



UNIVERZITET U NOVOM SADU FAKULTET TEHNIČKIH NAUKA



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Departman za računarstvo i automatiku
Odsek za računarsku tehniku i računarske komunikacije

ISPITNI RAD

Kandidat: Vladimir Getmantsev

Broj indeksa: SV84-2023

Predmet: Objektno orijentisano programiranje 2

Tema rada: Sudoku Game

Mentor rada: _____

Novi Sad, December, 2023.

0. Table of Contents

1. [Sudoku Game Structure](#)
2. [Automatic Puzzle Solving Solution](#)
3. [Puzzle Generation Solution](#)
4. [Project Structure](#)
 - 4.1 [App.h/App.cpp](#)
 - 4.2 [FileManager.h/FileManager.cpp](#)
 - 4.3 [Game.h/Game.cpp](#)
 - 4.4 [Cell.h/Cell.cpp](#)
 - 4.5 [Matrix.h/Matrix.cpp](#)
 - 4.6 [Sudoku9x9.h/Sudoku9x9.cpp](#)
 - 4.7 [SudokuSolver.h/SudokuSolver.cpp](#)
 - 4.8 [Main.cpp](#)
5. [User Interaction](#)
 - 5.1. [Application Launch](#)
 - 5.2. [Main Menu](#)
 - 5.2.1 [Initial Startup](#)
 - 5.2.2 [Startup With Command Line Argument](#)
 - 5.2.3 [Start New Game on Initial Start](#)
 - 5.2.4 [Start New Game With Command Line Args or With Active Session](#)
 - 5.2.5 [Load Game](#)
 - 5.2.6 [Continue Game](#)
 - 5.2.7 [Save Current Session](#)
 - 5.2.8 [Exit](#)
 - 5.2.9 [Invalid Input](#)
 - 5.3. [Game Menu](#)
 - 5.3.1 [Print Game Menu](#)
 - 5.3.2 [Set Cell](#)
 - 5.3.3 [Check If Board Solved](#)
 - 5.3.4 [Solve Board](#)
 - 5.3.5 [Pause](#)
 - 5.3.6 [Invalid Input](#)
6. [Testing](#)
 - 6.1 [Testing Sudoku Engine](#)
 - 6.1.1 [Matrix Tests](#)

- 6.1.1.1 [Constructor Test](#)
- 6.1.1.2 [Print Test](#)
- 6.1.1.3 [Row Contains Test](#)
- 6.1.1.4 [Column Contains Test](#)
- 6.1.2 [Sudoku9x9 Tests](#)
 - 6.1.2.1 [Sudoku9x9_load_dump_Test](#)
 - 6.1.2.2 [Sudoku9x9_setCell](#)
 - 6.1.2.3 [Sudoku9x9_checkRow](#)
 - 6.1.2.4 [Sudoku9x9_checkColumn](#)
 - 6.1.2.5 [Sudoku9x9_checkSubgrid](#)
 - 6.1.2.6 [Sudoku9x9_checkIfSolved](#)
- 6.1.3 [SudokuSolver Test Cases](#)
 - 6.1.3.1 [testSudokuSolver_solveBoard](#)
 - 6.1.3.2 [testSudokuSolver_unitTest](#)
- 6.2 [Testing sudoku app](#)

7. [Source code](#)

8. [Conclusion](#)

1. Sudoku game structure

The Sudoku game typically follows a structured format, involving a grid of cells, rules for filling in the cells, and a solving objective. Here's a description of the Sudoku game structure:

1. Game Grid:

- The game is played on a 9x9 grid, divided into nine 3x3 subgrids or regions. Each subgrid is referred to as a "box."
- The entire grid is initially filled with a partially completed puzzle. Some cells have given numbers, forming the puzzle's initial state.

2. Cell Values:

- Cells can be empty or filled with a digit from 1 to 9.
- The objective is to fill in the entire grid so that each row, each column, and each of the nine 3x3 subgrids (boxes) contains all of the digits from 1 to 9 without repetition.

3. Rules:

- Each row must contain all the digits from 1 to 9 without repetition.
- Each column must contain all the digits from 1 to 9 without repetition.
- Each 3x3 subgrid (box) must contain all the digits from 1 to 9 without repetition.

4. Initial Puzzle:

- The puzzle is presented with some cells pre-filled with numbers.
- The placement of these initial numbers adheres to the rules, ensuring that a unique solution exists.

5. **User Input or Program Generation:**

- Players can either solve a pre-defined puzzle (from file) or let the program generate a new puzzle for them.
- For generated puzzles, a valid and solvable arrangement of numbers is created.

6. **User Interaction:**

- Players interact with the puzzle by filling in empty cells with numbers based on the rules of the game.
- The inputted numbers must conform to the rules to create a valid solution.
- Player can check if the current puzzle solved.
- Player can use option to automatically solve current puzzle

7. **Validation:**

- The game should validate whether the player's input conforms to the rules.
- If a player completes the puzzle, the solution is checked for correctness.

8. **Replay Option:**

- After completing a game, players have the an option to create new puzzle of load another one from file (throw main menu), initiating a new iteration of the game.

2. Automatic Puzzle Solving Solution

The backtracking algorithm is a common and efficient approach to solving Sudoku puzzles.

The step-by-step description of the algorithm:

1. **Choose an Empty Cell:**

- Start with an empty cell in the Sudoku grid.

2. **Try a Digit:**

- Try placing a digit (1 to 9) in the chosen empty cell.

3. **Check Validity:**

- Check if the digit placement is valid according to Sudoku rules (no repetition in the row, column, or 3x3 subgrid).

4. **Recursive Approach:**

- If the digit placement is valid, move on to the next empty cell and repeat steps 2-3.
- If the digit placement is not valid, backtrack to the previous cell and try a different digit.

5. **Repeat Until Solved:**

- Repeat the process until the entire grid is filled and Sudoku rules are satisfied.
- If at any point the algorithm reaches an invalid state (no valid digit for an empty cell), backtrack to the previous cell and try a different digit.

Implementation in pseudocode

input:

board - initial puzzle that should be solved

output:

board(modified) - solved puzzle according to the rules

function solveSudoku(board):

for each empty cell in board:

for digit in 1 to 9:

if digit is valid in the current cell:

place digit in the current cell

recursively try to solve the remaining puzzle

if the puzzle is solved:

return true (success)

if the puzzle cannot be solved with the current placement:

remove digit from the current cell (backtrack)

return false (no valid digit for the current cell)

This algorithm explores different possibilities using recursion and backtracks when it encounters an invalid state. The key to efficiency is the early rejection of invalid placements, reducing unnecessary exploration of the solution space. The algorithm continues until a valid solution is found or all possibilities are exhausted.

3. Puzzle Generation Solution

The step-by-step description of the algorithm:

1. Create Randomly Three 3x3 Subgrids:

- Begin by creating three distinct 3x3 subgrids, each filled with random numbers following the Sudoku rules (no repetition in rows, columns, or subgrids).
- The goal is to generate diverse and non-overlapping subgrids.

2. Arrange Subgrids to Avoid Intersection:

- Place the three generated subgrids in a way that they do not intersect. This means they should be in different rows and columns.

- Ensure that no numbers are repeated in the rows and columns where the subgrids are placed.

3. **Solve Current Puzzle:**

- Solve the partially filled Sudoku puzzle created by combining the three subgrids.
- Utilize a solving algorithm, like the backtracking algorithm, to fill in the remaining cells in a way that adheres to the Sudoku rules.
- This step ensures that the generated puzzle has a unique solution.

4. **Drop N Numbers from the Solved Puzzle:**

- Randomly select N cells from the solved puzzle.
- Remove the numbers from these cells to create empty spaces.
- Ensure that the resulting puzzle still maintains its unique solution.
- The number N determines the difficulty level of the Sudoku puzzle; a higher N results in fewer given numbers, making the puzzle more challenging.

By following these steps, we can generate a Sudoku puzzle with a specified difficulty level, maintaining the uniqueness of the solution.

input:

initialPuzzle - Empty puzzle filled with zeroes

output:

finalPuzzle - Puzzle ready to solve to

function generateSudokuBoard(N):

subgrids[] <- generateRandomSubgrid(i) : i = 0...3

initialPuzzle = createEmptyPuzzle()

for each subgrid in subgrids:

initialPuzzle = putSubgrid(subgrid, initialPuzzle)

initialPuzzle = solveSudoku(initialPuzzle)

finalPuzzle = dropNumbers(solvedPuzzle, N)

return finalPuzzle

4. Project Structure

4.1 App.h/App.cpp:

This code creates a structure for organizing a gaming application, facilitating user interaction, and managing gaming sessions.

```

/*!
 * \brief Structure representing an application for playing Sudoku.
 *
 * Provides the main user interface and manages game sessions.
 */
struct App {

    /*!
     * \brief Constructor for the application.
     *
     * Takes command line arguments and initializes the application based on the provided
     *
     * \param argc The number of command line arguments.
     * \param argv An array of strings representing command line arguments.
     */
    App(int argc, char **argv);

    /*!
     * \brief Main method to run the application.
     *
     * Guides the user through the main menu and handles the user's choice, performing c
     */
    void run();

    /*!
     * \brief Destructor for the application.
     */
    ~App() = default;

private:

    /*!
     * \brief Method to start a new game.
     *
     * Allows the user to choose the difficulty level and starts a new game.
     */
    void startNewGame();

    /*!
     * \brief Method to load a game from a file.
     *
     * Allows the user to choose a file to load a saved game.
     */

```



```

void loadGame();

/*!
 * \brief Method to save the current game to a file.
 *
 * Allows the user to choose whether to save the current game and, if yes, specify the filename.
 */
void saveGame();

/*!
 * \brief Method to start a game session.
 *
 * Initiates a game session if one is not already started.
 */
void startGame();

/*!
 * \brief Method to read a string from input.
 *
 * Reads a string from standard input and stores it in the filename variable.
 */
void getFile();

/*!
 * \brief Method to read a single value from input.
 *
 * \tparam T Type of the value to be read.
 * \param choose Reference to the variable where the read value will be stored.
 */
template<typename T>
static void getKey(T &choose) {
    std::cin >> choose;
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

Sudoku9x9 board; ///< Sudoku game board.
std::string filename; ///< Filename for loading or saving the game.
bool is_started{false}; ///< Flag indicating whether a game session is started.
};

```

4.2 FileManager.h/FileManager.cpp:

Provides functionality for reading Sudoku puzzles from files and saving them.

```

#pragma once

#include <string>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <iostream>
#include <filesystem>
#include "Sudoku9x9.h"

namespace fs = std::filesystem;

/*!
 * \brief The FileManager struct provides functionality for loading and saving Sudoku games
 */
struct FileManager {

    /*!
     * \brief Load a Sudoku game from a file.
     *
     * \param board Reference to a Sudoku9x9 object where the loaded game will be stored.
     * \param filename Reference to a string containing the name of the file to load from.
     * \return True if the load operation is successful, false otherwise.
     */
    static bool loadFromFile(Sudoku9x9 &board, std::string &filename) noexcept;

    /*!
     * \brief Save a Sudoku game to a file.
     *
     * \param board Reference to a Sudoku9x9 object containing the game to be saved.
     * \param filename Reference to a string containing the name of the file to save to.
     * \return True if the save operation is successful, false otherwise.
     */
    static bool saveToFile(Sudoku9x9 &board, std::string &filename) noexcept;

private:

    /*!
     * \brief Create a new file.
     *
     * \param savefile Reference to a string containing the name of the file to create.
     * \return True if the file creation is successful, false otherwise.
     */

```

```
static bool createFile(std::string &savefile) noexcept;  
};
```

4.3 Game.h/Game.cpp:

The main purpose of this module is to provide a simple way to start the game, ensuring user-friendly input. It displays an in-game menu and offers a mechanism for obtaining user input, supporting various data types through template functions.

```

#pragma once

#include <limits>
#include "Sudoku9x9.h"
#define CLEAR "\033[2J\033[1;1H"

/*!
 * \brief The Game struct provides functionality for running a Sudoku game.
 */
struct Game {
    /*!
     * \brief Run the Sudoku game.
     *
     * \param board Reference to a Sudoku9x9 object representing the game board.
     */
    static void run(Sudoku9x9 &board);

private:
    /*!
     * \brief Print the in-game menu.
     */
    static void printIngameMenu();

    /*!
     * \brief Implementation of getting a single value from input.
     *
     * \tparam T Type of the value to be read.
     * \param arg Reference to the variable where the read value will be stored.
     */
    template<typename T>
    static void getKeyImpl(T &arg) {
        std::cin >> arg;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    /*!
     * \brief Implementation of getting multiple values from input using recursion.
     *
     * \tparam T Type of the first value to be read.
     * \tparam Args Types of the remaining values to be read.
     * \param first Reference to the variable where the first read value will be stored.
     * \param rest References to the variables where the remaining read values will be st

```

```

*/
template<typename T, typename... Args>
static void getKeyImpl(T &first, Args &... rest) {
    std::cin >> first;
    getKeyImpl(rest...);
}

/*!
 * \brief Get multiple values from input using recursion.
 *
 * \tparam Args Types of the values to be read.
 * \param args References to the variables where the read values will be stored.
 */
template<typename... Args>
static void getKey(Args &... args) {
    getKeyImpl(args...);
}
};

```

4.4 Cell.h/Cell.cpp:

The `cell` structure encapsulates the properties and behaviors of a single cell within a Sudoku board. Each cell has an associated value and a modifiability flag, allowing it to represent both filled-in numbers and empty spaces. The structure is equipped with various member functions and overloaded operators to facilitate easy manipulation, comparison, and I/O operations.

```

#pragma once

#include <string>
#include <iostream>

/**
 * @struct Cell
 * @brief Represents a cell in a Sudoku board.
 */
struct Cell {
    /**
     * @brief Default constructor for Cell.
     */
    Cell() = default;

    /**
     * @brief Parameterized constructor for Cell.
     *
     * @param value The initial value of the cell.
     * @param modifiable Specifies if the cell is modifiable (default is true).
     */
    explicit Cell(int value, bool modifiable = true);

    /**
     * @brief Copy assignment operator for Cell.
     *
     * @param item The Cell object to copy from.
     * @return Reference to the assigned Cell.
     */
    Cell &operator=(Cell const &item) = default;

    /**
     * @brief Assignment operator to set the value of the cell.
     *
     * @param new_val The new value to assign to the cell.
     * @return Reference to the assigned Cell.
     */
    Cell &operator=(int new_val);

    /**
     * @brief Equality operator for Cell.
     *
     * @param right The Cell object to compare.

```

```

* @return true if the cells are equal, false otherwise.
*/
bool operator==(Cell &right) const;

/**
* @brief Equality operator for comparing with an integer.
*
* @param right The integer to compare.
* @return true if the cell's value is equal to the integer, false otherwise.
*/
bool operator==(int right) const;

/**
* @brief Conversion operator to int, allowing Cell to be used as an integer.
*
* @return The value of the cell.
*/
explicit operator int() const;

/**
* @brief Conversion operator to bool, indicating whether the cell is non-zero.
*
* @return true if the cell's value is non-zero, false otherwise.
*/
explicit operator bool() const;

/**
* @brief Overloaded stream insertion operator to print the cell.
*
* @param out The output stream.
* @param item The Cell object to print.
* @return Reference to the output stream.
*/
friend std::ostream &operator<<(std::ostream &out, Cell &item);

/**
* @brief Overloaded stream extraction operator to read the cell from input.
*
* @param in The input stream.
* @param item The Cell object to read into.
* @return Reference to the input stream.
*/
friend std::istream &operator>>(std::istream &in, Cell &item);

```



```

/**
 * @brief Switches the modifiability of the cell (modifiable <-> non-modifiable).
 */
void switchMod();

private:
/**
 * @brief Converts the cell's value to a string.
 *
 * @return The string representation of the cell's value.
 */
std::string toStr() const;

int value{0};    ///< The value of the cell.
bool modifiable{true}; ///< Specifies if the cell is modifiable.
};

```

4.5 Matrix.h/Matrix.cpp:

The Matrix structure represents a matrix of cells, commonly used to represent Sudoku boards. It includes functionalities for accessing and manipulating rows and columns, as well as providing information about the matrix's dimensions. The structure employs a 2D vector to store cells, and overloaded operators enhance the ease of working with matrices.

```

#pragma once

#include <vector>
#include "Cell.h"

/**
 * @struct Matrix
 * @brief Represents a matrix of cells, commonly used for Sudoku boards.
 */
struct Matrix {
    /**
     * @brief Parameterized constructor for Matrix.
     *
     * @param rows_num The number of rows in the matrix.
     * @param cols_num The number of columns in the matrix.
     */
    Matrix(int rows_num, int cols_num);

    /**
     * @brief Returns a vector of indices in the specified row that contain the given value.
     *
     * @param index The index of the row to search.
     * @param val The value to search for.
     * @return Vector of indices containing the given value in the specified row.
     */
    std::vector<int> rowContains(size_t index, int val);

    /**
     * @brief Returns a vector of indices in the specified column that contain the given value.
     *
     * @param index The index of the column to search.
     * @param val The value to search for.
     * @return Vector of indices containing the given value in the specified column.
     */
    std::vector<int> columnContains(size_t index, int val);

    /**
     * @brief Returns the number of rows in the matrix.
     *
     * @return The number of rows.
     */
    size_t getRows();

```

```

/**
 * @brief Returns the number of columns in the matrix.
 *
 * @return The number of columns.
 */
size_t getColumns();

/**
 * @brief Overloaded subscript operator to access a specific row in the matrix.
 *
 * @param index The index of the row to access.
 * @return Reference to the specified row.
 */
std::vector<Cell> &operator[](int index);

/**
 * @brief Prints the matrix to the console.
 */
void print();

private:
    std::vector<
        std::vector<
            Cell
        >> values; ///< The 2D vector storing the matrix of cells.
};

```

4.6 Sudoku9x9.h/Sudoku9x9.cpp:

The `Sudoku9x9` structure, which inherits from the previously defined `Matrix` structure, represents a 9x9 Sudoku puzzle board. This structure extends the functionality of the `Matrix` to include Sudoku-specific operations and checks, providing methods for setting cells, checking for collisions, loading and dumping the puzzle board, and printing the board to the console.

```

#pragma once

#include <sstream>
#include <unordered_map>
#include "Matrix.h"

#define SUBGRID_SIZE 3
#define PUZZLE_SIZE (SUBGRID_SIZE*SUBGRID_SIZE)

struct Sudoku9x9 : public Matrix {
    Sudoku9x9() : Matrix(PUZZLE_SIZE, PUZZLE_SIZE) {}

    /**
     * @brief Checks if the Sudoku puzzle is solved.
     *
     * @return true if the puzzle is solved, false otherwise.
     */
    bool checkIfSolved();

    /**
     * @brief Sets the value of a cell in the Sudoku grid.
     *
     * @param row The row index of the cell (1 to 9).
     * @param column The column index of the cell (1 to 9).
     * @param val The value to set in the cell.
     * @return true if the cell was set successfully, false if there is a collision.
     */
    bool setCell(int row, int column, int val);

    /**
     * @brief Loads a Sudoku puzzle board from a string of numbers.
     *
     * The string should contain 81 characters representing the puzzle grid, with '0' for
     *
     * @param string_of_nums A string representing the puzzle grid.
     */
    void loadBoard(const std::string &string_of_nums);

    /**
     * @brief Dumps the current state of the Sudoku puzzle board as a string.
     *
     * @return A string representation of the puzzle grid.
     */
};

```

```
std::string dumpBoard();
```

```
/**
```

```
* @brief Checks if a given number is present in the specified row of the Sudoku board.  
*
```

```
* @param row The row index to check (0 to 8).
```

```
* @param num The number to check for presence (1 to 9).
```

```
* @return true if the number is not present in the row, false if it is present.
```

```
*/
```

```
bool isInRow(int row, int num);
```

```
/**
```

```
* @brief Checks if a given number is present in the specified column of the Sudoku board.  
*
```

```
* @param col The column index to check (0 to 8).
```

```
* @param num The number to check for presence (1 to 9).
```

```
* @return true if the number is not present in the column, false if it is present.
```

```
*/
```

```
bool isInColumn(int col, int num);
```

```
/**
```

```
* @brief Checks if a given number is present in the specified subgrid of the Sudoku board.  
*
```

```
* @param sg_idx The index of the subgrid to check (0 to 8).
```

```
* @param num The number to check for presence (1 to 9).
```

```
* @return true if the number is not present in the subgrid, false if it is present.
```

```
*/
```

```
bool isInSubgrid(int sg_idx, int num);
```

```
/**
```

```
* @brief Prints the current state of the Sudoku board to the console.  
*
```

```
* This function prints the entire Sudoku board, displaying the numbers in each cell.  
*/
```

```
void printBoard();
```

protected:

```
/**
```

```
* @brief Checks for collisions in the specified row.  
*
```

```
* Indexes are formatted with an offset of 1 (from 1 to 9 instead of 0 to 8).
```

```
* Returns {-1, -1} if the row has no collisions.
```

```

* Returns {i, -1} if the row has '0' on index 'i'.
* Returns {i, j} if the row has a collision between indexes 'i' and 'j' (row[i-1] ==
*
* @param index The index of the row to check (1 to 9).
* @return std::pair<int, int> Result of the collision check.
*/
std::pair<int, int> checkRow(int index);

/**
* @brief Checks for collisions in the specified column.
*
* Indexes are formatted with an offset of 1 (from 1 to 9 instead of 0 to 8).
* Returns {-1, -1} if the column has no collisions.
* Returns {i, -1} if the column has '0' on index 'i'.
* Returns {i, j} if the column has a collision between indexes 'i' and 'j' (column[i-1] ==
*
* @param index The index of the column to check (1 to 9).
* @return std::pair<int, int> Result of the collision check.
*/
std::pair<int, int> checkColumn(int index);

/**
* @brief Checks for collisions in the specified subgrid.
*
* Indexes are formatted with an offset of 1 (from 1 to 9 instead of 0 to 8).
* Returns {-1, -1, -1, -1} if the subgrid has no collisions.
* Returns {i_x, i_y, -1, -1} if the subgrid element 'i' is zero.
* Returns {i_x, i_y, j_x, j_y} if the subgrid has a collision between elements 'i' &
*
* @param index The index of the subgrid to check (1 to 9).
* @return std::pair<int, int> Result of the collision check.
*/
std::tuple<int, int, int, int> checkSubgrid(int index);
};

```

4.7 SudokuSolver.h/SudokuSolver.cpp:

The `SudokuSolver` structure is designed to facilitate the generation and solving of Sudoku boards. It incorporates static methods that employ backtracking algorithms to efficiently solve Sudoku puzzles. The overall structure leverages a combination of random number generation, placement strategies, and recursive algorithms to achieve its functionalities.

```

#pragma once

#include "Sudoku9x9.h"

#include <iostream>
#include <vector>
#include <algorithm>
#include <random>

/**
 * @class SudokuSolver
 * @brief A class for generating and solving Sudoku boards.
 */
struct SudokuSolver {
    /**
     * @brief Generates a Sudoku board with a specified number of numbers dropped.
     *
     * @param board The Sudoku9x9 object to generate the board for.
     * @param nums_to_drop The number of numbers to drop from the generated board.
     */
    static void generateBoard(Sudoku9x9 &board, int nums_to_drop);

    /**
     * @brief Solves the provided Sudoku board.
     *
     * @param board The Sudoku9x9 object to solve.
     * @return true if the board is successfully solved, false otherwise.
     */
    static bool solveBoard(Sudoku9x9 &board);

protected:
    /**
     * @brief Fills a 3x3 grid with random numbers.
     *
     * @param grid The grid to fill.
     * @param g The random number generator.
     */
    static void fillGrid(std::vector<std::vector<int>> &grid, std::mt19937 &g);

    /**
     * @brief Places a filled grid onto the Sudoku board at a specified position.
     *
     * @param grid The filled grid to place.
     */

```

```

* @param board The Sudoku9x9 object to place the grid on.
* @param grid_number The number identifying the position to place the grid.
*/
static void putGrid(std::vector<std::vector<int>> &grid, Sudoku9x9 &board, int grid_

/**
* @brief Drops a specified number of random numbers from the Sudoku board.
*
* @param board The Sudoku9x9 object to drop numbers from.
* @param k The number of random numbers to drop.
* @param g The random number generator.
*/
static void dropNums(Sudoku9x9 &board, int k, std::mt19937 &g);

/**
* @brief Checks if it is safe to place a number in a specific cell of the Sudoku board.
*
* @param board The Sudoku9x9 object.
* @param row_idx The row index of the cell.
* @param col_idx The column index of the cell.
* @param num The number to check for safety.
* @return true if it is safe to place the number, false otherwise.
*/
static bool isSafe(Sudoku9x9 &board, int row_idx, int col_idx, int num);

/**
* @brief Solves the Sudoku board using a recursive backtracking algorithm.
*
* @param board The Sudoku9x9 object to solve.
* @param row The current row during the solving process.
* @param col The current column during the solving process.
* @return true if the board is successfully solved, false otherwise.
*/
static bool solveBoardHelper(Sudoku9x9 &board, int row, int col);
};

```

4.8 Main.cpp

Entry point of the program, that passes command line arguments to the main sudoku loop.

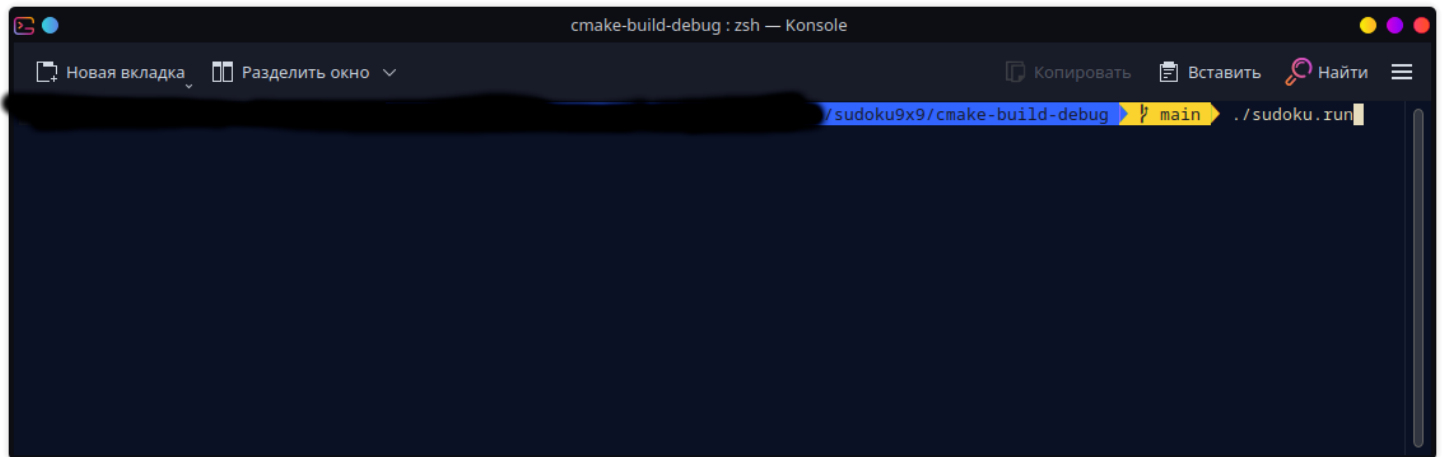

```
#include "App.h"

int main(int argc, char **argv) {
    auto app = App(argc, argv);
    app.run();
    return 0;
}
```

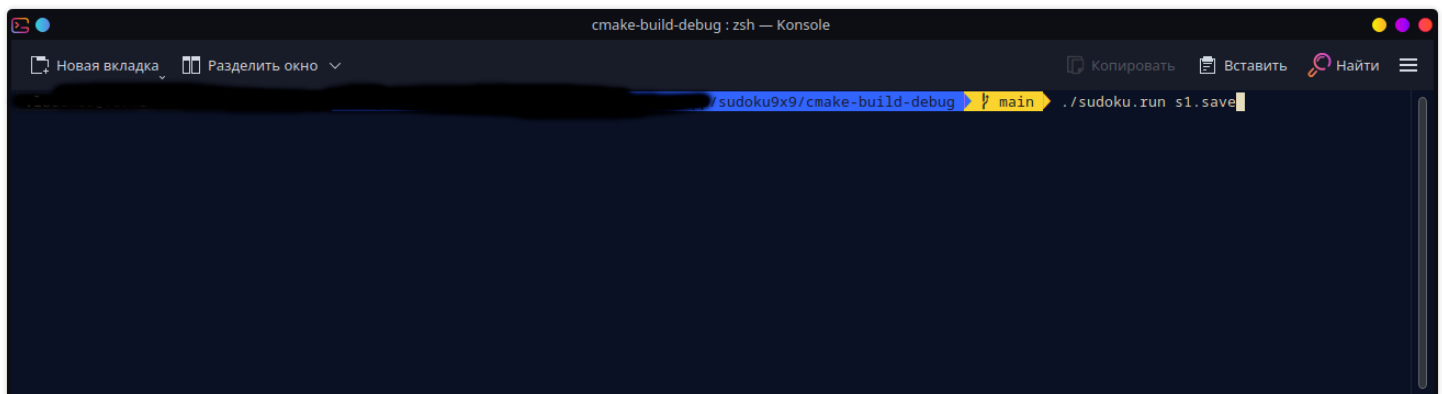
5. User Interaction

5.1 Application Launch

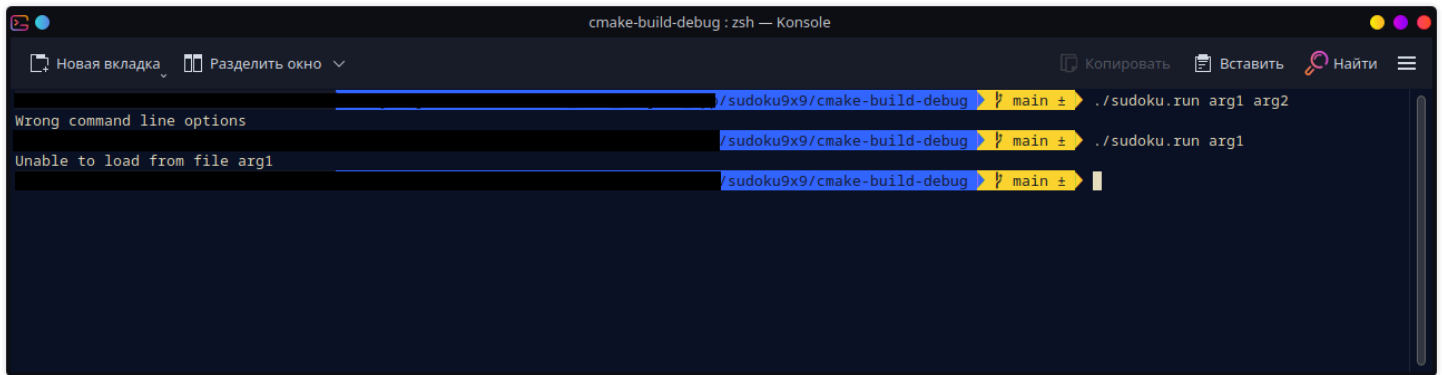
Normal launch:



The application supports loading a save file at launch by passing the filename as a command line argument:



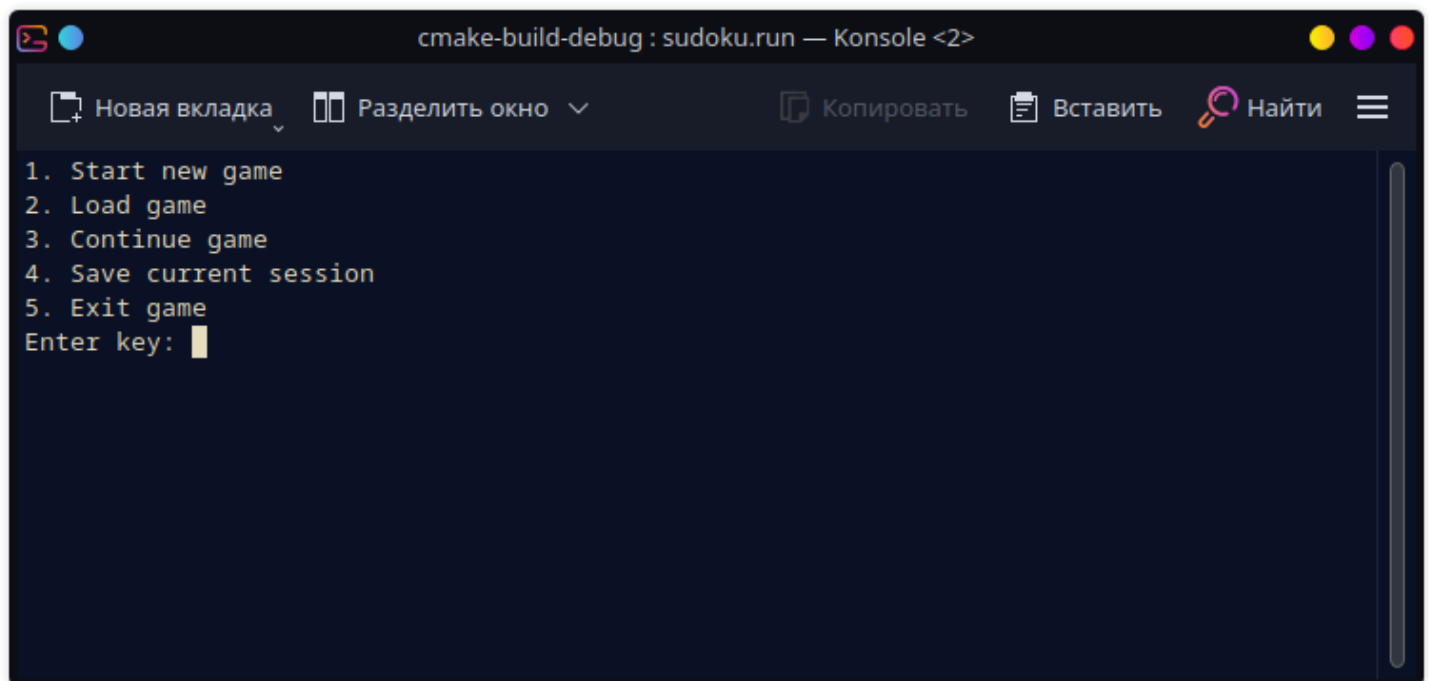
When entering invalid command line arguments, the program will not start and will issue a warning:



```
cmake-build-debug : zsh — Konsole
Новая вкладка Разделить окно
Копировать Вставить Найти
/sudoku9x9/cmake-build-debug main ± ./sudoku.run arg1 arg2
Wrong command line options
/sudoku9x9/cmake-build-debug main ± ./sudoku.run arg1
Unable to load from file arg1
/sudoku9x9/cmake-build-debug main ±
```

5.2 Main Menu

5.2.1 Initial Startup



```
cmake-build-debug : sudoku.run — Konsole <2>
Новая вкладка Разделить окно
Копировать Вставить Найти
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: 
```

5.2.2 Startup With Command Line Arg

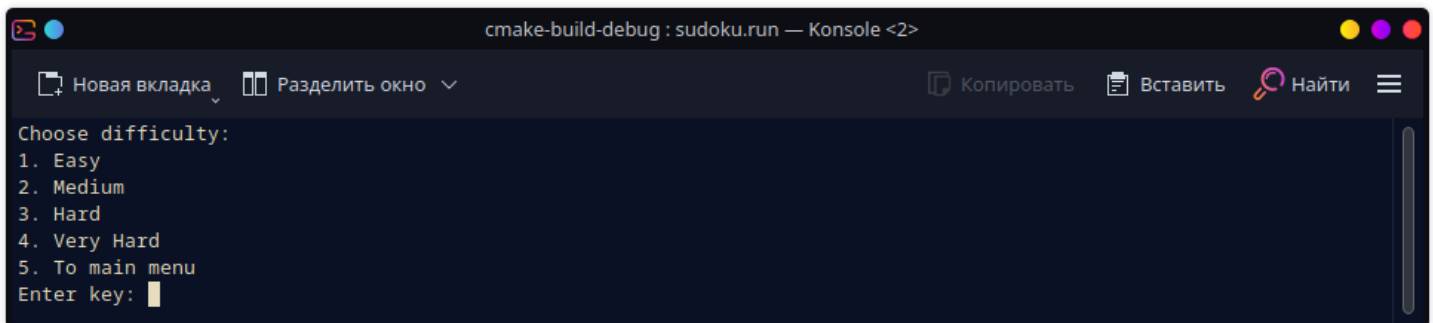


```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Game is loaded from file s1.save
To start loaded game choose '3. Continue game'
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: █
```

5.2.3 Start New Game on Initial Start

Option № 1

When the user chooses to start a new game, the program prompts them to choose the difficulty or go back:

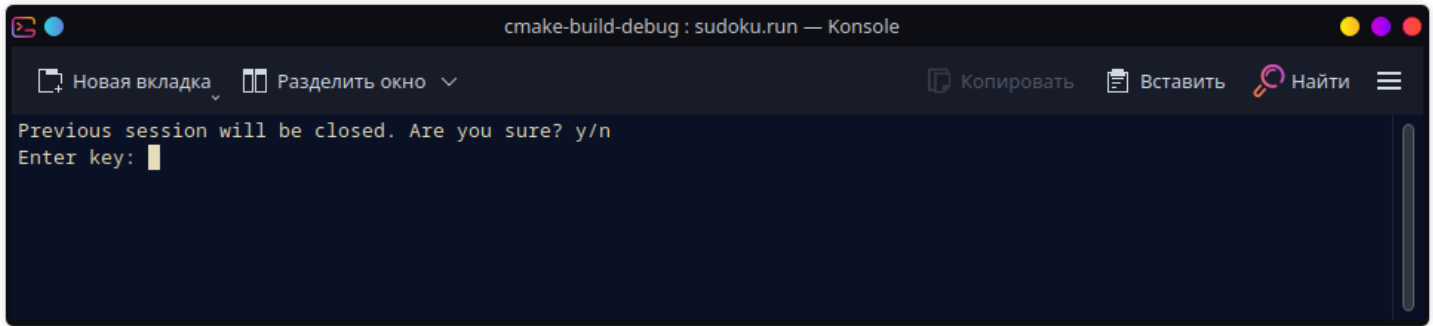


```
cmake-build-debug : sudoku.run — Konsole <2>
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Choose difficulty:
1. Easy
2. Medium
3. Hard
4. Very Hard
5. To main menu
Enter key: █
```

5.2.4 Start New Game With Command Line Args or With Active Session

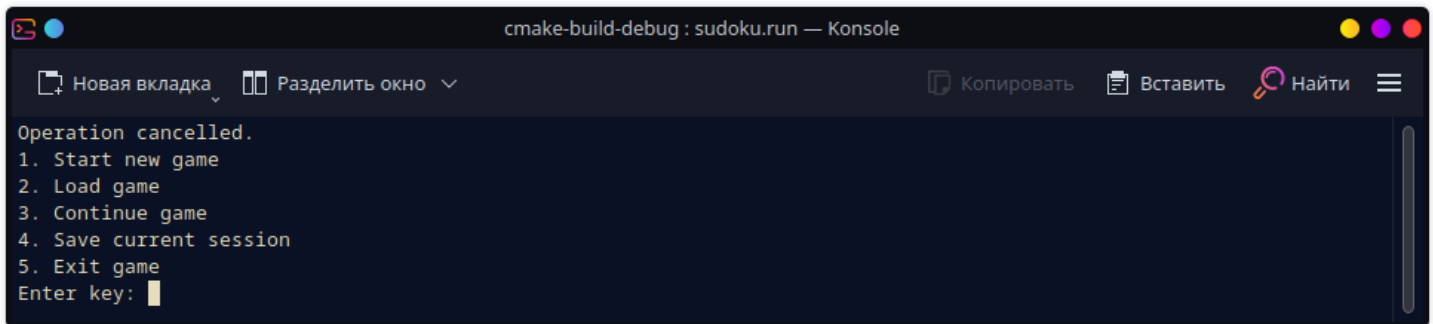
If the user wants to reset the current session, after confirmation, the user goes to the difficulty selection screen.

After selection, the game begins.

A terminal window titled 'cmake-build-debug : sudoku.run — Konsole'. The window has a dark background and a light-colored text. At the top, there is a menu bar with options: 'Новая вкладка', 'Разделить окно', 'Копировать', 'Вставить', 'Найти', and a hamburger menu icon. The main text area shows the prompt 'Previous session will be closed. Are you sure? y/n' followed by 'Enter key: ' and a cursor. The window has standard macOS window controls (red, yellow, green buttons) in the top right corner.

```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
Previous session will be closed. Are you sure? y/n
Enter key: 
```

If the user declines to start a new game, the program returns to the main menu.

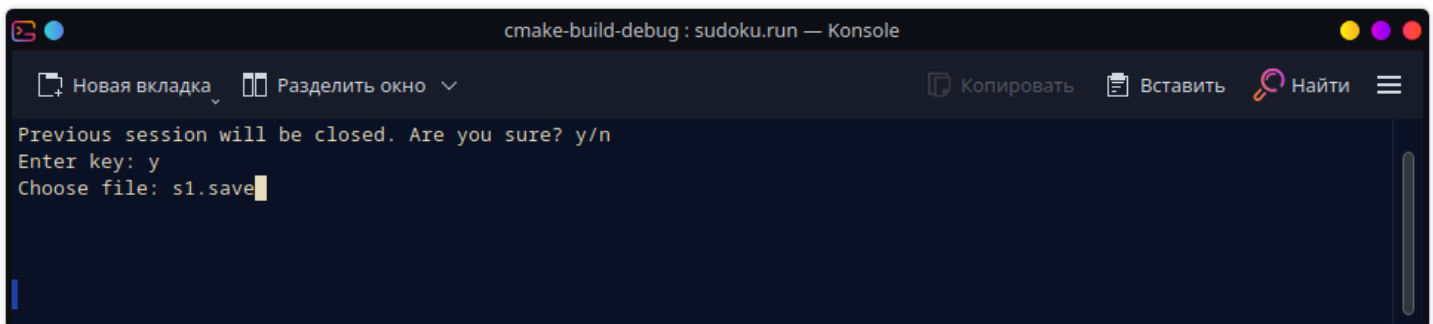
A terminal window titled 'cmake-build-debug : sudoku.run — Konsole'. The window has a dark background and a light-colored text. At the top, there is a menu bar with options: 'Новая вкладка', 'Разделить окно', 'Копировать', 'Вставить', 'Найти', and a hamburger menu icon. The main text area shows the prompt 'Operation cancelled.' followed by a list of options: '1. Start new game', '2. Load game', '3. Continue game', '4. Save current session', '5. Exit game', and 'Enter key: ' with a cursor. The window has standard macOS window controls (red, yellow, green buttons) in the top right corner.

```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
Operation cancelled.
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: 
```

5.2.5 Load Game

Option № 2

When loading the game, the session is also reset, so the user needs to confirm the operation. Then the user needs to enter the save file name:

A terminal window titled 'cmake-build-debug : sudoku.run — Konsole'. The window has a dark background and a light-colored text. At the top, there is a menu bar with options: 'Новая вкладка', 'Разделить окно', 'Копировать', 'Вставить', 'Найти', and a hamburger menu icon. The main text area shows the prompt 'Previous session will be closed. Are you sure? y/n' followed by 'Enter key: y' and 'Choose file: s1.save' with a cursor. The window has standard macOS window controls (red, yellow, green buttons) in the top right corner.

```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
Previous session will be closed. Are you sure? y/n
Enter key: y
Choose file: s1.save
```

After confirmation, the game begins.

5.2.6 Continue Game

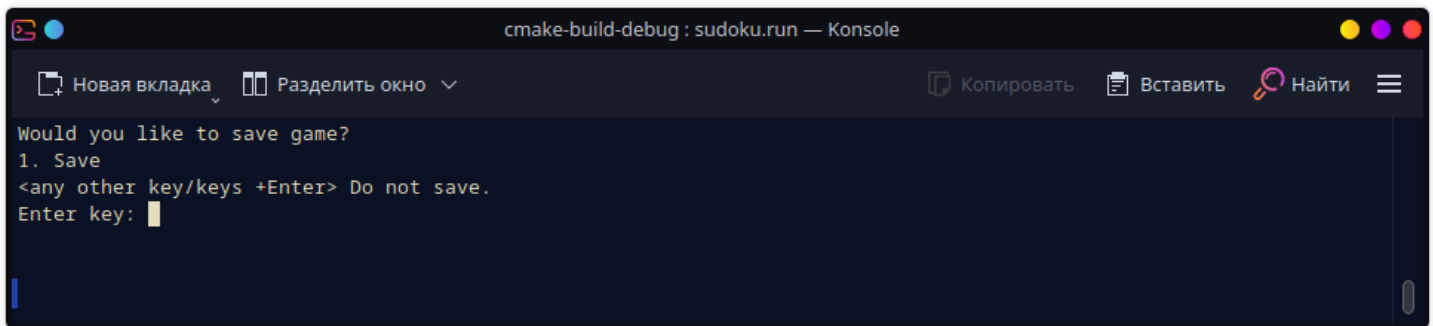
Option № 3

During initial startup, the user has no active session, so the program issues a notification to the user. Otherwise, if there is already a started or loaded session (for example, after starting a new game or loading through the program menu or as a command line argument), the user will be shown the game menu.

5.2.7 Save Current Session

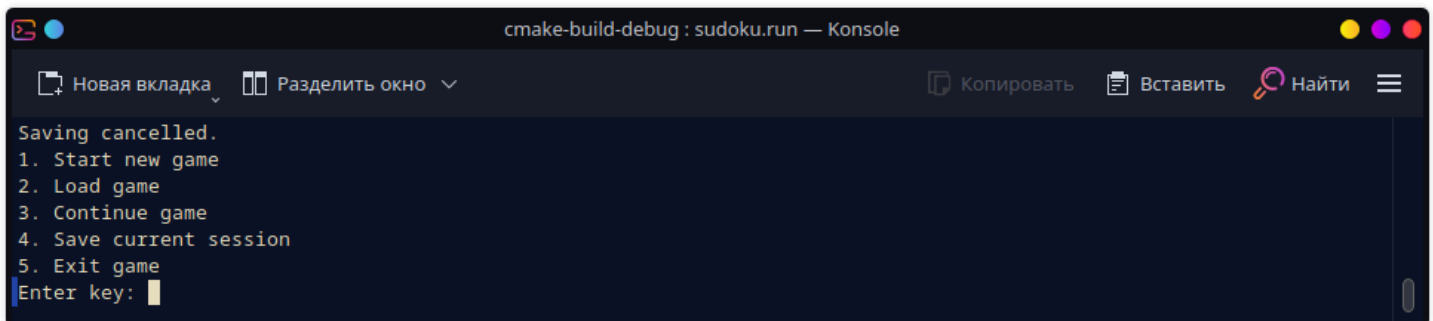
Option № 4

When selecting option 4, the program clarifies with the user whether they want to save the game:



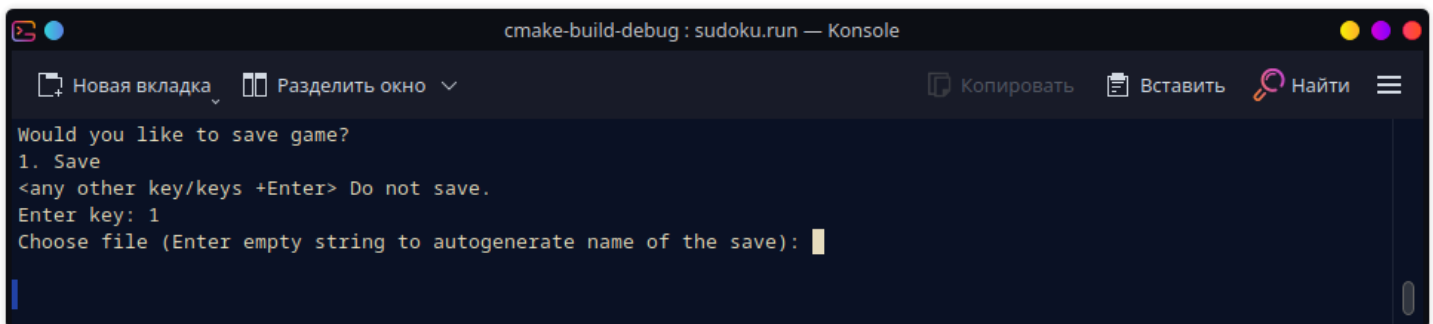
```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Would you like to save game?
1. Save
<any other key/keys +Enter> Do not save.
Enter key: 
```

If declined, the program returns the user to the main menu:



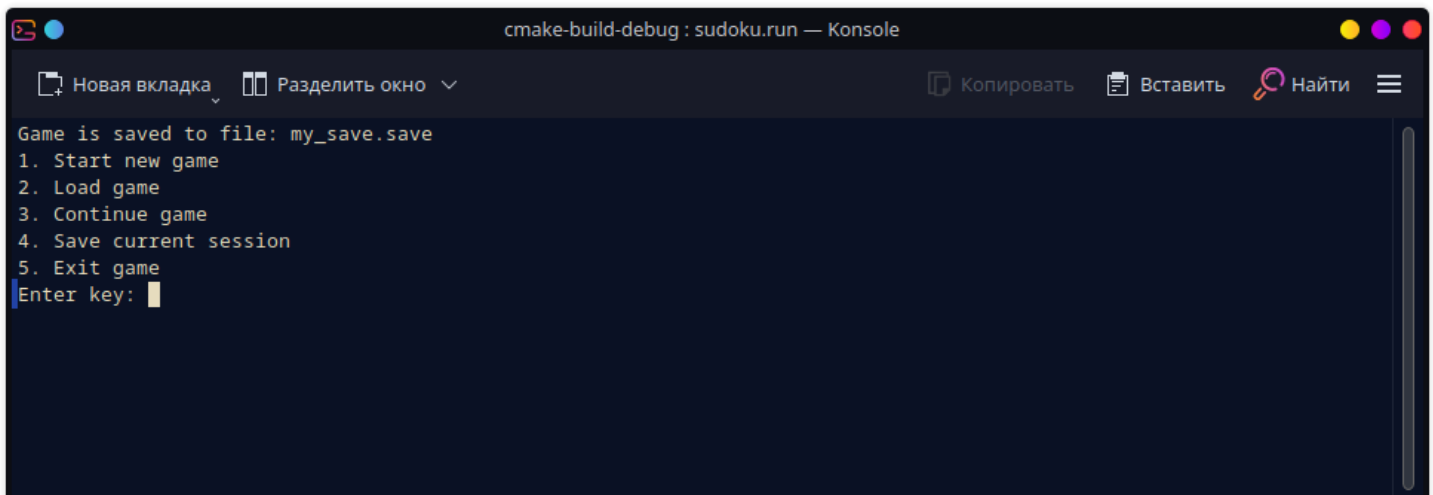
```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Saving cancelled.
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: 
```

Otherwise, it asks the user to enter a filename:



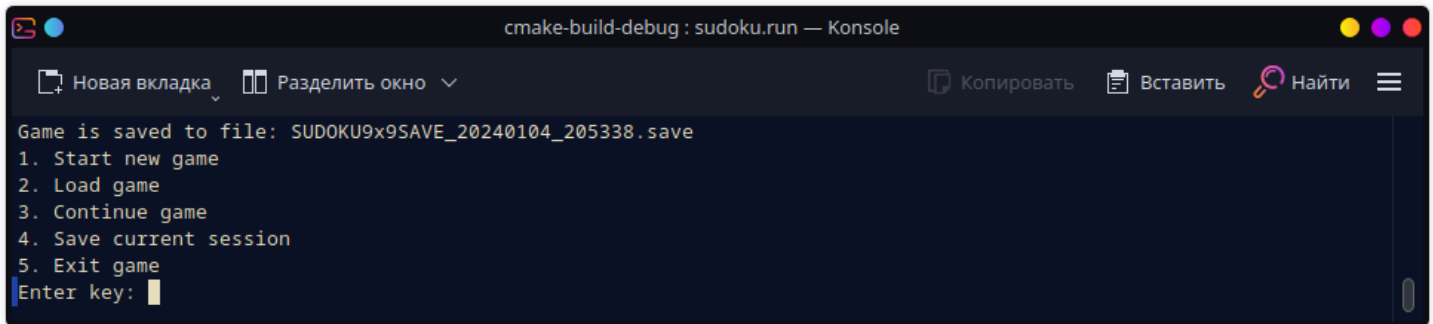
```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Would you like to save game?
1. Save
<any other key/keys +Enter> Do not save.
Enter key: 1
Choose file (Enter empty string to autogenerate name of the save): 
```

And after saving the current session:



```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Game is saved to file: my_save.save
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: 
```

If the user does not enter any filename, the program generates one automatically based on the date and time of saving:

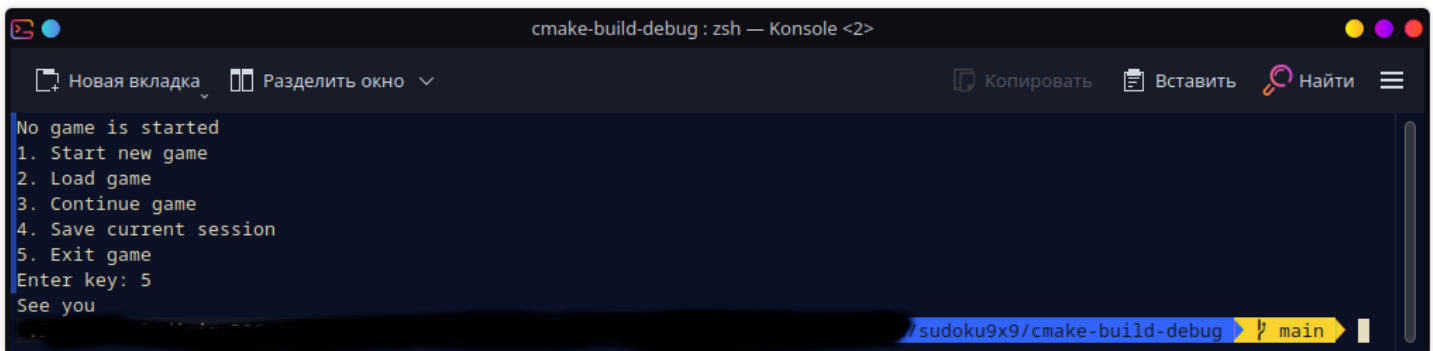


```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
Game is saved to file: SUDOKU9x9SAVE_20240104_205338.save
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: █
```

5.2.8 Exit

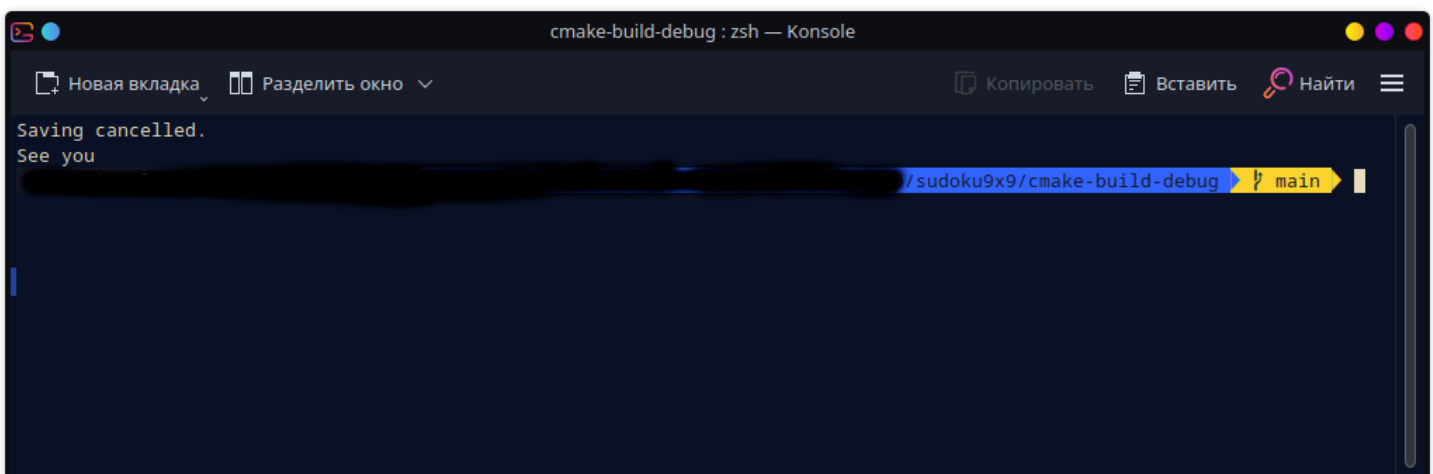
Option № 5

When exiting the program without an active session, the program terminates:

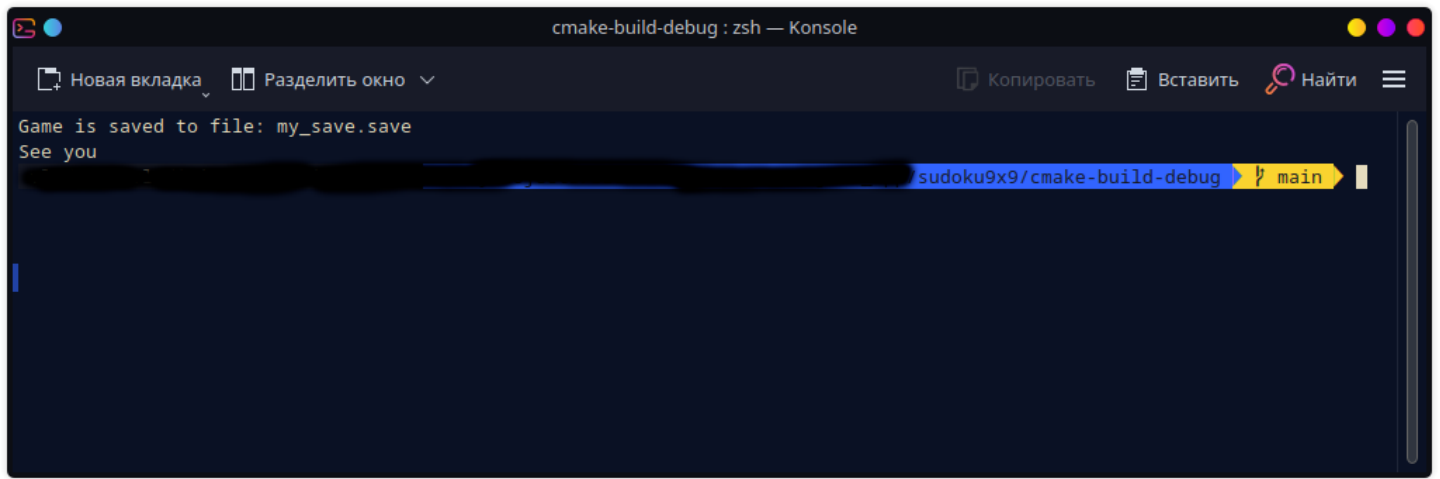


```
cmake-build-debug : zsh — Konsole <2>
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
No game is started
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: 5
See you
/sudoku9x9/cmake-build-debug > main █
```

Otherwise, it asks the user whether to save the session and does so if the user indicated it. It also informs in which file the session was saved:



```
cmake-build-debug : zsh — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
Saving cancelled.
See you
/sudoku9x9/cmake-build-debug > main █
```

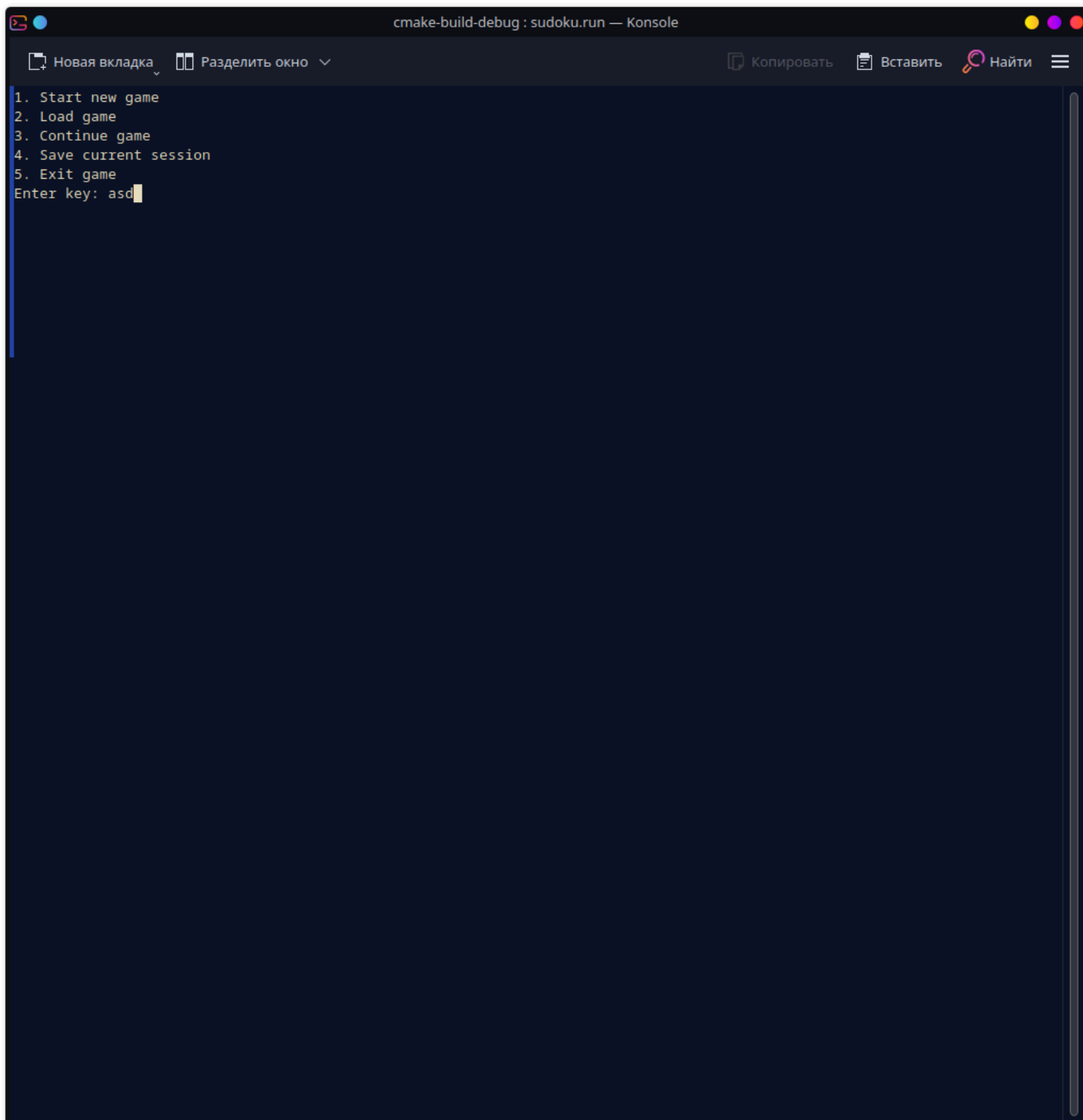


```
cmake-build-debug : zsh — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Game is saved to file: my_save.save
See you
[Redacted] sudoku9x9/cmake-build-debug main
```

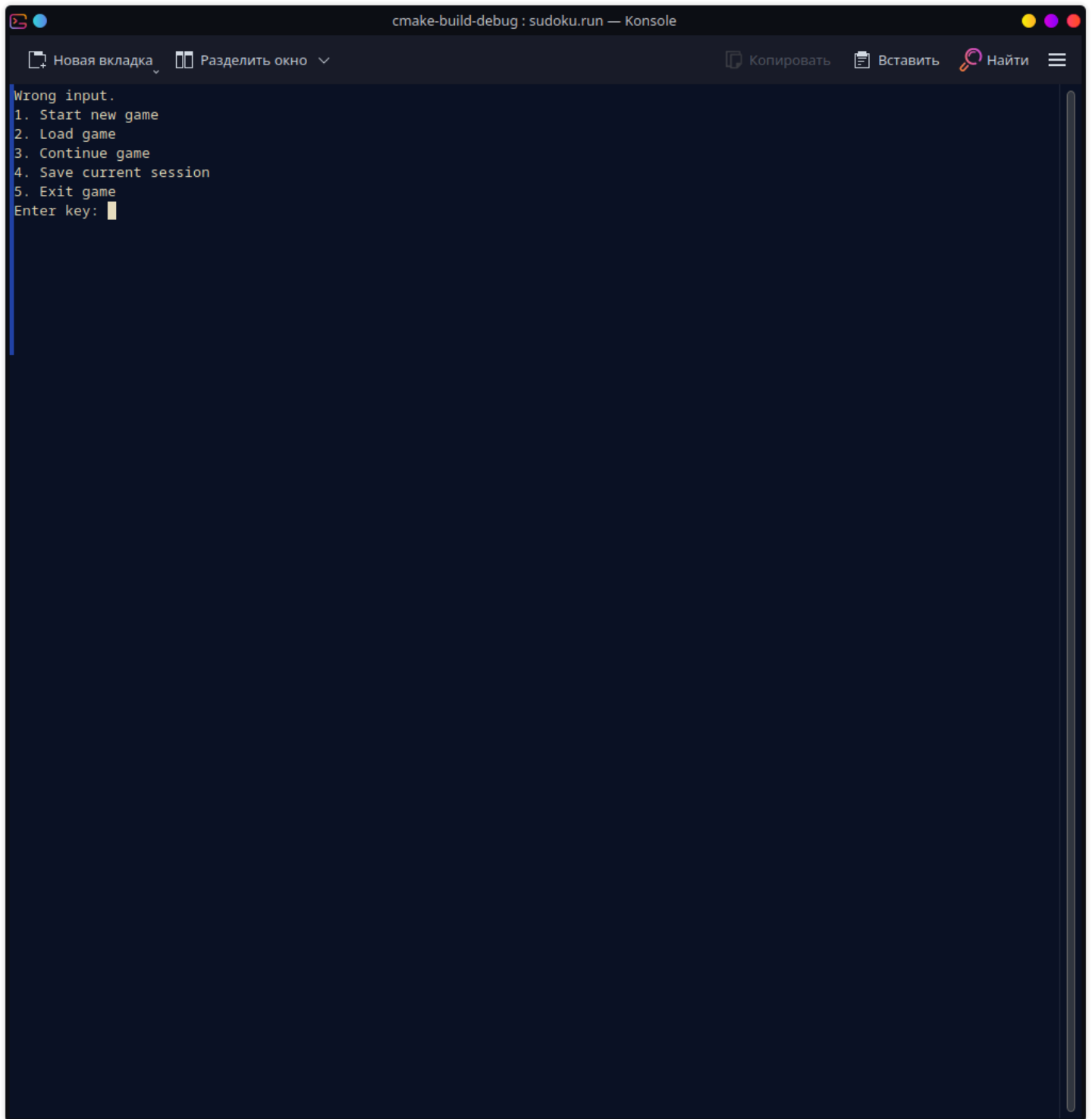
The session-saving process looks similar to choosing option № 4 .

5.2.9 Invalid Input

On any invalid input, the program will notify the user:



```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: asd
```

```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  Найти  ≡
Wrong input.
1. Start new game
2. Load game
3. Continue game
4. Save current session
5. Exit game
Enter key: █
```

5.3 Game Menu

After starting the game, the user sees the following output:

```
cmake-build-debug : sudoku.run — Konsole
Новая вкладка  Разделить окно  Копировать  Вставить  >

  1    2    3    4    5    6    7    8    9
1  4    1    3    5    9    7    6    2    8
2  2    7    0    3    6    8    5    1    4
3  5    6    0    2    1    4    3    7    9
4  7    0    0    4    3    0    8    5    0
5  3    0    6    7    5    9    1    4    0
6  1    4    5    8    2    6    7    9    3
7  8    2    7    1    4    3    9    0    5
8  6    0    1    0    8    2    0    3    7
9  9    3    4    6    7    5    0    8    1

0. Print game menu
1. Set Cell (row, column, number)
2. Check if board solved
3. Solve board
4. Pause
Enter key: █
```

Each underlined number indicates that this cell can be changed, and the non-underlined one indicates that this cell cannot be changed.

5.3.1 Print Game Menu

Option № 0

Displays the current state of the game on the screen.

Used to clear the result of the last option execution.

5.3.2 Set Cell

Option № 1

When choosing option 1, the user will be prompted to enter the row, column, and value they want to place on the board:

cmake-build-debug : sudoku.run — Konsole

Новая вкладкаРазделить окноКопироватьВставить

	1	2	3	4	5	6	7	8	9
1	4	1	3	5	9	7 —	6	2	8
2	2	7	0 —	3	6	8 —	5	1	4
3	5	6	0 —	2	1	4	3	7	9
4	7	0 —	0 —	4	3	0 —	8	5	0 —
5	3	0 —	6	7	5	9	1	4	0 —
6	1	4	5	8	2	6	7	9	3
7	8	2	7	1	4	3	9	0 —	5
8	6	0 —	1	0 —	8	2	0 —	3	7
9	9	3	4	6	7	5	0 —	8	1

0. Print game menu

1. Set Cell (row, column, number)

2. Check if board solved

3. Solve board

4. Pause

Board is not solved

Enter key: 1

Enter row column number: 2 3 7

If the data is valid, the program will display that the operation is complete:

cmake-build-debug : sudoku.run — Konsole

Новая вкладка

Разделить окно

Копировать

Вставить

	1	2	3	4	5	6	7	8	9
1	4	1	3	5	9	7 —	6	2	8
2	2	7	7 —	3	6	8 —	5	1	4
3	5	6	0 —	2	1	4	3	7	9
4	7	0 —	0 —	4	3	0 —	8	5	0 —
5	3	0 —	6	7	5	9	1	4	0 —
6	1	4	5	8	2	6	7	9	3
7	8	2	7	1	4	3	9	0 —	5
8	6	0 —	1	0 —	8	2	0 —	3	7
9	9	3	4	6	7	5	0 —	8	1

0. Print game menu

1. Set Cell (row, column, number)

2. Check if board solved

3. Solve board

4. Pause

Number 7 is set on 2:3

Enter key:

Otherwise, it will display a notification of invalid input:

cmake-build-debug : sudoku.run — Konsole

Новая вкладка

Разделить окно

Копировать

Вставить

Найти

	1	2	3	4	5	6	7	8	9
1	4	1	3	5	9	7 —	6	2	8
2	2	7	0 —	3	6	8 —	5	1	4
3	5	6	0 —	2	1	4	3	7	9
4	7	0 —	0 —	4	3	0 —	8	5	0 —
5	3	0 —	6	7	5	9	1	4	0 —
6	1	4	5	8	2	6	7	9	3
7	8	2	7	1	4	3	9	0 —	5
8	6	0 —	1	0 —	8	2	0 —	3	7
9	9	3	4	6	7	5	0 —	8	1

0. Print game menu

1. Set Cell (row, column, number)

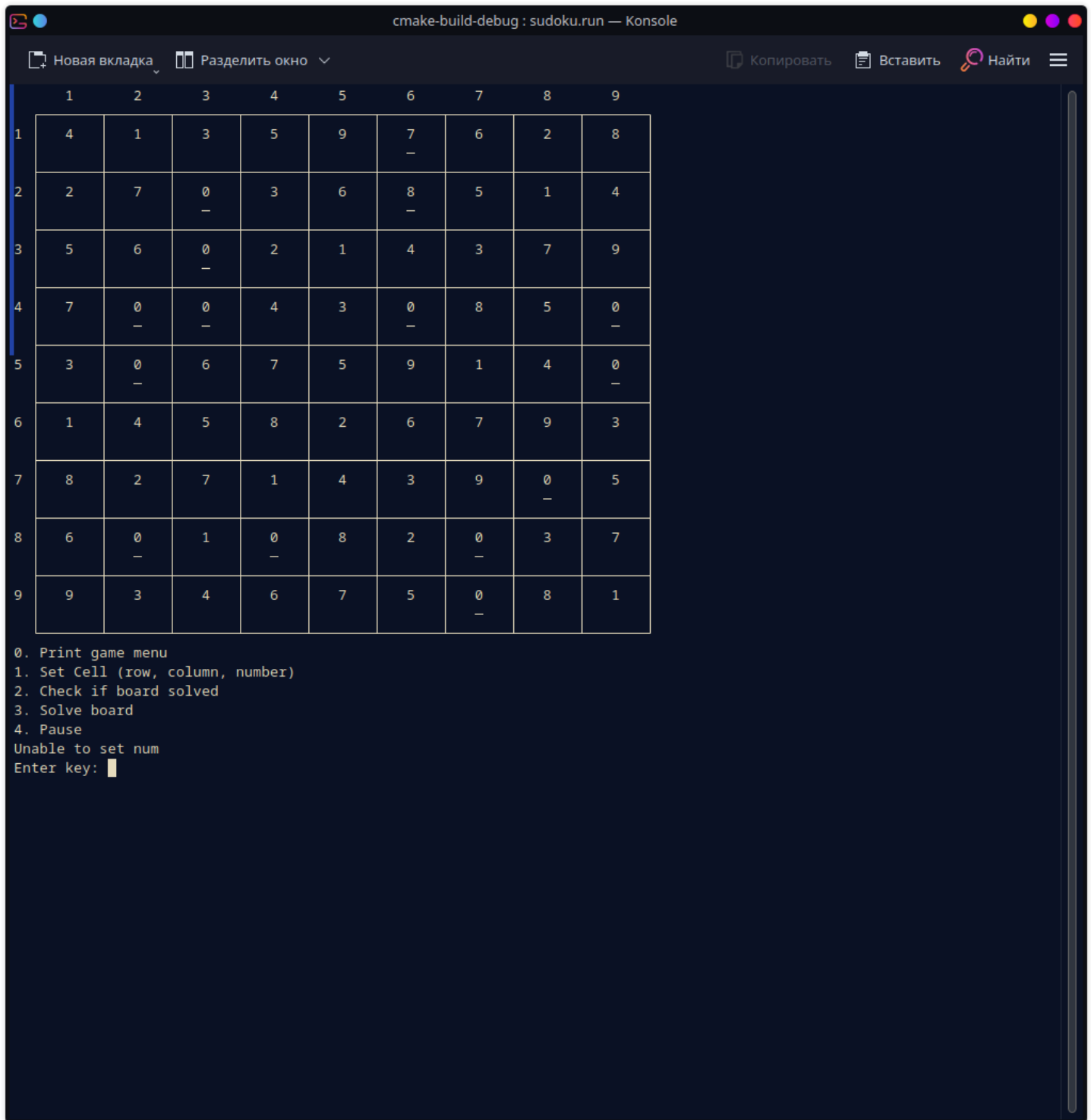
2. Check if board solved

3. Solve board

4. Pause

Enter key: 1

Enter row column number: 1111 1111 111 asd



5.3.3 Check If Board Solved

Option № 2

Tells the user if the current state of board solved or not.

cmake-build-debug : sudoku.run — Konsole

Новая вкладка

Разделить окно

Копировать

Вставить

>

	1	2	3	4	5	6	7	8	9
1	4	1	3	5	9	7 —	6	2	8
2	2	7	0 —	3	6	8 —	5	1	4
3	5	6	0 —	2	1	4	3	7	9
4	7	0 —	0 —	4	3	0 —	8	5	0 —
5	3	0 —	6	7	5	9	1	4	0 —
6	1	4	5	8	2	6	7	9	3
7	8	2	7	1	4	3	9	0 —	5
8	6	0 —	1	0 —	8	2	0 —	3	7
9	9	3	4	6	7	5	0 —	8	1

0. Print game menu

1. Set Cell (row, column, number)

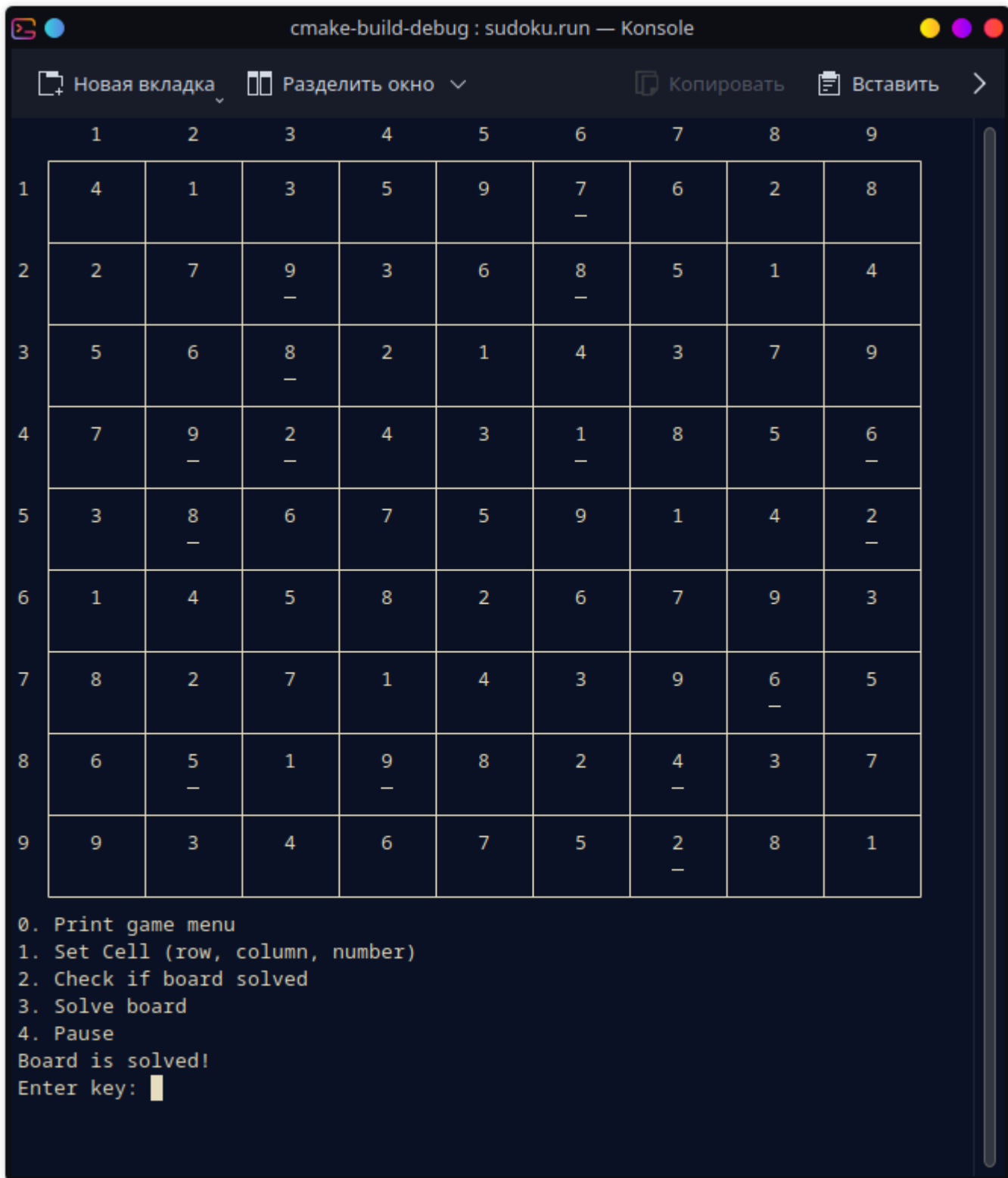
2. Check if board solved

3. Solve board

4. Pause

Board is not solved

Enter key:



5.3.4 Solve Board

Option № 3

Program solves the user board:

cmake-build-debug : sudoku.run — Konsole

Новая вкладкаРазделить окноКопироватьВставить

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

0. Print game menu

1. Set Cell (row, column, number)

2. Check if board solved

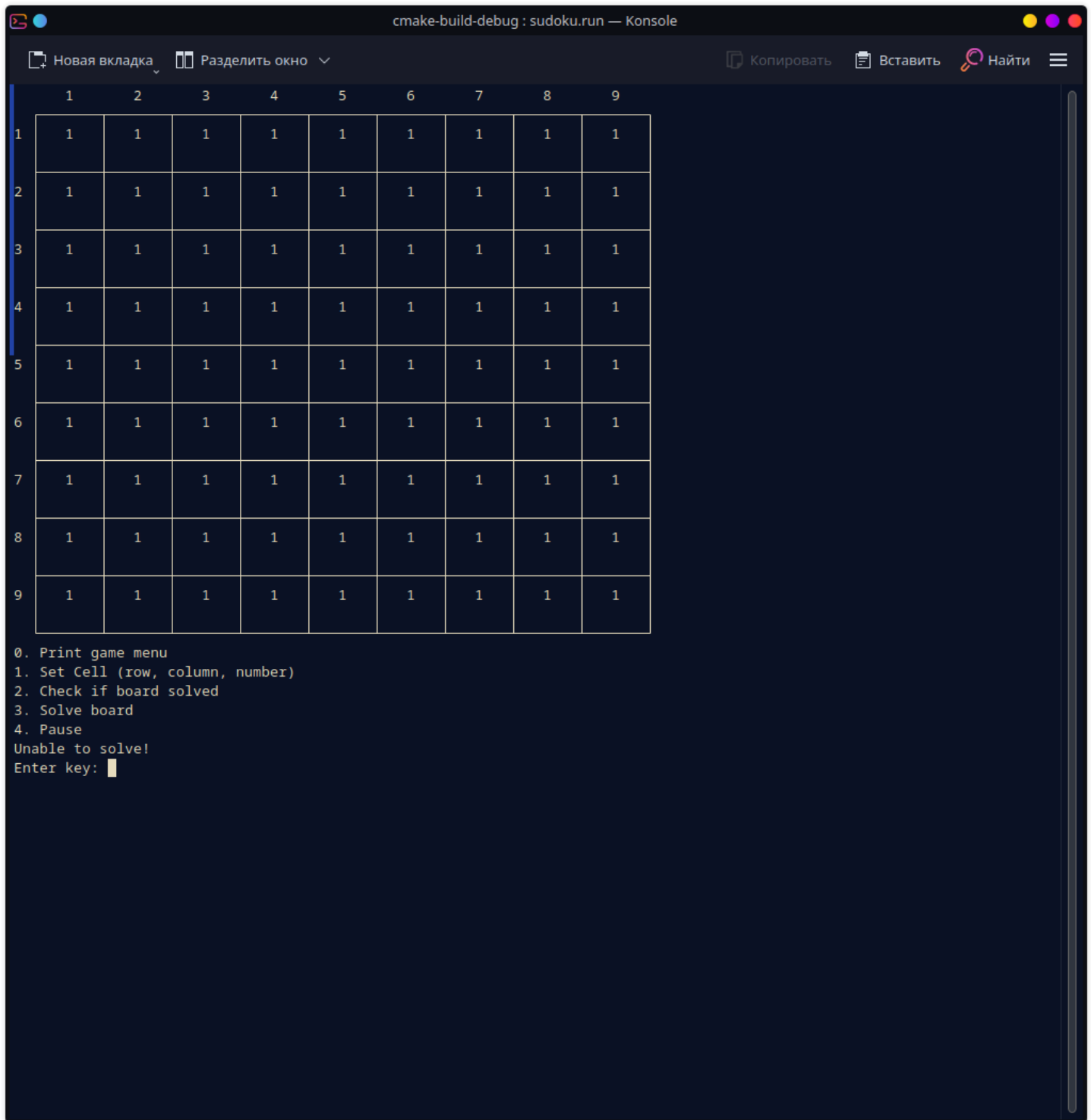
3. Solve board

4. Pause

Solved!

Enter key:

Or tells him, that it is impossible to do



5.3.5 Pause

Option № 4

Pauses the session and returns to the main menu.

5.3.6 Invalid Input

On any invalid input, the program will notify the user:

cmake-build-debug : sudoku.run — Konsole

Новая вкладкаРазделить окно

КопироватьВставитьНайти

	1	2	3	4	5	6	7	8	9
1	4	1	3	5	9	7	6	2	8
2	2	7	0	3	6	8	5	1	4
3	5	6	0	2	1	4	3	7	9
4	7	0	0	4	3	0	8	5	0
5	3	0	6	7	5	9	1	4	0
6	1	4	5	8	2	6	7	9	3
7	8	2	7	1	4	3	9	0	5
8	6	0	1	0	8	2	0	3	7
9	9	3	4	6	7	5	0	8	1

0. Print game menu

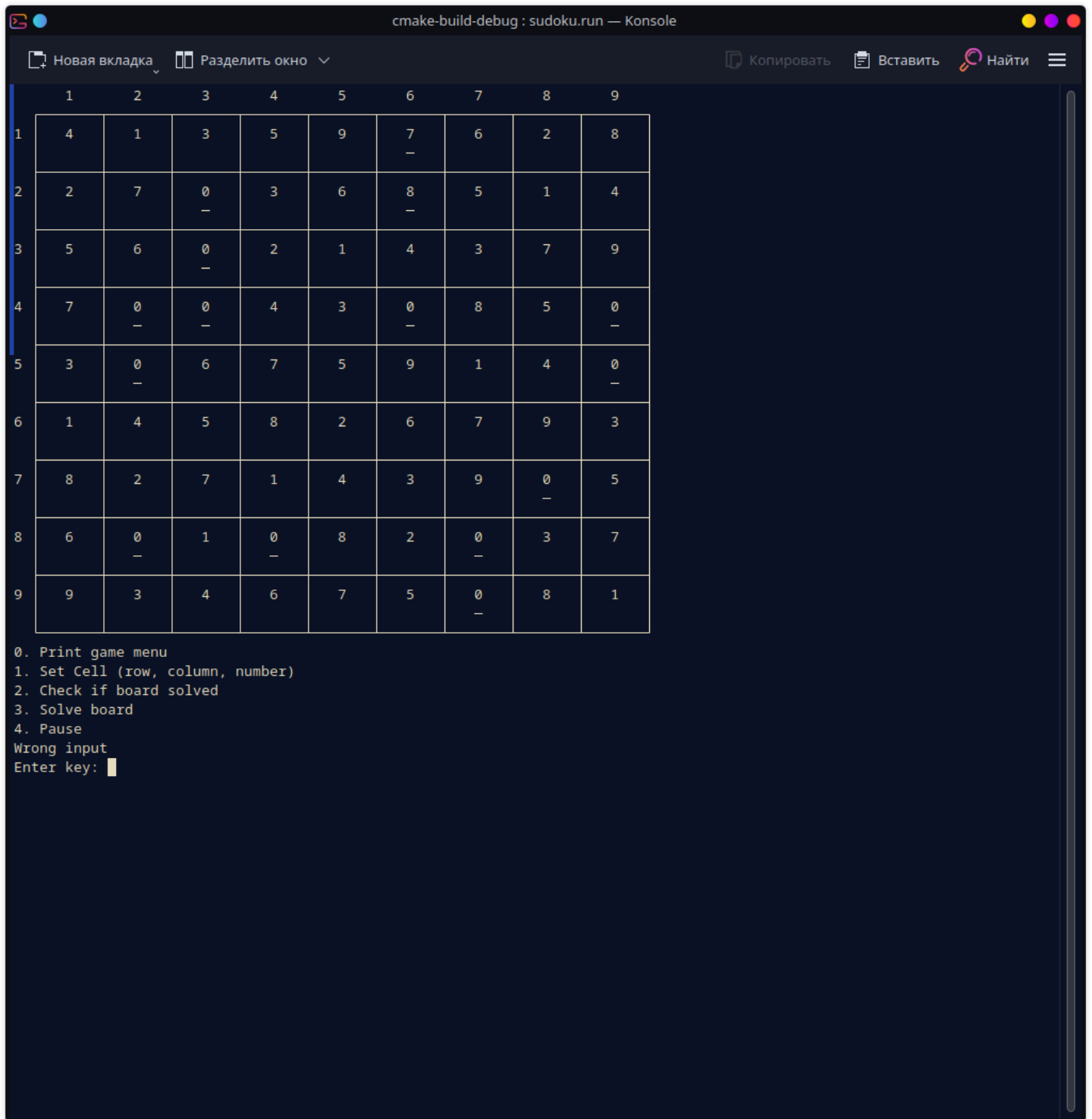
1. Set Cell (row, column, number)

2. Check if board solved

3. Solve board

4. Pause

Enter key: asd



6. Testing

- For testing, I used the `boost.Test` framework.

6.1 Testing Sudoku engine

6.1.1 Matrix Tests Description

These tests collectively ensure the proper functioning of crucial methods in the Matrix class, contributing to the overall reliability of the matrix-related operations in your Sudoku application.

6.1.1.1 Constructor Test (`BOOST_AUTO_TEST_CASE(ConstructorTest)`)

This test case checks the initialization of a Matrix object. It verifies that the constructor properly sets the number of rows and columns and initializes each cell to zero.

6.1.1.2 Print Test (`BOOST_AUTO_TEST_CASE(PrintTest)`)

The Print Test case demonstrates the functionality of the `print` method in the Matrix class. It initializes a Matrix object, prints its content, and allows visual inspection of the printed matrix.

6.1.1.3 Row Contains Test (`BOOST_AUTO_TEST_CASE(RowContainsTest)`)

The Row Contains Test case focuses on the `rowContains` method of the Matrix class. It verifies that the method correctly identifies the numbers present in a specified row and returns them in a vector. The test covers various scenarios, including empty vectors when the row or column index is out of bounds.

6.1.1.4 Column Contains Test (`BOOST_AUTO_TEST_CASE(ColumnContainsTest)`)

The Column Contains Test case assesses the correctness of the `columnContains` method in the Matrix class. It checks if the method accurately identifies the numbers present in a specified column and returns them in a vector. The test covers scenarios where the column or row index is out of bounds, resulting in empty vectors.

6.1.2 Sudoku9x9 Tests

- For this class, a testing wrapper `Sudoku9x9TestWrapper` was created.

6.1.2.1 Sudoku9x9_load_dump_Test

Description

This test suite focuses on the `loadBoard` and `dumpBoard` functionalities of the `Sudoku9x9`. The `loadBoard` function reads a Sudoku board from a string representation, while `dumpBoard` converts

the internal board state back to a string. The test ensures that the board can be loaded and dumped without losing information.

Test Cases

1. **Test:** Load and dump a Sudoku board with positive and negative cell values.
2. **Test:** Load and dump a Sudoku board with all positive cell values.
3. **Test:** Load and dump a Sudoku board with all negative cell values.
4. **Test:** Load and dump an empty Sudoku board.

6.1.2.2 Sudoku9x9_setCell

Description

This test suite focuses on the `setCell` functionality of the `Sudoku9x9`. The `setCell` function sets a value in a specified cell if it is allowed according to the Sudoku rules. The tests cover different scenarios with positive and negative cell values.

Test Cases

1. **Test:** Set positive and negative values in cells based on a predefined board.
2. **Test:** Set positive values in all cells on a predefined board.
3. **Test:** Set negative values in all cells on a predefined board.

6.1.2.3 Sudoku9x9_checkRow

Description

This test suite verifies the `checkRowT` functionality of the `Sudoku9x9`. The `checkRowT` function checks the validity of rows in the Sudoku board and returns the count of positive and negative values in each row.

Test Cases

1. **Test:** Check rows on a predefined board with mixed positive and negative values.
2. **Test:** Check rows on a predefined board with all positive values.
3. **Test:** Check rows on a predefined board with all negative values.
4. **Test:** Check rows on a predefined board with repeated positive values.

6.1.2.4 Sudoku9x9_checkColumn

Description

This test suite focuses on the `checkColumnT` functionality of the `Sudoku9x9`. The `checkColumnT`

function checks the validity of columns in the Sudoku board and returns the count of positive and negative values in each column.

Test Cases

1. **Test:** Check columns on a predefined board with mixed positive and negative values.
2. **Test:** Check columns on a predefined board with all positive values.
3. **Test:** Check columns on a predefined board with all negative values.
4. **Test:** Check columns on a predefined board with repeated positive values.

6.1.2.5 `sudoku9x9_checkSubgrid`

Description

This test suite verifies the `checkSubgridT` functionality of the `Sudoku9x9`. The `checkSubgridT` function checks the validity of 3x3 subgrids in the Sudoku board and returns the coordinates of conflicting cells if any.

Test Cases

1. **Test:** Check subgrids on a predefined board with mixed positive and negative values.
2. **Test:** Check subgrids on a predefined board with all zeros (empty board).
3. **Test:** Check subgrids on a predefined board with conflicting values.
4. **Test:** Check subgrids on a predefined board with repeated positive values.

6.1.2.6 `sudoku9x9_checkIfSolved`

Description

This test suite focuses on the `checkIfSolved` functionality of the `Sudoku9x9`. The `checkIfSolved` function checks whether the Sudoku board is solved based on the Sudoku rules.

Test Cases

1. **Test:** Check if the predefined solved Sudoku board is recognized as solved.
2. **Test:** Check if a predefined unsolved Sudoku board is recognized as unsolved.

6.1.3 SudokuSolver Test Cases

6.1.3.1 Test Case: `testSudokuSolver_solveBoard`

This test case aims to validate the `solveBoard` functionality of the `SudokuSolver` class. It involves creating an instance of the `Sudoku9x9` board, loading a predefined Sudoku board, and then

attempting to solve it using the `solveBoard` method. The final step includes printing the solved board to verify the correctness of the solution.

6.1.3.2 Test Case: `testSudokuSolver_unitTest`

This test case focuses on the unit testing of the `SudokuSolver` class. It involves generating a Sudoku board with 45 numbers initially filled and then dropping 20 numbers. The primary objective is to test the `generateBoard` and `solveBoard` methods, ensuring that the Sudoku board is correctly generated and successfully solved. The intermediate and final board states are printed for visual inspection.

6.2 Testing sudoku app

This part of program was tested manually.

7. Source code

All source can be found here -> https://github.com/xxXINFARKTxXx/OOP2_project

8. Conclusion

In the implementation of this project, a reliable and easily extendable architecture for a Sudoku game application was designed. The project successfully realizes key functionalities, including saving and loading game sessions both at launch and during runtime, a user interface for puzzle-solving, and tools to aid in puzzle resolution.

At each stage of the application's operation, scenarios of incorrect interaction between the program and the user were considered, with checks for invalid input.

The code is organized into modules, contributing to ease of maintenance and functionality expansion. Testing of modules includes both functional and unit testing. The test suite covers critical functionalities of the Sudoku game engine, including the `Matrix` and `Sudoku9x9` classes, as well as the `SudokuSolver` component. The application's functionality was also tested manually.

Thus, the goal of the project has been achieved, and the set tasks have been successfully completed.