UNIVERSIDADE FEDERAL DO PARANÁ
CURSO DE MATEMÁTICA

MARINA SAYURI VIEIRA
MONIQUE BAPTISTA FRAGOZO
OTÁVIO DITTRICH MOREIRA

JOGO SUDOKU

CURITIBA
2018

MARINA SAYURI VIERIA (GRR20185652)
MONIQUE BAPTISTA FRAGOZO (GRR20185650)
OTÁVIO DITTRICH MOREIRA (GRR20185659)

JOGO SUDOKU

Relatório apresentado à disciplina Fundamentos de Programação de Computadores do Curso de Graduação em Matemática da Universidade Federal do Paraná.

Orientador: Prof. Jackson Antônio do Prado Lima

CURITIBA, NOVEMBRO DE 2018

SUMÁRIO

1. INTRODUÇÃO

Neste trabalho adaptamos o tradicional jogo Sudoku para a linguagem Python. Para tal, utilizamos os conteúdos aprendidos durante o semestre, tais como funções, arquivos, listas, além de diversas bibliotecas que auxiliam na jogabilidade.

O Sudoku é um jogo de raciocínio lógico no qual, numa grade nxn dividida em n subgrades de tamanho menor, devem ser posicionados os números de 1 até n, sem que nenhum desses números se repita em nenhuma linha, coluna ou subgrade.

2. OBJETIVOS

O trabalho visa buscar o entretenimento do usuário, apresentando um jogo desafiador de forma a agradar desde os mais experientes no jogo, novatos, e até mesmo os que fogem de puzzle que exigem raciocínio lógico, mostrando que eles podem ser (e são) muito divertidos.

O jogo traz duas opções para o usuário:

1. o programa resolve o sudoku inserido pelo usuário.
2. o usuário resolve um sudoku gerado pelo programa.

3. DESENVOLVIMENTO

Ao longo do programa, foram utilizadas 14 funções.

3.1 FUNÇÕES

3.1.1 FUNÇÕES LINHA(n)/ COLUNA(n)/ QUADRADO(n)

As funções *linha(n), coluna(n)* e *quadrado(n)* criam uma matriz com listas que denotam a posição de cada linha, coluna e quadrado, e em cada lista os respectivos elementos de cada um.

```
def linha(n): #linha(n) é a lista com as posições de todas as linhas de um sudoku nxn
    linha = []
    for k in range(n): #fórmula da k-ésima linha
        l = []
        for p in range(n): #p é cada elemento da k-ésima linha
            l += [(k*n) + p] #deduzimos a fórmula para cada elemento da k-ésima linha
        linha += [l]
    return linha

def coluna(n): #coluna(n) é a lista com as posições de todas as colunas de um sudoku nxn
    coluna = []
    for k in range(n): #fórmula da k-ésima coluna
        c = []
        for p in range(n): #p é cada elemento da k-ésima coluna
            c += [k + (p*n)] #deduzimos essa fórmula pra cada elemento da k-ésima coluna
        coluna += [c]
    return coluna

def quadrado(n): #quadrado(n) é a lista com as posições de todos os quadrados de um sudoku nxn
    quadrado = []
    j = int(n**(1/2)) #porque o sudoku terá sempre j "blocos de linhas"
    for a in range(j): #pegamos o "bloco de linha"
        for k in range(j): #pegamos os quadrados desses "blocos de linhas"
            q = []
            prim_el = (j**3)*a + j*k #fórmula encontrada para o primeiro elemento do k-ésimo quadrado
            for p in range(j): #pegamos a linha do quadrado
                l = [] #lista vazia para cada linha p do quadrado
                for i in range(j): #i é cada elemento do quadrado
                    l += [prim_el + i] #a linha será sempre o primeiro elemento + i
                prim_el += n #para encontrar o primeiro elemento da próxima linha
                q += l
            quadrado += [q]
    return quadrado
```

Figura 1- Funções *linha(n), coluna(n) e quadrado (n)*

3.1.2 FUNÇÃO OP(sudoku)

A função *op(sudoku)* vai verificar cada entrada do sudoku e validar quais as opções de preenchimento, armazenando-as em uma lista. Para tal, verificaremos os elementos que estão na mesma linha, coluna e quadrado (para isso as funções anteriores *linha(n), coluna(n)* e *quadrado(n)*).

Não existir opção para uma casa significa que ela já está preenchida, e por convenção, utilizamos "" (aspas duplas).

A função retornará uma lista, na qual cada elemento dela é um conjunto que possui as opções de preenchimento para cada posição do sudoku. Usamos conjuntos ao invés de listas pois, nesse caso, estamos interessados em saber quais são as opções, sem ordem e sem elementos repetidos.

```
def op(sudoku):
#opcoes para cada casa do sudoku
    m = len(sudoku)
    n = int(m**(1/2))
    #vamos chamar as funções linha, coluna e quadrado
    linhaa =  linha(n)
    colunaa =  coluna(n)
    quadradoo = quadrado(n)
    i = 0
    op = [] #op = opcoes

    while i < m: #i é cada posição dos elementos do sudoku
        if sudoku[i] != "": #se não tem aspas é pq a casa já está preenchida com um numero, então não tem opção nenhuma
            op += [""] #aspas pq é convenção
        else:
            usados = set() #usados será o conjunto de elementos já usados na mesma linha, coluna e quadrado de cada elemento
        #set() é conjunto vazio
            l = c = q = 0
        #l é qual linha o i está, c coluna e q quadrado
            while not(i in linhaa[l]): #queremos achar a linha em que o i está
                l+=1
            while not(i in colunaa[c]): #queremos achar a coluna em que o i está
                c+=1
            while not(i in quadradoo[q]): #queremos achar a quadrado em que o i está
                q+=1
            for k in range(n): #esse for serve para variar os elementos de cada linha, coluna e quadrado que o elemento está
                s=sudoku[linhaa[l][k]] #encontramos o elemento que está na posição linhaa[l][k]
                usados.add(s)
                usados.add(sudoku[colunaa[c][k]])#encontramos o elemento que está na posição colunaa[l][k]
                usados.add(sudoku[quadradoo[q][k]])#encontramos o elemento que está na posição quadradoo[l][k]
            u={str(i+1) for i in range(n)} #todas as possibilidades para preencher uma casa (sem contar outras casas já preenchidas)
            #{'1','2',...,'n'}
            op_i = u (usados) #op_i = opções para cada entrada i
            op += [op_i] #op será uma lista com m listas, onde cada lista i terá as opções para a posição i
        i+=1
    return op
```

Figura 2- Função *op(sudoku)*

3.1.3 FUNÇÃO *QT_OP(op)*

A função *qt_op(op)* vai contar a quantidade de opções para cada entrada do sudoku. Para isso, contará a quantidade de elementos de cada lista da matriz op (definida na função anterior).

Colocaremos 1000 no lugar de "", pois futuramente precisaremos encontrar o mínimo de possibilidades, e as "" (que representam que a posição já está preenchida, ou seja, não há opções de preenchimento) vão atrapalhar.

```
def qt_op(op):
#QUANTIDADE de opções para cada entrada do sudoku
    qt = []
    for k in op:
        if k == "":
            qt+= [1000]
        else:
            qt += [len(k)] #contamos o número de opções de cada entrada k
    return qt
```

Figura 3- Função *qt_op(sudoku)*

3.1.4 FUNÇÃO VALIDACAO(sudoku)

A função validação(sudoku) verificará se o jogador inseriu um sudoku válido, pois o jogador pode inserir elementos iguais na mesma linha, coluna ou quadrado.

```
def validacao(sudoku):
    n = len(sudoku)**(1/2)
    n = int(n)
    m = (n**2)
    linhaa =  linha(n)
    colunaa =  coluna(n)
    quadradoo = quadrado(n)
    if type(sudoku) != type([]):
        return "Favor inserir um vetor!"
    num_entradas = len(sudoku)**(1/4) #o sudoku sempre deve sempre ter um quadrado perfeito de um quadrado perfeito entradas
    #o Sudoku é um quadrado cheio de quadrados dentro, então a dimensão tem que ser o quadrado de um quadrado
    #len é int mas quando tira raiz quarta fica float
    if num_entradas != int(num_entradas):
        #no python, 3.0 == 3
        return "Número de entradas não corresponde a de um sudoku."
    possibilidade_de_entradas = [""]
    for a in range(1, n+1): #vamos verificar se a entrada é válida
        possibilidade_de_entradas.append(str(a)) #possibilidade_de_entradas = ["","1","2",...,"n"]
    #for k in sudoku:
    #    if k not in possibilidade_de_entradas:
    #        return "Entrada inválida!" #se não é "" ou se não é nenhum número no range(1, n+1), a entrada é inválida
    for k in range(n): #agora, vamos ver se há dois num iguais na mesma linha/coluna/quadrado
        col = []
        li = []
        qua = []
        for j in range(n):
            li += [sudoku[linhaa[k][j]]] #elemento da pos linhaa[k][j]
            col += [sudoku[colunaa[k][j]]]#elemento da pos colunaa[k][j]
            qua += [sudoku[quadradoo[k][j]]]#elemento da pos quadradoo[k][j]

        for i in li:
            if i!="" and i!="#":
                if li.count(i) > 1:
                    return """Sudoku inválido.
Dois números iguais na mesma linha!"""
        for i in col:
            if i!="":
                if col.count(i) > 1:
                    return """Sudoku inválido.
Dois números iguais na mesma coluna!"""
        for i in qua:
            if i!="":
                if qua.count(i) > 1:
                    return """Sudoku inválido.
Dois números iguais no mesmo quadrado!"""
    return "Sudoku válido."
```

Figura 4 - Função *validacao(sudoku)*

3.1.5 FUNÇÃO SALVE(v)

A função salve(v) foi criada para salvar um vetor e será necessária em próximos passos.

```
def salve(v): #pra não estragar o vetor
    k=[i for i in v]
    return k
```

Figura 5 - Função *salve(v)*

3.1.6 FUNÇÃO REPLACE(v, i, j)

A função replace(v, i, j) troca o i-ésimo elemento com o j-ésimo elemento de um vetor v.

```
def replace(v,i,j):
    v[i],v[j]=v[j],v[i]
```

Figura 6 - Função *replace(v, i, j)*

3.1.7 FUNÇÃO RESOLVERSUDOKU(sudoku)

Para resolver o sudoku, a função *resolversudoku(sudoku)* vai verificar se o sudoku é válido por meio da função validação(sudoku) definida anteriormente. A função vai preencher o sudoku com os elementos que na lista de *qt_op* tem “1”, ou seja, uma opção. Em seguida, pegaremos o menor dos valores de *qt_op* (por isso colocamos “1000” nas " ", para não ocorrer problemas na hora de utilizar a função min()). Com a função .index(), encontramos a posição desse mínimo e pegaremos a lista relacionada a ele em *op*. Agora com a função .pop() pegaremos um elemento dessa lista e testaremos a resolução do sudoku, validando-o novamente. Caso ele seja impossível, testaremos outro elemento, e assim por diante, até que o sudoku seja válido. Se o sudoku não for possível para nenhuma das opções, ele retornará uma mensagem dizendo que o sudoku é impossível.

```
def resolversudoku(sudoku):
    if validacao(sudoku) != "Sudoku válido.":
        return validacao(sudoku)
    opc = op(sudoku)
    qt_opc = qt_op(opc)
    if not("" in sudoku):
        return sudoku
    elif 0 in qt_opc:
        return "Sudoku impossível"
    elif 1 in qt_opc: #primeiro precisamos ver onde está o 1, pra isso serve o seguinte index:
        j = qt_opc.index(1)
        num = opc[j].pop()
        sudoku[j] = num
        if not("" in sudoku):
            return sudoku
        else:
            return resolversudoku(sudoku)
    else:
        k=salve(sudoku)
        minimo = min(qt_opc) # por isso que usamos aquele 1000
        j = qt_opc.index(minimo) #achar a posição do mínimo
        min_op = opc[j] #min_op é o conjunto com as opções da entrada com menos opções
        b = min_op.pop() #pegamos um elemento aleatório entre as opções para tentar resolver o sudoku
        min_op -= {b} #tiramos o elemento aleatorio porque, se não conseguirmos resolver o sudoku com ele, tentaremos com os outros elementos
        sudoku[j] = b #o j antes era "", agora trocamos as aspas por b, que é um cara aleatório para tentar resolver com esse elemento
        rvs = resolversudoku(sudoku)
        while type(rvs)==type("") and min_op!=set(): #set() é conjunto vazio
            b = min_op.pop()
            min_op -= {b}
            sudoku[j] = b
            rvs = resolversudoku(sudoku)
            if type(rvs)!= type([ ]):
                sudoku=salve(k)
            else:
                return rvs
        return rvs
```

Figura 7 - Função *resolver(sudoku)*

3.1.8 FUNÇÃO GETCH()

A função *getch()* serve para o usuário não precisar utilizar a tecla “enter “ a cada entrada digitada, ou seja, o programa receberá a primeira tecla digitada pelo usuário sem a necessidade de teclar “enter”. Ela serve para dar mais mobilidade ao jogo.

```
import sys, termios, tty, os, time
import fcntl

def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)

    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

button_delay = 0.2

fd = sys.stdin.fileno()
fl = fcntl.fcntl(fd, fcntl.F_GETFL)
```

Figura 8 - Função *getch()*

3.1.9 FUNÇÃO ESCREVA(v)

A função escreve(v) é necessária para dar o print do sudoku em uma tabela (como é normalmente encontrado em revistas, jornais e passatempos). Ela receberá um vetor e retorna um print de maneira conveniente. Nela é utilizado o trabalho manual bruto linha para printar linha por linha, identificando o momento que é preciso fazer uma divisão do quadrado. Por exemplo, se o sudoku é 4x4, printará primeiro o “teto” do sudoku, depois printará quatro linhas de quatro elementos, cada uma divida ao meio por |, assim como a divisória do meio de duas linhas.

```
def escreva(v):
    i=0
    lista=[]
    n=len(v)
    m=pow(n,1/2)
    q=pow(m,1/2)
    if type(v)==type(lista):
        print("┌",end="")
        l=0
        for j in range(int(q)):
            if l==0:
                print("─"*(int(q**3-1)),end="┬")
            elif l==int(q-1):
                print("─"*(int(q**3)),end="┐")
                print("")
                print("|",end="")
            else:
                print("─"*(int(q**3)),end="┬")
            l+=1
        print("",end="")#até aqui printamos o cantinho de cima
        while i<n:
            if (i+1)%q!=0:
                if v[i]==0:
                    print("#",end="  ")
                elif v[i]==-1:
                    print("_",end="  ")
                else:
                    if int(v[i])<10:
                        print(v[i],end="  ")
                    else:
                        print(v[i],end=" ")
            else:
                if v[i]==0:
                    print("#",end=" ")
                elif v[i]==-1:
                    print("_",end=" ")
                else:
                    if int(v[i])<10:
                        print(v[i],end=" ")
                    else:
                        print(v[i],end="")
            if ((i+1)/m)%q==0 and i+1!=n:
                print("|")
                print("├",end="")
                l=0
                for j in range(int(q)):
                    if l==0:
                        print("─"*(int(q**3-1)),end="┼")
                    elif l==int(q-1):
                        print("─"*(int(q**3)),end="┤")
                        print("")
                        print("|",end="")
                    else:
                        print("─"*(int(q**3)),end="┼")
                    l+=1
                print("",end="")
            elif (i+1)%m==0 and (i+1)!=n:
                print("|")
                print("|",end="")
            elif (i+1)%q==0 and i+1!=n:
                print("|",end=" ")
            i+=1
        print("|")
        print("└",end="")
        l=0
        for j in range(int(q)):
            if l==0:
                print("─"*(int(q**3-1)),end="┴")
            elif l==int(q-1):
                print("─"*(int(q**3)),end="┘")
                print("")
            else:
                print("─"*(int(q**3)),end="┴")
            l+=1
        print("",end="")
    else:
        print(v)
    print("\n")
```

Figura 9 - Função *escreva(v)*

3.1.10 FUNÇÃO JOGAR(u)

A função jogar(u) receberá o vetor Sudoku (com algumas casas preechidas), e termina quando o usuário completar todas as entradas que faltam do Sudoku. Após isso, utilizaremos novamente a função validação(sudoku) para então retornar uma mensagem (string) dizendo se o Sudoku foi digitado corretamente (sem elementos repetidos em linhas, colunas e quadrados), ou se houve algum erro.

```
def jogar(u):
    n=len(u)
    m=int(n**(1/2))
    ns=[]
    for i in range(m):
        ns+=[str(i+1)]
    i=1
    k=0
    r=[]
    k=u.index("")
    u[k]="_"
    a=123
    while True:
        os.system('cls' if os.name == 'nt' else 'clear')
        escreva(u)
        if m == 16:
            a=input()
        else:
            a=getch()
        if a=="d":
            i=1
            while u[(k+i)%n]!="" and i<n:
                i+=1
            replace(u,k,(k+i)%n)
            k=(k+i)%n
        elif a=="w":
            i=m
            while u[(k-i)%n]!="" and i<4*n:
                i+=m
            replace(u,k,(k-i)%n)
            k=(k-i)%n
        elif a=="s":
            i=m
            while u[(k+i)%n]!="" and i<4*n:
                i+=m
            replace(u,k,(k+i)%n)
            k=(k+i)%n
        elif a=="a":
            i=1
            while u[m*int(k/m)+(k-i)%m]!="" and i<n:
                i+=1
            replace(u,k,m*int(k/m)+(k-i)%m)
            k=m*int(k/m)+(k-i)%m
        elif a in ns:
            if not "" in u:
                del u[k]
                u.insert(k,int(a))
                os.system('cls' if os.name == 'nt' else 'clear')
                escreva(u)
                break
            i=k+1
            p=0
            while u[i%n]!="" and p<=n:
                i+=1
                p+=1
            replace(u,i%n,k%n)
            del u[k%n]
            u.insert(k%n,a)
            r+=[k%n]
            i,k=k,i
            k=k%n
        elif a=="r" and r!=[]:
            del u[r[-1]]
            u.insert(r[-1],"")
            del r[-1]
        elif a=="b":
            os.system('cls' if os.name == 'nt' else 'clear')
            return menu()
    if validacao(u) != "Sudoku válido.":
        return validacao(u)
    else:
        return "Parabéns! Você conseguiu!!!!"
```

Figura 10 - Função *jogar(u)*

3.1.11 FUNÇÃO DIGITEV()

A função *digitev()* serve para o usuário digitar o Sudoku. Inicialmente é apresentado um sudoku em branco (com a função *escreva()*) mas com uma entrada "#", que ele poderá movimentá-la pelo Sudoku usados as teclas ASWD. Por exemplo, se ele deseja inserir um número 3 na 6° casa, inicialmente ele deverá digitar “d” 5 vezes para se deslocar até lá, e depois digitar o número 3. Para se movimentar, o que o programa faz é trocar as entradas do vetor que o usuário está com a que ele deseja ir. A função termina quando o usuário digita x, e ela retorna o vetor associado ao sudoku inserido.

```
def digitev():
    print("\tResolver Sudoku\nDigite o tamanho do Sudoku:\n1. 4x4\n2. 9x9\n3. 16x16\n4. Voltar\n ")
    n=getch()
    if n=="1":
        n=2
    elif n=="2":
        n=3
    elif n=="3":
        n=4
    elif n=="4":
        os.system('cls' if os.name == 'nt' else 'clear')
        menu()
    else:
        os.system('cls' if os.name == 'nt' else 'clear')
        return digitev()
    n=int(n)
    m=n**2
    n=m**2
    ns=[]
    for i in range(m):
        ns+=[str(i+1)]
    i=1
    u=["_"]
    while i<n:
        u+=[""]
        i+=1
    a=12355939
    k=0
    r=[]
    escreva(u)
    while a!="x":
        os.system('cls' if os.name == 'nt' else 'clear')
        escreva(u)
        if m==16:
            a=input()
        else:
            a=getch() #input sem dar enter
        if a=="d": #se aperta d vai pra direita
        #para "andar" pelo sudoku, em geral, permutamos duas entradas
            i=1
            while u[m*int(k/m)+(k+i)%m]!="" and i<n:
                i+=1
            replace(u,k,m*int(k/m)+(k+i)%m)
            k=m*int(k/m)+(k+i)%m
        elif a=="w":  #se aperta w vai pra cima
            i=m
            while u[(k-i)%n]!="" and i<4*n:
                i+=m
            replace(u,k,(k-i)%n)
            k=(k-i)%n
        elif a=="s": #se aperta s vai pra baixo
            i=m
            while u[(k+i)%n]!="" and i<4*n:
                i+=m
            replace(u,k,(k+i)%n)
            k=(k+i)%n
        elif a=="a":  #se aperta a vai pra esquerda
            i=1
            while u[m*int(k/m)+(k-i)%m]!="" and i<n:
                i+=1
            replace(u,k,m*int(k/m)+(k-i)%m)
            k=m*int(k/m)+(k-i)%m
        elif a in ns:
            if not "" in u:
                del u[k]
                u.insert(k,int(a))
                break
            i=k+1
            p=0
            while u[i%n]!="" and p<=n:
                i+=1
                p+=1
            replace(u,i%n,k%n)
            del u[k%n]
            u.insert(k%n,a)
            r+=[k%n]
            i,k=k,i
            k=k%n
        elif a=="r" and r!=[]:
            del u[r[-1]]
            u.insert(r[-1],"")
            del r[-1]
        elif a=="b":
            os.system('cls' if os.name == 'nt' else 'clear')
            return digitev()
    del u[k%n]
    u.insert(k%n,"")
    os.system('cls' if os.name == 'nt' else 'clear')
    escreva(u)
    return u
```

Figura 11 - Função *digitev()*

3.1.12 FUNÇÃO MENU()

A função menu inicialmente apresentará as opções: jogar, resolver sudoku, tutorial e sair.

Na opção *jogar*, o usuário escolherá o tamanho do jogo: 4x4, 9x9 ou 16x16. Em seguida, o programa irá abrir o arquivo do tamanho do jogo selecionado e sorteará um dos 1000 jogos. Pela função *jogar()*, o usuário tentará resolver o sudoku. Ao final da tentativa, retornará ao menu.

Já na opção *resolver* sudoku, o usuário vai inserir o jogo e, chamando a função *resolver(sudoku),* o programa retornará a solução. Ao final, retornará ao menu.

Na opção tutorial, o usuário terá as instruções do jogo tais como as teclas e as regras do jogo.

```
def menu():
    os.system('cls' if os.name == 'nt' else 'clear')
    print(" Menu\n1. Jogar\n2. Resolver sudoku\n3. Tutorial\n4. Sair\n ")
    escolha = getch()
    if escolha == "1":
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\tJogar\nDigite o tamanho do sudoku:\n1. 4x4\n2. 9x9\n3. 16x16\n4. Voltar\n ")
        p = getch()
        while p not in {"1", "2", "3", "4"}:
            os.system('cls' if os.name == 'nt' else 'clear')
            print("\tJogar\nDigite o tamanho do sudoku:\n1. 4x4\n2. 9x9\n3. 16x16\n4. Voltar\n ")
            p = getch()
        if p == "1":
            with open('Jogo 4x4.txt','r') as f:#abrir o arquivo com os jogos 4x4
                u=random.choice(f.readlines())#sortear uma linha (jogo) para jogar
                u=u.split()
                u = [k if k!="#" else "" for k in u]
                #del u[-1]#apareceu um "\n" no final, sei la o que é, mas tiramos ne
            print(jogar(u))
        elif p == "2":
            with open('Jogo 9x9.txt','r') as f:#abrir o arquivo com os jogos 9x9
                u=random.choice(f.readlines())#sortear uma linha (jogo) para jogar
                u=u.split()
                u = [k if k!="#" else "" for k in u]
                #del u[-1]#apareceu um "\n" no final, sei la o que é, mas tiramos ne
            print(jogar(u))
        elif p == "3":
            with open('Jogo 16x16.txt','r') as f:#abrir o arquivo com os jogos 16x16
                u=random.choice(f.readlines())#sortear uma linha (jogo) para jogar
                u=u.split()
                u = [k if k!="#" else "" for k in u]
                #del u[-1]#apareceu um "\n" no final, sei la o que é, mas tiramos ne
            print(jogar(u))
        elif p == "4":
            os.system('cls' if os.name == 'nt' else 'clear')
            menu()
        a = input("\nPressione enter para voltar para o menu.")
        os.system('cls' if os.name == 'nt' else 'clear')
        menu()
    elif escolha == "2":
        os.system('cls' if os.name == 'nt' else 'clear')#apaga a tela para recomeçar
        u = digitev()
        escreva(resolversudoku(u))
        a = input("Pressione enter para voltar. ")
        menu()
    elif escolha == "3":
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\tTutorial\n\n1. Comandos:\n(a) esquerda\n(d) direita\n(w) cima\n(s) baixo\n(x) confirma\n(r) apaga\n(b) voltar\n\n2. Objetivos:\n A ideia do jogo é bem simples: No 9x9, por exemplo,\n completar todas as
        a = input("Pressione enter para voltar. ")
        menu()
    elif escolha == "4":
        os.system('cls' if os.name == 'nt' else 'clear')
        print("Tem certeza?\n1. Sim\n2. Não\n ")
        bla = getch()
        if bla == "1":
            os._exit(45)
        else:
            os.system('cls' if os.name == 'nt' else 'clear')
            menu()


    else:
        os.system('cls' if os.name == 'nt' else 'clear')
        menu()

menu()
```

Figura 9 - Função *menu()*

4. CONCLUSÃO

Apesar do estresse e frustrações vindas das diversas tentativas de criar um bot no Telegram, o trabalho nos ajudou a fixar todo o conteúdo aprendido no decorrer do semestre, além de ter sido uma atividade divertida que aproximou os membros de nossa equipe. Pela falta de tempo, infelizmente não conseguimos colocar no programa tudo o que planejamos, porém estamos felizes com o resultado e esperamos que o professor também esteja.

REFERÊNCIAS

http://www.portal.ufpr.br/normalizacao
https://ubuntuforums.org/showthread.php?t=2394609