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: He did flowcharts and pseudocode for 4 game functions, did the description from 19-27 functions, did flowchart for game workflow, did secret door logic analysis, designed the FSA diagram and did it on the computer, committed in GitLab.

Giannis Fourlas: He did flowcharts and pseudocode for 4 game functions, did the description from 28-36 functions, did flowchart for game workflow, did description of the FSA, designed the FSA diagram, committed in GitLab, collaborated with game code extension (2 new blocks and 1 new crafting recipe).

Maarten Koji: He did flowcharts and pseudocode for 4 game functions, did the description from 1-9 functions, wrote the introduction, designed the FSA diagram, committed in GitLab, collaborated with game code extension (2 new blocks and 1 new crafting recipe).

Alexandra Plishkin: She did flowcharts and pseudocode for 4 game functions, did the description from 10-18 functions, did pseudocode for game workflow, designed the FSA diagram, did the Transition Function Table (FSA Secret Door), created the GitLab branch group66, committed and uploaded the final documents in it, reviewed the report document, collaborated with game code extension (2 new blocks and 1 new crafting recipe), fixed bugs in game after implementation of new blocks and crafting recipes.

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 has in the inventory. It uses the integer variable "blockType" and determines what's the integer value of the block the player is trying to place. If the value is bigger or equal to zero and smaller or equal to seven, there are three different possibilities; this is determined by if-else statements. In case of being smaller or equal to 4, it checks if the inventory contains the type of block, and in case of being there, it's removed from the inventory and placed in the world, determining the place with playerX/playerY; in case of not having the block in the inventory, it will tell the player that there's no block there. If the block is not smaller or equal to 4, it would generate an integer value "craftedItem", which is related to the block type, as you get the crafted item from a block type; if that crafted item is one of the items you have in the inventory designated as "Crafted items", the item will be removed from the inventory and placed in the world; on the other side, a message will pop up saying that there's no crafted item. If the integer typed wasn't any of the previous options, the program will say that it's invalid and will provide the player with a list of the 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 condition is true, the items of WOOD are removed from the inventory and an item of CRAFTED_WOODEN_PLANKS is added. If the condition is false, a message informs the player of the lack of the necessary 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 item of WOOD in the inventory necessary to make it work. If this condition is true, the item of WOOD is removed from the inventory and an item of CRAFTED_STICK is added. If the condition is false, a message informs the player of the lack of the necessary 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 items of iron ore in the inventory necessary to make it work. If this condition is true, the items of iron ore are removed from the inventory and an item of CRAFTED_IRON_INGOT is added. If the condition is false, a message informs the player of the lack of the necessary 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 will be counted, and if the item count is equal to what is needed for a crafting recipe, it will return true, if not it will return false.
24. [removeItemsFromInventory]: this function regards the inventory. It takes two integers as parameters, called "item" and "count". The items that will be counted by "count" and selected by "item". The variable removedCount is responsible for tracking the number of the items removed.
25. [addCraftedItems]: this function has a parameter called "craftedItem", and it has the items that the player has crafted. If there are no crafted items the program then creates an Arraylist to store the crafted items, and after that it adds the craftedItem to the Arraylist 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 then print a different message for each block. For example, when the player interacts with wood, the system will print "you gather wood from the tree", and if the block doesn't get recognized

the system will print “unrecognized block. Cannot interact”. The system will then wait for the player to type “enter” in order to confirm the operation.

27. [saveGame]: this function saves the game state to a specific file. This function includes the width and the height of the new world, the player position, the inventory contents, the crafted items and the unlock mode. These data are written into the file. If the game state has been successfully saved to the file, then it prints a message to inform the user. If there is an error during the process, then an error message appears. To ensure that the user has seen the message, the user has to press the “enter” button to proceed.
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 the width and the height for the new world, the player position, the inventory contents, the crafted items and the unlock mode. If the game state loads successfully from the file, then it prints a message. Otherwise, it prints an error message. To ensure that the user has seen the message, the user has to press the “enter” button to proceed.
29. [getBlockName]: this function has as a parameter an integer called “blockType” and returns a string representing the name of the block. This function uses a switch statement to understand the “blockType”. If the user enters strings, such as AIR, LEAVES, STONE, then the program returns the name of that specific block. If the user doesn’t enter the name of any known “blockType”, then the program returns “Unknown”.
30. [displayLegend]: this function prints the legend or different block types, using the System.out.println statement. These symbols \u00A7\u00A7 represent the colors that are printed with the different block types, helping users to identify them in the game. This function provides information, such as the appearance of the empty blocks, wood blocks, etc.
31. [displayInventory]: this function displays the player’s inventory. If the inventory is empty, then it prints “Empty” in yellow, otherwise it counts the occurrence of the different block types and prints a list of the block types with the number of occurrences. After that the program prints the crafted items. If there are no crafted item, then the program displays in yellow color “None”. If there are crafted items name and the color of the crafted item. At the end of the function, there is an empty line printed.
32. [getBlockColor]: this function takes an integer as an input and returns an ANSI color. It is used to apply different colors to the types of blocks.
33. [waitforEnter]: this function waits the user to press the “enter” button. It outputs the message “Press Enter to continue”. When the user presses the button, the 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 in the game. If it is a known craftedItem it returns a string, otherwise it returns "Unknown".
35. [getCraftedItemColor]: this function takes as an input an integer craftedItem and returns the ANSI color. If it is an unknown craftedItem 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 secret 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 in order to make it true the player needs to type the command "unlock".
Once that is 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, in order to get the sequence right, and that is "open".
"Open" has to 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 the 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.

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 in order 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 in order 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.

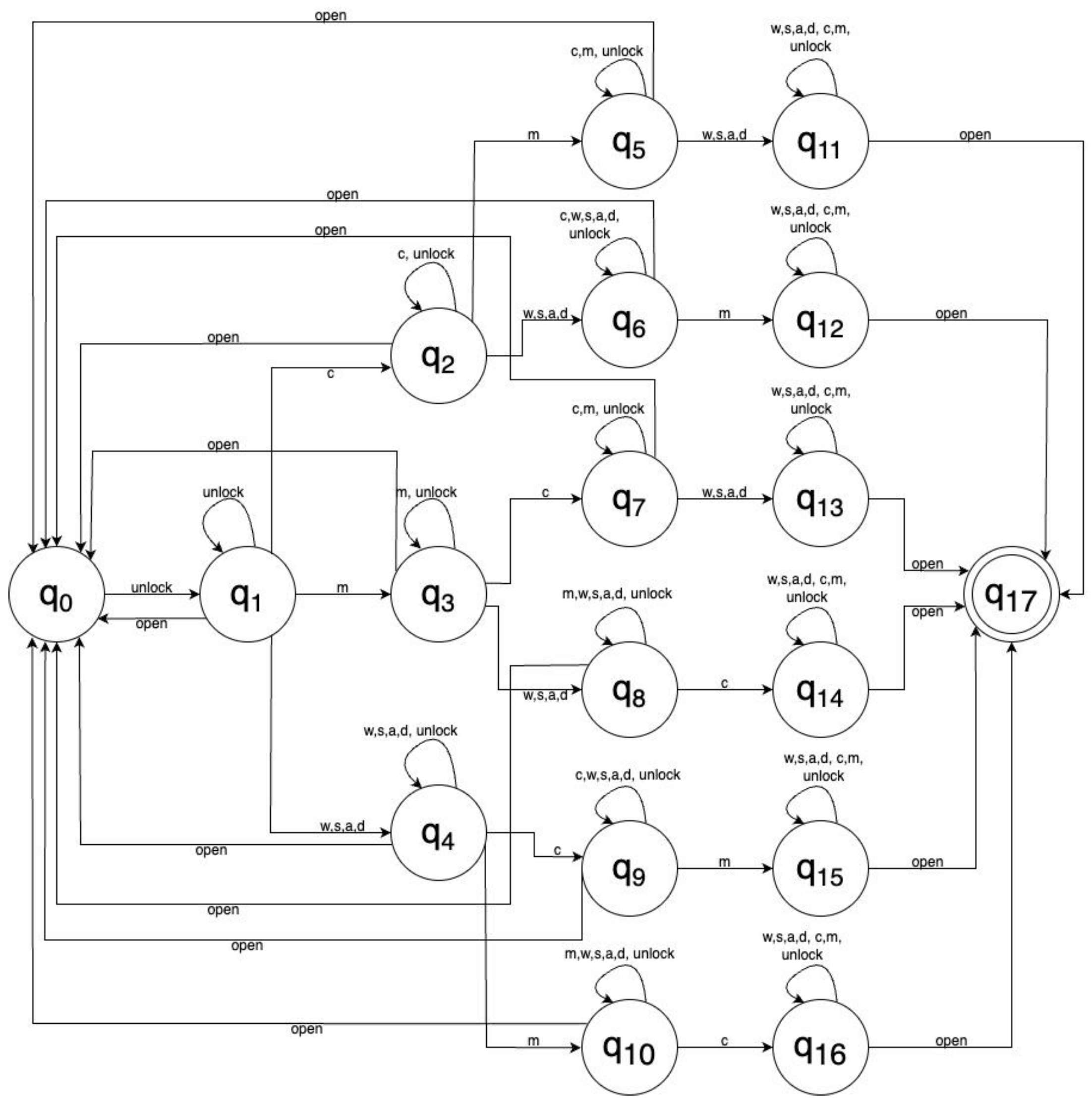
The user must follow this sequence of state changes for reaching the accepting state, if the user enters "open" before completing the sequence, the NFA returns to initial state.

Transition Function Table (FSA Secret Door):

	unlock	c	m	w,a,s,d	open
q_0	q_1	q_0	q_0	q_0	q_0
q_1	q_1	q_2	q_3	q_4	q_0
q_2	q_2	q_2	q_5	q_6	q_0
q_3	q_3	q_7	q_3	q_8	q_0
q_4	q_4	q_9	q_{10}	q_4	q_0
q_5	q_5	q_5	q_5	q_{11}	q_0
q_6	q_6	q_6	q_{12}	q_6	q_0
q_7	q_7	q_7	q_7	q_{13}	q_0
q_8	q_8	q_{14}	q_8	q_8	q_0
q_9	q_9	q_9	q_{15}	q_9	q_0
q_{10}	q_{10}	q_{16}	q_{10}	q_{10}	q_0
q_{11}	q_{11}	q_{11}	q_{11}	q_{11}	q_{17}
q_{12}	q_{12}	q_{12}	q_{12}	q_{12}	q_{17}

q_{13}	q_{13}	q_{13}	q_{13}	q_{13}	q_{17}
q_{14}	q_{14}	q_{14}	q_{14}	q_{14}	q_{17}
q_{15}	q_{15}	q_{15}	q_{15}	q_{15}	q_{17}
q_{16}	q_{16}	q_{16}	q_{16}	q_{16}	q_{17}
q_{17}	q_{17}	q_{17}	q_{17}	q_{17}	q_{17}

<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).

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 (For Final Submission)

[Provide details on the new block types, craft recipes, and their integration into the game. Include code snippets where appropriate]

```
[Provide Java code here]
```

7 Interacting with Flags API (For Final Submission)

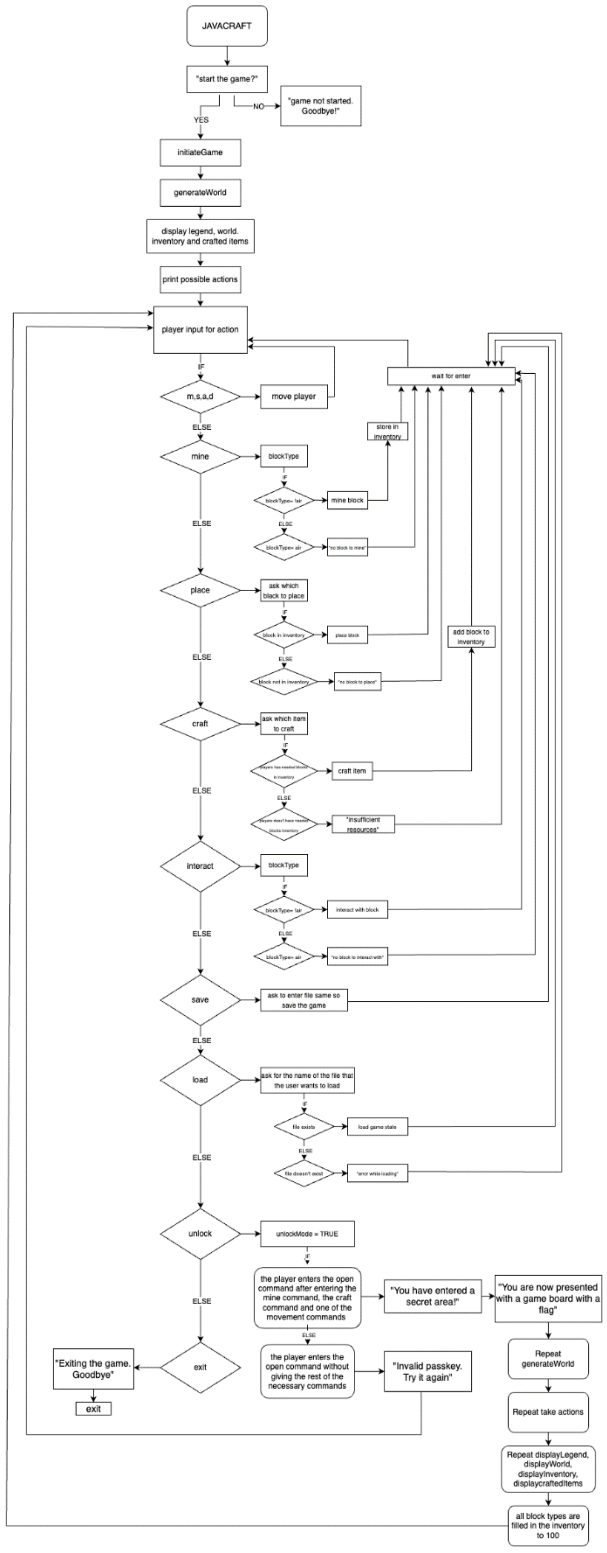
[Details on Flags API exploration and flag rendering on the grid.]

8 Conclusion (For Final Submission)

[Provide a summary of achievements, challenges, and learnings.]

9 Appendix

<u>Flowchart Game Workflow:</u>


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

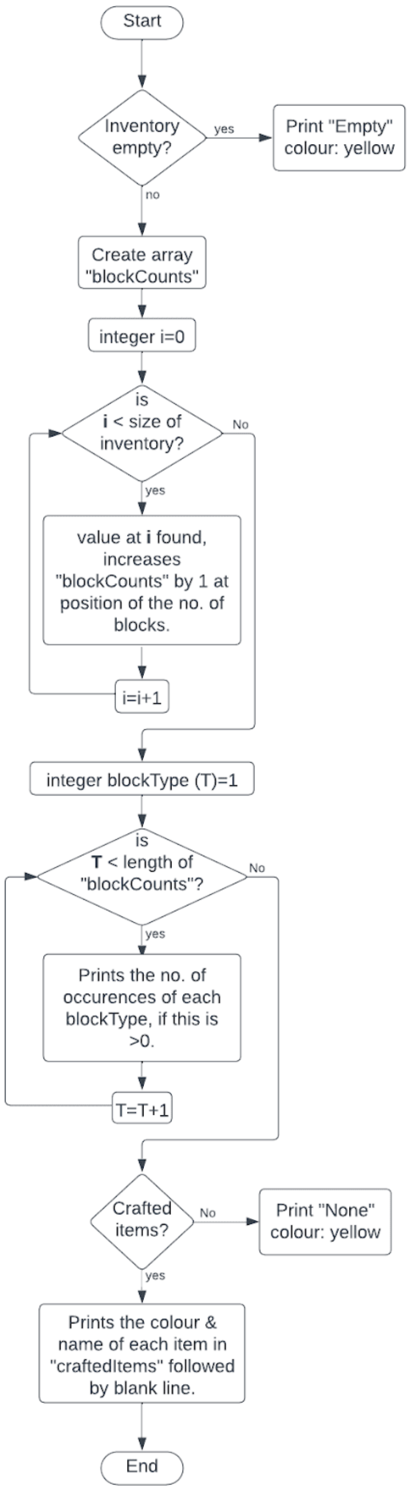
Pseudocode Game Workflow:

```
[
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
```

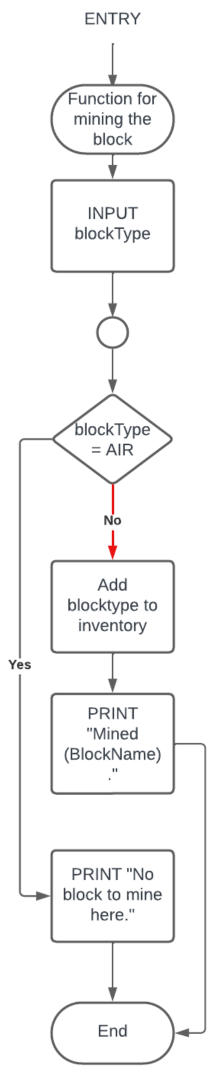
]


Start
Inventory empty?
yes
Print "Empty" colour: yellow
no
Create array "blockCounts"
integer i=0
is i < size of inventory?
No
yes
value at i found, increases "blockCounts" by 1 at position of the no. of blocks.
i=i+1
integer blockType (T)=1
is T < length of "blockCounts"?
No
yes
Prints the no. of occurences of each blockType, if this is >0.
T=T+1
Crafted items?
No
Print "None" colour: yellow
yes
Prints the colour & name of each item in "craftedItems" followed by blank line.
End

1. <u>Flowchart & pseudocode displayInventory:</u>

```
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
```

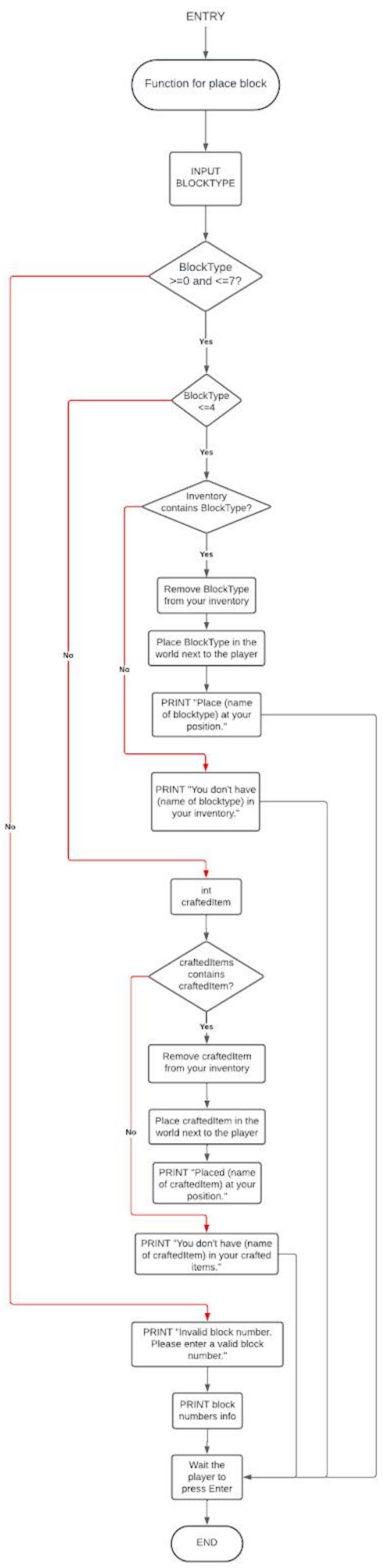
2. <u>Flowchart & pseudocode mineBlock:</u>


ENTRY
Function for mining the block
INPUT blockType
blockType = AIR
No
Yes
Add blocktype to inventory
PRINT "Mined (BlockName) ."
PRINT "No block to mine here."
End

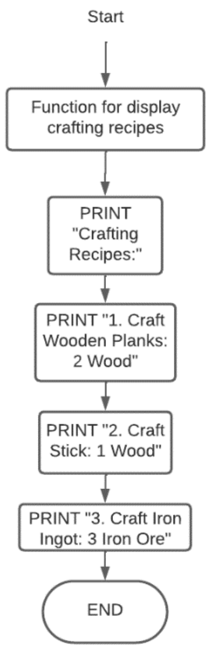
```
Void mineBlock
        Define the blockType within the playerX and playerY
        If the block is not air
                Add the blockType
                Substitute the block type into air
                Print (“Mined (blockType) .”
        Else
                Print “No block to mine here.”
        Wait for the player to press Enter
        End
```

3. <u>Flowchart & pseudocode placeBlock:</u>

```
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
```

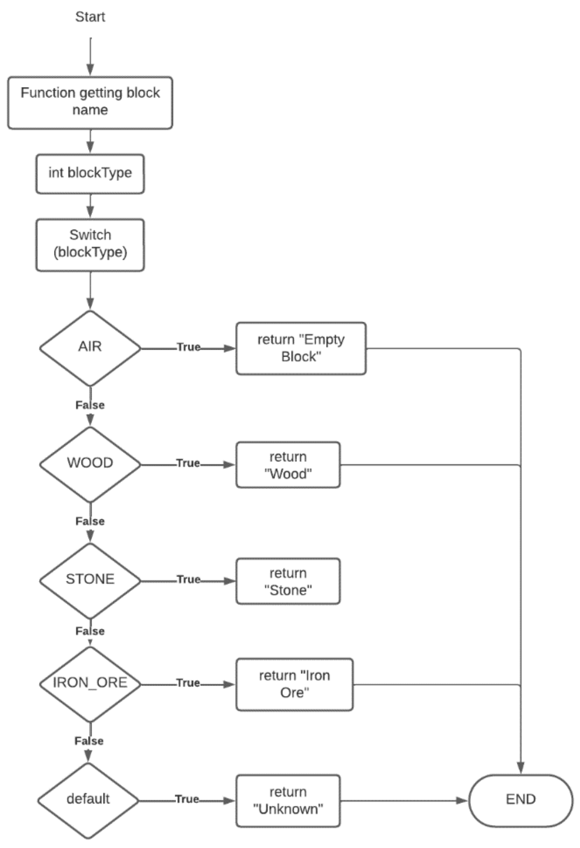

ENTRY
Function for place block
INPUT BLOCKTYPE
BlockType >=0 and <=7?
Yes
No
BlockType <=4
Yes
No
Inventory contains BlockType?
Yes
No
Remove BlockType from your inventory
Place BlockType in the world next to the player
PRINT "Place (name of blocktype) at your position."
PRINT "You don't have (name of blocktype) in your inventory."
int craftedItem
craftedItems contains craftedItem?
Yes
No
Remove craftedItem from your inventory
Place craftedItem in the world next to the player
PRINT "Placed (name of craftedItem) at your position."
PRINT "You don't have (name of craftedItem) in your crafted items."
PRINT "Invalid block number. Please enter a valid block number."
PRINT block numbers info
Wait the player to press Enter
END

4. <u>Flowchart & pseudocode displayCraftingRecipes:</u>


Start
Function for display crafting recipes
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

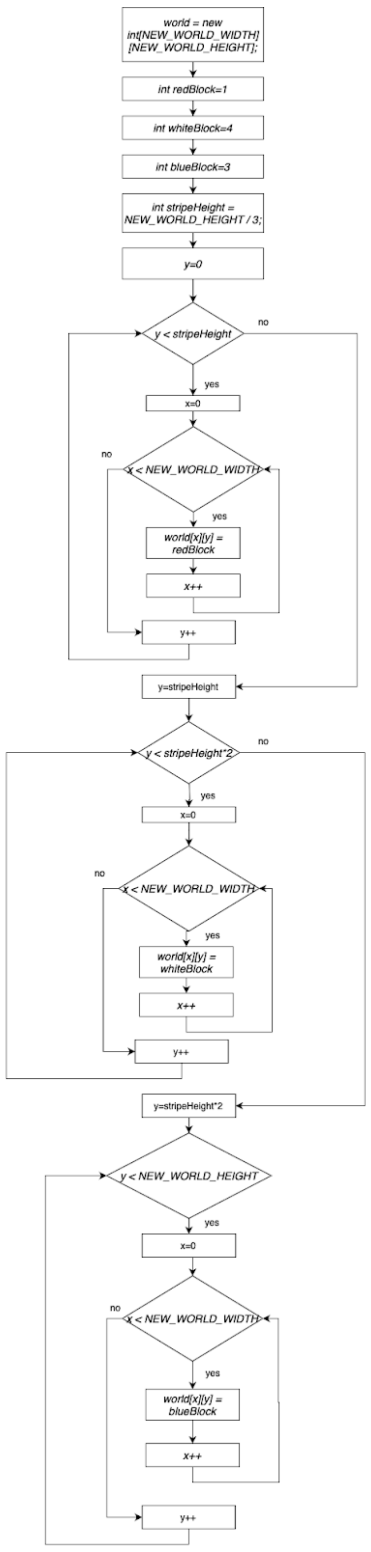
```
Void displayCraftingRecipes
        Print (“Crafting Recipes.”)
        Print (“1. Craft Wooden Planks: 2 Wood”)
        Print (“2. Craft Stick: 1 Wood)
        Print (“3. Craft Iron Ingot: 3 Iron Ore”)
            End
```

5. <u>Flowchart & pseudocode getBlockName:</u>

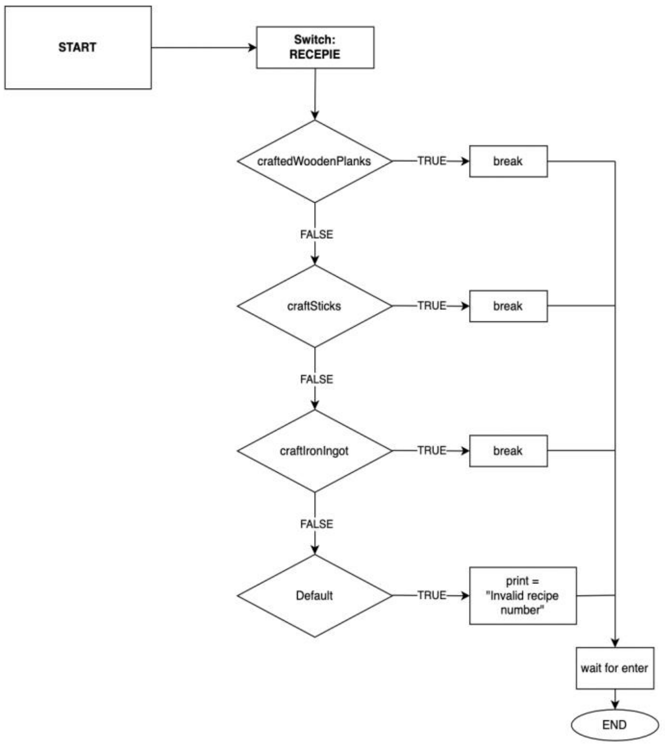


```
String getBlockName
int blockType
Switch (blockType)
        Case AIR:
                Return "Empty Block"
        Case WOOD:
                Return "Wood"
        Case STONE:
                Return "Stone"
        Case IRON_ORE:
                Return "Iron Ore"
        Default:
              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

6. Flowchart & pseudocode generateEmptyWorld:



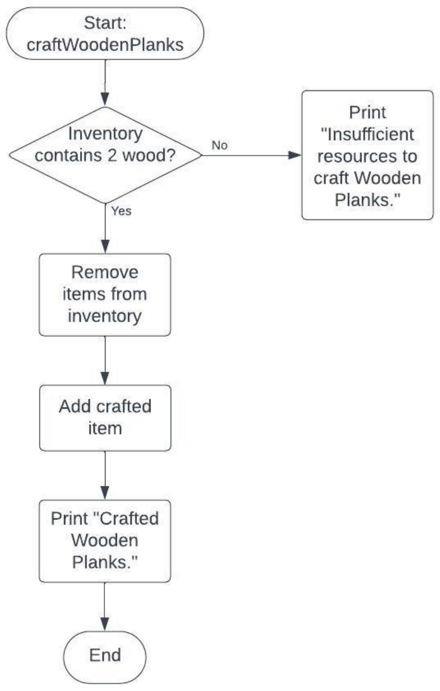
```
function generateEmptyWorld():
    create an empty 2D array called world with dimensions NEW_WORLD_WIDTH by NEW_WORLD_HEIGHT
    set redBlock to 1
    set whiteBlock to 4
    set blueBlock to 3
    set stripeHeight to NEW_WORLD_HEIGHT divided by 3  // Divide the height into three equal parts
    // 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:

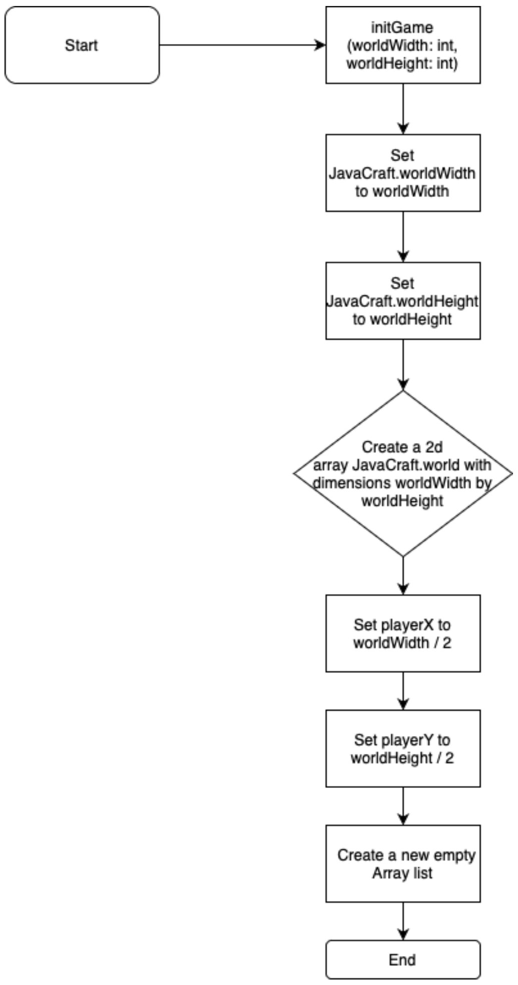
```
void CraftItem
        Switch: four possible cases
        Case 1:  call
    craftWoodenPlanks
    function
        Case 2: call craftSticks
    function
        Case 3: call craftIronIngot
        default: print “invalid
    recipe number”
Wait for the player to write Enter
End
```

8. Flowchart & pseudocode craftWoodenPlanks:



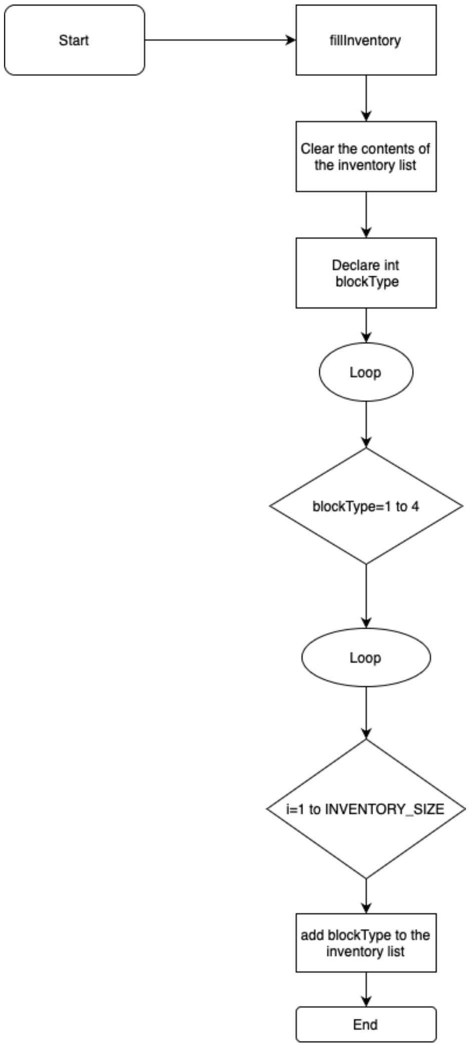
Start: craftWoodenPlanks
Inventory contains 2 wood?
No
Print "Insufficient resources to craft Wooden Planks."
Yes
Remove items from inventory
Add crafted item
Print "Crafted Wooden Planks."
End

```
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
```


Start
initGame (worldWidth: int, worldHeight: int)
Set JavaCraft.worldWidth to worldWidth
Set JavaCraft.worldHeight to worldHeight
Create a 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
End

9. <u>Flowchart & pseudocode initGame;</u>

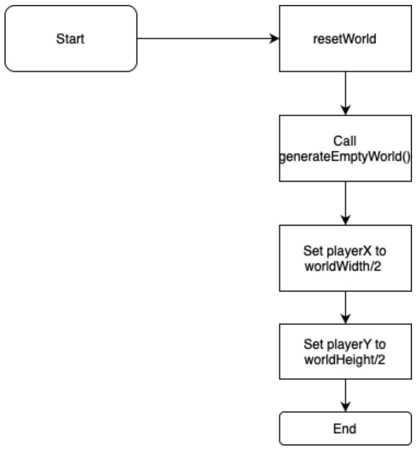
```
function initGame(worldWidth: int, worldHeight: int):
    Set JavaCraft.worldWidth to worldWidth
    Set JavaCraft.worldHeight to worldHeight
    Create a new 2D array JavaCraft.world with dimensions
worldWidth by worldHeight
    Set playerX to worldWidth / 2
    Set playerY to worldHeight / 2
    Create a new empty Array list
```



10. Flowchart & pseudocode fill_Inventory;

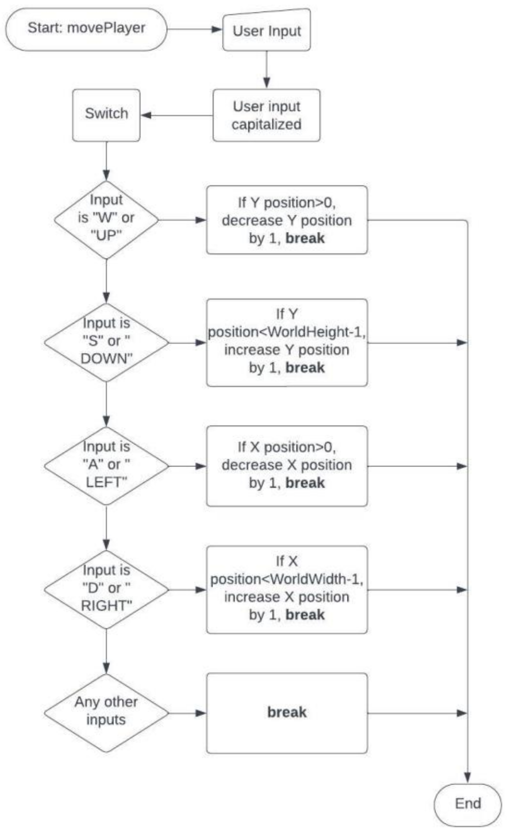
```
fillInventory()
   Clear the contents of the inventory list
   loop for blockType from 1 to 4
     loop for i from 1 to INVENTORY_SIZE
        add blockType to the inventory list
     End loop
   End loop
End
```

11. Flowchart & pseudocode resetWorld:


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

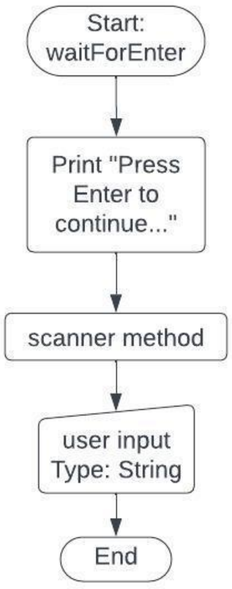
```
resetWorld()
   Call generateEmptyWorld()
   Set playerX to worldWidth/2
   Set playerY to worldHeight/2
End
```

12. Flowchart & pseudocode movePlayer:

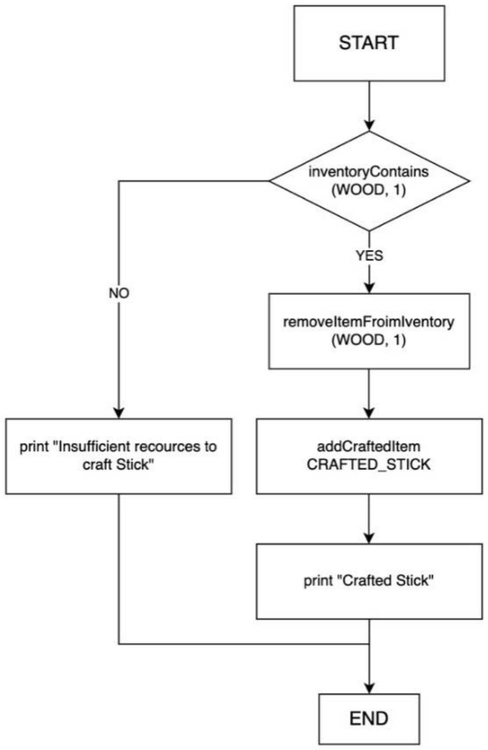

Start: movePlayer
User Input
User input capitalized
Switch
Input is "W" or "UP"
If Y position>0, decrease Y position by 1, **break**
Input is "S" or " DOWN"
If Y position<WorldHeight-1, increase Y position by 1, **break**
Input is "A" or " LEFT"
If X position>0, decrease X position by 1, **break**
Input is "D" or " RIGHT"
If X position<WorldWidth-1, increase X position by 1, **break**
Any other inputs
break
End

```
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:



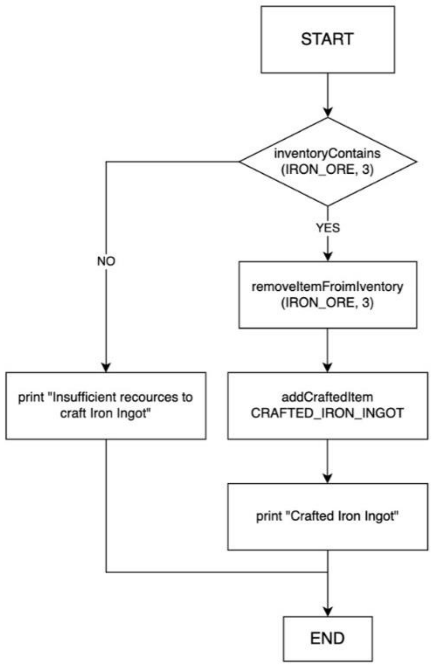
```
Print "Press Enter to continue…"
Read user input, type: String.
End.
```


START
inventoryContains (WOOD, 1)
YES
NO
removeItemFroimIventory (WOOD, 1)
print "Insufficient recources to craft Stick"
addCraftedItem CRAFTED_STICK
print "Crafted Stick"
END

14. Flowchart & pseudocode craftSticks:

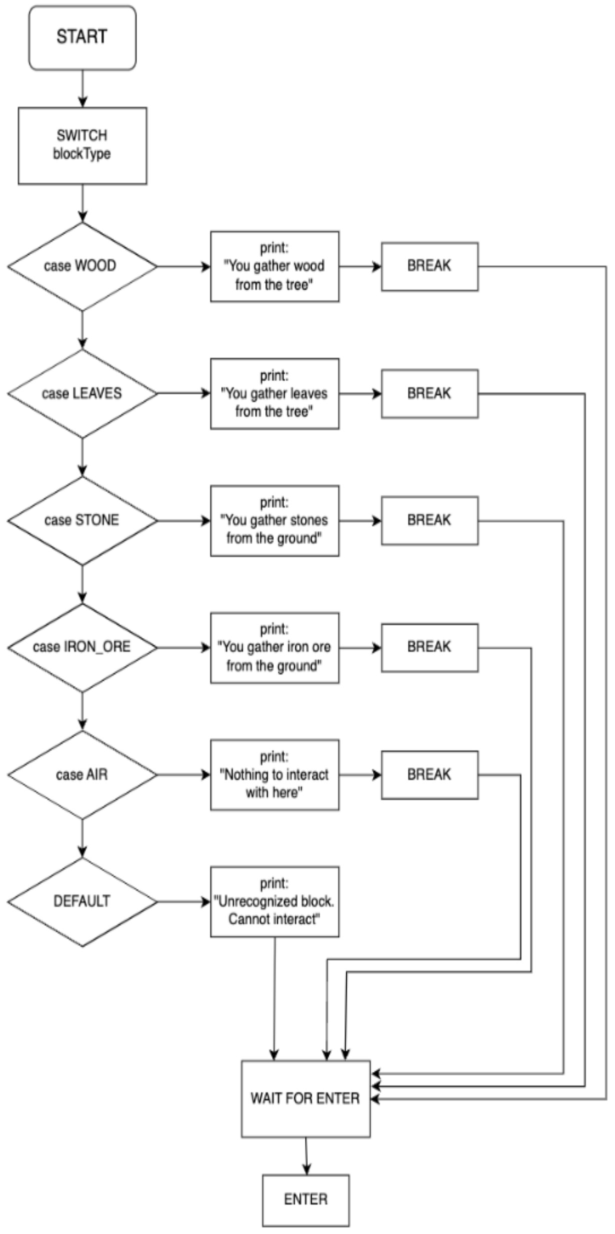
```
If the inventory contains one items of WOOD
        remove one items of WOOD from the        inventory
        add to the inventory one items of    CRAFTED_STICK
        print "Crafted Stick."
Else
          print "Insufficient resources to craft
        Stick."
```

15. Flowchart & pseudocode craftIronIngot:


START
inventoryContains (IRON_ORE, 3)
YES
NO
removeItemFroimIventory (IRON_ORE, 3)
print "Insufficient recources to craft Iron Ingot"
addCraftedItem CRAFTED_IRON_INGOT
print "Crafted Iron Ingot"
END

```
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. <u>Flowchart & pseudocode interactWithWorld:</u>


START
SWITCH blockType
case WOOD
print: "You gather wood from the tree"
BREAK
case LEAVES
print: "You gather leaves from the tree"
BREAK
case STONE
print: "You gather stones from the ground"
BREAK
case IRON_ORE
print: "You gather iron ore from the ground"
BREAK
case AIR
print: "Nothing to interact with here"
BREAK
DEFAULT
print: "Unrecognized block. Cannot interact"
WAIT FOR ENTER
ENTER

```
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

1. Source Name - Description
2. …