<table>
<tr><td colspan="3">Politechnika Świętokrzyska w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Informatyki Stosowanej</td></tr>
<tr><td>Kierunek:
Informatyka</td><td colspan="2">Laboratorium:
Projekt - Języki Skryptowe</td></tr>
<tr><td>Grupa dziekańska:
3ID15B</td><td>Temat projektu
Statki</td><td>Wykonał :
Maksymilian Szeląg
Julia Kusztal</td></tr>
<tr><td>Data wykonania:
20.05.2025</td><td>Data oddania:
27.05.2025</td><td>Ocena i podpis:</td></tr>
</table>

1. Opis projektu

Celem projektu było stworzenie sieciowej gry typu statki, umożliwiającej rozgrywkę dwóch graczy w czasie rzeczywistym. Aplikacja została zbudowana w języku Python, z wykorzystaniem bibliotek Pygame (interfejs graficzny) oraz socket (komunikacja sieciowa TCP/IP).

2. Architektura aplikacji

Projekt składa się z dwóch głównych komponentów:

`game.py` – Aplikacja kliencka

- Graficzny interfejs użytkownika (GUI) zbudowany w Pygame.
- Obsługa logiki gry, plansz, rozmieszczania statków i tury graczy.
- Komunikacja z serwerem za pomocą gniazda TCP.

`server.py` – Serwer gry

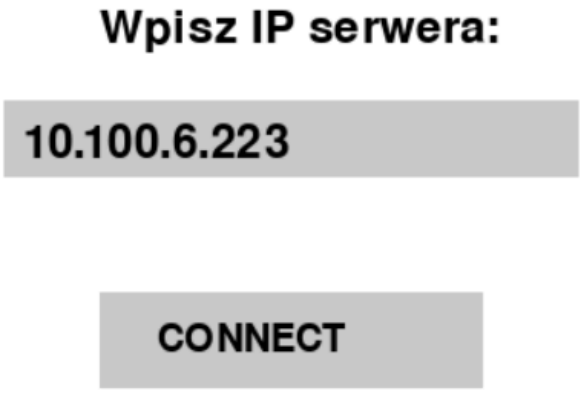
- Oczekuje na połączenia dwóch graczy.
- Przesyła dane między klientami i zarządza turami.
- Obsługuje sytuacje końca gry i rozłączenia przeciwnika.

3. Etapy działania gry

a) Ekran początkowy i sposób uruchomienia

Gra jest uruchamiana poprzez skrót na pulpicie(kliknięcie dwukrotnie w ikonę)

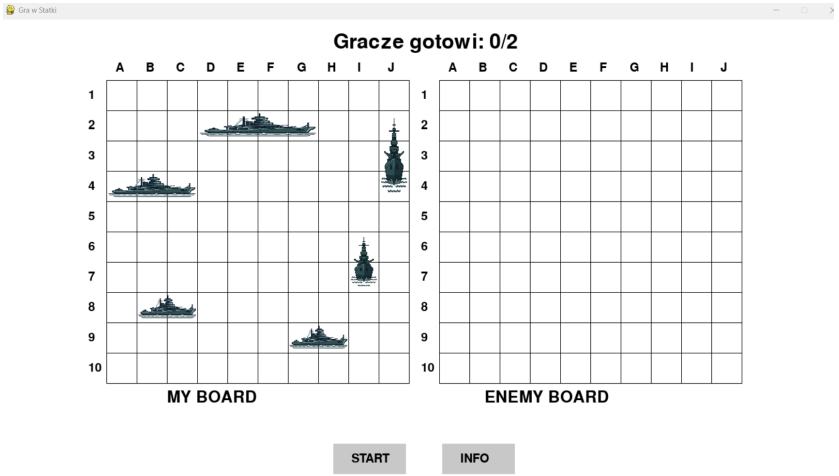
Następnie użytkownik wpisuje adres IP serwera.



Po kliknięciu przycisku „CONNECT” następuje próba połączenia z serwerem.

b) Rozmieszczanie statków

Gracz przeciąga statki na swoją planszę.



Statki można obracać prawym przyciskiem myszy.Każdy statek zajmuje odpowiednią liczbę pól (2x, 3x, 4x jednostki).

c) Start gry

Po rozmieszczeniu wszystkich statków gracz ma możliwość kliknięcia przycisku „START” “INFO”. Klikając “INFO” wyskakują nam zasady gry.

Zasady gry w Statki:

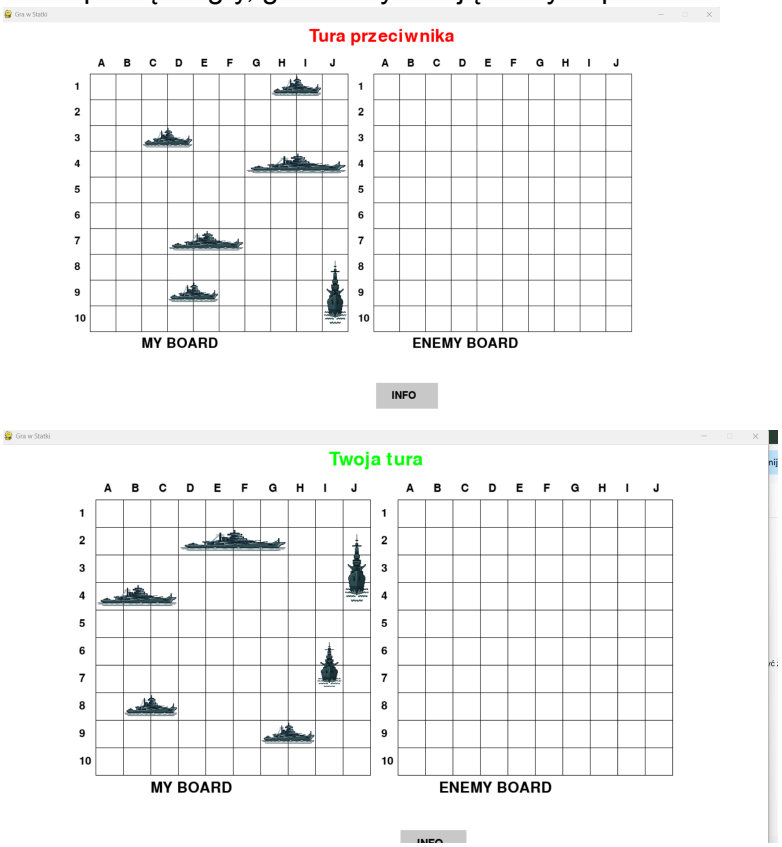
1. **Każdy gracz rozmieszcza swoje statki na planszy (MY BOARD).**
2. **Po rozpoczęciu gry gracze na zmianę oddają strzały.**
3. **Trafienie statku oznaczone jest kolorem czerwonym.**
4. **Pudło oznaczone jest kolorem niebieskim.**
5. **Wygrywa gracz, który pierwszy zatopi wszystkie statki przeciwnika.**
6. **Kliknij prawym przyciskiem myszy, aby obrócić statek podczas rozmieszczania.**

Po kliknięciu przycisku “START” gracz sygnalizuje że jest gotowy do rozpoczęcia rozgrywki.

Gracze gotowi: 1/2

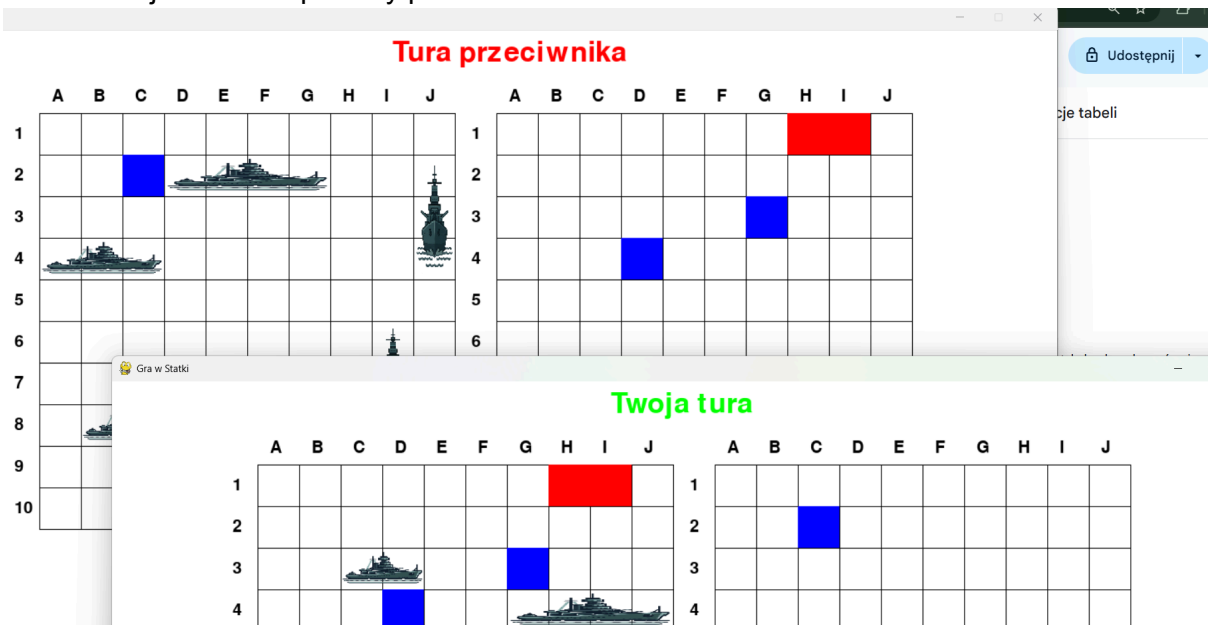
Gra rozpoczyna się, gdy obaj gracze są gotowi (Gracze gotowi: 2/2).

Po rozpoczęciu gry, gracze wykonują ruchy naprzemiennie.



d) Tura gracza

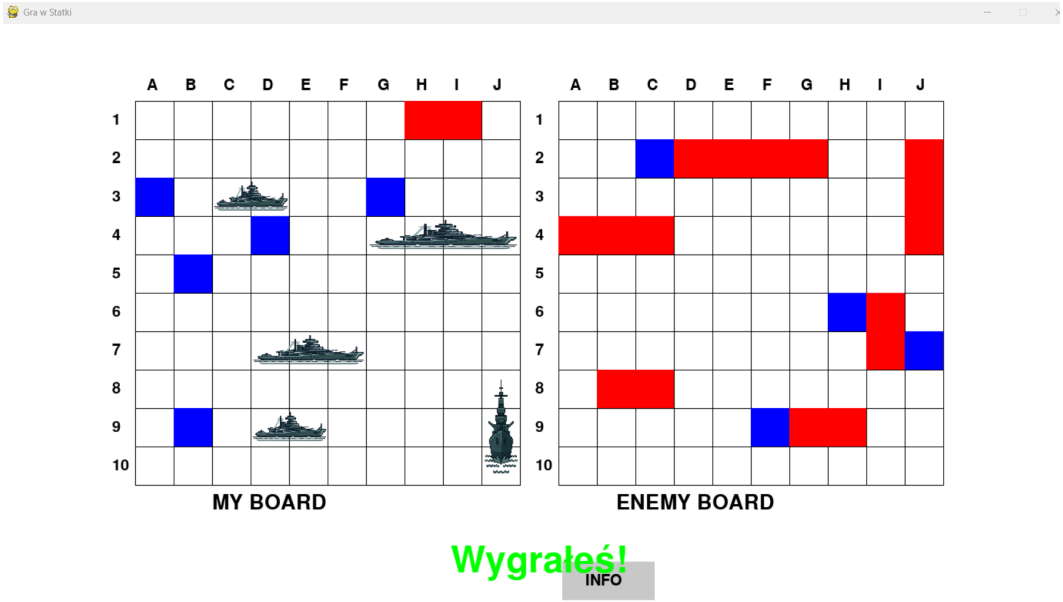
Gracz oddaje strzał na planszy przeciwnika.



Trafienie zaznaczone jest **czerwonym kolorem**, pudło – **niebieskim**.

e) Zakończenie gry

Gra kończy się po zatopieniu wszystkich statków przeciwnika.



Wyświetlany jest komunikat o wygranej jak w przykładzie powyżej i w taki sam sposób pojawia się komunikat o przegranej.
(Wygrałeś na tym screenie znajduje się na przycisku info, ponieważ zmniejszyłem okno aplikacji)

Po wygraniu gry, po kilku chwilach przeniesie nas do momentu, gdzie wpisujemy IP, przez co możemy znowu rozpocząć grę na nowo

4. Elementy techniczne

a) Komunikacja klient-serwer

- Protokół: TCP/IP
- Dane przesyłane są w formie serializowanej (moduł `pickle`).
- Klient nasłuchuje danych w osobnym wątku (`threading.Thread`).

b) Interfejs graficzny (Pygame)

- Plansze o rozmiarze 10x10 pól (każde po 50x50 pikseli).
- Dwie plansze: **MY BOARD** i **ENEMY BOARD**.
- Statki wyświetlane jako obrazy (`.png`) lub prostokąty w przypadku braku obrazów.

c) Logika rozmieszczania

- Zakazane nakładanie statków oraz sąsiadowanie (bufor bezpieczeństwa).

- Obsługa przeciągania i upuszczania (drag & drop).
- Walidacja poprawnego rozmieszczenia

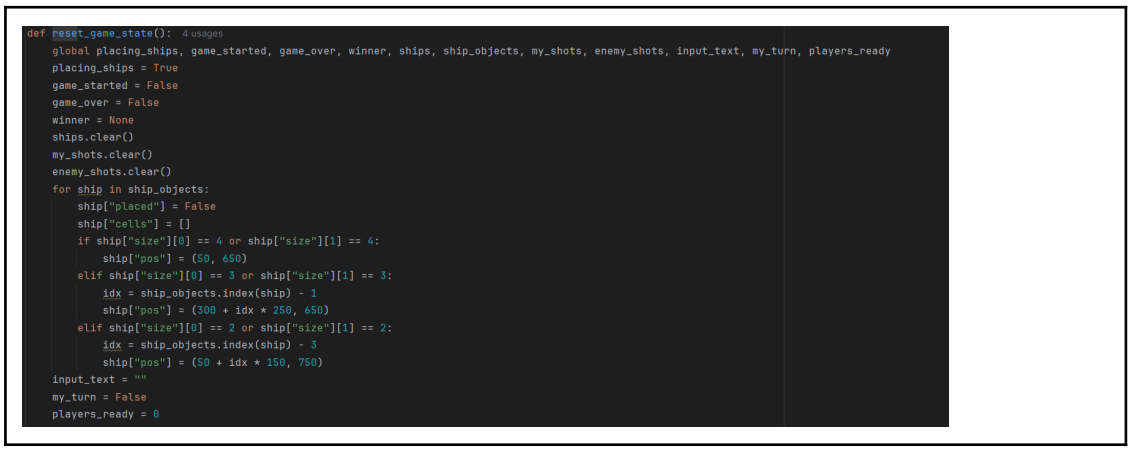
5. Kody do gry i ich krótkie wytłumaczenie:

Najważniejsze funkcje i ich opisy

Kod game.py

1. `reset_game_state()`

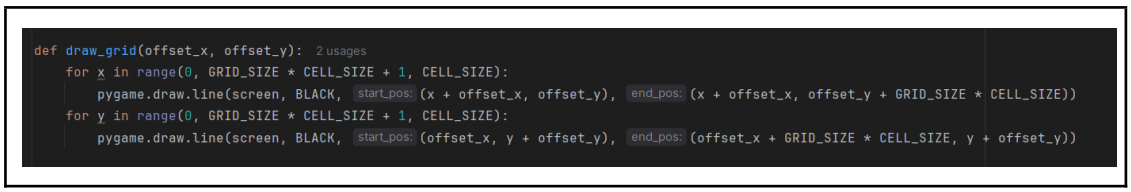
Resetuje stan gry – czyści plansze, ustawia statki w pozycjach początkowych, zeruje wszystkie zmienne (np. tura, gotowość graczy).



```
def reset_game_state():  4 usages
    global placing_ships, game_started, game_over, winner, ships, ship_objects, my_shots, enemy_shots, input_text, my_turn, players_ready
    placing_ships = True
    game_started = False
    game_over = False
    winner = None
    ships.clear()
    my_shots.clear()
    enemy_shots.clear()
    for ship in ship_objects:
        ship["placed"] = False
        ship["cells"] = []
        if ship["size"][0] == 4 or ship["size"][1] == 4:
            ship["pos"] = (50, 650)
        elif ship["size"][0] == 3 or ship["size"][1] == 3:
            idx = ship_objects.index(ship) - 1
            ship["pos"] = (300 + idx * 250, 650)
        elif ship["size"][0] == 2 or ship["size"][1] == 2:
            idx = ship_objects.index(ship) - 3
            ship["pos"] = (50 + idx * 150, 750)
    input_text = ""
    my_turn = False
    players_ready = 0
```

2. `draw_grid(offset_x, offset_y)`

Rysuje **siatkę planszy** (linie pionowe i poziome), bazując na podanych współrzędnych początkowych.



```
def draw_grid(offset_x, offset_y):  2 usages
    for x in range(0, GRID_SIZE * CELL_SIZE + 1, CELL_SIZE):
        pygame.draw.line(screen, BLACK, start_pos: (x + offset_x, offset_y), end_pos: (x + offset_x, offset_y + GRID_SIZE * CELL_SIZE))
    for y in range(0, GRID_SIZE * CELL_SIZE + 1, CELL_SIZE):
        pygame.draw.line(screen, BLACK, start_pos: (offset_x, y + offset_y), end_pos: (offset_x + GRID_SIZE * CELL_SIZE, y + offset_y))
```

3. `draw_coordinates(offset_x, offset_y)`

Dodaje **litery i cyfry** do planszy – oznaczenia kolumn i wierszy (A–J, 1–10).

```python
def draw_coordinates(offset_x, offset_y):  2 usages
    font = pygame.font.Font( name: None,  size: 30)
    for i in range(GRID_SIZE):
        text = font.render(NUMBERS[i],  antialias: True, BLACK)
        screen.blit(text,  dest: (offset_x - 30, offset_y + i * CELL_SIZE + 15))
    for i in range(GRID_SIZE):
        text = font.render(LETTERS[i],  antialias: True, BLACK)
        screen.blit(text,  dest: (offset_x + i * CELL_SIZE + 15, offset_y - 30))
```

4. `draw_button(rect, text)`

Rysuje **przycisk** z podanym tekstem w określonym miejscu.

```python
def draw_button(rect, text):  4 usages
    pygame.draw.rect(screen, GRAY, rect)
    font = pygame.font.Font( name: None,  size: 30)
    text_surf = font.render(text,  antialias: True, BLACK)
    screen.blit(text_surf,  dest: (rect.x + 30, rect.y + 15))
```

5. `draw_turn_message()`

Wyświetla informację **czyja tura** – gracza lub przeciwnika.

```python
def draw_turn_message():  1 usage
    if not game_over and game_started:
        font = pygame.font.Font( name: None,  size: 50)
        if my_turn:
            text = font.render( text: "Twoja tura",  antialias: True, GREEN)
        else:
            text = font.render( text: "Tura przeciwnika",  antialias: True, RED)
        screen.blit(text,  dest: (WIDTH // 2 - 100, 10))
```

6. `draw_game_over_message()`

Pokazuje **komunikat o wygranej lub przegranej** po zakończeniu gry.

```
def draw_game_over_message():  1 usage
    if game_over:
        font = pygame.font.Font( name: None,  size: 70)
        if winner:
            text = font.render( text: "Wygrałeś!",  antialias: True, GREEN)
        else:
            text = font.render( text: "Przegrałeś!",  antialias: True, RED)
        text_rect = text.get_rect(center=(WIDTH // 2, 700))
        screen.blit(text, text_rect)
```

7. `draw_ships_to_place()`

Pokazuje **statki do rozmieszczenia** na dole ekranu – te, które jeszcze nie zostały ustawione.

```
def draw_ships_to_place():  1 usage
    for ship in ship_objects:
        if not ship["placed"]:
            x, y = ship["pos"]
            if ship["image"] is not None:
                screen.blit(ship["image"],  dest: (x, y))
            else:
                w, h = ship["size"]
                pygame.draw.rect(screen, GREEN,  rect: (x, y, w * CELL_SIZE, h * CELL_SIZE))
```

8. `place_ship_on_board(ship, cell)`

Próbuje **umieścić statek** na planszy w wybranym miejscu. Sprawdza poprawność i unikanie kolizji.

```python
def place_ship_on_board(ship, cell):  1 usage
    col, row = cell
    w, h = ship["size"]
    if col + w > GRID_SIZE or row + h > GRID_SIZE:
        return False
    new_cells = [(col + i, row + j) for i in range(w) for j in range(h)]
    for existing_cell in ships:
        if existing_cell in new_cells:
            return False
    forbidden_zone = set()
    for x in range(col - 1, col + w + 1):
        for y in range(row - 1, row + h + 1):
            if 0 <= x < GRID_SIZE and 0 <= y < GRID_SIZE:
                forbidden_zone.add((x, y))
    for existing_cell in ships:
        if existing_cell in forbidden_zone:
            return False
    ship["cells"] = new_cells
    ship["placed"] = True
    ships.extend(new_cells)
    return True
```

9. `get_ship_at_pos(pos)`

Zwraca **statek, który został kliknięty** (jeśli jakiś został trafiony myszką).

```python
def get_ship_at_pos(pos):  2 usages
    x, y = pos
    for ship in ship_objects:
        if not ship["placed"]:
            ship_x, ship_y = ship["pos"]
            w, h = ship["size"]
            if ship_x <= x < ship_x + w * CELL_SIZE and ship_y <= y < ship_y + h * CELL_SIZE:
                return ship
    return None
```

10. `rotate_ship(ship)`

Obraca wybrany statek (zmienia orientację pion/poziom) i aktualizuje jego obrazek.

```python
def rotate_ship(ship):  1 usage
    if not ship["placed"]:
        ship["size"] = (ship["size"][1], ship["size"][0])
        ship_length = max(ship["size"])
        ship["orientation"] = "v" if ship["size"][0] == 1 else "h"
        ship["image"] = ship_images[f"{ship_length}_{ship['orientation']}"]
```

11. `shoot(cell)`

Oddaje **strzał** na pole przeciwnika – zapisuje go lokalnie i wysyła do serwera.

```python
def shoot(cell):  1 usage
    global my_turn
    if not placing_ships and my_turn and cell not in [shot[0] for shot in my_shots]:
        my_shots.append((cell, False))
        client_socket.send(pickle.dumps(("shot", cell)))
```

12. `get_cell(pos)`

Przekształca współrzędne kliknięcia myszki na **współrzędne planszy** – określa, które pole kliknięto i na której planszy.

```python
def get_cell(pos):  2 usages
    x, y = pos
    if LEFT_BOARD_X <= x < LEFT_BOARD_X + GRID_SIZE * CELL_SIZE and BOARD_Y <= y < BOARD_Y + GRID_SIZE * CELL_SIZE:
        col = (x - LEFT_BOARD_X) // CELL_SIZE
        row = (y - BOARD_Y) // CELL_SIZE
        return (col, row), 0
    elif RIGHT_BOARD_X <= x < RIGHT_BOARD_X + GRID_SIZE * CELL_SIZE and BOARD_Y <= y < BOARD_Y + GRID_SIZE * CELL_SIZE:
        col = (x - RIGHT_BOARD_X) // CELL_SIZE
        row = (y - BOARD_Y) // CELL_SIZE
        return (col, row), 1
    return None, None
```

13. `receive_data()`

Nasłuchuje wiadomości z serwera – obsługuje trafienia, tury, koniec gry, gotowość graczy itd. Działa w tle w osobnym wątku.

```python
def receive_data():  1 usage
    global my_turn, game_over, winner, players_ready, game_started, screen_mode, placing_ships
    while running:
        try:
            data = pickle.loads(client_socket.recv(1024))
            print(f"Received: {data}")
            if isinstance(data, tuple) and data[0] == "shot":
                cell = (int(data[1][0]), int(data[1][1]))
                if cell not in [shot[0] for shot in enemy_shots]:
                    hit = cell in ships
                    enemy_shots.append((cell, hit))
                    client_socket.send(pickle.dumps(("hit" if hit else "miss", cell)))
                    if all(cell in [shot[0] for shot in enemy_shots if shot[1]] for cell in ships):
                        game_over = True
                        winner = False
                        client_socket.send(pickle.dumps(("game_over", False)))
                    my_turn = True
            elif isinstance(data, tuple) and data[0] in ["hit", "miss"]:
                cell = (int(data[1][0]), int(data[1][1]))
                for i, (shot_cell, _) in enumerate(my_shots):
                    if shot_cell == cell:
                        my_shots[i] = (cell, data[0] == "hit")
                hits = [shot for shot in my_shots if shot[1]]
                if len(hits) == len(ships):
                    game_over = True
                    winner = True
                    client_socket.send(pickle.dumps(("game_over", True)))
            elif data == "your_turn":
                my_turn = True
            elif data == "wait":
                my_turn = False
            elif isinstance(data, tuple) and data[0] == "game_over":
                game_over = True
                winner = data[1]
            elif isinstance(data, tuple) and data[0] == "players_ready":
                players_ready = data[1]
                if players_ready == 2:
                    game_started = True
                    placing_ships = False
            elif isinstance(data, tuple) and data[0] == "opponent_disconnected":
                screen_mode = "connect"
                reset_game_state()
                print("Opponent disconnected. Returning to connect screen.")
        except (ConnectionResetError, ConnectionAbortedError) as e:
            print(f"Server disconnected: {e}")
            screen_mode = "connect"
            reset_game_state()
            break
```

14. `connect_to_server(ip)`

Łączy się z serwerem po podanym IP, **rozpoczyna komunikację** z serwerem i uruchamia odbieranie danych.

```python
def connect_to_server(ip):  2 usages
    global client_socket, running
    # Reset game state before connecting to ensure a fresh start
    reset_game_state()
    PORT = 5555
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((ip, PORT))
        threading.Thread(target=receive_data, daemon=True).start()
        return True
    except Exception as e:
        print(f"Connection error: {e}")
        return False
```

kod Server.py

1. `get_local_ip()`

Pobiera lokalny adres IP komputera, aby móc go wyświetlić i użyć do połączenia klientów.

```python
def get_local_ip():  1 usage
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("8.8.8.8", 80))
    ip = s.getsockname()[0]
    s.close()
    return ip
```

2. `broadcast(message)`

Wysyła podaną wiadomość do wszystkich połączonych graczy (klientów).
Używane np. do przekazania: „gra rozpoczęta”, „koniec gry”, „gracze gotowi”.

```python
def broadcast(message):
    for client in clients:
        client.sendall(pickle.dumps(message))
```

4. `start_server()`

Uruchamia główną pętlę serwera, w której:

- Oczekuje na połączenie dwóch graczy.
- Przypisuje im obsługę w osobnych wątkach (`handle_client`).
- Resetuje stan po rozłączeniu.
- Obsługuje zamknięcie serwera klawiszem (Ctrl+C).

```python
def start_server():  1 usage
    global players_ready, game_started
    while True:
        try:
            while len(clients) < 2:
                client, addr = server_socket.accept()
                print(f"Player connected: {addr}")
                with lock:
                    clients.append(client)
                    client.sendall(pickle.dumps(("players_ready", players_ready)))
            threading.Thread(target=handle_client, args=(clients[0], clients[1], 0)).start()
            threading.Thread(target=handle_client, args=(clients[1], clients[0], 1)).start()
            while len(clients) > 0:
                pass
            with lock:
                players_ready = 0
                game_started = False
        except KeyboardInterrupt:
            print("Shutting down server...")
            for client in clients:
                client.close()
            server_socket.close()
            break
```

3. `handle_client(client, opponent, player_id)`

Główna funkcja do obsługi klienta (gracza). Odbiera dane, analizuje je i przekazuje odpowiednie informacje do przeciwnika lub obu graczy.

Działa w osobnym wątku dla każdego gracza.

Obsługuje:

- gotowość do gry,
- strzały i odpowiedzi (trafienie/pudło),
- przełączenie tury,
- zakończenie gry,
- rozłączenie gracza.

```python
def handle_client(client, opponent, player_id):  2 usages
    global current_player, players_ready, game_started
    try:
        while True:
            data = client.recv(1024)
            if not data:
                break
            decoded_data = pickle.loads(data)
            print(f"Received from player {player_id}: {decoded_data}")

            with lock:
                if isinstance(decoded_data, tuple) and decoded_data[0] == "ready":
                    if not game_started:
                        players_ready += 1
                        broadcast(("players_ready", players_ready))
                        if players_ready == 2:
                            game_started = True
                            clients[0].sendall(pickle.dumps("your_turn"))
                            clients[1].sendall(pickle.dumps("wait"))
                            print("Game started!")
                elif isinstance(decoded_data, tuple) and decoded_data[0] == "shot":
                    if game_started and player_id == current_player:
                        opponent.sendall(data)
                    else:
                        client.sendall(pickle.dumps("wait"))
                elif isinstance(decoded_data, tuple) and decoded_data[0] in ["hit", "miss"]:
                    opponent.sendall(data)
                    if decoded_data[0] == "miss":
                        current_player = 1 - current_player
                        client.sendall(pickle.dumps("your_turn"))
                        opponent.sendall(pickle.dumps("wait"))
                    else:
                        opponent.sendall(pickle.dumps("your_turn"))
                        client.sendall(pickle.dumps("wait"))
                elif isinstance(decoded_data, tuple) and decoded_data[0] == "game_over":
                    broadcast(decoded_data)
                    game_started = False
                    players_ready = 0
                    break
    except Exception as e:
        print(f"Error in handle_client: {e}")
    finally:
        client.close()
        with lock:
            if opponent in clients:
                opponent.sendall(pickle.dumps(("opponent_disconnected", None)))
                clients.remove(opponent)
            if client in clients:
                clients.remove(client)
            if not game_started:
                players_ready = 0
        print(f"Player {player_id} disconnected.")
```

6. Wnioski

Projekt pozwolił na praktyczne zastosowanie programowania sieciowego, wielowątkowości, grafiki 2D oraz obsługi zdarzeń. Dzięki połączeniu interaktywnego GUI z komunikacją w czasie rzeczywistym, gra jest dynamiczna, intuicyjna i wciągająca.