# Testing Report

## Module Testing

Document input, expected output, and actual output for ten modules  
■ These modules must be essential to the functionality of the program  
■ You will need multiple test cases to complete thorough testing  
■ If applicable, identify the cause of and solution to errors

## Program Testing

Test each feature of the program  
■ Document the functionality, usability and performance of each  
■ If applicable, identify the cause of and solution to errors

## System Testing

Test your software in as many different environments as possible.  
This could include different operating systems, different browsers,  
different hardware, etc.  
■ Document the functionality and performance of the software. This  
may include the use of screenshots, traces, or response times.

# Evaluation

○ A detailed description of the finished program, identifying what features were  
implemented and how well they work  
○ Identify any features that were not implemented and explain why they were  
not included  
○ Discuss any major issues that were encountered during development  
○ Describe how the project challenged and/or improved your programming  
ability throughout its development

# Logbook & Bibliography

## Log 1: Basics

### Ctrl + C; Ctrl + V

#### Context

After not doing anything for the majority of the holidays, I have finally forced myself to begin.

The easiest and arguably most beneficial thing I could do at the moment would be familiarising myself with [matter.js](https://brm.io/matter-js/), the physics engine to be used throughout the project.

#### Implementation

I had nothing better to do so I copied the example code off their ‘Getting Started’ page. The code created two rectangles falling down onto solid ground with gravity and collisions handled by the engine.

A screenshot of a video game

Description automatically generated

I have no idea why the viewport is tiny

The code started by assigning aliases to modules, which I’d assume makes them important.

A screen shot of a computer

Description automatically generated

From what I can work out:

Engine includes the world in which the physics of bodies are calculated in, and some of the fundamental physics attributes (e.g. gravity)

Render is the canvas of matter.js, including methods and properties relating to rendering the objects in the world (e.g. how often objects are redrawn, displaying performance information)

Runner is a substitute for the normal main loop of a program, recalculating the physics of all objects in the world at set intervals. Alternatively, Engine.update calculates the physics of all objects a single time, allowing for more precise control over updating the physics simulation.

Bodies contains methods of creating common bodies to be simulated (e.g. rectangles, circles). Body, on the other hand, not only is able to create more complex bodies by inputting vertices, but also allows for manipulation of physical properties (e.g. angular velocity, friction, time scaling).

Composite is used for creating containers to sort bodies, somewhat like the usage of layers in digital art. The root world in which bodies are placed, engine.world is also a Composite.

#### Sources

##### [Matter.js source code](https://github.com/liabru/matter-js/blob/master/build/matter.js)

Downloaded the file to use the physics engine.

* By [liabru](https://github.com/liabru)
* Under MIT licence, which allows modification, distribution and private use so long as the MIT licence is still included in the software. Licence conditions are met as the MIT licence takes up the first 27 lines of the matter.js file.
* Accessed 21/01/2024

##### [Matter.js getting started page](https://github.com/liabru/matter-js/wiki/Getting-started)

Followed instructions to install matter.js and copied the code for learning purposes.

* By [liabru](https://github.com/liabru)
* Accessed 21/01/2024

### Visuals

#### Context

After having a basic scene, I wanted to see the capabilities of the renderer and how I could include these visuals in a game.

#### Implementation

I started by changing the size of the canvas, since a platformer would probably need a full screen display.

Initially I tried putting width and height properties alongside the others that already exist in the given code for Render.create. That didn’t work since width and height properties are for some reason nested within another options record, so it ended up looking like this in code:

A screen shot of a computer code

Description automatically generated

I considered dynamically resizing the canvas to fit the size of the viewport as I did with the orbital simulation. It didn’t seem like a great idea however since pixel precision would be required in a platformer, unlike an orbital simulation where most of the canvas is just empty space. These dimensions looked alright on the screen, so I kept it as so for the time being.

A screenshot of a video game

Description automatically generated

There’s meant to be a large white bar at the bottom of the screen though I don’t think it’s visible in the document because of the white background

Reading the documentation, it turns out that matter.js also has a lot of debug options in its renderer, so I decided to turn all of them on while adding a few more bodies:

A screenshot of a computer game

Description automatically generated

It’s also possible to give fill colour to bodies once wireframe is disabled. By default, matter.js assigns each body a random colour chosen from a pre-written set. I also changed the background colour along the way.

A screenshot of a computer

Description automatically generated

There’s also a method called setPixelRatio for Render. The idea is that pixels specified in CSS can be scaled on the user’s screen. Normally, zooming in on elements would result in blurriness due to a single CSS pixel taking up multiple pixels on the user’s screen. However, if the pixel ratio is adjusted accordingly, the element will maintain its resolution despite zooming in or out.

That’s under ideal conditions anyhow. As of now I’m not sure how the pixel ratio will handle fractional pixels and whether or not it will affect the positioning of elements on screen. Quite important for the placement of platforms and items.

#### Sources

##### [Matter.js documentation – Matter.Render](https://brm.io/matter-js/docs/classes/Render.html)

Used for understanding render options.

* By [liabru](https://github.com/liabru)
* Accessed 23/01/2024

##### [MDN web docs - Window: devicePixelRatio property](https://developer.mozilla.org/en-US/docs/Web/API/Window/devicePixelRatio)

A page linked to by the matter.js documentation to explain the pixel ratio.

* By Mozilla
* Accessed 23/01/2024

### Keyboard Controls and Moving

#### Context

As per the platformer genre, the player should be able to move an in-game character through some sort of keyboard interaction – in this case the arrow keys. This will hopefully be a rough sketch of how basic keyboard controls should work.

#### Implementation

Writing it in code should be simple enough, just add event listeners for each key and use the applyForce method to move the object.

That was not the case. Turns out, JavaScript event listeners for the keydown event don’t detect the key being held down properly. When a key is held down, instead of the event being constantly fired, it fires once, then pauses for a while, before it finally starts firing constantly.

I’ve seen the exact same problem in graphics.py, and I assume it occurs due to the code not actually reading keystrokes but the text that the keys type out. Due to how typing in a text document works, holding down a key types a single letter first, pauses, and then spams the letter out (try typing ‘aaaaaaa’ by holding down ‘a’ and you’ll see what I mean). While this makes sense in typing, it does not make sense in a game.

This also results in some weird behaviour when multiple keys are pressed. Only if keys are pressed simultaneously (on the same frame I’d assume) do they both register as inputs, and the pausing midway problem still persists. On the other hand, if one key is pressed later than another, the later key will override the previous key. You really don’t think much into how differently the keyboard is used in gaming and office environments until this happens.

I considered using both keydown and keyup events to precisely track when a key is pressed and when it is released with flags, but felt it may be too complex for a simple problem, and decided to have a look at what the internet says.

As it turns out, this problem is stupidly common to the point that people have made entire libraries dedicated to fixing how JavaScript responds to holding down keys. While I could import perhaps [p5](https://p5js.org/reference/#/p5/keyIsDown) or [Keydrown](https://jeremyckahn.github.io/keydrown/) to fix this problem, it just doesn’t feel quite as fun as making it myself, and I have no idea how the 85% code originality works so I’ll just play it safe. Anyhow, the Stack Overflow responses gave similar approach as my initial solution so it should be fine.

And it was, surprisingly enough, fine.

#### Sources

##### [KeyDrown](https://jeremyckahn.github.io/keydrown/)

A library that just fixes the key hold problem. Had a look at it. Didn’t import it in the end.

* By [jeremyckahn](https://github.com/jeremyckahn)
* Accessed 24/01/2024

##### [p5 - keyIsDown](https://p5js.org/reference/#/p5/keyIsDown)

A rendering library that also fixes the key hold problem. Looked at it. Didn’t import.

* By [Processing Foundation](https://processingfoundation.org/)
* Accessed 24/01/2024

##### [A Stack Overflow question about the key hold problem](https://stackoverflow.com/questions/29279805/keydown-doesnt-continuously-fire-when-pressed-and-hold)

Used to check if my logic was correct on solving the key hold problem

* Response from [Ghis](https://stackoverflow.com/users/1311952/ghis)
* Accessed 24/01/2024

## Log 2: Movement Physics

### Acceleration and Deceleration

#### Context

#### Implementation

May need to mess with friction to get acceleration correct and cap top speed, would it be easier to just mess with velocity and acceleration

Matter.js is too accurate!!

Box would not move

Stupid matter makes the function namespaced under Body rather than just being a method of boxA

Also no way to just change the velocity or increment have to set the velocity of the object everytime

Wdum acceleration isn’t a thing in matter.js

Add velocity cap; subject to change

Swap out friction for more predictable acceleration

Added an else statement to decelerate when key not pressed; deceleration in two directions cancel out

Very goofy but I guess I can finally move it and play flappy bird now

Stupid matter.js only allows to use methods to change x y components all properties are read only

### Collision

#### Context

#### Implementation

Collsion.collide stupid, can only check one pair and returns some illegible collision object or smth

Fogor how to do for loops in js, for of instead for in

Add an extra flag for jumping

### Wall Jump

#### Context

#### Implementation

How do you prevent the player from just scaling the wall

Cooldown timer?

Just make the player bounce off the wall

How do you make sure it bounces off in the right direction instead of going into the wall again allowing for another wall jump and then just scaling the wall again

Make grounds grounds make walls walls

Stupid thing started walljumping left on a wall to the left on the player turning it into scaling the wall again

Put only = forgor == why does js not give an error for that

## Log 3: Gravity💀

### Just changing gravity

#### Implementation

Matter.js has this thing called engine.gravity that changes the x-y vector of gravity.

Add key detection to that and we’re done – simple enough

#### Sources

### Changing keystrokes and directions with gravity

#### Context

Suppose gravity pulls you onto the right wall, and from your perspective the right wall is the ground.

If we assume that gravity always pulls down, we can say that pressing the right arrow (moving right on screen) will make you fall. Similarly, pressing the left arrow should make you jump.

Pressing the up arrow would result in you moving up on screen, but right from your point of view. Pressing the left arrow would result in moving down on screen, but left from your point of view.

Since the functions jump, fastFall and move are called depending on keystrokes, the program needs to translate the keys pressed to the actual directions in your perspective for the functions to be called correctly.

Furthermore, when “right” in your perspective is passed into move in this case (pulled onto the right wall), the program needs to know that “right” in this case is not increasing the x value, but actually decreasing the y value. So the values that are assigned to moving in each direction respectively will also have to be changed.

#### Implementation

Use a bunch of records to convert this stuff

Took a while to figure out the best way to structure records

A computer screen shot of a code

Description automatically generated

And then every single movement function had to be changed so that movements could be carried out on any axis. This was made a bit more annoying as matter.js has read only properties, so I would have to assign values to a record every time to move.

Initially I resorted to a lot of if statements to check which axis the movement was meant to occur in, but I realised later that thankfully, records don’t care about the order of items, so I could do something like this: (also apparently adding square brackets just turns a string into an expression?)

A screen shot of a computer

Description automatically generated

While it is nearly impossible to read, I can assure that it does in fact work. To be fair, the if statement approach worked too, just that it took up twice the amount of lines.

#### Sources

### Changing detection with gravity

#### Context

Initially I wanted to just start on the next heading down there, but I realised that no matter how hard I tried, I could not get the block to jump when gravity was flipped in any direction other than downwards.

After a lot of console logging, the error was that the jump function was called at all.

After a bit more console logging, and realising that one of my flags couldn’t be accessed directly from terminal (why?), it was because the block didn’t actually think it was on the ground when gravity was reversed, and refuses to jump in mid-air.

It also resulted in weird stuff like it trying to fall even though it was already on the ground.

So, the thing I had to do first was make it realise it was colliding with the ground

#### Implementation

Obstacles of each direction are stored in arrays. The arrays are then put into a record, with the key of the record being the four directions.

Initially, I just copied the directions of the KEYSTROKE\_TO\_DIRECTION record which changed depending on gravity. While it was perfectly functional for upwards and downwards gravity, it was different for left and rightwards gravity.

e.g. when gravity to the left, pressing up would result in moving left from block’s perspective. However, obstacles to the right of the block from its perspective, would actually be rightwards on screen (I had to read this 4 times to make sure it made sense, not sure if it does now)

A screen shot of a computer program

Description automatically generated

### Changing physics with gravity

#### Context

Now we can finally do this.

#### Implementation

Since DIRECTION\_TO\_VALUE had a sign component already, all that was left was the multiply it to every movement.

And now that the block actually thinks it’s on the ground, ground friction and jumping finally work now.

Not much to say, just a lot of replacing stuff.

[insert screenshot later]

#### Sources

○ Regular, detailed and reflective entries. Each entry may include:  
■ Description of what was attempted  
■ Discussion of problems and possible solutions  
■ Evaluation of progress and/or feature/program  
■ Ideas and considerations  
○ Screenshots must be taken of the program at regular intervals to show the  
development of the software

○ For all external resources used (including images, code snippets, etc),  
identify:  
■ The original author (may be a username)  
■ Where the resource was accessed (provide a link if applicable)  
■ When the resource was accessed  
■ Any licence requirements of the resource  
■ How the resource was used  
○ Note that links must be specific, e.g., to a discussion page on Stack  
Overflow, not to the Stack Overflow home page