

Rose Plumley Assignment 2

by Rose Plumley

Submission date: 04-Jan-2025 10:21AM (UTC+0000)

Submission ID: 247112493

File name: 40436_Rose_Plumley_Rose_Plumley_Assignment_2_628793_395683598.docx (495.85K)

Word count: 1503

Character count: 8121

Assignment 2

Rose Plumley

2218443

Games Architecture and Mechanics

ACCB5017

Nathan Morgan

Table of Contents

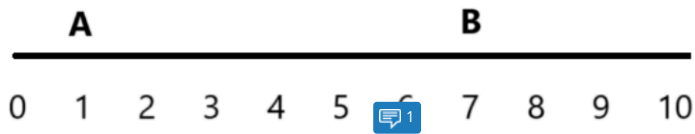
1. Introduction.....	3
2. Distance Between Two Points.....	3
3. Plane Class.....	12
4. Sphere Class with Collision Detection.....	14
5. Point to Plane and Sphere to Plane Collision Detection.....	14
6. Conclusion.....	14
7. References.....	15

1. Introduction

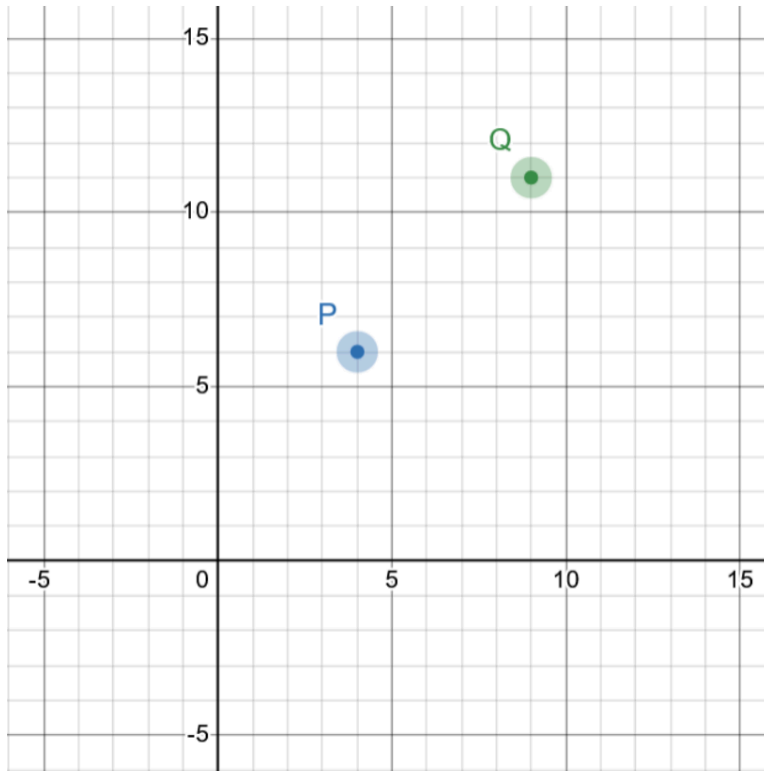
In the context of computer graphics, and by extension games, objects are represented in image form by providing software with geometric descriptions of the object to be displayed, which determine variables such as size, shape, and location in space (3-dimensional or otherwise) [1]. These descriptions can be stored in code as individual classes, such as a vector class. This report aims to cover work on a vector class for a game engine in the C++ programming language, investigating involved mathematical concepts, exploring the underlining theoretical concepts therein, and implementing these into code.

2. Distance Between Two Points

First, consider the distance between two points on a line, A and B:



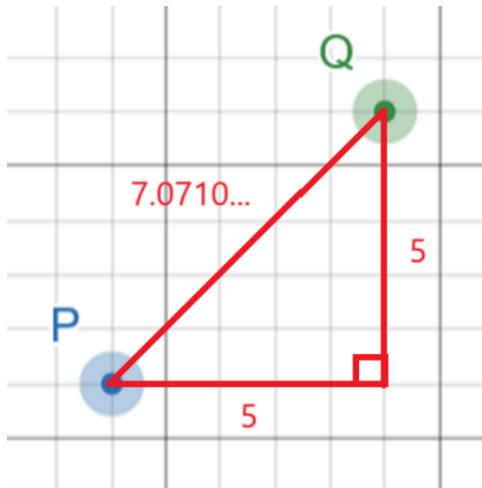
As a 1-dimensional vector, point A exists at (1), while point B exists at (7). The distance between these two points is simple: $7 - 1 = 6$, so $AB = 6$. Next, consider two dimensions; instead of a single axis, there are two, which can be represented as an x and a y on a graph. Much like the example above, the distance between two vectors (such as P and Q) can be described as the length of the distance vector $P - Q$ [2]. However, assuming the difference between vectors is not strictly on the x or y axis, the resulting distance will be at a diagonal. For example:



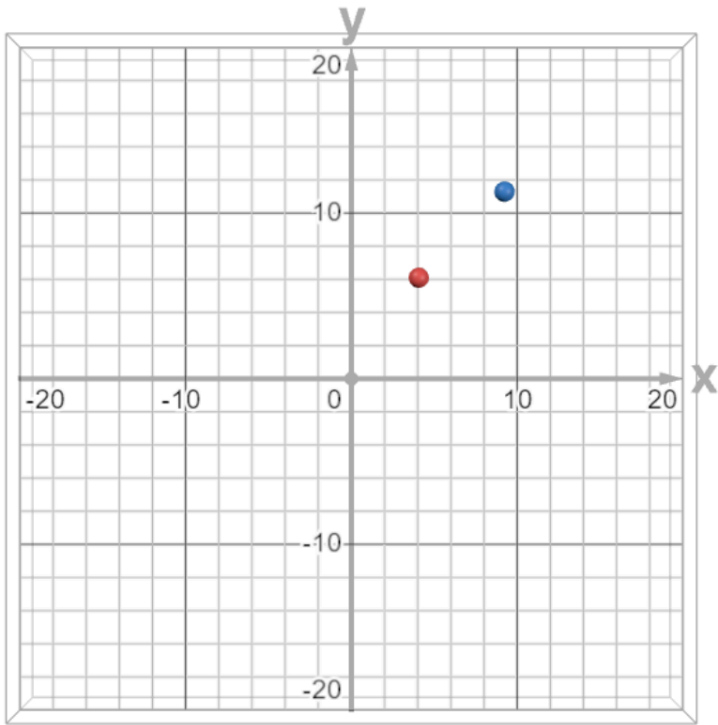
[3]

The graph above plots: *Vector P* = (4, 6), *Vector Q* = (9, 11). The distance between these two vectors however, can be defined through a variety of methods. One of the most common however, is “Euclidean Distance,” which defines the “length of a straight-line segment that would connect two points” [4], which conversely, can also be understood as “the distance as the crow flies”. Because Euclidean distance relies on Cartesian coordinates, it can be calculated through the Pythagorean theorem $a^2 + b^2 = c^2$, where a and b can be calculated as the absolute difference between x and y coordinates of each point vector respectively [5], like in the one-dimensional example above. Thus, Euclidean distance in two dimensions can be calculated with the formula: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ [6][7] (where x and y are the absolute distances, thus greater than 0 – note as well that Euclidean distance is a scalar quantity, and thus cannot be negative). With respect to the two-dimensional graph example with vectors P and Q, the Euclidean distance is calculated as:

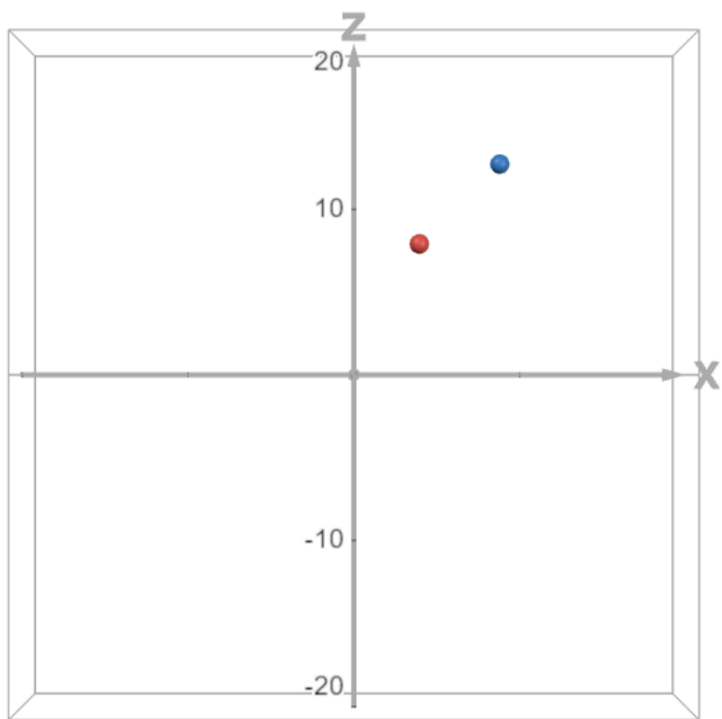
$$\sqrt{(9 - 4)^2 + (11 - 6)^2} = 7.07 \text{ (2dp)}.$$



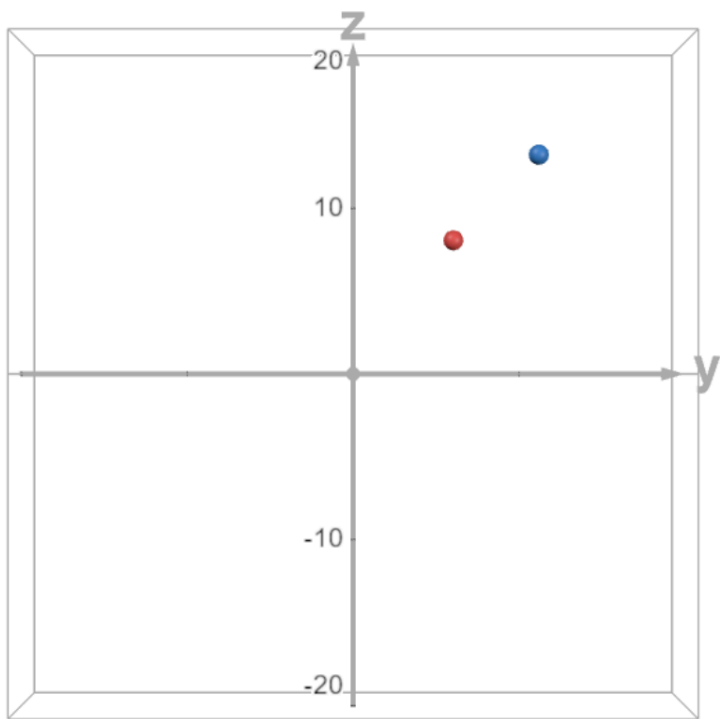
In three-dimensional space, the equation expands to include the absolute distance between the z coordinates of two points: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ [8]. Using the P and Q vector example:



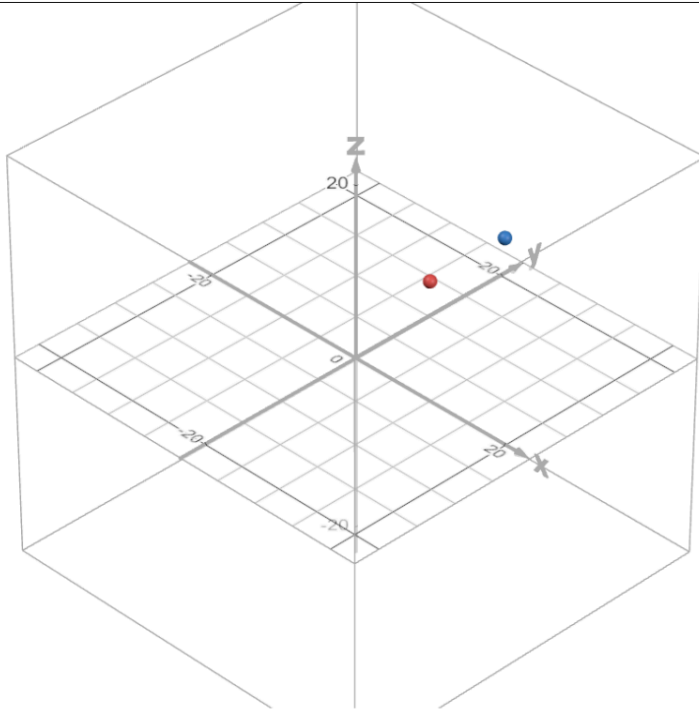
[3]



[3]



[3]



[3]

Where: $P = (4, 6, 8)$, $Q = (9, 11, 13)$. When input into the equation accounting for z coordinates, this results in $\sqrt{(9-4)^2 + (11-6)^2 + (13-8)^2} = 8.66$ (2dp).

Below is an implementation of this as C++ code:

Vector.h:

```
#pragma once
#ifndef VECTOR_H
#define VECTOR_H

#include <cmath>

class Vector
{
public:

    //Constructors
```

```

Vector(float x, float y, float z);
Vector(void);

//Methods
void Scale(float s);
void Add(Vector& v1);
void Add(Vector& v1, Vector& v2);
float euclidPToPDistance(Vector& v1, Vector& v2);

//Getter
Vector getVector(void);

private:
    float x;
    float y;
    float z;
};
#endif

```



Vector.cpp:

```

#include "Vector.h"
//Constructor
Vector::Vector(void) {};
Vector::Vector(float myx, float myy, float myz)
{
    x = myx;
    y = myy;
    z = myz;
}
void Vector::Scale(float s)
{
    x = s * x;
    y = s * y;
    z = s * z;
}
void Vector::Add(Vector& v1, Vector& v2)
{
    v1.x = v2.x + v1.x;
    v1.y = v2.y + v1.y;
    v1.z = v2.z + v1.z;
}
float Vector::euclidPToPDistance(Vector& v1, Vector& v2)
{
    float euclidDist = sqrt(((v1.x - v2.x) * (v1.x - v2.x)) + ((v1.y - v2.y) * (v1.y - v2.y)) + ((v1.z - v2.z) * (v1.z - v2.z)));
    return euclidDist;
    //This *should* always result in a positive number, but if it doesn't, come back to this.
}
Vector Vector::getVector(void)
{

```

```

        return Vector(x, y, z);
    }
    void Vector::Add(Vector& v1)
    {
        x = x + v1.x;
        y = y + v1.y;
        z = z + v1.z;
    }

```

Main .cpp file:

```

float x1, x2, y1, y2, z1, z2, dist;

std::cout << "\nPlease enter the x value for vector 1: ";
std::cin >> x1;
std::cout << "Please enter the y value for vector 1: ";
std::cin >> y1;
std::cout << "Please enter the z value for vector 1: ";
std::cin >> z1;
std::cout << "Vector 1 coordinates: (" << x1 << ", " << y1 << ", " << z1 << ")" << std::endl;

std::cout << "\nPlease enter the x value for vector 2: ";
std::cin >> x2;
std::cout << "Please enter the y value for vector 2: ";
std::cin >> y2;
std::cout << "Please enter the z value for vector 2: ";
std::cin >> z2;
std::cout << "Vector 2 coordinates: (" << x2 << ", " << y2 << ", " << z2 << ")" << std::endl;

Vector v1 = Vector(x1, y1, z1);
Vector v2 = Vector(x2, y2, z2);

dist = v1.euclidPToPDistance(v2); //Method needs identifier, v1 is there for now.
//A test shows this works, but this is likely wrong on some level.
//Research and/or revise?

std::cout << "\nEuclidean distance between vectors: " << dist << std::endl;

```

Console output of a test using earlier P and Q vectors example:

```

----- Assignment 2 Menu -----
1. Distance between two points task
2. Plane class task
3. Sphere class with collision detection task
4. Point to plane and sphere to plane collision detection task

Choose which task to perform: 1

Please enter the x value for vector 1: 4
Please enter the y value for vector 1: 6
Please enter the z value for vector 1: 8
Vector 1 coordinates: (4, 6, 8)

Please enter the x value for vector 2: 9
Please enter the y value for vector 2: 11
Please enter the z value for vector 2: 13
Vector 2 coordinates: (9, 11, 13)

Euclidean distance between vectors: 8.66025

C:\Users\queen\Documents\Visual Studio 2022\Projects\Game Architecture and Mechanics\Assignment2\x64\Debug\Assignment2.exe (process 15764) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

MAY BE WORTH TALKING ABOUT HOW EUCLIDEAN DISTANCE IS USED. MOVE ON TO EXPLAINING MANHATTAN, COSINE, AND DOT PRODUCT DISTANCES, ASSUMING THEY DON'T TURN UP IN NEXT TASKS.

3. Plane Class

2. A Plane Class (weighting 30%).

A **Plane** class is also required. This class uses two attributes, a **Vector** which defines the normalized plane normal, and a point, **P₀** which represents a point on the plane. Create and test a method that normalizes the plane normal vector. Create a method that calculates the distance of a point from the plane. Next create a method that tests whether a point is on the plane, in front of the plane, or behind the plane. Finally, include a helper method that calculates a plane's normal, given two vectors. In order to help with the point to plane distance calculation you may want to create a method that calculates the dot product of two vectors.

Consider vector (defines normalised plane normal)

Consider point P₀ (represents a point on the plane)

PLANE FORMED FROM A POINT AND A VECTOR. THE PLANE CAN ALSO ALTERNATIVELY BE UNDERSTOOD AS A LINE.

Need a method that normalises the plane normal vector

Also need a method that calculates the distance of a point from the plane (need to make a method that calculates dot product of two vectors)

Also also need a method that tests whether a point is on the plane, in front of the plane, or behind the plane (collision detection?)

Also also also need a helper method that calculates a plane's normal, given two vectors

Previously mentioned above under section 2 was the dot product. This involves the multiplication of two vectors together; Take the previous examples, $P = (4, 6, 8)$, $Q = (9, 11, 13)$. To obtain the dot product, the method is as follows: $\vec{P} \cdot \vec{Q} = P_1Q_1 + P_2Q_2 + P_3Q_3$, where subscripts are interchangeable for x, y and z terms [9]. With terms substituted, the equation becomes $\vec{P} \cdot \vec{Q} = (4 \times 9) + (6 \times 11) + (8 \times 13)$, resulting in 206, notably a scalar number, not a vector (product only has a size, not a direction [10]). In essence, this is because the dot product results in the length

(component) of one vector on another, multiplied by the normal length of the other vector (alternatively: the component of $\vec{P}$ along $\vec{Q}$ (the length), defined as $\|\vec{P}\| \cos \theta$, multiplied by $\|\vec{Q}\|$. In short, $\|\vec{P}\| \cdot \|\vec{Q}\| \cos \theta$) [9].

Now, consider a plane defined by a point, P_0 , and a vector.

Recall a plane defined by	Plane equation
normal $\mathbf{n} = (A, B, C)$	$Ax + By + Cz + D = 0$
point on plane P_0	$D = -(A \cdot P_{0x} + B \cdot P_{0y} + C \cdot P_{0z})$

Taking above, to test if a point is above, $Ax + By + Cz + D > 0$

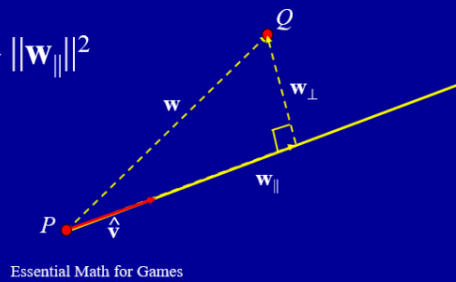
To test if point is below, $Ax + By + Cz + D < 0$

I'll be real, I think the powerpoint has the answers, but my ass is overloading trying to figure this out.



Line-Point Distance

- Line defined by point P and vector $\hat{\mathbf{v}}$
- Break vector $\mathbf{w} = Q - P$ into $\mathbf{w}_\perp$ and $\mathbf{w}_\parallel$
- $\mathbf{w}_\parallel = (\mathbf{w} \cdot \hat{\mathbf{v}}) \hat{\mathbf{v}}$
- $\|\mathbf{w}_\perp\|^2 = \|\mathbf{w}\|^2 - \|\mathbf{w}_\parallel\|^2$



Essential Math for Games

$\mathbf{w} \cdot \mathbf{v}^\wedge = \|\mathbf{w}\| \|\mathbf{v}^\wedge\| \cos A$. But $\|\mathbf{v}^\wedge\| = 1$ as it's normalized so $\mathbf{w} \cdot \mathbf{v}^\wedge = \|\mathbf{w}\| \cos A$, but from trig $\|\mathbf{w}\| \cos A = \|\mathbf{w}_\parallel\|$. So $\mathbf{w} \cdot \mathbf{v}^\wedge = \|\mathbf{w}_\parallel\|$. Multiply $\|\mathbf{w}_\parallel\|$ by the normalized vector $\mathbf{v}^\wedge$ to get the vector which points in the same direction as $\mathbf{v}$ but has length $\|\mathbf{w}_\parallel\|$. This is $\mathbf{w}_\parallel$ and is called the $\text{proj}_{\mathbf{w}}$. This is shown as bullet point 3 above.

- Final formula if $\mathbf{v}$ isn't normalized:

$$\text{dist}(P, \mathbf{v}, Q) = \sqrt{\left(\mathbf{w} \bullet \mathbf{w} - \frac{(\mathbf{w} \bullet \mathbf{v})^2}{\mathbf{v} \bullet \mathbf{v}} \right)}$$

4. Sphere Class with Collision Detection

3. A Sphere class with collision detection (weighting 25%).

A **Sphere** class is also required. You have also been asked to investigate sphere to sphere collision detection and to implement this collision detection method as part of the sphere class. It should detect a collision between two sphere bounding volumes. Your supervisor has not provided you with any further details and you may develop your own class and test harness for this method and you may use the Vector class you have already developed.

Make a sphere class

Implement sphere to sphere collision detection (as part of the sphere class)

Sphere collision is usually distance between centers, e.g. sphere to sphere collision is if distance "d" is less than radius1 + radius2, then colliding ((point to point distance measurement like in section 2))

Line to sphere collision is distance from line/plane to center (if $d < r$, colliding) ((This is why point to line distance above is important, see section 3))

Eggs!

5. Point to Plane and Sphere to Plane Collision Detection

4. Point to plane and sphere to plane collision detection. (weighting 25%).

This task asks you to create two further methods that implement collision tests. First create a method that tests for a collision between a point and a plane (place this in the plane class). Then extend this to create a sphere to plane collision test (place this in the sphere class).

Eggs!

Point to plane takes from section 3, i.e., check if point is on plane/line, as opposed to in front of/behind

Sphere to plane takes from section 4 almost 1-1.

6. Conclusion

Eggs!

7. References

- [1] D. D. Hearn, *Computer Graphics with OpenGL*, Pearson New International ed., Pearson Education, 2013, ch. 2, sec.I, pp. 30. [E-Book]
- [2] "Digression on Length and Distance in Vector Spaces," *Mit.edu*. [Online]. Available: <https://ocw.mit.edu/ans7870/18/18.013a/textbook/HTML/chapter03/section08.html>. [Accessed: 22-Dec-2024].
- [3] "Desmos," *Desmos*. [Online]. Available: <https://www.desmos.com/calculator>. [Accessed: 22-Dec-2024].
- [4] The Editors of Encyclopedia Britannica, "Euclidean distance," *Encyclopedia Britannica*. 12-Nov-2024.
- [5] K. Smith, *Precalculus: A Functional Approach to Graphing and Problem Solving*, 6th ed., Jones and Bartlett Publishers Inc, 2011, pp. 8.
- [6] D. Cohen, *Precalculus: A Problems-Oriented Approach*, 6th ed., Cengage Learning, 2004.
- [7] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, A. Nannarelli, M. Re, and S. Spano, "N -Dimensional Approximation of Euclidean Distance," *IEEE transactions on circuits and systems. II, Express briefs*, vol. 67, no. 3, pp. 565–569, 2020, doi: 10.1109/TCSII.2019.2919545.
- [8] "Euclidean distance," *GeeksforGeeks*, 13-Mar-2024. [Online]. Available: <https://www.geeksforgeeks.org/euclidean-distance/>. [Accessed: 23-Dec-2024].
- [9] J. Gibson, TMW Media Group., and TMW Media Group., Venice, CA. *Calculus 3 tutor. Vector dot product*, (2011).
- [10] "Vectors and Scalars - National 5 Physics Revision," *BBC Bitesize*. [Online]. Available: <https://www.bbc.co.uk/bitesize/guides/zydyxfr/revision/1>. [Accessed: 03-Jan-2025].
- [11]
- [12]
- [13]
- [14]
- [15]
- [16]
- [17]
- [18]
- [19]
- [20]

Rose Plumley Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

50/100

Up to page 9: Very well written report, fantastic research and great implementations.

What happened after? Eggs?

PAGE 1

PAGE 2

PAGE 3



Comment 1

Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering, equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10



Comment 2

Good catch

PAGE 11

PAGE 12



Comment 3

do not overload your ass.