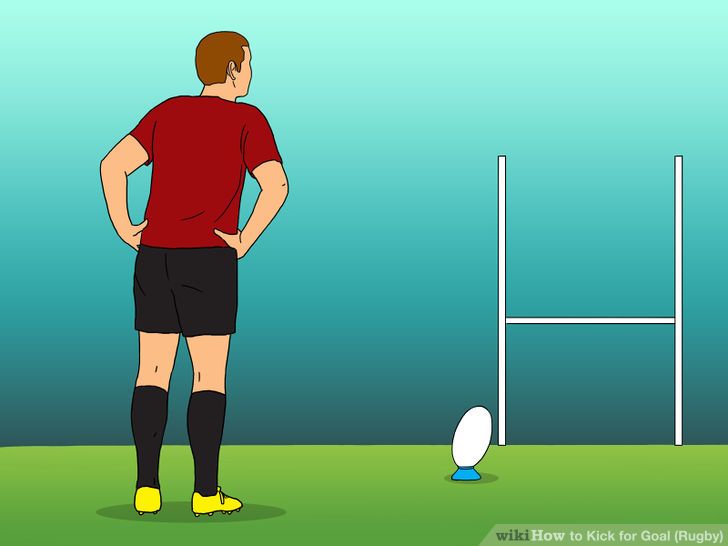
# Related image**Assignment 2 - A SHOT on Goal (PS3)**

**The deadline for this assignment is midnight Monday 19th March, 2018 via Bb.**

**angle?**

**?**

In the game of rugby a touchdown is converted by booting the ball over the 3m high goal cross bar from some point on the pitch, but orthogonal to the try lin where the touchdown occurred (see diagram of a rugby pitch, below).

**speed?**

**?**

The supplied program calculates what speed and at what vertical angle you would have to kick the ball to get it over the cross bar from a set distance (which is input by the user). It does this by cycling through a series of angles and speeds, and figuring out using standard trajectory equations whether the ball would make it over the bar (ignoring factors such as air resistance, ball spin, air density, and so on) until the first working combination of angle and speed is found.

Using the kicking angle and speed thus found, a series of (x,y) coordinates of the ball's trajectory path are calculated and saved in an array. It is assumed that the ball is kicked on target and goes through the uprights.

A 'display' function then plots these on a rugby-stylised graph for inspection (see screen layout later), along with some performance and other statistics.

Units are (m) metres, (s) seconds, (deg) degrees, all float values.

For more information, read the C++ source (key parts can be found at the end of this document).

This article (also available on BB) provides more information on the background to the 'science' of booting a rugby ball,

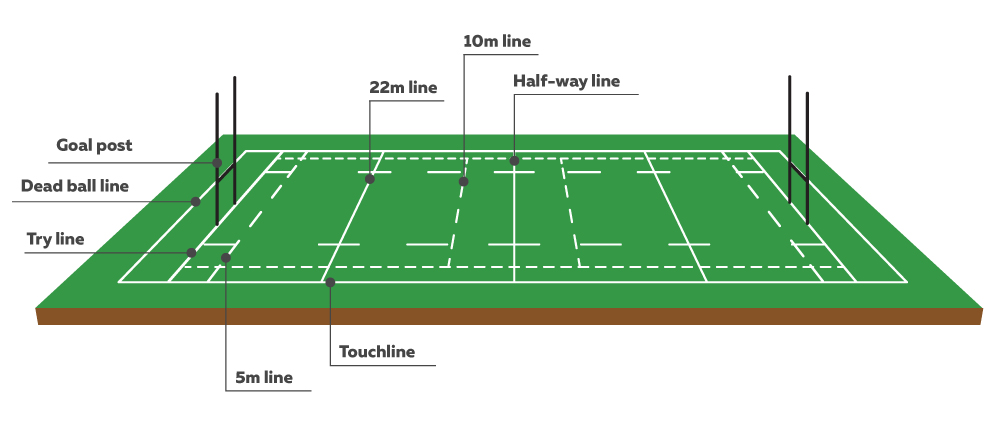
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3918559/>

but all we're interested in is this trajectory equation, embedded in the code:

**height of ball at distance 'x' =**

- where 'g' is gravity (= 9.81 ms-2) and 'x' is the chosen distance of the ball on its 'tee' from the goal posts. Note the angle must be in radians, not degrees, and this conversion is performed in the code too. In C++ this equation looks like this:

**height = ((-g\*x\*x) / (2.0F\*cos(AngleRads)\*cos(AngleRads)\*(nextSpeed\*nextSpeed))) + (x\*tan(AngleRads));**

This equation is used with a minimum speed (currently 5m/s) with a low angle (15º) to begin with, and compares the calculated height of the ball at the goal post distance with the height of the cross bar (3 metres plus a margin of error to be sure it goes over!). If it looks like it'll go over the bar then that's the solution and the speed and angle used are recorded.

***X touch***

***down!***

If it doesn't reach that high the speed is increased by 0.5m/s and another calculation attempt made, up to the maximum speed of 32m/s (note that research suggests that the optimum kick speed is around 26 ±1.7 m/s).

If no solution is found at that angle, the angle is increased by 0.5º, and the whole cycle gone through again. The maximum angle is currently 45º as it's pretty hard to kick at steeper angles than this with sufficient force, and the minimum distance of 5m to the goal posts doesn't require such steep 'punting' angles (and would probably see you charged down before you boot it!).

With a solution found a second function then calculates a series of (x,y) coordinates at 0.5m intervals from the 'tee' position to a maximum of 50.0m (thus crossing the goal post position), with each vertical 'y' coordinate being obtained using the same equation as above. This set of coordinates (dubbed the 'flight path') are then plotted on a crude console-based (text) graphic, thus showing the flight path and other information. See overleaf for an example.

For convenience, the program has been written so that with minimal effort it can run on both the desktop PC ('x86') and the PS3. There are two ZIP files on Bb with ready-made projects for both platforms (under **Assessment , PS3 Assignment**). The only practical differences are the method of timing the code and the lack of easy file I/O on the PS3. Download both ZIP files to somewhere convenient, unpack them, and run the x86 version (initially as it is a lot quicker than the reset cycle of the PS3). **Use 12m as the distance** (although feel free to experiment too). The build mode should be **'Release'**.

*Note: false colour – it’s actually quite boring black and white!*

Height (m) above pitch

in 0.25m increments SHOT ON GOAL CALCULATOR (Adrian Oram)

(Vert. exag.~= x5) ~~~~~~~~~~~~~~~~~~~~~~~

8.5 | **Time to solution (us) 151**

| **Time generating (us) 58**

8.0 | **Kick angle (deg) 20**  . . . . . . . .

| **Kick speed (m/s) 30.5**

7.5 | **10000 Iterations**

|

7.0 | . . . . . . . . . . .

| |

6.5 | |

| |

6.0 | . . . | . . . . . . . .

| |

5.5 | | OOOOOOOOOOO

| | OOOOOOOOO OOOOOOOOO

5.0 | . . . | . . OOOOO . . . OOOOO . .

| | OOOO OOOO

4.5 | | OOO OOO

| | OOO OOO

4.0 | . . . | OOO . . . . . .OOO .

| | OO OO

3.5 | |OO OO

| OO| OOO

3.0 | . . OO | . . . . . . . .OO

| OO + O

2.5 | OO |

| OO |

2.0 | . . O . | . . . . . . . .

| OO |

1.5 | OO |

| O |

1.0 | **\_Q** OO . . | . . . . . . . .

| **| |\\_o** O |

0.5 | **o |\_\_\_\_#** OO |

| **/**  O |

0.0 | **/** BOOT!! |

+-**##**----^+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++

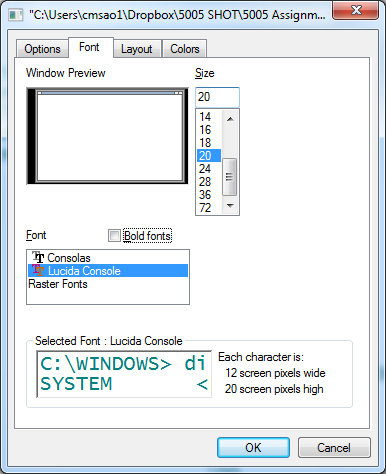
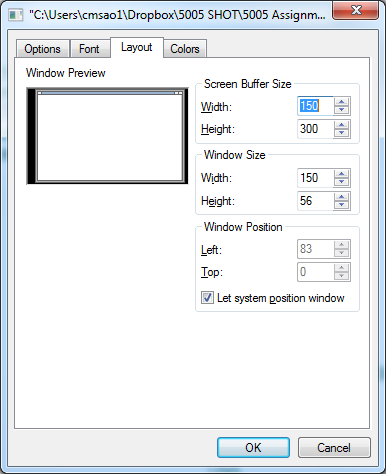
0 5 10 15 20 25 30 35 40 45 50

+ = cross bar

Distance (m) to Fulchester RC's goal in 0.5m increments

Done...

**Note**: you may need to alter the PC console display - right click in the top bar of the console window, and alter the properties to adjust the way it displays, I'm using **Font** **size 20, Lucida Console**…and changed the **Layout** **Width** to **150** characters…

When running on the x86 the performance data (highlighted above) should show something like:

**Time to solution (us) 9.12323** *finding the initial solution for angle and speed (shown below).*

**Time generating (us) 3.70631** *calculating the (x,y) coords of the flight path.*

**Kick angle (deg) 20**

**Kick speed (m/s) 30.5**

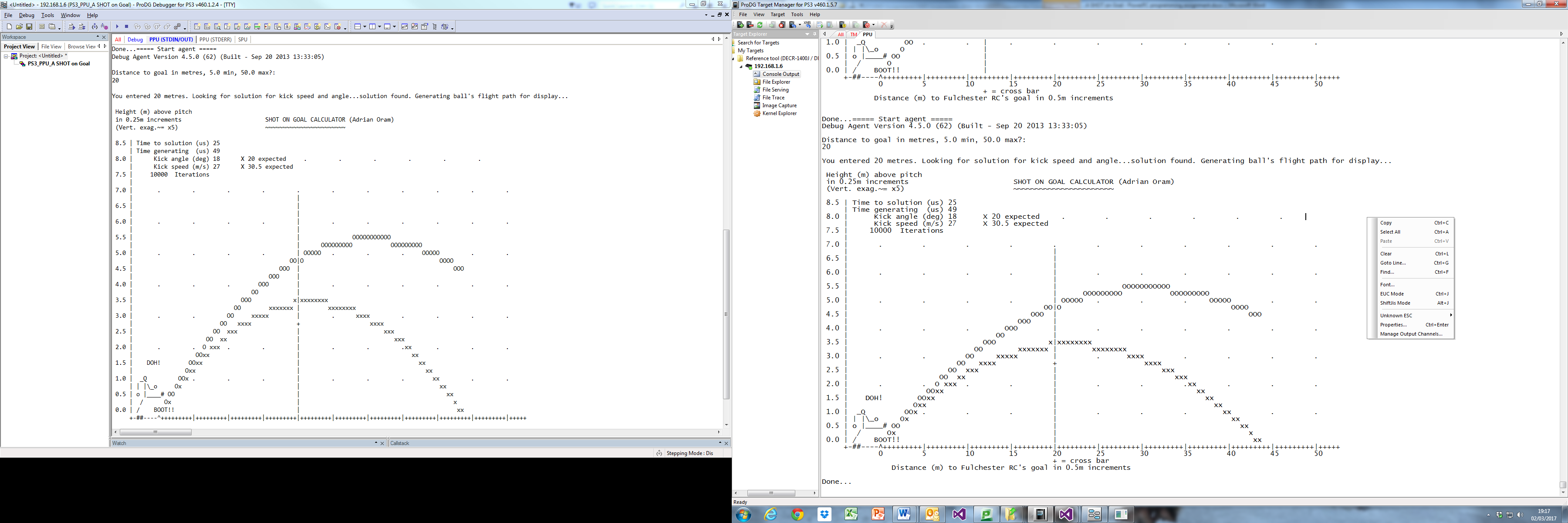
**10000 Iterations**

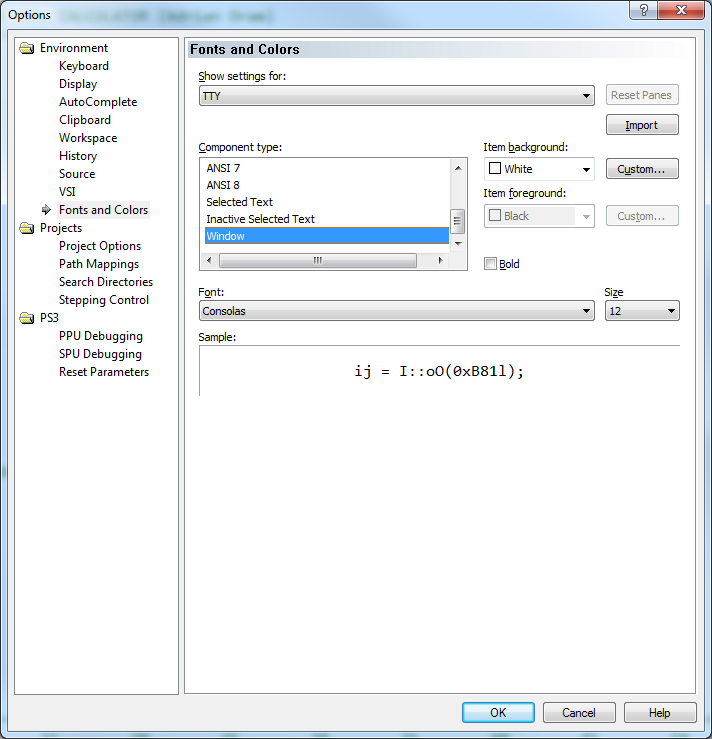
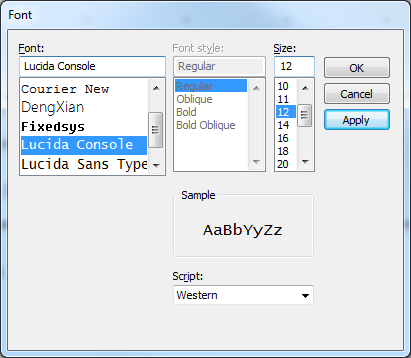
The code repeats the calculations 10,000 times (adjustable) and records the quickest time seen; hopefully this will find the actual elapsed times for these operations.

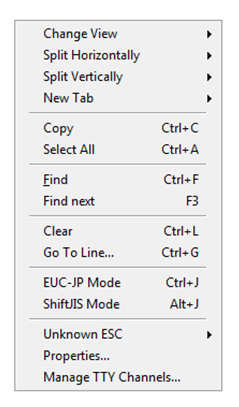
Run the equivalent program on the PS3 (open a second Visual Studio instance) - with some adjustment to the console display you should see an identical graph but with very different timings (as show on the example graph above).

To adjust the PS3 console use these dialogue boxes from the **Tools**, **Options** menu in the PS3 debugger environment:

In the **debugger**: Or, in the **Target Manager** right-click then choose **Font**:





Note that these PS3 console displays can be left to continuously scroll (leaving you with an audit trail of runs - this can be cut and pasted into a document if you wish to keep this data) or easily cleared by right-clicking and choosing **Clear**:

As mentioned the two programs (x86 vs PS3) are 99% identical. You can currently freely cut and paste the whole source from one platform to the other (one Visual Studio project to the other) with the only proviso that you alter the **\_PS3** flag at the top of the code as commented so that it will build:

//#define **\_PS3** // Build for PS3 system otherwise x86 if commented out.

//#define \_trace // comment out to remove trace output used for testing

//#define \_longTrace // comment out to avoid lengthy trace of trial height calculations

Using the x86 version may make some early efforts a bit easier due the speed of building and running, but eventually you will adopt the PS3 version for the assignment.



**Note**: experiment with the other flags to see what they do (look for the affected code in the source). Be aware that the **\_longTrace** option could have you staring at the screen for a long time watching numbers roll by! If you decide to try it, it's a good idea to reduce the const int repeats = 10000; in the code to 10 or less first…

The other **\_trace** allows certain extra information to appear on the screen and on the x86 create some data in an external data file (**FlighPathData.txt**). Take a look.

**Preliminary tasks to be done first!**



**Task 1**

This assignment should be **done in pairs** so please buddy up. If you wish to work alone ('solo') a reduced workload will be asked for (see page 5). Give yourself a team name in any case.

Alter these lines at the top of the code (in both versions too) as appropriate, insert both names in the 'yourName' string, please:

#define yourName "Adrian Oram" // Please change these as appropriate!

#define yourTeamName "Fulchester RC"



**Task 2**

This assignment is about optimising and speeding up the key calculations by employing PowerPC assembly (and other tricks you know) in the PS3 version. In order to have comparable data, some datum figures need to be established, so there is a second version of the program (in both projects) called:

**A 12m SHOT on Goal - USE FOR ASSIGNMENT.cpp**

which, as the title would suggest, has been tailored somewhat to a **12.0m distance**. Swap the current general version of the code for this specific one in your projects. Running this with a distance of 12.0 metres should not appear much different, although you may notice a couple of '**OK**'s after the 'Kick speed' and 'Kick angle' messages in the graph display, and a 'speech bubble' above the kicker.

Using the times and data obtained for this distance you are asked to speed up the "**Time to solution**" and the "**Time generating**" whilst maintaining the correct result, initially in C++ but mainly by using PowerPC assembly. On the PS3 the correct result for a distance of **12.0 m** is a speed of **30.5 m/s** and an angle of **20º**. The times on my PS3 came out to:

**Time to solution (us) 151**

**Time generating (us) 58**

and this is our starting benchmark.

Now run the program and enter **20m** for the distance and note what appears. For each "incorrect" coordinate not on the 12m solution flight path an '**x**' is produced and the correct one placed. This can be used as a quick visual check that the coordinate calculations arising from your improved code are working correctly - you should never see 'x's! Any incorrect values for angle and speed are also highlighted - and the kicker isn't happy!...

Distance to goal in metres, 5.0 min, 50.0 max?:

20

You entered 20 metres. Looking for solution for kick speed and angle...solution found. Generating ball's flight path for display...

Height (m) above pitch

in 0.25m increments SHOT ON GOAL CALCULATOR (Adrian Oram)

(Vert. exag.~= x5) ~~~~~~~~~~~~~~~~~~~~~~~

8.5 | Time to solution (us) 25

| Time generating (us) 49

8.0 | Kick angle (deg) **18** **X 20 expected** . . . . . .

| Kick speed (m/s) **27** **X 30.5 expected**

7.5 | 10000 Iterations

|

7.0 | . . . . . . . . . . .

| |

6.5 | |

| |

6.0 | . . . . | . . . . . .

| |

5.5 | | OOOOOOOOOOO

| | OOOOOOOOO OOOOOOOOO

5.0 | . . . . | OOOOO . . . OOOOO . .

| OO|O OOOO

4.5 | OOO | OOO

| OOO |

4.0 | . . . OOO | . . . . . .

| OO |

3.5 | OOO x|xxxxxxxx

| OO xxxxxxx | xxxxxxxx

3.0 | . . OO xxxxx | . xxxx . . . .

| OO xxxx + xxxx

2.5 | OO xxx | xxx

| OO xx | xxx

2.0 | . . O xxx . . | . . .xx . . .

| OOxx | xx

1.5 | **DOH!** OOxx | xx

| Oxx | xx

1.0 | **\_Q** OOx . . . | . . . xx . .

| **| |\\_o** Ox | xx

0.5 | **o |\_\_\_\_#** OO | xx

| **/** Ox | x

0.0 | **/** BOOT!! | xx

+-**##**----^+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++++++|+++++

0 5 10 15 20 25 30 35 40 45 50

+ = cross bar

Distance (m) to Fulchester RC's goal in 0.5m increments

Done...

**Note**: when testing with the 12m distance the goal posts will be in their correct position, here they're at the requested 20m point.

In this version a further change has been made: the range of angles to be tested for has been reduced from 15º - 45º to just **18º - 23º** (a range of 5º) as shown below in the code:

// Kick metrics

//\*\*\*SHOT const float minAngle(15.0F); // (deg) pretty low!

//\*\*\*SHOT const float maxAngle(45.0F); // (deg) pretty steep.

//\*\*\*SHOT Limit range of angles to help speed things up, and help elsewhere?

const float minAngle(**18.0F**); // (deg) \*\*\*SHOT We know the answer for kick distance 12.0m is 20 degrees.

const float maxAngle(**23.0F**); // (deg)

*(Notice how the decoration "***\*\*\*SHOT***" is being used in comments to indicate differences between this version and the original code)*

**Note**: the number of angles involved in the initial calculations has been reduced from 61 to **just 11** due to this change…

This makes the calculations a little quicker but we know the answer lies in that range so it will still work. However, it can now be demonstrated that such a limitation precludes finding a solution for certain distances - try running it with any distance between 5m and 9m and observe the result.



**Note**: Despite the modified program clearly assuming a known value for angle and speed, you should consider the program to be **general purpose**, so quick solutions which generate a speed of 30.5 m/s and an angle of 20º directly are not allowed, for obvious reasons! Your code should allow correct solutions to be found (or not) for any distance specified - a table of some other distances and their solutions is given on the last page - you should check yours matches up!

**Another 'obvious' solution** (similar to the 'cheating' version of the Xmas Tree program that stored all of the trees as ready-made strings rather than generating them) is to store each solution 'flight path' for each possible distance in a large array. There are 91 possible distances and each requires 104 float 'y' values per flight path, so this needs 91\*104\*4 bytes = 37kB, not massive but enough to be of concern with a potentially limited memory budget. **As you can imagine, such solutions are disbarred here!**

**Task 3**

Now create a new C++ source in the project and name it as follows (this is in preparation for the final submission):

*(pairs)* **your\_surname1\_your\_surname2\_SHOT\_on\_Goal\_2017.cpp**

or

*(solo)* **your\_surname\_SHOT\_on\_Goal\_2017.cpp**

**Select All** everything in the **USE FOR ASSIGNMENT** source and paste it into this new source file. Use this to develop your versions; keep the original intact in case you need to refer back to it for any reason. Remember to **Exclude** the original from the project.



**Task 4**

Optionally, to avoid entering 12m from this point onwards you can edit the code that prompts for the value and uncomment the assignment to **distanceToGoal** at the top of 'main':

int main(void)

{

bool foundCombo(false);

getDistanceToKick(&distanceToGoal); // comment this out if required

//distanceToGoal = 12.0F; //\*\*\*SHOT use this rather than entering it each run!

**Please ensure this is re-instated in your final submission as other distances will be tested.**



**What you are asked to do …**

**(for those in pairs)**

Optimise the functions: **findSHOTonGoalSpeedAndAngle (&kickSpeed, &kickAngle, distanceToGoal);**

and: **generateFlightPath (kickSpeed, kickAngle);**

in order to speed up their respective execution times as much as possible, using the 12m distance as your benchmark of performance. **You are expected to develop the core of the functions (the 'maths') using PowerPC assembly for the PS3**, but utilise other optimising methods as you see fit too. No code/functionality should be moved outside the timed zones.

**(for those going solo)**

As above but only optimise the function: **generateFlightPath (kickSpeed, kickAngle);**

**Marking Scheme (this assignment is worth 35% of the module assessment)**

1. Speed up marks for **Time to solution** (roughly pro rata) **10% (for pairs)**

2. Speed up marks for **Time generating** (roughly pro rata) **10% (for pairs)**

*or* 2. Speed up marks for **Time generating** only (roughly pro rata) **20% (solo only)**

3. Good use of PPC assembly - registers, input/output/clobber lists, branch instructions, etc **50%**

4. Optimisation of calculations and other methods employed (reducing dependencies, etc) **20%**

5. Properly and usefully documenting your code **10%**

**Total 100%**

Feedback and advice during the assignment can be solicited and will be freely given. The same mark will normally be awarded to both students in a pair.

**What you should submit …**

You should submit your final **.cpp** version of the program you produced (*working*, please, or with an explanation of what's wrong with it) via the assignment link on BB. Only one person per pair need submit the work. Your code should build and run in **Release** mode for the **PS3** and be **comprehensively** **commented**. I will be marking your program code, so make it understandable, structured, and well formatted. Your code should have your names embedded in it anyway, please confirm that's the case!

Your code will be run to check it, and to obtain your performance figures.

Please ensure your final submitted **C++** file is named as follows (pairs or solo accordingly):

*pairs* **surname1\_surname2\_SHOT\_on\_Goal\_2017.cpp**

*solo* **surname\_SHOT\_on\_Goal\_2017.cpp**

**“BEER” PRIZE** – *A league table of the fastest combined times (solution + generating) will be compiled and some sort of prize (not necessarily beer) will be awarded for the fastest - it won't be a rugby ball, or a trophy. Solo submissions will be ranked against the times seen for the Generate Flight Path (Time Generating) times only.*

**Redacted version of the relevant code**

// Some constants

const float g(9.81F); // (m/s/s) gravity

const float Pi(3.14159265358979323846F); // This value stolen from M\_PI defines in Math.h) - used to convert degrees to radians

const float dataEnd = -1.0F; // End of data marker

const int x = 0; // coordinate system

const int y = 1;

#define SPACE ' '

const double micro(1.0e6F); // scaling factor to give microseconds

const double milli(1.0e3F); // scaling factor to give milliseconds

// Kick metrics

//\*\*\*SHOT Limit range of angles to help speed things up, and help elsewhere?

const float minAngle(18.0F); // (deg) \*\*\*SHOT We know the answer for kick distance 12.0m is 20 degrees.

const float maxAngle(23.0F); // (deg)

const float minSpeed(5.0F); // (m/s) pretty pathetic!

const float maxSpeed(32.0F); // (m/s) who let Superman on the pitch!? 26 +/-1.7 m/s is optimal kick speed.

const float maxHeight(8.5F); // (m) trajectories above this height can't be displayed (out the park!)

float kickSpeed; // (m/s) calculated

float kickAngle; // (deg) calculated

// Pitch metrics

const float minDistanceToGoal(5.0F); // (m) Probably too close! Might get charged down!

const float maxDistanceToGoal(50.0F); // (m) This is almost half a standard rugby pitch length.

const float crossBarHeight(3.0F); // (m) 3m is the standard rugby cross bar height.

const float margin(0.5F); // (m) allow for a margin of error getting it over.

const float goalPostHeight(7.0F); // (m) Upto 16m usually. Don't make higher, it won't fit on display!

float distanceToGoal; // (m) Kicking distance, input by user.

// Final flight path of ball as a series of x,y coordinates

const float deltaD(0.5F); // (m) increment used for speed and angle.

const float deltaY(0.25F); // (m) increment in the height direction (vertical exaggeration of x2)

const float yScale = 1.0F / deltaY; // vertical scaling factor

const float xScale = 1.0F / deltaD; // horizontal scaling factor

const int maxDataPoints = (int)((maxDistanceToGoal + 2.0F) / deltaD); // calculate a data point for each 0.5 metre along

//+ 2.0 so that ball appears beyond the goal at maxDistance

float flightPath[104 + 1][2]; // (x,y) coords (m,m) of ball flight. Terminate with 'dataEnd' if fewer than

maxDataPoints used.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int main(void)**

{

bool foundCombo(false);

for (int reps(0); reps < repeats; ++reps)

{

start = sys\_time\_get\_system\_time(); //\*\*\* PS3

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

foundCombo = findSHOTonGoalSpeedAndAngle(&kickSpeed, &kickAngle, distanceToGoal);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

stop = sys\_time\_get\_system\_time(); //\*\*\* PS3 timers return time in microseconds

if ((double)(stop - start) < SpeedAndAngleTime) **SpeedAndAngleTime** = (double)(stop - start); // record fastest time

}

if (foundCombo)

{

cout << " solution found. Generating ball's flight path for display...";

for (int reps(0); reps < repeats; ++reps)

{

start = sys\_time\_get\_system\_time(); //\*\*\* PS3

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

generateFlightPath(kickSpeed, kickAngle);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

stop = sys\_time\_get\_system\_time(); //\*\*\* PS3 timers return time in microseconds

if ((double)(stop - start) < GenerateFlightPathTime) **GenerateFlightPathTime** = (double)(stop - start);

}

showFlightPathResults(kickSpeed, kickAngle, distanceToGoal);

}

else cout << "no solution found.\n";

cout << "\nDone...";

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Figure out the first working combination of Speed and Angle.

// Using the lowest kick speed to begin with, increment through the angles from low to high until one is found that gets

// the ball over the bar. If none found increase the speed and repeat. Rationale: it's better to use as little

// energy in the kick as possible! DeltaD is used as the increment for both angle and speed.

**bool findSHOTonGoalSpeedAndAngle(float\* speed, float\* angle, float x)**

{

float nextSpeed;

float nextAngle(minAngle); // Start with shallowest angle...

float height;

bool foundCombo(false); // Found combination of speed and angle that gets ball over bar?

while (!foundCombo && !(nextAngle > maxAngle)) // Think de Morgan's Theory, perhaps.

{

float AngleRads = (nextAngle \* (Pi / 180.0F)); // Need radians for cos and tan functions

nextSpeed = minSpeed; // reset minimum speed

while (!foundCombo && !(nextSpeed > maxSpeed))

{

height = ((-g \* x\*x) / (2.0F \*cos(AngleRads) \*cos(AngleRads) \* (nextSpeed\*nextSpeed))) + (x \*tan(AngleRads));

if (height > crossBarHeight + margin) // Success!

{

\*speed = nextSpeed; // Record the working combination...

\*angle = nextAngle;

foundCombo = true; // ... and stop looking.

}

else nextSpeed += deltaD; // Otherwise try next speed up (+0.5 m/s).

}

nextAngle += deltaD; // no joy, try next angle up (+0.5 degrees).

}

return (foundCombo);

}

// With metrics found, calculate the flight path coords. Uses 'flightPath[104][2]' array as global.

**void generateFlightPath(float speed, float angle)**

{

float yValue(0.001F); // ball is sitting on a tee just above the ground begin with, of course!

float xValue(0.0F); // ...and hasn't moved yet.

const float AngleRads = (angle \* (Pi / 180.0F)); // Need radians for cos and tan functions

int i(0);

for (; i < maxDataPoints && (yValue > 0.0) && (yValue <= maxHeight); ++i) // If height goes negative or too high, STOP!

{

flightPath[i][x] = xValue; // store data points

flightPath[i][y] = yValue;

xValue += deltaD; // do for each increment tick across the pitch

// find the 'y' (height) for each 'x' distance using the angle and speed previously found (same equation as above)

yValue = ((-g \* xValue \* xValue) / (2.0F \* cos(AngleRads) \* cos(AngleRads) \* (speed \* speed))) + (xValue \* tan(AngleRads));

}

// Finished generating required data points, now mark end-of-data with -1.0 (dataEnd)

flightPath[i][x] = dataEnd;

flightPath[i][y] = dataEnd;

}

**Solutions to some other distances.**

Your final version should still produce correct answers for distances other than 12.0m, and this will be checked when your code is run.

**Distance (m) Angle Speed**

**found (°) found (m/s)**

**5-9m *do not produce a solution with the reduced angle range.***

10 22.5 30

10.5 21.5 31.5

11 21 31

11.5 20.5 30.5

**12 20 30.5**

12.5 19.5 31

13 19 31

13.5 18.5 31.5

14 18 32

14.5 18 31

15 18 30

15.5 18 29.5

16 18 29



17.5 18 28

18.5 18 27.5

20 18 27

21 18 27

22.5 18 27

25 18 27.5

28 18 28

31 18 28.5

35 18 29.5

37.5 18 30

39.5 18 30.5

41.5 18 31

43 18 31

45 18 31.5

47.5 18.5 32

50 19 32

 2017

#### Software-Hardware Optimisation Techniques

**B.Sc./M.Sc. Games Software Development**

**Assignment 2 – A SHOT on Goal**

**Marks Sheet**

**Name 1**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Student ID** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name 2**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Student ID** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Marking Scheme Marks distribution**

1. Speed up marks for **Time to solution** (roughly pro rata) **(for pairs)** /**10%**

2. Speed up marks for **Time generating** (roughly pro rata) **(for pairs)** /**10%**

2. Speed up marks for **Time generating** only (roughly pro rata) **(solo)** /**20%**

3. Good use of PPC assembly - registers, Input/output/clobber lists, branch instructions /**50%**

4. Optimisation of calculations and other methods employed (reducing dependencies etc) /**20%**

5. Properly and usefully documenting your code /**10%**

## TOTAL /100\*

***\*(This assignment constitutes 35% of the overall module assessment.)***

*General Feedback (see submission for further comments)*

**The deadline for this assignment is midnight, Monday 19th March, 2018, via Blackboard**

*Dr Adrian Oram, Cantor Building, ACES.*

**What you were asked to do …**

**(for those in pairs)**

Optimise the functions:

**findSHOTonGoalSpeedAndAngle (&kickSpeed, &kickAngle, distanceToGoal);**

and: **generateFlightPath (kickSpeed, kickAngle);**

in order to speed up their respective execution times as much as possible, using the 12m distance as your benchmark of performance. **You are expected to develop the core of the functions using PowerPC assembly for the PS3**, but utilise other optimising methods as you see fit too. No code/functionality should be moved outside the timed zones.

The same mark will normally be awarded to both students in a pair.

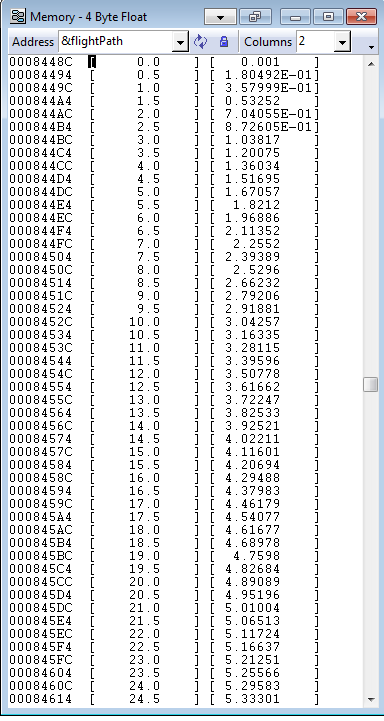
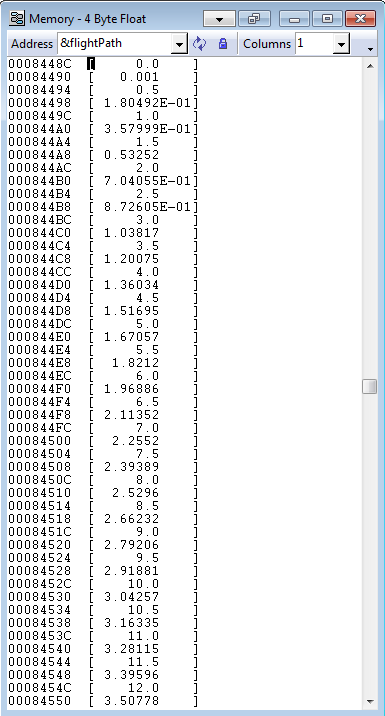


**(for those going solo)**

As above but only optimise the function: **generateFlightPath (kickSpeed, kickAngle);**

**SHOT on Goal - Addendum.**

The **flightPath** array looks like this in memory, the **x**’s and **y**’s follow consecutively… accessing it in assembly can be straightforward…



**x** y

**x y**

**x y**

**x y**

**. .**

**. .**

**. .**

**x**

**y**

**x**

**y**

**x**

**y**

**x**

**y**

**.**

**.**

**.**