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

## O DESIGN INICIAL

O aplicativo Sudoku 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 tem dimensões 600x420. No widget central, os objetos principais são duas `QTableWidget`: a da esquerda deve apresentar o tabuleiro atual, enquanto a da direita exibe o tabuleiro inicial. Dois botões `QPushButton` correspondem às ações de fazer uma jogada (botão JOGAR) e sair de uma pausa que exige uma ação do usuário (clicar no botão CONTINUAR). Um `QSpinBox` é utilizado para selecionar o valor da jogada, ou seja, o número a ser colocado na casa selecionada no tabuleiro (figura 1).

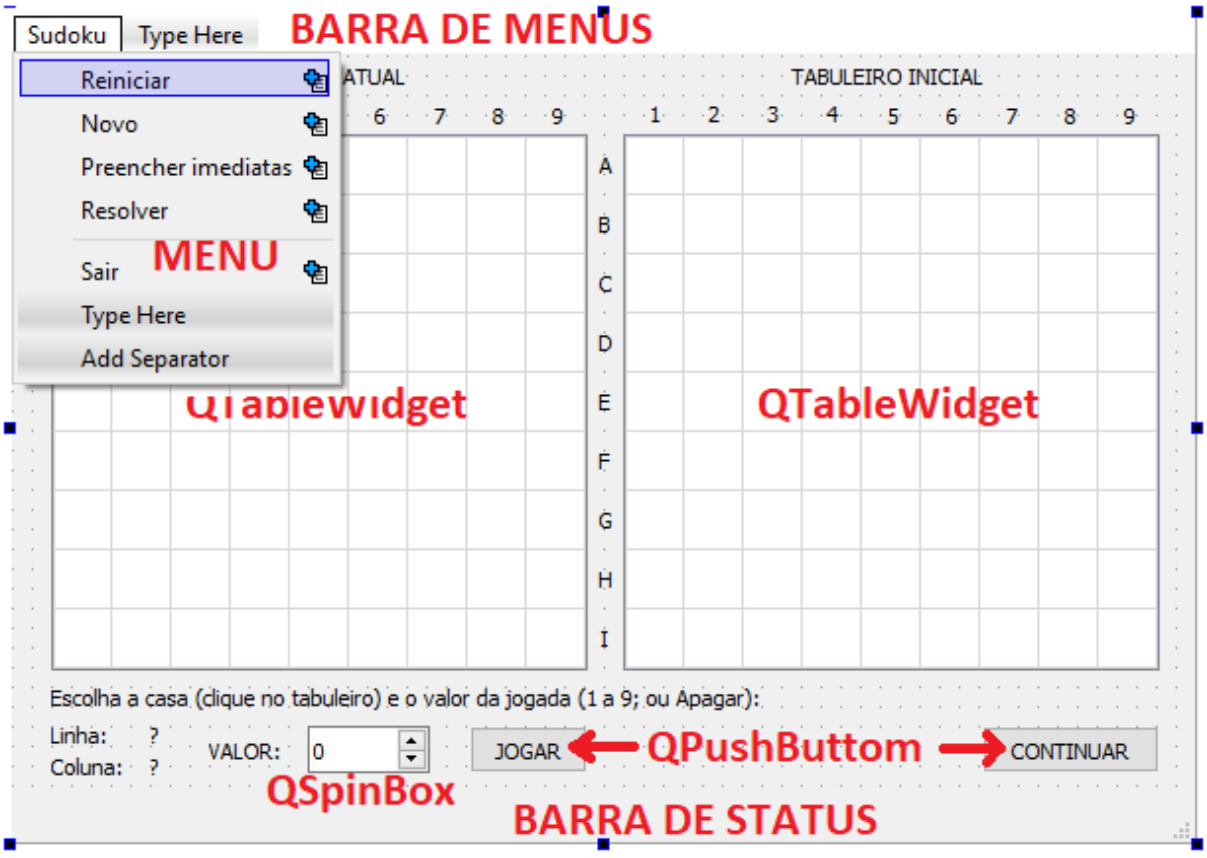


Figura 1 - Design geral da QMainWindow do Sudoku

Cada tabela exibe linhas entre as células (propriedade `showGrid` é verdadeira) e tem 9 linhas e 9 colunas (propriedades `rowCount` e `columnCount` têm valor 9). As tabelas não têm cabeçalhos nem horizontal nem vertical (as propriedades `horizontalHeaderVisible` e `verticalHeaderVisible` devem ser falsas). Cada célula das tabelas deve ter dimensão 30x30 (as propriedades `verticalHeaderDefaultSectionSize`, `verticalHeaderMinimumSectionSize`, `horizontalHeaderDefaultSectionSize` e `horizontalHeaderMinimumSectionSize` devem ter valor 30).
Devem ser criados slots que reajam às seguintes ações nos widgets:

- `on_..._cellClicked(int row, int column)`: uma célula na tabela do tabuleiro atual foi clicada.
- `on_..._valueChanged(int arg1)`: o valor da jogada foi alterado.
- `on_..._clicked()`: o botão JOGAR foi apertado.

A barra de menus contém um único menu, denominado “Sudoku”, com as opções “Reiniciar”, “Novo”, “Preencher imediatas” e “Resolver”, seguidas de um separador, após o qual está a opção “Sair”. Para cada uma dessas ações, deve ser criado um slot correspondente:

- `on_actionSair_triggered()`
- `on_actionPreencher_triggered()`
- `on_actionResolver_triggered()`
- `on_actionNovo_triggered()`
- `on_actionReiniciar_triggered();`

INICIALIZAÇÃO

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

- Um objeto da classe `Jogada`, para armazenar a movimentação desejada pelo usuário.
- Dois objetos da classe `Sudoku`, para conter o tabuleiro inicial e o atual do jogo.

O tabuleiro inicial é lido a partir do arquivo “`sudoku.txt`”, usando o construtor específico da classe `Sudoku`. O tabuleiro atual é inicialmente idêntico ao tabuleiro inicial.

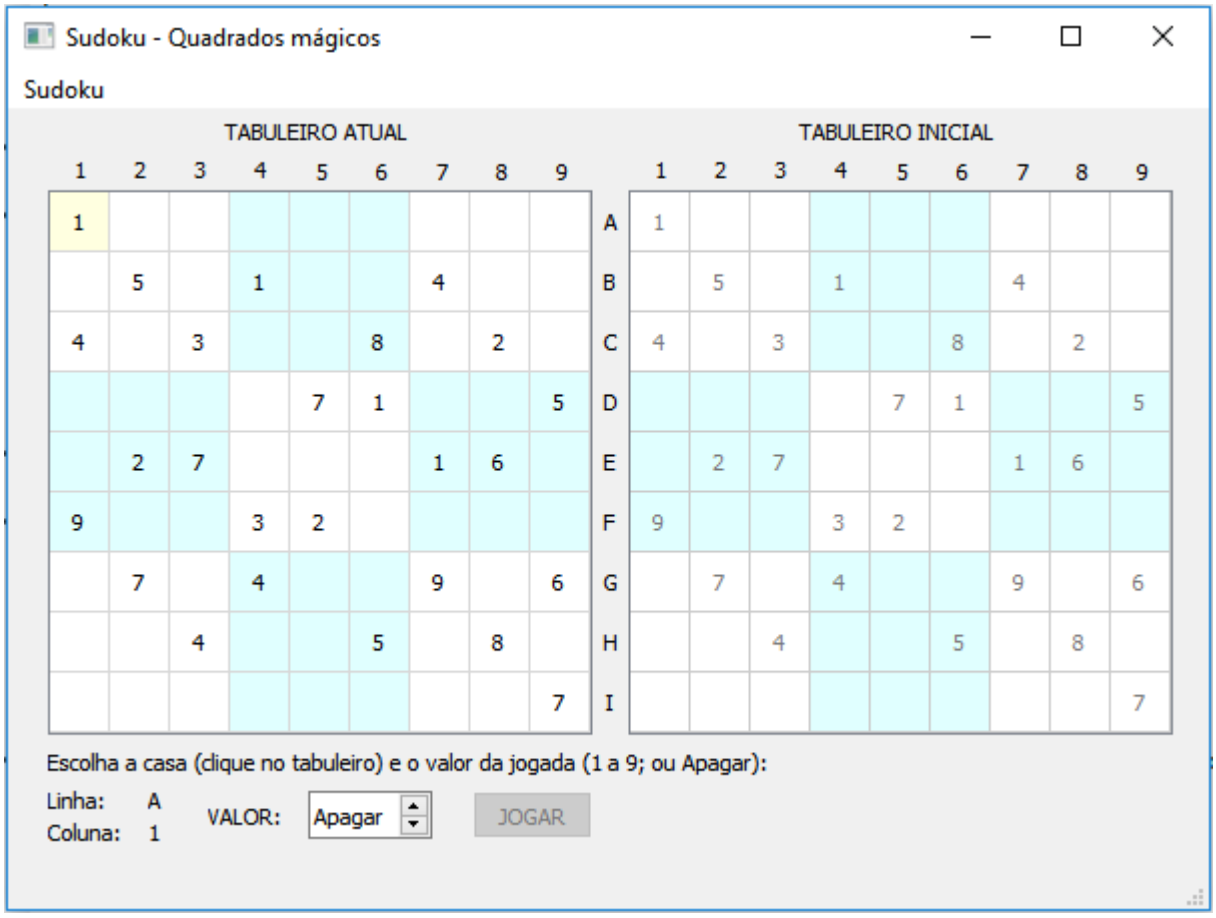


Figura 2 - Inicialização do programa

Ao iniciar o jogo, o aplicativo deve ter a aparência da figura 2. Algumas propriedades dos widgets que compõem o ambiente devem ser fixadas:

- A janela principal deve ter o título “Sudoku - Quadrados mágicos”.
- O botão CONTINUAR deve estar escondido, chamando o método `hide` do `QPushButton`.
- O tabuleiro inicial deve permanecer insensível a eventos (clicar, digitar, etc) durante toda a execução. Para desabilitá-lo, use o método `setEnabled(false)` da `QTableWidget`.

- O tabuleiro atual não deve permitir seleção de células, usando shift+clique ou ctrl+click (método `setSelectionMode(QAbstractItemView::NoSelection)`) nem navegação entre células usando TAB (`setTabKeyNavigation(false)`).
- Cada célula dos tabuleiros deve conter um objeto do tipo `QLabel`. Esses objetos devem ser criados dinamicamente (`prov = new QLabel`) e ter suas propriedades definidas usando métodos da classe `QLabel`:
  - Texto centralizado: `setAlignment(Qt::AlignCenter)`
  - Dimensão 30x30: `setGeometry(0,0,30,30)`
  - Cor de fundo (`setStyleSheet("background: nome_da_cor")`) de acordo com a célula: branco (`white`) e azul claro (`lightcyan`) de acordo com o bloco, e amarelo claro (`lightyellow`) para a célula que está selecionada (inicialmente, a célula A1).

  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`.
- O `QSpinBox` deve ter as propriedades definidas:
  - Faixa de valores de 0 a 9: `setRange(0,9)`
  - Comportamento circular (8→9→0→1): `setWrapping(true)`
  - Valor mínimo (zero, no caso) com significado especial, devendo ser exibido texto “Apagar” ao invés do valor: `setSpecialValueText("Apagar")`
- Os objetos `QLabel` das células dos tabuleiros devem exibir os valores das casas correspondentes dos objetos `Sudoku` que representam o tabuleiro inicial e atual (ver abaixo).

EXIBINDO O TABULEIRO

Para exibir o valor de uma célula de um dos tabuleiros:

- Obtenha o valor numérico a ser exibido, usando o método de consulta da classe `Sudoku`, que é o `operator()(i,j)`
- Obtenha um ponteiro que aponta para o `QLabel` que está associado com a célula a ser exibida, usando o método de consulta da classe `QTableWidget`, que é `cellWidget(i,j)`
- Caso exista um `QLabel` associado à célula (não seja `NULL`), altere seu texto para o valor correto: `setNum(valor)` se for um número de 1 a 9 e `setText("")` se o valor for zero (vazia).

O ARQUIVO `SUDOKU_FORM_QT.CPP`

O código fonte da classe `Sudoku` não deve sofrer alteração para poder ser utilizado com uma interface Qt. Ou seja, o mesmos arquivos `sudoku.cpp` e `sudoku.h` devem ser utilizados para rodar o jogo tanto na interface em modo texto quanto na interface visual.
Para isso, o método `resolver()` da classe `Sudoku` chama os seguintes métodos, também da classe `Sudoku`, que devem ser implementados no arquivo `sudoku_form_Qt.cpp`:

- `exibir()`
- `exibir_origem()`
- `exibir_andamento(Ntab_testados, Ntab_gerados, Ntab_analisar)`

Esses métodos devem chamar métodos da classe principal da interface Qt que sejam capazes de exibir tabuleiros na tabela da esquerda ou da direita ou de apresentar uma mensagem na barra de status. Para que os métodos no arquivo `sudoku_form_Qt.cpp` possam utilizar métodos da classe Qt, deve ser criado um ponteiro global no arquivo `main.cpp` que aponta para a variável principal:

```
MainSudoku *pt_w;
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

```
MainSudoku w;
pt_w = &w;    ...
```

No arquivo `sudoku_form_Qt.cpp`, esse ponteiro é declarado como uma variável externa (`extern MainSudoku *pt_w`). Dessa forma, os métodos da classe `Sudoku` podem chamar métodos da classe `MainSudoku`:

```
void Sudoku::exibir() const
{
    ...
    pt_w->nome_do_metodo_da_classe_MainSudoku();
    ...
}
```

O BOTÃO JOGAR

Durante o jogo, o `QPushButton` do botão JOGAR deve ficar habilitado ou desabilitado de acordo com o fato da jogada ser válida ou inválida para a célula selecionada, conforme a figura 3.

- Para habilitar ou desabilitar o botão: `setEnabled(true)` ou `setEnabled(false)`
- A habilitação ou não do botão deve ser reavaliada sempre que o usuário clicar em uma célula ou alterar o valor da jogada no `QSpinBox`.
- Para determinar se uma jogada é válida ou não para a célula selecionada, deve ser utilizado o método `jogada_valida` ou o método `apagamento_valido` da classe `Sudoku`.


Figura 3- Botão JOGAR habilitado ou desabilitado, conforme a jogada é válida ou não.

CLICANDO EM UMA CÉLULA

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

- Recuperar o `QLabel` correspondente à célula que estava anteriormente selecionada, cujas coordenadas estão armazenadas na `Jogada` que faz parte da classe principal.
- Retornar sua cor de fundo para branco ou azul claro, conforme o bloco.
- Armazenar as coordenadas da célula clicada na `Jogada`.
- Recuperar o `QLabel` correspondente à célula clicada.
- Fazer sua cor de fundo passar a ser amarelo claro.
- Alterar os labels ao lado do spinbox para refletirem a linha e coluna selecionadas
- Atualizar o botão JOGAR para habilitado ou desabilitado, conforme a jogada seja válida ou não.

MUDANDO O VALOR DA JOGADA

Ao alterar o valor do spin box, um slot `on_..._valueChanged()` deverá executar as seguintes operações:

- Armazenar o valor na `Jogada`.
- Atualizar o botão JOGAR para habilitado ou desabilitado, conforme a jogada seja válida ou não.

FAZENDO UMA JOGADA

Ao pressionar o botão JOGAR, o que só será possível se ele estiver habilitado, um slot `on_..._clicked()` deverá executar as seguintes operações:

- Preencher (valor = 1 a 9) ou apagar (valor = 0) uma célula no tabuleiro atual, usando os métodos `fazer_jogada()` ou `apagar_jogada()`.
- Exibir o novo valor da célula no tabuleiro atual.
- Atualizar o botão JOGAR para habilitado ou desabilitado, conforme a jogada seja válida ou não.

O MENU

O menu Sudoku (figura 4) dá acesso às seguintes operações:

- Reiniciar: o slot `on_actionReiniciar_triggered` faz o tabuleiro atual ser igual ao inicial e exibe o tabuleiro atual.
- Novo: o slot `on_actionNovo_triggered` gera um novo tabuleiro inicial, usando o método `gerar` da classe `Sudoku`. Em seguida, faz o atual igual ao inicial e exibe ambos os tabuleiros.
- Preencher imediatas: calcula as casas que podem ser preenchidas imediatamente, usando o método `resolver_casas_faceis` da classe `Sudoku`. Em seguida, se o número de casas preenchidas retornado pelo método for diferente de zero, exibe uma mensagem usando o método `showMessage` (ver a seguir) da barra de status, de acordo com o valor retornado:

| Nº de casas | Mensagem |
|---|---|
| 5 | 5 casas preenchidas |
| 1 | 1 casa preenchida |
| -1 | TABULEIRO INSOLÚVEL! 1 casa preenchida |
| -5 | TABULEIRO INSOLÚVEL! 5 casas preenchidas |
| -100 (< -81) | TABULEIRO INSOLÚVEL! |

  Se houve casas preenchidas, exibe o tabuleiro atual e entra em modo de espera, até que o usuário pressione o botão CONTINUAR (ver a seguir).
- Resolver: o slot `on_actionResolver_triggered` inicialmente exibe o tabuleiro atual no lado direito do aplicativo, na posição normalmente destinada ao tabuleiro inicial. Isso é feito para

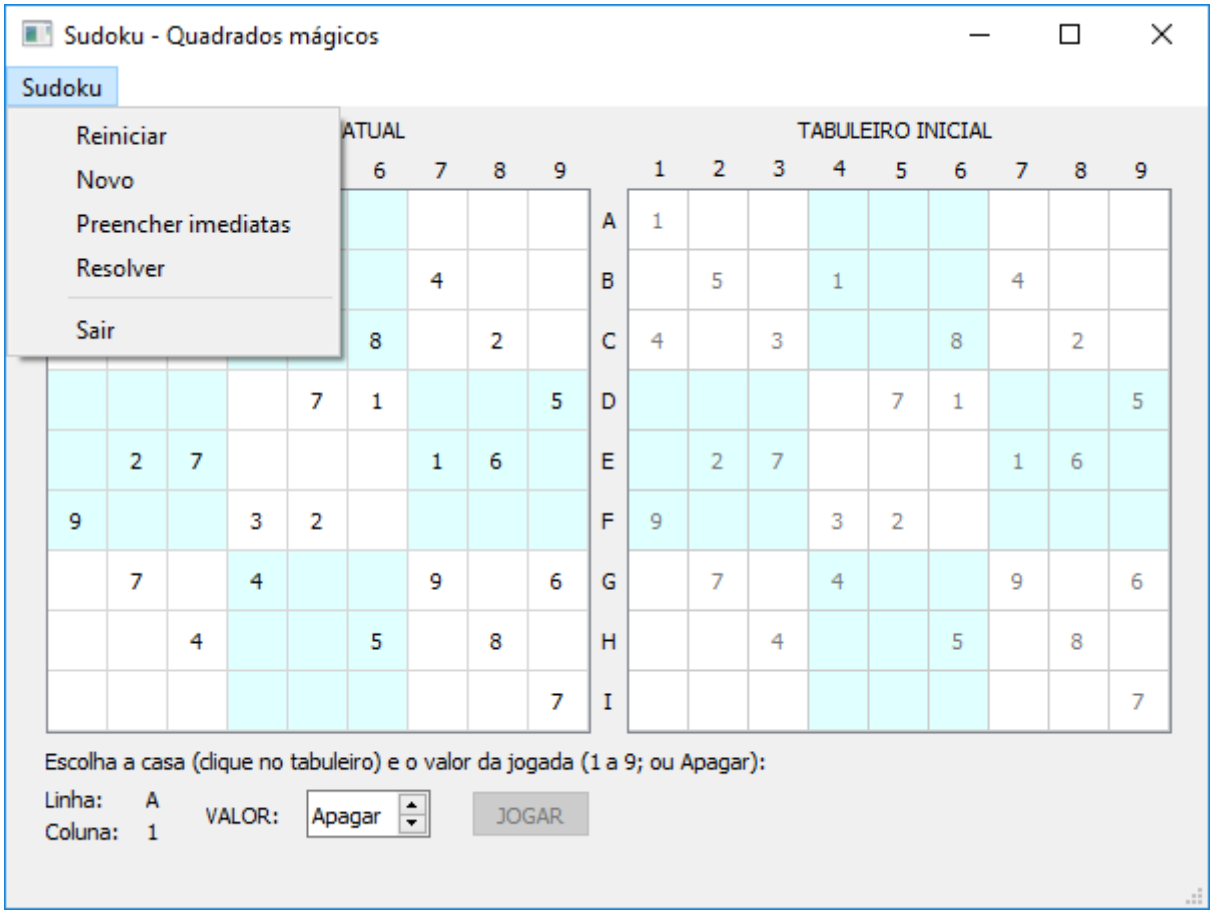


Figura 4 - O menu Sudoku.

que o usuário possa visualizar o tabuleiro a partir do qual está sendo feita a resolução automática durante a evolução do algoritmo. Em seguida, chama o método `resolver` da classe `Sudoku`. Durante a execução do algoritmo de busca, são exibidos os tabuleiros sendo testados e a evolução do número de nós, atividades essas que são feitas pelo método `resolver` (ver a seguir). Concluída a resolução, entra em modo de confirmação, até que o usuário pressione o botão CONTINUAR (ver a seguir). Após a confirmação, reexibe os tabuleiros atual e inicial nas tabelas da esquerda e da direita, respectivamente.

- Sair: o slot `on_actionSair_triggered` chama `QCoreApplication::quit()`.

EXIBINDO MENSAGENS NA BARRA DE STATUS

- As mensagens podem ser exibidas usando o método `showMessage` da barra de status, que recebe como parâmetro uma `QString`. Para apagar a mensagem, use `clearMessage`.
- Já existe um objeto do tipo `QStatusBar` dentro de um objeto do tipo `QMainWindow`.
- Objetos do tipo `QString` podem ser somados (concatenados).
- Existem vários métodos de conversão, tais como o método `QString::number()`, que gera a string correspondente a um número.

ESPERANDO CONFIRMAÇÃO

A espera por confirmação ocorre quando se deseja que o usuário tome ciência de um fato, expresso por uma mensagem previamente exibida na barra de status, sendo que a execução normal do jogo só deve recomeçar após o usuário clicar no botão CONTINUAR.


Figura 5 - Espera por confirmação.

Ao ser chamada a espera por confirmação, deve ocorrer o seguinte (figura 5):

- O menu, o tabuleiro atual, o spin box com o valor da jogada e o botão JOGAR devem ser desabilitados (ficarem insensíveis a eventos): `setEnabled(false)`.
- O botão CONTINUAR deve ser exibido (`show`) e habilitado (`setEnabled(true)`).
- Cria-se um loop de espera por um evento, associando-se o sinal de clicar no botão CONTINUAR com o slot de terminar o loop:

```
QEventLoop espera;
connect(ui->botaoContinuar, SIGNAL(clicked()),
        &espera, SLOT(quit()));
espera.exec();
```

- Apaga a mensagem na barra de status (`clearMessage`).
- Desabilita e esconde o botão CONTINUAR.
- Habilita o menu, o tabuleiro atual, o spin box e o botão JOGAR, dependendo da jogada ser válida.

RESOLUÇÃO

O algoritmo de resolução não tem especificamente a ver com a interface Qt, já que a classe `Sudoku` e o seu método `resolver` devem ser genéricos para poderem ser utilizados tanto com interface em modo texto quanto em interface visual.
Quando o usuário solicita a resolução automática do jogo, o programa deve executar os seguintes passos, em qualquer uma das interfaces:

- Exibir temporariamente o tabuleiro atual no espaço destinado ao tabuleiro inicial.
- Chamar o método `resolver` da classe `Sudoku`.
- Ao término do método, esperar confirmação pelo usuário (figura 6).
- Exibir os tabuleiros atual e inicial nas posições apropriadas (esquerda e direita, na interface Qt). O tabuleiro atual pode ou não ter sido alterado pelo método de resolução, caso tenha ou não encontrado uma solução (figura 7).

O algoritmo de resolução permite acompanhar o seu desenvolvimento através das seguintes ações:

- A cada passo, exibe o tabuleiro analisado, chamando o método `exibir` da classe `Sudoku`, e o número de tabuleiros gerados, analisados e que ainda não foram analisados, chamando o método `exibir_andamento` da classe `Sudoku`. Esses dois métodos são implementados no arquivo `sudoku_form_ . . . .cpp`, que possui uma versão diferente para cada interface.
- Ao final da execução, caso não tenha encontrado solução, exibe o melhor tabuleiro encontrado (menor número de casas vazias), sem alterar o tabuleiro atual (figuras 6 e 7, esquerda). Caso exista solução, o tabuleiro atual recebe o valor dessa solução (figuras 6 e 7, direita).

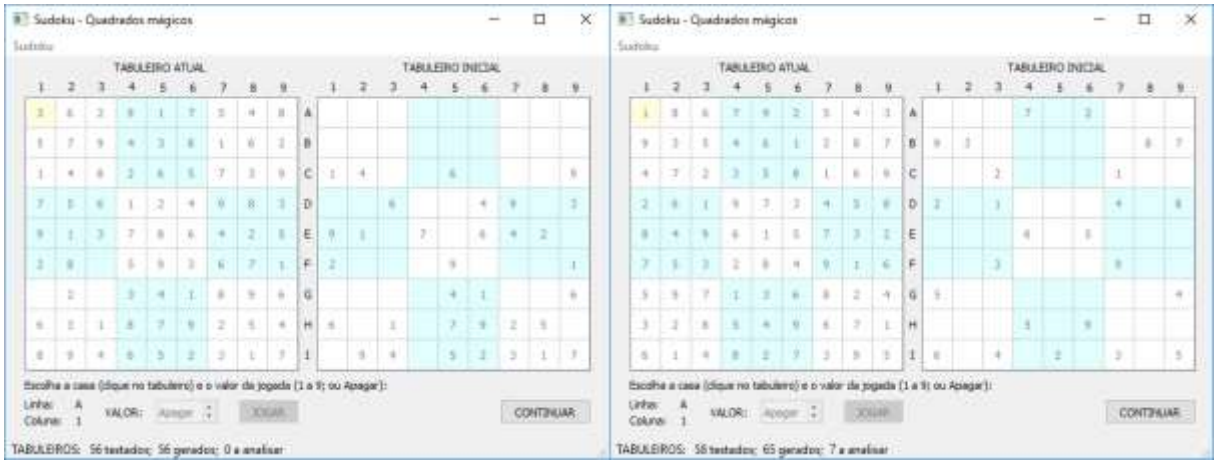


Figura 6 - Exibição final do algoritmo de resolução, antes da confirmação, sem e com solução encontrada.

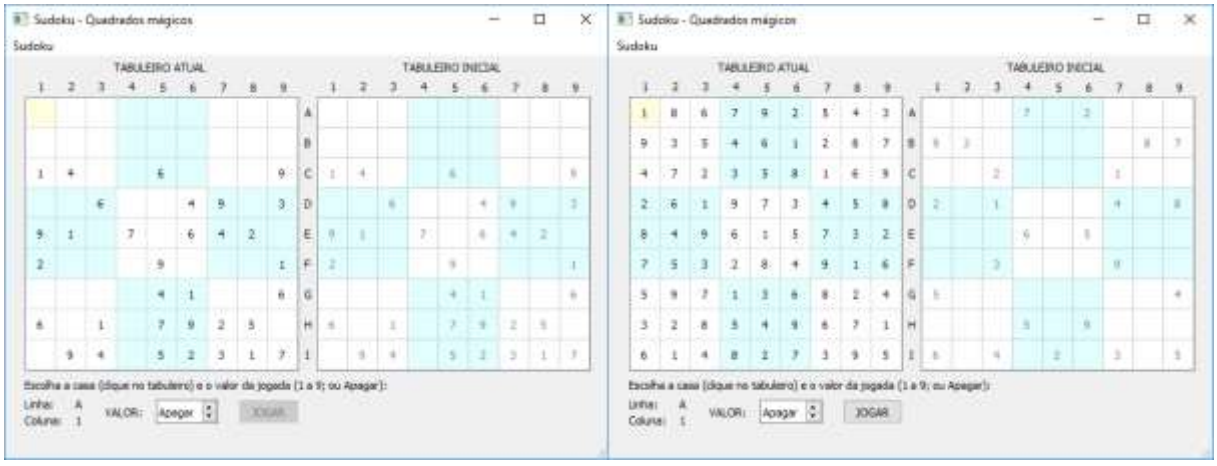


Figura 7 - Exibição após confirmação do algoritmo de resolução, sem e com solução encontrada.