Tutorial: Missile help

```
#!/usr/bin/env python

import pygame, sys, time, random, bres
from pygame.locals import *

ramp_one, ramp_two, ramp_three = None, None, None

wood_light = (166, 124, 54)
wood_dark = (76, 47, 0)
blue = (0, 100, 255)
dark_red = (166, 25, 50)
dark_green = (25, 100, 50)
dark_blue = (25, 50, 150)
black = (0, 0, 0)
white = (255, 255, 255)
yellow = (240, 230, 140)
grey = (153, 153, 153)

width, height = 1024, 768
screen = None


maxRadius = 60
allObjects = []
```

```
delay = 15   # number of milliseconds delay before generating a USEREVENT
missileSize = 3

silos = [[80, 700], [500, 700], [1000, 700]]


def sqr (x):
    return x*x

class explosion:
    def __init__ (self, pos, colour):
        self._radius = 1
        self._maxRadius = maxRadius
        self._increasing = True
        self._pos = pos
        self._colour = colour
    def update (self):
        if self._increasing:
            pygame.draw.circle (screen, self._colour, self._pos, self._radius, 0)
            self._radius += 1
            if self._radius == self._maxRadius:
                self._increasing = False
        else:
            pygame.draw.circle (screen, black, self._pos, self._radius, 0)
            self._radius -= 1
            if self._radius > 0:
                pygame.draw.circle (screen, self._colour, self._pos, self._radius, 0)
            else:
                globalRemove (self)
    def erase (self):
        pygame.draw.circle (screen, black, self._pos, self._radius, 0)
    def ignite (self, p):
        return sqr (self._pos[0]-p[0]) + sqr (self._pos[1]-p[1]) < sqr (self._radius)
```

```
def drawTrail (p):
    pygame.draw.rect (screen, white, (p[0], p[1], missileSize, missileSize), 0)

def drawMissile (p):
    pygame.draw.rect (screen, yellow, (p[0], p[1], missileSize, missileSize), 0)

def eraseBlock (p):
    pygame.draw.rect (screen, black, (p[0], p[1], missileSize, missileSize), 0)


class missile:
    def __init__ (self, start_pos, end_pos):
        self.route = bres.bres (start_pos, end_pos)
        self.erase_route = bres.bres (start_pos, end_pos)
    def update (self):
        if self.route.finished ():
            globalRemove (self)
            createExplosion (self.route.get_current_pos (), white)
        elif ignites (self.route.get_current_pos ()):
            createExplosion (self.route.get_current_pos (), grey)
        drawTrail (self.route.get_current_pos ())
        drawMissile (self.route.get_next ())
    def erase (self):
        while not self.erase_route.finished ():
            eraseBlock (self.erase_route.get_next ())
    def ignite (self, p):
        return False


def ignites (p):
    for o in allObjects:
        if o.ignite (p):
            return True
    return False
```

```
def createMissile (start_pos, end_pos):
    global allObjects
    allObjects += [missile (start_pos, end_pos)]
    pygame.time.set_timer (USEREVENT+1, delay)

def createExplosion (pos, colour):
    global allObjects
    allObjects += [explosion (pos, colour)]
    pygame.time.set_timer (USEREVENT+1, delay)


def globalRemove (e):
    global allObjects
    e.erase ()
    allObjects.remove (e)
    pygame.display.flip ()


def updateAll ():
    if allObjects != []:
        for e in allObjects:
            e.update ()
    if allObjects != []:
        pygame.display.flip ()
        pygame.time.set_timer (USEREVENT+1, delay)


def wait_for_event ():
    global screen
    while True:
        event = pygame.event.wait ()
        if event.type == pygame.QUIT:
            sys.exit(0)
```

```
        if event.type == KEYDOWN and event.key == K_ESCAPE:
            sys.exit (0)
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button >= 1 and event.button <= 3:
                createMissile (silos[event.button-1], pygame.mouse.get_pos ())
        if event.type == USEREVENT+1:
            updateAll ()


def main ():
    global screen
    pygame.init ()
    screen = pygame.display.set_mode ([width, height])
    wait_for_event ()


main ()
```

Tutorial

- extend your missile command program to include a `city` class
- give your city class an `__init__`, `update`, `ignite`, `erase` and `check` method
- the method prototypes are:

Tutorial

- ```
    # create a city at pos
    # calculcate the epicenter of the city
    # store it in the class
    def __init__ (self, pos):
    # draw the city
    def draw_city (self):
    # remove the city
    def erase (self):
    # determine whether city should catch fire given exposion at p with a radius
    def check (self, p, radius):
  ```

- extend your game to include cities and their destuction!

- now create a `gun` class (which will be very similar to the city class)
  ```