Aula 5
13 e 14/11/2023

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (P)

Licenciatura em Videojogos
Ano 1 / Semestre 1

Filipe Pinto

E-mail - filipe.pinto@ulusofona.pt
Sala U.0.5: das 11.00-19.00
Discord do curso

- Stacks

Funciona como um pilha de dados com uma estrutura ordenada

Lógica LIFO - Last In First Out

Exemplos práticos "***undo/redo functions***"

Funções: push and pop

=> introduzir e remover elementos no fim de um stack

- Queues

Funciona como um fila de dados com
uma estrutura ordenada

Lógica FIFO - First In First Out

Exemplos práticos "***execute sequential events***"

Funções: enqueue and dequeue
=> introduzir elementos no inicio (appendleft) e remove do fim (pop)

- Módulos

Criar módulo permite manter estrutura e organização no desenvolvimento

Permite a reutilização de código e simplificação de projectos complexos

Permite o acesso a bibliotecas e funcionalidades adicionais

Fundamental no processo de desenvolvimento de jogos

Fundamentos de Programação

Aula 5

- Números aleatórios

A randomização de número é um sistema muito usado em jogos, especialmente RPG (RNG - Random number generation)

Existem diferentes formas de fazer essa geração de números

Em jogos esses procedimentos podem obedecer a duas lógicas - aleatória ou procedimental
(Random generation ou procedural generation)

Seeds permitem a reprodutibilidade da “aleatorização”

Estas lógicas são usadas das mais variadas formas em game development

Fundamentos de Programação
Exercícios: Stacks

Exercício 1:
#Histórico de Ações do Jogador:

#Implemente uma stack para armazenar as ações de um jogador .
#Cada ação é uma string descrevendo o movimento.
#Implemente métodos - undo_last_action() e add_new_action().
stack => actions_history , actions_list = vv

#Exemplo de Execução:

```
Actions history: deque(['move pawn', 'take card', 'play dice', 'capture enemy pawn'])
Action undone: capture enemy pawn
```

Oprint(show stack with all actions)
print(undo_last_action())

Exercício 2:
#Gerenciador de Telas:

#Crie uma stack para gerenciar as telas de um jogo.
#A navegação dever ser feita com recurso a input,
com uma opção fechar janela e para voltar à ultima

stack => navigation_history,
screens_list = ["main menu", "settings",
"game settings", "key bindings"]

#Exemplo de Execução:
Executar interface

```
Screens available: ['main menu', 'settings', 'game settings', 'key bindings']
Choose screen to move in or 'close' to go back. Type exit to leave:
main menu
Current navigation history: deque(['main menu'])

Screens available: ['settings', 'game settings', 'key bindings']
Choose screen to move in or 'close' to go back. Type exit to leave:
settings
Current navigation history: deque(['main menu', 'settings'])

Screens available: ['game settings', 'key bindings']
Choose screen to move in or 'close' to go back. Type exit to leave:
game settings
Current navigation history: deque(['main menu', 'settings', 'game settings'])

Screens available: ['key bindings']
Choose screen to move in or 'close' to go back. Type exit to leave:
close
Closing: game settings
Current navigation history: deque(['main menu', 'settings'])

Screens available: ['key bindings', 'game settings']
Choose screen to move in or 'close' to go back. Type exit to leave:
close
Closing: settings
Current navigation history: deque(['main menu'])

Screens available: ['key bindings', 'game settings', 'settings']
Choose screen to move in or 'close' to go back. Type exit to leave:
exit
```

Exercício 3:
#Inventário de Itens com Limite de Capacidade:

#Implemente um inventário para um personagem de jogo, que pode conter até 5 itens.
#Adicionar items sem exceder a capacidade
stack => inventario
items = ['Sword', 'Shield', 'Potion', 'Bow', 'Arrows', 'Axe', 'Armor']
items_capacity = 5

#Exemplo de Execução:
adicionar lista items à stack
print(stack)
print(discard_item())

```
Adding items to inventory:
Inventory capacity exceded: Axe discarded
Inventory capacity exceded: Armor discarded
Inventory before discarding: deque(['Sword', 'Shield', 'Potion', 'Bow', 'Arrows'])
Descarding: Arrows
Current inventory: deque(['Sword', 'Shield', 'Potion', 'Bow'])
```

Fundamentos de Programação
Exercícios: Stacks
Soluções possíveis

```python
from collections import deque

actions_history_stack = deque()
actions_list = ["move pawn", "take card", "play dice", "capture enemy pawn"]

def add_new_action(action):
    if action in actions_list:
        actions_history_stack.append(action)

def undo_last_action():
    if actions_history_stack:
        return "Action undone: " + actions_history_stack.pop()
    return "No actions left to undo"

for action in actions_list:
    add_new_action(action)

print("Actions history:", actions_history_stack)

print(undo_last_action())
```

```python
from collections import deque

navigation_history_stack = deque()
screens_list = ["main menu", "settings", "game settings", "key bindings"]

def open_screen(screen):
    if screen in screens_list:
        navigation_history_stack.append(screen)
        screens_list.remove(screen)

def close_current_screen():
    if navigation_history_stack:
        closed_screen = navigation_history_stack.pop()
        screens_list.append(closed_screen)
        return "Closing: " + closed_screen
    return "No window left to close"

def screen_interface():
    while True:
        print("\nScreens available:", screens_list)
        print("Choose screen to move in or 'close' to go back. Type exit to leave:")
        choice = input().lower()

        if choice == 'exit':
            break
        elif choice in screens_list:
            open_screen(choice)
        elif choice == 'close':
            print(close_current_screen())
        else:
            print(choice + " is not available")

        if(navigation_history_stack):
            print("Current navigation history:", navigation_history_stack)
        else:
            print("You are back where you started")

screen_interface()
```

```python
from collections import deque

inventory_stack = deque()
items = ['Sword', 'Shield', 'Potion', 'Bow', 'Arrows', 'Axe', 'Armor']
items_capacity = 5

def add_items_to_inventory(items_list):
    print("Adding items to inventory:")
    for item in items_list:
        inventory_stack.append(item)
        if len(inventory_stack) > items_capacity:
            print("Inventory capacity exceded: " + inventory_stack.pop() + " discarded")

def discard_item():
    if inventory_stack:
        return "Descarding: " + inventory_stack.pop()
    return "Inventory is empty"

add_items_to_inventory(items)

print("Inventory before discarding:", inventory_stack)
print(discard_item())
print("Current inventory:", inventory_stack)
```

Fundamentos de Programação
Exercícios: Queues

Exercício 1:
#Gerenciamento de Eventos do Jogo:

Crie uma queue para gerenciar eventos de jogo. Implemente uma função para adicionar eventos à queue, outra para processar os eventos, esta deve receber um boolean, para executar todos os eventos ou apenas o próximo.

queue => events

#Exemplo de Execução:
Adicionar evento => 'Attack enemy'
Adicionar evento => 'Move to position'
Adicionar evento => 'Pick up item'

```
Next event:
Attack enemy
```

```
Executing all events:
Attack enemy
Move to position
Pick up item
```

execute_events(boolean)

Exercício 2:
#Sistema de Turnos para Jogos Multiplayer:

#Implemente uma queue para um jogo multiplayer. Cada jogador é representado por seu nome.
#Adicionar uma lista de jogadores e uma função que finaliza o turno desse jogador que o coloca no final da queue

queue => turn_queue
players_list = ['Jack', 'Alice', 'Bob', 'Joane']

#Exemplo de Execução:
adicionar players à queue
print(end_turn())

```
Queue before end of turn: deque(['Joane', 'Bob', 'Alice', 'Jack'])
Turn finished: Jack
Queue after end of turn: deque(['Jack', 'Joane', 'Bob', 'Alice'])
```

Exercício 3:
#Sistema de gerenciamento de animações

#Implemente um sistema para gerir animações em sequência para um jogo usando uma queue.
#Cada animação é representada por um string com o nome da animação.
#Escrever animações com input que serão colocadas numa queue que depois executadas por ordem

queue => animations_queue

#Exemplo de Execução
animation_interface()

```
Type the animations you want to execute. Type 'done' to execute.
Add animation: run
Add animation: walk
Add animation: jump
Add animation: done

Executing all animations
Executing animation: run
Executing animation: walk
Executing animation: jump
```

Fundamentos de Programação

Exercícios: Queues

Soluções possíveis

```python
from collections import deque

events = deque()

def add_event(event):
    events.appendleft(event)

def execute_events(is_all):
    if is_all:
        print("Executing all events:")
        while events:
            print(events.pop())
    else:
        print("Next event:")
        if events:
            print(events.pop())
        else:
            print("No events left")

add_event('Attack enemy')
add_event('Move to position')
add_event('Pick up item')

#execute false or true
#execute_events(False)
execute_events(True)
```

```python
from collections import deque

turn_queue = deque()
players_list = ['Jack', 'Alice', 'Bob', 'Joane']

def add_players_to_queue(players_list):
    for name in players_list:
        turn_queue.appendleft(name)

def end_turn():
    if turn_queue:
        current_player = turn_queue.pop()
        turn_queue.appendleft(current_player)
        return "Turn finished: " + current_player
    return "No players in queue"

add_players_to_queue(players_list)
print("Queue before end of turn: " + str(turn_queue))
print(end_turn())
print("Queue after end of turn: " + str(turn_queue))
```

```python
from collections import deque

animations_queue = deque()

def add_animation(animation):
    animations_queue.appendleft(animation)

def execute_animations():
    print("\nExecuting all animations")
    while animations_queue:
        print("Executing animation:", animations_queue.pop())

def animation_interface():
    print("Type the animations you want to execute. Type 'done' to execute.")
    while True:
        animation_input = input("Add animation: ")
        if animation_input.lower() == 'done':
            break
        else:
            add_animation(animation_input)

    execute_animations()

animation_interface()
```

Fundamentos de Programação

Exercícios: Módulos

Exercício 1:

#Criar um módulo para manipular stacks com funções push, pop e check_empty

Exercício 2:

Criar um módulo para manipulação de queues com funções de enqueue, dequeue, move_to_end_of_the_queue e check_empty

list_strings = ['helmet', "boots', 'gloves', 'armor', 'pants']

#Não introduzir manualmente cada elemento
#Criar uma função para introduzir a lista na stack e na queue
#Executar todas as funções dos módulos

Exercício 3:

#Criar módulo com duas funções, uma que sirva para processar uma string, outra para contar repetições de palavras ou letras

string = 'It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way – in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.'

Exercício 4:

Criar um módulo para manipular dicionários, uma função para introduzir e remover, outra para listar keys, values ou o dicionário completo e outra para criar um dicionário através de uma lista de strings

lista de strings => ['helmet', "boots', 'gloves', 'helmet', 'armor', 'pants', 'helmet', 'gloves', 'armor']

Fundamentos de Programação

Exercícios: Módulos

Soluções possíveis

```python
from Module import *

stack = deque()

list_strings = ["aaa", "bbb", "ccc", "ddd"]

for string in list_strings:
    push_value(stack, string)

print(stack)

while stack:
    pop_value(stack)

check_empty(stack)
print(stack)
```

```python
from collections import deque

def push_value(lista, value):
    return lista.append(value)

def pop_value(lista):
    return print(lista.pop())


def check_empty(lista):
    if(lista):
        print("Is not empty")
    else:
        print("Is empty")
```

```python
from Module import *

queue = deque()

list_strings = ["aaa", "bbb", "ccc", "ddd"]

for string in list_strings:
    enqueue_value(queue, string)

print(queue)

check_empty(queue)

move_to_end_of_queue(queue)

print(queue)

while queue:
    dequeue_value(queue)

check_empty(queue)
print(queue)
```

```python
from collections import deque

def enqueue_value(lista, value):
    return lista.appendleft(value)

def dequeue_value(lista):
    return print(lista.pop())

def move_to_end_of_queue(lista):
    value = lista.pop()
    lista.appendleft(value)


def check_empty(lista):
    if(lista):
        print("Is not empty")
    else:
        print("Is empty")
```

```python
string = """It was the best of times, it was the worst of times,
it was the age of wisdom, it was the age of foolishness,
it was the epoch of belief, it was the epoch of incredulity,
it was the season of Light, it was the season of Darkness,
it was the spring of hope, it was the winter of despair,
we had everything before us, we had nothing before us,
we were all going direct to Heaven,
we were all going direct the other way - in short,
the period was so far like the present period,
that some of its noisiest authorities insisted on its being received,
for good or for evil, in the superlative degree of comparison only."""

print(count_repetitions(process_string(string, True)))
```

```python
def process_string(string, isList):
    if(isList):
        string = string.split()
        for words in string:
            words = words.strip(";:!?'").lower()
    else:
        string = string.replace(" ", "").strip(";:!?").lower()
    return string

def count_repetitions(lista):
    result = {}

    for item in lista:
        if item in result:
            result[item] += 1
        else:
            result[item] = 1

    return result
```

Fundamentos de Programação

Exercícios: Módulos

Soluções possíveis

```
dict = {'A': 1, 'B' : 2, 'C' : 3}

print(manage_dict_entries(dict, 'D', 4, True))

print(manage_dict_entries(dict, 'D', 4, False))

list_dictionary(dict, "keys")

list_dictionary(dict, "values")

list_dictionary(dict, "all")

print(create_dict(['helmet', 'boots', 'gloves', 'helmet', 'armor', 'pants', 'helmet', 'gloves', 'armor']))
```

```
def manage_dict_entries(dict, key, value, is_adding):
    if is_adding:
        if key in dict:
            print(str(key) + " already exists in dictionary")
        else:
            dict[key] = value
    else:
        if key in dict:
            del (dict[key])
        else:
            print(str(key) + " does not exist in dictionary")
    return dict

def list_dictionary(dict, string):
    if string == "keys":
        for keys in dict:
            print("key: " + str(keys))
    elif string == "values":
        for value in dict.values():
            print("value: " + str(value))
    elif string == "all":
        for key, value in dict.items():
            print("key: " + str(key) + " value: " + str(value))
    else:
        print("Choose keys, values or all")

def create_dict(list):
    result = {}

    for item in list:
        if item in result:
            result[item] +=1
        else:
            result[item] = 1

    return result
```

Fundamentos de Programação

Exercícios: Random

Exercício 1:

##Definir um valor seed random entre 0 e 100000 e aplicá-lo a uma função que define uma chance 10% de um evento ser raro

```
You found noting special.
PS C:\Users\filip> & C:/Users
You found a rare treasure!
```

Exercício 2:

#Aleatorizar os atributos de força, agilidade e inteligência de uma personagem, com valores entre 1 e 10 e arredondar para um numero inteiro.

```
print("Strength:", randomizar)
print("Agility:", randomizar)
print("Inteligence:", randomizar)
```

```
Strength: 9
Agility: 9
Inteligence: 5
```

Exercício 3:

#Randomizar items de uma lista

```
items = ['Sword', 'Shield', 'Potion', 'Gold', 'Key']
```

```
Items inside the chest: ['Sword', 'Key', 'Gold', 'Potion', 'Shield']
```

Exercício 4:

#Aleatorizar um objecto baseado no nível do jogador

```
items_by_level = {
    1: ['Apple', 'Rusty sword', 'Wooden shield'],
    2: ['Steel sword', 'Helm', 'Healing potion'],
    3: ['Longbow', 'Leather armor', 'Agility boots'],
    4: ['Magical sword', 'Steel armor', 'Strength ring'],
    5: ['Magical wand', 'Invisibility cloak', 'King\'s crown']
}
```

```
Item received: Invisibility cloak
```

Exercício 5:

#Criar um mapa de jogo onde cada posição é gerada aleatoriamente para representar diferentes tipos de terreno e garantir que o mapa seja o mesmo a cada execução.
#O mapa pode ser representado com nested lists e os tipos de terreno representados por strings.
#O mapa deverá ter 5 linhas e 5 colunas

```
terrain_types = ['Water', 'Grass', 'Mountain', 'Road', 'Sea']
```

```
['Grass', 'Mountain', 'Road', 'Grass', 'Road']
['Road', 'Sea', 'Grass', 'Grass', 'Road']
['Sea', 'Grass', 'Grass', 'Road', 'Road']
['Road', 'Mountain', 'Road', 'Grass', 'Mountain']
['Road', 'Water', 'Mountain', 'Grass', 'Sea']
```

Fundamentos de Programação

Exercícios: Random

Soluções possíveis

```python
import random

value = random.uniform(0, 100000)

random.seed(int(value))

def event_rarity():
    if random.random() < 1:
        return "You found a rare treasure!"
    else:
        return "You found noting special."

print(event_rarity())
```

```python
import random

def generate_atribute(min_val, max_val):
    return random.uniform(min_val, max_val)

print("Strength:", int(generate_atribute(1, 10)))
print("Agility:", int(generate_atribute(1, 10)))
print("Inteligence:", int(generate_atribute(1, 10)))
```

```python
import random

items = ['Sword', 'Shield', 'Potion', 'Gold', 'Key']
random.shuffle(items)
print("Items inside the chest:", items)
```

```python
import random

items_by_level = {
    1: ['Apple', 'Rusty sword', 'Wooden shield'],
    2: ['Steel sword', 'Helm', 'Healing potion'],
    3: ['Longbow', 'Leather armor', 'Agility boots'],
    4: ['Magical sword', 'Steel armor', 'Strength ring'],
    5: ['Magical wand', 'Invisibility cloak', 'King\'s crown']
}

def get_random_item(level):
    return random.choice(items_by_level[level])

player_level = 5
print("Item received:", get_random_item(player_level))
```

```python
import random

random.seed(111)
terrain_types = ['Water', 'Grass', 'Mountain', 'Road', 'Sea']
map = []
for i in range(5):
    line = []
    for i in range(5):
        terrain = random.choice(terrain_types)
        line.append(terrain)
    map.append(line)

for line in map:
    print(line)
```