<div style="border:1px solid black; display:inline-block; padding:4px 12px;">**Documentation**</div>

Table of Contents

/home/runner/work/Tanksta/Tanksta//Project/bullet.py

```python
def __init__(self, coordonates, angle, puissance):

    puissance = puissance * 19

    self.image = pygame.image.load(

        directory + 'assets/objects/tank_bullet5.png')

    self.origin_image = self.image

    self.rect = coordonates

    self.image = pygame.transform.rotate(self.origin_image, angle)

    self.angle = angle

    self.damage = 20

    self.puissance = puissance

    self.vx = puissance * math.cos(math.radians(angle))

    self.vy = puissance * math.sin(math.radians(angle))

    self.ax = 0  # puissance # vent

    self.ay = -9.8

    self.x = self.rect[0]

    self.y = self.rect[1]

    self.time = 0

  # This is my commentary
```

-> Convert the image from Puisy .

Possible memory error in this function

```python
def updateVx(self, dt):

    self.vx = self.vx + self.ax * dt
```

```
        return self.vx
```

-> Update the x and y

```
    def updateVy(self, dt):

        self.vy = self.vy + self.ay * dt

        return self.vy
```

-> Updates the Vy of the current dt

```
    def updateX(self, dt):

        self.x = self.x + 0.5 * (self.vx + self.updateVx(dt)) * dt

        return self.x
```

-> Updates the value in the data field with the given dt .

```
    def updateY(self, dt):

        self.y = self.y - 0.5 * (self.vy + self.updateVy(dt)) * dt

        return self.y
```

-> Updates the given dt in the multiple - precisions .

```
    def updateBulletPosition(self, dt):

        self.rect[0] = self.updateX(dt)

        self.rect[1] = self.updateY(dt)

        self.time = self.time + dt
```

-> Updates this gate s duration to the specified dt .

```python
def display(self, screen, delta):

    screen.blit(self.image, self.rect)

    self.updateBulletPosition(delta)
```

-> Send a rain REs .

/home/runner/work/Tanksta/Tanksta//Project/request.py

```python
def __init__(self):

    self.player = None

    self.game = None

    self.players = []

    self.current_player = None
```

-> ! \ ~english Initialize the player and store it .

```python
def playOnline(self):

    payload = {'pseudo': 'losabit', 'want_to_play': False}

    headers = {'Content-type': 'application/json'}

    r = requests.post(url + '/players/?format=json', data=json.dumps(payload), headers=headers)

    self.player = json.loads(r.text)
```

-> Write the json .

```python
def wantToPlay(self):

    if self.player is None:

        return

    self.player['want_to_play'] = True

    headers = {'Content-type': 'application/json'}

    r = requests.put(url + '/players/' + str(self.player["id"]) + "?format=json", data=json.dumps(self.player),

            headers=headers)

    self.player = json.loads(r.text)
```

-> Changes the player in the current game .

Documentation

```
def checkGameIsFind(self):

    if self.player is None:

        return

    r = requests.get(url + "/players/" + str(self.player["id"]) + "?format=json")

    result = json.loads(r.text)

    if result["play_on"] is not None:

        self.player = result

        r = requests.get(url + "/games/" + str(result["play_on"]) + "?format=json")

        self.game = json.loads(r.text)

        if self.game["turn_id"] is not None:

            self.getInfo()

            return True

        else:

            return False

    else:

        return False
```

-> Returns True if the game was the first one

Possible memory error in this function

```
def loadGame(self):

    r = requests.get(url + '/players/?format=json')

    result = json.loads(r.text)

    for player in result:

        if player['play_on'] == self.game['id'] and player['id'] != self.player['id']:
```

```python
        self.players.append(player)

    r = requests.get(url + "/players/" + str(self.player["id"]) + "?format=json")

    self.player = json.loads(r.text)
```

-> save the PlayerRecord settings from saved disk file

```python
def endTurn(self):

    if self.player is None:

        return

    self.player['end_of_turn'] = True

    headers = {'Content-type': 'application/json'}

    r = requests.put(url + '/players/' + str(self.player["id"]) + "?format=json", data=json.dumps(self.player), headers=headers)

    self.player = json.loads(r.text)

    self.getInfo()
```

-> Returns the Turn object .

```python
def getInfo(self):

    r = requests.get(url + "/games/" + str(self.game["id"]) + "?format=json")

    self.game = json.loads(r.text)

    r = requests.get(url + "/players/" + str(self.game["turn_id"]) + "?format=json")

    self.current_player = json.loads(r.text)
```

-> Cache the info from the given game and store it in self . _activity

```python
def sendInfo(self, tank):

    headers = {'Content-type': 'application/json'}
```

Documentation

```
        self.player['health'] = tank.current_health

        self.player['pos_x'] = tank.body_rect.x

        self.player['pos_y'] = tank.body_rect.y

        self.player['shoot'] = len(tank.bullets) >= 1

        if self.player['shoot']:

            self.player['puissance'] = tank.bullets[0].puissance

        self.player['canon_orientation'] = tank.canon_angle

        r = requests.put(url + '/players/' + str(self.player["id"]) + "?format=json", data=json.dumps(self.player), headers=headers)

        self.player = json.loads(r.text)
```

-> Checks the player for security .

```
    def getNbValidate(self, online_manager):

        initt = 5

        if self.game["players"] >= self.game["players_want_play"]:

            font = pygame.font.SysFont("comicsansms", 30)

            text = font.render("{1} on {0} as validate".format(self.game["players"], self.game["players_want_play"]),

                        True, (120, 0, 0))

            label = pygame_gui.elements.UILabel(relative_rect=pygame.Rect((900, 400), (200, 200)),text="{1} on {0} as "

                                                "validate".format(self.game["players"], self.game["players_want_

            #print("{1} on {0} as validate".format(self.game["players"], self.game["players_want_play"]))

            t = TimeCapsule(initt)
```

-> Runs the flags through the given editor .

```
    def delete(self):
```

```
if self.player is None:

    return

headers = {'Content-type': 'application/json'}

r = requests.delete(url + '/players/' + str(self.player["id"]) + "?format=json", data=json.dumps(self.player),

        headers=headers)
```

-> Delete the given game from the Google sheets API .

/home/runner/work/Tanksta/Tanksta//Project/gui.py

```python
def __init__(self,Tanks,manager):

    self.tanks =  Tanks

    self.manager = manager

    self.initHealthBar()
```

-> Initialize the simulation s DependentCollection .

```python
def initHealthBar(self):

    inc = 0

    for tank in self.tanks:

        pygame_gui.elements.UIScreenSpaceHealthBar(relative_rect=pygame.Rect(((1600 / len(self.tanks))*inc, 100), (1600 / le

        inc += 1
```

-> Generate a new view of the given geometry .

```python
def drawCurrentTurn(self,turn,screen):
```

-> Turns on the given screen .

```python
font = pygame.font.Font(pygame.font.get_default_font(), 36)

# now print the text

text_surface = font.render(f'Playing: Tank {str(turn+1)}' , True, pygame.Color('black'))

text_rect = text_surface.get_rect(center=(1600/2, 50))

screen.blit(text_surface, text_rect)
```

-> Returns a unicode string with the given text_width .

```
def draw(self,turn,screen):

    self.drawCurrentTurn(turn,screen)
```

-> If we are running as a screen it should kick off the stack . If it is negative it

/home/runner/work/Tanksta/Tanksta//Project/offline.py

```python
def __init__(self, numberOfPlayers, y, screen):

    self.manager = pygame_gui.UIManager((1600, 900))

    self.tanks = self.initTankPositions(numberOfPlayers, y)

    self.player = Player(self.tanks[0])

    self.ai = [AI(self.tanks[i]) for i in range(1, len(self.tanks))]

    self.turn = -1

    self.origin_tank_position = None

    self.nextTurn()

    self.gui = GUI(self.tanks, manager=self.manager)
```

-> Generate a new view of the current position .

```python
def initTankPositions(self, numberOfTank, y):

    tanks = []

    for i in range(numberOfTank):

        tanks.append(Tank(tuple([350 + i * 400, y])))

    return tanks
```

Fonction to update tank position

-> Generate an HttpResponse of counts form a certain contribution to an existing Crane .

Possible memory error in this function

```python
def update(self, screen):

    # Draw info GUI

    self.gui.draw(self.turn, screen)
```

```python
time_delta = clock.tick(60)/1000.0

if self.turn == 0:

    self.player.update()

    if self.difference_position(self.origin_tank_position, self.player.tank.body_rect) > MOVEMENT_LIMIT or len(self.player.ta

        self.nextTurn()

        self.player.stop()

else:

    self.ai[self.turn - 1].random_controller(self.player.tank)

    if self.difference_position(self.origin_tank_position, self.ai[self.turn - 1].tank.body_rect) > MOVEMENT_LIMIT or len(self.a

        self.nextTurn()

for tank in self.tanks:

    tank.display(screen, self.tanks)

for i in range(len(self.tanks)):

    if i >= len(self.tanks):

        i = len(self.tanks) - 1

    if self.tanks[i].current_health <= 0:

        del self.tanks[i]

if len(self.tanks) == 1:

    return False

for event in pygame.event.get():

    if self.turn == 0:

        self.player.controller(event)

    if event.type == pygame.QUIT:

        pygame.quit()
```

```
        print("Game Closed")

        running = False

    self.manager.update(time_delta)

    self.manager.draw_ui(screen)

    return True
```

-> Generate the str for insertion into a stream .

Possible memory error in this function

```
def difference_position(self, pos1, pos2):

    return abs(pos1[0] - pos2.x) + abs(pos1[1] - pos2.y)
```

-> Return the difference between two given sets of parameters .

```
def nextTurn(self):

    self.turn += 1

    if self.turn == len(self.tanks):

        self.turn = 0

    if self.turn == 0:

        self.origin_tank_position = [self.player.tank.body_rect.x, self.player.tank.body_rect.y]

    else:

        self.origin_tank_position = self.ai[self.turn - 1].tank.body_rect
```

-> Converts all the rolls to an ElasticSearch .

/home/runner/work/Tanksta/Tanksta//Project/online.py

```python
def __init__(self, server):

    if len(server.players) == 0:

        print("need more than 1 player")

        pygame.quit()

    print(server.players)

    self.server = server

    self.ids = self.initIds(server.players)

    self.tanks = self.initTankPositions(server.players)

    self.player_tank = Tank(tuple([server.player['pos_x'], server.player['pos_y']]))

    self.player_tank.moveCanon(server.player['canon_orientation'] - self.player_tank.canon_angle)

    self.tanks.append(self.player_tank)

    self.player = Player(self.player_tank)

    self.origin_position = None

    self.nextTurn = False

    self.last_indice = -1

    self.last_player = None

    self.manager = pygame_gui.UIManager((1600, 900))

    self.gui = GUI(self.tanks, manager=self.manager)
```

-> Generate the References for the current player .

Possible memory error in this function

```python
def initTankPositions(self, players):

    tanks = []
```

```
    for player in players:

        tanks.append(Tank(tuple([player['pos_x'], player['pos_y']])))

        tanks[len(tanks) - 1].moveCanon(player["canon_orientation"] - tanks[len(tanks) - 1].canon_angle)

    return tanks
```

-> Generate a list of residues given a list of numbers

Possible memory error in this function

```
    def initIds(self, players):

        ids = {}

        count = 0

        for player in players:

            ids[player['id']] = count

            count += 1

        return ids
```

-> Convert the number of journals to journal ints .

Possible memory error in this function

```
    def update(self, screen):

        self.gui.draw(0, screen)

        if self.server.current_player == None:

            print("error")

        if self.server.current_player['id'] == self.server.player['id']:

            if self.player.tank.current_health == 0 or self.nextTurn:

                self.origin_position = None
```

```
                self.server.endTurn()

                self.nextTurn = False

            else:

                if self.origin_position == None:

                    self.origin_position = [self.player.tank.body_rect.x, self.player.tank.body_rect.y]

                self.server.sendInfo(self.player.tank)

                for event in pygame.event.get():

                    self.player.controller(event)

                    self.player.update()

                    if event.type == pygame.QUIT:

                        self.server.delete()

                        pygame.quit()

                        print("Game Closed")

                if self.difference_position(self.origin_position, self.player.tank.body_rect) > MOVEMENT_LIMIT or len(self.player.tank.

                    self.nextTurn = True

        else:

            self.server.getInfo()

            current_player = self.server.current_player

            if current_player["id"] in self.ids:

                indice = self.ids[current_player["id"]]

                self.tanks[indice].move(current_player["pos_x"] - self.tanks[indice].body_rect.x)

                self.tanks[indice].current_health = current_player["health"]

                self.tanks[indice].moveCanon(current_player["canon_orientation"] - self.tanks[indice].canon_angle)

                if self.last_indice == -1:
```

```
            self.last_indice = indice

            self.last_player = current_player

        if self.last_indice != indice:

            self.tanks[indice].shoot(self.last_player["puissance"])

            self.last_indice = indice

            self.last_player = current_player

    for tank in self.tanks:

        if tank.current_health != 0:

            tank.display(screen, self.tanks)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            self.server.delete()

            pygame.quit()

            print("Game Closed")

    return True
```

-> Returns an event or updates the stack with the given id .

```
    def difference_position(self, pos1, pos2):

        return abs(pos1[0] - pos2.x) + abs(pos1[1] - pos2.y)
```

-> Return the difference between two given sets of parameters .

Documentation

/home/runner/work/Tanksta/Tanksta//Project/tank.py

Possible memory error in this function

```python
def __init__(self, position):

    super().__init__

    self.health_capacity = 100

    self.current_health = 100

    self.body_image = pygame.image.load(directory + 'assets/objects/tanks_tankGreen_body3.png')

    self.body_rect = self.body_image.get_rect()

    self.body_rect.x = position[0]

    self.body_rect.y = position[1]

    self.wheel_image = pygame.image.load(directory + 'assets/objects/tanks_tankTracks3.png')

    self.wheel_rect = self.wheel_image.get_rect()

    self.wheel_rect.x = position[0] + 5

    self.wheel_rect.y = position[1] + 40

    self.canon_image = pygame.image.load(directory + 'assets/objects/tanks_turret4.png')

    self.canon_rect = self.canon_image.get_rect()

    self.canon_image = pygame.transform.scale(self.canon_image, tuple([int(self.canon_rect.w * 1.5), self.canon_rect.h * 2]))

    self.canon_rect = self.canon_image.get_rect()

    self.canon_rect.x = position[0] + 15

    self.canon_rect.y = position[1] + 5

    self.canon_originImage = self.canon_image

    self.canon_angle = 0

    self.bullets = []
```

-> Checks the image for the unknown type .

```python
def display(self, screen, tanks):

    time_delta = clock.tick(60)/100.0

    screen.blit(self.canon_image, self.canon_rect)

    screen.blit(self.wheel_image, self.wheel_rect)

    screen.blit(self.body_image, self.body_rect)

    for i in range(len(self.bullets)):

        if i == len(self.bullets):

            i -= 1

        bullet = self.bullets[i]

        bullet.display(screen, time_delta)

        if bullet.rect[1] > 710:

            del self.bullets[i]

        elif bullet.rect[0] > 1500 or bullet.rect[0] < -500:

            del self.bullets[i]

        else:

            for tank_ennemi in tanks:

                if abs(bullet.rect[0] - tank_ennemi.body_rect.x) < 30 and abs(bullet.rect[1] - tank_ennemi.body_rect.y) < 30:

                    tank_ennemi.touched(bullet.damage)

                    del self.bullets[i]
```

-> Generate the badge for the given image .

Possible memory error in this function

```python
def moveCanon(self, value):

    self.canon_angle += value
```

```
        self.canon_image = pygame.transform.rotate(self.canon_originImage, self.canon_angle)

        self.canon_rect = self.canon_image.get_rect(center=self.canon_rect.center)
```

-> Move the hand .

```
    def shoot(self, puissance):

        coordonates = list(self.canon_rect.center)

        coordonates[0] += math.cos(math.radians(self.canon_angle)) * 30

        coordonates[1] -= math.sin(math.radians(self.canon_angle)) * 30 + 5

        self.bullets.append(Bullet(coordonates, self.canon_angle, puissance))
```

-> Appends the coordintance to the polygons .

Possible memory error in this function

```
    def move(self, value):

        self.wheel_rect.x += value

        self.body_rect.x += value

        self.canon_rect.x += value
```

-> Move the plot by value .

Possible memory error in this function

```
    def touched(self, value):

        self.current_health -= value
```

-> Set the state of the pin to a value .

Documentation

/home/runner/work/Tanksta/Tanksta//Project/utils/timeCapsule.py

Possible memory error in this function

```python
def __init__(self, time):

    self.time_to_end = time

    self.time_spend = time

    self.getTicksLastFrame = 0

    self.start_ticks = pygame.time.get_ticks()
```

-> Call this from the pygame . time_to_end method


```python
def can_execute(self):

    t = pygame.time.get_ticks()

    self.time_spend += (t - self.getTicksLastFrame) / 1000.0

    self.getTicksLastFrame = t

    if self.time_spend >= self.time_to_end:

        self.time_spend = 0

        return True

    return False
```

-> Check if the time falls before the next frame .

Possible memory error in this function

```python
def modify_time(self, time):

    self.time_to_end = time

    self.time_spend = 0
```

-> Set the time_to_end and end of the file .

/home/runner/work/Tanksta/Tanksta//Project/controllers/ai.py

```python
def __init__(self, Tank):

    Controller.__init__(self, Tank)
```

-> Sets the current configuration to the database .

```python
def random_controller(self, tank):

    self.increase_puissance = True

    if random.randint(1, 1000) < 30:

        self.tank.move(-random.randint(1, 60))

    if random.randint(1, 1000) < 60:

        self.tank.moveCanon(self.predictAngle(self.tank.canon_angle,tank))

        self.tank.shoot(self.puissance if self.puissance < max_puissance else max_puissance)

        self.puissance = random.randint(1, 20)

        self.increase_puissance = False
```

-> wuis the thermostat is like a fake executable that uses puisis

```python
def predictAngle(self,canon_angle, tank):

    if self.tank.body_rect[0] > tank.body_rect[0]:

        if canon_angle == -200:

            return 0

        else:

            deg = -200

    else:

        if canon_angle == 200:
```

```
        return 0

    else:

        deg = 200

    return deg
```

-> The result of this method in the order that it is conform to the given operations .

```
def basic_controller(self, tank, tanks_ennemies):

    self.increase_puissance = True

    if random.randint(1, 1000) < 60:

        self.tank.shoot(self.puissance if self.puissance < max_puissance else max_puissance)

        self.puissance = min_puissance

        self.increase_puissance = False
```

-> Update the puisis with a random selection .

Documentation

/home/runner/work/Tanksta/Tanksta//Project/controllers/player.py

```python
def __init__(self, Tank):

    Controller.__init__(self, Tank)
```

-> Sets the current configuration to the database .

```python
def controller(self, event):

    if event.type == pygame.KEYDOWN:

        if event.key == pygame.K_SPACE:

            self.increase_puissance = True

        if event.key == pygame.K_q:

            self.can_move_canon = True

            self.canon_direction = 1

        elif event.key == pygame.K_d:

            self.can_move_canon = True

            self.canon_direction = -1

        if event.key == pygame.K_RIGHT and self.move == False:

            self.move = True

            self.direction = 1

        elif event.key == pygame.K_LEFT and self.move == False:

            self.move = True

            self.direction = -1

    if event.type == pygame.KEYUP:

        if event.key == pygame.K_SPACE:

            self.tank.shoot(self.puissance if self.puissance < max_puissance else max_puissance)
```

```
        self.puissance = min_puissance

        self.increase_puissance = False

    if event.key == pygame.K_q and self.tank.canon_angle < 180 + ecart_angle:

        self.can_move_canon = False

    elif event.key == pygame.K_d and self.tank.canon_angle > -ecart_angle:

        self.can_move_canon = False

    if event.key == pygame.K_RIGHT and self.direction > 0:

        self.move = False

    elif event.key == pygame.K_LEFT and self.direction < 0:

        self.move = False
```

-> Move the sensor to a specific key .

Documentation

/home/runner/work/Tanksta/Tanksta//Project/controllers/controller.py

```python
def __init__(self, Tank):

    self.tank = Tank

    self.puissance = min_puissance

    self.increase_puissance = False

    self.move = False

    self.direction = 0

    self.can_move_canon = False

    self.canon_direction = 0

  # Update the tank's position and angle of canon
```

-> Updates the puisiss dictionary with the given Puisissysance

Possible memory error in this function

```python
def update(self):

    if self.increase_puissance:

        print("Increasing power", self.puissance)

        self.puissance += value_puissance

    if self.move and self.direction != 0:

        self.tank.move(move_value if self.direction > 0 else -move_value)

    if self.can_move_canon and self.canon_direction != 0:

        if self.canon_direction > 0 and self.tank.canon_angle > 180 + ecart_angle:

            return

        if self.canon_direction < 0 and self.tank.canon_angle < -ecart_angle:

            return
```

```
self.tank.moveCanon(canon_angle_value if self.canon_direction > 0 else -canon_angle_value)
```

-> The puisission is an integer array of bytes that contain orifices .

```
def stop(self):

    self.puissance = min_puissance

    self.move = False

    self.can_move_canon = False

    self.increase_puissance = False
```

-> stop the HA cycle