# Pygame - Dev Notes - Game Extras - Explosions

A brief intro on adding fun options such as explosions to a game window with Pygame.

### Contents


- Intro
- Explosions for sprite objects
- Load explosion images
- Create explosion sprite object
- Add explosions to collisions
- Add explosions to player's ship
- Scale explosion images
- References
- Demo


### Intro

For an example game, such as *Space Invaders* or a similar classic STG, we may wish to add a sense of *explosions* for some of our game sprites.

### Explosions for sprite objects

To create realistic explosion effects for our sprite objects, we need to create or import a series of images for these explosions.

We may then use these defined images to create an animated sequence for each required explosion. In effect, as we've seen with standard animation, we specify a starting image and then cycle or animate our way through this set of images.

In our game, we may introduce this animation for each projectile collision with an asteroid, or perhaps as our player's ship finally runs out of lives and explodes.

These are fun extras that add a sense of achievement to the underlying challenge of destroying advancing asteroids or alien ships.

### Load explosion images

As we've done so far for other game sprite objects, we need to be able to define and load our images for the explosions. We'll use a list for these images, as we did for the random asteroids, and then we can cycle through these explosions as required.

So, our first example will use a list to simply load these explosion images. As we know the directory for these images, and the required number of images, we'll use a `for` loop to iterate over this directory and load our images, e.g.

```
# explosions
explosion_imgs = []

# iterate over explosion images in directory
for i in range(9):
    file = 'explosion{}.png'.format(i)
    # load image from os
    expl_img = pygame.image.load(os.path.join(img_dir, file)).convert()
    # set colour key for image
    expl_img.set_colorkey(BLACK)
    # append to specified list for explosion images
```

```
    explosion_imgs.append(expl_img)
```

As we loop through the directory of images, we can use the built-in function, `format()`, to specify an abstracted value for the iterator number, in this example, of the created filename. So, for each iteration, the value for the `{}` braces will be replaced with the index value of the iterator.

Then, we may create our image for the Pygame window, and set the colour key to black to create our transparency for the containing shape's background.

We can then append these images to our list for explosions.

### Create explosion sprite object

As we've done for other sprite objects, we may now create a new class to help us represent and organise our sprite object for *explosions*.

So, we'll add a new class for this object, and then start by initialising this sprite, e.g.

```
# create a generic explosion sprite - use for asteroids, player explosions &c.
class Explosion(pygame.sprite.Sprite):
    # initialise sprite
    def __init__(self, center):
        pygame.sprite.Sprite.__init__(self)
        ...
```

After initialising this new sprite object, we can set the starting image for our explosions to the first index position of our list for the explosion images. Then, we need to add the rectangle for this image, and set its centre to the specified value of the pass parameter.

We also need to set the initial frame for our animation, which we can set to a starting default of `0`.

As we're working with an animation, which is cycling through images, we need to try to make this steady and constant. One way to achieve this desired animation effect is to create a steady framerate for the animation itself. So, we may now check the time in ticks for the last update, following the pattern we've seen earlier in our game.

Then, we can set a default framerate for this animation. We can test this animation, and modify this framerate as required.

```
# create a generic explosion sprite - use for asteroids, player explosions &c.
class Explosion(pygame.sprite.Sprite):
    # initialise sprite
    def __init__(self, center):
        pygame.sprite.Sprite.__init__(self)
        # specify image for explosion sprite
        self.image = explosion_imgs[0]
        # set rect for image
        self.rect = self.image.get_rect()
        self.rect.center = center
        # set initial frame for animation
        self.frame = 0
        # check last update to animation
        self.last_update = pygame.time.get_ticks()
        # set framerate delay between animation frames - sets speed for explosion
        self.frame_rate = 50
```

Then, we need to add an `update` function to our class, which will allow us to update the image of the explosion for this sprite object as time progresses. i.e. as the framerate advances, we can switch the explosion images to create the animation itself.

```
...
# change image as time progresses for explosion sprite
```

```
def update(self):
    # get current time
    now = pygame.time.get_ticks()
    # check if enough time has passed between animations
    if now - self.last_update > self.frame_rate:
        self.last_update = now
        # if enough time passed - add 1 to frame
        self.frame += 1
        # check if end of explosion images reached
        if self.frame == len(explosion_imgs):
            # kill if end of image reached
            self.kill()
        else:
            center = self.rect.center
            self.image = explosion_imgs[self.frame]
            # update rect for image
            self.rect = self.image.get_rect()
            self.rect.center = center
```

In this `update` function, we need to check the current time in the game, which then allows us to check if enough time has passed between each animation. If enough time has elapsed, we can update the value for the `last_update` time record, and advance our animation frame by an increment of 1. i.e. we can move to the next available image in the series of explosions.

If we reach the end of these explosions images, i.e. when enough frames have passes, we can then end or `kill()` this animation for the explosions. If not, we can set the centre of our explosion image's rectangle, and update the image itself.

### Add explosions to collisions

We've now create our sprite object for explosions, but we still need to use this object in the logic of our game.

So, we can now call this explosion whenever we record a collision between, for example, a projectile and an asteroid.

In the game loop's update section, as we check for collisions we can now add an animation for the explosions. e.g.

```
...
# add more mobs for those hit and deleted by projectiles
for collision in collisions:
    # calculate points relative to size of mob object
    game_score += 40 - collision.radius
    # play explosion sound effect for collision
    explosion_effect.play()
    # add animation for explosion images if collision
    explosion = Explosion(collision.rect.center)
    # add explosion sprite to game sprites group
    game_sprites.add(explosion)
    # create a new mob object
    createMob()
...
```

As we're checking for collisions, we can now create the animation for the explosions after we play the sound effect for each explosion.

### Add explosions to player's ship

Adding these explosions to another sprite object, such as collisions against the player's ship, is as simple as updating the game loop again. So, as we decrease the relative level for the shield of our player's ship, we can then trigger an explosion to reinforce this collision for our player. e.g.

```
# add check for collision - enemy and player sprites (True = hit object is now
```

```
deleted from game window)
collisions = pygame.sprite.spritecollide(player, mob_sprites, True,
pygame.sprite.collide_circle)
# check collisions with player's ship - decrease shield for each hit
for collision in collisions:
    # decrease player's shield for each collision
    player.stShield -= 20
    # add animation for explosion images if collision
    explosion = Explosion(collision.rect.center)
    # add explosion sprite to game sprites group
    game_sprites.add(explosion)
    # create a new mob object
    createMob()
    # check overall shield value - quit game if no shield
    if player.stShield <= 0:
        running = False
```

### Scale explosion images

Our explosions are now being shown for both initial types of collision within our game. As a projectile hits a mob object, and then as a cascading mob object strikes the shield of our player's ship.

However, there is still a lingering issue with these explosions that is not reinforcing the gameplay for our shooter style game. Effectively, there is no differentiation in the relative size of an explosion, and therefore no semblance of feedback to our player.

So, we might add a standard scale transform to the image for each explosion sprite object,

```
# explosions
explosion_imgs = []

# iterate over explosion images in directory
for i in range(9):
    file = 'explosion{}.png'.format(i)
    # load image from os
    expl_img = pygame.image.load(os.path.join(img_dir, file)).convert()
    # set colour key for image
    expl_img.set_colorkey(BLACK)
    # append to specified list for explosion images
    explosion_imgs.append(expl_img)
```

This gives us a smaller, less overwhelming explosion for each mob object, and collision against the player's ship.

However, it would also be useful to be able to scale these explosions relative to the actual size of a given sprite object. So, a smaller relative explosion image for a smaller mob object, and likewise for a collision against the player's ship.

The first thing we need to do is update our class for the `Explosion` object, which will allow us to dynamically modify each explosion image in the animation relative to a specified size. In effect, we can scale each frame of the explosion animation to match the size of the collision object.

So, we can update this class as follows,

```
# create a generic explosion sprite - use for asteroids, player explosions &c.
class Explosion(pygame.sprite.Sprite):
    # initialise sprite
    def __init__(self, center, size):
        pygame.sprite.Sprite.__init__(self)
        # specify size for explosion sprite
        self.size = size
        # get initial image for explosion
        self.image = pygame.transform.scale(explosion_imgs[0], self.size)
...
```

We'll start by adding a parameter for `size`, which allows us to pass a variable size for each collision object. We can then use this size to scale the initial image for the explosion animation.

As we update this object, each frame of the animation will also require scaling of the explosion image. e.g.

```
# change image as time progresses for explosion sprite
def update(self):
    # get current time
    now = pygame.time.get_ticks()
    # check if enough time has passed between animations
    if now - self.last_update > self.frame_rate:
        self.last_update = now
        # if enough time passed - add 1 to frame
        self.frame += 1
        # check if end of explosion images reached
        if self.frame == len(explosion_imgs):
            # kill if end of image reached
            self.kill()
        else:
            center = self.rect.center
            self.image = pygame.transform.scale(explosion_imgs[self.frame],
self.size)
            # update rect for image
            self.rect = self.image.get_rect()
            self.rect.center = center
```

The key update is in the block of code for the `if/else` conditional statement. As we output each frame of the explosion animation, we may then scale this image to match the passed `size` for the explosion object.

So, different size mob objects will have a matching explosion animation, which we may update in the game loop, e.g.

```
# add check for sprite group collide with another sprite group - projectiles hitting
enemy objects - use True to delete sprites from each group...
collisions = pygame.sprite.groupcollide(mob_sprites, projectiles, True, True)
# add more mobs for those hit and deleted by projectiles
for collision in collisions:
    # calculate points relative to size of mob object
    game_score += 40 - collision.radius
    # play explosion sound effect for collision
    explosion_effect.play()
    # get size of collision object
    col_size = collision.rect.size
    #print("collision size = " + str(col_size))
    # add animation for explosion images if collision
    explosion = Explosion(collision.rect.center, col_size)
    # add explosion sprite to game sprites group
    game_sprites.add(explosion)
    # create a new mob object
    createMob()
```

And, the same for the player's sprite object as well.

### References

- pygame.image
- pygame.sprite
- pygame.time
- pygame.transform

### Demo

- objectexplosions1.py
- shooter1.2.py
  - add some fun explosions
    - create sprite object for explosion
    - cycle through images to create explosion animation
    - add explosion for each collision
  - extra explosions
    - explode a player's ship for a collision
  - scale explosions
    - rescale and size explosions in game window