# BlockWorks: Building the Future One Block at a Time.

LAUNCHING INTO COMPUTER SCIENCE

Table of Contents

# BlockWorks Design and Test Plan

## Introduction

Space Engineers inspires creativity and engineering with the motto, "Unleash your need to create" (Keen, 2024). This philosophy drives the game's immersive world, where players construct complex structures, vehicles, and systems in space. The BlockWorks application supports this creativity by helping players organise and manage the blocks and components needed for their builds. It simplifies resource management, allowing users to focus on creative and engineering challenges.

BlockWorks offers features to view, search, add, delete, and sort blocks efficiently. Players can track material quantities for each block, ensuring they have the resources needed to complete their projects.

## Instructions

### Display All Blocks

Purpose: View all blocks and their components.

How to use: Select option to see a list of blocks with their component details.

### Search Blocks

Purpose: Search for blocks by name.

How to use: Select option, enter a search term to view matching blocks.

### Add Blocks

Purpose: Add a new block.

How to use: Select option, enter a unique block name, specify component quantities, confirm the addition and the block will be added.

### Delete Blocks

Purpose: Remove a block by name.

How to use: Select option, enter the name of the block, confirm deletion, and the block will be removed.

### Sort Blocks

Purpose: Sort blocks alphabetically.

How to use: Select option, the blocks will be displayed in the chosen order.

# Design

## Data Storage

| Display | Data Type / Structure | Name | Description |
|---|---|---|---|
| | list | blocks | A list of the blocks to display |
| | dict | block | The individual block in the blocks list |
| | str | component | The component in the current block |
| | int | quantity | The quantity of the component |

| Add | Data Type / Storage | Name | Description |
|---|---|---|---|
| | str | name | The name of the new block entered by the user. |
| | dict | components | A dictionary to store the components and their quantities. |
| | str | component | The name of a specific component |
| | int | quantity | Quantity of each component. |
| | dict | new_block | The new block to be added to the blocks list. |

| Delete | Data Type / Storage | Name | Description |
|---|---|---|---|
| | str | name | The name of the block to be deleted, entered by the user. |
| | dict | block | The block being check against the name |

| Sort | Data Type / Storage | Name | Description |
|---|---|---|---|
| | int | index | Used to track the block being sorted. |
| | dict | current_block | The block dictionary currently being sorted |
| | int | position | The index of the previous block being compared to current_block |
| | list | blocks | The list of all blocks, sorted alphabetically by name. |

| Search | Data Type / Storage | Name | Description |
|---|---|---|---|
| | list | blocks | The list of all blocks, sorted alphabetically by name. |
| | str | search_name | The name of the block being searched for, entered by the user. |
| | int | low | The lower bound of the search |
| | int | high | The upper bound of the search |
| | int | mid | The middle index compared to the search name |

## Algorithms

### Display

The display function is responsible for displaying all the blocks in the list. If the list is empty, it informs the user that there are no blocks available, otherwise, it prints the name and components of each block.

**Prerequisite:** The list of blocks is initialised and contains dictionaries representing each block.

**Parameter:** A list of dictionaries representing the blocks to be displayed.

```
FUNCTION display_blocks(blocks)
  CHECK IF the collection OF blocks IS EMPTY:
    IF empty, INFORM the user that no blocks are available
    EXIT the function

  FOR EACH block in the collection OF blocks:
    SHOW the name of the block

  FOR EACH component IN the block:
    SHOW the component name and its quantity
END FUNCTION
```

**Testing:** The display function will be tested with empty lists, varying block counts, and component numbers to ensure it handles all cases correctly.

## Add

The add function will allow users to add a new block to a list by providing a unique name and valid components. Blocks with no components will not be added.

**Prerequisite:** The list of blocks is initialised and contains dictionaries representing each block.

**Parameter:** A list of dictionaries representing the blocks. Name of the new block. Number of components.

```
FUNCTION add_block(blocks)
  DEFINE a list of available components

  PROMPT the user to enter a name for the new block
  STORE the name as block_name

  CHECK IF block_name IS valid:
    IF the name is empty, INFORM the user and STOP
    IF the name already exists in blocks, INFORM the user and
STOP

  FOR EACH component IN the available components:
    PROMPT the user to provide a quantity
    REPEAT the UNTIL a valid quantity is provided
    IF the quantity is greater THAN zero:
      RECORD the component and its quantity

  CHECK IF any components were recorded:
    IF no components, INFORM the user and STOP

  CREATE a NEW block using block_name and the recorded
components
  DISPLAY the NEW block

  PROMPT the user to confirm whether to add the block
  IF the user confirms:
    ADD the new block to blocks
    INFORM the user of success
  ELSE:
    INFORM the user that the block was not added

  RETURN blocks
END FUNCTION
```

**Testing:** The add function will be tested with cases such as an empty list, duplicate names, invalid or empty names, missing components, and valid components to ensure correct functionality and handling of edge cases.

## Delete

The delete function will allow users to remove a block by name, verifying its existence and confirming intent before deletion. Blocks will not be deleted if unmatched or declined by the user.

**Prerequisite:** The list of blocks is initialised and contains dictionaries representing each block.

**Parameter:** A list of dictionaries representing the blocks. Name of the block to be delete.

```
FUNCTION delete_block(blocks)
  PROMPT the user to enter the exact name of the block to
delete
  STORE the input as block_name

  IF block_name IS EMPTY:
    INFORM the user that the block name cannot be empty
    RETURN the unchanged list of blocks

  CALL the SEARCH with block_name to locate the matching block
  STORE the result as found_block

  IF found_block IS NOT EMPTY:
    DISPLAY the details of found_block
    PROMPT the user to confirm the deletion of the block
    IF the user confirms:
      REMOVE found_block from blocks
      INFORM the user that the block has been deleted
    ELSE:
      INFORM the user that the block was not deleted
  ELSE:
    INFORM the user that no block was found with the provided
name

  RETURN the updated list of blocks
END FUNCTION
```

**Testing:** The delete function will be tested with cases such as an empty list, single and multiple blocks, non existent block names, and user interactions to ensure proper deletion, error handling, and accurate user prompts.

## Sort - Insertion

The sorting algorithm will organise blocks alphabetically by name using insertion sort, chosen for its time O(n2) complexity on average, which is suitable for small datasets, and its simplicity and stability over the faster O(n log n) merge sort (Cormen et al., 2022; Knuth, 2011).

**Prerequisite:** The list of blocks is initialised and contains dictionaries representing each block.

**Parameter:** A list of dictionaries representing the blocks.

```
FUNCTION insertion_sort_blocks(blocks)
  FOR Each block starting from the second element in blocks:
    STORE the current block as key_block
    SET position to the index of the current block

    WHILE position > than 0 AND the name of key_block < the
name of the block at position - 1:
      SHIFT the block at position - 1 one position forward
      MOVE position one step back

    PLACE key_block at the updated position

  RETURN the sorted list of blocks
END FUNCTION
```

**Testing:** The insertion sort function will be tested with cases such as empty lists, single elements, already sorted lists, reverse sorted lists, and random order to verify its behaviour.

## Search - Binary

The search function will find blocks in the list by name. It uses binary search, which requires sorted input, adding overhead for dynamic datasets but improving efficiency as datasets grow with a time complexity of log O(log n)( Cormen et al., 2022; Knuth, 2011). This choice supports potential program expansion, though the sorting algorithm may need to be reconsider if the dataset scales significantly.

**Prerequisite:** The list of blocks is initialised and contains dictionaries representing each block.

**Parameter:** A list of dictionaries representing the blocks. Name of block to find.

```
FUNCTION search_block(blocks, block_name, sort_order)
  CALL insertion_sort_blocks(blocks, sort_order)

  SET lowerbound to 0
  SET upperbound to the last index of blocks

  WHILE lowerbound <= upperbound:
    SET mid to (lowerbound + upperbound) / 2
    STORE mid_block as the block at index mid

    IF the name of mid_block is equal to block_name:
      RETURN mid_block
    ELSE IF the name of mid_block is smaller than block_name:
      SET lowerbound to mid + 1
    ELSE:
      SET upperbound to mid - 1

  RETURN "Block not found"
END FUNCTION
```

**Testing:** The search function will be tested with cases such as an empty list, exact and partial matches, case insensitive searches, and no matching blocks to ensure it performs correctly and handles various scenarios.

| | Test | Description | Data | Expected Outcome |
|---|---|---|---|---|
| Add Block | A1 | Test adding a new block to an empty list | List = []<br>Block Name = "Armour Cube"<br>Components = {"Steel Plate": 1} | The new block is added to the list. |
| | A2 | Test with a list containing blocks and a unique block name. | List = ["Armour Cube", "Interior Wall"]<br>Block Name = "Door"<br>Components = {"Steel Plate": 2} | The new block is added to the list. |
| | A3 | Test with a list containing blocks and a duplicate block name. | List = ["Armour Cube", "Interior Wall", "Door"]<br>Block Name = "Armour Cube"<br>Components = {"Steel Plate": 2} | The function prevents adding the block and notifies the user that the block name already exists. |
| | A4 | Test with an invalid or empty block name. | List = [],<br>Block Name = ""<br>Components = {"Steel Plate": 1} | The function notifies the user that the block name cannot be empty or invalid. |
| | A5 | Test with no components entered. | List = []<br>Block Name = "Ramp"<br>Components = {} | The function does not add the block and notifies the user that no components were entered. |

| | Test | Description | Data | Expected Outcome |
|---|---|---|---|---|
| Delete Block | D1 | Test with a non-existent block name. | List = ["Armour Cube"]<br>Block Name = "Door" | The function informs the user that there are no blocks to delete. |
| | D2 | Test with multiple blocks and a matching block name. | List = ["Armour Cube", "Steel Plate"]<br>Block Name = "Armour Cube" | The block is deleted from the list after user confirmation. |
| | D3 | Test with user declining the deletion. | List = ["Armour Cube"]<br>Block Name = "Armour Cube"<br>User Response = "no" | The block is not deleted, and the function informs the user that the block was not deleted. |
| | D4 | Test with user confirming the deletion. | List = ["Armour Cube"]<br>Block Name = "Armour Cube"<br>User Response = "yes" | The block is deleted, and the function informs the user that the block was deleted. |

| | Test | Description | Data | Expected Outcome |
|---|---|---|---|---|
| Sort - Insertion | I1 | Test with an empty list. | List = []<br>Sort Order = "A-Z" | The function returns an empty list since there are no blocks to sort. |
| | I2 | Test with a list of blocks sorted in ascending order (A-Z). | List = ["Armour Cube", "Steel Plate"]<br>Sort Order = "A-Z" | The blocks are already sorted, so the list remains unchanged. |
| | I3 | Test with a list containing blocks that are out of order. | List = ["Steel Plate", "Armour Cube", "Iron Rod"]<br>Sort Order = "A-Z" | The function correctly sorts the blocks in ascending order. |
| | I4 | Test with a list containing a single block. | List = ["Armour Cube"]<br>Sort Order = "A-Z" | The list remains unchanged since a single block does not need sorting. |
| | I5 | Test with a list containing blocks sorted in descending order (Z-A). | List = ["Steel Plate", "Armour Cube"]<br>Sort Order = "A-Z" | The function correctly sorts the blocks in ascending order. |

| | Test | Description | Data | Expected Outcome |
|---|---|---|---|---|
| Search - Binary | B1 | Test with an empty list. | List = []<br>Search Term = "Armour Cube" | The function returns no results or an appropriate message indicating no blocks are available. |
| | B2 | Test with a list containing blocks and an exact match search term. | List = ["Armour Cube", "Steel Plate"]<br>Search Term = "Armour Cube" | The function returns the block(s) whose name exactly matches the search term. |
| | B3 | Test with a case-insensitive search term. | List = ["Armour Cube"]<br>Search Term = "armour cube" | The function correctly matches blocks regardless of case (upper/lower). |
| | B4 | Test with a search term that does not match any block name. | List = ["Armour Cube"]<br>Search Term = "Iron Plate" | The function returns no results or an appropriate message indicating no matching blocks. |

# References

Cormen, T. H., Leiserson, C. E., Stein, C., & Rivest, R. L.(n.d.) *Introduction to Algorithms, 4th Edition*. Available from: https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf [Accessed 13 December 2024].

Keen Software House (n.d.) *Space Engineers Official Website*. Available from: https://www.spaceengineersgame.com/ [Accessed 13 December 2024].

Knuth, D. (2011) The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. [Extended ed.]. Vol. Part 1. Hoboken: Pearson Education, Limited.