JavaCraft

Full Project Specification

Sunday, October 22, 2023

Table of contents

1 Introduction

Welcome to JavaCraft, a multifaceted text-based Java game. This project is a holistic academic exercise in computer science, logical thinking, and collaboration. Working in teams of four, you will use your creativity, analytical skills, and technical skills. Please see the Project Overview for more information on the intended learning goals of the project.

2 Tasks

2.1 JavaCraft's WorkFlow

JavaCraft is a (*relatively*) complex Java program that brings together over 35 functions to create a diverse gameplay experience. This task is your chance to understand the core functionalities of JavaCraft, putting them together to form the game's overarching structure.

2.1.1 Sub-Tasks

- **Create a Flowchart**: Develop a detailed flowchart that broadly explains the gameplay. This flowchart should contain the main components of the game, illustrating how they interact and connect to create the gaming experience.
- **Write a Pseudocode**: Complement the flowchart with a pseudocode that outlines the main game logic, providing a textual representation of the game's flow and structure.

2.1.2 Deliverables:

- **Flowchart & Pseudocode Submission**: Include the flowchart and pseudocode in the final PDF document, showcasing your understanding of the game's overall architecture[1].

2.2 Functionality Exploration

Dive deep into the codebase of JavaCraft by exploring the 35+ functions that constitute the game's core:

[1]There are two main submissions for this project, you can see the details of which task is part of which submission on the Submissions page.

2.2.1 Sub-Tasks

For each function, perform the following detailed analysis:

1. **Identify Key Components**: Recognize the essential parts of the function, focusing on what makes each function unique and how it contributes to the overall game.
2. **Recognize Patterns**: Delve into the code of each function to find patterns or repetitions, understanding the patterns.
3. **Break Down the Function**: Divide the function into logical steps or subproblems, providing a step-by-step analysis that reveals the function's inner workings.
4. **Represent the Function**: Develop a flowchart and pseudocode for each function, translating the textual analysis into visual and algorithmic representations[2].

2.2.2 Deliverables:

- **Project Documentation**: Prepare a project documentation that includes descriptions of **at least 35** functions within the code. Create flowcharts and pseudocodes for **at least 15** of these functions, showcasing your analytical skills and understanding of the game's mechanics. This documentation will be a part of the provisional and final PDF submission.

2.3 Finite State Automata (FSA) Design

Your next task is to design a Finite State Automata (FSA) for the secret (hidden) door logic:

2.3.1 Sub-Tasks

1. **Study the Secret Door Logic**: Examine the secret door's functionality in the provided code, understanding how the secret door operates within the game's context.
2. **Create an FSA**: Design an FSA (DFA or NFA) that mirrors the behavior of the secret door, ensuring that the FSA accurately captures every aspect of the secret door's functionality.
3. **Illustrate the FSA**: Draw a clear and well-structured FSA diagram, labeling states and transitions, making sure that the diagram is in line with what was covered in the lectures.
4. **Describe the FSA**: Write an explanation of the FSA's operation, detailing the conditions that lead to state changes, and how the FSA represents the secret door logic within the game.

[2] In 2.1, you designed a flowchart and pseudocode for the overall program, here you have to do it for individual functions in the program.

2.3.2 Deliverables:

- **FSA Documentation**: Include a detailed explanation of the secret door logic, a well-designed FSA, and concise documentation of the FSA's functioning and trigger conditions. This documentation will be part of the provisional and final PDF document, reflecting your ability to apply theoretical concepts to practical game elements.

2.4 Extending the Game Code

In this task, you apply your creativity and technical skills by extending JavaCraft's existing code:

2.4.1 Sub-Tasks

1. **Create New Block Types**: Design at least 2 unique block types, considering their defining attributes, interactions, appearances, and how they will integrate into the existing game dynamics.
2. **Formulate a New Craft Recipe**: Develop at least 1 crafting recipe that creatively integrates existing blocks to yield a new and distinct block.
3. **Integrate the Extension**: Incorporate the new block types and craft recipe into the existing game code, ensuring that the new elements work with the existing gameplay mechanics.

2.4.2 Deliverables:

- **Extension Documentation**: Provide a detailed and comprehensive blueprint of the new block types and craft recipe. Include this in the final PDF document, showcasing your ability to innovate within an existing codebase.

2.5 Git Usage

As a part of the project, you will learn Git, a powerful tool for version control, and apply it to the project.

2.5.1 Sub-Tasks

1. **Create a branch on the JavaCraft repository**: Create a branch on the JavaCraft repository on GitLab for your group, all the team members should have access to it. Re-read the lab from week 2 if you are still struggling with this.
2. **Make at least 1 commit per person**: Get each team member to make at least 1 commit on your branch. The commit can be as simple as adding their names to Readme file or adding a comment on the JavaCraft code.
3. **Handle Changes and Conflicts**: Use Git's capabilities to manage modifications and resolve conflicts.
4. **Upload the Final Code [either on GitLab or Canvas]**: Upload the completed code, including 2 new block types and 1 craft recipe, to the your Git branch, only one person needs to do it in your group [3].

2.5.2 Deliverables:

- **Git Summary**: Include a summary of your Git usage in the final PDF document. Detail the repository setup, task allocation, change management, and final submission.

2.6 Interacting with Flags API

Once you are done with unlocking the secret door and your intermediary submission[4], you can move on to the last part. The challenge in this part is interacting with the Flags API (flag.ashish.nl). You use this API to know which flag you need to plot. You can choose different levels of difficulty.

2.6.1 Sub-Tasks

1. **Explore the Flags API**: Understand the Flags API (documentation at flag.ashish.nl), grasping how it can be utilized within the game.
2. **Draw the Flag**: Configure the difficulty (easy, medium, and hard) and draw the flag on a 50x30 grid, either manually or through automation[5].

[3] If you have trouble uploading your completed code to GitLab, you can also alternatively submit your code along with your provisional report on Canvas. Please make sure you upload `.java` file if you do not want to use Git.

[4] Please see the submission guidelines for more information on this.

[5] To get you started, I have plotted a flag in the hidden door. You can see how I have automated the process of plotting the flag.

2.6.2 Deliverables:

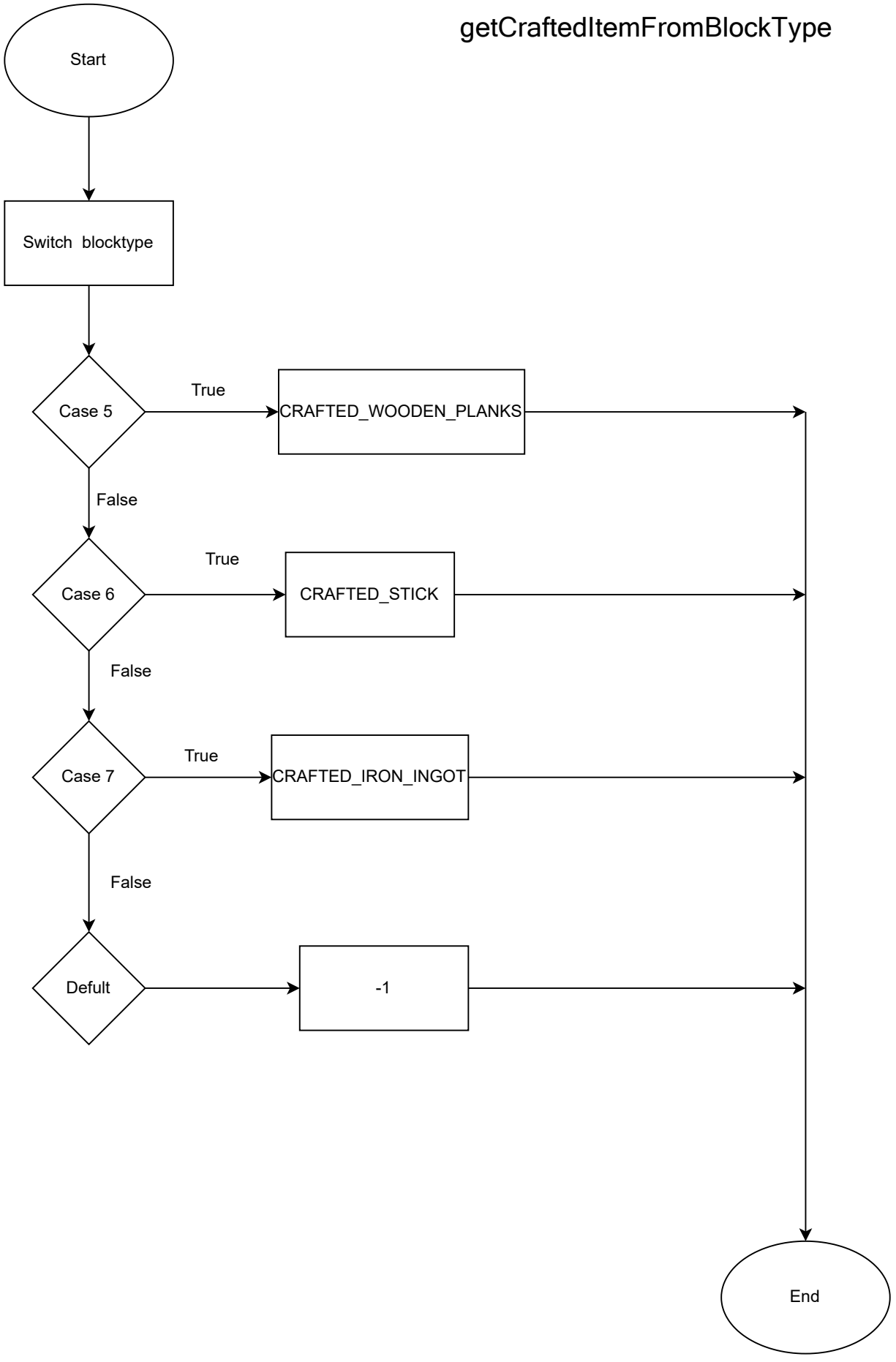
Flags Documentation: Detail the Flags API's functionality and your flag drawing process in the final PDF document. Include the chosen difficulty level and how the flag drawing integrates into the game, reflecting your ability to work with external APIs.

3 Final Deliverable

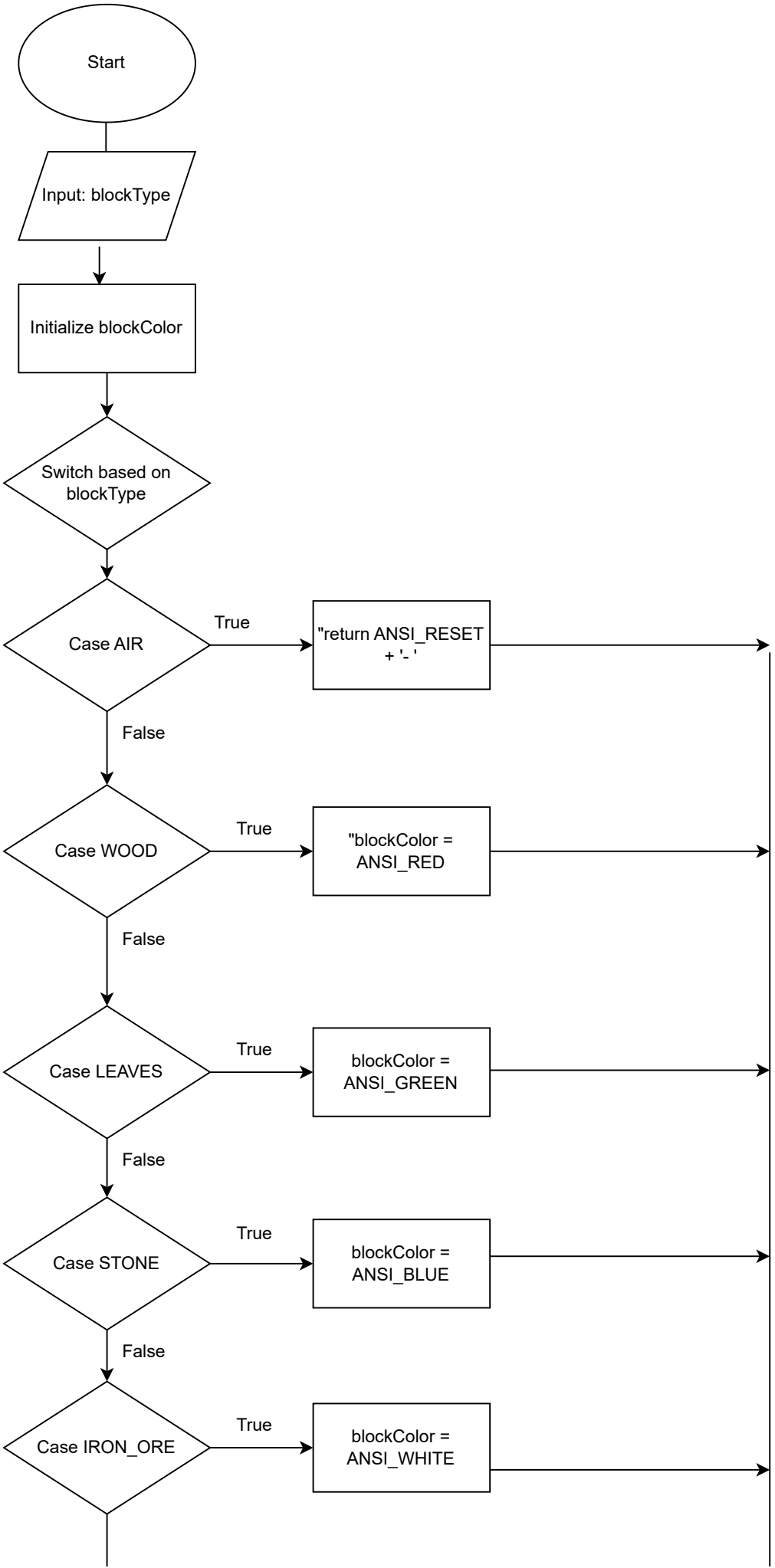
Compile a single PDF document, not exceeding 10 pages, detailing all the above steps in a structured and coherent format[6]. Include pseudocode and flowcharts for part 3 in the appendix, as lengthy as needed. Ensure that all deliverables mentioned are included in this document, except for the Git submission, which will be submitted separately to the TA's repository.

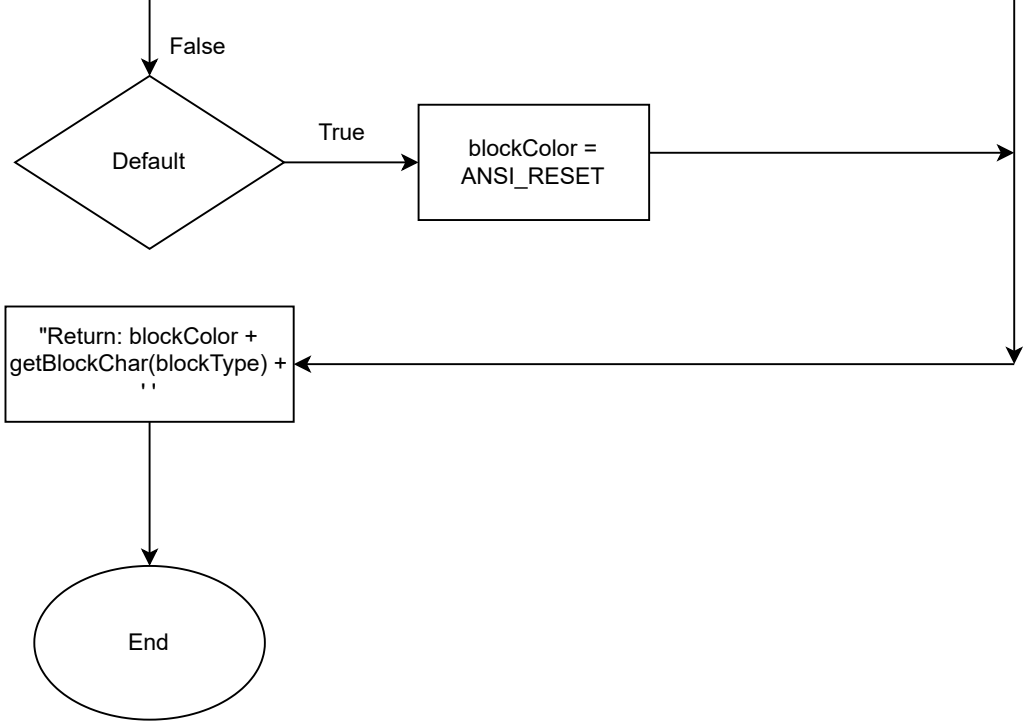
[6]You can use the template provided at bcs1110.ashish.nl.

Task 2.1.1 Flowcharts:

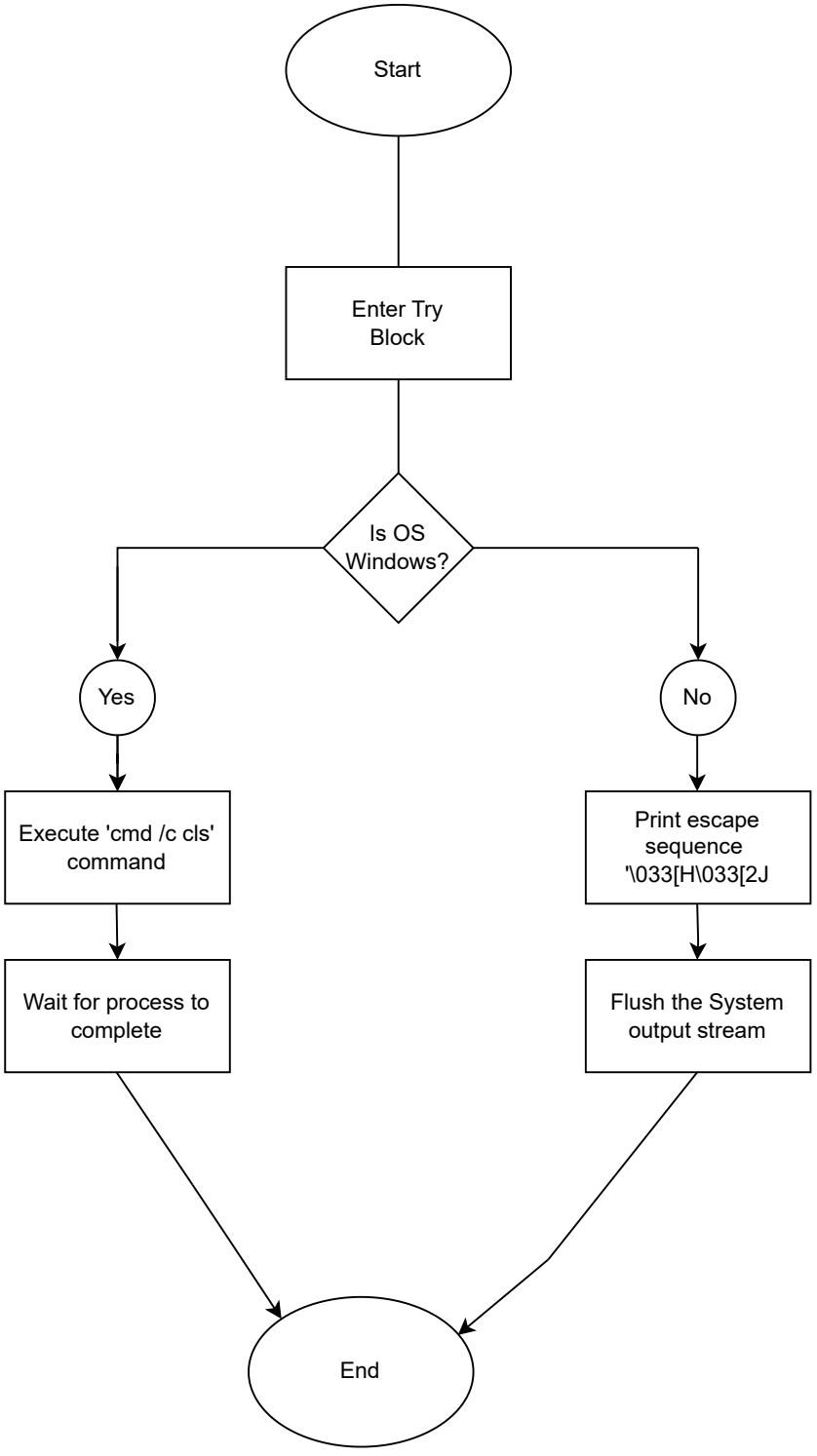


getBlockSymbol

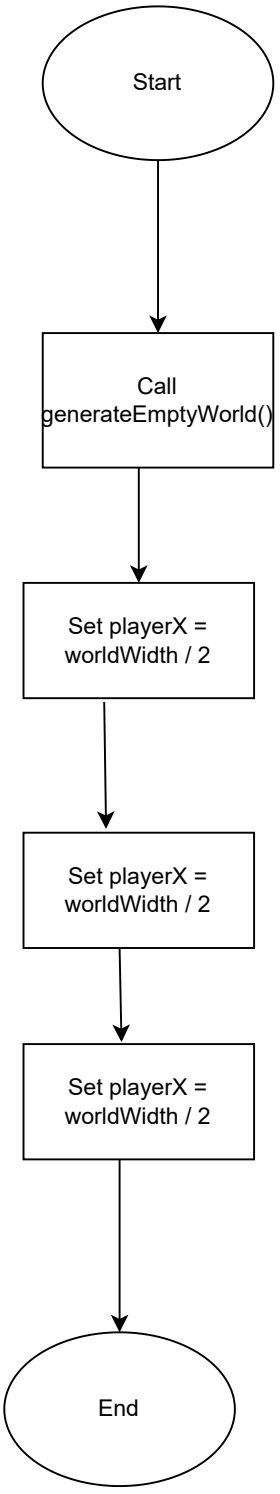



False
Default
True
blockColor =
ANSI_RESET
"Return: blockColor +
getBlockChar(blockType) +
' '
End

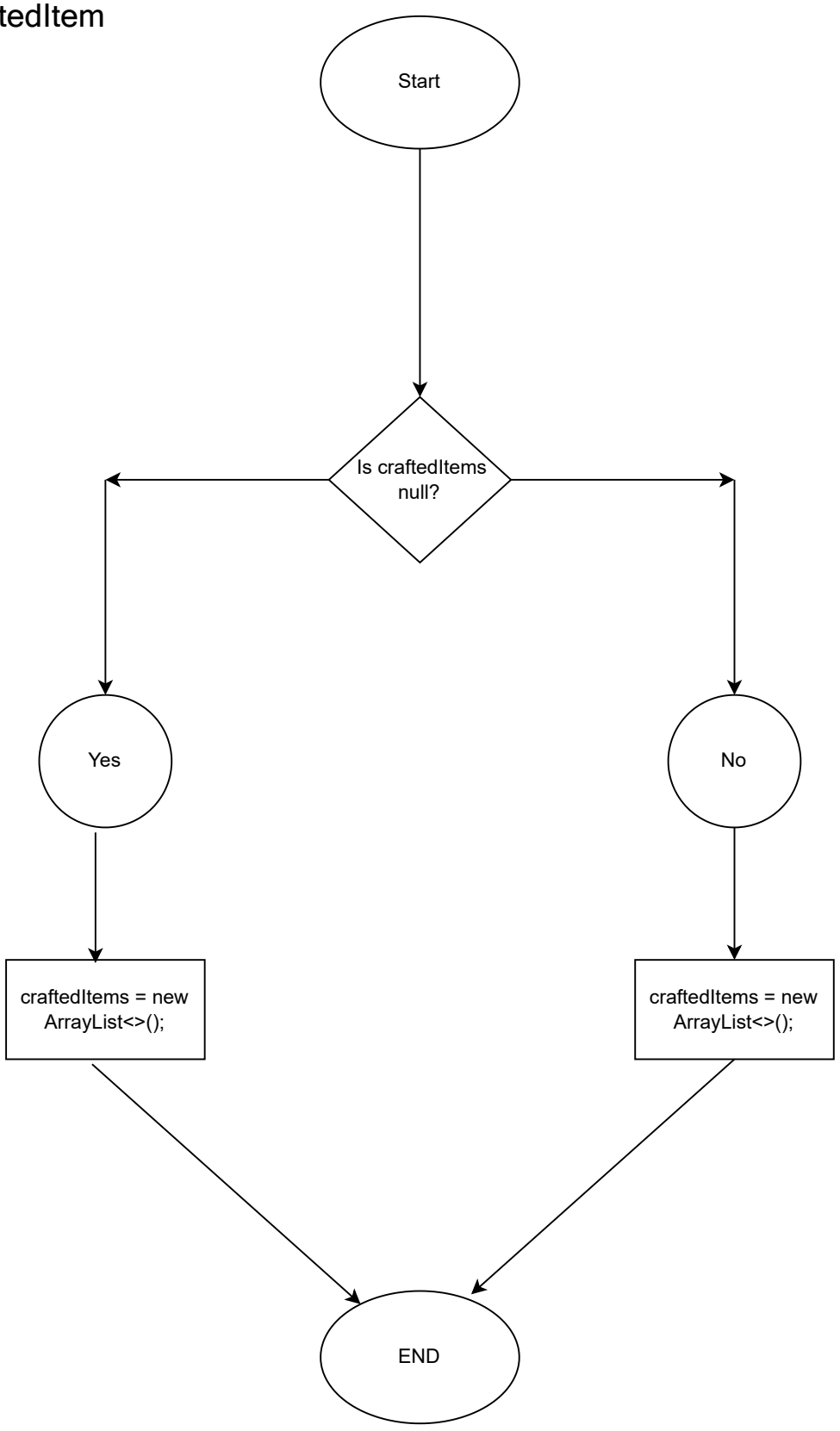
clearScreen()



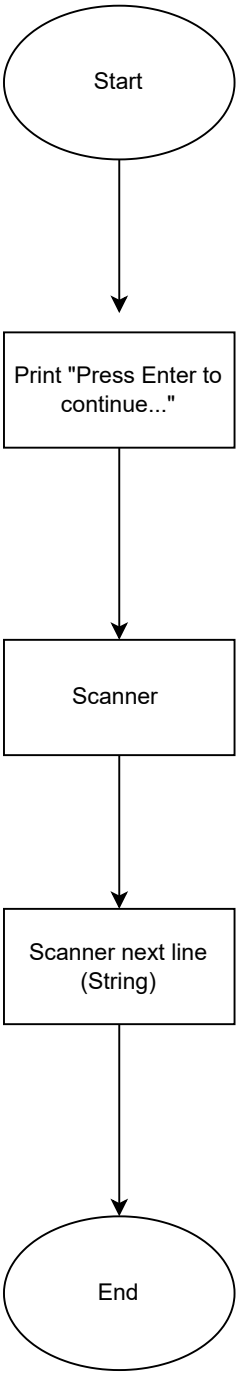
resetWorld


Start
Call generateEmptyWorld()
Set playerX = worldWidth / 2
Set playerX = worldWidth / 2
Set playerX = worldWidth / 2
End

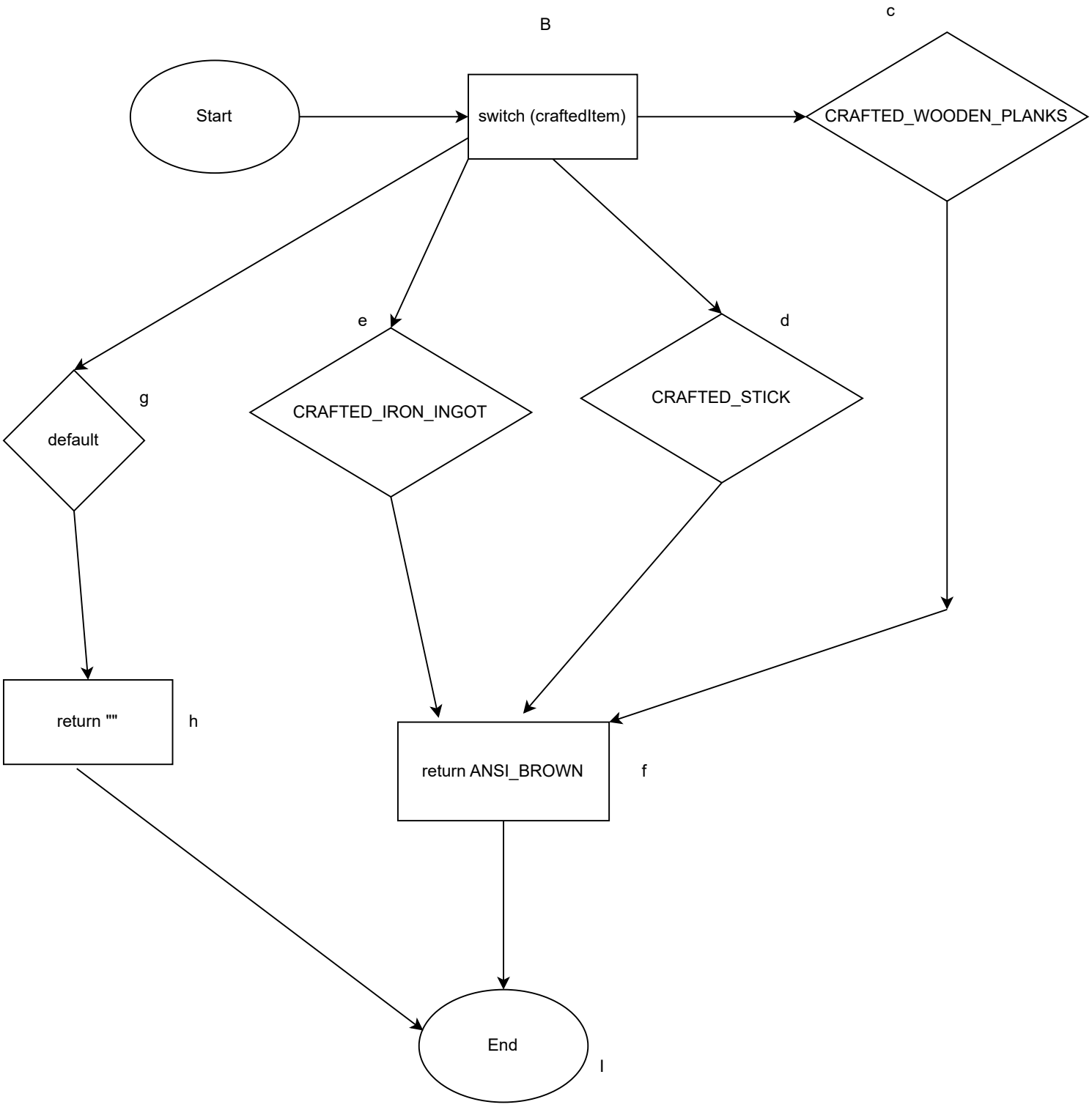
addCraftedItem



waitForEnter

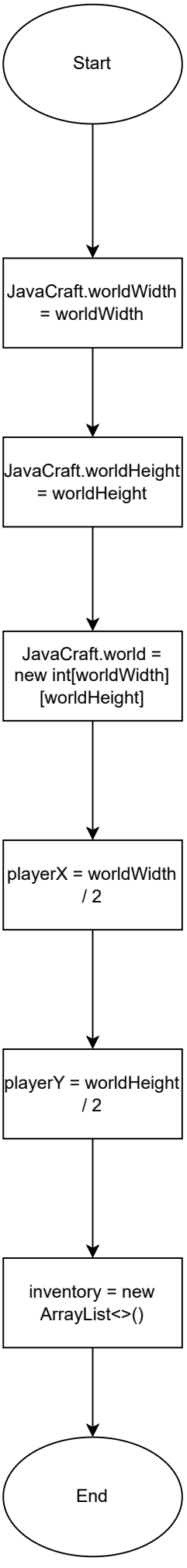


getCraftedItemColor

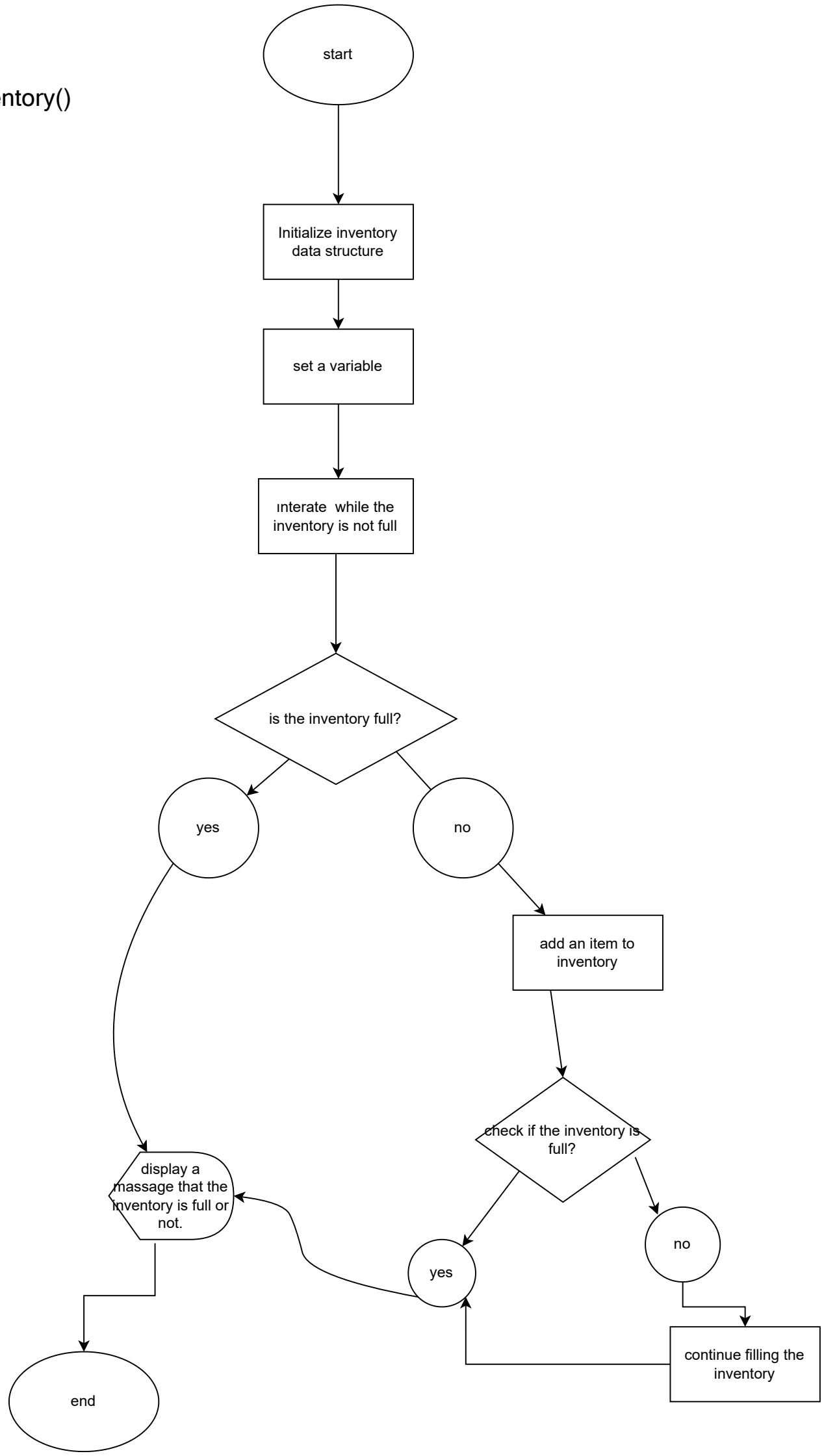



```
//      A --> B
    B -->C
    B --> D
    B --> E
    C --> F
    D --> F
    E --> F
   B -->  G
   G -->H
    F--> I
    H -->I
```

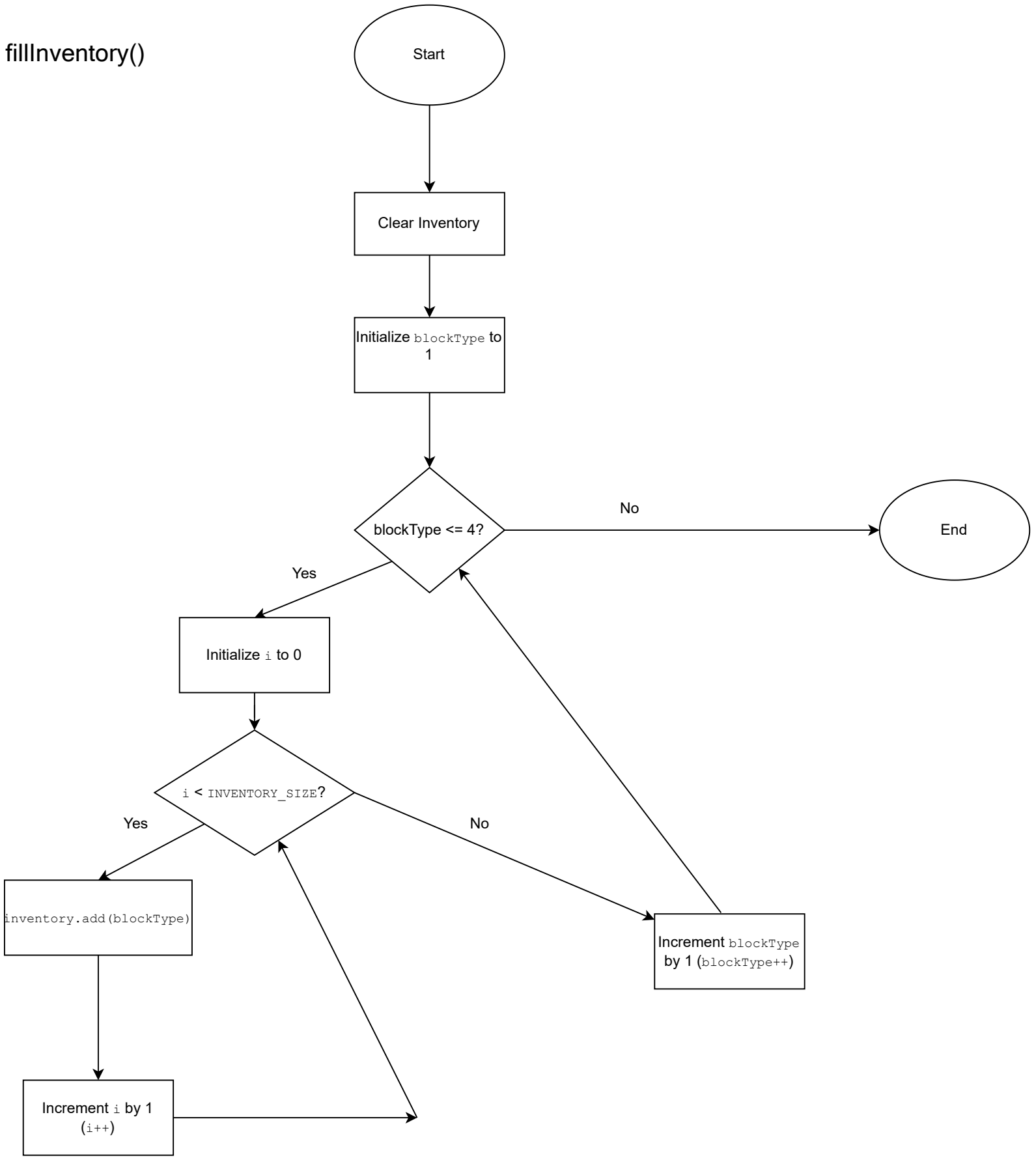
initGame


Start
JavaCraft.worldWidth = worldWidth
JavaCraft.worldHeight = worldHeight
JavaCraft.world = new int[worldWidth][worldHeight]
playerX = worldWidth / 2
playerY = worldHeight / 2
inventory = new ArrayList<>()
End

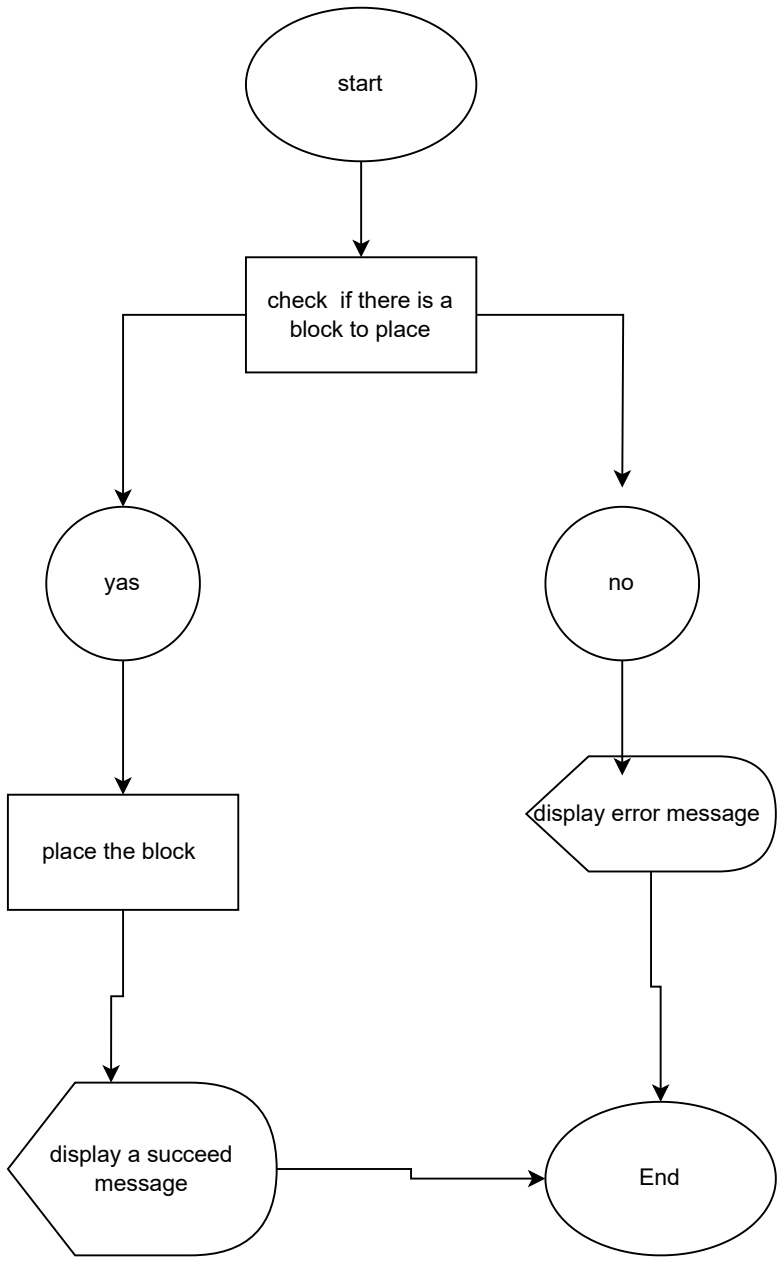
fillInventory()



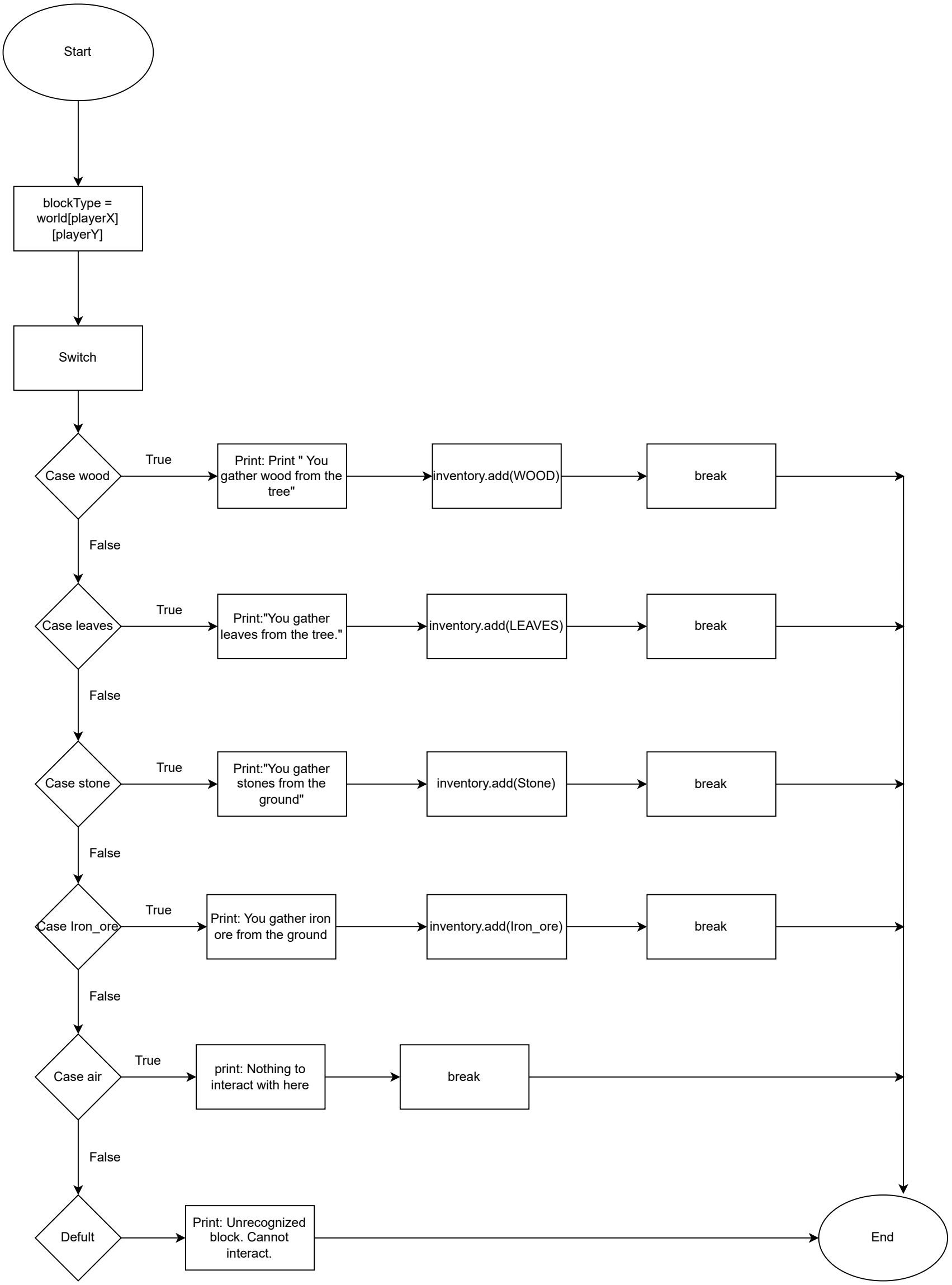
fillInventory()



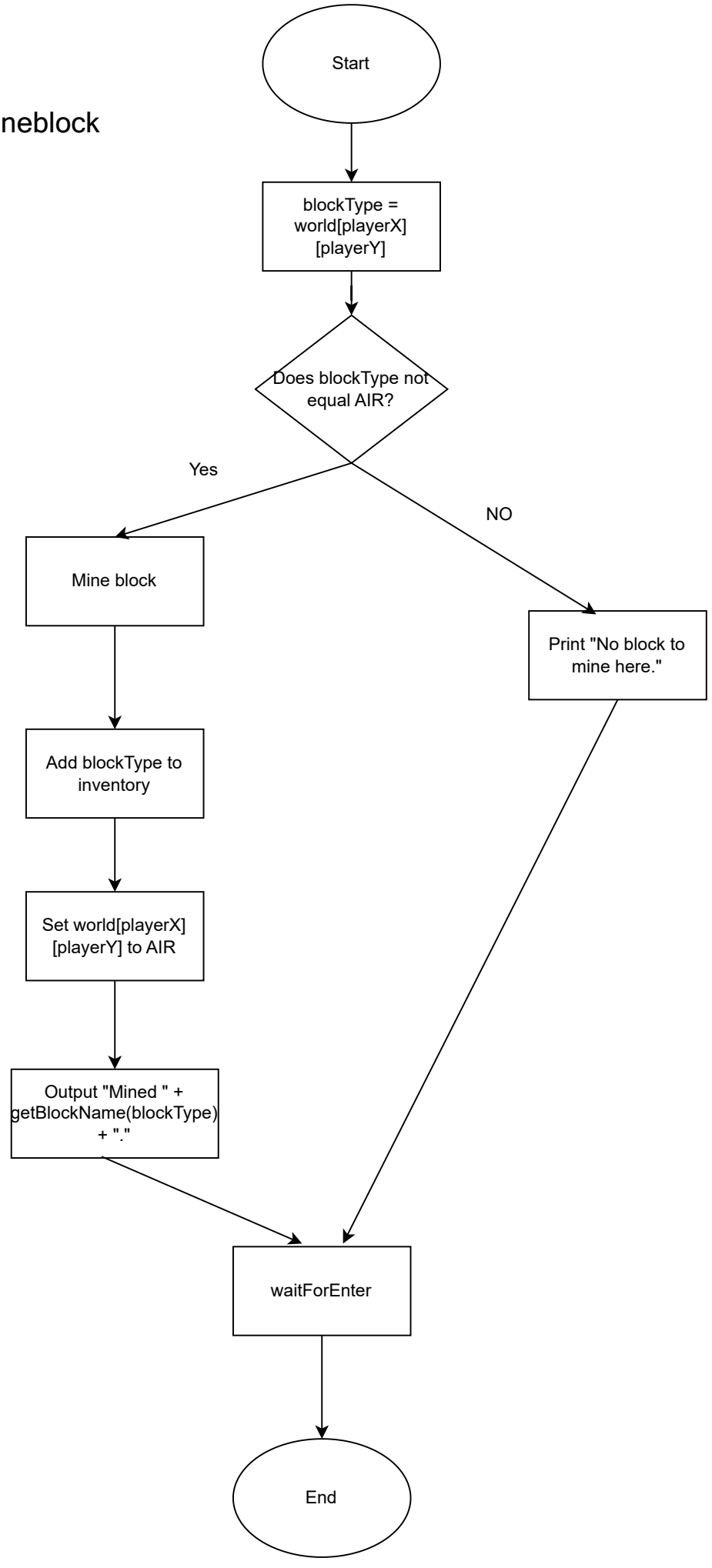
placeBlock()

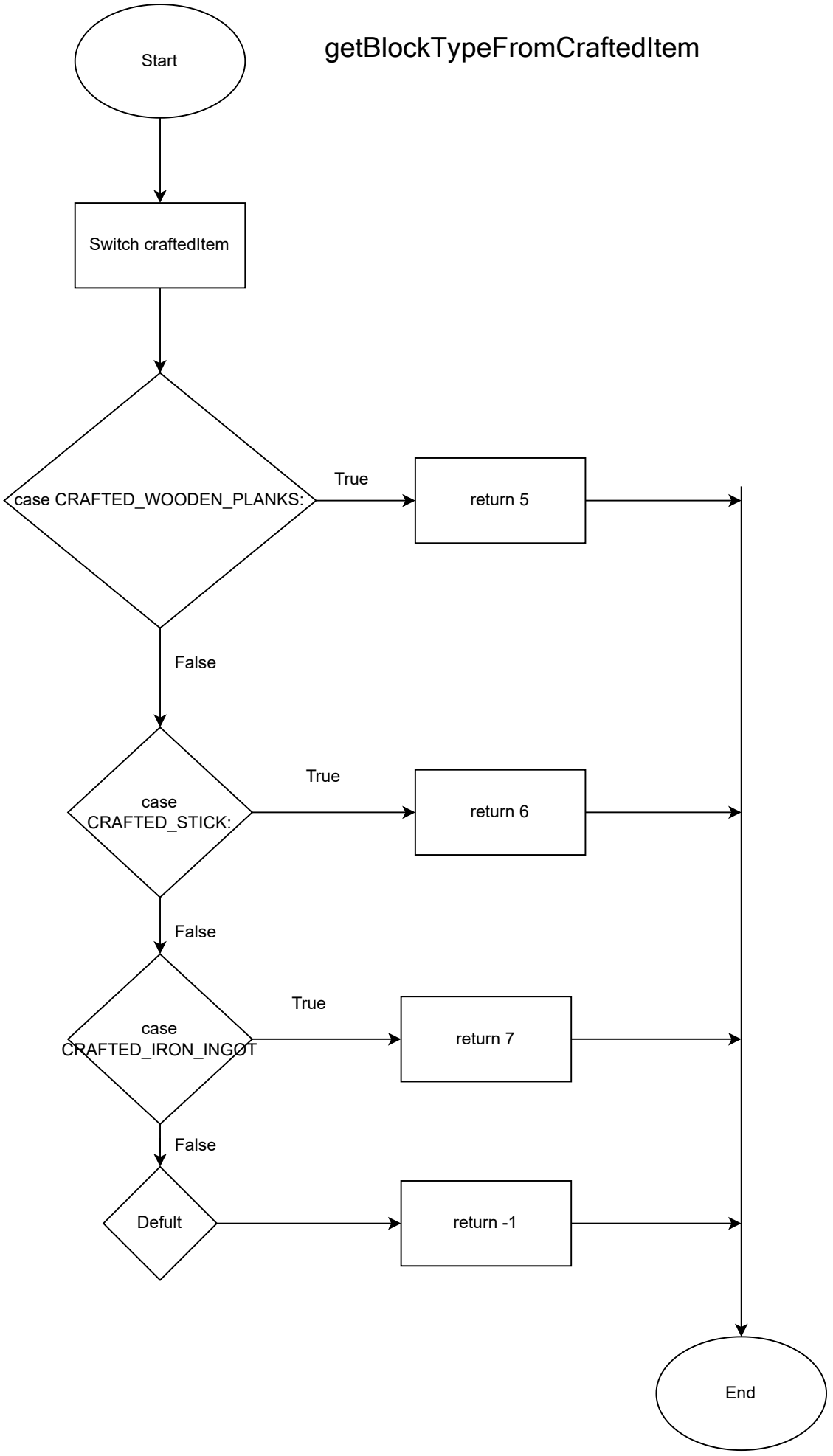

start
check if there is a block to place
yas
no
place the block
display error message
display a succeed message
End

InteractWithWorld()

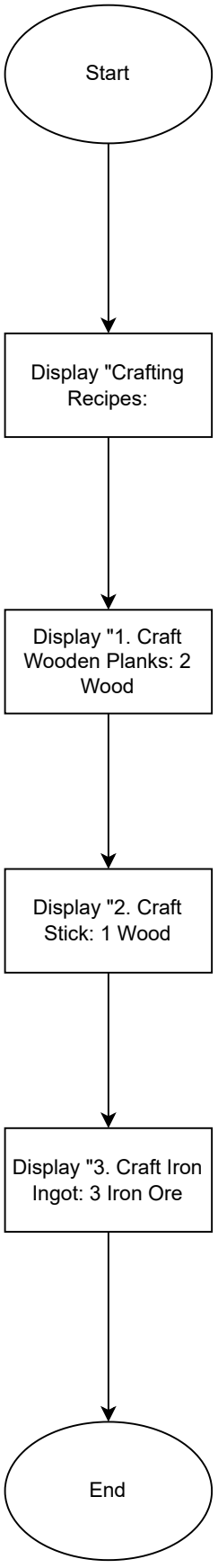


mineblock

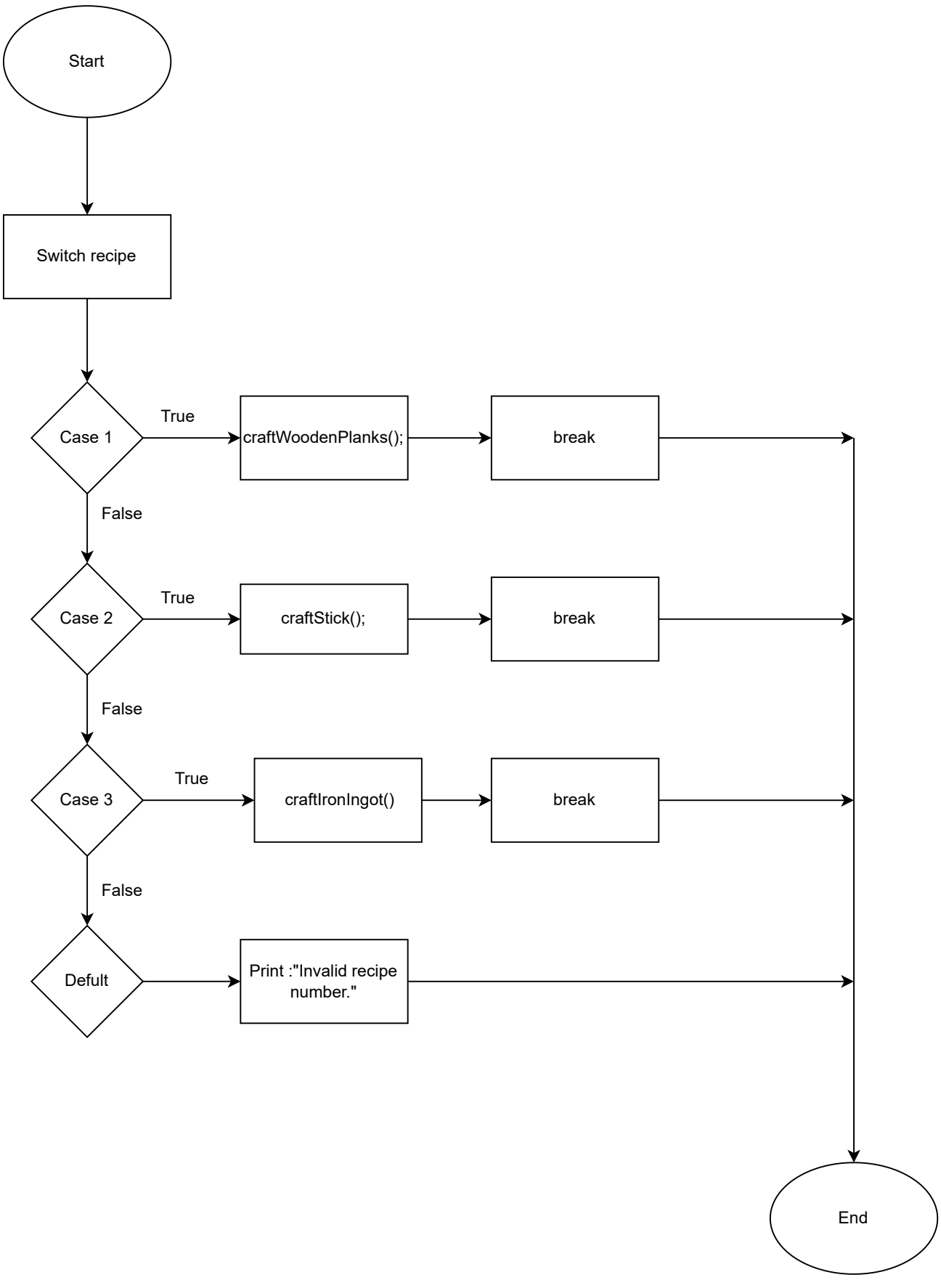

Start
blockType = world[playerX] [playerY]
Does blockType not equal AIR?
Yes
NO
Mine block
Print "No block to mine here."
Add blockType to inventory
Set world[playerX] [playerY] to AIR
Output "Mined " + getBlockName(blockType) + "."
waitForEnter
End


getBlockTypeFromCraftedItem
Start
Switch craftedItem
case CRAFTED_WOODEN_PLANKS:
True
return 5
False
case CRAFTED_STICK:
True
return 6
False
case CRAFTED_IRON_INGOT
True
return 7
False
Defult
return -1
End

displayCraftingRecipes()



craftItem()


Start
Switch recipe
Case 1
True
craftWoodenPlanks();
break
False
Case 2
True
craftStick();
break
False
Case 3
True
craftIronIngot()
break
False
Defult
Print :"Invalid recipe number."
End

Task 2.1.1 Pseucodes:

1. generateWorld()

Pseudocode:

```
FOR y = 0 to worldHeight
  FOR x = 0 to worldWidth
    randValue = random(0, 100)
    IF randValue < 20
      world[x][y] = WOOD
    ELSE IF randValue < 35
      world[x][y] = LEAVES
    ELSE IF randValue < 50
      world[x][y] = STONE
    ELSE IF randValue < 70
      world[x][y] = IRON_ORE
    ELSE
      world[x][y] = AIR
END FOR
END FOR
```

Description: Generates the game world by randomly assigning block types based on probability.

2. displayWorld()

Pseudocode:

```
PRINT "World Map:"
PRINT top border
FOR y = 0 to worldHeight
  PRINT left border
  FOR x = 0 to worldWidth
    IF player at x,y AND not in secret area
      PRINT "P" with player color
    ELSE IF player at x,y AND in secret area
      PRINT "P" with secret color
    ELSE
      PRINT block symbol and color for world[x][y]
  PRINT right border
PRINT bottom border
```

Description: Displays the game world map with borders, coloring, player marker.

3. movePlayer(direction)

Pseudocode:
```
IF direction is UP AND playerY > 0
  playerY--
ELSE IF direction is DOWN AND playerY < worldHeight
  playerY++
ELSE IF direction is LEFT AND playerX > 0
  playerX--
ELSE IF direction is RIGHT AND playerX < worldWidth
  playerX++
```

Description: Moves the player in the specified direction if possible.

4. mineBlock()

Pseudocode:
```
block = world[playerX][playerY]
IF block is not empty
  add block to inventory
  set world[playerX][playerY] to empty
ELSE
  PRINT "No block to mine"
```

Description: Mines the block at the player position if not empty.
5. craftItem(recipe)

Pseudocode:
```
IF recipe is 1
  call craftWoodenPlanks()
ELSE IF recipe is 2
  call craftStick()
ELSE IF recipe is 3
  call craftIronIngot()
ELSE
  PRINT "Invalid recipe"
```

Description: Performs crafting of different items based on recipe number.

6. craftWoodenPlanks()

Pseudocode:
```
IF inventory has 2 wood
  remove 2 wood from inventory
  add crafted wooden planks to crafted items
ELSE
  PRINT "Insufficient resources"
```

Description: Crafts wooden planks if wood available.

7. getBlockName(block)

Pseudocode:
```
SWITCH block
  CASE WOOD: return "Wood"
  CASE LEAVES: return "Leaves"
  CASE STONE: return "Stone"
  CASE IRON_ORE: return "Iron Ore"
  CASE EMPTY: return "Empty Block"
  DEFAULT: return "Unknown"
```

Description: Returns name string for a block type.

8. displayInventory()

Pseudocode:
```
PRINT "Inventory:"
IF inventory empty
  PRINT "Empty"
ELSE
  FOR each block type
    PRINT block name and count
PRINT "Crafted Items:"
IF crafted items empty
 PRINT "None"
ELSE
 FOR each crafted item
   PRINT crafted item name
```

Description: Prints inventory contents and counts.

9. saveGame(filename)

Pseudocode:
open file filename for writing
serialize world array and write to file
serialize playerX, playerY and write to file
serialize inventory and write to file
serialize craftedItems and write to file
serialize unlockMode and write to file
close file

Description: Saves entire game state to file using serialization.

10. loadGame(filename)

Pseudocode:
open file filename for reading
deserialize world array and read from file
deserialize playerX, playerY and read from file
deserialize inventory and read from file
deserialize craftedItems and read from file
deserialize unlockMode and read from file
close file

Description: Loads entire game state from file using deserialization.

11. getCountryAndQuoteFromServer()

Pseudocode:
send API request to server
get JSON response
parse response to get country and quote
print country and quote

Description: Gets data from external API.

12. unlockSecretDoor()

Pseudocode:
```
IF in unlock mode AND craft/mine/move done
  unlock secret door
  generate secret world

  fill inventory
  reset unlock flags
ELSE
  print "Invalid passkey"
  reset unlock mode and flags
```

Description: Unlocks secret door if correct sequence of actions performed.

13. generateSecretWorld()

Pseudocode:
```
SET world dimensions to new values
FOR each stripe in world height
  Fill stripe with red/white/blue blocks
```

Description: Generates a world with red, white and blue stripes for the secret area.

14. fillInventory()

Pseudocode:
```
CLEAR inventory
FOR each block type
  FOR number of slots in inventory
    Add block type to inventory
```

Description: Fills inventory with blocks when entering secret area.

15. clearScreen()

Pseudocode:
```
IF Windows OS
  Execute 'cls' command
ELSE
  Print control characters to clear terminal
Print blank line
```

Description: Clears the terminal/consolescreen

Task 2.2:

1. main() - Main game loop, starts game (line 18)
2. initGame() - Initializes new game world (line 63)
3. generateWorld() - Randomly generates world grid (line 70)
4. displayWorld() - Prints world grid to console (line 84)
5. getBlockSymbol() - Gets display symbol for a block (line 104)
6. getBlockChar() - Gets character for a block type (line 114)
7. startGame() - Handles main game input loop (line 126)

8. fillInventory() - Fills inventory when entering secret area (line 223)

9. resetWorld() - Resets world grid and player position (line 233)

10. generateEmptyWorld() - Generates new empty world grid (line 242)

11. clearScreen() - Clears console screen (line 266)

12. lookAround() - Prints nearby blocks around player (line 273)

13. movePlayer() - Handles player movement on input (line 286)

14. mineBlock() - Handles mining block at player position (line 298)
15. placeBlock() - Handles placing block from inventory (line 307)
16. getBlockTypeFromCraftedItem() - Gets block type from crafted item (line 329)
17. getCraftedItemFromBlockType() - Gets crafted item from block type (line 338)
18. displayCraftingRecipes() - Prints available recipes (line 346)
19. craftItem() - Handles crafting items (line 353)
20. craftWoodenPlanks() - Crafts wooden planks (line 361)

21. craftStick() - Crafts stick (line 370)
22. craftIronIngot() - Crafts iron ingot (line 379)
23. inventoryContains() - Checks if inventory contains item (line 388)
24. removeItemsFromInventory() - Removes items from inventory (line 413)
25. addCraftedItem() - Adds crafted item to list (line 422)
26. interactWithWorld() - Handles world interactions (line 425)
27. saveGame() - Saves game state to file (line 443)
28. loadGame() - Loads game state from file (line 459)

29. getBlockName() - Gets name string for block (line 472)

30. displayLegend() - Prints game legend/key (line 478)

31. displayInventory() - Prints player inventory (line 486)

32. getBlockColor() - Gets ANSI color code for block (line 514)

33. waitForEnter() - Waits for user input (line 517)

34. getCraftedItemName() - Gets name for crafted item (line 524)

35. getCraftedItemColor() - Gets color code for crafted item (line 533)

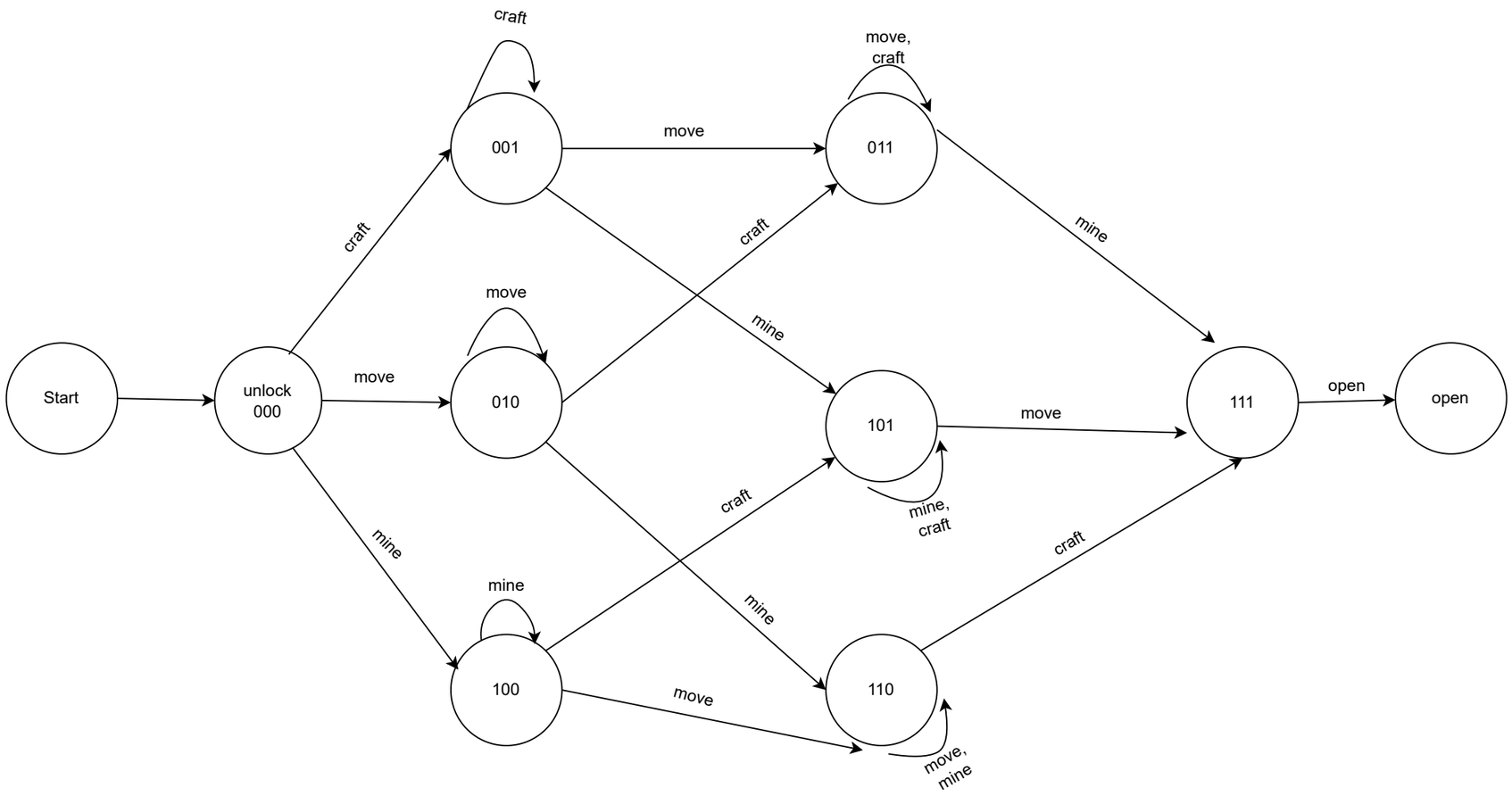
36. getCountryAndQuoteFromServer() - API call fordata(line536)

Task 2.3.1:

1:To access the door you need to follow a sequence of actions. This includes entering an " unlock mode" and performing tasks such, as crafting, mining and moving. Once you successfully unlock the door you will be able to enter an area with a new game board. You will notice the flag of The Netherlands displayed on the map.

2:

DFA:


craft
move,
craft
move
001
011
craft
craft
mine
move
mine
Start
unlock
000
move
010
101
move
111
open
open
craft
mine,
craft
mine
mine
craft
100
move
110
move,
mine

FSA EXPLANATION:

How to Open the Secret Door?

Unlock is the first command, followed by three other commands in any order: Craft, Move, or Mine. After these commands, the player should enter the Open command. After making this steps secret door will be opened and Dutch flag will be appear in game board.

As you can see from the diagram, the diagram starts with the Unlock command. After the Unlock command, the player has three options: Craft, Mine, and Move.

In the diagram, we represent these commands as follows:

Craft: 001
Move: 010
Mine: 100

After choosing one of these combinations, the player can choose the two remaining commands. If the player chooses the same command, they will return to the same node until they enter a different command. After these processes, the player should select the second command among "craft" , "mine" or "move".

As you can see from the FSA, every node chosen in the second pick has two probabilities. Let's take the Unlock and Move combination as an example. After using this combination, the player can choose Craft or Mine to go further. As I mentioned before, if the player chooses the same command, which is Move in this situation, they will loop back to the same node. Furthermore, after choosing one of these commands, every node has one option to the last node, which is Open. The rest of the commands loop back to the same node. Moreover, after selecting one of these commands, each node has a single option if the last node is "110" the last comment which go further is "craft", if the remaining node is "011" last command is "mine" lastly if the renaming node is "101" the last command is "move. After this all process player should type "open" and the DFA finishes and requires all conditions which written in the first paragraph

Who did what?:

Student Name	Provisional Report
Tolga Efe Savash	%50 of Flowchart, %33 of pseudocode, %33 of FSA design, %38 of Git
Mert Doğan	%20 of Flowchart, %33 of pseudocode, %33 of FSA design, %31 of Git
Metehan Bilir	%30 of Flowchart, %33 of pseudocode, %33 of FSA design, %31 of Git
Onur Özgen	%0 of Flowchart, %1 of Pseudocode, %1 of FSA design,