UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO

TRABALHO FINAL:
SUDOKU

MANAUS/AM
2013

Equipe:

ANTÔNIO CÉSAR VIEIRA DA CRUZ JUNIOR - 21203278

LUCAS ROCHA SAHDO - 21200812

TRABALHO FINAL:
SUDOKU

Trabalho destinado à matéria Algoritmos e Estruturas de Dados 1 como critério para obtenção de nota final do 2° período do curso de engenharia da computação.

MANAUS/AM

2013

ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

O nome Sudoku é a abreviação japonesa para a frase "*suuji wa dokushin ni kagiru",* que significa "os dígitos devem permanecer únicos".

O jogo é uma grade de dimensão 9 x 9 constituída de subgrades de dimensão 3 x 3 chamadas de regiões. Algumas células já contêm números, chamadas como números dados ou pistas. O objetivo é preencher as células vazias, com um número em cada célula, de maneira que cada coluna, linha e região contenham os números naturais 1 a 9 apenas uma vez. Portanto, na solução do jogo, cada número aparece apenas uma vez em qualquer um dos sentidos ou regiões, daí, portanto "únicos números" originaram o nome do jogo ou enigma.

As primeiras publicações do Sudoku ocorreram nos Estados Unidos no final dos anos 70. A editora deu ao jogo o nome de Number Place, que é usado até hoje nos Estados Unidos. Em 1984, a Nikoli, maior empresa japonesa de quebra-cabeças, descobriu o Number Place e decidiu levá-lo ao Japão.

2. IMPLEMENTAÇÃO

2.1 As bibliotecas

As bibliotecas que foram usadas para desenvolver o jogo foram as seguintes:

2.1.1 Bibliotecas padrão

- #include <stdio.h>

A mais comum das bibliotecas do c. Conhecida pela função "printf()".

- #include <stdlib.h>

Desta biblioteca utilizou-se a função "rand()", que gera valores aleatórios

- #include <stdio_ext.h>

Desta biblioteca especial utilizou-se a função "__fpurge(stdin)", que limpa o buffer do teclado, impedindo que o programa acesse "lixo" ao solicitar entrada de dados do usuário pelo teclado.

2.1.2 Bibliotecas alternativas

- #include "SahdoLib.h"

Possui diversas funções gráficas com manipulação de coordenadas da tela e cores.

- #include "Partidas.h"

Contém várias partidas de sudoku divididas em nível fácil, médio e difícil.

2.2 As definições - defines

A definição na linguagem C é um recurso que te permite nomear valores para melhor compreensão do programa, pois à medida que o código vai crescendo, fica difícil identificar o objetivo de certo valor que se está utilizando em determinado trecho de código, por isso, usa-se a definição.

Abaixo estão as definições utilizadas no jogo com suas respectivas funções, lembrando que o nome de cada definição já é bem intuitivo, deixando claro o objetivo de cada uma.

2.2.1 Definições sobre a matriz 9x9

- #define ORDEM 9

2.2.2 Definições booleanas

A linguagem c não possui o tipo booleano, mas considera o inteiro “0” como falso e diferente de “0” como verdadeiro, logo, criamos definições chamadas pelos nomes VERDADEIRO e FALSO, que correspondem respectivamente aos valores numéricos “0” e “1”.

- #define VERDADEIRO 1
- #define FALSO 0

2.2.3 Definições auxiliares

Essas definições servem de auxílio para a variável “sinal”, localizada dentro da estrutura de dados “Matriz”. O nomeação dos números deixa o código mais limpo e organizado.

- #define LIVRE 2
- #define USUARIO 1
- #define AUTOMATICO 0

2.2.4 Definições sobre o nível do jogo

Essas definições servem de auxílio para a seleção do nível de dificuldade do jogo. Os números também foram nomeados devido à estética do código e sua organização.

- #define FACIL 0
- #define MEDIO 1
- #define DIFICIL 2
- #define RESOLVIDO 3

2.3 As variáveis globais

As variáveis globais que foram utilizadas no programa do jogo são as seguintes:

- int CoordX

Coordenada horizontal inicial da tabela de sudoku na tela de shell do linux.

- int CoordY

Coordenada vertical inicial da tabela de sudoku na tela de shell do linux.

- int varNivel

Variável que recebe o nível do jogo na função “Nivel()”. A partir desta variável que se construiu a lógica de carregar jogos fáceis, médios e difíceis para o programa.

- matriz* MatSudoku

Lista principal do programa que recebe, ao iniciar o jogo, todos os valores iniciais do jogo de sudoku.

- matriz* MatSudokuResolvidoBackUp;

Lista que recebe a solução do jogo atual de sudoku e deixa armazenada.

- matriz* HistoricoJogadas;

Lista que recebe todas as jogadas válidas do usuário durante uma partida.

2.4 A estrutura de dados

A estrutura de dados foi criada de forma que se pudesse criar uma lista encadeada com características de uma matriz quadrada comum, ou seja, com linhas e colunas e o valor a ser armazenado. A vantagem nisso é que ganhamos espaço para salvar outras informações da célula na matriz além da linha e da coluna, pois a estrutura permite você agregar quantas informaçõe você desejar. Na figura 1 temos a estrutura de dados criada para satisfazer a lógica e a às necessidades do programa desejado.

```
typedef struct Matriz
{
        int num; //valor de cada célula da matriz
        int lin,col; //linha e coluna da matriz
        int sinal; //AUTOMATICO - valores inseridos
        pelo programa | USUARIO - valores inseridos
        pelo usuario | LIVRE - valores que se pode
        inserir (= 0)
        struct Matriz* prox;

} matriz;
```

Figura 1 – Estrutura de dados do programa

A lista criada a partir da estrutura recebe oitenta e um valores correspondentes ao número de células de uma tabela de sudoku clássico. A figura 2 ilustra como é a configuração desta lista após iniciar o jogo e como os dados se organizam desde a cabeça da lista até a calda da mesma.

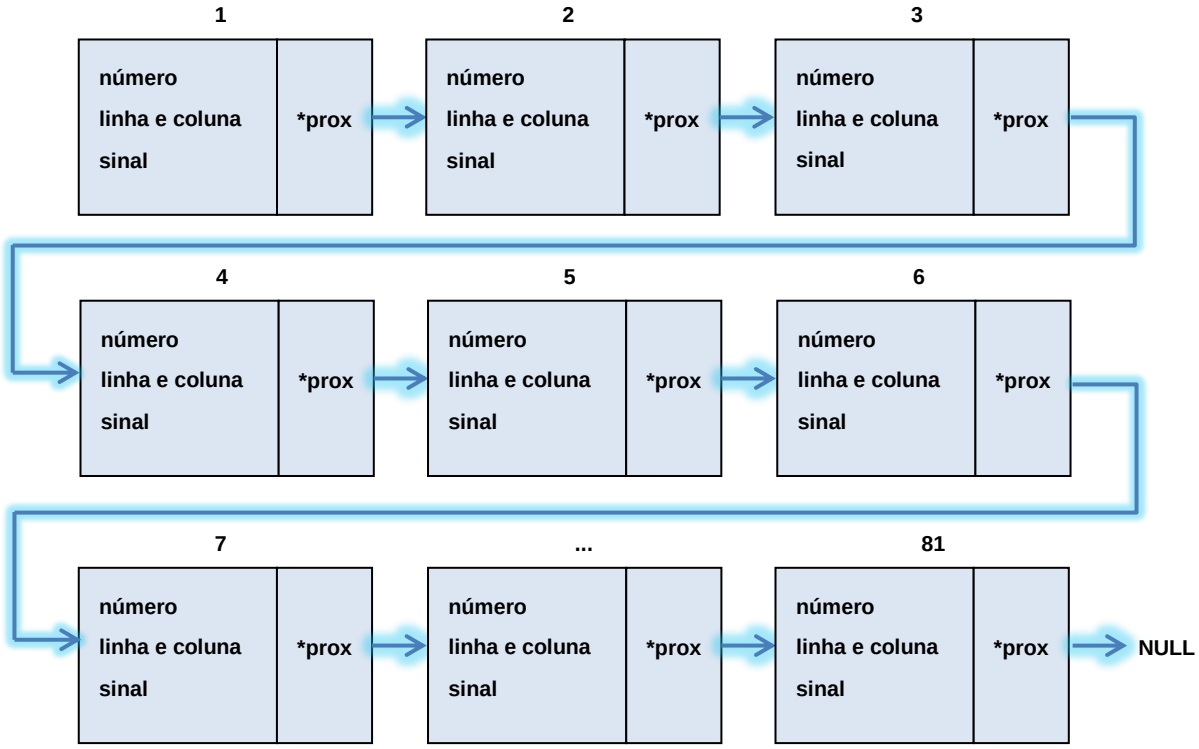

Figura 2 – Lista encadeada principal do jogo

2.5 As funções

2.5.1 Funções estéticas

- void ImprimeMatriz(matriz *mat)

Esta função, como intuitivamente se percebe pelo nome, tem a função de imprimir cada valor da matriz em uma tabela de Sudoku. Nesta função está definida também toda a lógica de cores. As cores são definidas a partir da biblioteca “SahdoLib.h”

- void ImprimeHistorico(matriz *l)

Esta função tem como objetivo imprimir uma lista passada por parâmetro, que no nosso caso é a lista onde foram salvas cada jogada que o usuário fez durante o jogo.

- void Menu()

É o ponto de partida do jogo e o ponto de finalização, pois ao iniciar o programa, o menu é o primeiro a ser chamado, e a partir dele que se chama as outras rotinas. Se o usuário escolher iniciar o jogo, a partida finalizará apenas se o usuário perder ou vencer jogo, e caso uma dessas situações ocorra, o programa retornará ao menu inicial.

- void Nivel()

Chamada logo após iniciar o jogo. Nesta o usuário deve informar qual o nível da dificuldade do jogo, e a partir daí, o programa carregará um jogo de sudoku no nível solicitado.

- void Instrucoes()

Trata-se de uma simples função que imprime na tela as instruções de como jogar o sudoku. Ela é chamada dentro da função “Menu”.

2.5.2 Funções resolutivas

- void IniciarJogo()

Esta função é a principal de uma partida, pois nela está a lógica principal de funcionamento do jogo. Nesta função também se encontra a lógica de manter o jogo em andamento enquanto o usuário não perder ou vencer o jogo. Isto é possível graças ao laço infinito utilizado. Pela sua importância esta função foi convertida em fluxograma, conforme podemos ver na figura x, localizada no anexo 1 deste trabalho.

- matriz* Inicializa()

Esta função é importantíssima, pois ela carrega um dos vários jogos de sudoku disponíveis na biblioteca “Partidas.h” para a lista principal do jogo, a “MatSudoku”. Nesse processo de carregamento do jogo na lista, cada célula (endereço) da recebe seu respectivo valor, linha, coluna e sinal, conforme a estrutura de dados foi definida. Esta função é chamada logo após se escolher o nível do jogo pelo usuário.

- void IniciaBackUp(matriz* mat)

Chamada dentro da função acima “IniciarJogo”, seu objetivo principal é iniciar o processo de backup da solução do jogo. Após a lista ”MatSudoku” receber os respectivos valores do jogo da partida atual, antes que o usuário insira qualquer valor na lista, é necessário resolver o sudoku e salvar em uma outra lista a solução do sudoku da partida, e é justamente este o objetivo desta função, iniciar o processo de backup da solução em uma outra lista chamada “MatSudokuResolvidoBackUp”.

- void ResolveSudokuEfazBackUp(matriz* mat, int lin, int col)

Esta função é a função mais importante do todo o jogo, pois nela está o algoritmo que resolve o sudoku. Este algoritmo recursivo percorre cada célula da lista e atribui um valor de 1 a 9 até conseguir chegar ao caso base, que é o momento que o algoritmo chegou a uma solução. Ou seja, a função faz tentativas de 1 a 9 em cada célula da lista até conseguir achar uma solução para o respectivo jogo, e dependendo do jogo, são milhares de tentativas, logo esta função pode levar bastante tempo chegar ao fim.

O resultado deste percurso pode ser ilustrado em uma árvore, em que o algoritmo percorre cada nó tentando inserir valores e quando em um nó ele não consegue inserir nenhum dos nove valores o algoritmo insere os valores que estavam antes na lista de volta para poder tentar outra possibilidade de combinações, sendo assim ele vai retornando ao nó pai reinserindo os valores que estavam antes.

Devido a este processo de inserir de e desinserir valores na lista, ao finalizar o algoritmo, a lista volta ao seu estado original, ou seja, sem a solução. Devido a este motivo, ao chegar no caso base do algoritmo, deve-se fazer um backup da solução, e é exatamente este objetivo que a função abaixo realiza.

- void FazBackUp(matriz* mat)

Neste momento estamos no caso base da recursão da função “ResolveSudokuEfazBackUp” acima, logo, a lista passada por parâmetro contêm a solução em sua células, nos restando apenas passar estes valores para uma outra lista que se chama, como já mencionado, “MatSudokuResolvidoBackUp”.

- void InsereValorUsuario(matriz* mat)

Esta função solicita do usuário valor, linha e coluna para ser inserido na lista principal. O valor só será inselido se não houver erro de sintaxe e nem quebra das regras do sudoku.

- void InsereValor(matriz* mat, int num, int lin, int col);

Chamada dentro da função “InsereValorUsuario”, esta função insere um valor de 1 a 9 na lista “MatSudoku”.

- int verificaSintaxe(int num);

Chamada dentro da função “InsereValorUsuario”, esta função verifica se existe erro de sintaxe nos valores que o usuário digitar.

- int VerificaJogada(matriz* mat, int num, int lin, int col);

Chamada dentro da função “InsereValorUsuario” e também pela função “ResolveSudokuEfazBackUp”, esta função verifica quebras nas regras do sudoku, tais como valores repitidos na mesma linha, coluna ou região.

- int VerificaSePerdeu()

Esta função faz uma comparação entre a lista “MatSudoku”, que é a lista do jogo, com a lista “MatSudokuResolvida”, que é a lista que armazenou a solução do jogo. Se houver diferença nos valores, um sequer que seja, significa que é impossível vencer, pois o jogo possui apenas uma única solução.

- int VerificaSeVenceu();

Se o usuário conseguir preencher todos os valores livres da lista, significa que ele venceu, pois ele conseguiu inserir todos os valores sem que o programa detectasse qualquer tipo de erro, logo, significa que os valores inseridos são os corretos e que o jogo está vencido. Esta faz justamente esta verificação percorrendo toda a lista e verificando se ainda existem valores livres, e se não houver, houve uma vitória.

- matriz* SalvaJogada(matriz* mat, int num, int lin, int col);

Esta função é chamada dentro da função “InsereValorUsuario”, e seu objetivo é armazenar as jogadas que são válidas do usuário, criando assim, um histórico de jogadas em uma lista chamada “HistoricoJogadas”.

2.5.3 Funções auxiliares

- int getNumById(matriz* mat, int lin, int col)

Esta função é importante, pois permite acessar um valor da lista pela linha e pela coluna, da mesma maneira como é realizado com matrizes, ou seja, o objetivo desta função é indexar o acesso na lista.

- int getSinalById (matriz* mat, int lin, int col);

Possui o mesmo propósito da função acima, mas ao invés de acessar o valor “num”, ela acessa o valor “sinal”, de acordo com a estrutura de dados construída.

- int ValorAleatorio()

Possui o objetivo de gerar e retornar um valor aleatório. Esta função é utilizada na lógica de seleção de jogos ao iniciar o jogo, pois para que seja possível ter jogos diferentes em cada partida, deve-se fazer uma escolha aleatória.

3. METODOLOGIA

A metodologia utilizada no desenvolvimento do programa pode ser ilustrada pelo fluxograma da figura 3.

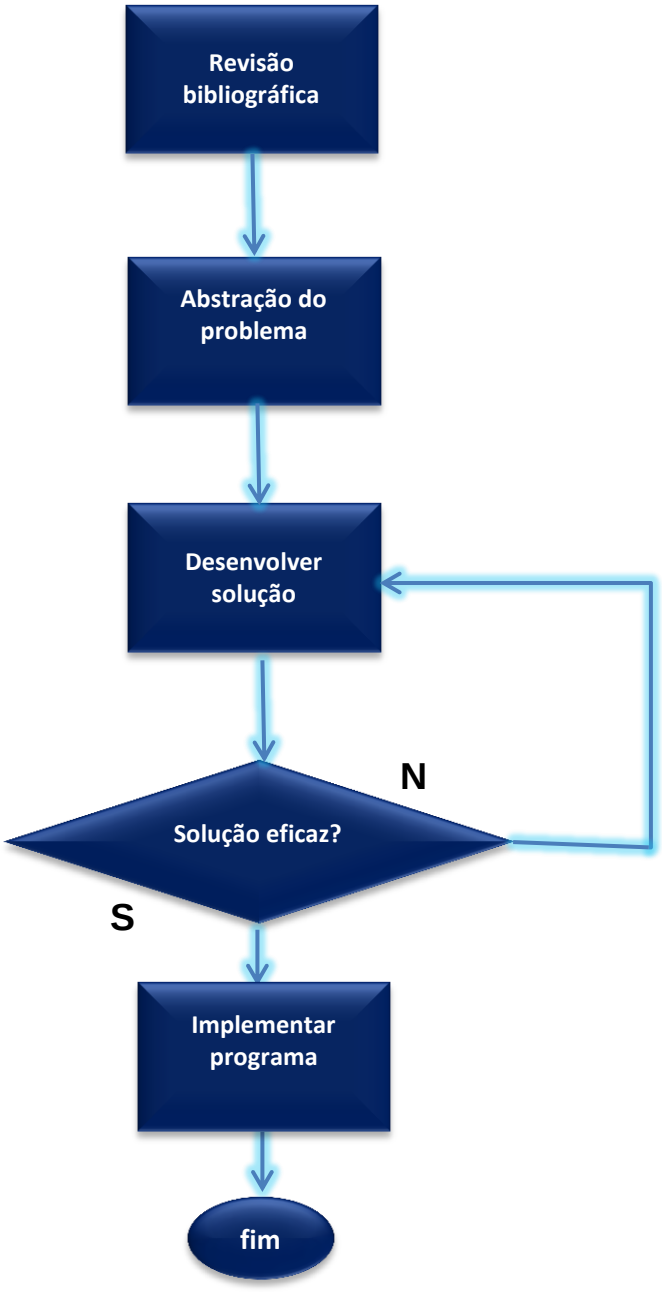

Figura 3 – Fluxograma da metodologia

4. CONCLUSÃO

O objetivo deste trabalho foi desenvolver um jogo eficiente de sudoku que seguisse todas as suas respectivas regras. O programa desenvolvido atendeu a todos os requisitos solicitados e mostrou ser eficiente tanto na teoria, quanto também na prática através dos testes.

Apesar de o trabalho ser na área de computação, este envolveu estudos em outras áreas, como matemática de matrizes e probabilidade. Esse fato proporcionou um aprendizado muito mais amplo e um aproveitamento maior do aprendizado.

É certo que houve alguns problemas como a demora da função “ResolveSudokuEfazBackUp”, pelo fato de ela ser uma complexa função recursiva, mas tais problemas puderam ser contornados pelos resultados obtidos, ou seja, houveram mais vantagens do que desvantagens no algoritmo desenvolvido.

O que se espera em trabalhos futuros é que se possa melhorar a quantidade e a qualidade dos recursos que o programa oferece. Também se espera aumentar a quantidade de partidas disponíveis e a jogabilidade do usuário, permitindo meios muito mais rápidos de inserção de valores através das coordenadas escolhidas.

7. REFERÊNCIAS BIBLIOGRÁFICAS

Schildt, H. **C Completo e Total**. Tradução de: Roberto C. Mayer. 3. Ed. São Paulo: Makron Books, 1997.

Jamsa, K; Klander, L. **Programando em C/C++ A Bíblia**. Tradução de: Jeremias René D. P. dos Santos. São Paulo: Makron Books, 1999.

MONOGRAFIA_NET, Regras da ABNT, Disponível em: <http://www,monografia,net/abnt/index,htm> Acesso em: 19/03/2013,

8. ANEXO 1

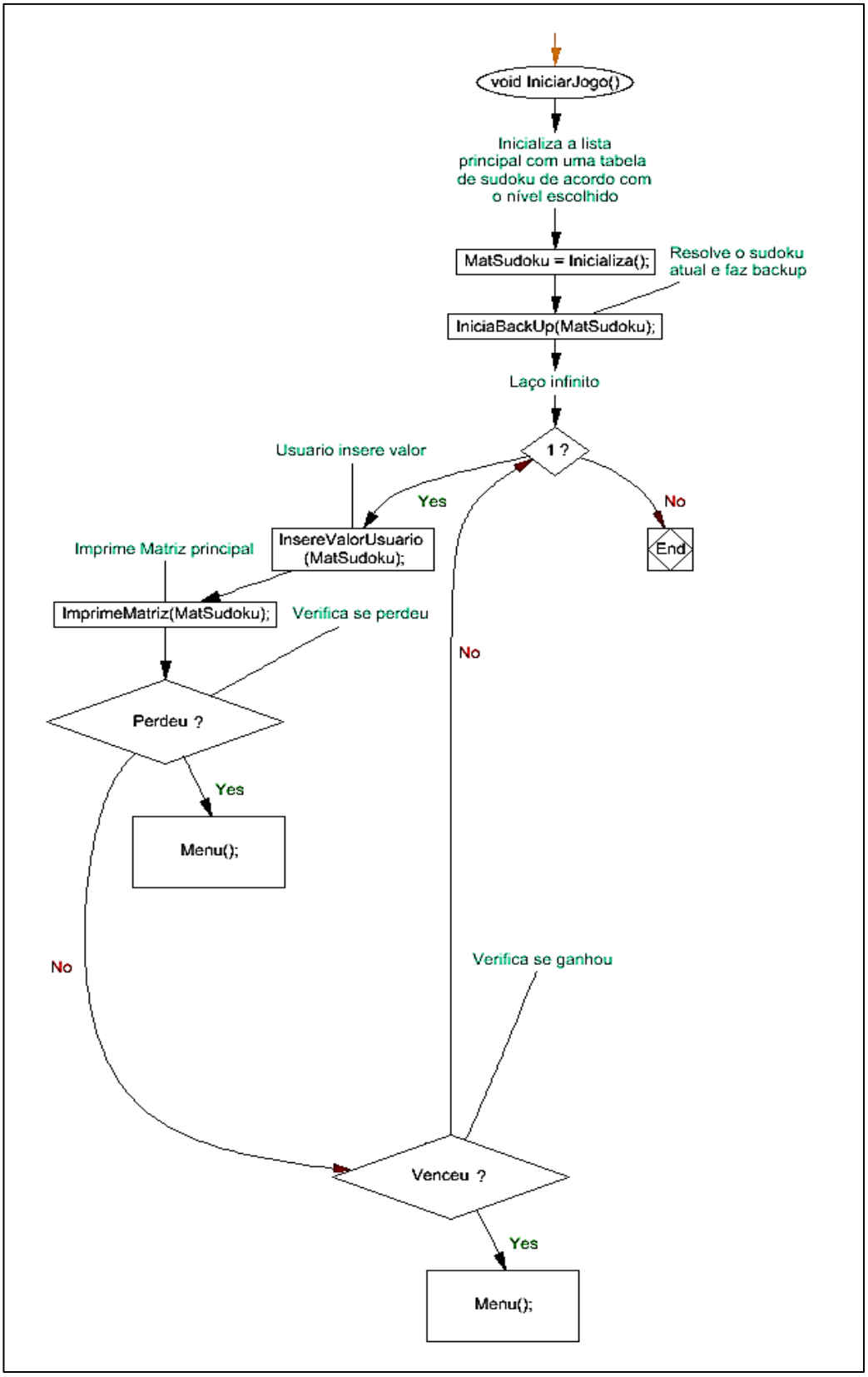

Figura 4 – Fluxograma da função IniciarJogo()