**Skeet Project Description**

**Overview**

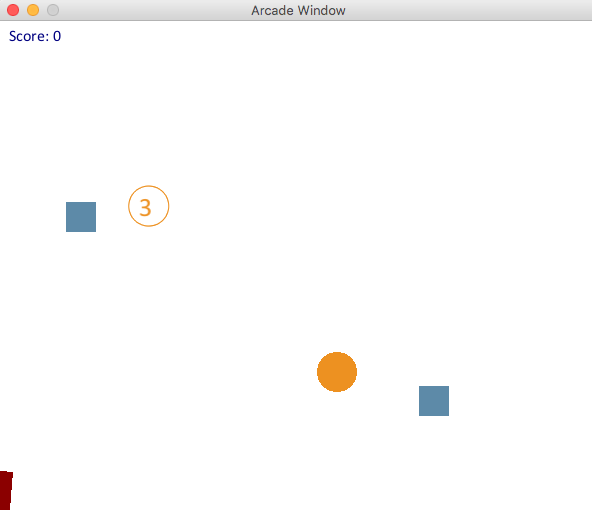
Use your knowledge of object-oriented programming to write a basic target shooting game.

This project will use the same arcade library as the previous project. Please refer to that project for instructions about configuring your environment.

**Instructions**

Your assignment is to create a game that simulates skeet shooting. On the left side of the screen, clay pigeons, or targets are randomly thrown across the screen. On the bottom left corner of the screen, the "marksman" (the term is used very loosely here) aims the rifle. The object of the game is to hit the intended targets, and not other "safe" ones.

The following shows the game in action:



**GAME RULES AND SPECIFICATION**

1. When there is no pigeon on the screen, a new one is created with a 1/50 probability.
2. To make it more of a challenge, there are three types of pigeons:
   1. Standard Target
      * Rendered as a circle with a 20px diameter.
      * Destroyed with one hit.
      * 1 point is awarded for hitting it.
      * Use the **arcade.draw\_circle\_filled** to assist you.
   2. Strong Target
      * Rendered as a circle with a number inside of it.
      * The strong target should move more slowly than the others as defined below.
      * It takes 3 hits to destroy this target.
      * 1 point is awarded for each of the first two hits.
      * 5 points are awarded for the third hit that destroys the target.
   3. Safe Target
      * Rendered as a square.
      * Use the **arcade.draw\_rectangle\_filled** function to assist you.
      * This target should not be hit.
      * It is destroyed with a single hit.
      * A penalty of 10 points is incurred if this target is hit.
3. The target type, direction, velocity, and timing to release (delay) are random according to the following constraints:
   1. The initial position of the target is anywhere along the top half of the left side the screen.
   2. The horizontal component of the velocity should be between 1 and 5 pixels/frame.
   3. The vertical component of the velocity should be between -2 and +5 pixels/frame.
   4. To give the user a greater chance to hit the strong target, it should move more slowly than the others. In particular, its horizontal velocity should be taken from the range: 1 to 3, and it's vertical velocity from the range -2 to +3.
   5. New targets should be created in the "update" function with 1/50 probability. This can be achieved by drawing a random number from 1 to 50 and checking if it's 1.
   6. There is no limit to the number of targets that can be on the screen at a time, but they should be removed from the game when they leave the screen.
4. Rifle
   1. Rendered as a rectangle.
   2. The aim is controlled to match the mouse cursor.
5. Bullets
   1. Rendered as a filled-in circle.
   2. There is no limit to the number of bullets.
   3. Clicking the mouse fires a new bullet.
   4. New bullets should be aimed in the direction of the rifle.
   5. Bullets travel at 10 pixels/frame at that angle at which they are fired.
   6. Bullets should be removed if they leave the borders of the screen.

You are responsible to implement the following:

* The **Bullet** class
* Classes for the various types of targets
* A base class for flying objects
* In the game class: creating, storing, drawing and advancing the targets. (Hint: look for the "TODO" comments in the game class.)

Provided you correctly implement the methods and data members for targets and bullets, the collision detection and keeping of the game score will be taken care of for you.

The game class expects the following interface to be present for targets and bullets:

|  |
| --- |
| **Bullet** |
| center : Point |
| velocity : Velocity |
| radius : float |
|  |
| \_\_init\_\_() |
| +advance() : None |
| +draw() : None |
| +is\_off\_screen(screen\_width, screen\_height) : Boolean |
| +fire(angle:float) : None |

|  |
| --- |
| **Target** |
| center : Point |
| velocity : Velocity |
| radius : float |
| alive : Boolean |
|  |
| +\_\_init\_\_() |
| +advance() : None |
| +draw() : None |
| +is\_off\_screen(screen\_width, screen\_height) : Boolean |
| +hit() : int |

The **hit()** method for the **Target** represents the target being hit and should either kill the target (or decrement the number of hits remaining for the strong target) and return an integer representing the points scored for that hit.

**ARCHITECTURAL DESIGN**

For this project, you will be expected to use the principles of inheritance and polymorphism. These are the topics we will be studying over the next two weeks.

In order to demonstrate correct use of these design principles, in your game class, the targets should all be put in the same list and then treated "identically" throughout the game code. In other words, it should not have a separate list for each type of target.

**CODE HINTS AND SUGGESTIONS**

* Start by creating all the classes you need with the correct data members and blank methods. Then, you can go through and fill in the methods one by one.
* You can get a random float from a range as follows:

import random

...

random.uniform(-2, 2)

* Trig functions (sin, cos, tan) will likely be helpful in determining the x and y components of an angle. They can be found in the **math** library and used as follows. (Note that these fucntions expect radians rather than degrees, and there are handy functions to convert between them.)

import math // used for sin, cos, and M\_PI

...

speed = 10

angle = 60

dx = math.cos(math.radians(angle)) \* speed

dy = math.sin(math.radians(angle)) \* speed

* You can draw a circle with a number inside it with code similar to this:

arcade.draw\_circle\_outline(self.center.x, self.center.y, self.radius, TARGET\_COLOR)

text\_x = self.center.x - (self.radius / 2)

text\_y = self.center.y - (self.radius / 2)

arcade.draw\_text(repr(self.lives), text\_x, text\_y, TARGET\_COLOR, font\_size=20)