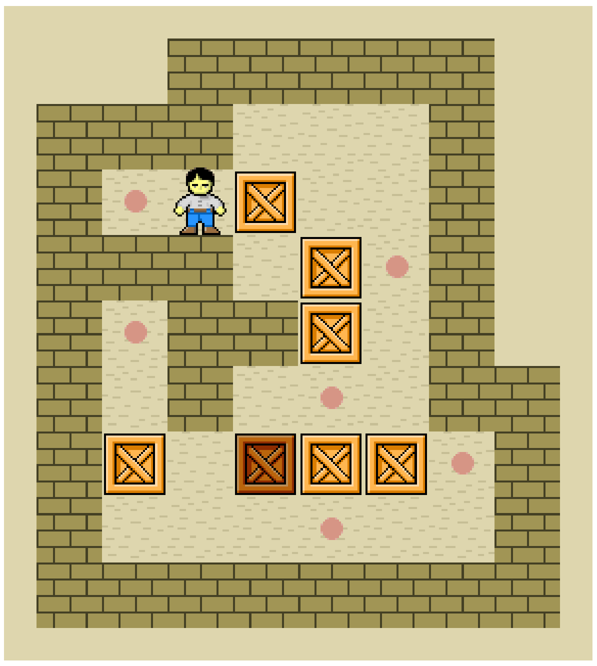
Sokoban

Overview



Sokoban is a simple Japanese puzzle game from the 80's, although the rules are simple, levels can be deceptively hard.

The main idea of Sokoban is a small grid board, containing walls, boxes, storage locations and a single player. The player moves around the board pushing boxes around until they manage to get all boxes stored in one of the storage locations! Feel free to check out the wikipedia entry for Sokoban too.

For this assignment, you will be building both a level generator for the game, as well as the mechanics to play your created levels!

Getting Started

1. Create a new folder for your assignment. Here is an example:

```
$ mkdir ass1
$ cd ass1
```

2. Fetch the starter code using the command below. Alternatively download the starter code here.

```
$ 1511 fetch-activity cs_sokoban
```

3. Check that everything works by running the autotest.

```
$ 1511 autotest cs_sokoban
```

(These should fail initially. If you want to exit running the autotest midway, press `[ctrl-c]` .)

Initial Code and Data Structures

The starter code for this assignment includes some functions and defined types:

- A `print_board(...)` function:
  - This prints out the current state of the board, allowing the user to play the game.
  - This ensures that the board will be consistent between you and the autotest.
- A `init_board(...)` function:
  - This sets up the board with default values.
- A `struct tile` struct:
  - Each square of the board (aka the 2D array) holds a `struct tile` . This tells us information about what is at that location in the board.
  - Note that the player location is stored separately to the contents of the board.
- A `enum base` enum:
  - This is used within `struct tile` . Every position on the board must be exactly 1 of these values (`NONE` , `WALL` , or `STORAGE`).

+ Closer look at [struct tile board]

Reference Implementation

To help you understand how the assignment works, we have written a reference implementation for you to run. You can run it via the command `1511 cs_sokoban` .

```
$ 1511 cs_sokoban
=== Level Setup ===
...
```

+ Example Program

FAQ

+ FAQ

Stages

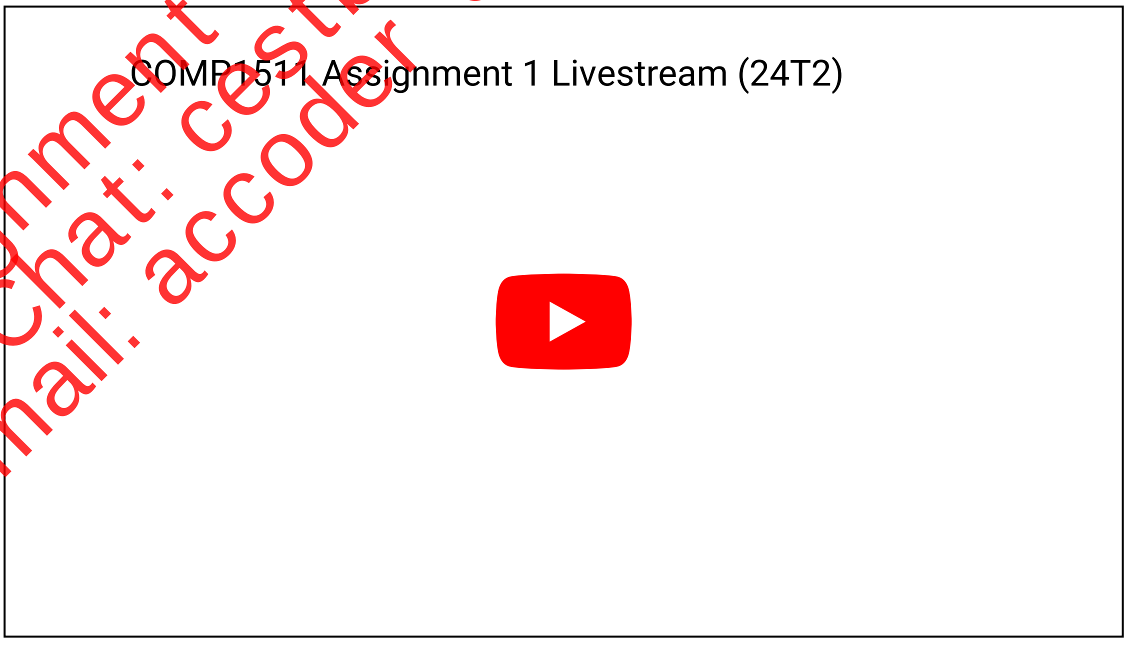
We have broken the assignment spec down into incremental stages:

- **Stage 1:** Level Builder - adding boxes, walls, and storage locations.
- **Stage 2:** Player Movement - adding player starting location, player movement and move counter.
- **Stage 3:** Core Game Mechanics - moving boxes, win condition, and reset level.
- **Stage 4:** Advanced Game Mechanics - undo command, pushing multiple boxes, and linking boxes.
- **Extension (no marks):** Adding graphics, multiple levels per game, and storing levels in text files.

Your Tasks

This assignment consists of four stages. Each stage builds on the work of the previous stage, and each stage has a higher complexity than its predecessor. You should complete the stages in order.

A video explanation to help you get started with the assignment can here found here:



Stage 1 ●○○ | Stage 2 ●○○ | Stage 3 ●●○ | Stage 4 ●●● | Extension | Tools

Stage 1

In Stage 1, you will set up the level builder for the Sokoban game. It includes asking the user for commands to place walls, boxes, and storage locations.

There is a reference implementation that you can play by running the command `1511 cs_sokoban` in the terminal.

Here is an example of input for a finished stage 1 that you can try within the reference implementation.

```
$ 1511 cs_sokoban
w 0 0
s 1 1
s 10 10
b 1 0
b 1 1
W 0 0 9 0
[ctrl-d]
```

(note: the lowercase `w` and capital `W` are different commands)

Stage 1.1: Setting up level builder

To create a level builder, we need to give the user the ability to add items onto the board via commands until they decide to start playing the game. In this substage you will create a command loop that lets players add `WALL` or `STORAGE` to the board.

In this substage, you will need to do the following:

1. Print out `=== Level Setup ===\n` .
2. Setup a loop for reading in commands.
3. Given the command: `w [row] [col]` a wall will be placed at position `[row][col]` .
4. Given the command: `s [row] [col]` a storage location will be placed at position `[row][col]` .
5. Before reading in the next command, print out the board with the provided `print_board()` function.
6. The loop should stop when the user inputs `[ctrl-d]` .

Don't worry about error checking for this, we will cover that in a future stage :)

HINT:

Have a look back at your code for "CS Calculator" from week 4.

NOTE:

Since we don't have a player position yet, just pass `-1, -1` into `print_board()` for the player positions.

NOTE:

Remember that we need a space before the `" %c"` as we don't want to read in the new lines that get entered between commands as characters!

Clarifications

- If a wall gets placed on a storage location, or vice versa, override the value.
- We will only test valid `row` and `col` values within the map during this stage.
- We will only test valid commands.

Examples:

+ Example 1.1.1: Adding Walls and Storage Location

+ Example 1.1.2: Overriding Values

Autotest

NOTE:

You may like to autotest this section with the following command:

```
1511 autotest-stage 01_01 cs_sokoban
```

Remember to do your own testing!

Stage 1.2: Array bounds checking

In stage 1.1, we didn't check if the given values are outside of the bounds of our array/board. Currently our program will crash in these cases, so let's fix that now!

In this substage, you will need to do the following:

1. When the user enters the `w` or `s` command, check that `row` and `col` are within the bounds of the board. If either one is out of bounds, print out `Location out of bounds` and don't modify the board.

HINT:

You might want to create a function to do this, as we will need to check this in future stages too!

Clarifications

- `row` and `col` will always be integer values.

Examples

+ Example 1.2.1: Out of Bounds

Autotest

NOTE:

You may like to autotest this section with the following command:

```
1511 autotest-stage 01_02 cs_sokoban
```

Remember to do your own testing!

Stage 1.3: Add boxes

Add functionality for boxes to be added to the board. Although boxes can exist in the same tile as a storage location, we should never have a box and a wall at the same position.

In this substage, you will need to do the following:

1. Given the command `b [row] [col]`, add a box at that location.
   - If there is a storage location there, we will end up with both a storage location AND a box.
   - If there is a wall there, change the base from `WALL` to `NONE` and add in the box.
2. Check that `row` and `col` are within the bounds, else, print `Location out of bounds`.
3. Edit your `w` so that if someone places a wall where a box already exists, it removes the box.

Examples

+ Example 1.3.1: Adding Boxes

+ Example 1.3.2: Adding Boxes and Storage Location

+ Example 1.3.3: Boxes and Walls

+ Example 1.3.4: Out of Bounds

+ Example 1.3.5: Walls overriding boxes

Autotest

NOTE:

You may like to autotest this section with the following command:

```
1511 autotest-stage 01_03 cs_sokoban
```

Remember to do your own testing!

Stage 1.4: Add lines of walls

Add the **capital** `W` commmand to add lines of walls.

In this substage, you will need to the following:

1. Given the command `W [start_row] [start_col] [end_row] [end_col]` , add a line of walls from `[start_row] [start_col]` to `[end_row][end_col]` .
2. If both start and end locations are out of bounds, print `Location out of bounds` .
3. If only part of the line is out of bounds, add in the valid positions (ignore the out of bounds values) and don't print an error message.

NOTE:

Remember that walls will override any boxes or storage locations that currently exist there.

Clarifications

- We will only ask for values that form horizontal or vertical lines. We will never ask for a diagonal line.
- We will only test values when `start_row` <= `end_row` and `start_col` <= `end_col`

Examples

+ Example 1.4.1: Adding Lines of Walls

+ Example 1.4.2: Overriding

+ Example 1.4.3: Out of Bounds

Autotest

NOTE:

You may like to autotest this section with the following command:

```
1511 autotest-stage 01_04 cs_sokoban
```

Remember to do your own testing!

Testing and Submission

Are you finished with this stage? If so, you should make sure to do the following:

- Run `1511 style`, and clean up any issues a human may have reading your code. Don't forget -- **20%** of your mark in the assignment is based on style!
- Autotest for this stage of the assignment by running the `autotest-stage` command as shown below.
- Remember -- *give early, and give often*. Only your last submission counts, but why not be safe and submit right now?

```
1511 style cs_sokoban.c
1511 autotest-stage 01 cs_sokoban
give cs1511 ass1_cs_sokoban cs_sokoban.c
```

Assessment

Assignment Conditions

- **Joint work** is **not permitted** on this assignment.

  This is an individual assignment.

  The work you submit must be entirely your own work. Submission of any work even partly written by any other person is not permitted.

  Except, you may use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from a site such as Stack Overflow or other publicly available resources. You should attribute clearly the source of this code in an accompanying comment.

  Assignment submissions will be examined, both automatically and manually for work written by others.

  Do not request help from anyone other than the teaching staff of COMP1511, e.g. in the course forum & help sessions.

  Do not post your assignment code to the course forum - the teaching staff can view assignment code you have recently autotested or submitted with give.

  Rationale: this assignment is designed to develop the individual skills needed to produce an entire working program. Using code written by or taken from other people will stop you learning these skills. Other CSE courses focus on the skill needed for work in a team.

- The use of **code-synthesis tools**, such as **GitHub Copilot**, is **not permitted** on this assignment.

  Rationale: this assignment is intended to develop your understanding of basic concepts. Using synthesis tools will stop you learning these fundamental concepts.

- **Sharing, publishing, distributing** your assignment work is **not permitted**.

  Do not provide or show your assignment work to any other person other than the teaching staff of COMP1511. For example, do not message your work to friends.

  Do not publish your assignment code via the internet. For example, do not place your assignment in a public GitHub repository.

  Rationale: by publishing or sharing your work you are facilitating other students using your work which is not permitted. If they submit your work, you may become involved in an academic integrity investigation.

- **Sharing, publishing, distributing your assignment work after the completion of COMP1511** is **not permitted**.

  For example, do not place your assignment in a public GitHub repository after COMP1511 is over.

Rationale: COMP1511 sometimes reuses assignment themes using similar concepts and content. Students in future terms find your code and use it which is not permitted and you may become involved in an academic integrity investigation.

Violation of the above conditions may result in an academic integrity investigation with possible penalties, up to and including a mark of 0 in COMP1511 and exclusion from UNSW.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalised, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalised if your work is taken without your consent or knowledge.

For more information, read the UNSW Student Code, or contact the course account. The following penalties apply to your total mark for plagiarism:

0 for the assignment	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 for the assignment	Submitting any other person's work. This includes joint work.
0 FL for COMP1511	Paying another person to complete work. Submitting another person's work without their consent.

Submission of Work

You should submit intermediate versions of your assignment. Every time you autotest or submit, a copy will be saved as a backup. You can find those backups here, by logging in, and choosing the yellow button next to 'cs_sokoban.c'.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below.

It is fine if intermediate versions do not compile or otherwise fail submission tests.

Only the final submitted version of your assignment will be marked.

You submit your work like this:

```
$ give cs1511 ass1_cs_sokoban cs_sokoban.c
```

Assessment Scheme

This assignment will contribute 20% to your final mark.

80% of the marks for this assignment will be based on the performance of the code you write in `cs_sokoban.c`

20% of the marks for this assignment will come from manual marking of the readability of the C you have written. The manual marking will involve checking your code for clarity, and readability, which includes the use of functions and efficient use of loops and if statements.

Marks for your performance will be allocated roughly according to the below scheme.

100% for Performance	Completely Working Implementation, which exactly follows the spec (Stage 1, 2, 3 and 4).
85% for Performance	Completely working implementation of Stage 1, 2 and 3.
65% for Performance	Completely working implementation of Stage 1 and Stage 2.
35% for Performance	Completely working implementation of Stage 1.

The Challenge stage of the assignment is NOT worth any marks, but is something fun for you to work on getting to know a new library and building something more visual!

Style Marking Rubric

	0	**1**	**2**	**3**	**4**

Formatting (/5)					
Indentation (/2) - Should use a consistent indentation scheme.	Multiple instances throughout code of inconsistent/bad indentation	Code is mostly correctly indented	Code is consistently indented throughout the program		
Whitespace (/1) - Should use consistent whitespace (for example, 3 + 3 not 3+ 3)	Many whitespace errors	No whitespace errors			
Vertical Whitespace (/1) - Should use consistent whitespace (for example, vertical whitespace between sections of code)	Code has no consideration for use of vertical whitespace	Code consistently uses reasonable vertical whitespace			
Line Length (/1) - Lines should be max. 80 characters long	Many lines over 80 characters	No lines over 80 characters			
Documentation (/5)					
Comments (incl. header comment) (/3) - Comments have been used throughout the code above code sections and functions to explain their purpose. A header comment (with name, zID and a program description) has been included	No comments provided throughout code	Few comments provided throughout code	Comments are provided as needed, but some details or explanations may be missing causing the code to be difficult to follow	Comments have been used throughout the code above code sections and functions to explain their purpose. A header comment (with name, zID and a program description) has been included	
Function/variable/constant naming (/2) - Functions/variables/constants names all follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names do not follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names somewhat follow naming conventions in style guide and help in understanding the code	Functions/variables/constants names all follow naming conventions in style guide and help in understanding the code		
Organisation (/5)					
Function Usage (/4) - Code has been decomposed into appropriate functions separating functionalities	No functions are present, code is one main function	Some functions are present, but functions are all more than 50 lines	Some functions are present, and all functions are approximately 50 lines long	Most code has been moved to sensible/thought out functions, but they are mostly more than 50 lines *max* (incl. main function)	A b m d in fu a li m fu
Function Prototypes (/1) - Function Prototypes have been used to declare functions above main	Functions are used but have not been prototyped	All functions have a prototype above the main function or no functions are used			
Elegance (/5)					
Overdeep nesting (/2) - You should not have too many levels of nesting in your code (nesting which is 5 or more levels deep)	Many instances of overdeep nesting	<= 3 instances of overdeep nesting	No instances of overdeep nesting		

Code Repetition (/2) - Potential repetition of code has been dealt with via the use of functions or loops	Many instances of repeated code sections	<= 3 instances of repeated code sections	Potential repetition of code has been dealt with via the use of functions or loops	
Constant Usage (/1) - Any magic numbers are #defined	None of the constants used throughout program are #defined	All constants used are #defined and are used consistently in the code		
Illegal elements				
Illegal elements - Presence of any illegal elements indicated in the style guide	**CAP MARK AT 16/20**			

Allowed C Features

In this assignment, there are no restrictions on C Features, except for those in the style guide. If you choose to disregard this advice, you **must** still follow the style guide.

You also may be unable to get help from course staff if you use features not taught in COMP1511. Features that the Style Guide identifies as illegal will result in a penalty during marking. You can find the style marking rubric above.

Due Date

This assignment is due 08 July 2024 20:00:00. For each day after that time, the maximum mark it can achieve will be reduced **by 5%** (off the ceiling).

- For instance, at **1 day past the due date**, the maximum mark you can get is **95%**.
- For instance, at **3 days past the due date**, the maximum mark you can get is **85%**.
- For instance, at **5 days past the due date**, the maximum mark you can get is **75%**.
  No submissions will be accepted after 5 days late, unless you have special provisions in place.

Change Log

Version 1.0 (2024-06-17 13:00)	◦ Assignment Released
Version 1.1 (2024-06-18 13:00)	◦ Modifications to style marking rubric regarding illegal elements.
Version 1.2 (2024-06-18 23:00)	◦ Fix autotests so stage 1 tests pass after implementing subsequent stages.
Version 1.3 (2024-06-19 14:00)	◦ Removed sentence that said arrays were not allowed
Version 1.4 (2024-06-19 16:00)	◦ Fixed autotests so that stage 2.1 tests pass after implementing stage 2.2
Version 1.5 (2024-06-20 19:30)	◦ Fixed wording in stage 1.4 to be clearer, made sure stages 1.1 - 1.4 wording is consistant.
Version 1.6 (2024-06-29 23:50)	◦ Removed [ctrl-d] from examples in stage 2.1 and 3.2 which were not necessary.
Version 1.7 (2024-06-30 19:20)	◦ Adding in splashkit example image becasue its so cute!
Version 1.8 (2024-07-01 19:20)	◦ Fixing reference solution with error messges when linking boxes. Adding in an autotest to test this, Fixing wording in spec (4.3, dot point 2.