

# Tacita's JavaCraft - Provisional Report (Group 18)

---

## Table of Contents

1. Tacita's JavaCraft - Provisional Report (Group 18)
  1. Table of Contents
  2. Group Details
    1. Students
  3. JavaCraft's Workflow
    1. Class JavaCraft
  4. Functionality Exploration
  5. Finite State Automata (FSA) Design
    1. Secret door logic (boolean secretDoorUnlocked)
  6. Git Collaboration & Version Control
    1. Overview
    2. Who did what?
  7. Appendix
    1. void clearScreen()
    2. void craftIronIngot()
    3. void craftItem(int recipe)
    4. void craftStick()
    5. void craftWoodenPlanks()
    6. void displayCraftingRecipes()
    7. void displayInventory()
    8. void fillInventory()
    9. void generateWorld()
    10. char getBlockChar(int blockType)
    11. String getBlockName(int blockType)
    12. String getBlockSymbol(int blockType)
    13. String getCraftedItemName(int craftedItem)
    14. void loadGame(String fileName)
    15. void lookAround()
    16. void placeBlock(int blockType)
    17. Additional documentation
  8. References

# Group Details

Group Name	Tacita
Group Number	18
TA	TA assigned to Group 18

## Students

Student Name	Student ID
Leopold Meinel	i6352276
Anton Haarmann	i6367288
Sian Lodde	i6343174
Tristan Dormans	i6343359

# JavaCraft's Workflow

## Class JavaCraft

### Pseudocode

BEGIN

Define global constants/variables and assign values to some;

Initialize game by assigning some global variables;

Generate world with different blocks by using randomness;

PRINT INFO `instructions`;

PRINT INFO "Start the game? (Y/N): ";

IF `

Set `

Set `

Set `

Set `

WHILE true

PRINT INFO `initial UI containing legend, world, inventory`;

PRINT INFO "Enter your action: 'WASD': Move, 'M': Mine, 'P': Place, 'C': Craft, 'I': Interact, 'Save': Save, 'Load': Load, 'Exit': Quit, 'Unlock': Unlock Secret Door\n" (colored in green);

IF `

IF `

Set `

Move player;

ELSE IF `

IF `

Set `

Mine block;

ELSE IF `

PRINT INFO `players inventory`;

PRINT INFO "Enter the block type to place: ";

Place block `

ELSE IF `

PRINT INFO `crafting recipes`;

PRINT INFO "Enter the recipe number to craft: ";

Craft item `

ELSE IF `

Interact with world;

ELSE IF `

PRINT INFO "Enter the file name to save the game state: ";

Save game as `

ELSE IF `

PRINT INFO "Enter the file name to load the game state: ";

Load game from `

ELSE IF `

```

        PRINT INFO "Exiting the game. Goodbye!\n";
        Exit game;
    ELSE IF `<String> READ user input` == "look" (caseless check)
        Print all blocks surrounding player;
    ELSE IF `<String> READ user input` == "unlock" (caseless check)
        Set `<boolean> unlockMode` = true;
    ELSE IF `<String> READ user input` == "getflag" (caseless check)
        TRY TO
            Set up connection to a server;
            PRINT INFO " " + `<String> get country from server via a
POST request`;
            PRINT INFO " " + `<String> get quote from server via a POST
request`;
        ON EXCEPTION
            PRINT ERROR containing `stacktrace`;
            PRINT ERROR "Error connecting to the server";
            Wait on player to press ENTER;
    ELSE IF `<String> READ user input` == "open" (caseless check)
        IF `<boolean> unlockMode` == true AND `<boolean>
craftingCommandEntered` == true AND `<boolean> miningCommandEntered` ==
true AND `<boolean> movementCommandEntered` == true
            Set `<boolean> secretDoorUnlocked` = true;
            Reset world to an empty world;
            PRINT INFO "Secret door unlocked!\n";
            Wait on player to press ENTER;
        ELSE
            PRINT WARNING "Invalid passkey. Try again!\n";
            Set `<boolean> unlockMode` = false;
            Set `<boolean> craftingCommandEntered` = false;
            Set `<boolean> miningCommandEntered` = false;
            Set `<boolean> movementCommandEntered` = false;
    ELSE
        PRINT WARNING "Invalid input. Please try again." (colored in
yellow);
    IF `<boolean> unlockMode` == true
        IF `<String> READ user input` == "c" (caseless check)
            Set `<boolean> craftingCommandEntered` = true;
        IF `<String> READ user input` == "m" (caseless check)
            Set `<boolean> miningCommandEntered` = true;
    IF `<boolean> secretDoorUnlocked` == true
        PRINT INFO `description of current state`;
        Set `<boolean> inSecretArea` = true;
        Reset world to an empty world;
        Set `<boolean> secretDoorUnlocked` = false;
        Fill `<Integer list> inventory` with all available blockTypes;
        Wait on player to press ENTER;
    ELSE
        Exit game;

END

```



## Functionality Exploration

See [Appendix](#) for documentation of all functions and flowcharts and pseudocodes of 16 functions.

# Finite State Automata (FSA) Design

## Secret door logic (boolean secretDoorUnlocked)

### General description

The secret door logic is triggered when `<boolean> secretDoorUnlocked` is true and will replace the map with an empty map containing a dutch flag. It will also replace the green player symbol with a blue one.

The `<boolean> secretDoorUnlocked` is true if the player supplies the following input in order:

1. `y` (caseless check)
2. Nothing OR anything other than `exit` (caseless check)
3. `unlock` (caseless check)
4. Nothing OR anything other than `exit` (caseless check)
5. Mandatory `a`, `c` AND `m` plus optional `y` AND/OR `unlock` in any order (caseless check, repetition is possible)
6. Nothing OR anything other than `exit` (caseless check)
7. `open` (caseless check)

After point 7, the `<boolean> secretDoorUnlocked` is true and the secret door logic triggers.

## Automaton

$$D=(Q, \Sigma, \delta, q_0, F)$$

$a=w, up, s, down, a, left, d, right$

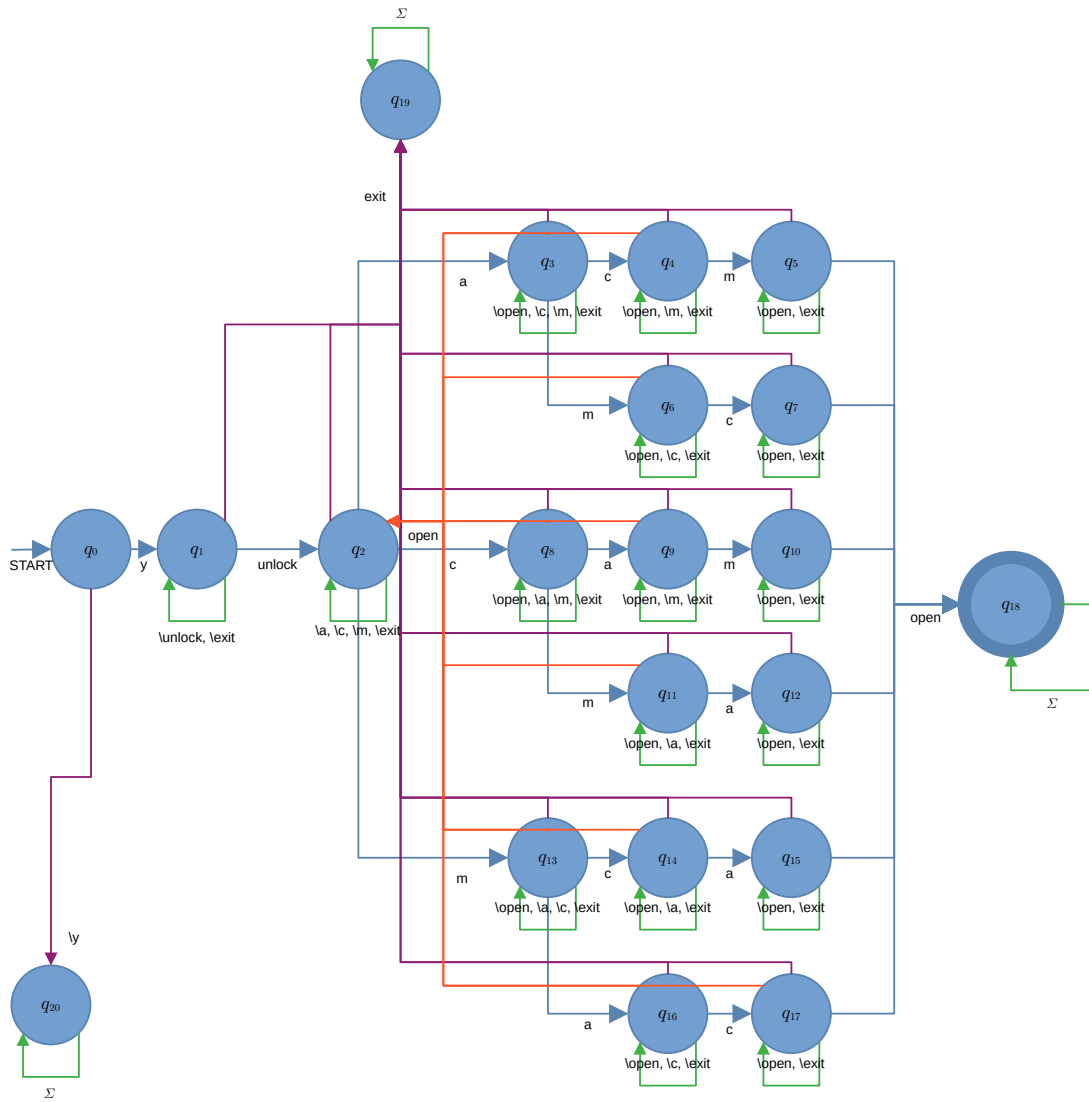
$\Sigma=\{y, unlock, a, c, m, open, exit\}$  (caseless check)

$\delta$ : Transition Function

$L(D)=\{y, unlock, \{\text{mandatory } a, c, m \text{ and optional } y, \text{unlock in any order; repetition is possible}\}, open\}$

$Q=\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}, q_{17}, q_{18}, q_{19}, q_{20}\}$

$F=\{q_{18}\}$





[illegible]

# Git Collaboration & Version Control

## Overview

### UM Gitlab Repository, Branch Group 18

#### Changes & Conflicts

Merge conflicts were handled efficiently and quickly. As a team we all had our experiences with these conflicts, one example was that a local repository was a few key commits behind. This was solved by choosing what parts of the code to keep, and what parts of the code needed to be replaced by the newer version on the repository.

Some other issue we faced was not being able to merge in the first place, which was inevitably resolved by re-cloning the repository and pasting in our modified files, which we wanted to replace older files on the remote repository.

#### Git usage

We used Gitlab as our main collaboration method. By splitting up the tasks in a fair manner we divided the workload to be more efficient. Through Gitlab we kept each other up to date by making commits after every completed task. That way everybody knew in what state the project was and how much still needed to be done. We also made sure to document our commits well, in an effort to better our understanding of the changes made. Each one of us made multiple commits and used Gitlab extensively. This in return improved our team performance and also kept each other motivated to work on the project.

## Who did what?

Task	Who worked on the task	Participation in percentage
Setting up the Github repository	Sian	100%
Creating the initial pseudocode and flowcharts	Leopold Anton Tristan Sian	Even across all participants
Setting up the Gitlab repository	Leopold	100%
Creating the documentation for the JavaCraft code	Leopold Anton Tristan Sian	Even across all participants
Finding repetition in the code	Sian	100%
Creating the flowchart and pseudocode for the global game	Tristan	100%
Creating FSA for automaton	Leopold Tristan	90% 10%
Creating the table for automaton	Leopold	100%
Writing the description of the automaton	Leopold	100%
Converting ODF Flowcharts to .graphml	Tristan	100%
Deciding on the uniform format for the flowcharts	Leopold Anton Tristan Sian	Even across all participants
Deciding on the uniform format for the pseudocode	Leopold Anton Tristan Sian	70% 10% 10% 10%
Converting flowcharts to uniform format	Sian Tristan Anton	80% 10% 10%
Converting pseudocode to uniform format	Leopold	100%
Updating the documentation	Leopold	100%
Cleaning up the repository folders	Sian	100%
Exporting flowcharts to SVG format	Sian	100%

Task	Who worked on the task	Participation in percentage
Implenting two blocks and two crafting items to the game	Anton	100%
Updating the functions involved with new blocks and crafting items	Anton	100%
Creating the provisional report document	Leo	70%
	Tristan	10%
	Anton	10%
	Sian	10%
Merging the flowchart images with the report document int one PDF	Sian	100%
Setting repository naming of folders to industry standard	Leopold	100%

# Appendix

## void clearScreen()

### Documentation

#### clearScreen

```
private static void clearScreen()
```

Clears the screen.

This method clears the screen and uses different logic depending on the OS.

#### Catched Exceptions:

- On IOException: Prints stacktrace when I/O exception of some sort has occurred.
- On InterruptedException: Prints stacktrace when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity.

### Java

```
private static void clearScreen() {
    try {
        if (System.getProperty("os.name").contains("Windows")) {
            new ProcessBuilder("cmd", "/c",
"cls").inheritIO().start().waitFor();
        } else {
            System.out.print("\033[H\033[2J");
            System.out.flush();
        }
    } catch (IOException | InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

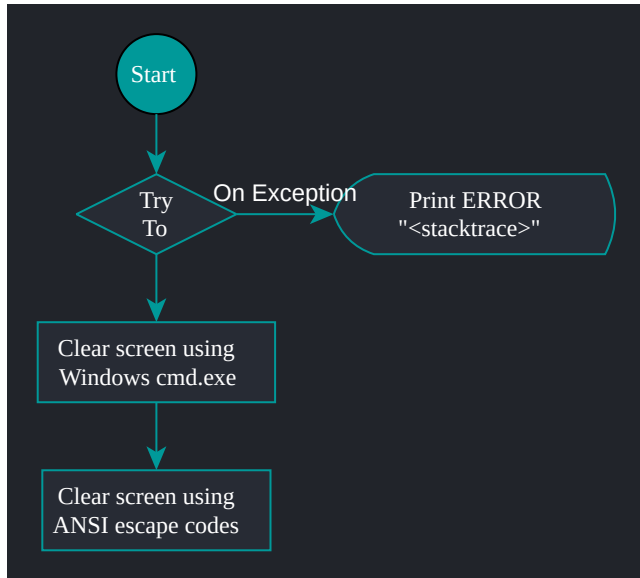
### Pseudocode

```
BEGIN

TRY TO
    IF current operating system matches Windows
        Clear screen using Windows cmd.exe by calling "/c cls";
        Wait on process to finish;
    ELSE
        Clear screen using ANSI code;
ON EXCEPTION
    PRINT ERROR containing `stacktrace`;

END
```

## Flowchart



void craftIronIngot()

## Documentation

### craftIronIngot

```
public static void craftIronIngot()
```

Crafts CRAFTED\_IRON\_INGOT.

This method crafts CRAFTED\_IRON\_INGOT from 3 IRON\_ORE that is taken from the players inventory.

Prints message if the player doesn't have the correct items in his inventory.

## Java

```
public static void craftIronIngot() {
    if (inventoryContains(IRON_ORE, 3)) {
        removeItemsFromInventory(IRON_ORE, 3);
        addCraftedItem(CRAFTED_IRON_INGOT);
        System.out.println("Crafted Iron Ingot.");
    } else {
        System.out.println("Insufficient resources to craft Iron Ingot.");
    }
}
```

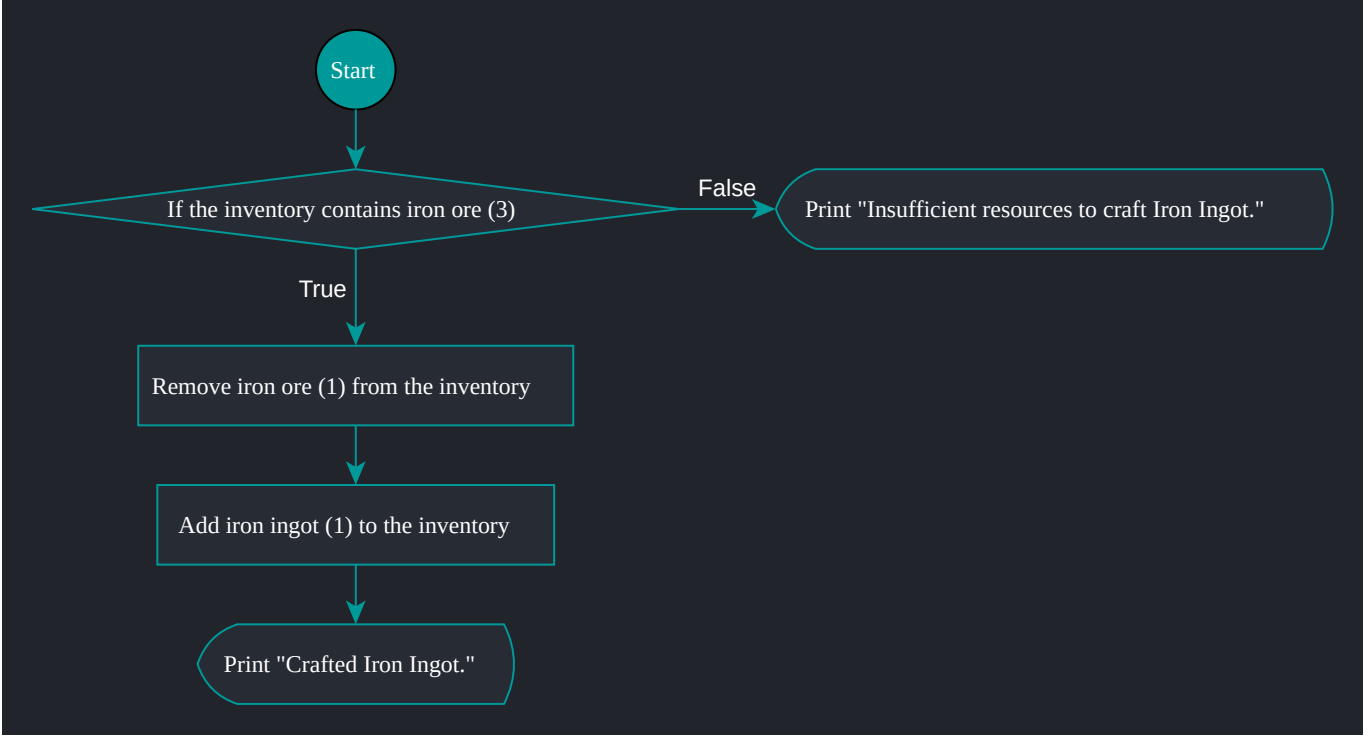
## Pseudocode

```
BEGIN

IF `<list> inventory` contains at least 3 iron ore
    Remove 3 iron ore from `<list> inventory`
    Add the crafted item 1 iron ingot to `<list> inventory`
    PRINT INFO "Crafted Iron Ingot.\n";
ELSE
    PRINT WARNING "Insufficient resources to craft Iron Ingot.\n";

END
```

Flowchart





void craftItem(int recipe)

## Documentation

### craftItem

```
public static void craftItem(int recipe)
```

Crafts an item.

This method crafts an item from a recipe.

Prints message if invalid recipe was supplied.

#### Parameters:

`recipe` - The recipe used to craft the item

## Java

```
public static void craftItem(int recipe) {
    switch (recipe) {
        case 1:
            craftWoodenPlanks();
            break;
        case 2:
            craftStick();
            break;
        case 3:
            craftIronIngot();
            break;
        case 4:
            craftStonePickaxe();
            break;
        case 5:
            craftIronPickaxe();
            break;
        default:
            System.out.println("Invalid recipe number.");
    }
    waitForEnter();
}
```

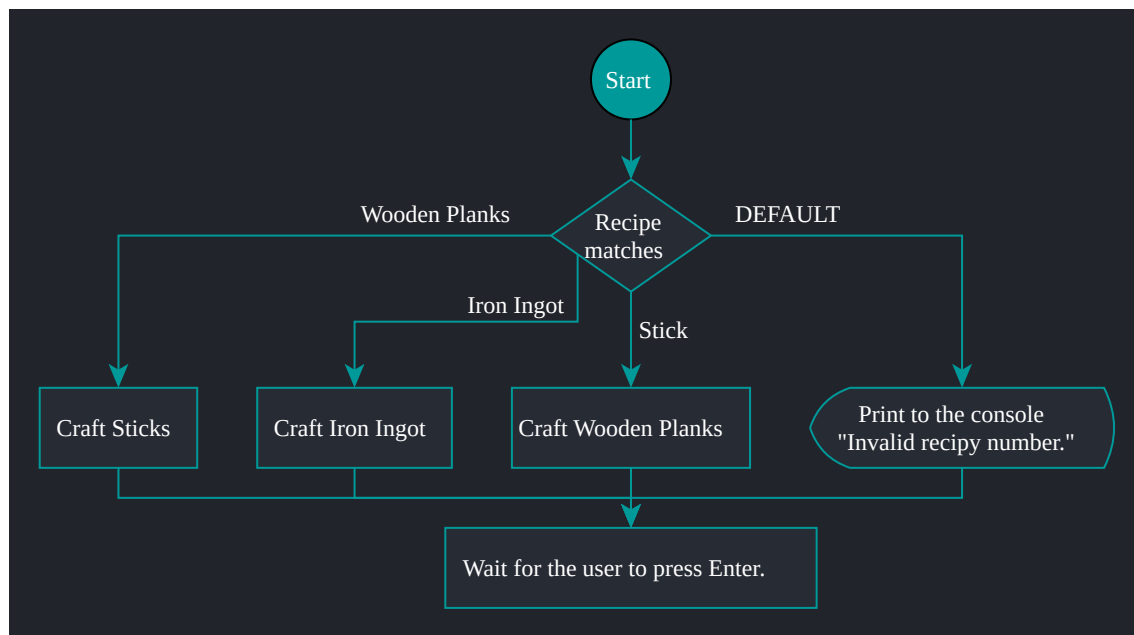
## Pseudocode

```
BEGIN

IF `<Integer> recipe` == 1
    Craft wooden planks;
ELSE IF `<Integer> recipe` == 2
    Craft stick;
ELSE IF `<Integer> recipe` == 3
    Craft iron ingot;
ELSE IF `<Integer> recipe` == 4
    Craft stone pickaxe;
ELSE IF `<Integer> recipe` == 5
    Craft iron pickaxe;
ELSE
    PRINT WARNING "Invalid recipe number.\n";
Wait on player to press ENTER;

END
```

## Flowchart



void craftStick()

## Documentation

### craftStick

```
public static void craftStick()
```

Crafts CRAFTED\_STICK.

This method crafts CRAFTED\_STICK from 1 WOOD that is taken from the players inventory.

Prints message if the player doesn't have the correct items in his inventory.

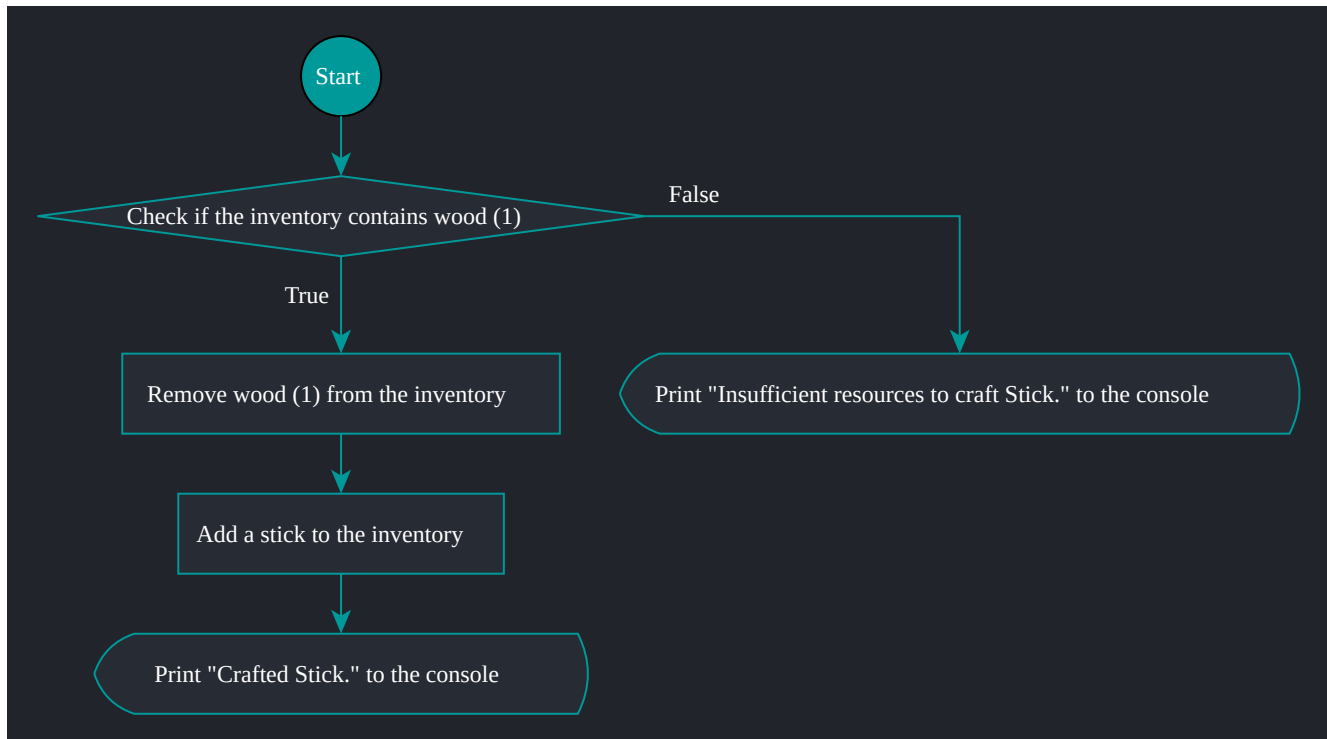
## Java

```
public static void craftStick() {  
    if (inventoryContains(WOOD)) {  
        removeItemsFromInventory(WOOD, 1);  
        addCraftedItem(CRAFTED_STICK);  
        System.out.println("Crafted Stick.");  
    } else {  
        System.out.println("Insufficient resources to craft Stick.");  
    }  
}
```

## Pseudocode

```
BEGIN  
  
IF `<list> inventory` contains wood  
    Remove 1 wood from `<list> inventory`;  
    Add the crafted item 1 stick to `<list> inventory`;  
    PRINT INFO "Crafted Stick.\n";  
ELSE  
    PRINT WARNING "Insufficient resources to craft Stick.\n";  
  
END
```

## Flowchart



void craftWoodenPlanks()

## Documentation

### craftWoodenPlanks

```
public static void craftWoodenPlanks()
```

Crafts CRAFTED\_WOODEN\_PLANKS.

This method crafts CRAFTED\_WOODEN\_PLANKS from 2 WOOD that are taken from the players inventory.

Prints message if the player doesn't have the correct items in his inventory.

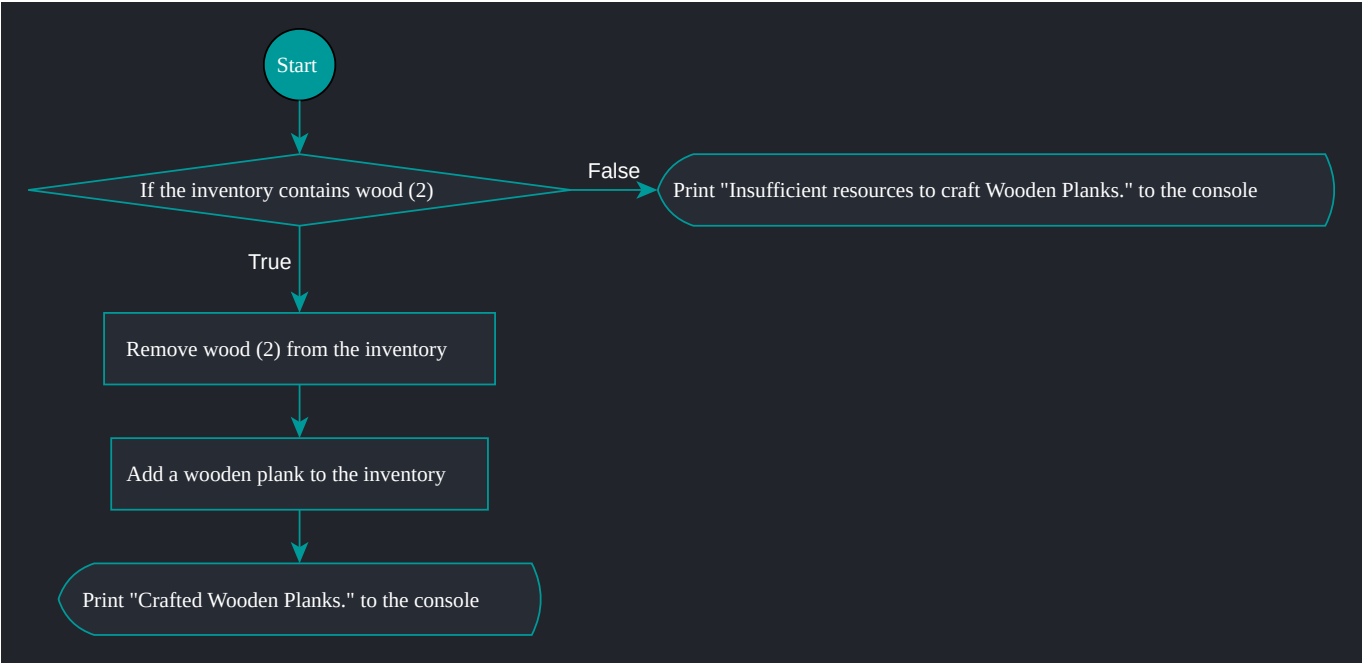
## Java

```
public static void craftWoodenPlanks() {  
    if (inventoryContains(WOOD, 2)) {  
        removeItemsFromInventory(WOOD, 2);  
        addCraftedItem(CRAFTED_WOODEN_PLANKS);  
        System.out.println("Crafted Wooden Planks.");  
    } else {  
        System.out.println("Insufficient resources to craft Wooden  
Planks.");  
    }  
}
```

## Pseudocode

```
BEGIN  
  
IF `<list> inventory` contains at least 2 wood  
    Remove 2 wood from `<list> inventory`;  
    Add the crafted item 1 wooden planks to `<list> inventory`;  
    PRINT INFO "Crafted Wooden Planks.\n";  
ELSE  
    PRINT WARNING "Insufficient resources to craft Wooden Planks.\n";  
  
END
```

Flowchart



void displayCraftingRecipes()

## Documentation

### displayCraftingRecipes

```
public static void displayCraftingRecipes()
```

Prints crafting recipes.

This method prints the available crafting recipes.

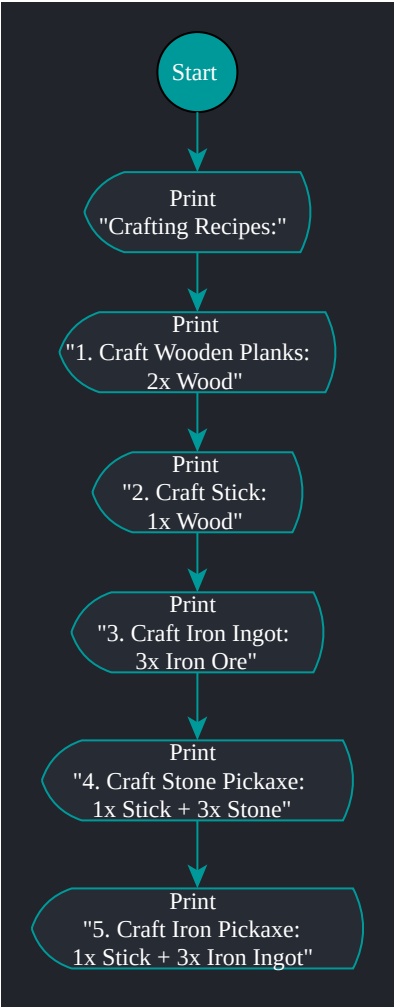
## Java

```
public static void displayCraftingRecipes() {  
    System.out.println("Crafting Recipes:");  
    System.out.println("1. Craft Wooden Planks: 2 Wood");  
    System.out.println("2. Craft Stick: 1 Wood");  
    System.out.println("3. Craft Iron Ingot: 3 Iron Ore");  
    System.out.println("4. Craft Stone Pickaxe: 1 Stick, 3 Stone");  
    System.out.println("5. Craft Iron Pickaxe: 1 Stick, 3 Iron Ingot");  
}
```

## Pseudocode

```
BEGIN  
  
PRINT INFO "Crafting Recipes:\n";  
PRINT INFO "1. Craft Wooden Planks: 2 Wood\n";  
PRINT INFO "2. Craft Stick: 1 Wood\n";  
PRINT INFO "3. Craft Iron Ingot: 3 Iron Ore\n";  
PRINT INFO "4. Craft Stone Pickaxe: 1 Stick, 3 Stone\n";  
PRINT INFO "5. Craft Iron Pickaxe: 1 Stick, 3 Iron Ingot\n";  
  
END
```

Flowchart





void displayInventory()

## Documentation

### displayInventory

```
public static void displayInventory()
```

Prints players inventory.

This method prints the players inventory including craftedItems.

## Java

```
public static void displayInventory() {
    System.out.println("Inventory:");
    if (inventory.isEmpty()) {
        System.out.println(ANSI_YELLOW + "Empty" + ANSI_RESET);
    } else {
        int[] blockCounts = new int[7];
        for (int i = 0; i < inventory.size(); i++) {
            int block = inventory.get(i);
            blockCounts[block]++;
        }
        for (int blockType = 1; blockType < blockCounts.length;
blockType++) {
            int occurrences = blockCounts[blockType];
            if (occurrences > 0) {
                System.out.println(getBlockName(blockType) + " - " +
occurrences);
            }
        }
        System.out.println("Crafted Items:");
        if (craftedItems == null || craftedItems.isEmpty()) {
            System.out.println(ANSI_YELLOW + "None" + ANSI_RESET);
        } else {
            for (int item : craftedItems) {
                System.out.print(
                    getCraftedItemColor(item) + getCraftedItemName(item) +
", " + ANSI_RESET);
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

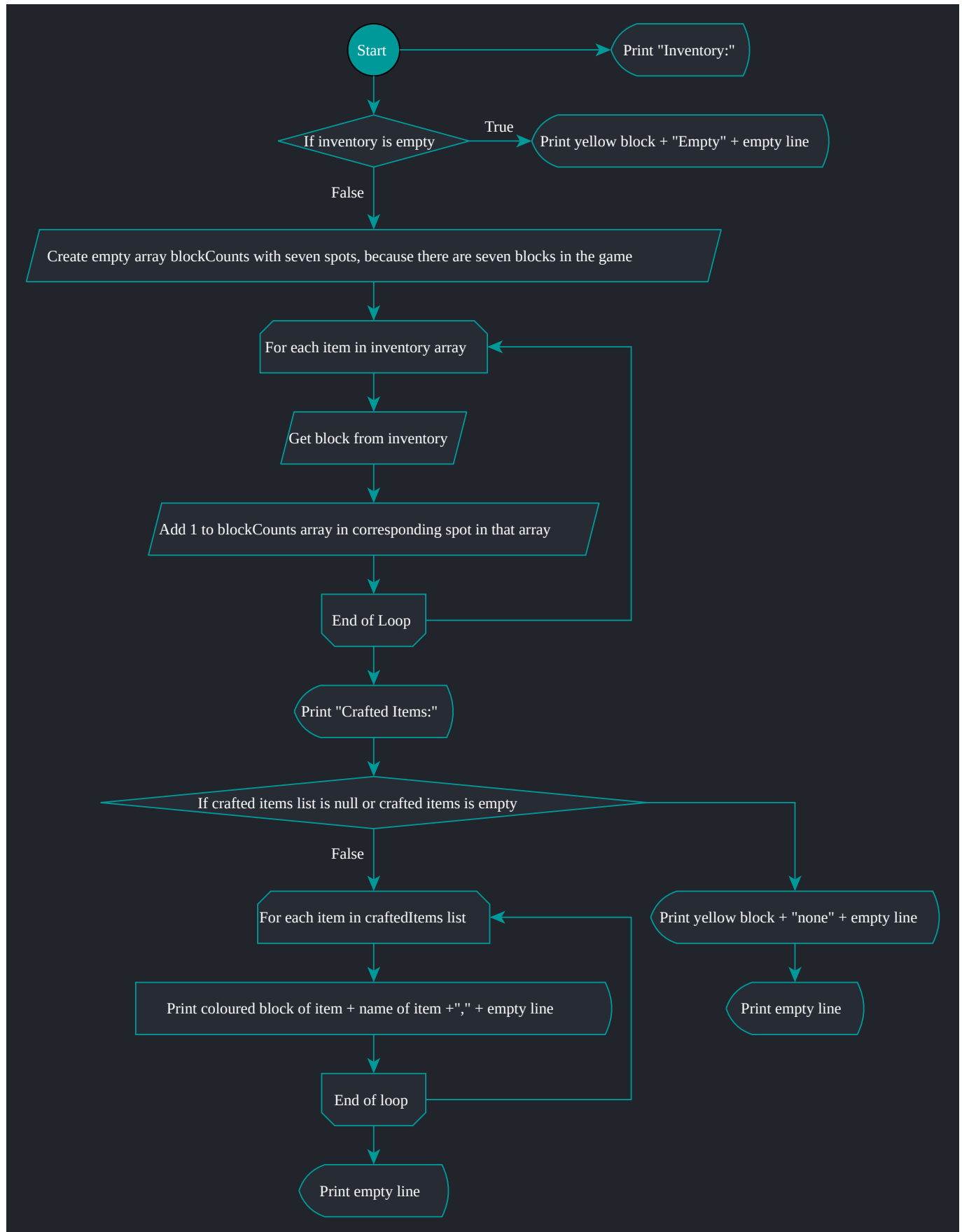
## Pseudocode

```
BEGIN

PRINT INFO "Inventory:\n";
IF `<Integer list> inventory` is empty
    PRINT INFO "Empty\n" (colored in yellow);
ELSE
    CREATE `<Integer array> blockCounts` of size 7;
    FOR EACH `<Integer> element` in `<Integer list> inventory`
        Assign `<Integer> block` = `<Integer> element`;
        Set `<Integer array> blockCounts @ index <Integer> block` += 1;
    FOR `<Integer> blockType` = 1; `<Integer> blockType` < `length of
<Integer array> blockCounts`
        Assign `<Integer> occurrences` = `<Integer array> blockCounts @
index <Integer> blockType`;
        IF `<Integer> occurrences` > 0
            PRINT INFO `<String> get block name matching <Integer>
blockType` + " - " + `<Integer> occurrences\n`;
            Set `<Integer> blockType` += 1;
PRINT INFO "Crafted Items:\n";
IF `<Integer list> craftedItems` is non-existent or empty
    PRINT INFO "None\n" (colored in yellow);
ELSE
    FOR EACH `<Integer> item` in `<Integer list> craftedItems`
        PRINT INFO `<String> get name matching <Integer> item` + ", "
(colored in `<String> get color matching <Integer> item`);
        PRINT INFO "\n";
PRINT INFO "\n";

END
```

## Flowchart



void fillInventory()

## Documentation

### fillInventory

```
private static void fillInventory()
```

Fills players inventory with all blocks.

This method fills the players inventory with all available blockTypes.

Part of secret door logic.

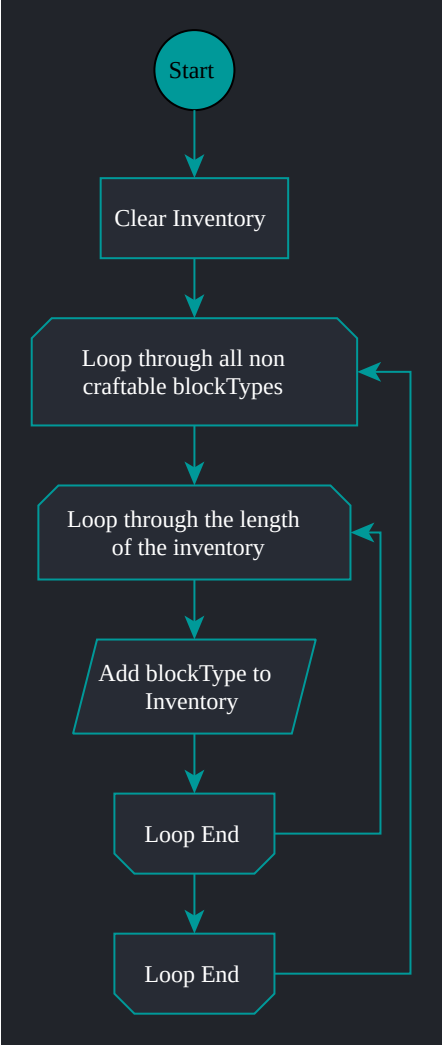
## Java

```
private static void fillInventory() {  
    inventory.clear();  
    for (int blockType = 1; blockType <= 6; blockType++) {  
        for (int i = 0; i < INVENTORY_SIZE; i++) {  
            inventory.add(blockType);  
        }  
    }  
}
```

## Pseudocode

```
BEGIN  
  
Clear `<Integer list> inventory`;  
FOR `<Integer> blockType` = 1; `<Integer> blockType` <= 6  
    FOR EACH `<Integer> element` in `<Integer list> inventory`  
        Set `<Integer> member` = `<Integer> blockType`;  
        Set `<Integer> blockType` += 1;  
  
END
```

Flowchart



## void generateWorld()

### Documentation

#### generateWorld

```
public static void generateWorld()
```

Generates the world.

This method uses randomness to generate a world out of different materials.

### Java

```
public static void generateWorld() {
    Random rand = new Random();
    for (int y = 0; y < worldHeight; y++) {
        for (int x = 0; x < worldWidth; x++) {
            int randValue = rand.nextInt(100);
            if (randValue < 17) {
                world[x][y] = WOOD;
            } else if (randValue < 30) {
                world[x][y] = LEAVES;
            } else if (randValue < 45) {
                world[x][y] = STONE;
            } else if (randValue < 57) {
                world[x][y] = COAL_ORE;
            } else if (randValue < 65) {
                world[x][y] = IRON_ORE;
            } else if (randValue < 70) {
                world[x][y] = EMERALD_ORE;
            } else {
                world[x][y] = AIR;
            }
        }
    }
}
```

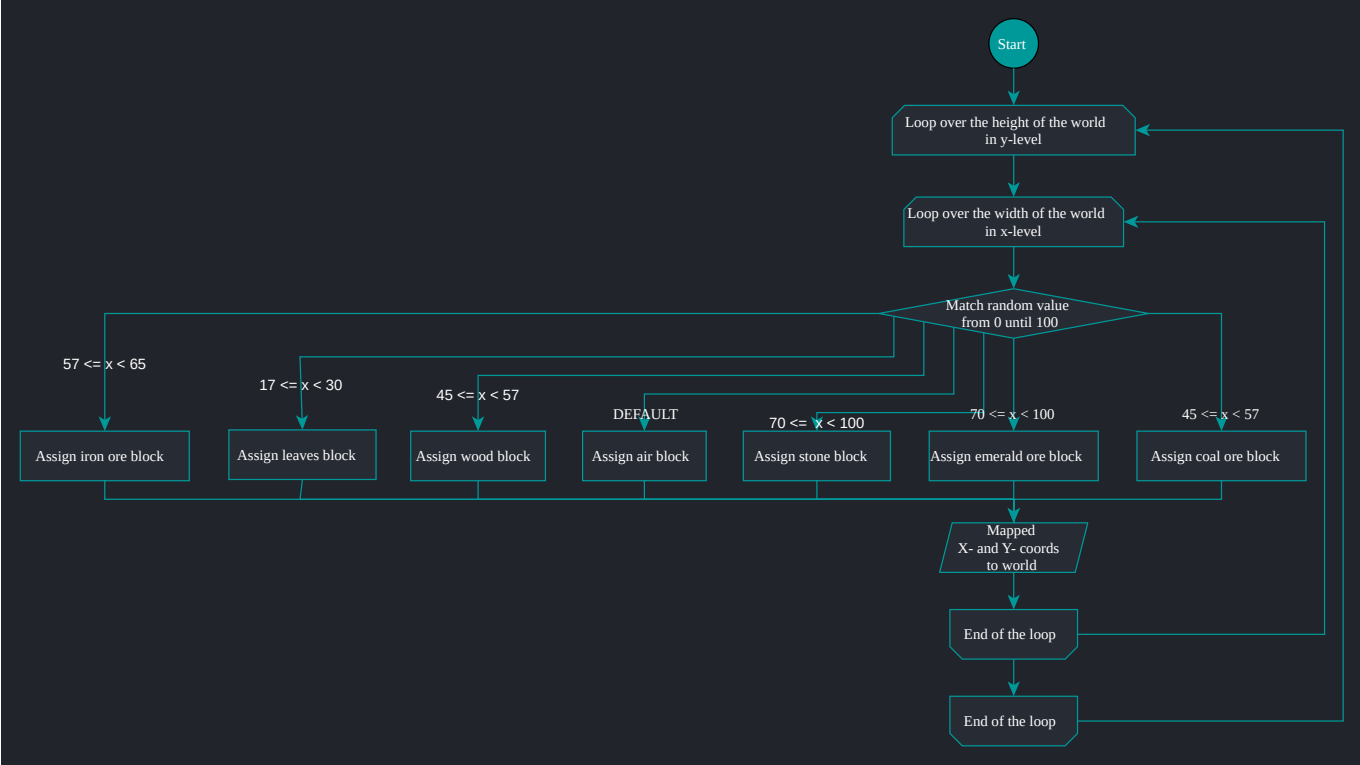
## Pseudocode

```
BEGIN

FOR `<Integer> y` = 0; `<Integer> y` < `<Integer> worldHeight`
  FOR `<Integer> x` = 0; `<Integer> x` < `<Integer> worldWidth`
    Assign `<Integer> randValue` = `random value between 0 and 99`;
    IF `<Integer> randValue` < 17
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> wood`;
    ELSE IF `<Integer> randValue` < 30
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> leaves`;
    ELSE IF `<Integer> randValue` < 45
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> stone`;
    ELSE IF `<Integer> randValue` < 57
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> coal ore`;
    ELSE IF `<Integer> randValue` < 65
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> iron ore`;
    ELSE IF `<Integer> randValue` < 70
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> emerald ore`;
    ELSE
      Set `<two dimensional Integer array> world @ indexes <Integer>
x, <Integer> y` = `<Integer> air`;
      Set `<Integer> x` += 1;
      Set `<Integer> y` += 1;

END
```

Flowchart





## char getBlockChar(int blockType)

### Documentation

#### getBlockChar

```
private static char getBlockChar(int blockType)
```

Returns the symbol for blockType.

This method returns the mapped char for blockType.

**Parameters:**

blockType - The type of block

**Returns:**

char The mapped symbol for blockType

### Java

```
private static char getBlockChar(int blockType) {
    switch (blockType) {
        case WOOD:
            return '\u2592';
        case LEAVES:
            return '\u00A7';
        case STONE:
            return '\u2593';
        case IRON_ORE:
            return '\u00B0';
        case COAL_ORE:
            return '\u2593';
        case EMERALD_ORE:
            return '\u00B0';
        default:
            return '-';
    }
}
```

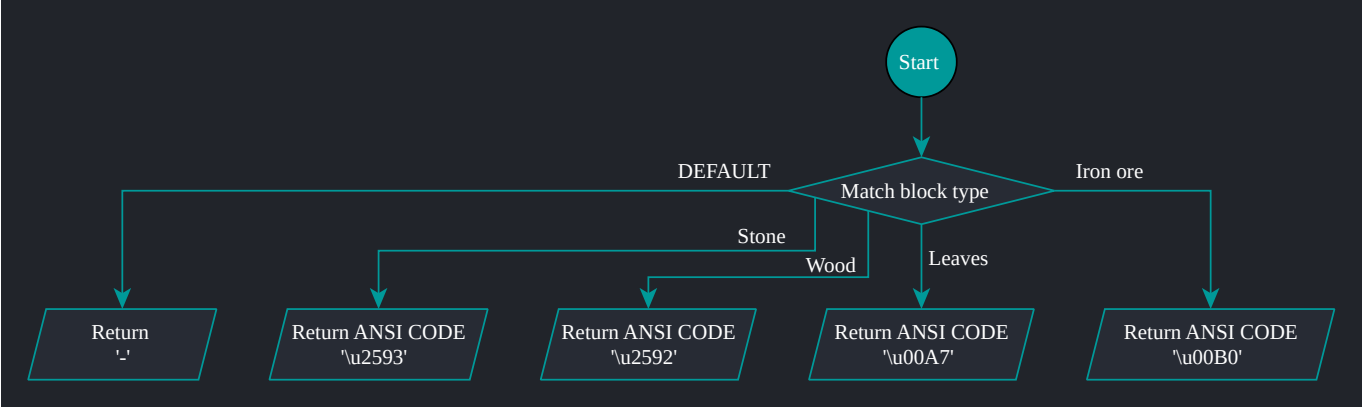
## Pseudocode

```
BEGIN

IF `<Integer> blockType` == `<Integer> wood`
    RETURN `<Character> medium shade`;
ELSE IF `<Integer> blockType` == `<Integer> leaves`
    RETURN `<Character> section sign`;
ELSE IF `<Integer> blockType` == `<Integer> stone`
    RETURN `<Character> dark shade`;
ELSE IF `<Integer> blockType` == `<Integer> iron ore`
    RETURN `<Character> degree sign`;
ELSE IF `<Integer> blockType` == `<Integer> coal ore`
    RETURN `<Character> dark shade`;
ELSE IF `<Integer> blockType` == `<Integer> emerald ore`
    RETURN `<Character> degree sign`;
ELSE
    RETURN `<Character> -`;

END
```

Flowchart



## String getBlockName(int blockType)

### Documentation

#### getBlockName

```
private static StringⒺ getBlockName(int blockType)
```

Returns human readable block name.

This method returns a human readable block name for blockType.

Defaults to "Unknown"

**Parameters:**

blockType - The type of block

**Returns:**

String The human readable block name.

### Java

```
private static String getBlockName(int blockType) {  
    switch (blockType) {  
        case AIR:  
            return "Empty Block";  
        case WOOD:  
            return "Wood";  
        case LEAVES:  
            return "Leaves";  
        case STONE:  
            return "Stone";  
        case IRON_ORE:  
            return "Iron Ore";  
        case COAL_ORE:  
            return "Coal Ore";  
        case EMERALD_ORE:  
            return "Emerald Ore";  
        default:  
            return "Unknown";  
    }  
}
```

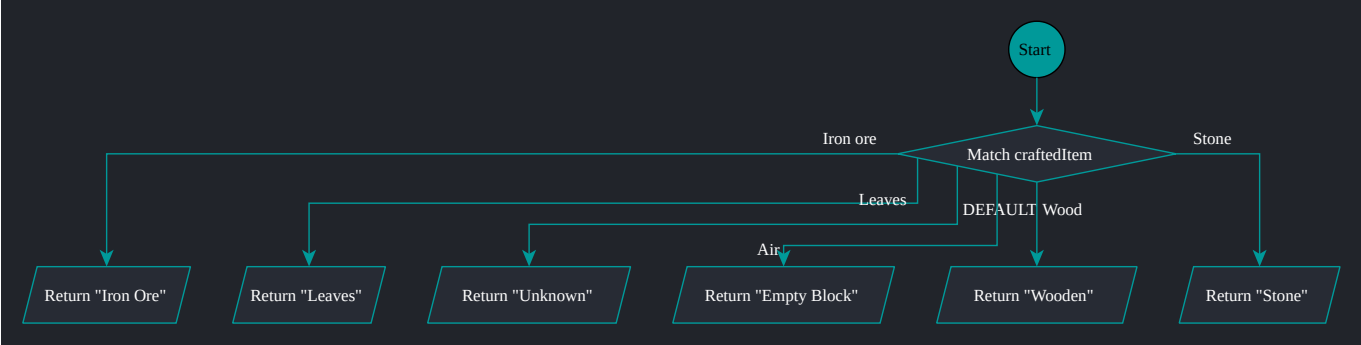
## Pseudocode

```
BEGIN

IF `<Integer> blockType` == `<Integer> air`
    RETURN "Empty Block";
ELSE IF `<Integer> blockType` == `<Integer> wood`
    RETURN "Wood";
ELSE IF `<Integer> blockType` == `<Integer> leaves`
    RETURN "Leaves";
ELSE IF `<Integer> blockType` == `<Integer> stone`
    RETURN "Stone";
ELSE IF `<Integer> blockType` == `<Integer> iron ore`
    RETURN "Iron Ore";
ELSE IF `<Integer> blockType` == `<Integer> coal ore`
    RETURN "Coal Ore";
ELSE IF `<Integer> blockType` == `<Integer> emerald ore`
    RETURN "Emerald Ore";
ELSE
    RETURN "Unknown";

END
```

Flowchart



## String getBlockSymbol(int blockType)

### Documentation

#### getBlockSymbol

```
private static StringⒺ getBlockSymbol(int blockType)
```

Returns the symbol and color for blockType.

This method returns the mapped char and blockColor for blockType.

#### Parameters:

blockType - The type of block

#### Returns:

String The mapped symbol and blockColor for blockType

### Java

```
private static String getBlockSymbol(int blockType) {
    String blockColor;
    switch (blockType) {
        case AIR:
            return ANSI_RESET + "- ";
        case WOOD:
            blockColor = ANSI_RED;
            break;
        case LEAVES:
            blockColor = ANSI_GREEN;
            break;
        case STONE:
            blockColor = ANSI_BLUE;
            break;
        case IRON_ORE:
            blockColor = ANSI_WHITE;
            break;
        case COAL_ORE:
            blockColor = ANSI_COAL_GRAY;
            break;
        case EMERALD_ORE:
            blockColor = ANSI_EMERALD_GREEN;
            break;
        default:
            blockColor = ANSI_RESET;
            break;
    }
    return blockColor + getBlockChar(blockType) + " ";
}
```

## Pseudocode

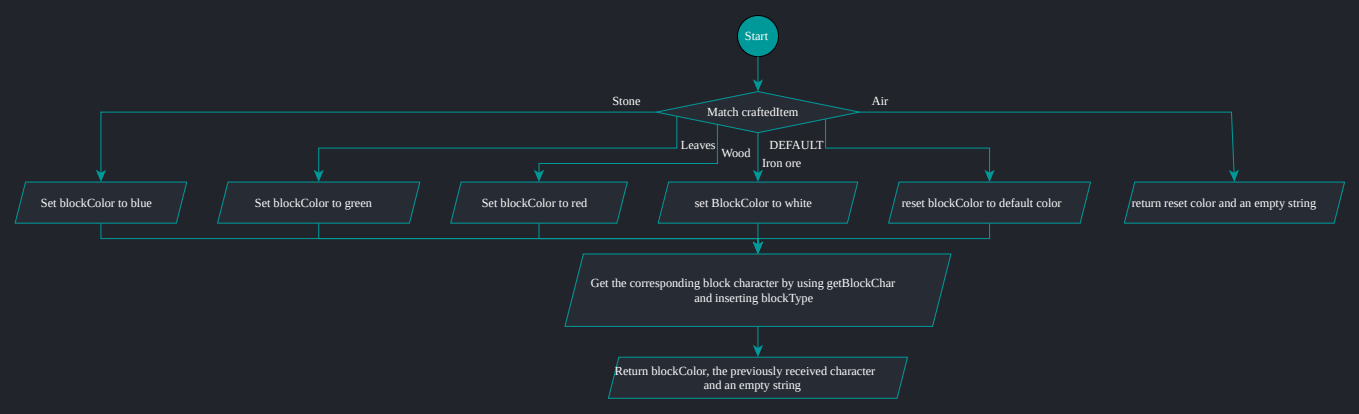
```
BEGIN

Define `<String> blockColor`;
IF `<Integer> blockType` == `<Integer> air`
    RETURN "Empty Block";
ELSE IF `<Integer> blockType` == `<Integer> wood`
    Set `<String> blockColor` = `(color red)`;
ELSE IF `<Integer> blockType` == `<Integer> leaves`
    Set `<String> blockColor` = `(color green)`;
ELSE IF `<Integer> blockType` == `<Integer> stone`
    Set `<String> blockColor` = `(color blue)`;
ELSE IF `<Integer> blockType` == `<Integer> iron ore`
    Set `<String> blockColor` = `(color white)`;
ELSE IF `<Integer> blockType` == `<Integer> coal ore`
    Set `<String> blockColor` = `(color coal gray)`;
ELSE IF `<Integer> blockType` == `<Integer> emerald ore`
    Set `<String> blockColor` = `(color emerald green)`;
ELSE
    Set `<String> blockColor` = `(reset color)`;
RETURN `<String> blockColor` + `<Character> get symbol matching blockType`
+ " ";

END
```



Flowchart



## String getCraftedItemName(int craftedItem)

### Documentation

#### getCraftedItemName

```
private static StringⒺ getCraftedItemName(int craftedItem)
```

Returns human readable item name.

This method returns a human readable item name for craftedItem.

**Parameters:**

craftedItem - The crafted item

**Returns:**

String The human readable name of craftedItem

### Java

```
private static String getCraftedItemName(int craftedItem) {  
    switch (craftedItem) {  
        case CRAFTED_WOODEN_PLANKS:  
            return "Wooden Planks";  
        case CRAFTED_STICK:  
            return "Stick";  
        case CRAFTED_IRON_INGOT:  
            return "Iron Ingot";  
        case CRAFTED_STONE_PICKAXE:  
            return "Stone Pickaxe";  
        case CRAFTED_IRON_PICKAXE:  
            return "Iron Pickaxe";  
        default:  
            return "Unknown";  
    }  
}
```

## Pseudocode

```
BEGIN

IF `<Integer> craftedItem` == `<Integer> wooden planks`
    RETURN "Wooden Planks";
ELSE IF `<Integer> blockType` == `<Integer> stick`
    RETURN "Stick";
ELSE IF `<Integer> blockType` == `<Integer> iron ingot`
    RETURN "Iron Ingot";
ELSE IF `<Integer> blockType` == `<Integer> stone pickaxe`
    RETURN "Stone Pickaxe";
ELSE IF `<Integer> blockType` == `<Integer> iron pickaxe`
    RETURN "Iron Pickaxe";
ELSE
    RETURN "Unknown";

END
```

## Flowchart



void loadGame(String fileName)

## Documentation

### loadGame

```
public static void loadGame(StringⒺ fileName)
```

Loads the game.

This method loads the game from a file.

#### Parameters:

fileName - The file name

#### Catched Exceptions:

- On IOException: Prints error with message when I/O exception of some sort has occurred.
- On ClassNotFoundException: Prints error with message when no definition for the class with the specified name could be found.

## Java

```
public static void loadGame(String fileName) {  
    // Implementation for loading the game state from a file goes here  
    try (ObjectInputStream inputStream = new ObjectInputStream(new  
        FileInputStream(fileName))) {  
        // Deserialize game state data from the file and load it into the  
        program  
        NEW_WORLD_WIDTH = inputStream.readInt();  
        NEW_WORLD_HEIGHT = inputStream.readInt();  
        world = (int[][]) inputStream.readObject();  
        playerX = inputStream.readInt();  
        playerY = inputStream.readInt();  
        inventory = (List<Integer>) inputStream.readObject();  
        craftedItems = (List<Integer>) inputStream.readObject();  
        unlockMode = inputStream.readBoolean();  
        System.out.println("Game state loaded from file: " + fileName);  
    } catch (IOException | ClassNotFoundException e) {  
        System.out.println("Error while loading the game state: " +  
            e.getMessage());  
    }  
    waitForEnter();  
}
```

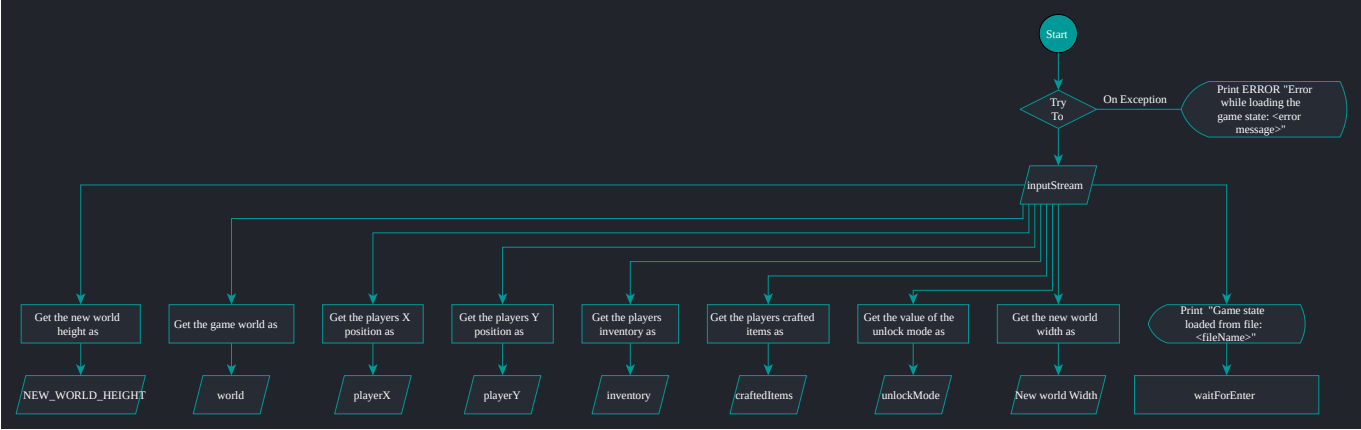
## Pseudocode

```
BEGIN

TRY TO
    Set `<stream> inputStream` = `<stream> of contents from file matching
    <String> fileName relative to current working directory`;
    Set `<Integer> NEW_WORLD_WIDTH` = `<Integer> get next line containing
    serialized <Integer> in <stream> inputStream`;
    Set `<Integer> NEW_WORLD_HEIGHT` = `<Integer> get next line containing
    serialized <Integer> in <stream> inputStream`;
    Set `<two dimensional Integer array> world` = `<two dimensional Integer
    array> get next line containing any serialized object in <stream>
    inputStream`;
    Set `<Integer> playerX` = `<Integer> get next line containing
    serialized <Integer> in <stream> inputStream`;
    Set `<Integer> playerY` = `<Integer> get next line containing
    serialized <Integer> in <stream> inputStream`;
    Set `<Integer list> inventory` = `<Integer list> get next line
    containing any serialized object in <stream> inputStream` and cast to
    <Integer list>;
    Set `<Integer list> craftedItems` = `<Integer list> get next line
    containing any serialized object in <stream> inputStream` and cast to
    <Integer list>;
    Set `<boolean> unlockMode` = `<boolean> get next line containing
    serialized <boolean> in <stream> inputStream`;
    PRINT INFO "Game state loaded from file: " + `<String> fileName` +
    "\n";
    Close `<stream> inputStream`;
ON EXCEPTION
    PRINT ERROR "Error while loading the game state: " + `errorMessage` +
    "\n";
    Close `<stream> inputStream`;
Wait on player to press ENTER;

END
```

Flowchart



void lookAround()

## Documentation

### lookAround

```
private static void lookAround()
```

Prints all blocks surrounding the player.

This method prints all blocks surrounding the player. This is meant to make the players life easier.

## Java

```
private static void lookAround() {
    System.out.println("You look around and see:");
    for (int y = Math.max(0, playerY - 1); y <= Math.min(playerY + 1,
worldHeight - 1); y++) {
        for (int x = Math.max(0, playerX - 1); x <= Math.min(playerX + 1,
worldWidth - 1); x++) {
            if (x == playerX && y == playerY) {
                System.out.print(ANSI_GREEN + "P" + ANSI_RESET);
            } else {
                System.out.print(getBlockSymbol(world[x][y]) + ANSI_RESET);
            }
        }
        System.out.println();
    }
    System.out.println();
    waitForEnter();
}
```

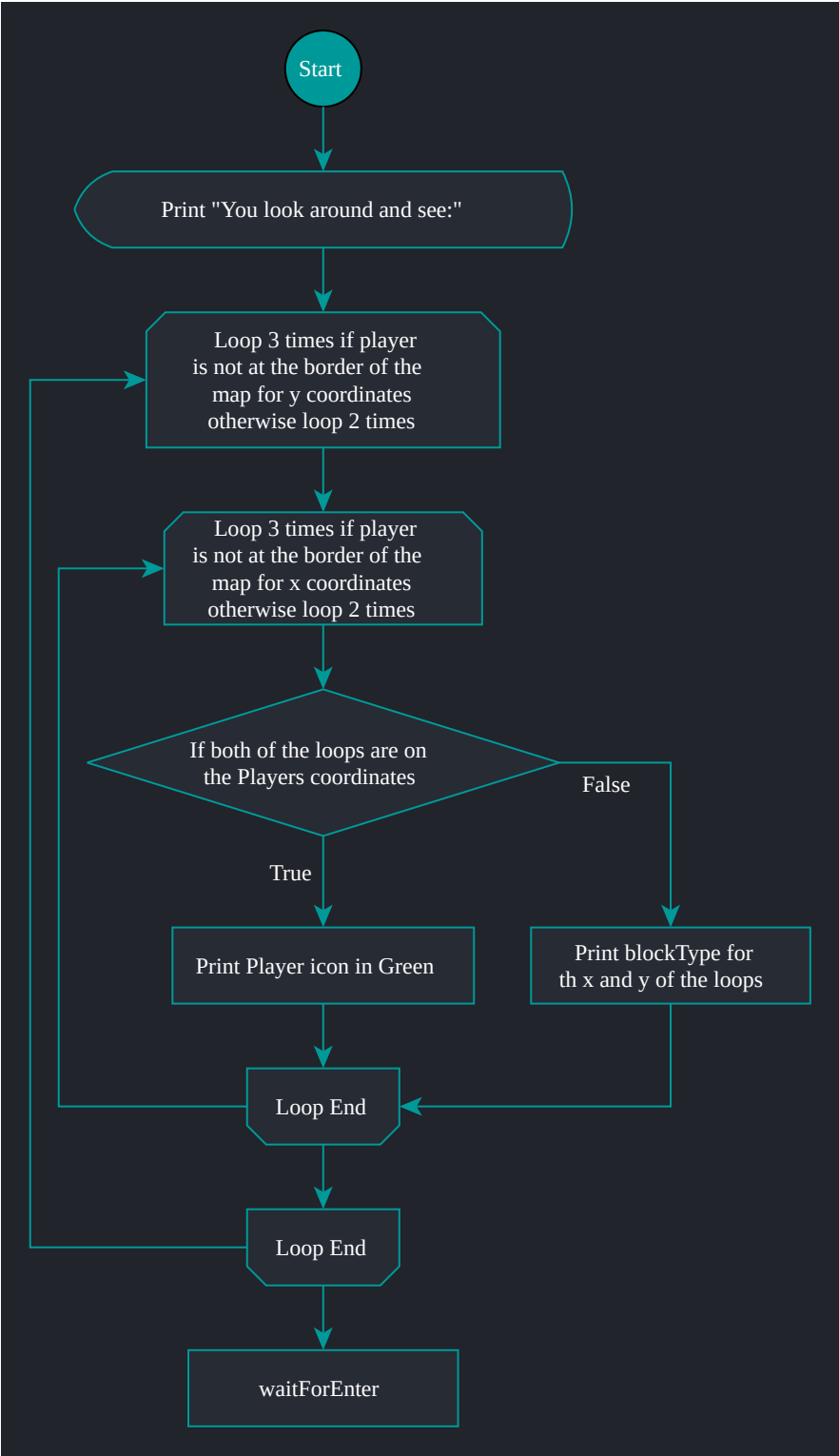
## Pseudocode

```
BEGIN

PRINT INFO "You look around and see:";
FOR `
```



Flowchart



void placeBlock(int blockType)

## Documentation

### placeBlock

```
public static void placeBlock(int blockType)
```

Places a block.

This method places a block that is of blockType 0 to 9 and removes it from the players inventory if the players inventory contains that block.

Parameters:

blockType - The type of block

## Java

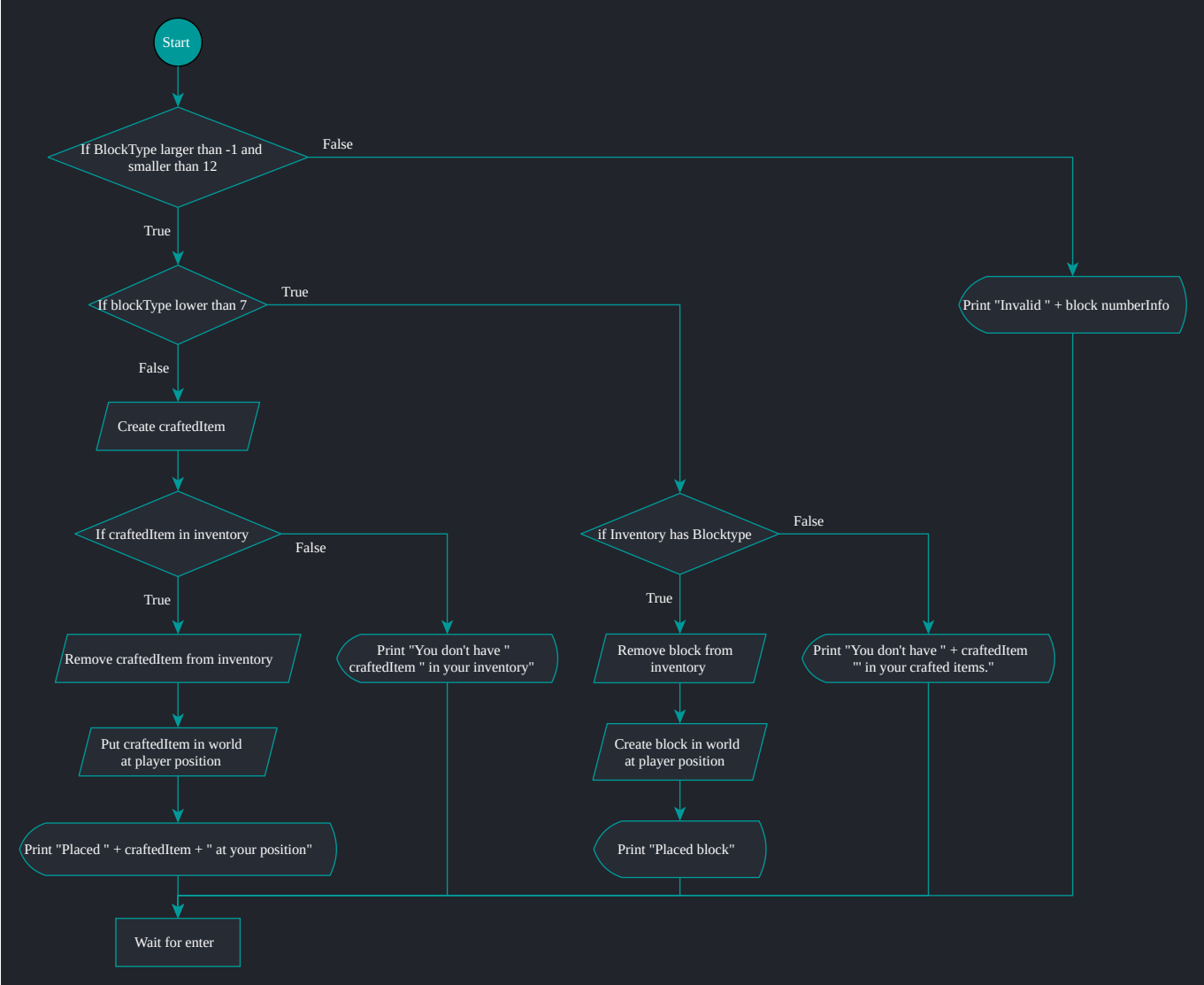
```
public static void placeBlock(int blockType) {
    if (blockType >= 0 && blockType <= 11) {
        if (blockType <= 6) {
            if (inventory.contains(blockType)) {
                inventory.remove(Integer.valueOf(blockType));
                world[playerX][playerY] = blockType;
                System.out.println("Placed " + getBlockName(blockType) + "
at your position.");
            } else {
                System.out.println(
                    "You don't have " + getBlockName(blockType) + " in
your inventory.");
            }
        } else {
            int craftedItem = getCraftedItemFromBlockType(blockType);
            if (craftedItems.contains(craftedItem)) {
                craftedItems.remove(Integer.valueOf(craftedItem));
                world[playerX][playerY] = blockType;
                System.out.println(
                    "Placed " + getCraftedItemName(craftedItem) + " at
your position.");
            } else {
                System.out.println("You don't have " +
getCraftedItemName(craftedItem)
                    + " in your crafted items.");
            }
        }
    } else {
        System.out.println("Invalid block number. Please enter a valid
block number.");
        System.out.println(BLOCK_NUMBERS_INFO);
    }
    waitForEnter();
}
```

## Pseudocode

```
BEGIN

IF `
```

Flowchart



# Additional documentation

<b>addCraftedItem</b>
<pre>public static void addCraftedItem(int craftedItem)</pre> <p>Adds a crafted item to craftedItems.</p> <p>This method adds a crafted item to craftedItems that are part of the players inventory.</p> <p><b>Parameters:</b> <b>craftedItem</b> - The crafted item</p>
<b>craftedItemsContains</b>
<pre>public static boolean craftedItemsContains(int craftedItem)</pre> <p>Queries craftedItems for an item.</p> <p>This method queries the players crafted item inventory for an item.</p> <p><b>Parameters:</b> <b>craftedItem</b> - The item to query the crafted item inventory for</p> <p><b>Returns:</b> boolean true if craftedItems contains item, false in any other case</p>
<b>craftedItemsContains</b>
<pre>public static boolean craftedItemsContains(int craftedItem,  int count)</pre> <p>Queries craftedItems for if it has enough of an crafted item.</p> <p>This method queries the players craftedItems for an crafted item and if it contains at least as much as the supplied count.</p> <p><b>Parameters:</b> <b>craftedItem</b> - The crafted item to query the crafted items inventory for <b>count</b> - The count that the crafted items inventory should contain of the item</p> <p><b>Returns:</b> boolean true if craftedItems contains crafted item at least as many times as the supplied count, false in any other case</p>
<b>craftIronPickaxe</b>
<pre>public static void craftIronPickaxe()</pre> <p>Crafts CRAFTED_IRON_PICKAXE.</p> <p>This method crafts CRAFTED_IRON_PICKAXE from 1 Stick and 3 Iron Ingots that are taken form the players inventory.</p> <p>Prints message if the player doesn't have the correct items in his inventory.</p>
<b>craftStonePickaxe</b>
<pre>public static void craftStonePickaxe()</pre> <p>Crafts CRAFTED_STONE_PICKAXE.</p> <p>This method crafts CRAFTED_STONE_PICKAXE from 1 Stick and 3 Stone that are taken form the players inventory.</p> <p>Prints message if the player doesn't have the correct items in his inventory.</p>
<b>displayLegend</b>
<pre>public static void displayLegend()</pre> <p>Prints a legend.</p> <p>This method prints a legend of items on the map.</p>
<b>displayWorld</b>
<pre>public static void displayWorld()</pre> <p>Prints the world as ASCII text.</p> <p>This method is responsible for displaying the world.</p> <p>Part of secret door logic.</p>
<b>generateEmptyWorld</b>
<pre>private static void generateEmptyWorld()</pre> <p>Generates an empty world.</p> <p>This method generates an empty world which only contains the dutch flag.</p> <p>Part of secret door logic.</p>

### getBlockColor

```
private static StringⒶ getBlockColor(int blockType)
```

Returns block color.

This method returns the blocks color.

Defaults to empty String

**Parameters:**

**blockType** - The type of block

**Returns:**

String The human readable name of craftedItem

### getBlockTypeFromCraftedItem

```
private static int getBlockTypeFromCraftedItem(int craftedItem)
```

Returns the block type of craftedItem.

This method returns the block type of craftedItem.

Defaults to -1.

**Parameters:**

**craftedItem** - The crafted item

**Returns:**

int The block type of craftedItem

### getCountryAndQuoteFromServer

```
public static void getCountryAndQuoteFromServer()
```

Gets country and quote from server.

This method gets country and quote from server via a POST request.

**Catched Exceptions:**

- On Exception: Prints an error for any encountered exception.

### getCraftedItemColor

```
private static StringⒶ getCraftedItemColor(int craftedItem)
```

Returns item color.

This method returns the items color.

Defaults to empty String

**Parameters:**

**craftedItem** - The crafted item

**Returns:**

String The human readable name of craftedItem

### getCraftedItemFromBlockType

```
private static int getCraftedItemFromBlockType(int blockType)
```

Returns the crafted item of blockType.

This method returns the crafted item of blockType.

Defaults to -1.

**Parameters:**

**blockType** - The type of block

**Returns:**

int The crafted item of blockType

### getRequiredItemForMining

```
public static int getRequiredItemForMining(int blockType)
```

Returns the crafted item that is required to mine blockType.

This method returns the crafted item that is required to mine blockType.

Defaults -1.

**Parameters:**

**blockType** - The type of block

**Returns:**

int The crafted Item required to mine blockType

### initGame

```
public static void initGame(int worldWidth,  
                             int worldHeight)
```

Initializes the game.

This method sets worldWidth, JworldHeight, world, playerX, playerY and initializes inventory.

**Parameters:**

**worldWidth** - The width of world in blocks

**worldHeight** - The height of world in blocks

### interactWithWorld

```
public static void interactWithWorld()
```

Handles interaction with the game world.

This method handles interaction with the game world and prints messages for blocks that the player can interact with. It also adds certain blocks to the players inventory if he interacts with them.

**inventoryContains**

```
public static boolean inventoryContains(int item)
```

Queries inventory for an item.

This method queries the players inventory for an item.

**Parameters:**

**item** - The item to query the inventory for

**Returns:**

boolean true if inventory contains item, false in any other case

**inventoryContains**

```
public static boolean inventoryContains(int item,
                                       int count)
```

Queries inventory for if it has enough of an item.

This method queries the players inventory for an item and if it contains at least as much as the supplied count.

**Parameters:**

**item** - The item to query the inventory for

**count** - The count that the inventory should contain of the item

**Returns:**

boolean true if inventory contains item at least as many times as the supplied count, false in any other case

**main**

```
public static void main(StringⒺ[] args)
```

Main method.

This method is called upon execution of the game.

**Parameters:**

**args** - The supplied commandline arguments

**mineBlock**

```
public static void mineBlock()
```

Mines a block.

This method mines a block and adds it to the players inventory if it is not AIR.

**movePlayer**

```
public static void movePlayer(StringⒺ direction)
```

Moves the player

This method moves the player UP/DOWN/LEFT/RIGHT depending on the supplied direction.

**Parameters:**

**direction** - The direction the player should be moved towards.

**removeItemFromCraftedItems**

```
public static void removeItemFromCraftedItems(int craftedItem,
                                              int count)
```

Removes a count of item from craftedItem.

This method removes a count of an item from the players crafted items inventory.

**Parameters:**

**craftedItem** - The item to remove from the crafted items inventory

**count** - The count that should be removed from the crafted items inventory

**removeItemsFromInventory**

```
public static void removeItemsFromInventory(int item,
                                           int count)
```

Removes a count of item from inventory.

This method removes a count of an item from the players inventory.

**Parameters:**

**item** - The item to remove from the inventory

**count** - The count that should be removed from the inventory

**resetWorld**

```
private static void resetWorld()
```

Resets the world to an empty world.

This method resets the world to an empty world via generating an empty world and resetting the players position.

Part of secret door logic.

## saveGame

```
public static void saveGame(String19 fileName)
```

Saves the game.

This method saves the game in a file.

### Parameters:

fileName - The file name

### Catched Exceptions:

- On IOException: Prints error with message when I/O exception of some sort has occurred.

## startGame

```
public static void startGame()
```

Starts the game.

This method handles the following:

- Printing of initial UI, instructions and informational messages
- Player input
- Secret door logic

Part of secret door logic.

## waitForEnter

```
private static void waitForEnter()
```

Waits for input ENTER.

This method waits for player to input ENTER.



## References

- [Template](#) - Canvas task on which this document is based
- [yEd](#) - Graph Editor we used to make the flowcharts