Relatório Copas

Discente: André Gustavo Silveira
GRR: 20232345

Para a atividade denominada Trabalho 2, foi definido o desenvolvimento de um programa que simulasse uma rede em anel, onde as máquinas conectadas jogassem o jogo Copas. Por questão de familiaridade com a linguagem, foi escolhida a linguagem Python para o desenvolvimento do projeto, sendo uma linguagem poderosa o suficiente para atender as especificações propostas.

Estrutura

Todas as máquinas que participam da comunicação são originárias da mesma classe, sendo que o ID de criação delas determina quem inicia o jogo. O arquivo “servidor.py” é inicializado para todas as máquinas, e ele é bem sucinto, apenas iniciando a máquina selecionada:

servidor.py > ...

```python
import sys

from machine import  RingMachine

def main():
    '''
        Utilization: python servidor.py [machine_id]
            machine_id -> varia de 0 a 3. Se for zero, vai esperar a
            mensagem das próximas
    '''

    if len(sys.argv) < 2 or int(sys.argv[1]) > 3:
        print("Utilization: python servidor.py [0 <= machine_id < 4]")

    machine_id = int(sys.argv[1])

    try:
        machine = RingMachine(machine_id)
        machine.run()
    except Exception as e:
        print(f"Falha ao inicializar máquinas: {e}")
        return

    except KeyboardInterrupt:
        print("\nCtrl+C detectado. Iniciando desligamento...")
        machine.close_socket()

    finally:
        print("Limpando recursos...")
        print("Programa finalizado.")

if __name__ == "__main__":
    main()
```

A função “run()” da classe **RingMachine** realiza a inicialização da máquina, e no caso da primeira máquina, espera as outras conectarem aos seus sockets e mandarem mensagens de conexão:

```python
def run(self):
    if(self.machine_id == 0):
        print("Esperando pelas outras máquinas")
        while(len(self.connected_machines) < 3):
            data, _ = self.sock.recvfrom(1024)
            message_str = data.decode('utf-8')

            self._handle_incoming_message(message_str)

        self._initiate_cards()
        self._play_card()
        self._send_token()
    else:
        self._send_connection_message()

    while True:
        try:
            data, _ = self.sock.recvfrom(1024) # Tamanho do buffer de 1024 bytes
            message_str = data.decode('utf-8')

            self._handle_incoming_message(message_str)

            if self.has_token:
                print("Token recebido. Enviando para a próxima máquina")
                self._send_token()

        except socket.error as e:
            print(f"[M{self.machine_id}] Erro de socket: {e}. Saindo do loop de execução.")
            self.close_socket()

            break
        except Exception as e:
            print(f"[M{self.machine_id}] ERRO durante recebimento/processamento: {e}")
            self.close_socket()
            break
```

Por falar em mensagem de conexão, o protocolo montado tem o seguinte formato: Utiliza-se uma classe chamada **RingMessage**, que possui os campos “type”, para qual tipo de mensagem está sendo enviado, “payload”, para as informações a serem transmitidas, e target, para a máquina que irá receber a mensagem.

```python
class RingMessageType(Enum):
    CONNECT = 1
    SETUP = 2
    TOKEN = 3
    POINTS = 4
    END = 5
    TURN_OFF = 6


class RingMessage:

    def __init__(self, type: RingMessageType, content: str, target: int):
        self.type = type.value
        self.payload = content
        self.target = target

    def __str__(self):
        return f"{self.type};{self.target};{self.payload}"

    def to_bytes(self):
        return self.__str__().encode("utf-8")
```

Como mostrado, foram previstos seis tipos:

- CONNECT (1)
  - Exclusivo para mensagens de conexão
- SETUP (2)
  - Exclusivo para as ações de recebimentos de cartas, durante a fase de preparação do jogo
- TOKEN (3)
  - Utilizado para mensagens comuns e para passagens de bastão
- POINTS (4)
  - Chamada para adição de pontos a uma máquina quando ela perdeu a rodada anterior. Funciona como passagem de bastão também
- END (5)
  - Aviso para contagem de pontos final, quando uma máquina já atingiu a mão vazia
- TURN_OFF (6)
  - Aviso de que o anel está sendo quebrado, deve ocorrer o desligamento da máquina

Admito que o tipo 3 e 4 poderiam ser juntados, alterando-se o payload da mensagem, mas achei que a decisão certa a se tomar seria a que facilitasse a visualização da função "_handle_incoming_message", já que ela trata todos os casos de recebimento de mensagem.