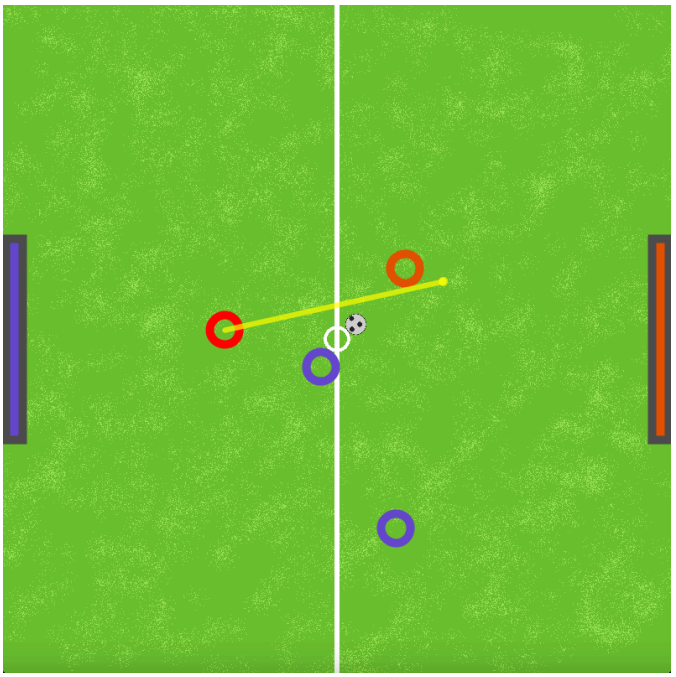
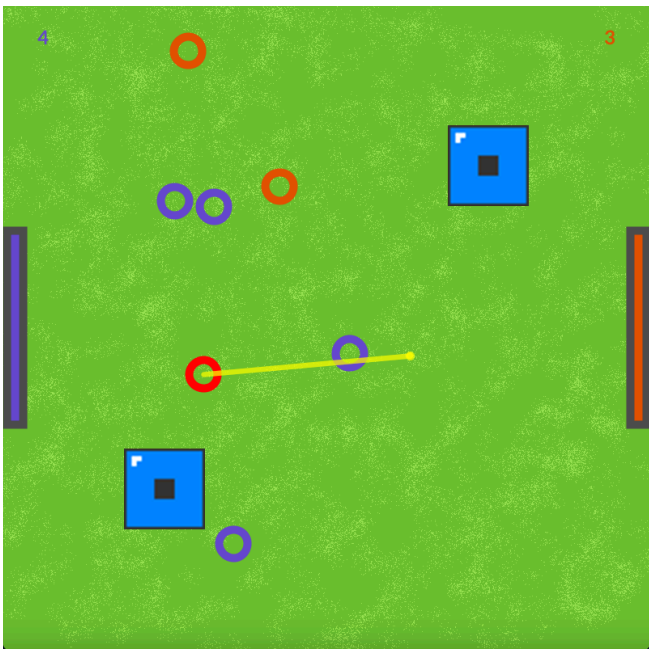
Hoop Sports is a physics-based team sports game played by bumping players or the ball. The user controls a single player, but can swap to another player on their team by clicking them. Players move in short bursts of speed, referred to as “jumps”. The user can jump anywhere by clicking the location they wish to jump toward.

Hoop Sports features three game modes: football, bumpers, and dodgeball.



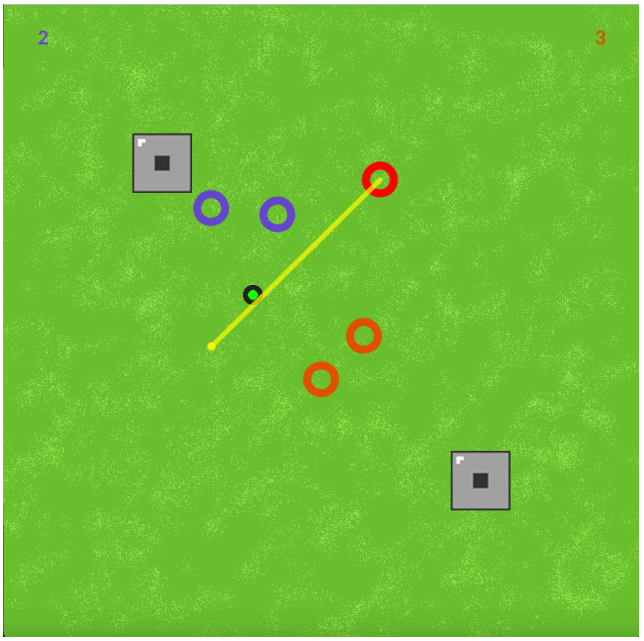
Football

Football plays as expected: the first team to get the ball into the other team’s goal wins. AI players will bump the ball toward the enemy goal and rush back to defend the home goal when the ball passes them. They will also try to keep the ball from getting trapped in a corner by sending a single player to retrieve it.




Bumpers

In bumpers, there is no ball. The goal is to bump enemies into your goal to eliminate them. The last team standing wins. AI players will bump into players on the opposite team, interfering with their movement and bumping them toward the goal. This game mode's map sometimes has two to four bouncy objects.




Dodgeball

The objective in dodgeball is to bump the dodgeball into enemy players to eliminate them while avoiding the ball yourself. The ball can safely be hit when not moving, as indicated by its colour. The last team standing wins. AI players will hit the ball if it is safe to and hit other players if not. This game mode's map sometimes has a couple objects on the map that colliding objects slide along.

From the main menu, the user can start the game, change the game mode, read tips and tutorials, and open the settings menu. The settings menu saves your settings on returning to the menu and contains three settings.

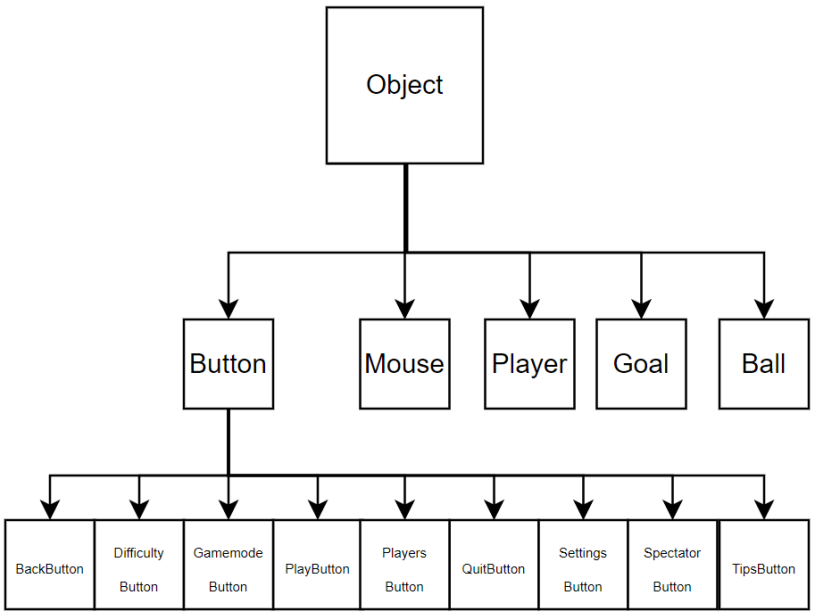

Settings

The first setting is the difficulty level. It affects the AI players and ranges from one to five. Raising the difficulty level makes AI players jump farther and more often. This also increases their accuracy, meaning they will hit their target more directly. Games will be faster-paced and tougher at higher difficulty levels.

The next setting is the player count, which can be set to a few different levels from two to 16 players. At higher player counts, players will be smaller in-game to help prevent

players from getting stuck. The game is most fun at 8 or less players, while larger numbers are usually more fun to watch play in spectate mode.

The final setting toggles spectate mode. With spectate mode enabled, the user does not control any players and watches the AI players play. This can be used to learn how the AI players behave or just to watch matches. The player can choose to play at any time by clicking a player on the red team.




Class Hierarchy

Before making the actual game, I implemented basic collision and physics in the Object class. This is the base class for all other classes. It manages location, size, targeting, collision attributes and methods, and applying forces, among other various things. It handles collision by checking where every moving Object wants to go, iterating to its final location one pixel at a time for higher accuracy. If the location to check is occupied by another Object with collision enabled, the moving object will either bounce off, slide along, stick to, or repel from its surface, depending on the Object's surface type. If a moving Object collides with another moving Object, the two will trade directions and velocities.

Collision is mainly detected based on two Object attributes: collType and collShape. collType is normally BLOCK, meaning the Object will not overlap with other Objects of collType BLOCK. When two Objects check collision, the collision bounds are determined by collShape. Both Objects will be treated as having rectangular collision if either Object has collType RECT. They will check inside their x and y boundaries for the other Object. However, if both Objects have collType CIRCLE, collision is detected by checking if the distance between the two Objects is less than the average of the two Objects' size. This results in perfectly accurate circular hitboxes, even if the circles are different sizes.

The Object and Player classes are responsible for AI and targeting. AI Players jump at random intervals toward the most fitting target they can find. The targeting system includes a host of behaviours ranging from simply aiming at the nearest Player to leaving the goalie position to get the ball away from the wall nearest the goal. In football, AI will also prevent the ball from getting trapped near a corner by sending a single player to retrieve it. This player moves faster and is less accurate in order to hit the ball at more angles and get it out of the corner faster.

I solved every significant bug I encountered. However, there are some improvements I could have made, primarily with the AI. On bumpers, the AI will bump players without considering where they will end up. The dodgeball AI will target other players and cluster up when the ball is moving. AI players never consider the angle at which they are hitting their target, meaning they will often accidentally score on themselves.

Many quality of life and ease of use improvements could be made as well. At higher difficulties or player counts, it can be difficult to click a player to play as them. Sometimes the user may click a player when they intend to jump. The pause menu could have an option to restart the game so the user does not have to return to the menu to play again. Seeing as football games can go very quickly, football could include multiple rounds. I made pixel art to allow more players on screen without hurting performance as much, but seeing as I capped the player count at 16, the graphics could be improved as well.

The biggest issue I faced with Processing was how it works with class relationships, namely its inability to cast to subclasses. I had to place methods related to AI on the Object class rather than Player and then use “instanceof” to ensure the code would execute for the correct class. This resulted in implementing other unnecessary methods and using bad coding practices. Another issue I faced was that Processing would sometimes call a method on the parent class even when it was overridden in a child class. This required even more methods to be placed on the parent class rather than the child class.

Word count: 1089