedObjectID].y, 3)
    popupWindow.querySelector('#v').value = roundTo(pyth(objects[selectedObjectID].vx, objects[selectedObjectID].vy), 3)
    //converting from components to degrees isn't straightforward. bear with me.
    speedAngle = toDegrees(Math.atan(objects[selectedObjectID].vy/objects[selectedObjectID].vx))
    //this is just because the domain of arctan is a little dodgy. we need to accomodate in quadrants 2 & 3
    if (objects[selectedObjectID].vx < 0) {
        speedAngle += 180
    }
    //this is only to make sure that the range of angles is -180 to 180 rather than 0 to 360
    //really it's just a personal choice. I think it's more user friendly
    if (speedAngle > 180) {
        speedAngle -= 360
    }
    popupWindow.querySelector('#d').value = roundTo(speedAngle,3)
    popupWindow.querySelector('#r').value = roundTo(objects[selectedObjectID].radius, 3)

    break
```

list all distances

find closest

check cursor over

show popup

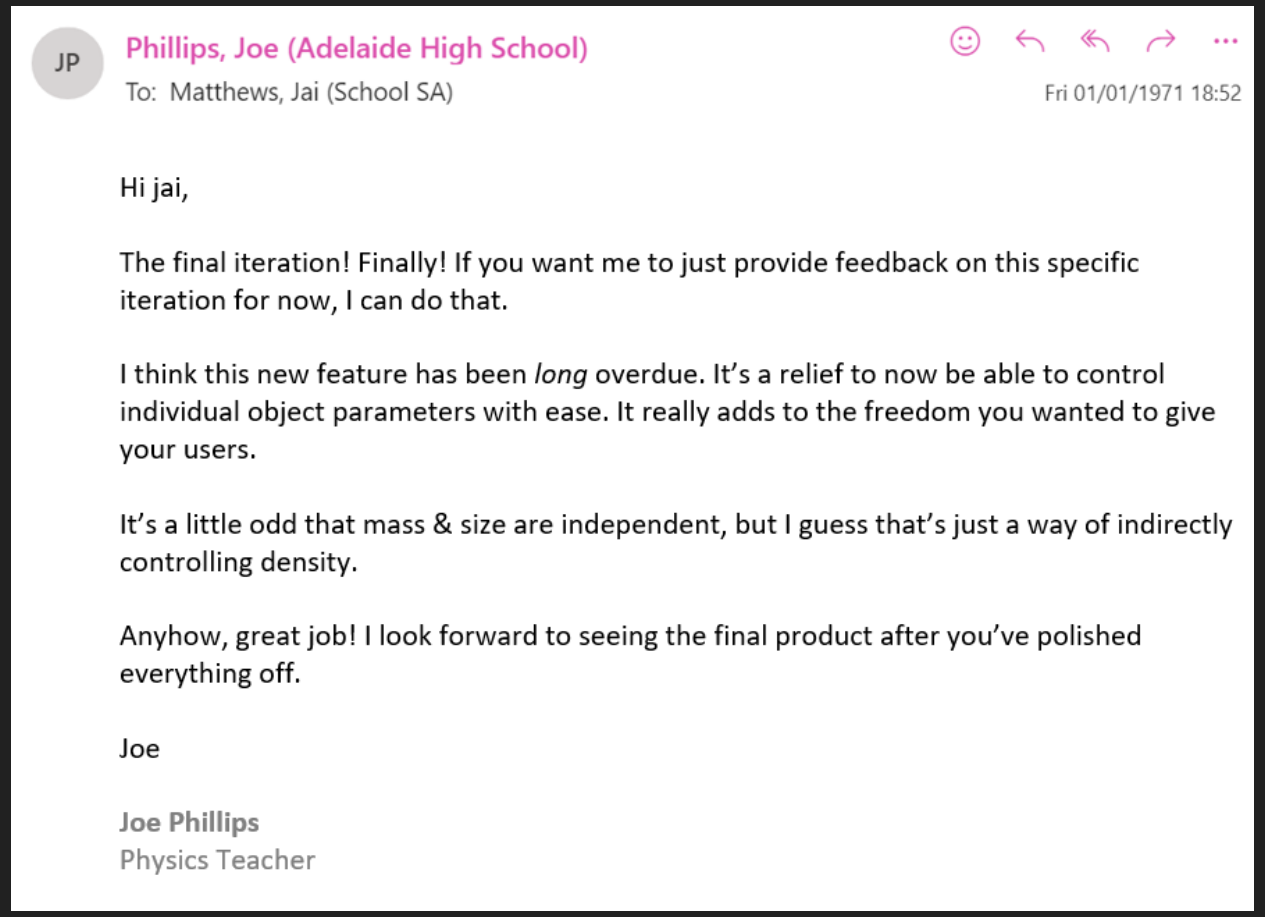
show params

```
//I also quickly implemented these so that I could present degrees to the user
//JS does trig in radians
function toDegrees(angle) {
    return angle * (180 / Math.PI);
}

function toRadians(angle) {
    return angle * (Math.PI / 180);
}
```

Vector trig :(

Feedback

- My stakeholder was very complemental of the additions in this iteration.
- He pointed out that this is extremely helpful in making the simulation general-purpose as per the original goals.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

Hi jai,

The final iteration! Finally! If you want me to just provide feedback on this specific iteration for now, I can do that.

I think this new feature has been *long* overdue. It's a relief to now be able to control individual object parameters with ease. It really adds to the freedom you wanted to give your users.

It's a little odd that mass & size are independent, but I guess that's just a way of indirectly controlling density.

Anyhow, great job! I look forward to seeing the final product after you've polished everything off.

Joe

Joe Phillips
Physics Teacher

Evaluation

- I believe this iteration was incredibly good in terms of realising my mission statement. It added a wealth of options for the user and meant that there was a lot of freedom in what the user could do.
- I am tremendously glad to have aided in my initial innovate goal for this project.

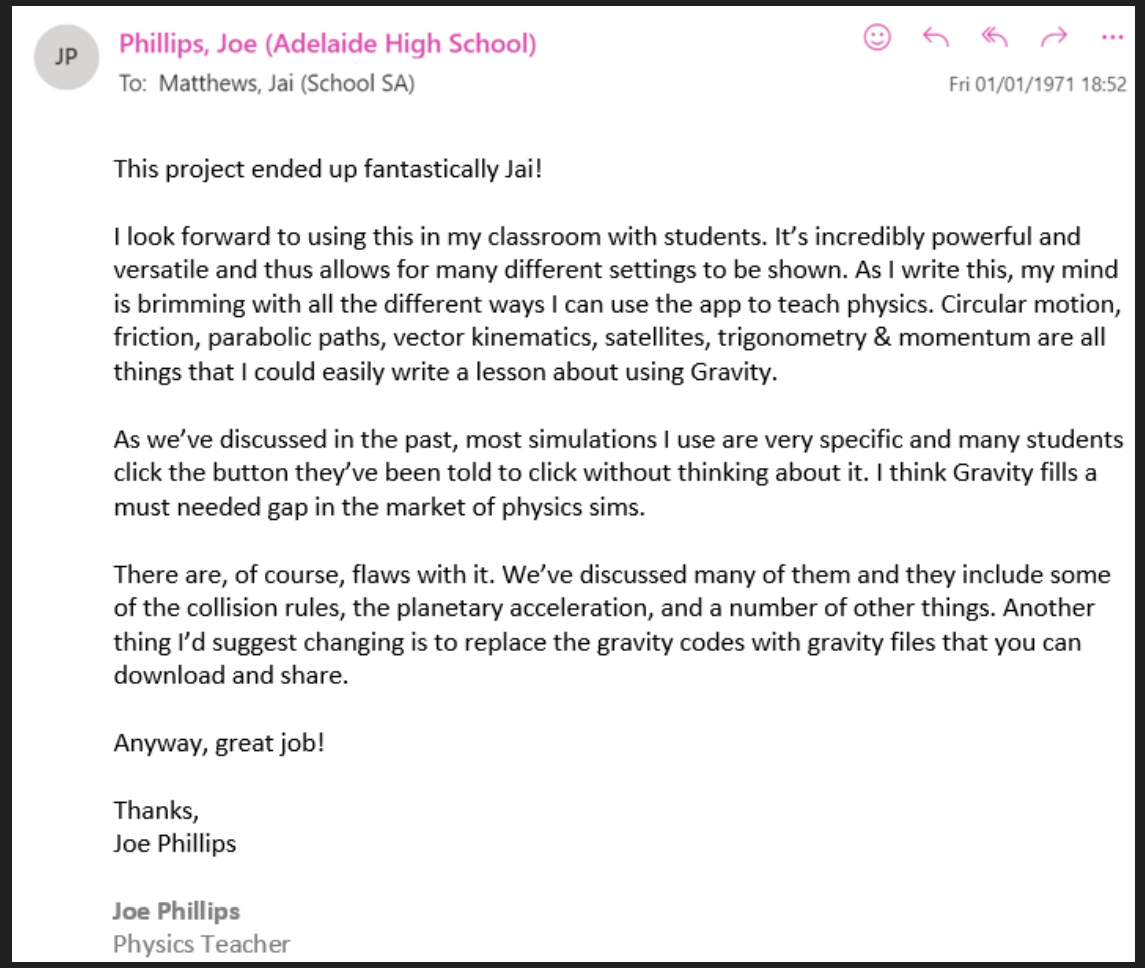
Evaluation

AT3 Major Project

Project Demonstration

Stakeholder Feedback

- My stakeholder really enjoyed the project and thought it to be of a very high quality.
- He particularly enjoyed the versatility that I programmed into it.
- Things he suggested changing included the collision errors discussed and the acceleration due to gravity issues from earlier.
- Also, he suggested create files rather than codes for sharing set ups.



JP

Phillips, Joe (Adelaide High School)

To: Matthews, Jai (School SA)

Fri 01/01/1971 18:52

This project ended up fantastically Jai!

I look forward to using this in my classroom with students. It's incredibly powerful and versatile and thus allows for many different settings to be shown. As I write this, my mind is brimming with all the different ways I can use the app to teach physics. Circular motion, friction, parabolic paths, vector kinematics, satellites, trigonometry & momentum are all things that I could easily write a lesson about using Gravity.

As we've discussed in the past, most simulations I use are very specific and many students click the button they've been told to click without thinking about it. I think Gravity fills a must needed gap in the market of physics sims.

There are, of course, flaws with it. We've discussed many of them and they include some of the collision rules, the planetary acceleration, and a number of other things. Another thing I'd suggest changing is to replace the gravity codes with gravity files that you can download and share.

Anyway, great job!

Thanks,
Joe Phillips

Joe Phillips
Physics Teacher

Innovative Features

- The primary innovative feature is the fact that it is a general-purpose simulation for many different settings.
- It's also innovative in that teachers are empowered to create their own physics settings that they can share with students using a Gravity Code.
- A biproduct of the versatility of this project is that you can simulate multiple things simultaneously that other products on the market are incapable of doing. For instance, planetary gravity & object collision. Or parabolic motion & changing sources of gravity.

Good things

Things I am particularly proud of in this project include the following:

- All the innovative features discussed on the previous slide.
- The ability to display & scale velocity & acceleration vectors.
- The ability to share lessons with students.
- The seamless parabolic motion that arose from simple kinematic code.
- The fact that this all runs cleanly in pure JS

Bad things

As discussed throughout this presentation, there were a number of issues. They are summarised here:

- Energy Transfer wasn't coded in the final product
- Acceleration due to gravity did not increase with proximity
- The drag equation wasn't entirely accurate to real world situations.
- The collisions were glitchy and there wasn't a clear solution.
- The save codes were too big.
- The screen size can not be scaled whilst using the editor without breaking it.

Energy transfer & showing object paths were two ideas that my stakeholder gave me that, unfortunately, I did not have time to implement.

Relation to Original Problem

- The original project's goal was primarily to create a simulation that empowered teachers & students with the versatility to create whatever setting was appropriate to them.
- I strongly feel that this goal was satisfied. Almost everything in the simulation is controllable and there are ample settings & options for the user.

Purpose of Project

- The purpose is to provide teachers with a physics tool to communicate motion topics to students.
- Many services available are limited to only performing specific tasks. Mine would be general-purpose and can be used for a variety of different reasons.
- This may mean, for example, that students will need to know how to set up a digital experiment and thus show an extra level of understanding.

An example of an existing physics sim (that I myself have been subjected to) that can *only* handle projectile motion and nothing else.

Relation to Original Goals

Goal	Did I meet it?
Create a general-purpose physics simulation.	Yes. As discussed on previous slide it was very effective in doing this.
Allow teachers to send setups to students that they can watch.	Yes once again. There is a save/load feature teachers can use.
Make it easy to use so students can set up their own settings.	Absolutely! With only 3 tools to learn and some very clear settings, students will have no problem.
Include elements that allow for a greater understanding of the underlying physics.	I included the ability to show vectors which was great, but perhaps in future versions I could also show an object's path?
Make objects behave in it as they would in the real world.	Yes. I strictly sticked to genuine physics formulas whenever I could. In the cases where I couldn't, reasons were given.
Include options for how the world works (e.g., controls for acceleration due to gravity)	Yep. You can control gravity, drag, vector scales, and every property of every object.
Make it enjoyable to use to spark students' interest.	I myself find that these simulations can be quite mesmerising. For this reason and others, students should enjoy the app.

The End!

AT3 Major Project