Throw A Ball With More Functionalities

Baorong Wei & Sohail Hasan Shaik

I. Introduction and Motivation

In our project, Throw a Ball, we experimented with physics-based animation to simulate inbound dynamic motion in Unity. In particular, we have animated two cases: Throwing a ball on bricks and a Ball collision with the cloth.
This project aims to demonstrate the use of principles of physics to create realistic and interactive animations. We will leverage techniques like collision detection, rigid body dynamics, and cloth simulation based on Verlet integration to increase the realism and life of virtual environments commonly found in games and simulations.
It's inspired by games such as Bowling, Billiards, and Angry Birds (if there'll be a 3D version later), where interactions between objects are dynamic and interesting. Additionally, we sought to deepen our understanding of Unity's physics engine and how it replicates real-world physics in virtual environments.
We also add more functionalities, like the ball can bounce back from the floor, like a basketball; the bricks can now be destroyed into pieces, instead of just being pushed to the ground; the ball can have a tracking function, like a homing missile, and it can hit the target wherever it is while avoiding the obstacles.

II. Related Work

Since neither of us has much experience building a game, we used some online tutorials to make this game from scratch. We got the idea of throwing an object from some discussion boards [1]&[2] and also learnt about AddForce [3]. Further inspiration for shattering effects came from research [4], which discusses using nonlinear finite element analysis to simulate the fracture and destruction of solid objects in animations. We got to know about how this effect is made, but for building it in Unity, we utilized the RayFire plugin, which provides robust functionality for runtime demolition and fragmentation of 3D objects. RayFire is widely recognized for its ability to simulate realistic destruction effects, seamlessly integrating with Unity's physics engine. The plugin's features have been explored in user-driven tutorials and guides, which helped us implement visually compelling destruction mechanics and cloth dynamics.

III. Method and Implementation

At the outset, we analyzed the core requirements of the project. The primary goal was to simulate the physical behaviour of a thrown ball and its interactions with static objects, such as bricks. From this foundation, we planned to enhance the project by introducing features that would improve interactivity and visual realism. These included adding cloth simulation, enabling the ball to bounce, making it capable of tracking a target dynamically and shattering the bricks upon impact. The challenge was to design these features in a way that seamlessly integrated into the game environment without compromising performance or user experience.
The foundation of the simulation involved animating the motion of a ball thrown at a set of bricks. We used Unity's Rigidbody component to apply physics-based motion to the ball and bricks. Colliders were added to detect and manage collisions, ensuring the ball interacted realistically with the environment. Mouse events were used to implement throwing

mechanics, where the ball’s trajectory and force were calculated based on user input. Afterwards, we made our modifications based on this base model.

Sohail – Used Bricks as Rigidbodies and colliders using built-in unity features. Made the cloth (Pink color) detect collisions using unity in-built physics such as cloth, mesh renderer, and sphere colliders. Made the ball interactive using the BallThrower class. Which manages the ball's movement based on user input. It calculates throw speed, and direction, and simulates the ball's physics, including gravity and collision. Start() → Calls setupBall() to initialize the ball object and its properties. setupBall() → Finds the ball using the "Player" tag, sets up its Rigidbody for physics, and initializes default values with ResetBall(). ResetBall() → Resets the ball's position, stops its movement, disables gravity, and prepares it for the next interaction. PickupBall() → Allows the player to pick up and move the ball with the mouse, converting screen space to world space and smoothly following the cursor using Lerp. CalAngle() → Determines the throw angle by normalizing the swipe direction and incorporating the camera's movement. If the swipe meets the criteria (distance > 30px, time < 0.5s), it triggers the throw. Update() → Handles the main input and throwing logic. CalSpeed(): Calculates the ball's velocity using swipe distance and time, then clamps the speed to a maximum value for balance. I also made a dynamic cloth (Blue Color) using Verlet to figure out which works better (Verlet or Unity dynamics). Used ClothSimulator Class to write the logic. InitializeParticles() → Creates a grid of particles spaced by width, height, and spacing, with fixed particles on the edges (y == 0 and x == 0 or x == width - 1). InitializeConstraints() → Creates horizontal and vertical constraints between adjacent particles to maintain their relative distances in the grid. AddConstraint() → Adds a constraint between two particles by creating a Constraint object that stores their rest length and the method to satisfy the constraint. VerletIntegrate() → Updates each particle's position using Verlet integration with gravity and damping, while adjusting the previous position for non-fixed particles. ResolveSphereCollision() → Detects and resolves collisions between a particle and a sphere by adjusting the particle's position and velocity if the distance is less than the sphere's radius. CreateMesh() → Generates a mesh for the cloth simulation by defining vertices from particle positions and creating triangles between adjacent particles. UpdateMesh() → Updates the mesh vertices to reflect the cloth simulation's progress by applying the current particle positions. Implementing all these resulted in a dynamic collision of the ball on the cloth.

Baorong – My idea is to go beyond just throwing a ball. We could add more functionalities that can make this more realistic, like bouncing for the ball. We could make the ball more powerful so that it can hit the bricks and shatter them into pieces. We could also add tracking functions to the ball, like homing missiles in Sci-Fi games, which means the ball can always hit the target once thrown. One brick would be the target and the rest will be obstacles that simulate buildings in the scene. The target will be spawned randomly each time the player clicks “play” and it will move randomly in all three axes. The ball will automatically follow a smooth curve line and hit the target no matter where it is while avoiding the obstacles. However, just like a homing missile, if the target is too close to an obstacle, it will still hit the obstacle.
To simulate bouncing, the ball was assigned a Physics Material with appropriate bounciness settings. This enabled the ball to rebound off surfaces like the floor, adding a layer of interaction and realism to the simulation.

The destruction of bricks was implemented using the RayFire tool. In the `Block.cs` script, we assigned each brick a RayFire rigid body, enabling them to fragment dynamically upon impact. Randomized parameters for fragmentation density and count were used to create unique shattering effects for each collision. The `Bullet.cs` script handled the collision detection and triggered the `Demolish()` method on the impacted bricks, ensuring a visually engaging destruction effect.
Tracking was one of the more challenging functionalities to implement. In the `Missle.cs` script, we calculated the direction vector from the missile to its target and used `Quaternion.Slerp` to rotate the missile smoothly toward the target. To ensure continuous tracking, the missile's position was updated incrementally along the forward vector. If the missile reached the target, it applied a physical force to simulate an impact, adding another layer of realism to the interaction.
Obstacle navigation was handled primarily in the `Target.cs` script. Before moving to a new position, the target used Unity's Physics.BoxCast to detect potential obstacles in its path. By ensuring that the chosen path was clear, the target avoided collisions with static objects. This design indirectly guided the missile along an obstacle-free trajectory, simplifying the implementation of the missile's navigation.

IV. Results and Evaluation

Sohail – Bricks fall realistically as rigid bodies after collision. The Unity Cloth also detects collision once the ball is thrown at it. The Verlet Cloth flows dynamically but is a bit glitchy in detecting collisions. The ball is user-interactive with speed adjustments and detects a collision with other objects.
Code Link – CodeDemo-SohailHasanShaik
Demo Link – https://youtu.be/aEVTe9n-bgc

Baorong – The final implementation achieved the desired objectives. The simulation successfully depicted dynamic interactions, such as realistic ball motion, shattering bricks, and target tracking. The shattering effects, powered by RayFire, added a visually compelling element to the simulation. The tracking and obstacle avoidance mechanisms worked seamlessly, ensuring the missile reliably reached its target. Performance remained stable, even with complex interactions, demonstrating the efficiency of the implementation. These results underscore the effectiveness of Unity and RayFire in simulating real-world physics in a virtual environment. There are two scenes. The first is simple with just the ball hitting the bricks and the bricks shatter. The second scene is more complicated with the tracking function and many more obstacles. After clicking the "play" button, these two scenes can be easily switched by pressing "space", and the player can also reset the scene by pressing "R".
Link for the codes with README file: Baorong.zip
Demo Link – https://youtu.be/ACDKaYxJocY

V. Acknowledgements and Contribution

Sohail – Bricks using inbuilt unity features + Ball Thrower script + Cloth Simulation script (Verlet) + Inbuilt cloth Simulation using physics.
Baorong – Add more functionalities to the base simulation. Add bouncing, shattering, tracking and some other techniques to make the gameplay smoother.

Special thanks to the research community, discussion boards and online tutorials which played a crucial role in providing insights and techniques for implementing advanced animation and physics.

VI. References

[1] How to throw a ball - Unity Engine

[2] Throwing Objects - Unity Engine

[3] Throwing ball using AddForce - Unity Engine

[4] James F. O'Brien, Jessica K. Hodgins, *"Animating Fracture"*, arXiv link.

[5]How to throw a ball to a specific point on a plane? - Stack Overflow