Augmented Reality Two-player Pong Game Using Computer Vision

Richard Ballaux

Viktor Deturck

Leon Santen

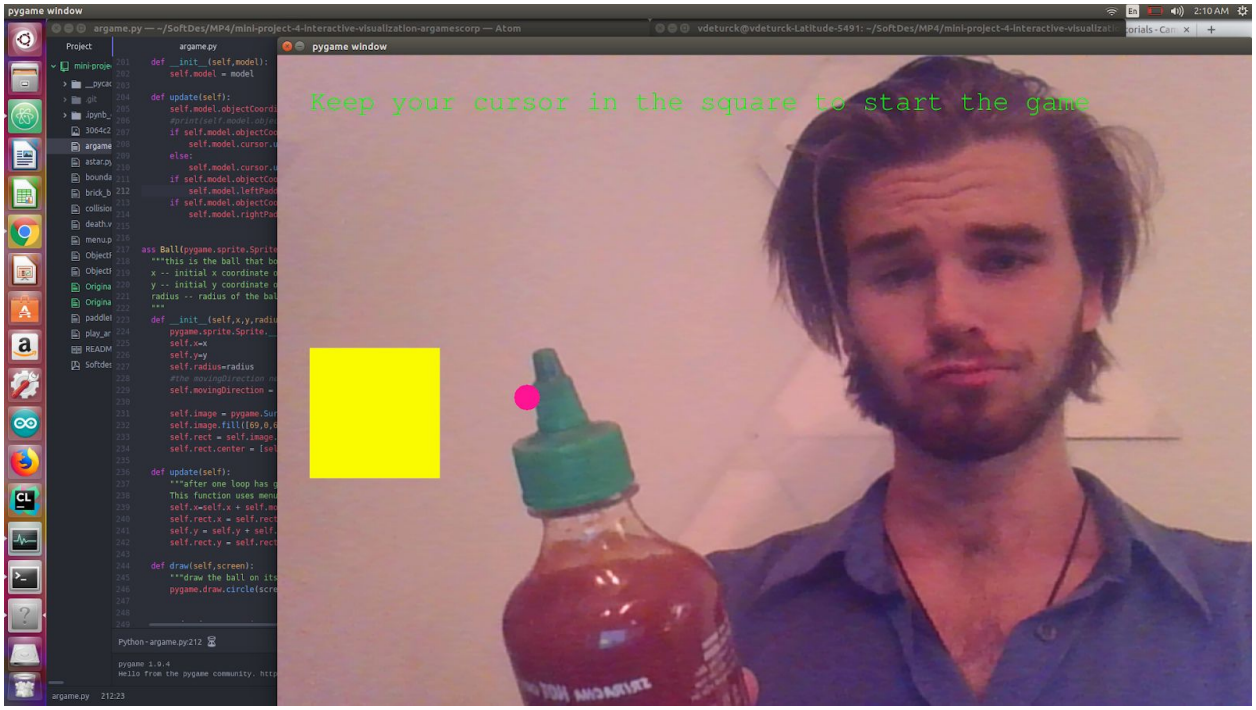
This game is a two-player augmented reality pong game. Each player uses a green object as their controller. The game consists of a menu section and a game section. When the game is started, the two players see the camera live feed in the background and the pong paddles and ball in the front. The position of the controllers is detected by OpenCV. Hovering over an object with the controller selects the corresponding settings. In the game, the green object's height controls the height of the pong paddles on the screen.

Results

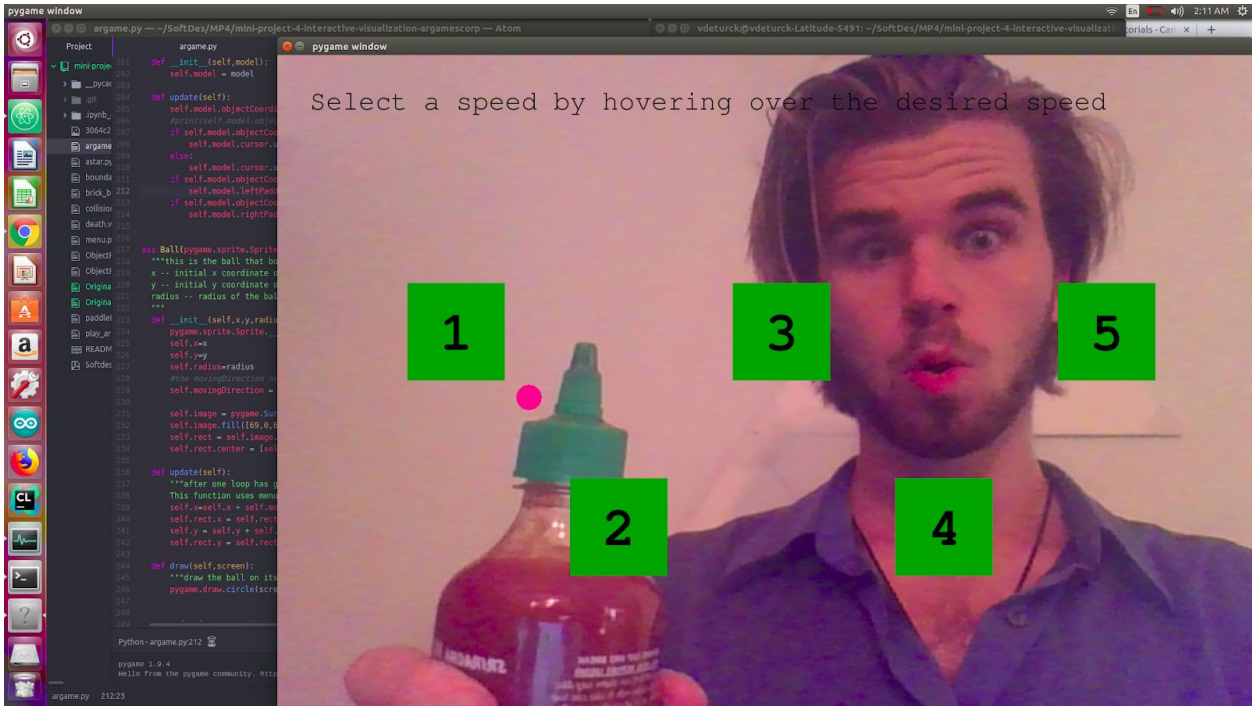
The game starts with a screen that prompts the player to hover over a specific area to get the feeling for how to use the controller. After hovering over this area, the player can choose between 5 different levels of difficulty. The level of difficulty changes the speed of the ball.

The actual game shows the mirrored camera image in the background. The ball, boundaries, and paddles are shown on top of the background image. When a player cannot hit the ball and the ball goes out the other player receives a point.

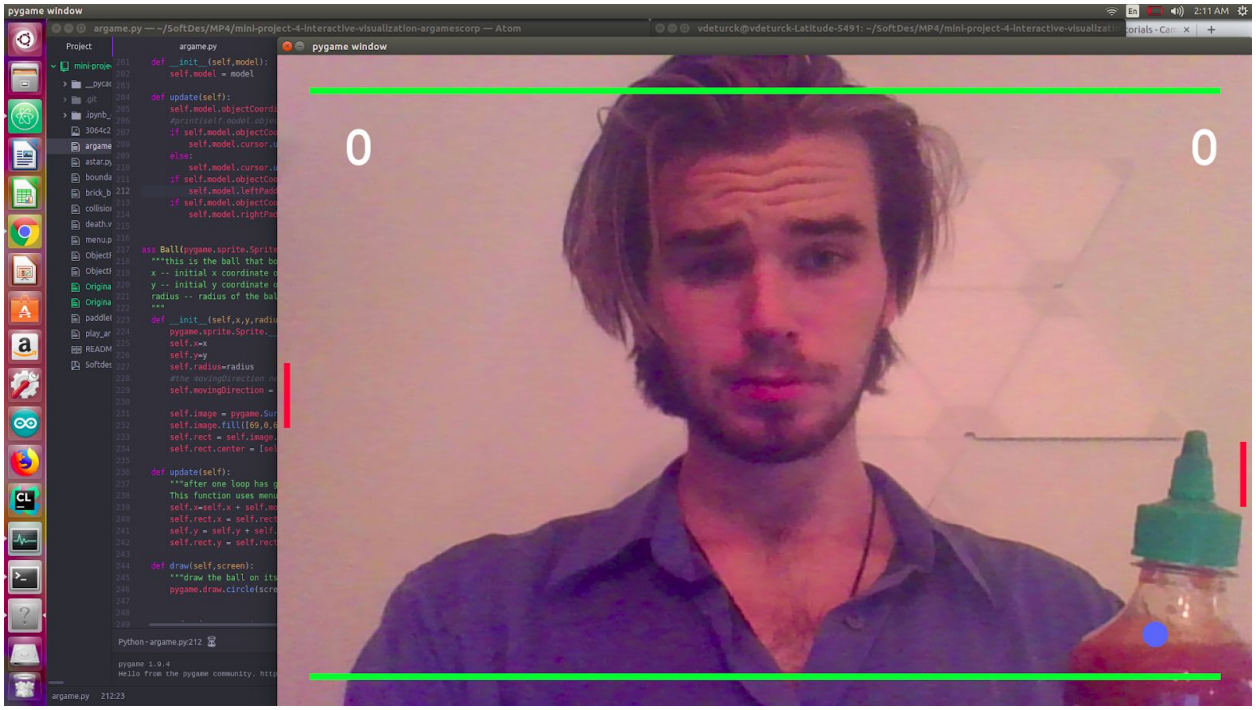
The tracking of the two green objects, one for each player, is dependent on the light situation in the room. Objects that reflect light are not eligible. The right player has to use the right side of the camera, the left player has to use the left side of the mirrored camera image.



To start the game, the player needs to hover with his arbitrary green object over the yellow box.



In the next menu, the speed of the ball can be set.



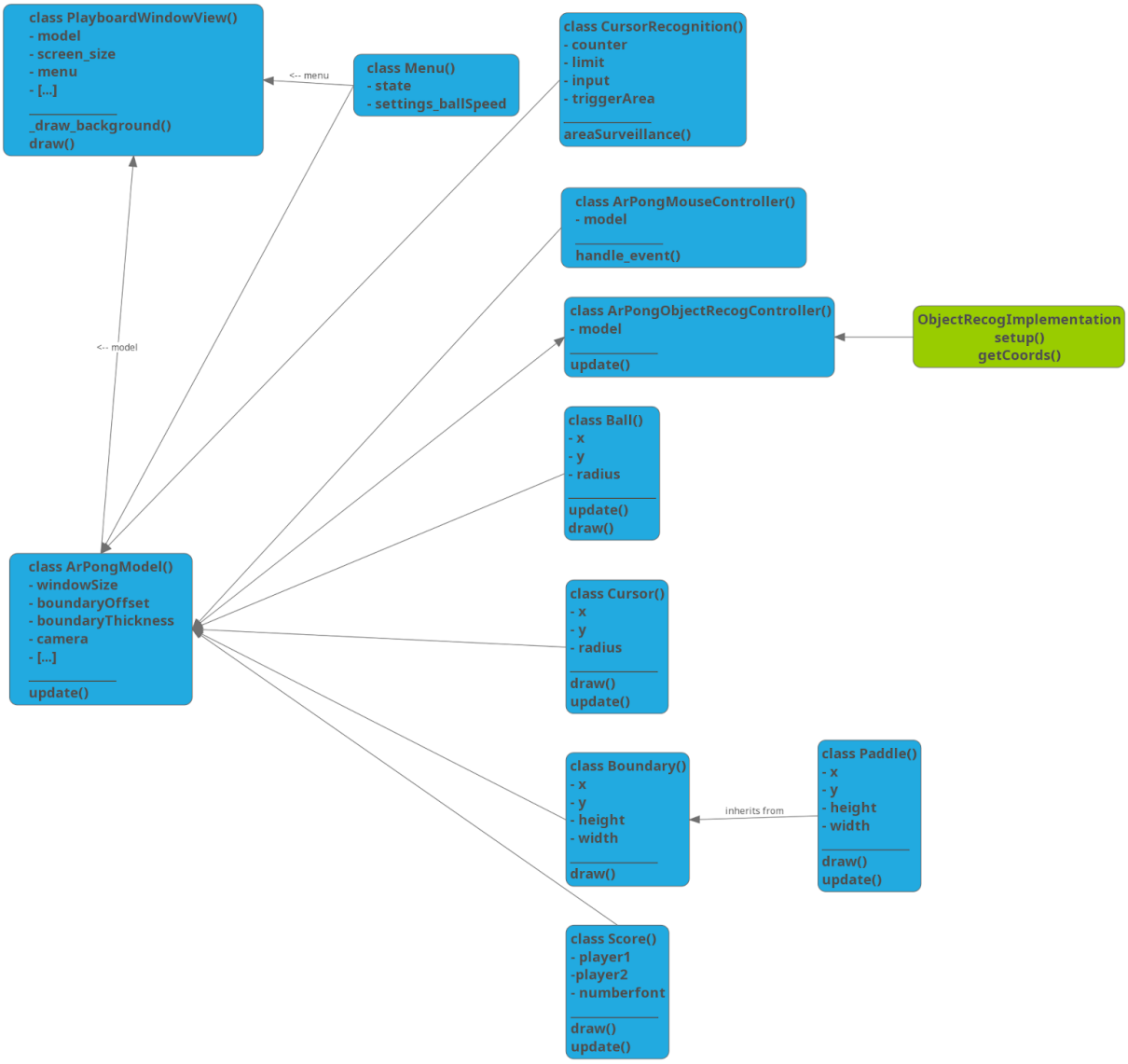
The actual game. The right paddle will follow the height of the most right green object, the left paddle will follow the most left object.

Implementation

The ArPongModel() is the model of the program. It updates the view class. Further, the model receives the status of the controller. The controller accesses another python script that outputs the position of the two controllers by using OpenCV.

The class Menu() provides the settings for the view and the model. The class Menu() also contains a state machine. The update functions of view and model both access the state of the menu state machine and change their update functions based on the state of the state machine. There are three different states at the moment: menu, select speed, game, endgame

One of the bigger design decision we had to make was how to analyze the camera image with OpenCV. We could not input the same picture from the camera that OpenCV used because the background of the game had to be bigger than the image OpenCV analyzed. The ObjectRecogImplementation.py file accesses the resolution of the game and calculation scaling factor that is used to output the position of the cursor that is calculated by analyzing a smaller camera picture.



Reflection

Besides small technical challenges, working on the same files simultaneously turned out to be a big challenge. To avoid merge conflicts, we had to text each other to confirm that no one is working on the same file. We did not feel comfortable enough to create branches and we lacked the knowledge to do so. On the performance side, we could have done more research to figure

out how to improve the object recognition while outputting a high-quality live feed as the background without analyzing the high-quality live feed.
We also could have done some more research on implementing our video recognition with the game. We found out there are some Pygame modules that implement the camera with some basic object recognition, which would have made the connection between the two elements in our project much easier. However, it was still interesting to learn and experiment with two different libraries.