WYKORZYSTNIE SOCKET'ÓW W PYTHONIE

Historia gniazd

- RFC 147(1971)
- ARPANET
- Berkeley sockets(1983)

Socket'y w Python'ie

Moduł socket I jego podstawowe metody:

- Socket()
- Bind()
- Listen()
- Accept()
- Recv()
- Send()
- Close()

Metoda Socket(family, type)

Metoda ta zwraca obiekt typu socket dla określonej rodziny adresów I typu.

- Family
  - AF_UNIX - komunikacja między procesami
  - AF_INET – komunikacja sieciowa IPv4
  - AF_INET6 – komunikacja sieciowa IPv6
- Type
  - SOCK_STREAM - połączenia TCP
  - SOCK_DGRAM - połączenia UDP

```
setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Metoda bind((HOST,PORT))

Służy do przywiązania gniazda do danego interfejsu za pomocą krotki.

- HOST
  - IP
  - Gethostbyname()
  - ' ' - akceptuje połączenia na wszystkich hostach
- PORT
  - 1-65535
  - 1-1023 privileged ports
  - Port 0 – OS wybiera port za nas

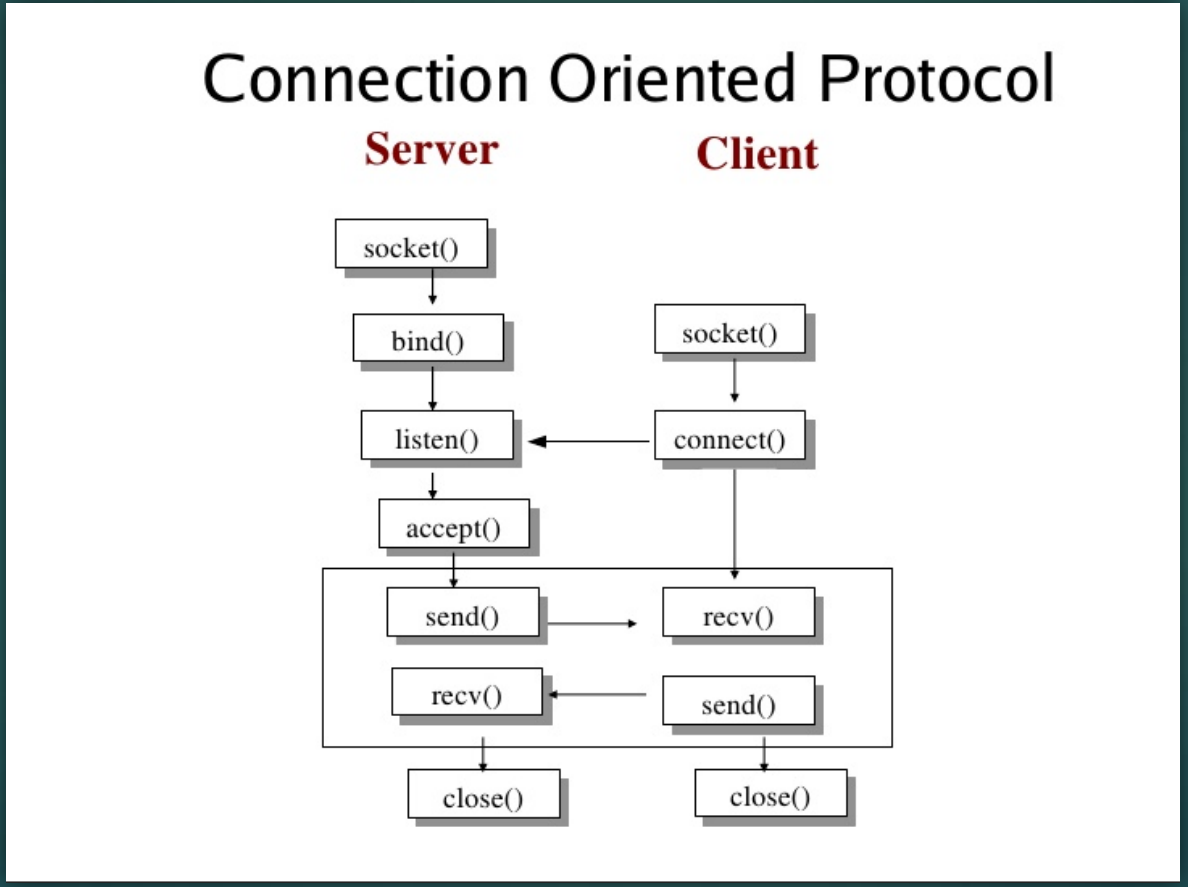
Metody listen(backlog) I accpet()

- Listen() zmienia stan gniazda na nasłuchiwanie
  - Posiada atrybut backlog do określenia wielkości kolejki
  - Wartość domyślna dla systemu Linux:
  - cat `/proc/sys/net/core/somaxconn`
- `Accept() nomen omen akceptuje nowe połączenie`
  - `Zwraca parę` (conn, address)
  - Conn to socket, który służy do przesyłania danych
  - Address to adres po drugiej stronie połączenia, czyli klienta

Recv() I Send()

- Metoda revc(bufsize) służy do odbierania danych
  - Bufsize maksymalny rozmiar danych otrzymanych na raz
  - Bez podania argumentów przyjmuje wartość 1024
  - Powinien być jak najmniejszą potęgą 2
- Metodą send() wysyłamy dane

Socket'y TCP


Connection Oriented Protocol
Server
Client
socket()
bind()
listen()
accept()
send()
recv()
close()
socket()
connect()
recv()
send()
close()

Serializacja w Python'ie

- Do przesyłania danych client-server wykorzystaliśmy moduł pickle
- serializacja polega na zamianie obiektów w byte'y możliwe do przesyłania do pamięci czy przez sieć.
- Funkcja dumps() zamienia obiekt w bytecode
- Operacja w drugą stronę to loads().

Porównanie

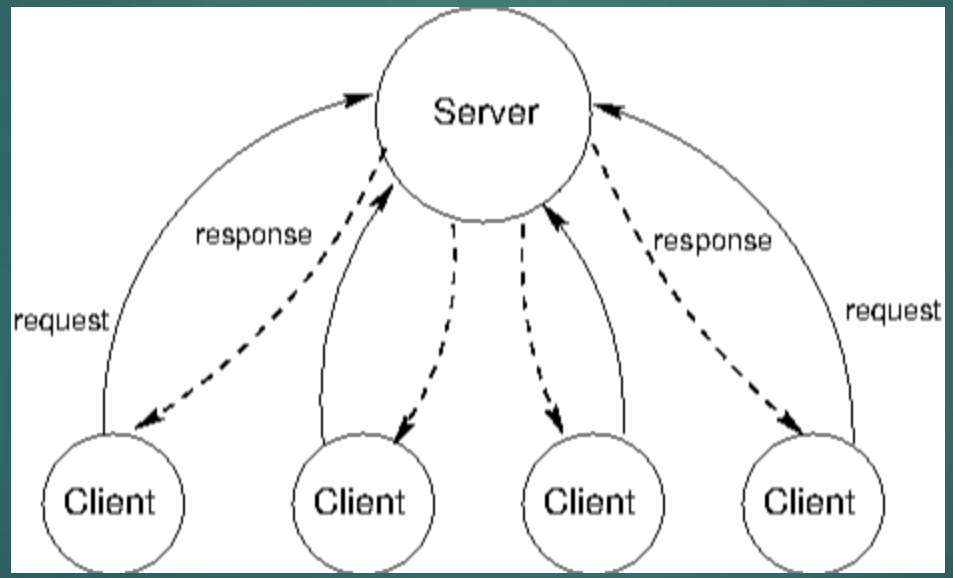
Pickle

- Dostępny tylko w Pythonie
- Zamienia objekty w bytecode
- Wspiera większość typów danych w tym klasy I funkcje
- Nie jest w pełni bezpieczny
- Napisany częściowo w c

JSON

- Powszechnie używany
- Przechowuje dane w formie tekstu
- Czytelny dla człowieka
- Nie może seralizować wszystkich o biektów

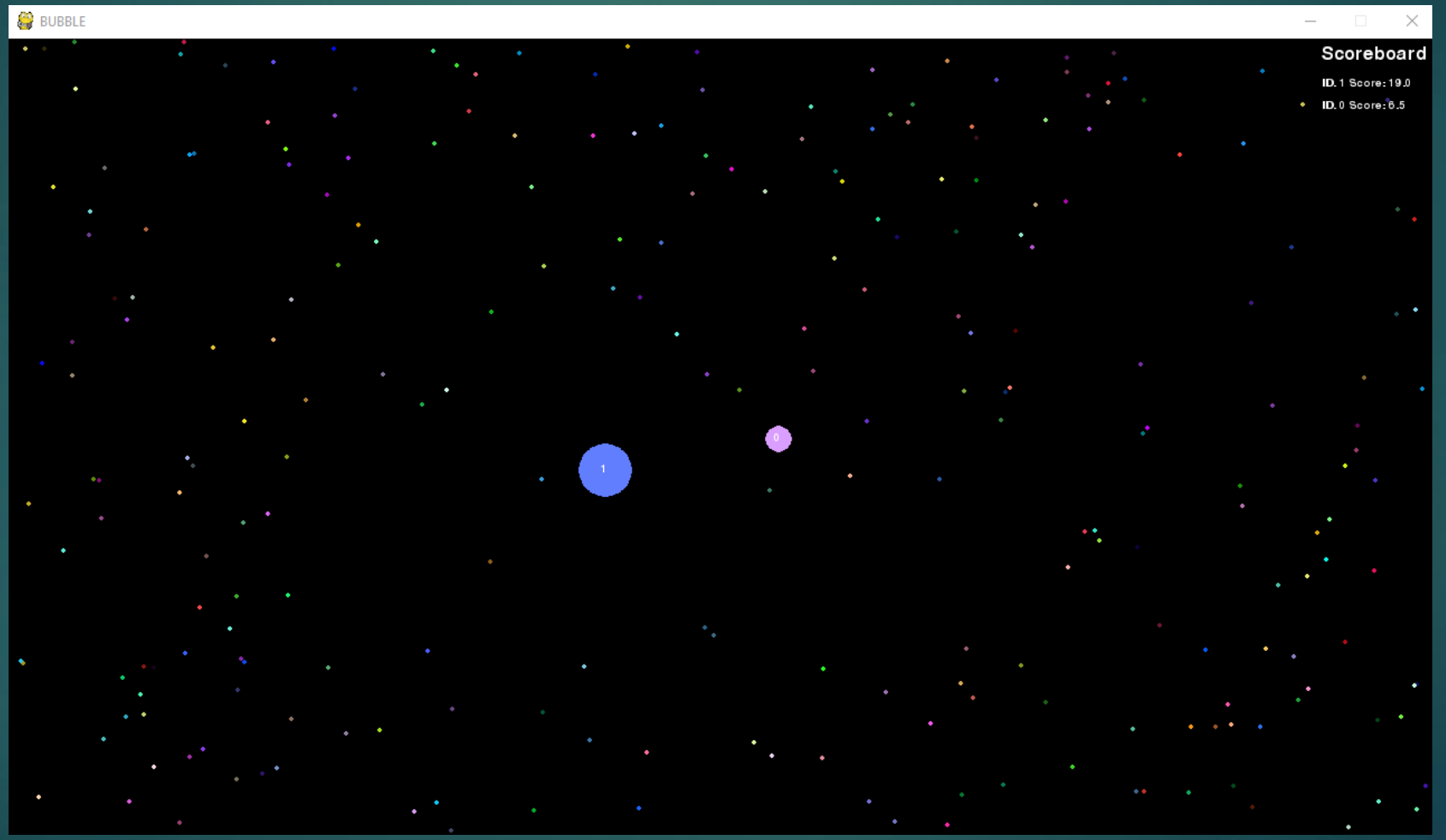
Architektura client-server



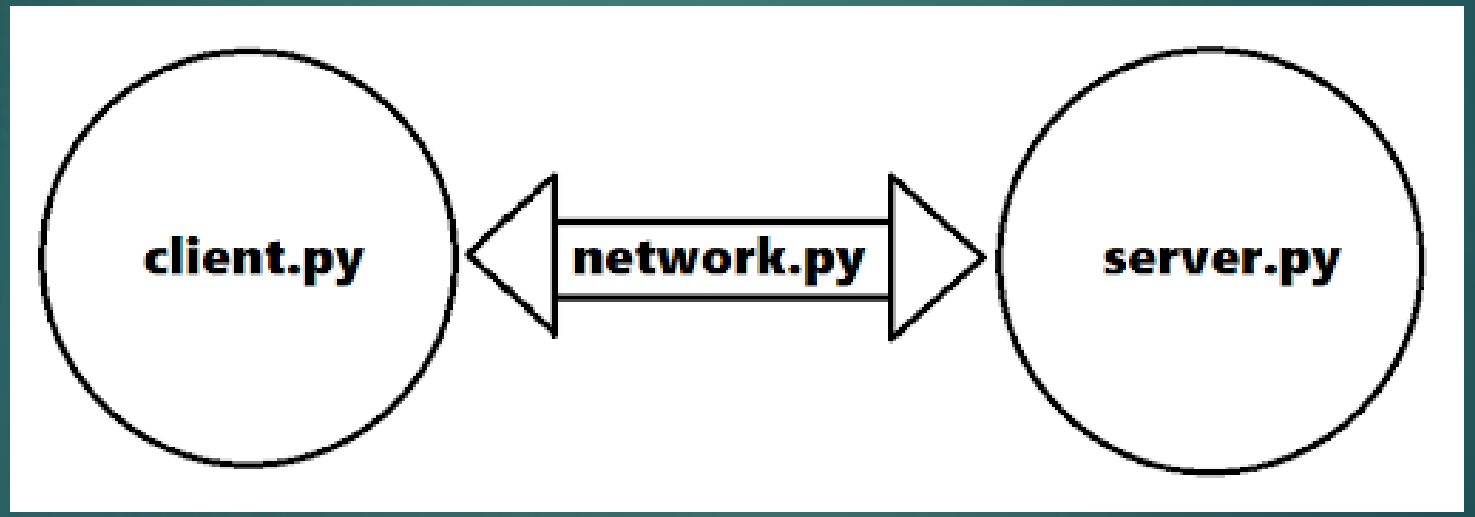
Wielowątkowość w Python'ie

- Do obsułgi wielu client'ów można wykorzystać moduł _threads
- `_thread.start_new_thread`(*function, args*)
  - Tworzy nowy wątek na serwerze
  - Wywołuje function(args)
  - Kończy działanie gdy function coś zwróci

Zaprezentowanie gry



Przedstawienie połączenia między client.py a serwer.py



Inicjalizacja serwera

- Tworzenie i nadanie ustawień gniazda (linie 13-17)
- Zmiana stanu gniazda na nasłuchiwanie (linia 21)
- Zdefiniowanie zmiennych globalnych (linie 27-31)

```
8   from player import Player, Ball
9
10  HOST = "192.168.0.10"
11  PORT = 8999
12
13  s = socket(AF_INET, SOCK_STREAM)
14  s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
15
16  try:
17      s.bind((HOST, PORT))
18  except error as e:
19      print(str(e))
20  try:
21      s.listen(4)  # max. 4 users in queue
22  except error as e:
23      print(str(e))
24  print("Waiting for a connection")
25
26  # GAME VARIABLES
27  players = list()
28  balls = list()
29  connections = 0
30  my_id = 0
31  resolution = (1280, 720)
```

Oczekiwanie na nowe połączenie do serwera, tworzenie wątków

- Tworzenie jednego wątku serwera
- W pętli oczekiwanie na nowe połączenie z serwerem
- Tworzenie osobnych wątków dla każdego gracza

```
147    start_new_thread(threaded_server,())
148    while True:
149        # W tej pętli serwer stale wyczekuje nowych połączeń
150        # Serwer powinien akceptować takie połączenia (socket przypisany jest do zmiennej 's')
151        # Następnie wywołać funkcje threded_client z gniazdem i unikalnym id klienta
152        # Pro tip: Skorzystaj ze zmiennej my_id
```

Inicjalizacja połączenia od strony client.py

- Użycie globalnych zmiennych players oraz balls
- Stworzenie obiektu klasy Network
- Łączenie się z serwerem metodą network.connect()

```
49    def main():
50        global players, balls
51
52        # connect to server and get data
53        network = Network()
54        (players, balls, my_id) = network.connect()
55
```

Klasa Network

- Służy do wymiany danych między clientem a serwerem
- Metoda connect wywoływana jest przy pierwszym połączeniu

```
5   class Network:
6       def __init__(self):
7           self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8           self.host = "172.17.177.158"
9           self.port = 8999
10          self.addr = (self.host, self.port)
11
12      def connect(self):
13          self.client.connect(self.addr)
14          data = self.client.recv(2048 * 12)
15          return pickle.loads(data)
```

Client.py - główna pętla

- Pętla while wykonywana do momentu wyjścia z gry (linie 58-79)
- clock.tick() = liczba odświeżeń na sekundę (linia 60)
- zmiana pozycji gracza po wciśnięciu strzałki na klawiaturze (linia 73)
- Wysłanie pozycji naszego gracza (zmienna data) przy użyciu metody network.send()
- Otrzymanie od serwera pozycji graczy i NPC (linia 79)

```
49  def main():
50      global players, balls
51
52      # connect to server and get data
53      network = Network()
54      (players, balls, my_id) = network.connect()
55
56      clock = pygame.time.Clock()
57      run = True
58      while run:
59
60          clock.tick(30)  # fps = frames per second
61          player = players[my_id]
62
63          for event in pygame.event.get():
64              if event.type == pygame.QUIT:
65                  run = False
66                  sys.exit()
67
68              if event.type == pygame.KEYDOWN:
69                  # if user hits a escape key close program
70                  if event.key == pygame.K_ESCAPE:
71                      run = False
72                      pygame.quit()
73          player.move(resolution)
74
75          redrawWindow(window, players, balls)
76          pygame.display.update()
77
78          data = player
79          (players, balls) = network.send(data)
80
81      network.disconnect()
82      pygame.quit()
83      quit()
```

Metoda send() w klasie Network

- Wysyłanie informacji między serwerem a clientem
- self.client.send() = metoda służąca do wysyłania danych od clienta do serwera
- reply = to dane wysłane z serwera do clienta

```
20      def send(self, data):
21
22          try:
23
24              self.client.send(pickle.dumps(data))
25
26              reply = self.client.recv(2048 * 12)
27              try:
28                  reply = pickle.loads(reply)
29              except Exception as e:
30                  print(e)
31
32              return reply
33          except socket.error as e:
34              print(e)
```

Wątek serwera

- Tworzony jest tylko jeden
- Modyfikujemy w nim globalne zmienne balls oraz players
- Wywołujemy funkcje które sprawdzają kolizje między graczami oraz graczem a NPC
- Dodajemy NPC

```
89  def threaded_server():
90      global balls, players
91      create_balls(100)
92      while True:
93          balls_collision(players, balls)
94          players_collision(players)
95          if len(balls) < 150:
96              create_balls(100)
```

Wątek clienta

- Tworzonych jest tyle wątków ile jest podłączonych graczy
- Dodajemy kolejnego gracza (linia 111)
- Wysyłamy clientowi dane w chwili dołączenia do gry (linia 114)
- Otrzymujemy dane z aktualną pozycją danego gracza (linia 120)
- Aktualizujemy jego pozycję w zmiennej globalnej (linie 123-128)
- Wysyłamy do clienta zmienne players oraz balls (linie 130-131)
- W przypadku utracenia połączenia usuwamy gracza z listy plyayers i zamykamy połączenie (linie 139-142)

```
108  def threaded_client(connection, player):  # player =                layer
109      global balls, players, connections
110      id = player
111      add_player(id)
112
113      init_data = (players, balls, id)
114      connection.send(pickle.dumps(init_data))
115
116      while True:
117
118          try:
119
120              data = connection.recv(2048 * 12)
121              update = pickle.loads(data)
122
123              if isinstance(update, Player):  # recv client object
124                  if players[id].radius == 0:
125                      players[id].respawn(resolution)
126                  else:
127                      players[id].x = update.x
128                      players[id].y = update.y
129
130              send_data = pickle.dumps((players, balls))
131              connection.send(send_data)  # sending players, balls
132
133          except Exception as e:
134              print(e)
135              break
136
137      # disconneted
138      try:
139          print(f"Connection {id} Close")
140          remove_player(id)
141          connections -= 1
142          connection.close()
```

Bibliografia

- https://docs.python.org/3/library/socket.html
- https://docs.python.org/3/library/pickle.html
- https://realpython.com/python-sockets/#using-hostnames
- https://www.youtube.com/watch?v=_fx7FQ3SP0U&list=PLzMcBGfZo4-kR7Rh-7JCVDN8lm3Utumvq&index=1
- https://www.educba.com/python-pickle-vs-json/