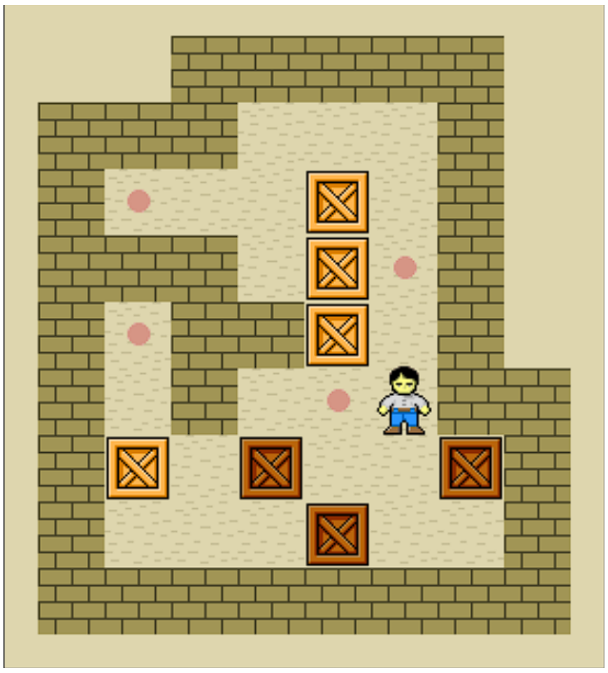
RecurseBox

Designer: Xiao Jiachen, Gong Linghu

1 Introduction

Sokoban is a puzzle video game in which the player pushes boxes around in a warehouse, trying to get them to storage locations.



This is a classical game first published in December 1982. In this project, you are required to develop a game based on *Sokoban*, while adding some extra rules to make this classic more interesting.

2 Requirements

2.1 Basic Part (80pts)

1. **Programming language: C/C++**.
   You must mainly use C/C++ to implement. It's OK to use other languages, but no more than 30% of all your source code. **IF THIS REQUIREMENT IS NOT SATISFIED, YOU WILL GET A VERY LOW SCORE.**

2. **Command Line Interface (10pts)**
   You may use CLI to interact with the players (GUI is in bonus part). A typical CLI game is like this

```
# # # # # # # # # # # | P: Player                 |
# P . . . . . . . = # | p: Player on checkpoint   |
# . . . . . . . . . # | O: Box with no internal   |
# . . . . . . . . . # | =: Checkpoint             |
# . . . O . . . . . # | -: Storage Point          |
# . . . . O . . . . # | #: Wall                   |
# . . . . . O - - - # | o: Box on storage point   |
# . . . . . . . . . # |                           |
# . . . . . . O . - # | W/A/S/D: Move             |
# O o . . . . . . . # | Esc: Pause                |
# # # # # # # # # # # | Current Level: 1          |
```

More specifically, you should have the following components

- Home page of a game, i.e. title of a game, and with a start menu
- Interface to select the game levels
- Interface for game displaying and operating
- Notification of game status, e.g. if you finish a level, tell the player in any possible form

3. **Implementation of Basic *Sokuban* Rules (15pts)**
The following is the description of *Sokuban* from Wikipedia (https://en.wikipedia.org/wiki/Sokoban).

> The game is played on a board of squares, where each square is a floor or a wall.
>
> Some floor squares contain boxes, and some floor squares are marked as storage locations.
>
> The player is confined to the board and may move horizontally or vertically onto empty squares (never through walls or boxes).
>
> To move a box, the player walks up to it and pushes it to the square beyond. Boxes cannot be pulled, and they cannot be pushed to squares with walls or other boxes.
>
> The number of boxes equals the number of storage locations. The puzzle is solved when all boxes are placed at storage locations.

Here are things different from original game:

1. You can push more than one box simultaneously.

2. Not only all the storage points should have one box, but also the player should arrive at the checkpoint.

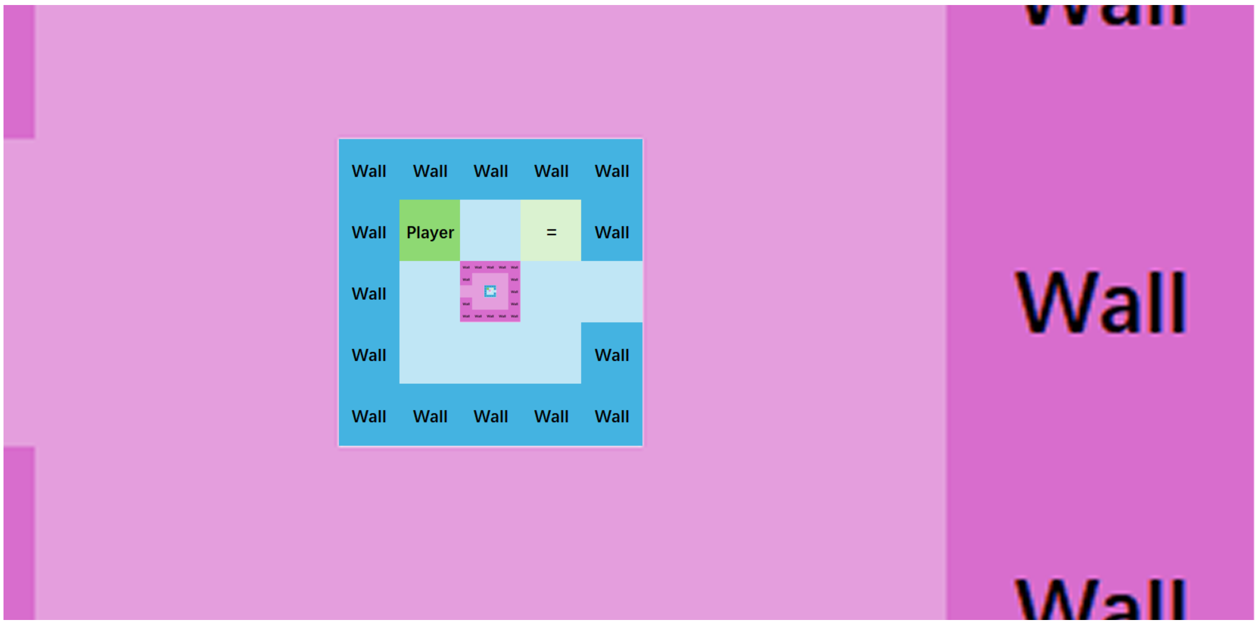
4. **Implementation of Extra Rules (45pts)**

- **Internal Structures (10pts)**
  The boxes can **have internal structures** like a common *Sokoban* level. This means that the player, or other boxes can enter the box with internal structure when that box cannot be moved. Entering point should be the round down middle point of that entering edge.

  What's more, all the internal structures is visible as in a box.

```
B1:
# # # # # | Bn: A box with internal structure and id:n |
# P . = # | bn: that box sit on storage point          |
# . B2. .
# . . . #
# # # # #

B2:
# # # # #
# . . . #
. . B1. #
# . . . #
# # # # #
```



  If an object tries to enter a box but find that some other thing block the entering point, that object will try to push that thing away, or entering that thing if cannot be moved.

- One box can only be contained by other boxes only once.

- **Recursive Box (5pts)**
  If objects go out of the border of a box A, it will return to the box B that contains the box A, start from the position of A and move one step forward. If go out of B's

border, recursively go out. If no box contains A, you should go into a default 5x5 empty space. (5pts)

- If go out of box A to B and find that the target point is a box C that cannot be moved, you should directly slides from A into C with the round position. (10pts)

  ```
  B1:
  # # # #
  # B2B3#
  # # # #

  B2:
  # . P
  # . .
  # . .

  B3:
  w . # | w is the point that player
  . . # | moves to right
  . . #
  ```

- **Recursion Overflow (5pts)**
  If box A contains A, and you go out from it infinite times, you should be seen as go out of a special box called `inf`. That box can be contained in other box or stay in the default 5x5 empty space.

- **Greedy Snake (5pts)**
  If you push an object that eventually pushing yourself, all the objects should move one step.

```
B1:
# # # # # # # # #
# # # # # # # # #
# B1O B2B3O O P O
# # . # # # . . #
# # . . . . . . #
# # . . . . . . #
# # . . . . . . #
# # # # # # # # #
# # # # # # # # #

B2:
# # #
# - #
# O #

B3:
# # #
# = #
# . #
```

- **Multiple reference (10pts)**
  A box can have multiple copies with one true father. That means copies of one box can appear in multiple boxes (from multiple boxes can enter that box), but go out of that box will only go to the true father.

5. **Save and load (10pts)**
   You should be able to save and load the game status. You can store the game data in the format you want (e.g. a save file). Also, the program should be able to detect whether the save file is broken.

2.2 Bonus Part (up to 30pts in total)

Here are some aspects that you can get the bonus.

New Functions

1. **Graphical Interface (up to 15pts)**
   You are encouraged to use game development frameworks (or engines) like SDL2 to develop a graphical interface, or you can build it from scratch like OpenGL if you like.

   You will get the points of CLI implementation in the basic part if you develop a graphical interface. You should not build a beautiful GUI but lack features of this game.

2. **Undo (up to 5pts)**
   You can implement undo and restart so that user can cover their mistakes. You are not encouraged to just directly store all the status(although this still count), a better way is to just store the difference.

3. **Auto solving (up to 10pts)**
   You can use some algorithms to automatically solve the puzzle.

   > Original *Sokoban* can be studied using the theory of computational complexity. The problem of solving Sokoban puzzles was first proved to be NP-hard. Further work showed that it was significantly more difficult than NP problems. This is of interest for artificial intelligence research because solving *Sokoban* can be compared to the automated planning required by some autonomous robots.

   Score will depends on the method you are using. You may not be able to solve it but what matters is the process.

More Extra Rules

5. **Time travel (up to 10pts)**
   There are two time lines, before and after. If you changed something before, that will affect after as just reset the position. Boxes in after can be pushed into before, but before cannot push in after. If pushing before boxes have conflics with after, should be seen as the before boxes already takes place and after boxes go into that place will push it.

   Travel between before and after can be determined by yourself, as a box, or another keyboard event, or anything you like.

6. **Multiple infinite and multiple epsilon (10pts)**
   Infinite go out will cause infinite, Infinite entering will cause epsilon. When a box contains infinite, and go out from infinite and then go out infinite times again, then it will be in 2-infinite.

   ```
   B1:
   #  inf B1 P  #
   #  _   _  _  # | move upwards will cause a infinity,|
   #  #   #  #  # |   then go out inf and make another |
   #  #   #  #  # |   infinity, result in 2-infinity   |
   #  #   #  #  #
   ```

   Also works for n-infinite and n-epsilon.

Anything you think is worth doing (up to 20pts)

You can try to develop the following functions, which is much harder than previous ones:

- Multiple players
- Level editor

- Sharing levels through network
- …🤗

3 Grading Policy

Note that your project score contains the following functional aspects:

- Basic gameplay development (80pts)
  - Command line interface (10pts)
  - Implementations of basic *Sokuban* rules (15pts)
  - Implementation of extra rules (45pts)
  - Save and load (10pts)
- Bonus part (up to 30pts in total)

Also, we will judge the following non-functional aspects:

- Project management
- Unified code style - your team should have a standard
- Robustness

You may earn extra points if you do well on the aspects above, nevertheless deductions on the score will be made if you do poorly.

About the Presentation

> Talk is cheap, show me your game!

Remember to send us a executable copy of your game in advance. Players' experience is significant.

During the presentation, please briefly show the functions you developed. We tend to ask you details of implementaion during the presentation, instead of the game play.

4 References

1. Wikipedia, Sokuban: https://en.wikipedia.org/wiki/Sokoban (https://en.wikipedia.org/wiki/Sokoban)
2. Simple DirectMedia Layer (SDL): https://www.libsdl.org/ (https://www.libsdl.org/)
3. The Godot Engine: https://godotengine.org/ (https://godotengine.org/)