CMPE2300 – Lab #02 – Pool-o-tron

In this lab you will create a Billiard Ball Simulator, exercising basic classes, properties, methods, and composition.

Program Specification

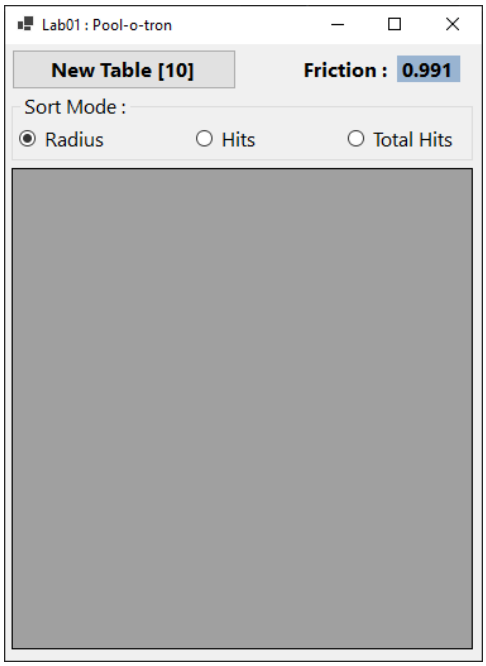
The application will consist of a Ball class, and a Table class as the controller for the Balls. The main form will operate the simulation through the Table class. The goal of the simulation is to direct the "cue" ball towards other randomly placed, randomly sized balls, attempting to achieve the maximum number of collisions between all balls. The stats of each attempt will be held in each ball and the Table will provide the main form with access for display purposes.

There are 2 states of the simulation. When all balls have stopped moving, the game is in Display/Aim state, where the user may provide the direction for the cue ball to shoot. The user will "shoot" by clicking on the CDrawer, starting the simulation. The cue ball will have an initial velocity applied, and table will let all balls interact until they stop, at which time the simulation transitions back to the Display/Aim state, showing the statistics for the "shot".

A constantly running timer will perform all ball movement.

The application will consist of a controller form where the user can set the simulation parameters, and then create a Table. The Table will start in Display/Aim state.

Main/Controller form :



You will require a button, accessible label for friction, a groupBox with sort selection RadioButtons, and a DataGridView. Anchor your UI appropriately.

Other members include a timer, a Table class instance and some flags. The form constructor will be used to bind all necessary events. See the Main Form section of this lab for more details.

The classes required are Ball and Table

class **Ball**, has members :

- public static float property, representing the friction used in ball movement simulation, default = 0.991f
- static Random object, initialized
- Vector2 member called _center, representing the ball center
- Vector2 member called _velocity, representing the ball velocity
- public property Vector2 called Center, get only returning _center
- public property Vector2 called Velocity, get only returning _velocity
- property int called Radius, public get, private set
- property int called Hits, public get, private set
- property int called TotalHits, public get, private set
- property Color called BallColor, public get, private set

** **Note** : If you differ from the suggested naming of the members, the supplied ProcessCollision() method will require modifications to match your member names.

Constructors :

- a public constructor accepting a CDrawer and Color argument. Set the BallColor. Randomize a Radius between 20 and 50, and using that Radius, set the _center to be randomly anywhere fully inside the supplied CDrawer.
- a public constructor accepting a CDrawer. This constructor will make our cue ball. Set the Radius to a constant of 30, BallColor to White, and set the _center to be randomly anywhere fully inside the supplied CDrawer.

Methods :

- public ResetHits(), set the Hits property to 0.
- public SetVelocity(), accepting a Vector2, assign _velocity to the supplied argument.
- override Equals() - return true if the supplied argument is a Ball and overlaps
- override GetHashCode() - return 1
- override ToString() - return a string in the format “Radius : Hits”
- provide IComparable support. Return the appropriate value such that Sort() will result in a descending Radius collection.
- provide a Comparison compliant function with null checking**, such that Sort() will result in a descending Hits collection.
- provide a Comparison compliant function with null checking**, such that Sort() will result in a descending TotalHits collection.
- public Show() accepting a CDrawer. Render the Ball, if it is the cue ball, use a border of 2 in Yellow. You can use a readonly Color member to define the cue ball color. Using ToString(), center the string in the ball in Black.

- public Move(), accepting a CDrawer and List of Ball.
  - slow things down by multiplying the balls velocity by the friction. (*=)
  - if the velocity vector LengthSquared() is less than 0.1f, assign it Vector2.Zero and return. Why ?
  - process CDrawer wall bounces,
  - adjust the _center by the _velocity, ie. move it.
  - Iterate all "target" balls from the supplied collection :
    - Skip yourself
    - Using Equals(), if an overlap is determined with the iteration ball, invoke supplied method ProcessCollision with iteration ball - this bounces the overlapping iteration ball with itself.
    - ** Review the ProcessCollision method to see power of the Vector2 type in performing vector operations like reflection and scaling. Also note the modifications to the Hits/TotalHits properties, tracking collisions.

At this point, you should test your Ball class. Make a single "regular" and cue ball and insert into a form owned collection, let your timer callback Move/Show the balls in a CDrawer. You should be able to verify :

- balls bounce off the walls,
- ball bounce off each other
- ball Hit and TotalHits accumulate appropriately.

Completion only to this point can satisfy a minimum mark award, but you need to confirm with your instructor that required elements are complete, and can be visually verified - see rubric.

class **Table**, has members :

- public property CDrawer, private set, initialized to null
- List of Ball - the balls on our table
- Vector2 member representing the current location of the mouse
- Ball member representing the cue ball, initialized to null
- public property of List of Ball - get only, returns a NEW copy of member list
- public property of bool called Running - representing if simulation is running - returns true if any balls have a non-Zero velocity

Constructor :

- Single default, explicit, but does nothing.

Methods :

- public MakeTable(), accepting integers for table width, height and number of ball required.
  - close any existing CDrawer, initialize a new one to the supplied width, height, no continuous update, redundamouse = true. Assign to the CDrawer member.
  - bind event callback for CDrawer MouseMoveScaled
  - bind event callback for CDrawer MouseLeftClickScaled
  - Invoke MakeBalls() with the required number of balls
  - Invoke ShowTable()
- event handler for MouseLeftClickScaled
  - for all balls, invoke ResetHits()
  - retrieve the subtraction of the current mouse position vector from the cue ball Location vector, this provides the direction vector for the “shot”
  - Normalize the vector and multiply by a constant shot velocity of 40f.
  - Invoke the cue ball SetVelocity() method, with your calculated shot vector. This should result in the cue ball starting the simulation
- event handler for MouseMoveScaled
  - using the supplied point, create and assign it into the mouse position vector
  - if not Running, invoke ShowTable
- public ShowTable()
  - if no CDrawer, return
  - Iterate the balls collection, Invoke Move() and Show()
  - if not Running :
    - Add a yellow line from the cue ball to the current mouse position.
    - Optionally, add an arrow the end of the line to show the vector.
- public MakeBalls(), accepting the number of balls to make.
  - remove all current balls
  - add the required number of “regular” balls, no-overlaps allowed.
  - create a single cue ball, but it must not overlap any existing balls, assign this cue ball as the member cue ball, and add it to the list of balls

Putting it all together : your **Form**

has members :
- consts as required*
- member of type Table, initialized to null
- member of type System.Windows.Forms.Timer, initialized to new instance
- member of type bool, the running flag
- member of type int, representing the number of balls requested by the user

has UI elements :
- Button, Make table
- Label, display/input current Friction value
- RadioButtons - Radius, Hit, HitTotal selections for DataGridView sorting
- DataGridView

Constructor :
- bind [Make Table] button MouseWheel event and Click event to New Table button, mousewheel will allow number of ball requested value to change, and update button text as shown, min = 1
- bind [Friction] label MouseWheel, allow for altering Ball static friction value to incr/decr by 0.001f, display as shown.
- bind all radio buttons to a single Click handler, set Radius button as checked.
- Set the timer interval as 35ms, enable it and bind to an explicit Tick event callback.
- Set the datagridview DataSource to null.

Event Handlers :
- [Make Table] MouseWheel handler - incr/decr balls requested by 1, update button as shown.
- [Make Table] Click handler - assign member to new Table, adjust CDrawer beside the main UI.
- [Radio Buttons] Click handler - invoke helper method UpdateGridView()
- [Friction Label] MouseWheel handler - incr/decr static Ball member for Friction by +/-0.001f, update Label as shown.
- Timer Tick handler :
  - if no Table, return
  - invoke ShowTable(),
  - if flag indicates running, but table Running flag indicates the shot is complete, invoke UpdateGridView()

Helper methods :
- UpdateGridView() - no args, no return
  - Retrieve a copy of the List of Ball from the Table
  - Based on the appropriate check RadioButton, Sort the collection
  - Using the demo supplied :
    - assign the DataGridView Datasource to null, then to your sorted collection – reassigning the Datasource forces a refresh on the control. There are other ways but this is the most straight-forward.

- turn RowHeadersVisible off
- Set the BallColor, Center and Velocity columns to not visible
- Iterate the rows, setting the Radius column Backcolor to the BallColor column value (with Color() cast)
- Invoke AutoResizeColumns(to DisplayedCells)

Base version of the simulation complete.

Rubric :

30% - Ball Class complete only - cue ball and regular ball(s) can be created and output verifies balls bounce, hit accumulation correct. To get this, your main form must have the core to demonstrate this (no marks for just completion without demonstration).

- Form - add a list of balls, and an enabled timer. You should be able to make at least 10 non-overlapping balls. Assign a velocity the first ball, and the balls should bounce around - indicating their accumulated hits.
- Note : this is a minor amount of code, and should be used to verify that your Ball class is "mostly" correct. Addition of the Table class can then be added, form processing added, and this verification code commented/deleted.

60% - Ball, Table and main form UI complete to base version.

+10% - Enhancement on complete Base version, possible enhancements include :

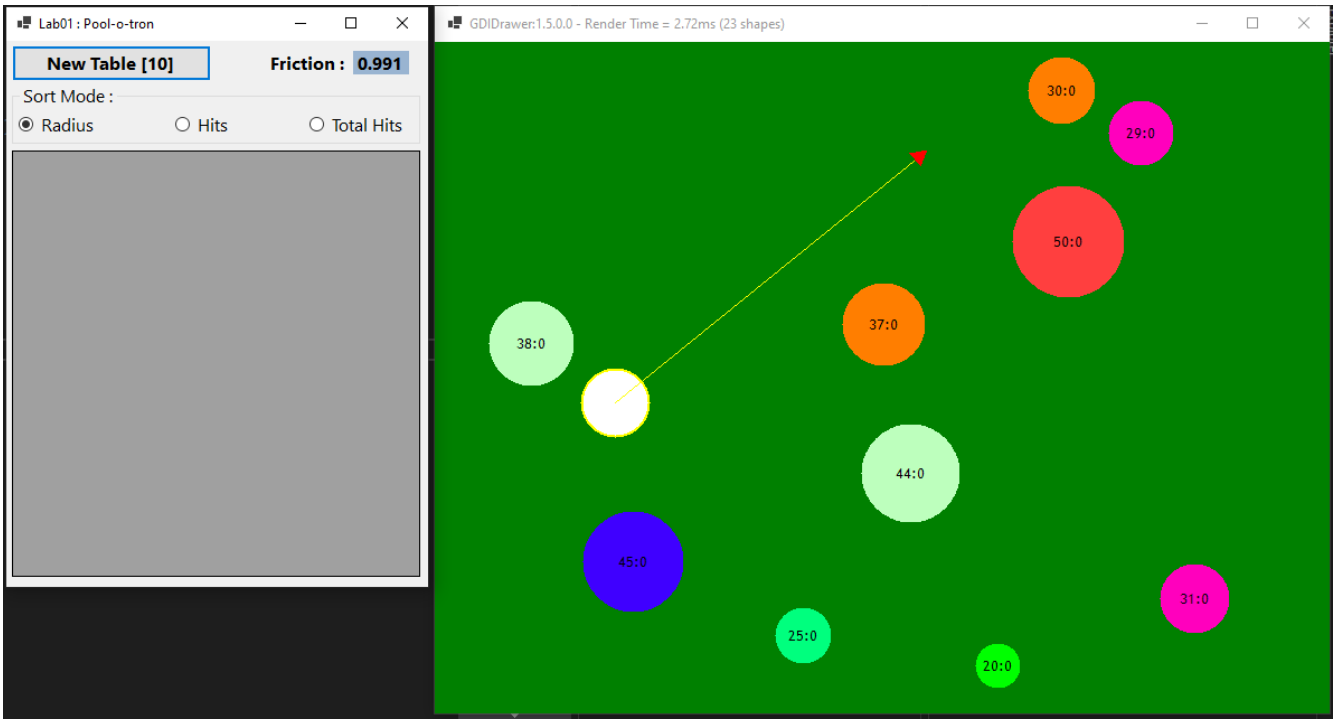
- Modify the shot velocity - remove the 40f multiplier, instead, multiply by the length of the direction vector divided by 15. This will mean a longer vector will be a higher initial velocity for the cue ball (like games do).
- On completion of each shot - #BallsHit %
- Total of all collisions
- animation elements - ie. border on collision highlight for (N)ms.
- Arrow on shot direction vector
- [Your idea here] - ask about whether it would qualify

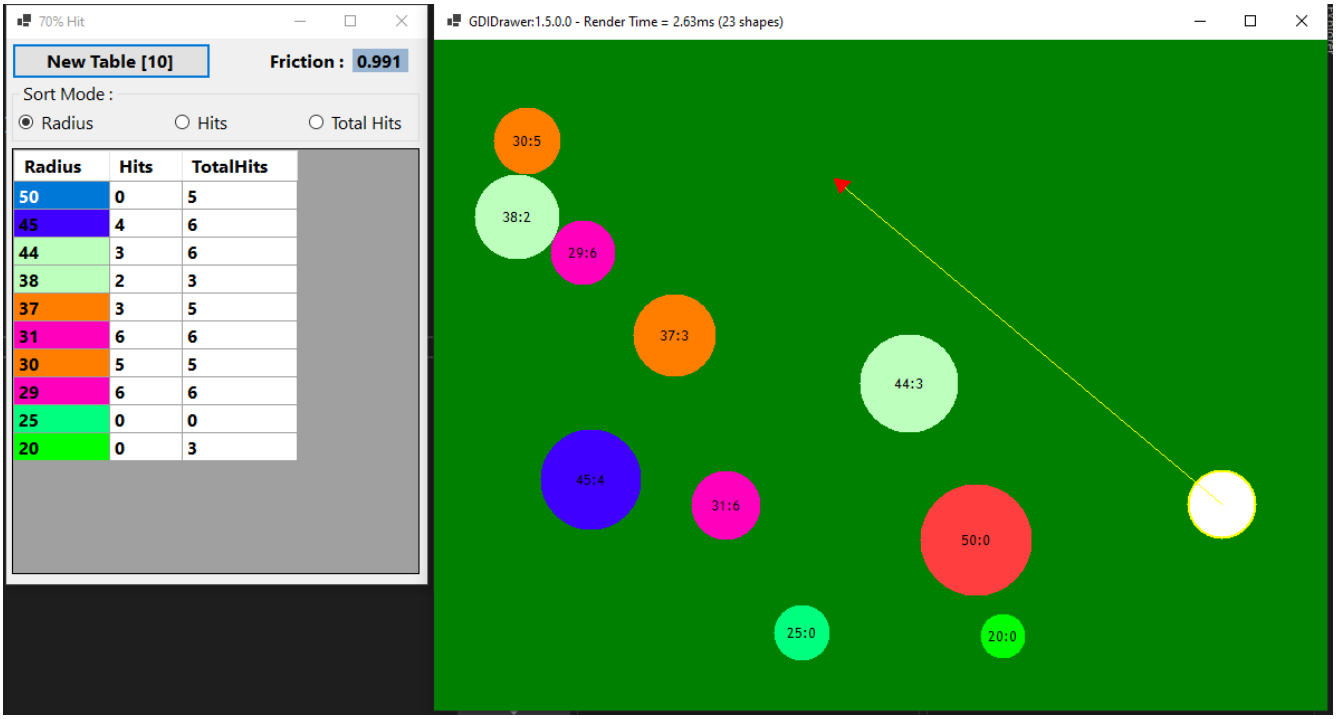
Supplement

You will likely have success coding as specified, but there is a potential issue that may arise. While there are no additional threads required for this lab, the CDrawer events fire appropriately, but execute in the CDrawer thread..

What would this “alien” thread access that the main UI thread accesses too ? Add the appropriate data marshaling construct, on the appropriate data to ensure no thread issues occur. This is actually minimal - examine your CDrawer event handlers - what data object(s) are accessed ? Do other (ie. UI) threads access these objects too ?

Vector2 - A handy library that can be a simple substitute for PointF, but with enhancements for dealing with vector math. Includes overloaded operators for ease of use, see the bounce function provided. [Vector2 - MSDN - Framework 4.8]




70% Hit
New Table [10]
Friction : 0.991
Sort Mode :
Radius
Hits
Total Hits
Radius Hits TotalHits
50 0 5
45 4 6
44 3 6
38 2 3
37 3 5
31 6 6
30 5 5
29 6 6
25 0 0
20 0 3
GDIDrawer:1.5.0.0 - Render Time = 2.63ms (23 shapes)
30:5
38:2
29:6
37:3
44:3
45:4
31:6
50:0
25:0
20:0