

# PROJETO DE ESTRUTURA DE DADOS

SEM GRUPO

---

05/02/2020

Ivo Martins – 8180159

João Carlos – 8180546

## Índice

Objetivo do Projeto.....	3
Manual.....	4
Menu Inicial.....	4
Menu do modo de jogo .....	6
Detalhes adicionais.....	8
Packages.....	8
Resolução do problema proposto.....	9
Diagrama de classes .....	11
Controlo de Versões .....	12
Apreciação critica .....	13

## Objetivo do Projeto

Este projeto tem como âmbito a criação de um jogo a partir de um mapa (ficheiro json).

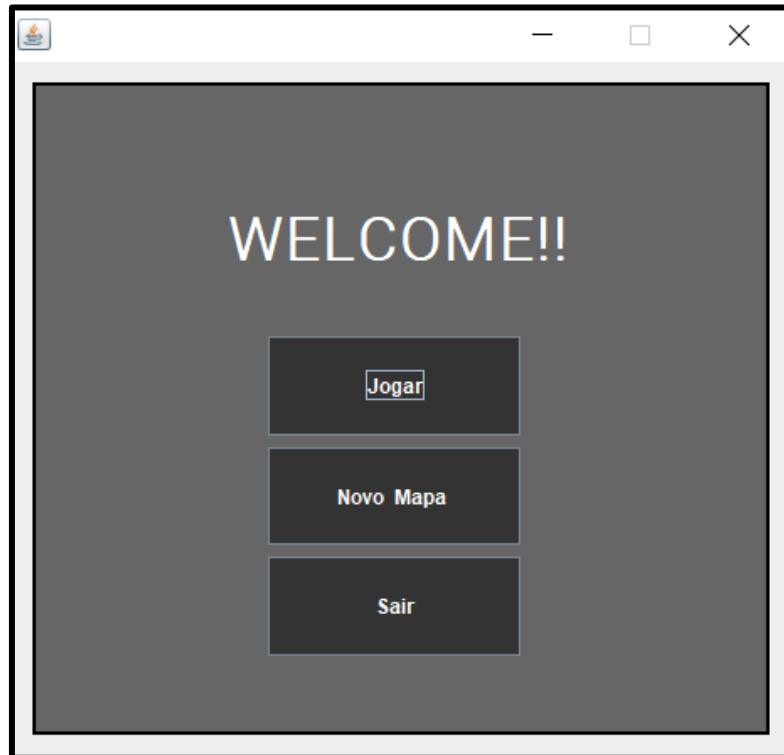
O objetivo do jogo em si é o jogador chegar ao exterior sem que os seus pontos (ou vidas) cheguem a zero. Ou seja, o jogador começa na entrada, sendo que esta tem conexão com apenas uma divisão (neste caso: o hall/corredor), e a partir dessa divisão, pode aceder a outras, em que em certos casos terão fantasmas (devidamente indicados), que causaram a perda de pontos, finalizando-se o jogo quando o jogador/utilizador alcançar o exterior ou a sua vida acabar.

Também é possível carregar outros mapas se estes contiverem as mesmas características do mapa predefinido do jogo, tendo em conta que o mapa esteja dentro da pasta do projeto.

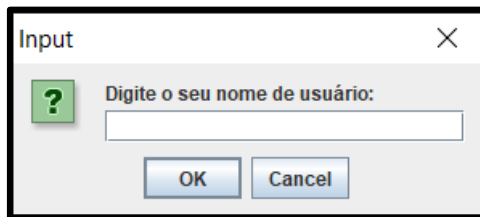
## Manual

### Menu Inicial

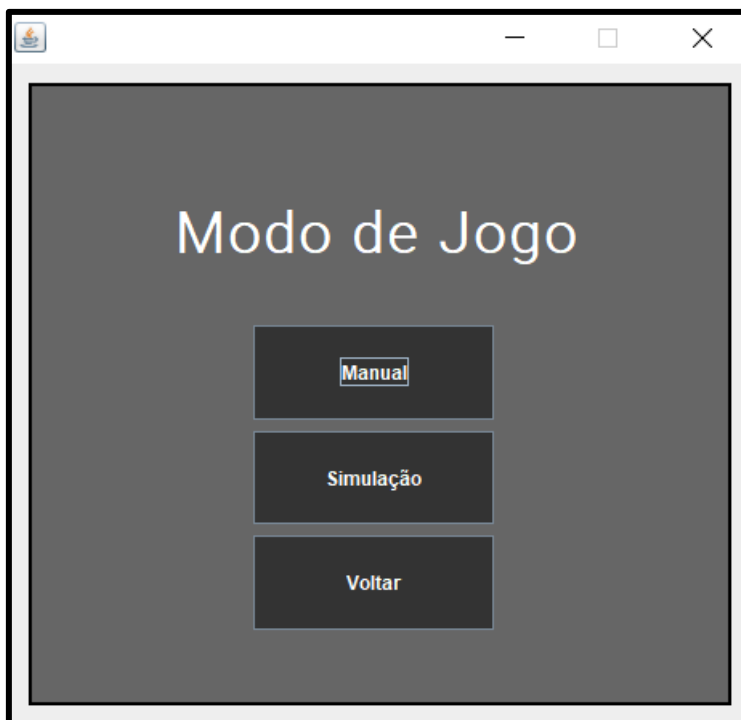
- Quando o programa é corrido no netbeans será aberta uma janela, sendo essa uma janela com o menu inicial:



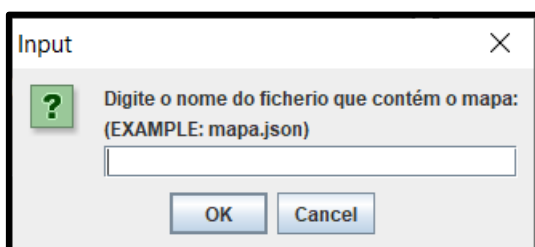
- Ao selecionar “Jogar” será aberta outra janela, que pedirá o nome de utilizador:



- ❖ E de seguida aparecerá outro menu, que permite escolher o modo de jogo:



- Ao selecionar “Novo Mapa”, aparecerá a seguinte janela:

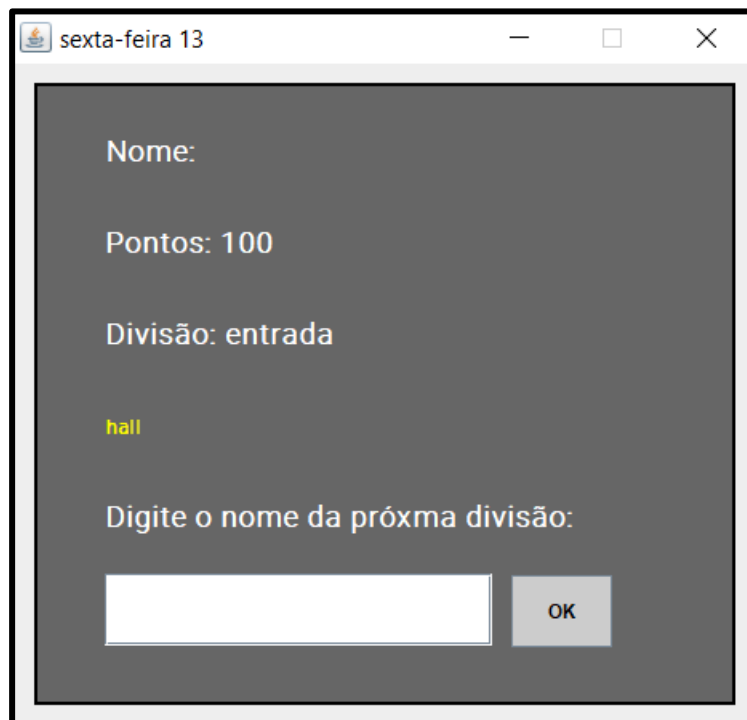


- ❖ Que pede o nome do ficheiro que contem o novo mapa e o tipo de ficheiro, que supostamente deveria de ser json.

## Menu do modo de jogo

Existem dois modos de jogo sendo esses: Manual e Simulação.

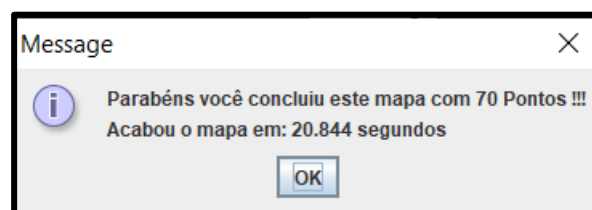
- O Manual pede ao jogador onde deseja ir a partir da entrada, considerando a sua localização corrente, até que seja alcançado o exterior ou até que perca todos os pontos de vida.



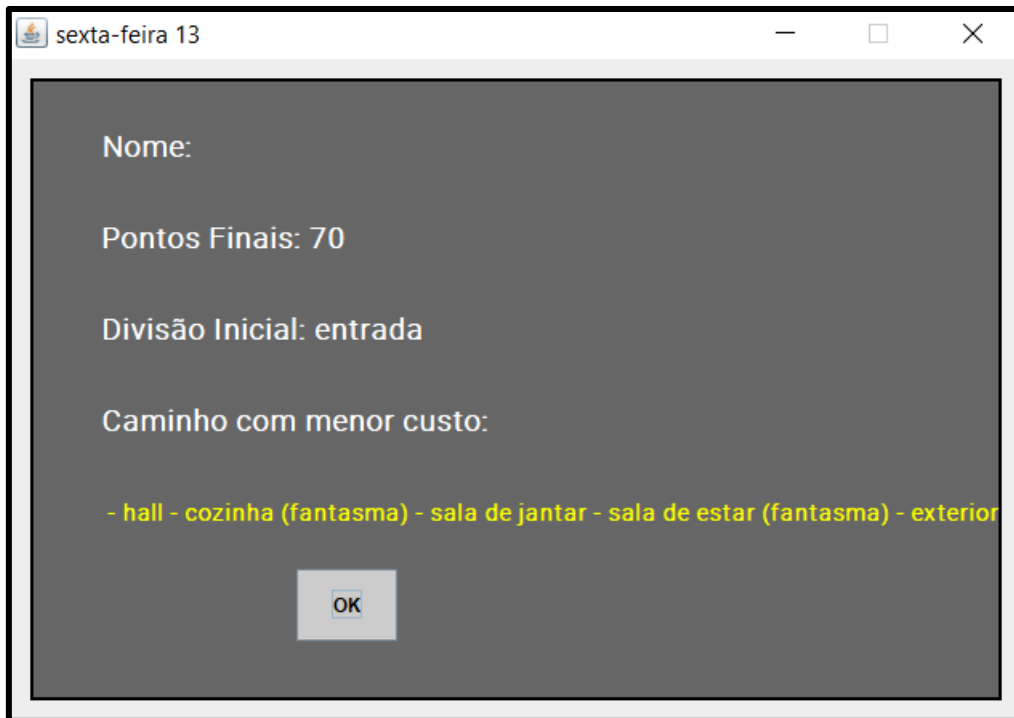
A screenshot of a game window titled "sexta-feira 13". The window has a dark gray background and contains the following text and elements:

- Nome:
- Pontos: 100
- Divisão: entrada
- hall
- Digite o nome da próxima divisão:
- A text input field.
- An "OK" button.

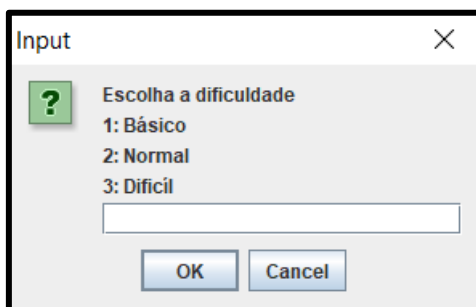
- ❖ Em amarelo aparecem todas as divisões que é possível aceder a partir da divisão atual. Caso apareça ghost à frente da divisão significa que se for para essa divisão irá perder pontos de vida.
- ❖ Quando o jogador chegar ao exterior irá aparecer uma janela com o número de pontos com que terminou o mapa e o tempo gasto, sendo que estes valores serão guardados num ficheiro ("Classificações").



- Simulação é o tem como objetivo mostrar o caminho/Path com menor peso do mapa, ou seja, mostra o caminho que menos pontos de vida serão perdidos.



- Quando selecionado qualquer um dos modos é pedido o nível de dificuldade que o jogador quer selecionar:



- ❖ Em Básico, a perda de pontos será (pontos perdidos nessa divisão) \* 1; em Normal, será: (pontos perdidos nessa divisão) \* 2; e em Difícil, será (pontos perdidos nessa divisão) \* 3.

## Detalhes adicionais

### Packages

O nosso projeto contém 5 packages, sendo esses:

- **Collection** → contém todas as classes necessárias para a execução do programa, criadas nas aulas práticas.
  - ❖ ArrayList
  - ❖ ArrayList
  - ❖ ArrayOrderedList
  - ❖ ArrayUnoderedList
  - ❖ BinaryTree
  - ❖ BinaryTreeNode
  - ❖ Graph
  - ❖ HeapNode
  - ❖ LinearNode
  - ❖ LinkedHeap
  - ❖ LinkedList
  - ❖ LinkedQueue
  - ❖ LinkedStack
  - ❖ LinkedUnorderedList
  - ❖ Network
- **Exceptions** → contém todas a exceções criadas para uma boa execução do programa.
  - ❖ ElementNotFoundException
  - ❖ EmptyCollectionException
  - ❖ EmptyException
  - ❖ RecruiterAlreadyExistsException
- **Interfaces** → contém todas as interfaces criadas nas aulas práticas, criadas nas aulas.
  - ❖ BinaryTreeADT
  - ❖ GraphADT
  - ❖ HeapADT
  - ❖ ListADT
  - ❖ NetworkADT
  - ❖ OrderedListADT
  - ❖ QueueADT
  - ❖ StackADT
  - ❖ UnorderedListADT



- **Jogo** → contém todas as classes criadas por nós relacionadas às interfaces gráficas e à música do jogo.
  - ❖ Main
  - ❖ Manual
  - ❖ Menu Inicial
  - ❖ Menu Jogo
  - ❖ Music
  - ❖ Simulação
- **Mapa** → contém as classes criadas para a leitura de mapas e para a criação de um jogador.
  - ❖ Aposentos
  - ❖ ArrayOrderedUti
  - ❖ Jogador
  - ❖ Mapa

Este projeto, também contém uma biblioteca adicional sendo essa **json-simple-1.1.1.jar**.

## Resolução do problema proposto

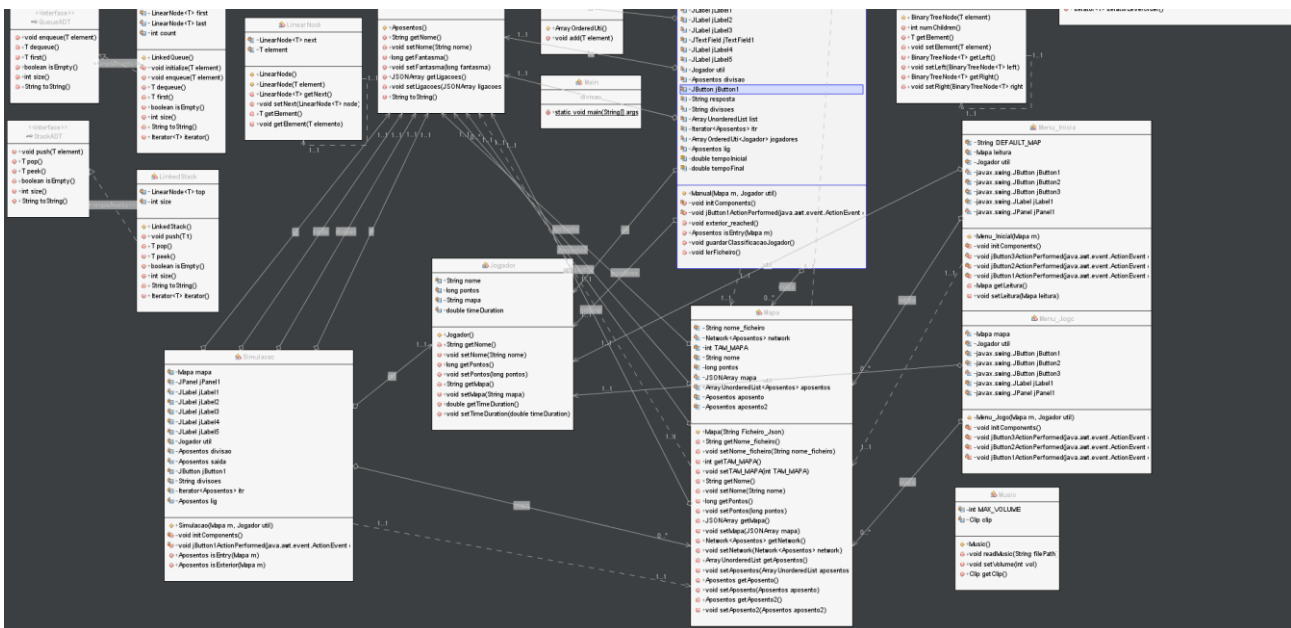
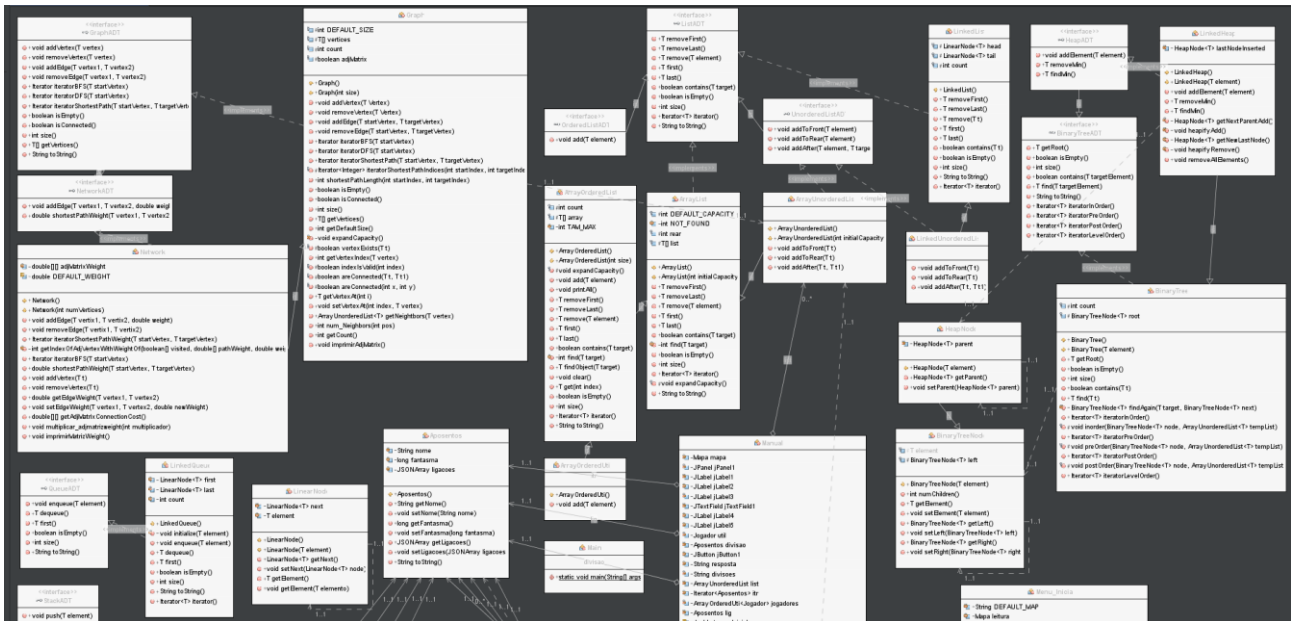
Na nossa opinião, foram propostos dois problemas, sendo esses, o da realização de um modo **Manual** e o da realização do modo **Simulação**, sendo que de certa forma, criaram-se dois algoritmos. A base destes algoritmos foi o grafo, pois este é a estrutura de dados mais adequada para a resolução do problema proposto.

- Para **Manual**, utilizamos Iterator e o método getNeighbors (presente na classe Graph).
  - ❖ No método getNeighbors é dado um objeto genérico que neste caso vai ser do tipo Aposento, em que é pesquisada a posição do Aposento no Array de vértices, e uma vez obtida essa posição, será pesquisada na Matriz de adjacência a existência de uma conexão do Aposento dado, com outros Aposentos, adicionando esses Aposentos a uma ArrayUnorderedList do tipo Aposentos.
  - ❖ Com o Iterator percorremos o ArrayUnorderedList e obtemos os vizinhos da divisão em que o jogador se encontra.

- Para **Simulação**, utilizamos o algoritmo de Dijkstra.
  - ❖ Começamos por verificar qual é o Aposento em que começa o caminho e o Aposento em que o caminho acaba, sendo esses os Aposentos com os nomes: “**entrada**” e “**exterior**” (obtidos a partir dos métodos **isEntry** e **isExit** na classe **Simulação**). Depois utilizamos o método **iteratorShortestPathWeight**, que recebe o Aposento de partida e o de destino, e verifica todos os caminhos/paths em que existe uma ligação entre os aposentos dados e vai verificando o caminho/path com menor custo. Adaptando para este projeto, é o caminho que menos pontos de vida tira, e assim retorna um **Iterator** para esse caminho.
  - ❖ Utilizando esse **Iterator** imprimimos na interface gráfica, criada a partir da classe **Simulação**, os nomes das divisões do caminho com menor custo e o número de pontos de vida finais.

## Diagrama de classes

O Diagrama de Classes foi gerado pelo netbeans.



## Controlo de Versões

Todo o controlo de versões foi efetuado a partir do github, sendo que o desenvolvimento do nosso projeto se encontra no seguinte link:

[https://github.com/ivomartins29/Projeto\\_ED.git](https://github.com/ivomartins29/Projeto_ED.git)

## Apreciação crítica

Este projeto, revelou que é necessário ter um bom entendimento do conteúdo ensinado nas aulas, teóricas ou práticas. Sendo que foram utilizadas coleções criadas nas aulas práticas para a realização do projeto.

Uma das partes em que foi requerida mais atenção, foi a elaboração do algoritmo Dijkstra, utilizado no modo de jogo Simulação. Este algoritmo tem vários conteúdos aprendidos nas aulas, para obter o caminho/path com menor custo.

Foi um projeto bastante elaborado no ponto de vista em que é preciso estarmos atentos quanto à estrutura de dados utilizadas.