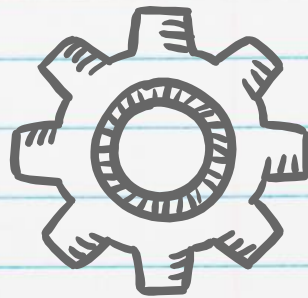




O RATO E

O LABIRINTO

CONFIGURAÇÃO DO TRABALHO:



- *Linguagem: Python 3*
- *Ambiente: Linux/Windows*
- *Objetivo do Trabalho: Criar um algoritmo capaz de apresentar um "rato" que descobre o caminho correto do labirinto. E na próxima execução ser capaz de escolher um caminho mais rápido (que exige menos passos).*
- *Integrantes: Leonardo Barbosa de Farias*

DIVISÃO DO PROGRAMA (CLASSES):

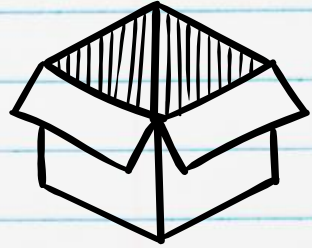
O programa foi dividido em um *total de 5 classes* sendo elas:

- Actioner
- Checker
- Commander
- DataManager
- Interface



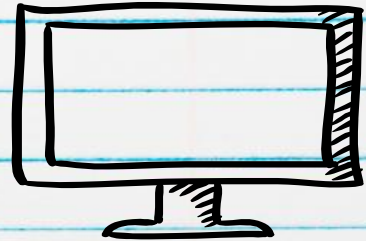
DATAMANAGER:

É a classe responsável por relacionar as respostas do rato e alterá-las na lista que representa o labirinto. Além disso, ela também envia as posições adjacentes ao rato, na horizontal e vertical.



INTERFACE:

A interface, como nome sugere, é quem utiliza a biblioteca do Tkinter. Ela recebe as informações da classe DataManager e exibe sempre que for alterada.



CHECKER:



A classe checker faz a manipulação dos arquivos txt criados durante a execução do programa. O labirinto e o caminho correto até a saída são salvos em arquivos txt.

Sendo assim, na próxima execução o rato será capaz de identificar se está no mesmo labirinto e mesma posição que antes.

ACTIONER:

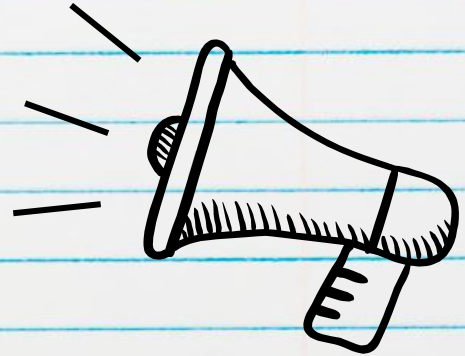


É a classe que representa a principal parte lógica do programa. Pois ela que é responsável pelas escolhas do rato dentro do labirinto. Algoritmo de busca, armazenamento e comunicação com o **DataManager**, isso tudo ela faz. Possui a maior quantidade de métodos.

Ps: A forma que armazena e realiza as buscas serão discutidas mais à frente.






COMMANDER:

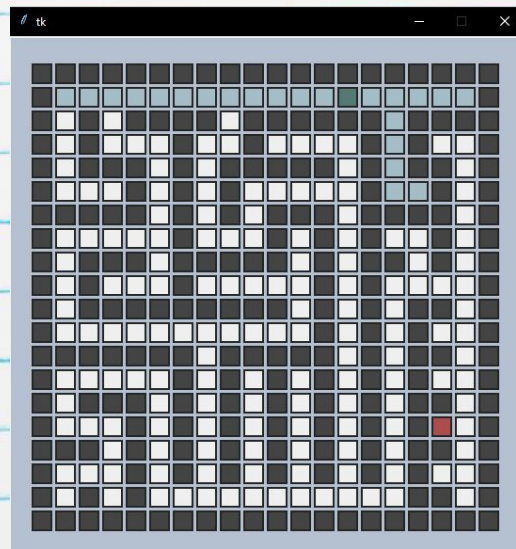
A classe commander funciona como uma função "main" nas demais linguagens. Já que é ela quem vai instanciar e relacionar as outras classes. Ela também é responsável pela velocidade do rato e dizer se o rato está em um novo labirinto ou se deve percorrer o caminho correto já registrado em uma busca passada.



COMO O PROGRAMA É EXIBIDO:

O programa é exibido através de uma tela composta por 400 cubos (20x20), os quais suas cores representam um tipo de espaço:

-  -> Parede (1);
-  -> Espaço ainda não visitado(0);
-  -> Espaços já visitados pelo rato(4);
-  -> Posição atual do Rato(3);
-  -> Saída(2).



Ps: Ao redor do labirinto são colocados paredes (a fim de evitar que o rato caia do labirinto)

REPRESENTAÇÃO DE DADOS:



O programa interage com diversas formas de dados, por isso é necessário saber como representá-las da melhor forma.

- **Labirinto:** Lista na qual cada item também é uma lista, a intenção é que cada item da lista principal represente uma linha e os subitens colunas:
 - 0 -> Espaço vazio;
 - 1 -> Parede;
 - 2 -> Saída;
 - 3 -> Posição atual do rato;
 - 4 -> Caminho já percorrido.
- **Direções do Rato:** Dicionário de funções. Cima, direita, baixo e esquerda;

REPRESENTAÇÃO DE DADOS:

- **Espaços adjacentes:** Lista com 4 itens, cada posição representa uma direção. Os itens possuem a mesma representação do labirinto (0, 1, 2, 3, 4);
- **Bifurcação:** Tupla de valores booleanos. Na qual true significa que há um espaço de bifurcação. Sendo a 1º posição para cima/direita e 2º posição para baixo/esquerda.
- **Localização das bifurcações:** Lista na qual os itens são tuplas. O primeiro valor é referente ao número de passos dados pelo rato e o segundo valor, sua direção.
- **Caminho Correto:** Lista de direções tomadas corretamente pelo rato.

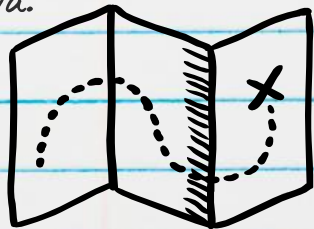
REPRESENTAÇÃO DE DADOS:

- *Labirinto*: $[[...], [...], ...]$
- *Direções*: $\{ \text{'Cima': func()}, ... \}$
- *Espaços Adjacentes*: $[0, 1, 4, 1]$
- *Bifurcação*: $(\text{True}, \text{True})$
- *Loc. da Bifurcação*: $[(32, \text{'direita'}), (38, \text{'baixo'}), ...]$
- *Caminho Correto*: $[\text{'direita'}, \text{direita'}, \text{'cima'}, ...]$

LÓGICA PARA ANDAR PELO LABIRINTO:

A ideia do programa é de enviar a menor quantidade de informações possíveis para o rato, sendo assim ele realmente parece que foi jogado em um labirinto e tudo que sabe é andar, o que tem na sua frente e quantos passos foram feitos. Por conseguinte, sua forma de andar é a seguinte:

- Anda reto até "dar de cara" com uma parede.
- Enquanto anda reto, salva as possíveis bifurcações que deixou de realizar.
- Ao encontrar uma parede, ele checa se é possível alternar sua direção.
- Caso não seja possível, ele retorna para a última bifurcação salva.
- A partir dessa troca ele volta a seguir um caminho reto.

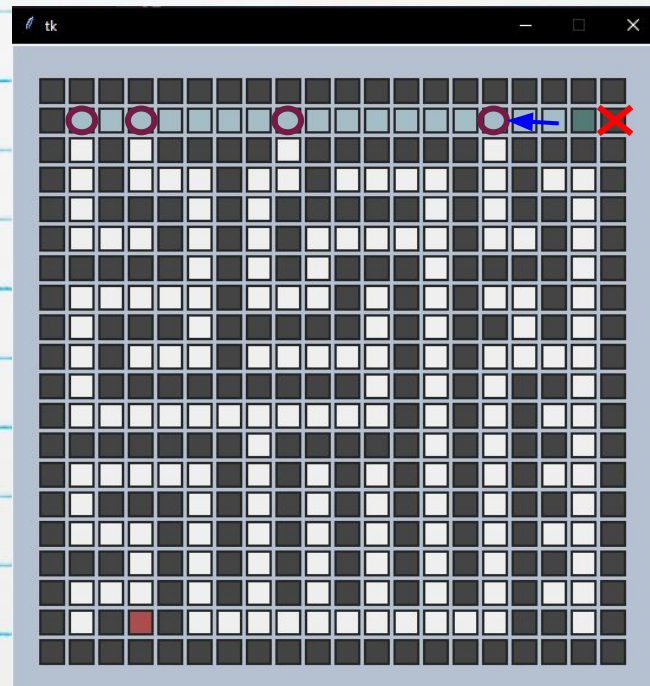


Ps: Lembrando que sempre vai priorizar caminhos que ainda não passou.

LÓGICA PARA ANDAR NO LABIRINTO:

Exemplo:

- -> Salvando bifurcações;
- ✗ -> Após encontrar uma parede, analisa se é possível trocar sua direção.
- ➡ -> Como não foi possível trocar sua direção, o rato volta para a última bifurcação que encontrou.



BIFURCAÇÕES:



14

A ideia por trás das bifurcações veio do vídeo [Maze Solving - Computerphile](#), nele Michael Pound, pesquisador da universidade de Nottingham, criou um algoritmo capaz de solucionar labirintos, apesar de não mostrar nenhum código ele compartilha sua lógica.

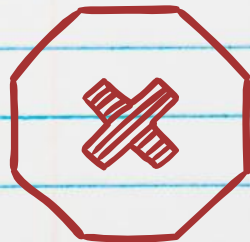
Simplificando, uma bifurcação é registrada sempre que o rato precisa alterar o plano da sua direção (horizontal para vertical ou vice-versa). O que não for bifurcação é tratado como um caminho reto, por isso o rato não precisa se preocupar tanto assim.



COMO ESCOLHER UM CAMINHO MELHOR:

Após uma execução o rato é capaz de identificar um caminho mais efetivo. É importante frisar que *não é o melhor caminho* porém, que exige menos passos em relação a primeira busca.

Isto é capaz devido à lógica de andar do rato. Sempre que o rato precisa retornar para a última bifurcação, ele deleta seus passos. A razão é que se o rato está sendo obrigado a voltar então o caminho que tomou foi errado.





LIMITAÇÕES E BUGS DO PROGRAMA:

- O programa *não foi projetado para suportar alterações drásticas no labirinto*. Por exemplo: Mudar seu modelo de 20x20 para 17x18.
- O labirinto precisa ser contornado por paredes, caso contrário, o rato acaba recebendo *valores nulos ao procurar pelos espaços adjacentes*.
- O rato possui uma limitação em relação a loops dentro do labirinto, como sempre salva as bifurcações quando passa por uma *acaba perdendo o controle de quantas vezes já passou pelo mesmo ponto*.