SME0827- Estrutura de dados
Trabalho - Jogo de Ficção Interativa

Augusto Sousa Nunes	N°USP 11319677
Caio Vinicius de Oliveira	N°USP 11932272
Enzo Fagionato Peira Ruffino	N°USP 11916902
Marcelo Nascimento da Silva	N°USP 10668518

22 de Julho de 2021

Introdução

Esse relatório tem como objetivo apresentar o conhecimento obtido por nosso grupo na disciplina. Para isso foi escolhido pelo docente o desenvolvimento de um jogo de ficcão interativa como forma de aplicar tal conteúdo.

História

Inpirado pelo jogo "ZORK", nosso grupo utilizou como ambiente principal um navio em alto mar no estilo "Escape", que repentinamente é invadido e passa a ser danificado. Cabe ao jogador percorrer os ambientes, procurar por itens e pensar em uma maneira de utilizá-los para completar tarefas e vencer o jogo.

Personagem

Roberto: Você é um técnico de manutenção de um grande navio de contâineres chamado CS UNION. Depois de passar pelo canal de Suez, piratas somalianos entraram no navio, danificaram o sistema de comunicação e levaram seus colegas de trabalho como reféns. Por sorte, você conseguiu se esconder na sala de máquinas e ninguém te encontrou. Agora, sozinho no navio e sem ter como chamar por ajuda, você precisa usar suas habilidades mecânicas para poder escapar de alguma maneira.

Instruções do jogo

O jogo baseia-se num sistema de escolhas, ao adentrar em um ambiente, o jogador poderá realizar uma ação digitando ela em seu display, por exemplo:

esq ou esquerda: Movimentação pelo navio;

dir ou direita: Movimentação pelo navio;

int ou interagir: O personagem interage com algo;

ajuda: Abre o menu de interações possíveis;

pegar (ITEM): Pega um item específico;

pegar todos: Pega todos os itens disponíveis;

olhar (ITEM): Retorna uma descrição do ambiente;

procurar: procura por algo em dado ambiente.

Algumas interações são disponíveis apenas para itens específicos. A tarefa do jogador será arrumar um meio de consertar o gerador e depois escapar pelo bote salva vidas.

Implementação e estruturas utilizadas

A codificação do jogo foi realizada em Python 3, utilizando POO e conceitos de herança como base. Por exemplo, "Movel" sendo uma classe, tem seus parâmetros herdados para as classes "QuartoInterativo" e outras como "ArmarioFerramentas", que são consideradas móveis interativos.

O código é dividido da seguinte maneira:

main.py: O jogo é executado pela função main() em que se utiliza 3 objetos: 1 Scenario e 2 Inventory. Todo início do jogo, a função init() é chamada para a inicialização dos objetos necessários. Depois disso, é criado um loop para ler os comandos do jogador até que decida sair ou que ganhe jogo.

init.py: A função init(), chamada no inicio do jogo, é responsável por criar e empacotar todos os objetos, para que no final retorne os 3 objetos principais para funcionalidade da main(). Na primeira inicialização, há a criação dos objetos e armazenamento destes em arquivos utilizando o módulo joblib. Fizemos isso para evitar a criação desses objetos toda vez que há a inicialização do jogo, de modo com que poupe processamento no momento da inicialização.

Scenario - Mapa do navio, construído como uma lista duplamente encadeada adaptada para o nosso jogo cujo conteúdo são objetos do tipo Environment (Figura 1). Inicialmente, o percurso entre os ambientes era representado por um dicionário (Figura 2), mas visto que teríamos que realizar uma busca sempre que acessássemos o cenário atual, descobrimos que não seria a maneira mais eficiente de implementar.

Além disso, uma lista duplamente encadeada tem custo linear $O(n)$ de acesso, porém a lista implementada acessa apenas as posições vizinhas, logo o custo seria constante $O(1)$. Mas diferente de um deque ou uma pilha, não há a necessidade de descarte ou armazenamento, se encaixando perfeitamente na necessidade do projeto.

No caso, cada ambiente no cenário é um nó, e sendo um nó, possuí duas variáveis que apontam para o ambiente anterior e para o próximo. O maquinário

seria a "cabeça" e o convés a "cauda" dessa cadeia de nós. Abaixo, segue a disposição das salas. As funções de ir para o próximo e para o anterior foram implementas para alertar o jogador se está no inicio ou no final do Scenario.

Diagrama

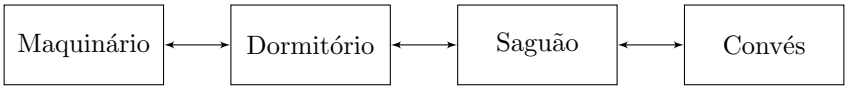

Figura 1

```
def __init__(self):
    self.actual = None
    self.envs = {}

def add_environment(self, env, env_idx):
    """
        Method to add the given environment to the scenario
      Parameters:
        env (Environment): environment to add
    """
    if not self.actual:
        self.actual = env_idx
    self.envs[env_idx] = env
```

Figura 2

```
class Scenario:
    """
      Abstract data structure of the game's scenario
    """

    def __init__(self):
        self.head = None

    def add_environment(self, data):
        """
            Method to add the given environment to the scenario
          Parameters:
            env (Environment): environment to add
        """
        new = Node(data)
        new.next = self.head
        if self.head != None:
            self.head.prev = new
        self.head = new
```

Environment - São os ambientes. Cada ambiente tem no máximo 6 móveis que podem ser itens interativos ou não interativos. Nesta classe, temos os métodos para adicionar móvel e olhar o ambiente.

Móveis - São os móveis do ambiente. Móveis não interativos são criados com a estrutura MovelNaoIter, que retorna apenas uma mensagem contendo a descrição do móvel. Os demais móveis interativos são criados com classes

específicas, pois cada móvel interativo terá métodos próprios de interação, estas se encontram no diretório "./moveis/".

Item - É a classe mais primitiva do código. Tem apenas o nome e mensagem.

Inventory - Inventário do personagem. Armazena os nomes dos itens e os objetos do tipo Item. Durante o jogo são utilizados dois inventários, um que será utilizado para armazenar itens do personagem e o outro, chamado de "Shadow Inventory", é utilizado para armazenar a energia, um "item" que é utilizado em algumas interações entre personagem e móvel.

```
class Environment:
  """
  Classe de um ambiente cujo jogador pode interagir
  """

  def __init__(self, name, descricao):
    self.name = name
    self.descricao = descricao
    self.moveis = [None, None, None, None, None, None]
    self.nomes_moveis = [None, None, None, None, None, None]
```