Hyper Star Shooter

Isaiah Ramirez, Kuba Ferguson

Forming a basis/starting point

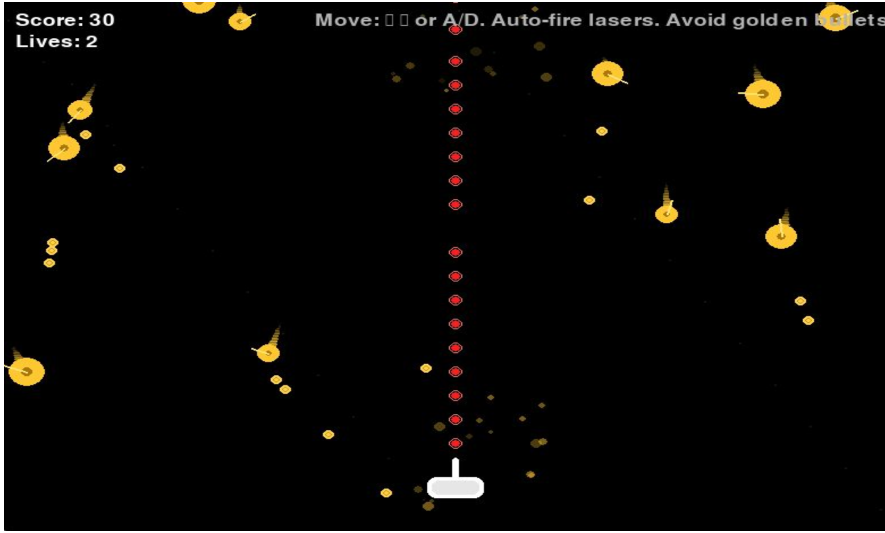
Inspiration:

Galaga and fruit ninja

- Formed a Galaga like game with falling star and a shooter shooting at these stars.

Chat can have some issues:

- Game would start lagging seconds into the game
- Clearly those are not stars
- The “stars” shooting out stuff was a cool surprise, not exactly what we were aiming for



Score: 30
Lives: 2

Move: □ □ or A/D. Auto-fire lasers. Avoid golden bullets

Process and learning along the way

Important Terms:

Pygame - turns Python into a mini game engine, letting you create interactive graphics and animations.

Alpha - How transparent a color is

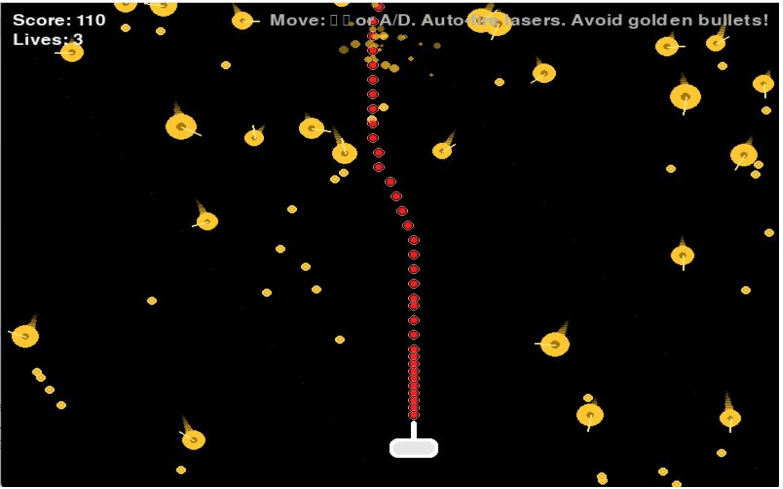
Alpha blending - Combining transparency with whatever was drawn on screen

1st version:

Alpha blending caused visual clutter, lag, and over-complicated code

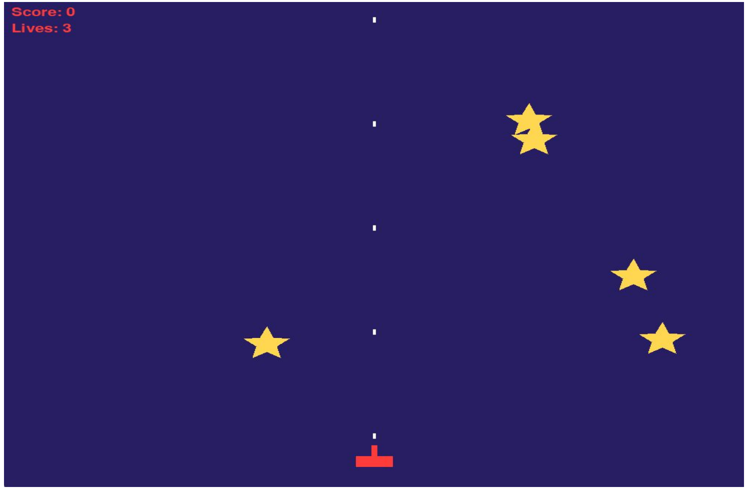
New Version:

- Removed all alpha Transparency
- Reduced amount of moving parts
- Cut over 100 lines of code
- Runs cleanly (No lag)



Score: 110
Lives: 3

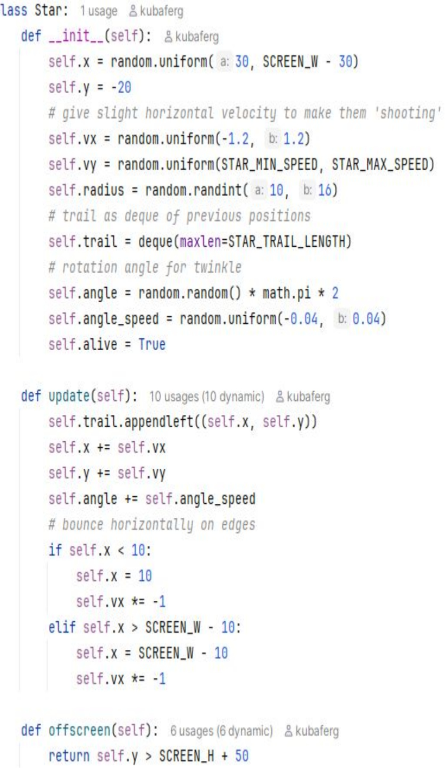
Move: □ □ or A/D. Auto-fire lasers. Avoid golden bullets!



Score: 0
Lives: 3

Condensing it

Star class (What star does)



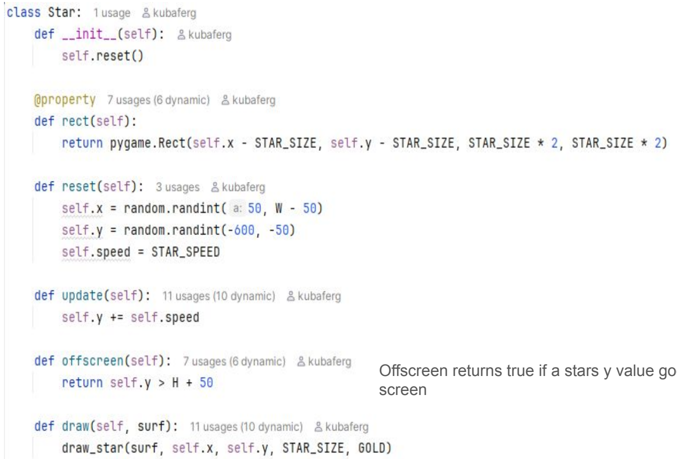
```
lass Star:  1 usage  kubaferg
  def __init__(self):  kubaferg
      self.x = random.uniform( a: 30, SCREEN_W - 30)
      self.y = -20
      # give slight horizontal velocity to make them 'shooting'
      self.vx = random.uniform(-1.2,  b: 1.2)
      self.vy = random.uniform(STAR_MIN_SPEED, STAR_MAX_SPEED)
      self.radius = random.randint( a: 10,  b: 16)
      # trail as deque of previous positions
      self.trail = deque(maxlen=STAR_TRAIL_LENGTH)
      # rotation angle for twinkle
      self.angle = random.random() * math.pi * 2
      self.angle_speed = random.uniform(-0.04,  b: 0.04)
      self.alive = True

  def update(self):  10 usages (10 dynamic)  kubaferg
      self.trail.appendleft((self.x, self.y))
      self.x += self.vx
      self.y += self.vy
      self.angle += self.angle_speed
      # bounce horizontally on edges
      if self.x < 10:
          self.x = 10
          self.vx *= -1
      elif self.x > SCREEN_W - 10:
          self.x = SCREEN_W - 10
          self.vx *= -1

  def offscreen(self):  6 usages (6 dynamic)  kubaferg
      return self.y > SCREEN_H + 50
```

Terms:

@Property - Basically allows you to create a method that behaves like a variable, (update/ changes values automatically instead of having to calculate them manually over and over)

Pygame.rect - rectangle object used for position, size, and in this case to sense collision (hit-box)

VS



```
class Star:  1 usage  kubaferg
    def __init__(self):  kubaferg
        self.reset()

    @property  7 usages (6 dynamic)  kubaferg
    def rect(self):
        return pygame.Rect(self.x - STAR_SIZE, self.y - STAR_SIZE, STAR_SIZE * 2, STAR_SIZE * 2)

    def reset(self):  3 usages  kubaferg
        self.x = random.randint( a: 50, W - 50)
        self.y = random.randint(-600, -50)
        self.speed = STAR_SPEED

    def update(self):  11 usages (10 dynamic)  kubaferg
        self.y += self.speed

    def offscreen(self):  7 usages (6 dynamic)  kubaferg
        return self.y > H + 50

    def draw(self, surf):  11 usages (10 dynamic)  kubaferg
        draw_star(surf, self.x, self.y, STAR_SIZE, GOLD)
```

Offscreen returns true if a stars y value goes off screen

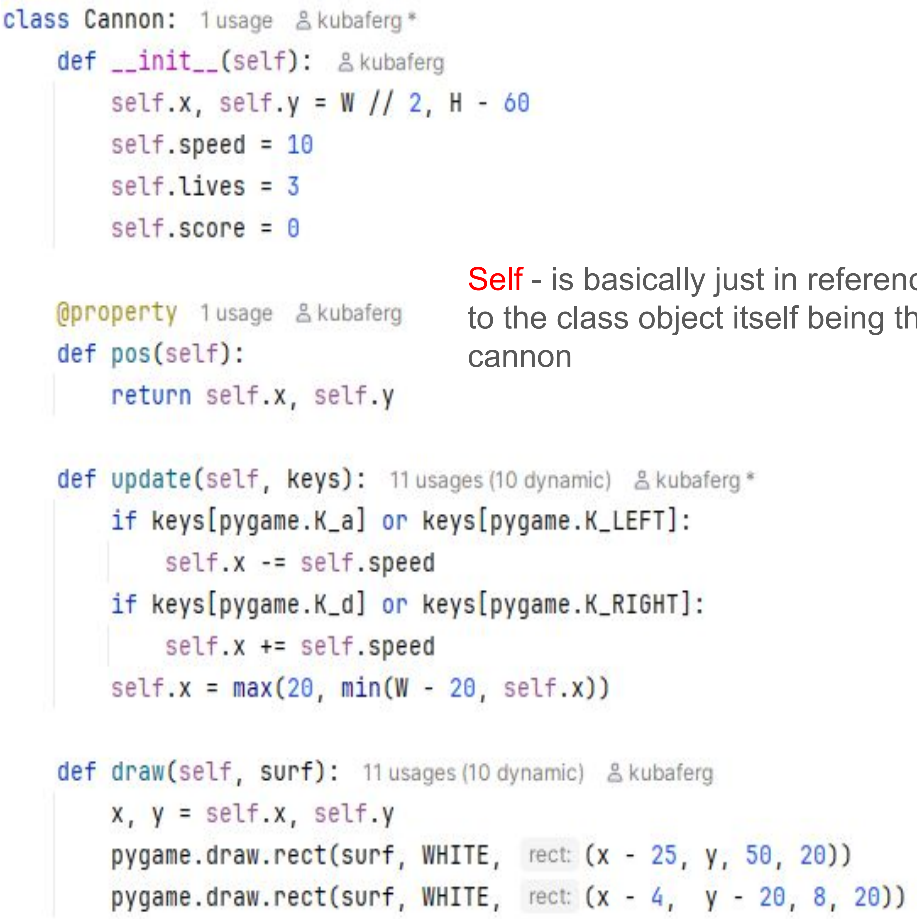
Old:

- Too busy adding trails to “stars”, making them go in all directions, making them twinkle, and trying to keep track of recent movements for the trail to takeover

New:

- Makes stars fall straight down
- adds rectangle around star that senses collison with laser
- speed of star is based on difficulty chosen
- returns true if a star has fallen to bottom of screen, taking a life away

Cannon



```
class Cannon:  1 usage  kubaferg *
    def __init__(self):  kubaferg
        self.x, self.y = W // 2, H - 60
        self.speed = 10
        self.lives = 3
        self.score = 0

    @property  1 usage  kubaferg
    def pos(self):
        return self.x, self.y

    def update(self, keys):  11 usages (10 dynamic)  kubaferg *
        if keys[pygame.K_a] or keys[pygame.K_LEFT]:
            self.x -= self.speed
        if keys[pygame.K_d] or keys[pygame.K_RIGHT]:
            self.x += self.speed
        self.x = max(20, min(W - 20, self.x))

    def draw(self, surf):  11 usages (10 dynamic)  kubaferg
        x, y = self.x, self.y
        pygame.draw.rect(surf, WHITE,  rect: (x - 25, y, 50, 20))
        pygame.draw.rect(surf, WHITE,  rect: (x - 4,  y - 20, 8, 20))
```

Self - is basically just in reference to the class object itself being the cannon

The cannon/shooter:

1st section:

- Places cannon in bottom center of the screen
- Sets initial parameters for speed, lives and score (state variables)

2nd section:

- Provides cannon’s current x, y coordinates as a tuple

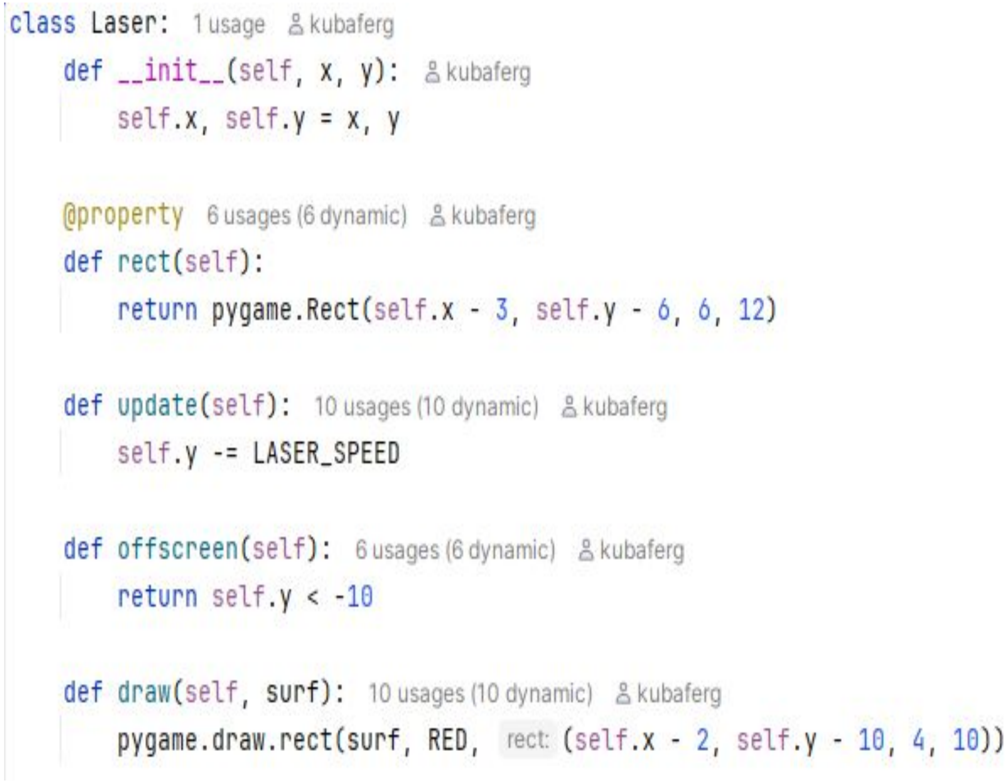
3rd section:

- Establishes which keys to press to move cannon left to right and if they’re currently being pressed
- Adds boundary constraints so that cannon won’t be able to move offscreen

4th section:

- Draws cannon on surface (surf), top is base and bottom is barrel
- Surf: refers to main window for game

Laser shots



```
class Laser:  1 usage  kubaferg
    def __init__(self, x, y):  kubaferg
        self.x, self.y = x, y

    @property  6 usages (6 dynamic)  kubaferg
    def rect(self):
        return pygame.Rect(self.x - 3, self.y - 6, 6, 12)

    def update(self):  10 usages (10 dynamic)  kubaferg
        self.y -= LASER_SPEED

    def offscreen(self):  6 usages (6 dynamic)  kubaferg
        return self.y < -10

    def draw(self, surf):  10 usages (10 dynamic)  kubaferg
        pygame.draw.rect(surf, RED,  rect: (self.x - 2, self.y - 10, 4, 10))
```

1st Section:

- Sets starting position for laser (basically where cannon is)

2nd section:

- Creates hitbox for laser shots to sense collision with star, #'s are pixels for size of laser

3rd section:

- Moves laser upward in movement with certain speed, in pygame decreasing y moves things upward
- Overall controls the laser's movement each time the game updates.

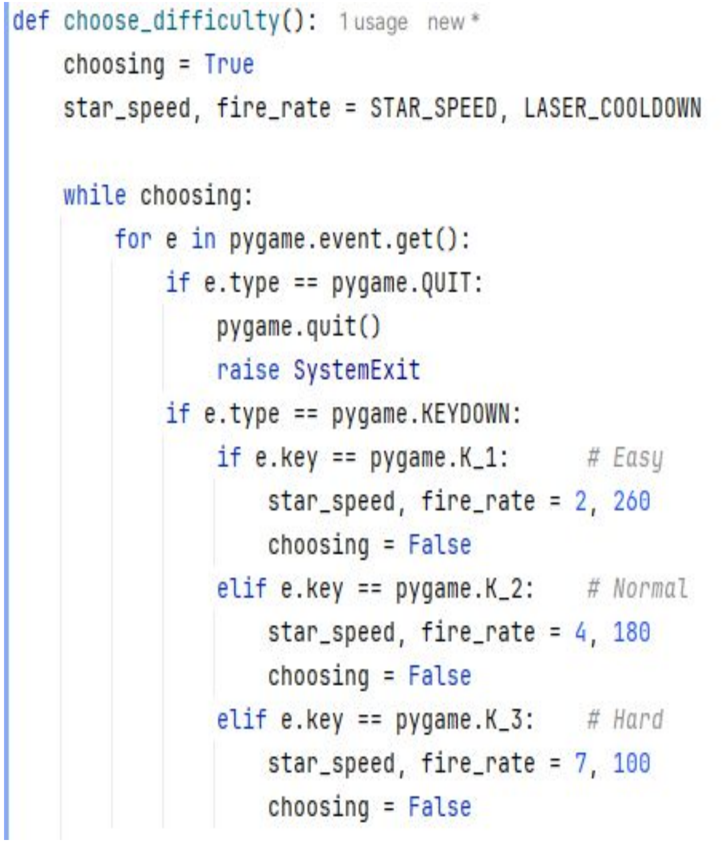
4th section:

- Returns True when laser has reached beyond top boundary of screen, then removes that laser since it is no longer visible

5th section:

- Draws the actual laser(s)

User choosing difficulty



```
def choose_difficulty():  1 usage  new *
    choosing = True
    star_speed, fire_rate = STAR_SPEED, LASER_COOLDOWN

    while choosing:
        for e in pygame.event.get():
            if e.type == pygame.QUIT:
                pygame.quit()
                raise SystemExit
            if e.type == pygame.KEYDOWN:
                if e.key == pygame.K_1:      # Easy
                    star_speed, fire_rate = 2, 260
                    choosing = False
                elif e.key == pygame.K_2:    # Normal
                    star_speed, fire_rate = 4, 180
                    choosing = False
                elif e.key == pygame.K_3:    # Hard
                    star_speed, fire_rate = 7, 100
                    choosing = False
```

1st section:

- Stays on menu until user chooses difficulty level
- Initializes variables that will change based on users chosen difficulty level

2nd chunk:

- Uses a loop that continuously checks for keyboard input
- Once certain key is pressed for difficulty level, variables will adjust accordingly

What can be expanded on / Other options?

- Stars could shoot back
- As score gets higher, cannon gets bigger
- Custom cannon (Color, shape, etc.)
- Adding a trail to each star to become shooting stars\
- More action to star movement
- Game changers (Ex: a bomb dropping that could explode all stars)
- Add star that requires more shots hit