Particle Motion Simulation Using Priori Collision Detection

Initialization Code (optional)

Manipulate

```
(*add trace*)
Manipulate[
 gTick;

 Module[{g},
  If[setIC,
   setIC = False;
   annotation = False;
   simPriori@setSize[nParticles];
   If[r0 ≥ 1 / (2 * nParticles), r0 = 0.99 / (2.0 * nParticles)];
   simPriori@setR[r0];
   simPriori@setE[e];

   Do[simPriori@add[{(i * 2 r0) + r0, .5},
      {RandomReal[{-1, 1}], RandomReal[{-1, 1}]},
      If[massType == "random", RandomInteger[{1, 10}], 1]
     ],
    {i, 0, nParticles - 1}
   ]
  ];

  If[runningState == "RUNNING" || runningState == "STEP",
   timeUsed = simPriori@makeStep[delT];
   simTime += timeUsed
  ];

  g = simPriori@updateImage[annotation, addGridLines];
  If[runningState == "RUNNING", gTick += del];
  FinishDynamic[];
  Text@g
 ],
 (*---------- control layout ------------*)

 Grid[{
   {
    Grid[{
      {Button[Style["run", 12], {runningState = "RUNNING"; gTick += del}, ImageSize -> {55, 35}],
       Button[Style["stop", 12], {runningState = "STOP"; gTick += del}, ImageSize -> {55, 35}]},
      {Button[Style["step", 12], {runningState = "STEP"; gTick += del}, ImageSize -> {55, 35}],
       Button[Style["reset", 12],
        {
         setIC = True;
         runningState = "STOP";
         gTick += del
        }, ImageSize -> {55, 35}]
      }}], SpanFromLeft
   },
```

```
{
 Grid[{{
    Grid[{
      {Style["simulation time (sec)", 11]},
      {Style[Dynamic@padIt2[Mod[simTime, 1000], {7, 4}], 11]}
     }, Spacings → {0, 0}],
    Grid[{
      {Style["time step (sec)", 11]},
      {Style[Dynamic@padIt2[delT, {4, 4}], 11]}
     }, Spacings → {0, 0}]
   }}, Spacings → {1.5, .7}, Frame → All, FrameStyle -> Directive[Thickness[.005], Gray]], SpanFromLeft
},
{
 Grid[
  {
   {Row[{"speed", Spacer[10], "(slow)"}],
    Manipulator[Dynamic[delT, {delT = #} &],
     {0.01, 0.2, 0.01}, ImageSize → Tiny, ContinuousAction -> False],
    "(fast)"
   }
  }, Spacings → {.6, .5}, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
 ]
},
{
 Grid[{
   {"how many",
    Manipulator[Dynamic[nParticles,
      {nParticles = #; setIC = True; gTick += del} &],
     {1, 6, 1}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[nParticles, 2], 11], SpanFromLeft
   },

   {"radius",
    Manipulator[Dynamic[r0,
      {r0 = #; setIC = True; gTick += del} &], {0.01, 1, 0.01}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[r0, {2, 2}], 11], SpanFromLeft
   },

   {"mass",
    RadioButtonBar[Dynamic[massType, {massType = #; setIC = True; gTick += del} &],
     {"random" → Style["random", 11], "equal" → Style["equal", 11]}], SpanFromLeft
   },
   {Row[{"annotation", Spacer[5],
      Checkbox[Dynamic[annotation, {annotation = #; gTick += del} &]], Spacer[40], "grid lines",
      Spacer[5], Checkbox[Dynamic[addGridLines, {addGridLines = #; gTick += del} &]]}], SpanFromLeft
   }
  }, Spacings → {1.2, .3}, Alignment → Center, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
 ]
},
{
 Grid[{
   {Style["coefficient of restitution", 12], SpanFromLeft},
   {Grid[{
      {"(inelastic)",
       Manipulator[Dynamic[e,
         {e = #; simPriori@setE[e]; gTick += del} &],
        {0, 1, 0.01}, ImageSize -> Tiny, ContinuousAction -> False],
       "(elastic)"
      },
      {Style[Dynamic@padIt2[e, {2, 2}], 11], SpanFromLeft}
     }, Spacings → {0.3, 0}]
   }
  }, Spacings → {.85, 0.4},
  Alignment → Center, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
```

```
 ]
},
{
 Grid[{
   {Style["select test case", 12]},
   {
    PopupMenu[Dynamic[testCase, {testCase = #;
         runningState = "STOP";
         Which[testCase == 1,

          (delT = 0.1; nParticles = 2; r0 = .12; e = 1;

           simPriori@setSize[nParticles];
           simPriori@setR[r0];
           simPriori@setE[e];
           simPriori@add[{.28, .5}, {.1, 0}, 1];
           simPriori@add[{.72, .5}, {-.1, 0}, 1];
           annotation = False; massType = "equal"; gTick += del

          ),
          testCase == 2,
          (delT = 0.2; nParticles = 2; r0 = .12; e = 1;

           simPriori@setSize[nParticles];
           simPriori@setR[r0];
           simPriori@setE[e];
           simPriori@add[{.28, .5}, {.1, 0}, 10];
           simPriori@add[{.72, .5}, {-.1, 0}, 1];
           annotation = False; massType = "random"; gTick += del
          ),
          testCase == 3,
          (delT = 0.2; nParticles = 2; r0 = .12; e = 1;

           simPriori@setSize[nParticles];
           simPriori@setR[r0];
           simPriori@setE[e];
           simPriori@add[{.38, .5}, {0, 0}, 1];
           simPriori@add[{.88, .5}, {-.1, 0}, 1];
           annotation = False; massType = "equal"; gTick += del
          ),
          testCase == 4,
          (delT = 0.2; nParticles = 2; r0 = .1; e = 1;
           simPriori@setSize[nParticles];
           simPriori@setR[r0];
           simPriori@setE[e];
           simPriori@add[{.1, .1}, {.1, .1}, 1];
           simPriori@add[{.8, .8}, {-.1, -.1}, 1];
           annotation = False; massType = "equal"; gTick += del
          ),
          testCase == 5,
          (delT = 0.2; nParticles = 3; r0 = .1; e = 1;

           simPriori@setSize[nParticles];
           simPriori@setR[r0];
           simPriori@setE[e];
           simPriori@add[{.2, .2}, {.1, 0}, 1];
           simPriori@add[{.5, .2}, {0, 0}, 1];
           simPriori@add[{.8, .2}, {0, 0}, 1];
           annotation = False; massType = "equal"; gTick += del
          ),
          testCase == 6,
          (delT = 0.2; nParticles = 3; r0 = .1; e = 0.5;

           simPriori@setSize[nParticles];
```

```
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.2, .2}, {.1, 0}, 1];
 simPriori@add[{.5, .2}, {0, 0}, 1];
 simPriori@add[{.8, .2}, {0, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
testCase == 7,
(delT = 0.2; nParticles = 3; r0 = .1; e = 1;

 simPriori@setSize[nParticles];
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.2, .2}, {.1, 0}, 1];
 simPriori@add[{.5, .2}, {0, 0}, 1];
 simPriori@add[{.7, .2}, {0, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
testCase == 8,
(delT = 0.2; nParticles = 3; r0 = .1; e = 1;

 simPriori@setSize[nParticles];
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.2, .2}, {.1, 0}, 1];
 simPriori@add[{.5, .2}, {0, 0}, 1];
 simPriori@add[{.8, .2}, {-.1, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
testCase == 9,
(delT = 0.2; nParticles = 3; r0 = .1; e = 1;

 simPriori@setSize[nParticles];
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.2, .4}, {.2, 0}, 1];
 simPriori@add[{.5, .4}, {0, 0}, 1];
 simPriori@add[{.7, .4}, {0, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
testCase == 10,
(delT = 0.2; nParticles = 5; r0 = .08; e = 1;

 simPriori@setSize[nParticles];
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.1, .4}, {.2, 0}, 1];
 simPriori@add[{.3, .4}, {0, 0}, 1];
 simPriori@add[{.46, .4}, {0, 0}, 1];
 simPriori@add[{.62, .4}, {0, 0}, 1];
 simPriori@add[{.78, .4}, {0, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
testCase == 11,
(delT = 0.2; nParticles = 3; r0 = .1; e = 1;

 simPriori@setSize[nParticles];
 simPriori@setR[r0];
 simPriori@setE[e];
 simPriori@add[{.1, .4}, {.2, 0}, 1];
 simPriori@add[{.4, .4}, {0, 0}, 1];
 simPriori@add[{.65, .4}, {0, 0}, 1];
 annotation = False; massType = "equal"; gTick += del
),
```

```
            testCase == 12,
            (delT = 0.2; nParticles = 2; r0 = .1; e = 1;

             simPriori@setSize[nParticles];
             simPriori@setR[r0];
             simPriori@setE[e];
             simPriori@add[{.1, .1}, {.1, .1}, 1];
             simPriori@add[{.1, .9}, {.1, -.1}, 1];
             annotation = False; massType = "equal"; gTick += del
            ),
            testCase == 13,
            (delT = 0.1; nParticles = 3; r0 = .1; e = 1;

             simPriori@setSize[nParticles];
             simPriori@setR[r0];
             simPriori@setE[e];
             simPriori@add[{.1, .5}, {.1, 0}, 1];
             simPriori@add[{.5, .5}, {.05, 0}, 1];
             simPriori@add[{.8, .5}, {-.2, 0}, 1];
             annotation = False; massType = "equal"; gTick += del
            ),
            testCase == 14,
            (delT = 0.05; nParticles = 3; r0 = .1; e = 1;

             simPriori@setSize[nParticles];
             simPriori@setR[r0];
             simPriori@setE[e];
             simPriori@add[{.1, .5}, {.5, 0}, 1];
             simPriori@add[{.5, .45}, {0, 0}, 1];
             simPriori@add[{.7, .35}, {0, 0}, 1];
             annotation = False; massType = "equal"; gTick += del
            )
           ];
           gTick += del} &],
        {
         1 → Style["2 equal mass towards each others", 10],
         2 → Style["2 unequal mass towards each others", 10],
         3 → Style["2 equal mass only one moving", 10],
         4 → Style["2 moving at angle", 10],
         5 → Style["3 equal masses e=1", 10],
         6 → Style["3 equal masses e=0.5", 10],
         7 → Style["one mass hitting 2 masses e=1", 10],
         8 → Style["2 masses hitting 1 from either side", 10],
         9 → Style["one mass hitting 2 masses, e=1", 10],
         10 → Style["one mass hitting 3 masses, e=1", 10],
         11 → Style["one mass hittng 2, small gap, e=1", 10],
         12 → Style["2 masses hittng at angle, e=1", 10],
         13 → Style["3 masses asymmetric hit, e=1", 10],
         14 → Style["3 masses off center hit, e=1", 10]
        }, ImageSize -> All, ContinuousAction -> False]
      }
     }, Frame → False
    ]
   }
  }, Spacings → {0, .6}, Alignment → Center, Frame → None],
(*----control variables--*)
{{gTick, 0}, None},
{{del, $MachineEpsilon}, None},
{{nParticles, 3}, None},
{{setIC, False}, None},
{{runningState, "STOP"}, None},
{{r0, 0.1}, None},
{{e, 1}, None},
{{delT, 0.1}, None},
```

```
{{massType, "random"}, None},
{{annotation, False}, None},
{{simTime, 0}, None},
{{timeUsed, 0}, None},
{{addGridLines, False}, None},

TrackedSymbols :> {gTick},
ControlPlacement → Left,
SynchronousUpdating → False,
SynchronousInitialization → False,
ContinuousAction → False,
Alignment -> Center,
ImageMargins → 0,
FrameMargins → 0,
Paneled → True,
Frame → False,
AutorunSequencing → {1},
Initialization :>
 (

  (*definitions used for parameter checking*)
  integerStrictPositive = (IntegerQ[#] && # > 0 &);
  integerPositive = (IntegerQ[#] && # ≥ 0 &);
  numericStrictPositive = (Element[#, Reals] && # > 0 &);
  numericPositive = (Element[#, Reals] && # ≥ 0 &);
  numericStrictNegative = (Element[#, Reals] && # < 0 &);
  numericNegative = (Element[#, Reals] && # ≤ 0 &);
  bool = (Element[#, Booleans] &);
  numeric = (Element[#, Reals] &);
  integer = (Element[#, Integers] &);

  (*------------------  Priori class  ----------------------------------------------*)
  simClassPriori[$size_?integerStrictPositive, $r0_?numericStrictPositive, $e_?numericPositive] :=
   Module[{size, r0, e, list, resolveCollisionWithWalls, resolveCollisionWithOther, getAllImages,
     couldCollideWithAnotherObject, timeToCollide, move, timeToCollideWithWall,  self},

    (*-------------------------------------------------------------------------*)
    self@makeStep[stepSize_?numericPositive] :=
     Module[{collisionTime, i, j, k, minTimes, stepUsed, listA},

      collisionTime = Table[Infinity, {size}, {size + 1}];
      (*find smallest time to next collision*)
      Do[
       collisionTime[[i, -1]] = self@timeToCollideWithWall[i];
       Do[If[self@couldCollideWithAnotherObject[i, j],
          collisionTime[[i, j]] = self@timeToCollide[i, j]
        ], {j, i + 1, size}
       ], {i, 1, size - 1}
      ];

      collisionTime[[-1, -1]] = self@timeToCollideWithWall[size];
      (*now we have list of collision times, find the smallest*)
      minTimes = Min[collisionTime];
      listA = Position[collisionTime, _?(#1 ≤ minTimes &)];

      If[minTimes ≥ stepSize, (*collistion not in this time step, ignore*)
       (
        stepUsed = stepSize;
        self@move[stepSize]
       )
       ,
       (
```

```
    (*collision is in this time step. See if collision is now*)
    If[minTimes > $MachineEpsilon, (*collision not now but in this step*)
     stepUsed = minTimes;
     self@move[minTimes]
     , (*collision now, do not move, just resolve*)
     stepUsed = 0
    ];

    (*resolve all collisions with same smallest time*)
    Do[{i, j} = listA[[k]];
     If[j ≤ size, (*collision with another particle*)
      self@resolveCollisionWithOther[i, j],
      self@resolveCollisionWithWalls[i](*collision with wall*)
     ],
     {k, 1, Length[listA]}
    ]
   )
  ];

  N@stepUsed
 ];
(*------------------------------------------------------------------------*)
self@timeToCollideWithWall[idx_?integerStrictPositive] :=
 Module[{x, y, vx, vy, time, smallestTime = Infinity},

  x = list[[2, idx, 1, 1]];
  y = list[[2, idx, 1, 2]];
  vx = list[[2, idx, 2, 1]];
  vy = list[[2, idx, 2, 2]];

  If[vx > 0,
   time = (1 - (x + r0)) / vx
   ,
   If[vx < 0,
    time = (x - r0) / Abs@vx,
    time = Infinity
   ]
  ];

  If[time < smallestTime, smallestTime = time];

  If[vy > 0,
   time = (1 - (y + r0)) / vy
   ,
   If[vy < 0,
    time = (y - r0) / Abs@vy
    ,
    time = Infinity
   ]
  ];

  If[time < smallestTime, smallestTime = time];
  smallestTime
 ];
(*------------------------------------------------------------------------*)
self@move[currentStepSize_?numericPositive] := Module[{i},
  Do[
   list[[2, i, 1, 1]] = list[[2, i, 1, 1]] + list[[2, i, 2, 1]] * currentStepSize;
   list[[2, i, 1, 2]] = list[[2, i, 1, 2]] + list[[2, i, 2, 2]] * currentStepSize,
   {i, size}
  ]
 ];
(*------------------------------------------------------------------------*)
```

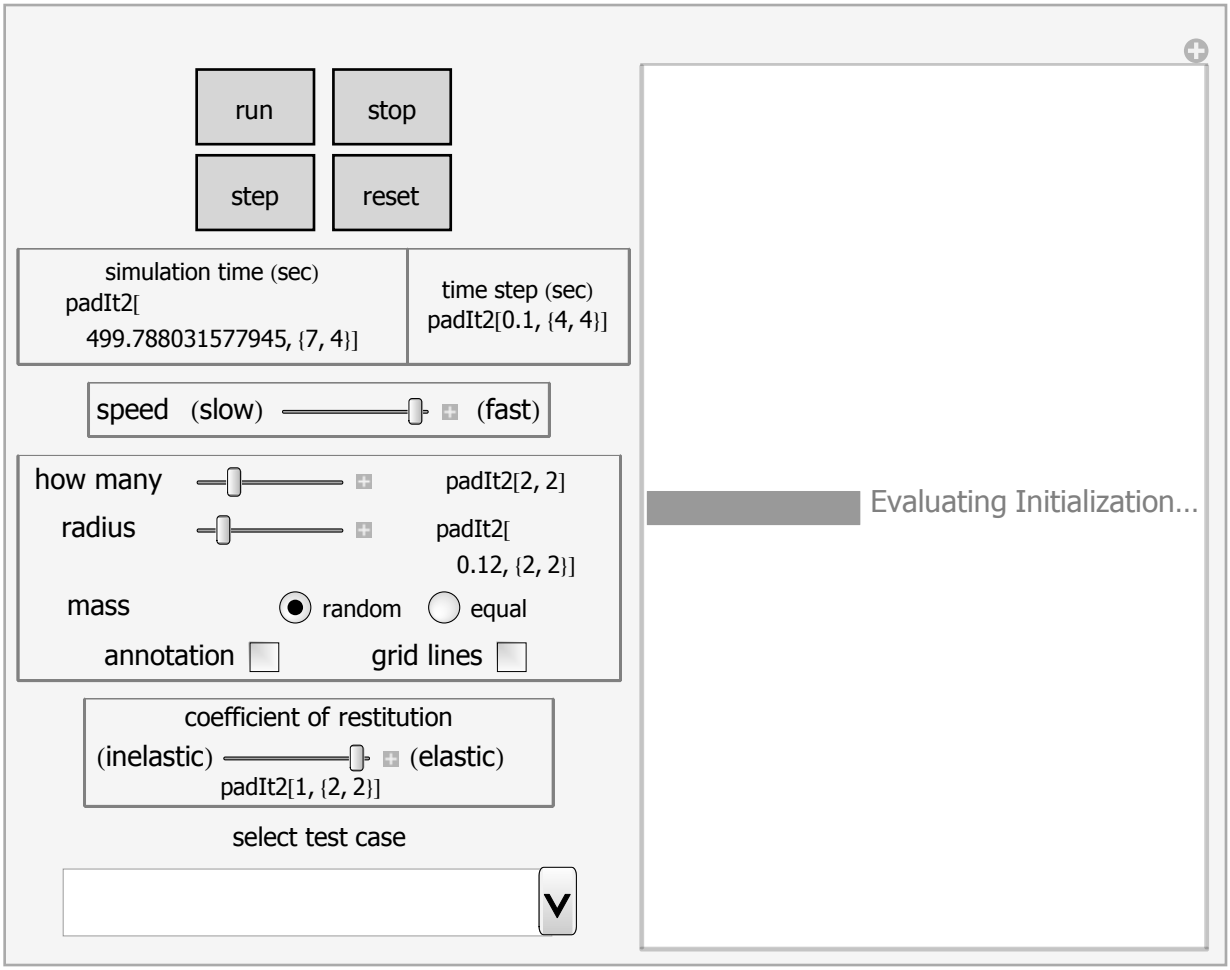
```
self@timeToCollide[i_?integerStrictPositive, j_?integerStrictPositive] :=
 Module[{x1, x2, y1, y2, vx1, vx2, vy1, vy2, t, eq, sol},

  x1 = list[[2, i, 1, 1]];
  y1 = list[[2, i, 1, 2]];
  vx1 = list[[2, i, 2, 1]];
  vy1 = list[[2, i, 2, 2]];

  x2 = list[[2, j, 1, 1]];
  y2 = list[[2, j, 1, 2]];
  vx2 = list[[2, j, 2, 1]];
  vy2 = list[[2, j, 2, 2]];

  If[Abs[vx1 - vx2] ≤ $MachineEpsilon && Abs[vy1 - vy2] ≤ $MachineEpsilon,
   sol = Infinity
   ,
   If[(Chop@EuclideanDistance[{x1, y1}, {x2, y2}] - 2 r0) ≤ 0,
    sol = 0
    ,
    eq = ((x1 - x2) + t (vx1 - vx2)) ^ 2 + ((y1 - y2) + t (vy1 - vy2)) ^ 2 - 4 r0 ^ 2;
    sol = Chop[t /. NSolve[eq == 0, t]];
    sol = Select[sol, Element[#, Reals] &];
    If[Length[sol] > 1,
     sol = Min[Select[sol, # ≥ 0 &]]
     ,
     sol = Infinity
    ]
   ]
  ];

  sol
 ];
(*------------------------------------------------------------------------*)
self@setSize[n_?integerStrictPositive] := (
   size = n ;
  list = {0, Table[0, {n}]}
 );
(*------------------------------------------------------------------------*)
self@setR[rr0_?numericStrictPositive] := r0 = rr0;
(*------------------------------------------------------------------------*)
self@setE[ee_?numericStrictPositive] := e = ee;
(*------------------------------------------------------------------------*)
self@add[{x_?numericPositive, y_?numericPositive},
   {vx_?numeric, vy_?numeric},
   mass_?numericStrictPositive] :=
 (
  list[[1]]++;
  list[[2, list[[1]] ]] = {{x, y}, {vx, vy}, mass, 0}
 );
(*------------------------------------------------------------------------*)
self@resolveCollisionWithWalls[idx_?integerStrictPositive] := Module[{m, x, y, vx, vy},
  x = list[[2, idx, 1, 1]];
  y = list[[2, idx, 1, 2]];
  vx = list[[2, idx, 2, 1]];
  vy = list[[2, idx, 2, 2]];
  m = list[[2, idx, 3]];

  If[x ≥ 1 - r0,
   If[vx > 0,
    vx = - Sqrt[e] vx
   ]
   ,
   If[x ≤ r0,
```

```
    If[vx < 0,
     vx = - Sqrt[e] vx
    ]
   ]
  ];

  If[y ≥ 1 - r0,
   If[vy > 0,
    vy = -Sqrt[e] vy
   ]
   ,
   If[y ≤ r0,
    If[vy < 0,
     vy = -Sqrt[e] vy
    ]
   ]
  ];

  list[[2, idx, 2]] = {vx, vy}
 ];
(*------------------------------------------------------------------------*)
self@resolveCollisionWithOther[i_?integerStrictPositive, j_?integerStrictPositive] :=
 Module[{m1, m2, x1, y1, vx1, vy1, x2, y2, vx2, vy2, theta, r, vvx1, vvx2, leftSpeed, rightSpeed},

  x1 = list[[2, i, 1, 1]];
  y1 = list[[2, i, 1, 2]];
  vx1 = list[[2, i, 2, 1]];
  vy1 = list[[2, i, 2, 2]];
  m1 = list[[2, i, 3]];

  x2 = list[[2, j, 1, 1]];
  y2 = list[[2, j, 1, 2]];
  vx2 = list[[2, j, 2, 1]];
  vy2 = list[[2, j, 2, 2]];
  m2 = list[[2, j, 3]];

  If[Abs[x1 - x2] ≤ $MachineEpsilon,
   theta = Pi / 2
   ,
   theta = ArcTan[(y1 - y2) / (x1 - x2)]
  ];

  r = RotationMatrix[theta];
  {vx1, vy1} = {vx1, vy1}.r;
  {vx2, vy2} = {vx2, vy2}.r;
  {x1, y1} = {x1, y1}.r;
  {x2, y2} = {x2, y2}.r;

  If[x1 < x2,
   leftSpeed = vx1; rightSpeed = vx2
   ,
   leftSpeed = vx2; rightSpeed = vx1
  ];

        m1 vx1 + m2 (vx2 + e (-vx1 + vx2))
  vvx1 = ----------------------------------;
                   m1 + m2
        m2 vx2 + m1 (vx1 + e vx1 - e vx2)
  vvx2 = ----------------------------------;
                   m1 + m2
  r = Transpose[r];
  {vx1, vy1} = {vvx1, vy1}.r;
  {vx2, vy2} = {vvx2, vy2}.r;
```

```
  list[[2, i, 2]] = {vx1, vy1};
  list[[2, j, 2]] = {vx2, vy2}
 ];
(*------------------------------------------------------------------------*)
self@updateImage[idx_?integerStrictPositive, annotation_?bool] := Module[{theta, m, vx, vy, x, y},
  x = Chop@list[[2, idx, 1, 1]];
  y = Chop@list[[2, idx, 1, 2]];
  vx = Chop@list[[2, idx, 2, 1]];
  vy = Chop@list[[2, idx, 2, 2]];
  m = list[[2, idx, 3]];

  theta = If[vx == 0, Pi / 2, ArcTan[vx, vy]];
  list[[2, idx, 4]] =
   {
    {EdgeForm[Black], FaceForm[Hue[.5]], Disk[{x, y}, r0]}, (*FaceForm[Hue[.5]]*)
    If[annotation,
     {Text[Grid[{
          {"#", idx},
          {"mass ", m},
          {"x ", x},
          {"y ", y},
          {"vx ", vx},
          {"vy ", vy},
          {"v ", Sqrt[vy^2 + vx^2]}
         }, Spacings → {0, 0}], {x, y}
       ]},
     Sequence @@ {}
    ],
    If[Abs[vx] > 0 || Abs[vy] > 0, {Arrowheads[r0 / 4],
      Arrow[{{x, y}, {x + (r0 Cos[theta]), y + (r0) Sin[theta]}}, {0, 0}]}, Sequence @@ {}]
   }
 ];
(*------------------------------------------------------------------------*)
self@updateImage[annotation_?bool, addGridLines_?bool] := Module[{idx, g},

  Do[self@updateImage[idx, annotation], {idx, 1, list[[1]]}];

  g = Graphics[{
      {EdgeForm[Black], FaceForm[], Rectangle[{0, 0}, {1, 1}]},
      self@getAllImages[]
    },
    PlotRange → {{0, 1}, {0, 1}},
    ImageSize → {contentSizeW, contentSizeH},
    Axes → False,
    AspectRatio → 1,
    ImageMargins → 0,
    ImagePadding → 5,
    PlotRangePadding → 0,
    If[addGridLines, GridLines → {Range[0, 1, 0.1], Range[0, 1, 0.1]}, GridLines → None],
    GridLinesStyle → Directive[LightGray]];
  g
 ];
(*-----------------------------------------------*)
self@couldCollideWithAnotherObject[i_?integerStrictPositive, j_?integerStrictPositive] :=
 Module[{x1, x2, vx1, vx2, y1, y2, vy1, vy2, bottomYSpeed, leftXSpeed,
   rightXSpeed, topYSpeed, sameXLevel = False, sameYLevel = False},

  y1 = list[[2, i, 1, 2]];
  vy1 = list[[2, i, 2, 2]];
  y2 = list[[2, j, 1, 2]];
  vy2 = list[[2, j, 2, 2]];
  x1 = list[[2, i, 1, 1]];
  vx1 = list[[2, i, 2, 1]];
```

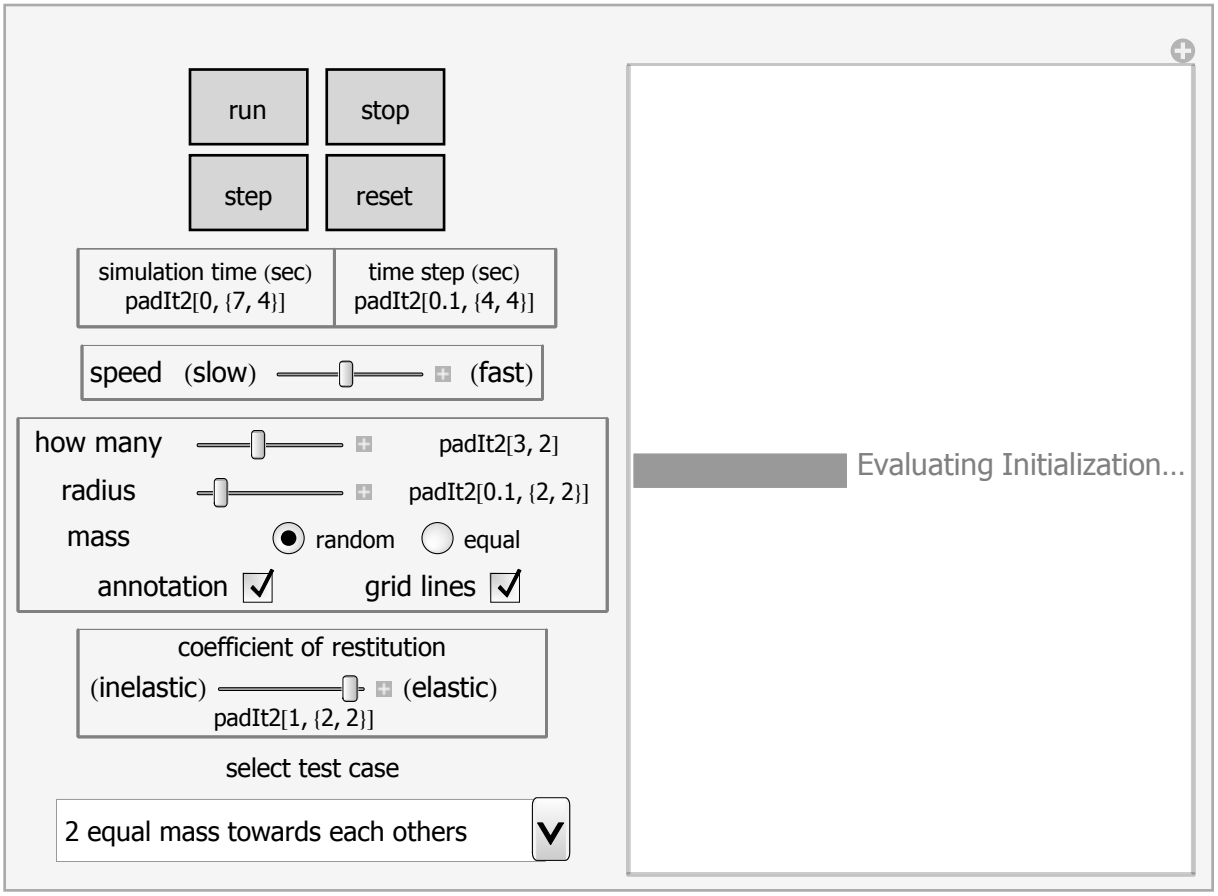
```
x2 = list[[2, j, 1, 1]];
vx2 = list[[2, j, 2, 1]];

If[x1 < x2,

 leftXSpeed = vx1;
 rightXSpeed = vx2
 ,
 If[x1 > x2,
  leftXSpeed = vx2;
  rightXSpeed = vx1
  ,
  sameXLevel = True
 ]
];

If[y1 < y2,
 bottomYSpeed = vy1;
 topYSpeed = vy2
 ,
 If[y1 > y2,
  bottomYSpeed = vy2;
  topYSpeed = vy1
  ,
  sameYLevel = True
 ]
];

If[sameXLevel,
 If[((bottomYSpeed ≥ 0 && topYSpeed ≥ 0 && bottomYSpeed <= topYSpeed) ||
     (bottomYSpeed ≤ 0 && topYSpeed ≥ 0) ||
     (bottomYSpeed ≤ 0 && topYSpeed ≤ 0 && bottomYSpeed ≤ topYSpeed)),
  Return[False, Module]
  ,
  Return[True, Module]
 ]
];

If[sameYLevel,
 If[(leftXSpeed ≥ 0 && rightXSpeed ≥ 0 && leftXSpeed ≤ rightXSpeed) ||
    (leftXSpeed ≤ 0 && rightXSpeed ≥ 0) ||
    (leftXSpeed ≤ 0 && rightXSpeed ≤ 0 && leftXSpeed ≤ rightXSpeed),
  Return[False, Module]
  ,
  Return[True, Module]
 ]
];

If[Abs[Chop@EuclideanDistance[{x1, y1}, {x2, y2}] - 2 r0] ≤ 2 * $MachineEpsilon, (*they are touching*)
 If[(bottomYSpeed ≥ 0 && topYSpeed ≥ 0 && topYSpeed ≥ bottomYSpeed)
    ||
    (bottomYSpeed ≤ 0 && topYSpeed ≥ 0)
    ||
    (bottomYSpeed ≤ 0 && topYSpeed ≤ 0 && bottomYSpeed ≤ topYSpeed)
    ||
    (leftXSpeed ≥ 0 && rightXSpeed ≥ 0 && rightXSpeed ≥ leftXSpeed)
    ||
    (leftXSpeed ≤ 0 && rightXSpeed ≥ 0)
    ||
    (leftXSpeed ≤ 0 && rightXSpeed ≤ 0 && leftXSpeed <= rightXSpeed),
  Return[False, Module]
  ,
  Return[True, Module]
 ]
```

```
      ];

      If[((bottomYSpeed ≥ 0 && topYSpeed ≥ 0 && bottomYSpeed <= topYSpeed) ||
           (bottomYSpeed ≤ 0 && topYSpeed ≥ 0) ||
           (bottomYSpeed ≤ 0 && topYSpeed ≤ 0 && bottomYSpeed ≤ topYSpeed))
         &&
         ((leftXSpeed ≥ 0 && rightXSpeed ≥ 0 && leftXSpeed ≤ rightXSpeed) ||
           (leftXSpeed ≤ 0 && rightXSpeed ≥ 0) ||
           (leftXSpeed ≤ 0 && rightXSpeed ≤ 0 && leftXSpeed ≤ rightXSpeed)),
       Return[False, Module]
      ];

      True
     ];
    (*---------------------------------------------*)
    self@getAllImages[] := list[[ 2, 1 ;; list[[1]], 4 ]];
    (*--- constructor---*)
    self@setSize[$size];
    r0 = $r0;
    size = $size;
    e = $e;
    self
   ];

  contentSizeW = 375;
  contentSizeH = 380;

  (*---------------------------------------------*)
  (* helper function for formatting             *)
  (*---------------------------------------------*)
  padIt1[v_?numeric, f_List] :=
   AccountingForm[Chop[v] , f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True];
  (*---------------------------------------------*)
  (* helper function for formatting             *)
  (*---------------------------------------------*)
  padIt2[v_?numeric, f_List] :=
   AccountingForm[Chop[v] , f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True];
  padIt2[v_?numeric, f_Integer] := AccountingForm[Chop[v] , f, NumberSigns → {"", ""},
    NumberPadding → {"0", "0"}, SignPadding → True];

  simPriori = simClassPriori[2, 0.05, 1];
  simPriori@add[{.28, .5}, {.1, 0}, 1];
  simPriori@add[{.72, .5}, {-.1, 0}, 1];
  simPriori@setE[1];
  simPriori@setR[.12];
 )
)
]
```



Caption

This Demonstration implements a physics collision detection engine using priori (continuous) collision detection to simulate the motion of particles in 2D. You can have up to 6 particles at same time moving inside a closed unit size square. Collisions between particles and the walls can be elastic or inelastic. Mass of particles can be the same or different (randomly generated). No friction or gravity is present in this simulation, hence speed of particles between collisions is constant.



Details (optional)

description of the user interface

Four buttons are used to control running the Demonstration. You select the number of particles and the radius of the particles and click run to see the motion in 2D. All particles have the same size. You can change the collision from being elastic to inelastic using the slider for the coefficient of restitution. You can see the current position, speed and mass of each particle as it moves using the check box labeled "annotation".

possible issues

As this is not a real-time simulation, keeping the frame rate synchronized with the time of collision in order to achieve smooth motion all the time was difficult. The occasional non-smooth motion can be reduced by keeping the simulation speed slow (keeping the simulation time step size small).

References

[1] David Morin, *Introduction to Classical Mechanics: With Problems and Solutions,* New York: Cambridge University Press, 2008.

[2] David Eberly, *Game Physics*, 2nd edition. New York: Morgan Kaufmann, 2010.

[3] Wiki page on collision detection

Control Suggestions (optional)

- [x] Resize Images
- [x] Rotate and Zoom in 3D
- [ ] Drag Locators

- [ ] Create and Delete Locators
- [x] Slider Zoom
- [x] Gamepad Controls
- [x] Automatic Animation
- [ ] Bookmark Animation

Search Terms (optional)

collision detection
physics engine
coefficient of restitution
elastic collision

Related Links (optional)

coefficient of restitution
elastic collision
collision

Authoring Information

Contributed by: Nasser M. Abbasi