OOP Project#3 Report: 4 Balls 3D Billiard Game

[Team 13]

<Contents>

1. Brief project description

This project is a simple 3D crash game simulation developed using DirectX 9. Four spheres and walls interact to simulate physical collisions, and the user controls a specific ball to proceed with the game. We made billiard game with 4 balls. We additionally implemented three functions, “Start” button, using white ball and yellow ball alternately, and fixing the billiard board using backspace.

2. How to compile and execute

Run Visual Studio 2022 on the Window OS(Window 10) and click the run button or press f5 key.

3. Description on functionality

1) “START” button

When the code is executed, a “START” button with a notice below appears informing the player that by pressing the esc key, the game will end and it will start only when you press the "START" button.

2) Using white and yellow ball alternately

During the game, white ball and yellow ball are automatically used alternately To indicate each player’s turn.

3) Press the ‘Backspace’ key to fix the billiard table

It was uncomfortable because the billiard table kept moving in the process of aiming with the blue sphere in the example code. So we implemented a function so that the billiard table could be fixed by pressing the ‘Backspace’ key.

4) CSphere :: hasIntersected()

Determines whether a collision occurred between the current sphere(ball) and another sphere.

5) CSphere :: hitBy()

When two spheres collide, this function calculates and reflects the velocity after the collision.

6) CWall :: hasIntersected()

Determines whether a collision occurred between the wall and the sphere.

7) CWall :: hitBy()

When a sphere collides with a wall, this function calculates and reflects the velocity after the impact.

4. How to implement important implementation issues

1) “START” button

‘isGameStarted’ indicates whether the game has started or not, and it is initialized to false to show the start button. ‘startg_Font’ and ‘infog_Font’ are a font object for Direct3D text rendering, startHr and infoHr is created using D3DXCreateFont(). ‘startButtonRect’ and ‘infoRect’ are the RECT structures that indicate the position of the button and information. In Display(), if the game has not started (isGameStarted == false), draw the "START" button and the "press ESC to exit" notice on the screen. The button and information are drawn using ‘startg_Font->DrawText()’, ‘infog_Font->DrawText()’. In WndProc(), checks whether the mouse click occurs inside the startButtonRect, if true, set ‘isGameStarted’ and ‘worldMove’ to true to start the game. When the “ESC” key is pressed before starting the game, exit the program.

2) Use balls alternately

The ‘isTargetYellow’ variable indicates the turn of the white ball if false and the yellow ball if true. In WndProc(), each time the ‘SPACE’ key is pressed, reverse the ‘isTargetYellow’ value so that the ball moves alternately. Compare the position of the blue sphere with the current position of the white/yellow ball to calculate the angle and the speed, and move it based on this information.

3) Press the “Backspace” key to fix the billiard table.

In WndProc(), when the user presses the ‘Backspace’ key, it reverses the ‘worldMove’ variable. If ‘worldMove’ is true, world movement is enabled, and if false, world movement is disabled.

4) CSphere :: hasIntersected()

To get the center coordinate, call ‘this->getCenter()’ and ‘ball.getCenter()’. By storing the difference between the X and Z axes of the two spheres in dx and dz respectively,

calculate the distance between the two points using the Pythagorean theorem(distance = sqrt(dx * dx + dz * dz)). If the distance is less than or equal to the sum of the radius of the two spheres, the two spheres are considered to have collided.

5) CSphere :: hitBy()

Using getCenter() and getVelocity(), we can obtain the center coordinates of the two spheres (myCenter and otherCenter) and their respective velocities (myVelocity, otherVelocity). To calculate the direction of the collision, calculate the direction vector between the two spheres and store it in 'collisionNormal'. This vector represents the direction of the collision between two spheres and is used to handle it. Call 'hasIntersected()' to see if the collision occurred, if true, normalize the 'collisionNormal' and calculate the relative speed of the two spheres. If velocityAlongNormal(which is a scalar projection of relative velocity between the two balls onto collisionNormal using dot product) is bigger than 0, the balls are considered to be moving apart, so the collision response is skipped. Impulse magnitude is obtained using rebound coefficient(e) to calculate the velocity after the collision, and velocities of both balls are updated.

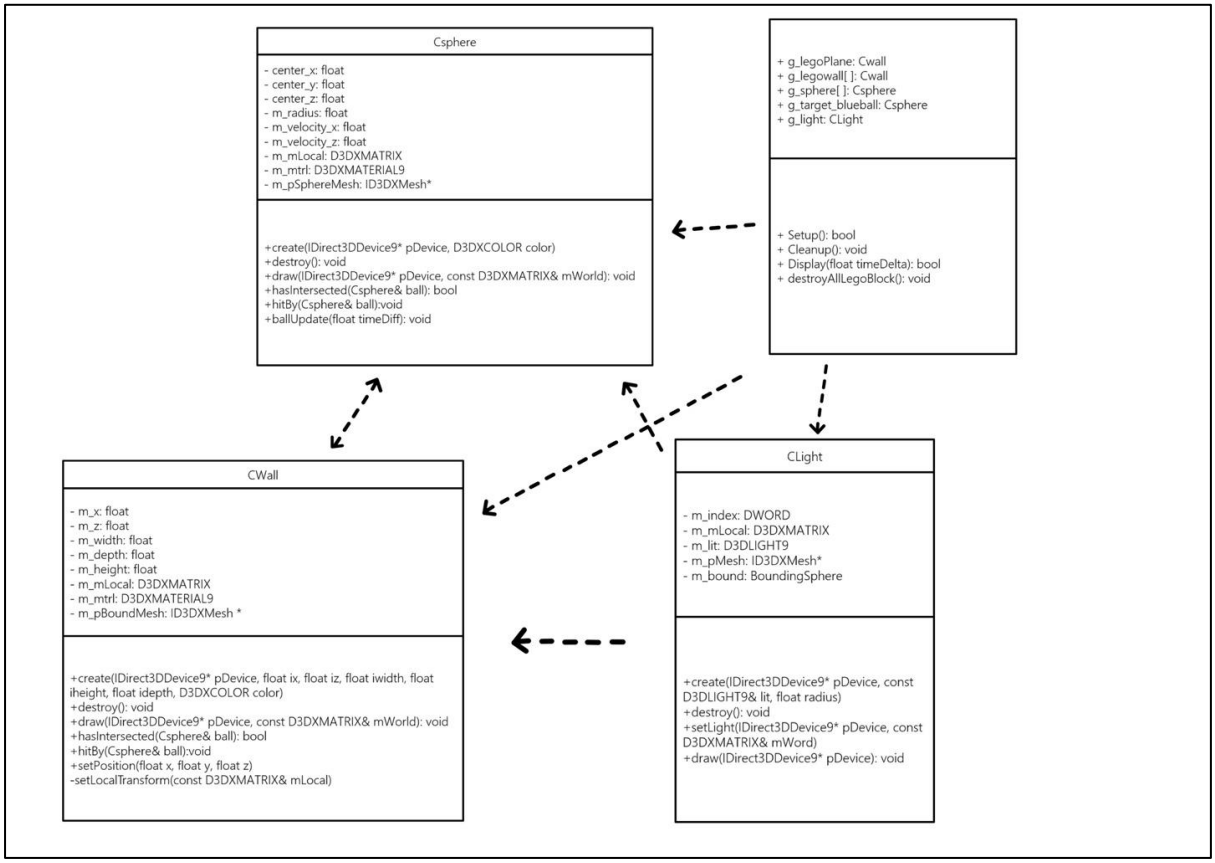
6) CWall :: hasIntersected()

Call 'ball.getCenter()' and 'ball.getRadius'. Calculate the boundary of the wall using the center position, width, and depth of the wall. Consider the radius of the sphere to ensure that the sphere is within the boundaries of the wall. This function determines whether the sphere and wall overlap on the X- and Z-axes, if true, they are considered to have collided.

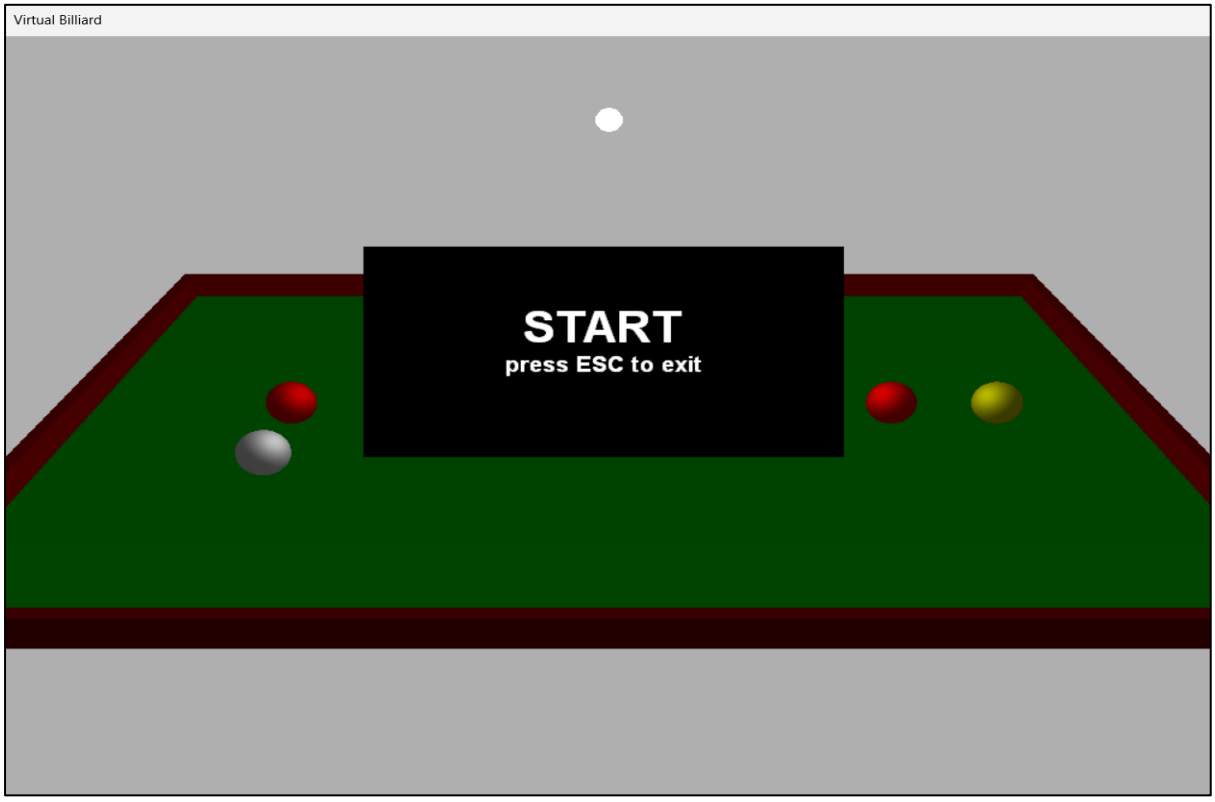
7) CWall :: hitBy()

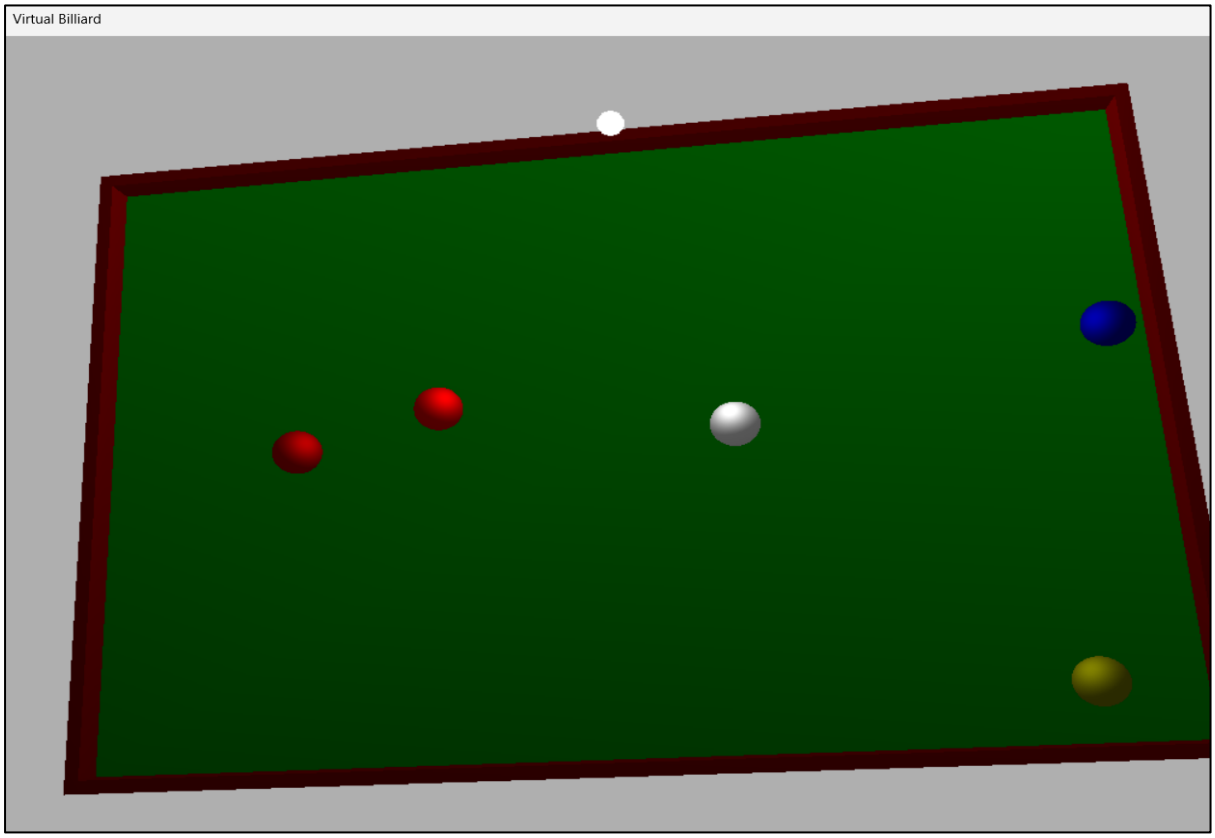
Gets the 'ballCenter' and 'ballRadius' of the sphere, and obtains the 'ballVelocity'. Calculate the boundaries of the wall based on the center and size of the wall. Call 'hasIntersected()' to see if the collision occurred, If true, set the collision direction vector(collisionNormal) according to which direction the collision occurred. By using dot product, calculate the scalar velocity component in the direction of the collision normal. Impulse magnitude is calculated using the coefficient of restitution (e), and the impulse vector is derived from the impulse magnitude and collision normal. Finally, update the new velocity of the sphere, and adjust the position of the sphere so that it does not cross the wall.

5. The result of SW system design



6. Execution results





7. Applying objected oriented concepts

In CSphere, and CWall class we implemented ‘hasIntersected()’ and ‘hitBy()’ relatively so that when the ball collides, it follows the physical laws of the real world. Using ‘this’ in CSphere to represent the current sphere object calling the function, while ball is other sphere involved in the interaction. These classes manages its data(center, radius, velocity) and provides controlled access through methods like getCenter, getRadius, getVelocity .etc (Encapsulation). ‘hasIntersected()’ and ‘hitBy()’ separates responsibilities of detecting and resolving collision. This enhances the code’s readability and maintainability,.

We created global variables(startg_Font, infog_Font, startButtonRect, infoRect) to make it usable throughout the code and functions. And created font for each purpose relatively: start button and information about exit.

In Display function, class objects and their functions of CSphere and Cwall are used to execute the game.

8. What we learned and Conclusion

Through this project, we could actually feel not only theoretically but also in the aspect of readability and scalability of why it is good to embody elements using classes, create and manage them as objects using object-oriented concepts. And it was interesting and convenient to allow objects to perform interrelated functions through interaction using interclass methods. In addition, we learned for the first time that there is a program called ‘DirectX 9’, which allows us to draw graphics and make games on Windows OS. Unlike the existing tasks, it was difficult to consider visual elements, but we were able to learn new things. In this project, we have additionally implemented functions such as starting buttons, alternating balls, and fixing billiard tables. If given more time, we would like to further implement the score mark, such as scoring if the billiard ball hits both red balls, and losing points if it hits the opponent's ball or hits nothing, and developing other visual elements such as billiard cue.