Mini Project 4: Interactive Programming

Software Design

Rachel Yang

and

Jeremy Garcia

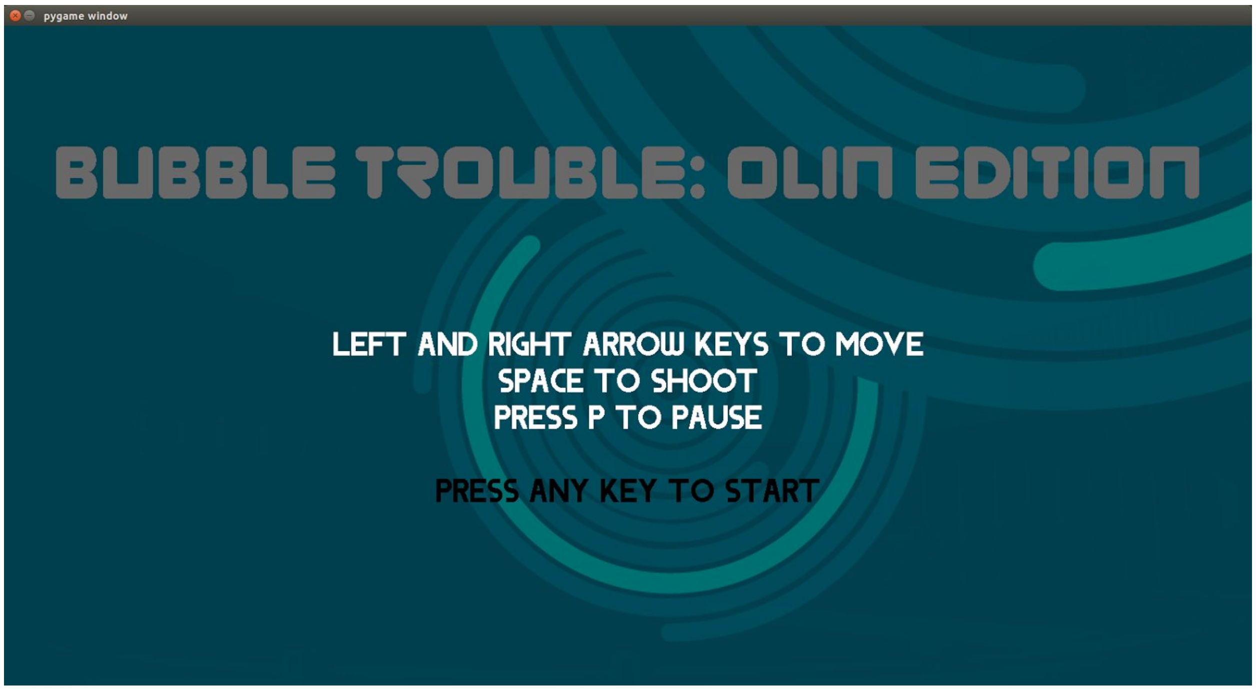
March 10, 2016

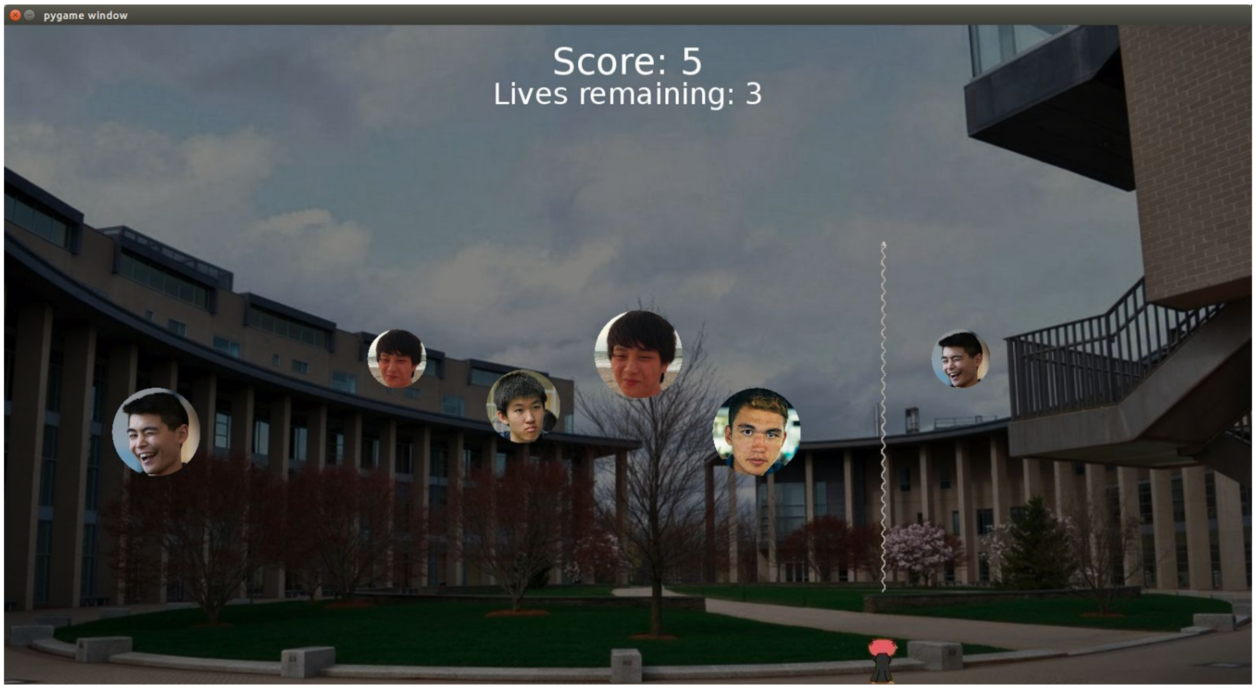
Project Overview

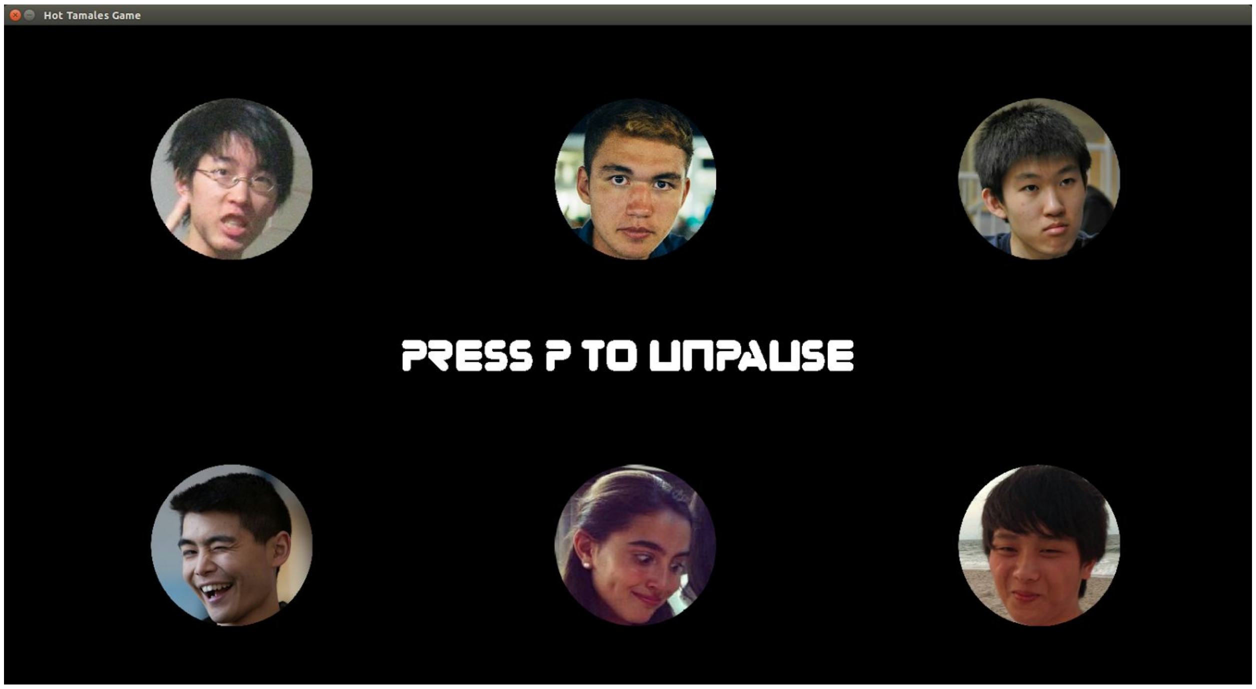
Our project was greatly inspired by the game *Bubble Trouble*, a classic arcade game in which the main character must either avoid death by dodging or popping the falling bubbles. Our main goal was to reinvent the *Bubble Trouble* experience by integrating the Olin community in some way. When we first started this project, we hoped to produce a fun, addictive, and nostalgic experience that paid tribute to *Bubble Trouble,* and we believe that we have accomplished this goal.

Results

The goal of our game, *Bubble Trouble: Olin Edition* is to pop as many bubbles as possible before running out of lives. However, instead of having normal bubbles, our bubbles are pictures of random faces of different Olin students. Once a bubble is hit, it splits into two smaller bubbles which travel in opposite directions. This occurs until the bubble is a sufficiently small size, upon which it disappears when hit.







Implementation

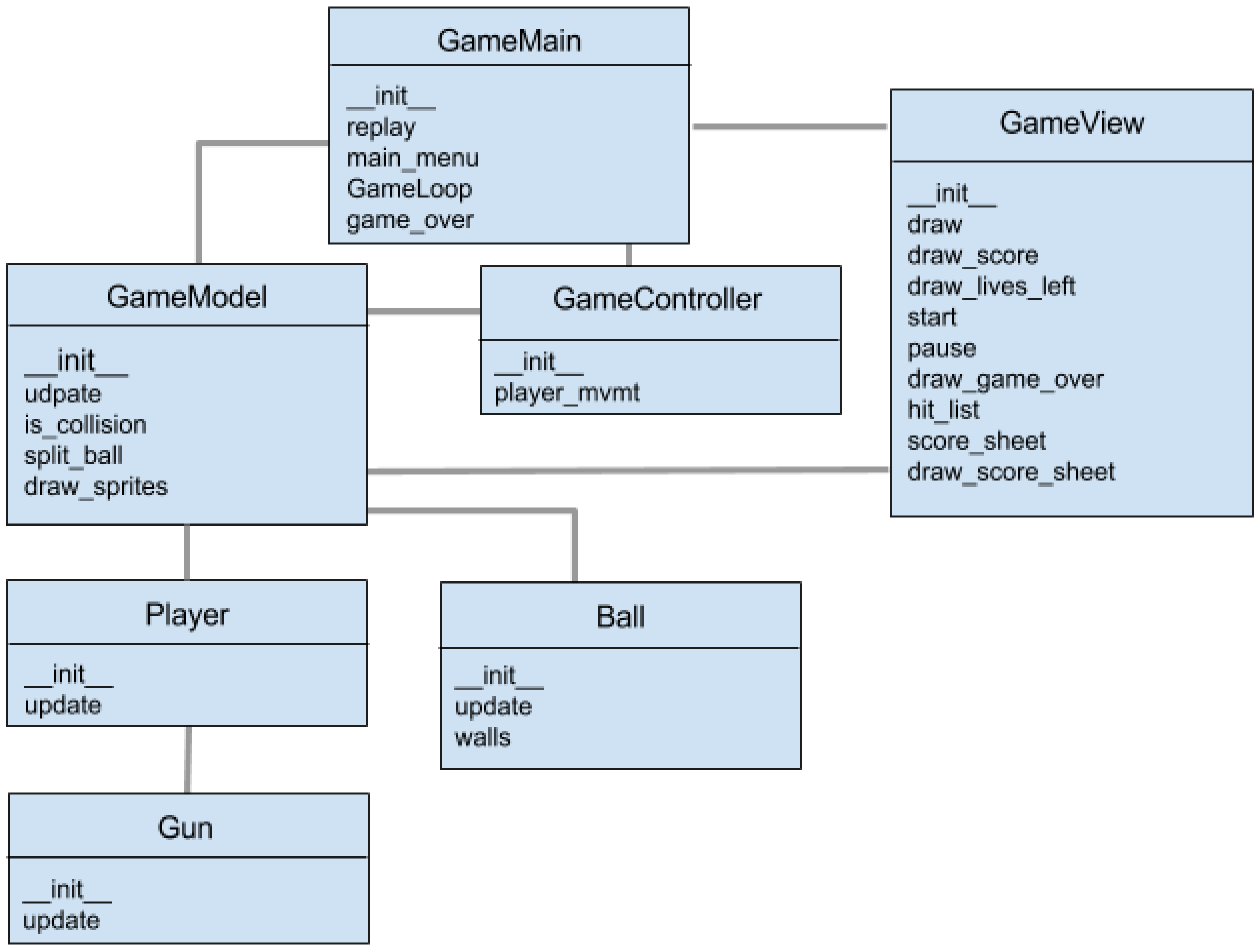
We chose to structure our game with a Model-View-Controller setup. As you can see in our UML diagram below, the three are somewhat interconnected--the Game Model is utilized in the Game Controller as well as the Game View. The biggest class that envelops the rest of the classes is GameMain. It calls instances of the model, view, and controller, displays the main menu, and essentially runs our event loop for the game, implementing the game as long as it is not in pause and as long as the player is still alive. When the player isn't alive, GameMain also calls the Game Over screen to display.

GameView focuses on what the user sees when he or she is playing our game. Our __init__ class initializes the fonts (including color, size, font type, etc.) and what messages are displayed on what screens. GameView is what allows us to have multiple screens: a main menu screen, a pause screen, a game over screen, and the screen for the game itself while it's running. It also lets us display information such as the score, the lives remaining, the highest score achieved, which Oliners you hit and which ones hit you. Finally, GameView draws all of our game pieces onto the screen, including our main player, the weapon, its arrows, and the bubbles.

GameController organizes the key presses for the game. The right arrow key moves the character right; the left arrow key, left; and space shoots the gun.

GameModel calls an instance of the player and the balls, or bubbles. It initializes the important constant variables such as the starting score and the starting number of lives. It updates the sprites at each moment the game runs and handles the interactions between the sprites, especially detecting and implementing collisions between them.

We had many choices in how to make the game continuous. We could have added different levels and once the player cleared one level, it would have moved on to the next, but we wanted to keep the game construction simple. We could have had a horizontally scrolling game, but we wanted the balls to bounce off of the left and right walls in addition to the floor. We could have randomly added a new ball once the user cleared all balls on the screen, or added a new ball once a certain amount of time has passed. We ended up choosing to add a ball when the user clears the others because that seemed more fair to the player in terms of gauging what difficulty they need to be playing at.



Reflection

One aspect of our game that we are pleased with is its similarity to the original *Bubble Trouble* while still remaining a unique game. Our project was adequately scoped since we focused on producing a minimum viable product and then added extra features once we had implemented the basics. This project, in addition to showing us the pygame module, was helpful in teaching us how to effectively structure code in Python classes. In regards to our team process, we planned to do pair programming when our schedules allowed, while also dividing the work as we saw fit. This worked out well since we were able to both program together, and also work separately when the other person was busy.