Gra multiplayer typu shooter

Miłosz Szumiec Maria Kowalczyk

Cel projektu

- zapoznanie się z modułami pozwalającymi na zdalną synchronizację wielu aplikacji
- zapoznanie się z modułem pygame pozwalającym na tworzenie responsywnych aplikacji graficznych
- zmierzenie się z wyzwaniami jakie stawia synchronizacja dwóch gier bazujących na bardzo dużej liczbie atrybutów

Przedmiot demonstracji

- przybliżenie działania modułu socket i podstawowego zastosowania modułu pickle
- przedstawienie ogólnego zarysu działania modułu pygame na przykładzie wcześniej stworzonej gry

Zakres demonstracji

- zdobytą wiedzę można użyć w przyszłych projektach, wymagających zastosowania moduły pygame, bądź połączenia ze sobą większej liczby aplikacji
- ograniczeniem jest fakt, że musimy mieć bezpośrednie połączenie po ip (serwer - klient), co często uniemożliwia protokół NAT stosowany w ruterach
  - można to obejść port-forwardingiem (nie do końca bezpieczne rozwiązanie) lub zastosowaniem wirtualnych sieci lokalnych

Teoria - Połączenie - Serwer

Pierwsze kroki w tworzeniu serwera...

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Rysunek 1: Tworzenie socketa z odpowiednimi argumentami

- AF_INET -> IPv4
- SOCK_STREAM -> TCP
- SOCK_DGRAM -> UDP

```
try:
    s.bind((server, port))
except socket.error as e:
    str(e)

s.listen()
print("Server Started")
```

Rysunek 2: Przypisanie do socketa odpowiedniego adresu ip i portu + ustawienie serwera na tryb nasłuchiwania

Teoria - Połączenie - Serwer

```
while True:
    conn, addr = s.accept()
    print("Connected to:", addr)

    start_new_thread(threaded_client, (conn, playerindexes.pop(0)))
```

Rysunek 3: Nieskończona pętla akceptująca nowe połączenia

```
g = Gameplay()
playerindexes = []
base = []
for i in range(g.max_players):
    playerindexes.append(i)
    base.append([g.allPlayersToPickle[i], [], []])
```

Rysunek 4: Stworzenie instancji gry na serwerze w celu rozesłania początkowych danych do graczy

Teoria - Połaczenie - Serwer

```
def threaded_client(conn, player):
    conn.sendall(pickle.dumps([base, player, g.boxes]))
```

Rysunek 5: Przesłanie początkowych danych do nowego klienta

```
def threaded_client(conn, player):
    conn.sendall(str.encode(strPlayers + ',' + str(player)))
```

Rysunek 6: Przykład przesyłania danych BEZ modułu pickle

Teoria - Połączenie - Serwer

```
playerData = pickle.loads(conn.recv(8192))
```

Rysunek 7: Przykład odbierania danych poprzez moduł pickle

```
info = conn.recv(2048).decode()
```

Rysunek 8: Przykład odbierania danych BEZ modułu pickle

Teoria - Połączenie - Serwer

```
def threaded_client(conn, player):
    conn.sendall(pickle.dumps([base, player, g.boxes]))
    while True:
        try:
            playerData = pickle.loads(conn.recv(8192))  # pobieramy tylko playera
            base[player] = playerData

            if not playerData:
                print("Disconnected")
                break
            else:
                reply = []

                for user in base:
                    if base.index(user) == player:
                        reply.append('')
                    else:
                        reply.append(user)

            conn.sendall(pickle.dumps(reply))  # wysyłamy wszystkich graczy oprócz playera
        except:
            break

    print("Lost connection")
    conn.close()
    base[player][0]['weapon'] = 'hand'
    playerindexes.insert(0, player)
```

Rysunek 9: Przykład funkcji obsługującej pojedynczego klienta

Teoria - Połaczenie - Klient

```
class Network:
    def __init__(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server = "localhost"
        self.port = 5555
        self.addr = (self.server, self.port)
        self.p = self.connect()

    def getP(self):
        return self.p

    def connect(self):
        try:
            self.client.connect(self.addr)
            return pickle.loads(self.client.recv(8192))
        except:
            pass

    def send(self, data):
        try:
            self.client.sendall(pickle.dumps(data))
            return pickle.loads(self.client.recv(8192))
        except socket.error as e:
            print(e)
```

Rysunek 10: Klasa obsługująca połączenie ze strony klienta

Teoria - Połączenie - Klient

```
n = Network()
base = n.getP()
data = base[0]
player = base[1]
boxes = base[2]
self.gameplay.c_f = player  # ustawiamy id gracza
self.gameplay.current_focus = self.gameplay.players[player]
self.gameplay.sync_windows(data)
self.gameplay.generate_obst_from_data(boxes)
```

Rysunek 11: Sekwencja odbywająca się po dołączenia do serwera, polegająca na synchronizacji danych lokalnej gry z tą z serwera

Teoria - Połaczenie - Klient

```
while not stopped:
    p1 = [self.gameplay.allPlayersToPickle[player], self.gameplay.bulletsToPickle, self.gameplay.boxes]
    self.gameplay.boxes = []
    self.gameplay.bulletsToPickle = []
    p2 = n.send(p1)
    self.gameplay.update(p2)

    # Triggering events: using mouse and keyboard
    for event in pygame.event.get():
        self.gameplay.on_press(event)

        # Exiting a game either by closing event or by game option
        if event.type == pygame.QUIT or self.gameplay.has_quited():
            # self.gameplay.quit()
            stopped = True

    timedelta = 0.4 * clock.tick(60)
    self.gameplay.play(timedelta)
    self.gameplay.pickle_player(self.gameplay.c_f)
    self.gameplay.draw()

    # refreshes screen
    pygame.display.flip()
```

Rysunek 12: Nieskończona pętla odpowiadająca zarówno za działanie samej gry, jak i wymianię informacji pomiędzy klientem a serwerem

Podstawowy szkielet aplikacji pygame

```python
import pygame

pygame.init()

screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("title")

running = True

while running:

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

        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                running = False

pygame.quit()
```

Rysunek 13: Najprostszy program w Pygame

Podstawowy szkielet aplikacji pygame

```
5   screen = pygame.display.set_mode((800, 600))
6   pygame.display.set_caption("title")
7   img = pygame.image.load('player&hand.png')
8   img_x = 400
9   img_y = 300
10  running = True
11
12  while running:
13      screen.blit(img, (img_x, img_y))
```

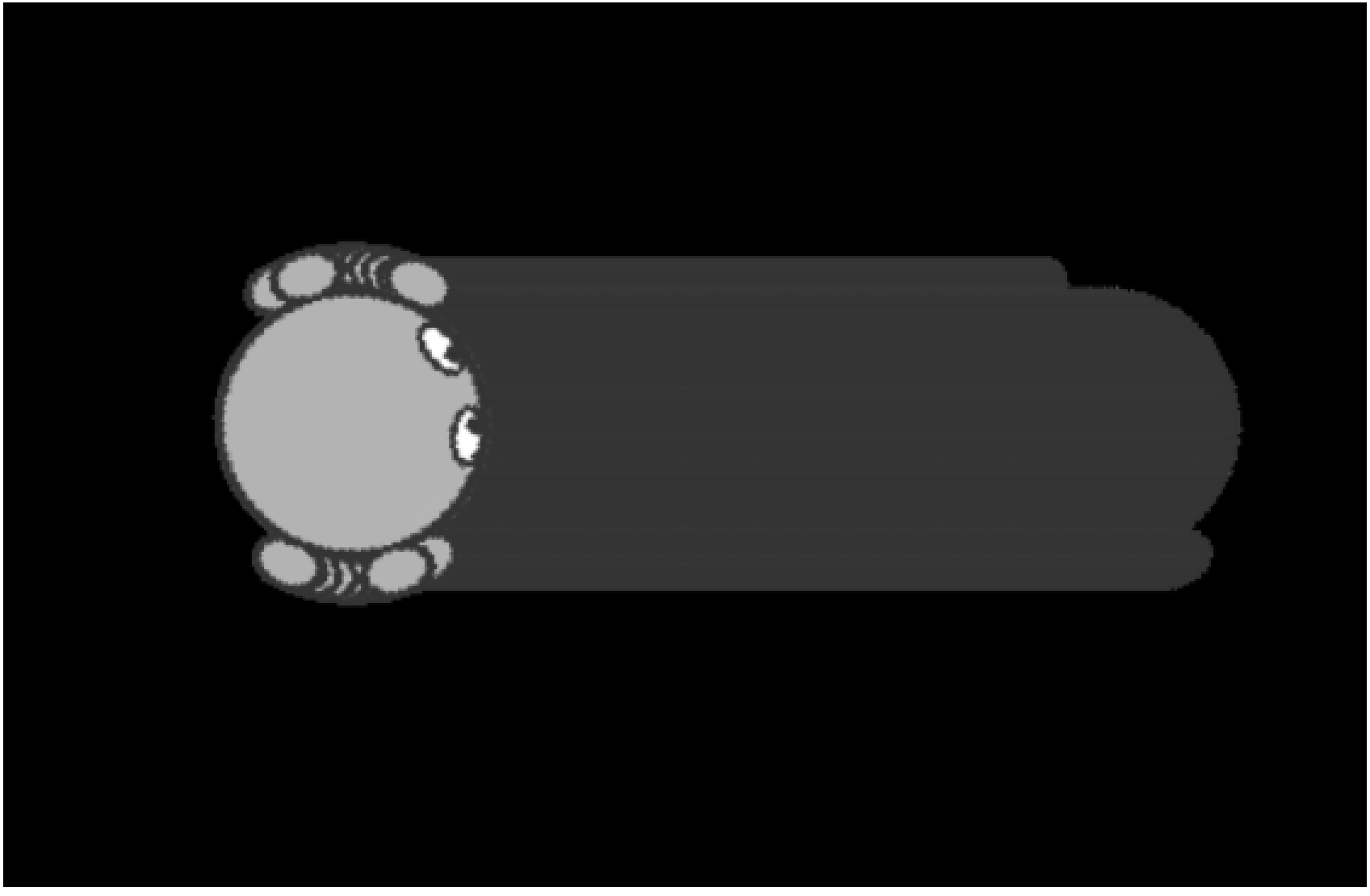
Rysunek 14: Tworzenie obiektu

Podstawowy szkielet aplikacji pygame

```
50      keys = pygame.key.get_pressed()
51
52      if keys[pygame.K_a]:
53          player1.x -= player1.speed
54      if keys[pygame.K_d]:
55          player1.x += player1.speed
56      if keys[pygame.K_w]:
57          player1.y -= player1.speed
58      if keys[pygame.K_s]:
59          player1.y += player1.speed
```

Rysunek 15: Mechanizm poruszania się postaci

Podstawowy szkielet aplikacji pygame



Rysunek 16: Dotychczasowy wynik działania naszego programu

Podstawowy szkielet aplikacji pygame

```
34      def redraw():
35          screen.fill((0, 0, 0))
36          player1.draw()
37          pygame.display.flip()
38          pygame.display.update()
```

Rysunek 17: Mechanizm odpowiadający za brak rysowania śladu podczas przesuwania obiektem

Śledzenie kursora myszki

```
10    rotated_img = img.copy()
11    rect = img.get_rect()
```

Rysunek 18: Tworzenie kopii naszego obrazu oraz uzyskanie z niego kwadratu

```
def rotate(self, angle):
    self.rotated_surface = pygame.transform.rotate(self.surface, angle)
    self.rect = self.rotated_surface.get_rect(center=screen.get_rect().center)

def draw(self):
    screen.blit(self.rotated_surface, self.rotated_surface.get_rect(center=(self.x, self.y)))
```

Rysunek 19: Obrót naszego obrazu o kąt ANGLE i rysowanie funkcją blit

Śledzenie kursora myszki

```
elif e.type == pygame.MOUSEMOTION:

    mouse_x, mouse_y = pygame.mouse.get_pos()
    angle = math.atan2(mouse_x - player1.x, mouse_y - player1.y)
    angle = math.degrees(angle) + 180
    player1.rotate(angle)
```

Rysunek 20: Kąt ANGLE

Podsumowanie

- Przedstawiliśmy jak w sposób podstawowy wdrożyć do programu bibliotekę socket i pickle, oraz jakie niosą ze sobą ograniczenia.
- Nakreśliliśmy możliwości, jakie niesie ze sobą moduł pygame, przy którym głównym ograniczeniem potencjalnego użytkownika jest wyobraźnia.
- Po wysłuchaniu naszej demonstarcji oraz przerobieniu ćwiczeń, będziecie w stanie stworzyć podstawowy szkielet gry typu shooter, który w przyszłości będziecie mogli rozszerzać.

Bibliografia

- https://docs.python.org/3/library/socket.html
- https://docs.python.org/2/library/socket.html
- https://docs.python.org/3/howto/sockets.html
- https://docs.python.org/2/library/thread.html
- https://docs.python.org/3/library/pickle.html
- https://stackoverflow.com/questions/34252273/what-is-the-difference-between-socket-send-and-socket-sendall
- https://techwithtim.net/tutorials/python-online-game-tutorial/connecting-multiple-clients/
- https://www.pygame.org/news
- "Game Programming" by Andy Harris 2007
- https://www.youtube.com/watch?v=PVY46hUp2EM

ZADANIA PRAKTYCZNE - TEORETYCZNE

1. Jaki argument odpowiada za połączenie TCP?
   - **a)** AF_INET
   - **b)** AF_INET6
   - **c)** SOCK_DGRAM
   - **d)** SOCK_STREAM
2. Chcąc wysłać obiekt x = [1, 2, 3] użyjemy komendy:
   - **a)** conn.sendall(str.encode(x))
   - **b)** conn.sendall(pickle.dumps(x))
3. Mając obiekt client = socket.socket(socket.AF_INET, socket.SOCK_STREAM), napisz, jaką komendą połączysz się z serwerem o adresie 192.168.2.1 na porcie 5525.
4. Do czego służy funkcja blit?
   - **a)** tworzy kopię grafiki i uzyskuje z niej kwadrat
   - **b)** służy do rysowania jednych obiektów Surface na drugich
   - **c)** obraca grafikę o podany kąt
   - **d)** wypełnia całe okno danym kolorem
5. Napisz jaki wynik zwraca funkcja atan2 i dlaczego dodajemy do niego wartość 180°,aby uzyskać prawidłowe śledzenie kursora myszy.

ZADANIA PRAKTYCZNE - PRACA NA KODZIE

1. **Pobierz do jednego foldera wszystkie zadania z części połączeniowej.**
   Otwórz plik ZADANKO_1.py i postępuj zgodnie z instrukcjami. Twoim zadaniem jest napisanie prostego przykładowego serwera.
2. Otwórz plik ZADANKO_2.py i postępuj zgodnie z instrukcjami. W tym zadaniu tworzysz prostą aplikację kliencką. Po wykonaniu wszystkich instrukcji przeklej całość do pliku ZADANKO_2.2.py, a następnie uruchom kolejno ZADANKO_1.py , ZADANKO_2.py i ZADANKO_2.2.py. **Miej na uwadze, że dla uproszczenia serwera każdy kolejny użytkownik przyjmuje kolejny indeks, a komunikacja przewidziana jest dla dwóch pierwszych użytkowników. Przy każdym teście restartuj serwer!**
3. Otwórz plik ZADANKO_3.py i postępuj zgodnie z instrukcjami. Twoim zadaniem jest napisanie bardzo prostej aplikacji, korzystając z moduły pygame.