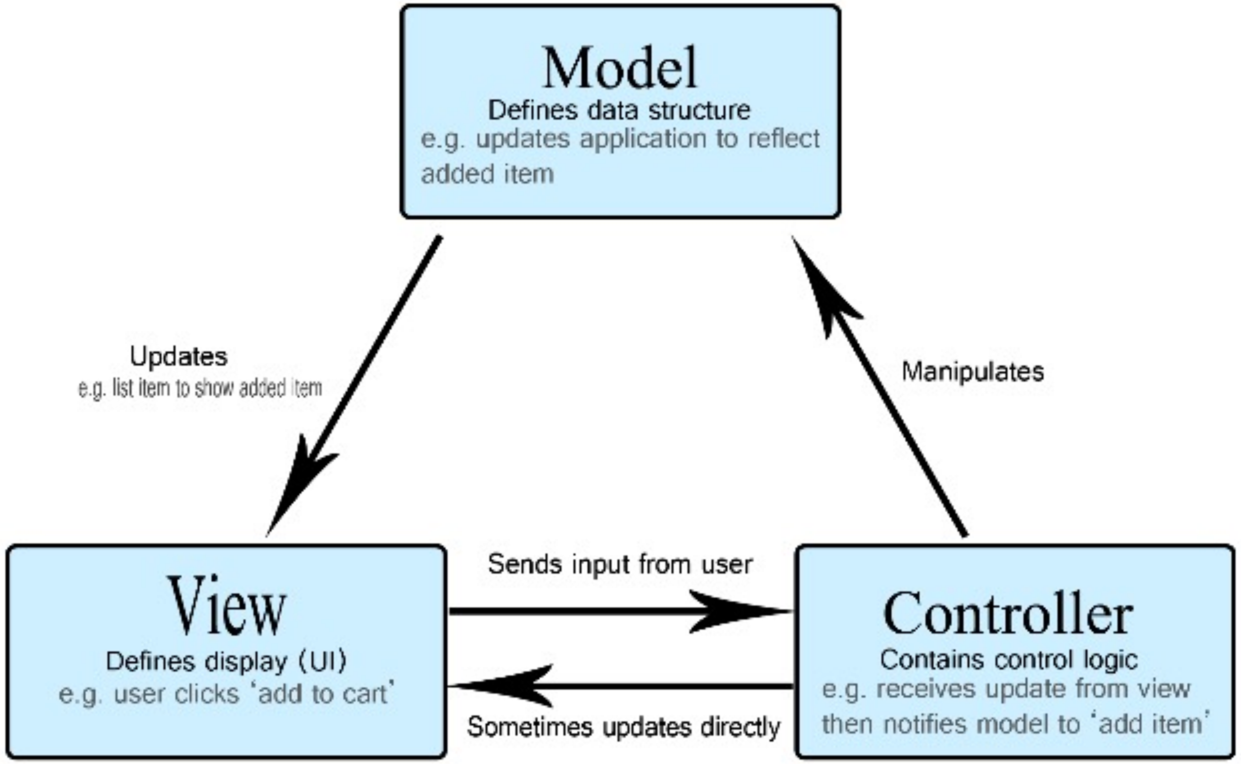
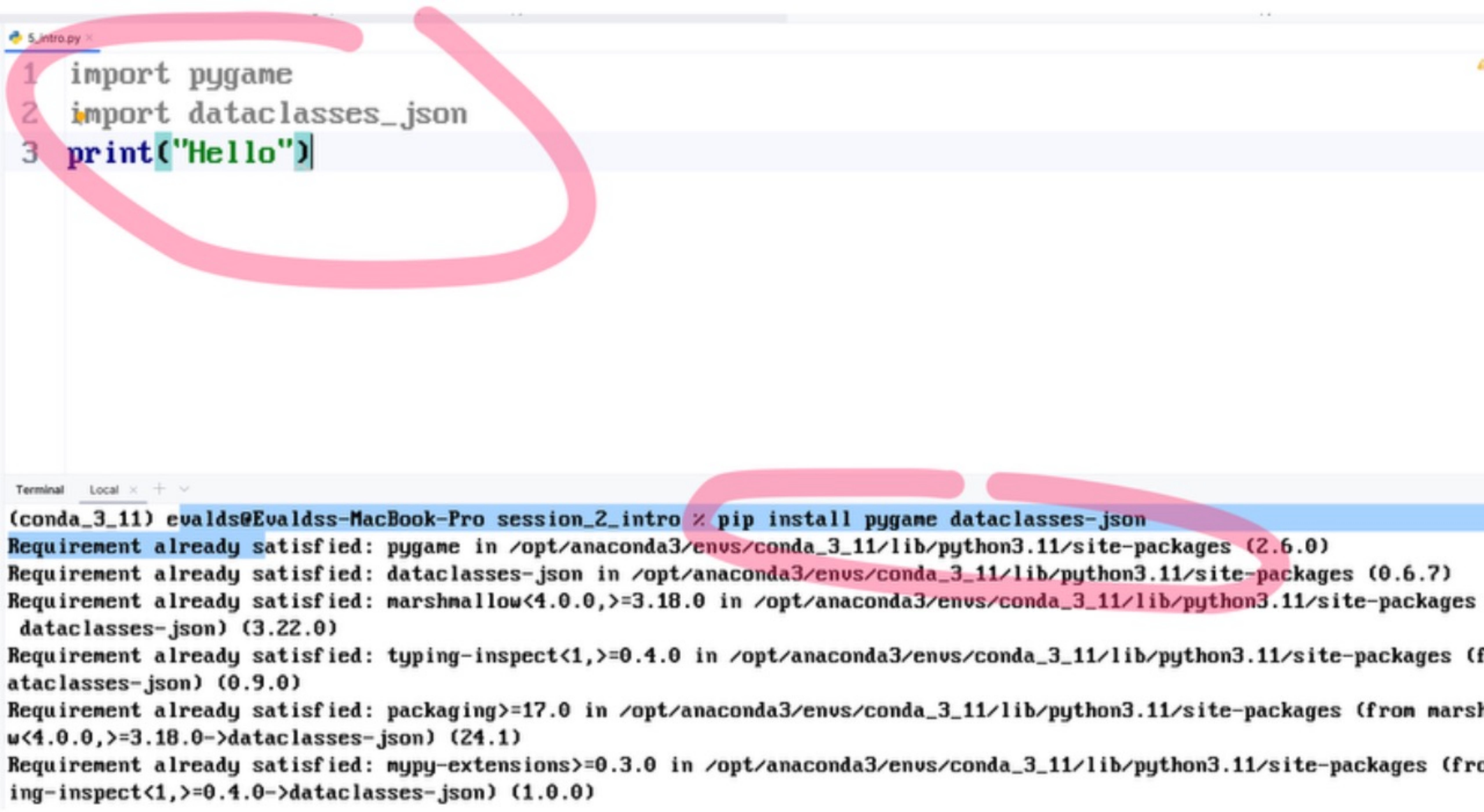
2024-Q4-DesignPatterns 2. Model View Controller (MVC), PyGame


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'



```python
import pygame
import dataclasses_json
print("Hello")
```

```
(conda_3_11) evalds@Evaldss-MacBook-Pro session_2_intro % pip install pygame dataclasses-json
Requirement already satisfied: pygame in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages (2.6.0)
Requirement already satisfied: dataclasses-json in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages (0.6.7)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages
 dataclasses-json) (3.22.0)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages (f
ataclasses-json) (0.9.0)
Requirement already satisfied: packaging>=17.0 in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages (from marsh
w<4.0.0,>=3.18.0->dataclasses-json) (24.1)
Requirement already satisfied: mypy-extensions>=0.3.0 in /opt/anaconda3/envs/conda_3_11/lib/python3.11/site-packages (fro
ing-inspect<1,>=0.4.0->dataclasses-json) (1.0.0)
```

1.3. Implementēt Python doto UML klašu diagrammu

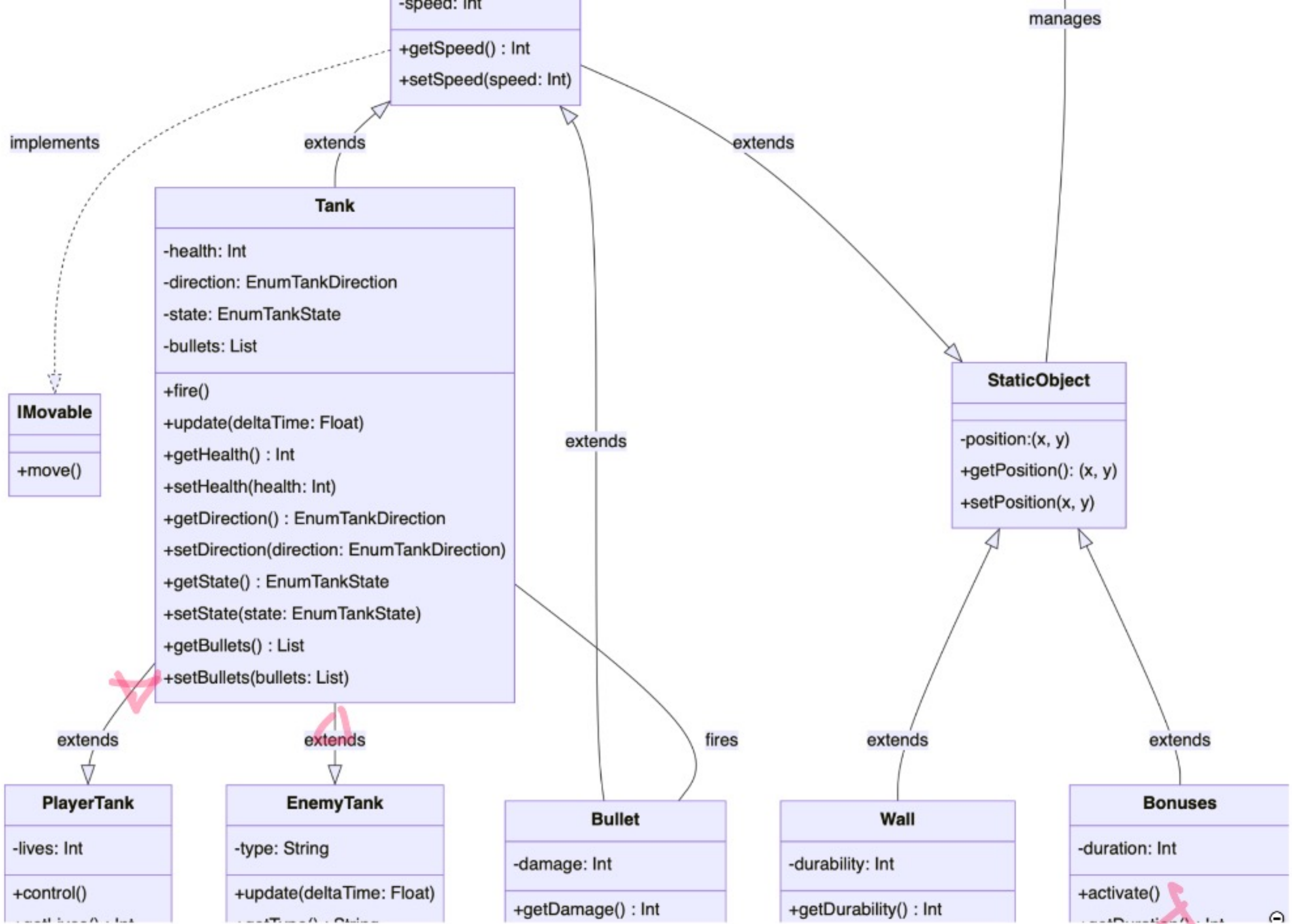
Implementēt UML diagramā doto shēmu Python valodā un vienā failā iesniegt līdz ar uzdevumu. Papildus punkti, implementējot.

Shēma: http://share.yellowrobot.xyz/upic/46175827e374b23b19f988e61f2b1ba2_1694030930.png

Interface Python valodā implementē ar ABCMeta un @abstractmethod.
Implementēt tikai getters un setters, pārējās funkcijas atstāt tukšas ar pass,

piemēram.

```
class Game:
   def new_game(self):
     pass
```



-speed: Int

+getSpeed() : Int

+setSpeed(speed: Int)

Tank

-health: Int

-direction: EnumTankDirection

-state: EnumTankState

-bullets: List

+fire()

+update(deltaTime: Float)

+getHealth() : Int

+setHealth(health: Int)

+getDirection() : EnumTankDirection

+setDirection(direction: EnumTankDirection)

+getState() : EnumTankState

+setState(state: EnumTankState)

+getBullets() : List

+setBullets(bullets: List)

IMovable

+move()

StaticObject

-position:(x, y)

+getPosition(): (x, y)

+setPosition(x, y)

PlayerTank

-lives: Int

+control()

+getLives() : Int

EnemyTank

-type: String

+update(deltaTime: Float)

+getType() : String

Bullet

-damage: Int

+getDamage() : Int

Wall

-durability: Int

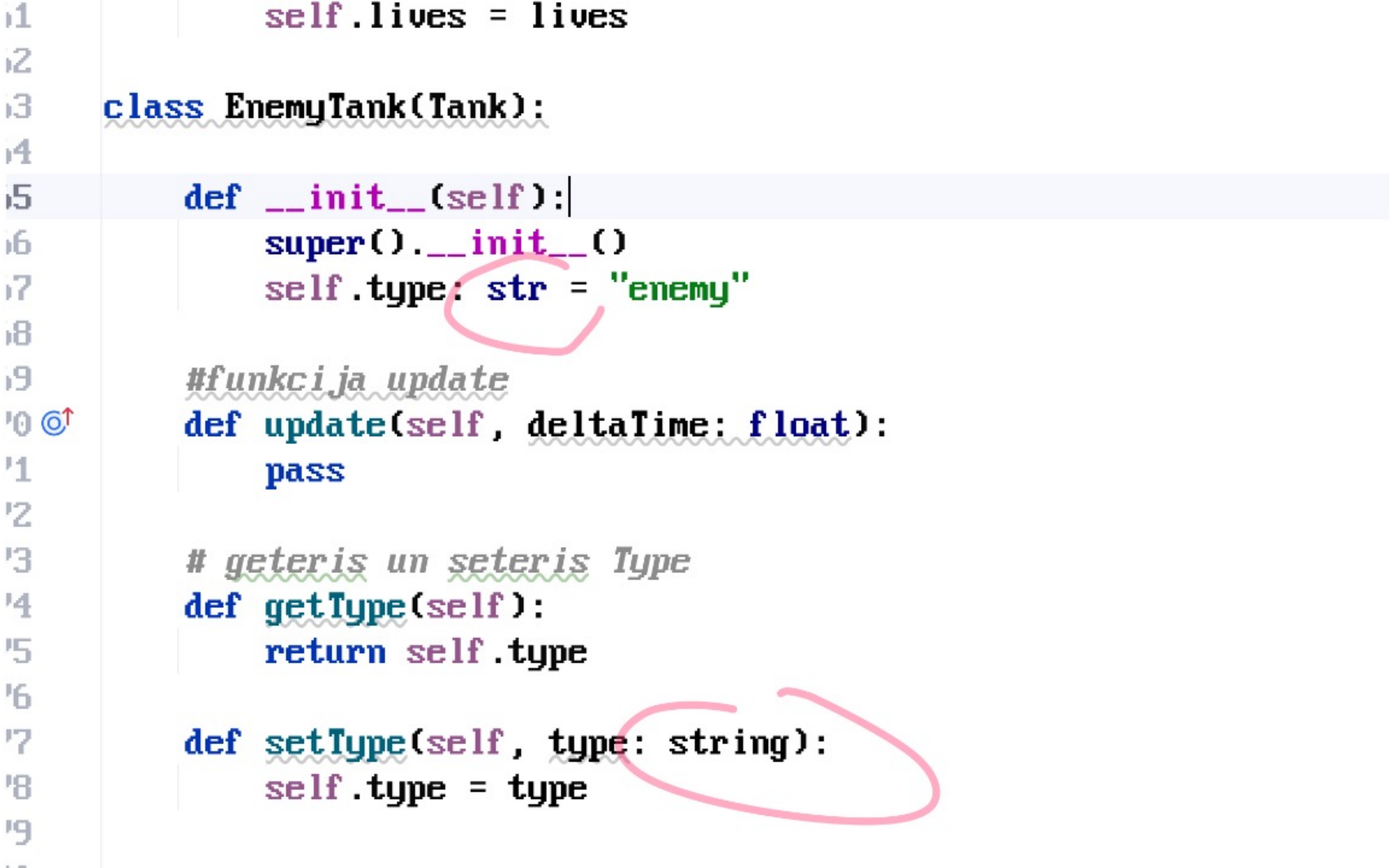
+getDurability() : Int

Bonuses

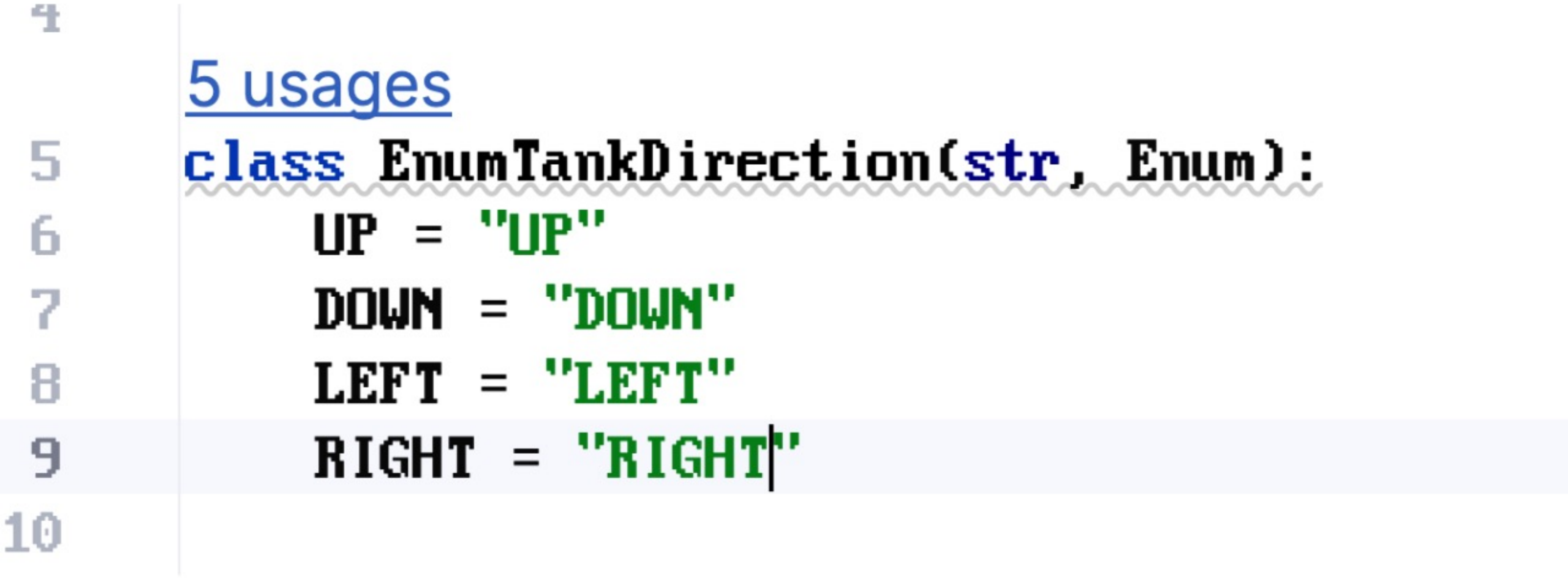
-duration: Int

+activate()

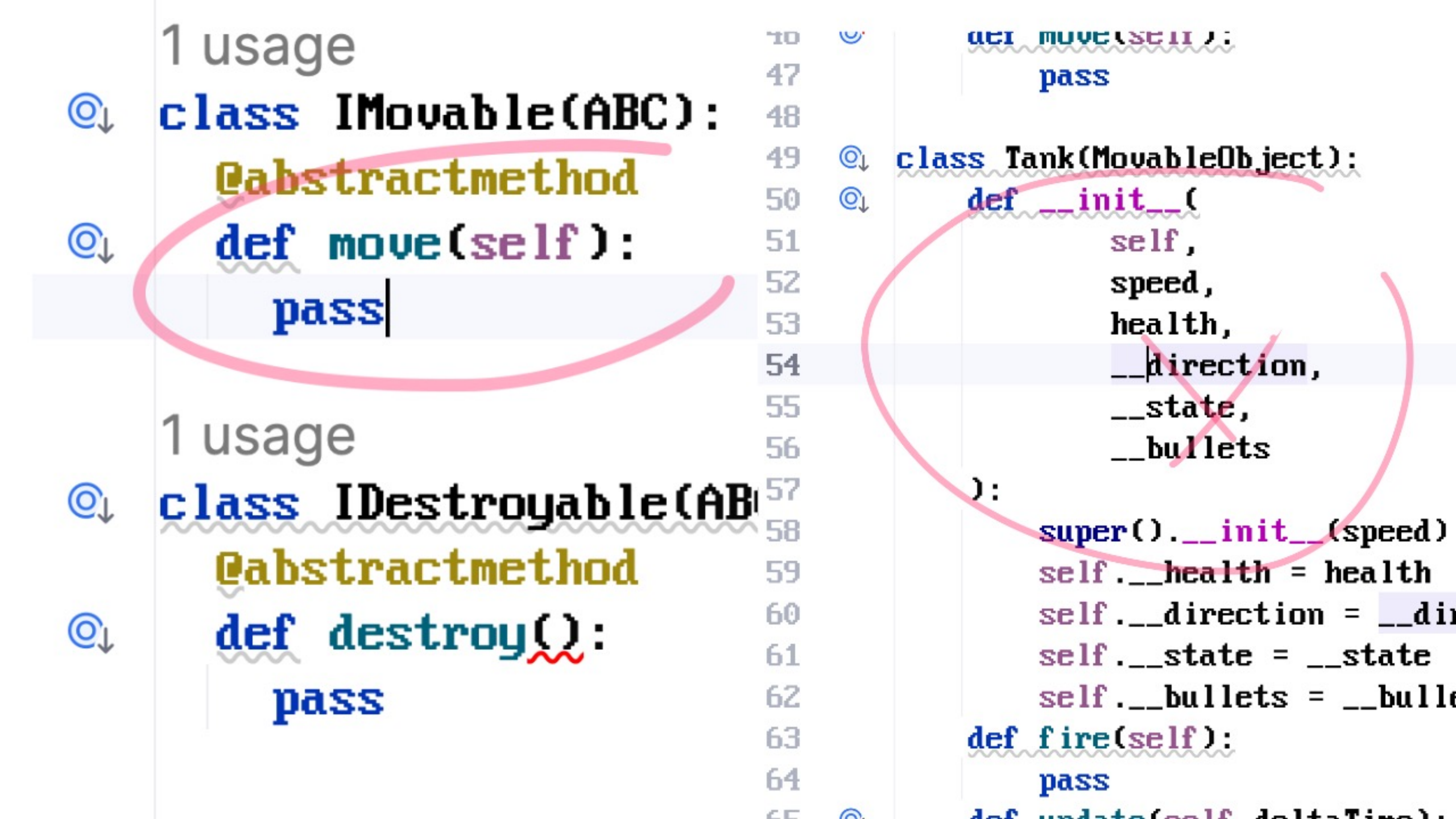
+getDuration() : Int



```python
        self.lives = lives

class EnemyTank(Tank):

    def __init__(self):
        super().__init__()
        self.type: str = "enemy"

    #funkcija update
    def update(self, deltaTime: float):
        pass

    # geteris un seteris Type
    def getType(self):
        return self.type

    def setType(self, type: string):
        self.type = type

```

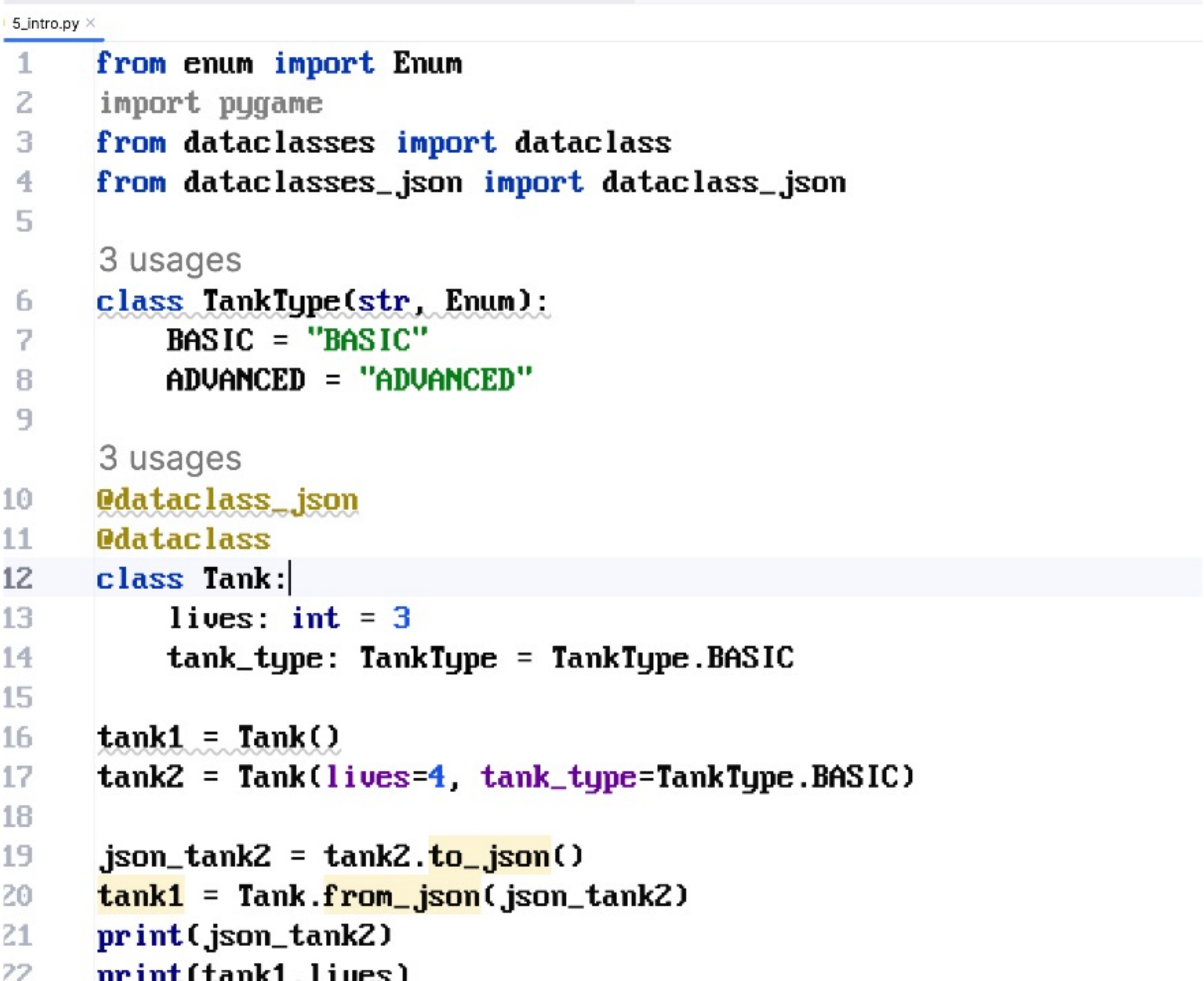


```python
class EnumTankDirection(str, Enum):
    UP = "UP"
    DOWN = "DOWN"
    LEFT = "LEFT"
    RIGHT = "RIGHT"
```

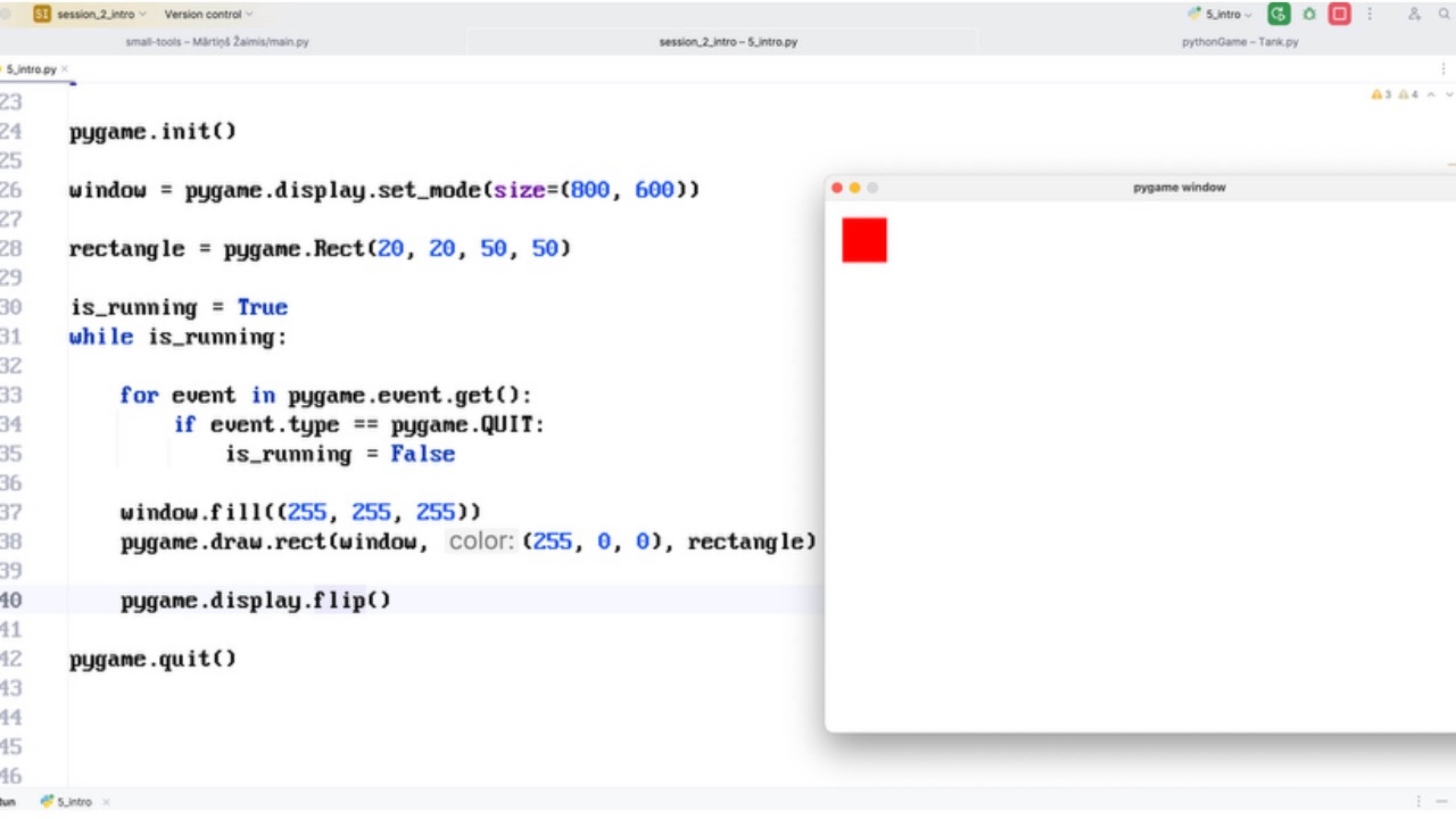


```
1 usage
class IMovable(ABC):
    @abstractmethod
    def move(self):
        pass

1 usage
class IDestroyable(AB
    @abstractmethod
    def destroy():
        pass
```

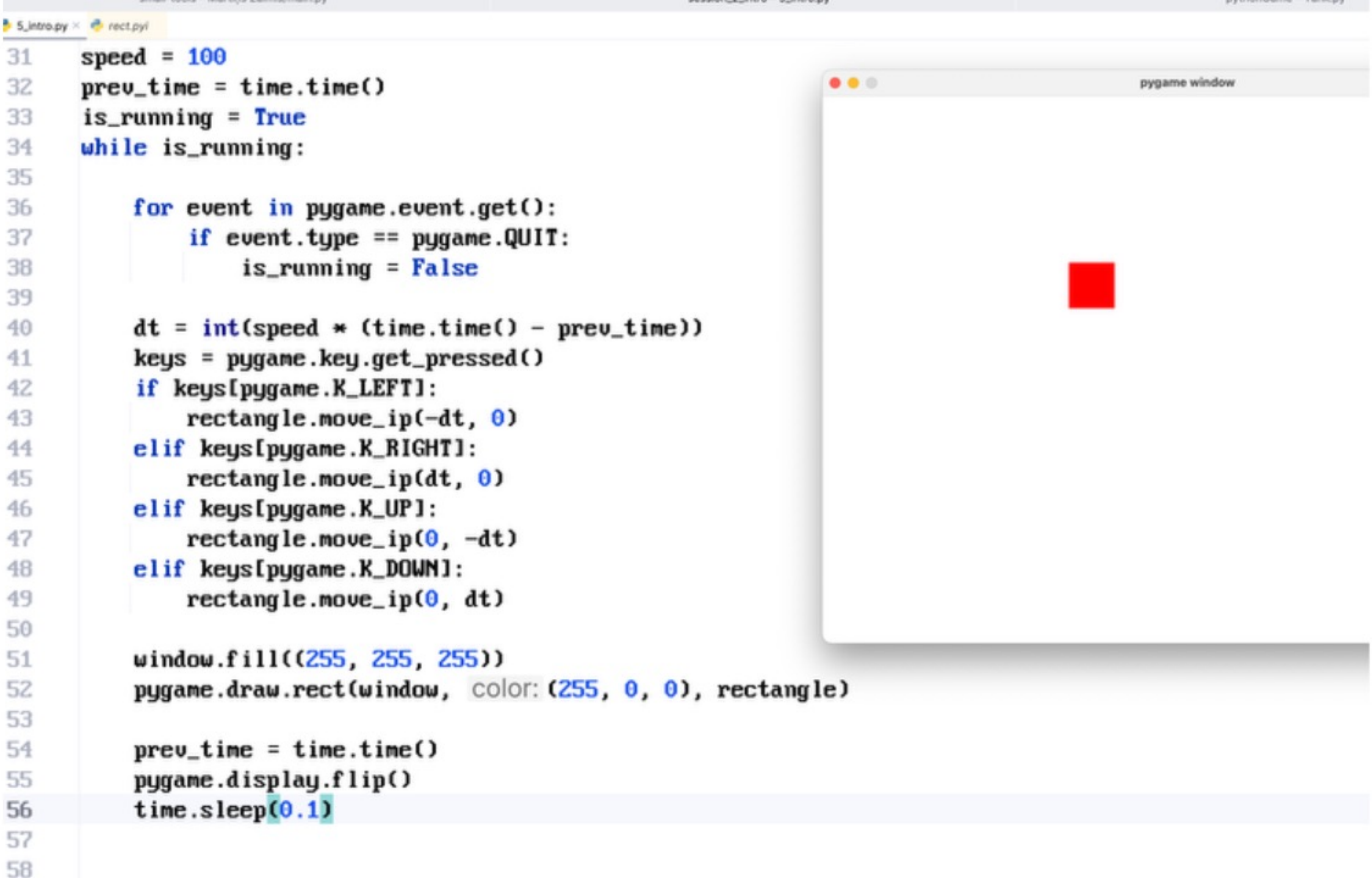
```
    def move(self):
        pass

class Tank(MovableObject):
    def __init__(
            self,
            speed,
            health,
            __direction,
            __state,
            __bullets
    ):
        super().__init__(speed)
        self.__health = health
        self.__direction = __di
        self.__state = __state
        self.__bullets = __bulle
    def fire(self):
        pass
    def update(self, deltaTime):
```



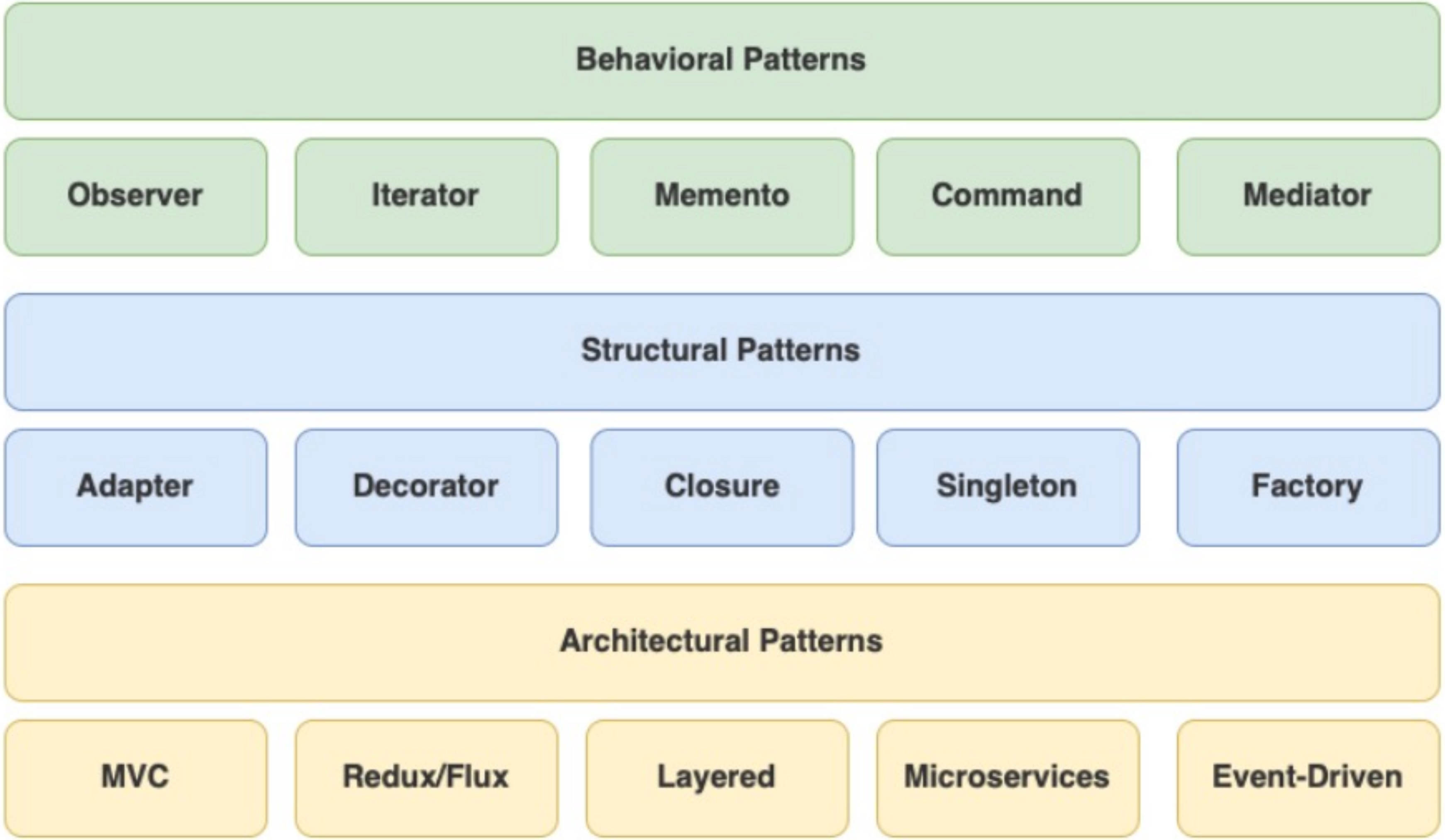
```
from enum import Enum
import pygame
from dataclasses import dataclass
from dataclasses_json import dataclass_json

3 usages
class TankType(str, Enum):
    BASIC = "BASIC"
    ADVANCED = "ADVANCED"

3 usages
@dataclass_json
@dataclass
class Tank:
    lives: int = 3
    tank_type: TankType = TankType.BASIC

tank1 = Tank()
tank2 = Tank(lives=4, tank_type=TankType.BASIC)

json_tank2 = tank2.to_json()
tank1 = Tank.from_json(json_tank2)
print(json_tank2)
print(tank1.lives)
```



```python

pygame.init()

window = pygame.display.set_mode(size=(800, 600))

rectangle = pygame.Rect(20, 20, 50, 50)

is_running = True
while is_running:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            is_running = False

    window.fill((255, 255, 255))
    pygame.draw.rect(window, color: (255, 0, 0), rectangle)

    pygame.display.flip()

pygame.quit()
```



```
speed = 100
prev_time = time.time()
is_running = True
while is_running:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            is_running = False

    dt = int(speed * (time.time() - prev_time))
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        rectangle.move_ip(-dt, 0)
    elif keys[pygame.K_RIGHT]:
        rectangle.move_ip(dt, 0)
    elif keys[pygame.K_UP]:
        rectangle.move_ip(0, -dt)
    elif keys[pygame.K_DOWN]:
        rectangle.move_ip(0, dt)

    window.fill((255, 255, 255))
    pygame.draw.rect(window, color: (255, 0, 0), rectangle)

    prev_time = time.time()
    pygame.display.flip()
    time.sleep(0.1)
```



Behavioral Patterns

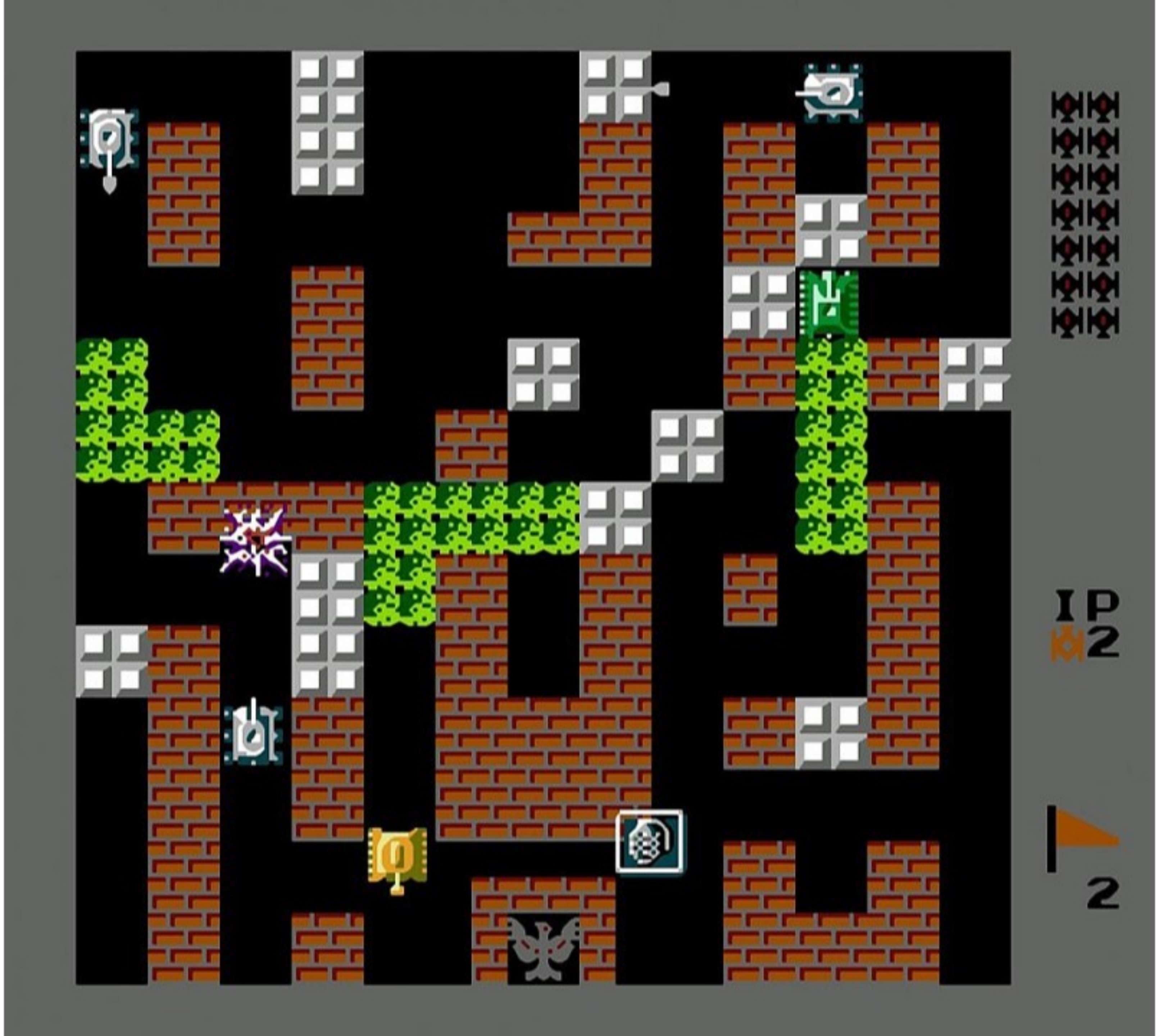
Observer Iterator Memento Command Mediator

Structural Patterns

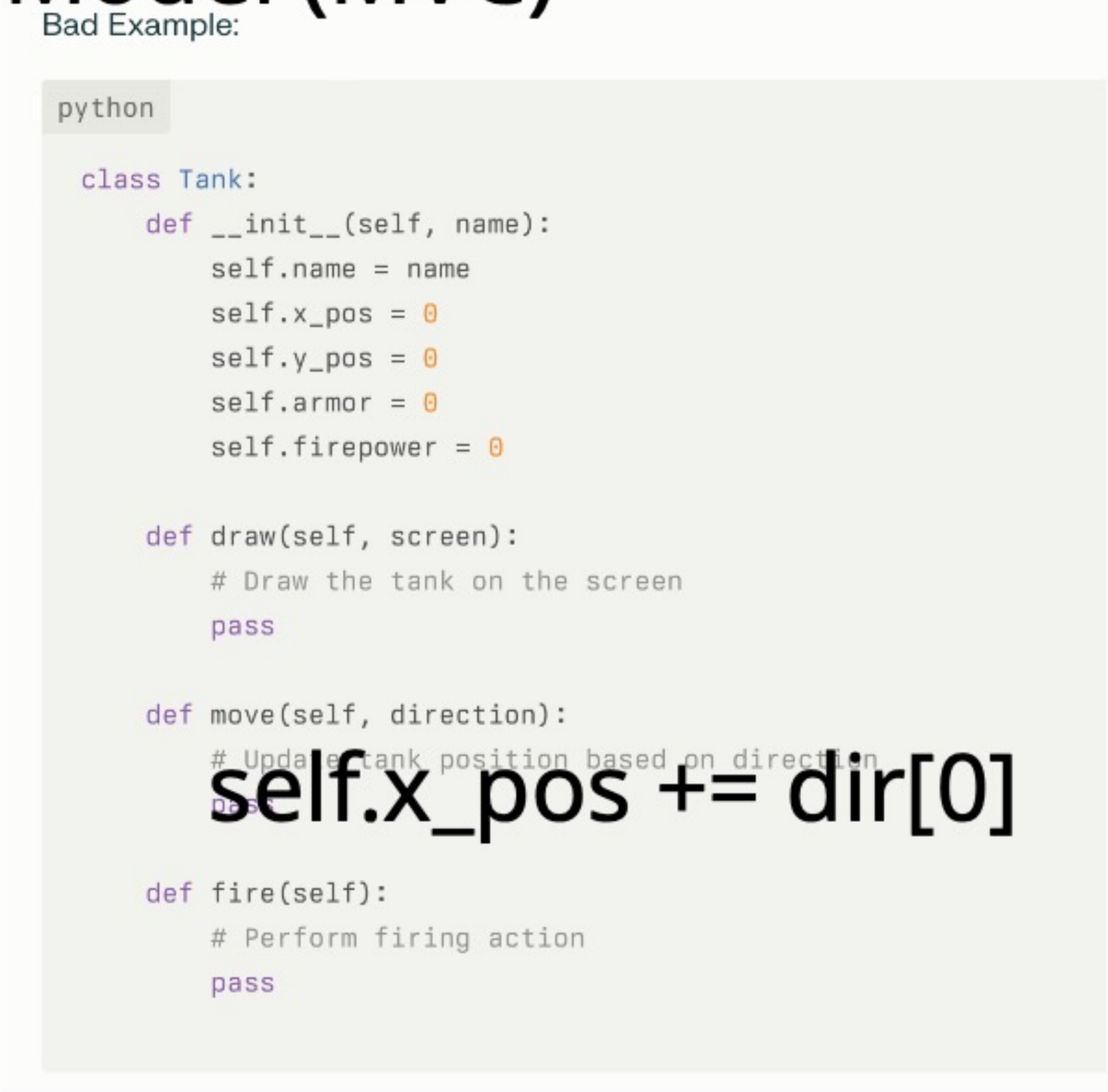
Adapter Decorator Closure Singleton Factory

Architectural Patterns

MVC Redux/Flux Layered Microservices Event-Driven


IP
2
2

Model (MVC)

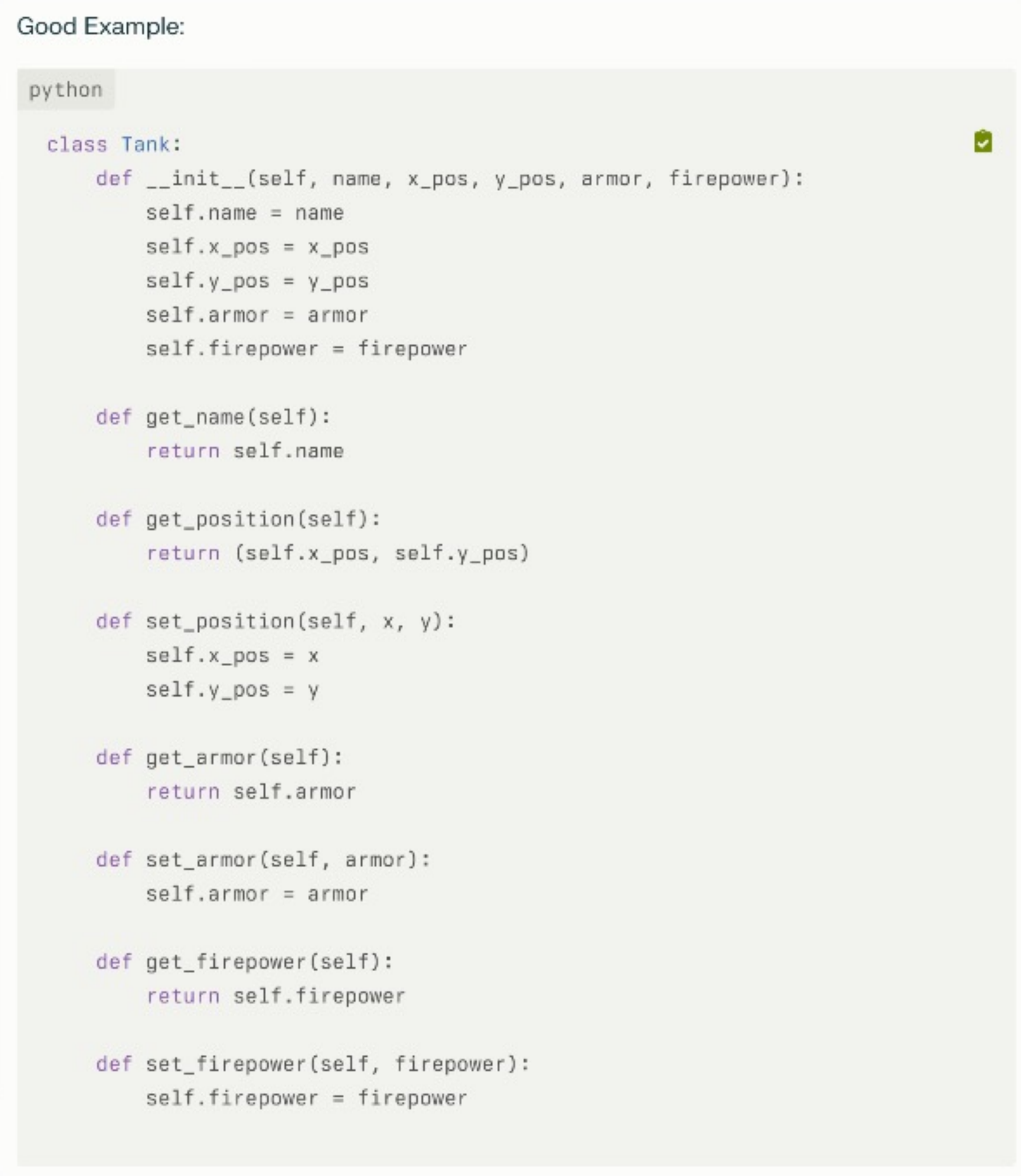


Bad Example:

python

```
class Tank:
    def __init__(self, name):
        self.name = name
        self.x_pos = 0
        self.y_pos = 0
        self.armor = 0
        self.firepower = 0

    def draw(self, screen):
        # Draw the tank on the screen
        pass

    def move(self, direction):
        # Update tank position based on direction
        pass

    def fire(self):
        # Perform firing action
        pass
```

self.x_pos += dir[0]

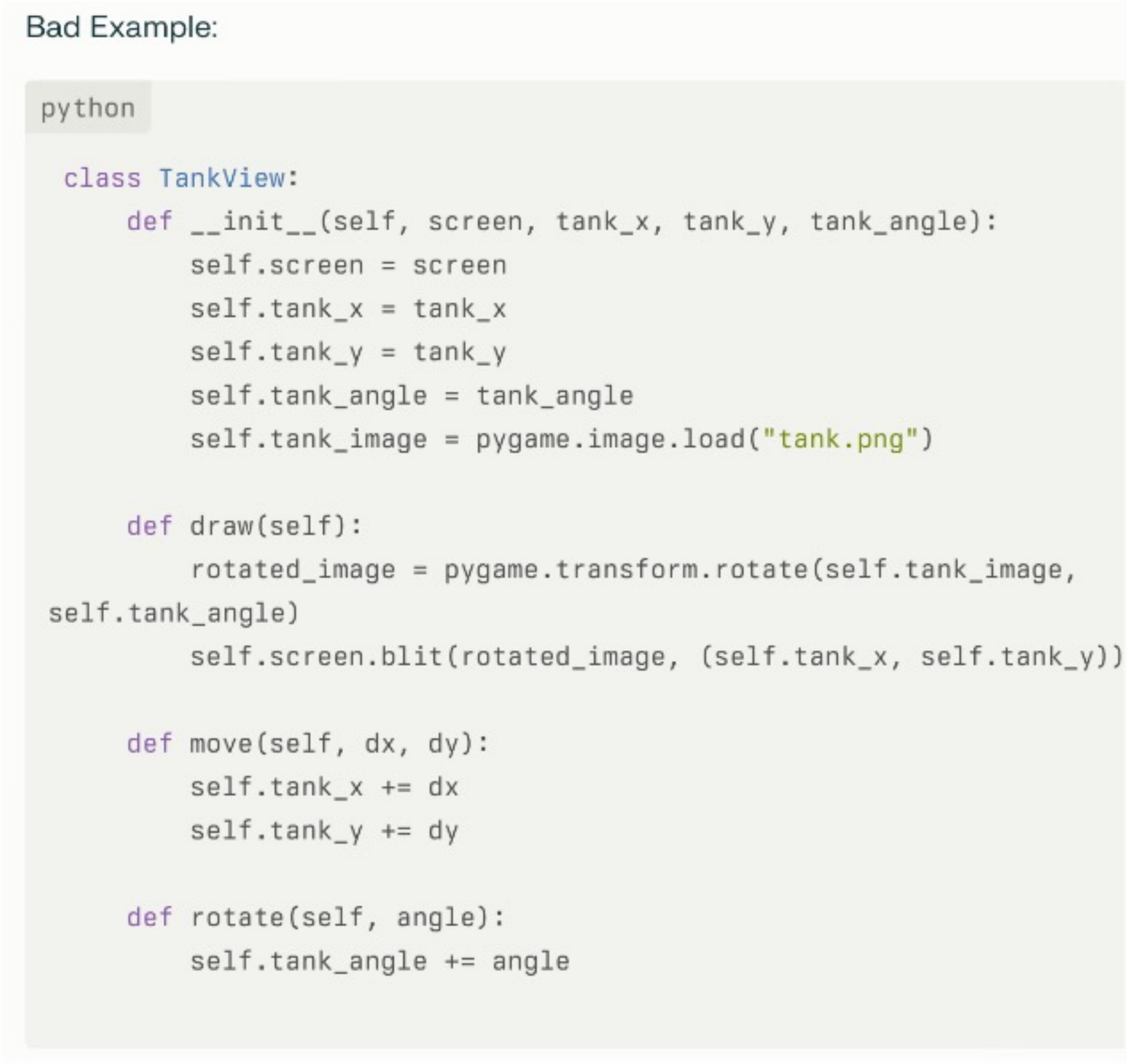


Good Example:

python

```
class Tank:
    def __init__(self, name, x_pos, y_pos, armor, firepower):
        self.name = name
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.armor = armor
        self.firepower = firepower

    def get_name(self):
        return self.name

    def get_position(self):
        return (self.x_pos, self.y_pos)

    def set_position(self, x, y):
        self.x_pos = x
        self.y_pos = y

    def get_armor(self):
        return self.armor

    def set_armor(self, armor):
        self.armor = armor

    def get_firepower(self):
        return self.firepower

    def set_firepower(self, firepower):
        self.firepower = firepower
```

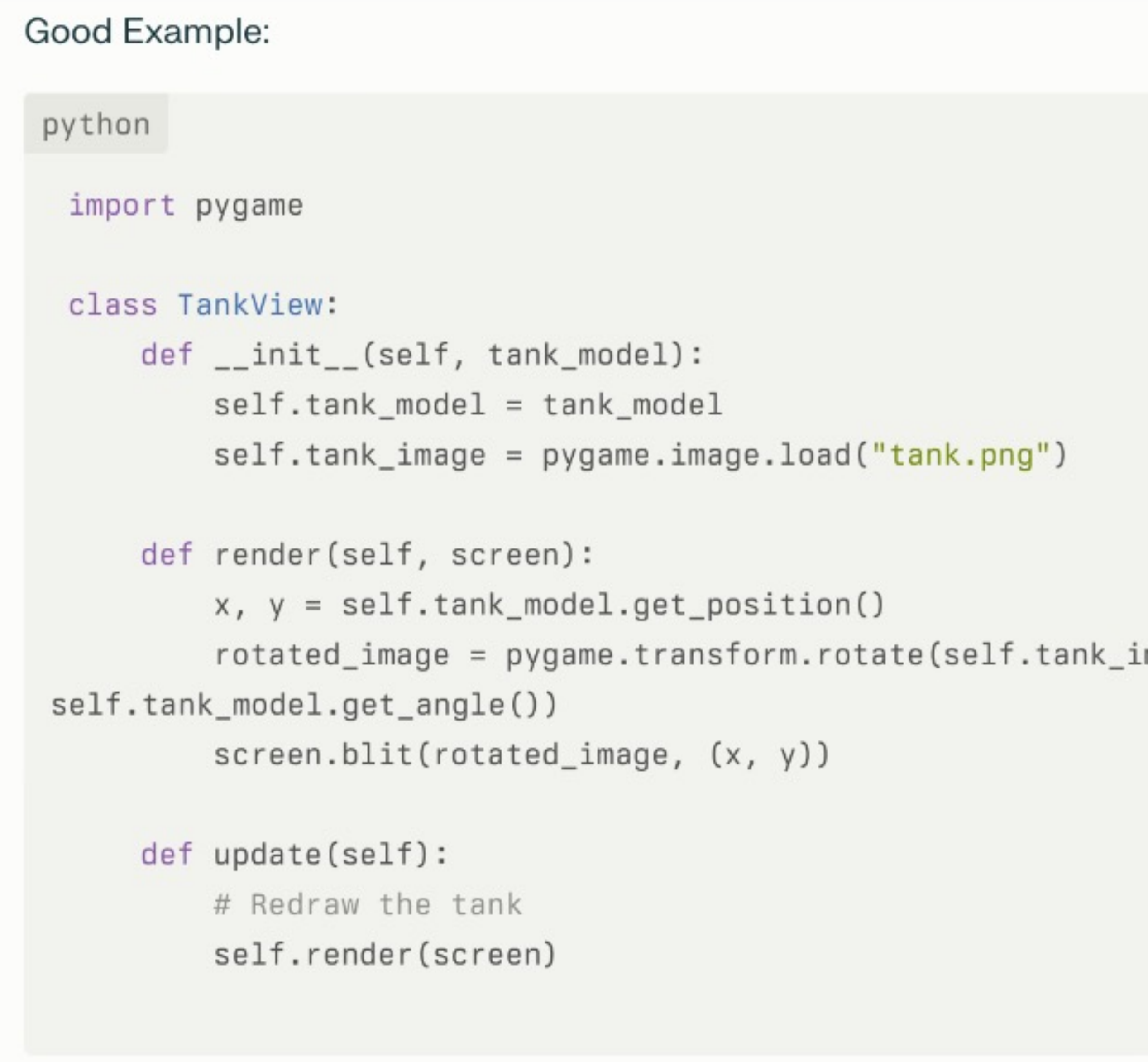
View (MVC)



Bad Example:

python

```python
class TankView:
    def __init__(self, screen, tank_x, tank_y, tank_angle):
        self.screen = screen
        self.tank_x = tank_x
        self.tank_y = tank_y
        self.tank_angle = tank_angle
        self.tank_image = pygame.image.load("tank.png")

    def draw(self):
        rotated_image = pygame.transform.rotate(self.tank_image,
self.tank_angle)
        self.screen.blit(rotated_image, (self.tank_x, self.tank_y))

    def move(self, dx, dy):
        self.tank_x += dx
        self.tank_y += dy

    def rotate(self, angle):
        self.tank_angle += angle
```

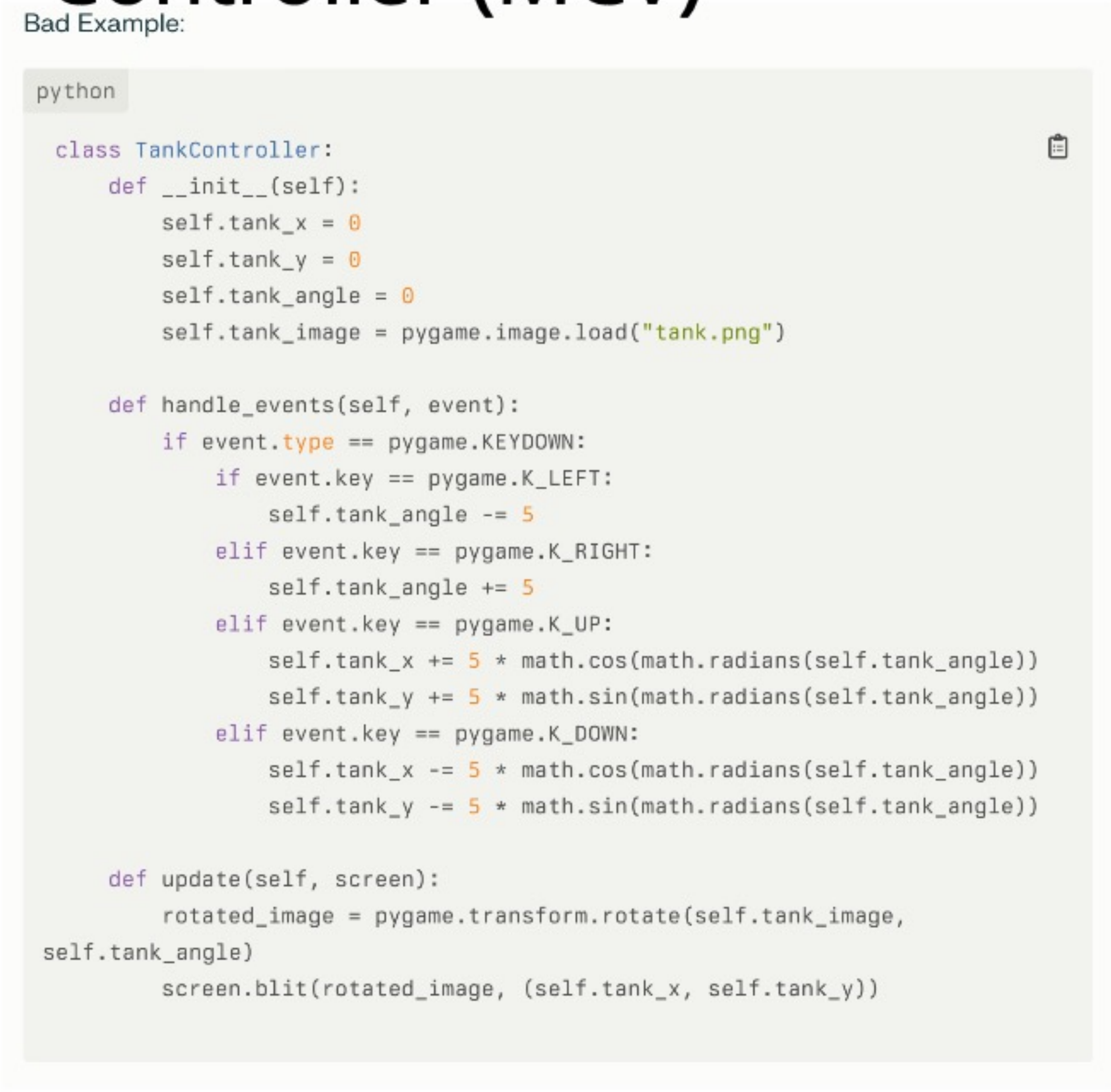


Good Example:

python

```python
import pygame

class TankView:
    def __init__(self, tank_model):
        self.tank_model = tank_model
        self.tank_image = pygame.image.load("tank.png")

    def render(self, screen):
        x, y = self.tank_model.get_position()
        rotated_image = pygame.transform.rotate(self.tank_i
self.tank_model.get_angle())
        screen.blit(rotated_image, (x, y))

    def update(self):
        # Redraw the tank
        self.render(screen)
```

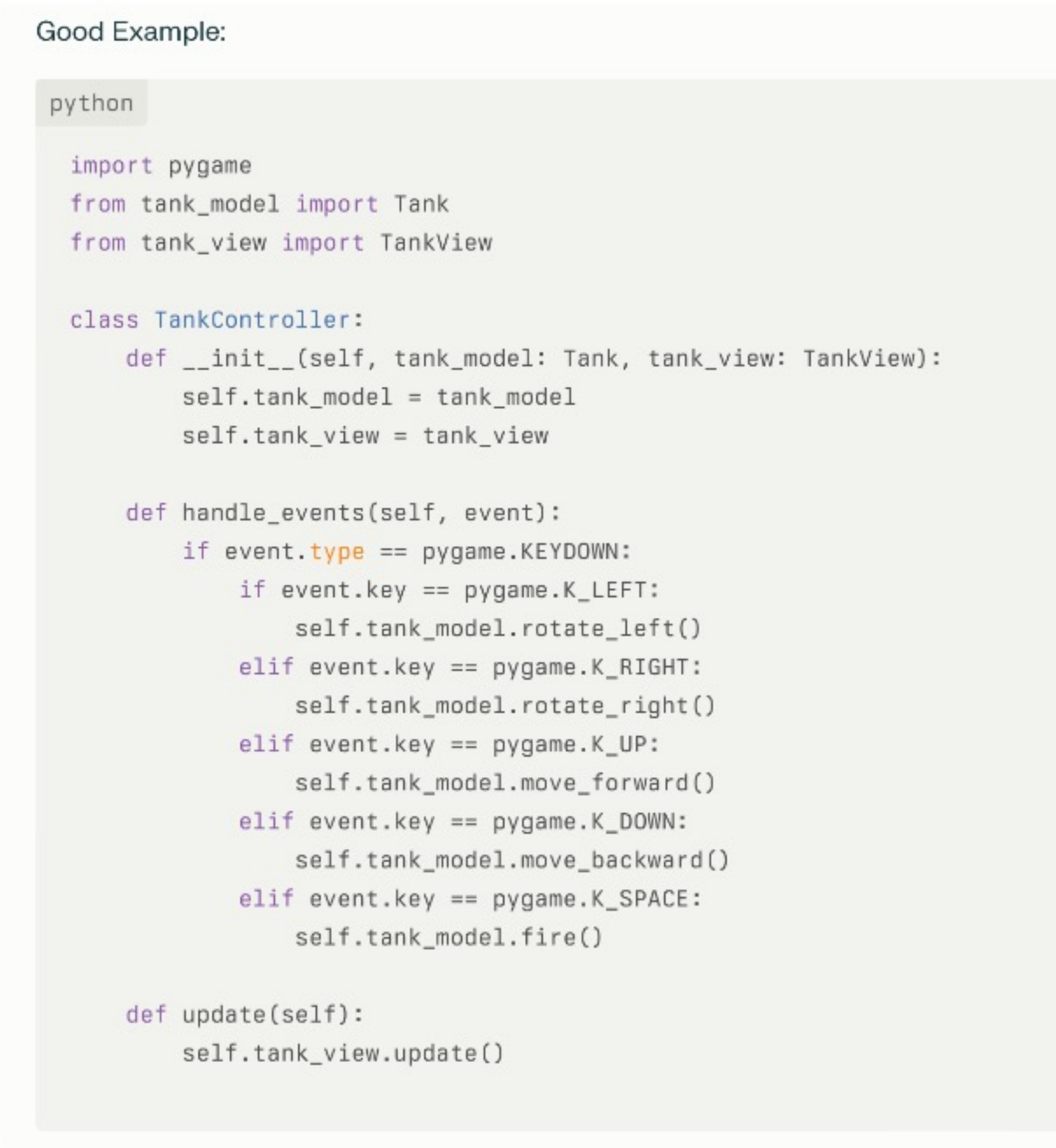
Controller (MCV)



Bad Example:

python

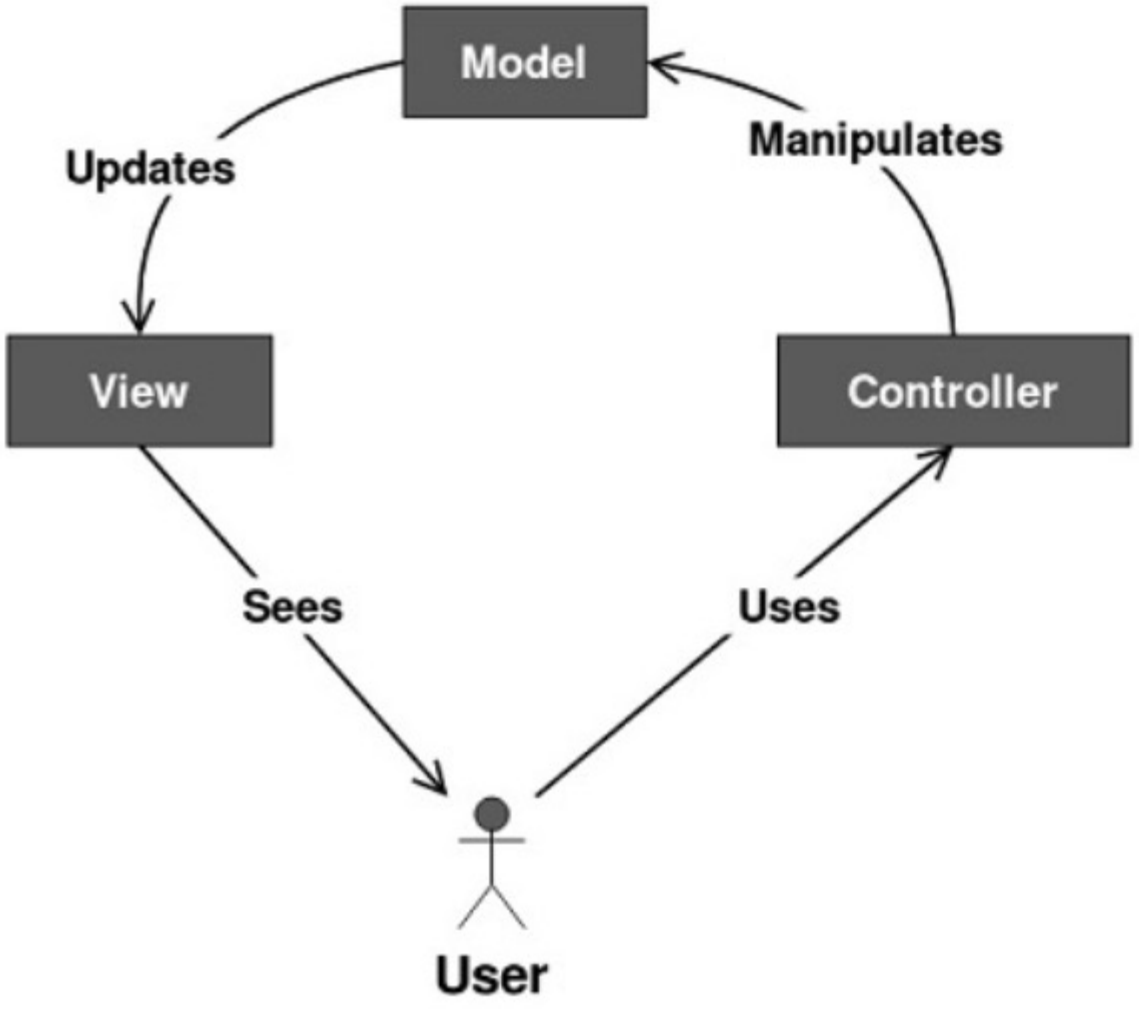
```python
class TankController:
    def __init__(self):
        self.tank_x = 0
        self.tank_y = 0
        self.tank_angle = 0
        self.tank_image = pygame.image.load("tank.png")

    def handle_events(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                self.tank_angle -= 5
            elif event.key == pygame.K_RIGHT:
                self.tank_angle += 5
            elif event.key == pygame.K_UP:
                self.tank_x += 5 * math.cos(math.radians(self.tank_angle))
                self.tank_y += 5 * math.sin(math.radians(self.tank_angle))
            elif event.key == pygame.K_DOWN:
                self.tank_x -= 5 * math.cos(math.radians(self.tank_angle))
                self.tank_y -= 5 * math.sin(math.radians(self.tank_angle))

    def update(self, screen):
        rotated_image = pygame.transform.rotate(self.tank_image,
self.tank_angle)
        screen.blit(rotated_image, (self.tank_x, self.tank_y))
```

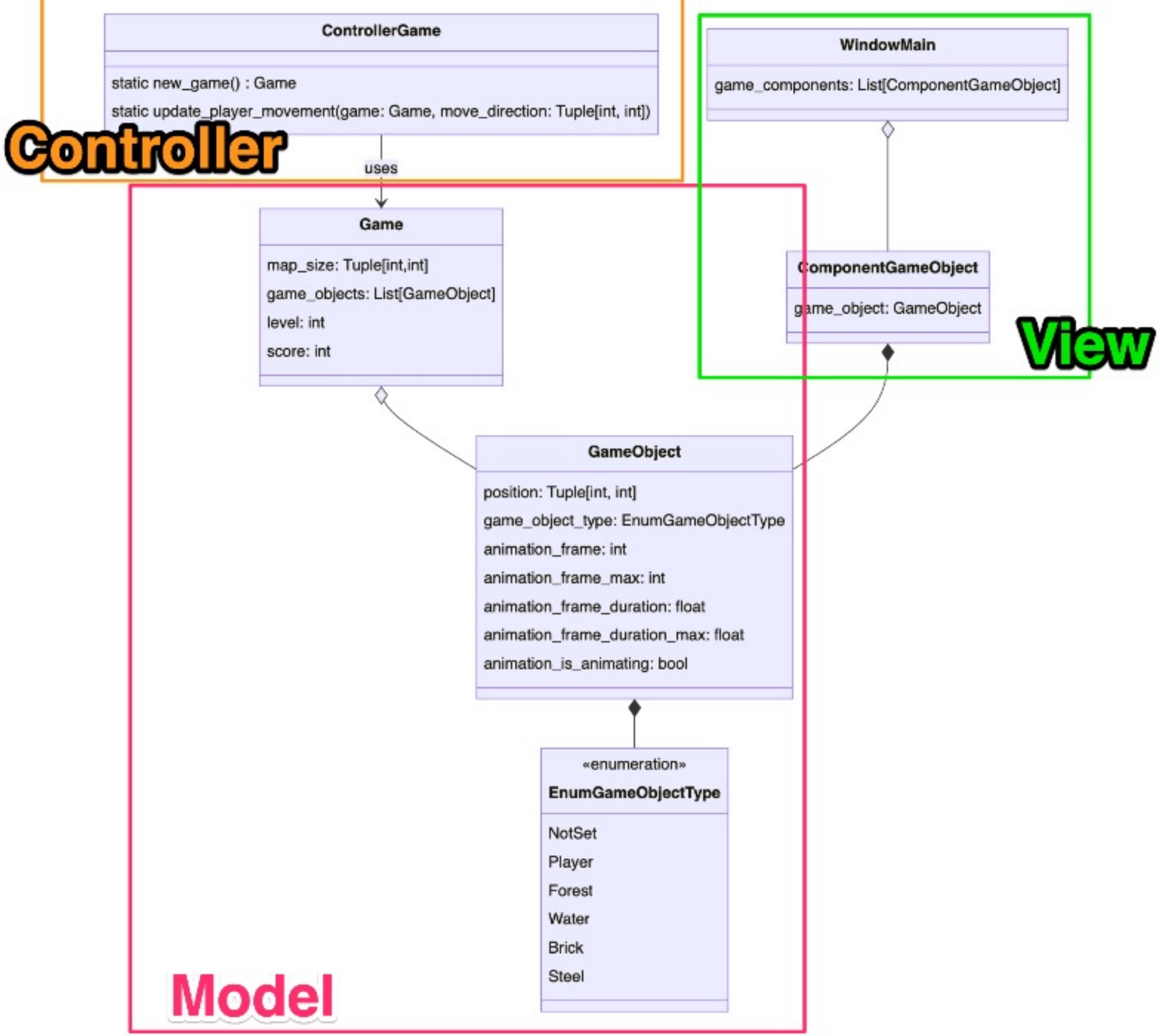


Good Example:

python

```python
import pygame
from tank_model import Tank
from tank_view import TankView

class TankController:
    def __init__(self, tank_model: Tank, tank_view: TankView):
        self.tank_model = tank_model
        self.tank_view = tank_view

    def handle_events(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                self.tank_model.rotate_left()
            elif event.key == pygame.K_RIGHT:
                self.tank_model.rotate_right()
            elif event.key == pygame.K_UP:
                self.tank_model.move_forward()
            elif event.key == pygame.K_DOWN:
                self.tank_model.move_backward()
            elif event.key == pygame.K_SPACE:
                self.tank_model.fire()

    def update(self):
        self.tank_view.update()
```


Model
Updates
Manipulates
View
Controller
Sees
Uses
User



Controller

ControllerGame

static new_game() : Game

static update_player_movement(game: Game, move_direction: Tuple[int, int])

uses

View

WindowMain

game_components: List[ComponentGameObject]

ComponentGameObject

game_object: GameObject

Model

Game

map_size: Tuple[int,int]

game_objects: List[GameObject]

level: int

score: int

GameObject

position: Tuple[int, int]

game_object_type: EnumGameObjectType

animation_frame: int

animation_frame_max: int

animation_frame_duration: float

animation_frame_duration_max: float

animation_is_animating: bool

«enumeration»

EnumGameObjectType

NotSet

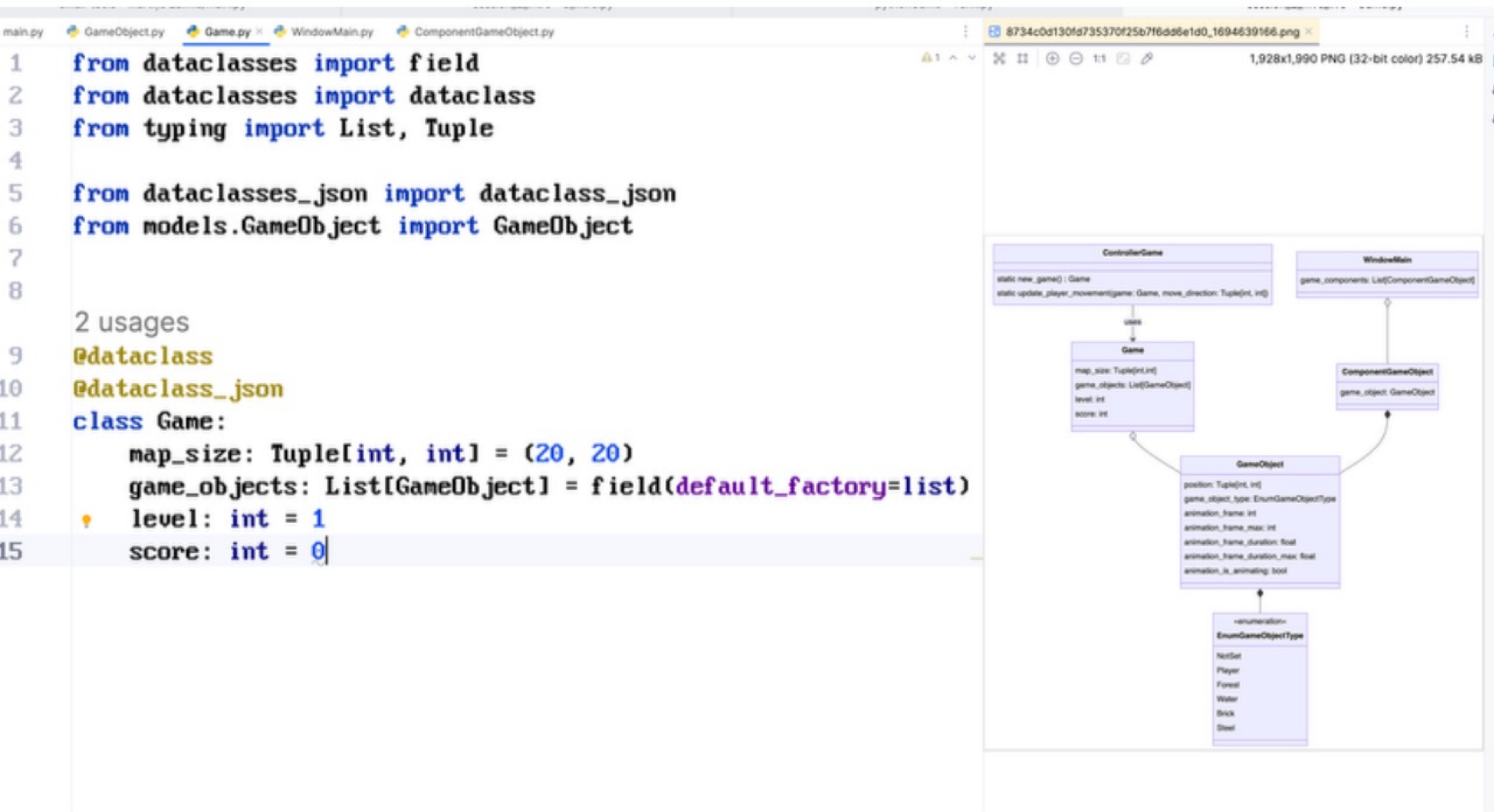
Player

Forest

Water

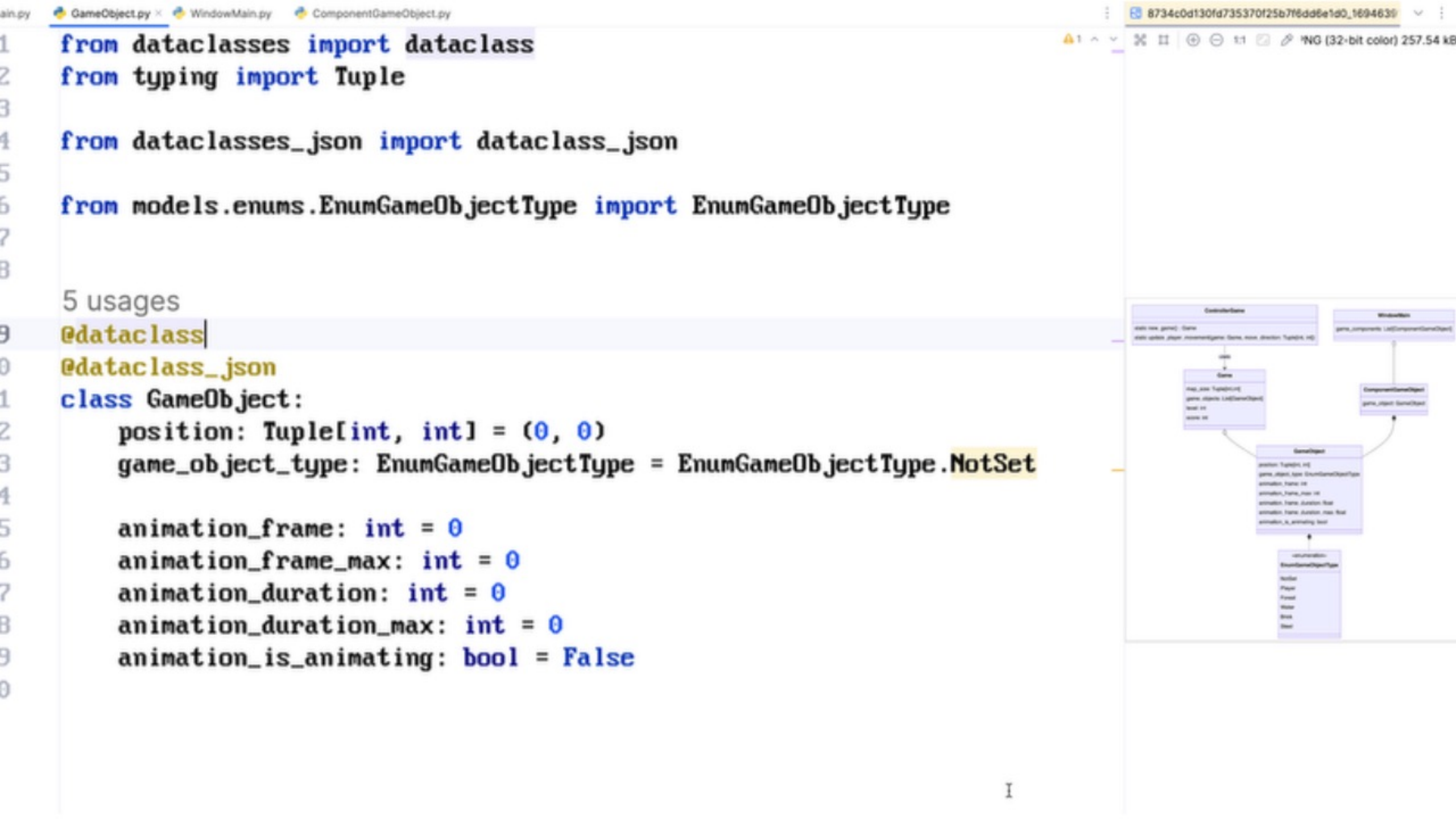
Brick

Steel

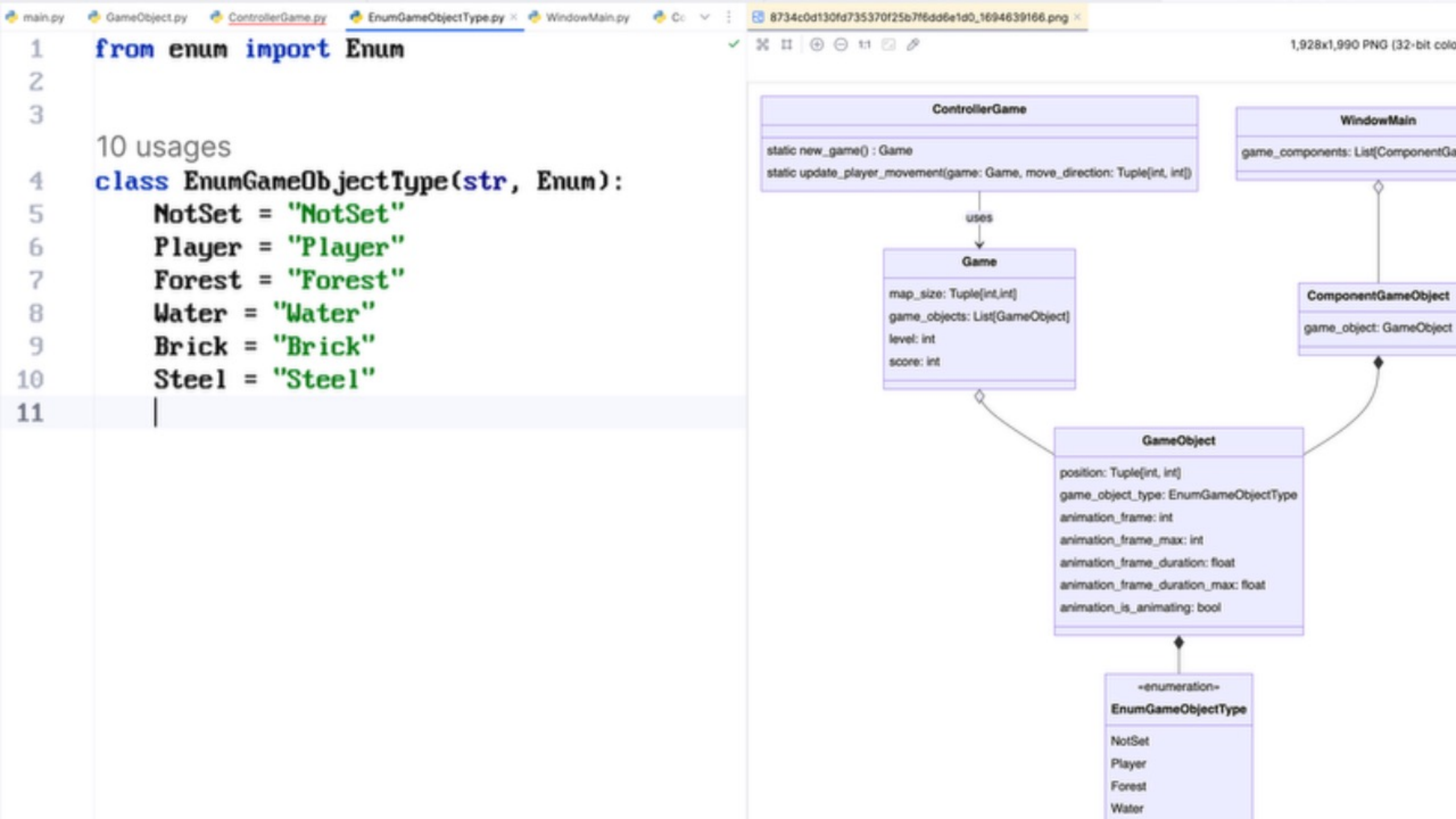


main.py
GameObject.py
Game.py
WindowMain.py
ComponentGameObject.py
8734c0d130fd735370f25b7f6dd6e1d0_1694639166.png
1:1
1,928x1,990 PNG (32-bit color) 257.54 kB
2 usages
ControllerGame
static new_game() : Game
static update_player_movement(game: Game, move_direction: Tuple[int, int])
uses
WindowMain
game_components: List[ComponentGameObject]
Game
map_size: Tuple[int,int]
game_objects: List[GameObject]
level: int
score: int
ComponentGameObject
game_object: GameObject
GameObject
position: Tuple[int, int]
game_object_type: EnumGameObjectType
animation_frame: int
animation_frame_max: int
animation_frame_duration: float
animation_frame_duration_max: float
animation_is_animating: bool
«enumeration»
EnumGameObjectType
NotSet
Player
Forest
Water
Brick
Steel

```python
from dataclasses import field
from dataclasses import dataclass
from typing import List, Tuple

from dataclasses_json import dataclass_json
from models.GameObject import GameObject


@dataclass
@dataclass_json
class Game:
    map_size: Tuple[int, int] = (20, 20)
    game_objects: List[GameObject] = field(default_factory=list)
    level: int = 1
    score: int = 0
```


ain.py
GameObject.py
WindowMain.py
ComponentGameObject.py
8734c0d130fd735370f25b7f6dd6e1d0_1694638
PNG (32-bit color) 257.54 kB
1:1
5 usages
ControllerGame
static new_game() : Game
static update_player_movement(game: Game, move_direction: Tuple[int, int])
uses
Game
map_size: Tuple[int,int]
game_objects: List[GameObject]
level: int
score: int
WindowMain
game_components: List[ComponentGameObject]
ComponentGameObject
game_object: GameObject
GameObject
position: Tuple[int, int]
game_object_type: EnumGameObjectType
animation_frame: int
animation_frame_max: int
animation_frame_duration: float
animation_frame_duration_max: float
animation_is_animating: bool
«enumeration»
EnumGameObjectType
NotSet
Player
Forest
Water
Brick
Steel

```python
from dataclasses import dataclass
from typing import Tuple

from dataclasses_json import dataclass_json

from models.enums.EnumGameObjectType import EnumGameObjectType


@dataclass
@dataclass_json
class GameObject:
    position: Tuple[int, int] = (0, 0)
    game_object_type: EnumGameObjectType = EnumGameObjectType.NotSet

    animation_frame: int = 0
    animation_frame_max: int = 0
    animation_duration: int = 0
    animation_duration_max: int = 0
    animation_is_animating: bool = False

```



```python
from enum import Enum


class EnumGameObjectType(str, Enum):
    NotSet = "NotSet"
    Player = "Player"
    Forest = "Forest"
    Water = "Water"
    Brick = "Brick"
    Steel = "Steel"
    
```

ControllerGame

static new_game() : Game

static update_player_movement(game: Game, move_direction: Tuple[int, int])

WindowMain

game_components: List[ComponentGa

Game

map_size: Tuple[int,int]

game_objects: List[GameObject]

level: int

score: int

ComponentGameObject

game_object: GameObject

GameObject

position: Tuple[int, int]

game_object_type: EnumGameObjectType

animation_frame: int

animation_frame_max: int

animation_frame_duration: float

animation_frame_duration_max: float

animation_is_animating: bool

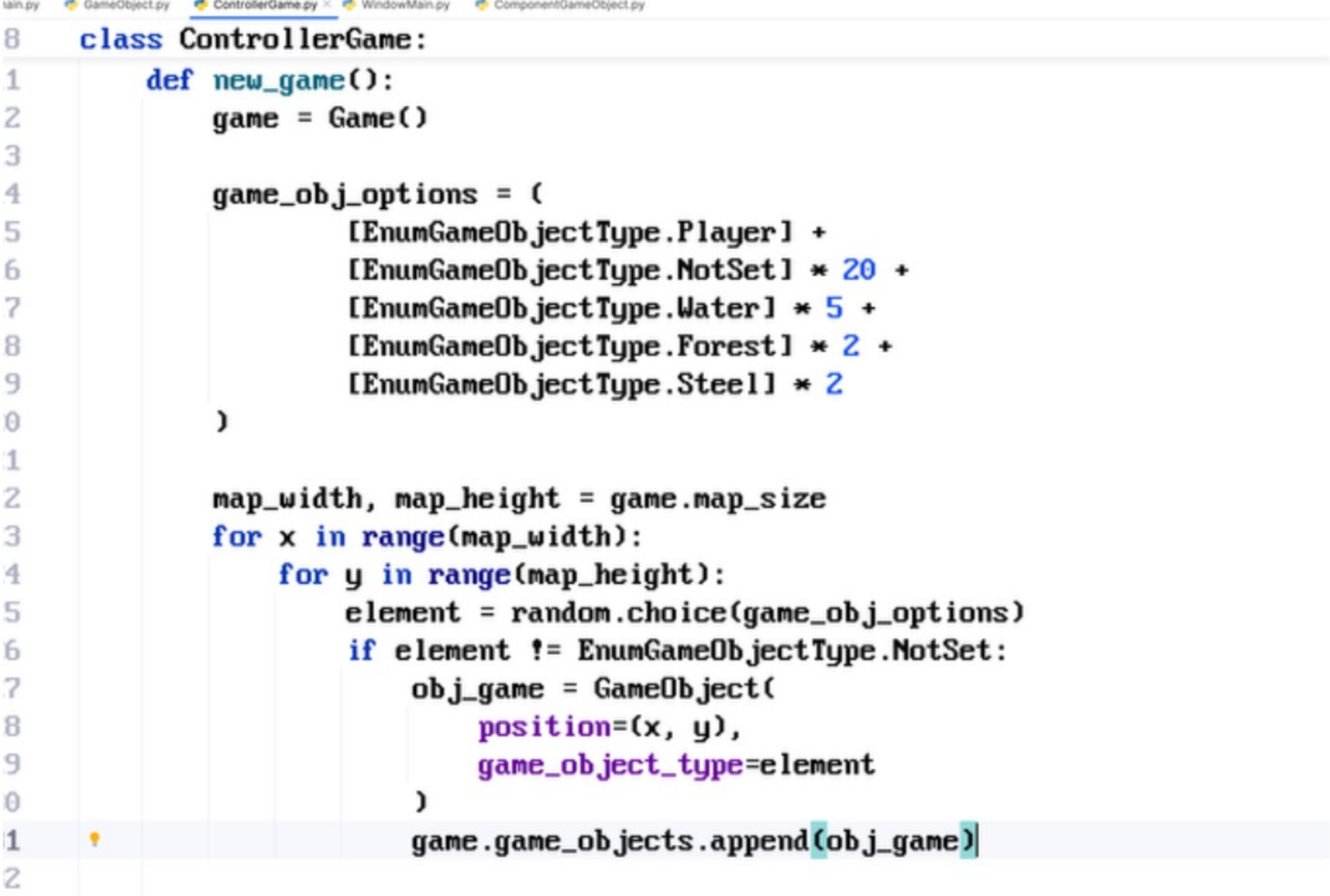
«enumeration»
EnumGameObjectType

NotSet

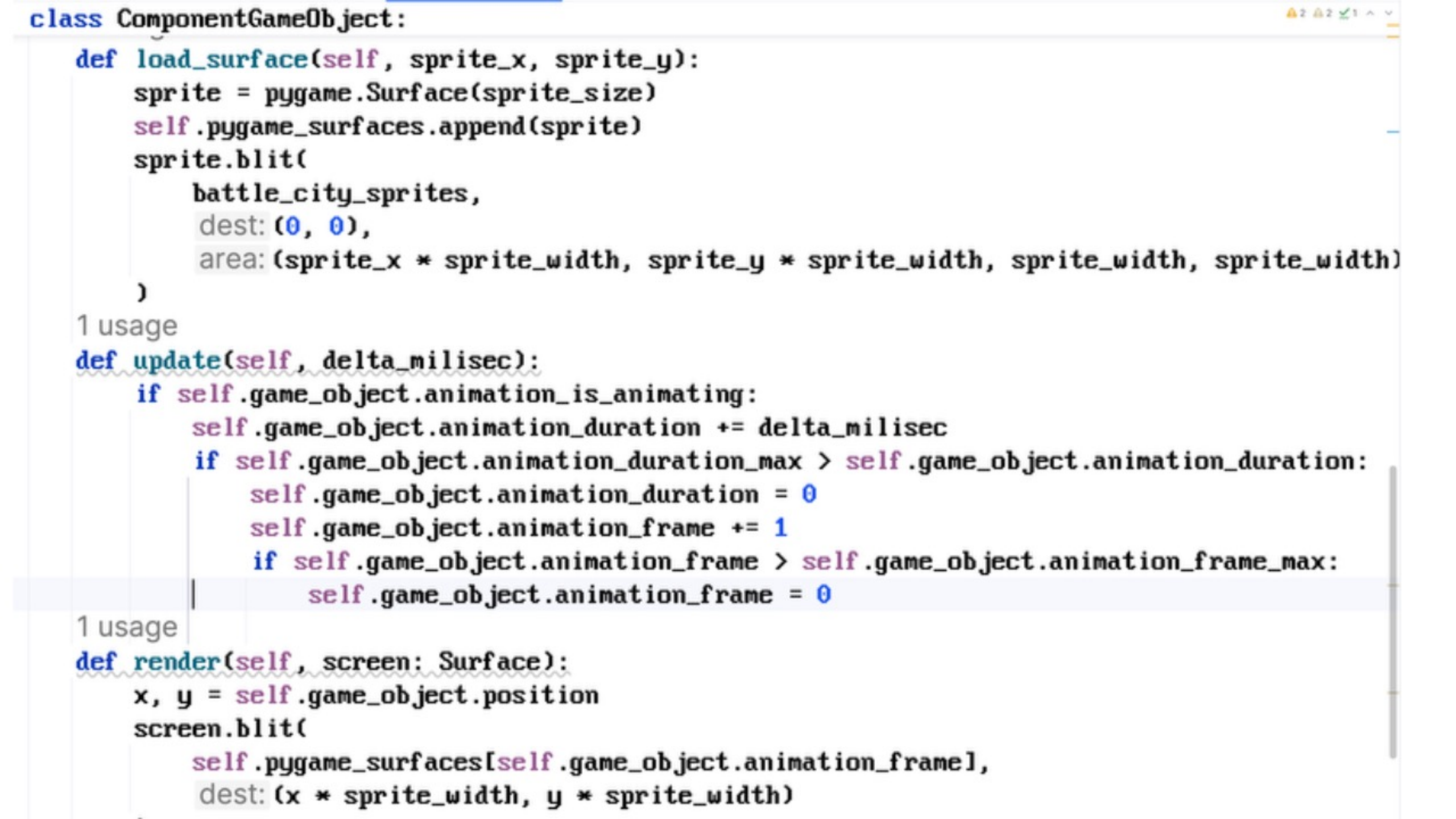
Player

Forest

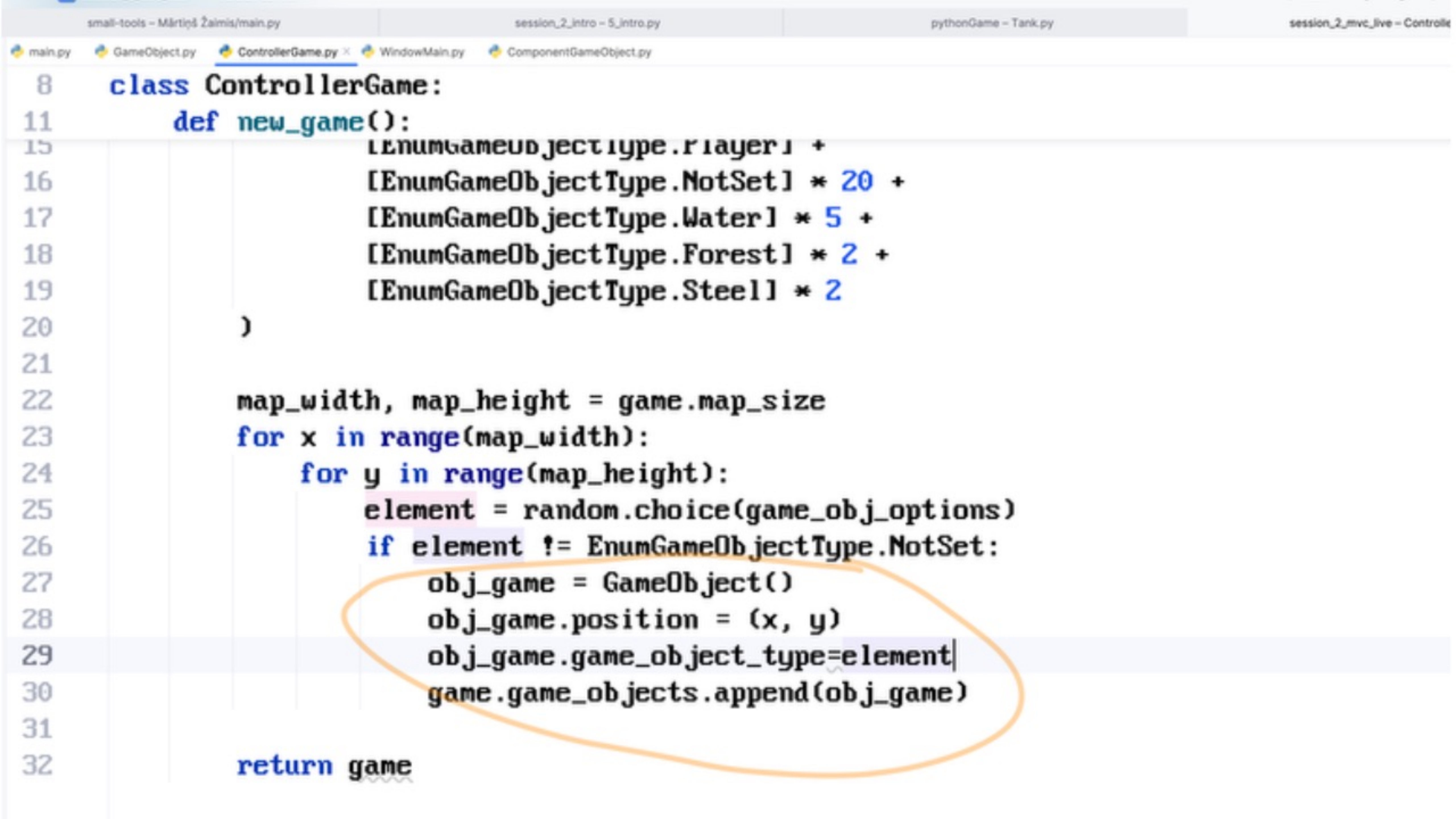
Water



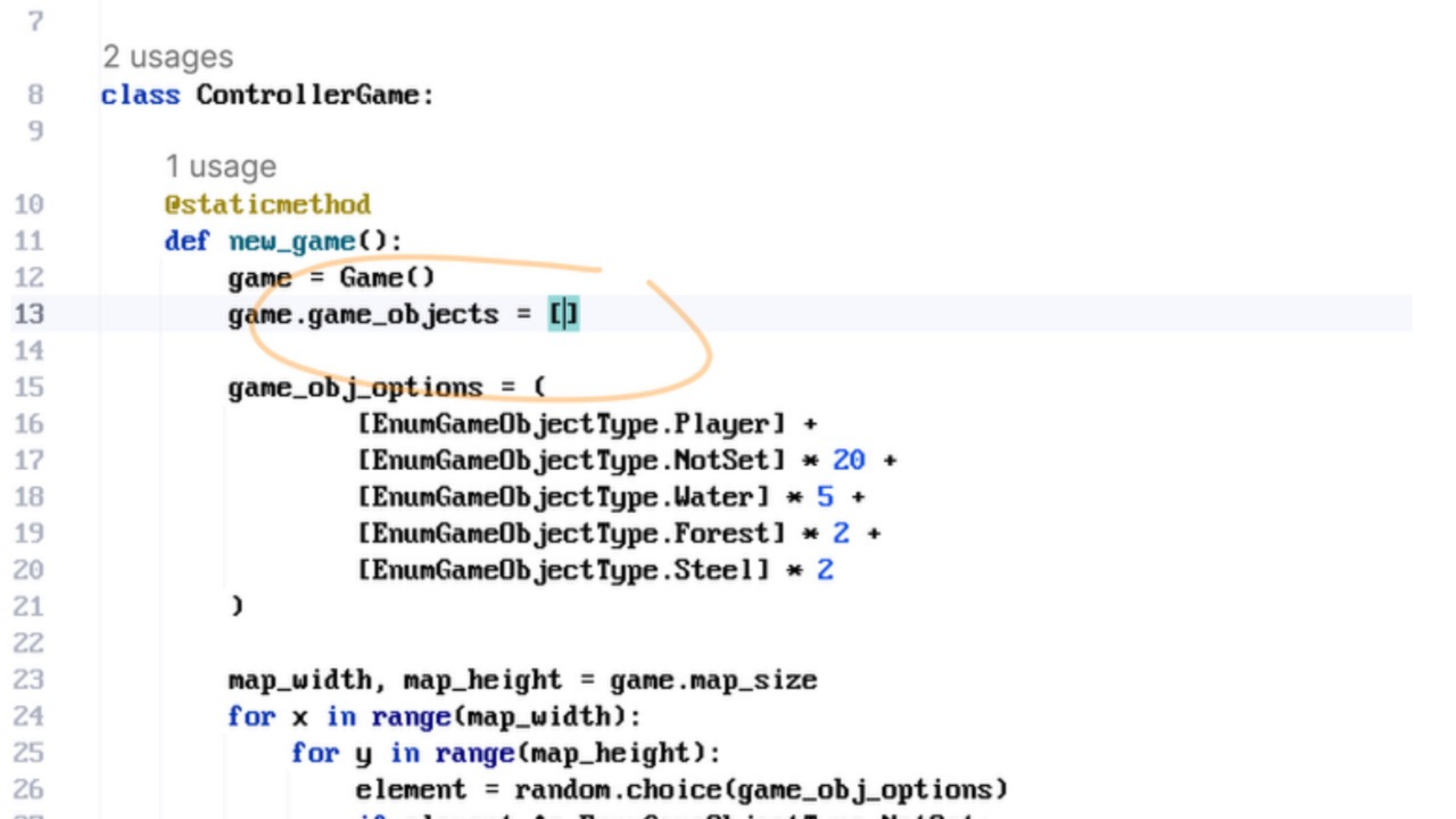
```python
class ControllerGame:
    def new_game():
        game = Game()

        game_obj_options = (
                [EnumGameObjectType.Player] +
                [EnumGameObjectType.NotSet] * 20 +
                [EnumGameObjectType.Water] * 5 +
                [EnumGameObjectType.Forest] * 2 +
                [EnumGameObjectType.Steel] * 2
        )

        map_width, map_height = game.map_size
        for x in range(map_width):
            for y in range(map_height):
                element = random.choice(game_obj_options)
                if element != EnumGameObjectType.NotSet:
                    obj_game = GameObject(
                        position=(x, y),
                        game_object_type=element
                    )
                    game.game_objects.append(obj_game)

```



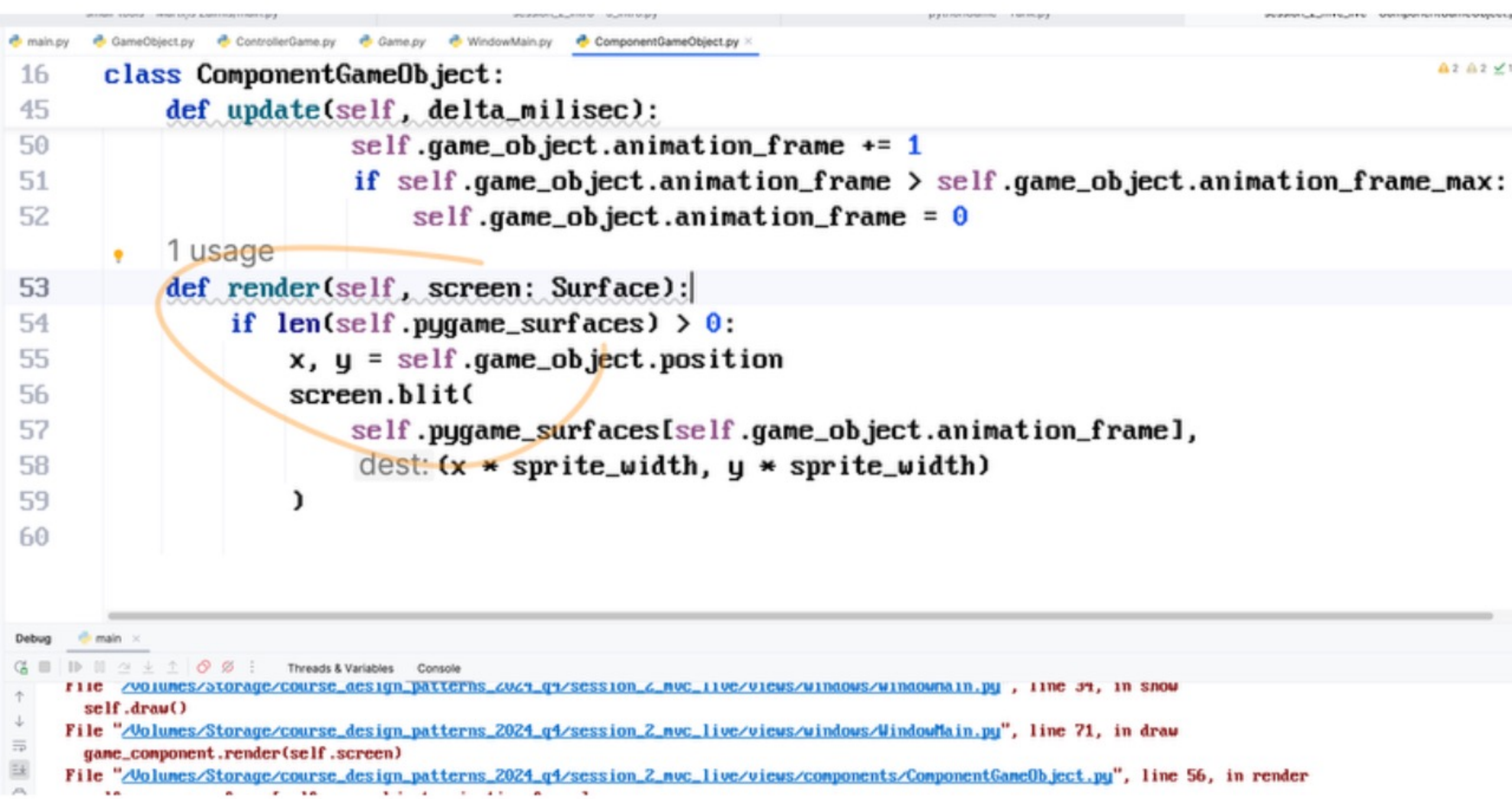
```
class ComponentGameObject:
    def load_surface(self, sprite_x, sprite_y):
        sprite = pygame.Surface(sprite_size)
        self.pygame_surfaces.append(sprite)
        sprite.blit(
            battle_city_sprites,
            dest: (0, 0),
            area: (sprite_x * sprite_width, sprite_y * sprite_width, sprite_width, sprite_width)
        )
    1 usage
    def update(self, delta_milisec):
        if self.game_object.animation_is_animating:
            self.game_object.animation_duration += delta_milisec
            if self.game_object.animation_duration_max > self.game_object.animation_duration:
                self.game_object.animation_duration = 0
                self.game_object.animation_frame += 1
                if self.game_object.animation_frame > self.game_object.animation_frame_max:
                    self.game_object.animation_frame = 0
    1 usage
    def render(self, screen: Surface):
        x, y = self.game_object.position
        screen.blit(
            self.pygame_surfaces[self.game_object.animation_frame],
            dest: (x * sprite_width, y * sprite_width)
```


small-tools – Mārtiņš Zaimis/main.py
session_2_intro – 5_intro.py
pythonGame – Tank.py
session_2_mvc_live – Controlle
main.py
GameObject.py
ControllerGame.py
WindowMain.py
ComponentGameObject.py

```
class ControllerGame:
    def new_game():
                [EnumGameObjectType.Player] +
                [EnumGameObjectType.NotSet] * 20 +
                [EnumGameObjectType.Water] * 5 +
                [EnumGameObjectType.Forest] * 2 +
                [EnumGameObjectType.Steel] * 2
        )

        map_width, map_height = game.map_size
        for x in range(map_width):
            for y in range(map_height):
                element = random.choice(game_obj_options)
                if element != EnumGameObjectType.NotSet:
                    obj_game = GameObject()
                    obj_game.position = (x, y)
                    obj_game.game_object_type=element
                    game.game_objects.append(obj_game)

        return game
```

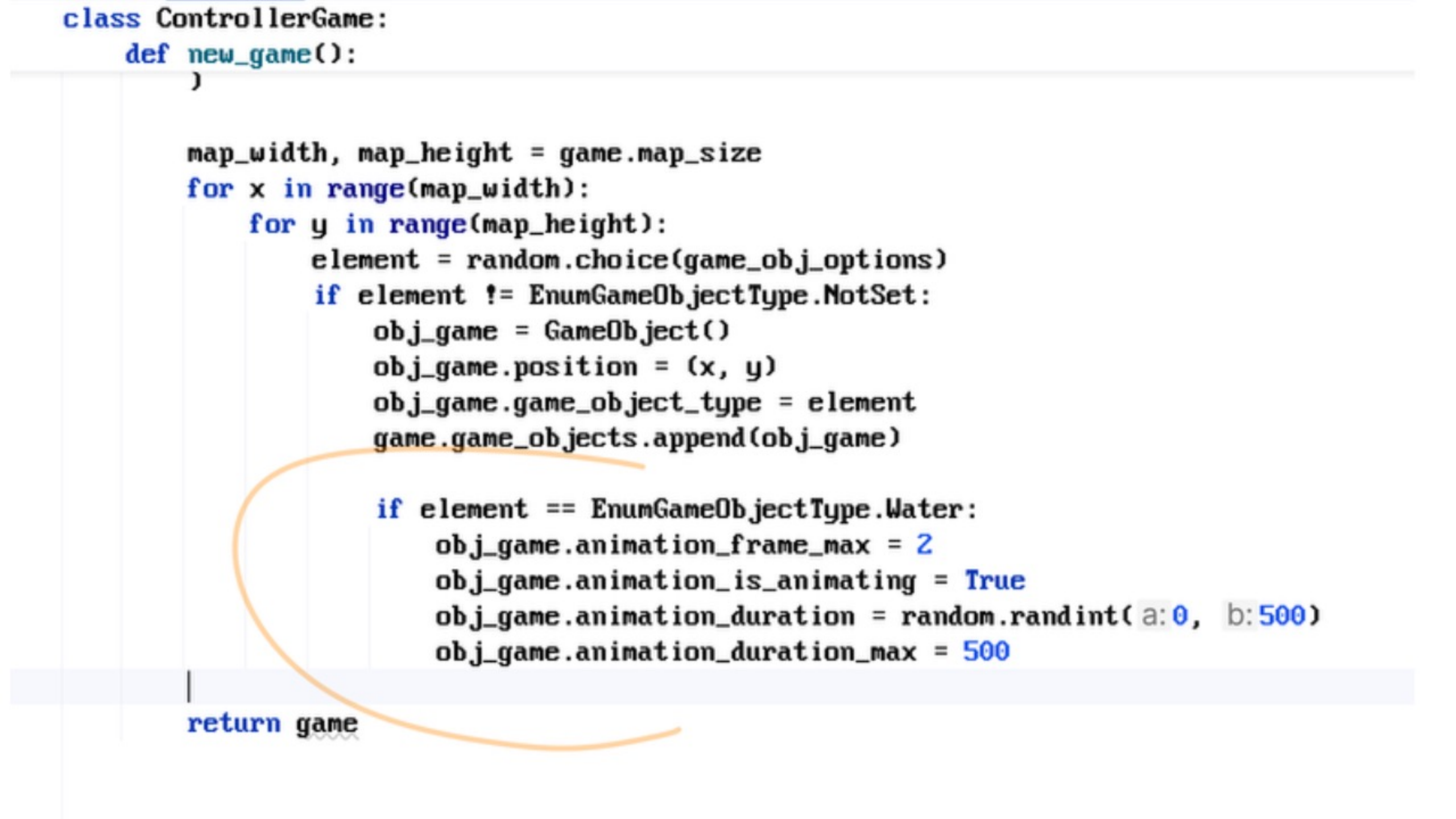


```
class ControllerGame:

    @staticmethod
    def new_game():
        game = Game()
        game.game_objects = []

        game_obj_options = (
                [EnumGameObjectType.Player] +
                [EnumGameObjectType.NotSet] * 20 +
                [EnumGameObjectType.Water] * 5 +
                [EnumGameObjectType.Forest] * 2 +
                [EnumGameObjectType.Steel] * 2
        )

        map_width, map_height = game.map_size
        for x in range(map_width):
            for y in range(map_height):
                element = random.choice(game_obj_options)
```

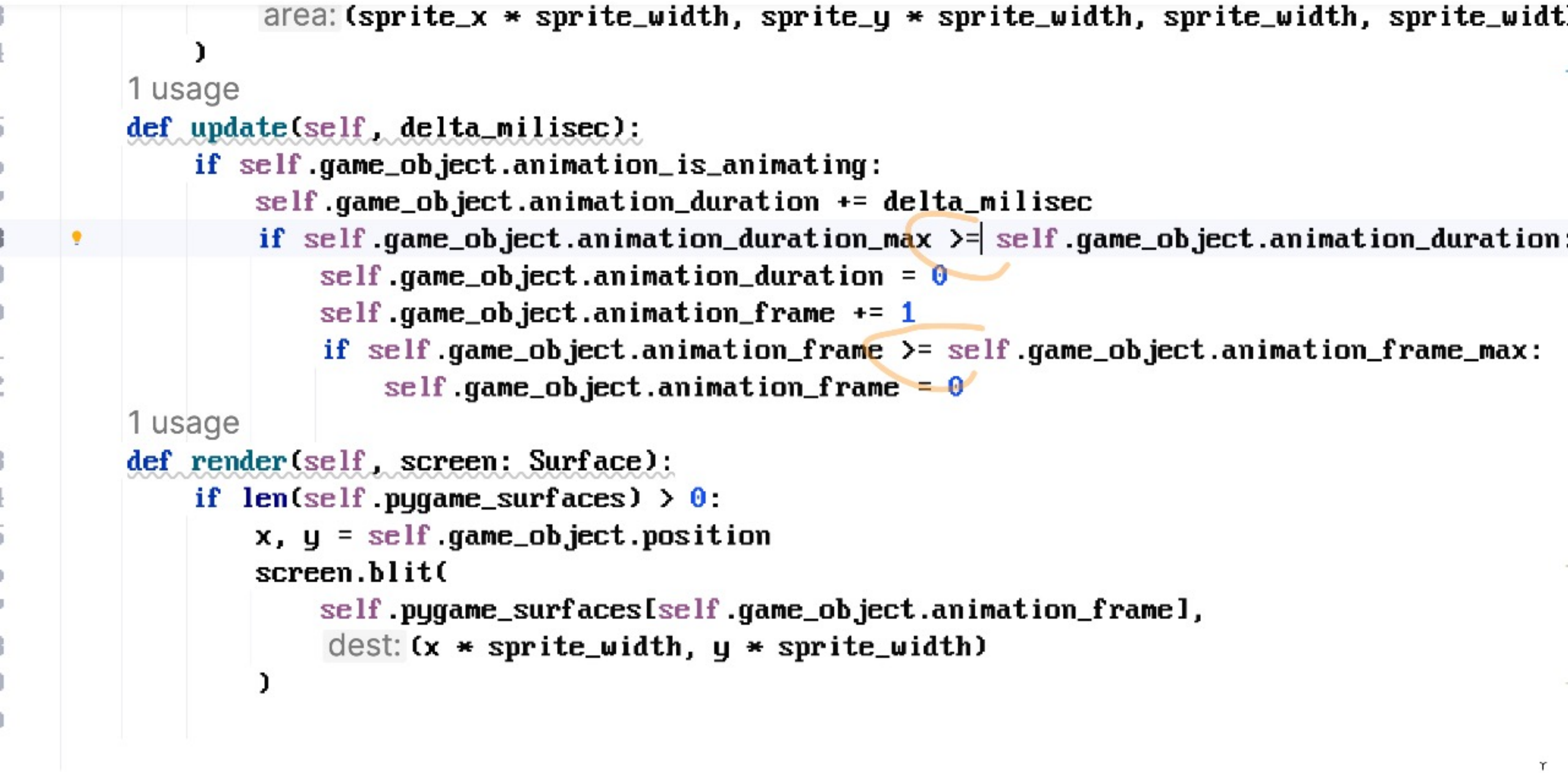


```
16  class ComponentGameObject:
45      def update(self, delta_milisec):
50                  self.game_object.animation_frame += 1
51                  if self.game_object.animation_frame > self.game_object.animation_frame_max:
52                      self.game_object.animation_frame = 0

53      def render(self, screen: Surface):
54          if len(self.pygame_surfaces) > 0:
55              x, y = self.game_object.position
56              screen.blit(
57                  self.pygame_surfaces[self.game_object.animation_frame],
58                  dest: (x * sprite_width, y * sprite_width)
59              )
60
```

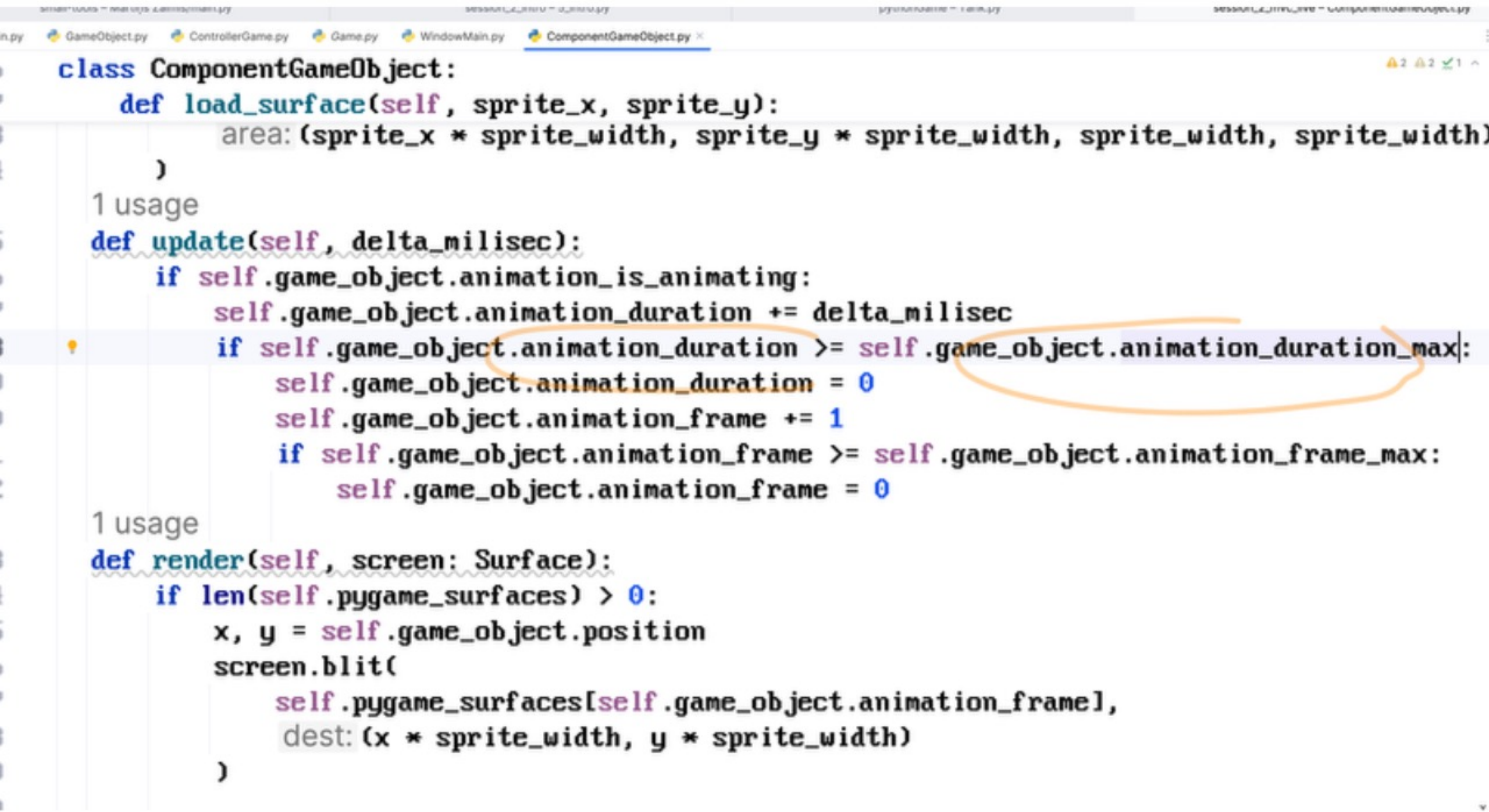
```
File "/Volumes/Storage/course_design_patterns_2024_q4/session_2_mvc_live/views/windows/WindowMain.py", line 34, in show
  self.draw()
File "/Volumes/Storage/course_design_patterns_2024_q4/session_2_mvc_live/views/windows/WindowMain.py", line 71, in draw
  game_component.render(self.screen)
File "/Volumes/Storage/course_design_patterns_2024_q4/session_2_mvc_live/views/components/ComponentGameObject.py", line 56, in render
```



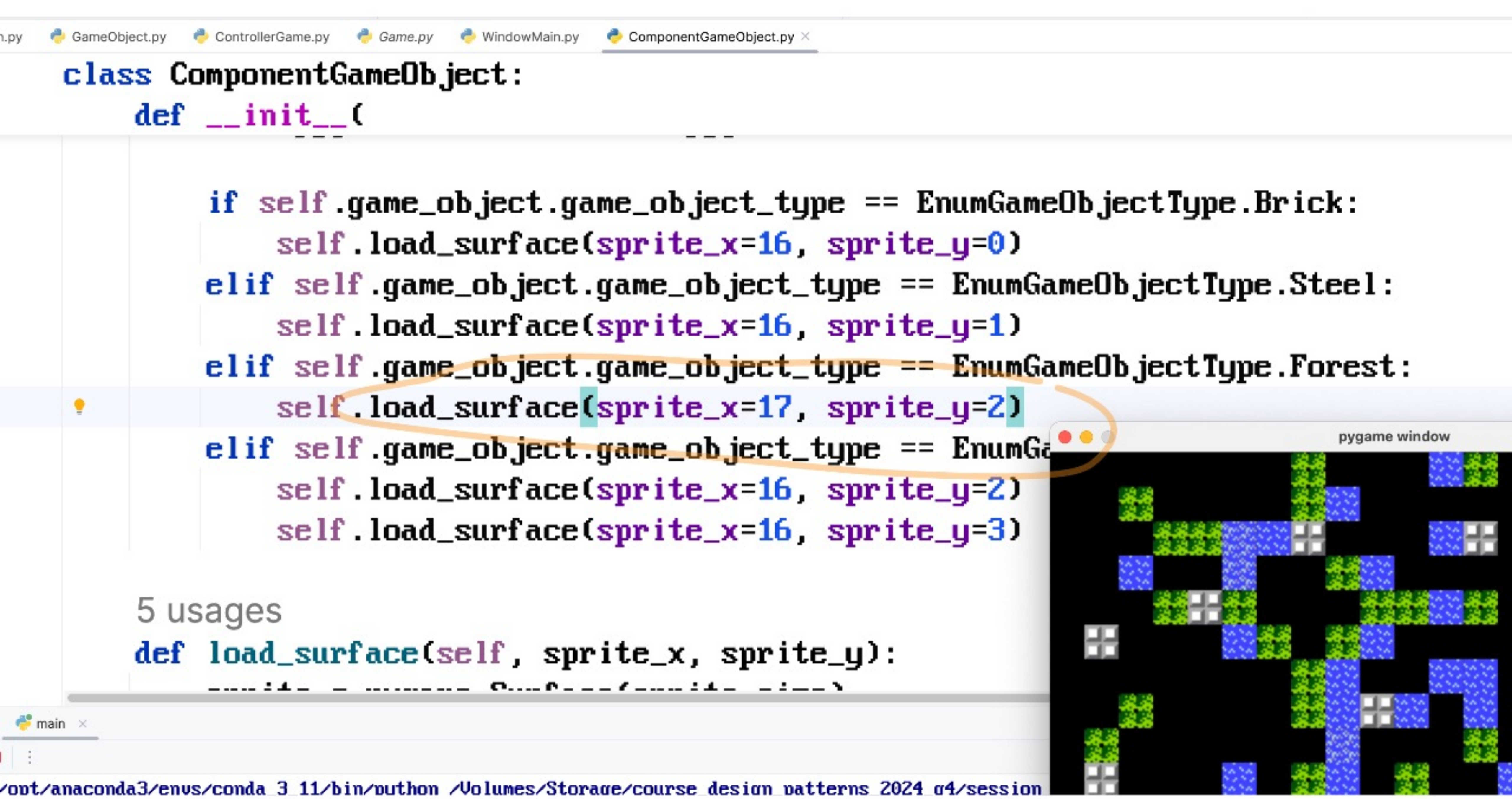
```
class ControllerGame:
    def new_game():
        )

        map_width, map_height = game.map_size
        for x in range(map_width):
            for y in range(map_height):
                element = random.choice(game_obj_options)
                if element != EnumGameObjectType.NotSet:
                    obj_game = GameObject()
                    obj_game.position = (x, y)
                    obj_game.game_object_type = element
                    game.game_objects.append(obj_game)

                    if element == EnumGameObjectType.Water:
                        obj_game.animation_frame_max = 2
                        obj_game.animation_is_animating = True
                        obj_game.animation_duration = random.randint( a:0,  b:500)
                        obj_game.animation_duration_max = 500

        return game
```



```
            area: (sprite_x * sprite_width, sprite_y * sprite_width, sprite_width, sprite_widt
        )
    1 usage
    def update(self, delta_milisec):
        if self.game_object.animation_is_animating:
            self.game_object.animation_duration += delta_milisec
            if self.game_object.animation_duration_max >= self.game_object.animation_duration
                self.game_object.animation_duration = 0
                self.game_object.animation_frame += 1
                if self.game_object.animation_frame >= self.game_object.animation_frame_max:
                    self.game_object.animation_frame = 0
    1 usage
    def render(self, screen: Surface):
        if len(self.pygame_surfaces) > 0:
            x, y = self.game_object.position
            screen.blit(
                self.pygame_surfaces[self.game_object.animation_frame],
                dest: (x * sprite_width, y * sprite_width)
            )
```

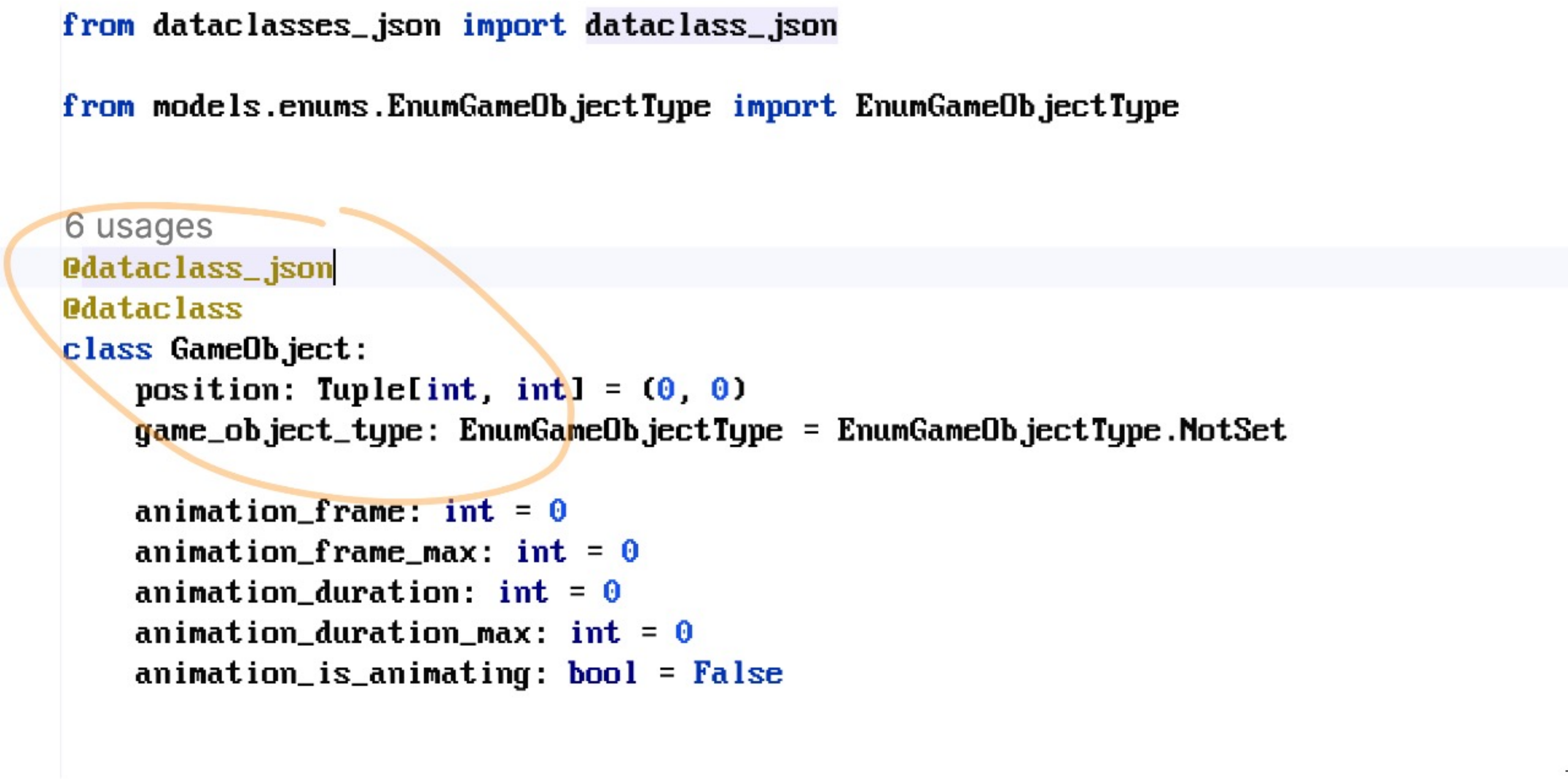


```
class ComponentGameObject:
    def load_surface(self, sprite_x, sprite_y):
            area: (sprite_x * sprite_width, sprite_y * sprite_width, sprite_width, sprite_width)
        )
    1 usage
    def update(self, delta_milisec):
        if self.game_object.animation_is_animating:
            self.game_object.animation_duration += delta_milisec
            if self.game_object.animation_duration >= self.game_object.animation_duration_max:
                self.game_object.animation_duration = 0
                self.game_object.animation_frame += 1
                if self.game_object.animation_frame >= self.game_object.animation_frame_max:
                    self.game_object.animation_frame = 0
    1 usage
    def render(self, screen: Surface):
        if len(self.pygame_surfaces) > 0:
            x, y = self.game_object.position
            screen.blit(
                self.pygame_surfaces[self.game_object.animation_frame],
                dest: (x * sprite_width, y * sprite_width)
            )
```



```
class ComponentGameObject:
    def __init__(

        if self.game_object.game_object_type == EnumGameObjectType.Brick:
            self.load_surface(sprite_x=16, sprite_y=0)
        elif self.game_object.game_object_type == EnumGameObjectType.Steel:
            self.load_surface(sprite_x=16, sprite_y=1)
        elif self.game_object.game_object_type == EnumGameObjectType.Forest:
            self.load_surface(sprite_x=17, sprite_y=2)
        elif self.game_object.game_object_type == EnumGa
            self.load_surface(sprite_x=16, sprite_y=2)
            self.load_surface(sprite_x=16, sprite_y=3)

    5 usages
    def load_surface(self, sprite_x, sprite_y):
        sprite = pygame.Surface(sprite_size)
```

```
/opt/anaconda3/envs/conda_3_11/bin/python /Volumes/Storage/course_design_patterns_2024_q4/session
```



```python
from dataclasses_json import dataclass_json

from models.enums.EnumGameObjectType import EnumGameObjectType


6 usages
@dataclass_json
@dataclass
class GameObject:
    position: Tuple[int, int] = (0, 0)
    game_object_type: EnumGameObjectType = EnumGameObjectType.NotSet

    animation_frame: int = 0
    animation_frame_max: int = 0
    animation_duration: int = 0
    animation_duration_max: int = 0
    animation_is_animating: bool = False
```