Distribuirani računarski sistemi u EE
- Space Invaders Game-

Bogdan Kondić PR 58/2017
Cvijetin Mlađenović PR 55/2017
Nikola Mijonić PR 49/2017
Sonja Tomčić PR 53/2017

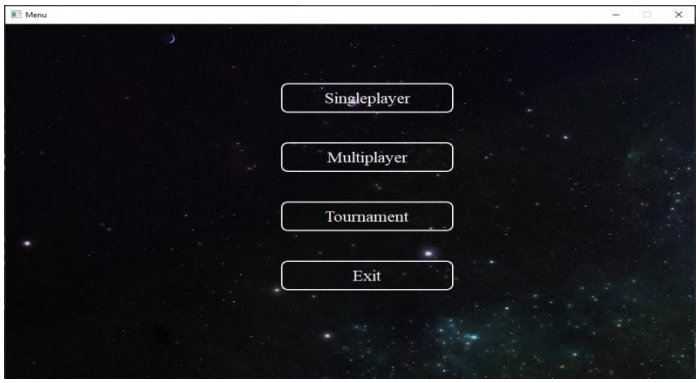
Space Invaders je arkadna igra koju je napravio japanski programer Tomohiro Nišikado. Izdata je 1978. godine. Igru je u početku distribuirala kompanija Taito iz Japana. Ova igra predstavlja jednu od prvih pucačkih video-igara. Cilj je poraziti vanzemaljce i osvojiti što više poena.

Projekat je pravljen po uzoru na Space Invaders igricu. Igrica je rekreirana u programskom jeziku Python, koristeći PyQt5 paket.

Opis igre

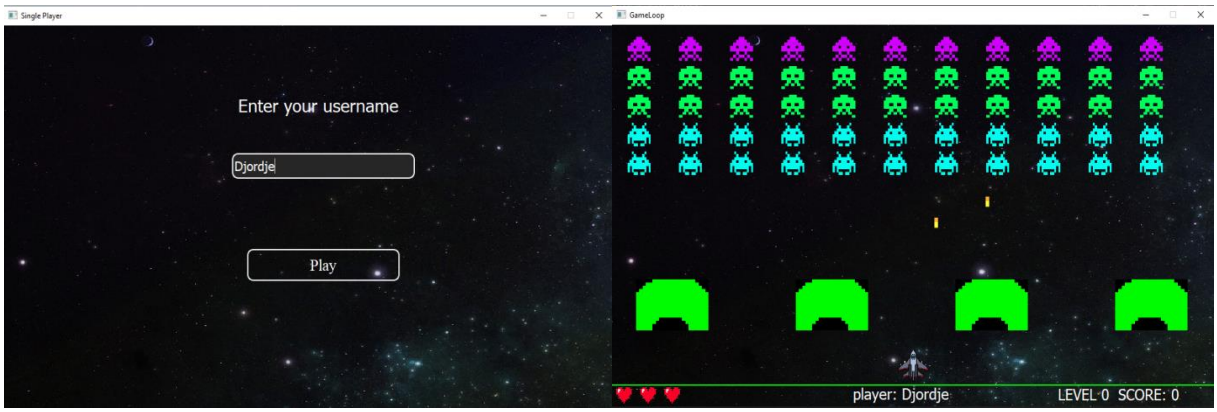
Kada se igra pokrene otvara se početni prozor gde korisnik ima mogućnost da izabere mode u kojem želi da igra. Korisnik da bira između 3 mode-a:

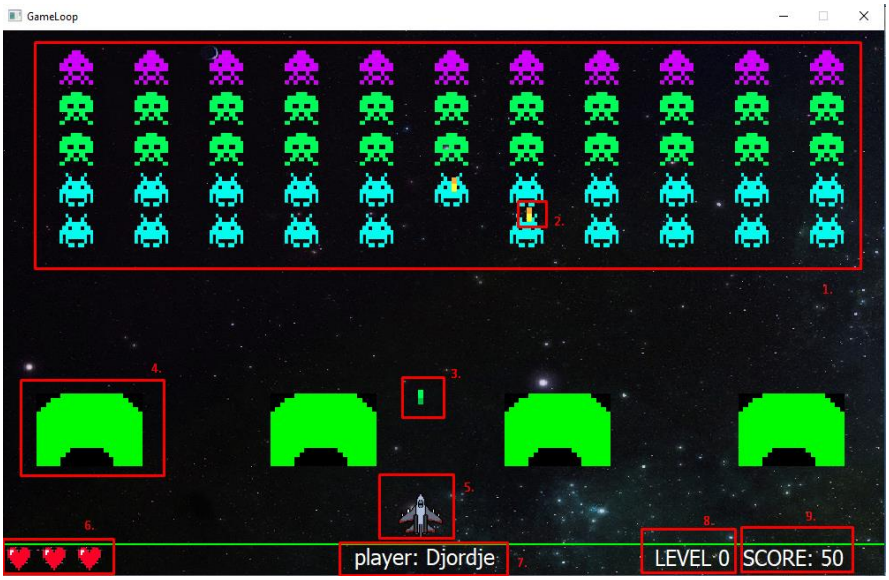
1. Singleplayer
2. Multiplayer
3. Tournament



Singleplayer:

Kada se izabere Singleplayer mode, korisniku se prikazuje polje za unos username-a. Nakon unosa, igrica se pokreće klikom na dugme „play“.



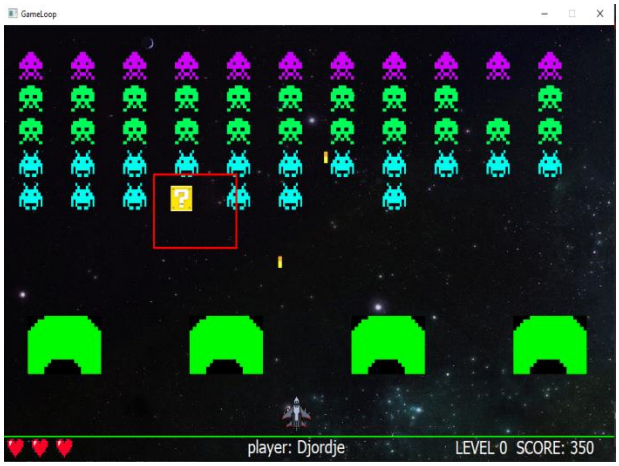


Elementi igre

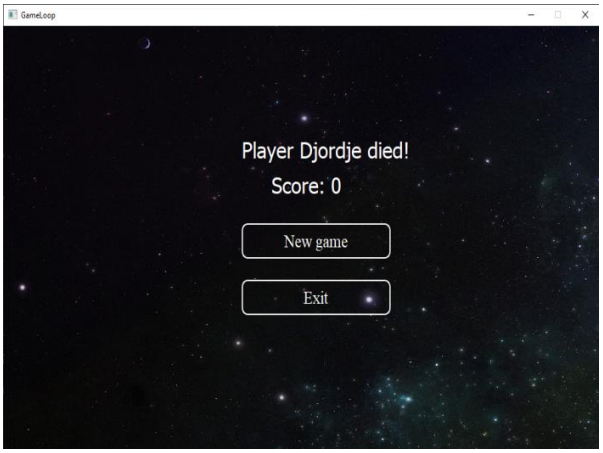
Elementi singleplayera:

1. vanzemaljci
2. metak vanzemaljca
3. metak igrača
4. štit
5. svemirski brod
6. životi
7. username igrača
8. nivo
9. rezultat

Korisnik pokreće svemirski brod na levu i desnu strelicu. Da bi igrač ubio vanzemaljca, potrebno je ispaliti metak pritiskom na dugme space. Vanzemaljci sve vreme ispaljuju metke. Ako metak vanzemaljca pogodi svemirski brod, igrač gubi život. U igrici postoje četiri štita. Njihov zadatak je da sačuvaju svemirski brod od metaka vanzemaljaca. Nakon što vanzemaljci 3 puta pogode štit, on gubi snagu. Potrebno je da vanzemaljci pogode štit deset puta kako bi on nestao. Nakon određenog vremena, na ekranu se pojavljuje upitnik kao faktor iznenađenja. Ukoliko igrač pogodi upitnik, dobija ili gubi život. Vanzemaljci se kreću s leva na desno i spuštaju se dole kako bi se ubijanje istih otežalo. Različite boje vanzemaljaca označavaju broj poena koji se osvaja nakon što se vanzemaljac ubije. Plavi vredi 50 bodova, zeleni 100, ljubičasti 200. Nakon što igrač ubije sve vanzemaljce, prelazi na sledeći nivo. Kako nivoi rastu, brzina vanzemaljaca i ispaljivanja njihovih metaka se povećava. Kada igrač izgubi živote, igrica je gotova i korisnik može da izabere da li želi da započne novu igru ili ne.



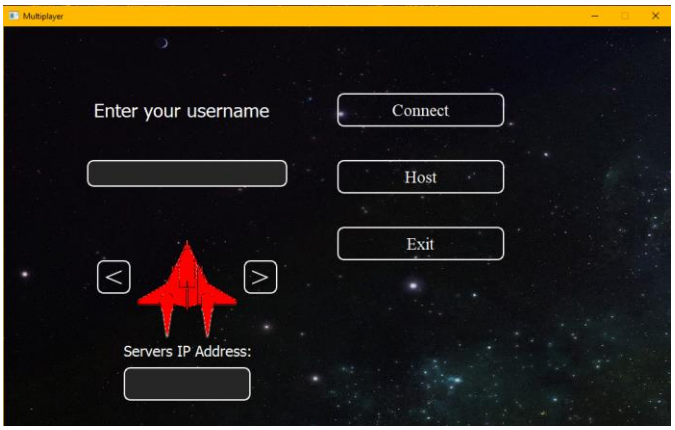
Faktor iznenađenja



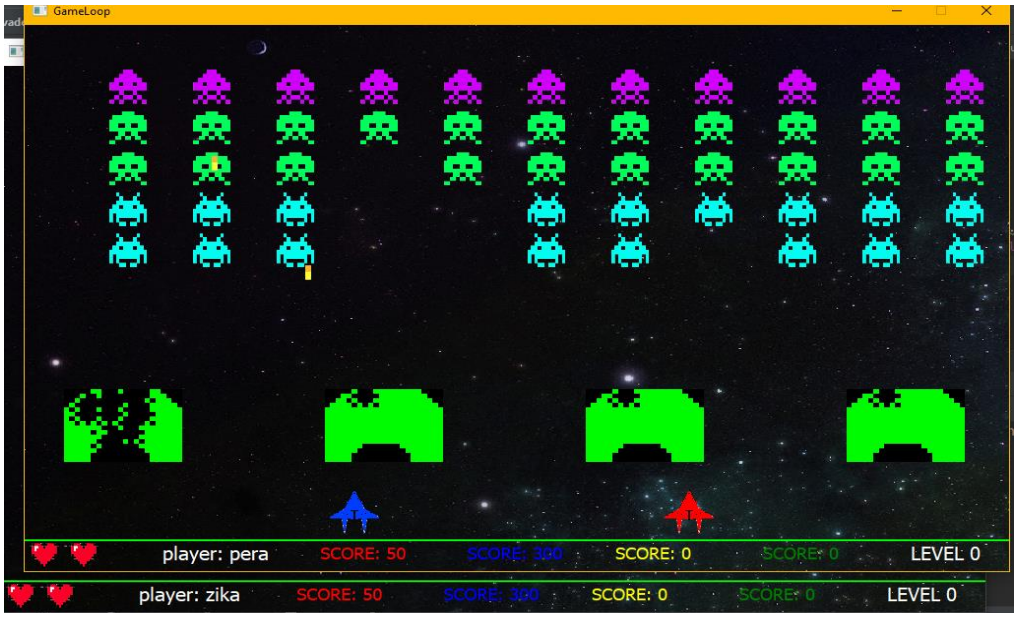
Završetak igre

Multiplayer:

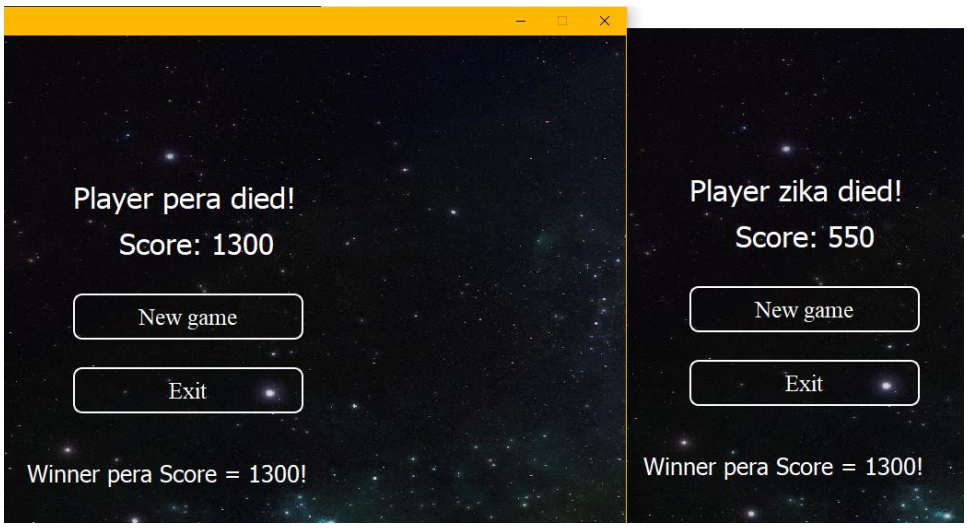
Logika same igrice je ostala nepromenjena u odnosu na singleplayer pa se iz tog razloga neće dodatno objašnjavati. Prikom izbora Multiplayer iz početnog menija korisnik dobija sledeći prikaz.



Korisnik ima mogućnost unosa korisničkog imena, IP adrese servera i izbor slike spaceshipa kako bi se razlikovali međusobno. Klikom na connect button client se konektuje na server. Server je prethodno podignut.



Kada se određen broj klijenata (konfigurabilan broj) konektuje na server igrica započinje. Slika iznad pokazuje ono što klijent vidi nakon konketovanja dovoljnog broja klijenata. Labele koje označavaju score svakog od konektovanih klijenata kako bi se mogla voditi evidencija o scoru protivnika. Pobednik je onaj igrač koji skupi najviše bodova.



Tournament:

Turnir mode je relizovan lokalno tako sto dva igrača igraju uporedno na ekranu jedan pored drugog krećući se na A i D tj. Na levu I desnu strelicu I pucajući na Space tj na Delete dugme kod drugog igraca. Pocetak turnira je relizovan tako sto se unese Korisnicko ime za 8 igrača I potom pritisne dugme Play, Potom se izgenerišu random 4 para za prvu rundu turnira. Prvi par kada odigra svoj meč, belezi se pobednik u listu pobednika odakle se kreće u novo ukraštanje nakon sledeće runde.

U drugoj rundi se ukrstaju naredna dva para I pobednici respektivnih parova odlaze u finalni mec gde se odredjuje pobednik celokupnog turnira.

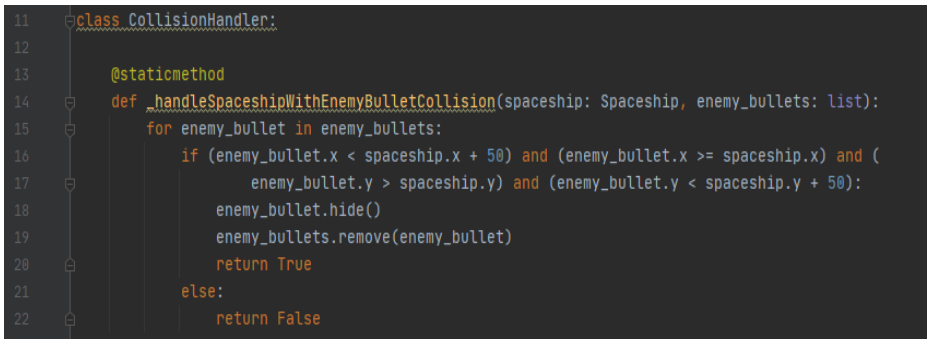
Implementacija

Singleplayer:

Singleplayer mode je kreiran u fajlu „SinglePlayer.py“. U njemu se kreira glavni prozor aplikacije – objekat GameLoop, koji nasleđuje Qwidget. Na samom početku GameLoopa se inicjalizuje GUI, kao i handleri koji se koriste tokom igre.

Handleri:

1. **CollisionHandler** obezbđuje nekoliko statičkih metoda koje su zadužene za hendlovanje kolizije u igrici.



```
class CollisionHandler:

    @staticmethod
    def _handleSpaceshipWithEnemyBulletCollision(spaceship: Spaceship, enemy_bullets: list):
        for enemy_bullet in enemy_bullets:
            if (enemy_bullet.x < spaceship.x + 50) and (enemy_bullet.x >= spaceship.x) and (
                    enemy_bullet.y > spaceship.y) and (enemy_bullet.y < spaceship.y + 50):
                enemy_bullet.hide()
                enemy_bullets.remove(enemy_bullet)
                return True
            else:
                return False
```

2. **KeyHandler** je zadužen za handlovanje pomeranja svemirskog broda i za pucanje

```python
class KeyHandler:
    def __init__(self):
        super().__init__()
        print('1')

    @staticmethod
    def handle_key(screen, event, spaceship):
        key = event.key()

        if key == Qt.Key_Left:
            if spaceship.x == 0:
                pass
            else:
                spaceship.move(-5, 0)
        elif key == Qt.Key_Right:
            if spaceship.x == screen.width() - 50:
                pass
            else:
                spaceship.move(5, 0)
        elif key == Qt.Key_Space:
            bullet = BulletFactory.create_object(screen, "bullet_id", spaceship.x + 22, spaceship.y + 16 - 40,
                                                 "../Sources/Images/Player/player_laser.png", 6, 16, "player_id")
```

Na početku GameLoopa se poziva metoda „create_new_level“ koja je zadužena za inicijalizaciju svih objekata i tredova se koriste u igri. Nakon prelaska na novi nivo, metoda se ponovo poziva, stari objekti se brišu i inicijalizuju se novi.

Za implementaciju tredova napravljeni su workeri koji nasleđuju Qthread. U metodi „create_new_level“ se inicijalizuju svi workeri i konektuju na metode koje će biti pozvane kada worker emituje signal.

```python
self.keyNotifierWorker = KeyNotifierWorker()
self.keyNotifierWorker.start()
self.keyNotifierWorker.key_signal.connect(self.moveSpaceship)
self.keyNotifierWorker.finished_signal.connect(self.finishedWithMoveSpaceshipThread)

self.enemiesWorker = EnemiesWorkerThread(self.enemies, self.shields, self.level_num)
self.enemiesWorker.start()
self.enemiesWorker.finished_enemies_moving_signal.connect(self.finishedWithEnemiesWorker)
self.enemiesWorker.update_enemies_position.connect(self.moveEnemies)
self.enemiesWorker.new_level.connect(self.create_new_level)

# testiranje

self.bulletEnemyWorker = BulletEnemyWorkerThread(self, self.enemies, self.enemy_bullets, self.level_num)
self.bulletEnemyWorker.start()
self.bulletEnemyWorker.finish_enemy_shooting.connect(self.finishedWithEnemyBulletWorker)
self.bulletEnemyWorker.update_enemy_bullet.connect(self.updateEnemiesBullet)
```

Workeri: BulletEnemyWorker, BulletWorker, DeusExWorker, EnemiesWorker, KeyNotifier, SpaceshipWorker.

```python
class BulletWorkerThread(QThread):
    update_bullet = pyqtSignal(Bullet)
    user_dead = pyqtSignal()
    update_lifes_number = pyqtSignal(int)

    def __init__(self, screen: QWidget, spaceship: Spaceship, enemies, score_label, score_list, box: Box, hearts: list):
        super().__init__()
        self.collisionHandler = CollisionHandler()
        self.enemies = enemies
        self.spaceship = spaceship
        self.screen = screen
        self.box = box
        self.hearts = hearts

        self.bullet = BulletFactory.create_object(self.screen, "bullet_id", self.spaceship.x + 22,
                                                  self.spaceship.y + 16 - 40,
                                                  "../Sources/Images/Player/player_laser.png", 6, 16, "player_id")

        self.score_list = score_list
        self.score_label = score_label

        self.allEnemies = []
```

```python
    def run(self):

        y = self.bullet.y
        while True:
            if y >= - 15:
                if self.collisionHandler._handleSpaceshipBulletWithEnemyCollision(self.bullet, self.enemies, self.allEnemies, s
                    break

                if self.collisionHandler._handleSpaceshipBulletWithBoxCollision(self.bullet, self.box, self.screen):
                    #if self.collisionHandler.updateNumberOfLivesLuckyFactor(self.screen, self.hearts, self.box.luckyFactor):
                        #self.user_dead.emit()
                    self.update_lifes_number.emit(self.box.luckyFactor)

                    break

                time.sleep(0.02)
                self.update_bullet.emit(self.bullet)
                y -= 15

            else:
                del self.bullet
```

Bullet Worker

Slučajna sila je implementirana upotrebom procesa i pipe-a.

```python
#deus ex


self.process = DeusExProcess(pipe=self.ex_pipe, max_arg=101)
self.process.start()

self.DeusExWorker = DeusExWorkerThread(self, self.in_pipe)
self.DeusExWorker.start()
#self.DeusExWorker.finish_enemy_shooting.connect(self.finishedWithEnemyBulletWorker)
self.DeusExWorker.show_random_force.connect(self.show_random_force)

self.level_label.setText("LEVEL " + str(level_num))
```

```python
class DeusExProcess(Process):

    def __init__(self, pipe: Pipe, max_arg: int):
        super().__init__(target=self.__count__, args=[pipe])
        self.max = max_arg

    def __count__(self, pipe: Pipe):
        pipe.recv()

        while True:
            time.sleep(15)
            x = random.randrange(200, 900)
            y = random.randrange(50, 350)
            luckyFactor = random.choice(range(-1, 2, 2))
            pipe.send([x, y, luckyFactor])

            '''
            for i in range(self.max):
                pipe.send(i)
                print("send {0}".format(i))
                time.sleep(0.05)
            '''
        pipe.send('end')
```

Za kreiranje objekata koriste se: BulletFactory, EnemyFactory, HeartFactory, ShieldFactory, SpaceshipFactory.

```python
class BulletFactory:

    @staticmethod
    def create_object(screen: QWidget, bullet_id: str,  x: int, y: int, img: str, width: int, height: int, player_id: str):
        return Bullet(screen=screen, bullet_id=bullet_id, x=x, y=y, img=img, width=width, height=height, player_id=player_id)
```

Bullet Factory

Multiplayer:

Multiplayer se sastoji od dve funkcionalne komponente. Client I Server. Server je zadužen za prihvatanje klijentske konkecije I za obradu zahteva klijenata. Client iscrtava GUI I u zavisnosti od zahteva koji stigne sa servera izvršava određene akcije. Sledi prikaz detalja implementacije.

```python
import socket
import pickle
import struct


class SocketManager:
    def __init__(self, socket):
        self.socket = socket

    def send_message(self, message):
        message_to_send = message
        # print(message_to_send)

        message_len = len(message_to_send.encode())

        self.send_all_bytes(self.socket, message_len, message_to_send)

    def recv_message(self):
        received_message = self.socket.recv(1024).decode()
        lista = received_message.split('|')
        flag = lista[0]
        message = lista[1]
        spaceship_image = lista[2]

        return flag, message, spaceship_image

    def send_all_bytes(self, socket, message_len, message_to_send):
        encoded_data = message_to_send.encode()
        bytes_sent = 0
        while bytes_sent < message_len:
            try:
                bytes_sent += socket.send(encoded_data)
            except Exception as e:
                print("Error while sending data : ", str(e))
```

SocketManager je klasa koja implementira kompletnu logiku slanja podataka kroz mrežu. Obezbeđeno je sigurno slanje podataka kroz mrežu. Podaci se salju sve dok se svaki byte ne pošalje. Takođe, u ovoj klasi nalazi se format poruke koji se koristi pri slanju podatak kroz mrežu.

```python
if __name__ == '__main__':
    listenSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listenSocket.bind((socket.gethostname(), 5000))
    listenSocket.listen(4)


    print(f"Server is open and waiting for clients at address: {socket.gethostbyname(socket.gethostname())}")
    Thread(target=startGame, args=()).start()

    while True:
        acceptedSocket, address = listenSocket.accept()
        print(f"New clieen has been accepted with address {address}")
        t = Thread(target=proccesingClient, args=[acceptedSocket])
        t.start()
```

```python
def proccesingClient(s: socket):
    socket_manager = SocketManager(s)
    global cnt
    global clients
    flag, message, spaceship_image = socket_manager.recv_message()
    username = message
    myUsername = message
    print("-" * 69)
    print(f"{flag}, {username}, {spaceship_image}")
    print("-" * 69)


    for client in clients:
        client[0].send_message(f"NEW CLIENT|{username}|{spaceship_image}")
        print(f"Klijent: {client[1]} | Username: {username}")
        time.sleep(0.2)
        socket_manager.send_message(f"NEW CLIENT|{client[1]}|{client[2]}")

while True:
    try:
        flag, username, spaceship_image = socket_manager.recv_message()
        print(f"{flag} { username}")

        if flag == "MOVE LEFT":
            for client in clients:
                if client[1] != username:
                    client[0].send_message(f"MOVE LEFT|{username}|")

        elif flag == "MOVE RIGHT":
            for client in clients:
                if client[1] != username:
                    client[0].send_message(f"MOVE RIGHT|{username}|")

        elif flag == "SHOOT":
            bullet_id = spaceship_image
            for client in clients:
                if client[1] != username:
                    client[0].send_message(f"SHOOT|{username}|{bullet_id}")
```

Prikaz detalja implementacije servera

Folder workers u okviru Network foldera sadrži Workere (Threadove) koji su objašnjeni u Singlepayer-u samo što je sada imaju instancu SocketManager-a koji je korišćen za mrežnu komunikaciju. Takođe, dodat je listenWorker koji se koristi za osluškivanje poruka sa servera I emitovanje signala na koji se konektuju odgovarajuće funkcije.

```
def __init__(self, s, all_spaceships : list, queue: Queue):
    super().__init__()
    self.connectSocket = s
    self.socket_manager = SocketManager(s)
    self.all_spaceships = all_spaceships
    self.queue = queue

def run(self):
    while True:
        try:
            flag, username, spaceship_image = self.socket_manager.recv_message()
            print(f"Primio sam od SERVERA: {flag}, {username}, {spaceship_image}")

            if flag == "NEW CLIENT":
                self.new_spaceship.emit(username, spaceship_image)
            elif flag == "START GAME":
                self.start_game.emit()
            elif flag == "MOVE LEFT":
                self.move_spaceship.emit(flag, username)
            elif flag == "MOVE RIGHT":
                self.move_spaceship.emit(flag, username)
            elif flag == "SHOOT":
                self.spaceship_shoot.emit(username, spaceship_image)
```

ListenThreadWorker