Programming Proverbs

- 17. ‘‘Never assume the computer assumes anything.’’

- Henry F. Ledgard, ‘‘Programming Proverbs: Principles of Good Programming with Numerous Examples to Improve Programming Style and Proficiency’’, (Hayden Computer Programming Series), Hayden Book Company, 1st edition, ISBN-13: 978-0810455221, December 1975.

Collision prediction between circles

- de-tour
  - recall we can predict the time a point travels a distance by:
- $$s = ut + \frac{1}{2} at^2$$
-
```
s = distance
u = initial velocity
a = acceleration
t = time
```

De-tour

- if we were to drop a small ball bearing from 1 metre, how long would it take to hit the ground?

- initial velocity `(u = 0)`

- acceleration approx `(a = 10)` ms^2

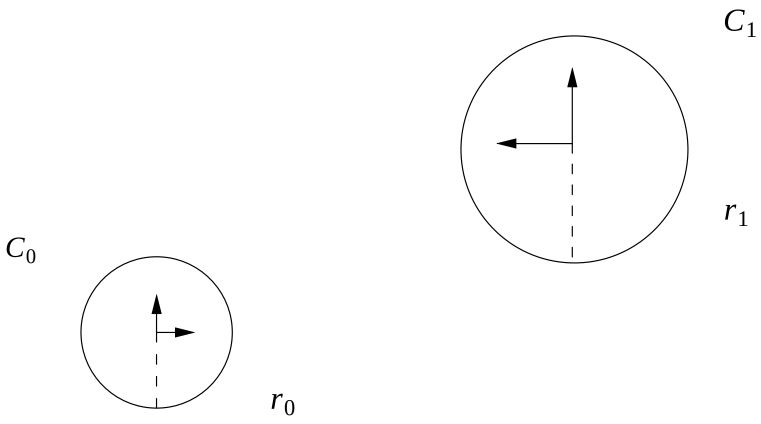
De-tour

- thus
- $1 = \frac{1}{2} at^2$
- $1 = \frac{1}{2} 10t^2$
- $1 = 5t^2$
- $\frac{1}{5} = t^2$
- $t = \pm\sqrt{\frac{1}{5}}$

De-tour

- t = 0.45
- t = -0.45
- we also note that for more complicated quadratic equations we can use the formula:
- $t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- for any equation matching: $at^2 + bt + c = 0$

Collision prediction between circles

- 

- each circle has a radius, position, velocity and acceleration
  - radius is a scalar, all others are vectors

Collision prediction between circles

- remember that when these circles collide the distance between the two circle mid points will be $r_0 + r_1$
- we know generally that the distance between the circles can be calculated as: $\sqrt{(c_{0x} - c_{1x})^2 + (c_{0y} - c_{1y})^2}$
- so we need to find the time when: $r_0 + r_1 = \sqrt{(c_{0x} - c_{1x})^2 + (c_{0y} - c_{1y})^2}$

Collision prediction between circles

- using the formula for initial position, velocity and acceleration:

  $s = s_0 + ut + \frac{at^2}{2}$ it is also known that the position circle c_0 at time, t, is: $\left[c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}, c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2} \right]$

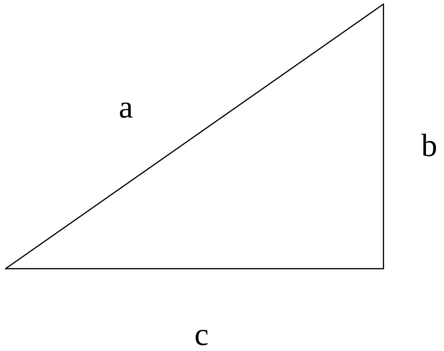
- correspondingly the position circle c_1 at time, t, is:

  $\left[c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}, c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2} \right]$

Collision prediction between circles

- recall Pythagorean theorem can be used to find the length of the hypotenuse from the length of the base and height of a right angled triangle

Collision prediction between circles

- 

- $a^2 = b^2 + c^2$

- $a = \sqrt{b^2 + c^2}$

Collision prediction between circles

- a is the radius of circle 0 and circle 1
  - $a = r_0 + r_1$
- b is the difference of the y-axis position of the circles
- c is the difference of the x-axis position of the circles

Collision prediction between circles

- therefore we need to find the time at which the distance between both circles is $r_0 + r_1$ which is:

- $$r_0 + r_1 = \sqrt{\left[\left(c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}\right) - \left(c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}\right)\right]^2 + \left[\left(c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2}\right) - \left(c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2}\right)\right]^2}$$

- which can be easily rearranged as:

- $$0 = \left[\left(c_{0x} + v_{0x}t + \frac{a_{0x}t^2}{2}\right) - \left(c_{1x} + v_{1x}t + \frac{a_{1x}t^2}{2}\right)\right]^2 + \left[\left(c_{0y} + v_{0y}t + \frac{a_{0y}t^2}{2}\right) - \left(c_{1y} + v_{1y}t + \frac{a_{1y}t^2}{2}\right)\right]^2 - (r_0 + r_1)^2$$

Collision prediction between circles

- we need to multiply the equation and separate out the values: A, B, C, D, E where:

- $At^4 + Bt^3 + Ct^2 + Dt + E = 0$

- then we solve for, t and the smallest value of t which is greater than zero is the next collision time

- see `pge/twoDsim.mod` in the function `findCollisionCircles`

wxmaxima

- fortunately, we have a tool called `wxmaxima` (similar to Mathematica) which will manipulate the algebra without mistake :-)

wxmaxima

```
ri + rj == sqrt(abs(xin-xjn)^2 + abs(yin-yjn)^2)     for values of t

ri + rj == sqrt(((xi + vxi * t + aix * t^2 / 2.0) - (xj + vxj * t + ajx * t^2 / 2.0))^2 +
                ((yi + vyi * t + aiy * t^2 / 2.0) - (yj + vyj * t + ajy * t^2 / 2.0))^2)

let:

a = xi
b = xj
c = vxi
d = vxj
e = aix
f = ajx
g = yi
h = yj
k = vyi
l = vyj
m = aiy
n = ajy
o = ri
p = rj
t = t
```

wxmaxima

```
o  + p  == sqrt(((a  + c   * t + e   * t^2 / 2.0) - (b  + d   * t +   f * t^2 / 2.0))^2 +
                ((g  + k   * t + m   * t^2 / 2.0) - (h  + l   * t +   n * t^2 / 2.0))^2)

o  + p  == sqrt(((a  + c   * t + e   * t^2 / 2.0) - (b  + d   * t +   f * t^2 / 2.0))^2 +
                ((g  + k   * t + m   * t^2 / 2.0) - (h  + l   * t +   n * t^2 / 2.0))^2)

0       == ((a  + c   * t + e   * t^2 / 2.0) - (b  + d   * t +   f * t^2 / 2.0))^2 +
           ((g  + k   * t + m   * t^2 / 2.0) - (h  + l   * t +   n * t^2 / 2.0))^2 -
           (o  + p)^2

now using wxmaxima
expand ; factor ; ratsimp
```

wxmaxima

```
p+o    ==  (sqrt((n^2-2*m*n+m^2+f^2-2*e*f+e^2)*t^4+
                ((4*l-4*k)*n+(4*k-4*l)*m+(4*d-4*c)*f+(4*c-4*d)*e)*t^3+
                ((4*h-4*g)*n+(4*g-4*h)*m+4*l^2-8*k*l+4*k^2+(4*b-4*a)*f+
                 (4*a-4*b)*e+4*d^2-8*c*d+4*c^2)*t^2+
                ((8*h-8*g)*l+(8*g-8*h)*k+(8*b-8*a)*d+(8*a-8*b)*c)*t+
                  4*h^2-8*g*h+4*g^2+4*b^2-8*a*b+4*a^2))/2

2*(p+o) ==  (sqrt((n^2-2*m*n+m^2+f^2-2*e*f+e^2)*t^4+
                ((4*l-4*k)*n+(4*k-4*l)*m+(4*d-4*c)*f+(4*c-4*d)*e)*t^3+
                ((4*h-4*g)*n+(4*g-4*h)*m+4*l^2-8*k*l+4*k^2+(4*b-4*a)*f+
                 (4*a-4*b)*e+4*d^2-8*c*d+4*c^2)*t^2+
                ((8*h-8*g)*l+(8*g-8*h)*k+(8*b-8*a)*d+(8*a-8*b)*c)*t+
                 4*h^2-8*g*h+4*g^2+4*b^2-8*a*b+4*a^2))
```

wxmaxima

```
(2*(p+o))^2 == ((n^2-2*m*n+m^2+f^2-2*e*f+e^2)*t^4+
                 ((4*l-4*k)*n+(4*k-4*l)*m+(4*d-4*c)*f+(4*c-4*d)*e)*t^3+
                 ((4*h-4*g)*n+(4*g-4*h)*m+4*l^2-8*k*l+4*k^2+(4*b-4*a)*f+
                  (4*a-4*b)*e+4*d^2-8*c*d+4*c^2)*t^2+
                 ((8*h-8*g)*l+(8*g-8*h)*k+(8*b-8*a)*d+(8*a-8*b)*c)*t+
                   4*h^2-8*g*h+4*g^2+4*b^2-8*a*b+4*a^2))

0            ==  (n^2-2*m*n+m^2+f^2-2*e*f+e^2)*t^4+
                 ((4*l-4*k)*n+(4*k-4*l)*m+(4*d-4*c)*f+(4*c-4*d)*e)*t^3+
                 ((4*h-4*g)*n+(4*g-4*h)*m+4*l^2-8*k*l+4*k^2+(4*b-4*a)*f+
                  (4*a-4*b)*e+4*d^2-8*c*d+4*c^2)*t^2+
                 ((8*h-8*g)*l+(8*g-8*h)*k+(8*b-8*a)*d+(8*a-8*b)*c)*t+
                 4*h^2-8*g*h+4*g^2+4*b^2-8*a*b+4*a^2)-
                 ((2*(p+o))^2)
```

wxmaxima

- ```
  solve polynomial:

  A := sqr(n)-2.0*m*n+sqr(m)+sqr(f)-2.0*e*f+sqr(e) ;
  B := (4.0*l-4.0*k)*n+(4.0*k-4.0*l)*m+(4.0*d-4.0*c)*f+(4.0*c-4.0*d)*e ;
  C := (4.0*h-4.0*g)*n+(4.0*g-4.0*h)*m+4.0*sqr(l)-8.0*k*l+4.0*sqr(k)+
       (4.0*b-4.0*a)*f+(4.0*a-4.0*b)*e+4.0*sqr(d)-8.0*c*d+4.0*sqr(c) ;
  D := (8.0*h-8.0*g)*l+(8.0*g-8.0*h)*k+(8.0*b-8.0*a)*d+(8.0*a-8.0*b)*c ;
  E := 4.0*sqr(h)-8.0*g*h+4.0*sqr(g)+4.0*sqr(b)-8.0*a*b+4.0*sqr(a)-sqr(2.0*(p+o)) ;
  ```

- fortunately, we can cut and paste the output from `wxmaxima` into our code and carefully convert it into program code and it works!
  ```

Conclusion

- we have seen that we can predict when two circles will collide in the future
  - using x and y components based on $s = ut + a \frac{t^2}{2}$
- in the next lecture we will extend this to look at calculating the time a circle hits a polygon