Kursinio darbo ataskaita

Atliko: Eimantas Bimba EDIF-24/2

Aprašo pradžia

Mano kursinio darbo programos tikslas buvo sukurti žaidimą. Jame tu turi išgyventi penkias minutes pries begalybe einančiu į tave priešų. Priešai meta sielas kurias galima panaudoti, kad tapti stipresniu. Kad paleisti programą reikia atidaryti main.exe. Tada reikia paspausti start, kad pradėti išgyvenimą. Žaidėja galima judinti tik su rodyklėmis.

Kursinio darbo reikalavimų įvykdymas

Github linkas: https://github.com/Mreimantas/University_Project

Štai keturi pagrindiniai OOP (objektinio programavimo) principai kuriuos aš panaudojau:

1. **Polymorphism**

Visi fabrikai turi create_enemy() metodą, tačiau naudoja ji skirtingai.

```
class EnemyFactory:
    def create_enemy(self, world_width, world_height, player):
        raise NotImplementedError

class ZombieFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.0, (0, 255, 255), world_width, world_height, player, health=150, damage=5)

class SkeletonFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.5, (255, 255, 255), world_width, world_height, player, health=100, damage=10)

class OrcFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.0, (0, 0, 255), world_width, world_height, player, health=200, damage=15)

class VampireFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.5, (255, 0, 255), world_width, world_height, player, health=120, damage=20)

class RandomFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        factory = random.choice([ZombieFactory(), SkeletonFactory(), OrcFactory(), VampireFactory()])
        return factory.create_enemy(world_width, world_height, player)
```

2. Abstraction:

EnemyFactory klasėje apibrėžia bendrą metodą create_enemy, bet jo neįgyvendina.

```
class EnemyFactory:
    def create_enemy(self, world_width, world_height, player):
        raise NotImplementedError

class ZombieFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.0, (0, 255, 255), world_width, world_height, player, health=150, damage=5)

class SkeletonFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.5, (255, 255, 255), world_width, world_height, player, health=100, damage=10)

class OrcFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.0, (0, 0, 255), world_width, world_height, player, health=200, damage=15)

class VampireFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.5, (255, 0, 255), world_width, world_height, player, health=120, damage=20)

class RandomFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        factory = random.choice([ZombieFactory(), SkeletonFactory(), OrcFactory(), VampireFactory()])
        return factory.create_enemy(world_width, world_height, player)
```

3. Inheritance

ZombieFactory, SkeletonFactory ir kt. klases paveldi iš EnemyFactory.


```
class EnemyFactory:
    def create_enemy(self, world_width, world_height, player):
        raise NotImplementedError

class ZombieFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.0, (0, 255, 255), world_width, world_height, player, health=150, damage=5)

class SkeletonFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.5, (255, 255, 255), world_width, world_height, player, health=100, damage=10)

class OrcFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.0, (0, 0, 255), world_width, world_height, player, health=200, damage=15)

class VampireFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.5, (255, 0, 255), world_width, world_height, player, health=120, damage=20)

class RandomFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        factory = random.choice([ZombieFactory(), SkeletonFactory(), OrcFactory(), VampireFactory()])
        return factory.create_enemy(world_width, world_height, player)
```

4. Encapsulation

GameSettings klaseje __game_state yra private

```python
class GameSettings:
    def __init__(self):
        self.__game_state = "menu"
        self.screen_width = 1280
        self.screen_height = 720
        self.is_fullscreen = False
        self.player_health = 100
        self.player_speed = 5
        self.player_armor = 0
        self.player_weapon = 0
        self.souls = 0
        self.upgrades = [0, 0, 0, 0]

    @property
    def game_state(self):
        return self.__game_state

    @game_state.setter
    def game_state(self, value):
        if value in ["menu", "game", "settings", "upgrades", "quit"]:
            self.__game_state = value
        else:
            raise ValueError(f"Invalid game state: {value}")
```

Buvo naudojamas Factory Method Design Pattern, kuris sukuria objektus Enemies kurie yra skirtingi.

```python
class EnemyFactory:
    def create_enemy(self, world_width, world_height, player):
        raise NotImplementedError

class ZombieFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.0, (0, 255, 255), world_width, world_height, player, health=150, damage=5)

class SkeletonFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.5, (255, 255, 255), world_width, world_height, player, health=100, damage=10)

class OrcFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.0, (0, 0, 255), world_width, world_height, player, health=200, damage=15)

class VampireFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.5, (255, 0, 255), world_width, world_height, player, health=120, damage=20)

class RandomFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        factory = random.choice([ZombieFactory(), SkeletonFactory(), OrcFactory(), VampireFactory()])
        return factory.create_enemy(world_width, world_height, player)
```

Mano kode naudojama Kompozicija ir agregacija.

Kompozicija:

Player, enemies ir souls yra priskirto Game klasei.

```
class Game:
    def __init__(self, settings, screen):
        self.settings = settings
        self.screen = screen
        self.clock = pygame.time.Clock()
        self.fps = 60
        self.internal_width = 1280
        self.internal_height = 720
        self.screen_width, self.screen_height = self.screen.get_size()
        self.world_width = self.internal_width * 2
        self.world_height = self.internal_height * 2
        self.world_surface = pygame.Surface((self.world_width, self.world_height))
        self.font = pygame.font.SysFont("Arial", 36)
        self.reset()

    def reset(self):
        self.settings.reset()
        self.settings.player = Player(self.settings, self.world_width, self.world_height)
        self.enemies = []
        self.souls = []
        self.last_spawn_time = 0
        self.spawn_delay = 2000
        self.elapsed_time = 0
```

Agregacija:

EnemySpawner klaseje sukuriame Enemy klasė kuri nėra priskirta prie tos klases

```
class EnemySpawner:
    @staticmethod
    def spawn_outside_view(speed, color, world_width, world_height, player, health=100, damage=10):
        margin = 600
        side = random.choice(['top', 'bottom', 'left', 'right'])

        if side == 'top':
            x = random.randint(0, world_width)
            y = random.randint(0, max(0, int(player.y - margin)))
        elif side == 'bottom':
            x = random.randint(0, world_width)
            y = random.randint(min(world_height, int(player.y + margin)), world_height)
        elif side == 'left':
            x = random.randint(0, max(0, int(player.x - margin)))
            y = random.randint(0, world_height)
        else:  # right
            x = random.randint(min(world_width, int(player.x + margin)), world_width)
            y = random.randint(0, world_height)

        return Enemy(x, y, speed, color, health, damage)
```

```python
class EnemyFactory:
    def create_enemy(self, world_width, world_height, player):
        raise NotImplementedError

class ZombieFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.0, (0, 255, 255), world_width, world_height, player, health=150, damage=5)

class SkeletonFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(1.5, (255, 255, 255), world_width, world_height, player, health=100, damage=10)

class OrcFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.0, (0, 0, 255), world_width, world_height, player, health=200, damage=15)

class VampireFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        return EnemySpawner.spawn_outside_view(2.5, (255, 0, 255), world_width, world_height, player, health=120, damage=20)

class RandomFactory(EnemyFactory):
    def create_enemy(self, world_width, world_height, player):
        factory = random.choice([ZombieFactory(), SkeletonFactory(), OrcFactory(), VampireFactory()])
        return factory.create_enemy(world_width, world_height, player)
```

Duomenys traukiami ir saugomi į settings.json faila.

Kodas buvo testuotas naudojant unittest framework. Testavimo kodas yra *tests* direktorijoje. Buvo ištestuota naudojant unittest bei savo sukurtais scenarijais.

```
----------------------------------------------------------------------
Ran 13 tests in 0.002s

OK
```

Kodas buvo parašytas atsižvelgiant į PEP8 stilių.

Rezultatai

- Programa gali rašyti ir skaityti json type file, kad gauti arba išsaugoti statistikas.
- Programoje yra menu kuriame yra 4 mygtukai.
- Paspaudžius start mygtuką prasideda žaidimas, kuriame reikia išgyventi penkias minutes.

Išvados

Sukuriau žaidimą kuriame galima judėti, už sielas tobulinti žaidėjo personažą ir pereiti žaidimą.

Dirbant prie šio kursinio darbo išmokau naudoti pygame biblioteka, kad kurti žaidimus.

Ši žaidimą dar galima tobulint: pavyzdžiui prailginti laika iki dešimt minučių arba ilgiau, pridėti daugiau priešų ir jų tipų, pridėti daugiau tobulinimų ir taip tolaiu.