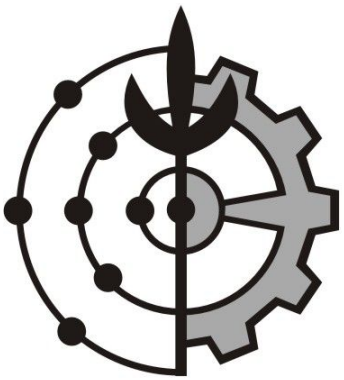


UFRRJ
UNIVERSIDADE FEDERAL RURAL
DO RIO DE JANEIRO

Laboratório de Redes de Computadores

Filomilitar com sockets TCP

Aluno:
Artur Alves Souza Silva
Bruno Ferreira

Departamento de Ciência de Computação
Professor: Marcel William

Julho de 2017

1. Introdução

O Filomilitar é um jogo de dois usuários. Cada jogador possui um grupo de pessoas, esse grupo tem a denominação de nação. Cada pessoa contém níveis de inclinação ideológica, os níveis são filosófico ou militar.

O jogador deve desenvolver conhecimentos, assim conseguindo pontos. O jogador também conta com 5 filósofos e 5 militares, que sao utilizados para desenvolver os conhecimentos.

O conhecimento leva em consideração o número de pessoas envolvidas e o número de especialistas da nação com a mesma ideologia do conhecimento, o número de pessoas define a dificuldade de desenvolvimento total do conhecimento. O desenvolvimento do conhecimento por ano é igual ao número de pessoas da nação dividido por 2, essa taxa tem como função não permitir que um conhecimento com um valor de dificuldade alto seja desenvolvido em um turno, sendo que esse valor pode variar dependendo das ideologias das pessoas. Assim que o jogador consegue desenvolver o conhecimento totalmente, isso é, quando a dificuldade do conhecimento chega em 0, o jogador ganha pontos igual ao número da dificuldade inicial do conhecimento.

O jogador que obter 50 pontos antes vence o jogo.

No relatório será apresentado a implementação do Filomilitar onde dois jogadores competem entre si e espectadores conseguem acompanhar o jogo.

2. Descrição

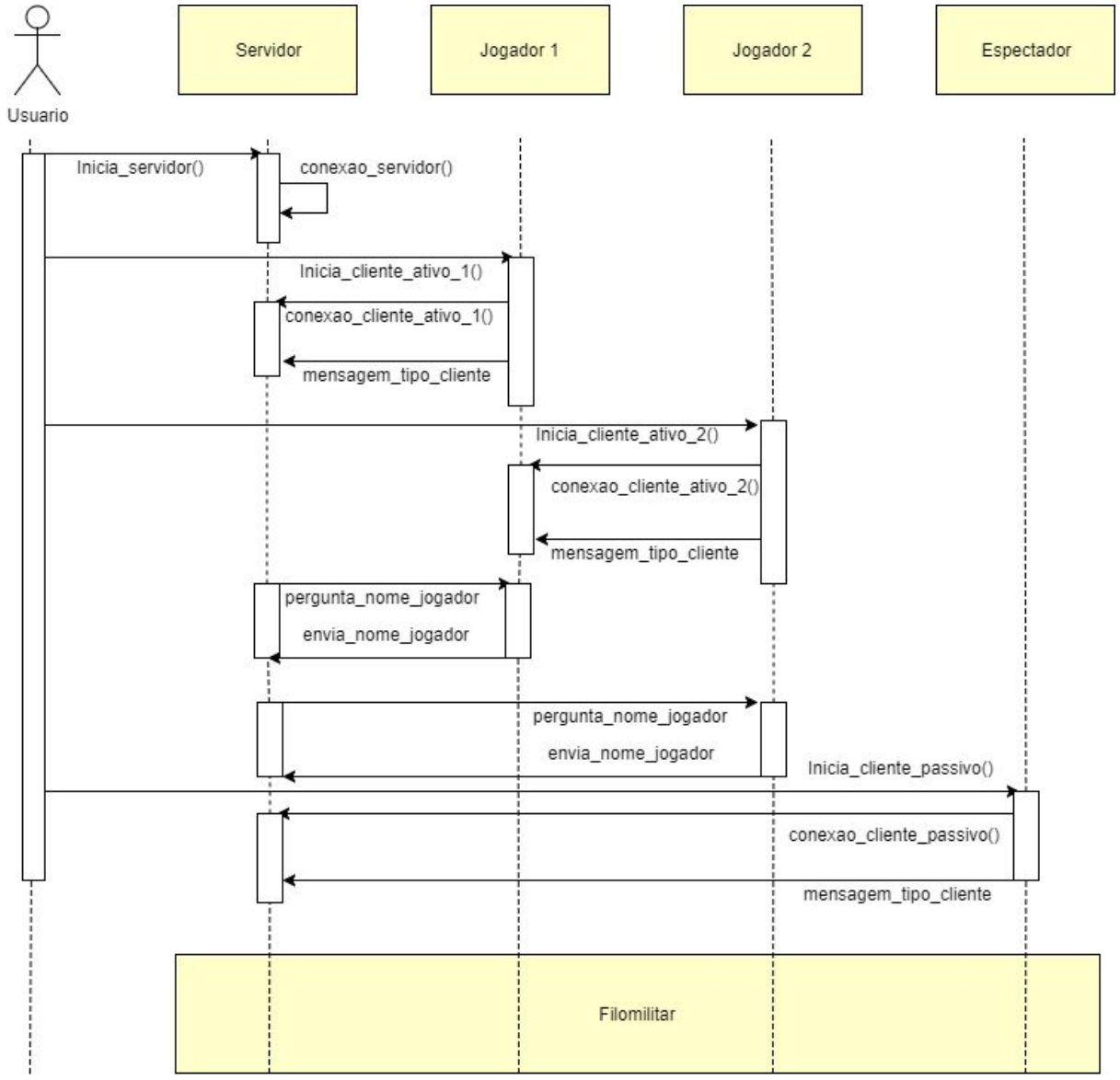
Conexão sendo estabelecida:

Quando o cliente ativo 1 e 2 estabelecerem conexão com o servidor, os clientes ativos enviam a mensagem “player” para o servidor, com a finalidade de informar que desejam se conectar ao servidor como jogadores.

Logo após. quando o cliente passivo estabelecer conexão com o servidor, o mesmo envia ao servidor a mensagem “espectador”, assim informando que deseja se conectar como espectador.

O servidor exibe quando recebe uma conexão, informando que tipo de cliente foi conectado.

Assim que detecta que dois jogadores se conectaram o servidor envia para os jogadores a seguinte mensagem “Escolha um nome.”. O servidor envia para o primeiro jogador, aguarda a resposta do mesmo e só depois de obter a resposta que a mesma mensagem é enviada ao segundo jogador. Quando obtido o nome dos jogadores, o servidor da início ao jogo.



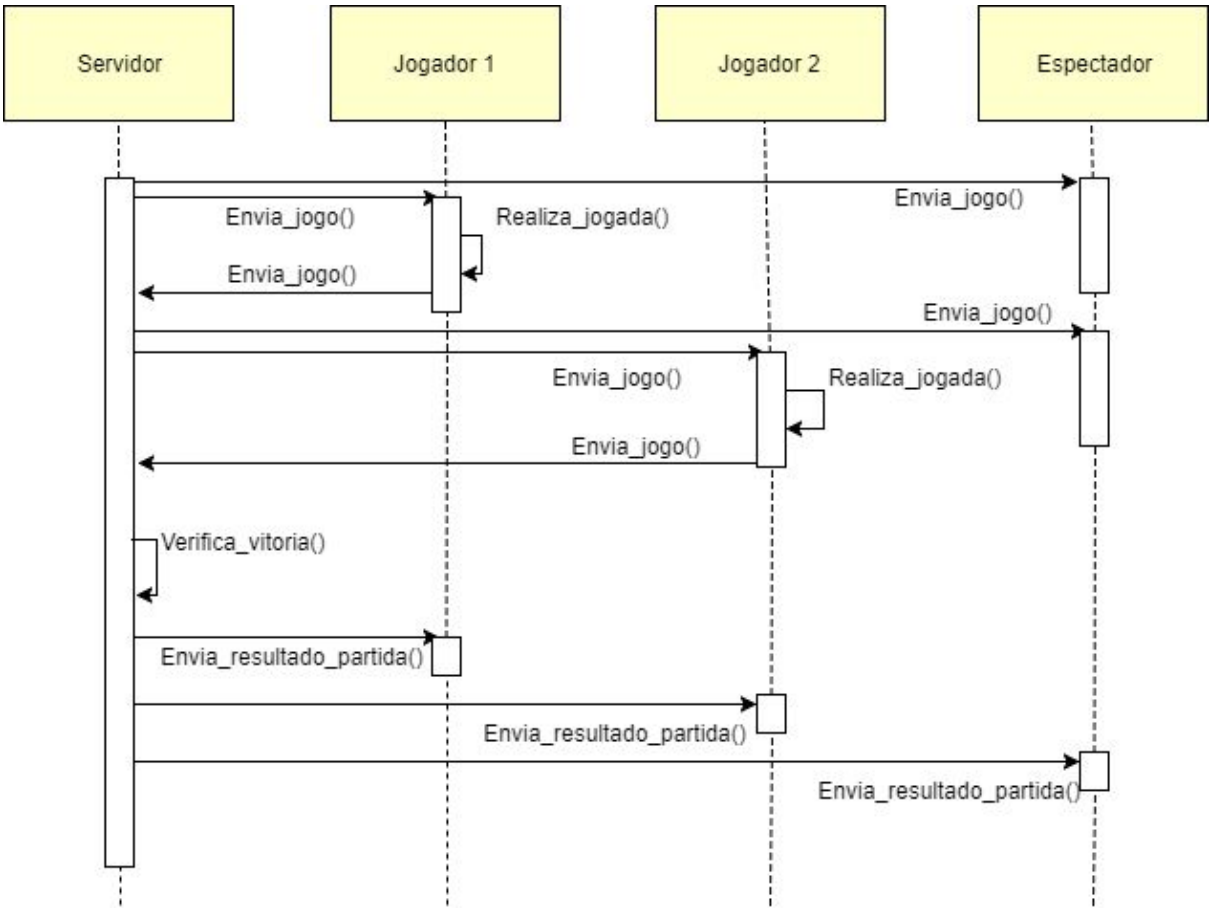
Partida de Filomilitar:

No início de cada turno o jogador deve decidir se deseja Gerar Um Conhecimento ou Desenvolver Um Conhecimento.

Se decidir em Gerar um conhecimento, o jogador deve escolher o tipo do conhecimento, entre Filosófico ou Militar. Logo após deve escolher o número de pessoas para compor o conhecimento, assim decidindo sua dificuldade. Por fim deve ser escolhido o número de especialistas para desenvolver o conhecimento, sejam Filósofos ou Militares.

Se decidir por Desenvolver Um Conhecimento, o jogador deve logo em seguida escolher qual conhecimento deseja desenvolver.

Ganha o jogador que chegar em 50 pontos primeiro.



3. Manual de uso da aplicação

A aplicação do Filo Militar foi desenvolvida em Python 3.5.1.

Os arquivos que compõem a aplicação estão listados no final do relatório e contidos na pasta *code*.

Instruções para execução do jogo em um sistema operacional Ubuntu ou Mac OS:

1. Abrir o terminal na pasta *code*
2. Executar o servidor com o comando python3 servidor.py
3. Abrir um terminal na pasta *code*
4. Executar o primeiro jogador com o comando python3 cli_at.py
5. Repetir as etapas 3 e 4 para o segundo jogador
6. Abrir um terminal na pasta *code*
7. Executar o espectador com o comando python3 cli_pas.py
8. As informações dos comandos do jogo sao enviadas ao cliente em cada turno
9. Assim que um jogador conseguir 50 pontos o servidor enviará as mensagens “vitória”, “derrota” e “Jogador com mais pontos ganhou o jogo.” para o vencedor, o derrotado e os espectadores respectivamente

3. Código fonte

servidor.py

```
host = "127.0.0.1"          # Set the server address to variable host
port = 4475            # Sets the variable port to 4444
from socket import *   # Imports socket module
from threading import Thread
import pickle
from classes.game import *
from classes.player import Player

import sys



players = []
espectadores = []

p0_nome = None
p1_nome = None


class Th(Thread):

      def __init__ (self, m_socket):
            Thread.__init__(self)
            self.m_socket = m_socket

            # criar salas depois
            # self.salas = {}

            self.players = []
            self.espectadores = []

      def run(self):
            while True:
                  # Accepts incoming request from client and returns
                  # socket and address to variables client and
Informations
                  client, Informations = self.m_socket.accept()

                  tipo = client.recv(1024) # mudar valor depois
                  # tipo = pickle.loads(b"".join(tipo))
```

```
                tipo = tipo.decode()

                print("{} conectado [ {} ]".format(tipo,
Informations))

                if tipo == "player" and len(self.players) < 2:
                      self.players.append(client)

                elif tipo == "espectador":
                      self.espectadores.append(client)


                # print(self.players)
                # print(self.espectadores)

           self.m_socket.close()


s = socket(AF_INET, SOCK_STREAM)

s.bind((host,port))    # Binds the socket. Note that the input to
                           # the bind function is a tuple
s.listen(3)            # Sets socket to listening state with a  queue
                           # of 1 connection
print ("Listening for connections.. ")


server_thread = Th(s)
server_thread.start()

# pega os nomes dos jogadores
def PlayersNomes(players):

      # pergunta_nome = "Escolha um nome.".encode()
      pergunta_nome = pickle.dumps("Escolha um nome.")

      players[0].send(pergunta_nome)
      p0_nome = players[0].recv(1024)
      # p0_nome = p0_nome.decode()
      p0_nome = pickle.loads(p0_nome)

      players[1].send(pergunta_nome)
      p1_nome = players[1].recv(1024)
      # p1_nome = p1_nome.decode()
      p1_nome = pickle.loads(p1_nome)


      return p0_nome, p1_nome
```

```
# instancia os objetos para o jogo iniciar
def GameStart(players):

      # if len(players) != 2:
      #     return

      p0_nome, p1_nome = PlayersNomes(players)

      # Valores que eu coloquei
      n_inicial_pessoas = 20
      n_pessoas_por_nacao = n_inicial_pessoas / 2

      player_0 = Player(p0_nome, n_pessoas_por_nacao, 5, 5)
      player_1 = Player(p1_nome, n_pessoas_por_nacao, 5, 5)

      jogo = Game(player_0, player_1, n_inicial_pessoas)

      jogo.StartGame()

      return jogo

# envia e espera o jogo do jogador da rodada
def Turno(player, jogo, espectadores_ativos):

      jogo_serializado = pickle.dumps(jogo)
      player.sendall(jogo_serializado)

      for espectador in espectadores_ativos:
            espectador.sendall(jogo_serializado)

      jogo_recebido = player.recv(4096)

      # chunks = []
      # bytes_recd = 0
      # MSGLEN = 4096
      # while bytes_recd < MSGLEN:
      #     chunk = player.recv(min(MSGLEN - bytes_recd, 4096))
      #     if chunk == b'':
      #           raise RuntimeError("socket connection broken")
      #     chunks.append(chunk)
      #     bytes_recd = bytes_recd + len(chunk)
      # jogo_recebido = b''.join(chunks)

      jogo_recebido = pickle.loads(jogo_recebido)
      print(jogo_recebido)
```

```
      return jogo_recebido

# nao deveria estar aqui, mas to sem tempo
# se houver vitoria retorna True, caso contrario retorna False
def ChecarVitoria(jogo, players, espectadores):
      # checa se alguem ganhou, fez 20 pontos

      if jogo.p0.pontos >= 50:
            jogo.player_turno = 666
            print("{} ganhou o jogo.".format(jogo.p0.nome))
            for espectador in espectadores:
                  espectador.sendall(pickle.dumps("Jogador com mais
pontos ganhou o jogo."))
            players[0].sendall(pickle.dumps("vitoria"))
            players[1].sendall(pickle.dumps("derrota"))
            return True
      elif jogo.p1.pontos >= 50:
            jogo.player_turno = 666
            print("{} ganhou o jogo.".format(jogo.p1.nome))
            for espectador in espectadores:
                  espectador.sendall(pickle.dumps("Jogador com mais
pontos ganhou o jogo."))
            players[0].sendall(pickle.dumps("derrota"))
            players[1].sendall(pickle.dumps("vitoria"))
            return True

      return False


# objeto de jogo
jogo = None

# loop do jogo
while True:

      players = server_thread.players
      espectadores = server_thread.espectadores

      # se jogo nao comecou
      if jogo == None and len(players) == 2:
            jogo = GameStart(players)

      elif jogo != None and len(players) == 2:

            # se alguem ganhou, volta no loop e nao chama nenhum
turno, fica dando continue
            if ChecarVitoria(jogo, players, espectadores):
```

```
                break

            print("Turno Player 0")

            jogo = Turno(players[0], jogo, espectadores)


            # inp = input("Proximo turno? ")
            print("Turno Player 1")

            jogo = Turno(players[1], jogo, espectadores)


            # inp = input("Proximo turno? ")



s.close()
```

cli_at.py

```
import socket, pickle

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.connect(('localhost', 4475))

data = "player".encode()
server.send(data)

jogo = None

while True:
      # inp = input("Msg: ")
      # data = inp
      # data = pickle.dumps(data)
      # server.send(data)
      data = server.recv(4096)
      if not data:
            break
      # data = data.decode()
      data = pickle.loads(data)

      if data == "Escolha um nome.":
            nome = input(data)
            # nome = nome.encode()
            nome = pickle.dumps(nome)
            server.send(nome)

      elif data == "vitoria" or data == "derrota":
            print(data)
```

```
      else:
            jogo = data

      # print(data)

      # servidor ja mandou o jogo
      if jogo != None:
            # exibe os dados do jogo
            jogo.ExibirDados()
            # joga o turno
            jogo.TurnoJogar()

            # preparando para enviar
            data = pickle.dumps(jogo)
            # envia o jogo
            server.sendall(data)

            # exclui o jogo daqui?!

            jogo = None



server.close()
```

cli_pas.py

```
import socket, pickle

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.connect(('localhost', 4475))

data = "espectador".encode()
server.send(data)

jogo = None


while True:
      data = server.recv(4096)
      if not data:
            break
      # data = data.decode()
      data = pickle.loads(data)

      if data == "Jogador com mais pontos ganhou o jogo.":
            print(data)
      else:
            jogo = data

      # servidor ja mandou o jogo
```

```
      if jogo != None:
            # exibe os dados do jogo
            jogo.ExibirDados()

            # exclui o jogo daqui?!

            jogo = None


server.close()
```

game.py

```
from .player import Player
from .pessoa import Pessoa
from random import randint


class Game():
      """docstring for Game"""

      def __init__(self, p0, p1, n_inicial_pessoas):

            self.p0 = p0
            self.p1 = p1
            self.n_inicial_pessoas = n_inicial_pessoas
            self.n_pessoas = n_inicial_pessoas # caso pessoas morram,
pessoas nao podem morrer por enquanto, mas vou deixar aqui pq vai que
um dia pode, e pessoas devem morrer as vezes pro equilibrio de toda
energia universal, flw vlw

            self.ano = 0

            # 0 para p0 e 1 para p1
            self.player_turno = 0


      def ExibirDados(self):
            print("_____________________________")
            print("Ano {}".format(self.ano))
            print("_____________________________")

            self.p0.ExibirDados()
            self.p1.ExibirDados()

      # vai ficar pra segunda versao do jogo
      def TrocaDeNacao(self, de, para, idp, pessoa):

            # da nacao 0 para nacao 1 (nacao = player)
            if de == 0 and para == 1:
                  if self.p0.n_pessoa > 0:
                        # tirar pessoa da nacao 0
```

```
                        self.p0.RemovePessoa(idp)
                        # add pessoa na nacao 1
                        self.p1.AddPessoa(idp, pessoa)
                  else:
                        print("{} ganhou o jogo".format(self.p1.nome))
                        self.player_turno = 666

            if de == 1 and para == 0:
                  if self.p1.n_pessoa > 0:
                        # tirar pessoa da nacao 0
                        self.p0.AddPessoa(idp, pessoa)
                        # add pessoa na nacao 1
                        self.p1.RemovePessoa(idp)
                  else:
                        print("{} ganhou o jogo".format(self.p0.nome))
                        self.player_turno = 666


      # def CriarPlayer(self, player, nome, pessoas, filosofos,
militares ):
      #     # Player 0
      #     if player == 0:
      #           self.p0 = Player(nome, pessoas, filosofos,
militares)

      #     # Player 1
      #     if player == 1:
      #           self.p1 = Player(nome, pessoas, filosofos,
militares)


      def CriarPessoa(self, idp, nacao, game):

            sexo = randint(0, 1)
            if sexo == 0:
                  sexo = "masculino"
            else:
                  sexo = "feminino"

            # nivel filosofo eh random e nivel militar eh o
complemento para 100
            nivel_filosofo = randint(0,100)
            nivel_militar = 100 - nivel_filosofo

            p = Pessoa(idp, sexo, nivel_filosofo, nivel_militar,
nacao, game)

            return p

      def GerarConhecidos(self, idp, n_pessoas_por_nacao):
            conhecidos = []

            n_conhecidos = randint(2,5)
```

```
            for x in range(0,n_conhecidos):

                  # gera o id do conhecido, se for o mesmo id da
pessoa, gero um novo
                  id_do_conhecido = randint(1, n_pessoas_por_nacao)
                  while id_do_conhecido == idp:
                        id_do_conhecido = randint(1,
n_pessoas_por_nacao)

                  # Se for do player 1, faz id ficar negativo
                  if idp < 0:
                        id_do_conhecido = id_do_conhecido * (-1)

                  conhecidos.append(id_do_conhecido) # adicionando o
id do conhecido na lista

            # adicionando um conhecido de outra nacao (do outro
player)
            id_do_conhecido = randint(1, n_pessoas_por_nacao)
            if idp > 0:
                  id_do_conhecido = id_do_conhecido * (-1)

            conhecidos.append(id_do_conhecido)


            return conhecidos


      def CriarPessoas(self):

            # se tem numero impar de pessoas, adiona mais um
            if self.n_inicial_pessoas % 2 != 0:
                  self.n_inicial_pessoas = self.n_inicial_pessoas + 1


            n_pessoas_por_nacao = int(self.n_inicial_pessoas / 2) #
numero de pessoas por nacao

            # id das pessoas e dicionario, _0 = player 1 e _1 = player
2
            pessoas_0 = {}
            pessoas_1 = {}

            id_pessoa_0 = 1
            id_pessoa_1 = -1
            for x in range(0, n_pessoas_por_nacao):

                  p0 = self.CriarPessoa(id_pessoa_0, 0, self) # cria a
pessoa
                  pessoas_0[id_pessoa_0] = p0 # adiciona pessoa na
lista com key = o id
```

```
                p1 = self.CriarPessoa(id_pessoa_1, 1, self) # cria a
pessoa
                pessoas_1[id_pessoa_1] = p1 # adiciona pessoa na
lista com key = o id

                # criando lista de conhecidos
                conhecidos_0 = self.GerarConhecidos(id_pessoa_0,
n_pessoas_por_nacao)
                conhecidos_1 = self.GerarConhecidos(id_pessoa_1,
n_pessoas_por_nacao)

                # Adicionando os conhecidos nas pessoas
                # pra ficar simples, nao to fazendo: se pessoa A
conhece pessoa B, pessoa B conhece pessoa A.
                # posso fazer depois, fazer um grafo de toda a
populacao
                p0.AdicionarConhecidos(conhecidos_0)
                p1.AdicionarConhecidos(conhecidos_1)


                # modificando o id das proximas pessoas
                id_pessoa_0 = id_pessoa_0 + 1
                id_pessoa_1 = id_pessoa_1 - 1

          # adicionando as pessoas nos players

          self.p0.AddPessoas(pessoas_0)
          self.p1.AddPessoas(pessoas_1)

          # print(self.p0.pessoas)
          # print(self.p1.pessoas)

          # print("pessoas criadas")
          # print(pessoas_0)



     def RealizarAcao(self, acao, player, tipo_conhecimento=None,
n_pessoas_conhecimento=None, n_ru=None, n_conhecimento=None):

          # Gerar Conhecimento
          if acao == "gc":
               player.CriarConhecimento(tipo_conhecimento,
n_pessoas_conhecimento, n_ru)

          elif acao == "dc": # desenvolver conhecimento
               player.DesenvolverConhecimento(n_conhecimento)


     def StartGame(self):
          self.CriarPessoas()

     def TurnoJogar(self):
```

```
            # controlador de turnos
            if self.player_turno == 0:
                  player = self.p0
                  self.player_turno = 1
            elif self.player_turno == 1:
                  player = self.p1
                  self.player_turno = 0
                  # Finalizando o turno aqui pois devo esperar os dois
jogadores jogar
                  self.FinalizarTurno()
            else:
                  # nao tem mais turno
                  print("Jogo finalizado, por favor, feche a conexao
:)")
                  inp = input("Tchau.")


            print("{} turno".format(player.nome))
            print("Acoes:")
            print("gc = Gerar conhecimento, logo apos voce deve
escolher o tipo do conhecimento, numero de pessoas envolvidas no
conhecimento e o numero de recurso especializado utilizado no
conhecimento")
            print("dc = Desenvolve conhecimento, logo apos voce deve
escolher o conhecimento desejado")
            comando = input("Escolha sua acao ")

            while comando != "gc" and comando != "dc":
                  # print(comando)
                  comando = input("Escolha sua acao: [gc] [dc] ")

            if comando == "dc" and len(player.conhecimentos) <= 0:
                  comando = "gc" # :)

            # se for pra DESENVOLVER CONHECIMENTO, pede os outros
dados
            if comando == "dc":
                  n_conhecimento = input("Escolha o numero do
conhecimento: min 0, max {} ".format(len(player.conhecimentos) - 1))
                  n_conhecimento = int(n_conhecimento)

                  # convertendo para passar para as funcoes
                  while n_conhecimento < 0 or n_conhecimento >
len(player.conhecimentos) - 1:
                        n_conhecimento = input("Escolha o numero do
conhecimento: min 0, max {} ".format(len(player.conhecimentos) - 1))
                        n_conhecimento = int(n_conhecimento)


                  # n_conhecimento = int(n_conhecimento)
                  self.RealizarAcao(comando, player,
n_conhecimento=n_conhecimento)
```

```
            # se for pra GERAR CONHECIMENTO, pede os outros dados
            elif comando == "gc":
                  tipo_conhecimento = input("Escolha o tipo de
conhecimento: [f] para filosofo ou [m] para militar ")
                  while tipo_conhecimento != "f" and tipo_conhecimento
!= "m":
                        tipo_conhecimento = input("Escolha o tipo de
conhecimento: [f] para filosofo ou [m] para militar ")

                  # poderia estar melhor, mas o tempo grita
                  n_pessoas_conhecimento = input("Escolha qual a
dificuldade do seu conhecimento, numero de pessoas envolvidas: min 1,
max {} ".format(player.n_pessoa))
                  n_pessoas_conhecimento = int(n_pessoas_conhecimento)
                  while n_pessoas_conhecimento < 1 or
n_pessoas_conhecimento > player.n_pessoa:
                        n_pessoas_conhecimento = input("Escolha qual a
dificuldade do seu conhecimento, numero de pessoas envolvidas: min 1,
max {} ".format(player.n_pessoa))
                        n_pessoas_conhecimento =
int(n_pessoas_conhecimento)

                  n_ru = input("Escolha quantos especialistas devem
trabalhar no conhecimento: min 0, max {} ou {}
".format(player.n_filosofo, player.n_militar))
                  n_ru = int(n_ru)
                  while n_ru < 0 or (tipo_conhecimento == "f" and n_ru
> player.n_filosofo) or (tipo_conhecimento == "m" and n_ru >
player.n_militar):
                        n_ru = input("Escolha quantos especialistas
devem trabalhar no conhecimento: min 0, max {} ou {}
".format(player.n_filosofo, player.n_militar))
                        n_ru = int(n_ru)


                  # convertendo para passar para as funcoes
                  # n_pessoas_conhecimento =
int(n_pessoas_conhecimento)
                  # n_ru = int(n_ru)
                  self.RealizarAcao(comando, player,
tipo_conhecimento=tipo_conhecimento,
n_pessoas_conhecimento=n_pessoas_conhecimento, n_ru=n_ru)

            print("--------------TURNO
FINALIZADO--------------------")
```

```
      def FinalizarTurno(self):
            # TODO as pessoas tem que conversar
            self.ano = self.ano + 1
```

player.py

```
from .conhecimento import Conhecimento

class Player():
      # A classe player tem tudo o que player TEM e PODE FAZER

      def __init__(self, nome, n_pessoas, filosofos=5, militares=5):
            self.nome = nome # nome player
            self.n_filosofo = filosofos # qtd de filosofos na nacao
            self.n_militar = militares # qtd de militares na nacao
            self.n_pessoa = n_pessoas # qtd de pessoas na nacao

            self.conhecimentos = []
            self.pessoas = None # pessoas da nacao, dict

            self.taxa_desenvolvimento_anual = self.n_pessoa / 2 # 2
anos necessarios para desenvolver um conhecimento envolvendo toda
nacao, podendo ser mais rapido se o conhecimento for igual as
ideologias das pessoas

            self.pontos = 0

            self.idp_pessoas_remover = []
            self.idp_pessoas_add = []
            self.pessoas_add = []

      def RemovePessoa(self, idp):
            self.idp_pessoas_remover.append(idp)
            # del self.pessoas[idp]
            # self.n_pessoa = self.n_pessoa - 1

      def AddPessoa(self, idp, pessoa):
            self.idp_pessoas_add.append(idp)
            self.pessoas_add.append(pessoa)

            # self.pessoas[idp] = pessoa
            # self.n_pessoa = self.n_pessoa + 1
            # self.taxa_desenvolvimento_anual = self.n_pessoa / 10 #
10 anos necessarios para desenvolver um conhecimento envolvendo toda
nacao, podendo ser mais rapido se o conhecimento for igual as
ideologias das pessoas
            # self.pontos = self.pontos + 1 # 1 ponto por conquistar
uma pessoa

      def RemovePessoa2(self, idp):
            del self.pessoas[idp]
            self.n_pessoa = self.n_pessoa - 1
```

```
      def AddPessoa2(self, idp, pessoa):
            self.pessoas[idp] = pessoa
            self.n_pessoa = self.n_pessoa + 1
            self.taxa_desenvolvimento_anual = self.n_pessoa / 10 # 10
anos necessarios para desenvolver um conhecimento envolvendo toda
nacao, podendo ser mais rapido se o conhecimento for igual as
ideologias das pessoas
            self.pontos = self.pontos + 1 # 1 ponto por conquistar uma
pessoa

      def AddPessoas(self, pessoasDict):
            self.pessoas = pessoasDict


      def AddPontos(self, valor):
            self.pontos = self.pontos + valor

      # gera um conhecimento novo -> n_ru: numero de filosofos ou
militares desenvolvendo o conhecimento
      def CriarConhecimento(self, tipo_conhecimento,
n_pessoas_conhecimento, n_ru):

            # se tipo_conhecimento for diferente de F (filosofo) e de
M (militar) -> tentar arrumar
            if tipo_conhecimento != "f" and tipo_conhecimento != "m":
                  tipo_conhecimento = input("Digite f (filosofo) ou m
(militar) para o tipo do conhecimento")

            # checar numero de pessoas, se for maior que pessoas na
nacao, usa o numero total de pessoas na nacao
            if n_pessoas_conhecimento > self.n_pessoa:
                  n_pessoas_conhecimento = self.n_pessoa

            # checar numero de RU, se for maior que RU na nacao, usa o
numero total de RU na nacao
            if tipo_conhecimento == "f" and n_ru > self.n_filosofo:
                  n_ru = self.n_filosofo
            if tipo_conhecimento == "m" and n_ru > self.n_militar:
                  n_ru = self.n_militar

            c = Conhecimento(tipo_conhecimento,
n_pessoas_conhecimento, n_ru)
            self.conhecimentos.append(c) # conhecimento criado e
adicionado


      def DesenvolverConhecimento(self, n_conhecimento):

            # se numero do conhecimento nao existe, desenvolve o
primeiro da lista
            if n_conhecimento < 0 or n_conhecimento >=
len(self.conhecimentos):
```

```
                n_conhecimento = 0

            # conhecimento que deve ser desenvolvido
            conhecimento = self.conhecimentos[n_conhecimento]

            # print("pessoas----------")
            # type(self.pessoas)
            # print("pessoas----------")

            conhecimento.DesenvolverConhecimento(self.pessoas,
self.taxa_desenvolvimento_anual, self)

            # para a segunda versao do jogo
            # for idp, pessoa in self.pessoas.items():
            #     pessoa.ConversarComConhecidos()

            # for idp in self.idp_pessoas_remover:
            #     self.RemovePessoa2(idp)

            # for i in range(0, len(self.idp_pessoas_add)):
            #     self.AddPessoa2(self.idp_pessoas_add[i],
self.pessoas_add[i])

            # pode ser melhorado

      def ExibirDados(self):
            print("Jogador {} -------- pontos: {}".format(self.nome,
self.pontos))
            print("numero de filosofos: {} - numero de militares:
{}".format(self.n_filosofo, self.n_militar))
            print("numero de pessoas: {}".format(self.n_pessoa))
            print("taxa de desenvolvimento anual:
{}".format(self.taxa_desenvolvimento_anual))
            print("Conhecimentos: {}".format(len(self.conhecimentos)))

            for conhecimento in self.conhecimentos:
                  conhecimento.ExibirDados()

            print("____________________")
```

pessoa.py

```
from threading import Thread

class Pessoa():
      """docstring for Pessoa"""

      def __init__(self, id_pessoa, sexo, nivel_filosofo,
nivel_militar, nacao, game):

            self.id_pessoa = id_pessoa
            self.sexo = sexo
```

```
            self.nivel_filosofo = nivel_filosofo
            self.nivel_militar = nivel_militar

            self.game = game

            self.nacao = nacao

            self.conhecidos = None # lista de conhecidos

            self.conhecimentos = None # de inicio as pessoas nao
ajudou em nenhum conhecimento ainda


      def AdicionarConhecidos(self, conhecidos):
            self.conhecidos = conhecidos

      def Ideologia(self):

            if self.nivel_filosofo >= self.nivel_militar:
                  return "filosofo"
            else :
                  return "militar"

      def MelhorarFilosofia(self, valor):
            # Se nivel militar ainda existe, tira um e bota em
filosofo
            if pA.nivel_militar > 0:
                  pA.nivel_filosofo = pA.nivel_filosofo + 1
                  pA.nivel_militar = pA.nivel_militar - 1
            if pB.nivel_militar > 0:
                  pB.nivel_filosofo = pB.nivel_filosofo + 1
                  pB.nivel_militar = pB.nivel_militar - 1


      def Conversa_MesmaNacao_MesmaIdeologia(self, pA, pB):

            pA_Ideologia = pA.Ideologia()
            pB_Ideologia = pB.Ideologia()

            # Verificando Ideologia e Nacao
            if pA_Ideologia == pB_Ideologia and pA.nacao == pB.nacao:

                  if pA_Ideologia == "filosofo":

                        # Se nivel militar ainda existe, tira um e bota
em filosofo
                        if pA.nivel_militar > 0:
                              pA.nivel_filosofo = pA.nivel_filosofo + 1
                              pA.nivel_militar = pA.nivel_militar - 1
                        if pB.nivel_militar > 0:
                              pB.nivel_filosofo = pB.nivel_filosofo + 1
                              pB.nivel_militar = pB.nivel_militar - 1
```

```
                if pA_Ideologia == "militar":

                      # Se nivel filosofo ainda existe, tira um e
bota em militar
                      if pA.nivel_filosofo > 0:
                            pA.nivel_militar = pA.nivel_militar + 1
                            pA.nivel_filosofo = pA.nivel_filosofo - 1
                      if pB.nivel_filosofo > 0:
                            pB.nivel_militar = pB.nivel_militar + 1
                            pB.nivel_filosofo = pB.nivel_filosofo - 1


                return True # se foi verdade

           return False # se nao foi verdade :)

     def Conversa_MesmaIdeologia_NacaoDiferente(self, pA, pB):

           pA_Ideologia = pA.Ideologia()
           pB_Ideologia = pB.Ideologia()

           # Verificando Ideologia e Nacao
           if pA_Ideologia == pB_Ideologia and pA.nacao != pB.nacao:

                if pA_Ideologia == "filosofo":

                      # Quem for mais "fraco" troca de nacao
                      if pA.nivel_filosofo > pB.nivel_filosofo:
                            self.game.TrocaDeNacao(pB.nacao,
pA.nacao, pB.id_pessoa, pB)
                            pB.nacao = pA.nacao
                      elif pB.nivel_filosofo > pA.nivel_filosofo:
                            self.game.TrocaDeNacao(pA.nacao,
pB.nacao, pA.id_pessoa, pA)
                            pA.nacao = pB.nacao

                if pA_Ideologia == "militar":

                      # Quem for mais "fraco" troca de nacao
                      if pA.nivel_militar > pB.nivel_militar:
                            self.game.TrocaDeNacao(pB.nacao,
pA.nacao, pB.id_pessoa, pB)
                            pB.nacao = pA.nacao
                      elif pB.nivel_militar > pA.nivel_militar:
                            self.game.TrocaDeNacao(pA.nacao,
pB.nacao, pA.id_pessoa, pA)
                            pA.nacao = pB.nacao


                return True # se foi verdade

           return False # se nao foi verdade :)
```

```
      def Conversa_IdeologiaDiferente(self, pA, pB):

            pA_Ideologia = pA.Ideologia()
            pB_Ideologia = pB.Ideologia()

            # Verificando Ideologia
            if pA_Ideologia != pB_Ideologia:
                  # diferenca entre as ideologias de cada pessoa
                  dif_entre_ideologias_pA = abs(pA.nivel_militar -
pA.nivel_filosofo)
                  dif_entre_ideologias_pB = abs(pB.nivel_militar -
pB.nivel_filosofo)

                  # Se Dif de A for maior que de B -e> A eh filosofo
                  if dif_entre_ideologias_pA >
dif_entre_ideologias_pB:

                        if pA_Ideologia == "filosofo":

                              # Se nivel militar ainda existe, tira 2 e
bota em filosofo
                              if pA.nivel_militar > 1:
                                    pA.nivel_filosofo =
pA.nivel_filosofo + 2
                                    pA.nivel_militar = pA.nivel_militar
- 2
                              if pB.nivel_militar > 1:
                                    pB.nivel_filosofo =
pB.nivel_filosofo + 2
                                    pB.nivel_militar = pB.nivel_militar
- 2

                        if pA_Ideologia == "militar":
                              # diferenca entre a diferenca das pessoas
                              dif_entre_pessoas =
abs(dif_entre_ideologias_pA - dif_entre_ideologias_pB)

                              # esse numero pode mudar, ele que define
quao diferente tem que ser pra modificar a ideologia da pessoa (30)
                              if dif_entre_pessoas > 30:

                                    # Se nivel filosofo ainda existe,
tira um e bota em militar
                                    if pA.nivel_filosofo > 1:
                                          pA.nivel_militar =
pA.nivel_militar + 2
                                          pA.nivel_filosofo =
pA.nivel_filosofo - 2
                                    if pB.nivel_filosofo > 1:
                                          pB.nivel_militar =
pB.nivel_militar + 2
                                          pB.nivel_filosofo =
```

```
pB.nivel_filosofo - 2
                              else :

                                    # Se nivel militar ainda existe,
tira 2 e bota em filosofo
                                    if pA.nivel_militar > 1:
                                          pA.nivel_filosofo =
pA.nivel_filosofo + 2
                                          pA.nivel_militar =
pA.nivel_militar - 2
                                    if pB.nivel_militar > 1:
                                          pB.nivel_filosofo =
pB.nivel_filosofo + 2
                                          pB.nivel_militar =
pB.nivel_militar - 2



                  return True

            return False


      def ConversarComConhecidos(self):

            # interando sobre os conhecidos, id dos conhecidos
            for idp in self.conhecidos:

                  try:
                        # tenta pegar a pessoa do player 0
                        pessoaB = self.game.p0.pessoas[idp]
                  except Exception as e:
                        # tenta pegar a pessoa do player 1
                        pessoaB = self.game.p1.pessoas[idp]


                  # se sao da mesma nacao
                  if self.Conversa_MesmaNacao_MesmaIdeologia(self,
pessoaB):
                        continue
                  elif
self.Conversa_MesmaIdeologia_NacaoDiferente(self, pessoaB):
                        continue
                  elif self.Conversa_IdeologiaDiferente(self,
pessoaB):
                        continue
```

conhecimento.py

```
class Conhecimento():
```

```
      def __init__(self, tipo_conhecimento,
valor_inicial_conhecimento, n_ru):

            self.tipo_conhecimento = tipo_conhecimento #
            self.n_ru = n_ru # numero de filosofos ou militares
            self.lista_conhecimento = [valor_inicial_conhecimento] #
valor inicial do conhecimento eh o valor inicial da lista, mesmo
valor de pessoas que "ajudaram no conhecimento"


      # usa as pessoas para desenvolver o conhecimento, dependendo do
nivel para o tipo de conhecimento, o desenvolvimento pode ficar mais
rapido ou mais lento
      def DesenvolverConhecimento(self, pessoas,
taxa_desenvolvimento_anual, player):

            # se conhecimento ainda nao foi totalmente desenvolvido
            if not (self.lista_conhecimento[-1] == 0):
                  # loop de 1 ate a taxa de desenvolvimente + 1 -> id
das pessoas
                  x = 0
                  # n_ru pois so de trabalhar no desenvolvimento do
conhecimento ele ja eh atualizado
                  # n_desenvolvimento = self.n_ru
                  n_desenvolvimento = 0

                  # print("pessoas----------")
                  # print(pessoas)
                  # print("pessoas----------")

                  for idp, pessoa in pessoas.items():

                        # 1 se a pessoa nao for da mesma ideologia do
conhecimento, 2 se for
                        n_desenvolvimento_atual = 1

                        if pessoa.Ideologia() == "filosofo" and
self.tipo_conhecimento == "f":
                              n_desenvolvimento_atual = 2

                        if pessoa.Ideologia() == "militar" and
self.tipo_conhecimento == "m":
                              n_desenvolvimento_atual = 2

                        # quantidade para desenvolver conhecimento
                        n_desenvolvimento = n_desenvolvimento +
n_desenvolvimento_atual

                        # quando X for igual taxa de desenvolvimento
devo sair
                        # da pra salvar o X e continuar de onde parou
na proxima, chegando no maximo eu preciso voltar pro 0, nao vou fazer
```

```
agora por preguica
                        x = x + 1
                        if x == taxa_desenvolvimento_anual:
                              break

                  # se o n_desen for maior que a diferenca entre o
ultimo elemento da lista e 0
                  # n_desen fica como a diferenca
                  if (self.lista_conhecimento[-1] - n_desenvolvimento)
< 0:
                        n_desenvolvimento = abs(0 -
self.lista_conhecimento[-1])

                  # n_desen = diferenca entre o ultimo elemento da
lista e n_desen antigo
                  n_desenvolvimento = self.lista_conhecimento[-1] -
n_desenvolvimento
                  self.lista_conhecimento.append(n_desenvolvimento)

                  # se desenvolveu tudo, player ganha numero de pontos
igual ao valor inicial do conhecimento (dificiuldade do memso)
                  if n_desenvolvimento == 0:
                        player.AddPontos(self.lista_conhecimento[0])

      def ExibirDados(self):
            print("Tipo do conhecimento:
{}".format(self.tipo_conhecimento))
            print("Quantidade de RU: {}".format(self.n_ru))
            print("Representacao do conhecimento:
{}".format(self.lista_conhecimento))
```