Our system includes multiple different objects moving and interacting on screen. We have a ship, bullets, and asteroids. The ship at its most basic level is an object that can rotate, move forward, and shoot bullets. To better simulate the original game, we incorporated acceleration and deceleration in the ship's movement, giving it a more space like feel and adding extra challenge to the game. Throughout the game, asteroids must be spawned off screen, then fly across the screen in some random direction at some random speed. If the ship collides with an asteroid, the game must end. The bullets the ship shoots can also collide with asteroids, but this instead increases your score and destroys the asteroid. As in the original game, any object that moves off screen must wrap around to the other side of the screen as if it is all connected. We also have added a "super" ability, which is unlocked upon achieving a high enough score. This can be activated to clear the entire screen of all asteroids instantly.

The user stories completed in our system are:

As a user, I'd like to have set up options for the game.

As a user, I would like a main menu.

As a user i'd like an aesthetically pleasing interface.

As a user i'd like to be able see my score, save my score, and see a list of high scores.

As a user, I want the ability to spin and move around the screen.

As a user, I would like a boss battle.

As a user, I would like asteroids to spawn randomly.

As a user, I want asteroids to split into more asteroids when destroyed.

As a user, I want the ability to have some kind of power-up.

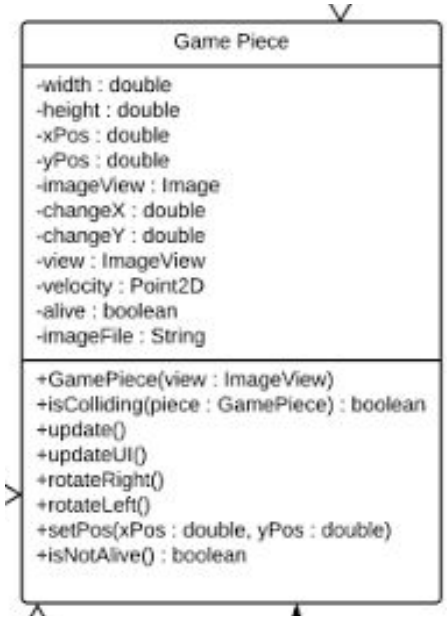
As a user, I want the ability to shoot bullets to destroy asteroids.

As a user, I want to be able to restart to play again.

These user stories are at the heart of the classic game of Asteroids. They can essentially be split into categories including gameplay, more specifically movement and functionality, and menus/graphics not part of the actual gameplay. Our system includes an introductory menu from which the game can be started, or a list of the controls to play the game can be brought up, satisfying these user stories. Once the game is over, we have an ending screen that allows you to quit or return to the main menu, possibly to play again if you so desire. We also added nice space related graphics to give the game a more modern feel. As for the gameplay, the user has three controls. They can move forward, rotate (either left or right), and shoot. These are the essential functions that make up the original classic game, and we also added a super ability that unlocks after a certain score is achieved, allowing the user to clear the screen of all asteroids, essentially counting as a powerup. Finally, we have asteroids generating randomly anywhere off screen, which can be any one of three different sizes. Then, each asteroid has a random velocity and direction. When larger asteroids are hit, they split into smaller ones as requested in the user stories. Anytime an asteroid is destroyed by a bullet shot by the user, their score increments and is displayed in the top right corner of the screen, again as requested.

The way we implemented object-oriented programming was fairly simple. We have one parent class, GamePiece, that contains all the attributes and methods that the ship, asteroids, and bullets all possess, and takes in an ImageView as a parameter in the constructor. The Ship, Asteroid, and Bullet classes are all children of the GamePiece parent class. By doing this, we were able to utilize polymorphism many times throughout our program, especially in the controller class. For example, the method that checks for collisions is an instance method of

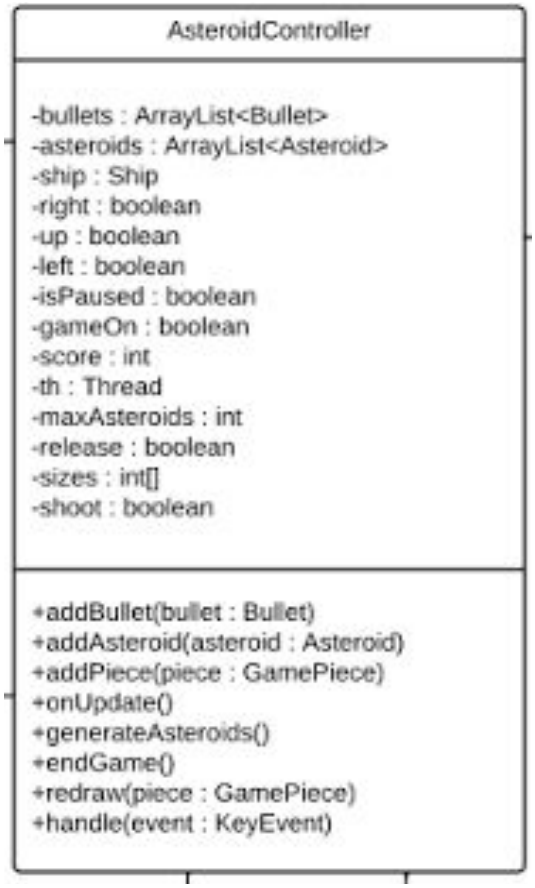
GamePiece and takes in another GamePiece object to compare the first one to. In our controller, we were able to iterate through lists containing the Asteroid objects and Bullet objects and check for collisions between those as well as the ship, only having to utilize one method. The GamePiece class is also useful for handling some very basic movement, such as objects wrapping from one side of the screen to the other. This attribute is shared between all child classes, so it is easily handled within the parent class. You can see from the UML all the methods utilized in GamePiece that are inherited by Ship, GamePiece, and Asteroid.



However, the Ship, Asteroid, and Bullet classes all have other , more specific variables and methods to handle aspects of movement specific to that class. The Ship class has several different methods for movement as well as a shoot method that spawns a bullet. The ship must be able to rotate, and travel in the direction it is facing. This is unique because the asteroids and bullets do not rotate at all, and travel in a direction unrelated to their orientation. The Ship class also incorporates aspects of acceleration, gradually speeding up when directed to move, and gradually slowing down after the move key is released, but still traveling in the same direction

even if it is then rotated to give a feeling of little or no friction in space. Bullets also have a unique aspect in that they must disappear after traveling for a certain distance untouched. The ship and asteroids never disappear but just keep moving around the screen until they are destroyed. We deal with all of these different class-specific movement patterns from within the child classes.

The controller is simply used to update what happens to each piece after every update loop, which just runs continuously. On each update, the controller must go through all the asteroids currently in the game and see if they are colliding with a bullet or the ship. It must then update the position of each object depending on all the inputs to the game and that object's specific move method. For example, it must move all asteroids from their current position to a new position depending on each individual asteroid's velocity. It must move and/or rotate the ship based on the keys pressed by the user, and the ship's current velocity and acceleration or deceleration. Finally, it must use the ship's shoot method to spawn a bullet at the ship's location if the user presses the key to do so. See this screenshot from the UML of the AsteroidController class:


AsteroidController
-bullets : ArrayList<Bullet>
-asteroids : ArrayList<Asteroid>
-ship : Ship
-right : boolean
-up : boolean
-left : boolean
-isPaused : boolean
-gameOn : boolean
-score : int
-th : Thread
-maxAsteroids : int
-release : boolean
-sizes : int[]
-shoot : boolean
+addBullet(bullet : Bullet)
+addAsteroid(asteroid : Asteroid)
+addPiece(piece : GamePiece)
+onUpdate()
+generateAsteroids()
+endGame()
+redraw(piece : GamePiece)
+handle(event : KeyEvent)