How it works and doesn’t Ball mover 9000

Intention/overview

This code will move three balls around an ellipse with squared corners.

The balls will start on the top right corner, with a gap for the ball that will land from pg2.

The balls will be moved by the users actions anticlockwise around the squared ellipse.

When they reach the right hand side vertical they will trigger an animation, the user will no longer have control and the viewport and balls will animate to the next screen.

Future: the cards will magnify in turn right to left then left to right in concordance with the balls progress.

Overhanging issues

1.)Converting arc to percentage of the arc

* Managing radian and degrees (the degrees are the angle css wants the radians distance travelled so we should be limiting the distance travelled so cant go less than starting point and more than 3/4s around and then convert to degrees last)(going anticlockwise is something to remember)
* crossing the zero 360 threshold when rotating in a circle

all make the code complex.

Potentially if dealt with by dedicated function would be easier to follow when angle/radian/percentages being used (and when being modifying in the case of the 0 360 threshold issue)

An approach is to turn angles and radians into percentages as soon as possible.

(really need to follow an angle through how often do they get normalised changed and turned to and from percentages)

Really feel im missing something super painfully obvious where my ifs will catch the similarity between -10 and 350 710 -370

It would be nice to work only in radians!

2.)currently use page center for ellipse but this may not be right! Should calculate and check before try and tighten ellipse to match the shape.

3.) fitting the balls ellipse to the shape by linear gradient where border gets thinner and maybe by squewing the ellipse, and changing the point on the image the code operates on (a the moment it is bottom right)

4) lots of variables in functions so if viewport size changes during the page being used the code will adjust – a listener for the change in viewport size may be more efficient???

Others notes: the if to check the range on the magnifier is not designed to go over the zero point,

The code doesn’t set a start point around the circle and an end point

How the user moves the ball

The user by dragging the magnifier anticlockwise (through the same anticlockwise arc that the balls traverse) will move the front ball. The balls behind will follow. Triggering the page transition when the magnifier moves the balls through the trigger threshold.

How the code should achieve this

**1.Initially the code should calculate the first ball position by calculating a position from the right hand corner of the elipse that is 3 ball widths inwards.**

Currently this is not calculated but given as a variable which is an angle in radians. This starting position variable is always added to ball1s angle. (the others are placed behind along the ellipse). The rest of the position is calculated by add the percentage (of the magnifiers arc the magnifier has covered) multiplied by 360 degrees. To give the percentage of a full circle in radians the balls should have covered (+the starting angle in radians). This would be better if it was not the full circle but the arc the balls cover as a percentage.

The code moves the balls based on percentage of the arc the magnifier has cover.

**2. To set the magnifier initial position and by doing so the balls**, a function should run to calculate the angle to put the magnifier at so it is pointing where the first ball will be positioned. This can be achieved by inputting to the ellipsePosition function 0% magnifier arc into it. This achieves 1. And by doing so calculates the end position of the magnifier arc by putting in 100% and the length by taking one from the other. However currently this has issues due to anticlockwise movement, crossing the zero 360 threshold and the magnifier image itself not having the handles start at zero degrees so the initial css rotation currently not being 0.

**3. The user can move the magnifier by dragging or mousescrolling**

The functions and what they do

* mouseDown

uses **getDegrees** to find the angle of the mouse drag of the handle to the handle center

uses **getDegrees** for each part of the mouse move by comparing the first click down degrees to distance dragged get the amount of degrees the magnifier should be dragged. These degrees are added to the initial angle of the magnifier this is zero at start and is saved on mouse up.

(the css does have an initial amount of rotation but the degrees is always added to it so it is not an issue?????) – check this is right

If the degrees calculated are between where i would like the magnifier to be able to move the if run **magnifierAngTopercOfEllipseArcAndRotate**(degrees)

* mouseUp

uses **rememberMagnifierAngle** to set the variable currentDragDeg so that magnifier can be dragged multiple times without reverting to its zero position.

(if rememberMagnifierAngle only used here which i think it is maybe it should just be in mouseUp)

* wheelListner

if the wheel listener value which is stored in a variable called percOfELlipseArc, is moved to a number between 0 and 100 it does the same as mouseDown. This is acheived by calculating the degrees based on the percentage with **percOfEllispeArcToMagnifierAngle.** Like mouseDown ituses the same “if” to see if degrees are in the acceptable range. This range defines the arc the magnifier can be turned through.

The code repeats code from other places. The limit to the turn of the magnifier is here an not in **magnifierAngTopercOfEllipseArcAndRotate** because sometimes this code is used to calculate something but not to place the magnifierfor example the function not used yet that dynamically sets the arc limits for the magnifier, considering page size.

also repeated from mouse up is **rememberMagnifierAngle**

* calcMyObjectPos

This doesn’t move the object it calculates the position it would be in if it was at a set percentage of the ellipse arc. (This may be dodgy but) it takes the object that is being moved and passes it back, so just a callback, it does nothing with it, its so you can put the percent in and then put it in the movefunction and the movefunction will be given all it needs by the calcMyObjPos. CalcmyObjPos doesn’t place because loopThroughArcOfTheEllipseFindNearestSuitableLocation uses it to work out if balls would be too close if they are we want to loop try a new percentage.

Calc my pos currently does not translate the percentage of the magnifier arc to the ellipse arc it directly uses it as a proportion of a full rotation. However we do not want a full rotation – so though it works its not beautiful.

The code makes a percentage of a 360 and adds the startAngle (this allows balls to start not at zero but at the top right)

The ‘if’ angles are hard set not variables. This is fine in that its a circle but it makes it harder to read, because its anticlockwise ellipse travel and the 360 to 0 jump causes difficulty this code is vital to be understood as i think the angles ive used arnt necessaryily right though they work, and if they were variables we could reuse them if a function to dynamically set start positions of the magnifier, its arc, and the first ball position.

else if(ang>5\*Math.PI/2){ this ‘else if’ is a mess because it returns a location now which is to be fair locked, but before it just triggered the transistion, it now returns a position so it can be used with dynamically calculating start positions (this may not be an issue im not sure)

the function returns an object that has the object it was given unaltered packed with the top and left position to put it

(This function maybe slow as recalculates all its variables every time if could put its variable maybe in a listener that listens for a change in viewport size this may be more efficient)

* moveMyObject actually moves the marbles once their position is calculated and provided to it
* checkIfSpaceToMoveIfSoMove this takes two objects and a separation looks at their locations works out the distance between. It returns false if they are too close, it places the second if its distance is okay.
  + The object it receives are from calcMyPos so they haven’t been placed yet
  + for ball one it just receives same object twice with zero separation because the first ball just gets placed, then it receives the ones after.

(in future it might be nice to have an array going on for placing balls but as only 3 and not likely to change on the cusp of unnecessary though would be nicer, maybe when its time to polish)

The reason it return true is because it is run in the loop as an if, if the if return true it means the check has checked and placed the marble and so the increment the number of the marble it needs to work on next loop

* looppercOfEllipseArcTillFindNearestAcceptableLocation

feel the loop isn’t very tight

i is taken from the percentage around the ellipse every loop because the later balls will be further behind, i goes up to the start position as a percentage, because the start position should be calculated such that there is room for two balls behind it

(– this isn’t the best as two balls behind it here, isn’t the same as two balls behind at different part of the ellipse)

The first marble is placed because it is set as the first and second marble with a zero separation so it will just be placed first cycle the others the separation is 100 which is the width of the marbles

(i think these loop for a while a smaller increment would mean tighter placement a larger one quicker placement, maybe when marbleToMove number is increase should inc the i value a bit extra as will know the first few values bound to be too close, )

* rotationMarbleBasedOnDelta this uses marbles actual location and where it will be moved to finds the straight line not the arc distance between and rotates it accordingly. It uses the circumference of the marble so it turns the amount it would of in real life (sort of were not using arc length)
* get rotation degrees – no my function just works out the css rotation
* get\_degrees work out the angle of a location from the magnifier center

(could change it to take the center positions so it can be used both for ellipse and for magnifier, currently use page center for ellipse)

* rememberMagnifierAngle just set currentMagDragDeg and get rotation – this is within moving the magnifier, maybe it should be on mouse up and every mouse wheel click – i think – the whole function is just setting one variable with another function so maybe it is redundant
* magnifierAngTopercofEllipseArcAmdRotate – take the magnifierAng

this does modify a bit the magnifier and to make percArcLength also take it from 100 so this is where it goes anticlockwise

this then goes into the loop function

THIS IS WHERE PERC AND ANTICLOCKWISE happens

Though the percentage then gets turned into degrees again in the ellipseCalculate so

magnifierAng is made a percentage, then reversed by being taken from 100 then turned into a proportion of 360, can all the be missed and just feed the angle,

im not sure what the different will be of have a percentage of the arc of the magnifier which is based on a center higher than the arc percentages center.

* percOfEllipseArcToMagnifierAngle

this reverse engineer the magnifier angle so the mouse scroll can change the magnifier ang

this normalises the angle as well should it

* page3To4Transition this can be ignored for now
* dynamicallySetStartPosVariables – this will set the values of where magnifier and first ball should start based on the shape which is centered vertically and horizontally, it works by using the 0 and 100% x y positions of the ellipse arc.
* It require shape center to be the same as screen center
* It works because extremes of the percentage specific coordinates

Should be easy to solve but not yet done

– Also which part of an arc to measure e.g getting ten to twenty degrees can be done by travelling ten or 350 degrees ... so you get the difference if its less than 180 your going the short way around so take that difference from 360 to find the length of the long way as an angle.