Path 1:

```
(* Error Key: floor = {-10,-10,-10),
2nd side wall = {-25, -25, -25}, ceiling = {25,25,25},
back before side = {50, 50, 50}, tin = {-100, -100, -100} *)

(* set initial parameters *)
(* dimensions of court *)
a = 6.4; (* width *)
b = 9.75; (* length *)
h = 5.64; (* height *)
tin = .0975; (* height tin *)

(* parameters for range of good boast *)
xGood = .5;
yGood = b - .5;

(* create a list for problem *)
problem = {0, 0, 0};

(* initial angles and velocity *)
theta = 1.6;
phi = .95;
v = 13;

(* position of boast *)
x0 = 5.01;
y0 = 1.68;
z0 = .67;

(* initial velocity components *)
vx0 = v * Sin[theta] * Cos[phi];
vy0 = v * Sin[theta] * Sin[phi];
vz0 = v * Cos[theta];

(* other parameters *)
deltaT = .001;
m = .024; (* mass of squash ball *)
c = .47; (* drag coefficient of a sphere *)
rho = 1.225; (* density of fluid (k/m^3) - air *)
r = .02; (* radius of ball *)
area = Pi * (r) ^ 2;
```

```
e = .62; (* coefficient of restitution of ball *)
k = (1/2) * c * rho * area;
g = 9.81;
t = 0;
dataTotal = {{x0, y0, z0, t, vx0, vy0, vz0}};


(* use Euler-Cromer Method to evaluate velocities and positions along path 1 *)
While[x0 ≤ a,
     (* x component *)
     vx1 = vx0 + (-k/m) * Abs[vx0] * vx0 * deltaT;
     x1 = x0 + vx1 * deltaT;

     (* y component *)
     vy1 = vy0 + (-k/m) * Abs[vy0] * vy0 * deltaT;
     y1 = y0 + vy1 * deltaT;

     (* z component *)
     vz1 = vz0 + (-g - (k/m) * Abs[vz0] * vz0) * deltaT;
     z1 = z0 + vz1 * deltaT;

     (* iterate time and store data *)
     t += deltaT;
     AppendTo[dataTotal, {x1, y1, z1, t, vx1, vy1, vz1}];

     (* update values *)
     vx0 = vx1; (* x values *)
     x0 = x1;
     vy0 = vy1; (* y values *)
     y0 = y1;
     vz0 = vz1; (* z values *)
     z0 = z1;

  (* address potential loops problems *)
     If[z0 ≤ r, problem = {-10, -10, -10}];
  (* if ball hits floor assign value and break *)
     If[problem == {-10, -10, -10}, Break[]];
     If[z0 ≥ h - r, problem = {25, 25, 25}];
  (* if ball hits ceiling assing value and break *)
     If[problem == {25, 25, 25}, Break[]];

  (* this statement allows it so the loop is stopped when
   the center of the ball is within its radius to the wall *)
```

```
If[x0 > a - r, Break[], Continue[]]]
```

In[ ]:=

Get New Positions and Velocities to start path 2:

In[ ]:=
```
(* positions are the same as x0, y0,
z0 already stored from previous code block *)
(* reset initial conditions velocity -
 multiplied by coefficient of restitution *)
vx0 = -e * vx0; (* note the negative sign becuse
 of the plane it hits changes directions*)
vy0 = e * vy0;
vz0 = e * vz0;
```

Path 2:

```
While[y0 ≤ b,
    (* x component *)
    vx1 = vx0 + (-k / m) * Abs[vx0] * vx0 * deltaT;
    x1 = x0 + vx1 * deltaT;

    (* y component *)
    vy1 = vy0 + (-k / m) * Abs[vy0] * vy0 * deltaT;
    y1 = y0 + vy1 * deltaT;

    (* z component *)
    vz1 = vz0 + (-g - (k / m)) * Abs[vz0] * vz0 * deltaT;
    z1 = z0 + vz1 * deltaT;

    (* iterate time and store data *)
    t += deltaT;
    AppendTo[dataTotal, {x1, y1, z1, t, vx1, vy1, vz1}];

    (* update values *)
    vx0 = vx1; (* x values *)
    x0 = x1;
    vy0 = vy1; (* y values *)
    y0 = y1;
    vz0 = vz1; (* z values *)
    z0 = z1;
```

```
(* address potential loops problems *)
    If[z0 ≤ r, problem = {-10, -10, -10}];
(* if ball hits floor assign value and break *)
    If[problem == {-10, -10, -10}, Break[]];
    If[y0 > b - r && z0 ≤ tin + r, problem = {-100, -100, -100}];
(* if ball hits tin assign value and break *)
    If[problem == {-100, -100, -100}, Break[]];
    If[z0 ≥ h - r, problem = {25, 25, 25}];
(* if ball hits ceiling assign value and break *)
    If[problem == {25, 25, 25}, Break[]];
    If[x0 < 0, problem = {-25, -25, -25}];
(* if ball hits side wall first assign value and break *)
    If[problem == {-25, -25, -25}, Break[]];

(* this statement allows it so the loop is stopped when
 the center of the ball is within its radius to the wall *)
    If[x0 < 0, Break[]];
    If[y0 > b - r, Break[], Continue[]]]
```

Get New Positions and Velocities to start Path 3:

```
In[•]:= (* positions are the same as x0, y0,
z0 already stored from previous code block *)

(* reset initial conditions velocity -
 multiplied by coefficient of restitution *)
vx0 = e * vx0;
vy0 = -e * vy0;
(* note the negative sign becuse of the plane it hits changes directions *)
vz0 = e * vz0;
```

Path 3:

In[ ]:=
```
While[x0 ≥ 0,
    (* x component *)
    vx1 = vx0 + (-k / m) * Abs[vx0] * vx0 * deltaT;
    x1 = x0 + vx1 * deltaT;

    (* y component *)
    vy1 = vy0 + (-k / m) * Abs[vy0] * vy0 * deltaT;
    y1 = y0 + vy1 * deltaT;

    (* z component *)
    vz1 = vz0 + (-g - (k / m)) * Abs[vz0] * vz0 * deltaT;
    z1 = z0 + vz1 * deltaT;

    (* iterate time and store data *)
    t += deltaT;
    AppendTo[dataTotal, {x1, y1, z1, t, vx1, vy1, vz1}];

    (* update values *)
    vx0 = vx1; (* x values *)
    x0 = x1;
    vy0 = vy1; (* y values *)
    y0 = y1;
    vz0 = vz1; (* z values *)
    z0 = z1;

  (* address potential loops problems *)
    If[z0 ≤ r, problem = {-10, -10, -10}];
  (* if ball hits floor assign value and break *)
    If[problem == {-10, -10, -10}, Break[]];
    If[z0 ≥ h - r, problem = {25, 25, 25}];
  (* if ball hits ceiling assign value and break *)
    If[problem == {25, 25, 25}, Break[]];
    If[y0 ≤ 0, problem = {50, 50, 50}];
    If[problem == {50, 50, 50}, Break[]];

  (* this statement allows it so the loop is stopped when
   the center of the ball is within its radius to the floor *)
    If[y0 < 0, Break[]];
    If[x0 < r, Break[], Continue[]]]
```

Extract Final Position:

In[∘]:=
```
finalPosition = dataTotal[[Length[dataTotal]]]
```

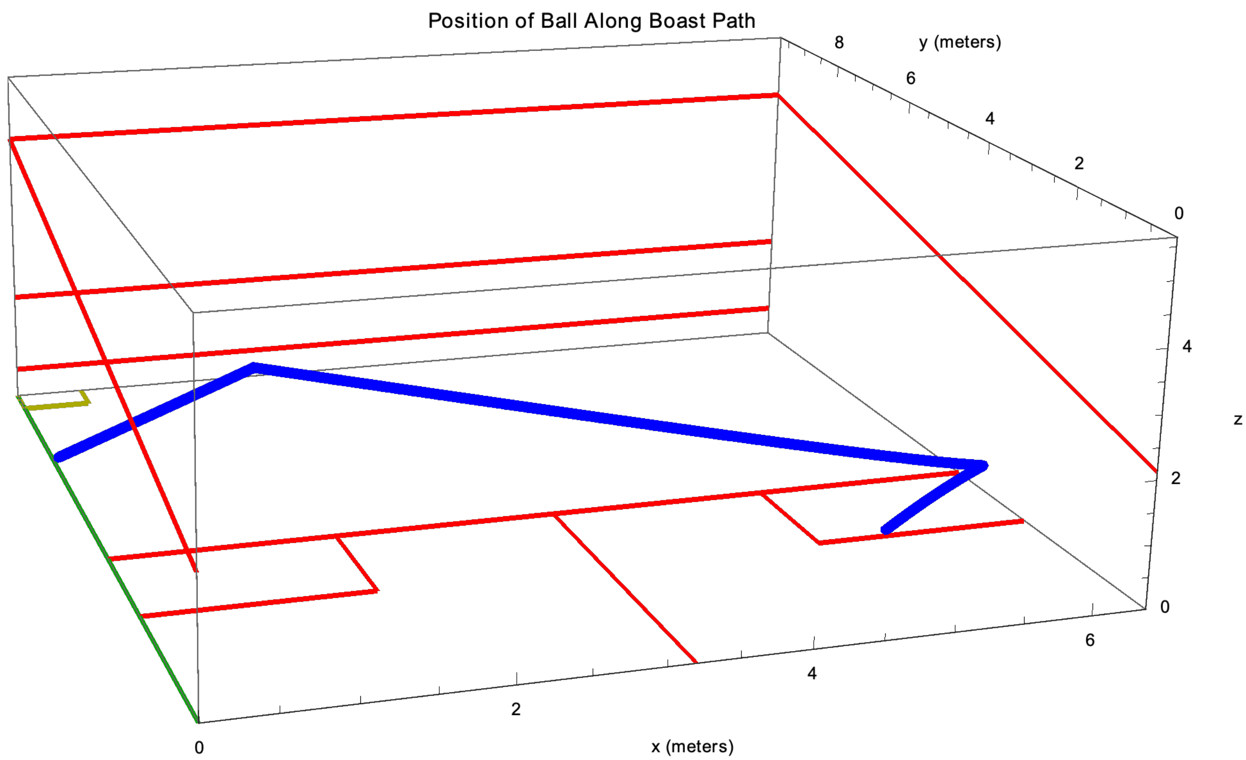
Out[∘]=
```
{0.0198284, 7.19583, 0.128648, 1.913, -2.58545, -3.46344, -0.0411369}
```

Final Plots:

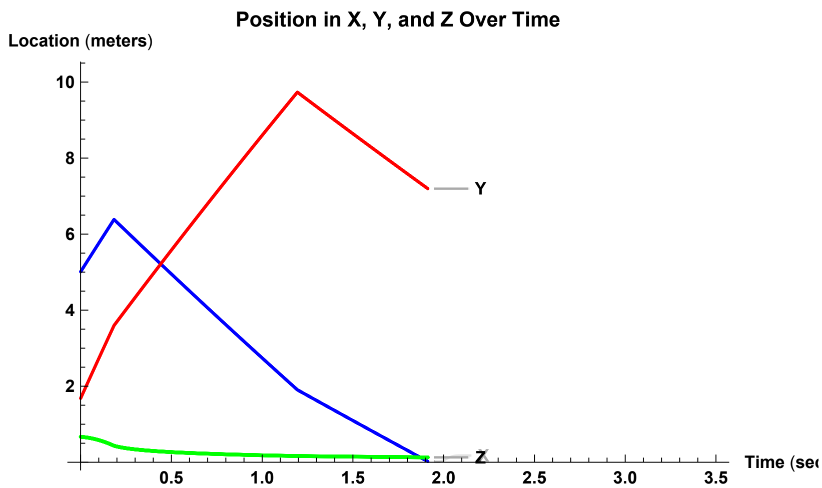
In[∘]:=
```
dataPosition = Drop[dataTotal, None, {4, 7}];

(* lines on singles court *)
tCross = Graphics3D[{Thick, Red, Line[{{0, 4.21, 0}, {6.4, 4.21, 0}}]}];
tMiddle = Graphics3D[{Thick, Red, Line[{{3.2, 0, 0}, {3.2, 4.21, 0}}]}];
leftBox =
  Graphics3D[{Thick, Red, Line[{{1.6, 4.21, 0}, {1.6, 2.61, 0}, {0, 2.61, 0}}]}];
rightBox = Graphics3D[{Thick, Red,
    Line[{{4.8, 4.21, 0}, {4.8, 2.61, 0}, {6.4, 2.61, 0}}]}];
tin = Graphics3D[{Thick, Red, Line[{{0, 9.75, .48}, {6.4, 9.75, .48}}]}];
service = Graphics3D[{Thick, Red, Line[{{0, 9.75, 1.78}, {6.4, 9.75, 1.78}}]}];
top = Graphics3D[{Thick, Red, Line[{{0, 9.75, 4.57}, {6.4, 9.75, 4.57}}]}];
backTop = Graphics3D[{Thick, Red, Line[{{0, 0, 2.13}, {6.4, 0, 2.13}}]}];
(* currently not in plot for aesthetics *)
leftHeight = Graphics3D[{Thick, Red, Line[{{0, 0, 2.13}, {0, 9.75, 4.57}}]}];
rightHeight = Graphics3D[{Thick, Red, Line[{{6.4, 0, 2.13}, {6.4, 9.75, 4.57}}]}];

(* lines representing good shots *)
lineNick = Graphics3D[{Thick, Darker[Green], Line[{{0, 0, 0}, {0, 9.75, 0}}]}];
boxGood = Graphics3D[{Thick, Darker[Yellow],
    Line[{{0, 9.75, 0}, {0, yGood, 0}, {xGood, yGood, 0}, {xGood, 9.75, 0}}]}];

(* plot path and court *)
path = ListPointPlot3D[dataPosition, PlotRange → {{0, a}, {0, b}, {0, h}},
   PlotStyle → Blue,  AxesLabel → {"x (meters)", "y (meters)" , "z (meters)"},
   PlotLabel → "Position of Ball Along Boast Path"];

Show[path, tCross, tMiddle, leftBox, rightBox, tin,
 service, top, leftHeight, rightHeight, lineNick, boxGood]

(* create data tables for x, y and z over time *)
(* z *)
xTime = {};
j = 1;
```

```
While[j ≤ Length[dataTotal],
     AppendTo[xTime, {dataTotal[[j, 4]], dataTotal[[j, 1]]}];
     j += 1]


(* z *)
yTime = {};
j = 1;
While[j ≤ Length[dataTotal],
     AppendTo[yTime, {dataTotal[[j, 4]], dataTotal[[j, 2]]}];
     j += 1]


(* z *)
zTime = {};
j = 1;
While[j ≤ Length[dataTotal],
     AppendTo[zTime, {dataTotal[[j, 4]], dataTotal[[j, 3]]}];
     j += 1]

(* look at graphs together *)
xGraph = ListPlot[xTime, PlotStyle → Blue, PlotLabels → {"X"}];
yGraph = ListPlot[yTime, PlotStyle → Red, PlotLabels → {"Y"}];
zGraph = ListPlot[zTime, PlotStyle → Green, PlotLabels → {"Z"}];

Show[xGraph, yGraph, zGraph, PlotRange → {{0, 3.5}, {0, 10}},
 PlotLabel → "Position in X, Y, and Z Over Time",
 AxesLabel → {"Time (seconds)", "Location (meters)"} ]
```

Out[∘]=



Out[∘]=



Compiled Results (Gluon):

In[∘]:=
```
(* import data compiled from gluon *)
data =
  Import["/Users/jdmcatee/Desktop/Computational Methods/Final Project/total.dat",
   "TSV"];
r = .02; (* radius of ball *)
```

```
leeway = 2 * (.0254); (*leeway of 2 inches in z direction *)

(* create separate lists for each occurence in order to plot with colors *)
pFloor = {};
pSide = {}; pCeiling = {};
pBack = {};
pTin = {}; pNick = {};
pGoodBoast = {}; pBadBoast = {};
i = 1;
While[i ≤ Length[data],
 If[data[[i, 4]] == -10, AppendTo[pFloor, data[[i]]],
  If[data[[i, 4]] == -25, AppendTo[pSide, data[[i]]],
   If[data[[i, 4]] == 25, AppendTo[pCeiling, data[[i]]],
    If[data[[i, 4]] == -100, AppendTo[pTin, data[[i]]],
     If[data[[i, 4]] == 50, AppendTo[pBack, data[[i]]],
      If[data[[i, 4]] ≤ r && data[[i, 6]] ≤ r + leeway, AppendTo[pNick, data[[i]]],
       If[data[[i, 4]] ≤ xGood && data[[i, 5]] ≥ yGood,
        AppendTo[pGoodBoast, data[[i]]], AppendTo[pBadBoast, data[[i]]]]]]]]]]];
     i += 1]

pFloor = Drop[pFloor, None, {4, 6}];
pSide = Drop[pSide, None, {4, 6}];
pCeiling = Drop[pCeiling, None, {4, 6}];
pTin = Drop[pTin, None, {4, 6}];
pBack = Drop[pBack, None, {4, 6}];
pNick = Drop[pNick, None, {4, 6}];
pGoodBoast = Drop[pGoodBoast, None, {4, 6}];
pBadBoast = Drop[pBadBoast, None, {4, 6}];

floor = ListPointPlot3D[pFloor, PlotStyle → Black];
side = ListPointPlot3D[pSide, PlotStyle → Blue];
ceiling = ListPointPlot3D[pCeiling, PlotStyle → Brown];
tin = ListPointPlot3D[pTin, PlotStyle → Orange];
back = ListPointPlot3D[pBack, PlotStyle → Purple];
nick = ListPointPlot3D[pNick, PlotStyle → Green];
goodBoast = ListPointPlot3D[pGoodBoast, PlotStyle → Yellow];
badBoast = ListPointPlot3D[pBadBoast, PlotStyle → Red];

Show[floor, side, ceiling, tin, back, nick, goodBoast, badBoast,
 AxesLabel → {"Phi (radians)", "Theta (radians)", "Velocity (meter / second)"},
 PlotLabel → "Monte Carlo Boast Simulation"]

(* calculate percentage of non problem shots that hit nick *)
numberNicks = Length[pNick];
```

```
numberGood = Length[pGoodBoast];
numberBad = Length[pBadBoast];
numberShots = (numberNicks + numberGood + numberBad);
percentNick = N[numberNicks / numberShots];
percentGood = N[numberGood / numberShots];
```

... **Last**: {} has zero length and no last element.

... **Last**: {} has zero length and no last element.

... **Last**: {} has zero length and no last element.

Out[ ]=