Creating a physics Library for use in mobile app development.

By Michael Shalaby

Supervised by Keith Mannock

Student ID: mshala02

Msc Computer Science Project Proposal

Abstract:

This proposal will outline the design requirements of to create a Physics Library for mobile game development using open source graphics libraries and rendering libraries. The proposed library will be built using kotlin as the language of choice in order to specialise the physics library for android app development. In order to assess the viability of the physics library, the library will require a lot more than unit testing to see if it works. Therefore the ideal testing ground for this library would be to use it to simulate the daring dam-busters raid "Operation Chastise" carried out by the British during WW2.

Contents

History of physical simulations in video games.

Since the introduction of video games in the 1970s, the first physics library came in the form of collision detection as seen in the game Pong (1972). Since then video game physics have seen a lot more development in recent years. These improvements vary by simulating variable motion in earlier racing games as demonstrated in pole position released 1982 to demonstrate how higher speed turning would lead to a larger turning radius to demonstrating the difference in driving on different surfaces like driving on tarmac then driving on grass or gravel.

Over the following decades, more and more physical properties to video games became more and more mainstream. These would range from more simple applications like detecting collision between two separate objects and the transition from using ray tracing to simulate gun fire to implementing projectile motion to simulate the motion of a projectile like a bullet as travels through the air, calculating its new position at every iteration of time.

Why build a new physics library for mobile apps

While there are many excellent physics libraries available with a few being suitable for android app development including jbullet, a physics engine ported to java. There isn't currently an existing physics library built with Kotlin. While java and kotlin both use the same compiler, the main issue java as a programming language face is the fact that it typically uses a lot more boilerplate code than kotlin.

As kotlin has become considered to be the default language for android development since the release of android studio 3.0 has led to popular IDE's like Jetbrain's IntelliJ to utilise a kotlin to java converter to help developers reduce the amount of code they need to write and create less error prone programs for android in comparison to using Java.

While Java is still being updated today with the latest release of Java 12. It is still considered by many to not be a fully modern programming language whereas kotlin is being viewed more and more as the successor to Java for android development.

Why simulate Operation Chastise:

During my undergraduate studies in 2013, one of my projects was to build a web app to simulate some aspects of the bouncing bomb operation. However, due to having an extremely limited understanding in programming, the amount of physical properties that was simulated was very limited. However, as my understanding in programming has improved, so has my understanding on how to implement a more realistic simulation of a bouncing bomb by using less constants like having the velocity of the bomb along the i direction dynamically be affected by skimming the surface rather than a flat value of velocity reduced every time it skims.

A brief history of operation chastise

During the night time on the 16th and 17th of May, the RAF launched one of the most daring bombing raids to destroy three dams located by the industrial heartland of Germany. The reason why dams where chosen in specific was because it was believed that by destroying the dams, it would flood the surrounding areas around the dam and prevent any electricity being generated by the dams. The idea behind this was that by destroying the dams, the British could cripple Germany's industrial capabilities a lot more effectively than launching conventional bombing raids directly over the city.

What makes operation chastice the ideal testing ground for the physics library

As the dam where protected by anti-torpedo nets. The bombs dropped where designed to bounce across the lakes in a similar manner to 'skipping stones'. Thus, negating the effect of torpedo nets underwater. Therefore, this particular event provides many interesting physical aspects to simulate. This would include:

- The height in which the bomb is dropped. During the original operation, the plane was required to fly 60ft above the surface of the water. Therefore, the user can be given the option to adjust the height in which the bomb is dropped from.
- The rate in which the bomb was spinning. During the original operation, the bouncing bomb was required to have a back spin at 700 rpm to increase the speed in which the bottom of the bomb would hit the water. Therefore the user could be given the option to change the rate in which the bomb is spinning in order to see

The physics Library should be able to do:

- Simulate the linear and Rotational Motion of an Object to simulate the motion of the bouncing bomb
- Simulate Collisions between objects and accurately simulate the result of collision in order to simulate the bouncing bomb colliding with the surface of the water and wall of the dam
- Simulate physical properties of both Soft and rigid body objects to simulate how the bouncing bomb will interact with surfaces of different properties i.e. water, concrete or more outlandish surfaces like grass or gravel.
- Be able to incorporate different shapes for different objects with different properties i.e. would the bomb be as effective if it was an oval cylinder rather than a circular cylinder.
- Be able to simulate multiple objects independent of each other in a realistic setting i.e. there was a lot more than one plane involved and therefore a lot more than one bouncing bomb being launched.

Roadmap:

1. Chapter one will discuss
   a. Start with deriving and programming linear motion
   b. Using the methodology to derive linear motion, apply it to demonstrate how rotational motion is derived
   c. Understand how to implement linear motion into usable code through the use of vectors
      i.e. for 2d motion: $\text{magnitude of velocity} = \sqrt{i^2 + j^2}$ or for 3d motion $\text{magnitude of velocity} = \sqrt{i^2 + j^2 + k^2}$(if applicable due to limitations in my current programming abilities
2. Implement trig equations for the calculation of angles
3. Understand how to implement rotational motion through the use of matrix algebra to simulate the rotation of an object
4. Implement boundary mechanics for objects, this should incorporate properties like:
   a. Rigid body properties
   b. Soft body properties
5. Implement a collision model between objects of different boundary mechanics this should include:
   a. Collision between different object with different physical properties properties

      b.   Collisions at an angle

Understanding objects in motion

As mentioned above, two types of motion will be examined, linear and rotational motion. By understanding linear motion, the ability to grasp circular motion will be much easier.

Linear motion

In the most basic sense, an object in motion could be derived from a graph charting uniform velocity against time as demonstrated below:
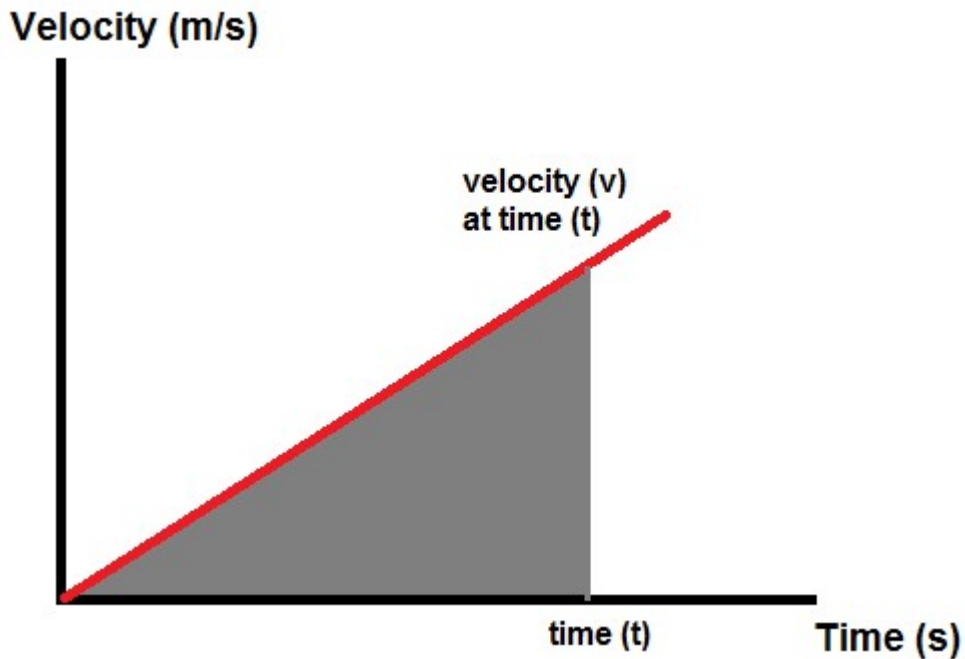
constant velocity vs time
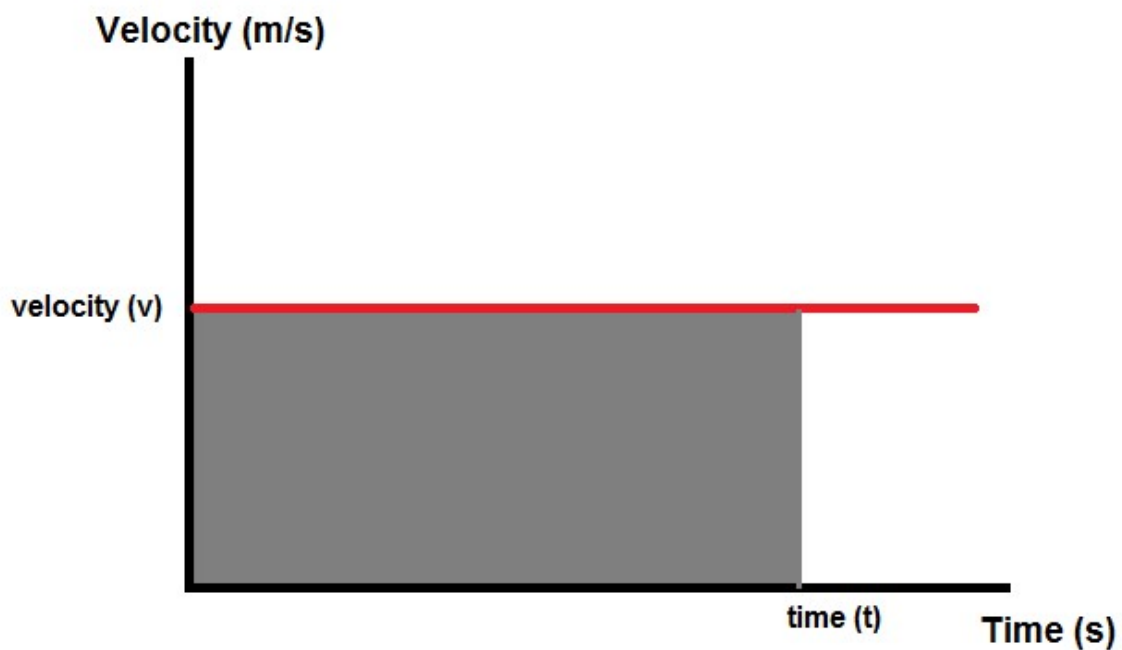
with a constant velocity, the total distance travelled in a certain time period can easily be determined by calculating the area of the rectangle below the velocity line as shown below:

$$s = vt \ [1]$$

Where:
- s = displacement
- v = velocity
- t = time elapsed

Uniform acceleration vs time

Velocity (m/s)

velocity (v)
at time (t)

time (t)

Time (s)

The total displacement(distance in this case) can still be determined by calculating the area of a triangle as shown below:

$$s = \frac{1}{2}v.\frac{m}{s}\ t.s = \frac{1}{2}vt \quad [2]$$

However, if the object is already in motion i.e. v > 0 when t = 0.

Velocity (m/s)

Velocity (v_1)
at time (t)

Velocity (v_0)
at t = 0

Time (s)
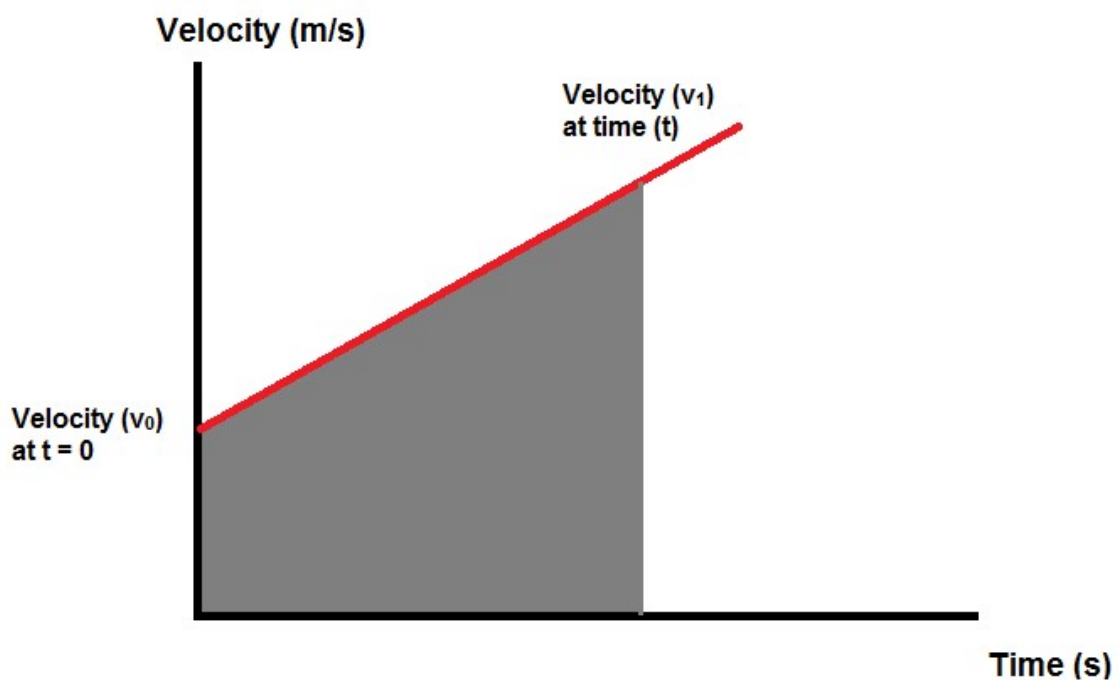
Then by finding the change in velocity during the total time elapsed **t**; we can determine the distance travelled during time elapsed **t** again through the equation below

$$s = \frac{1}{2}(v_1 - v_0)t \quad [3]$$

Where:
- v_0 = the initial velocity of the object
- v_1 = the final velocity of the object

Also as the initial velocity is greater than 0,

Also through the use of the graph above, we can determine the rate of change of velocity as the gradient of the line. This is known as the acceleration of an object and can be calculated as shown bellow:

$$a = \frac{v_0 - v_1}{t} \quad [5]$$

Where:
- a = acceleration of the object

Further equations of motions can also be derived from an acceleration vs time graph as shown below

(draw constant acceleration/time graph here)

As demonstrated earlier with the constant velocity against time diagram. The velocity can also be calculated using a constant acceleration by time graph. Assuming that the initial velocity is 0, this can be expressed as followed:

$$v = a\,t$$

However, if the initial velocity is greater than 0 when time = 0, then simply add the initial velocity to the expression above as follows

$$v = u + at \quad [6]$$

And the same approach can be used when calculating the current velocity with a constantly increasing acceleration by calculating the area of the triangle instead of a rectangle as shown below

(draw constantly increasing acceleration/time graph here)

$$v = \frac{1}{2}a\,t$$

If v > 0 when t = 0

$$s = u + \frac{1}{2}a\,t$$

By substituting the equation for velocity as shown in eq.1 we get the following

$$\frac{s}{t} = u + \frac{1}{2}at$$

And re-arrange to get an equation for the displacement s

$$s = ut + \frac{1}{2}at^2 \text{ [7]}$$

The final equation of motion can be derived by combining equation 6 and 7 as follows:

$$s = ut + \frac{1}{2}at^2$$
$$v = u + at$$

And re-arrange to get:

$$at = v - u$$
$$t = \frac{v - u}{a}$$

Then substitute back into a to get:

$$s = u\left(\frac{v - u}{a}\right) + \frac{1}{2}a\left(\frac{v - u}{a}\right)^2$$

$$s = \frac{2uv - 2u^2 + v^2 + u^2 - 2vu}{2a}$$

$$2as = 2uv - 2u^2 + v^2 + u^2 - 2vu$$

$$2as = v^2 - u^2$$

$$v^2 = u^2 + 2as \text{ [8]}$$

Non uniform displacement, velocity and acceleration

So far, all equations where derived from uniform displacement, velocity and acceleration. However, in realistic scenarios, the plotting of velocity against time would look something like this.

(insert non-uniform velocity / time)

Therefore, more approaches are required to determine the displacement or acceleration. One method would be to divide the graph up into segments

Segmentation and the trapezium rule
One method of determining values for either displacement, velocity or acceleration would be to segment the graph into multiple segments and then calculate the average values from those segments as demonstrated below.

(Demonstrate using the trapezium rule on the non-uniform velocity time graph)

In order to determine the acceleration at a given segment, we simply apply the same calculation used with the uniform velocity/time graph as shown in eq. 4 shown below

$$a = \frac{v_1 - v_0}{t}$$

Therefore, if we want to assess the average acceleration across the velocity/time graph, we could get the mean value of all acceleration segments as shown below

$$a = \frac{\left(\left(\frac{v_1 - v_0}{t_0}\right) + \left(\frac{v_2 - v_1}{t_1}\right) + \left(\frac{v_1 - v_0}{t_2}\right) + \cdots \left(\frac{v_{n+1} - v_n}{t_n}\right) \right)}{n} \quad [7]$$

Where:
n = number of segments

Also using this method of segmentation would also allow us to more accurately determine the area under the velocity curve to get a more accurate value of displacement through the use of the trapezium rule as demonstrated below.

$$s = \frac{1}{2} t \left[(v_0 + v_n) + 2(v_1 + v_2 + \cdots + v_{n-1}) \right] \quad [8]$$

And with a variable acceleration/time graph

(Insert variable acceleration/time here)

As demonstrated with determining the displacement using the trapezium rule, we can also determine the velocity using the trapezium rule as shown below

$$v = \frac{1}{2} t \left[(a_0 + a_n) + 2(a_1 + a_2 + \cdots + a_{n-1}) \right] \quad [9]$$

However, the values obtained through segmentation will never be 100% accurate. However, by introducing more segments to the graph, the accuracy of the value obtained will get much closer to the actual value.

Using differential and integral equations for non-uniform motion

As demonstrated with segmentation, the more segments in the graph, the more accurate the final result. Therefore, through observing each segments. It could be determined that velocity is a measure of the rate of change of displacement against the change in time as shown below

(insert variable velocity/time graph)

And that acceleration is a measure of the rate of change of velocity against the change in time as demonstrated below as well

(insert variable acceleration/time graph)

Therefore, we can determine that this will remain the case with an infinitely small value for t as shown bellow

Therefore allowing for the use of differential equations as shown from the graphs above, velocity is the differential of displacement as acceleration is the differential of velocity.

where:

- v = the velocity(speed with a vector)of the object in motion
- S_0 = the initial displacement (position) of the object
- S_1 = the final displacement of the object (think of $s_0 - s_1$ as the distance travelled)
- t_0 = the initial time of motion
- t_1 = the final time of motion (think of $t_1 - t_0$ as the total time elapsed

Therefore, from the equation above, It can be said that velocity is a measure of the change in displacement (distance travelled) against the change in time. Therefore to calculate the velocity at a given instance; we can take (s_0 - s_1) to be Δs and and ($t_0 - t_1$) to be Δt and can be written as followed:

$$v = \Delta s/\Delta t$$

where:
- Δ = the change (so Δs is the change in displacement or distance travelled)

Now velocity has been expressed with the change of displacement against the change in time, the equation above can be written as a differential equation for the changing **Δ** for **d** where we take the time elapsed as a value infinitely small as shown below:

$$v = \frac{ds}{dt} \text{ [10]}$$

The equation above does not account for the acceleration of the object. However, acceleration is simply the change of velocity over a given period of time. Therefore, the equation for acceleration can be written as the following:

$$a = v_1/t_1 - v_0/t_0$$

Where:

- v_1 = the final value for velocity
- v_0 = the initial value for velocity
- a = rate of change of velocity over time period ($t_0 - t_1$)

Therefore, Like the equation for velocity, the equation for acceleration over a change in velocity and time can be denoted as the following:

$$a = \Delta v/\Delta t$$

And for a given instance where **t** is infinitely small of acceleration:

$$a = dv/dt \text{ [11]}$$

Therefore, it can be said that velocity is a differential equation of displacement and acceleration is a differential equation of velocity. Therefore, acceleration can be viewed as the second differential of displacement as demonstrated below:

$$A = d^2s/dt^2 \text{ [12]}$$

As we have derived equations from displacement to velocity to acceleration. The same can be done in reverse through integrating acceleration to get the value for velocity and integrating velocity to get the value for displacement as shown below:

$$v = \int a \, dt \ [13]$$

And

$$s = \int v \, dt \ [14]$$

This can be applied whether the acceleration of an object is constant or is constantly varying with respect to time for example:
if an acceleration profile takes the value; a = vt² + 2t, then the velocity would be calculated as

$$v = \int vt^2 + 2t \ dt$$

Which would give us $v = \frac{vt^3}{3} + t^2$

Forces, Momentum and collisions

The momentum of an object can be described as the relationship between the object in motion and it's mass. This can be expressed as follows:

$$\text{p} = \text{m}.\,kg \ \times \text{v}\frac{m}{s} = \text{mv}\frac{kgm}{s}$$

Where:

P = momentum of particle in motion = kgms^{-1}
m = mass of the particle = kg
v = velocity of the particle = ms^{-1}

The Force of an object in motion can also be described by the relationship of the acceleration of the object with its mass. This can be expressed as follows:

$$\text{F} = \text{m} \ kg \times \text{A}\frac{m}{s^2} = ma\frac{kgm}{s^2}$$

The energy an object in motion has is commonly refered to as its kinetic energy. The kinetic energy of a particle is the force exerted by the particle in a given instance of distance. This can be equated as

$$\text{KE} = \text{F}\frac{kgm}{s^2} \times \text{s}\, m = Fs$$

Where:

F = force
s = distance travelled

If s can be expressed as

Circular motion

In order to understand Circular Motion, one must be familiar with Circular Geometry.

Circular geometry

Expressions for calculating the geometry of a circle can be derived from the following drawing of a circle and the equation for the area and the circumference of a circle as demonstrated below.
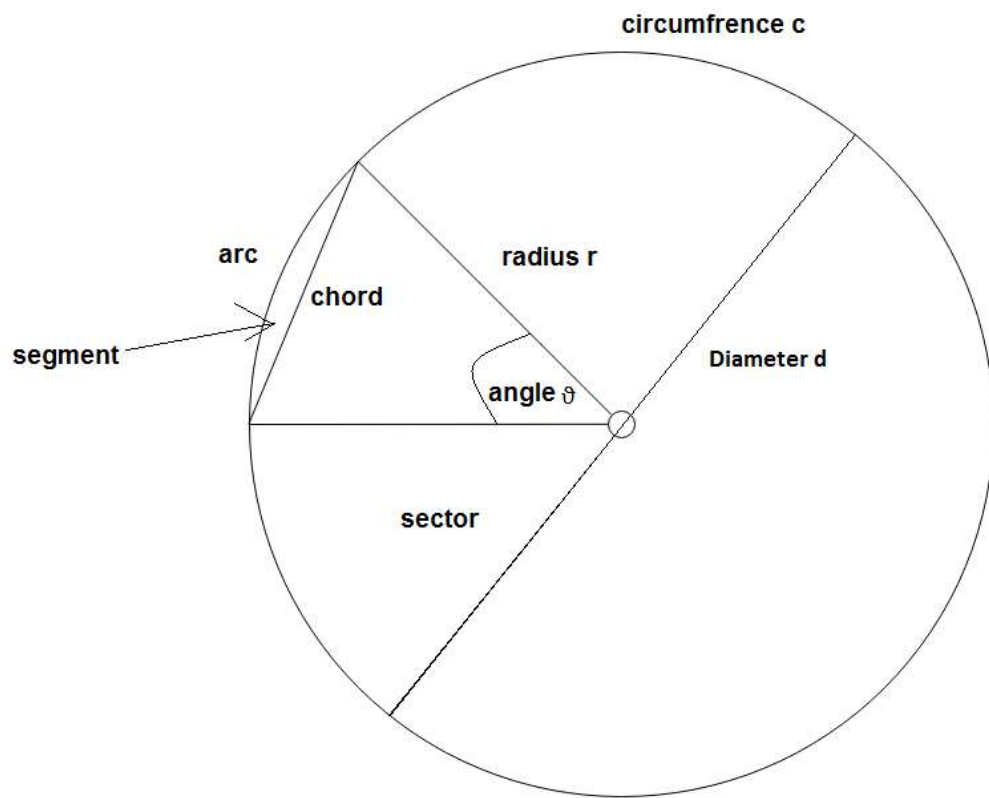
$$Area\ of\ circle\ A = \pi r^2\ or\ A = \pi\frac{d^2}{4}\ [15]$$

$$Circumfrence\ of\ a\ circle\ C = 2\pi r\ or\ C = \pi d\ [16]$$

Where:

r = radius
d = diameter

As the sector of a circle is a portion of a circle taken from the centre, the area of the sector can be calculated as well as the arc of the sector (the circumference of the given sector) Therefore, the ratio of the circle can be taken from the angle the sector encompasses as demonstrated below

$$ratio = \frac{\theta}{360^o} \text{ or } ratio = \frac{\theta}{2\pi \text{ radians}}$$

Where:

θ = angle of the sector in either degrees or radians.

Therefore the area of a sector can be expressed as the following(using radians):

$$Area\ of\ sector = \frac{\theta}{2\pi\ radians}\pi r^2 \text{ or } Area\ of\ sector = \frac{\theta}{2\pi\ radians}\pi\frac{d^2}{4}$$
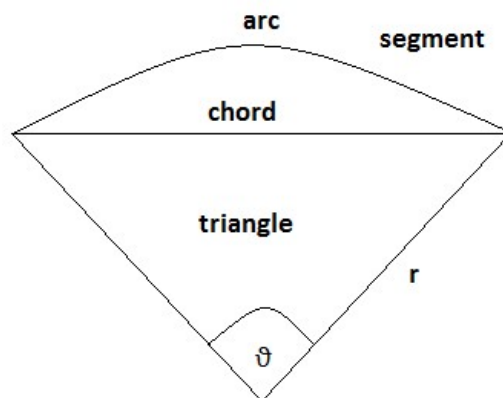
$$Area\ of\ sector = \frac{\theta}{2\ radians}r^2 \text{ or } Area\ of\ sector = \frac{\theta d^2}{8\ radians}$$

And

$$lenght\ of\ arc = \frac{\theta}{2\pi\ radians}\pi d \text{ or } \frac{\theta}{2\pi\ radians}2\pi r$$

$$lenght\ of\ arc = \frac{\theta}{2\ radians}d \text{ or } \frac{\theta}{radians}r$$

From the drawing of the circle, it can be observed that a sector is made up of two elements, the segment and the triangle as shown below

Then by splitting the sector as demonstrated in the diagram below, the value for chord and the height of the triangle can be derived from the following diagram.
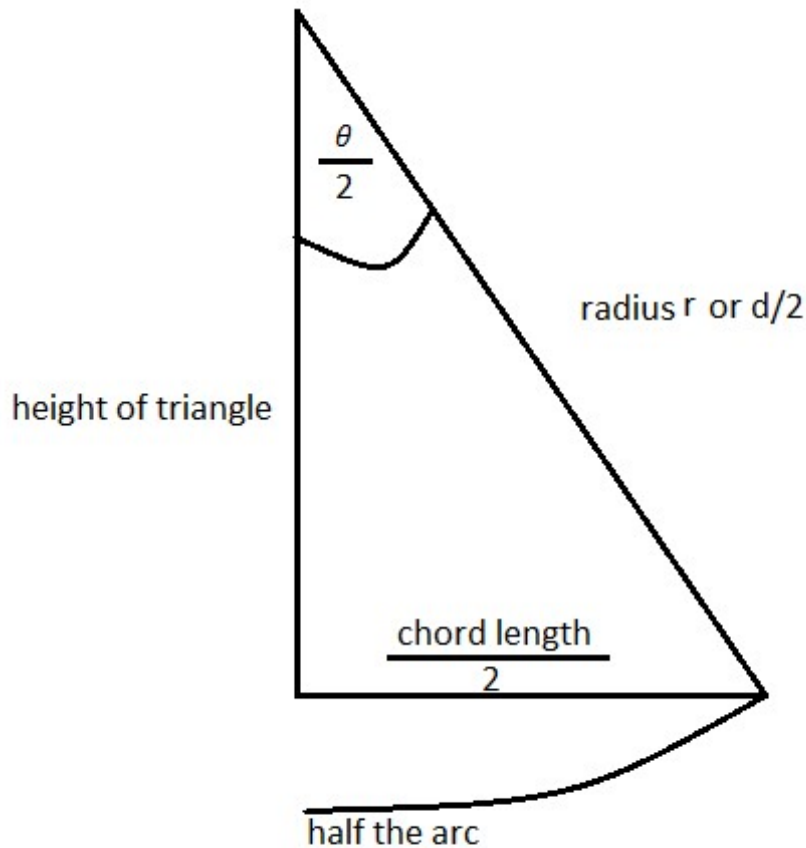
the height of the triangle can be derived and therefore the height and the chord length triangle can be determined using trig equations as demonstrated below:

$$height\ of\ triangle = \frac{d}{2}\cos\left(\frac{\theta}{2}\right)$$

$$chord\ length = 2\left(\frac{d}{2}\sin\left(\frac{\theta}{2}\right)\right)$$

$$chord\ length = d\ sin\left(\frac{\theta}{2}\right)$$
as its half the total length of the chord

Therefore if the area of a triangle is $\frac{1}{2} \times base \times heigh$

Then the area of the triangle can be expressed as shown:

$$area\ of\ triangle = \frac{1}{2} \times chord\ length \times height\ of\ triangle$$

And if the previous calculations for the chord and height are incorporated:

$$area\ of\ triangle = \frac{1}{4} \times d\ sin\left(\frac{\theta}{2}\right) \times d\ cos\left(\frac{\theta}{2}\right)$$

With the area of the triangle calculated, the area of the segment can be calculated with the following expression

$$area\ of\ segment = area\ of\ sector - area\ of\ triangle$$

Therefore expanding on the previous expression for the area of segment with the area of sector and area of triangle, the following expression can also be derived.

$$area\ of\ segment = \frac{\theta}{8\ radians}\ d^2 - \frac{1}{4} \times\ d\ sin\left(\frac{\theta}{2}\right) \times d\ cos\left(\frac{\theta}{2}\right)$$

Translating circular geometry into circular motion

Through observing the motion using the equations as discussed in linear motion. Many equations can be adapted to implement the motion of a particle rotating around a circle e.g:

If the velocity of a particle in motion can be denoted as

$$v = \frac{s}{t}$$

And assuming this particle is travelling in a circle, then the distance can be denoted as the circumference travelled around the circle.
as discussed with the geometry of a circle, the circumference can be expressed as

$$c = 2\pi r = s$$

Therefore, a constant velocity travelling around a circle can be expressed as follows.

$$v = \frac{2\pi r}{t}$$

The acceleration of an object traveling around a circle can be determined by the following equation

$$a = \frac{v^2}{r}$$

By taking the previous equation for the velocity and combining it to the equation for acceleration, a new expression for acceleration can be e derived as:

$$a = \frac{\left(\frac{2\pi r}{t}\right)^2}{r}$$

$$a = \frac{4\pi^2 r}{t^2}$$

The speed in which the circle rotates at its centre is known as its angular velocity. The angular velocity, like linear velocity is a measure of the angle θ travelled during the time t. this can be expressed as follows

$$\omega = \frac{\theta}{t}$$

Where:

θ = angle rotated
t = time period of rotation.
ω = angular velocity

As demonstrated with linear motion, circular motion can also be derived in a very similar manner.

For instance, if the angular velocity ω is not constant and the initial angular velocity is greater than 0, then a similar expression

Implementing real life equation into code

In order to express kinematic equations into code, it must be understood that a typical computer has no notion of how physics work in real life. Therefore every kinematic behaviour must be explicitly coded into the library.

Time, the most important variable

As demonstrated by the equations of motions above. The most important variable for calculating either the location, velocity or even acceleration would be time as the change in time would ultimately result in a change in displacement of an object in motion or a change in velocity for an object accelerating. Therefore, in order for everything to function as it should, time is to be considered the first variable to implement.

Location, using vectors to simulate the location of an object

In order for the computer to register the concept of a location, everything must be coded as a vector. Therefore it could state that the location of an object can be denoted by its vectors i and j (and also k for 3d vectors). For 2d vectors, the computer can take the vector co-ordinates as the location on the page depending on the size of a screen, determining the location of an object much like a navigator would do using the longitude and latitude of a map. In the most simplest sense, this can be expressed within a vector class on kotlin as followed:

```
//simulate change in position (displacement)
i_pos += i_pos_init + i_vel * d_time
j_pos +=j_pos_init +  j_vel * d_time
```

Where:

i_pos = location along the x axis
j_pos = location along the y axis

If the user wishes to incorporate 3D geometry, then an additional k_pos vector with a similar expression written down can be implemented as well.

Linear Motion and direction, using vectors to simulate motion

Unlike real life where speed (motion without direction) is easier to calculate that velocity (motion with direction) in a computer however, it is practically impossible to simulate motion without the use of vectors.
Therefore the total velocity of an object can be determined by using Pythagoras theorem

$$magnitude \; of \; velocity = \sqrt{i^2 + j^2}$$

And the direction could be determined through the use of trig equations

$$angle \; of \; travel = tan^{-1}\left(\frac{j}{i}\right)$$

Therefore, all that is really required for both the i and j vectors would be:

- position vectors for both i and j
- velocity vector for i and j
- acceleration for both i and j

In the most basic form, motion can be implemented on kotlin as follows:

Simulating rotations using equations for circular motion.

When programming for the rotation of an object or the circular motion around a point with a distance r a computer typically cannot rotate objects through the use of circular motion equaitions. Therefore, circular motion must be simulated by using matrix transforms as shown in the diagram below.
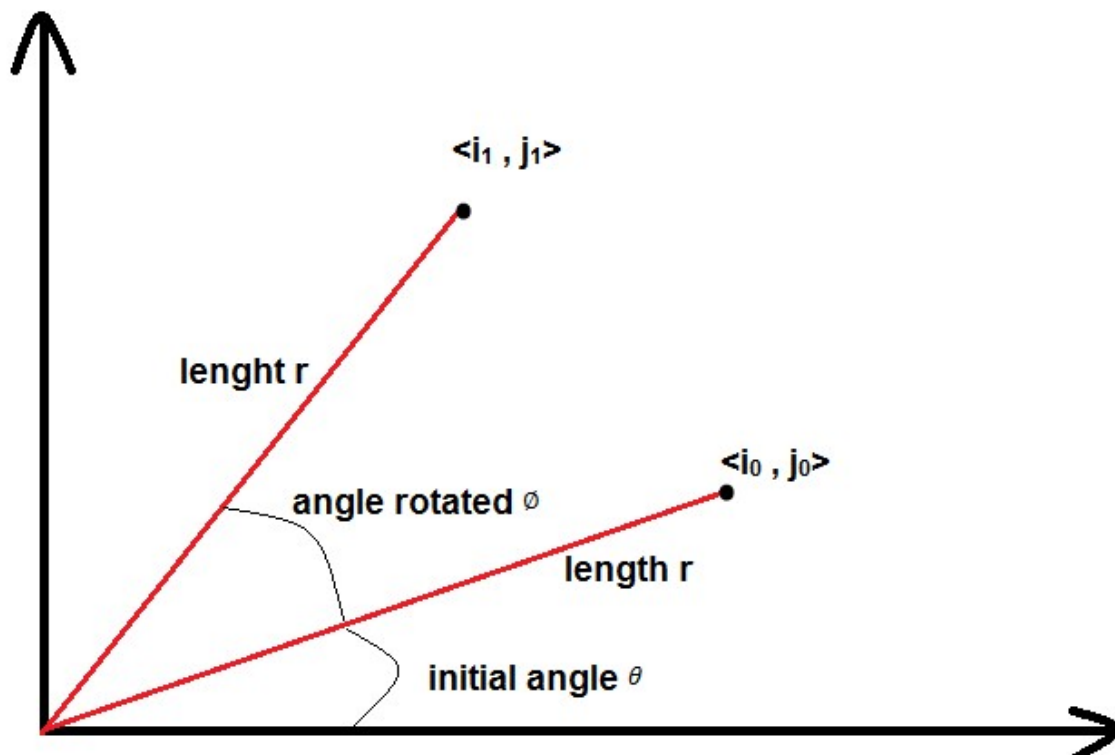
Matrix for 2d transforms:

$$\begin{matrix} i_1 \\ j_1 \end{matrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{matrix} i_0 \\ j_0 \end{matrix}$$

Where:

i_1 = the new i position vector
j_1 = the new j position vector
i_0 = the initial i position vector
j_0 = the initial j position vector

θ is the angle rotated.

From this matrix, two expressions for both the i and j vectors can be expressed from this matrix as follows:

$$i_1 = i_0\cos(\theta) - j_0\sin(\theta)$$
$$j_1 = j_0\sin(\theta) + i_0\cos(\theta)$$

Design patterns
- *discuss design patterns to be used to help implement the physics library and bouncing bomb sim:*
    - *as seen from sdp, the design patterns probably to be used the most will be bridge patterns and adapter patterns*

Production schedule

Incorporate a Gantt chart to outline the plan of action in how this project will be completed.

Todo:

1. Complete circular motions by incorporating these equations and how they are derived (note all calculations are done in radians)

   angle rotated $\Delta\vartheta = \frac{\Delta s}{r}$

   angular rotation $\omega = \frac{\Delta\vartheta}{\Delta t}$

   linear velocity (speed in which the circumference is spinning) $v = r\,\omega$

   and show further derivations much like we donein linear velocity

2. Implement equations of momentum to demonstrate momentum
   e.g.
   momentum $p = m_1 v_1$
   conservation of momentum $p_1 = p_2$
   momentum of multiple particles $= \sum_n m_n\, v_n$

3. Expand greatly on matrix algebra and demonstrate how that is used to simulate the rotation of a given object through matrix multiplication

4. Expand upon design patterns to implement during the project. I.e the adaptor pattern to adapt certain objects that will have similar behaviour like the motion of the bomb adapted to implement the motion of the bomber plane

5. Create a Gantt chart to provide some basis of a structured approach on tackling this project.

6. Expand on the road map to incorporate how each property will be adapted to the bouncing bomb operation.

References:

1. Physics for Game Developers, 2nd Edition by Bryan Bywalec; David M Bourg (viewed from https://learning.oreilly.com/library/view/physics-for-game/9781449361037/ch02.html)

2. A summarised comparison of kotlin vs java
https://www.konstantinfo.com/blog/kotlin-vs-java/

3. History of the bouncing bomb operation
http://www.dambusters.org.uk/the-dam-raids/the-bomb/the-bouncing-bomb/

4. How the bouncing bomb worked
http://www.chm.bris.ac.uk/webprojects2001/moorcraft/The%20Bouncing%20Bomb.htm

5. Circular motion
http://web.mit.edu/8.01t/www/materials/modules/chapter06.pdf

6. Relation of rotation vector and angular velocity
https://physics.stackexchange.com/questions/433102/relation-between-rotation-vector-derivative-and-angular-velocity-when-the-rotati?rq=1

7. Circular motion equaitons
https://www.khanacademy.org/science/ap-physics-1/ap-centripetal-force-and-gravitation/introduction-to-uniform-circular-motion-ap/a/circular-motion-basics-ap1

8. Physical simulations computed
https://www.falstad.com/mathphysics.html

9. https://gamedevelopment.tutsplus.com/tutorials/how-to-create-a-custom-2d-physics-engine-the-basics-and-impulse-resolution--gamedev-6331

10. https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics

11. https://pet.timetocode.org/

ϑ