UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

LICENCIATURA EM COMUNICAÇÃO E MULTIMÉDIA

FUNDAMENTOS DE ALGORITMIA E PROGRAMAÇÃO

Trabalho Prático - Sudoku

Ano Letivo 2022/2023

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Ana Ferreira, al72063

Gonçalo Carneiro, al62702

João Roxo, al71741

Índice

INTRODUÇÃO

No âmbito da unidade curricular Fundamentos de Algoritmia e Programação, foi-nos proposto desenvolver um programa em Portugol Studio, e nós optamos por fazer o jogo Sudoku, com recurso ao modo gráfico.

Para realizar o trabalho aplicamos tudo aquilo que aprendemos nas aulas e recorremos ainda à página de ajuda e aos projetos disponíveis como exemplo no próprio Portugol Studio para percebermos melhor o que cada função faz, quais as bibliotecas necessárias e, ainda, como funciona o modo gráfico, uma vez que não abordamos esta componente nas aulas.

O Sudoku é um jogo de raciocínio que consiste numa tabela 9x9 e o objetivo do jogo é que o jogador preencha toda a tabela com número de 1 a 9, sem que haja quaisquer repetições na mesma linha ou coluna.

Ao executar o programa, o jogador terá a possibilidade de selecionar o nível de dificuldade pretendido: principiante, básico ou profissional. Em função do grau de dificuldade, deverão ser preenchidas algumas células, respeitando as seguintes condições:

- Nível principiante: Preencher 35 células (cada área terá entre 1 e 7 células preenchidas);
- Nível básico: Preencher 30 células (cada área terá entre 1 e 6 células preenchidas);
- Nível profissional: Preencher 25 células (cada área terá entre 1 e 5 células preenchidas).

Portugol Studio

Portugol Studio é um simulador de linguagem algorítmica desenvolvido para apoio às aulas de Introdução à Programação, voltada para pessoas que falam o idioma português. Tem uma sintaxe fácil baseada em C e PHP e conta com diversos exemplos e materiais de apoio à aprendizagem e possibilita a criação de jogos e outros programas mais complexos com o uso de interface gráfica. A interface do programa é didática e simplificada, possui um sistema de ajuda que permite ao utilizador aprender a linguagem e como programar dentro da ferramenta, um depurador que permite executar o programa passo a passo e um inspetor de variáveis que permite a visualização dos valores das variáveis do código durante a execução ou depuração.

Estrutura

Para criar um programa em Portugol devemos seguir uma certa estrutura e algumas regras. O programa deve **sempre** incluir o comando *programa* e a função *início*.

Bibliotecas

Na elaboração de um algoritmo existe a possibilidade de utilizar um conjunto de funções e comandos já existentes que se dá o nome de Bibliotecas. Estes Bibliotecas contêm códigos e dados auxiliares que permitem o compartilhamento e a alteração de código e dados de forma modular. Existem vários tipos de bibliotecas, cada uma com funções para resolver determinados problemas.

Para utilizar uma biblioteca primeiro é necessário importá-la para o nosso programa, através do seguinte comando (utilizando a biblioteca útil como exemplo):

```
inclua biblioteca Util --> u
```

Para a realização do nosso programa utilizamos as seguintes bibliotecas:

Biblioteca Gráficos: permite inicializar e utilizar um ambiente gráfico com suporte ao desenho de primitivas gráficas e de imagens carregadas do sistema de arquivos. (abreviação **g**)

Biblioteca Tipos: contém funções que permitem converter os tipos de dado do Portugol entre si. (abreviação **ti**)

Biblioteca Teclado: contém um conjunto de funções para manipular a entrada de dados através do teclado do computador. Esta biblioteca não funciona no console de entrada e saída de dados do Portugol Studio, apenas funciona com a biblioteca Gráficos, se o modo gráfico estiver iniciado. (abreviação **te**)

Biblioteca Útil: contém diversas funções utilitárias, por exemplo, uma função que sorteia um número aleatório entre os valores mínimo e máximo especificado ou uma que descobre o número de elementos existentes num vetor. (abreviação **u**)

Biblioteca Mouse: contém um conjunto de funções para manipular a entrada de dados através do rato do computador. Esta biblioteca não funciona no console de entrada e saída de dados do Portugol Studio, ela só funciona com a biblioteca Gráficos, se o modo gráfico estiver iniciado. (abreviação **m**)

Funções

As funções, como aprendemos nas aulas teóricas, são pequenos programas onde se definem regras específicas e que podem, ou não, retornar valores. Ao encontrar a chamada de uma função, o programa principal será direcionado para a respetiva função e executará o que nela estiver definido. Executada a função, o programa prossegue com as instruções do programa principal. Algumas das vantagens da utilização de funções durante a programação são a redução de código duplicado num programa; a decomposição de problemas grandes em pequenas partes; melhorar a interpretação visual de um programa e esconder ou regular uma parte de um programa, mantendo o restante código alheio às questões internas resolvidas dentro dessa função.

Ao realizar o código do jogo utilizamos diversas funções que iremos explicar de seguida para que servem e o que fazem.

inicio()

A função inicio() regula todo o código, expressando a ordem inicial do código para as 3 funções principais:

- inicializar()
- executar()
- finalizar()

inicializar()

Esta função é responsável por iniciar o modo gráfico, estipular o tamanho da janela e o nome da mesma.

Para o desenvolvimento do ficheiro principal, auxiliamo-nos poucos vídeos do Youtube para encontrar informação pertinente relativamente áquilo que pretendíamos executar. Aprendemos que poderíamos escolher a cor de fundo da janela inicial, mas por algum motivo que não conseguimos apurar, não conseguimos executar o resultado inicialmente pretendido. Deixamos assim a cor de fundo branca e desenhamos um quadrado do mesmo tamanho da janela, o que será mencionado mais tarde.

finalizar()

Como próprio nome indica, fecha o jogo, neste caso em específico, termina o modo gráfico.

Não é necessário ser colocado, mas é boa política fazê-lo pois permite que o modo gráfico encerre antes de encerrar o próprio programa.

executar()

Dentro desta função temos um ciclo se repetição ENQUANTO, que é extremamente importante, pois todo o seu conteúdo continuará a ser executando enquanto a tecla ESC do teclado não for pressionada.

Esta foi uma das partes que aprendemos através de um dos exemplos disponíveis no próprio *Portugol Studio* (Secção Exemplos, pasta Música, programa Bateria).

Este ENQUANTO permite manter o modo gráfico aberto uma vez que existe um loop infinito, mas não permite atualizar constantemente em tempo real todo o gráfico desenhado.

De seguida temos as 3 funções principais e que são as veias deste Programa:

- confirmar_mouse_hover()
- tratar_cliques()
- desenhar()

confirmar_mouse_hover()

É a função responsável para no indicar exatamente onde o cursor do rato está, face à localização dos diferentes componentes desenhados no modo gráfico.

Adaptamos esta função do programa "Bateria" do *Portugol Studio*.

Conseguir adaptar e conseguir visualizar como iria ficar o resultado final, foi um processo complicado mas que conseguimos realizar com sucesso. Este segue as variáveis base do tamanho de cada quadrado da matriz Sudoku pelo que se adapta ao tamanho da grelha e é sempre possível localizar onde se encontra o cursor.

Consoante a localização do cursor, a variável matriz_jogo[linha X][coluna X] fica verdadeira para o quadrado da linha X e coluna X.

tratar_cliques()

Idêntico à função de cima, e dependente da mesma, esta função permite descobrir se o cursor clicou, ou seja, se pressionou um botão do rato e se o largou dentro do quadrado da matriz Sudoku.

Para tal, o código primeiro confirma se o cursor se situa dentro da matriz e consegue identificar qual o botão foi clicado, direito ou esquerdo, havendo uma ação diferente pra cada um.

O Programa guarda essa informação nas suas respetivas variáveis, anulando também a do outro botão, o que permite selecionar um quadrado previamente selecionado pelo botão do rato oposto, sem consequência para a utilização normal do jogo.

As variáveis que permitem isto acontecer são: inteiro Xclicado e inteiro Yclicado.

Estas variáveis guardam o valor, respetivamente, de Xhover e Yhover, responsáveis por dizer qual a coluna e linha, respetivamente, que o cursor está por cima.

Caso seja carregado fora da grelha, o sistema coloca estas variáveis com o valor = -1, valor negativo e impossível de obter normalmente para representar as linhas e colunas, o que orienta o Programa mais à frente a confirmar quando houver ou não clique dentro de uma zona válida.

desenhar()

Esta função, como o próprio indica, é a responsável pelos desenhos e toda a parte gráfica do programa

Todas as funções executadas dentro desta função desenham diferentes partes da mesma em diferentes situações, consoante a informação previamente dada pelo cursor do rato.

De modo a garantir que cada quadrado da matriz é desenhada do modo correto, antes de executar as diferentes funções (à exceção da função desenhar_fundo_branco()) são executadas duas linhas de repetição PARA. Isto garante que o código se repete para cada Linha e para cada Coluna da matriz, e como estão todos ligados pelas mesma variáveis, todas as diferentes funções estão em sintonia constante, e caso haja alguma alteração mais tarde de alguma das variáveis mãe, não será necessário alterar diretamente as restantes funções.

desenhar_fundo_branco()

Como foi mencionado acima, cria o quadrado branco de tamanho igual à tela do modo gráfico, de modo a cobrir e servir de fundo.

desenhar_clicou_direito()

Esta função só é iniciada se for confirmado que houve um clique do botão direito do rato, se o cursor está por cima da matriz e se a matriz **matriz_niveis[linha][coluna] = falso**, isto tudo enquanto confirma se o clique e o cursor do rato foram executados na mesma linha e na mesma coluna.

É um conjunto de código e de confirmações eficaz, e foi um grande desafio colocá-lo a funcionar.

Confirmada a condição e entrando na função em si, no interior da mesma temos 1 repetição "PARA". Essa repetição confirma qual a tecla clicada ao confirmar os 9 valores possíveis, estando constantemente a ser revista e atualizada por causa dos valores ClicadoY e ClicadoX guardados nas funções acima.

Isto permite o programa saber que ouve clique ali e que foi válido, e considerar o clique ativo até novo clique seja efetuado e sobreponha o antigo, o que o deixa continuar a trabalhar sem comprometer os restantes trabalhos.

A grande diferença entre esta função e a abaixo, ao clicar com o lado esquerdo, é que esta permite 9 espaços para guardar informação, pelo que se clicarmos em outros números, no código que permite desenhá-los vão aparecer todos os clicados, de 1 a 9 possibilidades, enquanto que quando clicado com o lado esquerdo, qualquer outro numero clicado substitui o antigo, de modo a ficar apenas um.

De modo a saber se o clique foi bem-sucedido, a cor do quadrado respetivo muda para orientar o utilizador que pode submeter um valor.

Caso o valor seja incorreto (letras, símbolos, tudo o que não for um número inteiro de 1 a 9 ou o botão DELETE ou de LIMPAR) o sistema continua aberto à espera que seja submetido um valor válido.

Esta parte é extremamente importante pois, como mais à frente vai ser mencionado, temos uma função que muda a cor do quadrado quando o cursor está por cima dele. Isto permite ao jogador saber se está numa zona clicável, pelo que é importante que continue a ser executado para os outros quadrados independentemente do quadrado que já tenha sido selecionado.

Isto não acontecia se fosse executada a função **te.ler_tecla()**, que pára o avanço do código até ser carregada uma tecla. Para evitar isto, é primeiro confirmar se houve alguma tecla pressionada, com a função **te.alguma_tecla_pressionada()**, e depois sim vemos qual foi a tecla, não havendo a quebra de espera da submissão pois já foi dada, permitindo ao sistema continuar a executar o código enquanto aguarda uma tecla ser carrega.

Para concluir, o sistema tem de confirmar se a tecla é a correta, pelo que aceita números de 1 a 9, mas como existem dois tipos de números (consideraremos os "normais", aqueles que estão representados a cima do teclado de texto, e os "numéricos" que se localizam no Numpad, normalmente representada à direita de alguns teclados / portáteis), terá de considerar ambos como possíveis respostas.

Sempre que se carrega uma tecla, a função **te.ler_tecla()** retorna um número inteiro positivo, e é esse número que consegue passar pela **função te.caracter_tecla**(X) e ser convertido para um número de 1 a 9 do tipo caracter. Como seria de esperar, o número positivo de um número "normal" não é igual ao seu semelhante "numérico".

Infelizmente vemos um erro no Software to Portugol Studio, pois ao fazer a conversão, o caracter dado por um número "normal" não é igual ao seu semelhante "numérico", sendo na realidade assumido que representa letras, em vez de números.

Foi confirmado esta situação várias vezes e mesmo na secção Ajuda do Portugol Studio os valores inteiros correspondentes a cada um dos caracteres é diferente e correto, mas a função que converte para caracteres não está a funcionar corretamente para os semelhantes "numéricos".

Para corrigir a situação usei a função **ajustar_num()**, que confirma se foi clicada o semelhante "numérico" e, caso seja, altera o valor inteiro para o seu semelhante "normal".

Concluindo, é possível também eliminar o conteúdo de uma célula com tecla DELETE ou a tecla LIMPAR (para esclarecer, é a tecla com a seta para a esquerda que usamos para eliminar texto nos ficheiros Word, por exemplo).

desenhar_clicou_esquerda()

Idêntico à função de cima, com as variações já mencionadas, esta função permite dar ao sistema, a informação, se foi submetido um valor válido após ter-se clicado com o lado esquerdo do rato. Caso seja válido, vai guardar essa informação, marcando o quadrado como estando ocupado através de uma variável lógica (**matriz_escolhas[linha][coluna]**, idêntico ao de cima que usa a **matriz_provisorio[(linha*3) + mini_y_prov][(coluna*3) + mini_x_prov]**, com a variação que este ultimo, à medida que confirma se o número for válido, também avança na zona onde vai guardar o lugar ocupado, permitindo que eles sejam escritos em zonas diferentes) e guardando o valor com outra variável: **numero_aceite[linha][coluna]** (sendo necessário converter primeiro para cadeia)

desenha_cursor_cima()

É a função que lê a posição do cursor do "rato" e marca alterando a cor, caso o utilizador esteja sobre uma posição possível de clicar.

desenha_esqueleto()

Função que cria a matriz externa do Sudoku, permitindo o efeito de grelha 9x9 que o utilizador vai visualizar assim que escolhe a dificuldade do nível pretendido.

desenha_quadrado_permanente()

Com esta função são desenhados todos os quadrados que estão previamente bloqueados e que são mostrados no início do nível, marcando-os de maneira diferente para serem mais facilmente visualizados.

desenha_valor_submetido()

Função que permite ir buscar o valor previamente submetido pelo utilizador após o mesmo clicar no lado esquerdo do rato, após isso o valor é desenhado no quadrado respetivo.

desenha_valor_provisorio()

Tal como a função anterior, porém os valores são submetidos após ser feito o clique com o botão direito do rato.

Níveis

Para selecionar o nível, o sistema confirma dentro da função **desenhar()** se já foi escolhido um nível.

Caso o nível tenha sido selecionado, o código segue as diferentes funções acima mencionadas. Pelo contrário, se não tiver sido selecionado um nível, obriga o utilizador a fazê-lo, antes de poder começar a jogar.

escolher_nivel()

Esta função inicia todo o processo de escolha do nível e, após a sua escolha, inicia a criação aleatória de uma solução 9x9 do Sudoku, mas já vemos isso mais à frente.

Aqui será dada a informação de qual nível pretendemos escolher na consola do Portugol Studio. Infelizmente devido ao tempo que demorou algumas secções do código, e como priorizamos a funcionalidade à estética, acabou por não dar para colocar esta parte no modo gráfico.

Existe 3 níveis:

- Fácil (bloqueia 35 células da matriz)
- Médio (bloqueia 30 células da matriz)
- Difícil (bloqueia 25 células da matriz)

sorteia_valores_aleatorios()

Esta é uma das funções mais importantes de todo o código.

Foi possível a sua execução com alguma orientação por parte do professor Joaquim Sousa que nos recomendou este website (https://github.com/vaithak/Sudoku-Generator).

Ao contrário do que se possa pensar, ter acesso ao código não facilitou em nada o processo de criação da função, visto que as linguagens de programação são diferentes, pelo que foi necessário proceder a várias pesquisas e descobrir soluções para os problemas dentro das diferentes funcionalidades entre a linguagem C++ (utilizada pelo código presente no link acima) e o Portugol Studio.

Depois de resolver o problema, começamos, por aleatoriamente preencher um vetor **adivinha_num[]** com valores de 1 a 9, sem serem repetidos, pois serão os números a ser utilizados para preencher a matriz do Sudoku.

resolver_grelha_sudoku()

Depois de várias tentativas para a criação de funções que permitisse criar um puzzle inteiro, dentro das regras do Sudoku, esta foi a única bem sucedida, dentro das outras tentativas, que mais à frente iremos comentar.

encontrar_quadrado_sem_valor()

Esta função permite confirmar se o quadrado em questão tem algum valor, ou seja, se já lhe foi atribuído algum valor.

É uma das mais importantes, pois permite constantemente retornar a linha e coluna do quadrado do Sudoku que não tem um valor atribuído, alterando permanentemente os valores inicialmente introduzidos de "row" e "col", dentro da função **resolver_grelha_sudoku()**, garantindo que o normal recorrer do código, lhe possa ser atribuído um valor.

A atribuição do novo valor, dentro da função **resolver_grelha_sudoku()**, passa por um sistema de repetição "PARA" que vai circular entre os números possíveis, até um máximo de 9x.

testa_regras()

Esta função é responsável por confirmar se o número em questão segue as 3 regras do Sudoku, que são as seguintes:

- não repetir números na mesma coluna;
- não repetir números na mesma linha;
- não repetir números no mesmo quadrado grande.

Isto é possível com a ajuda de três outras funções, dispostas na mesma ordem das regras:

- **usado_linha;**
- **usado_coluna;**
- **usado_quadrado_grande**.

Cada uma delas confirma a regra e só avança para o próximo quadrado se estiver tudo correto.

Haverá momentos em que nenhum valor é possível dentro das regras. Isso faz com que o código limpe qualquer valor dado para aquela linha e coluna, retornando-o como falso.

Devido à estrutura do código, sempre que um número é aceite e lhe é permitido avançar, ele retorna a si mesmo, como se o mesmo se estivesse a "enrolar". Isto é importantíssimo pois na conversão do código original, o facto de verificarmos que o código retornava valores lógicos sem nunca serem chamados, ou terem alguma variável que fosse capaz de armazenar esses valores, era algo estranho e que não fazia sentido.

Após serem feitos alguns testes, linha a linha, com a conversão direta chegamos á conclusão, que o código parava sempre na linha 1, coluna 6, sem conseguir guardar qualquer valor nesse quadrado.

Isto permitiu-nos seguir o código e assim conseguimos perceber como ele se "enrolava em si mesmo". O retorna falso ou verdadeiro não era para alguma variável ou função externa, mas sim para o próprio código, e era o que permitia voltar atrás à medida que batia num quadrado sem aparente solução.

Sistema Backtracking

O sistema de “backtracking”(ver Figura 1 em Anexos) foi de extrema importância, porém, tinha um grande problema.

Com a conversão direta do código original, após retornar a nova coluna e linha da função **encontrar_quadrado_sem_valor()** o código mantinha os mesmos valores, mesmo quando “desenrolava”, não alterava os valores para os quadrados anteriores (objetivo principal do “backtracking”). Isto acontecia até sair da função, o que obrigava o Programa a preencher o resto do Sudoku com espaços vazios.

De modo a impedir que isto acontecesse, foi colocada a função **manter_valores()**, que confirma quando o código chega ao último quadrado (importante para não dar erro, pois obriga-o a sair da função e a continuar o código quando todos os quadrados ficam preenchidos) e como é usado no “backtracking”, sempre que “col” = 0, retira um valor à variável “row” e colocar “col” = 8, senão retira apenas um valor a “col”, alterando novamente os valores de “row” e “col” antes de “desenrolar”, permitindo que o sistema esteja sempre com a coluna e a linha atualizados e corretos.

Esta solução foi testada em múltiplas situações e as regras foram bem implementadas, pelo que no final, o sistema “desenrolava” sempre dentro da condição “SE”, não alterando mais nada nos valores que foram submetidos, até sair da função e continuar com o resto do Programa.

sorteia_definitivo()

Com o puzzle criado, restou-nos decidir quais seriam os quadrados iniciais visíveis para o utilizador.

Esta função é considerada simples pois é, auxiliada por uma repetição “PARA” que repete com base na dificuldade selecionada, de modo a selecionar aleatoriamente “X” quadrados no meio dos 81 possíveis.

Durante as repetições é dado um valor aleatório entre 0 e 8 para a linha e coluna (para matrizes e vetores o primeiro valor a ser guardado tem index 0, daí não ser entre 1 e 9, permitindo guardar os 9 valores normalmente), seguido de confirmação para garantir que esse quadrado escolhido não foi bloqueado.

Caso esteja livre, é bloqueado esse quadrado e é concluído o ciclo da repetição “PARA”, repetindo as vezes necessárias para ter “X” quadrados diferentes, de acordo com o nível escolhido.

Concluída esta etapa, é atribuído o valor verdadeiro à variável **escolheu_nivel** e o código consegue finalmente desenhar o jogo Sudoku, permitindo que o utilizador possa interagir com o puzzle.

Validar

Quando o Utilizador sentiu que concluiu o jogo, pode carregar no botão Validar. Caso não esteja correto, não acontece nada, mas se tiver correto, o sistema desenha "PARABÉNS" em letras grandes e bloqueia o código por 3 segundos, antes de voltar a avançar de volta à matriz de jogo.

Como a **função validar()** compara a matriz **numero_aceite[][]** (valores submetidos pelo Utilizador) com a matriz **numeros_nivel[][]** (valores aleatoriamente gerados pelo Programa), é necessário acrescentar os valores bloqueados pelo sistema para que a confirmação seja concluída, pois o Utilizador não consegue preencher esses quadrados já preenchidos e isso implicaria que o sistema consideraria como estando errado sempre.

Voltar

Como o nome indica permite ao sistema voltar atrás e selecionar novo nível com novos resultados.

Para tal, é necessário que o Programa reinicia os valores armazenados nas matrizes responsáveis pela solução do Sudoku (**numeros_nivel[][]**) e por bloquear os quadrados iniciais (**matriz_niveis[][]**)

Estética

A forma como fizemos a estética do nosso Sudoku foi bem pensada para qualquer pessoa poder jogar sem problema e sem ter dificuldades quando estiver a se divertir (ver Figura 2 a 5 em Anexos)

Quando executamos o Programa deparamo-nos logo com 3 opções para jogar.

Decidimos colocar estes 3 modos para todo o tipo de utilizador poder usar. Os botões do menu foram colocados ao centro para melhor visualização dos mesmos.

Assim que o Utilizador clica em uma das opções é apresentada a grelha de jogo.

Neste momento é possível tanto jogar como voltar para trás se quisermos selecionar outro nível, e só nos é possível usar o botão "Validar" se o Sudoku for terminado corretamente.

Caso isso aconteça é apresentada a Mensagem a dizer "PARABÉNS" e uns segundos depois a janela volta ao sudoku terminado, podendo voltar a jogar outro nível.

Código

Todo o código está comentado de modo a melhor orientar qualquer individuo que leia o código, mesmo sem contextualização.

No início do código, declarámos várias variáveis e constantes, que serão utilizadas pelas matrizes e o tamanho da grelha do Sudoku em si.

Em anexos colocámos o código usado e funcional e imagens exemplo.

É possível ver que ao longo dessa zona temos várias matrizes com nomes diferentes, sendo estas responsáveis por armazenar e confirmar qual dos quadrados serão usados, da seguinte forma:

- **matriz_niveis[][] :** Confirma quais os quadrados dentro da grelha Sudoku que devem ser bloqueados e mostrados ao utilizador após escolha por parte do mesmo do nível que pretende;
- **numeros_nivel[][]** : Números apresentados em cadeia (*string*) que armazenam o resultado final do Sudoku. Serve para se confirmar se o jogador conseguiu ou não concluir o puzzle;
- **matriz_jogo[][]:** Responsável por informar quando o cursor entra na matriz desenhada do Sudoku e quando se desloca dentro dela, mudando para verdadeiro a linha e coluna correspondentes, abrindo uma possibilidade de ações consoante as intenções do jogador;
- **matriz_escolhas[][]:** Como próprio nome indica, confirma se o jogador selecionou algum quadro com intensão de guardar informação aceite pelo programa. Se sim, fica verdadeiro o quadrado em questão, permitindo ao programa escrever e manter escrito a informação que foi submetida;
- **numero_aceite[][]:** Capaz de guardar informação sobre os valores submetidos pelo jogador de modo a serem usados como meio de comparação com a matriz números_nivel[][] e assim o sistema saber se a conclusão do Sudoku foi válida ou se é necessário corrigir os valores errados;
- **matriz_provisorio[][]:** uma grande matriz usada para colocar os valores provisórios dentro de cada quadrado do Sudoku. Tem vários valores possível pois, ao contrário da tradicional matriz 9x9, esta precisa de contar que por cada quadrado tem de possibilitar a existência de 9 valores, ou seja, matrizes 3x3 dentro de cada quadrado, o que no final da 9 x 3 = 27 possibilidades para cada coluna e linha. De resto, funciona igual às outras matrizes para informar o sistema que é para desenhar um determinado número.

Para além destas estruturas de armazenamento, temos a variável x_un_table e y_un_table. Ambes guardam a informação do tamanho de cada quadro da matriz do Sudoku, respetivamente para o X e para o Y. Isto permite que se possa usar o nome da variável para todas as linhas de código e, caso seja necessário mudar o valor dos quadrados, basta mudar diretamente nestas variáveis que não entrará em conflito com o código previamente escrito, permitindo alterar o tamanho da grelha sem consequências.

Dificuldades ao longo do desenvolvimento do programa

Durante o desenvolvimento do programa deparamo-nos com alguns problemas.

O maior problema, e também o que nos dificultou mais foi conseguir gerar números aleatoriamente na grelha do Sudoku.

Foram tentadas várias soluções possíveis, sendo a primeira uma tentativa de apresentar valores aleatórios ao longo da grelha, da esquerda para a direita, de baixo para cima, sempre a confirmar com os dados previamente colocados se as regras estavam a ser cumpridas.

Não resultou devido à impossibilidade de o código voltar para trás e corrigir possíveis erros, problema recorrente com as outras soluções.

A segunda hipótese foi idêntica em termos de resultado, mas com uma abordagem. A solução passou em colocar primeiro todos os valores de um número, até preencher todas as posições possíveis para aquele número, e repetir o processo para os outros.

Pensou-se que isto poderia ultrapassar a necessidade de voltar atrás, mas não foi o caso, pelo que tivemos de arranjar outra solução.

A terceira tentativa envolveu usar uma Inteligência Artificial (ChatGPT) onde tentamos pedir uma solução para o código. A AI gerou um o código, porém, como nós, não foi capaz de prever a necessidade de ter de voltar atrás e reconfirmar valores, pelo que acabou por dar o mesmo resultado.

Por fim relatamos a situação ao Professor Joaquim Sousa, que prontamente nos orientou.

Com o as novas indicações, fizemos a conversão de um sistema em C++ para o Portugol Studio, conseguindo finalmente arranjar uma solução para gerar a grelha aleatoriamente e com precisão.

CONCLUSÃO

Ao longo do desenvolvimento deste trabalho, encontramos várias dificuldades, mas no final conseguimos obter o resultado desejado.

Infelizmente apercebemo-nos que Portugol Studio aparenta ter algumas irregularidades, dando diferentes resultados para um mesmo código (sem contar quando isto era pretendido), mas no geral foi uma experiencia enriquecedora que levamos connosco.

Estamos orgulhosos com o nosso trabalho e esperamos um dia poder utilizar as novas skills que aprendemos para crescermos mais como pessoas e profissionais.

ANEXOS

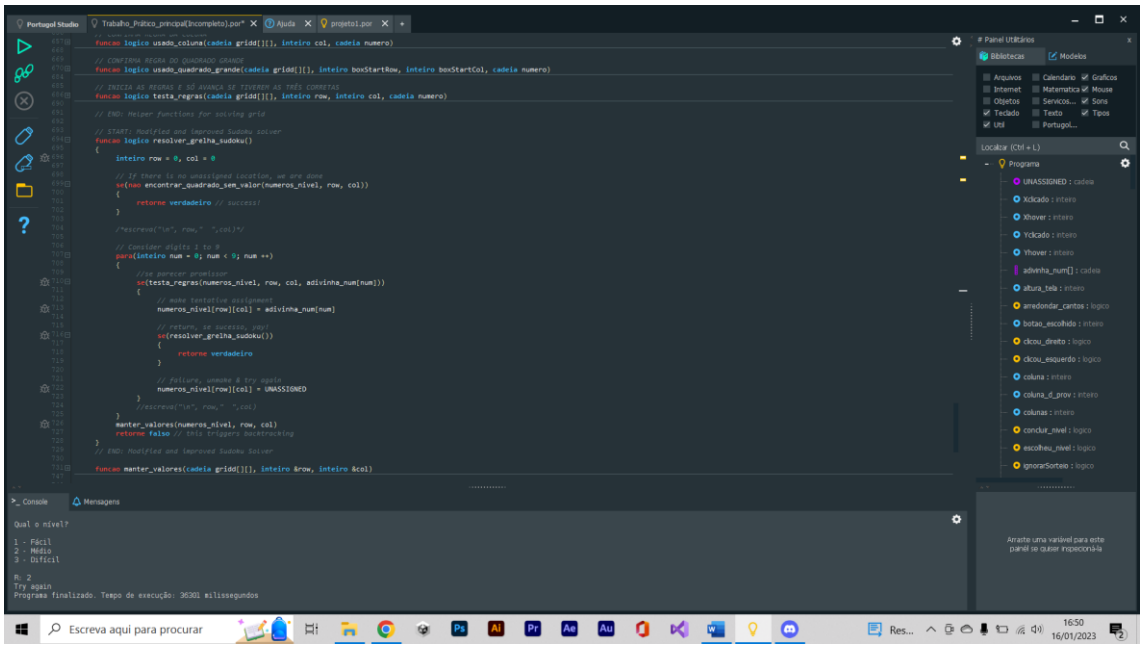

Figura 1 – Código responsável pelo "backtracking"

Figura 2 - Exemplo do resultado Final(1)

Figura 3 - Exemplo do resultado Final(2)

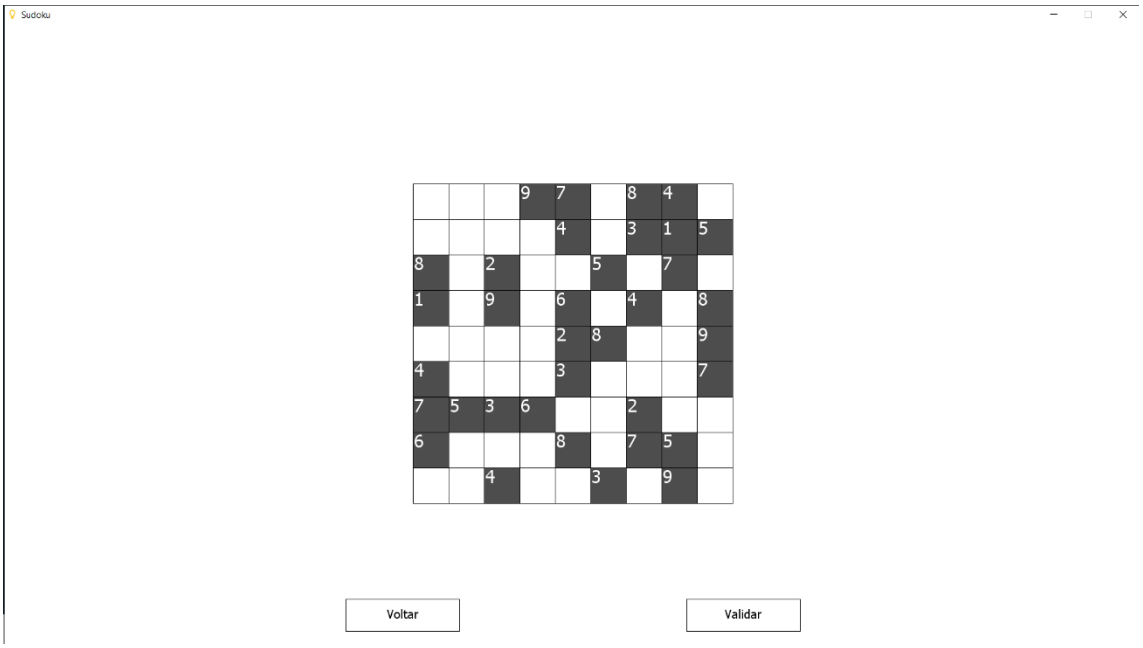

Figura 4 - Exemplo do resultado Final(3)

Figura 5 - Exemplo do resultado Final(4)