JavaCraft

Final Report: Group 43

Saturday, October 7, 2023

**Table of contents**

[1 Introduction 2](#_Toc148809260)

[2 JavaCraft’s Workflow 3](#_Toc148809261)

[3 Functionality Exploration 3](#_Toc148809262)

[4 Finite State Automata (FSA) Design 5](#_Toc148809263)

[5 Git Collaboration & Version Control 7](#_Toc148809264)

[6 Extending the Game Code 7](#_Toc148809265)

[7 Interacting with Flags API (For Final Submission) 8](#_Toc148809266)

[8 Conclusion 9](#_Toc148809267)

[9 References 9](#_Toc148809268)

[10 Appendix 10](#_Toc148809269)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute | | Details | | |
| Group Name | | Syntax Error | | |
| Group Number | | 43 | | |
| TA | | José Manuel Ros | | |
|  | Student Name | | Student ID |  |
|  | Botond Moksony | | [I6365627] |  |
|  | Vicente Muñoz | | [I6346464] |  |
|  | Zhili Yang Wu | | [I6362675] |  |
|  | Malo Coquin | | [I6302990] |  |

# Introduction

|  |  |
| --- | --- |
| Name | Tasks |
| Vicente Muñoz | * Function definitions (25%) * Flowcharts (20%) * Pseudocodes (20%) * Game flowchart (100%) * FSA illustration & design (33%) * Git student (40%) * Git trouble summary (100%) * Flag (80%) * Added blocks (100%) * Raw meat(100%) * Fix border colors (100%) |
| Botond Moksony | * Function definitions (25%) * Flowcharts (30%) * Pseudocode (30%) * Appendix (100%) * FSA illustration & design (33%) * Git teacher (100%) * Report final revision (100%) * Flag (20%) * Cow (50%) |
| Zhili Yang Wu | * Function definitions (25%) * Flowcharts (40%) (Did startGame flowchart) * Pseudocodes (50%) * Game pseudocode (30%) * FSA description (100%) * FSA illustration & design (13%) * Git student (40%) * Cooked meat (100%) * Cow (50%) * Eat meat (50%) |
| Malo Coquin | * Function definitions (25%) * Flowcharts (10%) * Game pseudocode (70%) * FSA illustration & design (20%) * SourceTree student (20%) * Added crafting recipes (100%) * Eat meat (50%) |

# JavaCraft’s Workflow

* Flowchart For Game (also included in the appendix in higher resolution):A diagram of a flowchart

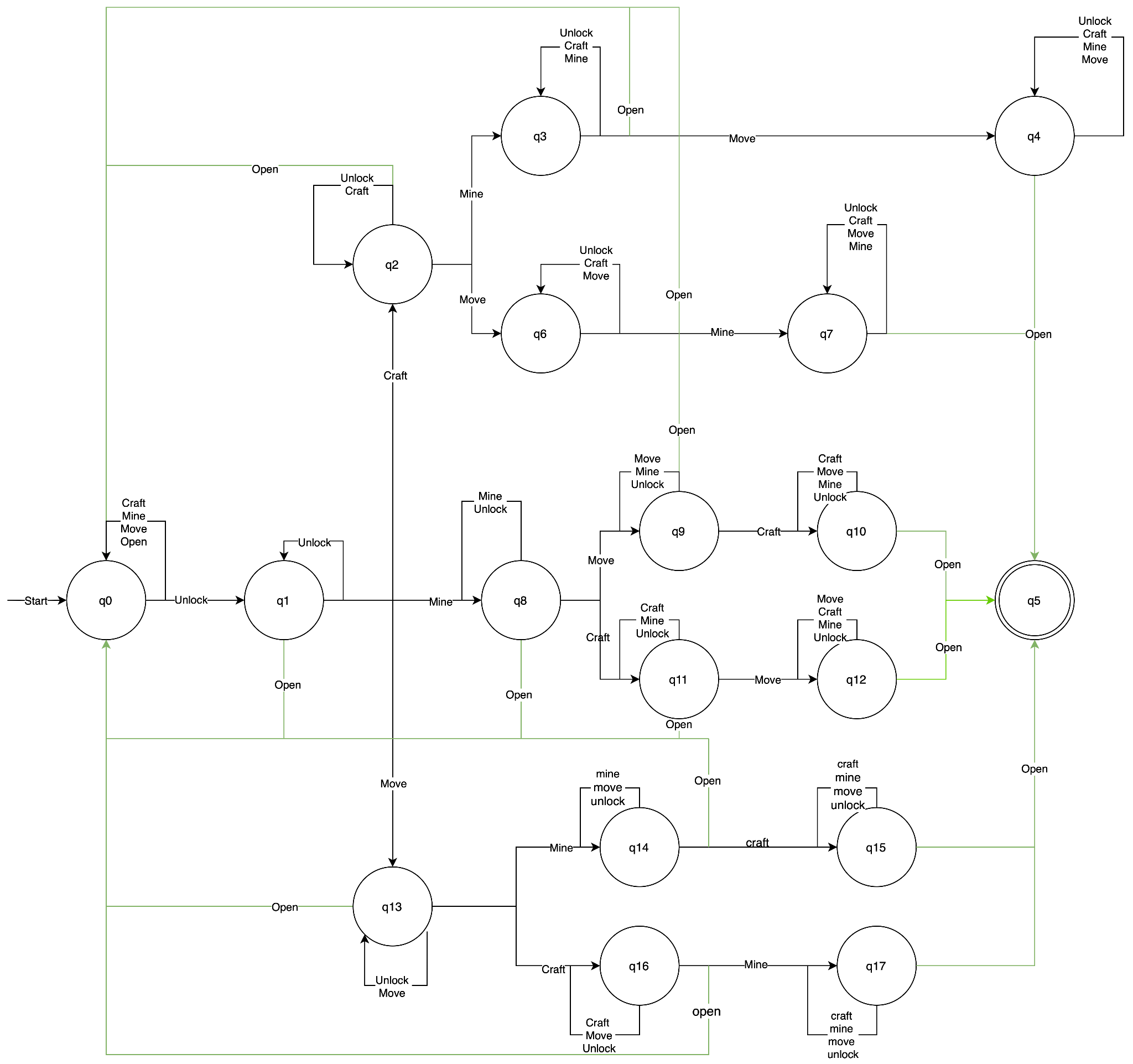
  Description automatically generated
* The pseudocode for the game can be found in the appendix.

# Functionality Exploration

List of key functionalities explored:

|  |  |  |
| --- | --- | --- |
| No. | Function Name | Description |
| 1 | main | Print out the game’s description and initiate the game after user confirmation |
| 2 | initGame | create the world width, set the initial position of the player & create the inventory |
| 3 | generateWorld | generate blocks for all the coordinates on the map based on random values |
| 4 | displayWorld | display map on terminal |
| 5 | getBlockSymbol | function for getting block colors and symbols (some blocks are represented by symbols, e.g.: $) |
| 6 | getBlockChar | function for getting the characters for the blocks |
| 7 | startGame | ask for the action, & call a function for the specific action we choose. It also can open the secret door but only with some conditions. |
| 8 | fillInventory | fills up the inventory with all the existing blocktypes |
|  | resetWorld | calls for the dutch flag function (generateEmptyWord) and moves the player to the middle of the screen |
| 9 | generateEmptyWorld | Generates the Dutch flag |
| 10 | clearScreen | clear the screen according to the os type |
| 11 | lookAround | prints out the blocks around the player |
| 12 | movePlayer | Changes player coordinates according to the input |
| 13 | mineBlock | It allows the player to mine the blocks to get materials if the specified coordinate is not air |
| 14 | placeBlock | It allows the player to put a material in a specific coordinate |
| 15 | getBlockTypeFromCraftedItem | Return the blockType from the itemID |
| 16 | getCraftedItemFromBlockType | Return the itemID from the blockType |
| 17 | displayCraftingRecipes | Prints out the crafting recipes. |
| 18 | craftItem | It allows the player to use their materials to create an item. |
| 19 | craftWoodenPlanks | It allows the player to use the wood they collected in the game and turn it into wooden planks |
| 20 | craftStick | It allows the player to use the wood they collected in the game and turn it into a stick |
| 21 | craftIronIngot | It allows the player to use the iron ore they collected in the game and turn it into iron ingots |
| 22 | inventoryContains | Checks whether the inventory contains a specific item. |
| 23 | inventoryContains | Checks whether the inventory contains a certain amount of items |
| 24 | removeItemsFromInventory | Remove a specified number of items from the inventory |
| 25 | addCraftedItem | Add itemID to the craftedItems list |
| 26 | interactWithWorld | Add block to inventory according to the place where the player stands |
| 27 | saveGame | Save the game variables, objects to an external file |
| 28 | loadGame | Load variables and objects from a previously saved game file |
| 29 | getBlockName | Returns the name of the block from blockType |
| 30 | displayLegend | print the legend with the symbols and names of the blocks |
| 31 | displayInventory | print the player’s inventory with the number of blocks that the player has |
| 32 | getBlockColor | return the color of each block |
| 33 | waitForEnter | It tells the player to “press enter to continue” whenever they need to, and waits for input. |
| 34 | getCraftedItemName | return the name of the item from craftedItem |
| 35 | getCraftedItemColor | return the color off block, but only for the iron ingot (brown). |
| 36 | getCountryAndQuoteFromServer | Communicates with an external server in order to get a flag of a country and a quote besides it. |

# Finite State Automata (FSA) Design



The secret door is a hidden option within the game that is activated when entering the following commands into the terminal:

* Unlock
* Mine (user has to input “m” in either upper or lower case in order to mine)
* Move (user has to either character from WASD in upper or lower case for moving)
* Craft (user has to select any crafting option in this command)
* Open

For opening the secret door function, the user needs to input these commands in specific orders, where each command is entered one and without any repetition in the following inputs. The main conditions needed to enter the sequence are the following:

* The Unlock command always needs to be the first command to be inputted (although you can input this command in intermediate steps as we will mention later).
* Open command has to be strictly the last command to be inputted.
* In between the first input (unlock) and the last output (open), the remaining commands and/or unlock are to be inputted in any order.

With the input of these commands, we can describe the sequencing of the input of these commands as a change of states within the game, where we can identify an initial state and a final/accepting state and determine a finite state automaton (FSA). Here, the FSA's alphabet is the set of all the commands accepted by the game as valid inputs (as described in the outputs of the game) and the language accepted by this is the set of commands mentioned earlier. By applying the conditions for the sequence of inputs for unlocking the door, we can get the following representation of the language:

𝚺 = { L {valid inputs within the game}\* | L contains ‘unlock’ as first string, has the following commands {‘unlock’, ‘mine’, ‘move’, ‘craft’} in any order and has ‘open’ as last string}

With this language, we will proceed to describe the steps required for reaching the secret door. First step, when the game locates us into the game right after entering the correct inputs for starting the game, we begin at the initial state q0, from which we move to q1 after entering the ‘unlock’ command. If we enter ‘unlock’ for a second time right after entering it from q0, we will not move from q1 as no new changes or alterations to the game are noticed, meaning we have a loop for ‘unlock’ in this state. From q1 and onwards, we will have loops in the states for all repeating commands that we input, meaning that we can only advance in the FSA by inputting the commands exactly once, as long as the condition that ‘unlock’ is the first command entered and ‘open’ is the last entered. For all cases, if the input ‘open’ is entered in intermediate states, the FSA returns the user to q0 losing all progress (as represented with the green line in the FSA diagram).

Upon opening the secret door the game plots a Dutch flag into the terminal and the player’s inventory gets filled up with all of the block types.

# Git Collaboration & Version Control

* Repository Link: [**https://gitlab.maastrichtuniversity.nl/bcs1110/javacraft/-/tree/group\_43**](https://gitlab.maastrichtuniversity.nl/bcs1110/javacraft/-/tree/group_43)
* Branch Details: **group\_43**

Git summary: We assigned different tasks of changing the code to different people on the team. Each one split off a different branch in which they worked on their task. After finishing their task, they would commit, push, and merge their branch back to the main group branch, in order to share their progress with the rest. Afterward, they would start a new task and thus a new branch, until the project is finished.

Changes & Conflicts & Problems:

* We had problems at the start understanding how to use git and how to commit/push/pull, but thankfully we had Botond there to help us understand all of this.
* We also had problems at one point when trying to push and pull, in which error messages kept appearing. We solved this by merging the previous branches, then pulling, and finally pushing.
* We had problems when setting up SourceTree, where we needed to create an access token and ssh keys in gitlab. We had many errors so we had to do the process all over again carefully and we made it work.
* Some conflicting branches were very problematic, so we had to copy paste parts of one to the other before effectively pushing them.

# Extending the Game Code

|  |  |
| --- | --- |
| New additions to the game | Integration into the game |
| Coal | We added coal as a basic block type in order to later use it as part of a new crafting recipe. |
| Diamond | We didn’t have any use for a second new block type, so we just added the most famous one from Minecraft to fill in the void. We made it rarer than all the other materials in the map, sometimes nonexistent. |
| Immortal Demon Cow  (Refer to appendix for code snippets) | The most complicated and tryhard thing we did. We made a cow represented by a purple “C”, that moves around the map in an interval between the player’s interactions. By “interacting” with the cow, the player can kill it and obtain meat (while a sad cow sound plays in the background), although the cow will remain alive and keep moving through the map, therefore its title. |
| Meat | Not exactly a block type, but if you “kill” the cow, it will add meat into your inventory and behave in the same way as other materials. |
| Crafting table  (Refer to appendix for code snippets) | Recipe: 4 wooden planks. It makes possible the crafting of the furnace. |
| Furnace  (Refer to appendix for code snippets) | Recipe: 1 Iron ore, 2 Coals. Only possible to craft it if you have a crafting table in your inventory. |
| Cooked meat  (Refer to appendix for code snippets) | Counts as a crafting recipe, but its kind of the final goal in a way. Recipe: 1 meat. Only possible to make if you have a furnace in your inventory. |

# Interacting with Flags API (For Final Submission)

In the getCountryAndQuoteFromServer() method, we send a request to Ashish's server (with the URL: https://flag.ashish.nl/get\_flag), that returns the name of a Country and a Random quote. For the request, we send our group number and group name, as well as the difficulty level that we chose. In our case that was “hard”, and the server sent us Sri Lanka as the country name.

The quote is:

*“Any code of your own that you haven't looked at for six or more months might as well have been written - someone else - Eagleson's Law”.*

To render the flag on the grid, we created a 2D int matrix that had numbers from 0 to 3, with each number representing a different color. After using a nested for loop to go through each part of the matrix, we used if-else if statements to print blocks of the color matching the number. We ordered the numbers on the matrix so that when it was printed, it would result in the Sri Lanka flag.

A red and yellow rectangular sign

Description automatically generated

# Conclusion

Our biggest achievement would be the implementation of the cow in the game, as it also was our biggest challenge to go through. We had many issues such as the cow completely breaking the game, or just spawning on a specific corner of the grid, plus many others. Another achievement would be the flag, as we chose a considerably hard one. And another important achievement would be how our team dynamics were excellent, with no ghost members or people who did nothing. Our biggest challenge next to the cow would have been the use of git, due to its less than friendly usage. After doing this project, we improved in our coding skills, teamwork skills, our familiarity with java, and team management skills considering this was our first ever project.

# References

* Guinto, Susan. *briks*. *Brik*, <https://cdn.apps.joltteam.com/brikbuild/flag-of-sri-lanka-pixel-art-191320199.pdf> . Accessed 18 Oct. 2023. - Image used for the flag
* Nabizade, M. (2021, June 21). *How to Play Background Music In Java Programs*. Murad Nabizade. Retrieved October 18, 2023, from <https://www.muradnabizade.com/backgroundmusicjava> - Code used for playing the cow sounds.

# Appendix