CS 1410: Asteroids Part 1

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 bullet 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 first part, you are required to implement the following features of the game:

1. A player-controlled spaceship that can turn in either direction, to point in any direction, and accelerate forward. All three spaceship functions should be controlled by keyboard or mouse input from the user. Remember, the ship may only accelerate in the forward direction (relative to the direction that the ship is currently oriented); it may not ever accelerate in the backward direction. The spaceship should be drawn using a simple polygon that resembles a spaceship. Either use the original game for inspiration, or use your own creativity. The polygon should rotate visually on the screen according to the rotation commands given by the player. Also, if the spaceship moves off the screen, then it should instantly reappear on the opposite side of the screen, still traveling in the same direction and at the same speed. This should apply to all four sides of the screen (left/right and top/bottom).

2. A number of rocks that move freely through space, at random directions and speeds. Rocks will rotate at a fixed rate; they simply move along a straight line, in a random direction, never accelerating. The shape and size of each rock should also be randomly different. The rocks should be drawn using a simple polygon that resembles an asteroid (i.e. a rock). Again, feel free to use the original game for inspiration. Just like the spaceship, rocks should reappear on the opposite side of the screen when they travel off the screen.

For part 1 of the assignment, no additional functionality is required (e.g. shooting bullets and destroying rocks or colliding with rocks). You will add these features in part 2. You are welcome to continue working on additional features once you complete the requirements for part 1, but it is your responsibility to complete the requirements for part 1 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. For instance, the `Movable` class implements the game logic for moving an object (and enabling it to wrap around the screen edges, etc.).

You should use our `PyGame` starter kit.

Part 1 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.

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

Movable Class

The `Movable` class handles most aspects of objects that have a location and may be able to move.

Movable Data Members

- `mX` and `mY` are the object's position on the window, measured in pixels. `mX` must be greater or equal to 0 and less than `mWorldWidth`. `mY` must be greater or equal to 0 and less than `mWorldHeight`.
- `mDX` and `mDY` are the object's speed in the horizontal and vertical directions, measured in pixels per second.
- `mWorldWidth` and `mWorldHeight` are the dimensions of the window, measured in pixels.

Movable Methods

- `__init__` creates and initializes the data members directly from the parameters. Assumes all parameter values are valid.
- `move` updates the values of `mX` and `mY` using the speeds `mDX`, `mDY` and the amount of elapsed time, `dt`. If the object moves off of the window, updates the coordinates to have the object appear on the other side of the window. For example, moving off of the right side will cause the object to appear on the left side.
- `accelerate` is an abstract method. It should `raise NotImplementedError`.
- `evolve` is an abstract method. It should `raise NotImplementedError`.
- `draw` is an abstract method. It should `raise NotImplementedError`.

Rotatable Class

The `Rotatable` class adds the ability to rotate objects, using inheritance to keep all of the functionality of the `Movable` class. It also implements the `accelerate` method, making it possible to change the motion of objects.

Rotatable Data Members

- `mRotation` is the object's orientation, measured in degrees. 0 degrees is to the right, 90 degrees is down, etc. Note that this is not the direction of travel. Direction of travel is controlled by `mDX` and `mDY` from `Movable`.

Rotatable Methods

- `__init__` uses contructor chaining to initialize the `Movable` data members, and sets the object's rotation, assuming the input values are all valid.
- `rotate` adds to the object's orientation. Note that the value of `delta_rotation` may be positive or negative. Adding a negative number is fine. That reduces the value. This method must make sure that the rotation is at least 0 and is less than 360. If the rotation goes out of this range, then update it appropriately. For example -10 degrees should be updated to 350, and 375 degrees should be updated to 15 degrees.
- `splitDeltaVIntoXAndY` receives a `rotation` in degrees, and a `delta_velocity` in pixels per second. The method returns a 2-tuple of the amount of velocity change in the horizontal and vertical directions. The easiest way to do this is using the `math` module's functions for converting degrees to radians and calculating the cosine and sine values of angles. Pay attention to the discussion in class, and ask questions.
- `accelerate` splits `delta_velocity` into horizontal and vertical components using other methods and the object's current rotation. It then adds to the object's `mDX` and `mDY` to change the speed of movement.
- `rotatePoint` receives `x` and `y`, the coordinates of an arbitrary point. It returns new values for `x` and `y` as a 2-tuple. The new values are rotated about the origin based on the object's current rotation. This is done using the trigonometric functions of the `math` class. Join in the class discussion, take notes, and ask questions.
- `translatePoint` receives `x` and `y`, the coordinates of an arbitrary point. It returns new values for `x` and `y` as a 2-tuple. The new values are calculated by adding the object's `mX` and `mY` values.
- `rotateAndTranslatePoint` receives `x` and `y`, the coordinates of an arbitrary point. It returns new values for `x` and `y` as a 2-tuple. The new values are calculated by first rotating the point and then translating the rotated coordinates.
- `rotateAndTranslatePointList` receives a list of points. The list is a normal Python list. Each point in the

list is a 2-tuple of `x` and `y` coordinates. The method constructs a new list of points to return. The new list has each point rotated and translated.

Polygon Class

The `Polygon` class adds the ability to track the shape of an object as a list of points and the ability to draw the object to the `Rotatable` class.

Polygon Data Members

- `mOriginalPolygon` is a list of 2-tuples, where each 2-tuple is a pair of `x` and `y` coordinates. This polygon should be described with values that are centered on the origin (x 0, y = 0).
- `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.

Polygon Methods

- `__init__` uses contructor chaining to initialize the `Rotatable` data members, and sets the object's polygon to the empty list and the color to white.
- `setPolygon` receives a list of points. It assigns this to the correct data member, assuming the list is valid.
- `setColor` receives a color. It assigns this to the correct data member, assuming the value is valid.
- `draw` gets a copy of the original polygon list of points that has been rotated and translated. Then, uses the PyGame functions to draw the polygon described by the rotated, translated outline.

Ship Class

The `Ship` class adds the ability to evolve (update) like a ship to the `Polygon` class.

Ship Data Members

- None

Ship Methods

- `__init__` uses contructor chaining to initialize the `Polygon` data members, and sets the object's polygon to the shape of the ship. The ship should not be moving and should have a rotation of 0.
- `evolve` causes the ship to move.

Rock Class

The `Rock` class adds the ability to evolve (update) like a rock to the `Polygon` class.

Rock Data Members

- `mSpinRate` The rate that the rock spins. Measured in degrees per second. May be positive or negative.

Rock Methods

- `__init__` uses contructor chaining to initialize the `Polygon` data members, and sets the object's polygon to the shape of a random rock. Rocks are initialized not moving, with a rotation randomly selected from 0.0 to 359.9. Also, sets the spin rate for the rock to random floating point value in the range -90 degrees per second to +90 degrees per second. Finally, rocks are finally accelerated a random amount from 10 to 20 pixels per second.
- `createRandomPolygon` creates a list of 2-tuples with the coordinates of the outline of a random rock shape, and returns the point list. The angles of the points must be equally spread around a circle. For example, if there are 5 points, then the points will be 360 / 5 = 72 degrees apart. Each point's distance from the origin is randomly chosen to be between 70% and 130% of the `radius` paramter.
- `evolve` causes the rock to move and spin.

Asteroids Class

The `Asteroids` class is the overall game class that creates and controls all objects.

Asteroids Data Members

- `mWorldWidth`, `mWorldHeight` the dimensions of the window, in pixels.
- `mShip` the `Ship` object created.
- `mRocks` a list of all `Rock` objects created.
- `mObjects` a list of all objects created.

Asteroids Methods

- `__init__` Sets the data members, including creating a `Ship` and 10 `Rock`s.
- `getShip`, `getRocks` and `getObjects` return the appropriate data members.
- `turnShipLeft` reduces the ship's rotation by `delta_rotation`.
- `turnShipRight` increases the ship's rotation by `delta_rotation`.
- `accelerateShip` accelerates the ship by `delta_velocity`.
- `evolve` evolves all objects by `dt`.
- `draw` draws all objects.

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