Project Template

Project Report: Group 66

Friday, October 20, 2023

Table of contents

Attribute	Details
Group Name	[group66]
Group Number	[66]
TA	[]

Student Name	Student ID
[Filippo Barbera]	[i6350569]
[Giannis Fourlas]	[i6343092]
[Maarten Koji]	[i6350359]
[Alexandra Plishkin]	[i6350941]

Overview of who did what

Filippo Barbera: Flowcharts and pseudocode for 4 game functions, description from 19-27 functions, flowchart for game workflow, secret door logic analysis, designed the FSA diagram and did it on the computer, committed in GitLab, interacted with API and drew flag, API documentation.
Giannis Fourlas: Flowcharts and pseudocode for 4 game functions, description from 28-36 functions, flowchart for game workflow, description of the FSA, designed the FSA diagram, committed in GitLab, collaborated with game code extension, interacted with API and drew flag, API documentation.
Maarten Koji: Flowcharts and pseudocode for 4 game functions, description from 1-9 functions, introduction, designed the FSA diagram, committed in GitLab, collaborated with game code extension, interacted with API and drew flag, extension code documentation.
Alexandra Plishkin: Flowcharts and pseudocode for 4 game functions, description from 10-18 functions, pseudocode for game workflow, designed the FSA diagram, created the GitLab branch group66, committed and uploaded the final documents in it, reviewed the report document, collaborated with game code extension, fixed bugs in game after implementation of new blocks/crafting recipes, interacted with API and drew flag, extension code documentation, flag drawing description.

1 Introduction

JavaCraft is a simple player-interactive game that models the popular 2009 game, Minecraft. Composed of 35+ java functions, JavaCraft produces an open world that allows the player to undertake a variety of actions that allow them to progress in the game. This report analyzes the game code and its functionalities through the use of rigorous explanations, flowcharts, and pseudocode, and extends the game code by introducing new block types and crafting recipes. A particular focus in this investigation lies in the secret door/area that is interwoven throughout, that requires a series of specific actions from the player in order to be unlocked.

2 JavaCraft's Workflow

Flowchart For Game: (In appendix (bigger version in GitLab branch group66))

Pseudocode For Game: (In appendix)

3 Functionality Exploration

Detailed description for each function:

1. [initGame]: Key variables are declared and initialized. These include the world width and height, the size of the world, player X and Y, and the inventory. The method declares static variables, with the integer parameters of the world width and world height.
2. [generateWorld]: Generates a random value and relies on probability density to set each coordinate (using two for loops) in the world grid to a particular block (wood, leaves, stone, iron ore, and air), with air being the most prominent, occurring 70% of the time.
3. [displayWorld]: Displays world borders through an equation that uses the world width and height values, as well as printing "World map:" in cyan. Also sets the character "p" to represent the player, with the color of the character changing based on if the player is in the secret area or not, which is accomplished through if-else-if statements.
4. [getBlockSymbol]: Sets the character of air to a hyphen, assigns colors to all the blocks through the use of a switch statement. Calls the getBlockChar method, and returns it with the parameter blockType of type int.
5. [getBlockChar]: Sets the shade of all the other blocks, by assigning them a specific Unicode character, again with a switch statement, considering 4 cases and a default.
6. [startGame]: Calls a multitude of static public methods to produce the backbone of the game. Also includes the code to unlock the secret door and enter the secret area, which are discussed in detail in a further section.
7. [fillInventory]: Clears the inventory, adds each blockType (1-4) to the inventory through a for loop.
8. [resetWorld]: Calls the static generateEmptyWorld method to print the secret area, and initializes the player's X and Y coordinates.
9. [generateEmptyWorld]: Prints the layout of the secret area, by dividing the grid in 3, with each division being assigned the color of red, white, and blue blocks in order, with double for loops to fill the 2D grid.
10. [clearScreen]: This function clears the user's screen, making a distinction between windows and non-windows systems. Therefore, it checks for the type of system and uses specific commands to clear the screen, for windows: (cmd, cls, /c) and for another operating system: (\033,[H\033[2J), which are ANSI escape codes. So, it allows you to clear the screen of a device with any operating system.
11. [lookAround]: This function is used to look around in the game. So, if the player types "look", it would tell the player which are the blocks around. For this, assigns *x* and *y* as integers values related to the

world's disposition, after this, those values are determined to playerY and playerX. And uses a loop with some mathematical equations to determine exactly the position in the world and what's around the player.

12. [movePlayer]: It's what allows the player to move through the map of the game. For this, it defines the keys and words to move the player in certain directions. It defines a string variable "direction", which is therefore used in a switch case with all the possible movements for the player. Also, in each case, there's an if statement that determines if the player is allowed to move to that direction or if he/she already arrived at the border of the map.
13. [mineBlock]: Allows the player to mine the blocks into the game. It uses an integer variable "blockType", which is related to the player position in the world. It determines if the type of block the player is trying to mine is not air, and if it's not air, the player is able to mine it and the block is removed from the map and substituted by air; on the other hand, if it's air, a message will pop up saying that there's no block to mine there.
14. [placeBlock]: Allows the player to place the mined blocks and crafted items the player from their inventory. Using the integer variable "blockType", determines the block's value. If the value is >=0 and <=7, there are three possibilities; this is determined by if-else statements. In case <=4, it checks if the inventory contains the block, if present, it's removed from the inventory and placed in the world, at playerX/playerY; otherwise, it informs the player of the absence. If it's >4, it would generate an integer value "craftedItem", related to the block type; if that crafted item is in your inventory designated as "Crafted items", it's removed from the inventory and placed in the world; on the other side, it notifies the player of the absence. For any other integer typed, the program will say that it's invalid, providing the player with block numbers information.
15. [getBlockTypeFromCraftedItem]: It's used to get the block type from a crafted item. It uses the integer variable "craftedItem" and a switch-case to return integer values that are designated to each "blockType".
16. [getCraftedItemFromBlockType]: It's used to get a crafted item from a block type. It uses the integer variable "blockType" and a switch-case to return integer values that are designated to each "craftedItem".
17. [displayCraftingRecipes]: This function displays the crafting recipes on a list to inform the player of what's allowed to be crafted and with which materials. It simply uses the print output to display the list.
18. [craftItem]**:** allows the players to craft items using what they have previously mined. Using a switch statement, it gives the chance to choose between the three different recipes that are then developed in the following functions. If the player's input is not valid a message informs the player.

19. [craftWoodenPlank]**:** it runs when the player chooses the first recipe in the previous function, but an if statement makes the presence of two items of WOOD in the inventory necessary to make it work. If this is true, WOOD items are removed from the inventory and CRAFTED_WOODEN_PLANKS is added. If is false, a message informs the player of the lack of resources.
20. [craftStick]**:** it runs when the player chooses the second recipe in the previous function, but an if statement makes the presence of one necessary item of WOOD in the inventory. If is true, WOOD item is removed from the inventory and CRAFTED_STICK is added. If is false, a message informs the player of the lack of resources.
21. [craftIronIngot]**:** it runs when the player chooses the third recipe in the previous function, but an if statement makes the presence of three necessary items of iron ore in the. If is true, iron ore items are removed from the inventory and CRAFTED_IRON_INGOT is added. If is false, a message informs the player of the lack of resources.
22. [inventoryContains]: this function checks the items in the inventory and return true if there's an item.
23. [inventoryContains]: this function sets the item count to 0, to use later a for statement that will go through all the items in the inventory. If the item is in the inventory, it's counted, and if item count is equal to requirements for a crafting recipe, returns true, if not it returns false.
24. [removeItemsFromInventory]: this function regards the inventory. It takes two integers as parameters, "item" and "count". The items will be counted by "count" and selected by "item". The variable removedCount is responsible for tracking the number of removed items.
25. [addCraftedItems]: this function has a parameter called "craftedItem", and it contains items crafted by player. If it's empty, the program creates an Arraylist to store the crafted items and adds the craftedItem to the craftedItems.
26. [interactWithWorld]: this function allows the player to interact with the world around him using a switch statement. Each case regards a different block (wood, leaves, stone, iron ore, air and empty blocks), the system will print a different message for each. For example, when player interacts with wood, system prints "you gather wood from the tree", and if the block isn't recognized, system prints "unrecognized block. Cannot interact". The system waits for the player to type "enter" to confirm the operation.
27. [saveGame]: this function saves the game state to a specific file. This function includes width and height of new world, player position, inventory contents, crafted items and unlock mode. These are written into the file. If the game state has been saved, it prints a message to inform user. Otherwise, an error message appears. To ensure that the user has seen the message, the user must press the "enter" button.
28. [loadGame]: this function loads the game state from a specific file. Then, it deserializes the game state from the specific file and loads it into the program. The deserialized data are width and height for new

world, player position, inventory contents, crafted items and unlock mode. If the game state loads successfully, it prints a message. Otherwise, it prints an error message. To ensure that the user has seen the message, the user must press the “enter” button.

29. [getBlockName]: this function has as a parameter an integer “blockType” and returns a string with the name of the block. This function uses a switch statement to understand the “blockType”. If the user enters the number corresponding to a block, then the program returns its name. If the user doesn’t enter the number of any known “blockType”, then program returns “Unknown”.
30. [displayLegend]: this function prints the legend of different block types. These symbols \u00A7\u00A7 represent the symbols that are printed with the different block types, helping users to identify them in the game. This function provides information, such as appearance of empty blocks, wood blocks, etc.
31. [displayInventory]: this function displays the player’s inventory. If the inventory is empty, then prints “Empty” in yellow, otherwise it counts the different block types and prints a list of them with number of occurrences. After that program prints the crafted items. If there are none, then the program displays in yellow color “None”. Otherwise, prints name and colour of it. At the end, an empty line printed.
32. [getBlockColor]: this function takes an integer as an input and returns an ANSI colour. It is used to apply different colours to the blocks.
33. [waitforEnter]: this function waits for the user to press the “enter” button. It outputs the message “Press Enter to continue”. When pressed, program moves to the next step.
34. [getCraftedItemName]: this function takes as an input an integer craftedItem and returns a string with the name of the crafted item. If it’s known, it returns a string, otherwise it returns “Unknown”.
35. [getCraftedItemColor]: this function takes as an input an integer craftedItem and returns the ANSI colour. If it’s unknown, it returns an empty string.
36. [getCountryAndQuoteFromServer]: this function makes a request to the server and extracts specific data. It receives a JSON response. If an error occurs during the process, the program outputs a message.

4 Finite State Automata (FSA) Design

- Secret Door Logic Analysis:

Studying the source code of JavaCraft, it’s possible to see that there is a sequence of commands hidden in it, that allows the player to access a secret area of the map.

This specific feature requires many functions to work, and the very first one is called unlockMode. This is a Boolean that is first set as false, and to make it true the player needs to type the command “unlock”. Once that’s done, there are three more commands that need to be entered by the player, which are “c” (for craft), “m” (for

mine) and any movement command (w, a, s, d). The order in which they are given is not relevant, but there is one command that must be entered last, to get the sequence right. “Open” must be the very final command, if it is entered before the others the Boolean value will turn back to false and the player will have to start over with the sequence. If this is the case the system will print “Invalid passkey. Try it again”.
If all commands have been given in the right order, the secret door will be unlocked, and the player will read “Secret door unlocked!”.
The screen will now be cleared, and the player will see the Dutch flag appear on the screen with the following messages:
“You have entered the secret area!” and “You are now presented with a game board with a flag!”.

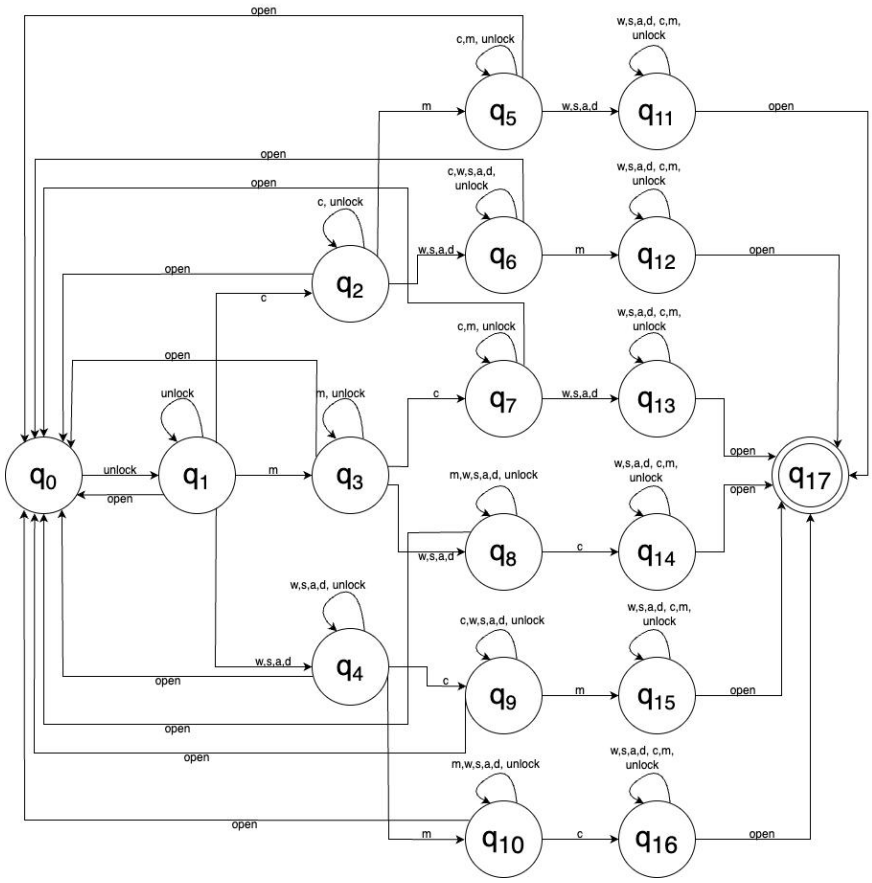
- FSA Illustration & Description:

The secret door is unlocked only when the user enters specific commands. More specifically, there are 18 different states in the FSA we designed. Our FSA is a Non-deterministic Finite Automaton (NFA) as the transition of states can be to multiple next states for each input symbol, there’s not exactly one transition defined for each symbol in Σ at every state.
The alphabet Σ of the FSA contains “unlock” for unlocking the process to unlock the secret door; “c” is for crafting action, “m” for mining action and “w, a, s, d” for moving action; finally, “open” is for trying to open the secret door.
The specific transitions between the states to reach the accepting state are the following:

- From **Locked (q0)** which is the initial state to **UnlockProcess (q1)** when user enters unlock.
- Now, it goes from **UnlockProcess (q1)** to **Action0 (q2-q4)** when the user enters one of the actions (craft, mine, move).
- From the **Action0 (q2-q4)** state to **Action1 (q5-q10)** when the user enters one of the actions (the action must be different from the action entered firstly to change state).
- From **Action1 (q5-q10)** to **Action2 (q11-q16)** when the user enters one of the actions (the action must be different from the two actions entered before to change state).
- From **Action2 (q11-q16)** to **Open (q17)** when the user attempts to open the secret door.

The accepting state in the finite state automata is the open state, where the secret door is unlocked.
When the NFA reaches this state, it means that the secret door has been successfully unlocked.
The user must follow this sequence of state changes for reaching the accepting state, if the user enters “open” before completing the sequence, the NFA returns to initial state.

<u>FSA Diagram:</u>



5 Git Collaboration & Version Control

- Repository Link: https://gitlab.maastrichtuniversity.nl/bcs1110/javacraft.git
- Branch Details: Branch group66
    - Members: Alexandra Plishkin, Maarten Koji, Giannis Fourlas and Filippo Barbera.
    - Documents in the branch: Code (JavaCraft.java), Report (Report_Group66.docx), Instructions (README.md), Flowchart (Flowchart_Game_Workflow.png), Provisional Report (Provisional_Report_Group66.pdf) and Final Report (Final_Report_Group66.pdf).

In the branch we added the report and uploaded our code. We managed to handle the conflicts and upload our changes to the files. Each member committed at least once.

6 Extending the Game Code

We added two new types of blocks: “Diamond” and “Cute animal”. Also, we added a new crafting recipe: magical animal. When you add a diamond, an iron ingot, and a cute animal; the result is a “Magical Animal”.

Blocks Explanation & Implementation Methods:

Creation of 2 additional block types involved two procedures to obtain functioning block types that appeared in the game when we ran it. The first procedure involved defining the key characteristics of the blocks. These include the probability of occurrence on the world map, the Unicode characters, and others (specified in data table below). The second procedure involved the extension of all methods requiring the index of block types as parameters, or the iteration over all the blockTypes, in arrays and loops. Thus, we extended the values from 5 to 7, with the additional indices corresponding to the created blockTypes.

Specifying new block type example: Specifying the probability of generation in the world map in the generateWorld() method, line 105”

```
} else if (randValue < 75) {
  world[x][y] = DIAMOND;
} else if (randValue < 80) {
  world[x][y] = ANIMAL;
```

BlockType iteration extension example: Size of array created in blockCounts to specify all block types. From the displayInventory() method, line 687:

```
int[] blockCounts = new int[5];
```

```
int[] blockCounts = new int[7];
```

Data table:

	Block 1: Diamond	Block 2: Animal
Probability of block occurrence [generateWorld()]	5%	5%
Index/block-number [javaCraft()]	5	6
Colour [getBlockColor()]	ANSI_PURPLE	ANSI_BROWN
Unicode char [getBlockChar()]	\u2591	\u00A9
Interactive text display [interactWithWorld()]	"You found DIAMOND!! You're rich."	"You want to adopt that animal..."
Name [getBlockName()]	“Diamond”	“Cute animal”
Legend [displayLegend()]	“\u2591\u2591 - Diamond block"	“\u00A9\u00A9 - Cute animal”

Crafting Recipe Explanation & Implementation Methods:

The crafting recipe is CRAFTED_MAGICAL_ANIMAL, its name is “Magical Animal”, and its colour is red. We started including it in the block numbers information, in the getBlockTypeFromCraftedItem and getCraftedItemFromBlockType functions, and in the displaying of the crafting recipes.

So on, we created a function craftMagicalAnimal and we called it in craftItem for allowing the player to craft the item when entering number “4” (the one assigned to this crafting recipe). In craftMagicalAnimal we check if inventory contains the necessary items, then we remove each of these from it and we add the crafted item to crafted items and game prints "Invoked Magical Animal."; if inventory doesn’t contain the necessary items, game prints "Insufficient resources to invoke Magical Animal.". As this recipe requires 3 different items, we added another inventoryContains which checked if the inventory had each of these items and returned true in that case. Also, if you place the magical animal, you are allowed to interact with it, the game will tell you that you are becoming good friends!

We realized there was a getBlockChar function and implemented getCraftedChar function to also obtain a character for the crafted item, we used Unicode values for representing the ASCII characters displayed in the map. Also, while implementing this recipe, we faced some bugs when running the game, as we weren’t able to place the crafted item in the world, we noticed that any crafted item could be placed. So, we started including getCraftedItemColor and getCraftedItemChar in the function getBlockSymbol, so the crafting item will have a symbol in the game. Now, in the placeBlock function we checked if the blockType was a crafted item (checking if it was >= 7), then we check if crafted items contained the crafted item and if that’s the case it will be removed from your crafted items inventory and placed in the world with a message "Placed (name of crafted item) at your position.".

7 Interacting with Flags API

The Flags API's functionality is to provide flag information. The game uses this API to get flag information based on parameters. The API responds with a JSON data containing country and quote details. The flag drawing process integrates this API response into the game by extracting and displaying the country and quote on the game's user interface. More specifically, the game sends an HTTP POST request to the flags API, having group number, group name, and difficulty level. The request is sent to the API, and then the API processes this request. After that, it generates a response. It gets flag information from the database based on parameters. Then, it returns a JSON response containing the country and quote data. Then, the game receives the JSON response from the API and extracts the country and quote information, which are displayed on the game.

The flag we had to implement was Japan and the difficulty level was hard. For the implementation we combined automation and manually plotting into the map. Firstly, we changed the map size to 50x30, for this we copied the displayWorld() function and created displayWorldinSecretArea() where we used the new world sizes to display the map, then we simply did an if statement which checks if you are in the secret area and displays the corresponding map size. Then, we also modified resetWorld() to situate the player in the middle of the new map.

Finally, for drawing the flag we divide the height in 3 parts (size 10) and the width in 5 parts (size 10). So, firstly we will iterate through each block in the first height (0-10) and fill all the line with white blocks, doing the same for the two remaining height parts (10-20 and 20-30). Now, as the Japanese flag has a red circle in the middle, we had to firstly fill a square in the centre of the flag, so we iterate in the width (10-40) and in the height (5-25) to create a red square of 20x30. Finally, we replaced manually the corners of the flag with white blocks to obtain a shape which is more like a circle.

8 Conclusion

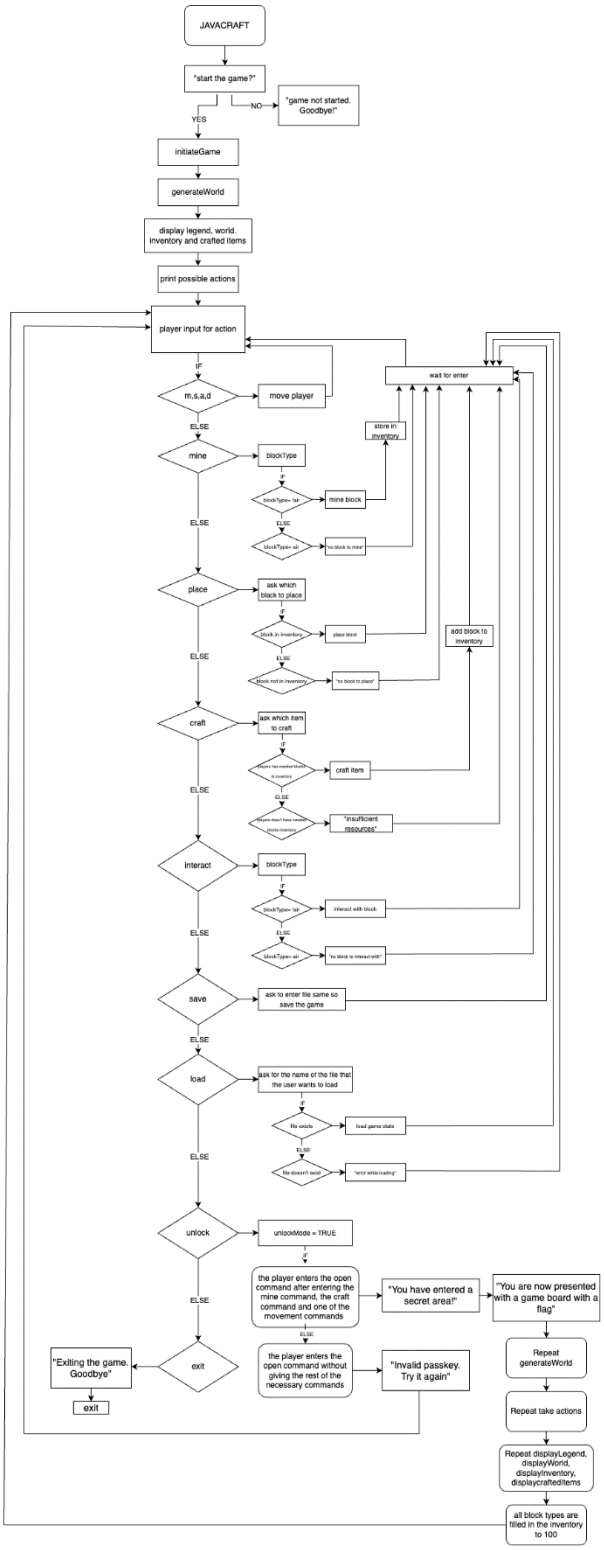
We believe that we have successfully achieved our aim of exploring the JavaCraft game code, understanding its functionalities and extending it with additional crafting recipes and block types. We designed flowcharts and pseudocode to understand the logical breakdown behind more complicated methods. We also produced a NFA to understand the logical sequencing behind the game's secret door mechanism.

One of the main challenges we faced was the difficulty in the flag drawing, generated by the API. As we had chosen a relatively complicated flag design, we had to find a solution that was both efficient, and produced a design that matched Japanese flag. We decided to go for a more efficient method than filling each block individually, as described above. Thereby we learnt the importance of putting in more effort to find more elegant solutions as opposed to attempting to create the design by force.

Throughout this project we achieved a higher-level understanding of not just the JavaCraft code but of code in general, how to present code, and how to extend it accordingly. We understood the different mechanisms involved in the production of it, such as the use of loops, arrays, and different character types. As we conclude this project, we aim to further our understanding of these coding mechanisms in Java.

9 Appendix

Flowchart Game Workflow:


JAVACRAFT
"start the game?"
NO
"game not started. Goodbye!"
YES
initiateGame
generateWorld
display legend, world, inventory and crafted items
print possible actions
player input for action
IF
m,s,a,d
move player
wait for enter
ELSE
mine
blockType
store in inventory
mine block
"no block to mine"
place
ask which block to place
place block
add block to inventory
"no block to place"
craft
ask which item to craft
craft item
"insufficient resources"
interact
blockType
interact with block
"no block to interact with"
save
ask to enter file name so save the game
load
ask for the name of the file that the user wants to load
file exists
load game state
"error while loading"
unlock
unlockMode = TRUE
the player enters the open command after entering the mine command, the craft command and one of the movement commands
"You have entered a secret area!"
"You are now presented with a game board with a flag"
the player enters the open command without giving the rest of the necessary commands
"Invalid passkey. Try it again"
Repeat generateWorld
Repeat take actions
Repeat displayLegend, displayWorld, displayInventory, displaycraftedItems
exit
"Exiting the game. Goodbye"
exit
all block types are filled in the inventory to 100

Pseudocode Game Workflow:

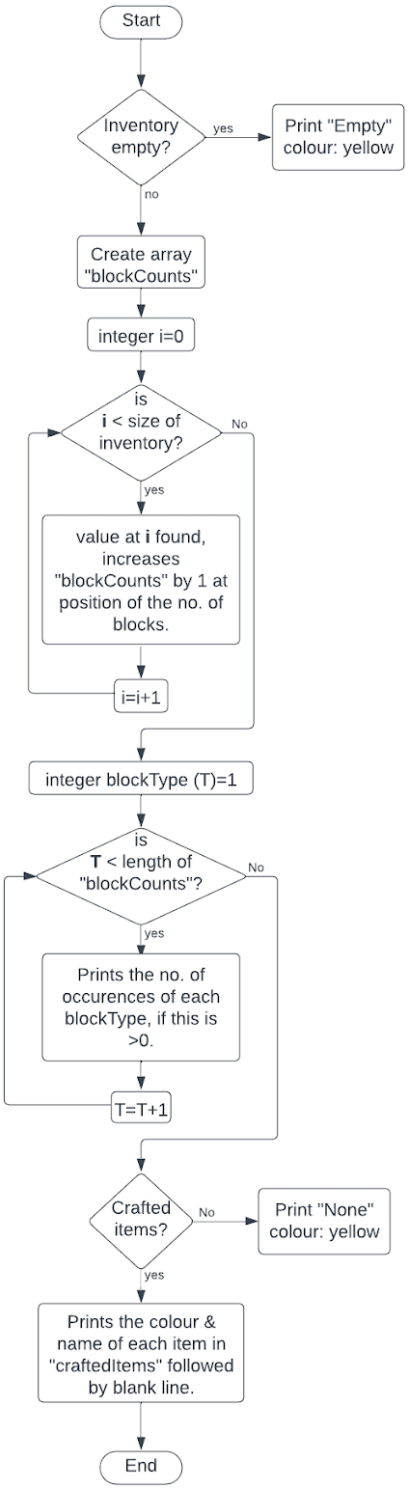
```
[
Print (Welcome + instructions)
Print(Start?)
        If (no)
                Print(Game not started. Goodbye!)
                Exit/Finish
        If (yes)
                Initiate game
                Generate world
                Display legend, world, inventory and crafted items
                Print possible actions
                Start game

                        If action is equal w, a, s, d
                                Move: w (up), a (left), s (down), d (right)
                        If action is equal m
                                If block in position is not air
                                        Mine block in position
                                        Store in inventory
                                Else
                                        Print(No block to mine)
                        If action is place p
                                Ask which block want to place
                                        If block in inventory
                                                Place block
                                        Else
                                                Print(No block to place)
                        If action is craft c
                                Print crafting recipes
                                Ask what wants to craft
                                        If player has the blocks needed
                                                Craft block
                                                Add block to inventory
                                        Else
                                                Print(Insufficient resources)
                        If action is interact i
                                If the block in position is not air
                                        Interact with block
                                Else
                                        Print(No block to interact with)
                        If action is save:
                                Ask for enter file name to save game state
                        If action is load:
                                Ask for enter file name that user wants to load
                                If file name exists
                                        Load game state
                                Else
                                        Error while loading
```

If action is unlock:
If User enters w, a, s, d and enters c and enters m and enters open
Secret door unlocks
If secret door unlocks
Repeat generate world
Repeat take actions
Repeat display legend, world, inventory and crafted items
All block types are filled in the inventory to 100

If action is exit:
Print(Exiting the game. Goodbye!)
Exit

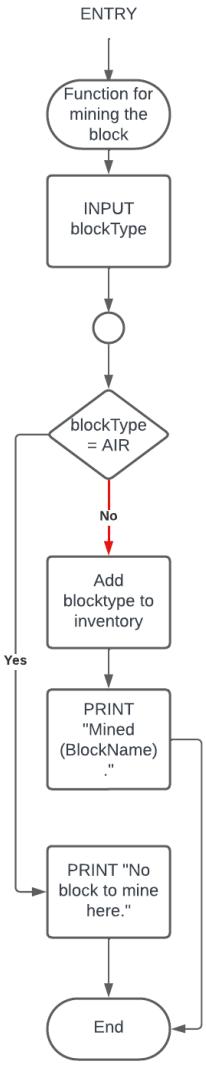
]



1. Flowchart & pseudocode displayInventory:

If the inventory is empty, print ‘Empty’ in *yellow*.
Create an array that lists the number of blocks in the inventory.
For all elements in the inventory,
    Set the integer ‘block’ to the value of that element,
    add 1 to the block type specified by ‘block’ in the array ‘blockCounts’.
For all elements in ‘blockCounts’,
    Set the integer ‘occurrences’ to the element,
    If ‘occurrences’ >0, print the name of the block along with the occurrences.
Print “Crafted Items:”
If there are none, print “None” in *yellow*.
Else, for all items in the ‘craftedItems’ array, print the color and name of the item.
Print 2 empty lines.
End

2. Flowchart & pseudocode mineBlock:



Void mineBlock

Define the blockType within the playerX and playerY

If the block is not air

Add the blockType

Substitute the block type into air

Print (“Mined (blockType) .”

Else

Print “No block to mine here.”

Wait for the player to press Enter

End

3. Flowchart & pseudocode placeBlock:

Void placeBlock

int blockType

If blockType>=0 and <=7

If blockType <=4

If inventory contains blockType

Remove blockType from inventory

Place blockType in the world within the playerX and playerY

Print (“Placed (blockType) at your position.”)

Else

Print (“You don’t have (blockType) in your inventory)

Else

int craftedItem (get the crafted item from the blocktype)

If craftedItems contains the craftedItem

Place blockType in the world within the playerX and playerY

Print (“Placed (craftedItem name) at your position.”)

Else

Print (“You don’t have (craftedItem name) in your crafted items.”)

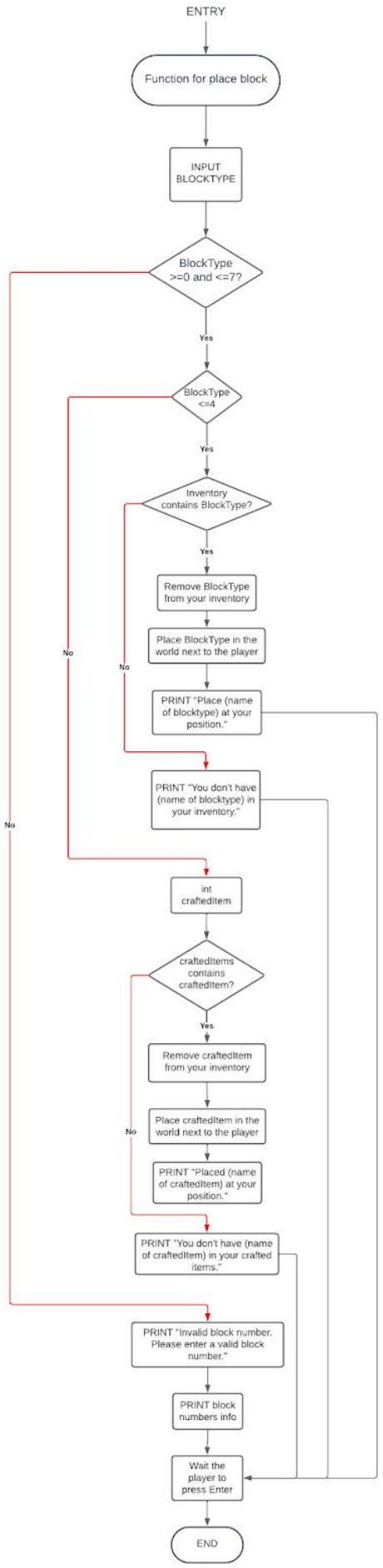
Else

Print (“Invalid block number. Please enter a valid block number.”)

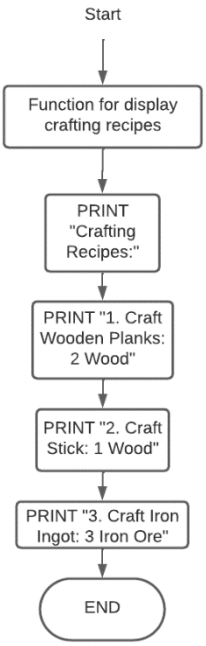
Print block numbers info

Wait for the player to press Enter

End

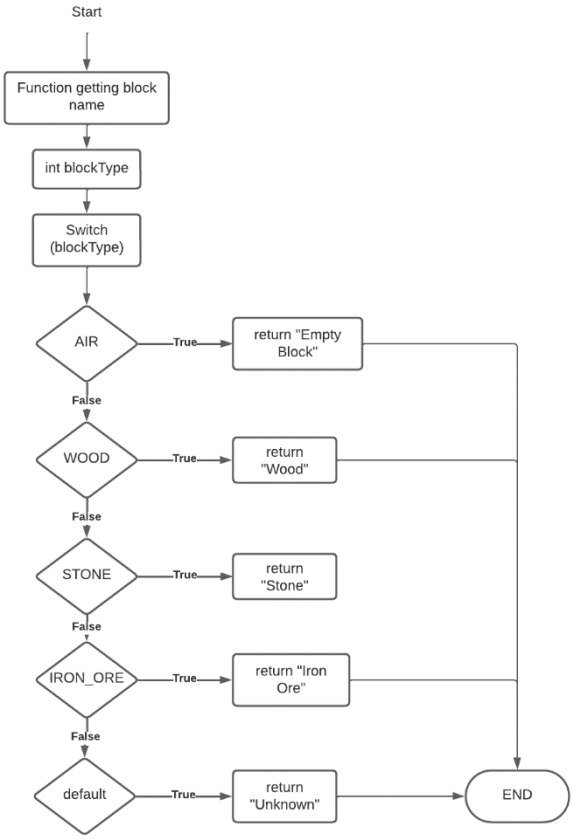


4. Flowchart & pseudocode displayCraftingRecipes:



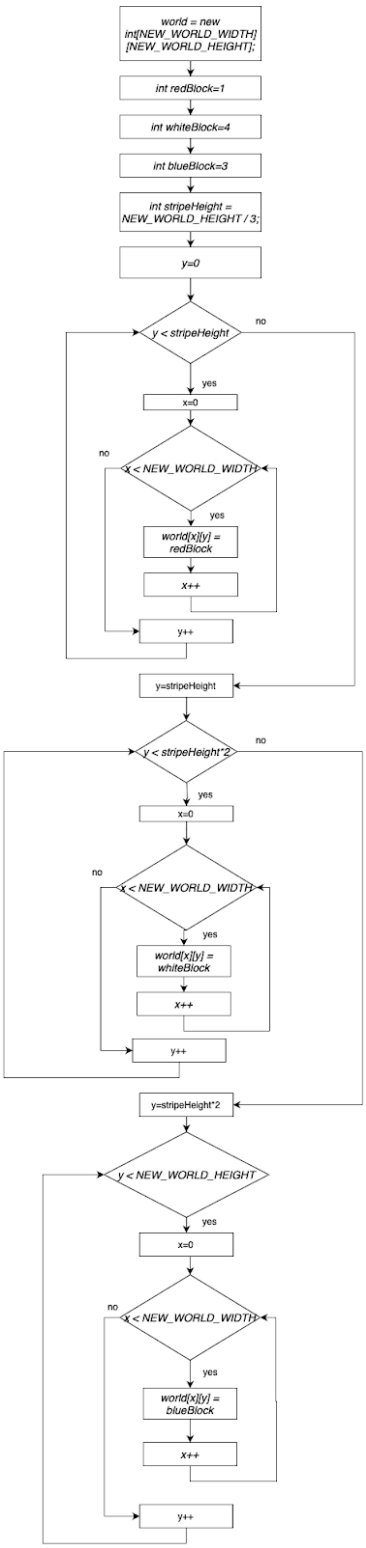

```
Void displayCraftingRecipes
    Print (“Crafting Recipes.”)
    Print (“1. Craft Wooden Planks: 2 Wood”)
    Print (“2. Craft Stick: 1 Wood)
    Print (“3. Craft Iron Ingot: 3 Iron Ore”)
    End
```

5. Flowchart & pseudocode getBlockName:



String getBlockName
int blockType
Switch (blockType)
    Case AIR:
        Return “Empty Block”
    Case WOOD:
        Return “Wood”
    Case STONE:
        Return “Stone”
    Case IRON_ORE:
        Return “Iron Ore”
    Default:
        Return “Unknown”
    End

6. Flowchart & pseudocode generateEmptyWorld:



function generateEmptyWorld():

create an empty 2D array called world with dimensions NEW_WORLD_WIDTH by NEW_WORLD_HEIGHT

set redBlock to 1

set whiteBlock to 4

set blueBlock to 3

set stripeHeight to NEW_WORLD_HEIGHT divided by 3 // Divide the height into three equal parts

// Fill the top stripe with red blocks

for y from 0 to stripeHeight - 1:

for x from 0 to NEW_WORLD_WIDTH - 1:

set world[x][y] to redBlock

// Fill the middle stripe with white blocks

for y from stripeHeight to (stripeHeight * 2) - 1:

for x from 0 to NEW_WORLD_WIDTH - 1:

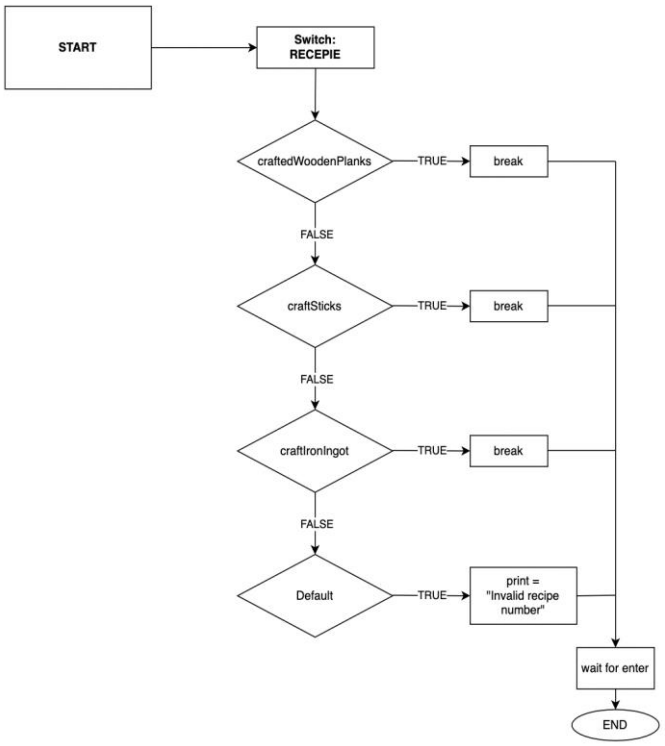
set world[x][y] to whiteBlock

// Fill the bottom stripe with blue blocks

for y from (stripeHeight * 2) to NEW_WORLD_HEIGHT - 1:

for x from 0 to NEW_WORLD_WIDTH - 1:

set world[x][y] to blueBlock



7. Flowchart & pseudocode craftItem:

void CraftItem

Switch: four possible cases

Case 1: call craftWoodenPlanks function

Case 2: call craftSticks function

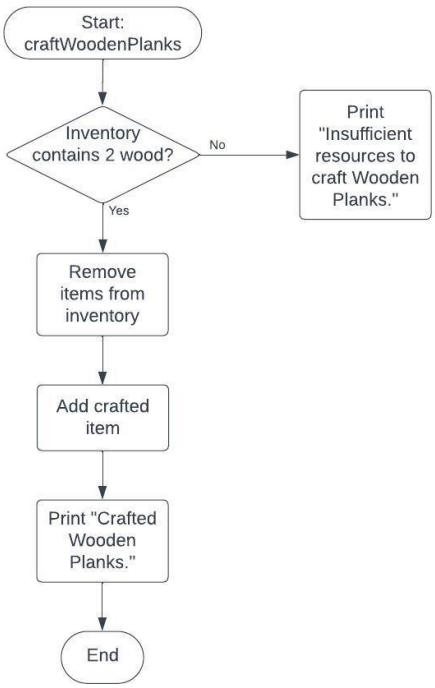
Case 3: call craftIronIngot

default: print “invalid recipe number”

Wait for the player to write Enter

End

8. Flowchart & pseudocode craftWoodenPlanks:



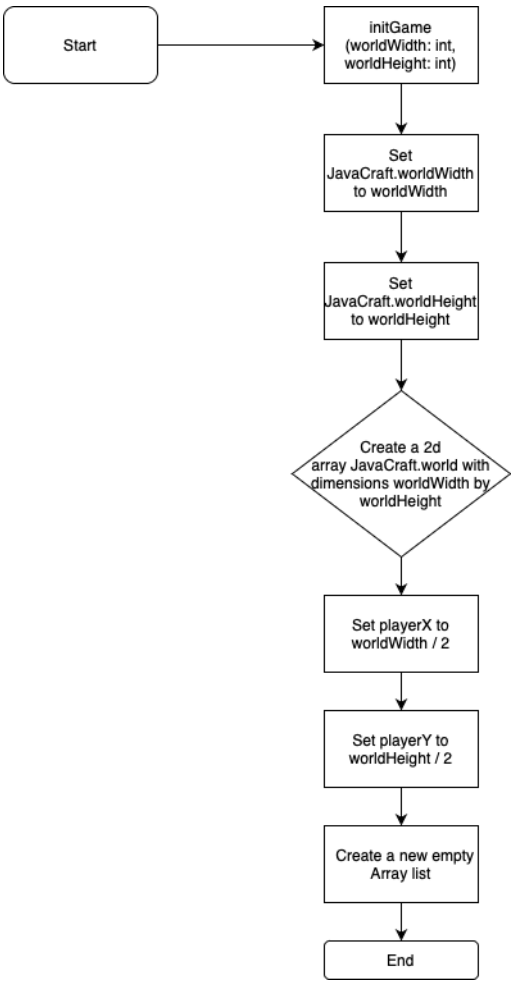
If the inventory does not contain 2 wood, print "Insufficient resources to craft Wooden Planks."

Else, remove the necessary items from inventory,

Add the crafted wooden plank,

Print "Crafted Wooden Planks."

End



9. Flowchart & pseudocode initGame;

function initGame(worldWidth: int, worldHeight: int):

Set JavaCraft.worldWidth to worldWidth

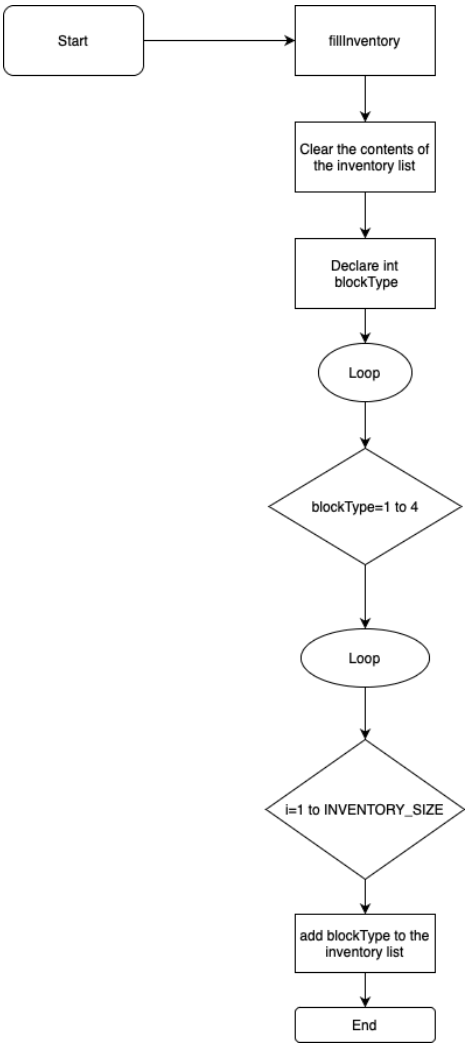
Set JavaCraft.worldHeight to worldHeight

Create a new 2D array JavaCraft.world with dimensions worldWidth by worldHeight

Set playerX to worldWidth / 2

Set playerY to worldHeight / 2

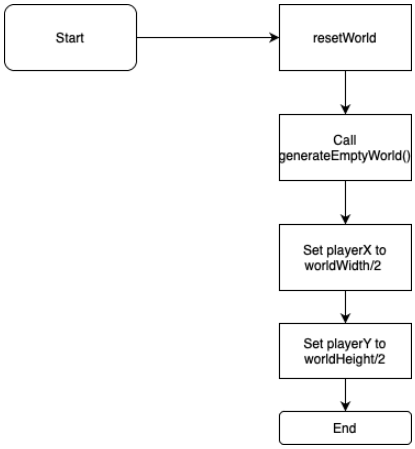
Create a new empty Array list



10. Flowchart & pseudocode fill_Inventory;

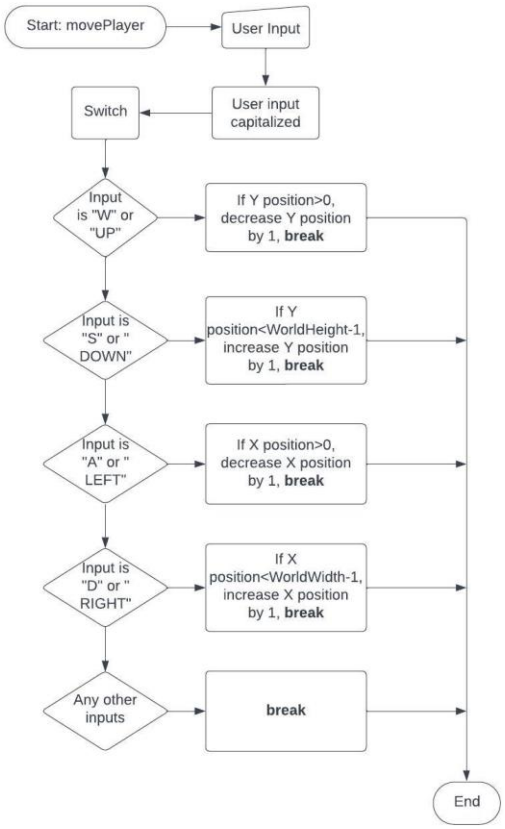
```
fillInventory()
    Clear the contents of the inventory list
    loop for blockType from 1 to 4
      loop for i from 1 to INVENTORY_SIZE
        add blockType to the inventory list
      End loop
    End loop
End
```

11. Flowchart & pseudocode resetWorld:




```
resetWorld()
    Call generateEmptyWorld()
    Set playerX to worldWidth/2
    Set playerY to worldHeight/2
End
```

12. Flowchart & pseudocode movePlayer:



Take user input

Capitalize user input

5 cases:

Case 1: Input is W / UP

If Y position>0, decrease Y position by 1. Terminate.

Case 2: Input is S / DOWN

If Y position<WorldHeight-1, increase Y position by 1. Terminate.

Case 3: Input is A / LEFT

If X position>0, decrease X position by 1. Terminate.

Case 4: Input is D / RIGHT

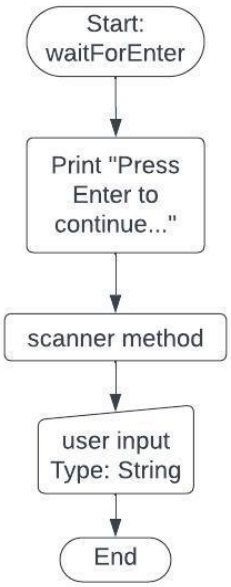
If X position<WorldWidth-1, increase X position by 1. Terminate.

Case 5: Any other inputs

Terminate.

End

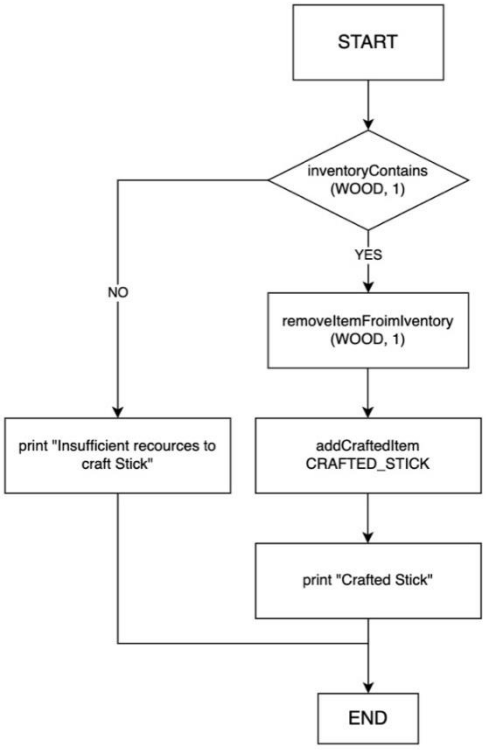
13. Flowchart & pseudocode waitForEnter:



Print “Press Enter to continue…”

Read user input, type: String.

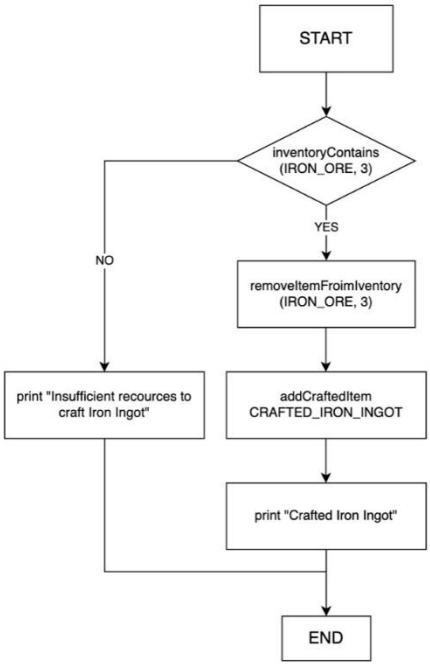
End.



14. Flowchart & pseudocode craftSticks:

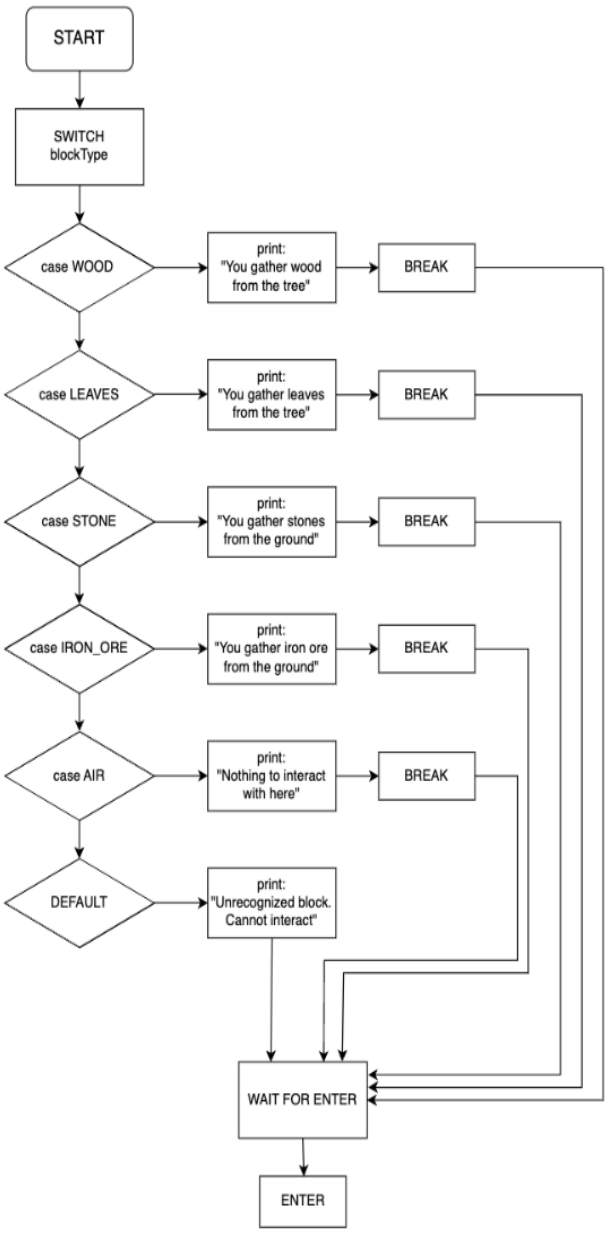
If the inventory contains one items of WOOD
    remove one items of WOOD from the inventory
    add to the inventory one items of CRAFTED_STICK
    print “Crafted Stick.”
Else
    print “Insufficient resources to craft Stick.”

15. Flowchart & pseudocode craftIronIngot:



If the inventory contains two items of IRON_ORE
    remove three items of IRON_ORE from the inventory
    add to the inventory one items of CRAFTED_IRON_INGOT
    print “Crafted Iron Ingot.”
Else
    print “Insufficient resources to craft Iron Ingot.”

16. Flowchart & pseudocode interactWithWorld:



define the type of the block that the player is on

switch: six possible cases depending on the type of block

case WOOD:

print “you gather wood from the tree”

add to the inventory one item of LEAVES

break

case STONE:

print “you gather stone from the ground.:”

add to the inventory one item of STONE

break

case IRON_ORE:

print “you gather iron ore from the ground.:”

add to the inventory one item of IRON_ORE

break

case AIR:

print “Nothing to interact with here.:”

break

case DEFAULT:

print “Unrecognized block. Cannot interact.:”

Wait for the player to type “ENTER”.

10 References