2-Dimensional Physics Simulation Environment (Sandbox)

Software Design Document

Branden Stahl

11/8/2023

CS 225, Fall 2023

Embry-Riddle Aeronautical University

Daytona Beach campus

1 Aerospace Boulevard

Daytona Beach, FL 32114

**Introduction:**

The project is a 2-dimensional (2-D) physics engine. The goal is that it can be interacted with by the user. There will be a Graphical User Interface (GUI) that the user can interact with that will show a User Interface (UI) version of the physics simulation. There will be a gravity that can be toggled between using global (base earth gravity of 9.81m/s2 (1)) or particulate gravity (gravity based around the weight and distance of the particles). The reasoning behind this project is to help better understand how physics operations work, but also provide entertainment of providing an interactable 2-D physics simulation, known as a physics sandbox. This document will explain how the project works, document the required math, and show the classes, the class interactions, and show a diagram of it.

**Problem Description:**

The project shows a GUI simulation of a 2-D gravitational physics engine with a focus on user interactions. Specifically, this will be focused on emulating particulate gravitational forces either with no gravitational effect or under gravitational forces of Earth, utilizing elastic collisions. The user can right click on the screen to open a spawning menu that can spawn one of two shapes (Circle or Rectangle), which will have varied weights and sizes that can either be randomly assigned or chosen by the user. There is also a left click method that allows the user to move the shapes. The shapes cannot collide with each other, and the edges of the screens act like solid walls that the shapes cannot get through. Within the settings (accessible by either menu or file) are settings to control the gravity type (Particulate or Global, as defined previously). There will be an exit menu that gives the option to save the simulation or not, which when saved will export it to a file that can then be re-opened later by the user. There are a few notable 2-D gravity physics simulators, such as *Mr. Doob’s Google Gravity*2, which is extremely similar in the sense of varied shapes can be interacted with by the user and will have gravity & the screen edges will be the wall of the program.

As far as math goes, the formulas that have been deduced as necessary are as follows:

Gravitational Constant: (1)

Force between two gravitational bodies: (3)

Acceleration due to gravity: (3)

Force: (3)

Law of Cosines: (4)

Inelastic Collision Formula: (5)

**Problem Solution:**

The program will read and write to a .DAT file (A type of CSV (Comma Separated Values) file), utilizing all the below classes and methods:

ManagerClass: This is the base management class, and from this class the entire project will be managed. This is where all physics and frame timing occur for the simulation. The most important method in this class is the startRender function, as this launches a while loop on a new thread that will ensure that no other code execution will affect the frame timing. This is important because if the frame timing is affected, then the simulation will appear choppy and will make interactions more challenging to handle.

GuiClass: This is the class that handles the creation and closing of all UI’s and will handle ensuring that each object is displayed correctly. The most important method in this class is the render() method, as this method will go through every body that exists and will figure out what shape it is before rendering it exactly how it should be rendered.

GameFrame: This is an extension of the swing library’s JFrame to override the close event to halt all game rendering. This also ensures that the manager class is only running the simulations while this frame is open.

GamePanel: This is an extension of the swing library’s JPanel to override the paintComponent() method, as well as implement some custom behavior that allows for high-framerate rendering.

GenericBody: This is an abstract class that creates the base references for all other shapes. This is only to be used as a reference piece for RectBody and SphereBody.

RectBody: This is a rectangular body class that inherits the GenericBody class. It stores all data pertaining to itself, as well as provide internal error handling for the creation of itself to ensure it wont create itself if it’s got illegal parameters.

SphereBody: This is a spherical body class that inherits the GenericBody class. Just like the RectBody class, this stores all data pertaining to itself as well as the internal error handling.

*Figure 1: UML Diagram*

**ManagerClass**

-manager: ManagerClass  
-gui: GuiClass  
-bodies: ArrayList<GenericBody>  
-constants: ArrayList<Object>  
-settings: ArrayList<Object>  
-timer: Timer  
-lastUpdate: Instant

+main(String): void  
-readData(): void  
+startRender(): void  
+stopRender(): void  
+createBody(int, double, int, double, int, boolean, boolean): void  
+onMouseClick(MouseEvent): void  
-Apply2BodyGravity(GenericBody, GenericBody, double): void  
-ApplyGravity(GenericBody, double): void  
-updatePosition(GenericBody, double): void  
-checkBounds(GenericBody): void  
-setCollisionVelocity(GenericBody, GenericBody, double): void  
-checkCollisions(GenericBody): void  
-updateAll(double): void

**GuiClass**

-mainFrame: GameFrame  
-panel: GamePanel  
-g: Graphics  
-manager: ManagerClass  
-settings: ArrayList<Object>  
-insets: Insets  
  
+GuiClass<ManagerClass, ArrayList<Object>>  
+createMenu(): void  
+createWindow(boolean): void  
+openSpawnMenu(MouseEvent): void  
+render(ArrayList<GenericBody>, int): void

**GameFrame**

GameFrame<ManagerClass>

**GamePanel**

-bodies: ArrayList<GenericBody>  
-framerate: int  
-wireframe: boolean  
-debugLines: boolean  
-manager: ManagerClass  
  
+GameFrame<JFrame, boolean, ManagerClass>  
+updateArgs(ArrayList<GenericBody>, int): void  
+paintComponent(Graphics): void  
-wireframeRender(Graphics): void  
-render(Graphics): void

**GenericBody**

+move(int[]): void  
+getBounds(): int[][]  
+getEdgeBounds(): int[][]  
+getCollidingBodies(ArrayList<GenericBody>): ArrayList<GenericBody>  
+setPreviousPos(): void

**RectBody**

-mass: double-size: double  
-position: int[]  
-velocity: double[]  
-elasticity: double  
-color: Color  
-canCollide: boolean  
-isStatic: boolean  
-prevPos: int[]  
  
+RectBody<double, int, double, int[], boolean, boolean, Color>  
+getCollidingBodies(ArrayList<GenericBody>): ArrayList<GenericBody>

**SphereBody**

-mass: double-size: double  
-position: int[]  
-velocity: double[]  
-elasticity: double  
-color: Color  
-canCollide: boolean  
-isStatic: boolean  
-prevPos: int[]  
  
+SphereBody<double, int, double, int[], boolean, boolean, Color>  
+getCollidingBodies(ArrayList<GenericBody>): ArrayList<GenericBody>

**References:**

1: COMSOL Physical Constants Reference <https://doc.comsol.com/5.5/doc/com.comsol.help.comsol/comsol_ref_definitions.12.025.html>

2: Mr. Doob’s Google Gravity  
<https://mrdoob.com/projects/chromeexperiments/google-gravity/>

3: United States Naval Academy Physics Chapter 13 PDF  
<https://www.usna.edu/Users/physics/finkenst/homepage_files/SP211/Chapter_13.pdf>

4: Wikipedia: Law of Cosines <https://en.wikipedia.org/wiki/Law_of_cosines>

5: Wikipedia: Inelastic Collision <https://en.wikipedia.org/wiki/Inelastic_collision>