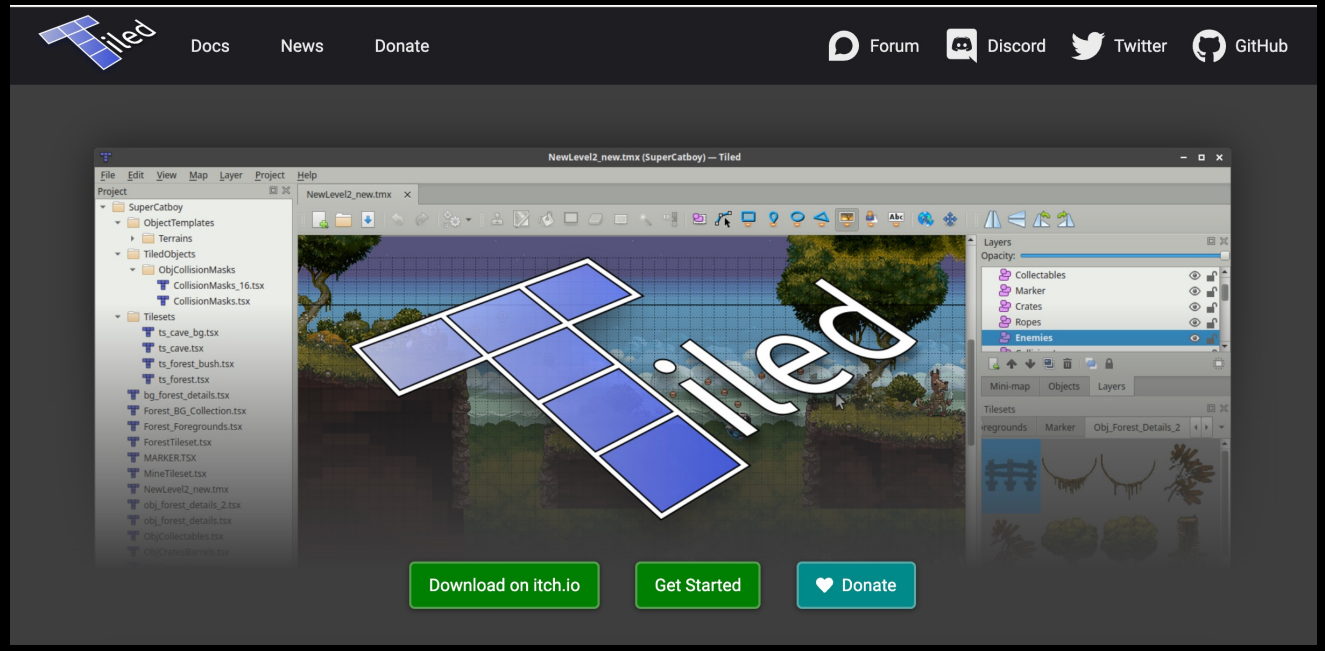
Project4:
Tile-Based Topdown Shooter

1. Creating the Map

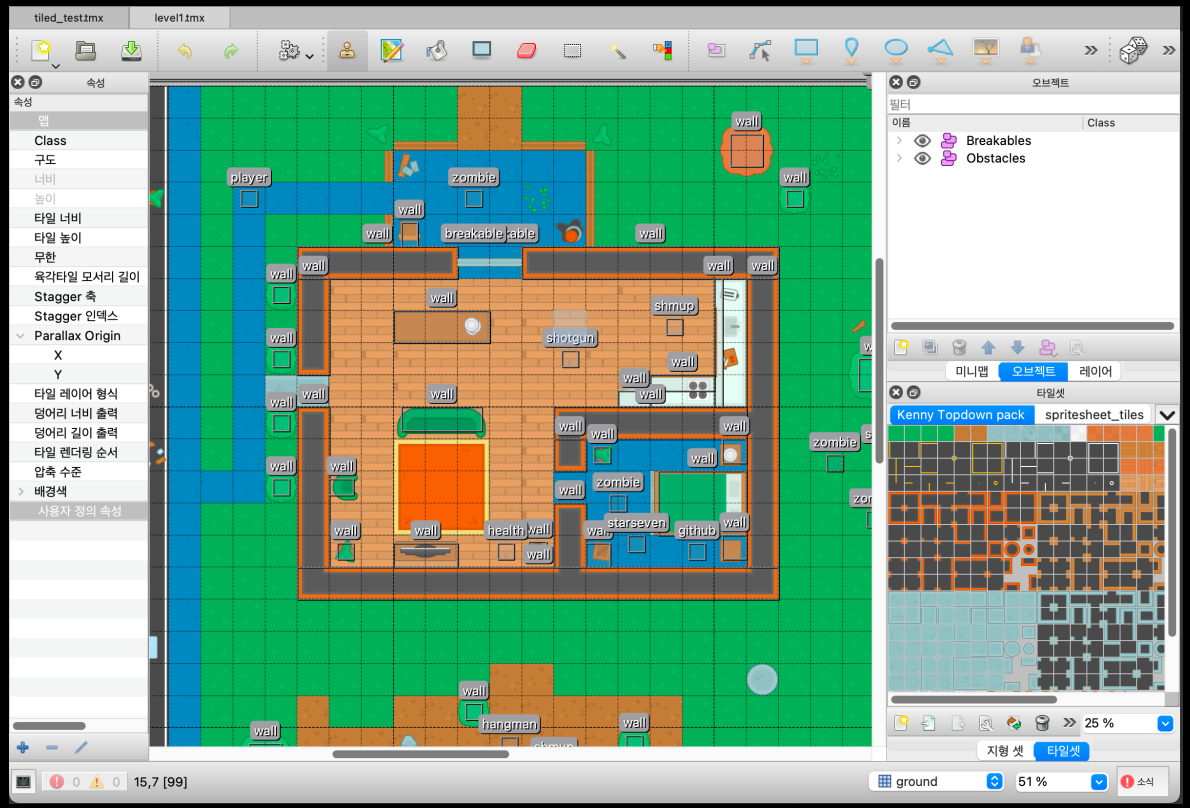
https://www.mapeditor.org/

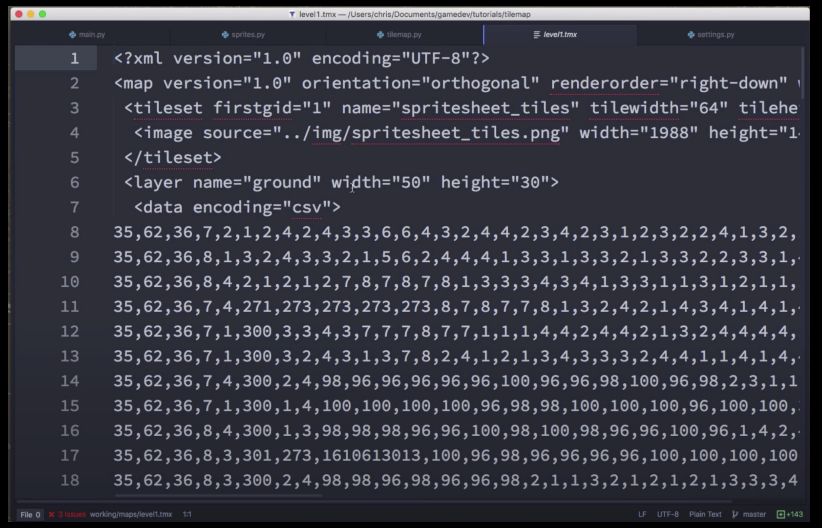


Creates a tmx file

Which can be read in python

Using the pytmx package


tiled_test.tmx
level1.tmx
속성
맵
Class
구도
너비
높이
타일 너비
타일 높이
무한
육각타일 모서리 길이
Stagger 축
Stagger 인덱스
Parallax Origin
X
Y
타일 레이어 형식
덩어리 너비 출력
덩어리 길이 출력
타일 렌더링 순서
압축 수준
배경색
사용자 정의 속성
오브젝트
필터
이름
Class
Breakables
Obstacles
미니맵
오브젝트
레이어
타일셋
Kenny Topdown pack
spritesheet_tiles
25 %
지형 셋
타일셋
player
zombie
wall
breakable table
shotgun
shmup
health
starseven
github
hangman
15,7 [99]
ground
51 %


level1.tmx — /Users/chris/Documents/gamedev/tutorials/tilemap
main.py
sprites.py
tilemap.py
level1.tmx
settings.py
1 <?xml version="1.0" encoding="UTF-8"?>
2 <map version="1.0" orientation="orthogonal" renderorder="right-down"
3 <tileset firstgid="1" name="spritesheet_tiles" tilewidth="64" tilehe
4 <image source="../img/spritesheet_tiles.png" width="1988" height="1
5 </tileset>
6 <layer name="ground" width="50" height="30">
7 <data encoding="csv">
8 35,62,36,7,2,1,2,4,2,4,3,3,6,6,4,3,2,4,4,2,3,4,2,3,1,2,3,2,2,4,1,3,2,
9 35,62,36,8,1,3,2,4,3,3,2,1,5,6,2,4,4,4,1,3,3,1,3,3,2,1,3,3,2,2,3,3,1,
10 35,62,36,8,4,2,1,2,1,2,7,8,7,8,7,8,1,3,3,3,4,3,4,1,3,3,1,1,3,1,2,1,1,
11 35,62,36,7,4,271,273,273,273,273,8,7,8,7,7,8,1,3,2,4,2,1,4,3,4,1,4,1,
12 35,62,36,7,1,300,3,3,4,3,7,7,7,8,7,7,1,1,1,4,4,2,4,4,2,1,3,2,4,4,4,4,
13 35,62,36,7,1,300,3,2,4,3,1,3,7,8,2,4,1,2,1,3,4,3,3,3,2,4,4,1,1,4,1,4,
14 35,62,36,7,4,300,2,4,98,96,96,96,96,96,100,96,96,98,100,96,98,2,3,1,1
15 35,62,36,7,1,300,1,4,100,100,100,100,96,98,98,100,100,100,96,100,100,
16 35,62,36,8,4,300,1,3,98,98,98,96,96,100,98,100,98,96,96,100,96,1,4,2,
17 35,62,36,8,3,301,273,1610613013,100,96,98,96,96,96,96,100,100,100,100
18 35,62,36,8,3,300,2,4,98,98,96,98,96,96,98,2,1,1,3,2,1,2,1,2,1,3,3,3,4

```python
class TiledMap:
    def __init__(self, filename):
        tm = pytmx.load_pygame(filename, pixelalpha=True)
        self.width = tm.width * tm.tilewidth
        self.height = tm.height * tm.tileheight
        self.tmxdata = tm

    def render(self, surface):
        ti = self.tmxdata.get_tile_image_by_gid
        for layer in self.tmxdata.visible_layers:
            if isinstance(layer, pytmx.TiledTileLayer):
                for x, y, gid, in layer:
                    tile = ti(gid)
                    if tile:
                        tile.set_colorkey(BLACK)
                        surface.blit(tile, (x * self.tmxdata.tilewidth,
                                            y * self.tmxdata.tileheight))

    def make_map(self):
        temp_surface = pg.Surface((self.width, self.height))
        self.render(temp_surface)
        return temp_surface
```

```python
self.map = TiledMap(path.join(self.map_folder, 'level1.tmx'))
self.map_img = self.map.make_map()
self.map.rect = self.map_img.get_rect()
for tile_object in self.map.tmxdata.objects:
    obj_center = vec(tile_object.x + tile_object.width / 2,
                     tile_object.y + tile_object.height / 2)
    if tile_object.name == 'player':
        self.player = Player(self, obj_center.x, obj_center.y)
    if tile_object.name == 'zombie':
        Mob(self, obj_center.x, obj_center.y)
    if tile_object.name == 'sun':
        Sun(self, obj_center.x, obj_center.y)
    if tile_object.name == 'numpy':
        Numpy(self, obj_center.x, obj_center.y)
    if tile_object.name == 'windmill':
        Windmill(self, obj_center.x, obj_center.y)
    if tile_object.name == 'hangman':
        Hangman(self, obj_center.x, obj_center.y)
    if tile_object.name == 'wall':
        Obstacle(self, tile_object.x, tile_object.y,
                 tile_object.width, tile_object.height)
```

Map will be rendered into image, and instances in will automatically be located where they were placed during map design Ex)

Map Scrolling


```python
class Camera:
    def __init__(self, width, height):
        self.camera = pg.Rect(0, 0, width, height)
        self.width = width
        self.height = height

    def apply(self, entity):
        return entity.rect.move(self.camera.topleft)

    def apply_rect(self, rect):
        return rect.move(self.camera.topleft)

    def update(self, target):
        x = -target.rect.centerx + int(WIDTH / 2)
        y = -target.rect.centery + int(HEIGHT / 2)

        # limit scrolling to map size
        x = min(0, x)  # left
        y = min(0, y)  # top
        x = max(-(self.width - WIDTH), x)  # right
        y = max(-(self.height - HEIGHT), y)  # bottom
        self.camera = pg.Rect(x, y, self.width, self.height)
```

It looks like there is camera following the moving player,
but the player is actually still, and the map around it is shifting
'Camera' follows a target, and its left top location updated a certain amount depending on where the target is. Then it applies the movement to the rest of the entities

Improved Move/Shoot Controls

```python
def get_keys(self):
    self.rot_speed = 0
    self.vel = vec(0, 0)
    keys = pg.key.get_pressed()
    if keys[pg.K_LEFT] or keys[pg.K_a]:
        self.rot_speed = PLAYER_ROT_SPEED
    if keys[pg.K_RIGHT] or keys[pg.K_d]:
        self.rot_speed = -PLAYER_ROT_SPEED
    if keys[pg.K_UP] or keys[pg.K_w]:
        self.vel = vec(PLAYER_SPEED, 0).rotate(-self.rot)
    if keys[pg.K_DOWN] or keys[pg.K_s]:
        self.vel = vec(-PLAYER_SPEED / 2, 0).rotate(-self.rot)
    if keys[pg.K_SPACE]:
        self.shoot()
```

```python
def get_keys(self):
    self.rot_speed = 0
    self.vel = vec(0, 0)
    keys = pg.key.get_pressed()
    mouse = pg.mouse.get_pressed()
    if keys[pg.K_a]:
        self.vel = vec(-PLAYER_SPEED,0)
    if keys[pg.K_d]:
        self.vel = vec(PLAYER_SPEED,0)
    if keys[pg.K_w]:
        self.vel = vec(0, -PLAYER_SPEED)#.rotate(-self.rot)
    if keys[pg.K_s]:
        self.vel = vec(0, +PLAYER_SPEED)#.rotate(-self.rot)
    if mouse[0] == 1:
        #print(1)
        self.shoot()
```

The original game used L, R arrow keys to rotate the player,
And U D arrow keys to move it front or back depending on its relative angle

However, this is very hard to control -> Change to WASD Movement, Mouse to shoot

Rotation Method While Scrolling

```
def update(self):
    self.get_keys()
    mouse_pos = pg.mouse.get_pos()
    #print(mouse_pos)
    actual_mouse = (mouse_pos[0] - self.game.camera.camera.x, mouse_pos[1] - self.game.camera.camera.y)
    #print(mouse_pos, actual_mouse)
    self.mouse_angle = math.atan2(actual_mouse[1] - (self.pos[1]), actual_mouse[0] -self.pos[0]) #+32 +26
    self.rot = (360 - self.mouse_angle*57.29)%360
    #self.rot = (self.rot + self.rot_speed * self.game.dt) % 360
    self.image = pg.transform.rotate(self.game.player_img, self.rot)
```

Player will rotate depending on where the mouse is
However, the mouse coordinates returned by pg.mouse.get_pos() only return the relative location of the mouse on the current VISIBLE SCREEN.
Therefore we must subtract the cameras' current topleft x,y coordinates in order to make it work while scrolling

Settings.py

```python
# Player settings
PLAYER_HEALTH = 100
PLAYER_SPEED = 350
PLAYER_ROT_SPEED = 200
PLAYER_IMG = 'student.png'
PLAYER_HIT_RECT = pg.Rect(0, 0, 35, 35)
BARREL_OFFSET = vec(30, 30)

# Weapon settings
BULLET_IMG = 'bullet.png'
WEAPONS = {}
WEAPONS['pistol'] = {'bullet_speed': 500,
                     'bullet_lifetime': 1000,
                     'rate': 150,
                     'kickback': 50,
                     'spread': 5,
                     'damage': 6,
                     'bullet_size': 'lg',
                     'bullet_count': 1}
WEAPONS['shotgun'] = {'bullet_speed': 400,
                      'bullet_lifetime': 500,
                      'rate': 900,
                      'kickback': 100,
                      'spread': 20,
                      'damage': 5,
                      'bullet_size': 'sm',
                      'bullet_count': 12}
WEAPONS['machine gun'] = {'bullet_speed': 600,
                      'bullet_lifetime': 800,
                      'rate': 50,
                      'kickback': 60,
                      'spread': 8,
                      'damage': 5.5,
                      'bullet_size': 'lg',
                      'bullet_count': 1}

# Mob settings

MOBS = {}
MOBS['zombie'] = {
    'img' : 'pysnake.png',
    'speed' : [150, 100, 75, 125],
    'hit_rect' : pg.Rect(0, 0, 30, 30),
    'health' : 75,
    'damage' : 10,
    'knockback' : 20,
    'avoid_radius' : 50,
    'detect_radius' : 400
```

Contains all necessary info of game including sprite health, speed, gun damage etc

Sprites - Player

```
def new(self):
    # initialize all variables and do all the setup for a new game
    self.all_sprites = pg.sprite.LayeredUpdates()
    self.walls = pg.sprite.Group()
    self.mobs = pg.sprite.Group()
    self.mobList = []
    self.bullets = pg.sprite.Group()
    self.items = pg.sprite.Group()
```

```
class Player(pg.sprite.Sprite):
    def __init__(self, game, x, y):
        self._layer = PLAYER_LAYER
        self.groups = game.all_sprites
        pg.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.hit_rect = PLAYER_HIT_RECT
        self.hit_rect.center = self.rect.center
        self.vel = vec(0, 0)
        self.pos = vec(x, y)
        self.last_shot = 0
        self.health = PLAYER_HEALTH
        self.weapon = 'pistol'
        self.damaged = False
        self.mouse_pos = pg.mouse.get_pos()
        self.mouse_angle = math.atan2(self.mouse_pos[1] - (self.pos[1]+32), self.mouse_pos[0]-self.pos[0]+26)
        self.rot = 0


    def get_keys(self):
        self.rot_speed = 0
        self.vel = vec(0, 0)
        keys = pg.key.get_pressed()
        mouse = pg.mouse.get_pressed()
        if keys[pg.K_a]:
            self.vel = vec(-PLAYER_SPEED,0)
        if keys[pg.K_d]:
            self.vel = vec(PLAYER_SPEED,0)
        if keys[pg.K_w]:
            self.vel = vec(0, -PLAYER_SPEED)#.rotate(-self.rot)
        if keys[pg.K_s]:
            self.vel = vec(0, +PLAYER_SPEED)#.rotate(-self.rot)
        if mouse[0] == 1:
            #print(1)
            self.shoot()
```

(parameter) self: Self@Player

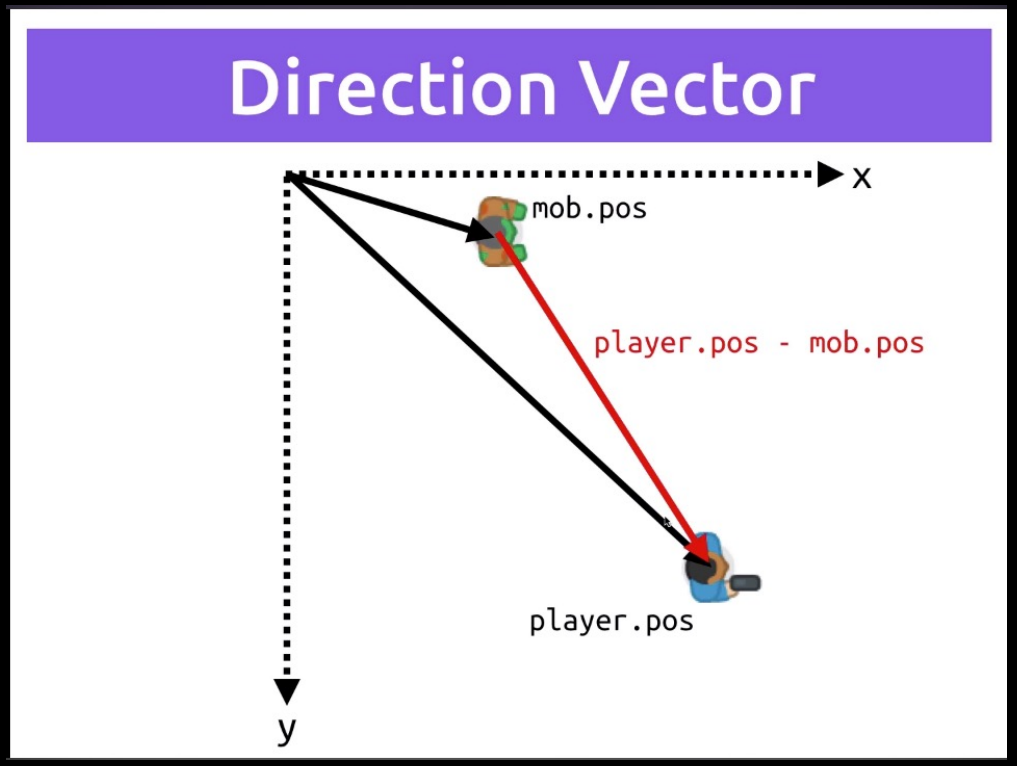
```
def shoot(self):
    now = pg.time.get_ticks()
    if now - self.last_shot > WEAPONS[self.weapon]['rate']:
        self.last_shot = now
        dir = vec(1, 0).rotate(-self.rot)
        pos = self.pos + BARREL_OFFSET.rotate(-self.rot)
        self.vel = vec(-WEAPONS[self.weapon]['kickback'], 0).rotate(-self.rot)
        for i in range(WEAPONS[self.weapon]['bullet_count']):
            spread = uniform(-WEAPONS[self.weapon]['spread'], WEAPONS[self.weapon]['spread'])
            Bullet(self.game, pos, dir.rotate(spread), WEAPONS[self.weapon]['damage'])
            snd = choice(self.game.weapon_sounds[self.weapon])
            if snd.get_num_channels() > 2:
                snd.stop()
            snd.play()
        MuzzleFlash(self.game, pos)

def hit(self):
    self.damaged = True
    self.damage_alpha = chain(DAMAGE_ALPHA * 4)

def update(self):
    self.get_keys()
    mouse_pos = pg.mouse.get_pos()
    #print(mouse_pos)
    actual_mouse = (mouse_pos[0] - self.game.camera.camera.x, mouse_pos[1] - self.game.camera.camera.y)
    #print(mouse_pos, actual_mouse)
    self.mouse_angle = math.atan2(actual_mouse[1] - (self.pos[1]), actual_mouse[0] -self.pos[0]) #+32 +26
    self.rot = (360 - self.mouse_angle*57.29)%360
    #self.rot = (self.rot + self.rot_speed * self.game.dt) % 360
    self.image = pg.transform.rotate(self.game.player_img, self.rot)
    if self.damaged:
        try:
            self.image.fill((255, 0, 0, next(self.damage_alpha)), special_flags=pg.BLEND_RGBA_MULT)
        except:
            self.damaged = False
    self.rect = self.image.get_rect()
    self.rect.center = self.pos
    self.pos += self.vel * self.game.dt
    self.hit_rect.centerx = self.pos.x
    collide_with_walls(self, self.game.walls, 'x')
    self.hit_rect.centery = self.pos.y
    collide_with_walls(self, self.game.walls, 'y')
    self.rect.center = self.hit_rect.center

def add_health(self, amount):
    self.health += amount
    if self.health > PLAYER_HEALTH:
        self.health = PLAYER_HEALTH
```

Sprites - Mob

```python
class Mob(pg.sprite.Sprite):
    def __init__(self, game, x, y):
        self._layer = MOB_LAYER
        self.groups = game.all_sprites, game.mobs
        pg.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.image = game.pysnake_img.copy()
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.hit_rect = MOB_HIT_RECT.copy()
        self.hit_rect.center = self.rect.center
        self.pos = vec(x, y)
        self.vel = vec(0, 0)
        self.acc = vec(0, 0)
        self.rect.center = self.pos
        self.rot = 0
        self.health = MOB_HEALTH
        self.speed = choice(MOB_SPEEDS)
        self.target = game.player
        self.game.mobList.append(self)

    def avoid_mobs(self):
        for mob in self.game.mobs:
            if mob != self:
                dist = self.pos - mob.pos
                if 0 < dist.length() < AVOID_RADIUS:
                    self.acc += dist.normalize()
```

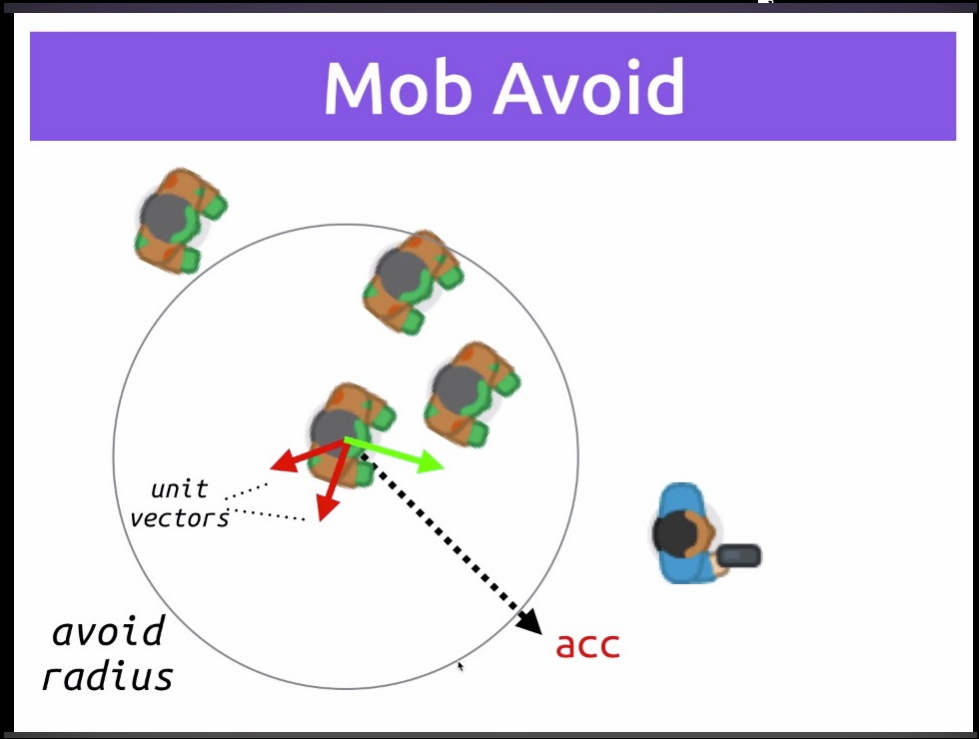
```python
def update(self):
    target_dist = self.target.pos - self.pos
    if target_dist.length_squared() < DETECT_RADIUS**2:
        if random() < 0.002:
            choice(self.game.zombie_moan_sounds).play()
        self.rot = target_dist.angle_to(vec(1, 0))
        self.image = pg.transform.rotate(self.game.pysnake_img, self.rot)
        self.rect.center = self.pos
        self.acc = vec(1, 0).rotate(-self.rot)
        self.avoid_mobs()
        self.acc.scale_to_length(self.speed)
        self.acc += self.vel * -1
        self.vel += self.acc * self.game.dt
        self.pos += self.vel * self.game.dt + 0.5 * self.acc * self.game.dt ** 2
        self.hit_rect.centerx = self.pos.x
        collide_with_walls(self, self.game.walls, 'x')
        self.hit_rect.centery = self.pos.y
        collide_with_walls(self, self.game.walls, 'y')
        self.rect.center = self.hit_rect.center
    if self.health <= 0:
        choice(self.game.zombie_hit_sounds).play()
        self.kill()
        self.game.map_img.blit(self.game.splat, self.pos - vec(32, 32))
        self.game.score += 1

def draw_health(self):
    if self.health > 60:
        col = GREEN
    elif self.health > 30:
        col = YELLOW
    else:
        col = RED
    width = int(self.rect.width * self.health / MOB_HEALTH)
    self.health_bar = pg.Rect(0, 0, width, 7)
    if self.health < MOB_HEALTH:
        pg.draw.rect(self.image, col, self.health_bar)
```

Mob Movement - Chasing Player



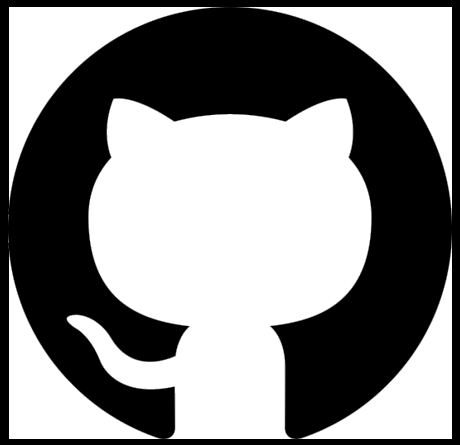
Speed and Velocity are stored as vectors. Each mob is given a DETECT_RADIUS
If player is within detection radius, we make the mob move to the player's direction, calculating their location by subtracting player.pos - mob.pos

Mob Movement - Improved



After a while mobs get mushed into one location because they all get the same velocity vector from the player. In order to solve this, we give each mob AVOID_RADIUS, and if there are other mobs in the radius, we calculate the unit vectors between target mob and invading mobs, and calculate direction by adding all three

Items - Github



- Randomly Chooses 0 ~ 2 Mobs and kills them
- Problem: you cannot apply random.choice to sprite groups
- Solution: When you create mobs, append them to a separate list, and apply random.choice to that list

```
class Sun(pg.sprite.Sprite):
    def __init__(self, game, x, y):
        self._layer = MOB_LAYER
        self.groups = game.all_sprites, game.mobs
        pg.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.image = game.sun_img.copy()
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.hit_rect = SUN_HIT_RECT.copy()
        self.hit_rect.center = self.rect.center
        self.pos = vec(x, y)
        self.vel = vec(0, 0)
        self.acc = vec(0, 0)
        self.rect.center = self.pos
        self.rot = 0
        self.health = 1500
        self.speed = SUN_SPEED
        self.target = game.player
        self.game.mobList.append(self)
```

```
    if hit.type == 'github':
        hit.kill()
        self.effects_sounds['github'].play()
        a1 = choice(self.mobList)
        a1.kill()
        self.mobList.remove(a1)
        a2 = choice(self.mobList)
        a2.kill()
        self.mobList.remove(a2)
        self.github += 1
        self.score += 2
if self.github >= 4:
    self.playing = False
```

```
self.score = 0
self.github = 0
self.start = True
self.prof = True
self.startTime = pg.time
self.nightDelay = 9000
self.FA = True
self.OOT = False
```

Items - Guns

```
BULLET_IMG = 'bullet.png'
WEAPONS = {}
WEAPONS['pistol'] = {'bullet_speed': 500,
                     'bullet_lifetime': 1000,
                     'rate': 150,
                     'kickback': 50,
                     'spread': 5,
                     'damage': 6,
                     'bullet_size': 'lg',
                     'bullet_count': 1}
WEAPONS['shotgun'] = {'bullet_speed': 400,
                      'bullet_lifetime': 500,
                      'rate': 900,
                      'kickback': 100,
                      'spread': 20,
                      'damage': 5,
                      'bullet_size': 'sm',
                      'bullet_count': 12}
WEAPONS['machine gun'] = {'bullet_speed': 600,
                      'bullet_lifetime': 800,
                      'rate': 50,
                      'kickback': 60,
                      'spread': 8,
                      'damage': 5.5,
                      'bullet_size': 'lg',
                      'bullet_count': 1}
```

- Info of guns stored in dict in settings.py

```
for mob in hits:
    # hit.health -= WEAPONS[self.player.weapon]['damage'] * len(hits[hit])
    for bullet in hits[mob]:
        mob.health -= bullet.damage
    mob.vel = mob.vel*0.85
```

```
for hit in hits:
    if hit.type == 'health' and self.player.health < PLAYER_HEALTH:
        hit.kill()
        self.effects_sounds['health_up'].play()
        self.player.add_health(HEALTH_PACK_AMOUNT)
    if hit.type == 'shotgun':
        hit.kill()
        self.effects_sounds['gun_pickup'].play()
        self.player.weapon = 'shotgun'
    if hit.type == 'machine gun':
        hit.kill()
        self.effects_sounds['metal_slug'].play()
        self.player.weapon = 'machine gun'
```

Items - Guns

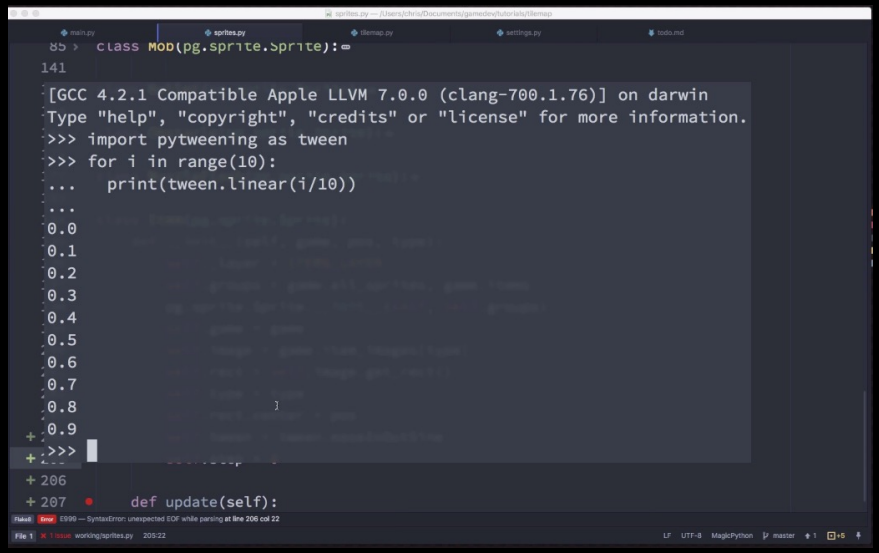
```
def shoot(self):
    now = pg.time.get_ticks()
    if now - self.last_shot > WEAPONS[self.weapon]['rate']:
        self.last_shot = now
        dir = vec(1, 0).rotate(-self.rot)
        pos = self.pos + BARREL_OFFSET.rotate(-self.rot)
        self.vel = vec(-WEAPONS[self.weapon]['kickback'], 0).rotate(-self.rot)
        for i in range(WEAPONS[self.weapon]['bullet_count']):
            spread = uniform(-WEAPONS[self.weapon]['spread'], WEAPONS[self.weapon]['spread'])
            Bullet(self.game, pos, dir.rotate(spread), WEAPONS[self.weapon]['damage'])
            snd = choice(self.game.weapon_sounds[self.weapon])
            if snd.get_num_channels() > 2:
                snd.stop()
            snd.play()
        MuzzleFlash(self.game, pos)
```

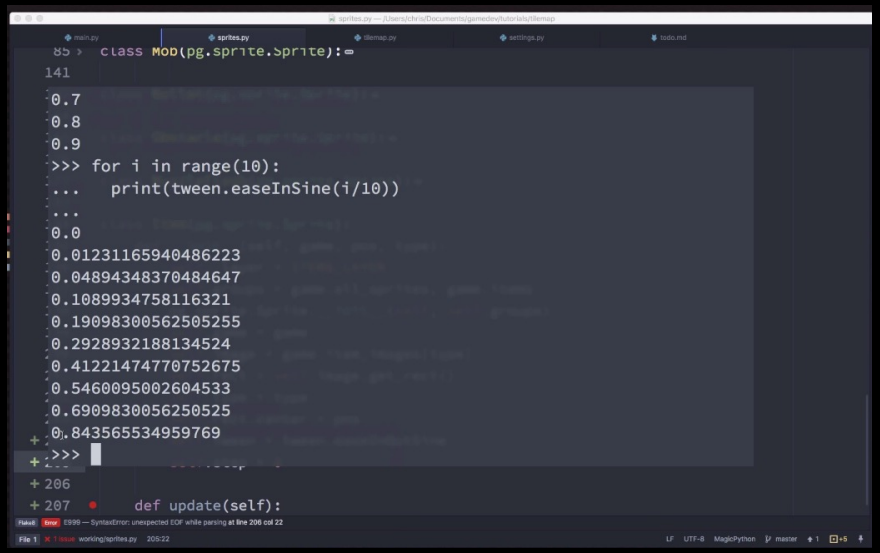
```
for mob in hits:
    # hit.health -= WEAPONS[self.player.weapon]['damage'] * len(hits[hit])
    for bullet in hits[mob]:
        mob.health -= bullet.damage
    mob.vel = mob.vel*0.85
```

- Rate
- Kickback
- Bullet_count
- Spread(Rotation for Inaccuracy)
- Range(Duration)
- Damage

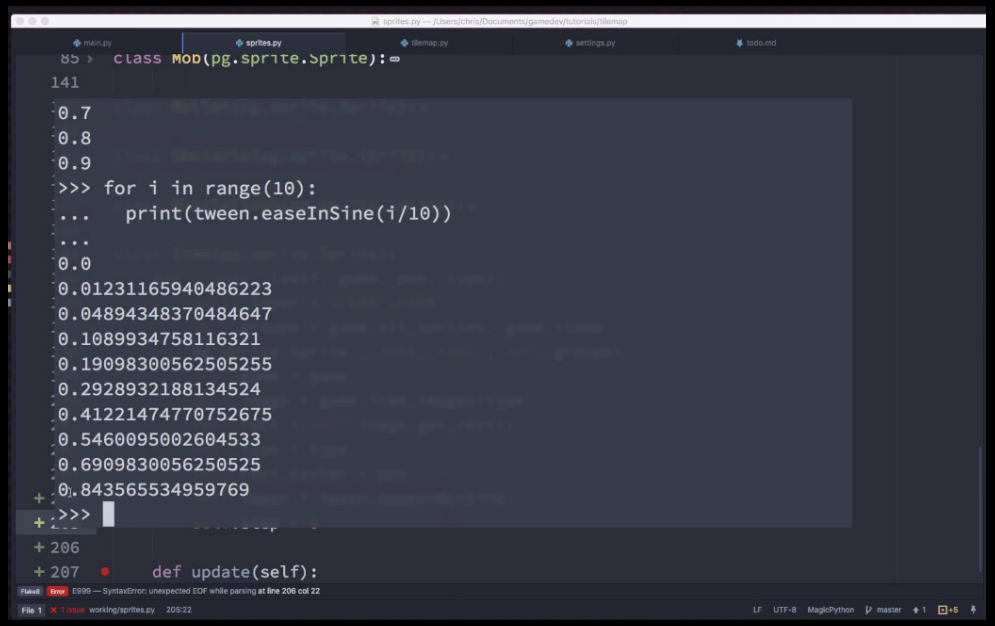
Items - Bouncing: Tweening/Easing

https://gamemechanicexplorer.com/





- Using the pytweening package
- Tweening allows us to move through a range in different rates




```python
class Item(pg.sprite.Sprite):
    def __init__(self, game, pos, type):
        self._layer = ITEMS_LAYER
        self.groups = game.all_sprites, game.items
        pg.sprite.Sprite.__init__(self, self.groups)
        self.game = game
        self.image = game.item_images[type]
        self.rect = self.image.get_rect()
        self.type = type
        self.pos = pos
        self.rect.center = pos
        self.tween = tween.easeInOutSine
        self.step = 0
        self.dir = 1

    def update(self):
        # bobbing motion
        offset = BOB_RANGE * (self.tween(self.step / BOB_RANGE) - 0.5)
        self.rect.centery = self.pos.y + offset * self.dir
        self.step += BOB_SPEED
        if self.step > BOB_RANGE:
            self.step = 0
            self.dir *= -1
```

Night Mode

```
def render_fog(self):
    # draw the light mask (gradient) onto fog image
    self.fog.fill(NIGHT_COLOR)
    self.light_rect.center = self.camera.apply(self.player).center
    self.fog.blit(self.light_mask, self.light_rect)
    self.screen.blit(self.fog, (0, 0), special_flags=pg.BLEND_MULT)
```

```
NIGHT_COLOR = (45, 45, 45)
LIGHT_RADIUS = (500, 500)
LIGHT_MASK = "light_350_soft.png"
```

Draw color over entire screen with special_flags
Set as BLEND_MULT

light_350_soft.png


```python
if (math.floor((pg.time.get_ticks() - self.startTime)/1000%50)) == 0:
    if pg.time.get_ticks() - self.nightDelay > 1000:
        self.night = not self.night
        self.nightDelay = pg.time.get_ticks()

if self.night:
    self.render_fog()
```

- Night mode is toggled every 50 seconds.
- Therefore 1 Day is 100 seconds

Ending Conditions

```
if len(self.mobs) == 1:
    self.playing = False
```

```
if self.prof == False:
    self.playing = False

if pg.time.get_ticks() - self.startTime >= 300000:
    self.FA = False

if pg.time.get_ticks() - self.startTime >= 1500000: #150000:
    self.OOT = True
    self.playing = False
```

- If all assignments are dead
- If Professor if defeated
- If 1500 seconds (15 Days) have passed
- If you don't make it through 300 seconds (3 Days), you get an F/A

Ending Conditions - End Screen

Different end screens depending on how the game ended

```
if self.github >=4 :
    self.screen.fill(BLACK)
    self.screen.blit(self.prof_img, (WIDTH/2 - 210, 50))
    self.draw_text("PLAGIARISM!", self.hud_font, 100, RED,
                   WIDTH / 2, HEIGHT / 2 + 160, align="center")
    self.draw_text("YOU GET AN INSTANT F!", self.hud_font, 75, WHITE,
                   WIDTH / 2, HEIGHT * 3.2 / 4, align="center")
    pg.display.flip()
    pg.time.delay(2000)
    self.wait_for_key()

if self.prof == False:
    self.screen.fill(BLACK)
    self.screen.blit(self.prof_img, (WIDTH/2 - 210, 50))
    self.draw_text("HOW DARE YOU", self.hud_font, 100, RED,
                   WIDTH / 2, HEIGHT / 2 + 160, align="center")
    # self.draw_text("YOU GET AN INSTANT F!", self.hud_font, 75, WHITE,
    #                WIDTH / 2, HEIGHT * 3.1 / 4, align="center")
    pg.display.flip()
    pg.time.delay(2000)
    self.wait_for_key()

if self.FA == True and len(self.mobs) != 1 and self.prof != False and self.github < 4: #and self.github <4
    self.screen.fill(BLACK)
    self.screen.blit(self.prof_img, (WIDTH/2 - 210, 50))
    self.draw_text("YOU DIDN'T LAST 3 DAYS", self.hud_font, 100, RED,
                   WIDTH / 2, HEIGHT / 2 + 160, align="center")
    self.draw_text("F/A!", self.hud_font, 75, WHITE,
                    WIDTH / 2, HEIGHT * 3.2 / 4, align="center")
    pg.display.flip()
    pg.time.delay(2000)
    self.wait_for_key()
```

Ending Conditions - Grades

```
if self.github >=4 or self.prof == False or (self.FA == True and len(self.mobs)!=1):
    self.score = 0
if self.score >= 0 and self.score<= 12:
    grade = 'F'
    self.emotional_damage.play()
elif self.score <=24:
    grade = 'D'
    self.failure.play()
elif self.score <=36:
    grade = 'C'
    self.failure.play()
elif self.score <=48:
    grade = 'B'
    self.weak_clapping.play()
elif self.score <=69:
    grade = 'A'
    self.yay.play()
else:
    grade = 'A+'
    self.tadaa.play()
```

Points per Mob

Pysnake: 1

Shmup: 1

Hangman: 1

Numpy: 2

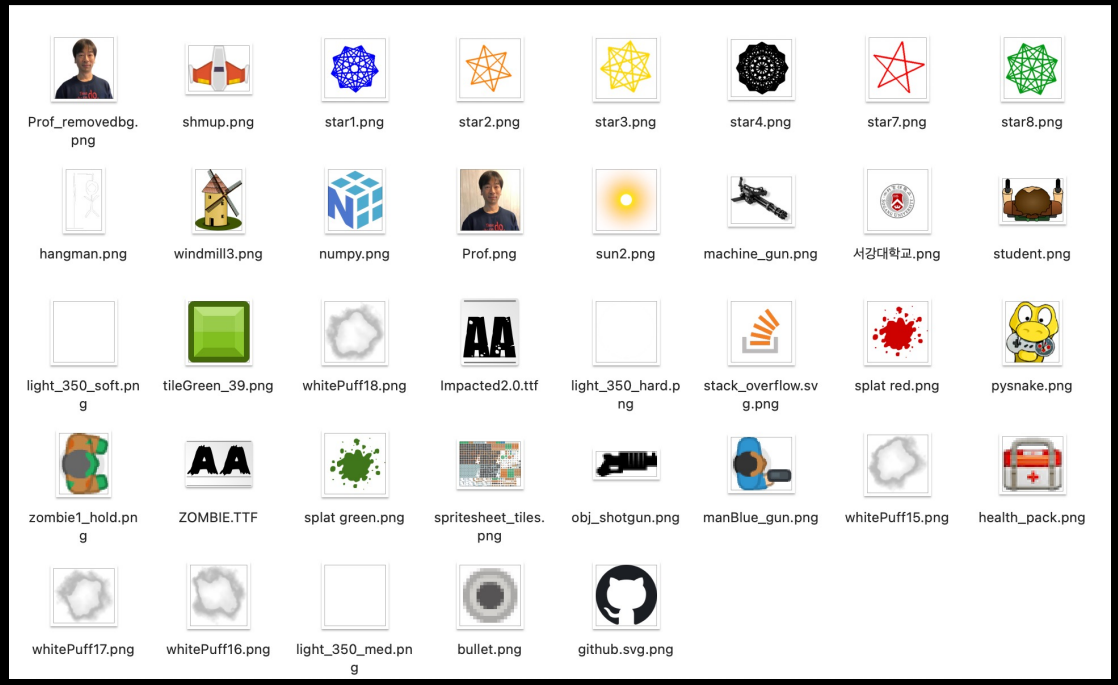
Windmill: 3

Stars1~7: 1,2,5

Sun: 10

You must kill all assignments WITHOUT using any Githubs to get A+

Assets




```python
def load_data(self):
    game_folder = path.dirname(__file__)
    img_folder = path.join(game_folder, 'img')
    snd_folder = path.join(game_folder, 'snd')
    music_folder = path.join(game_folder, 'music')
    self.map_folder = path.join(game_folder, 'maps')
    self.title_font = path.join(img_folder, 'ZOMBIE.TTF')
    self.hud_font = path.join(img_folder, 'Impacted2.0.ttf')
    self.dim_screen = pg.Surface(self.screen.get_size()).convert_alpha()
    self.dim_screen.fill((0, 0, 0, 180))
    self.sogang_img = pg.image.load(path.join(img_folder, '서강대학교.png')).convert_alpha()
    self.sogang_img = pg.transform.scale(self.sogang_img, (420, 420))
    self.player_img = pg.image.load(path.join(img_folder, PLAYER_IMG)).convert_alpha()
    self.prof_img = pg.image.load(path.join(img_folder, 'Prof.png')).convert_alpha()
    self.sun_img = pg.image.load(path.join(img_folder, 'sun2.png')).convert_alpha()
    self.sun_img = pg.transform.scale(self.sun_img, (250, 250))
    self.player_img = pg.transform.scale(self.player_img, (65, 60))
    self.player_img = pg.transform.rotate(self.player_img, -90)
    self.bullet_images = {}
    self.bullet_images['lg'] = pg.image.load(path.join(img_folder, BULLET_IMG)).convert_alpha()
    self.bullet_images['sm'] = pg.transform.scale(self.bullet_images['lg'], (10, 10))
    self.pysnake_img = pg.image.load(path.join(img_folder, PYSNAKE_IMG)).convert_alpha()
    self.pysnake_img = pg.transform.scale(self.pysnake_img, (64, 64))
    self.pysnake_img = pg.transform.rotate(self.pysnake_img, -90)
    self.shmup_img = pg.image.load(path.join(img_folder, 'shmup.png')).convert_alpha()
    self.shmup_img = pg.transform.scale(self.shmup_img, (72, 72))
    self.shmup_img = pg.transform.rotate(self.shmup_img, -90)
    self.numpy_img = pg.image.load(path.join(img_folder, 'numpy.png')).convert_alpha()
    self.numpy_img = pg.transform.scale(self.numpy_img, (64, 64))
    self.numpy_img = pg.transform.rotate(self.numpy_img, -90)
    self.profPNG_img = pg.image.load(path.join(img_folder, 'Prof_removedbg.png')).convert_alpha()
    self.profPNG_img = pg.transform.scale(self.profPNG_img, (150, 150))
    self.profPNG_img = pg.transform.rotate(self.profPNG_img, -90)
    self.windmill_img = pg.image.load(path.join(img_folder, 'windmill3.png')).convert_alpha()
```

Basis & References

https://kidscancode.org/lessons/
https://github.com/kidscancode/pygame_tutorials
https://www.youtube.com/@Kidscancode
https://www.youtube.com/@zacharystolter5502

Improvements

Several waves instead of 1 big game?

Power ups? (PngEgg -> Increase speed, Invinsible etc)

Some Animation?

Different Weapons?

Change Pointer Image?

A mob that shoots?

Typing minigame?