JavaCraft Provisional Report - Group 75

Table of Contents

Group Details

Group

Attribute	Details
Group Name	The Jokers
Group Number	75
TA	Thomas

Group Members

Student Name	Student ID
Mila Spasova	i6346060
Ethan Goetsch	i6293227
Amzu Andrei-Alexandru	i6361233
Aleksandra Yemelyanova	i6348514

Introduction

'JavaCraft' is a terminal-based game developed in Java, drawing inspiration from the iconic game 'Minecraft'. This report delves into 'JavaCraft,' covering its core functionalities, the intricacies of the Finite State Automata embedded in its source code, an analysis of the mechanics involved, and detailing how our team utilized Git for collaborative development and source code modifications.

The team's tasks are summarized as follows: Mila completed the function documentation, which was subsequently proofread by Ethan. The workflow pseudocode was a collaborative effort among Mila, Ethan, and Sasha. Both the function flowcharts and pseudocode were divided among Mila, Ethan, Sasha, and Alex. The Finite State Automata and Secret Door Logic Analysis were undertaken by Mila, Sasha, and Alex. The entire team reviewed and approved this report.

JavaCraft's Workflow

Flowchart

See *Appendix Figure 1A*

Pseudocode

See *Appendix Figure 1B*

Functionality Exploration

No.	Function Name	Description
2.	generateWorld	Randomly assigns a block type to each world block
1.	initGame	Takes in two integers for the world's width and height as input and defines the initial world and player values using the inputted values
3.	displayWorld	Iterates over the world matrix and prints each block's symbol and player's position
4.	getBlockSymbol	Takes in an integer for the block type as input and returns a string representing the colour and character of the corresponding block
5.	getBlockChar	Takes in an integer for the block type as input and returns the corresponding character
6.	fillInventory	Clears the player's inventory and fills it with four of each block type in the game
7.	resetWorld	Generates an empty world and sets the player's position to the centre of the world
8.	generateEmptyWorld	Initializes a new world matrix and divides it into three horizontal partitions of different colours
9.	clearScreen	Clears the screen using a CLS command on a Windows operating system, otherwise uses an escape sequence for other operating systems. If any errors occur during this process print the stack trace of the exception
10.	lookAround	Prints the symbols of the blocks adjacent to the player's positions
11.	movePlayer	Takes in a string for the direction to move in as input. Moves the player in a cardinal direction based on the given direction
12.	mineBlock	Mines the block at the player's position and adds it to the player's inventory if the block is not Air. Otherwise, informs the player they cannot mine the block
13.	placeBlock	Takes in an integer for the type of block to place as input. If the block type is not a crafted item then removes it from the player's inventory and places it at the player's position. Otherwise, removes the block type from the player's crafted items and places it at the player's position

No.	Function Name	Description
14.	getBlockTypeFromCraftedItem	Takes in an integer for the crafted item as input. Returns an integer corresponding to the crafted item inputted.
15.	getCraftedItemFromBlockType	Takes in an integer for the block type as input. Returns an integer corresponding to the block type inputted
16.	displayCraftingRecipes	Prints the recipe number, recipe name and crafting ingredients to the terminal for each recipe
17.	craftItem	Takes in an integer as input for the recipe. If it is a valid recipe then crafts the item corresponding to the inputted recipe. Otherwise, inform the player that it is not a valid recipe number.
18.	craftWoodenPlanks	If the player's inventory has the necessary ingredients, then craft wooden planks, add them to the player's inventory and remove the used ingredients
19.	craftStick	If the player's inventory has the necessary ingredients, then craft a stick, add them to the player's inventory and remove the used ingredients
20.	craftIronIngot	If the player's inventory has the necessary ingredients, then craft an iron ingot, add them to the player's inventory and remove the used ingredients
21.	craftEnchantmentTable	If the player's inventory has the necessary ingredients, then craft an enchantment table, add them to the player's inventory and remove the used ingredients
22.	inventoryContains	Takes in an integer as input for the item and returns true if the player's inventory contains the item. Otherwise, returns false.
23.	inventoryContains	Takes in an integer for the item and an integer for the amount of items as input. Returns true if the player's inventory contains the specified item the indicated number of times.
24.	removeItemsFromInventory	Takes in an integer for the item and an integer for the amount of items to remove as input. Remove the specified item from the player's inventory the indicated number of times.
25.	addCraftedItem	Takes in an integer for the crafted item as input and adds the crafted item to the player's inventory.
26.	interactWithWorld	Checks the block type at the player's position. If the block can be gathered then add the block type to the player's inventory. Otherwise, inform the player that the block cannot be gathered or is unrecognized.
27.	saveGame	Takes in a string for the file name as input. Writes the game world's data and player's data to the specified text file.
28.	loadGame	Takes in a string for the file name as input. Reading the data from the text file and initializes the game world and player data with the value read.

No.	Function Name	Description
29.	getBlockName	Takes in an integer for the block type as input and returns a string representing the name of the corresponding block type.
30.	displayLegend	Prints each blocks symbol and name to the terminal
31.	displayInventory	If the player's inventory is not empty then prints each item's name and amount to the terminal. Otherwise, informs the player their inventory is empty.
32.	getBlockColor	Takes in an integer for the block type as input and returns a string representing the colour code of the corresponding block type.
33.	waitForEnter	Waits for input from the enter key.
34.	getCraftedItemName	Takes in an integer for the crafted item as input and returns a string representing the name of the corresponding crafted item.
35.	getCraftedItemColor	Takes in an integer for the crafted item as input and returns a string representing the colour code of the corresponding crafted item.

For Flowcharts and Pseudocode, see Appendix Figures 2 - 16

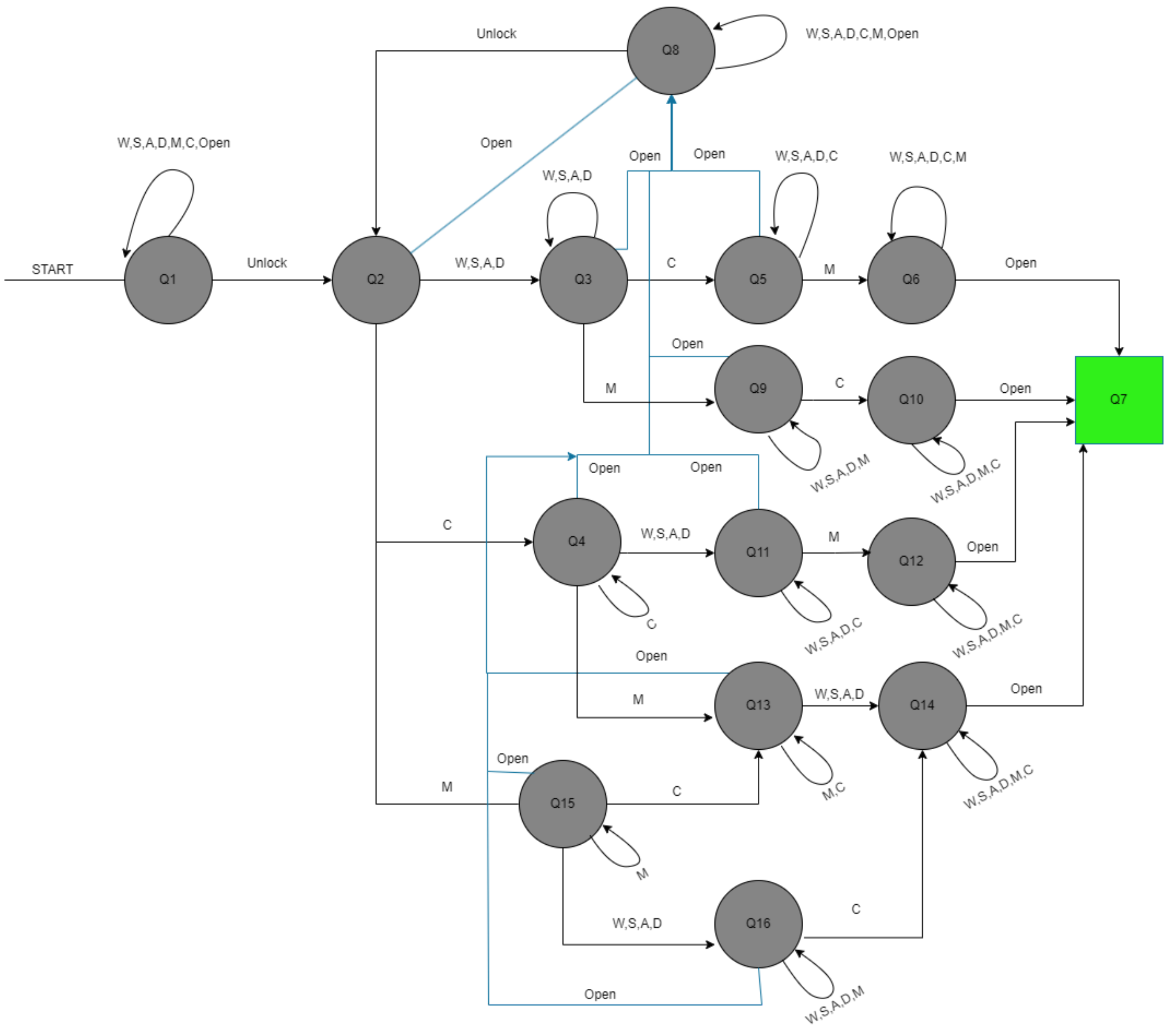
Finite State Automata (FSA) Design

Secret Door Logic Analysis

To unlock the secret door, players must first enter 'Unlock Mode' by using the 'unlock' command. Within this mode, players must perform three actions — Move, Craft, Mine — in any order. Once completed, they can use the 'open' command to access the door. However, if any actions are omitted, the system reverts to its default state, requiring players to re-enter 'Unlock Mode' and start the process anew.

FSA Illustration & Description

Σ = {Unlock, W, A, S, D, C, M, Open}



Git Collaboration & Version Control

Repository

https://github.com/Ethan-Goetsch/Intro-To-Computer-Science-Project/tree/develop

Branch Details

Branch: **develop**
Members: Alex, Mila, Ethan, Sasha

Changes & Conflicts

Our Git workflow prioritized proactive communication and clear task delegation to individual members, ensuring that everyone had distinct responsibilities. As a result, we experienced few conflicts and merging challenges. On the rare occasions that conflicts did arise, they were swiftly addressed through open communication, with team members being informed about the affected files.

Extending the Game Code

Interacting with Flags API

Conclusion

Appendix

Figure 1: GameFlow

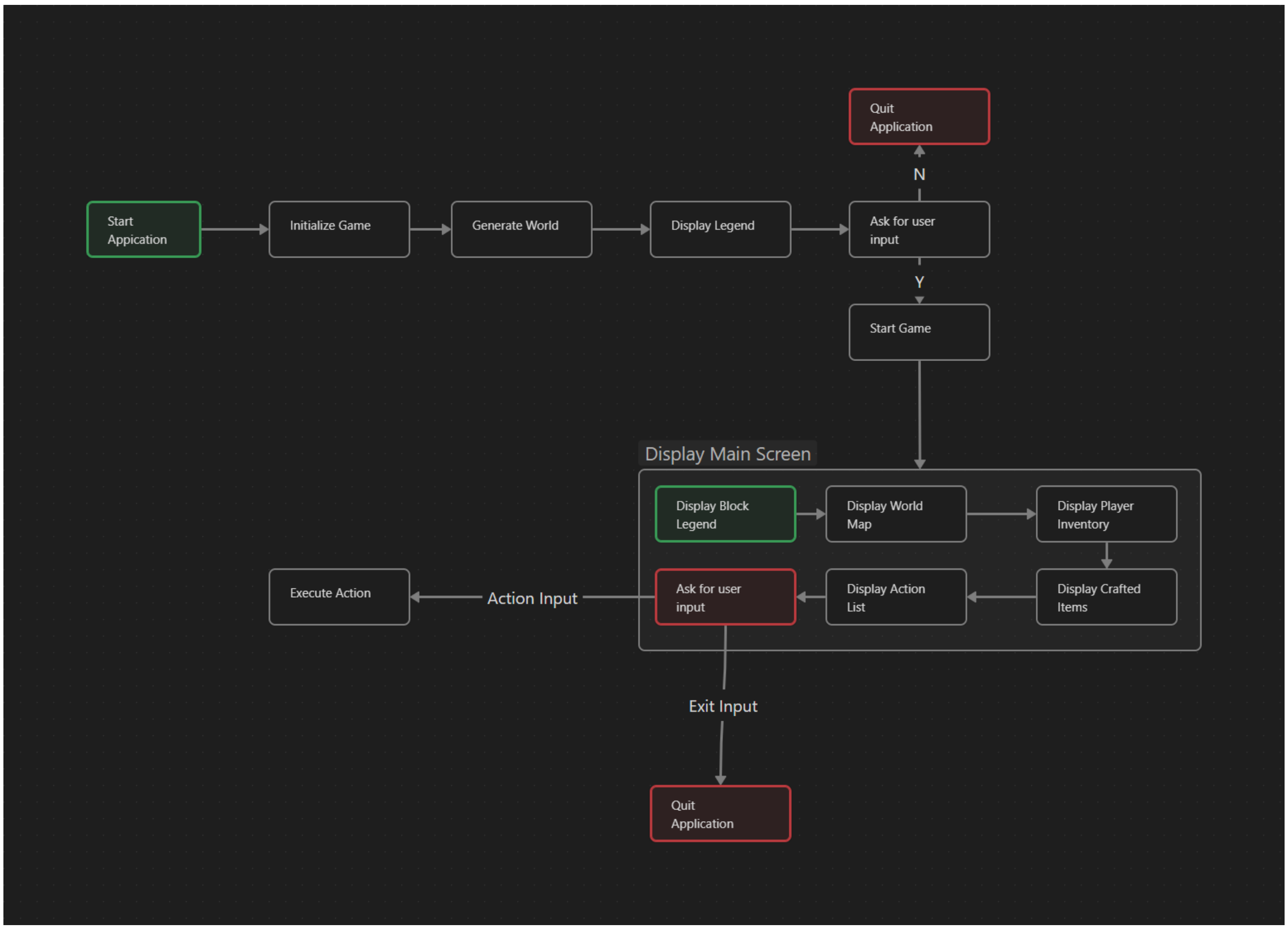


Figure 1A

```
Start

Initialize Game
Generate the World
Print Game Instructions

Start Game
    WHILE True DO
        Print Block Legend
        Print World Map
        Print Inventory
        Print Crafted Items
        Print Action List

        Player Enters Action
        SWITCH Player Action
            CASE “W”, “A”, “S”, “D”:
                Move Player Character in the Specified Direction
            CASE “M”:
                IF Block Exists THEN
                    Print “Mined {blockName}”
                ELSE
                    Print “No block to mine here”
            CASE “P”:
                Input blockType
                IF blockType is Valid THEN
                    IF blockType is in Inventory or Crafted Items THEN
                        Print “Placed {blockName}”
                    ELSE
                        Print “You don’t have {blockType} in your inventory”
                ELSE
                    Print “Invalid Block Type”
            CASE “C”:
                Display Craft Recipes
                Input Recipe Number
                IF Recipe Number is Valid THEN
                    IF Player has Recipe Blocks THEN
                        Craft Item
                    ELSE
                        Print “Not enough blocks”
                ELSE
                    Print “Invalid recipeNum”
            CASE “I”:
                Check Block Type at Player’s Coordinates
                SWITCH Block Type
                    CASE Wood:
                        Add Wood to Inventory
                    CASE Leaves:
                        Add Leaves to Inventory
                    CASE Stone:
                        Add Stone to Inventory
                    CASE Iron Ore:
                        Add Iron Ore to Inventory
                    CASE Air:
                        Do nothing
            CASE “Save”:
                Save Current World State
            CASE “Load”:
                Ask for File Name
                TRY
                    Load Saved File
                    Print “Game state loaded from {fileName}”
                CATCH Exception
                    Print “Error while loading the game state”
            CASE “Exit”:
                Print “Exiting the game. Goodbye!”
                Exit Game
        END SWITCH
    END WHILE

End
```

Figure 1B

Figure 2: InitGame

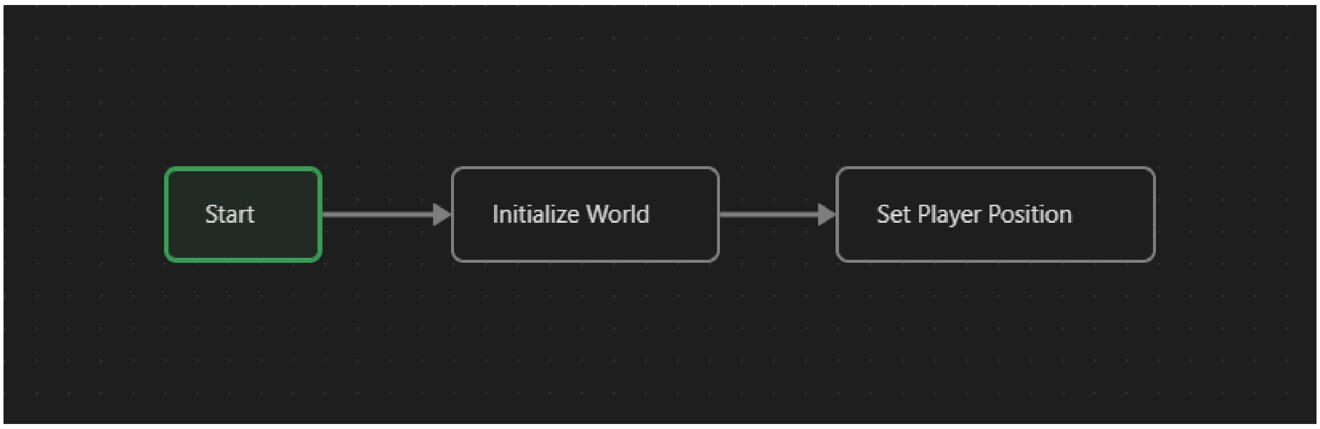


Figure 2A

```
Algorithm InitGame(int Width, int Height)
        World = [Width][Height]
        PlayerXCoordinate = Width / 2
        PlayerYCoordinate = Height / 2
END
```

Figure 2B

Figure 3: GenerateWorld

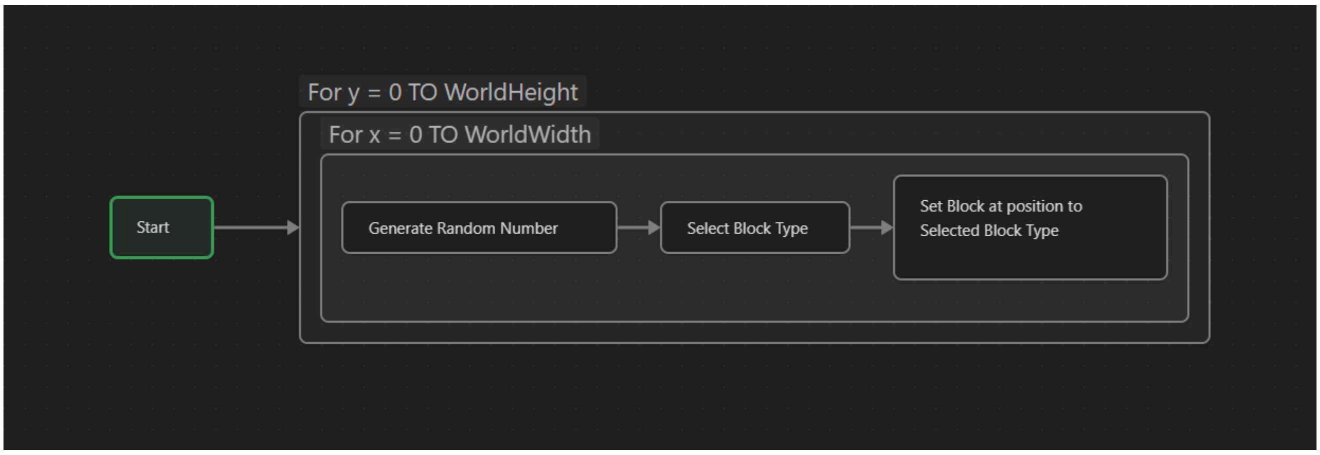


Figure 3A

```
Algorithm GenerateWorld(int WorldHeight,int WorldWidth)
        FOR Y = 0 TO WorldHeight-1
                FOR X = 0 TO WorldWidth-1
                        Random = Random number between 0 and 100
                        IF Random < 20 THEN
                                World[X][Y] = WOOD
                        ELSE IF Random < 35 THEN
                                World[X][Y] = LEAVES
                        ELSE IF Random < 50 THEN
                                World[X][Y] = STONE
                        ELSE IF Random < 70 THEN
                                World[X][Y] = IRON_ORE
                        ELSE
                                World[X][Y] = AIR
END
```

Figure 3B

Figure 4: GetBlockSymbol

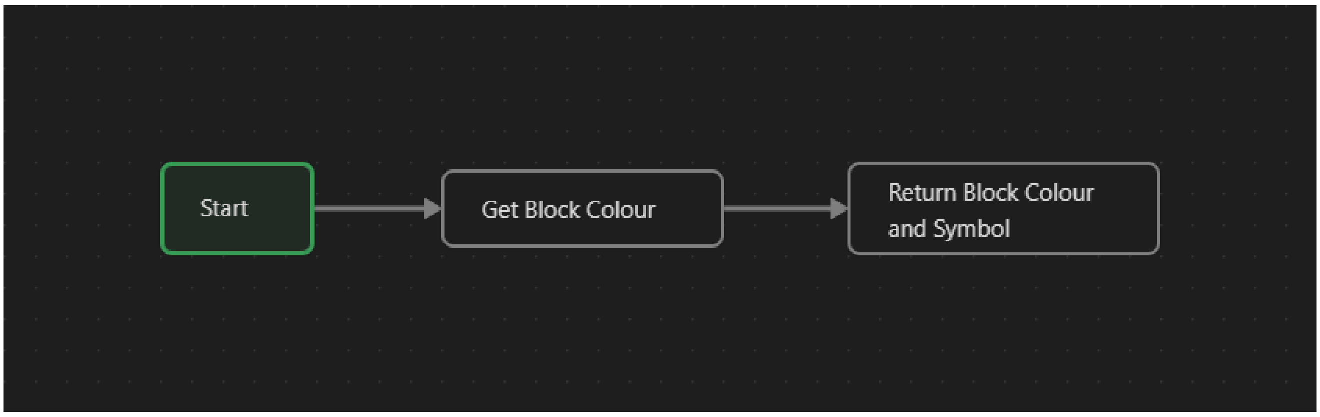


Figure 4A

```
Algorithm GetBlockSymbol(int BlockType)
        BlockColor = “”
        SWITCH BlockType
                CASE AIR:
                        return “-”
                CASE WOOD:
                        BlockColor = RED
                CASE LEAVES:
                        BlockColor = GREEN
                CASE STONE:
                        BlockColor = BLUE
                CASE IRON ORE:
                        BlockColor = WHITE

                RETURN "BlockColor Block Character “
END
```

Figure 4B

Figure 5: DisplayLegend

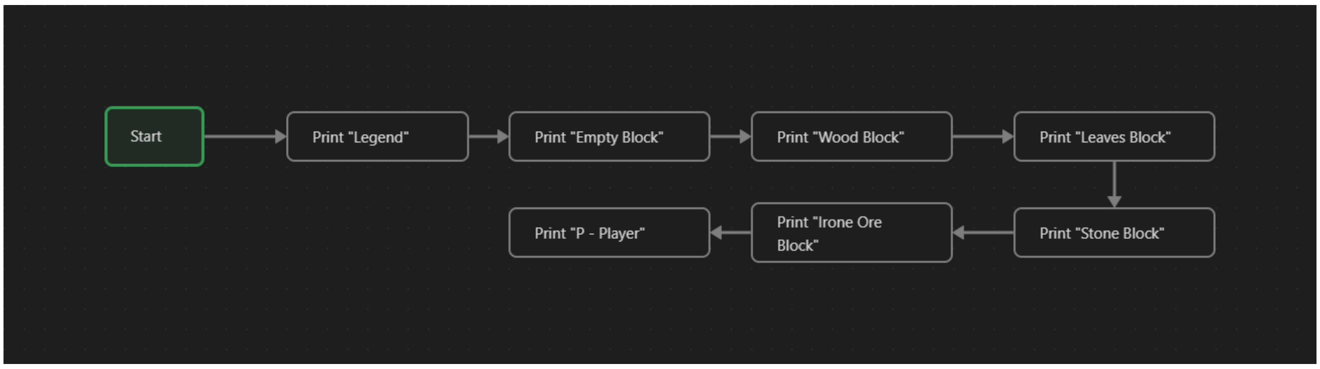


Figure 5A

```
Algorithm DisplayLegend()
        Print ”Legend:”
        Print ”-- - Empty block”
        Print ”WOOD BLOCK”
        Print ”LEAVES BLOCK”
        Print ”STONE BLOCK”
        Print ”IRON ORE”
        Print ”P - Player”
END
```

Figure 5B

Figure 6: DisplayWorld

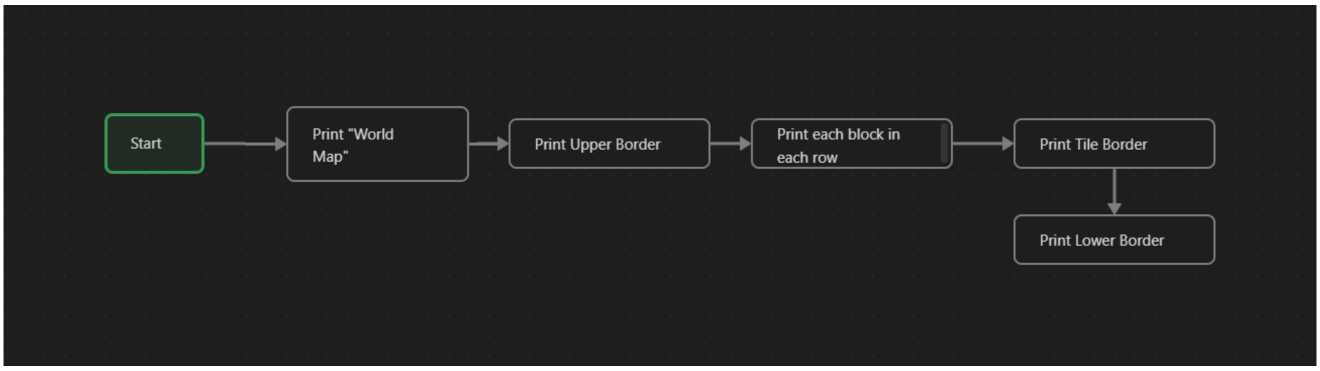


Figure 6A

```
Algorithm DisplayWorld(boolean InSecretArea)
        Print ”World Map”
        Print ”GAME BORDER”

        FOR Y = 0 TO WorldHeight
                Print(”║”)
                FOR x TO  WorldHeight
                        IF x = playerX AND y = playerY AND NOT InSecretArea THEN
                                Print ”P”
                        ELSE IF x = playerX AND y = playerY AND InSecretArea THEN
                                Print ”P”
                        ELSE
                                GetBlockSymbol of World[X][Y]
                                Print ”║”
                                Print "╚═" + "=" repeat(worldWidth * 2 - 2) + "╝"
END
```

Figure 6B

Figure 7: GenerateEmptyWorld

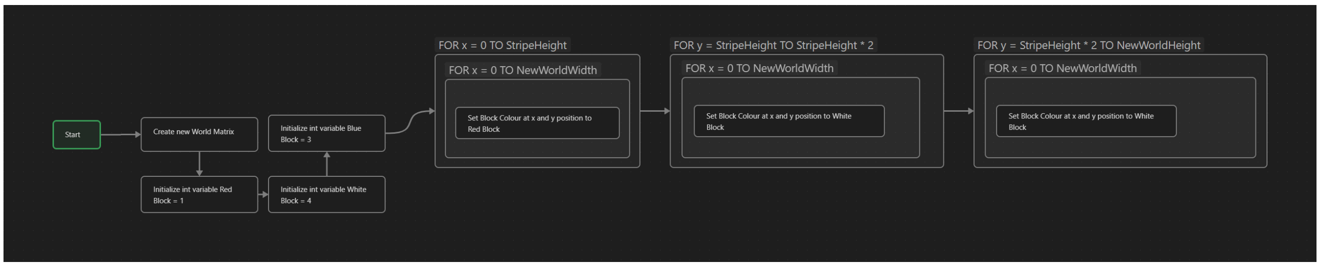
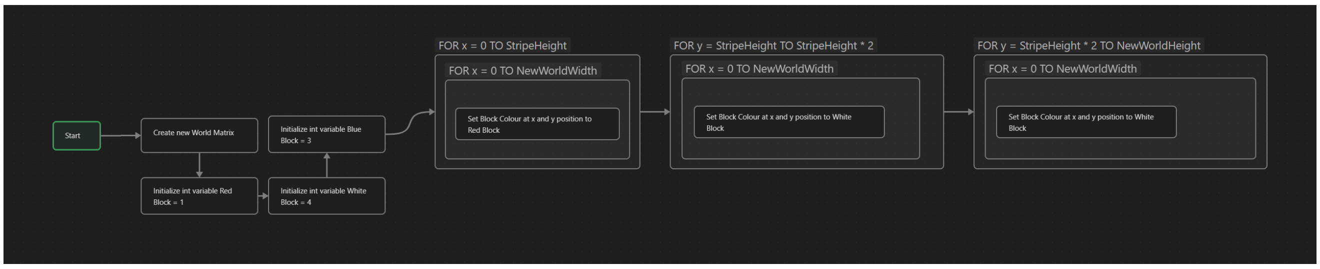

Start
Create new World Matrix
Initialize int variable Red Block = 1
Initialize int variable White Block = 4
Initialize int variable Blue Block = 3
FOR x = 0 TO StripeHeight
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to Red Block
FOR y = StripeHeight TO StripeHeight * 2
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to White Block
FOR y = StripeHeight * 2 TO NewWorldHeight
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to White Block

Figure 7A

```
Algorithm GenerateEmptyWorld(int WorldWidth, int WorldHeight)
        World = [WorldWith][WorldHeight]
        RedBlock = 1
        WhiteBlock = 4
        BlueBlock = 3

        StripeHeight = WorldHeight/3

        FOR y = 0 TO StripeHeight
                FOR x = 0 TO WorldWidth
                        World[x][y] = RedBlock
                End FOR
        End FOR

        FOR y = StripeHeight TO StripeHeight*2
                FOR x = 0 TO WorldWidth
                        World[x][y] = RedBlock
                End FOR
        End FOR
        FOR y = StripeHeight*2 TO WorldHeight
                FOR x = 0 TO WorldWidth
                        World[x][y] = RedBlock
                End FOR
        End FOR
END
```

Figure 7B

Figure 8: ClearScreen



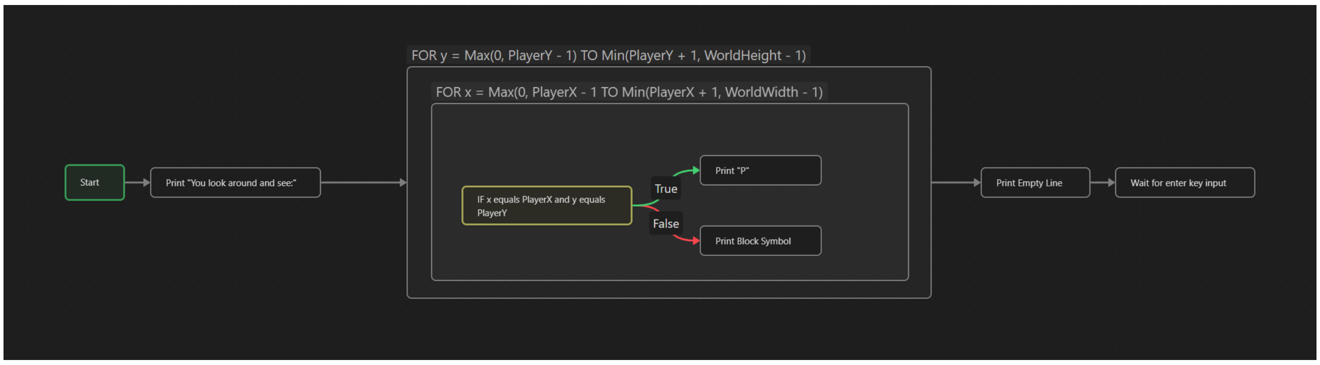
Start
Create new World Matrix
Initialize int variable Red Block = 1
Initialize int variable White Block = 4
Initialize int variable Blue Block = 3
FOR x = 0 TO StripeHeight
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to Red Block
FOR y = StripeHeight TO StripeHeight * 2
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to White Block
FOR y = StripeHeight * 2 TO NewWorldHeight
FOR x = 0 TO NewWorldWidth
Set Block Colour at x and y position to White Block

Figure 8A

```
Algorithm ClearScreen(boolean DebugState)
        IF NOT DebugState THEN
                TRY
                        IF Operating System is "Windows" THEN
                                Execute Command "cmd /c cls"
                        ELSE
                                Print Symbol
                                Flush System Output
                        CATCH IOException or InteruptedException
                                Print Stack Trace
                END TRY
        END IF
END
```

Figure 8B

Figure 9: LookAround



Start
Print "You look around and see:"
FOR y = Max(0, PlayerY - 1) TO Min(PlayerY + 1, WorldHeight - 1)
FOR x = Max(0, PlayerX - 1 TO Min(PlayerX + 1, WorldWidth - 1)
IF x equals PlayerX and y equals PlayerY
True
False
Print "P"
Print Block Symbol
Print Empty Line
Wait for enter key input

Figure 9A

```
Algorithm LookAround(int PlayerX, int PlayerY, int WorldWidth, int WorldHeight)
        Print "You look around and see:"
        FOR y = Max(0, PlayerY - 1) TO Min(PlayerY + 1, WorldHeight - 1)
                FOR x = Max(0, PlayerX - 1 TO Min(PlayerX + 1, WorldWidth - 1)
                        IF x == PlayerX AND y == PlayerY THEN
                                Print "P"
                        ELSE
                                Print Block Symbol
                        Print Empty Line
                END FOR
        END FOR
        Print Empty Line
        Wait For Enter Key Input
END
```

Figure 9B

Figure 10: MovePlayer

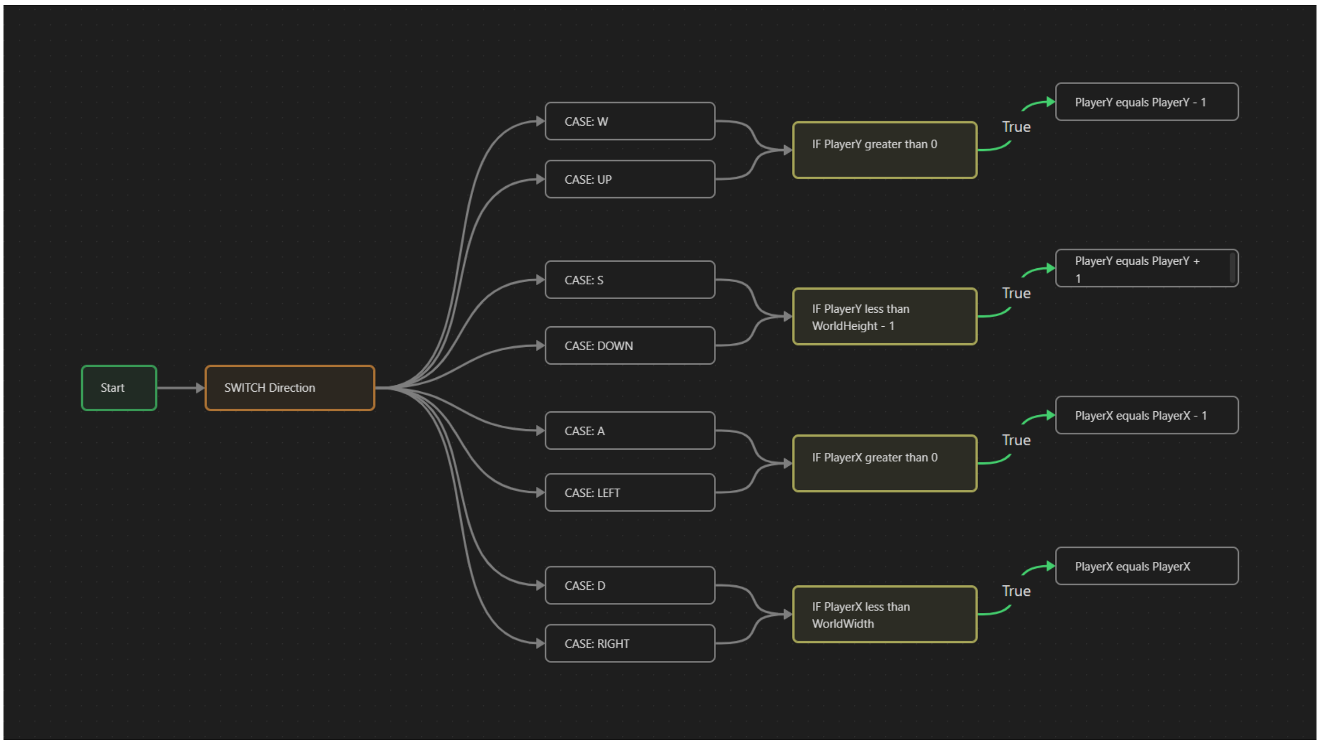

Start
SWITCH Direction
CASE: W
CASE: UP
CASE: S
CASE: DOWN
CASE: A
CASE: LEFT
CASE: D
CASE: RIGHT
IF PlayerY greater than 0
IF PlayerY less than WorldHeight - 1
IF PlayerX greater than 0
IF PlayerX less than WorldWidth
True
True
True
True
PlayerY equals PlayerY - 1
PlayerY equals PlayerY + 1
PlayerX equals PlayerX - 1
PlayerX equals PlayerX

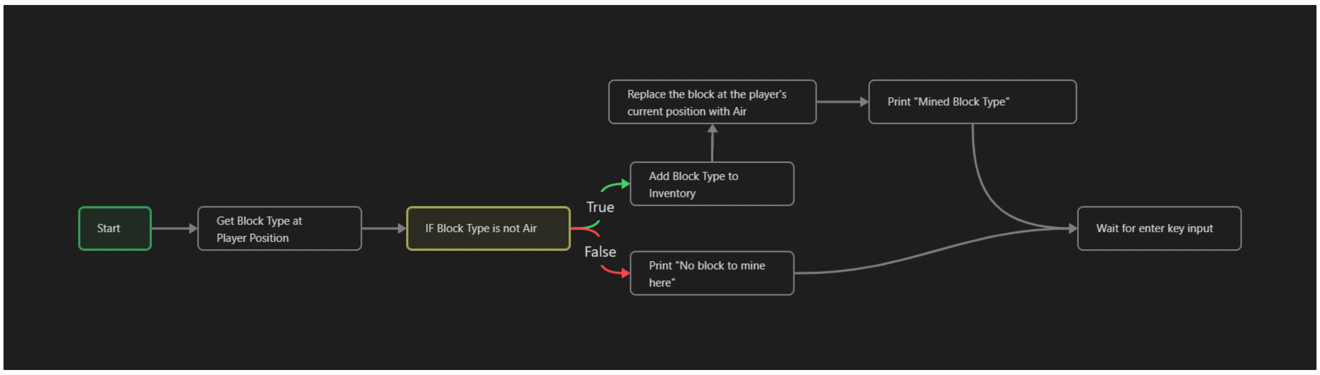
Figure 10A

```
Algorithm MovePlayer(string Direction, int PlayerX, int PlayerY, int WorldWidth, int
WorldHeight)
        SWITCH Direction
                CASE: W
                CASE: UP
                        IF PlayerY > 0 THEN
                                PlayerY = PlayerY -1
                        BREAK
                CASE: S
                CASE: DOWN
                        IF PlayerY < WorldHeight - 1 THEN
```

```
                                PlayerY = PlayerY + 1
                        BREAK
                CASE: A
                CASE: LEFT
                        IF PlayerX > 0
                                PlayerX = PlayerX - 1
                        BREAK
                CASE: D
                CASE: RIGHT
                        IF PlayerX < WorldWidth
                                PlayerX = PlayerX + 1
                        BREAK
        END SWITCH
END
```

Figure 10B

Figure 11: MineBlock



Start
Get Block Type at Player Position
IF Block Type is not Air
True
False
Add Block Type to Inventory
Replace the block at the player's current position with Air
Print "Mined Block Type"
Print "No block to mine here"
Wait for enter key input

Figure 11A

```
Algorithm MineBlock(int[][] World, int PlayerX, int PlayerY)
        BlockType = World[PlayerX][PlayerY]
        IF BlockType IS NOT Air THEN
                Add BlockType to Inventory
                World[PlayerX][PlayerY] = Air
                Print "Mined BlockType"
        ELSE
                Print "No block to mine here"
        Wait For Enter Key Input
END
```

Figure 11B

Figure 12: GetBlockTypeFromCraftedItem

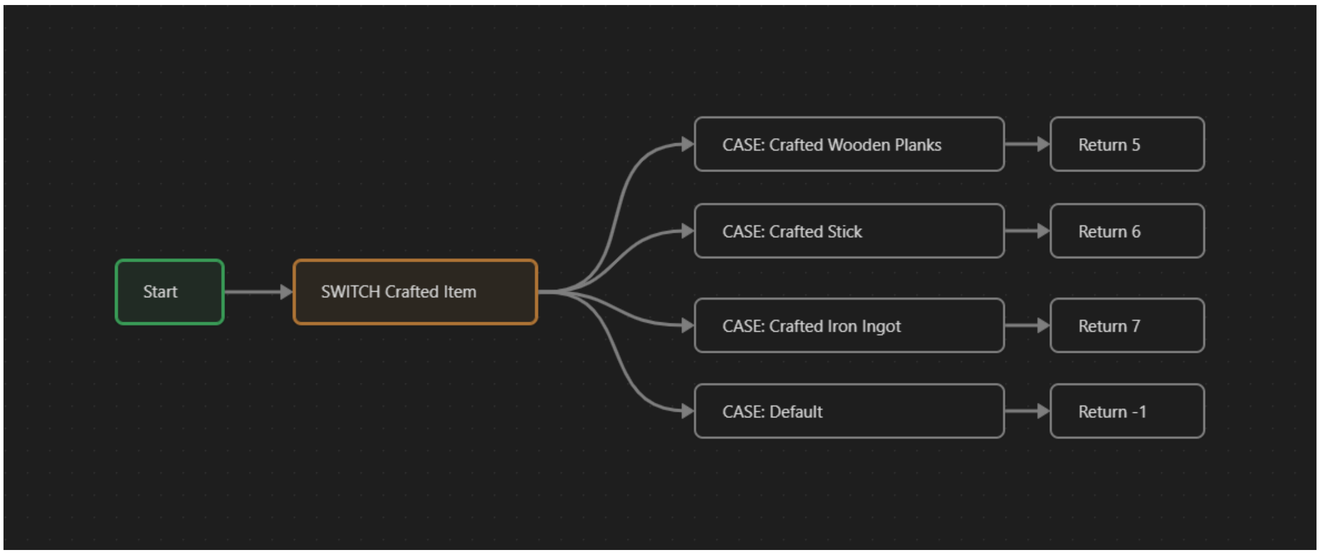


Figure 12A

```
Algorithm GetBlockTypeFromCraftedItem(int CraftedItem)
        SWITCH Crafted Item
                CASE Crafted Wooden Planks:
                        Return 5
                CASE Crafted Stick:
                        Return 6
                CASE Crafted Iron Ingot:
                        Return 7
                DEFAULT:
                        Return  -1
END
```

Figure 12B

Figure 13: GetCraftedItemFromBlockType

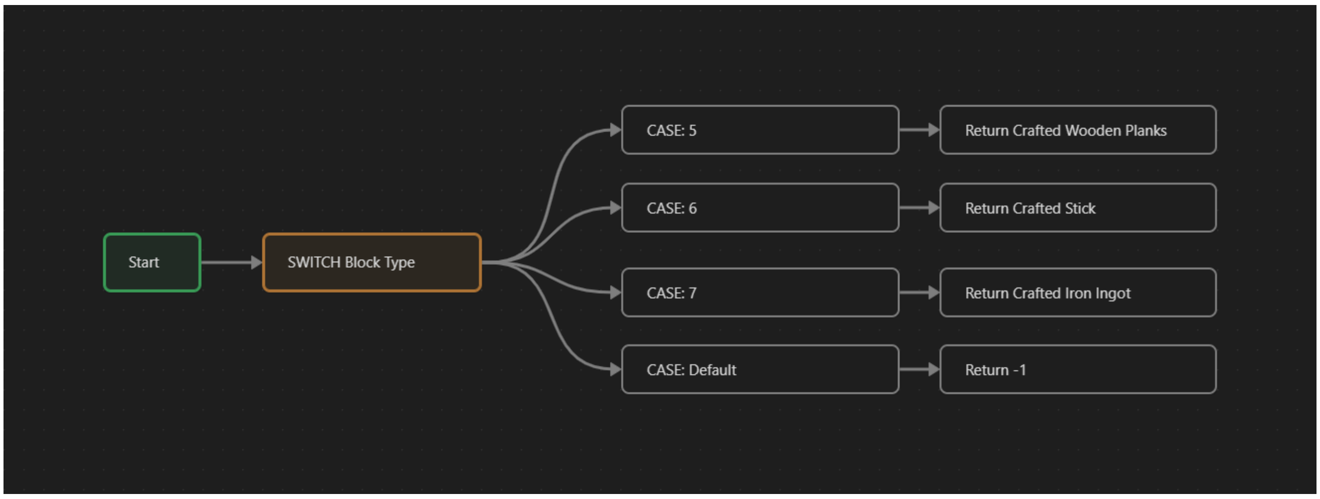


Figure 13A

```
Algorithm GetCraftedItemFromBlockType(int BlockType)
        SWITCH BlockType
                CASE 5:
                        Return Crafted Wooden Planks
                CASE 6:
                        Return Crafted Stick
                CASE 7:
                        Return Crafted Iron Ingot
                DEFAULT:
                Return -1
END
```

Figure 13B

Figure 14: DisplayCraftedRecipes

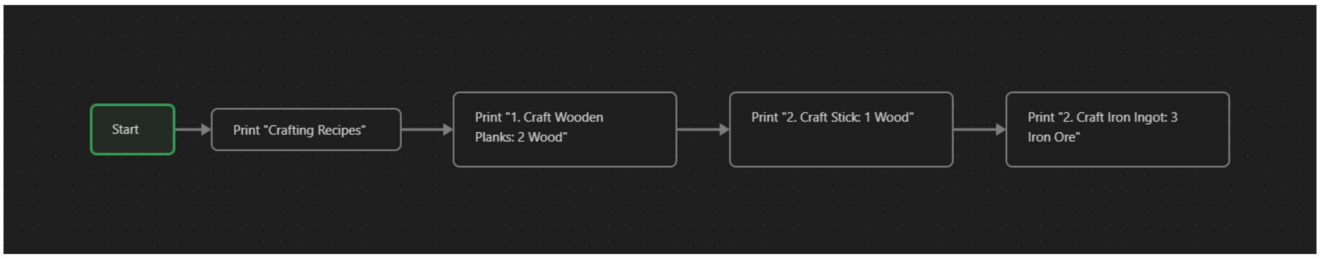

Start
Print "Crafting Recipes"
Print "1. Craft Wooden Planks: 2 Wood"
Print "2. Craft Stick: 1 Wood"
Print "2. Craft Iron Ingot: 3 Iron Ore"

Figure 14A

```
Algorithm DisplayCraftRecipes()
        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
```

Figure 14B

Figure 15: CraftItem

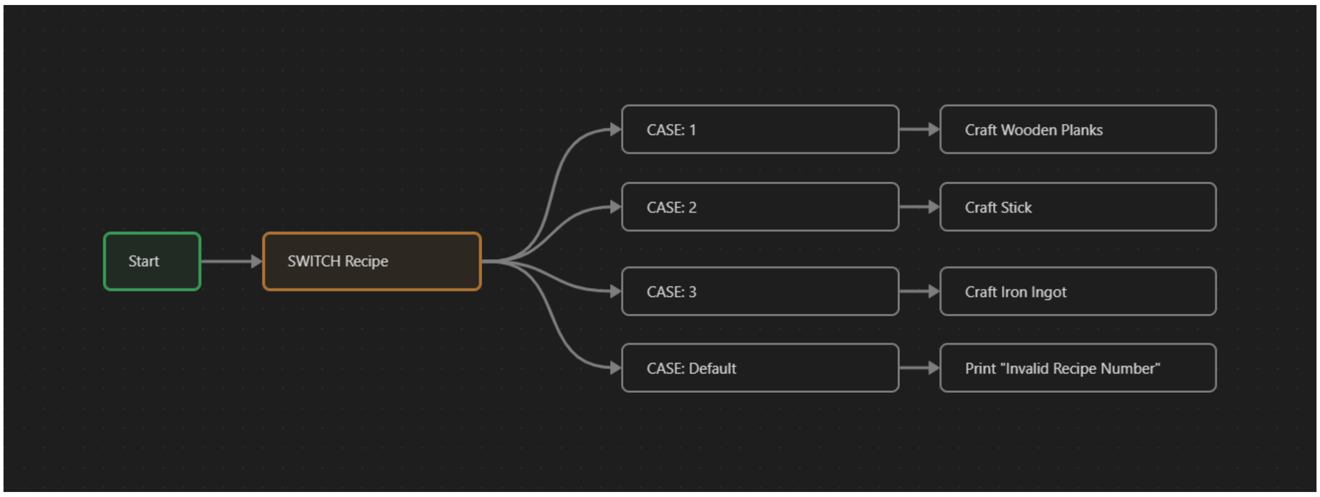


Figure 15A

```
Algorithm CraftItem(int Recipe)
        Switch (Recipe)
                CASE 1:
                        Craft Wooden Planks
                CASE 2:
                        Craft Stick
                CASE 3:
                        Craft Iron Ingot
                DEFAULT:
                Print “Invalid recipe number.”
END
```

Figure 15B

Figure 16: CraftWoodenPlanks

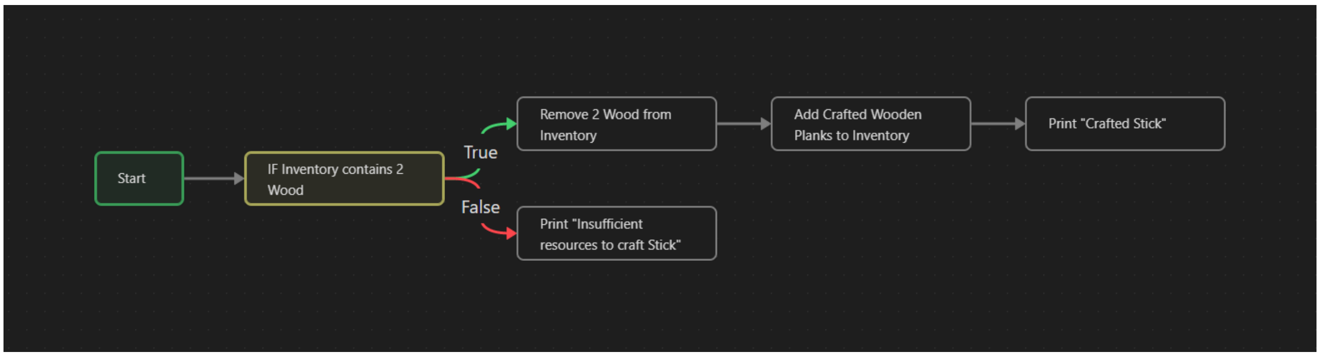

Start
IF Inventory contains 2 Wood
True
False
Remove 2 Wood from Inventory
Add Crafted Wooden Planks to Inventory
Print "Crafted Stick"
Print "Insufficient resources to craft Stick"

Figure 16A

```
Algorithm CraftWoodenPlanks
        IF Inventory contains 2 Wood THEN
                Remove 2 Wood from Inventory
                Add Crafted Wooden Planks to Inventory
                Print “Crafted Stick”
        ELSE
                Print “Insufficient resources to craft Stick”
END
```

Figure 16B

References