Ball Soccer - COS 426 Final Project

Benjamin Coles (bcoles), Martin Hito (mhito)

May 2021

1 Abstract

For this project we planned to implement a physics based, two-player soccer game. The project features a loading screen, scoring system, object based scenery, various physics based game mechanics, and a replay system which allows users to swiftly start a new game. The premise of the game is simple: score three goals before your opponent. Easy to learn, hard to master. Emphasis on the physics.

2 Introduction

3 Approach Methodology and Iterative Design

At the core of our project was the physical interactions between the players, ball and the surface area. No matter how sophisticated our intermediary screens, scenery and game-play mechanics were, the core component of our game which would ultimately make it exciting to play were the physical mechanics; if we could get this right, we knew we would have an enjoyable, re-playable game.

Our starting point was to base all interactions on real-world physics. We planned to encode interactions between walls, balls and players as realistically as possible, with the hope that this would provide us with interesting game-play. This meant, we provided all elements of the game with implicit or explicit physical properties such as mass and velocity, and we gave the walls of the arena friction coefficients that determined the direction in which balls would bounce off.

3.1 World Physics

Since all of our players and balls were spheres, it was fairly easy to detect a collision, just by checking if the central position of each sphere less than the combined radius away from the other. Then, we would obtain the vector the connecting the center of the two spheres, and project the velocity vector of the

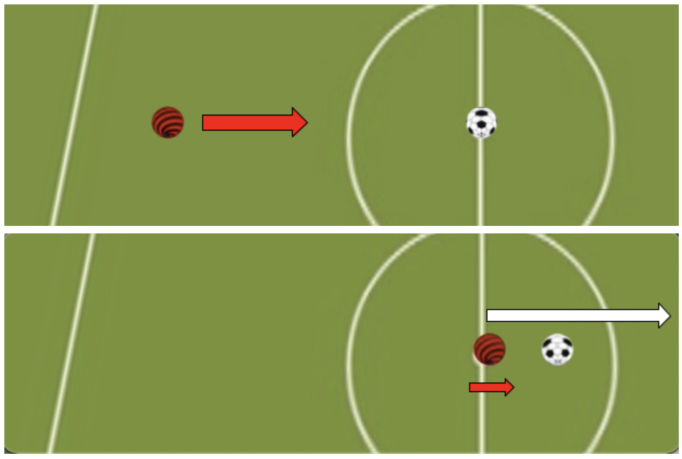

Figure 1: Collision between one of the players and the ball. The player comes in at velocity x, the ball moves away perfectly elastically at velocity x, and the ball continues in the same direction with velocity x/4.

two balls onto the line connecting to two spheres. This would allow us to calculate the momentum transfer between the two objects, and we would apply this momentum transfer in the direction of the vector that connected the two centers. By conserving momentum in collisions, we could also introduce the mass of each object very simply through the equation $p = mv$. After play testing, we found that giving the ball a mass that was 0.25 the mass of each player gave the most satisfying results. This basic collision system would be something that we kept constant throughout the development of our game, from the MVP to the final product. Our interactions with walls, however, went through an extensive amount of iteration. To simplify the idea, we first split the velocity of the sphere into the component that was perpendicular to the wall and the component that was parallel to the wall. The perpendicular component would need to be inverted and scaled such that we could allow a 'bounce' of the wall to happen. We found that simply multiplying this perpendicular component by -0.5 gave us the most realistic, haptic results.

The perpendicular component of velocity was much harder to handle. Initially, we explicitly encoded a friction parameter associated with each wall, and would apply an impulse to the ball according to this parameter and the magnitude of the perpendicular component of velocity. This meant that collisions with the wall happened as the would in a game like snooker or pool, in a way that reflected physical reality.

However, when we gave the game to our peers to play test, the most common complaint was that bounces off the wall seemed unpredictable. This unpre-

dictability complaint was initially surprising to us since we thought we had successfully encoded physics that reflected real world bounces. It turns out that after a collision, players expected the ball to bounce off at the same angle that it came in. We tested simply reducing the parallel component of velocity by half, as in the perpendicular case, such that the ball would bounce away at the same angle, just with a smaller velocity magnitude. This, despite being more simplistic and less physically accurate, gave us wall collisions that felt much more satisfying, predictable, and realistic than the physically 'correct' alternative.

Another core physical idea we had to encode was the interaction with and the friction provided by the ground. We implicitly gave the floor a friction coefficient such that at each time step the magnitude of velocity would reduce by a factor of 0.99. This gave a result where the ball and players would slowly come to a stop if no impulse was provided. This was an area which we can almost unanimous agreement from the play tests. Putting the coefficient any higher meant it felt like you were pushing, and not kicking the ball. Putting it any lower meant you could rarely catch up with the ball before it hit a wall. This compromise meant depending on your speed, you can either dribble or kick the ball. This decision, although simple, is ultimately what makes the game strategic and fun.

3.2 Movement System

In our MVP product, we initially designed the movement system such that pressing the arrow keys lead to an immediate change in position. Although this was the most obvious solution, we knew immediately that it simply felt wrong. Furthermore, this idea didn't bode well with our collision mechanics since it meant collisions could be immediately overcome by movement with the arrow keys.

As a result, we swiftly changed the movement system such that pressing the arrow keys lead to an immediate change in direction, instead of position. This gave the balls a sense of momentum, which also made some of the collision physics really natural to implement.

Our emphasis on real world physics made our implementation more simplistic since we already had established idea of what each idea represented. We could modularize each of our player and ball objects and give them implicit or explicit parameters than defined thier interactions. This not only made our implementation much cleaner and modularized, but it made the game very easy and intuitive to learn.

One thing we might like to touch on is to make a mechanical distinction between dribbling and shooting. With our current implementation, you can achieve both by hitting the ball at different speeds. We believe that by adding a hit mechanic,

we can make that distinction mechanically, such that hitting the ball acts as a dribble, but 'kicking' it can act like shooting. This may also lead to a higher skill ceiling for the game.

3.3 Scenery and Visuals

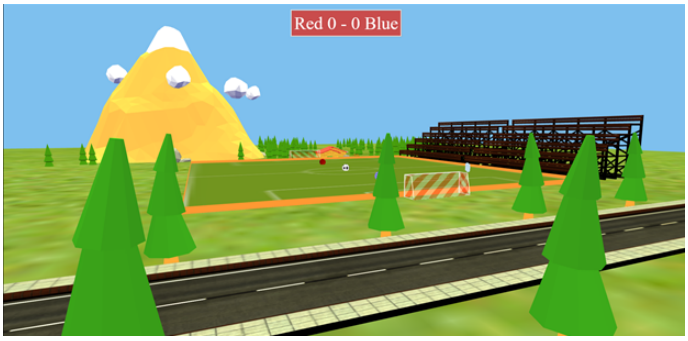

Figure 2: One angle of our scene. Here you can see the rod, which has a car drive on it every 40 seconds, the trees, mountain, house, and stands.

Initially, we decided to make our players and the ball a simple two dimensional disk, viewed directly from above. This felt a bit cartoon like, and whilst it was still fun to play, we were told by play-testers that it did not feel realistic.

We replaced all entities with spheres, and also added a visual roll to the balls. This pace of the roll was is simply proportional to the speed of the balls. We also added a striped texture to each ball such to make this rolling effect more visible.

We also added a variety of .gltf objects and image mappings to make the scene more appealing. These included the bleachers, the mountain, clouds, trees, the road, the goals, the field, and more. Whilst this doesn't add to much to the game-play itself, it definitely makes the experience of playing for more enjoyable and makes the game feel much more complete. These were all implemented and brought into the game in a similar way to the flower and land objects in the example project.

3.4 Additional Mechanics

A total of 7 power ups have been implemented into the game as of writing this paper. Power ups randomly spawn into the game 1 at a time, and only one power up can be on the field or affecting the scene at once. Power up generation is handled by a simple random number generator which generates a random integer between 1 and $FREQ$, inclusive. $FREQ$ is a variable related

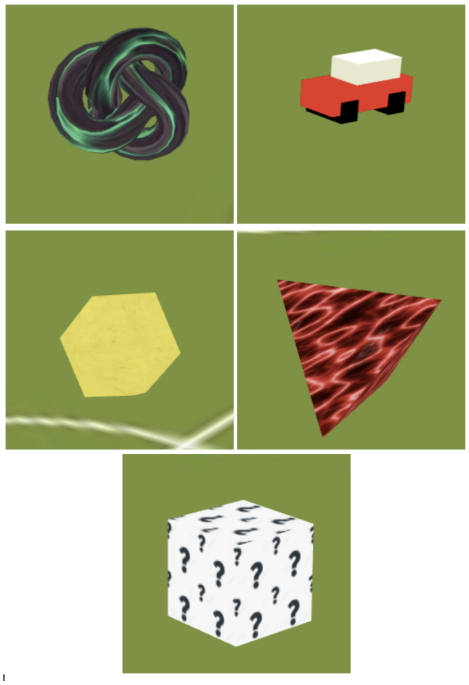

Figure 3: From top to bottom, left to right: Teleport power up icon, Speed up/slow down power up icon, Big Goal power up icon, Big Hit power up icon, Mystery power up icon.

to the frequency of power up spawning. Decreasing $FREQ$ allows for power ups to spawn more rapidly when there isn't one on the field, and increasing it has the opposite effect. Power ups are placed randomly on the field, anywhere greater than 10 units away from the edges to avoid clipping. While on the field, power ups spin in place and bob up and down sinusoidally to give a nice visual effect.

Once a player comes close enough to a power up to obtain it, it is removed from the scene, and the effects of the power up begin immediately. All power ups last for $TIME$ frames, which can be changed.

Initially, powerups were added as objects in the same way that our player, goal, stand, and other scenery objects were added. However, when the rotation and bobbing effects were added, the objects rotated around the global Y axis around the origin, so they did not spin in place. In order to fix this all power up generation was handled within our SeedScene.js file, and their effects were coded directly into the scene's update function. The effects and implementations of our power ups are described in the sections below.

3.4.1 Teleport Power Up

The teleport power up works by immediately switching the positions of the red and blue players upon pick up, regardless of which player obtained the power up. The teleport power up is the only one which does not last for a specified amount of time, because it is instantaneous. Power up regeneration begins immediately after the power up is picked up and the player's positions are swapped. The mesh used for the power up is a TorusKnot geometry.

3.4.2 Speed Change Power Ups

There are 2 speed-changing power ups in the game, one which increases the speed of the holder of the power up, the second decreases the speed of the holder of the power up. This was implemented by passing a number to the update function in our player object implementation. In the update function a check is done, and depending on the number passed, the ball either moves double the distance each time step or half the distance. The mesh used for the power up is a custom low-poly car icon. Because the speed up and slow down power ups both have the same icon, getting this power up is risky to the player, but potentially very rewarding.

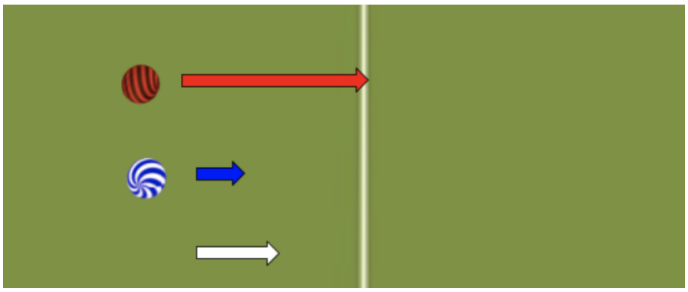

Figure 4: White arrow on bottom represents the normal max velocity of the player. The red arrow represents the max velocity when holding the speed up power up, the blue arrow represents the max velocity when holding the slow down power up.

3.4.3 Big Goal Power Ups

There are 2 big-goal power ups in the game. The first changes the size of the opponent's goal (the goal you are trying to score on) to make it twice as big. The second does the opposite, and changes the size of your own goal (where you opponent is trying to score) to be twice as big. This is implemented by removing from the scene the Goal objects and re adding new objects depending on which goal is increasing in size. The checker in our scene's update function that

checks whether the position of the ball is within the goal is changed depending on if one of these power ups are active, so that a score can still be counted on the larger goals. The mesh used for the power up is an Icosahedron geometry.

Currently, there is a slight delay between the goal's being removed and re added. This can potentially be solved by having a lower-poly version of the goal object, so that the object can be placed into the scene more quickly.

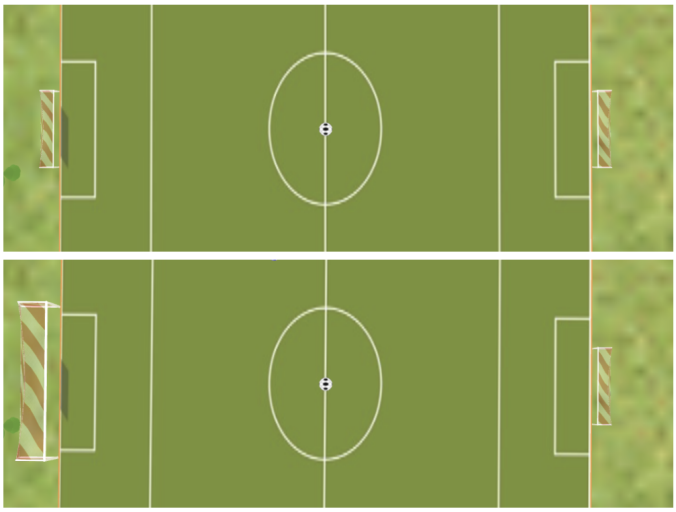

Figure 5: Top, normal goal sizes. Bottom, the goal that blue wants to score on is increased in size.

3.4.4 Big Hit Power Up

The big hit power up makes it so that the effects of any collision between the holder of the power up and either the ball or the other player is increased in magnitude by 3. This is implemented by adding a check for the power up in the collision handlers, and updating the offset due to a bounce accordingly. The mesh used for the power up is a Tetrahedron geometry. NOTE: The figure for this power up is at the beginning of the next page (figure 6).

3.4.5 Mystery Power Up

The mystery power up randomly selects one of the other power ups to be the active one. This is done with a simple random number generator. The mesh used for the power up is a Cube.

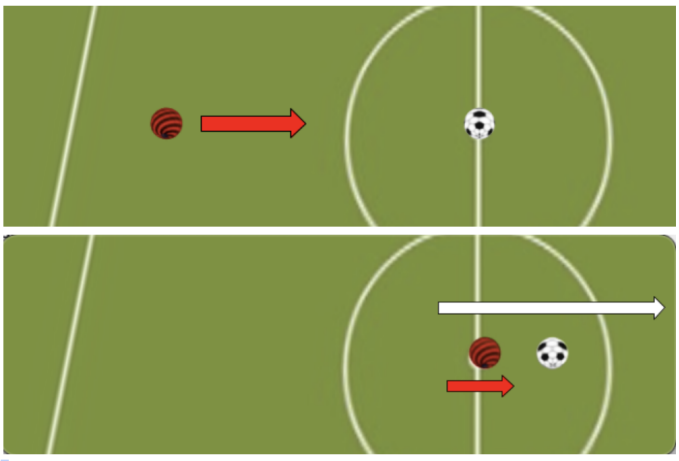

Figure 6: Hitting the ball if red has obtained the Big Hit power up. the white vector represents the movement of the ball. The size of the vectors correspond with the speed of the object as it moves.

4 Feedback

When we were asking our peers for feedback we were looking for feedback on one specific piece of design implementation as well as general feedback on the game. We did this by asking them to play two versions of the game with us; one had one implementation and the other had a different implementation of a feature. For example, one test we tried was with the style of wall interaction we had, and another was with the friction coefficient for the floor. This meant we could get general feedback from the player on the game as a whole, in addition to a specific opinion on how exactly changing how we implement one feature impacts the game. We switched to this model of testing so we could really localize our feedback on troublesome areas and focus less on sweeping statements about how 'good' or 'fun' the game is.

The specificity in our testing really helped us to fine tune some of the physical parameters of our game. Offering the player two different implementations meant the tester could understand what the options were and give us feedback on the benefits and drawbacks of both implementation. We were careful not to tell the testers exactly what these changes were so they weren't biased by knowledge of the implementation like we were. We wanted their opinion based purely on how fun and entertaining the final product was, and worked back from that point to inform or implementation. This was one of the biggest successes of our method of implementation.

4.1 What We Learned, Conclusion, and Next Steps

One of the most important things we learnt was how to progressively improve and iterate on a product. Most of our previous Princeton coursework has had a distinct goal in mind where we need to take a series of given steps to get to that point. In this project however, we had an endless stream of potential ideas and products, and we were only bounded by time.

The structure of our project was useful for this. Our minimum viable product was fairly simple to create (discs on a plane with naive physical interactions). However, from that point onward there were lots of directions in which we could iteratively improve our design.
We quickly learned the importance of modularity. Encoding something as an instance variable of an object, instead of having this variable implicitly exist across numerous chunks of code, meant we could swiftly change and test different values and examine how that impacted the game. Furthermore, it made collaboration really easy as we could swiftly see what changes the other had made and what impact they had on the project as a whole. This investment in modularity at the start paid off many times over as we iteratively improved the game.

We also learned how to prioritize features. Since the task was so open ended, we had to make sacrifices in what we choose to pursuit. This in turn meant we had actively decide what we valued in the game and have constantly changing image of the final product in our heads. I felt we found a good compromise between following the vision we had at the beginning, whilst also adapting to the problems we came into along the way.

Overall, we feel that we met our goal of creating a game that is genuinely fun to play. Our greatest success was fulfilling the 'easy to learn, hard to master' goal that we aimed for from the outset. We based our game on a solid base of engaging and realistic physical interactions, and fleshed out scenery, game play mechanics and menu-screens effectively from this baseline.

Our next steps would be to make more use of 3 dimensions. Whilst all our objects, players and balls are three-dimensional, most of the game play is limit to the 2D plane. Addition of mechanics like jumping, and an increased ball size would allow us to bring a whole new dimension into the game and give the players even more mechanics to master.

5 Contribution

With the modularity of our code, it was really easy to delegate tasks without interfering with each other's work. Here is what we both worked on.

Ben: Scenery, Core Physics, Loading Screens.

Martin: Powerup mechanics/visuals.

We both focused on what we worked on in the writeup.

6 Works Cited

We took some inspiration from the online flash game HaxBall and the online console/PC game Rocket League.

We took three pieces of code from the COS426 2020 Project PrinceTron Legacy. The first was the creation of a gradient box and box walls, which they themselves took from http://darrendev.blogspot.com/2016/03/gradients-in-threejs.html.

Additionally, we also started our menu/final screen code from the PrinceTron Legacy project, although the final code didn't end up looking too similar because of changes we made along the way.

Finally, we took the starter code from the course-site to give our code structure. We also learnt how to import gltf files from the example files Land and Flower.

We didn't use any additional libraries beyond those in the starter code.