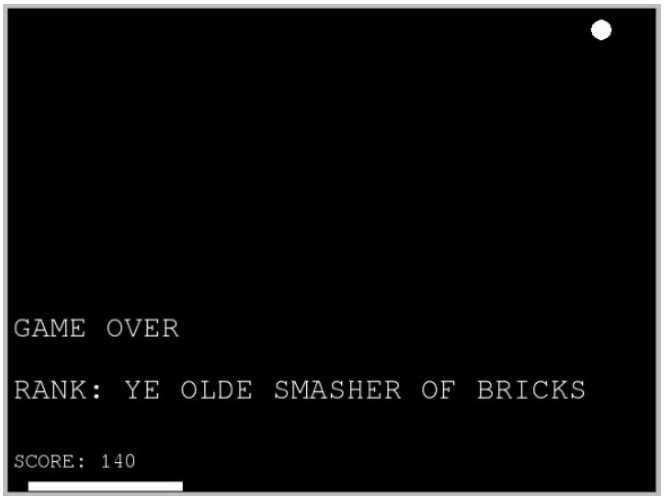
Overview

We built a version of the classic brick-break game in python using the pygame library. The player uses the arrow keys to move around a paddle at the bottom of the screen in order to bounce a ball up at the bricks. Each broken brick is worth one point, and the player's current score is displayed in the bottom left corner of the display screen. Noises are played upon impact and end game. When all bricks have been broken or the ball has been dropped, it is game over, and the player's rank is displayed.

Results

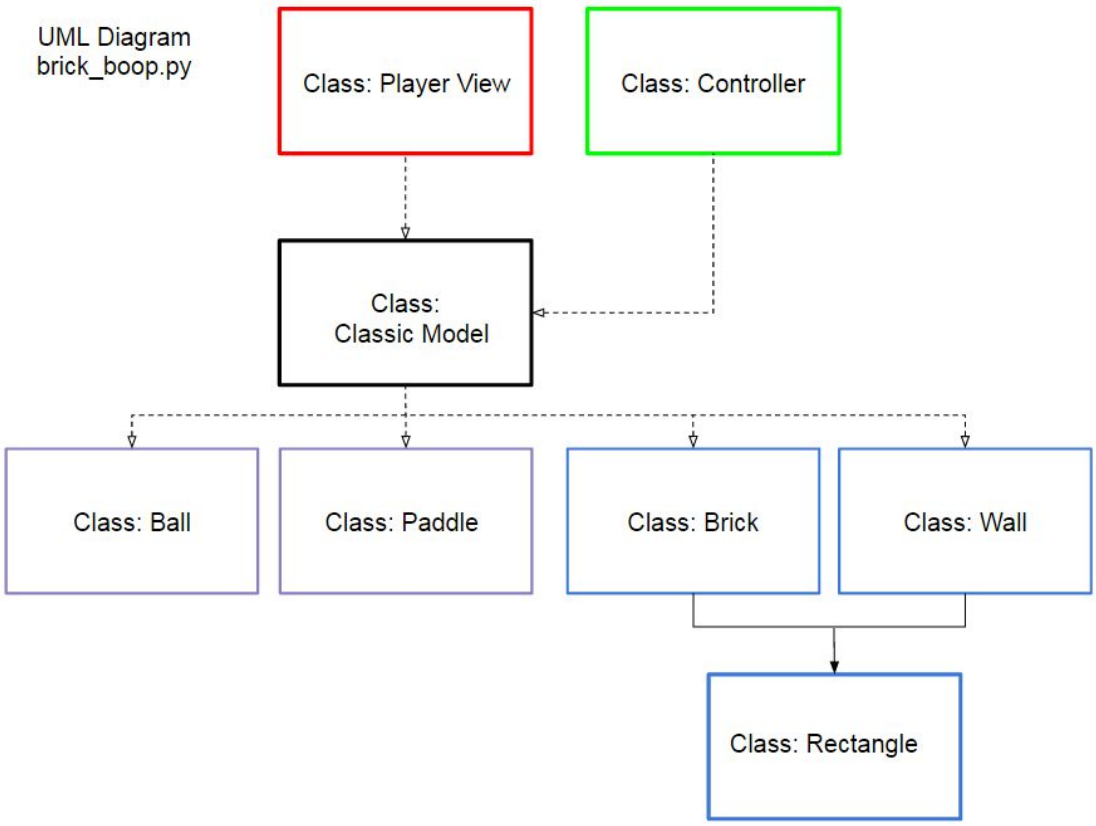
The gameplay is as intended, providing several minutes of sound enhanced brick booping joy to any player. Compete with yourself or friends to climb the ranks from Two Year old Terror to Ye Olde Smasher of Bricks! It is fairly straight-forward to obtain the highest rank in this game, but if you wish to learn of all the ranks, you must be willing to be a bit illogical! You will be greeted with a fanfare for clearing the screen… but if you drop the ball a trombone will laugh at you!

However, there are a few bugs with the physics between the ball and the bricks. There are times when the ball will come in at an angle and take out three bricks instead of hitting on and bouncing off. This is likely to do with the way that we checked for collisions and changed the ball's velocity in Classic Model. What is probably happening is that the ball touched two bricks at the same time, so the velocity is reversed twice and it continues in the original direction until it hits the third brick.



Implementation

This game is based on 4 classes: Rectangle, Controller, Classic Model, and Player View. The Rectangle class is a rectangle object that is inherited by the classes Brick and Wall. The Paddle and Ball classes have similar attributes to Rectangle, but are defined independently. Controller updates a dictionary that dictates what the paddle should do when there is a key-press. Classic Model defines how all of the elements of the game should interact and updates locations of objects based on the previous state and the interaction definitions. Player View draws the updated screen so that the player can see what is happening.



One of the larger design decisions we made in cleaning up our code was to not have Paddle and Ball inherit Rectangle. Although they both include Rectangle's attributes and are later used as rectangle objects, they have the additional feature of velocity which is not found in Rectangle. Initially we had attempted to use super with the Rectangle inheritance to accommodate these extra attributes, but we were plagued with errors and ultimately found it easier to implement and understand Ball and Paddle as independent classes.

Another design decision was to have the ball recognised as a rectangle object for collisions instead of a circle. While it is not a truly accurate determination of impact, it is close enough for our game resolution. It also enabled us to use colliderect() to determine when the ball collided with the paddle, walls, or brick, which increased our code clarity.

Reflection

We had ambitious plans that ended up being a bit too grandeur for our capabilities and the time that we had. We had wanted to implement multiple modes of gameplay, but there were some more basic pieces of the game that took us longer to sort out than anticipated. We were able to scale it down to be more manageable by only implementing one mode of gameplay and including a few additional features.

We did our best to implement our code in layers of functionality-- having a ball that could bounce off the sides of the screen, adding a paddle and debugging its motion, adding bricks, defining collisions with the paddle and then the bricks, and then finally adding UI elements such as a running score and sounds.

We planned on doing most of the work together, peer programming style, and that is what we ended up doing. We realized about midway through that there were communication issues resulting in small typos since one of us was constantly driving and the other constantly navigating/planning. For example, in implementing the sections of paddle that dictated what angle the ball bounced off at, we forgot to change one value in an if check inequality statement. -- which resulted in the ball not seeing that part of the paddle. Once we realized this we tried to be more aware of when we were working in parallel and check in with each other about what we just did.

One thing that went well was that we got faster at debugging and implementing game logic as we spent more time working through this project. We could still use some work on how efficient we are with the number of lines we need to implement logic. One lesson we learned very well in this mini-project is that print statements inside functions and loops are invaluable debugging tools.