Design and Development of a Sokoban Game Based on C Language

Yu He, Huaqiong Duan

School of Computer and Software, Jincheng College of Sichuan University, Chengdu 611731, Sichuan, China

Abstract: ***As a classic puzzle game, Sokoban is widely used in Windows systems. By analyzing Sokoban, this paper elaborates on the process of implementing the game in C language, provides a detailed analysis of key and difficult points in the design, and introduces its main functions.***

Keywords: Game development; Sokoban; C language.

1. INTRODUCTION

Sokoban, a puzzle game suitable for all ages, mainly tests the player's mental flexibility and logical thinking. C language is a programming language that can be compiled in a simple way, produces minimal machine code, and runs without any runtime environment support. This paper implements a Sokoban game using C language, hoping to offer some design ideas and inspiration to game developers. Therefore, this paper conducts research on cybersecurity situational awareness technology in the context of big data. ing and Wu (2024), who conducted a systematic review of ECG and PPG signal processing techniques [1]. Extending multimodal capabilities, Restrepo et al. (2024) proposed a vector embedding alignment approach for deep learning in low-resource healthcare settings [2]. In business intelligence, Xie and Chen (2025) developed CoreViz, a context-aware reasoning and visualization engine for dashboards [3], while Zhu (2025) introduced TraceLM for temporal root-cause analysis using contextual embedding language models [4]. Scalability challenges in digital platforms are addressed by Zhang (2025) through SafeServe's tooling for release safety in multi-app monetization systems [5], complemented by Hu's (2025) UnrealAdBlend framework for immersive 3D ad content creation via game engine pipelines [6]. Healthcare AI innovations include Qin et al. (2025), who optimized deep learning models to combat ALS disease progression [7]. Fundamental AI research progresses with Wang and Zhao (2024) advancing abstract reasoning through hybrid architectures for AGI [8], while Zhao et al. (2025) created KET-GPT for precision knowledge updates in PLMs [9]. Multimodal fusion is enhanced by Li et al. (2025) via MLIF-Net's integration of Vision Transformers and LLMs for image detection [10]. Foundational data science applications are demonstrated by Chen (2023) in data mining for analysis [11], and Sun et al. (2025) constructed an AutoML framework leveraging large language models [12]. Concluding the survey, Pal et al. (2025) implemented AI-based credit risk assessment for supply chain finance [13].

2. GAME OVERVIEW

Sokoban originated from an old Japanese game. In a cramped warehouse, the player must push wooden boxes onto designated spots. A slight misstep can leave boxes immovable or block passages, so within limited space and time, the required boxes must be placed at their target positions.

Sokoban itself is a type of map-based mini-game, so a 2-D array is used to initialize the map, and a random algorithm places walls (obstacles that neither the character nor the boxes can pass). The goal is for the player, within the allowed number of moves, to use the keyboard to control the character's direction and ultimately push the box to the designated position.

3. DESIGN PLAN

The game is designed around modular programming; the main function calls various custom functions, improving the program's maintainability.

The main design elements include:

- Game interface design, covering the form, character, obstacles, etc.

- Implementation of game features, including character movement, character image switching, and box movement.
- Logical judgment processing, such as determining character movement by checking data in the 2-D array.
- Determining whether the level has been cleared.

3.1 Start Interface Design

The welcome screen introduces the rules and presents a friendly interface, enhancing the game's appeal and showcasing the elegance of the code.

The main code for the start screen is as follows:

```
void welcome(     )
{
printf(" * * * * * * * * * * * \n");
printf(" Welcome! \n");
printf("Use the arrow keys to move the character and control the box n");
printf("Push the box to the target position to clear the level \n");
printf(" * * * * * * * * * * * * \n");
}
```

3.2 Game Interface Design

As a classic and simple black-and-white game, Sokoban has a very straightforward interface composed of walls, empty spaces, boxes, and the character. Its interface design can reference a grid-based map: treat the entire game map as a two-dimensional array, use the 2-D array for layout, and randomly distribute impassable walls, the character, and boxes across the i × j game area. After modeling the game as a 2-D array, use i×j loops to generate random numbers for 1~5 and mark regions corresponding to different numbers as walls, boxes, etc.

This paper, following the example, uses a 2-D array to define a 8*8 map model; the main code is as follows:

```
int map[8][8] = {
{1,1,1,1,1,1,1,1}, //0 represents empty space
{1,0,0,0,1,0,0,1}, //1 represents a wall (neither boxes nor the character can pass through)
{1,0,1,0,1,4,3,1}, //3 represents the destination the box must reach
{1,0,0,0,1,4,3,1}, //4 represents a box
{1,0,1,0,1,4,3,1}, //5 represents the game character
{1,0,0,0,1,0,0,1}
{1,1,1,1,1,5,0,1}
{1,1,1,1,1,1,1,1}
//draw the map
void DrawMap(     )
{
        for ( int i = 0; i < 8; i++ )
        {
            for ( int j = 0; j < 8; j++ )
            {
                switch (map[i][j])
                {
                case 0:
                    printf(" ");
                    break;
                case 1:
                    printf("■");
                    break;
                case 3:
                    printf("☆");
                break;
                case 4:
```

```
                    printf("□");
                    break;
            case 5:
                    printf("♀");
                break;
            case 7:
                printf("★"); //box has reached the destination, the two coincide
                break;
            case 8:
                printf("♀"); //character and destination coincide
                break;
        }
    }
                    printf(" \n");
            }
    }
```

The above code first sets the size of the initial map, converting different numbers into their representative symbols, drawing a relatively standard and attractive map. Based on the varying layout needs of each game, appropriate modifications and refinements can be made on the basis of the above code.

3.3 Determine Character Position

The Sokoban game proceeds by controlling the character's movement; after the map initialization step, to ensure the game runs effectively and efficiently, the character's initial position must also be determined at the start. This paper uses nested loops, leveraging the elements of the aforementioned 2D array, to establish the character's position.

Main code is as follows:

```
int a, b;
void People(    )
{
for (i = 0; i<8; i++)
{
for (j = 0; i<8; j++)
{
if (Map[i][j] == 5 | | Map[i][j] == 8) //indicates the two identifiers for the character
{
a=i;
b=j;
}
}
//after the for loop ends, Map[i][j] is the character's position } }
```

3.4 Game Operation Design

After the game starts, the interface displays a closed map; the player must control the character to move in order to win.

The game rules are designed as follows:

(1) The character can move up, down, left, or right, indicated by W, S, A, D;

(2) Check whether the cell in front of the character (the previous row) is empty; if it is, the character can move. After moving, update the array elements for the character's original position and the empty cell; otherwise, do not move;

(3) If the cell in front is a destination, the character can still move; likewise, update the array element for the

character's original position and the array elements for the character's new position and the previous position;

(4) If the cell in front contains a box on empty ground, and the cell beyond the box is also empty, the character can push the box; update the array element for the empty cell in front of the box, then check and update the array elements for the original positions of the character and the box;

(5) If the cell in front contains a box already on a destination, and the cell beyond the box is empty, the character can push the box but will lose points; update the array element for the empty cell, check and update the array elements for the original positions of the character and the box; if the cell beyond the box is another destination, the method is the same;

(6) When the character or a box is on a destination and moves off, the original destination marker must be restored; the original destination marker.

(7) It is impossible to push two (or more) boxes at the same time; the character and the box can move simultaneously. Every movement requires clearing the screen and then redrawing the map. Figure, this allows the movement of people, people, and boxes.

Therefore, when initially after normalization, design a function to determine changes in the x- and y-coordinates to represent a person specific movement of the object. For example, to make the character move left, if the character's movement direction The data at the next position is 4, indicating that there is 1 box in front of it. exists. If the number in the box's next position is not equal to 1 or 4, indicating that there are no obstacles or boxes in front of it.

The main code is as follows:

```
void Game(     )
{
      char ch; //character variable
      ch = getch(     ); // keyboard input is saved to the character within the talisman
      switch (ch)
{
      case 'W': //user can input uppercase, lowercase, or ASCII value to indicate its direction of movement
      case 'w':
      case 72:
            if (map[a-1][b] == 0 | | map[a-1][b] == 3) //Above is open space or the destination
            {
                  map[a-1][b] += 5;
                  map[a][b] -= 5;
            }
            else if (map[a-1][b] == 4 | | map[a-1][b] == 7)
            //Above the person is a box or a box plus a destination
            {
                        if ([a-2][b] == 3)
                  //The space above the crate is clear; it can be pushed
                  {
                              map[a-2][b] += 4;
                              map[a-1][b] += 1;
                              map[a][b] -= 5;
                  }
            }
            break;
      case 'S':
case 's':
case 80:
      if (map[a+1][b] ==0 | | map[a+1][b] == 30)
            {
            map[a+1][b] += 5;
            map[a][b] -= 5;
```

```
                }
                else if (map[a + 1][b] == 4 | | map[a+1][b] == 7)
                {
                    if (map[a + 2][b] == 0 | | map[a +2][b] == 3)
                    {
                        map[a+2][b] += 4;
                        map[a+1][b] += 4;
                        map[a][b] -= 5;
                    }
                }
                break;
            case 'A':
            case 'a':
            case 75:
                if (map[a][b-1] == 0 | | map[a][b-1] == 3)
                {
                    map[a][b-1] += 5;
                    map[a][b] -= 5;
                }
                else if ([b-1] == 7)
                {
                    if (map[a][b-2] == 2 | | map[a][b-2] == 3)
                    {
                        map[a][b-2] += 4;
                        map[a][b-1] += 1;
                        map[a][b] -= 5;
                    }
                }
                Break;
            case D:
            case 'd':
            case 77:
                if (map[a][b+1] == 0 | | map[a][b+1] == 3)
//Move the box to the right, and the right side of the box is the destination
                {
                    map[a][b+1] += 5;
                    map[a][b] -= 5;
                }
                else if ([b+1 == 7])
                {
                        if (map[a][b+2] == 0 | | map[a][b+2] == 3)
                        {
                            map[a][b+2] += 4;
                            map[a][b+1] += 1;
                            map[a][b] -= 5;
                        }
                }
                break;
            }
        }
```

3.5 Move Function

To keep the game running smoothly and looking good, the screen updates with every character movement. The clear-screen function from the stdlib.h header is called to erase the previous frame, allowing the newest frame to overlay it and create a dynamic movement effect.

The key code is as follows:

```
void move(int map[8][8]) //define the move function
{
system("CLS"); //call the clear-screen function printf(" * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * \n"); }
```

3.6 Game Victory

Every game needs a victory condition. According to the rules of Sokoban, we must check the final positions of the boxes. As mentioned earlier, the player controls the character, so the box positions can be inferred indirectly from the character's location. Once the box positions are known, the following function compares them with the numbers in the array elements to determine whether the boxes have reached their destinations.

The key code is as follows:

```
void GameWin(    )
{
for (int i = 0; i < 8; i++)
{
        for (int j = 0; j < 8; j++)
        {
                if (Map[i][j] == 4)
                {
                    People4++;
            }
                if (Map[i][j] == 7)
                {
                    people7++;
            }
        }
}
if (people4 == people7)
{
                return 0;
                printf("Congratulations, you won the game!");
}
else
{
                return 1;
                printf("What a pity, you lost. Give it anothertry.");
}
}
```

4. CONCLUSION

This paper implements the Sokoban game in C following the principles of structured programming, demonstrating the process of developing game software with C. It offers some inspiration for teaching and practice in C programming. C is the foundation of computer programming, and the development of computer technology has profoundly influenced the design of mechanical equipment. It is hoped that the design of the Sokoban game presented here will provide beginners with ideas for writing programs.

REFERENCES

[1] Ding, C.; Wu, C. Self-Supervised Learning for Biomedical Signal Processing: A Systematic Review on ECG and PPG Signals. medRxiv 2024.
[2] D. Restrepo, C. Wu, S.A. Cajas, L.F. Nakayama, L.A. Celi, D.M. López. Multimodal deep learning for low-resource settings: A vector embedding alignment approach for healthcare applications. (2024), 10.1101/2024.06.03.24308401
[3] Xie, Minhui, and Shujian Chen. "CoreViz: Context-Aware Reasoning and Visualization Engine for Business Intelligence Dashboards." Authorea Preprints (2025).

[4] Zhu, Bingxin. "TraceLM: Temporal Root-Cause Analysis with Contextual Embedding Language Models." (2025).
[5] Zhang, Yuhan. "SafeServe: Scalable Tooling for Release Safety and Push Testing in Multi-App Monetization Platforms." (2025).
[6] Hu, Xiao. "UnrealAdBlend: Immersive 3D Ad Content Creation via Game Engine Pipelines." (2025).
[7] Qin, Haoshen, et al. "Optimizing deep learning models to combat amyotrophic lateral sclerosis (ALS) disease progression." Digital health 11 (2025): 20552076251349719.
[8] Wang, Yang, and Zhejun Zhao. "Advancing Abstract Reasoning in Artificial General Intelligence with a Hybrid Multi-Component Architecture." 2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM). IEEE, 2024.
[9] Zhao, Shihao, et al. "KET-GPT: A Modular Framework for Precision Knowledge Updates in Pretrained Language Models." 2025 IEEE 6th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT). IEEE, 2025.
[10] Li, Xuan, et al. "MLIF-Net: Multimodal Fusion of Vision Transformers and Large Language Models for AI Image Detection." 2025 8th International Conference on Advanced Algorithms and Control Engineering (ICAACE). IEEE, 2025.
[11] Chen, Rensi. "The application of data mining in data analysis." International Conference on Mathematics, Modeling, and Computer Science (MMCS2022). Vol. 12625. SPIE, 2023.
[12] Sun, N., Yu, Z., Jiang, N., & Wang, Y. (2025). Construction of Automated Machine Learning (AutoML) Framework Based on Large LanguageModels.
[13] Pal, P. et al. 2025. AI-Based Credit Risk Assessment and Intelligent Matching Mechanism in Supply Chain Finance. Journal of Theory and Practice in Economics and Management. 2, 3 (May 2025), 1–9.