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

## Table of Contents

# Group Details

| | |
|---|---|
| **Group Name** | Tacita |
| **Group Number** | 18 |
| **TA** | TA assigned to Group 18 |

## Participating Students

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

# Introduction

Meet JavaCraft, the first project we were assigned in our University journey. JavaCraft is a very simplified version of the game Minecraft that is set in a two dimensional world that is visualized using ASCII characters.

For this project, we are given a code for the JavaCraft game. That code is, what we are meant to work on.

We are supposed to expand the game in different aspects like adding new items or crafting recipes to it and documenting and understanding it, which we should show in the form of code descriptions, flowcharts, pseudocodes, automatas.

So far we've already learned a lot from this project!

# JavaCraft's Workflow

## Class JavaCraft

**Pseudocode**

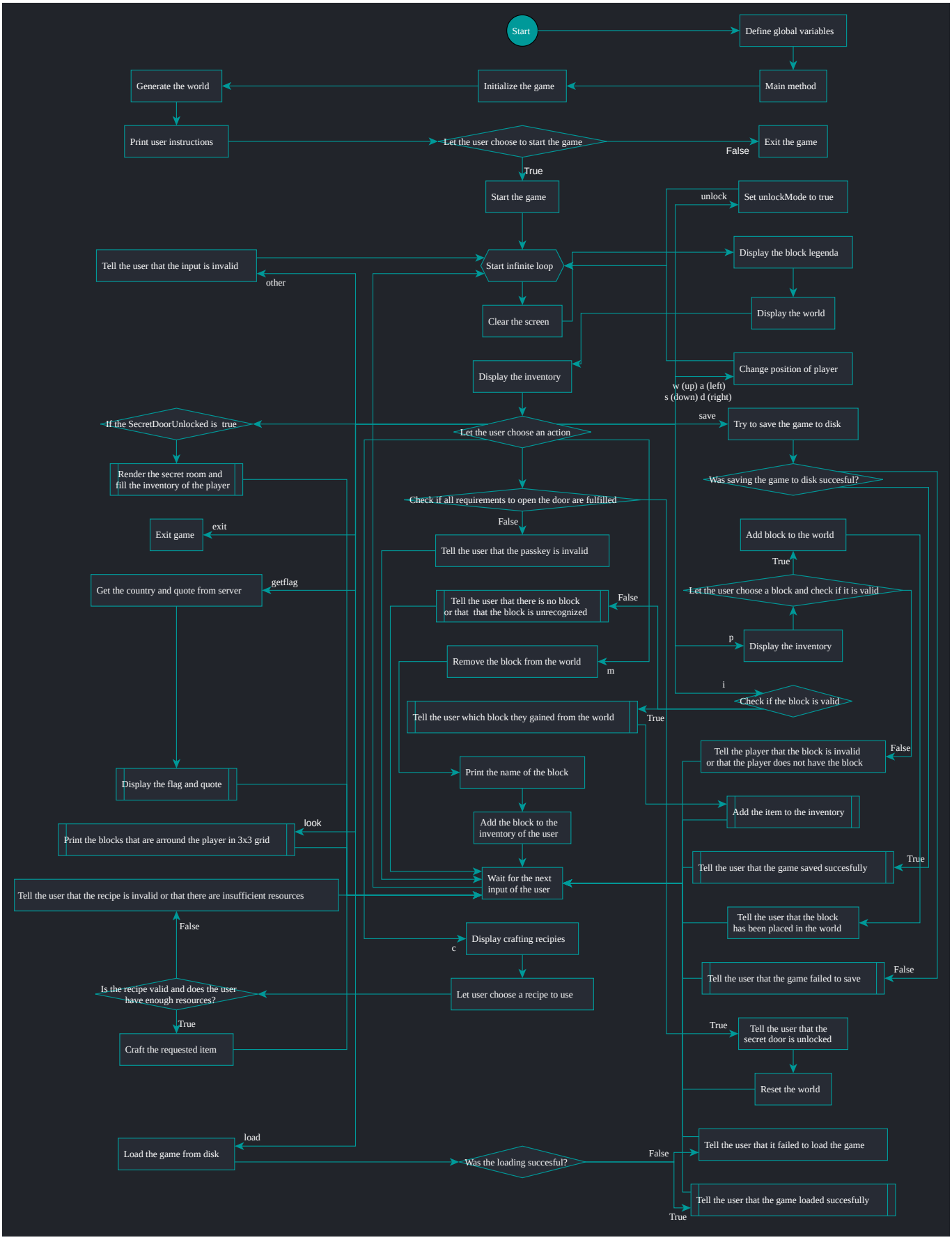
```
BEGIN

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 `<String> READ user input` == y (caseless check)
    Set `<boolean> unlockMode` = false;
    Set `<boolean> craftingCommandEntered` = false;
    Set `<boolean> miningCommandEntered` = false;
    Set `<boolean> movementCommandEntered` = false;
    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 `<String> READ user input` == "w" OR "up" OR "s" OR "down" OR
"a" OR "left" OR "d" OR "right" (caseless check)
            IF `<boolean> unlockMode` == true
                Set `<boolean> movementCommandEntered` = true;
            Move player;
        ELSE IF `<String> READ user input` == "m" (caseless check)
            IF `<boolean> unlockMode` == true
                Set `<boolean> miningCommandEntered` = true;
            Mine block;
        ELSE IF `<String> READ user input` == "p" (caseless check)
            PRINT INFO `players inventory`;
            PRINT INFO "Enter the block type to place: ";
            Place block `<String> READ user input`;
        ELSE IF `<String> READ user input` == "c" (caseless check)
            PRINT INFO `crafting recipes`;
            PRINT INFO "Enter the recipe number to craft: ";
            Craft item `<String> READ user input`;
        ELSE IF `<String> READ user input` == "i" (caseless check)
            Interact with world;
        ELSE IF `<String> READ user input` == "save" (caseless check)
            PRINT INFO "Enter the file name to save the game state: ";
            Save game as `<String> READ user input`;
        ELSE IF `<String> READ user input` == "load" (caseless check)
            PRINT INFO "Enter the file name to load the game state: ";
            Load game from `<String> READ user input`;
        ELSE IF `<String> READ user input` == "exit" (caseless check)
```

```
            PRINT INFO "Exiting the game. Goodbye!\n";
            Exit game;
        ELSE IF `<String> READ user input` == "look" (caseless check)
            Print all blocks sorrounding 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
```

## Flowchart


Start
Define global variables
Main method
Initialize the game
Generate the world
Print user instructions
Let the user choose to start the game
False
Exit the game
True
Start the game
unlock
Set unlockMode to true
Tell the user that the input is invalid
other
Start infinite loop
Display the block legenda
Clear the screen
Display the world
Display the inventory
Change position of player
w (up) a (left)
s (down) d (right)
If the SecretDoorUnlocked is true
Let the user choose an action
save
Try to save the game to disk
Render the secret room and fill the inventory of the player
Was saving the game to disk succesful?
Check if all requirements to open the door are fulfilled
False
Exit game
exit
Tell the user that the passkey is invalid
Add block to the world
True
Get the country and quote from server
getflag
Tell the user that there is no block or that that the block is unrecognized
False
Let the user choose a block and check if it is valid
p
Display the inventory
Remove the block from the world
m
i
Check if the block is valid
Tell the user which block they gained from the world
True
Tell the player that the block is invalid or that the player does not have the block
False
Print the name of the block
Display the flag and quote
Add the item to the inventory
look
Add the block to the inventory of the user
Print the blocks that are arround the player in 3x3 grid
Tell the user that the game saved succesfully
True
Wait for the next input of the user
Tell the user that the recipe is invalid or that there are insufficient resources
Tell the user that the block has been placed in the world
False
Display crafting recipies
c
Tell the user that the game failed to save
False
Is the recipe valid and does the user have enough resources?
Let user choose a recipe to use
True
True
Tell the user that the secret door is unlocked
Craft the requested item
Reset the world
load
Load the game from disk
Was the loading succesful?
False
Tell the user that it failed to load the game
Tell the user that the game loaded succesfully
True

# Functionality Exploration

See Appendix for documentation of all functions and flowcharts and pseudocodes of 16 functions.

## Code Repetition

`getBlockSymbol` contains code repetition in its switch statement, where each block contains a different color that corresponds to a different block.

This also occurs in multiple functions like `getBlockChar`, `getBlockTypeFromCraftedItem`, `getCraftedItemFromBlockType`, `getRequiredItemForMining`, `craftItem`, `craftStonePickaxe`, `craftIronPickaxe`, `craftWoodenPlanks`, `craftStick`, `craftIronIngot`, `interactWithWorld`, `getBlockName` and `getCraftedItemColor`.

`inventoryContains` and `craftedItemsContains` are almost identical and the general concepts are exactly the same.

# 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

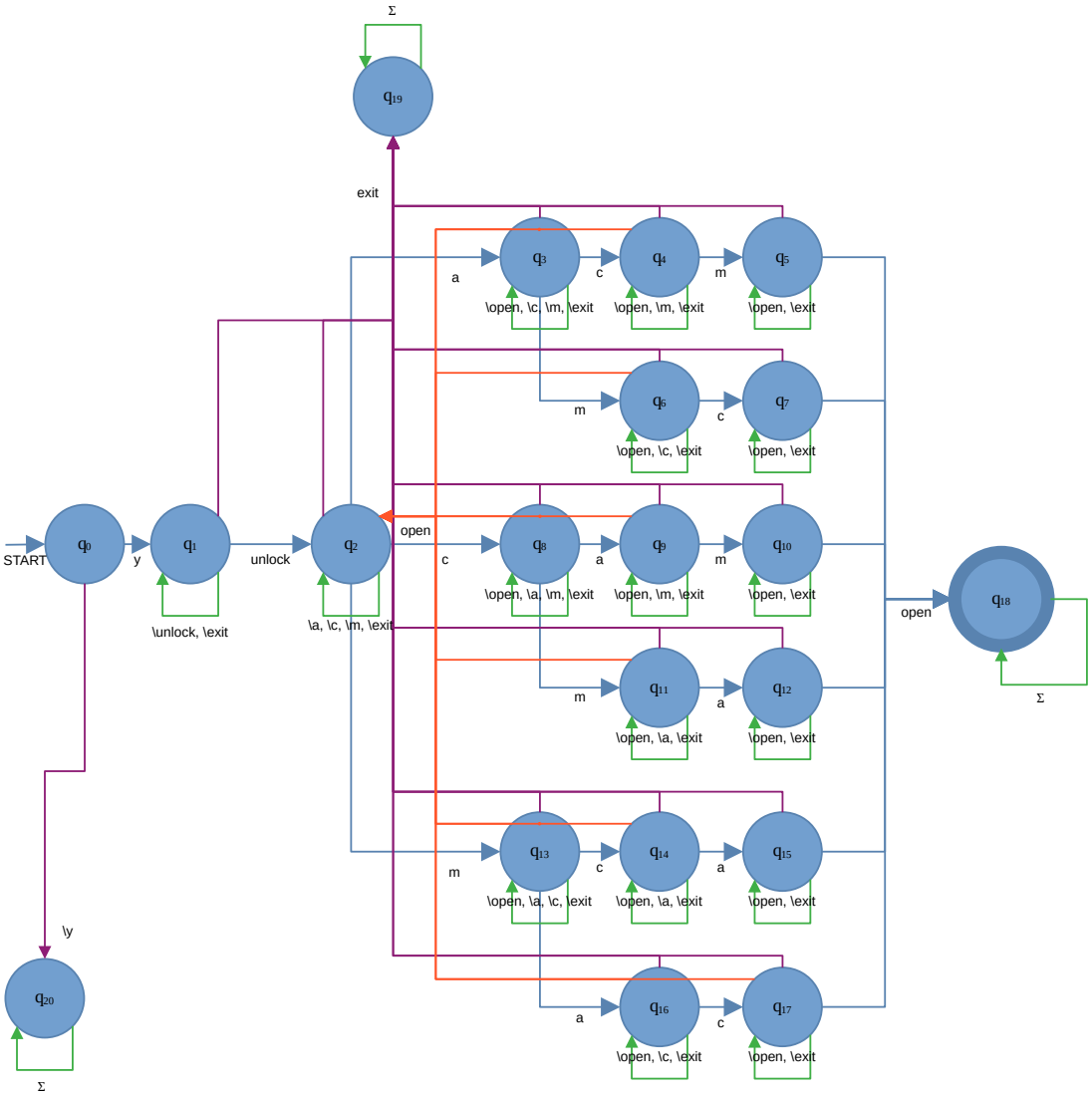
$$D=(Q, \Sigma, \delta, q_0, F)$$

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

Σ={y, unlock, a, c, m, open, exit} (caseless check)
δ: Transition Function

L(D)={y, unlock, {mandatory a, c, m and optional y, 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={$q_{18}$}



**Table**

| State | y | unlock | a | c | m | open | exit |
|---|---|---|---|---|---|---|---|
| $^{\rightarrow}q_{0}$ | $q_{1}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ |
| $q_{1}$ | $q_{1}$ | $q_{2}$ | $q_{1}$ | $q_{1}$ | $q_{1}$ | $q_{1}$ | $q_{19}$ |
| $q_{2}$ | $q_{2}$ | $q_{2}$ | $q_{3}$ | $q_{8}$ | $q_{13}$ | $q_{2}$ | $q_{19}$ |
| $q_{3}$ | $q_{3}$ | $q_{3}$ | $q_{3}$ | $q_{4}$ | $q_{6}$ | $q_{2}$ | $q_{19}$ |
| $q_{4}$ | $q_{4}$ | $q_{4}$ | $q_{4}$ | $q_{4}$ | $q_{5}$ | $q_{2}$ | $q_{19}$ |
| $q_{5}$ | $q_{5}$ | $q_{5}$ | $q_{5}$ | $q_{5}$ | $q_{5}$ | $q_{18}$ | $q_{19}$ |
| $q_{6}$ | $q_{6}$ | $q_{6}$ | $q_{6}$ | $q_{7}$ | $q_{6}$ | $q_{2}$ | $q_{19}$ |
| $q_{7}$ | $q_{7}$ | $q_{7}$ | $q_{7}$ | $q_{7}$ | $q_{7}$ | $q_{18}$ | $q_{19}$ |
| $q_{8}$ | $q_{8}$ | $q_{8}$ | $q_{9}$ | $q_{8}$ | $q_{11}$ | $q_{2}$ | $q_{19}$ |
| $q_{9}$ | $q_{9}$ | $q_{9}$ | $q_{9}$ | $q_{9}$ | $q_{10}$ | $q_{2}$ | $q_{19}$ |
| $q_{10}$ | $q_{10}$ | $q_{10}$ | $q_{10}$ | $q_{10}$ | $q_{10}$ | $q_{18}$ | $q_{19}$ |
| $q_{11}$ | $q_{11}$ | $q_{11}$ | $q_{12}$ | $q_{11}$ | $q_{11}$ | $q_{2}$ | $q_{19}$ |
| $q_{12}$ | $q_{12}$ | $q_{12}$ | $q_{12}$ | $q_{12}$ | $q_{12}$ | $q_{18}$ | $q_{19}$ |
| $q_{13}$ | $q_{13}$ | $q_{13}$ | $q_{16}$ | $q_{14}$ | $q_{13}$ | $q_{1}$ | $q_{19}$ |
| $q_{14}$ | $q_{14}$ | $q_{14}$ | $q_{15}$ | $q_{14}$ | $q_{14}$ | $q_{2}$ | $q_{19}$ |
| $q_{15}$ | $q_{15}$ | $q_{15}$ | $q_{15}$ | $q_{15}$ | $q_{15}$ | $q_{18}$ | $q_{19}$ |
| $q_{16}$ | $q_{16}$ | $q_{16}$ | $q_{16}$ | $q_{17}$ | $q_{16}$ | $q_{2}$ | $q_{19}$ |
| $q_{17}$ | $q_{17}$ | $q_{17}$ | $q_{17}$ | $q_{17}$ | $q_{17}$ | $q_{18}$ | $q_{19}$ |
| $^{*}q_{18}$ | $q_{18}$ | $q_{18}$ | $q_{18}$ | $q_{18}$ | $q_{18}$ | $q_{18}$ | $q_{18}$ |
| $q_{19}$ | $q_{19}$ | $q_{19}$ | $q_{19}$ | $q_{19}$ | $q_{19}$ | $q_{19}$ | $q_{19}$ |
| $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ | $q_{20}$ |

# Git Collaboration & Version Control

## Overview

### UM Gitlab Repository, Branch Group 18

#### 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.

#### 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.

# Extending the game code

## Blocktypes

The blocktypes we added are coal and emerald, we added them to the Game by assigning them an integer value and an ANSI color. We had to change a few functions to be able to fully integrate them into the game.

The first being `generateWorld()` in which we tweaked the rate at which the blocks spawn in the world. We made sure to match their rarity.

We also had to make some minor changes, for instance assigning the color to the integer value in `getBlockSymbol()`, and assigning them ASCII characters in `getBlockChar()`. Afterwards we changed integer values in `fillInventory()`, `placeBlock()` and `displayInventory()`. This had to be done to match the new amount of blocktypes. Otherwise the Game would've only used the old Blocktypes.

Additionally we assigned String values to the new blocktypes in `getBlockName()`, assigned each block to its color in `getBlockSymbol()` and added them to the legend in `displayLegend()`. Whenever one of our blocks is mined, a message will also be printed `interactWithWorld()`.

## Crafted Items

Our crafted Items we added to the game are iron and stone pickaxe, crafting the stone pickaxe requires three stone and one stick, crafting the iron pickaxe requires three iron ingot and one stick.

We chose these items because we wanted to implement a mechanic, that only lets a player mine a block if he fulfills certain requirements.

In this case for the player to be able to mine coal and iron blocks, he needs to have a stone pickaxe in his inventory. To be able to mine emerald blocks he needs an iron pickaxe.

To accomplish this, we had to first implement the crafted items. We did this in similar fashion as the blocktypes by assigning them integer values. And adding their values to the preexisting crafted items methods.

Afterwards we implemented the methods `craftStonePickaxe()` and `craftIronPickaxe()` in which we specified the crafting requirements for each Item. for this to work we had to add a new method `removeItemFromCraftedItem()`, that removes items from the crafted items inventory. And `craftedItemsContains()` that checks if the player has the amount of crafted items in his Inventory.

The biggest change was the implementation of the mine requirements in `mineBlock()`, we did this by checking for the blocktype that is going to be mined first and then checking if the player fits the requirements.

To do this we implemented a new method `getRequiredItemForMining()` which gets the Blocktype as parameter and gives back the needed Crafted Item to be able to mine it.

## Interacting with Flags API

We have rewritten the template function `getCountryAndQuoteFromServer()` to interact with the flags API at `https://flag.ashish.nl`.

The old code used a now deprecated constructor for URL: `new URL(String)`. Java complains with the following warning: `The constructor URL(String) is deprecated since version 20`. Therefore we decided on using `URI.create(String).toURL()` instead. This is not deprecated.

The rest of our code just uses the provided template which gets a country and a quote from the flags API via a POST request. Within the post request we send a json String containing the following:

- "group_number" : "18"
- "group_name" : "group18"
- "difficulty_level" : "hard"

This is meant to identify our group via it's name and number and lets the server know which difficulty level it should choose for the flag.

Since we only use this to know which flag we have to build, it wasn't necessary to pretty print any response we get. Therefore we didn't work on that and didn't really change the code.

In our current code we have replaced `https://flag.ashish.nl/get_flag` with `https://example.com` to avoid unnecessary interactions with the API.

We got Sri Lanka as our first response and used a string to represent it's flag. The result is the following:

# Conclusion

We created flowcharts for 16 funtions, tried to document the code in an organized fashion and as expected encountered no lack of issues along the way.

For instance, it was really challenging to fit the flowchart of the whole game on one page. Sian managed to do that anyways, even though he was grasping at straws.

Leo encountered some difficulties while constructing the FSA and had to redo the automaton multiple times. This was because of some misunderstandings about what was expected from us.

Anton faced some problems while adding new blocks and crafting recipes to the game. His difficulties were exaggerated due to the fact that he entered the course with minimal programming experience, nevertheless with enough persistance he managed to enrich JavaCraft's gameplay.

We learned how to work together in a team and to manage and divide team tasks. Also included in our learning experience was learning to maintain a functioning and readable codebase and fighting over who gets to do what. We became skilled at reading and understanding code written by someone else, via pseudocode and flowcharts, this in turn greatly helped us advance our java knowledge.

In the final stages of our project, we managed to create a proper looking and well formatted pdf using markdown. We also learned how to use an API and how to draw a challenging flag using only UNICODE characters and 16 ANSI colors.

This project has been a very good start to our BSc Computer Science and helped us a lot with getting used to working on university projects at Maastricht University.

# Who Did What?

| Task | Who worked on the task | Participation in percentage |
|---|---|---|
| Creating initial pseudocode and flowcharts | Leopold, Anton, Tristan, Sian | Even across all participants |
| Setting up Gitlab repository | Leopold, Sian | Even across all participants |
| Creating documentation for JavaCraft code | Leopold, Anton, Tristan, Sian | Even across all participants |
| Finding repetitions in code | Sian | 100% |
| Creating flowchart and pseudocode for class JavaCraft | Tristan | 100% |
| Creating FSA for automaton | Leopold, Tristan | 90%, 10% |
| Creating table and description for automaton | Leopold | 100% |
| Converting ODF Flowcharts to .graphml | Tristan | 100% |
| Deciding on the uniformal format for flowcharts | Leopold, Anton, Tristan, Sian | Even across all participants |
| Deciding on the uniformal format for pseudocode | Leopold, Anton, Tristan, Sian | 70%, 10%, 10%, 10% |
| Converting flowcharts to uniformal format | Sian, Tristan, Anton | 80%, 10%, 10% |
| Converting pseudocode to uniformal format | Leopold | 100% |
| Creating documentation | Leopold | 100% |
| Cleaning up repository directories | Sian | 100% |
| Exporting flowcharts to SVG format | Sian | 100% |
| Implementing two new blocks and two new crafting items | Anton | 100% |
| Updating functions involved with new blocks and crafting items | Anton | 100% |
| Creating provisional report document | Leo, Tristan, Anton, Sian | 70%, 10%, 10%, 10% |
| Merging flowchart images with report document into single PDF | Sian | 100% |
| Implementing uniformal directory structure | Leopold | 100% |

# Appendix

## Extending the Gamecode

### boolean craftedItemContains()

#### Documentation

**craftedItemsContains**

```
public static boolean craftedItemsContains(int craftedItem,
                                           int count)
```

Queries craftedItems for if it has enough of an crafted item.

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

**Parameters:**

`craftedItem` - The crafted item to query the crafted items inventory for

`count` - The count that the crafted items inventory should contain of the item

**Returns:**

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

#### Java

```java
public static boolean craftedItemsContains(int craftedItem, int count) {
        int craftedItemCount = 0;
        for (int i : craftedItems) {
            if (i == craftedItem) {
                craftedItemCount++;
                if (craftedItemCount == count) {
                    return true;
                }
            }
        }
        return false;
    }
```

### void craftIronPickaxe()

#### Documentation

**craftIronPickaxe**

```
public static void craftIronPickaxe()
```

Crafts CRAFTED_IRON_PICKAXE.

This method crafts CRAFTED_IRON_PICKAXE from 1 Stick and 3 Iron Ingots that are taken form the players inventory.

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

#### Java

```java
public static void craftIronPickaxe() {
        if (craftedItemsContains(CRAFTED_STICK) &&
craftedItemsContains(CRAFTED_IRON_INGOT, 3)) {
            removeItemFromCraftedItems(CRAFTED_STICK, 1);
```

```
            removeItemFromCraftedItems(CRAFTED_IRON_INGOT, 3);
            addCraftedItem(CRAFTED_IRON_PICKAXE);
            System.out.println("Crafted Iron Pickaxe");
        } else {
            System.out.println("Insufficient resources to craft Stone
Pickaxe");
        }
    }
```

### void craftStonePickaxe()

#### Documentation

**craftStonePickaxe**

`public static void craftStonePickaxe()`

Crafts CRAFTED_STONE_PICKAXE.

This method crafts CRAFTED_STONE_PICKAXE from 1 Stick and 3 Stone that are taken form the players inventory.

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

#### Java

```
public static void craftStonePickaxe() {
        if (craftedItemsContains(CRAFTED_STICK) && inventoryContains(STONE,
3)) {
            removeItemFromCraftedItems(CRAFTED_STICK, 1);
            removeItemsFromInventory(STONE, 3);
            addCraftedItem(CRAFTED_STONE_PICKAXE);
            System.out.println("Crafted Stone Pickaxe");
        } else {
            System.out.println("Insufficient resources to craft Stone
Pickaxe");
        }
    }
```

### void displayLegend()

#### Documentation

**displayLegend**

`public static void displayLegend()`

Prints a legend.

This method prints a legend of items on the map.

#### Java

```
public static void displayLegend() {
        System.out.println(ANSI_BLUE + "Legend:");
```

```
        System.out.println(ANSI_WHITE + "-- - Empty block");
        System.out.println(ANSI_RED + "\u2592\u2592 - Wood block");
        System.out.println(ANSI_GREEN + "\u00A7\u00A7 - Leaves block");
        System.out.println(ANSI_BLUE + "\u2593\u2593 - Stone block");
        System.out.println(ANSI_WHITE + "\u00B0\u00B0- Iron ore block");
        System.out.println(ANSI_COAL_GRAY + "\u2593\u2593 - Coal ore
block");
        System.out.println(ANSI_EMERALD_GREEN + "\u00B0\u00B0 - Emerald ore
block");
        System.out.println(ANSI_BLUE + "P - Player" + ANSI_RESET);
    }
```

## int getRequiredItemForMining()

### Documentation

**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

### Java

```
    public static int getRequiredItemForMining(int blockType) {
        switch (blockType) {
            case 4:
                return CRAFTED_STONE_PICKAXE;
            case 5:
                return CRAFTED_STONE_PICKAXE;
            case 6:
                return CRAFTED_IRON_PICKAXE;
            default:
                return -1;
        }
    }
```

## void interactWithWorld()

### Documentation

**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.

#### Java

```
public static void interactWithWorld() {
        int blockType = world[playerX][playerY];
        switch (blockType) {
            case WOOD:
                System.out.println("You gather wood from the tree.");
                inventory.add(WOOD);
                break;
            case LEAVES:
                System.out.println("You gather leaves from the tree.");
                inventory.add(LEAVES);
                break;
            case STONE:
                System.out.println("You gather stones from the ground.");
                inventory.add(STONE);
                break;
            case IRON_ORE:
                System.out.println("You mine iron ore from the ground.");
                inventory.add(IRON_ORE);
                break;
            case EMERALD_ORE:
                System.out.println("You mine emerald ore from the
ground.");
                inventory.add(EMERALD_ORE);
                break;
            case COAL_ORE:
                System.out.println("You mine coal ore from the ground.");
                inventory.add(COAL_ORE);
                break;
            case AIR:
                System.out.println("Nothing to interact with here.");
                break;
            default:
                System.out.println("Unrecognized block. Cannot interact.");
        }
        waitForEnter();
    }
```

### mineBlock()

#### Documentation

**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.

#### Java

```
public static void mineBlock() {
        int blockType = world[playerX][playerY];
        if (blockType == EMERALD_ORE) {
            if (craftedItems.contains(getRequiredItemForMining(blockType)))
{
                inventory.add(blockType);
                world[playerX][playerY] = AIR;
                System.out.println("Mined " + getBlockName(blockType) +
".");
            } else {
                System.out.println(
                        "You need: " +
getCraftedItemName(getRequiredItemForMining(blockType))
                                + ", to mine a " +
getBlockName(blockType));
            }
        } else if (blockType == IRON_ORE || blockType == COAL_ORE) {
            if (craftedItems.contains(getRequiredItemForMining(blockType)))
{
                inventory.add(blockType);
                world[playerX][playerY] = AIR;
                System.out.println("Mined " + getBlockName(blockType) +
".");
            } else {
                System.out.println(
                        "You need: " +
getCraftedItemName(getRequiredItemForMining(blockType))
                                + ", to mine a " +
getBlockName(blockType));
            }
        } else if (blockType != AIR) {
            inventory.add(blockType);
            world[playerX][playerY] = AIR;
            System.out.println("Mined " + getBlockName(blockType) + ".");
        } else {
            System.out.println("No block to mine here.");
        }
        waitForEnter();
    }
```

## void removeItemFromCraftedItem()

### Documentation

**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

#### Java

```
public static void removeItemFromCraftedItems(int craftedItem, int count) {
        int removedCount = 0;
        Iterator<Integer> iterator = craftedItems.iterator();
        while (iterator.hasNext()) {
            int i = iterator.next();
            if (i == craftedItem) {
                iterator.remove();
                removedCount++;
                if (removedCount == count) {
                    break;
                }
            }
        }
    }
```

## 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();
    }
}
```

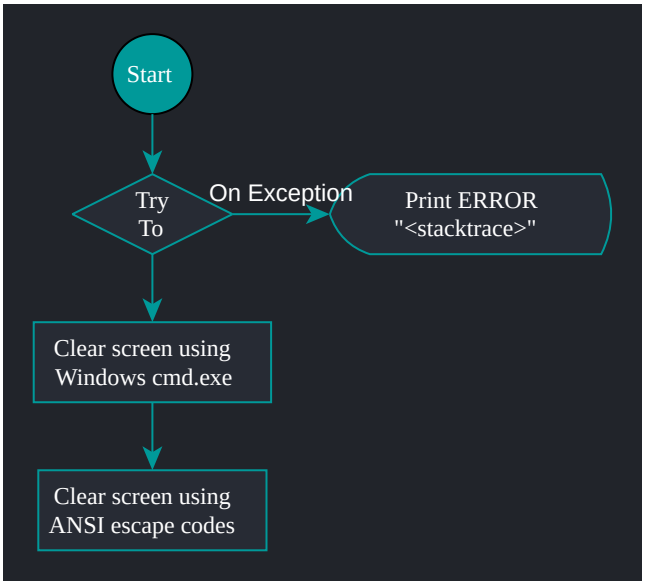
### Pseudocode

```
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


Start
Try
To
On Exception
Print ERROR
"<stacktrace>"
Clear screen using
Windows cmd.exe
Clear screen using
ANSI escape codes

## 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

```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.");
    }
}
```

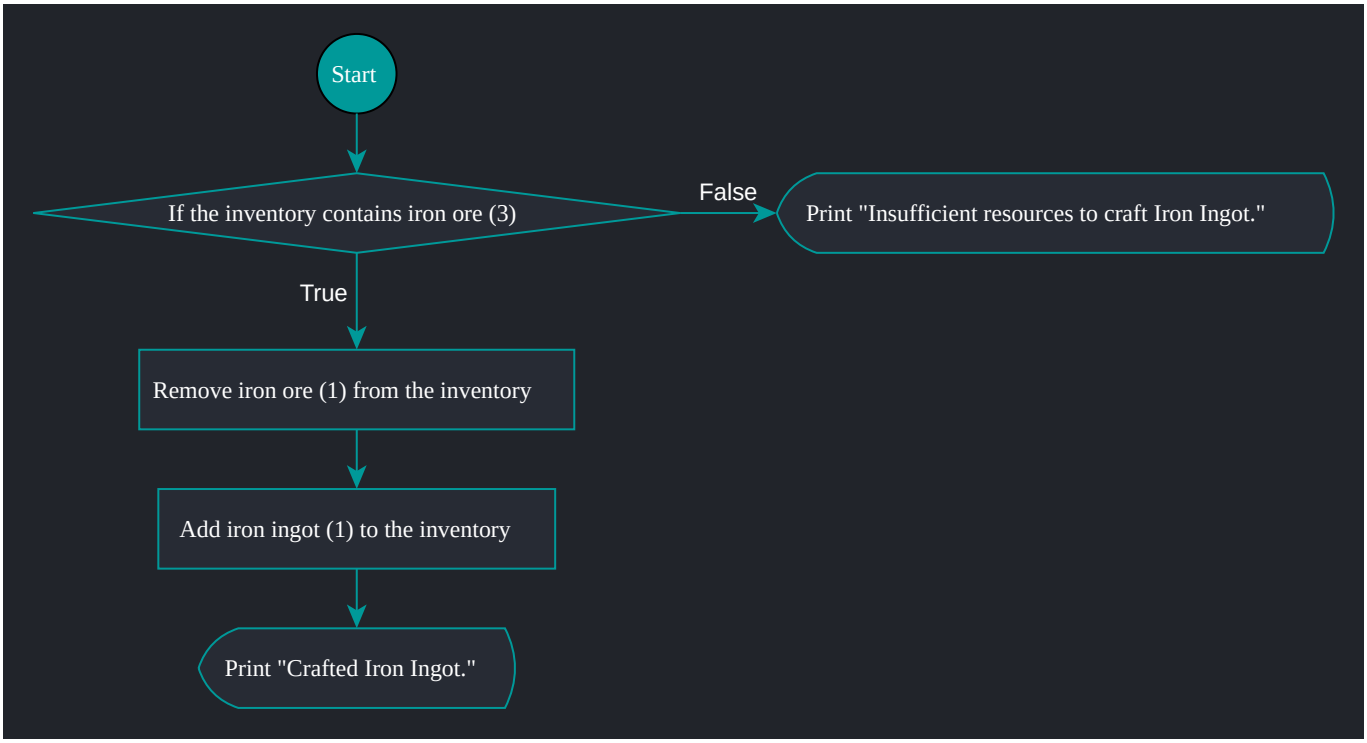
### Pseudocode

```
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


Start
If the inventory contains iron ore (3)
False
Print "Insufficient resources to craft Iron Ingot."
True
Remove iron ore (1) from the inventory
Add iron ingot (1) to the inventory
Print "Crafted Iron Ingot."

## 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

```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();
}
```

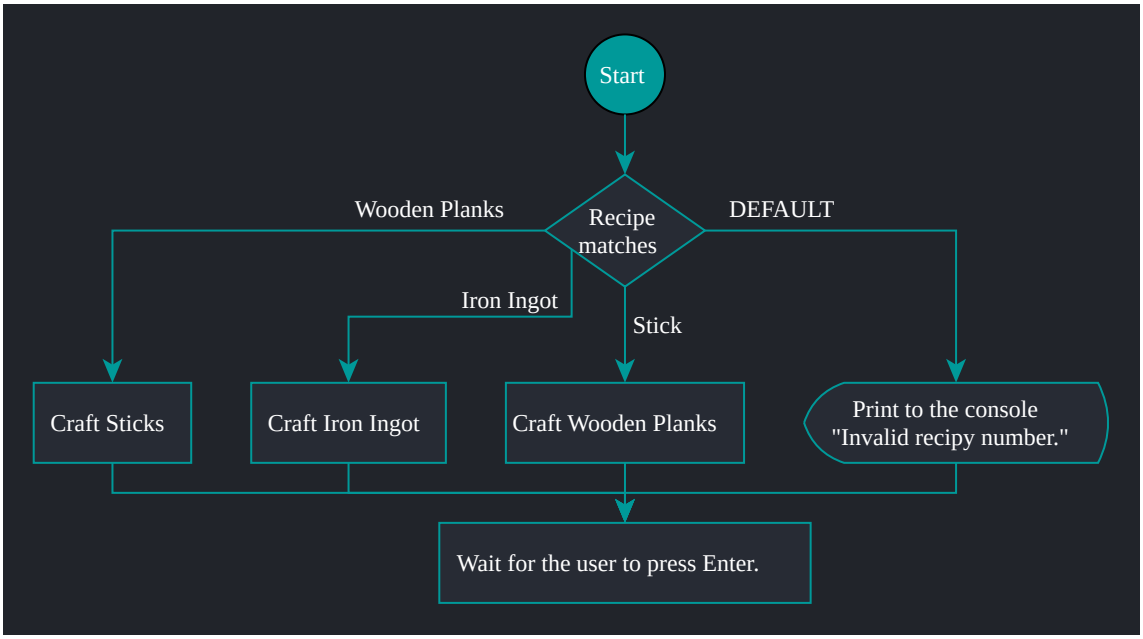
### Pseudocode

```
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

```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.");
    }
}
```

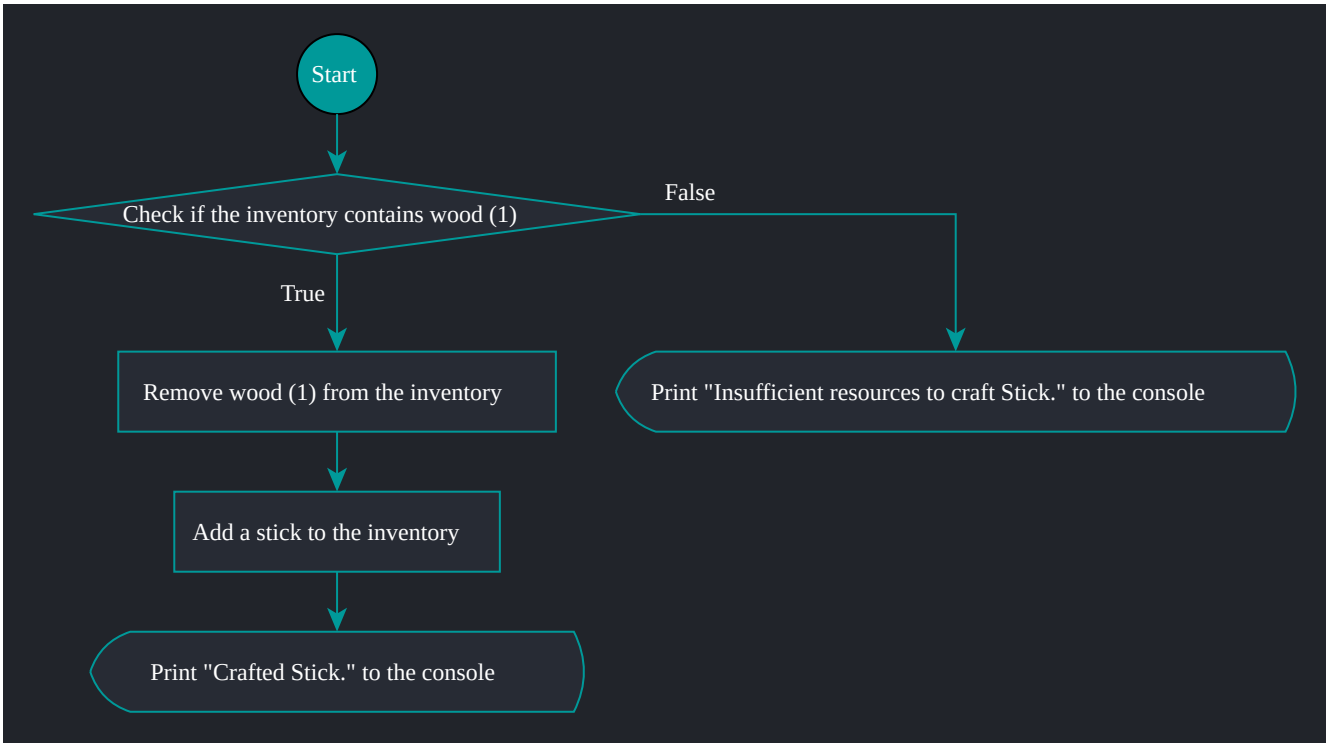
### Pseudocode

```
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


Start
Check if the inventory contains wood (1)
False
True
Remove wood (1) from the inventory
Print "Insufficient resources to craft Stick." to the console
Add a stick to the inventory
Print "Crafted Stick." to the console

## 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

```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.");
    }
}
```

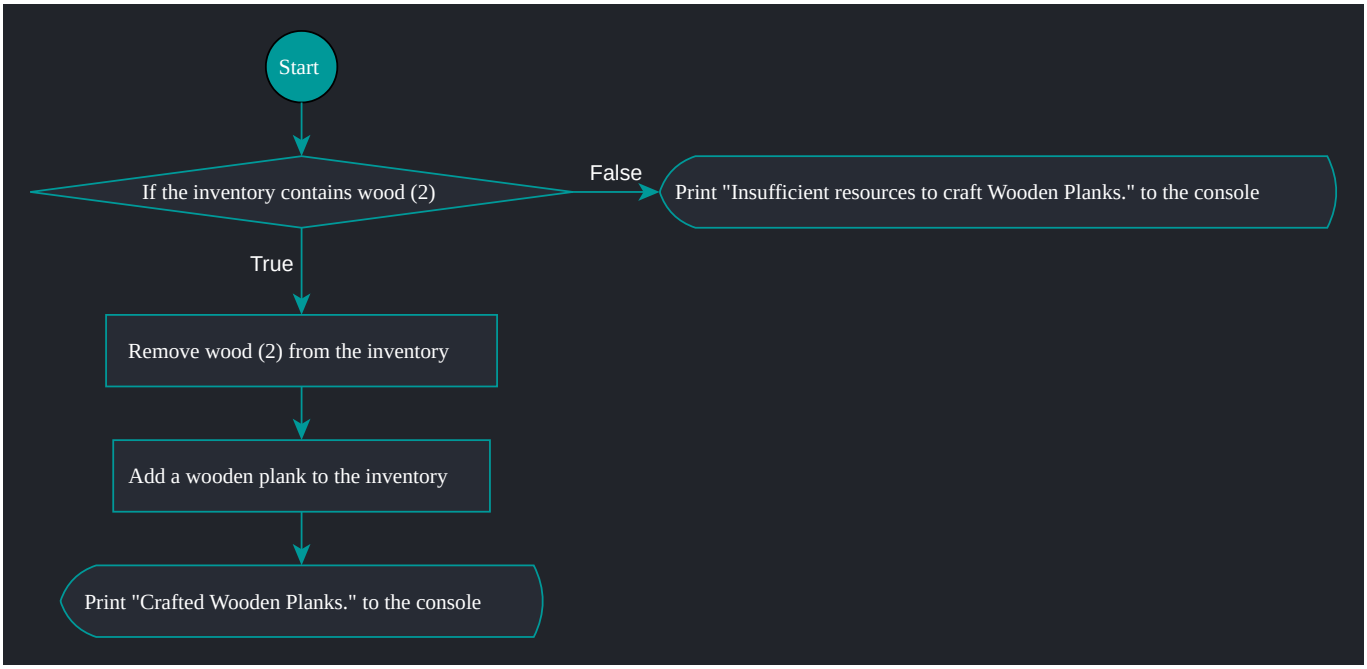
### Pseudocode

```
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


Start
If the inventory contains wood (2)
False
Print "Insufficient resources to craft Wooden Planks." to the console
True
Remove wood (2) from the inventory
Add a wooden plank to the inventory
Print "Crafted Wooden Planks." to the console

void displayCraftingRecipes()

### Documentation

**displayCraftingRecipes**

`public static void displayCraftingRecipes()`

Prints crafting recipes.

This method prints the available crafting recipes.

### Java

```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");
}
```

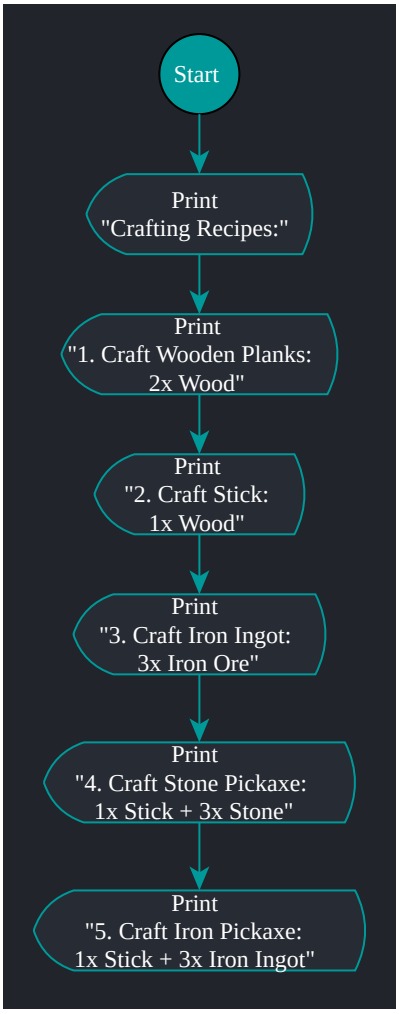
### Pseudocode

```
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


Start
Print "Crafting Recipes:"
Print "1. Craft Wooden Planks: 2x Wood"
Print "2. Craft Stick: 1x Wood"
Print "3. Craft Iron Ingot: 3x Iron Ore"
Print "4. Craft Stone Pickaxe: 1x Stick + 3x Stone"
Print "5. Craft Iron Pickaxe: 1x Stick + 3x Iron Ingot"

## 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();
}
```

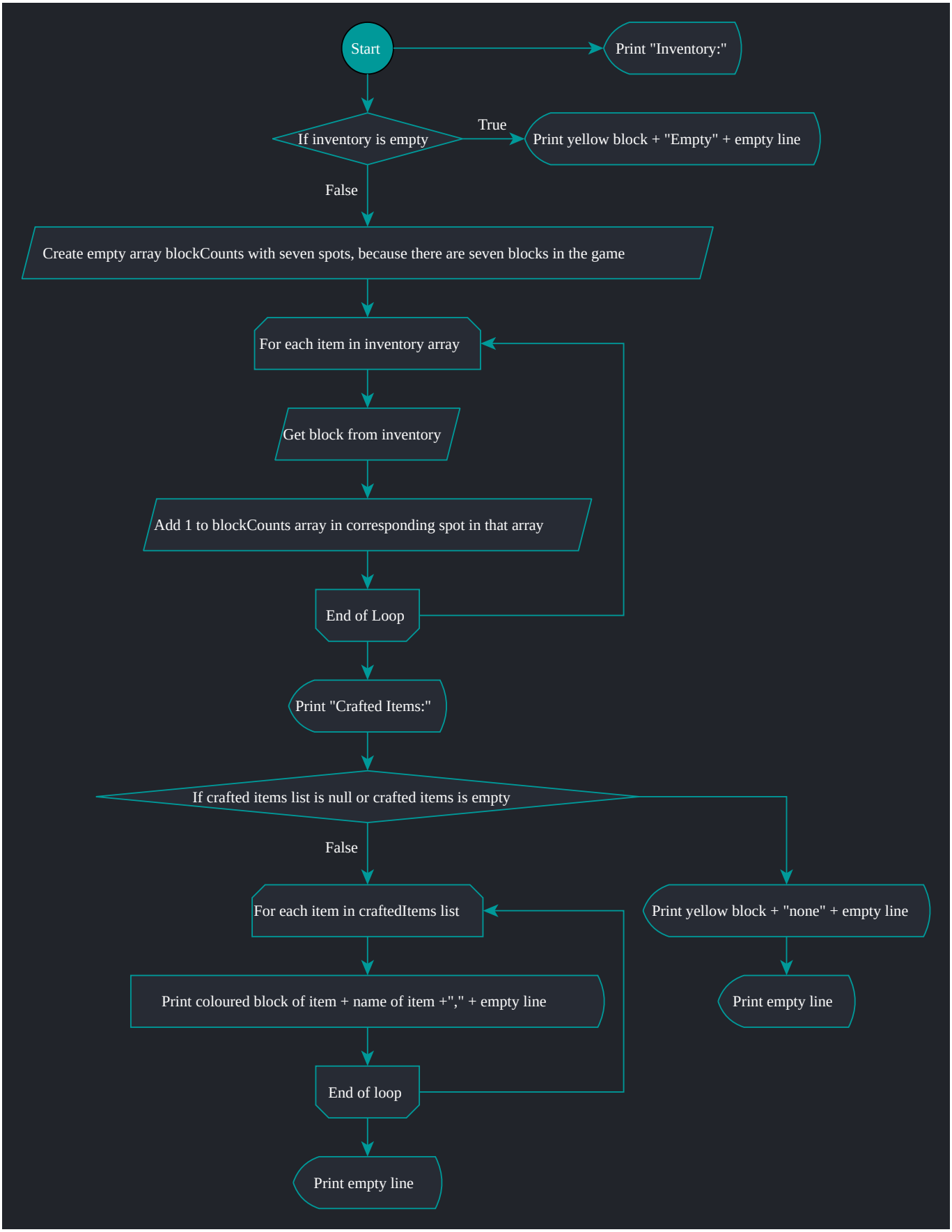
**Pseudocode**

```
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> occurences` = `<Integer array> blockCounts @
index <Integer> blockType`;
        IF `<Integer> occurences` > 0
            PRINT INFO `<String> get block name matching <Integer>
blockType` + " - " + `<Integer> occurences\n`;
        Set `<Integer> blockType` += 1;
PRINT INFO "Crafted Items:\n";
IF `<Integer list> craftedItems` is non-existant 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


Start
Print "Inventory:"
If inventory is empty
True
Print yellow block + "Empty" + empty line
False
Create empty array blockCounts with seven spots, because there are seven blocks in the game
For each item in inventory array
Get block from inventory
Add 1 to blockCounts array in corresponding spot in that array
End of Loop
Print "Crafted Items:"
If crafted items list is null or crafted items is empty
False
For each item in craftedItems list
Print coloured block of item + name of item +"," + empty line
End of loop
Print empty line
Print yellow block + "none" + empty line
Print empty line

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

```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);
        }
    }
}
```

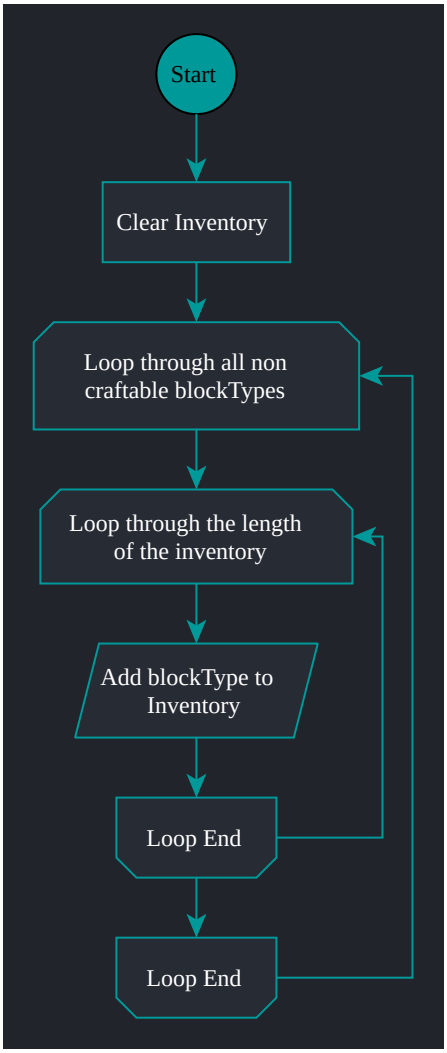
### Pseudocode

```
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


Start
Clear Inventory
Loop through all non craftable blockTypes
Loop through the length of the inventory
Add blockType to Inventory
Loop End
Loop End

# void generateWorld()

## Documentation

**generateWorld**

`public static void generateWorld()`

Generates the world.

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

## Java

```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;
            }
        }
    }
}
```

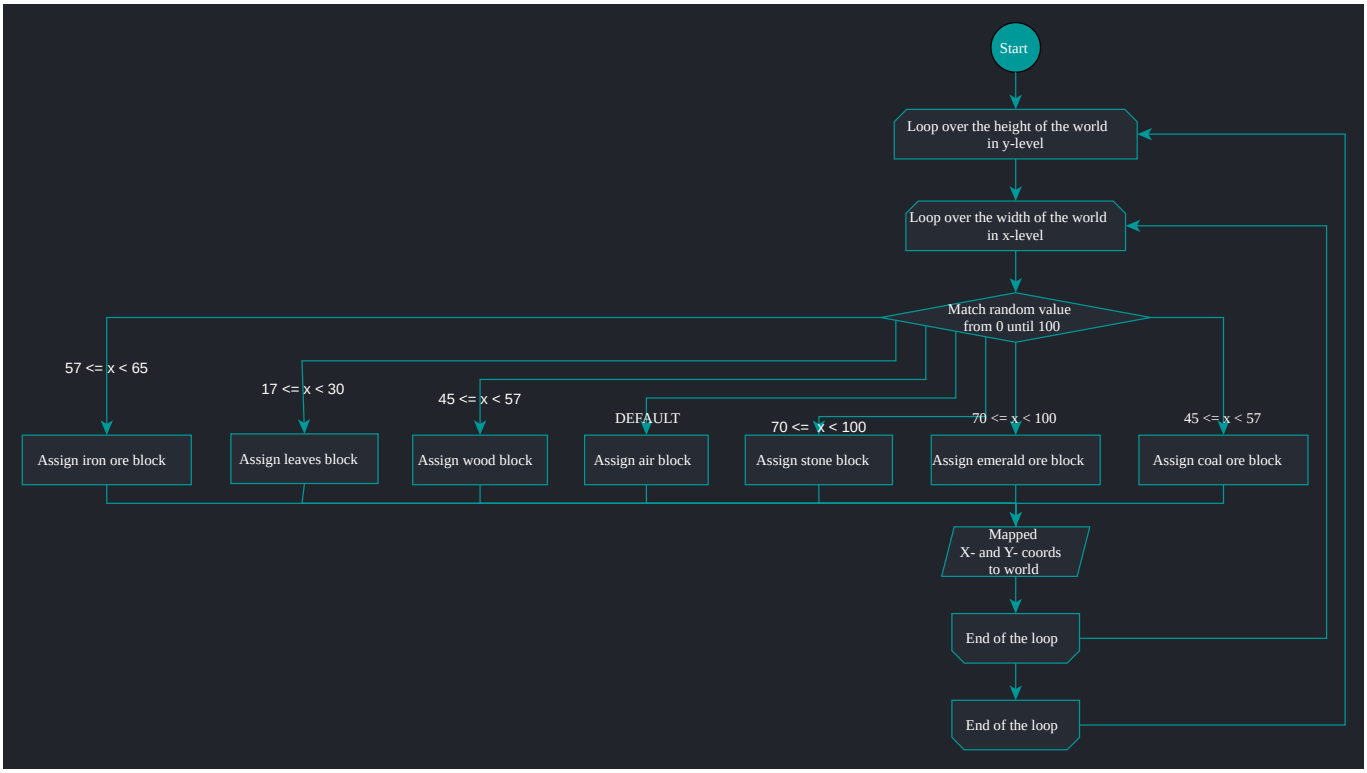
**Pseudocode**

```
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


Start
Loop over the height of the world in y-level
Loop over the width of the world in x-level
Match random value from 0 until 100
57 <= x < 65
17 <= x < 30
45 <= x < 57
DEFAULT
70 <= x < 100
70 <= x < 100
45 <= x < 57
Assign iron ore block
Assign leaves block
Assign wood block
Assign air block
Assign stone block
Assign emerald ore block
Assign coal ore block
Mapped X- and Y- coords to world
End of the loop
End of the loop

# 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

```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 '-';
    }
}
```

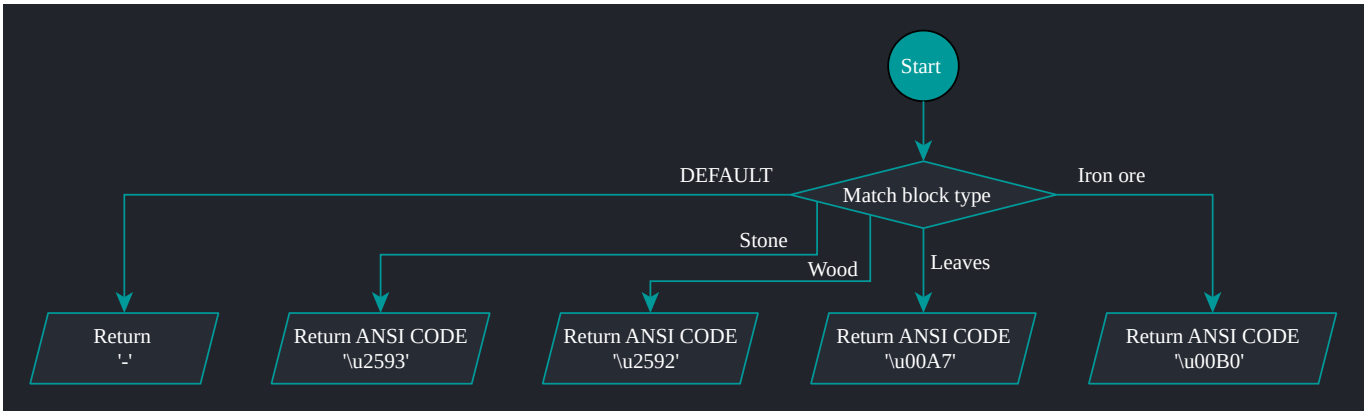
**Pseudocode**

```
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


Start
Match block type
DEFAULT
Iron ore
Stone
Wood
Leaves
Return
'-'
Return ANSI CODE
'\u2593'
Return ANSI CODE
'\u2592'
Return ANSI CODE
'\u00A7'
Return ANSI CODE
'\u00B0'

# 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

```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";
    }
}
```

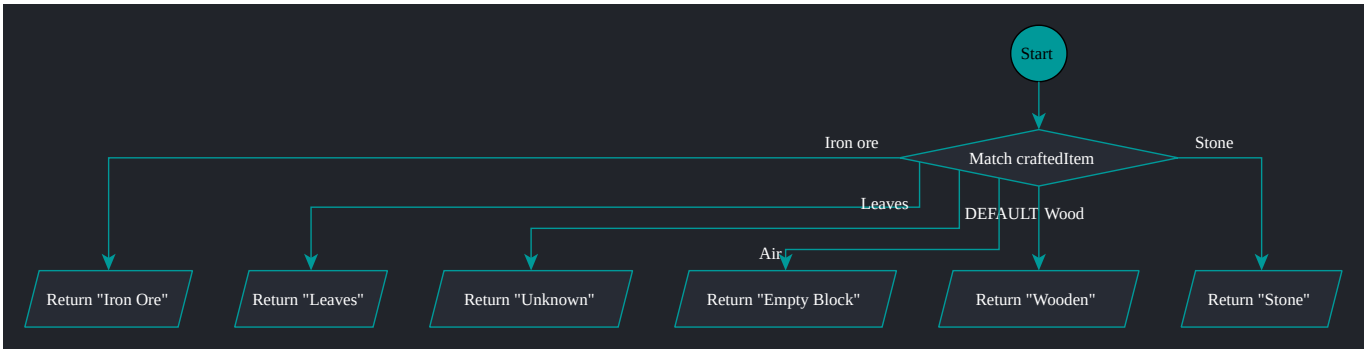
**Pseudocode**

```
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


Start
Match craftedItem
Iron ore
Stone
Leaves
DEFAULT
Wood
Air
Return "Iron Ore"
Return "Leaves"
Return "Unknown"
Return "Empty Block"
Return "Wooden"
Return "Stone"

## 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

```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) + " ";
}
```

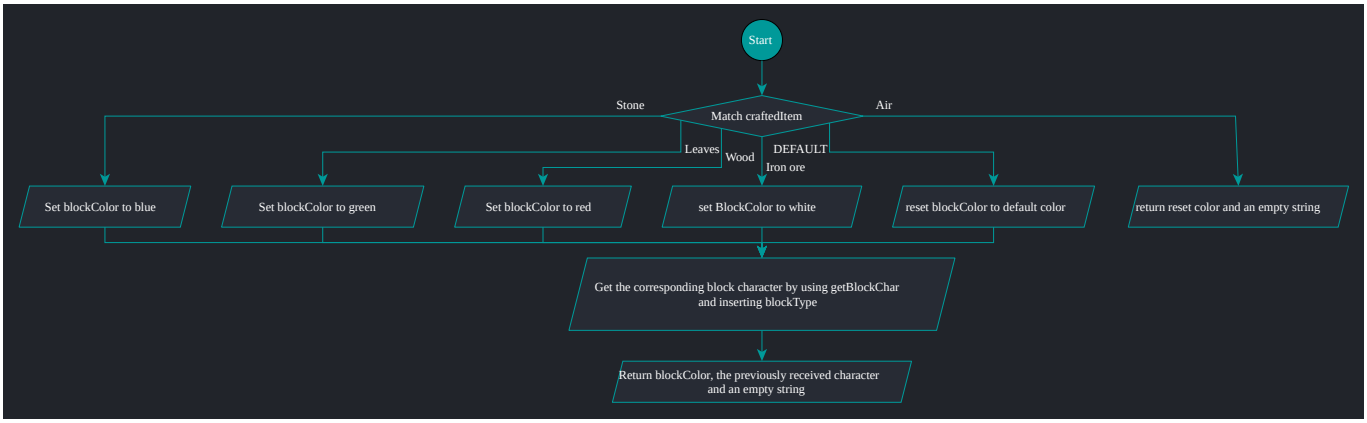
**Pseudocode**

```
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


Start
Match craftedItem
Stone
Air
Leaves
Wood
Iron ore
DEFAULT
Set blockColor to blue
Set blockColor to green
Set blockColor to red
set BlockColor to white
reset blockColor to default color
return reset color and an empty string
Get the corresponding block character by using getBlockChar and inserting blockType
Return blockColor, the previously received character and an empty string

# String getCraftedItemName(int craftedItem)

## Documentation

**getCraftedItemName**

`private static String getCraftedItemName(int craftedItem)`

Returns human readble 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

```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();
}
```

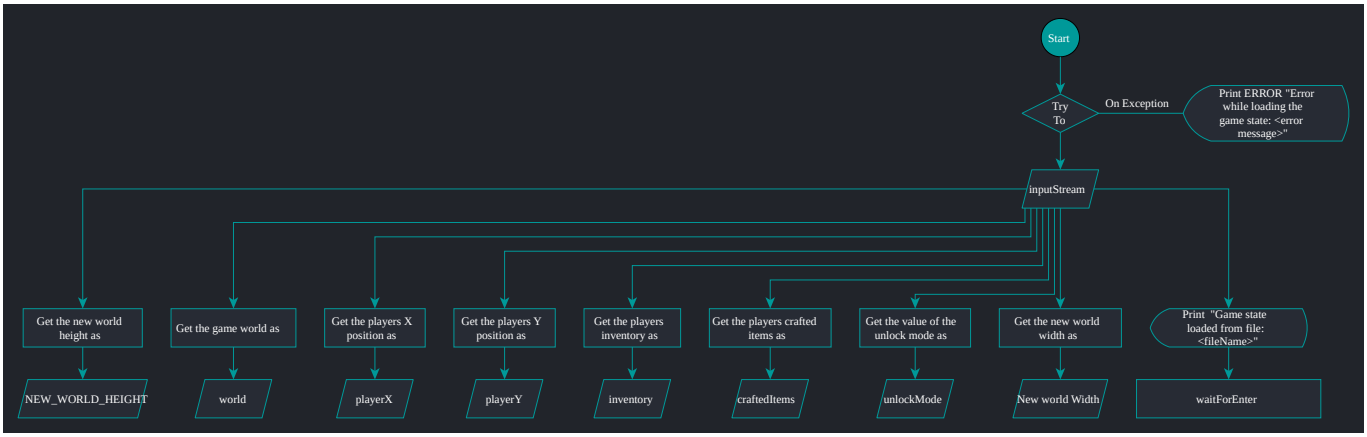
**Pseudocode**

```
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


Start
Try
To
On Exception
Print ERROR "Error while loading the game state: <error message>"
inputStream
Get the new world height as
Get the game world as
Get the players X position as
Get the players Y position as
Get the players inventory as
Get the players crafted items as
Get the value of the unlock mode as
Get the new world width as
Print "Game state loaded from file: <fileName>"
NEW_WORLD_HEIGHT
world
playerX
playerY
inventory
craftedItems
unlockMode
New world Width
waitForEnter

void lookAround()

### Documentation

**lookAround**

`private static void lookAround()`

Prints all blocks sorrounding the player.

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

### Java

```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();
}
```

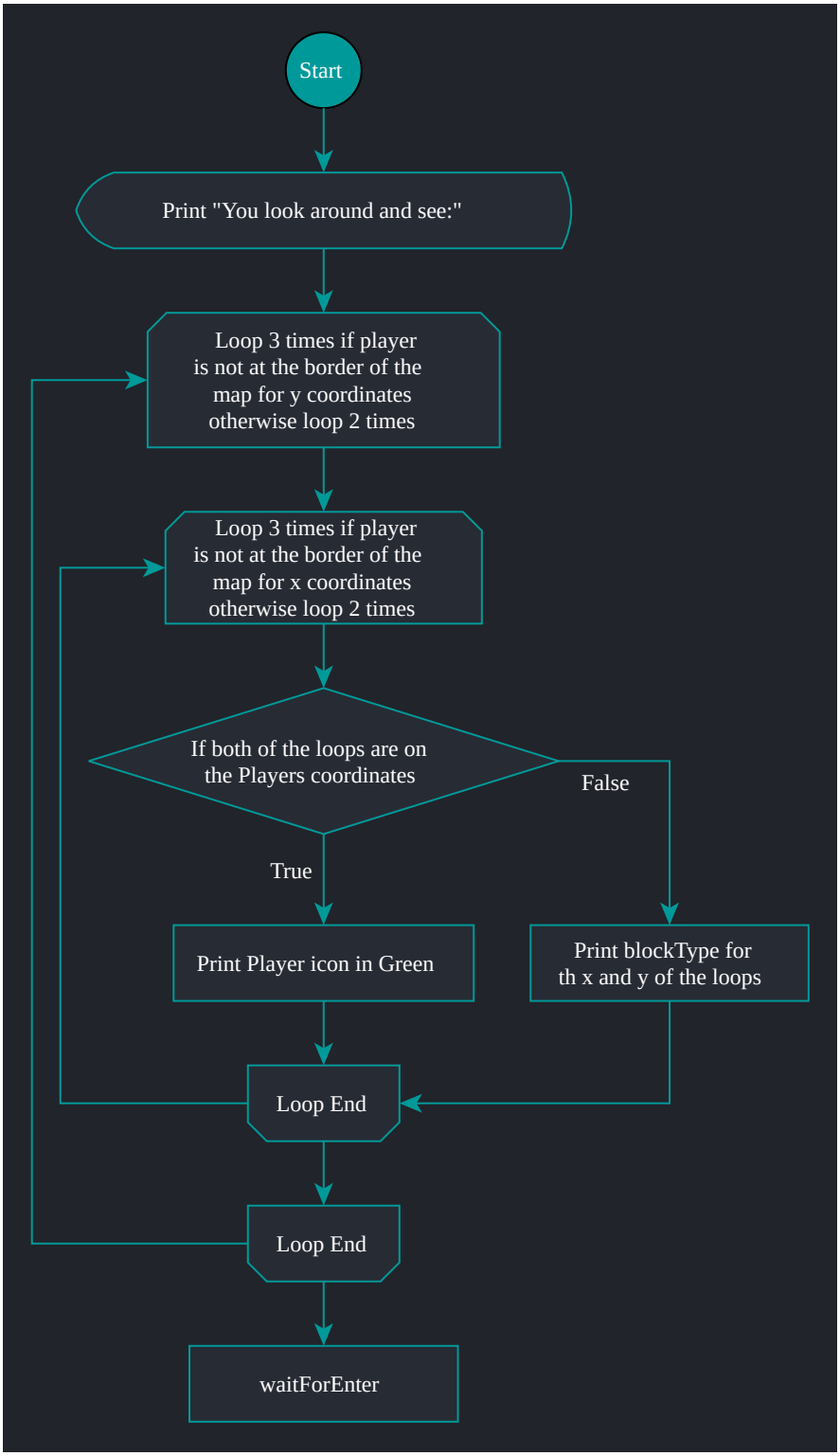
**Pseudocode**

```
BEGIN

PRINT INFO "You look around and see:";
FOR `<Integer> y` = `Maximum {of} 0 and {<Integer> playerY - 1}`;
`<Integer> y` <= `Minimum of {<Integer> playerY + 1} and {<Integer>
worldHeight - 1}`
    FOR `<Integer> x` = `Maximum of {0} and {<Integer> playerX - 1}`;
`<Integer> x` <= `Minimum of {<Integer> playerX + 1} and {<Integer>
worldWidth - 1}`
        IF `<Integer> x` == `<Integer> playerX` AND `<Integer> y` ==
`<Integer> playerY`
            PRINT INFO "P " (colored green);
        ELSE
            PRINT INFO `get block symbol from <two dimensional Integer
array> world @ indexes <Integer> x, <Integer> y`;
        Set `<Integer> x` += 1;
    PRINT INFO "\n";
    Set `<Integer> y` += 1;
PRINT INFO "\n";
Wait on player to press ENTER;

END
```

## Flowchart


Start
Print "You look around and see:"
Loop 3 times if player is not at the border of the map for y coordinates otherwise loop 2 times
Loop 3 times if player is not at the border of the map for x coordinates otherwise loop 2 times
If both of the loops are on the Players coordinates
False
True
Print Player icon in Green
Print blockType for th x and y of the loops
Loop End
Loop End
waitForEnter

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();
}
```

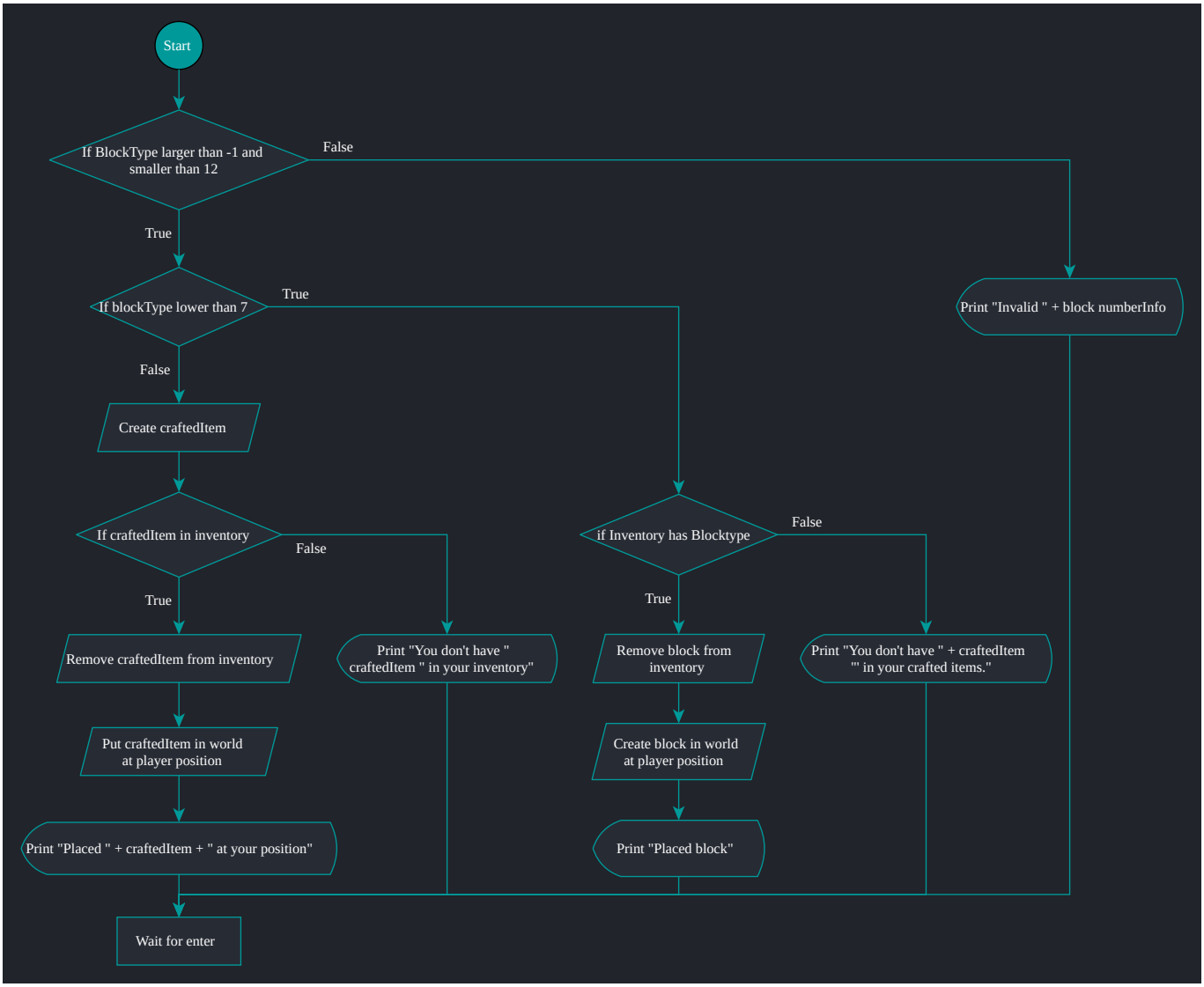
**Pseudocode**

```
BEGIN

IF `<Integer> blockType` >= 0 AND `<Integer> blockType` <= 11
    IF `<Integer> blockType` <= 6
        IF `<Integer list> inventory` contains `<Integer>` blockType
            Remove member `<Integer>` blockType from `<Integer list>
inventory`;
            Set `<two dimensional Integer array> world @ indexes <Integer>
playerX, <Integer> playerY` = `<Integer>` blockType;
            PRINT INFO "Placed " + `<String> get block name matching
<Integer> blockType` + " at your position.";
        ELSE
            PRINT WARNING "You don't have " + `<String> get block name
matching <Integer> blockType` + " in your inventory.";
    ELSE
        Assign `<Integer> craftedItem` = `<Integer> get crafted item of
<Integer> blockType`;
        IF `<Integer list> craftedItems` contains `<Integer>` craftedItem
            Remove member `<Integer>` craftedItem from `<Integer list>
craftedItems`;
            Set `<two dimensional Integer array> world @ indexes <Integer>
playerX, <Integer> playerY` = `<Integer>` blockType;
            PRINT INFO "Placed " + `<String> get block name matching
<Integer> craftedItem` + " at your position.";
        ELSE
            PRINT WARNING "You don't have " + `<String> get block name
matching <Integer> craftedItem` + " in your crafted items.";
ELSE
    PRINT WARNING "Invalid block number. Please enter a valid block
number.\n";
    PRINT WARNING `<String> BLOCK_NUMBERS_INFO` + "\n";
Wait on player to press ENTER;

END
```

## Flowchart


Start
If BlockType larger than -1 and smaller than 12
False
True
Print "Invalid " + block numberInfo
If blockType lower than 7
True
False
Create craftedItem
If craftedItem in inventory
False
True
Remove craftedItem from inventory
Print "You don't have " craftedItem " in your inventory"
Put craftedItem in world at player position
Print "Placed " + craftedItem + " at your position"
if Inventory has Blocktype
False
True
Remove block from inventory
Print "You don't have " + craftedItem "' in your crafted items."
Create block in world at player position
Print "Placed block"
Wait for enter

# Additional documentation

### addCraftedItem

```
public static void addCraftedItem(int craftedItem)
```

Adds a crafted item to craftedItems.

This method adds a crafted item to craftedItems that are part of the players inventory.

**Parameters:**

`craftedItem` - The crafted item

### craftedItemsContains

```
public static boolean craftedItemsContains(int craftedItem)
```

Queries craftedItems for an item.

This method queries the players crafted item inventory for an item.

**Parameters:**

`craftedItem` - The item to query the crafted item inventory for

**Returns:**

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

### displayWorld

```
public static void displayWorld()
```

Prints the world as ASCII text.

This method is responsible for displaying the world.

Part of secret door logic.

### generateEmptyWorld

```
private static void generateEmptyWorld()
```

Generates an empty world.

This method generates an empty world which only contains Sri Lanka's flag.

Part of secret door logic.

### 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

### 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

### 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

### 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.

### 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(String 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
- Flags API - API to get a flag