CS 1410: Asteroids Part 2

Nearly everyone has played or at least heard of the famous arcade game Asteroids. But, if you have not, you can play it here. The game involves a player-controlled spaceship that can turn left or right, accelerate forward, and shoot bullets. A collection of rocks (asteroids) move through space, potentially on a collision course with the spaceship. If a collision between the spaceship and a rock occurs, then the spaceship is destroyed. If a bullet collides with a rock, the rock and bullter are both destroyed. The objective of the game is to eliminate all of the rocks, by successfully shooting them from the spaceship, before a devastating collision incident occurs.

Assignment

Your assignment is to recreate a simple Asteroids game using Python and Pygame. The assignment will consist of two sequential parts. For this second part, you are required to add implementions the following features of the game:

1. A bullet must fire from the ship and travel in the direction the ship is facing at the time. The bullet will only stay active for a limited amount of time after being fired. Then, it should disappear.
2. Collisions between any rock and the bullet will cause the bullet and the rock to disappear.
3. Collisions between any rock and the ship will cause the game to end.
4. All rocks being destroyed will cause the game to end.
5. Stars will be displayed, and they will twinkle, becoming brighter and dimmer.

For part 2 of the assignment, only the above additional functionality is required. You are welcome to continue working on additional features once you complete the requirements for part 2, but it is your responsibility to complete the requirements for part 2 of the assignment first, and submit it by the due date.

For this assignment, you are required to demonstrate use of the object oriented principles inheritance, polymorphism and aggregation when implementing the classes that have been designed for you to represent the game and its various components.

You should use your solution to part 1 of the assignment as a starting point.

Part 2 Unit Tests

Required Classes

The required classes are listed in the UML Diagram. Not all of the methods will be described in detail here. For example, most getter methods will not be listed. However, if they are in the UML diagram, they are required. If you have questions about the required functionality, please ask questions in class or in the discussion forums.

Only changes to part 1 classes are listed here.

This browser does not support PDFs. Please download the PDF to view it: Download PDF.

Movable Class

Movable Data Members

- `mActive` boolean value, whether the object is active or not.

Movable Methods

- `__init__` initializes `mActive` to true.

- `getActive` returns the value of the `mActive` data member
- `setActive` updates the data member to the value in the parameter.
- `hits` returns true if this object's "circular" area overlaps with that of the other object. Use the circle collision technique discussed in class.
- `getRadius` is an abstract method. It should `raise NotImplementedError`.

Rotatable Class

Rotatable Data Members

- No changes

Rotatable Methods

- No changes

Polygon Class

Polygon Data Members

- No changes

Polygon Methods

- `getRadius` For each point in the original polygon, calculate the distance from the origin. Return the average of these distances. If there are no points, return 0.

Ship Class

Ship Data Members

- No changes

Ship Methods

- `fire` Creates a bullet and returns it. The bullet should be created with the location of the ship's rotated and translated first point. Also, the bullet will have the same velocity components as the ship and the same rotation as the ship.

Rock Class

Rock Data Members

- No changes

Rock Methods

- No changes

Circle Class

Circle Data Members

- `mRadius` is a number measured in pixels. The radius of the circle.
- `mColor` is a PyGame color, a 3-tuple of integers in the range 0-255 describing the red, green and blue channels of the color.

Circle Methods

- `__init__` uses constructor chaining to initialize the `Rotatable` data members, and sets the object's radius from the paramater, and sets the color to white.
- `setRadius` updates the data member, but only if the new value is at least 1.
- `setColor` updates the data member. It assumes the new color is valid.

- `draw` Uses the PyGame functions to draw the circle described by the `Movable` position and the radius.

Bullet Class

Bullet Data Members

- `mAge` the number of seconds the bullet has been in existence.

Bullet Methods

- `__init__` uses constructor chaining to initialize all `Circle` data members. Bullets all have a radius of 3. Initializes the age of the bullet to 0. Accelerates the bullet 100.0 units. Moves the bullet 0.1 seconds worth of movement. Without this, it will hit the ship.
- `setAge` updates the data member, assuming the parameter is correct.
- `evolve` moves the bullet. Adds `dt` to the age of the bullet. If the bullet is more than 6 seconds old, makes it inactive.

Star Class

Star Data Members

- `mBrightness` an integer value. The current star brightness, in the range 0 to 255.

Star Methods

- `__init__` uses constructor chaining to initialize all `Circle` data members. Stars have no speed, rotation of 0, and radius of 2. Initializes the brightness to a random value.
- `setBrightness` updates the data member, but only if the new brightness is within the range specified above. If the brightness changes, update the color as well.
- `evolve` Tries to changes the brightness by adding 10, subtracting 10, or doing nothing, each with equal probability.

Asteroids Class

Asteroids Data Members

- `mBullets` a list of all `Bullet` objects active in the game
- `mStars` a list of all `Star` objects created
- `mObjects` a list of all objects active in the game, including `Bullet`s and `Star`s.

Asteroids Methods

- `__init__` Create 20 `Star`s in random locations and an empty `Bullet` list. Store the `Star`s in the correct lists.
- `fire` If the maximum number of active bullets (3) has not been reached, create a new bullet, add it to the appropriate lists. Refer to the `Ship` and `Bullet` classes.
- `evolveAllObjects` Call evolve on all objects.
- `collideShipAndBullets` Check and handle collisions between bullets and ship.
- `collideShipAndRocks` Check and handle collisions between rocks and ship.
- `collideRocksAndBullets` Check and handle collisions between bullets and rocks.
- `removeInactiveObjects` Removes all inactive objects from all lists.
- `evolve` Be sure that all objects evolve. If any bullet collides with the ship, the ship and bullet should become inactive. If any bullet collides with any rock, the rock and bullet should become inactive. If any rock collides with the ship, the ship and bullet should become inactive. Remove any inactive rocks and bullets from the lists. Use the methods outlined above, in the correct order.
- `draw` Be sure that all objects draw, but only if they are active.

main

- `game_logic` Add a key press to call the `fire` method. Be sure to check `newkeys`, not `keys`.

Hints

- Refer to the Pygame documentation to understand which parameters are necessary when calling each of the Pygame draw methods. Specifically, you should be interested in `pygame.draw` and `pygame.Rect`.
- When creating colors, use a helpful tool to determine the RGB values. Here are two good options: color.adobe.com and colorpicker.com