LABIRINTO – ESPECIFICAÇÕES DA INTERFACE
PROFESSOR: ADELARDO ADELINO DANTAS DE MEDEIROS

O DESIGN INICIAL

O aplicativo Labirinto deve se basear na classe `QMainWindow` do Qt, mantendo a barra de menus e a barra de status, mas eliminando a barra de ferramentas. A janela principal deve ter o título "Labirinto – Algoritmo A*" (propriedade `windowTitle` da janela principal).

O objeto principal é uma `QTableWidget` que exibe o mapa. Dois botões `QRadioButton` indicam o que vai acontecer no caso dê um click duplo em alguma das células da tabela: ou a origem ou o destino do caminho a ser calculado serão fixados na posição onde ocorreu o click duplo (figura 1).

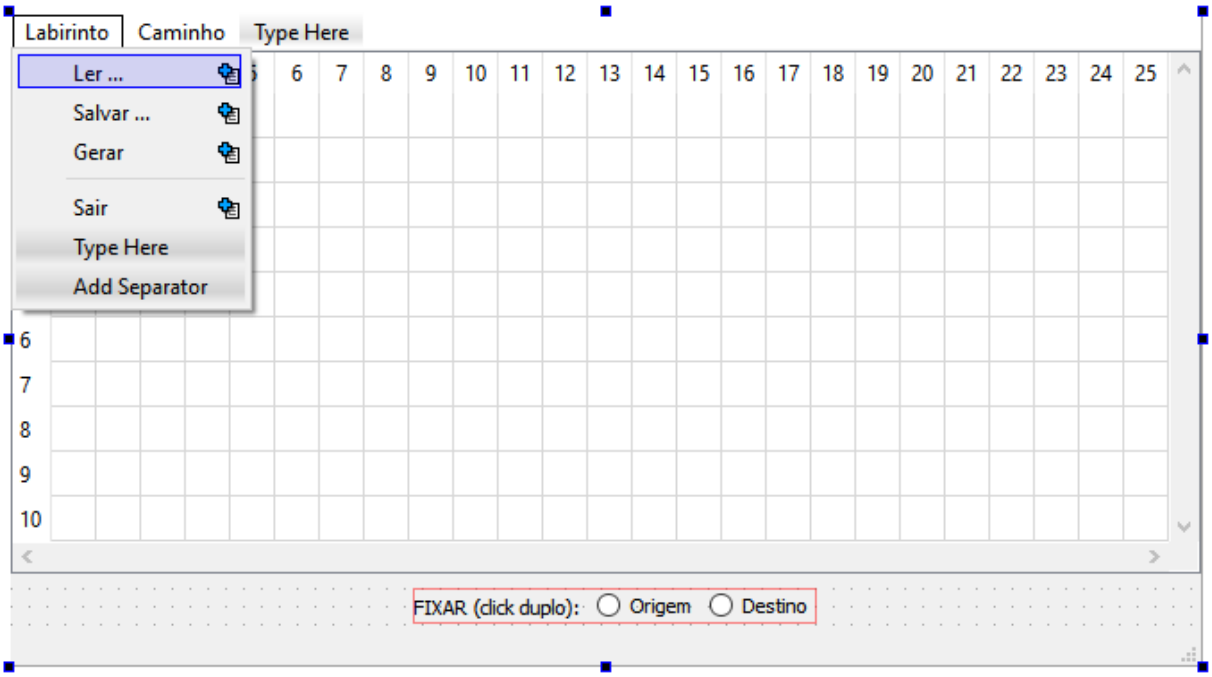

Figura 1 - Design geral da QMainWindow do Labirinto

A tabela `QTableWidget` tem as seguintes propriedades fixadas no design inicial:

- Exibe linhas entre as células (propriedade `showGrid` é verdadeira);
- Tem inicialmente 10 linhas e 25 colunas (propriedades `rowCount` e `columnCount`);
- As barras de rolagem horizontal e vertical estão sempre visíveis (as propriedades `verticalScrollBarPolicy` e `verticalHorizontalScrollBarPolicy` devem ser definidas como `ScrollBarAlwaysOn`);
- Tem cabeçalhos horizontal e vertical (as propriedades `horizontalHeaderVisible` e `verticalHeaderVisible` devem ser true);
- Não permite seleção de células usando shift+click ou ctrl+click (método `setSelectionMode(QAbstractItemView::NoSelection)`) nem navegação entre células usando TAB (`setTabKeyNavigation(false)`).
- Cada célula tem dimensão 25x25 (as propriedades `verticalHeaderDefaultSectionSize`, `verticalHeaderMinimumSectionSize`, `horizontalHeaderDefaultSectionSize` e `horizontalHeaderMinimumSectionSize` devem ter valor 25).

Deve ser criado um slot que reaja à seguinte ação na `QTableWidget`:

- `on_..._cellDoubleClicked(int row, int column)`: click duplo em uma célula.

A barra de menus contém dois menus:

- "Labirinto", com as opções "Ler", "Salvar" e "Gerar", seguidas de um separador, após o qual está a opção "Sair".
- "Caminho", com as opções "Calcular" e "Limpar".

Para cada uma das ações dos menus, deve ser criado um slot correspondente:

- `on_actionLer_triggered()`
- `on_actionSalvar_triggered()`
- `on_actionGerar_triggered()`
- `on_actionSair_triggered()`
- `on_actionCalcular_triggered()`
- `on_actionLimpar_triggered();`

INICIALIZAÇÃO (CONSTRUTOR DA CLASSE PRINCIPAL)

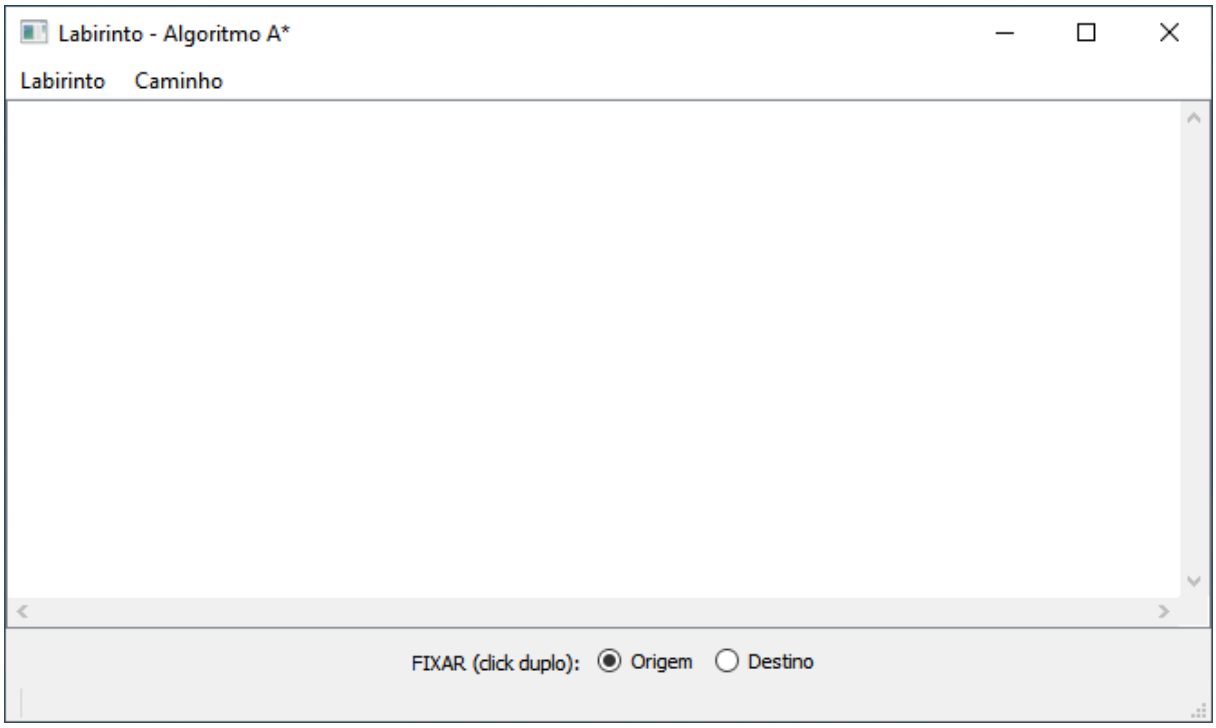

Figura 2 – Aparência inicial do programa

A classe principal, que herda da classe `QMainWindow`, deve incluir por composição ao menos:

- Um objeto da classe `Labirinto` para conter o mapa atual do jogo.
- Um objeto `QLabel*`:
  - alocado dinamicamente e com texto inicial nulo: `= new QLabel("");` e
  - inserido na barra de status para exibir mensagens: `statusBar()->addWidget(...)`

As células do cabeçalho da tabela devem ter fundo cinza claro, usando a folha de estilos: `setStyleSheet("QHeaderView::section {background-color:lightgray}")`.

O `QRadioButton` deve começar com a opção "Origem" selecionada: `setChecked(true)`.

A interface sempre lê as informações que deve exibir utilizando as funções de consulta do objeto da classe `Labirinto`. Esse procedimento (REDEFININDO A TABELA: veja a seção a seguir) é realizado na inicialização e sempre que um novo mapa é carregado, seja porque foi lido de um arquivo, seja porque foi gerado aleatoriamente. O mapa inicial é vazio, usando o construtor default da classe `Labirinto`. Por essa razão, ao iniciar o jogo, o aplicativo deve ter a aparência da figura 2.

LEITURA DE ARQUIVO

Quando a opção "Ler" do menu "Labirinto" é escolhida, o slot `on_actionLer_triggered` deve abrir uma caixa de diálogo para que o usuário escolha o arquivo a ser lido, usando a função `QFileDialog::getOpenFileName(...)`. Em seguida, o mapa deve ser lido do arquivo escolhido utilizando a função `ler()` da classe `Labirinto`.

Em caso de erro na leitura, deve ser aberta uma `QMessageBox` com a mensagem de erro e o nome do arquivo (`setText`) e em seguida aguardar que o usuário clique OK (`exec`). Em caso de sucesso, a tabela deve ser redefinida (REDEFININDO A TABELA: veja a seção a seguir).

SALVAMENTO DE ARQUIVO

Quando a opção "Salvar" do menu "Labirinto" é escolhida, o slot `on_actionSalvar_triggered` deve inicialmente verificar que o mapa não está vazio (função `empty()` da classe `Labirinto`) e emitir uma mensagem de erro (`QMessageBox`) se for esse o caso.

Caso esteja definido, deve abrir uma caixa de diálogo para que o usuário escolha o arquivo a ser salvo, usando a função `QFileDialog::getSaveFileName(...)`. Em seguida, o mapa deve ser salvo no arquivo escolhido utilizando a função `salvar()` da classe `Labirinto`.

Em caso de erro na escrita, deve ser aberta uma `QMessageBox` com a mensagem de erro e o nome do arquivo (`setText`) e em seguida aguardar que o usuário clique OK (`exec`).

GERAÇÃO DE MAPA ALEATÓRIO

Quando a opção "Gerar" do menu "Labirinto" é escolhida, o slot `on_actionGerar_triggered` deve criar um novo mapa utilizando a função `gerar()` da classe `Labirinto`.

Em caso de erro na geração, deve ser aberta uma `QMessageBox` com a mensagem de erro (`setText`) e em seguida aguardar que o usuário clique OK (`exec`). Em caso de sucesso, a tabela deve ser redefinida (REDEFININDO A TABELA: veja a seção a seguir).

REDEFININDO A TABELA (APÓS LEITURA DE ARQUIVO OU GERAÇÃO ALEATÓRIA)

Quando um novo mapa é carregado, o conteúdo anterior da tabela deve ser apagado (`clear`). As dimensões do mapa são lidas do objeto da classe `Labirinto` e, em seguida, a altura e largura do mapa são utilizadas para definir o número de linhas (`setRowCount`) e de colunas (`setColumnCount`) da tabela, respectivamente.

Cada célula da tabela deve conter um objeto do tipo `QLabel`. Esses objetos devem ser criados dinamicamente (`prov = new QLabel ...`) e terem suas propriedades gerais alteradas para que exibam texto centralizado: `setAlignment(Qt::AlignCenter)`.

Uma vez criado e corretamente definido, o `QLabel` deve ser inserido na sua célula respectiva, usando o método `setCellWidget(i,j,prov)` da classe `QTableWidget`.

Quando um novo mapa é carregado, qualquer eventual mensagem anterior exibida na barra de status deve ser suprimida, apagando-se o conteúdo do `QLabel` correspondente: `setText("")`

Uma vez definido o tamanho da tabela e criados os `QLabel` das células, o conteúdo do mapa pode ser exibido (EXIBINDO UM MAPA: veja a seção a seguir). Essa funcionalidade de exibição deve ser acionada quando um novo mapa for carregado ou quando o conteúdo do mapa sofrer alteração.

EXIBINDO UM MAPA (APÓS NOVO MAPA OU QUANDO OCORRE MODIFICAÇÃO DO CONTEÚDO)

Os objetos `QLabel` das células da tabela devem exibir os valores das casas correspondentes do `Labirinto`. A figura 3 mostra a exibição de um mapa com origem, destino e caminho definidos.

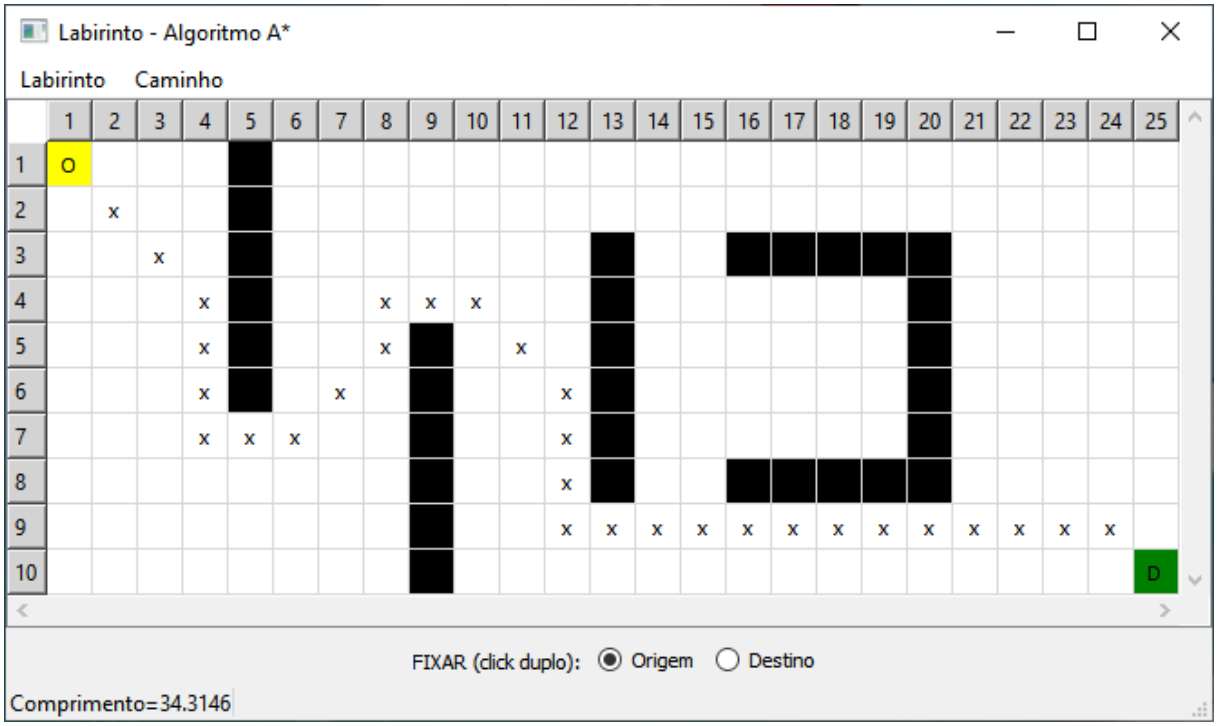

Figura 3 - Exibição de um mapa com caminho definido

Para recuperar o `QLabel*` que está armazenado em uma célula da tabela, usa-se a função de consulta `cellWidget(i,j)`. Em seguida, esses objetos recuperados devem ter suas propriedades alteradas usando métodos da classe `QLabel`:

- Cor de fundo (`setStyleSheet("background: nome_da_cor")`) de acordo com a célula:
  - branco (`white`) para células livres e de caminho;
  - preto (`black`) para obstáculos;
  - amarelo (`yellow`) para a origem; e
  - verde (`green`) para o destino.
- Texto (`setText`):
  - em branco para células livres e obstáculos;
  - “O” para origem;
  - “D” para destino; e
  - “x” para células do caminho.

CLICANDO EM UMA CÉLULA (CLIQUE DUPLO)

Um slot `on_..._cellDoubleClicked(row,column)` deve reagir ao evento de uma célula ser clicada duplamente no mapa, executando as seguintes operações:

- Verificar se a célula correspondente pode ser origem ou destino (não é obstáculo). Se não puder, retorne (encerre a função).
- Ler o `QRadioButton` para ver qual "Origem" ou "Destino" está selecionada: `isChecked()`
- Chamar a função correspondente da classe `Labirinto` para alterar a origem (`setOrigem`) ou o destino (`setDestino`).
- Reexibir o mapa.
- Limpar o texto da barra de status: `setText("")` no `QLabel` correspondente.
- Trocar o `QRadioButton` de "Origem" para "Destino" ou vice-versa: `setChecked(...)`.

ENCERRANDO O PROGRAMA

Quando a opção "Sair" do menu "Labirinto" é escolhida, deve ser chamada a função `QCoreApplication::quit()`.

RESOLUÇÃO DO CAMINHO

Quando a opção "Calcular" do menu "Caminho" é escolhida, o slot `on_actionCalcular_triggered` deve inicialmente verificar que o mapa não está vazio (função `empty()`) e que a origem e o destino do caminho estão definidas (função `origDestDefinidos`). Caso ocorra qualquer um desses casos, deve emitir uma mensagem de erro (`QMessageBox`) e encerrar.

Caso tudo esteja correto, o caminho deve ser calculado através dos seguintes passos:

- Ler o relógio interno: função `now` da classe `steady_clock` (ver main da versão console).
- Chamar o algoritmo de resolução: função `calculaCaminho` da classe `Labirinto`.
- Ler o relógio interno: função `now` da classe `steady_clock` (ver main da versão console).
- Calcular o intervalo de tempo decorrido (ver main da versão console).

Se existe caminho (comprimento retornado >= 0.0), o novo mapa deve ser exibido e o valor do comprimento calculado deve ser mostrado na barra de status: `setText` no `QLabel` correspondente.

Em qualquer caso (existindo ou não caminho), deve ser exibida em uma caixa de diálogo (`QMessageBox`) uma mensagem indicando se o caminho foi ou não encontrado, o número de nós em aberto e em fechado e o tempo decorrido. Essa caixa de diálogo permanece visível até que o usuário clique no botão OK da caixa de diálogo, chamando a função `exec` da `QMessageBox`.

LIMPEZA DO CAMINHO

Quando a opção "Limpar" do menu "Caminho" é escolhida, o caminho eventualmente calculado anteriormente é apagado. Para isso, o slot `on_actionLimpar_triggered` executa as seguintes ações:

- Chamar a função `limpaCaminho` da classe `Labirinto`.
- Reexibir o mapa.
- Limpar o texto da barra de status: `setText("")` no `QLabel` correspondente.