JavaCraft Provisional Report

Project Report: Group 78

Sunday, October 8, 2023

Table of contents

Attribute	Details
Group Name	Group 78
Group Number	78
TA	N/A

Student Name	Student ID
Andrei Visoiu	I6365974
Jan Ebenritter	I6357409
Armanto Tsollakou	I6349552
Vasileios Rallis	I6356800

1 Introduction

Welcome to our code report on JavaCraft, a game similar to minecraft, which is fully terminal based and easy to play. It allows the user to move around in a small world, mine elements, craft new items and look for the secret door to enter a new world.

This report serves as a structured analysis, breaking down the complexities of JavaCraft into understandable chunks.

1. Flowcharts and Pseudocode: Everybody contributed
2. Functionality exploration: Done by Andrei
3. FSA: Majority done by Andrei with support from Jan

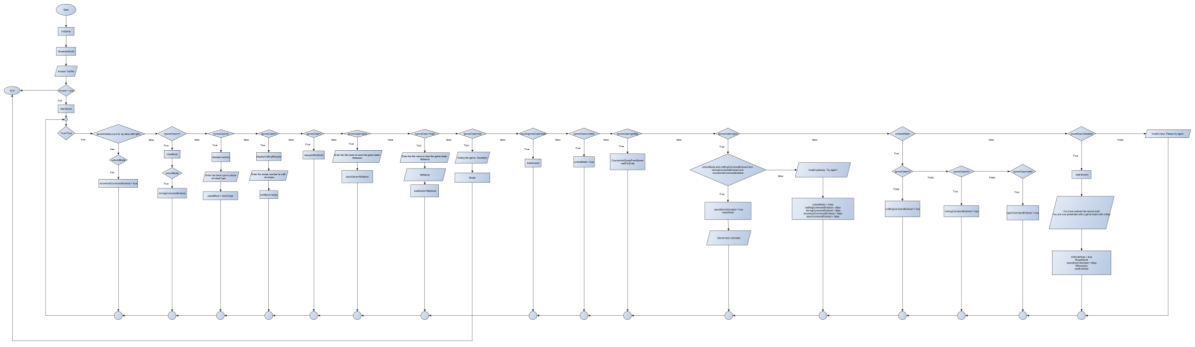
Furthermore it showcases the teamwork via GitLab and adjustments the team made on the code to add an extra layer of complexity to it.

4. Git collaboration & version control: Git managed by Vasileios
5. Extending the Game code: Blocks & functionality done by Andrei. Flag done by Vasileiosl/Jan
6. The process of interacting with flags API is summarized in this point.

We as Group78 worked well together and hope everything is understandable as well as lives up to the high standard we encountered in the last couple of weeks in BCS1110.

2 JavaCraft's Workflow

- Flowchart For Game:



Pseudocode for Game:

```
Initialize variables, constants and arrays

create main function
call initGame function and set parameters for worldWidth to 25 and
for worldHeight to 15
        initialize new array called worldWidth
        initialize new array called worldHeight
insert worldWidth and wordHeight into a 2D Array called world
set players location to the middle of the world
initialize Inventory to new ArrayList
call generateWorld function
        initialize new random object
        iterate through  array worldWidth and array worldHeight using
two for loops
        generate for each cell (x, y) a random integer between 0 and
99
        if random value < 20
then place wood
        else if random value < 35
 then place leaves
        else if random value < 50
then place stone
        else if random value < 70
then place iron ore
        else place air
print out welcome message and instruction
initialize scanner object
print message which should ask the player if he wants to play
scan next user input and bring it to upper case
store user input in startGameChoice variable
if user input = “y”
        then call startGame function
else
        print out goodby message and exit game

create startGame function
initialize new scanner object
initialize unlockMode as a boolean and set it to false
initialize craftingCommandEntered as a boolean and set it to false
initialize miningCommandEntered as a boolean and set it to false
initialize movementCommandEntered as a boolean and set it to
false
initialize new while function which runs forever
call clearScreen function
call displayLegend function
call displayWorld function
call displayInventory function

print instruction
call scanner object and set next input to lowercase
if input equals to a movementCommand
        set movementCommandEntered to true
                then call movePlayer function
else if input equals to miningCommand regardless of the
capitalization
        set miningCommandEntered to true
        then call mineBlock function
else if input equals to “p” regardless of the capitalization
    print message
if unlockMode is true check
        if user input equals to “c” regardless of the capitalization
        set craftingCommand to true
else if user input equals to “m” regardless of the capitalization
        set miningCommand to true
else if user input equals to “open”
    set openCommandentered to true
if secretDoorUnlocked if true
        call clearScreen function
        print message
        set boolean inSecretArea to true
        call resetWorld function
        set boolean value scretDoorUnlocked to true
        call fillInventory function
        call waitForEnter function

create fillInventory function
        clear inventory
use two for loops to iterate through all the blocktypes  and to fill the
inventory with all blocktypes

create movePlayer function
switch (user input, change it to upper case)
case “W” or “UP”
if player Y coordinate is bigger than 0
        then subtract player Y coordinate by 1
break
case “S” or “DOWN”
if player Y coordinate is bigger than worldHeight -1
        then increase player Y coordinate by 1
break
case “A” or “LEFT”
if player X coordinate is bigger than 0
        then decrease player coordinate by 1
break
case “D” or “RIGHT”
if player X coordinate is less than worldWidth -1
        then increase player X coordinate by 1
                break
default
                break

create mineBlock function
initialize blockType variable and set it to playersX and playersY
coordinate
if blockType does not equal to air
        then add blockType to players inventory
        set at the players location an air block
        print message
else
        print message
call waitForEnter function

create placeBlock function
if blockType >= 0 and blockType <= 7
        if blockType <= 4
                if inventory contains blockType
```

```
        then call displayInventory function
        print “Enter block type to place”
        scanner user next input and store it in variable called blocktype
call placeBlock function with user input as blocktype
else if user input equals to “c” regardless of the capitalization
        call displayCraftingRecipes function
        print “Enter recipe number to craft”
        scan next user input and stor it in variable called recipe
        call craftItem function with recipe as int
else if user input equals “i” regardless of the capitalization
        call interactWithWorld function
else if user input equals to “save” regardless of the capitalization
        print “enter file name to save the state”
        scan user next input and store it in variable called filename
        call saveGame function with filename as string
else if user input equals to “load” regardless of the capitalization
        print “enter file name to load the game state”
        scan user next input and store it in a variable called filename
        call loadGame function with filename as string
else if user input equals to “exit” regardless of the capitalization
        print “exiting the game. Goodbye!”
        break
else if user input equals to “look” regardless of the capitalization
        call lookAround function
else if user input equals to “unlock” regardless of the capitalization
        set unlockMode to true
else if user input equals to “getflag” regardless of the capitalization
        call getCountryAndQouteFromServer function
        call waitForEnter function
else if the user input equals to “open” regardless of the capitalization
if all the boolean values unlockMode, craftingCommandEntered,
miningCommandEntered and movementCommandEntered  are
true
        then set secretDoorUnlocked to true
        call resetWorld function
        print “Secret door unlocked”
        call waitForEnter function
else
        print message
        call waitForEnter function
        set unlockMode to false
        set craftingCommandEntered to false
        set miningCommandEntered to false
        set movementCommandEntered to false
        set openCommandEntered to false
else if user input did not match any of the instructions (possible
inputs)
```

```
                remove blockType from inventory via calling
removeItemsFromInventory function
                change block from players position (playerY and playerX
coordinate) to blockType
                print message
else
                print message
else
                initialize craftedItem variable and assign it to
getCraftedItemFromBlockType function
if craftedItems  contains a craftedItem
                remove craftedItem from crafted items via calling
                removeItemsFromInventory function
                change block from players position (playerY and playerX
coordinate) to blockType
                print message
else
                print message
else
                print messages
call waitForEnter function

create waitForEnter function
print message
initialize new scanner object
scan next line

create craftItem function
switch (recipe)
case 1
        call craftWoodenPlanks function
        break
case 2
        call craftStick function
        break
case 3
        call craftIronIngot function
        break
default
                print message
call waitForEnter function
```

3 Functionality Exploration

List of key functionalities explored:

No.	Function Name	Description
1	Main Function (main)	Generates a game map, displays the game instructions and asks the user if he wants to start the game. Within it initGame, generateWorld are called.
2	initGame()	This method generates the world, sets the player position and initialises/ creates the inventory.
3	generateWorld()	Based on random values, this functions adds blocks like WOOD, LEAVES, STONE, IRON_ORE and air and sets them on each of the blocks (spaces in the world matrix)
4	displayWorld()	This method prints a world map with a “World Map” title. It then creates a grid of characters that represent the world, where “P” denotes the player's current position. If the player is in the secret area the color of the “P” turns blue and otherwise it remains green . The rest of the world is filled using symbols obtained from the getBlockSymbol method.
5	getBlockChar()	Based on a switch with an input of what type of block it is, like WOOD, AIR , it returns its correspondent character. For example in the case of WOOD, the function returns '\u2592'.
6	startGame()	This is the main area of the code, here the input of the user is checked and other functions like initGame,clearScreen(),displayLegend(), displayWorld() are called to create and update the game according to the user actions.
7	fillInventory()	This function helps the game fill the inventory. The inventory is an array therefore it needs a function to go through all of the possible places within it and fill it with a certain blocktype.
8	resetWorld()	This method resets the generated world . It calls the generateEmptyWorld() method to clear the screen and create a new world as well as updates the player position to the starting one. (middle of the board)
9	generateEmptyWorld()	This method initializes the 2d matrix that stores the values for the world map . For each "stripe" of the world there exists a for loop which places either red, white or blue blocks on each of the positions in the 2d world matrix. This fills the world with blocks.
10	clearScreen()	This method runs the cmd /c and cls command if the operating system is windows to clear the command line window. If it isn't windows, a specific string gets printed out and the system gets flushed.
11	lookAround()	This private method simulates the player looking around the surrounding environment and shows a small area around the current player position. Currently the player position is marked green, while the blocks around the player are being represented by the symbols obtained from the “getBlockSymbol” method. It then waits for the player to press enter to continue the game.
12	movePlayer()	This method moves the player through the world based on the string that is given in startGame(). It updates the player position according to what the input was.
13	mineBlock()	This method "mines" the block by checking if the block at the current player position is different from air and if it is, the block gets changed to air and the block gets added to the players inventory.
14	getBlockTypeFromCraftedItem()	This private method takes the crafteditem integer value and returns the corresponding block types. It uses a switch statement where for each valid input there exists a case that returns the corresponding blocktype. If the crafted item is not valid, it returns -1.
15	getCraftedItemFromBlockType()	This private method takes the blocktype integer value as input and returns the crafted item

		integer. It uses a switch statement to map the blocktypes to the corresponding crafted items, such as sticks, iron ingots and wooden planks. If the blocktype integer value does not correspond to any case, the method returns -1.
16	displayCraftingRecipes()	This method lists the different crafting recipes available by purely printing out different strings for them.
17	craftitem()	This function checks which crafting recipe the player has picked and calls the appropriate method. This is done through a switch case. If the recipe number does not correspond to the valid cases, the method returns "invalid recipe number".
18	craftWoodenPlanks()	This method crafts the WoodenPlanks, by checking if the inventory contains 2 Wood blocks. If they exist, they get removed from the inventory (calls removeItemsFromInventory(WOOD,2)) , then the method adds the wooden planks to the crafted items list and finally prints out "crafted wooden planks". If the initial WOOD blocks don't exist, the method returns "insufficient resources to craft wooden planks".
19	craftStick()	This method crafts the Sticks, by checking if the inventory contains 1 Wood block. If it exists, it gets removed from the inventory (calls removeItemsFromInventory(WOOD,1)) , then the method adds the stick to the crafted items list and finally prints out "crafted sticks' '. If the initial WOOD block doesn't exist, the method returns "insufficient resources to craft stick".
20	craftIronIngot()	This method crafts the Iron ingots, by checking if the inventory contains 3 IRON_ORE blocks. If they exist, they get removed from the inventory (calls removeItemsFromInventory(IRON_ORE,3)) , then the method adds the Iron ingot to the crafted items list and finally prints out "crafted iron ingot". If the initial IRON_ORE blocks don't exist, the method returns "insufficient resources to craft iron ingot".
21	inventoryContains()	This method returns true if the inventory contains the requested count of a given item. Using a for loop the method traverses the inventory and checks if the integer matches the given one. If it does, the counter gets decreased. At the end of the for loop, the method checks if the counter is equal to the requested amount, if it is it returns true. Else it returns false.
22	removeItemsFromInventory()	This method removes a given amount of a specific item from the inventory. If the item is present in the inventory, the item gets removed and the counter initialised with zero at first grows by one . If the counter is equal to the specified amount , it exits the while loop.
23	addCraftedItem()	This method adds an item to the crafted items list. If the list doesn't exist, it gets initialized; else it adds the integer provided in the parameter to the list.
24	interactWithWorld()	This method allows you to interact with the world. It receives the blocktype from the player position and has a switch case with all the different types of block. Whenever you mine a block, it finds the corresponding block int value and gives you a message, then adds the block to your inventory. If the item is not found, the message "Unrecognized block. Cannot interact" gets displayed.
25	saveGame()	This method uses the ObjectOutputStream to serialise and write all of the game data to a specific file. Included are the world dimensions, player information, inventory, unlock mode and specifically the craftedItems. It prints a confirmation message if successful. Else it gives an error message with details. Lastly it waits for the input of the user (waitForEnter() gets called).
26	loadGame()	This method uses the ObjectOutputStream to load game data from a specific file. It deserialises it. It then reads and assigns values of the player information (position), inventory as well as the crafted items , unlock mode and finally world dimensions. It prints a confirmation message if successful, else it prints an error message with details.
27	getBlockName()	This method gives the block name based on the block type integer in the parameter. It uses a switch case to determine which integer corresponds to which string. For case WOOD (int value), "Wood" gets returned. If the integer value in the parameter does not correspond to any case, the function returns "Unknown".

28	displayLegend()	This method displays the legend for the world map. It prints out the different symbols and the colors of each block as well as the player symbol and its color.
29	getblockColor()	This method returns the color of the different blocks within the game. Using a switch case based on the integer blocktype (that is taken as an parameter), it decides which color to return. For Air the return is empty, for wood the return is red and so on. If the integer provided does not correspond to the given cases, it returns empty("").
30	waitForEnter()	This method waits for the input by the player. It prints a message, uses a scanner and just reads the next string entered by the player.
31	getCraftedItemName()	This method returns the name in a string format of the crafted items. Using a switch case based on the integer value of the crafted items, it returns the string equivalent of the name. If the integer doesn't match anything, the method returns unknown.
32	getCraftedItemColor()	This method returns the color of the crafted item. Yet for Wooden Planks and Sticks there is no color returned and for the rest the switch case returns ANSI_BROWN. Although, if the integer is different from CRAFTED_WOODEN_PLANKS, CRAFTED_STICK, CRAFTED_IRON_INGOT, the switch returns an empty string.
33	getCountryAndQuoteFromServer()	This method creates a POST request to a given URL with a given request body. It then receives the response and prints it out in the console. If an error occurs the method replies with "Error connecting to the server".
34	getBlockSymbol	This private method takes as input a block type and returns a colored symbol that represents that blocktype. It does so by using a switch statement to determine the color based on the int value given. It then combines it with the corresponding character obtained from the “getBlockChar” function to create the full block symbol. If the int value given as input is not found among the cases, it returns a default color and symbol.
35	placeBlock	This method allows the player to place blocks in the game world. It checks if the player has the required block in its inventory and whether the block type is within a specific range. If the conditions are met, the game map gets updated and it issues a message such as “ Placed *Wood* at your position”. If the blocktype given is out of that specific range, the following message is issued : “Invalid block number. Please enter a valid block number. “ or if the crafted item is not in the crafted item list, it also issues a similar message.

4 Finite State Automata (FSA) Design

Secret Door Logic Analysis

Overview of important commands and their initial state at game launch:

- unlockMode = false
- craftingCommandEntered = false
- miningCommandEntered = false
- movementCommandEntered = false
- openCommandEntered = false

Mechanics behind the secret door:

Further analysis revealed a special order of inputs needed to access the secret door.

The game initialises the following boolean variables and sets them to false:**"unlockMode”** ,**“craftingCommandsEntered”**,**“miningCommandEntered”**,**"movementCommandEntered”** and **"openCommandEntered”**. These are used to track what the player has done within the game.

Since the game is text-based, the player gets to see the inventory and the set of available commands. These commands entered by the player are then checked using a series of if statements.

Since the game functions based on the commands entered by the player, these if statements are contained within a while (true) loop that continuously asks the player for input and only exits the loop if the user issues the command **“exit”.**

Out of the available command set, of particular interest is the unlock command. If **“unlock”** is typed in by the player, the “unlockMode” flag is set to true.

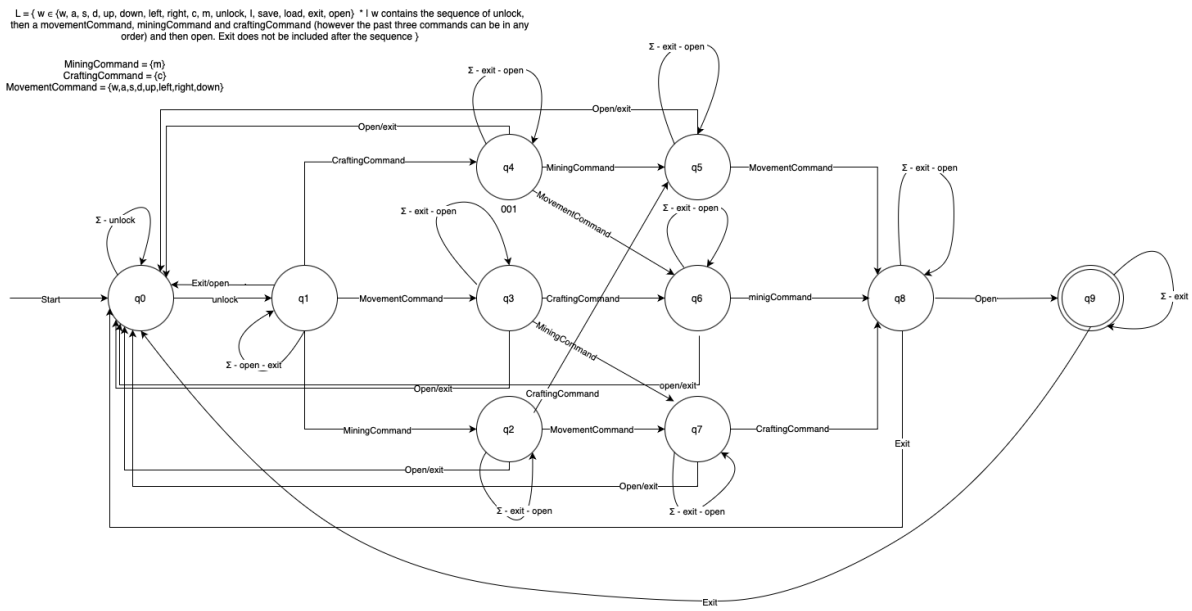
Within the source code of the game, actions like mining, moving, and crafting have a special if-clause in case the **“unlockMode”** is set to true. Therefore, this is the first necessary step the player has to take to get closer to unlocking the secret door.

Continuing, after setting “unlockMode” to true, the following commands can be run in any order: Mining any block will set **“MiningCommandEntered”** to true; moving throughout the game will set **“MovementCommandEntered”** to true; and crafting any item will set **“CraftingCommandEntered”** to true. After the completion of setting all the necessary flags to true through the inputs of the in-game commands, meaning unlocking : **“unlock”** , mining : **“m”** or **“M”**, moving: **“W,A,S,D”** or **“Up, down, right,left”** , and crafting : **“c” or “C”**, the player needs to take the final step and type in **“open”**. This in turn will set the **“OpenCommandEntered”** to true, which will trigger a special if-clause within the game and change the state of the **“secretDoorUnlocked”** flag from false to true.

The player has then entered and opened the secret door. The game screen clears, and the player is greeted with the following message: “You have entered the secret area! You are now presented with a game board with a flag!”.
The world gets generated again, yet with a different distribution of blocks, namely, wood and stone blocks in the form of the Dutch Flag. The player's inventory is also filled.

Overall, the player can only set the flag “secretDoorUnlocked” to true by performing specific actions while in unlock mode. If the player types **“open”** while failing to meet these conditions, the door remains closed, and the player is informed of the failure through a message. This action does reset the progress made by the player, meaning they have to start from the beginning. Even after a failure, the while (true) loop keeps going, allowing the player to try and potentially unlock the door yet again.

FSA Illustration & Description:



The FSA design we picked is that of an DFA, deterministic finite automaton.
“u” or “unlock” = unlock
“c” or “CraftingCommand” = craft
“m” or “MiningCommand”= mine

“w” or “MovementCommand” = move
“o” or “Open” = open
“Exit” or “Exit “ = exit

Set of States:
(Q): {q0,q1,q2,q3,q4,q5,q6,q7,q8,q9}
Alphabet:
Σ = {u,c,m,w,o,exit}
Initial State:
(q0): q0
Accepting State:
(F): q9
Language Recognized: L = { w ∈ Σ * | w contains the u,c,m,w,o commands at least once where u is at the start and o at the end of the string }

DFA - Table

States	u	c	m	w	o	exit
q0	q1	q0	q0	q0	q0	q0
q1	q1	q4	q2	q3	q0	q0
q2	q2	q5	q2	q7	q0	q0
q3	q3	q6	q7	q3	q0	q0
q4	q4	q4	q5	q6	q0	q0
q5	q5	q5	q5	q8	q0	q0
q6	q6	q6	q8	q6	q0	q0
q7	q7	q8	q7	q7	q0	q0
q8	q8	q8	q8	q8	q9	q0
q9	q9	q9	q9	q9	q0	q0

5 Git Collaboration & Version Control

- Repository Link: https://gitlab.maastrichtuniversity.nl/bcs1110/javacraft/-/tree/group78
- Branch Details: **Main Branch Name:** Group78, **Corresponding Members:** Andrei, Vasileios, Armanto, Jan

Introduction To Git

When we were introduced to git we were amazed by the possibilities of such a simple on-the-surface program. Our team made our git branch and implemented its usage with our workflow. Initially, we experienced no conflicts, but with continued use over the following days, conflicts became more frequent. We developed the ability to address and learn from these issues by implementing better organization and clearer categorization of each team member's work. This approach allowed everyone to work concurrently without encountering conflicts.

Code Versioning

Thanks to git’s in-depth versioning system and rollbacks our team was able to revert to early versions of code or save lost code files. The benefits that git gave to our team in regard to version control and code versioning did not make a big impact but was still useful.

Review Process

Thanks to git and the ease of updating and maintaining our branch in the javacraft repository, collaboration became quicker and more efficient. Each team member could easily push updates to his code and let other team members check the code. Same with the pseudocode, flowcharts and the DFA's.

Git Tools and Services

The main platform that was used for updating our repository was **GitLab** because the university provided us with it and it was very simple yet extensive when it comes to its tools and logging of commits and pushes.

Documentation

Using the git log and other tools like GitLab made documenting changes in our work very easy yet comprehensive.

Learning Best Practises

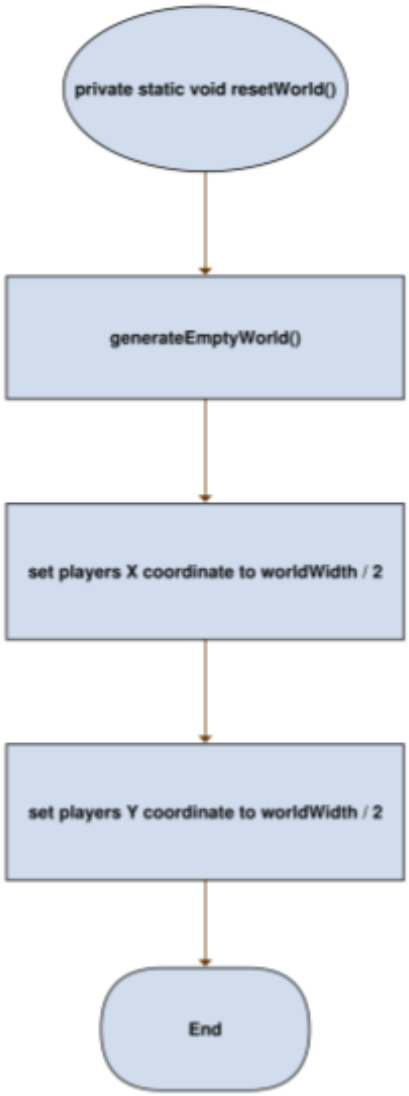
Thanks to git and GitLab we learned to make descriptive yet short comments on commits and changes to the database that made everything more clean and consistent. Like the color of the flowcharts.

Summary

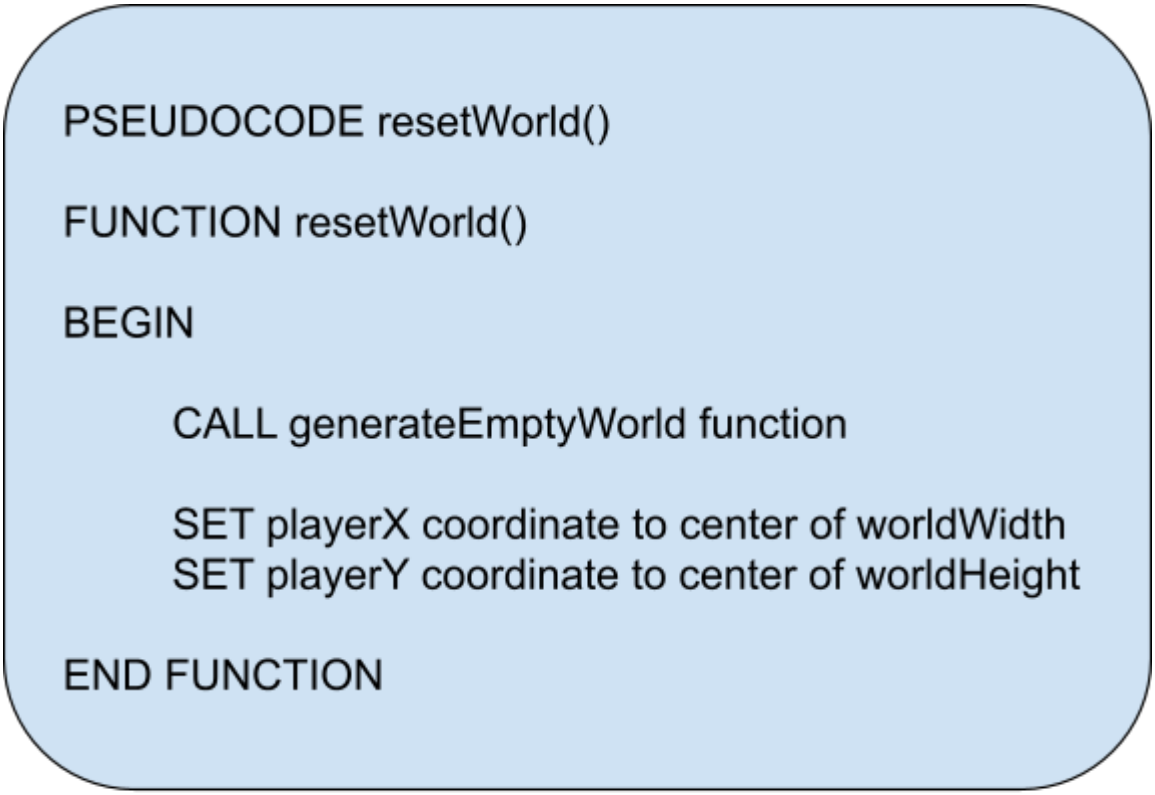
Git usage was a big part of our team's work. The team as a whole found git very useful so almost everything was being done through git to make sure there was consistency and transparency.

6 Appendix

Include any additional pseudocode, flowcharts, or supplementary material.



1.1 Flowchart: resetWorld by Jan

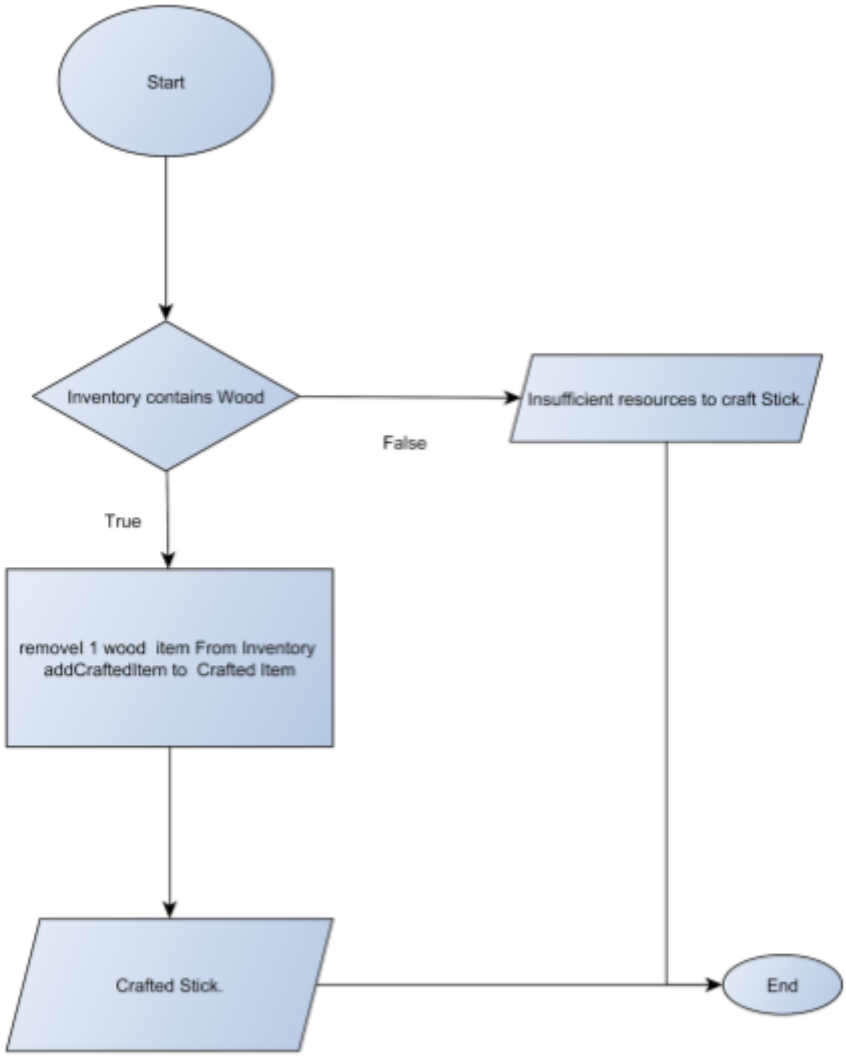


1.2 Pseudocode: resetWorld by Jan

2.1 Flowchart getBlockSymbol by Vasileios

```
PSEUDOCODE getBlockSymbol(int blockType)

FUNCTION getBlockSymbol(int blockType)

BEGIN

        CREATE STRING blockColor

        SWITCH on blockType
        AIR
                return ANSI_RESET and "- "
        WOOD
                SET blockColor to RED
        EXIT SWITCH
        LEAVES
                SET blockColor to GREEN
        EXIT SWITCH
        STONE
                SET blockColor to BLUE
        EXIT SWITCH
        IRON_ORE
                SET blockColor to WHITE
        EXIT SWITCH

        Everything else:
                SET blockColor to ANSI_RESET
        EXIT SWITCH

        END SWTICH

        return blockColor and CALL getBlockChar(blockType)

END FUNCTION
```

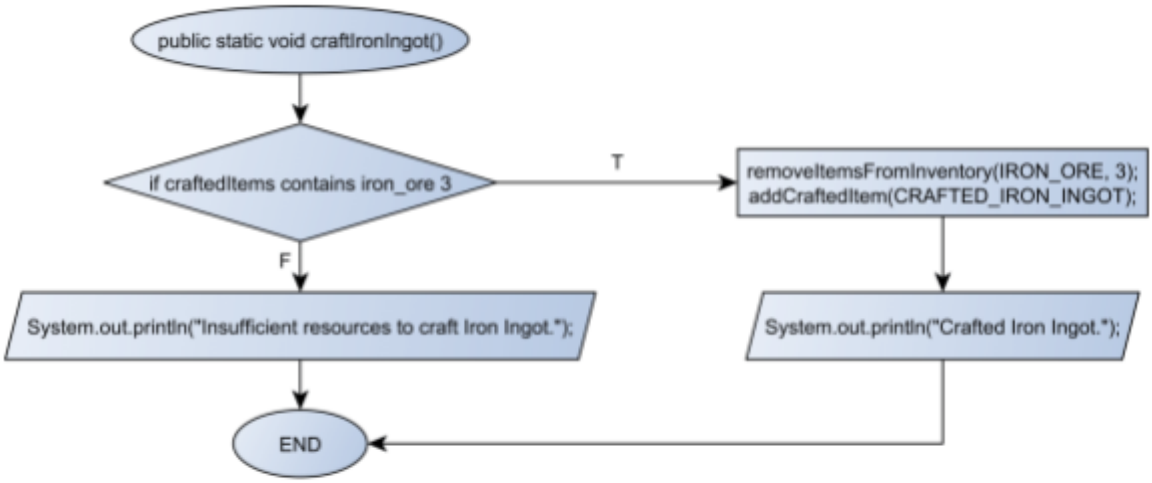
2.2 Pseudocode: getBlockSymbol by Jan



3.1 Flowchart craftStick by Armanto

```
PSEUDOCODE craftStick()

FUNCTION craftStick()

BEGIN

 IF inventory CONTAINS "WOOD" THEN

    REMOVE 1 ITEMS "WOOD" FROM inventory
                ADD STICK TO CraftedItem

    PRINT "Crafted Stick."
 ELSE
        PRINT "Insufficient resources to craft Stick."
 ENDIF

END FUNCTION
```

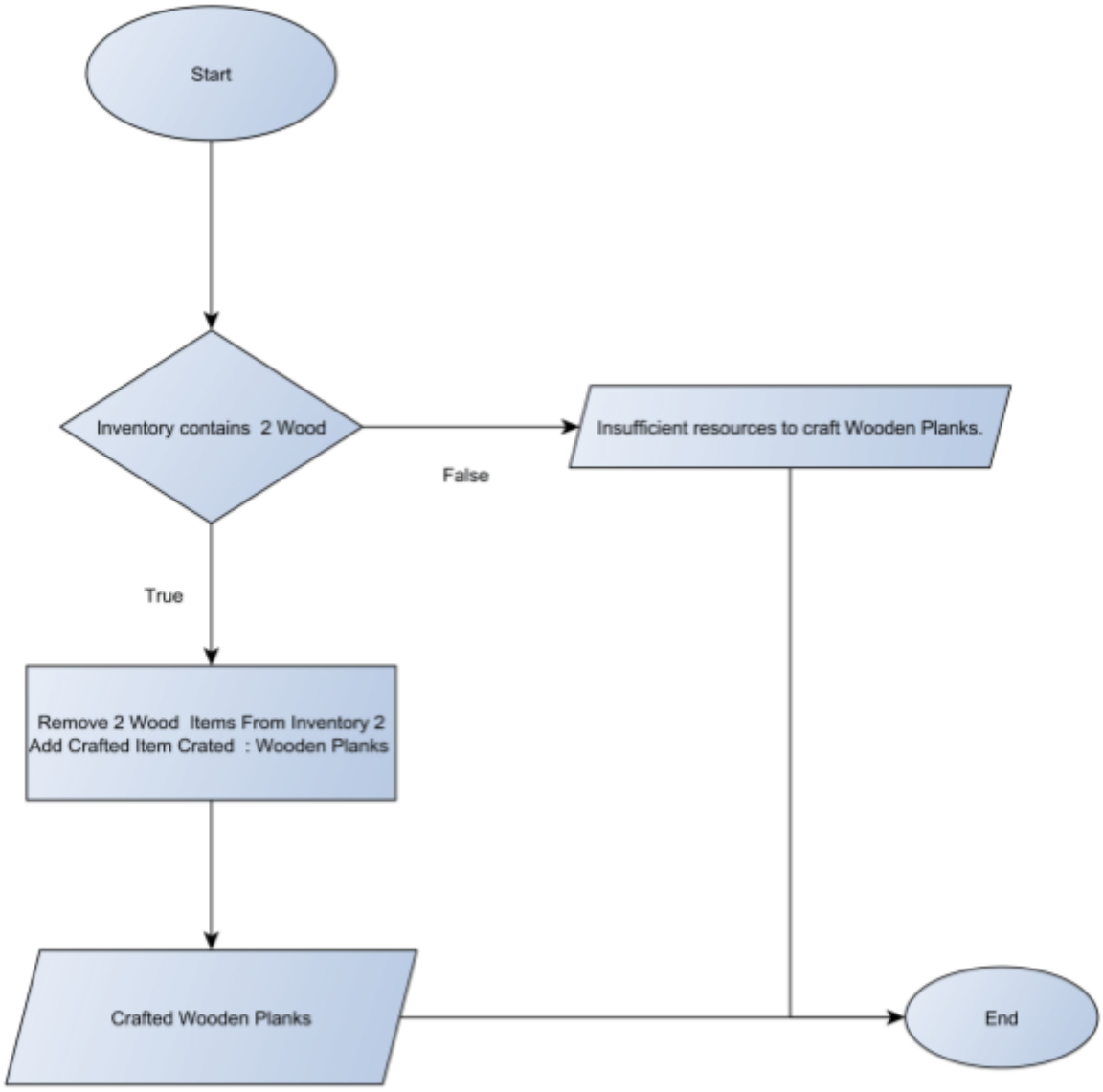
3.2 Pseudocode: craftStick by Armanto



4.1 Flowchart: craftIronIngot by Vasileios

```
FUNCTION  craftIronIngot()

        IF inventoryContains("IRON ORE",  3)  THEN

    REMOVE 3 "IRON ORE" FROM INVENTORY
        ADD "IRON INGOT" TO CraftedItem
        PRINT "Crafted Iron Ingot."

        ELSE
        PRINT "Insufficient resources to craft Iron Ingot."

        ENDIF

END FUNCTION
```

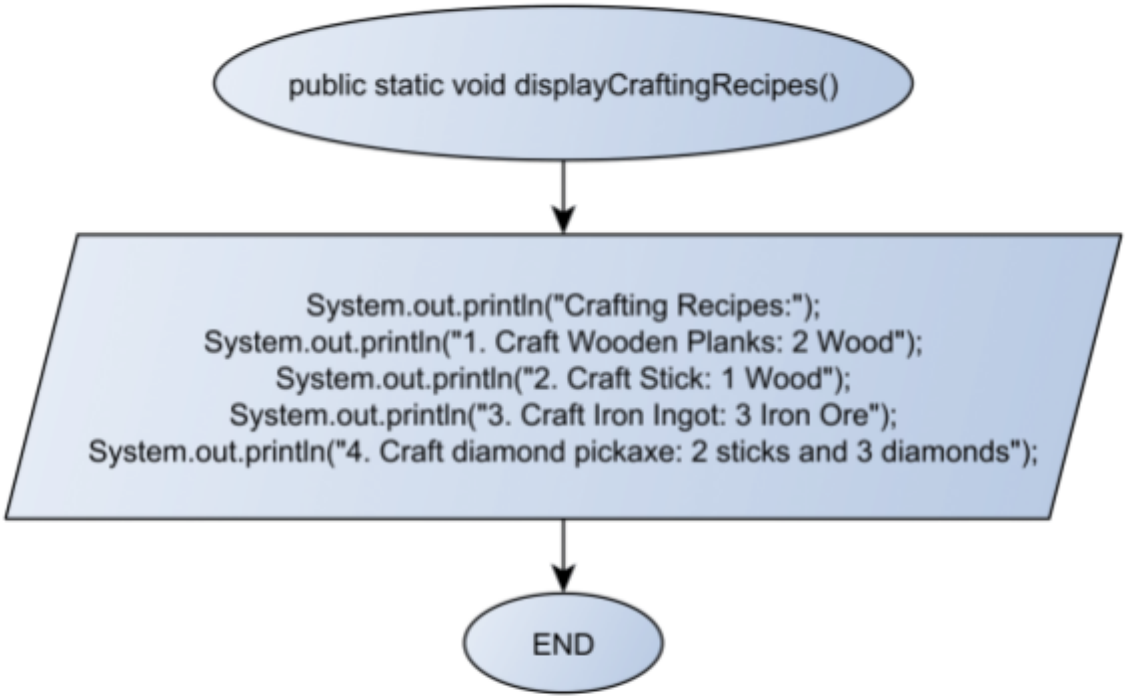
4.2 Pseudocode: craftIronIngot by Armanto



5.1 Flowchart: craftWoodenPlanks by Armanto

```
PSEUDOCODE craftWoodenPlanks()

FUNCTION craftWoodenPlanks()

BEGIN
  IF inventory CONTAINS 2 "WOOD" THEN

      REMOVE 2 "WOOD" FROM inventory
      ADD CRAFTED WOODEN PLANKS to craftedItem
      OUTPUT "Crafted Wooden Planks.'

  ELSE
      OUTPUT "Insufficient resources to craft Wooden Planks"

  END IF

END FUNCTION
```

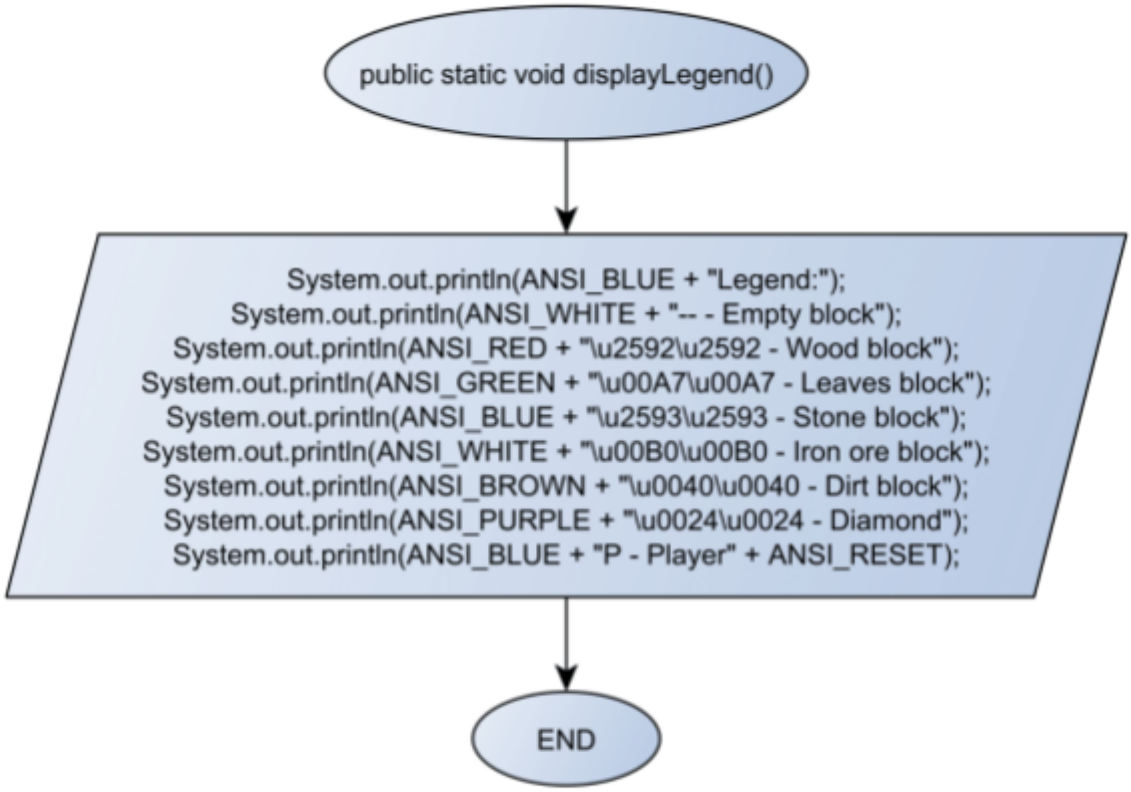
5.2 Pseudocode: craftWoodenPlanks by Armanto



6.1 Flowchart: displayCraftingRecipes by Vasileios

```
PSEUDOCODE displayCraftingRecipes()

FUNCTION displayCraftingRecipes()

BEGIN

        OUTPUT "Crafting Recipes:"
        OUTPUT "1. Craft Wooden Planks: 2 Wood"
        OUTPUT "2. Craft Stick: 1 Wood: 1 Wood"
        OUTPUT "3. Craft Iron Ingot: 3 Iron Ore"

END FUNCTION
```

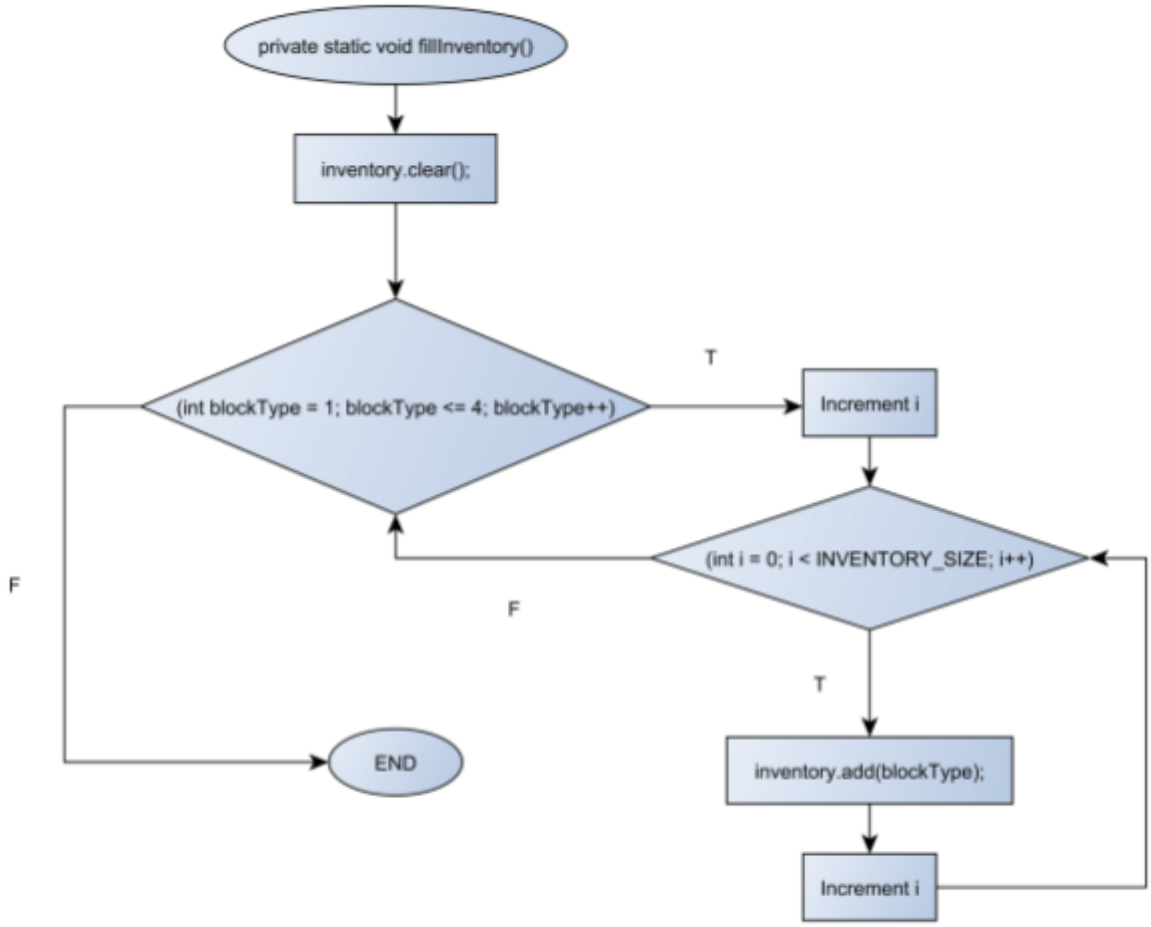
6.2 Pseudocode: displayCraftingRecipes by Armanto



7.1 Flowchart: displayLegend by Vasileios

```
PSEUDOCODE displayLegend()

FUNCTION displayLegend()

BEGIN

        OUTPUT "Legend:" in ANSI_BLUE"
        OUTPUT "-- - Empty block" in ANSI_WHITE"
        OUTPUT "\u2592\u2592 - Wood block" in ANSI_RED"
        OUTPUT "\u00A7\u00A7 - Leaves block" in ANSI_GREEN"
        OUTPUT "\u2593\u2593 - Stone block"iin ANSI_BLUE"
        OUTPUT "\u00B0\u00B0- Iron ore block in ANSI_WHITE"
        OUTPUT "P - Player" in ANSI_BLUE + ANSI_RESET"


END FUNCTION
```

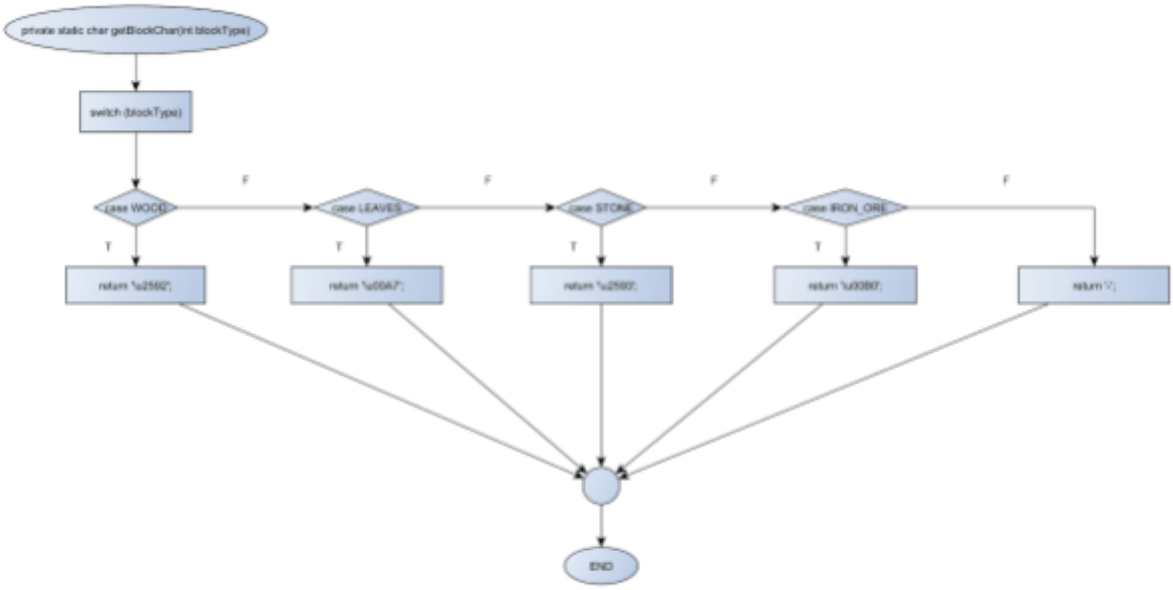
7.2 Pseudocode: displayLegend by Armanto



8.1 Flowchart fillInventory by Vasileios

```
PSEUDOCODE fillInventory()

FUNCTION fillInventory()

BEGIN

CLEAR inventory of user

FOR blocktype FROM 1 TO 4
        FOR i FROM 0 TO (inventory size)-1

        ADD blocktype to inventory

        END FOR
END FOR

END FUNCTION
```

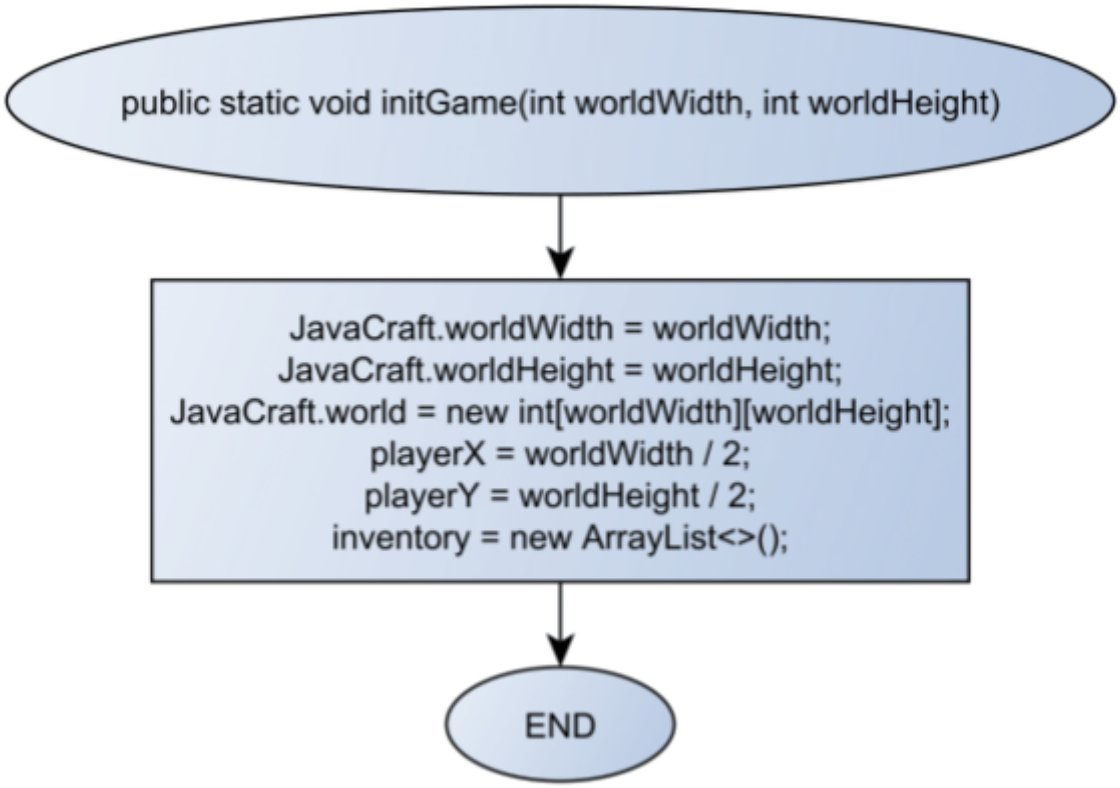
8.2 Pseudocode: fillInventory by Jan



9.1 Flowchart getBlockChar by Vasileios

```
PSEUDOCODE getBlockChar(int blockType)

FUNCTION getBlockChar(int blockType)

BEGIN
        SWTICH on blockType
        WOOD
                return '\u2592'
        LEAVES
                return '\u00A7'
        STONE
                return '\U2593'
        IRON_ORE
                return '\U00B0'

        Everything else:
                return '-'

        END SWITCH
END FUNCTION
```

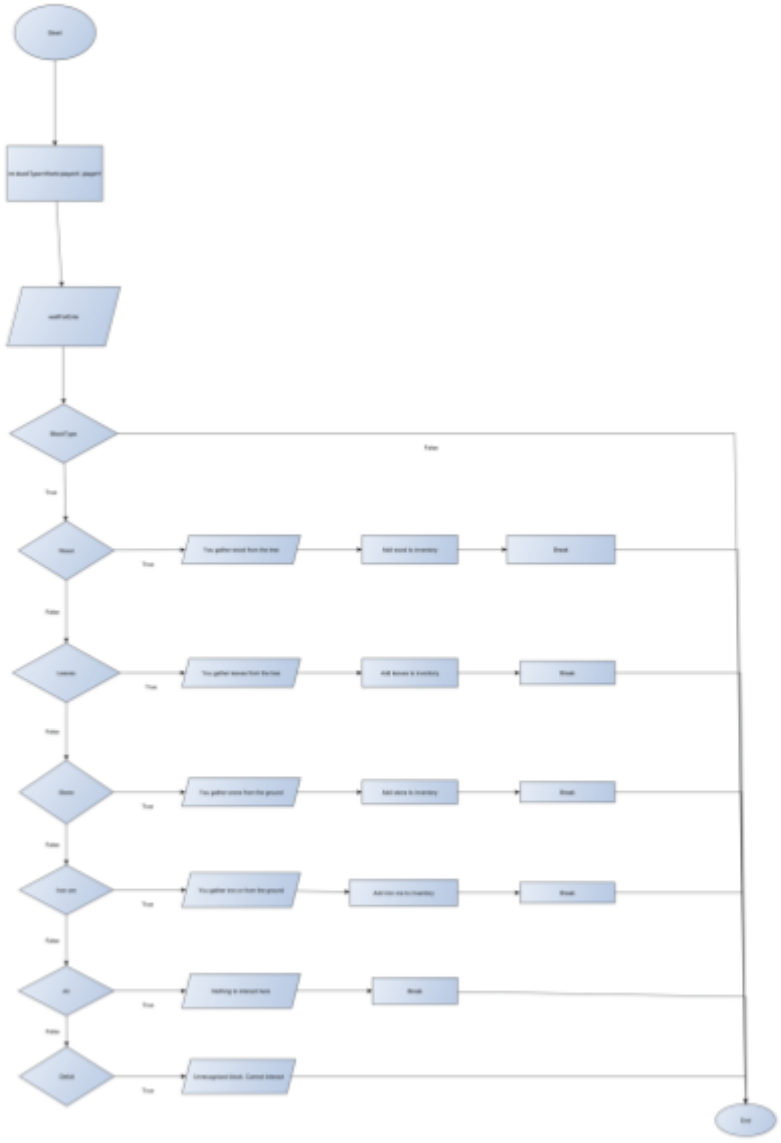
9.2 Pseudocode: getBlockChar by Jan



10.1 Flowchart initGame by Vasileios

```
PSEUDOCODE initGame(int worldWidth, int WorldHeight)

FUNCTION initGame(int worldWidth, int worldHeight)

BEGIN

    SET width of the game world to worldWidth
    SET height of the game world to worldHeight

    CREATE empty 2D array with dimensions worldWidth x worldHeight
    ASSIGN Javacraft.world to that array

    SET playerX variable the worldWidth/2 value
    SET playerY variable the worldWidth/2 value

    CREATE empty inventory array
    ASSIGN JavaCraft.inventory to that array

END FUNCTION
```

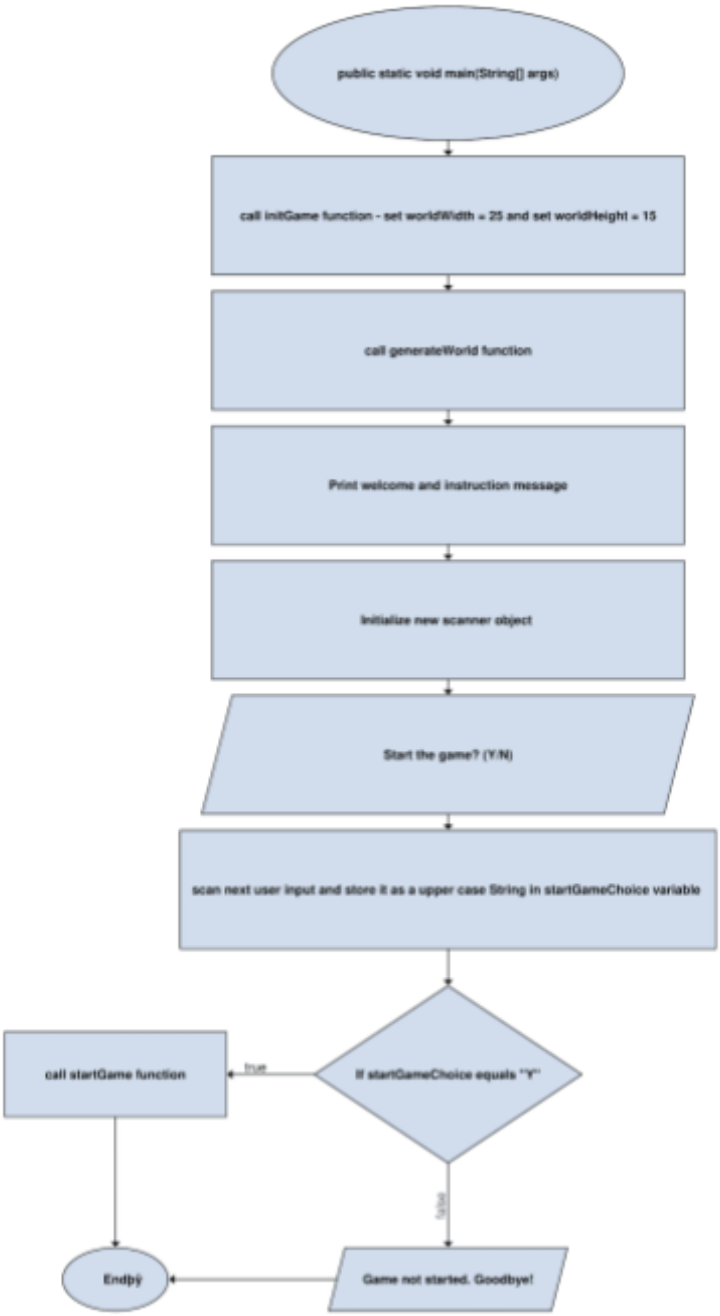
Pseudocode 10.2: initGame by Bill



11.1 Flowchart: interactWithWorld by Armanto

```
PSEUDOCODE interactWithWorld()

FUNCTION interactWithWorld()

BEGIN

CREATE blockType and ASSIGN it the block at the players coordinates

SWITCH based on blockType
        WOOD
                OUTPUT "You gather wood from the tree"
                ADD WOOD to inventory
        EXIT SWITCH

        LEAVES
                OUTPUT "You gather leaves from the tree."
                ADD LEAVES to inventory
        EXIT SWITCH

        STONE
                OUTPUT "You gather stones from the ground"
                ADD STONE to inventory
        EXIT SWITCH

        IRON_ORE
                OUTPUT "You mine iron ore from the ground"
                ADD IRON_ORE to inventory
        EXIT SWITCH

        AIR
                OUTPUT "Nothing to interact with here"
        EXIT SWITCH

        Everything else:
                SAY "Unrecognized block. Cannot interact"

END SWITCH

CALL waitForEnter()

END FUNCTION
```

11.2 Pseudocode: interactWithWorld by Armanto



12.1 Flowchart: main by Jan

PSEUDOCODE

Main - Jan
void main()

calls function initGame and sets value of variable worldWidth to 25 and value of variable worldHeight to 15

calls function generateWorld

prints out welcome message in green color

prints out instructions

prints out instructions on how to play the game in the following seven lines

prints out empty line

initializes scanner

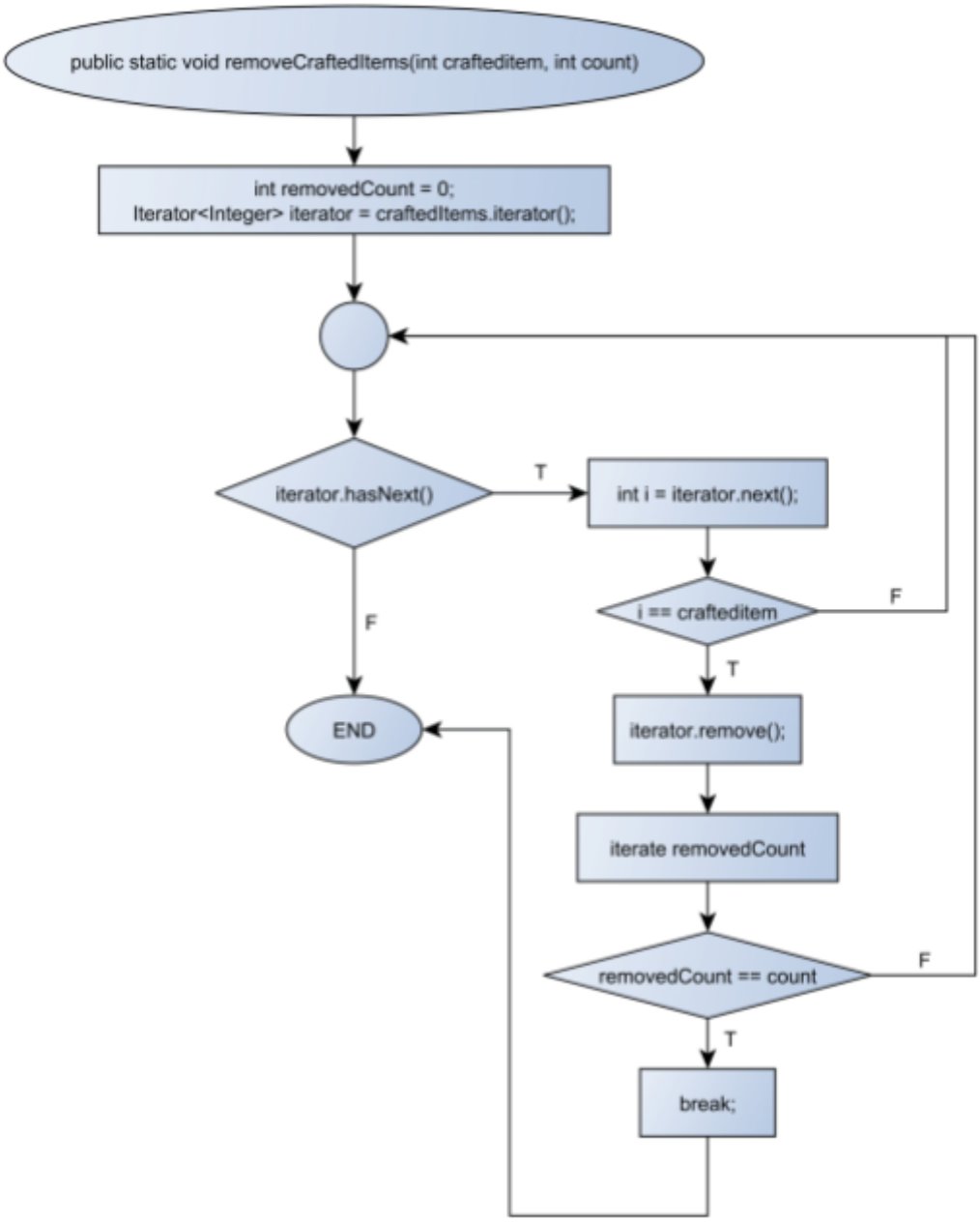
prints out “Start the game? (Y/N)” - asks the player if he wants to play the game

next character entered by the player will be automatically changed to an upper case character

if input equals “Y” start game

  else print “Game not started. Goodbye”

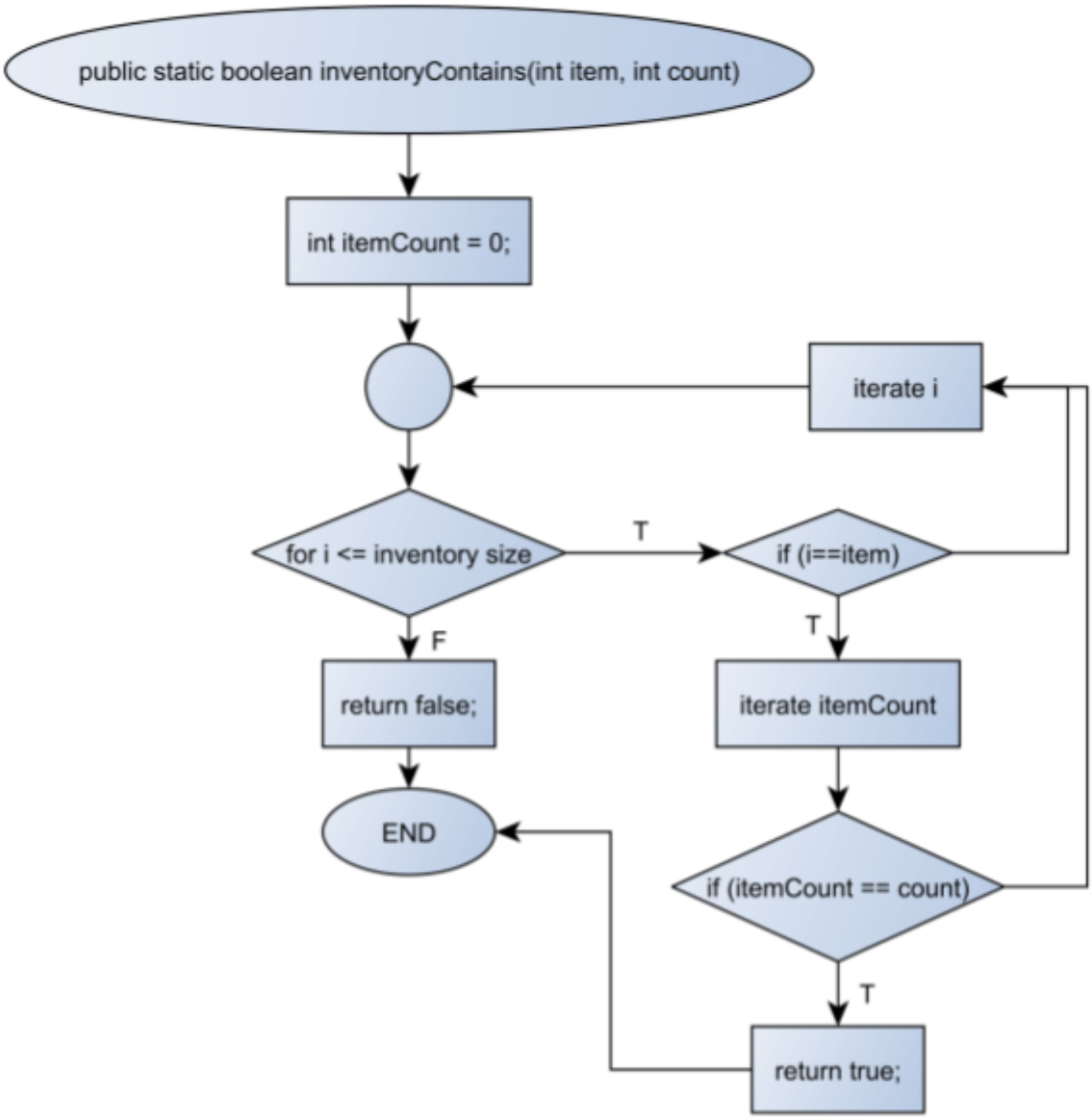
Pseudocode 12.2: main by Jan



13.1 Flowchart removeCraftedItems by Vasileios

```
PSEUDOCODE removeCraftedItems:
    removedCount = 0
    iterator = craftetItems of iterator
    while iterator has next:
        i = iterator.next
        if i equels craftetItems then:
            remove iterator
            add 1 to removeCount
            if removeCount equels count then:
                break loop
            IF END
        IF END
    WHILE LOOP END
PSEUDOCODE END
```

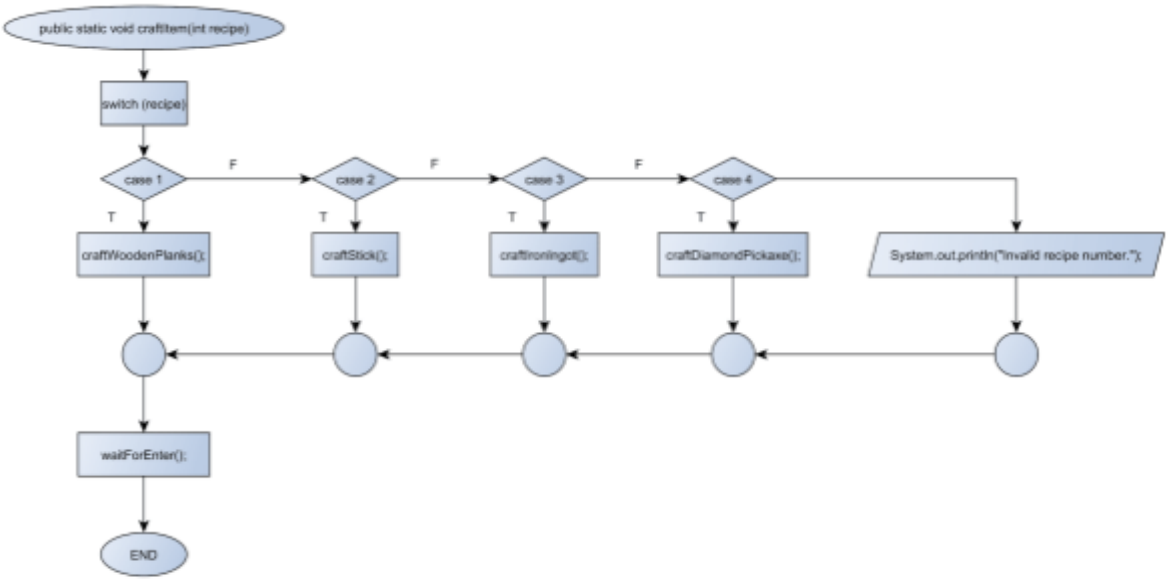
Pseudocode 13.2: removeCraftedItems by Bill



14.1 Flowchart: inventoryContains by Vasileios

```
Pseudocode

Create function inventoryContains(int item)
  if inventory contains item then
        return true
  else
        return false

Create function inventoryContains(int item, int count)
  Initialize itemCount = 0

  for each element i in inventory do
        if i is equal to item then
        Increment itemCount by 1

        if itemCount is equal to count then
        return true

  return false
```

Pseudocode 14.2: inventoryContains by Jan



15.1 Flowchart: craftItem by Vasileios

```
Pseudocode

Create craftItem(int recipe) function
  Switch recipe
        Case 1:
              call craftWoodenPlanks()
              Break
        Case 2:
              call craftStick()
              Break
        Case 3:
              call craftIronIngot()
              Break
        Default:
              print "Invalid recipe number."
  end Switch

  Call waitForEnter()

END FUNCTION
```

Pseudocode 15.2: craftItem by Jan

7 References

1. Draw.io - FSA Creation
2. yworks.com - Flowchart creation
3. Google Docs - Snippets & Appendix