

Games Architecture Mechanics

Part 1a

by May Davies

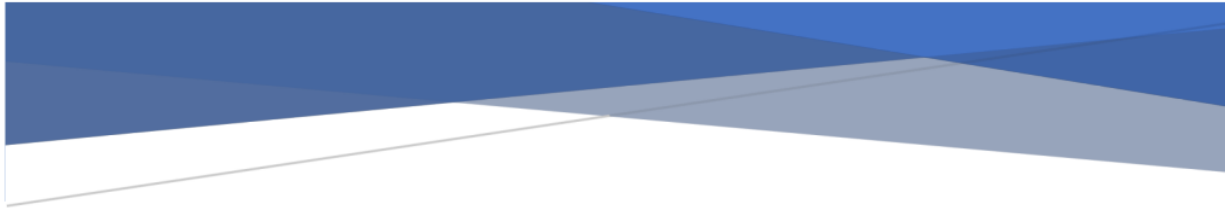
Submission date: 01-Dec-2020 12:00AM (UTC+0000)

Submission ID: 137918873

File name: 17857_May_Davies_Games_Architecture_Mechanics_Part_1a_234537_901843580.docx (1.67M)

Word count: 4341

Character count: 20816



GAMES ARCHITECTURE AND MECHANICS

Part 1a: Games Physics

May Davies

Contents

1.Declaration.....	2
2.Abstract.....	2
3.Introduction	2
4.Part One	2
4.1.Euler’s Numerical Integration	2
4.2.Verlet Integration.....	3
4.2.1Velocity Verlet.....	3
4.3.Scenario Work.....	4
4.3.1Testing Harness.....	7
5.Part Two.....	13
5.1 Task 1	13
5.1.1Task One Testing	14
5.2Task 2 – Plane Class.....	16
5.3Sphere to sphere collision.....	19
References	20
Appendix A – Euler Code.....	21
Appendix B – Velocity Verlet Integration Code	22
Appendix C – Euler and Velocity Verlet Tables	23
Appendix D.....	24
Appendix E	27
Appendix F	28

1. Declaration

The work of this report has been completed under the supervision of Gordon Dickers. I declare that this report is the product of my own work and is not being submitted for any other degree etc.

May Davies.

2. Abstract

This report goes over code for various different tasks involving vectors, and also involving numeric integration. Some of these tests also end up failing.

3. Introduction

This report is a report that aims to cover a few areas of games architecture and mechanics over, and as such will go over some areas of the physics involved in video games in depth for some of these tasks. The first task looks over Euler's numerical integration method, as well as some other integration methods, as well as a coded example for a spring powered model dragster car for each method discussed.

The rest of this report then goes over a scenario for writing a vector class for a games engine, using pre-existing vector classes and will go over some details of vector operations. The second task also goes over a planes class that should be capable of outputting a normalized plane vector and the distance of a point from a respective plane. the final task of the report involve sphere-to-sphere collision detection, of which will use the vector class developed within the second task.

4. Part One

4.1. Euler's Numerical Integration

One particular method of Numerical Integration is known as Euler's Method. Euler's Method was made by Leonhard Euler, and was published in his book *Institutionum Calculi Integralis* which was published in 1755 ^[1]



Euler's Method is a decent method of numerical integration, in that it can be used to make approximations for what is about to happen to something based on its current position and velocity, to then approximate the next position and velocity [2]. A simple way of showing a formula would be as follows, which assumes that there is a separate function for position and velocity:

$$\begin{aligned}X_{i+1} &= X_i + h \cdot X' \\ &= X_i + h \cdot V_i \\ V_{i+1} &= V_i + h \cdot V' \\ &= V_i + h \cdot F(X_i, V_i)/m\end{aligned}$$

It should be noted that there is a formula nearly identical to this in Bishop and Van Verth's book *Essential Mathematics For Games & Interactive Applications A programmers Guide* in that the last line also has a value t in the function of F(Xi, Vi) which represents Taylor's Series[2] however, this

formula is specifically on the current timestep only, so for Euler's numerical integration, the value of t is not shown.

In this particular example, X represents position, while V represents velocity, and h represents the timestep (which in some cases may be referred to as Δt for change in time). Attention should also be paid to the last equation. The section for F , is a function for acceleration, which is then divided by mass.

While Euler's Numerical Integration method can be pretty useful for video games, as it does not require a lot of processing as the physics are simplified in comparison to reality, it does however have one particular flaw. If the timestep (the value for h in the previous equations) is too big, then inaccuracies will begin to occur, as the next calculation for the position for example, requires the current position to be added to the product of h and X' . The reason for this inaccuracy is that for each and every iteration of X , is based on approximation, so if a timestep is huge, an inaccuracy will occur, which will result in wildly different results in comparison to a small timestep [3].

A good example of this, could be a car travelling at a fast speed, and then suddenly coming to a stop. In Euler's method, if the time step is too large, the car could come to a stop and then move in the opposite direction as there is effectively extra energy when there shouldn't be.

4.2 Verlet Integration

An alternative integration method to Euler's method could also be considered, which is known as Verlet Integration. Verlet Integration is a method of integration that can typically be used in video game programming, as it happens to be particularly handy for simulating bodies that could be considered soft like rope, blankets, vines and such other objects that can somewhat change shape in some way without losing their original shape (so for instance, not a liquid such as water)[4].

It should be noted that there are several different versions of Verlet Integration that can be used in video games. These include Verlet (aka positional verlet), Leapfrog Verlet, and lastly Velocity Verlet[4]. For this particular report, the chosen method of Verlet Integration is Velocity Verlet.

4.2.1 Velocity Verlet

Velocity Verlet is an interesting method of integration, as in Velocity Verlet, when actually performing the calculation, acceleration is calculated twice as opposed to only doing it once as seen in Euler. The following is an equation for Velocity Verlet:

$$\begin{aligned}V_{i+1/2} &= V_i + h_i / 2a \\X_{i+1} &= X_i + h_i V_{i+1/2} \\V_{i+1} &= V_{i+1/2} + h_i / 2a_{i+1}\end{aligned}$$

It should be noticed that upon close inspection, acceleration has to be calculated twice in this formula. First it must be calculated at the start in order to get the value of velocity at the half step, and then the next acceleration must be calculated before the full step of V , which can be seen in the last line of this formula with $2a_{i+1}$.

It should be noted that here there is no function for force, which was seen in Euler. The reason for this is because with verlet, a single step of Euler or any other first order integration must be done first, and then Verlet can pick up from there.

4.3Scenario Work

The scenario for this particular report that requires code to be written for is for a model dragster car that is being pushed by a spring. The tractive force of the spring is as follows:

$$F_t = 1.5(16 - x)$$

It should be noted that when x is greater than 8, F_t must be set to 0 newtons.

This particular scenario also requires that a drag force calculation also be done on the dragster car, which consists of the following formula (wherein F_d is drag force):

$$F_d = C_R mg + \frac{1}{2} C_D \rho A V^2$$

C_R is coefficient of rolling resistance. For this scenario this needs to be 0.21

m is mass. The car's mass in this scenario is 1kg.

g is the force of gravity. For this scenario, gravity is set to 9.81 m/s^2

C_D is the coefficient for drag, which is set to 0.34.

ρ is the measurement for air density. In this scenario, air density is 1.2 kg/m^3

A represents the area at the front of the vehicle which is 0.01 m^2

V is of course, the velocity of the vehicle, which will be calculated using the previously discussed methods of numerical integration.

It should also be noted that if the velocity is ever reach below 0, then F_d should also be set to 0. This is to prevent any oscillation to be shown in the simulation due to any possible issues with the timesteps.

This scenario's information will be presented twice to the user. The first time around the simulation will be ran using Euler Integration, and the second time around the simulation will be ran using Velocity Verlet Integration.

The Euler Integration method has been implemented first. The code for this can be seen in Appendix A, and for Verlet Integration please see Appendix B.

It should be noted that one of the requirements for the scenario is to save the data from the car scenario to a file, as such, the method that is used for this scenario is going to make include `<fstream>` in c++ in order to write and read to a file[5].

Proof of correct file handling can be seen below, with a text file created for each integration method used.

EulerResults.txt - Notepad					VelocityVerletResults.txt - Notepad				
Euler's method on car scenario					Velocity Verlet method on car scenario				
N,	time,	x,	v,	Acceleration	N,	time,	x,	v,	Acceleration
0	0.00s	0.43metres	2.85	19.00m/s/s	Euler 0.00s	0.43metres	2.85	19.00m/s/s	
1	0.15s	1.27metres	5.60	18.36m/s/s	0	0.15s	0.85metres	5.60	18.36m/s/s
2	0.30s	2.49metres	8.17	17.10m/s/s	1	0.30s	1.70metres	8.26	17.71m/s/s
3	0.45s	4.06metres	10.46	15.26m/s/s	2	0.45s	2.93metres	10.73	16.45m/s/s
4	0.60s	5.92metres	12.39	12.90m/s/s	3	0.60s	4.54metres	12.92	14.60m/s/s
5	0.75s	8.01metres	13.91	10.12m/s/s	4	0.75s	6.48metres	14.74	12.18m/s/s
6	0.90s	9.98metres	13.16	-5.00m/s/s	5	0.90s	8.69metres	16.14	9.28m/s/s
7	1.05s	11.84metres	12.41	-5.00m/s/s	6	1.05s	11.11metres	15.39	-5.00m/s/s
8	1.20s	13.59metres	11.66	-5.00m/s/s	7	1.20s	13.42metres	14.64	-5.00m/s/s
9	1.35s	15.23metres	10.91	-5.00m/s/s	8	1.35s	15.62metres	13.88	-5.00m/s/s
10	1.50s	16.75metres	10.16	-5.00m/s/s	9	1.50s	17.70metres	13.13	-5.00m/s/s
11	1.65s	18.16metres	9.41	-5.00m/s/s	10	1.65s	19.67metres	12.38	-5.00m/s/s
12	1.80s	19.46metres	8.66	-5.00m/s/s	11	1.80s	21.53metres	11.63	-5.00m/s/s
13	1.95s	20.64metres	7.91	-5.00m/s/s	12	1.95s	23.27metres	10.88	-5.00m/s/s
14	2.10s	21.72metres	7.15	-5.00m/s/s	13	2.10s	24.90metres	10.13	-5.00m/s/s
15	2.25s	22.68metres	6.40	-5.00m/s/s	14	2.25s	26.42metres	9.38	-5.00m/s/s
16	2.40s	23.53metres	5.65	-5.00m/s/s	15	2.40s	27.83metres	8.63	-5.00m/s/s
17	2.55s	24.26metres	4.90	-5.00m/s/s	16	2.55s	29.13metres	7.88	-5.00m/s/s
18	2.70s	24.89metres	4.15	-5.00m/s/s	17	2.70s	30.31metres	7.13	-5.00m/s/s
19	2.85s	25.40metres	3.40	-5.00m/s/s	18	2.85s	31.38metres	6.38	-5.00m/s/s
20	3.00s	25.79metres	2.65	-5.00m/s/s	19	3.00s	32.33metres	5.63	-5.00m/s/s
21	3.15s	26.08metres	1.90	-5.00m/s/s	20	3.15s	33.18metres	4.88	-5.00m/s/s
22	3.30s	26.25metres	1.15	-5.00m/s/s	21	3.30s	33.91metres	4.13	-5.00m/s/s
23	3.45s	26.31metres	0.40	-5.00m/s/s	22	3.45s	34.53metres	3.38	-5.00m/s/s
24	3.60s	26.26metres	-0.35	-5.00m/s/s	23	3.60s	35.04metres	2.63	-5.00m/s/s
25	3.75s	26.26metres	-0.35	0.00m/s/s	24	3.75s	35.43metres	1.88	-5.00m/s/s
26	3.90s	26.26metres	-0.35	0.00m/s/s	25	3.90s	35.71metres	1.13	-5.00m/s/s
27	4.05s	26.26metres	-0.35	0.00m/s/s	26	4.05s	35.88metres	0.38	-5.00m/s/s
28	4.20s	26.26metres	-0.35	0.00m/s/s	27	4.20s	35.94metres	-0.37	-5.00m/s/s
29	4.35s	26.26metres	-0.35	0.00m/s/s	28	4.35s	35.94metres	-0.37	-5.00m/s/s
					29	4.50s	35.94metres	-0.37	-5.00m/s/s

It should be noted that at this point, several of the original values for the Fd equation were changed in order to have the car stop at a reasonable point to demonstrate both the Euler integration, and the Velocity Verlet integration without having too much data as to speak. These are the values at the time of writing for simulation that have changed from the scenario:

$$C_R = 0.51$$

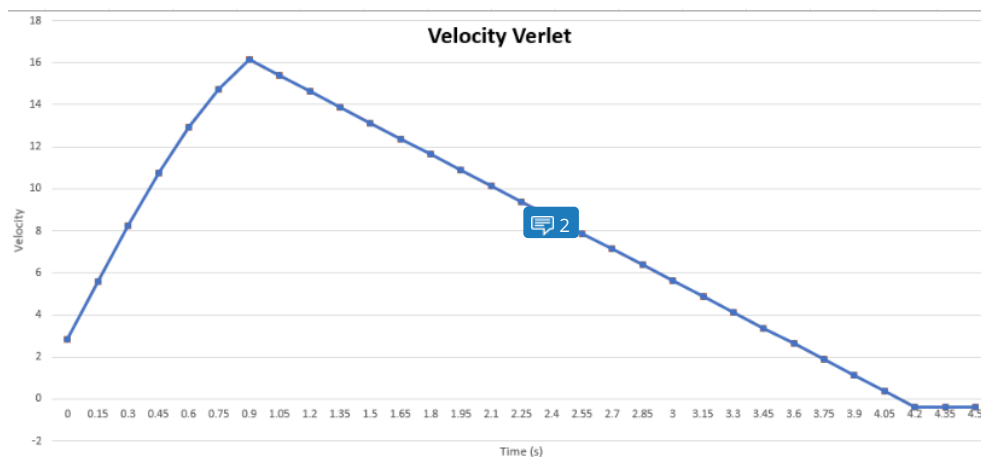
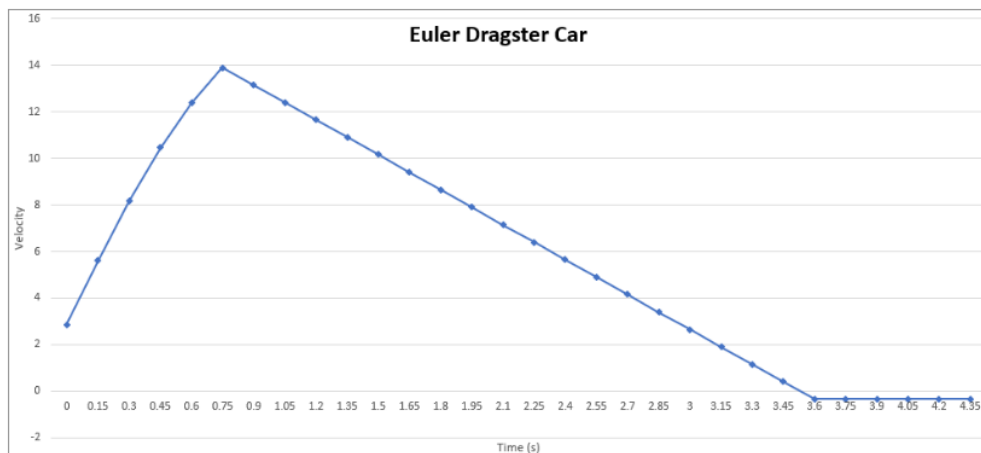
$$A = 0.10m^2$$

$$C_D = 0.45$$

$$\text{timestep: } 0.15 \text{ s}$$

This debatably could be too large of a timestep, however, this timestep allowed the car to come to a stop within 30 iterations, which was what would be decided to be a decent frame for testing that the integration methods actually do work.

The following graphs have been produced as a result of the data calculated through first the Euler method, and then the Velocity Verlet method. (See next page for Euler Chart and Velocity Verlet chart respectively). Additionally, for the results shown above, a table of results can also be seen in Appendix C.

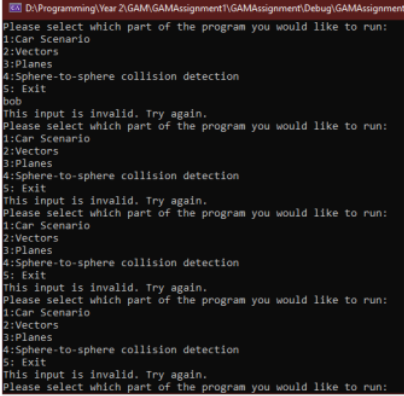
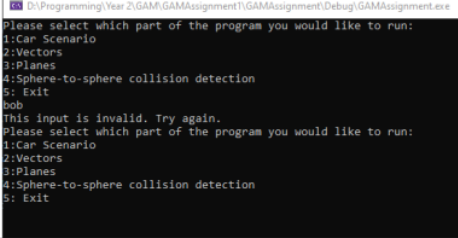
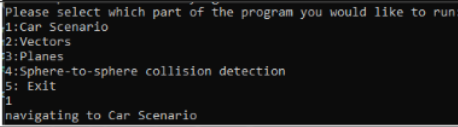
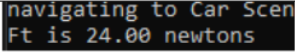
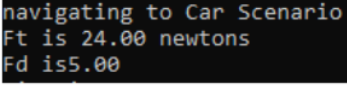


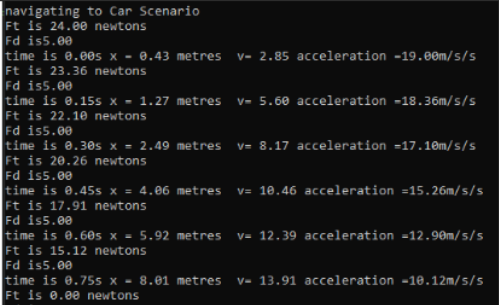
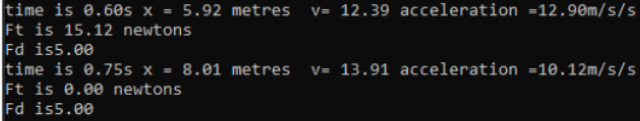
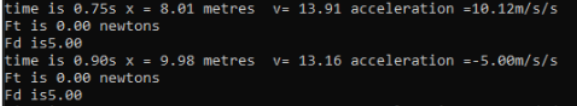
At a glance, these two charts may appear the same, however in closer inspection, it can be seen that there are some slight differences between Euler and Velocity Verlet. It can be seen that with each step that the margin of error between Euler and Verlet increasing as the simulation goes on. The only time that Euler and Velocity verlet are ever the same within this particular simulation is in the first 2 steps of integration, after this point there is a considerably increasing gap between the two. Euler's integration actually even ends in a shorter amount of time than Velocity Verlet does, as can be seen by the graphs above. It should however be noted that possibly due to timestep, that the car supposedly stops moving at a negative velocity, which of course is impossible, the reason for this is because a piece of code had to be written to prevent the position from continuing to increase at a steady rate, which checked to increase position based on if the velocity and the drag were both equal to 0. The reason this problem occurred of course is because the large timestep meant that eventually, a value for velocity would be written below 0.

A potential fix for this could have possibly been to decrease the timestep the closer and closer the simulation got to 0 velocity, however this in itself could be an inefficient use of Velocity Verlet. Aside from this, it could also be suggested that the reason that the car stops later in Velocity Verlet than it does in Euler could be due to it possibly being much more accurate, thanks to the acceleration having to be calculated twice within each iteration of Velocity Verlet.

4.3.1 Testing Harness

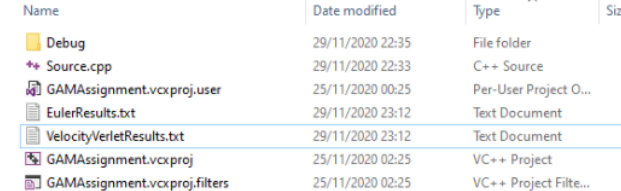
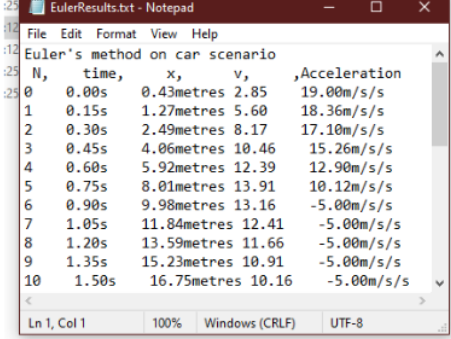
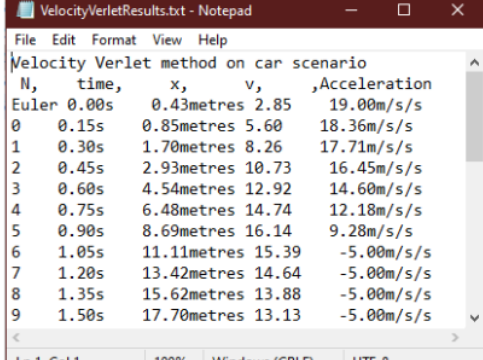
The following section of this report is a testing harness for the Dragster Car scenario, which goes over some of the tests done both during and post development of the Dragster Car simulation used for this particular report. It should be expected that not much validation testing should be necessary, as the program is being written in a way that does not allow user inputs outside of a main menu. (i.e, the values necessary for the simulation are not changeable by the user, only by the programmer.) tests 1-10 are on the Euler system, while tests 11 onwards are Velocity Verlet.

Test No.	Test	Expected Result	Actual Result	Success? Changes?
1	User enters an input that is not a number for the main menu to select the simulation	Input should fail. User should be told to pick again.		Fail. Didn't actually use stringstream, so cin >> menuInput would not get asked for again, and would instead be infinitely looped. Implement stringstream to fix this.
2	Repeat of Test 2.	See test 2 expected result		Success.
3	Enter 1 in order to go to the Car Scenario	Car Scenario should be navigated to.		Success
4	Test that initially Ft should be equal to 24, as $1.5 \cdot (16 - x)$ when x is 0 should output 24.	Ft should be output as 24 newtons.		Success
5	Test that the first value for Ft is correct,	Fd should be 5.00 (not 5.0031, as there is already		Success.

	given that Coefficient Roll = 0.51, m= 1.00, g = 9.81, the (1/2), Coefficient drag = 0.45, air density = 1.20, area =0.10, and at this stage velocity is 0, thus v^2 is 0. Using the calculation for Fd, this should be 5.0031	rounding to 2 decimal places implemented in code.		
6	Check that the outputs increase velocity as acceleration is positive.	Velocity should increase as acceleration is positive. Only the first few results will be checked as opposed to all 30.		Success.
7	Check that once x is 8 that Ft becomes 0.	Once x = 8, ft should become 0.		Success.
8	Check that once the value of x has past 8, that acceleration should instead become deceleration	Acceleration should move from being 10.12m/s/s to a negative acceleration (aka deceleration)		Success.
9	Check that x stops changing once velocity is less than 0	X should stop changing once velocity is less than 0.	No screenshot available. X did not stop increasing, uncertain as to why in particular. Could potentially be down to stepsize, though all step sizes attempted so far have yield the same result.	Fail. Forced the simulation to no longer calculate x and v once v

				is less than 0.
10	Check that when $v < 0$, F_d becomes 0.	F_d should become 0.	<pre> Ft is 24.00 time is 3.45s x = 26.31 metres v = 0.40 acceleration = -5.00m/s/s Ft is 0.00 newtons F_d is 5.00 time is 3.60s x = 26.26 metres v = -0.35 acceleration = -5.00m/s/s The car has stopped. Ft is 0.00 newtons F_d is 0.00 </pre>	Success. Note: only for the timestep that the car is first classed as stopped is acceleration still -5. After that step, acceleration becomes 0.
11	Check that the values used previously have been reset to the values that were changed in section 4.3.	Values should all be reset.		Success.
12	Check that the first step performed before going straight in to Velocity Verlet is Euler	Watching the variables from test 11 for the Euler section, then compare to the Euler that appears in the initial step for Velocity Verlet. Results should be the same.		Success.
13	Test that initially F_t should be equal to 24, as $1.5 \cdot (16 - x)$ when x is 0 should output 24	F_t should be output as 24 newtons.	<pre> Beginning Velocity verlet, with a single step of Euler Ft is 24.00 F_d is 5.00 x = 0.43 v= 2.85 acceleration =19.00 Euler done. </pre>	Success.
14	Test that the first value for F_t is correct, given that Coefficient	F_d should be 5.00 (not 5.0031, as there is already rounding to 2 decimal places	<pre> Beginning Velocity verlet, with a single step of Euler Ft is 24.00 F_d is 5.00 x = 0.43 v= 2.85 acceleration =19.00 Euler done. </pre>	Success.

	Roll = 0.51, m= 1.00, g = 9.81, the (1/2), Coefficient drag = 0.45, air density = 1.20, area =0.10, and at this stage velocity is 0, thus v^2 is 0. Using the calculation for F_d, this should be 5.0031	implemented in code.		
15	Check that the outputs increase while velocity as long as acceleration is positive.	Velocity should increase while acceleration is positive. Only the first few results will be checked as opposed to all 30.	<pre> Beginning Velocity verlet, with a single step of Euler Ft is 24.00 Fd is 5.00 x = 0.43 v= 2.85 acceleration =19.00 Euler done. Ft is 23.36 Fd is 5.00 time is 0.15x = 0.85 v= 5.60 acceleration =18.36 time is 0.15s x = 0.85 metres v= 5.60 acceleration =18.36m/s/s Ft is 22.72 Fd is 5.00 time is 0.30x = 1.70 v= 8.26 acceleration =17.71 time is 0.30s x = 1.70 metres v= 8.26 acceleration =17.71m/s/s Ft is 21.46 Fd is 5.00 time is 0.45x = 2.93 v= 10.73 acceleration =16.45 time is 0.45s x = 2.93 metres v= 10.73 acceleration =16.45m/s/s Ft is 19.60 Fd is 5.00 time is 0.60x = 4.54 v= 12.92 acceleration =14.60 time is 0.60s x = 4.54 metres v= 12.92 acceleration =14.60m/s/s Ft is 17.18 Fd is 5.00 time is 0.75x = 6.48 v= 14.74 acceleration =12.18 time is 0.75s x = 6.48 metres v= 14.74 acceleration =12.18m/s/s Ft is 14.28 </pre>	Success.
16	Check that once x is 8 that F_t becomes 0.	Once $x = 8$, F_t should become 0.	<pre> time is 0.75x = 6.48 v= 14.74 acceleration =12.18 time is 0.75s x = 6.48 metres v= 14.74 acceleration =12.18m/s/s Ft is 14.28 Fd is 5.00 time is 0.90x = 8.69 v= 16.14 acceleration =9.28 time is 0.90s x = 8.69 metres v= 16.14 acceleration =9.28m/s/s Ft is 0.00 Fd is 5.00 </pre>	Success.
17	Check that once the value of x has past 8, that acceleration should instead become deceleration	Acceleration should become negative after $x > 8$	<pre> time is 0.90x = 8.69 v= 16.14 acceleration =9.28 time is 0.90s x = 8.69 metres v= 16.14 acceleration =9.28m/s/s Ft is 0.00 Fd is 5.00 time is 1.05x = 11.11 v= 15.39 acceleration =-5.00 time is 1.05s x = 11.11 metres v= 15.39 acceleration =-5.00m/s/s Ft is 0.00 Fd is 5.00 time is 1.20x = 13.42 v= 14.64 acceleration =-5.00 time is 1.20s x = 13.42 metres v= 14.64 acceleration =-5.00m/s/s </pre>	Success.
18	Check that x stops changing	X should stop changing once	No screenshot available. X did not stop increasing, uncertain as to why in particular. Could potentially be down to stepsize, though all step sizes attempted so far have yield the same result.	Fail. Forced the simulation

	once velocity is less than 0	velocity is less than 0.		to no longer calculate x and v once v is less than 0.
19	Check that when $v < 0$, fd becomes 0.	Fd should become 0.	<pre> time is 3.60s x = 35.04 metres v= 2.63 acceleration =-5.00m/s/s t is 0.00 d is 5.00 time is 3.75s x = 35.43 v= 1.88 acceleration =-5.00 time is 3.75s x = 35.43 metres v= 1.88 acceleration =-5.00m/s/s t is 0.00 d is 5.00 time is 3.90s x = 35.71 v= 1.13 acceleration =-5.00 time is 3.90s x = 35.71 metres v= 1.13 acceleration =-5.00m/s/s t is 0.00 d is 5.00 </pre>	Success.
20	Check that File handling creates the files for Euler and for Velocity Verlet	Files for Euler and Velocity Verlet should be made once the simulation has been executed.		Success.
21	Check that the data from the simulations has been saved to the Euler text document.	Data from euler simulation should be written to EulerResults.txt		Success.
22	Check that the data from the Velocity Verlet simulation is saved to the Velocity Verlet text document.	Data from the Velocity Verlet simulation should be written to VelocityVerlet-results.txt		Success.

23	Check that neither Euler nor Velocity Verlet can be ran using a negative mass, as a negative mass is impossible.	Should reset mass to 1kg if a negative value for mass was set.	<pre> 1 navigating to Car Scenario Ft is 24.00 newtons Fd is 5.00 time is 0.00s x = -0.65 metres v= -4.35 acceleration =-29.00m/s/s Ft is 24.98 newtons Fd is 0.00 time is 0.15s x = -1.87 metres v= -8.10 acceleration =-24.98m/s/s Ft is 26.80 newtons Fd is 0.00 time is 0.30s x = -3.88 metres v= -12.12 acceleration =-26.80m/s/s Ft is 29.53 newtons Fd is 0.00 time is 0.45s x = -6.17 metres v= -16.55 acceleration =-29.53m/s/s Ft is 33.25 newtons Fd is 0.00 time is 0.60s x = -9.40 metres v= -21.53 acceleration =-33.25m/s/s Ft is 38.10 newtons Fd is 0.00 time is 0.75s x = -13.48 metres v= -27.25 acceleration =-38.10m/s/s Ft is 44.23 newtons Fd is 0.00 time is 0.90s x = -18.57 metres v= -33.88 acceleration =-44.23m/s/s Ft is 51.85 newtons Fd is 0.00 time is 1.05s x = -24.82 metres v= -41.66 acceleration =-51.85m/s/s Beginning Velocity verlet, with a single step of Euler Ft is 24.00 Fd is 5.00 x = -0.65 v= -4.35 acceleration =-29.00 Euler done. Ft is 24.98 Fd is 0.00 time is 0.15s x = -1.31 v= -8.10 acceleration =-24.98 time is 0.15s x = -1.31 metres v= -8.10 acceleration =-24.98m/s/s Ft is 25.96 Fd is 0.00 time is 0.30s x = -2.52 v= -11.99 acceleration =-25.96 time is 0.30s x = -2.52 metres v= -11.99 acceleration =-25.96m/s/s Ft is 27.78 Fd is 0.00 time is 0.45s x = -4.32 v= -16.16 acceleration =-27.78 time is 0.45s x = -4.32 metres v= -16.16 acceleration =-27.78m/s/s Ft is 29.48 Fd is 0.00 time is 0.60s x = -6.17 v= -21.53 acceleration =-29.48 time is 0.60s x = -6.17 metres v= -21.53 acceleration =-29.48m/s/s Ft is 33.25 Fd is 0.00 time is 0.75s x = -9.40 v= -27.25 acceleration =-33.25 time is 0.75s x = -9.40 metres v= -27.25 acceleration =-33.25m/s/s Ft is 38.10 Fd is 0.00 time is 0.90s x = -13.48 v= -33.88 acceleration =-38.10 time is 0.90s x = -13.48 metres v= -33.88 acceleration =-38.10m/s/s Ft is 44.23 Fd is 0.00 time is 1.05s x = -18.57 v= -41.66 acceleration =-44.23 time is 1.05s x = -18.57 metres v= -41.66 acceleration =-44.23m/s/s Ft is 51.85 Fd is 0.00 time is 1.20s x = -24.82 v= -49.49 acceleration =-51.85 time is 1.20s x = -24.82 metres v= -49.49 acceleration =-51.85m/s/s Ft is 59.47 Fd is 0.00 time is 1.35s x = -30.95 v= -57.32 acceleration =-59.47 time is 1.35s x = -30.95 metres v= -57.32 acceleration =-59.47m/s/s Ft is 69.00 Fd is 0.00 time is 1.50s x = -37.10 v= -65.15 acceleration =-69.00 time is 1.50s x = -37.10 metres v= -65.15 acceleration =-69.00m/s/s Ft is 79.53 Fd is 0.00 time is 1.65s x = -43.25 v= -72.98 acceleration =-79.53 time is 1.65s x = -43.25 metres v= -72.98 acceleration =-79.53m/s/s Ft is 91.00 Fd is 0.00 time is 1.80s x = -49.40 v= -80.81 acceleration =-91.00 time is 1.80s x = -49.40 metres v= -80.81 acceleration =-91.00m/s/s Ft is 103.48 Fd is 0.00 time is 1.95s x = -55.55 v= -88.64 acceleration =-103.48 time is 1.95s x = -55.55 metres v= -88.64 acceleration =-103.48m/s/s Ft is 116.90 Fd is 0.00 time is 2.10s x = -61.70 v= -96.47 acceleration =-116.90 time is 2.10s x = -61.70 metres v= -96.47 acceleration =-116.90m/s/s Ft is 131.25 Fd is 0.00 time is 2.25s x = -67.85 v= -104.30 acceleration =-131.25 time is 2.25s x = -67.85 metres v= -104.30 acceleration =-131.25m/s/s Ft is 146.53 Fd is 0.00 time is 2.40s x = -74.00 v= -112.13 acceleration =-146.53 time is 2.40s x = -74.00 metres v= -112.13 acceleration =-146.53m/s/s Ft is 162.75 Fd is 0.00 time is 2.55s x = -80.15 v= -120.00 acceleration =-162.75 time is 2.55s x = -80.15 metres v= -120.00 acceleration =-162.75m/s/s Ft is 179.90 Fd is 0.00 time is 2.70s x = -86.30 v= -127.87 acceleration =-179.90 time is 2.70s x = -86.30 metres v= -127.87 acceleration =-179.90m/s/s Ft is 197.98 Fd is 0.00 time is 2.85s x = -92.45 v= -135.74 acceleration =-197.98 time is 2.85s x = -92.45 metres v= -135.74 acceleration =-197.98m/s/s Ft is 216.98 Fd is 0.00 time is 3.00s x = -98.60 v= -143.61 acceleration =-216.98 time is 3.00s x = -98.60 metres v= -143.61 acceleration =-216.98m/s/s Ft is 236.90 Fd is 0.00 time is 3.15s x = -104.75 v= -151.48 acceleration =-236.90 time is 3.15s x = -104.75 metres v= -151.48 acceleration =-236.90m/s/s Ft is 257.75 Fd is 0.00 time is 3.30s x = -110.90 v= -159.35 acceleration =-257.75 time is 3.30s x = -110.90 metres v= -159.35 acceleration =-257.75m/s/s Ft is 279.53 Fd is 0.00 time is 3.45s x = -117.05 v= -167.22 acceleration =-279.53 time is 3.45s x = -117.05 metres v= -167.22 acceleration =-279.53m/s/s Ft is 302.25 Fd is 0.00 time is 3.60s x = -123.20 v= -175.09 acceleration =-302.25 time is 3.60s x = -123.20 metres v= -175.09 acceleration =-302.25m/s/s Ft is 325.90 Fd is 0.00 time is 3.75s x = -129.35 v= -182.96 acceleration =-325.90 time is 3.75s x = -129.35 metres v= -182.96 acceleration =-325.90m/s/s Ft is 350.48 Fd is 0.00 time is 3.90s x = -135.50 v= -190.83 acceleration =-350.48 time is 3.90s x = -135.50 metres v= -190.83 acceleration =-350.48m/s/s Ft is 375.98 Fd is 0.00 time is 4.05s x = -141.65 v= -198.70 acceleration =-375.98 time is 4.05s x = -141.65 metres v= -198.70 acceleration =-375.98m/s/s Ft is 402.40 Fd is 0.00 time is 4.20s x = -147.80 v= -206.57 acceleration =-402.40 time is 4.20s x = -147.80 metres v= -206.57 acceleration =-402.40m/s/s Ft is 429.75 Fd is 0.00 time is 4.35s x = -153.95 v= -214.44 acceleration =-429.75 time is 4.35s x = -153.95 metres v= -214.44 acceleration =-429.75m/s/s Ft is 458.03 Fd is 0.00 time is 4.50s x = -160.10 v= -222.31 acceleration =-458.03 time is 4.50s x = -160.10 metres v= -222.31 acceleration =-458.03m/s/s Ft is 487.25 Fd is 0.00 time is 4.65s x = -166.25 v= -230.18 acceleration =-487.25 time is 4.65s x = -166.25 metres v= -230.18 acceleration =-487.25m/s/s Ft is 517.40 Fd is 0.00 time is 4.80s x = -172.40 v= -238.05 acceleration =-517.40 time is 4.80s x = -172.40 metres v= -238.05 acceleration =-517.40m/s/s Ft is 548.48 Fd is 0.00 time is 4.95s x = -178.55 v= -245.92 acceleration =-548.48 time is 4.95s x = -178.55 metres v= -245.92 acceleration =-548.48m/s/s Ft is 580.50 Fd is 0.00 time is 5.10s x = -184.70 v= -253.79 acceleration =-580.50 time is 5.10s x = -184.70 metres v= -253.79 acceleration =-580.50m/s/s Ft is 613.48 Fd is 0.00 time is 5.25s x = -190.85 v= -261.66 acceleration =-613.48 time is 5.25s x = -190.85 metres v= -261.66 acceleration =-613.48m/s/s Ft is 647.40 Fd is 0.00 time is 5.40s x = -197.00 v= -269.53 acceleration =-647.40 time is 5.40s x = -197.00 metres v= -269.53 acceleration =-647.40m/s/s Ft is 682.25 Fd is 0.00 time is 5.55s x = -203.15 v= -277.40 acceleration =-682.25 time is 5.55s x = -203.15 metres v= -277.40 acceleration =-682.25m/s/s Ft is 718.03 Fd is 0.00 time is 5.70s x = -209.30 v= -285.27 acceleration =-718.03 time is 5.70s x = -209.30 metres v= -285.27 acceleration =-718.03m/s/s Ft is 754.75 Fd is 0.00 time is 5.85s x = -215.45 v= -293.14 acceleration =-754.75 time is 5.85s x = -215.45 metres v= -293.14 acceleration =-754.75m/s/s Ft is 792.40 Fd is 0.00 time is 6.00s x = -221.60 v= -301.01 acceleration =-792.40 time is 6.00s x = -221.60 metres v= -301.01 acceleration =-792.40m/s/s Ft is 830.90 Fd is 0.00 time is 6.15s x = -227.75 v= -308.88 acceleration =-830.90 time is 6.15s x = -227.75 metres v= -308.88 acceleration =-830.90m/s/s Ft is 870.25 Fd is 0.00 time is 6.30s x = -233.90 v= -316.75 acceleration =-870.25 time is 6.30s x = -233.90 metres v= -316.75 acceleration =-870.25m/s/s Ft is 910.48 Fd is 0.00 time is 6.45s x = -240.05 v= -324.62 acceleration =-910.48 time is 6.45s x = -240.05 metres v= -324.62 acceleration =-910.48m/s/s Ft is 951.50 Fd is 0.00 time is 6.60s x = -246.20 v= -332.49 acceleration =-951.50 time is 6.60s x = -246.20 metres v= -332.49 acceleration =-951.50m/s/s Ft is 993.40 Fd is 0.00 time is 6.75s x = -252.35 v= -340.36 acceleration =-993.40 time is 6.75s x = -252.35 metres v= -340.36 acceleration =-993.40m/s/s Ft is 1036.18 Fd is 0.00 time is 6.90s x = -258.50 v= -348.23 acceleration =-1036.18 time is 6.90s x = -258.50 metres v= -348.23 acceleration =-1036.18m/s/s Ft is 1079.75 Fd is 0.00 time is 7.05s x = -264.65 v= -356.10 acceleration =-1079.75 time is 7.05s x = -264.65 metres v= -356.10 acceleration =-1079.75m/s/s Ft is 1124.10 Fd is 0.00 time is 7.20s x = -270.80 v= -363.97 acceleration =-1124.10 time is 7.20s x = -270.80 metres v= -363.97 acceleration =-1124.10m/s/s Ft is 1169.25 Fd is 0.00 time is 7.35s x = -276.95 v= -371.84 acceleration =-1169.25 time is 7.35s x = -276.95 metres v= -371.84 acceleration =-1169.25m/s/s Ft is 1215.18 Fd is 0.00 time is 7.50s x = -283.10 v= -379.71 acceleration =-1215.18 time is 7.50s x = -283.10 metres v= -379.71 acceleration =-1215.18m/s/s Ft is 1261.90 Fd is 0.00 time is 7.65s x = -289.25 v= -387.58 acceleration =-1261.90 time is 7.65s x = -289.25 metres v= -387.58 acceleration =-1261.90m/s/s Ft is 1309.40 Fd is 0.00 time is 7.80s x = -295.40 v= -395.45 acceleration =-1309.40 time is 7.80s x = -295.40 metres v= -395.45 acceleration =-1309.40m/s/s Ft is 1357.68 Fd is 0.00 time is 7.95s x = -301.55 v= -403.32 acceleration =-1357.68 time is 7.95s x = -301.55 metres v= -403.32 acceleration =-1357.68m/s/s Ft is 1406.75 Fd is 0.00 time is 8.10s x = -307.70 v= -411.19 acceleration =-1406.75 time is 8.10s x = -307.70 metres v= -411.19 acceleration =-1406.75m/s/s Ft is 1456.60 Fd is 0.00 time is 8.25s x = -313.85 v= -419.06 acceleration =-1456.60 time is 8.25s x = -313.85 metres v= -419.06 acceleration =-1456.60m/s/s Ft is 1507.25 Fd is 0.00 time is 8.40s x = -320.00 v= -426.93 acceleration =-1507.25 time is 8.40s x = -320.00 metres v= -426.93 acceleration =-1507.25m/s/s Ft is 1558.68 Fd is 0.00 time is 8.55s x = -326.15 v= -434.80 acceleration =-1558.68 time is 8.55s x = -326.15 metres v= -434.80 acceleration =-1558.68m/s/s Ft is 1610.80 Fd is 0.00 time is 8.70s x = -332.30 v= -442.67 acceleration =-1610.80 time is 8.70s x = -332.30 metres v= -442.67 acceleration =-1610.80m/s/s Ft is 1663.60 Fd is 0.00 time is 8.85s x = -338.45 v= -450.54 acceleration =-1663.60 time is 8.85s x = -338.45 metres v= -450.54 acceleration =-1663.60m/s/s Ft is 1717.10 Fd is 0.00 time is 9.00s x = -344.60 v= -458.41 acceleration =-1717.10 time is 9.00s x = -344.60 metres v= -458.41 acceleration =-1717.10m/s/s Ft is 1771.30 Fd is 0.00 time is 9.15s x = -350.75 v= -466.28 acceleration =-1771.30 time is 9.15s x = -350.75 metres v= -466.28 acceleration =-1771.30m/s/s Ft is 1826.25 Fd is 0.00 time is 9.30s x = -356.90 v= -474.15 acceleration =-1826.25 time is 9.30s x = -356.90 metres v= -474.15 acceleration =-1826.25m/s/s Ft is 1881.90 Fd is 0.00 time is 9.45s x = -363.05 v= -482.02 acceleration =-1881.90 time is 9.45s x = -363.05 metres v= -482.02 acceleration =-1881.90m/s/s Ft is 1938.25 Fd is 0.00 time is 9.60s x = -369.20 v= -489.89 acceleration =-1938.25 time is 9.60s x = -369.20 metres v= -489.89 acceleration =-1938.25m/s/s Ft is 1995.30 Fd is 0.00 time is 9.75s x = -375.35 v= -497.76 acceleration =-1995.30 time is 9.75s x = -375.35 metres v= -497.76 acceleration =-1995.30m/s/s Ft is 2053.00 Fd is 0.00 time is 9.90s x = -381.50 v= -505.63 acceleration =-2053.00 time is 9.90s x = -381.50 metres v= -505.63 acceleration =-2053.00m/s/s Ft is 2111.35 Fd is 0.00 time is 10.05s x = -387.65 v= -513.50 acceleration =-2111.35 time is 10.05s x = -387.65 metres v= -513.50 acceleration =-2111.35m/s/s Ft is 2170.35 Fd is 0.00 time is 10.20s x = -393.80 v= -521.37 acceleration =-2170.35 time is 10.20s x = -393.80 metres v= -521.37 acceleration =-2170.35m/s/s Ft is 2230.00 Fd is 0.00 time is 10.35s x = -400.00 v= -529.24 acceleration =-2230.00 time is 10.35s x = -400.00 metres v= -529.24 acceleration =-2230.00m/s/s Ft is 2290.30 Fd is 0.00 time is 10.50s x = -406.15 v= -537.11 acceleration =-2290.30 time is 10.50s x = -406.15 metres v= -537.11 acceleration =-2290.30m/s/s Ft is 2351.25 Fd is 0.00 time is 10.65s x = -412.30 v= -544.98 acceleration =-2351.25 time is 10.65s x = -412.30 metres v= -544.98 acceleration =-2351.25m/s/s Ft is 2412.85 Fd is 0.00 time is 10.80s x = -418.45 v= -552.85 acceleration =-2412.85 time is 10.80s x = -418.45 metres v= -552.85 acceleration =-2412.85m/s/s Ft is 2475.10 Fd is 0.00 time is 10.95s x = -424.60 v= -560.72 acceleration =-2475.10 time is 10.95s x = -424.60 metres v= -560.72 acceleration =-2475.10m/s/s Ft is 2537.90 Fd is 0.00 time is 11.10s x = -430.75 v= -568.59 acceleration =-2537.90 time is 11.10s x = -430.75 metres v= -568.59 acceleration =-2537.90m/s/s Ft is 2601.35 Fd is 0.00 time is 11.25s x = -436.90 v= -576.46 acceleration =-2601.35 time is 11.25s x = -436.90 metres v= -576.46 acceleration =-2601.35m/s/s Ft is 2665.45 Fd is 0.00 time is 11.40s x = -443.05 v= -584.33 acceleration =-2665.45 time is 11.40s x = -443.05 metres v= -584.33 acceleration =-2665.45m/s/s Ft is 2730.20 Fd is 0.00 time is 11.55s x = -449.20 v= -592.20 acceleration =-2730.20 time is 11.55s x = -449.20 metres v= -592.20 acceleration =-2730.20m/s/s Ft is 2795.60 Fd is 0.00 time is 11.70s x = -455.35 v= -600.07 acceleration =-2795.60 time is 11.70s x = -455.35 metres v= -600.07 acceleration =-2795.60m/s/s Ft is 2861.65 Fd is 0.00 time is 11.85s x = -461.50 v= -607.94 acceleration =-2861.65 time is 11.85s x = -461.50 metres v= -607.94 acceleration =-2861.65m/s/s Ft is 2928.35 Fd is 0.00 time is 12.00s x = -467.65 v= -615.81 acceleration =-2928.35 time is 12.00s x = -467.65 metres v= -615.81 acceleration =-2928.35m/s/s Ft is 2995.70 Fd is 0.00 time is 12.15s x = -473.80 v= -623.68 acceleration =-2995.70 time is 12.15s x = -473.80 metres v= -623.68 acceleration =-2995.70m/s/s Ft is 3063.70 Fd is 0.00 time is 12.30s x = -479.95 v= -631.55 acceleration =-3063.70 time is 12.30s x = -479.95 metres v= -631.55 acceleration =-3063.70m/s/s Ft is 3132.35 Fd is 0.00 time is 12.45s x = -486.10 v= -639.42 acceleration =-3132.35 time is 12.45s x = -486.10 metres v= -639.42 acceleration =-3132.35m/s/s Ft is 3201.65 Fd is 0.00 time is 12.60s x = -492.25 v= -647.29 acceleration =-3201.65 time is 12.60s x = -492.25 metres v= -647.29 acceleration =-3201.65m/s/s Ft is 3271.60 Fd is 0.00 time is 12.75s x = -498.40 v= -655.16 acceleration =-3271.60 time is 12.75s x = -498.40 metres v= -655.16 acceleration =-3271.60m/s/s Ft is 3342.20 Fd is 0.00 time is 12.90s x = -504.55 v= -663.03 acceleration =-3342.20 time is 12.90s x = -504.55 metres v= -663.03 acceleration =-3342.20m/s/s Ft is 3413.45 Fd is 0.00 time is 13.05s x = -510.70 v= -670.90 acceleration =-3413.45 time is 13.05s x = -510.70 metres v= -670.90 acceleration =-3413.45m/s/s Ft is 3485.35 Fd is 0.00 time is 13.20s x = -516.85 v= -678.77 acceleration =-3485.35 time is 13.20s x = -516.85 metres v= -678.77 acceleration =-3485.35m/s/s Ft is 3557.90 Fd is 0.00 time is 13.35s x = -523.00 v= -686.64 acceleration =-3557.90 time is 13.35s x = -523.00 metres v= -686.64 acceleration =-3557.90m/s/s Ft is 3631.10 Fd is 0.00 time is 13.50s x = -529.15 v= -694.51 acceleration =-3631.10 time is 13.50s x = -529.15 metres v= -694.51 acceleration =-3631.10m/s/s Ft is 3704.95 Fd is 0.00 time is 13.65s x = -535.30 v= -702.38 acceleration =-3704.95 time is 13.65s x = -535.30 metres v= -702.38 acceleration =-3704.95m/s/s Ft is 3779.45 Fd is 0.00 time is 13.80s x = -541.45 v= -710.25 acceleration =-3779.45 time is 13.80s x = -541.45 metres v= -710.25 acceleration =-3779.45m/s/s Ft is 3854.60 Fd is 0.00 time is 13.95s x = -547.60 v= -718.12 acceleration =-3854.60 time is 13.95s x = -547.60 metres v= -718.12 acceleration =-3854.60m/s/s Ft is 3930.40 Fd is 0.00 time is 14.10s x = -553.75 v= -725.99 acceleration =-3930.40 time is 14.10s x = -553.75 metres v= -725.99 acceleration =-3930.40m/s/s Ft is 4006.85 Fd is 0.00 time is 14.25s x = -559.90 v= -733.86 acceleration =-4006.85 time is 14.25s x = -559.90 metres v= -733.86 acceleration =-4006.85m/s/s Ft is 4083.95 Fd is 0.00 time is 14.40s x = -566.05 v= -741.73 acceleration =-4083.95 time is 14.40s x = -566.05 metres v= -741.73 acceleration =-4083.95m/s/s Ft is 4161.70 Fd is 0.00 time is 14.55s x = -572.20 v= -749.60 acceleration =-4161.70 time is 14.55s x = -572.20 metres v= -749.60 acceleration =-4161.70m/s/s Ft is 4240.10 Fd is 0.00 time is 14.70s x = -578.35 v= -757.47 acceleration =-4240.10 time is 14.70s x = -578.35 metres v= -757.47 acceleration =-4240.10m/s/s Ft is 4319.15 Fd is 0.00 time is 14.85s x = -584.50 v= -765.34 acceleration =-4319.15 time is 14.85s x = -584.50 metres v= -765.34 acceleration =-4319.15m/s/s Ft is 4398.85 Fd is 0.00 time is 15.00s x = -590.65 v= -773.21 acceleration =-4398.85 time is 15.00s x = -590.65 metres v= -773.21 acceleration =-4398.85m/s/s Ft is 4479.20 Fd is 0.00 time is 15.15s x = -596.80 v= -781.08 acceleration =-4479.20 time is 15.15s x = -596.80 metres v= -781.08 acceleration =-4479.20m/s/s Ft is 4560.20 Fd is 0.00 time is 15.30s x = -602.95 v= -788.95 acceleration =-4560.20 time is 15.30s x = -602.95 metres v= -788.95 acceleration =-4560.20m/s/s Ft is 4641.85 Fd is 0.00 time is 15.45s x = -609.10 v= -796.82 acceleration =-4641.85 time is 15.45s x = -609.10 metres v= -796.82 acceleration =-4641.85m/s/s Ft is 4724.15 Fd is 0.00 time is 15.60s x = -615.25 v= -804.69 acceleration =-4724.15 time is 15.60s x = -615.25 metres v= -804.69 acceleration =-4724.15m/s/s Ft is 4807.10 Fd is 0.00 time is 15.75s x = -621.40 v= -812.56 acceleration =-4807.10 time is 15.75s x = -621.40 metres v= -812.56 acceleration =-4807.10m/s/s Ft is 4890.70 Fd is 0.00 time is 15.90s x = -627.55 v= -820.43 acceleration =-4890.70 time is 15.90s x = -627.55 metres v= -820.43 acceleration =-4890.70m/s/s Ft is 4974.95 Fd is 0.00 time is 16.05s x = -633.70 v= -828.30 acceleration =-4974.95 time is 16.05s x = -633.70 metres v= -828.30 acceleration =-4974.95m/s/s Ft is 5059.85 Fd is 0.00 time is 16.20s x = -639.85 v= -836.17 acceleration =-5059.85 time is 16.20s x = -639.85 metres v= -836.17 acceleration =-5059.85m/s/s Ft is 5145.40 Fd is 0.00 time is 16.35s x = -646.00 v= -844.04 acceleration =-5145.40 time is 16.35s x = -646.00 metres v= -844.04 acceleration =-5145.40m/s/s Ft is 5231.60 Fd is 0.00 time is 16.50s x = -652.15 v= -851.91 acceleration =-5231.60 time is 16.50s x = -652.15 metres v= -851.91 acceleration =-5231.60m/s/s Ft is 5318.45 Fd is 0.00 time is 16.65s x = -658.30 v= -859.78 acceleration =-5318.45 time is 16.65s x = -658.30 metres v= -859.78 acceleration =-5318.45m/s/s Ft is 5405.95 Fd is 0.00 time is 16.80s x = -664.45 v= -867.65 acceleration =-5405.95 time is 16.80s x = -664.45 metres v= -867.65 acceleration =-5405.95m/s/s Ft is 5494.10 Fd is 0.00 time is 16.95s x = -670.60 v= -875.52 acceleration =-5494.10 time is 16.95s x = -670.60 metres v= -875.52 acceleration =-5494.10m/s/s Ft is 5582.90 Fd is 0.00 time is 17.10s x = -676.75 v= -883.39 acceleration =-5582.90 time is 17.10s x = -676.75 metres v= -883.39 acceleration =-5582.90m/s/s Ft is 5672.35 Fd is 0.00 time is 17.25s x = -682.90 v= -891.26 acceleration =-5672.35 time is 17.25s x = -682.90 metres v= -891.26 acceleration =-5672.35m/s/s Ft is 5762.45 Fd is 0.00 time is 17.40s x = -689.05 v= -899.13 acceleration =-5762.45 time is 17.40s x = -689.05 metres v= -899.13 acceleration =-5762.45m/s/s Ft is 5853.20 Fd is 0.00 time is 17.55s x = -695.20 v= -907.00 acceleration =-5853.20 time is 17.55s x = -695.20 metres v= -907.00 acceleration =-5853.20m/s/s Ft is 5944.60 Fd is 0.00 time is 17.70s x = -701.35 v= -914.87 acceleration =-5944.60 time is 17.70s x = -701.35 metres v= -914.87 acceleration =-5944.60m/s/s Ft is 6036.65 Fd is 0.00 time is 17.85s x = -707.50 v= -922.74 acceleration =-6036.65 time is 17.85s x = -707.50 metres v= -922.74 acceleration =-6036.65m/s/s Ft is 6129.35 Fd is 0.00 time is 18.00s x = -713.65 v= -930.61 acceleration =-6129.35 time is 18.00s x = -713.65 metres v= -930.61 acceleration =-6129.35m/s/s Ft is 6222.70 Fd is 0.00 time is 18.15s x = -719.80 v= -938.48 acceleration =-6222.70 time is 18.15s x = -719.80 metres v= -938.48 acceleration =-6222.70m/s/s Ft is 6316.70 Fd is 0.00 time is 18.30s x = -725.95 v= -946.35 acceleration =-6316.70 time</pre>
----	--	--	---

5.Part Two

For this particular scenario, the rest of the report will be broken down in to 3 tasks. The first task, is to describe and program the mathematics necessary for a vector class that should include a distance between two points calculation, as well as a point to point distance squared.

The second task of this scenario is to create a plane class, which should have the attribute of a vector, as well as a point. The point will represent a point on the plane, and the vector will represent the plane normal. Then, there should be a method that can be used to calculate the distance of a particular point from the plane.

The final task is a task on creating a part of the program to be used for sphere to sphere collision, which should be used to calculate if two different spheres (or bounding volumes) have collided.

It should be noted that some of the code for this has already been given, and as such, means that the files Vector.cpp, and Vector.h already exist.

5.1 Task 1

Before beginning, it should be noted that past this point, Object Oriented Programming is being used. The reason for this is because it allows the idea of multiple instances of one “object” to exist at once, without having to repeat as much code as would be seen usually within procedural programming. OOP usually consists of a class, which can be used to create objects that have the properties of that class already configured[6].

Before any programming happens, first, the formula for the distance between 2 vectors in a 3D space is as follows:

$$X = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2}$$

In this Equation, the dimensions of each vector are represented by their respective x, y and z values. So, with this knowledge, a distance can be calculated as seen below.

$$\begin{aligned} A &= (2, 7, 5) \\ B &= (5, 0, 9) \end{aligned}$$

$$\sqrt{(2 - 5)^2 + (7 - 0)^2 + (5 - 9)^2} = 8.60 \text{ to } 2 \text{ decimal places.}$$

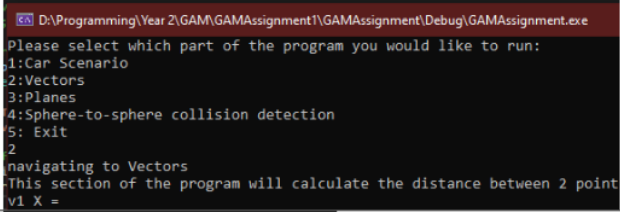
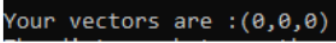
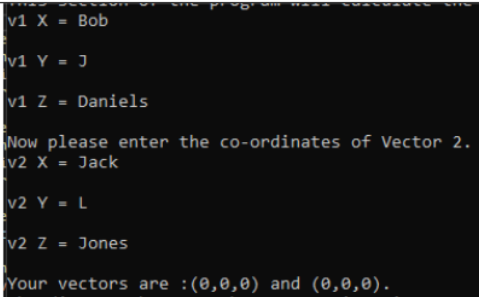
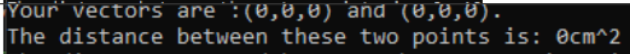
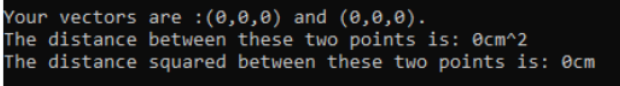
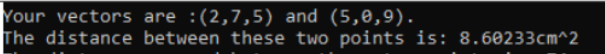
It should also be noted here that the next part of the program wants the distance squared between two points, in which case, the value for distance would simply be calculated again, except without square rooting[7], and as such would be the following formula:

$$\begin{aligned} X &= (A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2 \\ (2 - 5)^2 + (7 - 0)^2 + (5 - 9)^2 &= 74 \end{aligned}$$

To see the code that has been written (as well as the already included Vector.h and Vector.cpp), Please see Appendix D. It should be noted that as a square root is necessary to be performed, math.h has been included for this task, as math.h has a square root method that can be used for this particular task. Additionally, some changes have also been made to the files that were given, as some parts have been changed for the code. It has also been assumed that the measurements for the velocities is in cm, and as such will be output as cm in the program.

5.1.1 Task One Testing

This section of the report is a testing harness for Task 1 of Part Two. This will be used to test the function of the program for Task 2.

Test Number	Description	Expected Result	Actual Result	Pass/Fail
1	Check that the user can navigate to the Vectors part of the program	Program should navigate to the section on vectors.	 <pre> D:\Programming\Year 2\GAM\GAMAssignment1\GAMAssignment1\Debug\GAMAssignment.exe Please select which part of the program you would like to run: 1:Car Scenario 2:Vectors 3:Planes 4:Sphere-to-sphere collision detection 5: Exit 2 navigating to Vectors This section of the program will calculate the distance between 2 points v1 X = </pre>	Pass
2	Check that invalid inputs for vector 1 (such as (Bob, J, Daniels)) are invalid.	Should by default set x, y and z to 0.	 <pre> Your vectors are :(0,0,0) </pre>	Pass
3	Check that invalid inputs for vector 2 (such as (Jack, L, Jones)) are invalid	Should set x, y and z to 0.	 <pre> This section of the program will calculate the distance between 2 points v1 X = Bob v1 Y = J v1 Z = Daniels Now please enter the co-ordinates of Vector 2. v2 X = Jack v2 Y = L v2 Z = Jones Your vectors are :(0,0,0) and (0,0,0). Your vectors are :(0,0,0) and (0,0,0). The distance between these two points is: 0cm^2 </pre>	Pass
4	Check that Distance between 2 points outputs as 0 if all inputs are 0.	Distance between 2 points should be 0cm^2	 <pre> Your vectors are :(0,0,0) and (0,0,0). The distance between these two points is: 0cm^2 </pre>	Pass
5	Check that Squared Distance between 2 points is 0 if all inputs are 0.	Square distance between 2 points should be 0cm.	 <pre> Your vectors are :(0,0,0) and (0,0,0). The distance between these two points is: 0cm^2 The distance squared between these two points is: 0cm </pre>	Pass
6	Check that if vector 1 is (2,7,5) and vector 2 is (5,0,9),	Should output distance as 8.60~cm^2	 <pre> Your vectors are :(2,7,5) and (5,0,9). The distance between these two points is: 8.60233cm^2 </pre>	Pass

	that distance should be $8.60 \sim \text{cm}^2$			
7	Check that if vector 1 is (2,7,5) and vector 2 is (5,0,9), that square distance should be 74cm.	Should output 74cm for distance squared.	<pre> Your vectors are :(2,7,5) and (5,0,9). The distance between these two points is: 8.60233cm^2 The distance squared between these two points is: 74cm </pre>	Pass
8	Check that if the user enters nothing what so ever for both vectors that all values should default to 0.	All values should default to 0.	<pre> navigating to Vectors This section of the program will calculate the distance betwe r 1. v1 X = v1 Y = v1 Z = Now please enter the co-ordinates of Vector 2. v2 X = v2 Y = v2 Z = Your vectors are :(0,0,0) and (0,0,0). The distance between these two points is: 0cm^2 The distance squared between these two points is: 0cm </pre>	Pass

5.2Task 2 – Plane Class

For this part of this report, the task is to create a class that is used for a plane. This class should be able to calculate the plane normal, by normalising a vector, and then using a point (P_0) in order to represent a point that is somewhere on the plane. Once this is done, it will be necessary to calculate the distance that a point may be from the plane.

Firstly, in order to normalise a vector, the magnitude of the vector is needed[8]. In order to calculate a vector's magnitude, the following formula will suffice.

$$||\vec{v}|| = \sqrt{(A_x^2 + A_y^2 + A_z^2)}$$

Then, from here, to normalise the vector, this can be done by dividing each part of the vector by its magnitude with the following equation.

$$\hat{v} = \frac{\vec{v}}{||\vec{v}||}$$

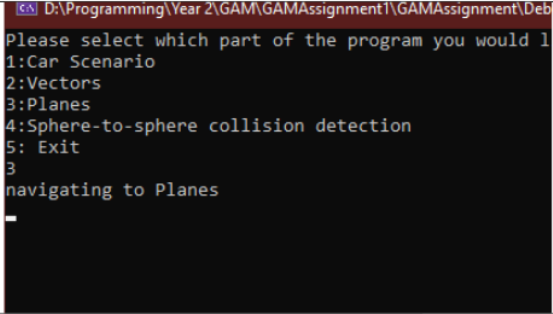
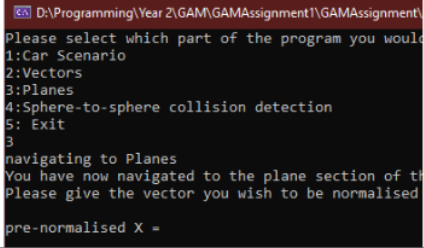
This in turn will produce a normalised vector, which can then be used as the plane normal for this task. The plane calculation is as follows (note that x is a 3 dimensional point on the plane).

$$x \cdot \hat{v} + d = 0$$

The value of d (for distance) is calculated by using:

$$d = -(P_0) \cdot \hat{v}$$

The code can be seen for this program within Appendix

Test Number	Description	Expected Result	Actual Result	Pass/Fail
1	Check that the user can navigate to the planes part of the program	Navigates to the planes area of the program.		Fail
2	Repeat of test 1 after adding in code to actually navigate	Navigates to the planes area of the program.		Pass

3	Test that the user can't enter invalid inputs, like typing bob for the pre-normalised values.	Results should appear as either 0 or NaN.	<pre> Please give the vector you wish to be normalised in order to be on the plane. pre-normalised X = Bob pre-normalised Y = Bob pre-normalized Z = Bob Normalising using a magnitude of 0. Normalised plane is (-nan(ind),-nan(ind),-nan(ind)) Please enter the co-ordinates of P0. </pre>	Pass
4	Test that the user can't enter invalid inputs like jeff for Pzero.	Should output 0 or NaN.	<pre> Please enter the co-ordinates of P0. X = Jeff Y = Jeff Z = Jeff Before becoming negative, pzero is: (0,0,0) Please enter a point on the plane. </pre>	Pass
5	Test that the user can't enter invalid inputs like jack for the point that is on the plane.	Should output 0 or NaN	<pre> Please enter a point on the plane. X = jack Y = jack Z = jack Point is : (0,0,0) </pre>	Pass
6	Check that the user can enter 0 all the way through out the program.	User should be able to enter 0, as 0 is a number.	<pre> Navigating to planes You have now navigated to the plane section of this program Please give the vector you wish to be normalised in order to be on the plane. pre-normalised X = 0 pre-normalised Y = 0 pre-normalized Z = 0 Normalising using a magnitude of 0. Normalised plane is (-nan(ind),-nan(ind),-nan(ind)) Please enter the co-ordinates of P0. X = 0 Y = 0 Z = 0 Before becoming negative, pzero is: (0,0,0) Please enter a point on the plane. X = 0 Y = 0 Z = 0 Point is : (0,0,0) distance is -nan(ind) plane is -nan(ind) </pre>	Pass

7	Check that with actual numbers the calculations output the correct results.	Entering different numbers should display correct equations matching with those already discussed.	<pre> C:\Programing\Year 2\QAM\QAMAssignment1\QAMAssignment1\Debug\QAMAssignment1.exe Please select which part of the program you would like to run: 1:Car Scenario 2:Vectors 3:Planes 4:Sphere-to-sphere collision detection 5: Exit 3 navigating to Planes You have now navigated to the plane section of this program. Please give the vector you wish to be normalised in order to use pre-normalised X = 1 pre-normalised Y = 2 pre-normalised Z = 3 Normalising using a magnitude of 3.74166. Normalised plane is (0.267261,0.534522,0.801784) Please enter the co-ordinates of P0. X = 1 Y = 1 Z = 1 Before becoming negative, pzero is: (1,1,1) Please enter a point on the plane. X = 1 Y = 1 Z = 0 xPoint is : (1,1,0) distance is = -1.60357 plane is = -0.801784 </pre>	Uncertain.
8	Check that the plane equation is equal to zero.	Plane equation should be equal to 0.	<pre> distance is = -1.60357 plane is = -0.801784 </pre>	Fail

5.3 Sphere to sphere collision

Sphere to sphere collision is also required for this report. Sphere to sphere collision is used to detect the collision between 2 different spheres. It is particularly useful for video games as it can be used for bounding volumes.

This can be detected using a sphere to sphere intersection formula [9].

The following formula can be used in order to calculate sphere to sphere collision in a 2 dimensional area.

$$(C_1 - C_2) \cdot (C_1 - C_2) \leq (R_1 + R_2)^2$$

In this equation, C represents the point for a circle, and R represents the radius of a circle.

As the C is made of points, it will be necessary to make use of the vector class that has previously been used.

To see the code for this particular part of the program, please see appendix F.

References

- [1] A. S.-G. Dick Jardine, in *Mathematical Time Capsules Historical Modules for the Mathematics Classroom*, The Mathematical Association of America, 2010, p. 215.
- [2] L. B. James Van Verth, "Euler's Method," in *Essential Mathematics For Games & Interactive Applications A Programmer's Guide*, London, Morgan Kaufmann, 2004, p. 587.
- [3] L. B. James Van Verth, "Euler's Method," in *Essential Mathematics For Games And Interactive Applications*, London, Morgan Kaufmann, 2004, p. 589.
- [4] L. B. James Van Verth, "Verlet Integration," in *Essential Mathematics For Games & Interactive Applications A Programmers Guide*, London, Morgan Kaufmann, 2004, pp. 594-595.
- [5] cplusplus.com, "Input/Output with files - C++ Tutorials," cplusplus.com, [Online]. Available: <http://www.cplusplus.com/doc/tutorial/files/>. [Accessed 29 11 2020].
- [6] W3Schools, "C++ OOP (Object Oriented Programming)," W3Schools, [Online]. Available: https://www.w3schools.com/cpp/cpp_oop.asp. [Accessed 30 11 2020].
- [7] N. Langdon, "How To Calculate Distance Between Two Co-ordinates," Leaf Group Media, 22 7 2019. [Online]. Available: <https://sciencing.com/calculate-distance-between-two-coordinates-6390158.html>. [Accessed 30 11 2020].
- [8] Khan Academy, "Vector Magnitude & normalization (article) | Khan Academy," Khan Academy, [Online]. Available: <https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/vector-magnitude-normalization>. [Accessed 30 11 2020].
- [9] L. B. James Van Verth, "Sphere-Sphere Intersection," in *Essential Mathematics For Games & Interactive Applications A Programmers Guide*, London, Morgan Kaufmann, 2004, p. 535.

Appendix A – Euler Code.

```
92 float Ft = 0.00; //Tractive force of the spring. //this is also the force for the initial Euler formula to start off with!
93 float Fd = 0.00; // Drag force.
94 float x = 0.00; //
95 float v = 0.00;
96 float timestep = 0.15;
97 float time = 0.00;
98 float CoefficientRoll = 0.51;
99 float m = 1.00; //Kilograms.
100 float g = 9.81; // m/s^2. Gravity.
101 float Cd = 0.45; //Drag Coefficient.
102 float p = 1.20; //kg/m^3. Air Density.
103 float Area = 0.10; //m^2. Area in front of the vehicle.
104 float a = 0.00; //acceleration.
105
106 //these two are for Verlet.
107 float vhalfStep = 0.00;
108 float nextA = 0.00;
109
110 //first run through is in Euler.
111 std::cout << std::setprecision(2) << std::fixed;
112
113
114 //This makes a file called EulerResults.txt, then writes to it.
115 ofstream theFile("EulerResults.txt");
116 theFile << std::setprecision(2) << std::fixed;
117 //validation for mass.
118 if (m < 0)
119 {
120     cout << "Negative mass is impossible. Reseting mass to 1.00kg." << endl;
121     m = 1.00;
122 }
123 theFile << "Euler's method on car scenario \n N,      time,      x,      v,      ,Acceleration \n";
124 cout << endl << endl << endl;
125 cout << "Beginning Euler car simulation." << endl;
126
127 for (int i = 0; i < 30; i++)
128 {
129     if (x > 0)
130     {
131         Ft = 0;
132     }
133     else
134     {
135         Ft = 1.5 * (16 - x);
136     }
137     if (v < 0)
138     {
139         Fd = 0;
140     }
141     else
142     {
143         Fd = CoefficientRoll * m * g + (1 / 2) * Cd * p * Area * (v * v);
144     }
145     a = (Ft - Fd) / m;
146     //had to use this if statement as for some reason, x would continue to slightly increase? even though the x should NOT be changing by this point???
147     if (Fd == 0 && Ft == 0)
148     {
149         cout << "The car has stopped." << endl;
150     }
151     else
152     {
153         //Euler formula.
154         v = v + (timestep * a);
155         x = x + (v * timestep);
156     }
157     cout << "Ft is " << Ft << " newtons" << endl;
158     cout << "Fd is " << Fd << endl;
159     cout << "time is " << time << "s" << "x = " << x << " metres " << "v = " << v << " acceleration = " << a << "m/s/s\n";
160     theFile << i << " " << time << "s" << "x << "metres " << v << " " << a << "m/s/s\n";
161     time = time + timestep;
162 }
163
164 //closes Euler Text file.
165 theFile.close();
166
167 cout << "please press any key in order to begin Velocity Verlet Simulation." << endl;
```

Appendix B – Velocity Verlet Integration Code

```

167 cout << "please press any key in order to begin Velocity Verlet Simulation." << endl;
168 //gives the user a pause before sending the user to the velocity verlet simulation.
169 cin.get();
170
171 cout << endl << endl << endl << "Beginning Velocity verlet, with a single step of Euler" << endl;
172 //resetting values for Verlet round.
173 Ft = 0.00; //Tractive force of the spring. //this is also the force for the initial Euler formula to start off with!
174 Fd = 0.00; // Drag force.
175 x = 0.00;
176 v = 0.00;
177 timestep = 0.15;
178 time = 0.00;
179 CoefficientRoll = 0.51;
180 m = 1.00; //Kilograms.
181 g = 9.81; // m/s^2. Gravity.
182 Cd = 0.45; //Drag Coefficient.
183 p = 1.20; //kg/m^3. Air Density.
184 Area = 0.10; //m^2. Area in front of the vehicle.
185 a = 0.00;
186 if (m < 0)
187 {
188     cout << "Negative mass is impossible. Reseting mass to 1.00kg." << endl;
189     m = 1.00;
190 }
191 //needs a single run of euler to get the car moving initially.
192 //Tractive force.
193 Ft = 1.5 * (16 - x);
194 //Drag formula.
195 Fd = CoefficientRoll * m * g + (1 / 2) * Cd * p * Area * (v * v);
196 a = (Ft - Fd) / m;
197 //Euler formula.
198 v = v + (timestep * a);
199 x = x + (v * timestep);
200 //Creates a file called VelocityVerletResults.txt and then writes to it.
201 ofstream velocityVerletFile("VelocityVerletResults.txt");
202
203 velocityVerletFile << std::setprecision(2) << std::fixed;
204
205 velocityVerletFile << "Velocity Verlet method on car scenario\n N, time, x, v, Acceleration\n";
206 velocityVerletFile << "Euler" << " " << time << "s" << x << "metres" << v << " " << a << "m/s/s\n";
207
208 time = time + timestep;
209
210 cout << "Ft is " << Ft << endl;
211 cout << "Fd is" << Fd << endl;
212 //cout << "x = " << x << " v= " << v << " acceleration = " << a << endl;
213
214
215 cout << "Euler done." << endl;
216 //Begins the iteration loop for Velocity Verlet!
217
218 for (int i = 0; i < 30; i++)
219 {
220     if (x > 0)
221     {
222         Ft = 0;
223     }
224     else
225     {
226         Ft = 1.5 * (16 - x);
227     }
228     if (v < 0)
229     {
230         Fd = 0;
231     }
232     else
233     {
234         Fd = CoefficientRoll * m * g + (1 / 2) * Cd * p * Area * (v * v);
235     }
236     //Forces x to stop increasing. This is most likely not the way of doing it, chances are theres some incorrect logic with how x is being calculated.
237     if (Fd == 0 && Ft == 0)
238     {
239         cout << "The car has stopped." << endl;
240     }
241     else
242     {
243         //Acceleration has to be calculated twice for Velocity Verlet. First is the current acceleration, and then just before the end of this iteration, the next acceleration is calculated.
244         a = (Ft - Fd) / m;
245         vHalfStep = v + timestep / 2 * a;
246         x = x + timestep * v;
247         nextA = (Ft - Fd) / m;
248         v = vHalfStep + timestep / 2 * nextA;
249     }
250     cout << "Ft is " << Ft << endl;
251     cout << "Fd is" << Fd << endl;
252     //cout << "time is " << time << " x = " << x << " v= " << v << " acceleration = " << a << "\n";
253     cout << "time is" << time << "s" << "x = " << x << "metres" << "v= " << v << " acceleration = " << a << "m/s/s\n";
254     velocityVerletFile << " " << time << "s" << x << "metres" << v << " " << a << "m/s/s\n";
255     time = time + timestep;
256 }
257 //closes the velocity verlet file.
258 velocityVerletFile.close();
259 //gives a pause for the user before sending them to the main menu.
260 cout << "Please press any key to go back to the main menu." << endl;
261 cin.get();
262 mainmenu();
263

```

Appendix C – Euler and Velocity Verlet Tables.

Euler

n	time	Velocity
1	0	2.85
2	0.15	5.6
3	0.3	8.17
4	0.45	10.46
5	0.6	12.39
6	0.75	13.91
7	0.9	13.16
8	1.05	12.41
9	1.2	11.66
10	1.35	10.91
11	1.5	10.16
12	1.65	9.41
13	1.8	8.66
14	1.95	7.91
15	2.1	7.15
16	2.25	6.4
17	2.4	5.65
18	2.55	4.9
19	2.7	4.15
20	2.85	3.4
21	3	2.65
22	3.15	1.9
23	3.3	1.15
24	3.45	0.4
25	3.6	-0.35
26	3.75	-0.35
27	3.9	-0.35
28	4.05	-0.35
29	4.2	-0.35
30	4.35	-0.35

Velocity Verlet

n	Time	Velocity
(Euler)	0	2.85
1	0.15	5.6
2	0.3	8.26
3	0.45	10.73
4	0.6	12.92
5	0.75	14.74
6	0.9	16.14
7	1.05	15.39
8	1.2	14.64
9	1.35	13.88
10	1.5	13.13
11	1.65	12.38
12	1.8	11.63
13	1.95	10.88
14	2.1	10.13
15	2.25	9.38
16	2.4	8.63
17	2.55	7.88
18	2.7	7.13
19	2.85	6.38
20	3	5.63
21	3.15	4.88
22	3.3	4.13
23	3.45	3.38
24	3.6	2.63
25	3.75	1.88
26	3.9	1.13
27	4.05	0.38
28	4.2	-0.37
29	4.35	-0.37
30	4.5	-0.37

Appendix D

```
void vectorsWork()
{
    float userInpX = 0.00;
    float userInpY = 0.00;
    float userInpZ = 0.00;
    float distance = 0.00;
    float distanceSquare = 0.00;

    //Vector v1;
    Vector v1;

    //Necessary for streaming string inputs to floats.
    string input;
    stringstream stream;

    cout << "This section of the program will calculate the distance between 2 points. Please enter the vector co-ordinates for vector 1." << endl;
    cout << "v1 X = ";
    //streams string inputs to floats.
    //inputs for vector 1.
    getline(cin, input);
    stream.str(input);
    stream >> userInpX;
    stream.clear();
    cout << endl << "v1 Y = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpY;
    stream.clear();
    cout << endl << "v1 Z = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpZ;
    stream.clear();
    Vector vect1(userInpX, userInpY, userInpZ); //Vector 1 instantiated as variables that the user has entered.

    cout << endl << "Now please enter the co-ordinates of Vector 2." << endl;
    cout << "v2 X = ";
    //streams string inputs to floats.
    //inputs for vector 2.
    getline(cin, input);
    stream.str(input);
    stream >> userInpX;
    stream.clear();
    cout << endl << "v2 Y = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpY;
    stream.clear();
    cout << endl << "v2 Z = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpZ;
    stream.clear();
    Vector vect2(userInpX, userInpY, userInpZ); //Vector 2 instantiated as variables that the user has entered.

    cout << endl << "Your vectors are :(" << vect1.getX() << ",(" << vect1.getY() << ",(" << vect1.getZ() << ") and (" << vect2.getX() << ",(" << vect2.getY() << ",(" << vect2.getZ() << ")." << endl;
    distance = vect1.PointToPointDistance(vect2); //calculates distance.
    distanceSquare = vect1.PointToPointDistanceSquare(vect2); //calculates distance squared.
    cout << "The distance between these two points is: " << distance << "cm" << endl; //outputs distance to user.
    cout << "The distance squared between these two points is: " << distanceSquare << "cm" << endl; //outputs distance squared to user.

    cin.get();
}
```

```
Vector.cpp  Source.cpp
Assignment  → Vector

#include "vector.h";
#include <math.h>;
//construct
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::magnitude(Vector& v1)
{
    return sqrt((v1.x * v1.x) + (v1.y * v1.y) + (v1.z * v1.z));
}
Vector Vector::getVector(void)
{
    return Vector(x, y, z);
}
float Vector::getX(void)
{
    return x;
}
float Vector::getY(void)
{
    return y;
}
float Vector::getZ(void)
{
    return z;
}
void Vector::Add(Vector& v1)
{
    x = x + v1.x;
    y = y + v1.y;
    z = z + v1.z;
}
float Vector::PointToPointDistance(Vector& v1)
{
    return sqrt(((x - v1.x) * (x - v1.x)) + ((y - v1.y) * (y - v1.y)) + ((z - v1.z) * (z - v1.z)));
}
float Vector::PointToPointDistanceSquare(Vector& v1)
{
    return ((x - v1.x) * (x - v1.x)) + ((y - v1.y) * (y - v1.y)) + ((z - v1.z) * (z - v1.z));
}
float Vector::dotProd(Vector& v1)
{
    return (x * v1.x + y * v1.y + z * v1.z);
}
```

```
Vector.h  Vector.cpp  Source.cpp
GAMAssignment

1  //****(note x,y,z are not private-should they be?)
2  #ifndef VECTOR_H
3  #define VECTOR_H
4
5  class Vector
6  {
7  public:
8
9      //construct
10     Vector(float x, float y, float z);
11     Vector(void);
12
13
14     void Scale(float s);
15     void Add(Vector& v1);
16     void Add(Vector& v1, Vector& v2);
17     float magnitude(Vector& v1);
18     float PointToPointDistance(Vector& v1);
19     float PointToPointDistanceSquare(Vector& v1);
20     float dotProd(Vector& v1);
21     /*float Normalise(Vector& v1, float mag);*/
22
23     Vector getVector(void);
24     float getX(void);
25     float getY(void);
26     float getZ(void);
27     //private: //these are currently public - should they be private?
28     private:
29         float x;
30         float y;
31         float z;
32 };
33 #endif
```

Appendix E

```
332 void planeMark()
333 {
334     float userInpX = 0.00;
335     float userInpY = 0.00;
336     float userInpZ = 0.00;
337     float normalX = 0.00;
338     float normalY = 0.00;
339     float normalZ = 0.00;
340     float d = 0.00;
341     float Pzero = 0.00;
342     float mag = 0.00;
343     float dotResult = 0.00;
344     float dotnorm = 0.00;
345     float planeCalc = 0.00;
346
347     string input;
348     stringstream stream;
349
350     cout << "You have now navigated to the plane section of this program." << endl;
351     cout << "Please give the vector you wish to be normalised in order to use for the Plane Normal." << endl;
352     cout << endl << "Pre-normalised X = ";
353     getline(cin, input);
354     stream.str(input);
355     stream >> userInpX;
356     stream.clear();
357     cout << endl << "Pre-normalised Y = ";
358     getline(cin, input);
359     stream.str(input);
360     stream >> userInpY;
361     stream.clear();
362     cout << endl << "Pre-normalised Z = ";
363     getline(cin, input);
364     stream.str(input);
365     stream >> userInpZ;
366     stream.clear();
367
368     Vector normalisedAndMag(userInpX, userInpY, userInpZ);
369     mag = sqrt((userInpX * userInpX + userInpY * userInpY + userInpZ * userInpZ));
370     cout << "Normalising using a magnitude of " << mag << "." << endl;
371     normalX = userInpX / mag;
372     normalY = userInpY / mag;
373     normalZ = userInpZ / mag;
374     Vector normalVect(normalX, normalY, normalZ);
375     cout << "Normalised plane is (" << normalX << ", " << normalY << ", " << normalZ << ")" << endl;
376
377     cout << "Please enter the co-ordinates of P0." << endl;
378     cout << endl << "X = ";
379     getline(cin, input);
380     stream.str(input);
381     stream >> userInpX;
382     stream.clear();
383     cout << endl << "Y = ";
384     getline(cin, input);
385     stream.str(input);
386     stream >> userInpY;
387     stream.clear();
388     cout << endl << "Z = ";
389     getline(cin, input);
390     stream.str(input);
391     stream >> userInpZ;
392     stream.clear();
393     Vector PzeroObject(userInpX-(userInpX*2), userInpY-(userInpY*2), userInpZ-(userInpZ*2));
394     cout << "Before becoming negative, pzero is: (" << userInpX << ", " << userInpY << ", " << userInpZ << ")" << endl;
395     cout << "Please enter a point on the plane. " << endl;
396     cout << endl << "X = ";
397     getline(cin, input);
398     stream.str(input);
399     stream >> userInpX;
400     stream.clear();
401     cout << endl << "Y = ";
402     getline(cin, input);
403     stream.str(input);
404     stream >> userInpY;
405     stream.clear();
406     cout << endl << "Z = ";
407     getline(cin, input);
408     stream.str(input);
409     stream >> userInpZ;
410     stream.clear();
411     Vector xPoint(userInpX, userInpY, userInpZ);
412     cout << "xPoint is : (" << userInpX << ", " << userInpY << ", " << userInpZ << ")" << endl;
413
414     dotResult = PzeroObject.dotProd(normalVect);
415     d = dotResult;
416     cout << "Distance is = " << d << endl;
417     dotnorm = xPoint.dotProd(normalVect);
418     planeCalc = dotnorm + d;
419     cout << "plane is = " << planeCalc << endl;
420 }
```

Appendix F

```
void sphereToSphere()
{
    float userInpX = 0.00;
    float userInpY = 0.00;
    float userInpZ = 0.00;
    float dotResult = 0.00;

    float r1 = 0.00;
    float r2 = 0.001;

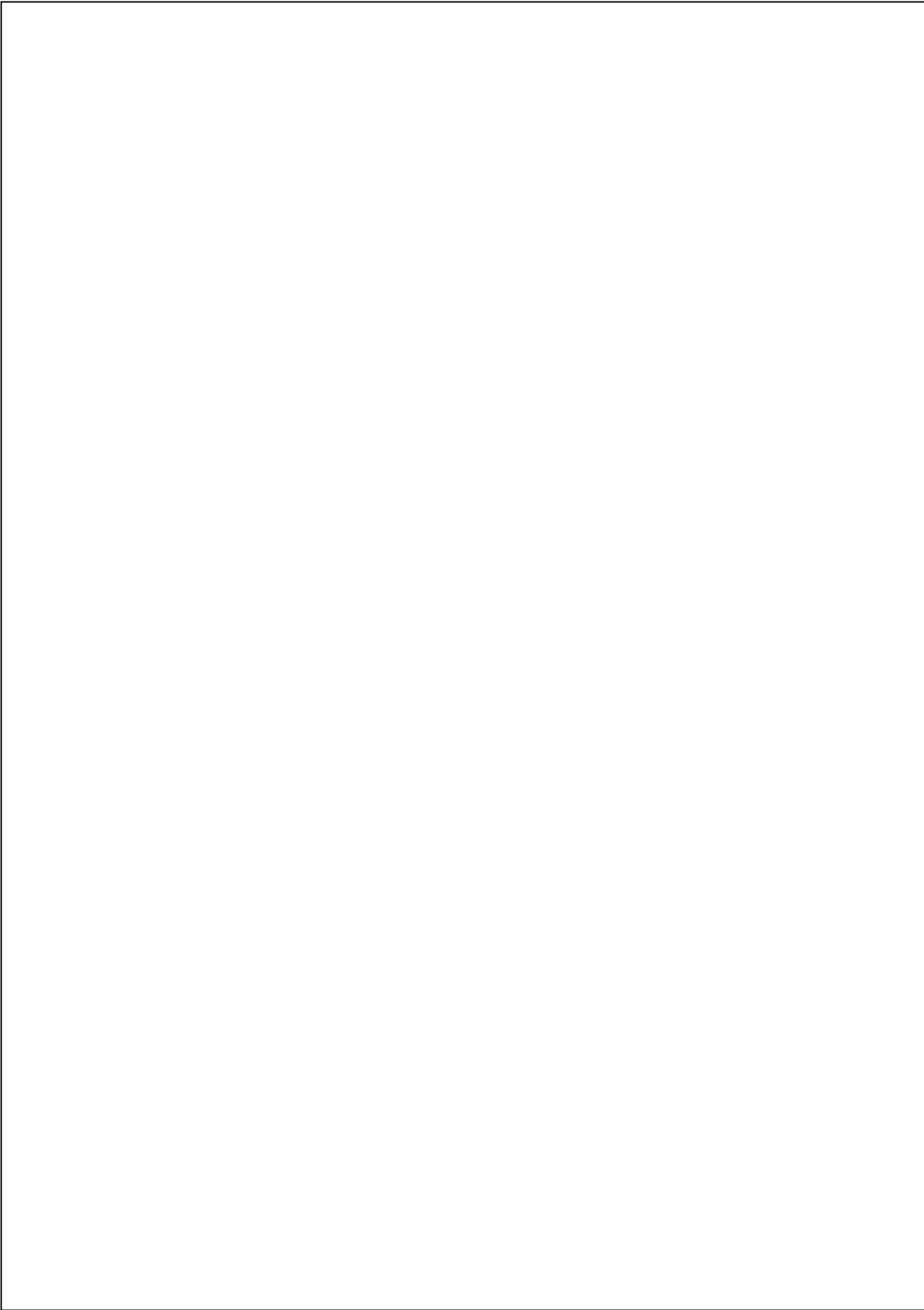
    string input;
    stringstream stream;

    cout << "Please enter the point for C1" << endl;
    cout << "C1 X = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpX;
    stream.clear();
    cout << endl << "v1 Y = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpY;
    stream.clear();
    cout << endl << "v1 Z = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpZ;
    stream.clear();
    Vector c1(userInpX, userInpY, userInpZ);

    cout << "c1 is " << userInpX << ", " << userInpY << ", " << userInpZ << endl;

    cout << "Please enter the point for C2" << endl;
    cout << "C2 X = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpX;
    stream.clear();
    cout << endl << "c2 Y = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpY;
    stream.clear();
    cout << endl << "c2 Z = ";
    getline(cin, input);
    stream.str(input);
    stream >> userInpZ;
    stream.clear();
    Vector c2(userInpX, userInpY, userInpZ);
    cout << "c2 is " << userInpX << ", " << userInpY << ", " << userInpZ << endl;
    cout << endl;
    cout << "please enter the radius for circle 1." << endl;
    getline(cin, input);
    stream.str(input);
    stream >> r1;
    stream.clear();
    cout << "please enter the radius for circle 2." << endl;
    getline(cin, input);
    stream.str(input);
    stream >> r2;
    stream.clear();
    cout << "r1 is " << r1 << endl;
    cout << "r2 is " << r2 << endl;

    dotResult = c1.dotProd(c2);
    if (dotResult > (r1 + r2))
    {
        cout << "Collided!" << endl;
    }
    else
    {
        cout << "Not collided!" << endl;
    }
}
```



Games Architecture Mechanics Part 1a

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

76/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

good

PAGE 4

PAGE 5

PAGE 6

PAGE 7



Comment 2

what about distance etc. Those graphs would be helpful

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

GRADING FORM: SMU GENERIC_GJ

MAY DAVIES

76

RESEARCH-INFORMED LITERATURE



Good use of references here. Key topics identified and researched. Nice to see ref to Euler's original work.

7

KNOWLEDGE AND UNDERSTANDING



A very good understanding demonstrated. Verlet covered, good, although you could have walked through an example calc to demonstrate the method. Plane class results could be better discredibed and results documented.

15

ANALYSIS



Some good work here - detailed discussion of results and epsilon errors, but only Velocity graphed - a missed opportunity here. Code design and implementation reflects the problem. Testing quite thorough. No need to document failed tests if they later pass.

15

PRACTICAL APPLICATION AND DEP.



Again, some good work here. The plane class and sphere collision could be better documented.

36

SKILLS FOR PROFESSIONAL PRAC.



Well structured. Abstract should summarize your report and include brief results/ findings. REfs good.

3

