Rudimentary 2D Physics Engine

Collin Jones | Medhaj Shrestha

Repo URL: https://github.com/collinjones/Rudimentary-2D-Physics-Engine

Status Summary

-Work done: Collin: create a window to add shapes (with keyboard shortcuts). Collision resolution between objects and boundaries. Vector math and object rendering. Created emitter, pachinko, and solar system. Medhaj: Friction, drag, attraction, repulsion algorithms. Factory Pattern.

-Changes or Issues Encountered: The patterns we wanted to use included singleton and flyweight. Flyweight will not work for what we wanted because we can not have our objects acting the same. Singleton was going to be used for our render object but problems with pointers and passing an argument arose. Going to now use State and Observer patterns.

-In our current prototype, we have the factory pattern implemented. This has helped to hide away creation logic and lessen the amount of repeated code. This has also helped consolidate the colors of specific objects.

Plan for next iteration:

- get a gui working with nanogui. This will help with sandboxing. Our gui will have adding shapes functionality along with turning on/off gravity. We also would like for there to be preset simulation (like a solar system or pachinko) options for the user to see. We also plan to implement a command pattern for the gui, a state pattern for the circles attraction, and an observer pattern for writing certain data to a txt file. We plan to have all of this done before our projects due date, and this will allow for sandboxing for the user.

Class Diagram:

shapeFactory

createCircle(**pos**: Vec2,**vel**: Vec2, **m**: double): circle
createCircle(**pos**: Vec2,**vel**: Vec2, **m**: double, **attractOrRepulse**: bool): circle
createPeg(**pos**: Vec2, **size**: int): Peg
createBoundary(**start**: Vec2, **end**: Vec2): Boundary
createEmitter(**x**: double, **y**: double): Emitter
createRectangle(**x**: int, **y**: int, **w**: int, **h**: int): Rectangle

---

VecMath

+<u>mult</u>(**v1**: Vec2, **scalar**: float): Vec2
+<u>div</u>(**v**: Vec2, **scalar**: int): void
+<u>magnitude</u>(**v**: Vec2): int
+<u>normalize</u>(**v**: Vec2):Vec2

Line

-**length**: double

+Draw(**renderer**: SDL_Renderer): void

---

vec2

-**vec**: SDL_FPoint

+add(**v**: Vec2): void
+sub(**v**: Vec2): void
+multiply(**scalar**: double): void
+divide(**scalar**: double): void
+limit(**msg**: double): void
+magnitude(): int
+Distance(**point**: Vec2):double

Simulation

-**window**: SDL_Window
-**render**: SDL_Renderer
-**e**: SDL_Event
-**init_error**: int
-**quit_flag**: bool

+ GenerateCircle(**pos**: Vec2, **vel**: Vec2, **mass**: double): Circle
+ check_for_errors(): int

0..1   0..*   0..1

---

object

#**position**: Vec2
#**velocity**: Vec2
#**acceleration**: Vec2
#**mass**: double
#**color**: SDL_Color

+ ApplyForce(**force**: Vec2): void
+ Update(): void

0..*

Box

-**length**: double
-**width**: double

+Draw(**renderer**: SDL_Renderer): void

Peg

-**radius**: double
-**diameter**: double
-**mass**: double
-**position**: Vec2
-**velocity**: Vec2
-**color**: SDL_Color

+Draw(**renderer**: SDL_Renderer): void

Boundary

-**pointA**: Vec2
-**pointB**: Vec2
-**length**: double

+ Draw(**renderer**: SDL_Renderer): void
+DistancePointA(**point**: Vec2): double
+DistancePointB(**point**: Vec2): double

Emitter

-**position**: Vec2

+Emit(**circles**: vector<Circle*>): void

Circle

-**radius**: double
-**diameter**: double

+Draw(**renderer**: SDL_Renderer): void
+Edges (**width**: int, **height**: int): void
+CollisionWithLine(**line**: Boundary): bool
+CollisionWithPoint(**point**: Vec2): bool
+PointCollisionLine(**point**: Vec2, **line**: Boundary): bool