README

This is a mini simulation of a pinball table. The player is given a board with obstacles, 2 paddles, and maximum 2 pinballs. The pinballs could collide with board borders, paddles, obstacles, and each other. The player could create pinballs and control the paddles.

Control: A, D: left/right paddles; Z: jitter ball, Space: create pinball.

Features:

1. Pinball (Pinball.cs, BallManager.cs):

When there are less than 2 pinballs on the board, a pinball could be generated by hitting the space key. The BallManager will instantiate it, then ask it to find a valid location to spawn, i.e. within the board and not overlapping with other obstacles, walls, or paddles. If all attempts fail, the location will fall back to (0,1,0).
Gravity accelerates balls along the -x direction, which is the direction the ball will move towards unless it hits any obstacles, walls, or paddles. Speed is capped to 22f, and Y (up / into board) direction is locked to board height. The pinball detects collisions by: 1. Add all colliders in the game to its list at initialization, 2. Update its physics for potential next position, 3. Iterate through all colliders with its next position and check for collision. Collision (change of ball's velocity) is handled differently within each collider class, united under a ICustomeCollider interface.
The manager resolves pairwise ball–ball interactions each frame. If centers are closer than the sum of radius, it splits positional correction along the collision normal. Then, an equal mass elastic impulse is applied along the normal (with restitution). Speed is clamped if needed.
A pinball is destroyed by the BallManager after it reaches the gutter area. Jitter is done by adding a small impulse to the ball.

2. Walls (WallCollider.cs, SWallCollider.cs):

The walls check for collision by checking the distance between them and the ball. If the distance is less than the sum of the wall's thickness and ball's radius, then a collision happens. It handles the collision by changing the ball's velocity, which reflects along contact normal with restitutions. The normal of border walls are trivial, but the Slant Walls are a bit trickier. The normal could be found by first getting the direction of a SWall from its start and end points, and in addition, it needs to check if the ball is hitting within its line segment. The normals of SWalls are checked and guaranteed to point inward the board.

3. Paddles (PaddleCollider.cs)

Paddles are rotated up to 45 degrees when a or d keys are hit. At rest, they are at -10 degrees. The paddle colliders could be simulated by triangles. When writing paddles, I did not know that I could get the 3 vertices from the mesh, so I used the base, height, and pivot point of the paddle to calculate them. Then, 3 sides are constructed from vertices in counterclockwise direction. Normal is then calculated for each side. To avoid weird behaviors for collision close to tip, normal is also calculated for each vertex, set to be the sum of normals of neighbouring sides. To check for collision, each side and each vertex are looped through, and the one with the minimum penetration is used. To handle collision, restitution is adjusted based on relative

velocity between the side/vertex and the ball when in contact. Paddle motion uses angular velocity (ω) to form a surface velocity v_surf = ω × r. The left and right paddles will multiply by negative of each other to correctly handle the symmetry.

4. Obstacles (TriangularPrismCollider.cs, CylinderCollider.cs, GenerateObjects.cs)

The Triangular Prism is simplified to a 2D triangular collider. First, we check the convex hull to get the triangle with the maximum area to prevent extracting vertices of rectangular sides. Then, we calculate the correct normals for each side, and make sure the normals are pointing “out” of the shape by ensuring counter clockwise direction. The edge with minimum penetration is chosen, and collision is applied in a similar manner as other colliders, except providing an additional velocity reduction.

The Cylinder obstacle is simplified to be a circular collider. Distance between centres of the ball and the obstacle is used to check collision, and if the distance is less than the sum of radius, a collision happens. Handling collision is done in a similar manner as other colliders, except for providing an additional velocity boost.

GenerateObjects will generate 4 Triangular Prisms and 3 Cylinders at the start of the game. It is given an area to select position from, and will loop to find a valid position for each object to generate. Overlapping is checked by estimating the possible spanning area in a circle from radius got from meshes of the shapes, as well as adding distance between obstacles and distance from walls to prevent the pinball stuck during play. The Z rotation of prisms is limited to between 10-80 degrees, to prevent being parallel from the border’s horizontal/vertical axis.

Limitation:

- The simulation is reduced to 2D. No rolling is implemented, Y values are locked.
- Fallback of spawning of ball at (0,1,0) is probably not the best pick, as this position is possible to be occupied by obstacles.