Canterbury Christ Church University

School of Engineering, Technology and Design

BSc/BEng Computer Science (Hons)

2024-2025

Individual Project (IP40)

Little Man Computer educational game

James Ramsey Haddad

Supervisor: Tina Eager

jh1662@canterbury.ac.uk

This report is submitted in partial fulfilment of the requirement for the BSc/BEng in Computer Science (Hons) at Canterbury Christ Church University.

I declare that this report is my own original work containing no personal data as defined in the Data Protection Act and that I have read, understood and accept the University's regulations on plagiarism/intellectual property rights/research ethics (the Research Governance Handbook) and the Individual Project Module Handbook in its entirety.

Further, I accept that digital and/or hard copies of my Individual Project, or parts thereof, may be made available to other students, individuals and organisations after it has been marked.

Finally, I accept that no copy of my Individual Project will ever be returned regardless of the circumstances.

Signed
James Haddad

Date of Submission:
3rd June 2025

1 Abstract

This dissertation presents the design, development, and evaluation of a browser-based Little Man Computer (LMC) educational game intended to deepen teenage students' understanding of the von Neumann architecture and introductory assembly programming. Grounded in a comprehensive literature review, the project fuses Behaviourist, Cognitivist, Constructivist, Humanist, and specialist theories with game-based learning principles to create an engaging, self-paced environment that minimises teacher intervention.

A systematic survey and weighted comparison of six widely used LMC simulators exposed functional gaps in usability, accessibility, and pedagogical scaffolding. These findings informed requirements that retained LMC's canonical instruction set while introducing left and right shift operators and ASCII output to broaden problem-solving possibilities.

The solution was engineered in TypeScript with Bootstrap-assisted framework, for GUI, and delivered through three Agile sprints. sprint #1 produced a standards-compliant, fully tested von Neumann cycle simulator; sprint #2 added a compiler, sandbox editor, dark/light modes, and keyboard-centric controls; sprint #3 gamified the platform with structured levels, a three-star performance rubric, and adaptive tutorials. Rigorous automated Jest testing, cross-compatibility checks against external LMC examples, and user tests with computer science students and teachers verified functional correctness and classroom viability by feedback implementation. The solution is ported to work on Windows, without internet connection, for scenarios that requires it. such scenarios as if the school computers have a web filter.

Legal, ethical, and accessibility considerations – including GDPR compliance, open-source asset licensing, and colour-blind-friendly palettes – were addressed throughout. The finished web app, deployable as a progressive web application, demonstrated smoother interaction, shorter learning curves, and higher student engagement than existing tools, according to qualitative feedback from teachers and learners.

Ultimately, the project shows that thoughtfully gamified processor simulators can bridge theoretical and practical learning, offering a scalable model for future computer-science education resources.

2 Acknowledgements

- The author for doing the project.
- My supervisor (Tina Eager) for advising the author during this project.
- Mr Jacob, from Canterbury Academy, and the anonymous students who helped with testing and providing feedback.
- Mr Lewis, from EKC Group Canterbury College, and the anonymous students who helped with testing and providing feedback.
- All authors listed in the **Bibliography**.

3 Contents

Below is a table of contents. Each entry is also a cross-reference hyperlink.

Table of Contents:

4 Introduction Chapter

4.1 Product proposal and objective

The proposed product will be an educational game as a CPU simulator, in the LMC standard. It is suited for the age range of teenagers (9-18), as the main audience, but will also be used by younger people and teachers. It will be a major improvement over the existing LMC simulators.

The game is based on 2 modes. One will have levels that give the user tasks to proceed while the other is a sandbox where users can advance their skills after completing the levels. Both the levels and sandbox trigger critical thinking and strategies (referring to/basing on previous work or using a logical thought pattern) to solve logical problems. Younger teenagers will have these concepts introduced to them smoothly, while older teenagers will appreciate and solidify them for more advanced uses.

The author hopes for the LMC educational game to be an attractive and effective method of teaching students problem-solving skills, the rough workings of the von Neumann processor architecture, and the ability to do so without the constant direct intervention of teachers.

The reader can try out the LMC educational game product using the URL below (Haddad, 2025b).

https://jh1662.github.io/LMC_simulator_game

This was made possible by GitHub's web hosting of the author's repository (Haddad, 2025c).

To see the source code please see the zip folder submitted alongside the dissertation. Zip folder also containers the ported Windows executable of the LMC educational game.

4.2 Nature of the dissertation

For the user's convenience, figures and tables are frequently used to help explain/showcase things simplistically and to reduce word count. Each one has a caption underneath, in a size-9 italic dark-blue font (still Times New Roman), which contains a unique identification and quick description. Captions can be referenced by cross-reference hyperlinks, such as **Table 5.1—1** (size-10 bold light-blue Times New Roman font), which leads/focuses to said caption when left-clicked with the control key pressed.

Other custom fonts the user must be aware of:
Code quotes/names – size-9 italic dark-green Consolas font.

File/folder names – size-9 bold dark-green Consolas font.

Significant quotes – size-10 centred italic grey Times New Roman font.

Whenever the user is lost in the dissertation, please refer to the table of contents.

Some of the chapters, such as the conclusion, most of its content are linked to other parts (mostly the appendices). Please utilise the cross-reference hyperlinks and treat the linked contents as part of the main chapters that referenced them.

With that, the author wishes a happy and inspiring reading to the reader.

5 Main Chapters

5.1 Literature review

5.1.1 The Problem

5.1.1.1 Lack of motivation

Education is a vital necessity for one's path in the future; however, most people only bother to do the bare minimum to pass each educational stage. Reasons are many, from "*impeding the smooth actualisation of the educational purpose*" to "*lack of motivation*" (Mauliya, Relianisa and Rokhyati, 2020) to how one person can influence another person's learning (Muhajirah, 2020). Not only does this risk failing higher education, but it also discourages people from entering advanced education beyond what is legally required or beyond living a good enough life. The problem for the individual is multiplied for the collective of society (Mauliya, Relianisa and Rokhyati, 2020) thus, causing academic stagnation.

5.1.1.2 Lack of qualified teachers and time

This problem is especially the case for subjects that lack either or both enough qualified teachers and teaching hours, one of them being computer science, as data shows (GOV.UK, 2024):

	Number of hours taught for all years	Number of teachers of all years
Computer science	66,216	7,852

Figure 5.1—1 - the computer science entry of the 2023/24 "Subjects taught" dataset published from the "school workforce in England" (GOV.UK, 2024).

Total	3,768,681	232,765

Figure 5.1—2 – entry displaying the cumulative total for all entries of the 2023/24 "Subjects taught" dataset published from the "school workforce in England" (GOV.UK, 2024).

The author has displayed and compared the computer science entry to the cumulative total by both quantity and percentage proportions, as shown in the project presentation below.

	Hours taught	Specialised teachers
Computer Science	66,216 (1.8%)	7,852 (3.4%)
Total	3,768,681	232,765

*Figure 5.1—3 - a table, from the author's presentation, that takes data from **Figure 5.1—1** and **Figure 5.1—2**, and compares them both quantitive and percentage-wise.*

1.8% and 3.4% (to one decimal point) are concerningly low proportions, indicating that very few hours of Computer Science lessons are taught to students, let alone the lack of specialised teachers, resulting in lessons being taught by underqualified teachers.

5.1.1.3 How it relates to the product

To help address and mitigate this problem, the author decided to work on an educational simulator game as described in **Product proposal and objective**.

The idea behind the project is that students will not only regain motivation via educational gaming, as explained in **The Solution**, but be able to effectively learn without much interference/teaching from qualified teachers – self-teaching. Self-teaching will help reduce the current stretch and pressure on qualified computer science teachers. By making the LMC educational simulator game engaging, the students will play the educational game out of school hours, sparing teaching hours for other Computer Science topics and students.

5.1.2 Educating by educational theories

5.1.2.1 Introduction

This phenomenon is caused using ineffective learning techniques. In the academic world, many researchers have come up with many learning theories that would exponentially improve the learning abilities of students if the requirements were met. There are 4 basic but popular theories that are widely accepted throughout schools, these are:

1. Behaviourism – Introduced by John B. Watson (Watson, 1913).
2. Cognitivism – Introduced by Jean Piaget (Piaget and Warden, 1927).
3. Humanism – Introduced by Abraham H. Maslow (Maslow, 1943).
4. Constructivism – Introduced by Jean Piaget (Piaget, 1955)

Deemed the most accepted and widely adopted theories (Muhajirah, 2020). Each of these theories was constantly improved on, to the current age, and branched into variations of itself.
Those are the theories for general theories, but in the Computer Science field, there are more modern and more specific learning theories that fit the field's speciality:

1. Multiple Intelligences Theory – Introduced by Howard E. Gardner (Howard Gardner, 1987).
2. Situated Learning Theory – Introduced by Jean Lave and Etienne C. Wenger (Lave and Wenger, 1991).
3. Mindset Theory – Introduced by Carol S. Dweck from her 2006 book "Mindset: The New Psychology of Success" (Dweck, 2016)

This paper will provide an overview of these theories and their relevance to a student's education.

5.1.2.2 General theories

5.1.2.2.1 Behaviourism

Behaviourism is a simplistic, systematic approach to linking learning and reflexes to stimuli as the same thing (Muhajirah, 2020; Watson, 1913). It states that a creature's behaviour is suited for learning when they interact with the stimuli of their surrounding environment (Burhanuddin *et al.,* 2021; Muhajirah, 2020). Students can reach this state of behaviour by observing simulations and witnessing the result and mechanics. An example of this is a Computer Science student witnessing how a piece of code gets executed to solve a problem, rather than being simply told what code will solve what problem.

5.1.2.2.2 Cognitivism

Cognitivism was made in response to counter Behaviourism (Brieger, Arghode and McLean, 2020), rather than reaction to the physical stimuli being the key factor in a person's learning, these theorists firmly believed that learning is how humans mentally process and manage information in response to comparing new information to old (Muhajirah, 2020; Brieger, Arghode and McLean, 2020). For education to take the cognitivist approach, the student must be presented with new information, in the form of a problem, but is tasked with critically thinking to solve the problem with whatever knowledge is already known. Such can be demonstrated by a Computer Science student thinking and then creating a piece of code that will solve the problem, rather than just being given the problem-solving code.

For more about Cognitivism and its linkage to other educational theories, please read articles regarding Case-Based Learning (CBL).

5.1.2.2.3 Constructivism

Constructivism takes some policies of Behaviourism and Cognitivism, yet it counters their beliefs. Constructivism states that one learns by taking the knowledge of observing new and different experiences, also known as "cognitive conflict" (Muhajirah, 2020), and using that to recognise, process, and solve new problems (Muhajirah, 2020; Piaget, 1955), different to those who experience rather than continuously learning from education (Muhajirah, 2020). It takes the Behaviourism policy of observing and taking in experiences. But rather than those experiences being the knowledge itself, the experience is used with cognitivism to solve problems that are different to those experiences and gain new knowledge from that (Muhajirah, 2020; Burhanuddin *et al.,* 2021). Taking both Behaviourism and Cognitivism allows the student to form a continuous cycle of first solving problems, then using that problem-solving experience to solve another slightly different problem. Constructivism does not teach how to solve one problem, but a series of connected problems (Burhanuddin *et al.,* 2021). The Computer Science student can enact this by using a task they solved before to solve another problem. Constructivism in this can be the Computer Science student using a coding problem, that the student solved with the student's code, from last academic year, to solve a slightly more complex problem using the knowledge and skills acquired from last time. As mentioned, constructivism differs from Behaviourism and Cognitivism by being longer-term, however, this brings up more issues, such as how the student can solve the first problem or how much more difficult or less relevant the next problem is before Constructivism can no longer apply.

5.1.2.2.4 Humanism

Humanism takes a completely different approach because it focuses more on the context surrounding the student's well-being rather than the learning process itself (Muhajirah, 2020). It is more philosophical when compared to the other 3 primary theories (Muhajirah, 2020). Focusing more on the context means that Humanism takes care of the students' needs and support (Muhajirah, 2020). This can range from making the student feel comfortable, such as the student's feelings, interests, dignity, safety, et cetera (Muhajirah, 2020; Maslow, 1943), to supporting the students' learning process, such as countering any learning disabilities and motivating the student (Maslow, 1943). A Computer Science student can its needs carried out by having frequent recesses in between to fulfil them. Meanwhile, taking care of a Computer Science student's support really depends on the student. For example, one student who is colourblind can be supported by making the learning material content colourblind-friendly/accessible or supporting a poor-sighted student by simply making that content (such as texts and diagrams) bigger instead. Motivating students can be done by making the learning process more fun, as seen when rewards or the yearning for a sense of accomplishment are involved.

5.1.2.2.5 Conclusion

In conclusion: Behaviourism is learning the acknowledging by observation, Cognitivism is learning problem-solving by critical thinking, Constructivism is learning by linking together what is already learned, and Humanism is supporting the learner and their needs to prevent hindrance in learning.

5.1.2.3 Specialised theories that fit Computer Science

These are not as widely used nor basic, but their specialism may help in the Computer Science field.

5.1.2.3.1 Multiple Intelligences theory

The Multiple Intelligences theory states that human intelligence is split up into multiple intelligences: linguistic, logical-mathematical, musical, bodily-kinesthetic, spatial, interpersonal, intrapersonal, and naturalistic (Howard Gardner, 1987). For a deeper explanation of each of these intelligences, please look at Howard Gardner's work (Howard Gardner, 1987). For example, a student with a dominant linguistic intelligence will learn better with written or spoken language, meanwhile a student with a dominant spatial intelligence will learn better using visualisation or real-world demonstrations (Howard Gardner, 1987). The theory believes that by this logic, teaching in a way to a variety of intelligences, students are more likely to have their dominant intelligence catered to – making their personalised learning more effective. A Computer Science student must simply learn in the way that suits their main intelligence, which students of the Computer Science subject may prefer intelligences that fits with "STEM", such as logical-mathematical intelligence.

5.1.2.3.2 Situated Learning theory

The Situated Learning theory says that successful learning must be meaningful in context (Lave and Wenger, 1991). This is achieved by making the practice of learning related to real life work (Lave and Wenger, 1991). Unlike the other theories, this is an industrial standard with students taking apprenticeships and internships, letting them learn through practising authentic tasks that have contextualised real-life meaning (Lave and Wenger, 1991). A Computer Science student may experience "situated learning" by doing coding tasks that refer to what tasks real companies give to their developer employees, but those tasks are simplified so the student can do them. While not as popular as the 4 primary learning theories, its principles are used in other theories, such as in Connectivism and Vygotsky's zone of proximal development (Brieger, Arghode and McLean, 2020).

5.1.2.3.3 Mindset theory

Lastly, there is the mindset theory. Unlike the Multiple Intelligences theory and the Situated Learning theory, which focus more on the student's mindset. The theory believes that the student's learning will be far more successful if the student prioritises the "*growth mindset*" over the "*fixed mindset*" (Dweck, 2016). An example of the differences in these mindsets is a student with a fixed mindset, who may avoid challenges to avoid the feeling of failure, however, a growth mindset will instead embrace the challenges as the effort and feedback of it will be an opportunity to grow. The change in mindsets is easier than it seems (Dweck, 2016), with simple changes such as following the Humanism theory by supporting and motivating the student to willingly do these challenges. A Computer Science student can easily lean more towards a growth mindset the same way as other students: support and satisfying needs, praise and rewards when successfully completing a task, use of role models, resilience strategies, encouragement and motivation, et cetera.

5.1.2.3.4 Conclusion

In conclusion, while these theories are more complex and specialised than the general theories, they offer great benefits to learning in their respective fields of learning. Multiple Intelligences theory promotes better understanding by more personalised learning, Situated Learning theory lets students understand by learning from a company worker's perspective, and Mindset theory makes students willingly want to learn better instead of compulsory learning.

5.1.2.4 The Solution

5.1.2.4.1 Utilising the popularity of gaming

One crucial (and most popular) thing people dedicate their leisure time to is entertainment (Nakatsu *et al.,* 2017), and one big part of it is gaming (Nakatsu *et al.,* 2017; Possler, Kümpel and Unkel, 2020). This is "especially for the young generation" (Nakatsu *et al.,* 2017). Gaming can vary from relaxing to interactive and intense. Games can be any genre (Sci-fi, fantasy, visual novel, et cetera), but the one relevant here is the education genre. Educational games let people of all ages (C. Girard, J. Ecalle and A. Magnan, 2012) educate themselves when happily and willingly spending time and effort playing the game (Osman, Bakar and Nurul, 2012). This different approach has been met successfully many times (Osman, Bakar and Nurul, 2012). So much, that schools use them the encourage their teenage students to learn to the best of their ability, mostly using a game collection website called Coolmath games (Coolmath) - a very popular website containing hundreds, if not thousands, of inspiring educational games – and the block-programming website Scratch.

5.1.2.4.2 How learning theories tie in with game theory

Gaming not only is popular among students, but it also satisfies the learning theories, making the game the perfect solution for students to learn effectively learning. The basis of the game can perform and display simulations, provide problem-solving campaign levels, progressive/sequential campaign levels with tutorials, and disability modes to counter any learning disabilities to satisfy Behaviourism, Cognitivism, Constructivism, and Humanism, respectively. More details on how these game features satisfy the primary theories, alongside the secondary theories, can be found in the table below:

Learning theories	**Game feature(s)**	**How does it satisfy the theory to enhance learning**
Behaviourism	Simulating the LMC processor.	Student enhances learning by observing and reacting to the stimuli brought by a display of the simulator LMC processor simulating.
Cognitivism	problem-solving campaign levels.	Student enhances learning by using critical thinking and problem-solving skills to solve tasks/levels of the LMC processor.
Constructivism	progressive/sequential campaign levels with tutorials.	Student enhances learning by using what was learnt from the previous level to do the next level. The tutorials help address Constructivism's issue of starting the learning cycle by giving knowledge and skills on how to solve the first level.
Humanism	Disability display modes and game fanciness (like sound effects).	Students enhance learning by countering needs with the displayed game being friendly, catering to as many disabilities (like displaying colourblind-friendly content) as possible, with modes that change the display itself to suit other disabilities, such as changing the background's colour to ease people with dyslexia.
Multiple Intelligences	Able to display information to the student in many ways.	Students' learning is enhanced by trying to convey information to the student in as many ways as possible, such as being able to code in mnemonics, numbers, or full words.
Situated Learning	Adding real-life context to each level's objective.	Students' learning is enhanced by making each level's objective seem like something an employee would do in the real world.
Mindset	Maximise rewards mechanisms and minimise any form of punishment in the game.	Students' learning is enhanced by being encouraged to finish the game level campaign for a sense of accomplishment. Also, there will not be any negative text or other responses to failed code to minimise punishment.

Table 5.1—1 - explaining how the LMC educational game features satisfy the primary and secondary educational learning theories.

5.1.2.4.3 Game's effect and speciality

This phenomenon has inspired this project to be a simple educational game to help people learn for enjoyment. By being a progression (by levels) and sandbox game, students of either genre will all be attracted to the game.

As this will be only one game, it has been decided to specialise this game to teach the users programming (specifically assembly programming). This is so as the author will use my own experience and skills in programming will be of essential use to make the game without much practice.

Not only is it in the author's expertise, but educational programming is very popular, take Scratch (Scratch Foundation, 2024b) for example, with them claiming "Since 2007, more than 130 million children in every country worldwide have created more than a billion Scratch projects" (Scratch Foundation, 2024a). This project, however, narrows the programming down to a LMC simulator. LMC simulator is a simplistic Von Neumann architecture CPU, programmed in assembly code. Students playing this game get to learn how the dominant CPU architecture works and the skills of coding in assembly code.

5.1.3 The processor simulator

5.1.3.1 The importance and dominance of Von Neumann architecture

Despite the Von Neumann architecture's immense age, the architecture of all LMC CPUs are implementations of the Von Neumann architecture as seen in **Figure 5.1—4** (Eigenmann and Lilja, 1998). This is so because the Von Neumann architecture is the dominant CPU architecture idea (Arikpo, Ogban and Eteng, 2007; Eigenmann and Lilja, 1998). All popular modern CPU architectures (x86, ARM, RISC-V, et cetera) are all fundamentally based on the Von Neumann architecture. It beats other

proposed CPU architecture ideas, the popular beaten one being the Harvard architecture. Harvard architecture differs by having a discrete data memory and instruction memory (Eigenmann and Lija, 1998), whereas Von Neumann has them unified as one memory (Arikpo, Ogban and Eteng, 2007; Eigenmann and Lilja, 1998) (also known as RAM). Harvard architecture excels at simultaneous performance by not using a shared bus for both memories and security, as the data memory can be manipulated without changing the instruction memory. Despite this, Harvard easily falls out of favour as its increased complexity severely makes its designs much more difficult to implement and is far more expensive to manufacture. Von Neumann's preference over Harvard architecture is amplified by Harvard architecture's performance advantage, of faster communication from having 2 buses, by simply increasing the buses' bandwidth in Von Neumann architecture, as mentioned by Eigenmann and Lija (1998).

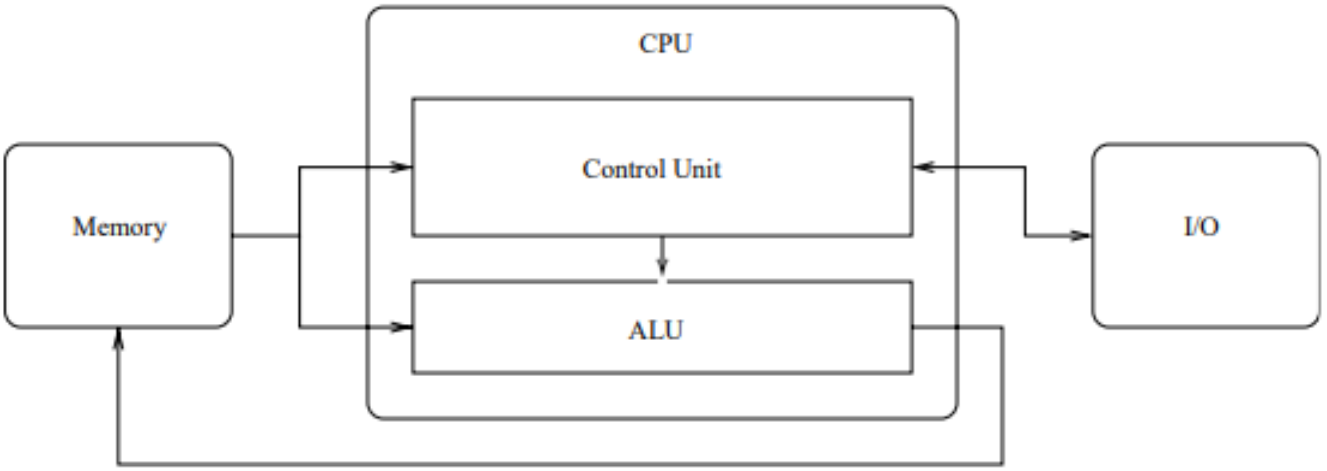


Figure 1: The original von Neumann architecture (Source: Lilja et al., 1998)

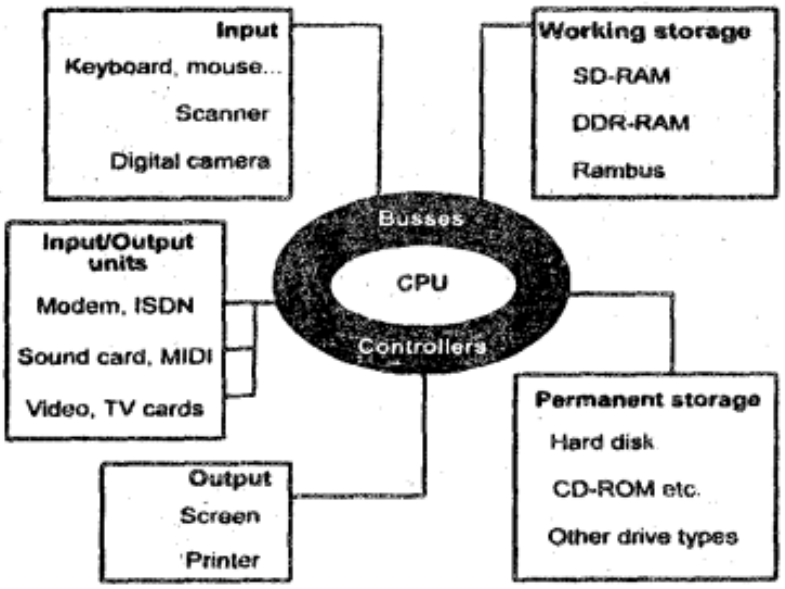


Figure 2: The von Neumann architecture in modern computers (Source: Lilja et al., 1998)

Figure 5.1—4 - Eigenmann and Lija (1998) compare the original 1945 Von Neumann architecture to it in modern computers.

Diagrams copied and supported by Arikpo et al. (2007). Sourced from Arikpo et al. (2007).

A clearer version of this has been made below when creating the poster presentation, which you can see for a clearer understanding.

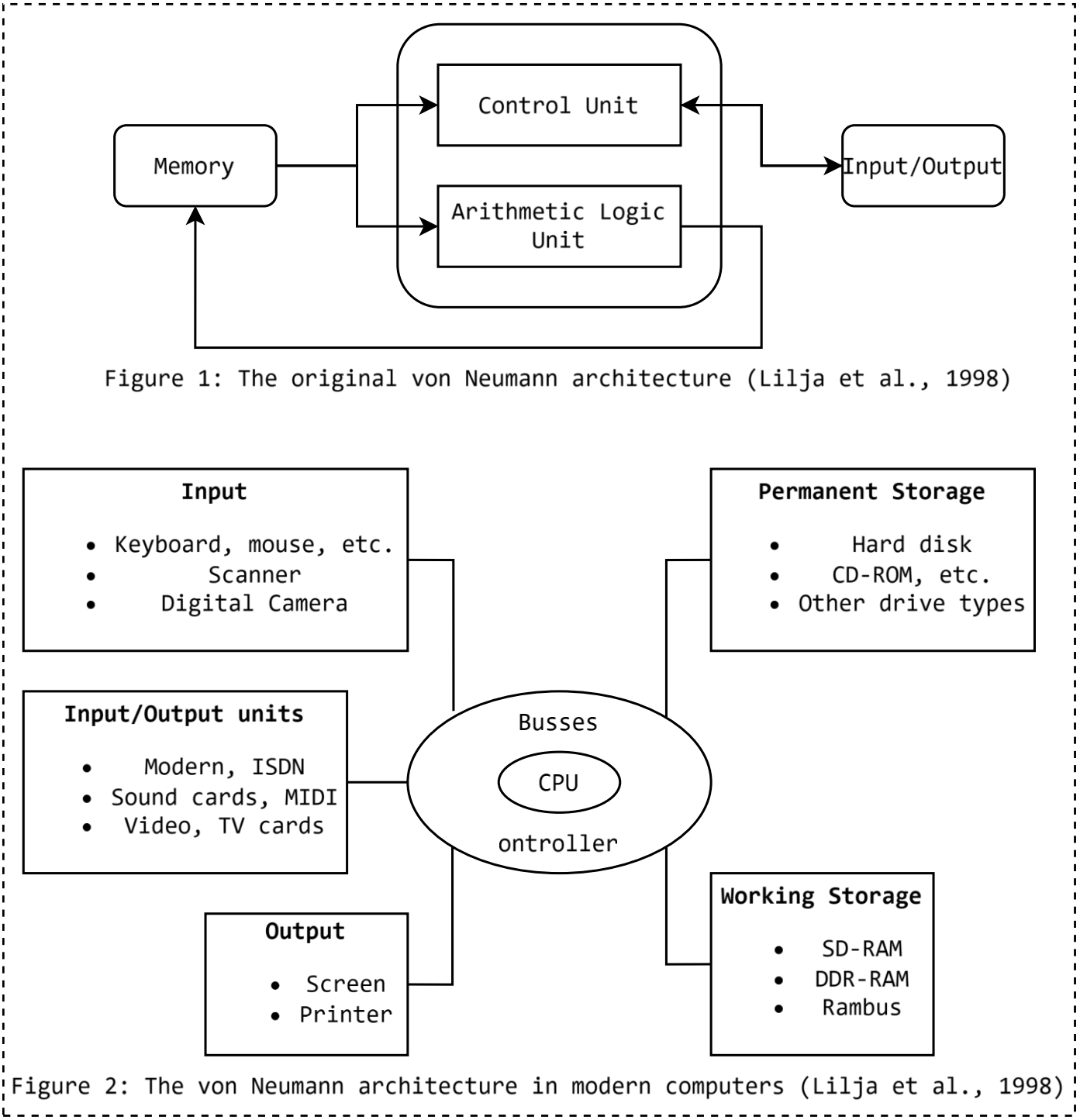


Figure 5.1—5 - a recreation of the diagram comparing the original 1945 Von Neumann architecture to it in modern computers.

5.1.3.2 Relevance

The von Neumann architecture has been fully researched for its accurate implementation into the product for two primary reasons. First, it sticks to the LMC standard. All explored LMC simulators are made with the architecture in mind, such as the unified RAM/memory, further backed up by the academic papers which went over the LMC concept. Second, by incorporating concepts (like the von Neumann architecture) which are widely used in the industry, we promote the Situated Learning theory for the student's better understanding when learning by playing the game.

5.1.3.3 LMC origins

Yurcik and Brumbaugh (2001) stated that the model of LMC was first introduced by Dr Stuart Madnick, from the Massachusetts Institute of Technology, in 1965 and then later improved by Irv Englander, from Bentley College (Yurcik and Brumbaugh, 2001). This statement is highly supported by other academic papers (Yurcik and Osborne, 2001; Osborne and Yurcik, 2002). At that time, LMC was just a paradigm in Englander's book (I. Englander, The Architecture of Hardware and Systems Software, John Wiley Sons, 2000.). Getting details of the paradigm directly is not possible because access to Englander's book is restricted by a paywall. The LMC paradigm uses the analogy of a "*little man*" in a mailroom (Yurcik and Brumbaugh, 2001; Yurcik and Osborne, 2001; Osborne and Yurcik, 2002) to visualise and teach the implementation of the von Neumann architecture (Yurcik and Brumbaugh, 2001). Despite LMC introduction being several decades ago, it is still widely taught and used. Academic sources

from the 2000s support this claim (Yurcik and Brumbaugh, 2001; Yurcik and Osborne, 2001; Osborne and Yurcik, 2002), and this still holds today with many modern and popular websites simulating LMC for the primary purpose of being taught in schools. Such websites can be found in the **Software research** section of this literature review.

5.1.3.4 LMC standard

Many decisions have been made in favour of sticking as close as possible to the LMC standard. Extra features may be added as a superset of LMC for learning extra concepts, but the core must remain untouched. The main reason for this standardisation is for the sake of the students' learning.

The consistency of standards makes sure that all students have the same uniform learning. Having this allows the development of extra comprehensive educational resources shared across multiple schools. Being across multiple schools, the shared resources are constantly updated up to date and removed and errors or missing information. This allows teachers not only to save more time than making their own resources but also to use more up-to-date and perfected resources for the students' learning. This is highly relevant for LMC as some of the Computer Science exam board syllabuses incorporate the topic of LMC, such as OCR (Oxford, Cambridge, and RSA Examinations). And LMC is still quite relevant for other exam-board syllabuses, because as mentioned, LMC excels in teaching how the fundamentals of a von Neumann CPU work, which is included in most (if not, then all) of the other exam-board syllabuses. Also, the standard of LMC is very effective to begin with, so there is no reason to reinvent the wheel.

5.1.4 Software research

5.1.4.1 LMC simulators

As the proposed project is an improvement/upgrade of existing software, software research will be based on other LMC simulators. The table below shows the most popular processor simulators that the author can find on the internet that follow LMC standards - CPU simulators based on Little Man Computers (LMC).

Links (for simulator)	Reference number (#)	Introduction page	Source code	Introduction has manual?
(Higginson, 2014)	1	(Higginson, 2024b)	(Higginson, 2021)	Yes
(101 Computing, 2019a)	2	(101 Computing, 2019b)	Not found	Yes
(Brinkmeier, 2017a)	3	(Brinkmeier, 2017b)	(Brinkmeier, 2017c)	Yes
(Gamble, 2022a)	4	(Gamble, 2022b)	(Gamble, 2024)	Yes
(trincot, 2021a)	5	(trincot, 2020)	(trincot, 2021b)	No
(Hankin, 2016b)	6	(Hankin, 2016a)	Not found	No

Table 5.1—2 - vital information of other LMC simulators.

Notes derived after reviewing each LMC simulator:

- LMCs #1 and #2 are far more popular and regularly used for education purposes while the others are more like side projects on GitHub.
- All LMCs refer to the Wikipedia page of LMC as a source of further information and learning about LMC. Specifically, the link to the Wikipedia page is linked in the manual page for LMC #1, #3, #4, #6 and in the simulator page for LMC #2 and #3. Wikipedia is not directly linked in LMC #5, but its simulator does say "*; For the syntax, see Wikipedia on "Little Man Computer".*"
- Points like the previous 2 bullet points will be accounted for in the weighting of which LMC is more reliable and hence preferred.
- Other CPU simulators that do not base themselves on LMC principles will be ignored, such as the AQA exam board CPU simulator aligned to the exam board's syllabus (Higginson, 2018) or based more complicated architecture (Higginson, 2016).
- LMC #5's introduction page shows the differences between LMC# 1,2, and 5. These will be used as a reference, but different when used. For example, testing the input range (and valid contents like decimal and letters) in LMC #1:

Table 5.1—3 - the top of the IDE of LMC #1 with a small assembly program coded to test what can and what cannot be accepted as input.

- While it may be possible to get the source code of all simulators by inspecting any JS file received by the browser, we do not know if the authors permit the use of their code that way. Ignoring the author's licenses may potentially lead to legal issues.

Note for Table 5.1—2: Please note that, as mentioned, not every LMC simulator has a manual and also the existing manuals does not explain everything about the LMC simulators. Therefore, this dissertation attempts to ensure its findings are nearly 100% accurate by trying out and testing the LMC simulators, it can never be 100% accurate.

5.1.4.2 Simulators' characteristics

LMC attributes listed below:

LMC #	RAM size	Valid inputs	Instruction set as mnemonics	Base	Shown registers	Compiler
1	100	integers -999 to 999	HLT, ADD, SUB, STA/STO*, LDA, BRA, BRZ, BRP, INP, OUT, OCT, DAT *'STA' and 'STO' are the same instruction – interchangeable **'OCT'	10 – decimal	PC, MIR, MAR, Accumulator	Detects syntax errors and prevents compilation if so. Uses double slashes (//) to initiate comments. Explains error when detected, for example "*unknown instruction at line 3 LDO*"
2	100	integers -999 to 999 and `NaN` for invalid inputs. However, text can be stored (as an address) if the STA instruction is called	INP, OUT, LDA, STA, ADD, SUB, BRP, BRZ, BRA, HLT, DAT	10 - decimal	PC, MIR, MAR, CIR, Accumulator	Detects syntax errors and compiles as if error lines never existed.
3	100	integers -999 to 999	ADD, SUB STA, LDA, BRA, BRZ, BRP**, INP, OUT, COB* * 'COB' (stands for 'coffee break') and 'HLT' does the same thing. ** 'BRP' here branches if carry register is '1' instead of the accumulator being positive.	10 - decimal	PC, Accumulator, Carry, Inbox (input), Outbox (output),	Detects syntax error and prevents execution but does not say the error's location. Instructions must be uppercase and labels in lowercase.
4	100	integers -999 to 999, input box doesn't allow other characters to be inputted	LDA, STA, ADD, SUB, INP, OUT, BRA, BRP, BRZ, HLT, DAT	10 - decimal	PC, MAR, MDR*, CIR *'MDR' stores whatever is stored to or retrieved from the RAM	Far simpler, and hence easier, as each line is split into 3 text boxes (labelled as "*Label*", "*Operator*", and "*Operand*"). Detects syntax errors and prevents compilation if so. Explains error when detected, for example "*Error, line 0: machine code instruction inp should not have an operand*".
5	?	integers 0 to 999	?	10 - decimal	Accumulator, Negative	Minimal UI and lack of manual makes it very difficult to use. Uses the semi-colon symbol (;) to initiate comments.
6	100	integers 0 to 999,	?	10 - decimal	Accumulator, PC, Neg, Ins*	Small width to bigger height ratio fits the LMC assembly language well (having a max of 3 syntax per line).

					* "*Ins*" appears to be counting how many "*clock cycles*" passed since execution (increases indefinitely after an error)	Detect only some errors before compilation. Doesn't detect errors in inputs despite being compulsorily entered before compilation.

Table 5.1—4 - features and characteristics of the other LMC simulators.

LMC simulator screenshots of their looks (when opened) in **Table 5.1—2** to analyse and compare their differing styles:

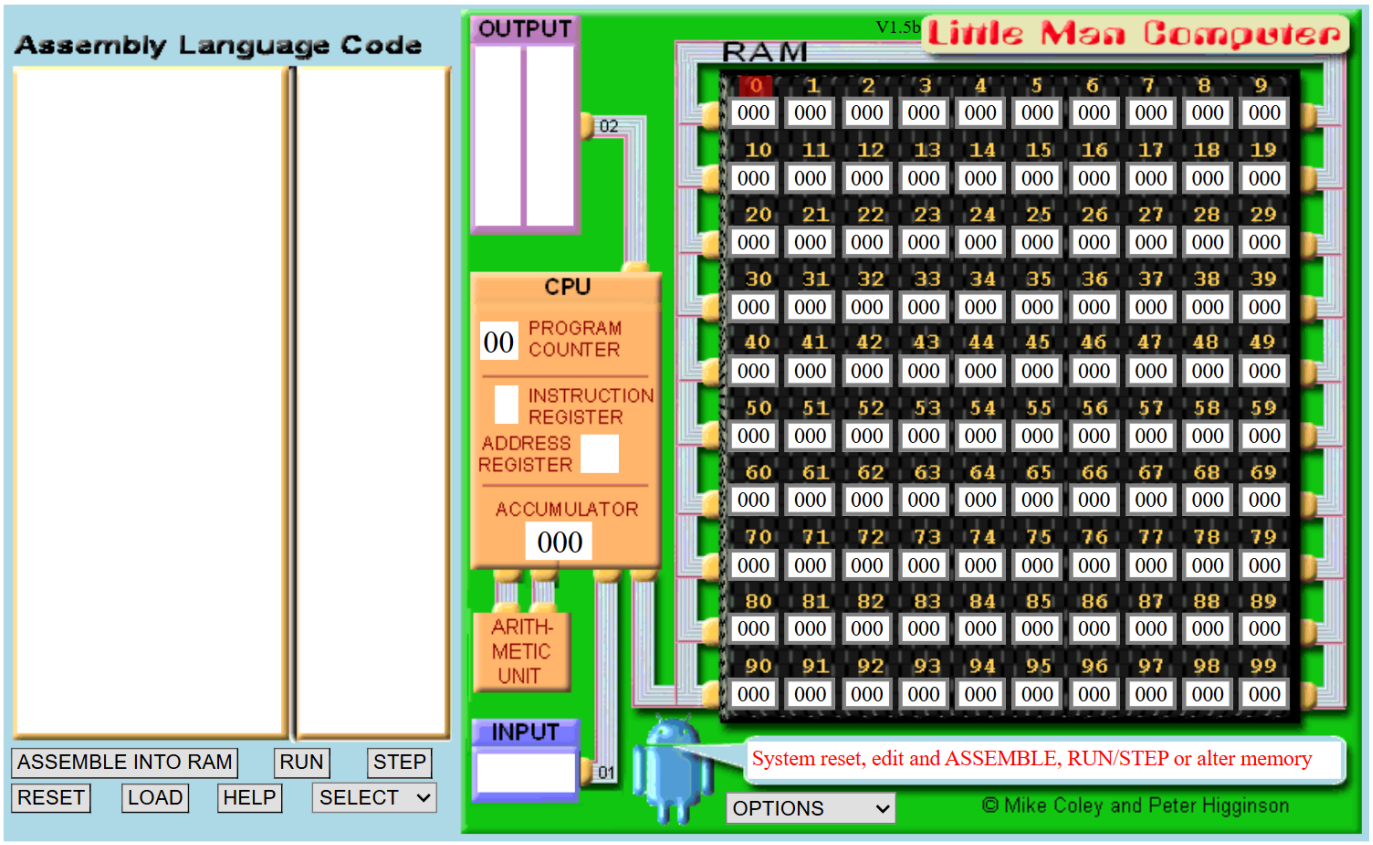


Figure 5.1—6 - screenshot of LMC #1

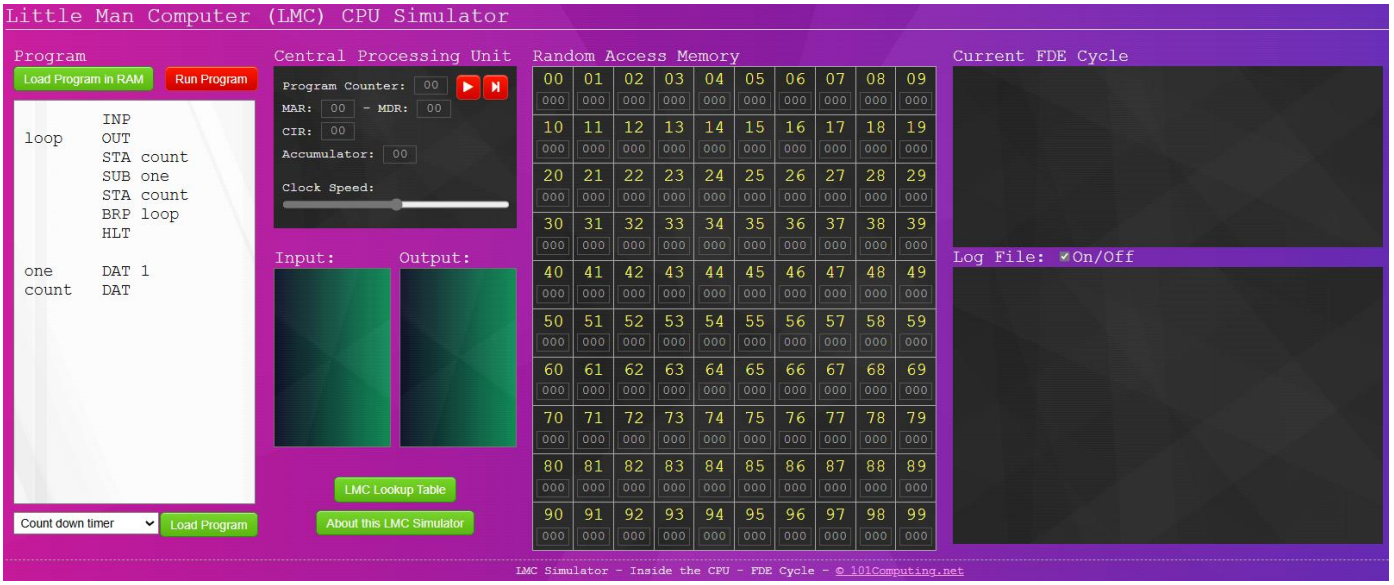


Figure 5.1—7 - screenshot of LMC #2

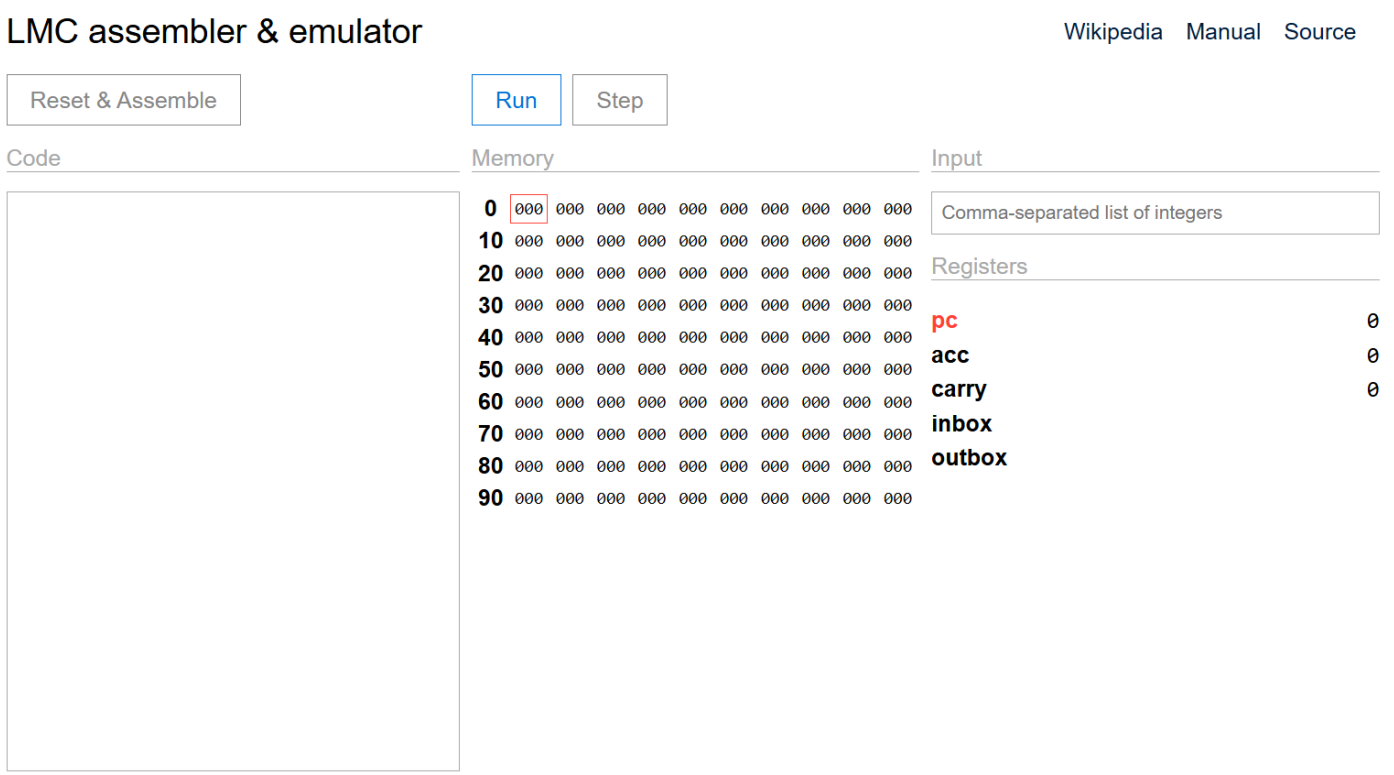


Figure 5.1—8 - screenshot of LMC #3

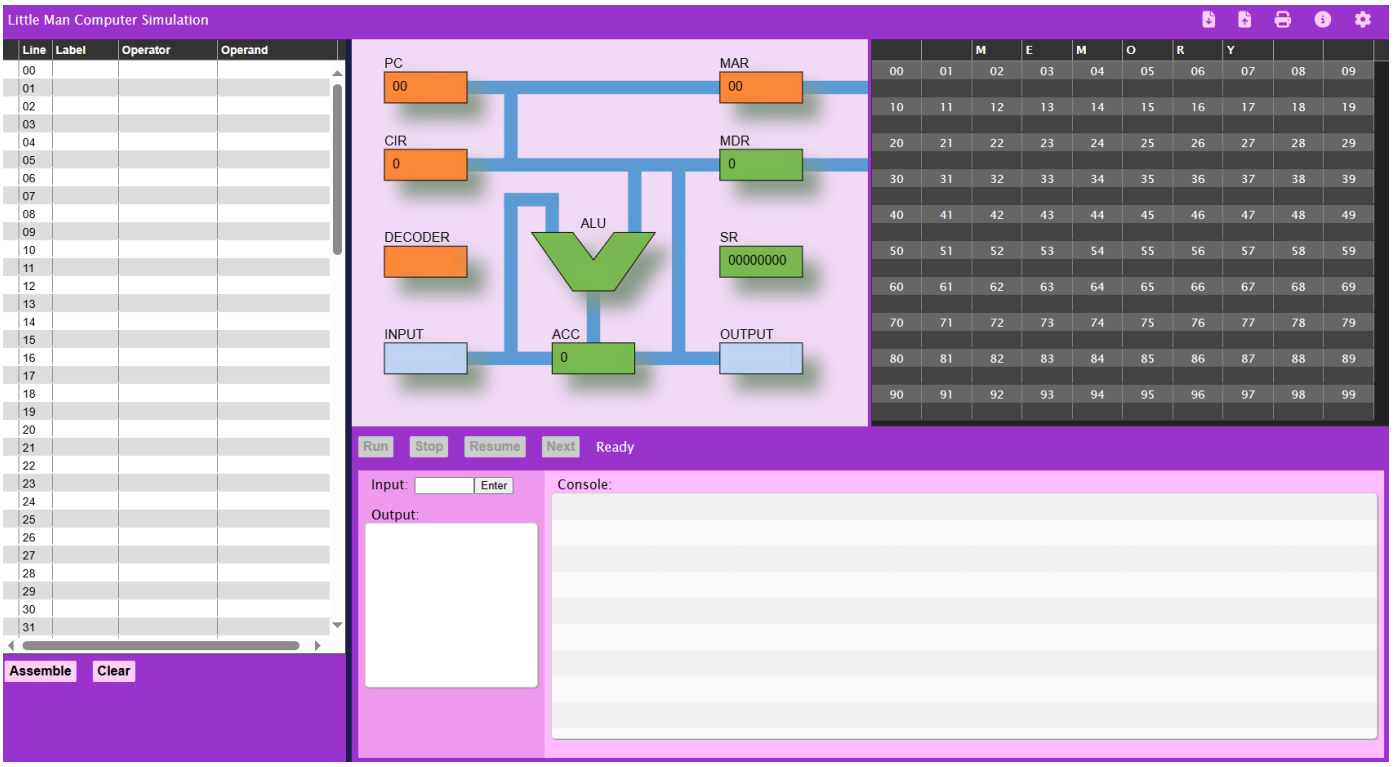


Figure 5.1—9 - screenshot of LMC #4

Figure 5.1—10 - screenshot of LMC #5

Figure 5.1—11 - screenshot of LMC #6

5.1.4.2.1 Analysis

LMC #1, #2 and #4 have a very colourful theme/style while LMC #5 and LMC #6 are slightly so. One of the main benefits of coloured themes is that they make the UI far easier to use by making different kinds of components distinct from one another. Examples of these include the RAM/memory cell addresses and their corresponding, directly below, cell values having different colours in LMC #1, #2, and #6. This helps users who may have difficulty reading and differentiating between different parts of the simulator. It also allows the compiler UI to be more compact without sacrificing readability. Easy-to-read themes will be included in the author's LMC educational game but will have the bonus of allowing the user to choose from multiple themes to allow specialised themes to specifically cater to different groups of people with different visual disorders or just difficulties. More about this will be discussed later in detail, and how it satisfies Humanism in the implementation.

5.1.4.3 Weighting

LMC weighting calculations in **Table 5.1—5**:

	LMC #1	LMC #2	LMC #3	LMC #4	LMC #5	LMC #6
Weight based on manual/ introduction (0-5)	4 – Explains instructions and registers. LMC has multiple assembly example programs, but not in the manual.	5 – Explains instructions, registers, and multiple assembly example programs.	4 – Explains instructions, registers, and an example assembly program.	4 - Explains instructions, registers, and an example assembly program.	2 – Instead of a manual, it has a feature comparison table to other LMCs and has multiple example assembly programs.	0 – A blog that doesn't explain much, with the most informative being the LMC Wikipedia page, but has multiple example assembly programs.
Weight based on Source code availability (0 or 5)	5	0	5	5	5	0

Weight based on how many attributes are found in Table 5.1—4 (0-6)	6	6	6	6	4	5
Weight based on author/company's background, such as: other projects, experience, and relevancy (0-5)	5 – Author made 3 other CPU simulators ([1,2,3]) and assisted with a book at [4]	4 – The company makes computer science books and lessons for GCSE [5] and A-level [6] students. Due to this, LMC must be made with the students in mind.	1 – The only thing of note is that the developer's contribution is a "compiler for a Java subset" at [7]. Not much experience nor relevancy.	3 – Company is literally a private school for children 3-18 years old ([8]). So LMC must be made with the students in mind. However, it is unknown how reputable (especially in the Computer Science field) this private school is as it is difficult to do so.	0 – developer's only other GitHub repository [9] is a HTML page of a table of comparison between the developer's LMC and LMC #1 and #2 [10]. Nothing else can be found on the GitHub account.	2 – developer's blogs ([11]) explores mostly mathematical concepts using programming. The much mathematical experience is only slightly relevant as LMC contains a bit of mathematics but not enough to be considered.
Total	20	14	16	18	12	7
% (to 2 d.p.)	22.99	16.09	18.39	20.69	13.79	10.34

*Table 5.1—5 - LMC weighting calculations, links are revealed in **Table 5.1—6**.*

Note: '?' means that the subject LMC program does not have enough information to be worth mentioning.
Note: The term "label" refers to establishing a reference point for looping in the assembly program.

Notation	Link
[1]	(Higginson, 2016)
[2]	(Higginson, 2024a)
[3]	(Higginson, 2018)
[4]	(Pawson and Higginson, 2020)
[5]	(101 Computing, 2024b)
[6]	(101 Computing, 2024a)
[7]	(Firmwehr, 2022)
[8]	(Wellingborough School, 2025)
[9]	(trincot, 2025)
[10]	(trincot, 2020)
[11]	(Hankin, 2022)

*Table 5.1—6 - revealed links for **Table 5.1—5**.*

5.1.4.4 Determined overall best attributes

Overall attributes based on weights of the LMCs and unique characteristics from one LMC where it is blank for others (exclusive characteristics explained later) in the table below. Remember that the core part of the game features must be LMC standard, as long as that remains the same, extra features can be added for more depth.

	RAM size	Valid inputs	Instruction set as mnemonics	Base	Shown registers	Compiler
Overall (based on weighting)	100	integers - 999 to 999 and `NaN` if invalid value	HLT, ADD, SUB, STA, LDA, BRA, BRZ, BRP, INP, OUT, OCT, DAT	10 – decimal	PC, MIR, MAR, Accumulator	Due to relative sizing, compiler traits are instead in the "*Compiler traits:*" list!

*Table 5.1—7 - in relation to **Table 5.1—5**, showing the overall attributes based on the weighting.*

Compiler traits using the weighting:

- Detects syntax errors (pointing out the error location), explains its nature, and prevents compilation if so.
- 3 text boxes per line (for the parts: label, operator, and operand). The right-hand-side textbox can be selected by click or the tab key/button, and the left text box of the next line can be selected by clicking or the enter key/button.
- Uses double slashes (//) to initiate comments.
- The entire compiler is not case sensitive.
- Minimal UI.

The table shows the planned unique characteristics, against the weighting, with their reasoning:

Unique characteristic	Reason for going against the weighting
Not using the `NaN` value.	Users should learn how to detect errors instead of being spoon-fed to get them used to and potentially prepare them for logic errors.
OTC instruction	Barely changes the core functionality of LMC as it simulates CPU simulator games, as it only changes the output from integer to any ASCII character. However, it also opens a lot more imaginative programs in the sandbox and can be used for making a lot more levels, exponentially expanding students' creativity and practising critical thinking respectfully.
Comments are done by double slashes	used in popular languages, such as JS, and for more accessible than other widely used comment notations such as the hashtag (#) for comments in Python.
3 text boxes per line	Let the student understand the structure of assembly code quickly enough to not lose interest before understanding how to code in assembly code and get on with the level's content. This will be implemented only if the author can map the tab, enter, and arrow keys to navigate between the textboxes not to impede coding speed and familiarity with the usual coding IDEs.
Non-case sensitive and minimal UI	Helps students with small screen sizes, such as a small laptop or even a smartphone.
The use of a hashtag (#) to denote sections, rather than a singular line, of lines of assembly script.	For easier organisation in bigger assembly code programs, especially with the more difficult campaign levels.
The optional use of predefined inputs, like how LMC **#3** and LMC **#6** implemented it, rather than only inputting values when asked.	Allows the user to rapidly develop assembly code programs by quicker testing. This also greatly assists the author himself as well because it allows to use of automatic tests for quicker development of the LMC game and test reliability for any possible runtime or logic errors.

Table 5.1—8 - planned unique characteristics, against the weighting, with their reasoning.

5.1.4.5 Missing instruction set

One confusing point of interest was how all the LMC simulators leaves the instruction code "4" unoccupied – not assigned to any instruction. This means that it might have to do with LMC standard. As previously mentioned in the *LMC simulator* section, all the LMC simulators state their source of reference and further reading which all a link to the Wikipedia page describing the concept of LMC simulator (Wikipedia contributors, 2025). The Wikipedia page references 3 academic papers (Yurcik and Brumbaugh, 2001; Yurcik and Osborne, 2001; Osborne and Yurcik, 2002) and one archived webpage (Illinois State University, 2014) with limited credibility. Strangely, all the said references (that shows an instruction set) together have a different assembly instruction set to that of Wikipedia.

Numeric code	**Mnemonic code**	**Instruction**	**Description**
1xx	ADD	ADD	Add the value stored in mailbox xx to whatever value is currently on the accumulator (calculator). Note: the contents of the mailbox are not changed, and the actions of the accumulator (calculator) are not defined for add instructions that cause sums larger than 3 digits. Similarly to SUBTRACT, one could set the negative flag on overflow.

2xx	SUB	SUBTRACT	Subtract the value stored in mailbox xx from whatever value is currently on the accumulator (calculator). Note: the contents of the mailbox are not changed, and the actions of the accumulator are not defined for subtract instructions that cause negative results - however, a negative flag will be set so that **7xx (BRZ)** and **8xx (BRP)** can be used properly.
3xx	STA	STORE	Store the contents of the accumulator in mailbox xx (destructive). Note: the contents of the accumulator (calculator) are not changed (non-destructive), but contents of mailbox are replaced regardless of what was in there (destructive)
5xx	LDA	LOAD	Load the value from mailbox xx (non-destructive) and enter it in the accumulator (destructive).
6xx	BRA	BRANCH ALWAYS (unconditional)	Set the program counter to the given address (value xx). That is, the value in mailbox xx will be the next instruction executed.
7xx	BRZ	BRANCH IF ZERO (conditional)	If the accumulator (calculator) contains the value 000, set the program counter to the value xx. Otherwise, do nothing. Whether the negative flag is taken into account is undefined. When a SUBTRACT underflows the accumulator, this flag is set, after which the accumulator is undefined, potentially zero, causing behaviour of BRZ to be undefined on underflow. Suggested behaviour would be to branch if accumulator is zero and negative flag is not set. Note: since the program is stored in memory, data and program instructions all have the same address/location format.
8xx	BRP	BRANCH IF POSITIVE (conditional)	If the accumulator (calculator) is 0 or positive, set the program counter to the value xx. Otherwise, do nothing. As LMC memory cells can only hold values between 0 and 999, this instruction depends solely on the negative flag set by an underflow on SUBTRACT and potentially on an overflow on ADD (undefined). Note: since the program is stored in memory, data and program instructions all have the same address/location format.
901	INP	INPUT	Go to the INBOX, fetch the value from the user, and put it in the accumulator (calculator) Note: this will overwrite whatever value was in the accumulator (destructive)
902	OUT	OUTPUT	Copy the value from the accumulator (calculator) to the OUTBOX. Note: the contents of the accumulator are not changed (non-destructive).
000	HLT/COB	HALT/COFFEE BREAK	Stop working/end the program.
	DAT	DATA	

Table 5.1—9 - showing the instruction set from Wikipedia (Wikipedia contributors, 2025) that every LMC# base their instruction sets on.

Table 5.1—9 will be called Assembly Instruction Set #1.

Opcode	Description	Mnemonic
1	LOAD contents of mailbox address into calculator	LDA XX
2	STORE contents of calculator into mailbox address	STA XX
3	ADD contents of mailbox address to calculator	ADD XX
4	SUBtract mailbox address contents from calculator	SUB XX
500	INPUT value from inbox into calculator	IN
600	OUTPUT value from calculator into outbox	OUT
700	HALT - LMC stops (coffee break)	HLT
800	**(SKIP)** SKN - skip next line if calculator value is negative	SKN
801	**(SKIP)** SKZ - skip next line if calculator value is zero	SKZ
802	**(SKIP)** SKP - skip next line if calculator is positive	SKP

9	JUMP - go to address	JMP XX

Table 5.1—10 - showing the consistent instruction set from Wikipedia's academic paper references (Osborne and Yurcik, 2002; Yurcik and Brumbaugh, 2001):

Table 5.1—10 will be called Assembly Instruction Set #2.

Notes:

- The archived webpage does have differently worded descriptions but still have the exact same instruction names, mnemonics and opcode.
- The articles that contain **Table 5.1—10**, also mention and use the "*DAT*" instruction but does not include it in the table.

The academic paper's use of terms is contextualised (by the analogy) that includes, but not limited to, "*calculator*", "*inbox*" and "*outbox*" (Osborne and Yurcik, 2002; Yurcik and Brumbaugh, 2001) which refers to the *accumulator*, *input*, and *output* respectively. The academic papers do mention this explicitly (Osborne and Yurcik, 2002; Yurcik and Brumbaugh, 2001). There are many similarities and differences between the 2 assembly instruction sets.

5.1.4.5.1 Similarities

- Both uses the little man in a mailroom analogy
- Both sets have the same load, store, add, subtract, input, output and halt instruction by naming, mnemonic, and execution nature.
- Both sets have the execution nature for unconditional branching/skipping (*BRA* and *JMP*).
- Both sets have the same idea of conditional branching/skipping if accumulator's value is positive or zero (*BRP* and *BRZ* and *SKP* and *SKZ*).

5.1.4.5.2 Differences

- None of the operands of the instruction's nature match between the 2 sets.
- Same nature but different name and mnemonic for unconditional branching/skipping regardless of accumulator value - *JMP*, called *JUMP*, instruction in Assembly Instruction Set #1 and *BRA*, called *BRANCH ALWAYS*, instruction Assembly Instruction Set #2.
- The conditional branching/skipping instructions of both sets have the same idea but different name, mnemonic and execution nature. This is especially when one set takes addresses to skip/branch to and the other set does not (except *JMP/BRA*).
- Input and output instructions are both in the same instruction code (but different address to differentiate) for Assembly Instruction Set #1 and are not for Assembly Instruction Set #2.
- Conditional branching/skipping instructions are all in the same instruction code (but different address to differentiate) for Assembly Instruction Set #2 and are not for Assembly Instruction Set #1.

5.1.4.5.3 Why are they 2 versions?

It is difficult to say. All found academic papers all uses Assembly Instruction Set #2 but all found online LMC simulators (that currently can be accessed) uses Assembly Instruction Set #1. There is no source in the Wikipedia page that is referenced for the exact set of Assembly Instruction Set #1. The most feasible explanation the author can make is the LMC standard evolved over time. This is because the original academic papers were published in 2001 (Yurcik and Brumbaugh, 2001) and 2002 (Osborne and Yurcik, 2002). The only source that explicitly states this is from LMC# 5's source code repository, stating "*Some (older) variants use SKZ, SKP and JMP instead of the BR* instructions*", however this source is not reliable so more reliable source needs to be found.

The author decided to go outside the original academic papers, indirectly referenced by the LMC simulator. The author does this by looking for other academic papers that references the original ones, from Wikipedia references, but more recent and contains their interpretation of the LMC assembly instruction set. It is highly preferable for modern academic papers to reference at least one of the original papers but none that can be found that shows an instruction set, let alone there not being many papers that mainly talk about LMC as the main subject. The closest modern academic paper, that author got, was about simulating LMC in Scratch (Javed and Zeeshan, 2022). Refer to a previous part in this dissertation for an explanation about Scratch.

Code	Name	Description
0	HLT	Stop execution
1xx	ADD	Add the contents of the memory address to the Accumulator
2xx	SUB	Subtract the contents of the memory address from the Accumulator
3xx	STA or STO	Store the value in the Accumulator in the memory address given.
4		Not in use
5xx	LDA	Load the Accumulator with the contents of the memory address given
6xx	BRA	Branch - use the address given as the address of the next instruction
7xx	BRZ	Branch to the address given if the Accumulator is zero
8xx	BRP	Branch to the address given if the Accumulator is zero or positive
9xx	INP or OUT	Input or Output. Take from Input if address is 1, copy to Output if address is 2.
	DAT	Used to indicate a location that contains data.

Table 1. LMC - Instruction Set

Figure 5.1—12 - shows the presented instruction set by Javed and Zeeshan (2022).

Javed and Zeeshan (2022) does reference LMC #1 for a basis of understanding of LMC and mention LMC's origin from Dr Stuart Madnick in 1965. After that is where the instruction set is displayed without much explanation – simply labelled as "*LMC – Instruction Set*". Two facts are gathered from here. The first solidifies the author's previous point of LMC's relevancy and popularity in the current years. The seconds shows that the exact LMC assembly instruction set does not matter that much if it still conveys the idea of a *little man in a mailroom* analogy with which Javed and Zeeshan (2022) also mentioned the analogy.

5.1.4.5.4 Substitution for the missing instruction

With the confusion cleared, we go back to the problem of a missing instruction set. LMC standard is not as enforced as once thought as long the analogy still holds. The author has ensured this by ensuring that the core part of the LMC game's assembly instruction set is still based on the modern LMC assembly instruction set, ensuring similarity with other LMC simulators. However, the author also mentioned the idea of adding extra features to the LMC for learning extra concepts.

The author's first idea was to add an instruction that branched (changes Program Counter's value) if the accumulator's value was negative. This idea was quickly scraped because having all branching instruction codes not adjacent to each other will make the instruction set unnecessarily complicated – doing more harm than good. It also does not teach any extra concepts to the student because the other branch instruction codes already do that. Besides, 'branching if negative' can very easily be achieved by adding 999 to the accumulator and then doing a 'branch if positive' which achieves the same goal and serves as learning concept as an early campaign level (doing a complex operation by doing smaller operations).

The second idea was a bitwise digit. This simply shifts the digits of significance, of the accumulator, to the left by one digit or to the right (depending which way the user wants to shift). This is not as easily replicated like the first choice but teaches the student about shifters - it represents the bit-wise shifters in computers. This is a far better concept to learn, especially because the RAM values are displayed in decimal rather than binary where students may forget the play of binary in processor architecture – this learning concept prevents it. The digit shifters, however, will need to be shifted by base-10 digits instead of binary due to the LMC's nature of storing values in range -999 to 999 instead of numbers that are the powers of 2. Because the author has to make sure to dedicate a small part of the LMC game to show to user how the digit shifters represent and relates to the bit-wise shifting operations in everyday computers.

For clarity, the Instruction set, as mnemonics, are now: *HLT, ADD, SUB, STA, LDA, BRA, BRZ, BRP, INP, OUT, OCT, DAT, RSH* and *LSH* where *LSH* and *RSH* is the mnemonic for *left shift* and *right shift* respectively. The reason for using two instructions is the same to why there is both *ADD* and *SUB* instructions instead of just one of them. Also, because the shifting instructions only affects the accumulator value and does not need an address, the author will implement the instructions similarly to how *INP* and *OUT* are – the instruction (first) digit '*4*' will be for shifting and the address dictates weather it is to shift left or right – "*401*" for left shifting and "*402*" for right shifting.

5.1.4.6 Assembly coding (not LMCs) games analysis

While there are not any games involving LMC standard simulators there are some that simulate assembly code in some architecture processor through various ways. **Table 5.1—11** shows all games, that can be found from the Google search engine at the time, where the player codes in some form of assembly code to play the game.

5.1.4.6.1 Games' characteristics

Other assembly code processor game (not LMC-compliant) in the **Table 5.1—11** below:

#	link	name	free?	is architecture von Neumann	Analysis
1	[1]	TIS-100	no	No because it, apart from registers, does not appear to have dedicated memory connected to each of the processors.	Focuses more on utilising parallel computing as it appears to utilise up to 12 processors where each one can be unlocked via campaign of its "*more than 45" levels*". The game also has "*3 sandbox [modes]*" as well as having different types of IO ranging from printing simple numbers and a numpad to a pixel screen. Does use mnemonics like LMC and has an accumulator register in each processor but that is about all the similarities to LMC.
2	[2]	Human Resource Machine	no	No as it describes the publisher describes the simulator as a "*Harvard Architecture with a single accumulator*". This is proven by seeing how the data set and instruction set are displayed in two different places and dealt/retrieved differently.	Appears to be single processor represented by a little person doing tasks as instructed by the written assembly code. This little person has access to inputs and outputs presented as inbox and outbox respectfully and deals with accumulator by "*hold[ing] exactly one box in his or her hands at a time" to be "like an accumulator*". While it does not follow LMC syntax structure and architecture, but it does it does follow the LMC analogy of little person in a closed-off room handling inboxes and outboxes.
3	[3]	SHENZHEN I/O	no	No, it is just like game #1 but it is possible to connect certain hardware together to create a von Neumann architecture.	Barely an assembly-code based game. The assembly programming is only a small part of the game, rather than being the primary focus, as it focuses more on the hardware side of things. As said, it can follow the von Neumann architecture that the LMC uses if the player purposely configures the hardware to do so but that is about it for LMC similarities.
4	[4]	EXAPUNKS	no	Difficult to tell, as it focuses on using assembly code to hack rather than making stand-alone programs and not having much pre-view, so it is assumed not.	Unlike the other analysed games, this game focuses on hacking rather than making stand-alone programs. Cannot really find any similarities between its simulation and LMC.

Table 5.1—11 - Other assembly code processor games (not LMC-compliant).

Notation	Link
[1]	(Zachtronics, 2015)
[2]	(Tomorrow Corporation, 2015)
[3]	(Zachtronics, 2016)
[4]	(Zachtronics, 2020)

*Table 5.1—12 - corresponding links to notations in **Table 5.1—11**.*

Notes for **Table 5.1—11**:

- 3 of the 4 found assembly code games (#1, #3, and #4) are all by the same developer company (called Zachtronics)
- All found games are not free (because no free one can be found), therefore, the limited reviewing and analysing these games can only be somewhat accurately done by looking at the previews (screenshots, videos, descriptions, tags, et cetera) and customer reviews.

5.1.4.6.2 Useful characteristics

From analysing these games, a list has been compiled listing all the game features that the author already planned or now considered planning to put them in the product:

- The sheer number of levels from game #1 – more levels equate to more learning/lesson content.
- Use the LMC analogy for the simulation from game #2 – satisfies the theory.
- Use of simulating hardware from game #3 – satisfies the theory and enhances learning the hardware side of assembly programming in LMC.

5.1.5 Methodology

Choosing a development methodology is critical to the project's life cycle. It provides a structured framework that guides how one plans, develop, and deliver the work/product. By using a development methodology to break the project into manageable tasks, allowing to set clear goals and track your progress over time within time constraints. With a properly defined approach, challenges

be easily identified early for which one can make informed decisions to adapt overcome them without losing momentum. Choosing a suitable methodology will greatly reduce risk of getting overwhelmed and ensure each step is effective in making the final product meets timeline expectations.

5.1.5.1 Which one and why

5.1.5.2 Agile development diagram

The development methodology shares traits with MoSCoW diagram (in what features are there and their priority).

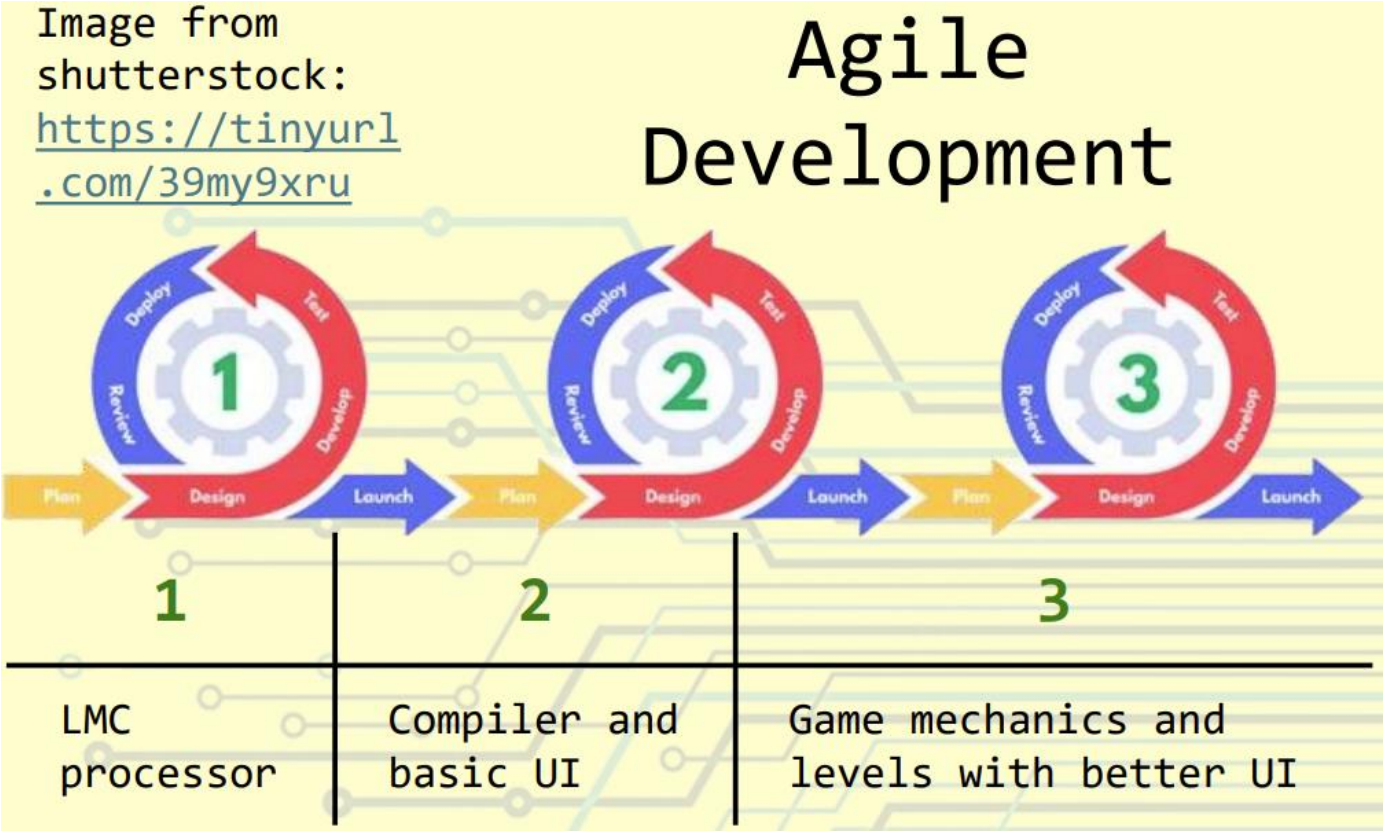


Figure 5.1—13 - screenshot snippet, showing the simplified Agile development methodology, from the project presentation poster.

Figure 5.1—13 shows the overall process of developing LMC educational games – not the more in-depth version. Explaining this figure further:

1. LMC processor – Focuses on the simulator itself working (without UI) – able to fetch, decode and execute compiled assembly code. Automatic testing (Jest) is used to make sure that the LMC will both not encounter any logic or runtime error, within expected parameters, and compatible with other LMC simulator to keep up to LMC standard. Errors relating to invalid assembly syntax are not covered in the sprint.

2. Compiler and basic UI – UI of one basic style in implement where user/developer can fully interact and try out without depending on heuristic testing. UI links to both the LMC simulator and its compiler. Compiler input is made of 3 textboxes per line (unless a comment or region declaration), but user will still be able to navigate thought those textboxes as if it was one whole textbox. Character restrictions and parsing (such as removing whitespaces) are done to reduce the number of error/warnings confrontations with the user. Compiler will display errors or warnings if the assembly code cannot be compiled – invalid syntax, using labels before declarations, et cetera. Compiler must compile the labels, opcode, and operand to numbers for the LMC simulator to understand and decode the assembly code as well as ignoring comments, region declarations and blank lines.

3. Game mechanics and levels with better UI – Expected to take at least the same amount of time and effort as the first two sprints combined. In short, this is the part where the LMC gets gamified. Game mechanics are added to: load objectives (depending on the chosen level), check if user's compiled assembly code meets what the objective said. A three-star rating system will be implemented to show how well the user done per level. The main point of the three-star rating system is to take computation stress of the compiled assembly code checker – it is important that it does not computationally heavy (relative to the LMC game as a light program) which is easy to accidently do if not careful.

The reasoning for the sprint diagram not being in-depth is because it only covers the "*must have*" priority of producing the online desktop application. In the PERT chart, it shows that 3 out platforms (offline desktop, online mobile, and offline mobile) are also expected but not shown in the diagram due to their lower priorities as seen in the **Priorities** section.

5.2 Development and implementation

5.2.1 Planning

In accordance with the sprint development methodology, the plan (including critical path analysis, diagrams, et cetera) should be done between each sprint as each sprint require their own plan and time organisation. However, an overall plan is also needed alongside the sprint-specific planning. This provides a balanced approach to project management. The bigger-picture plan outlines the overall goals, dependencies/requirements, and timelines, ensuring that all sprints are aligned with the same strategic vision of the project without deviation. Meanwhile, individual sprint planning breaks down the overall goals into manageable and actionable tasks that the author can focus on within the sprint's short timeframe without much time to reconsiderations. This dual-layer planning enhances clarity, resource allocation, and progress tracking, ensuring that immediate tasks contribute directly to long-term success with the LMC simulator educational game.

For the code of the project, the author plans to do so in OOP with code having traits modularity, reusability, loose coupling, high cohesion, easy-to-understand, configurable, et cetera. More details regarding the planning can be seen in the "*Planning*" section of each sprint as seen in appendixes L, M, and N.

5.2.1.1 System requirements

The LMC game will not be performance intensive due to the LMC simulator being simplistic in nature. The requirements for offline or online and mobile or desktop should be near identical. For those reasons, it is safe to assume that any computer that can run a somewhat modern browser capable of running HTML 5, CSS, and JS ES2016 is capable enough. Because nearly all the widely used browsers (*Google Chrome*, *Opera*, *Opera GX*, *Brave*, *Microsoft Edge*, *Epic Privacy Browser*, et cetera) all depend on the open-sourced Chromium rendering engine (also known as *Blink* for Chromium (The Chromium Project, 2023)) for interpretation, execution and rendering of HTML, CSS, and JS (compiled from TS).

5.2.1.1.1 Hardware

Due to *Chromium* being very lightweight, its minimum requirements are not a concern for both developers and users – hence not fully listed. The most reliable and descriptive requirements, that can be found, are taken from the *Google's Chrome Enterprise* and *Education Help centre* (Chrome Enterprise and Education, 2016). Minimum hardware requirements for Windows and Linux OSs are a CPU equivalent in power to an Intel Pentium 4 processor that is "SSE3 capable". It did not show the processor requirements for Mac and Android, which the author believes that it is due to that they use exclusive processor types and models, so we assume that it is of equivalence to that of Windows and Linux.

It is not said what the minimum RAM requirements are, probably because nearly all the require RAM will be based on what webpages/tabs are open and how many are currently open. Knowing the required RAM, that the LMC product needs, is near impossible to know until the author finish the product itself as it depends on exactly how much code is used, how many assets, how many CSS styles (dark mode, colourblind, et cetera), quality and quantity of image assets, will sound files be included, et cetera. This also applies the same for the required disk size for the offline versions.

5.2.1.1.2 Software

As mentioned, only a website engine such as *Blink* or an equivalent like *Firefox's* own rendering engine (called *Gecko*) is needed alongside a browser if using an online version of the LMC game.

More about online, the help centre page (about *Chromium* requirements) also states the minimum version of each operating system for *Chromium's* current version, summarised in the following table:

Operating system	**Minimum version**
Windows	Windows 10 or Windows Server 2016
MacOS	macOS Big Sur 11
Linux	64-bit Ubuntu 18.04, Debian 10, openSUSE 15.5, or Fedora Linux 39
Android	Android 8.0 Oreo

Table 5.2—1 - LMC educational game requirements, by OSs, based on Chromium's requirements.

The author would like to emphasise the fact that the OS requirements are only for modern versions of chromium and the project will use the ES2016 JS (default version that TS compiles to) which is, as of this writing, just below 9 years old. This means that lower-end computers running the lower version OSs can still run the product if they still meet the hardware requirements.

5.2.1.2 Priorities

5.2.1.2.1 Product platform priorities

As seen in the PERT chart, the author plans to make 4 versions of the LMC game – one per platform. Those are desktop online, desktop offline, mobile online and mobile offline. The priorities of each platform can easily be visualised and explained as seen below.

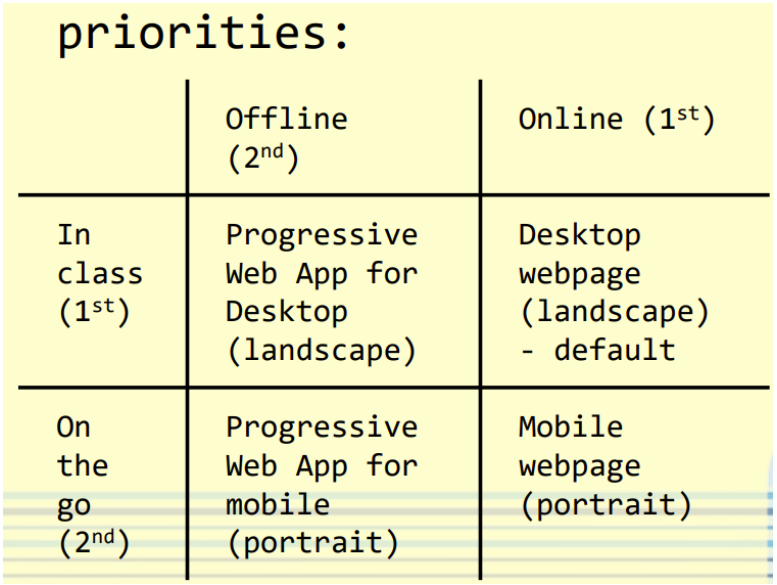


Figure 5.2—1 - screenshot snippet, showing the platform porting priorities, from the project presentation poster.

More information regarding product platform priories can be seen in the **Product platform priorities** of **Appendix J: Project Management**.

5.2.1.2.2 Overall priorities - MoSCoW

To present the different priorities, of this project overall, in a way that is easily absorbed by the reader, the author will present them using the well-known "MoSCoW" prioritisation technique/method. This method is demonstrated below for the project.

("Mo") Must Have	Fully functional LMC simulator; functional user interface; compiler detecting invalid code.
("S") Should Have	Tutorials; several campaign levels; user interface with graphics and assets that are comfortable to see to most people; compiler recognising why code is invalid. Product aside, automated tests should be included.
("Co") Could Have	Advance tools to help make coding and debugging easier; Different kinds of style to be comfortable to everyone and to people with disabilities; few dozens of campaign levels; simplistic save-file system for saving campaign progress and current code in sandbox mode.
("W") Would Have	More complex save-file system to load different programs in sandbox instead of current and save current code for each campaign level; user interface with graphics and assets that are very pleasing to the eye and attractive.

Table 5.2—2 - a rough draft of the overall priorities of the LMC educational game.

The author wants to note that this MoSCoW is a rough overview and not a specific analysis. For a more detailed version, see the sprint specific ones at the individual sprints of **Appendix L: Agile Development: Timebox 1**, **Appendix M: Agile Development: Timebox 2**, and **Appendix N: Agile Development: Timebox 3**.

5.2.1.3 Used technologies

5.2.1.3.1 Programming language

The author has put in deep thought for choosing the most suitable programming language to code the LMC educational game. As previously mentioned, in the section about platform versions with reasoning, the top priority is for the product to be accessible online rather than having to download it. For that reason, it needs to be a programming language that fully compatible with compiling to JS to be runnable in an internet browser. There are many languages that can do that (such as *Elm*, *Dart*, and *CoffeeScript*), but the more popular ones are TS, *Kotlin*, and *WebAssembly*. Popularity is considered here because the more widely used a programming language is, the more community support it tend to have. Bigger community support tends to mean that it includes, but not limited to, more code guides and tutorials, more native dependencies and frameworks, more compatible JS dependencies, more thought into the programming language creation itself, and more code examples to use in own code (will be noted, with source, if done so). Not to forget that JS is also an option, by its native interpretation of the browser.

The options are narrowed down again but this time by age. Besides the same reason of the community support (older programming languages tend have more community support), younger programming languages are usually far less mature. What

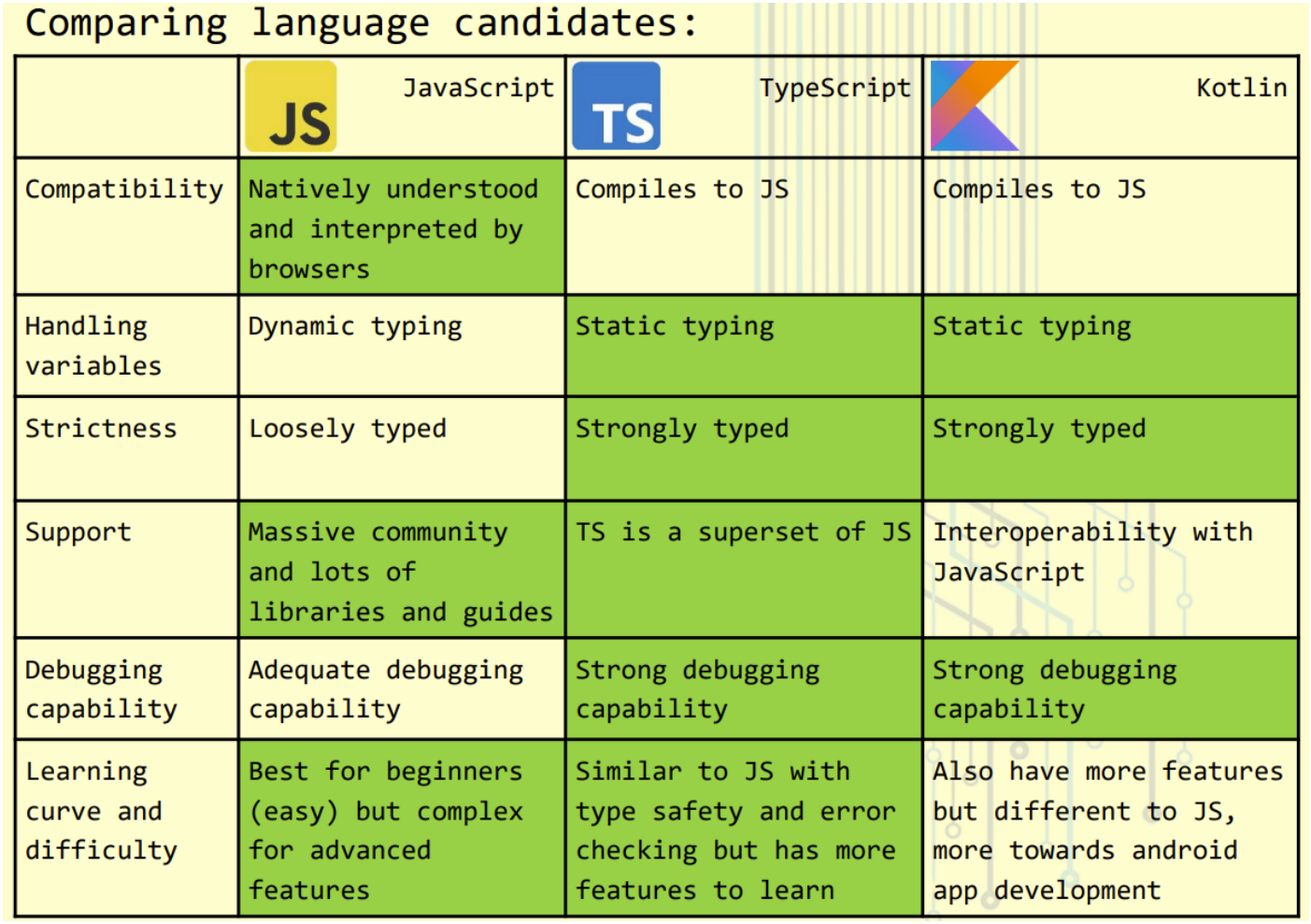
the author mean by that is that older programming languages will have the time to, do the feedback cycle of, collecting feedback from its users and then updating the programming language – refining and facilitating itself.

Programming language	Release date
TS	2012
Kotlin	2016
WebAssembly	2017
JS	1996

Table 5.2—3 - Shows the programming language candidates and their corresponding release dates.

Because *WebAssembly* is the youngest and the author has experience coding in only the other 3 languages (TS, JS, and Kotlin), *WebAssembly* is no longer a candidate.

Because there are only 3 languages left, the language of choice will be chosen by comparing the three languages' traits and characteristics. This comparison can be seen in the "*Comparing language candidates*" table from the poster presentation as seen below.



Comparing language candidates:

	JavaScript	TypeScript	Kotlin
Compatibility	Natively understood and interpreted by browsers	Compiles to JS	Compiles to JS
Handling variables	Dynamic typing	Static typing	Static typing
Strictness	Loosely typed	Strongly typed	Strongly typed
Support	Massive community and lots of libraries and guides	TS is a superset of JS	Interoperability with JavaScript
Debugging capability	Adequate debugging capability	Strong debugging capability	Strong debugging capability
Learning curve and difficulty	Best for beginners (easy) but complex for advanced features	Similar to JS with type safety and error checking but has more features to learn	Also have more features but different to JS, more towards android app development

Figure 5.2—2 - screenshot snippet of the project presentation poster showing the comparison for choosing a programming language (for the project) between JS, TS, and Kotlin.

From the comparison table, you can see the TS is the clear winner here by having more (and definitely not less significant) pros than either JS or Kotlin.

5.2.1.3.2 Dependencies

Because the product is to be a client run online web application, the only dependency to run is a compatible browser, but developing and testing the web application do require dependencies.

Because this more of a technical plan (practical rather than theory), the dependency part has been moved to Appendix L.

5.3 Developing

The development of each sprint can be seen in the "*Development*" section of appendix L, M, and N - 11.12.2.2, 11.13.2, and 11.14.1.2 respectfully.

6 Legal Considerations Chapter

6.1 Data Protection Law

The LMC educational game is fully client side and does not send any of the user's personal data online. The only scenario where personal data may be sent is of, they downloaded the offline version web application from a software distributor (such as *Steam* and *Itch.io*) where in case the responsibility of upholding to the Data Protection Law lies upon them.

6.2 Computer Misuse Law

Online or offline, the LMC educational game is a web application, contained by the internet browsers' (or *Microsoft HTML Application Host*) restrictions. The LMC educational game will not do unauthorised access, act as malware, or any other malicious and unlawful actions.

6.3 Use of intellectual property

6.3.1 LMC concept

The 1965 Little Man Computer concept (introduced by Dr. Stuart Madnick) is widely used as an educational tool and thus is not subject to any strict legal restrictions. This is mostly because that it is a conceptual model of a computer instead of an actual software hence not having any public licence like GNU.

6.3.2 Android mascot

The Android mascot is intellectual property of Google LLC and is hence subject to their legal requirements under the Creative Commons 3.0 Attribution License as explained on their webpage (Google Partner Marketing Hub, 2020). Fortunately, in that webpage they have the simplified the process of knowing what rules apply to what scenario with their table as seen below.

	Include the ™ symbol	Include an attribution statement	Include the Creative Commons attribution
Android name	Yes	No	No
Android logo(s)	Yes	Yes	No
Android icon	No	No	Yes

Table 6.3—1 - legal requirements for the use of the Android brand name, logo, and icon (Google Partner Marketing Hub, 2020).

As deduced from the table, the author only needs to include the Creative Commons attribution as the author is using the icon.

For the reader's clarity, the icon is a picture representing the brand only and the logo is the combination of the name and icon. This and the fact that the icon is allowed for modification is stated with the page's mention of the following:

> *The Creative Commons license only applies to the green Android robot pictured above. Do not include the Creative Commons attribution when using Android Pie, Oreo, Nougat, Marshmallow, Lollipop or KitKat robots, as these are trademarked and no modifications are allowed.*

This means that all the author needs to do is to make sure that the Creative Commons attribution is included which Google specifically mentioned that is done by including the following text in the work:

> *The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.*

Keep in mind that the modified Android mascot head SVG is sourced from a third party (Openmoji.org, 2020), however they have explicitly mentioned that they do not require any given attribution as their "SVG repo" are "Public Domain and Open source" (SVG Repo, 2023). To be on the safe side, the author will follow their suggested non-compulsory crediting by including the following HTML code in the work:

> *Vectors and icons by SVG Repo*

6.3.3 Sound effect

The sound effect taken from *Pixabay* is subject to their "Content License Summary" (SVG Repo, 2023) to which it says that, for content, the author can:

> *Use Content without having to attribute the author*

This means that the sound effect is free to take as long as it does not "*contains any recognisable trademarks, logos or brands*" or used for "commercial purposes" (SVG Repo, 2023).

7 Ethical Considerations Chapter

There are no concerns the volunteer participation using the LMC educational game because it will be accessed as an online website, meaning no downloading necessary. The program does not take nor store any of the user's data as the program only uses the URL to store the game's configuration and progression. Said data can be seen in screenshots of sprint #3's **Current state of code**.

Identities of the volunteers are mentioned because of their explicit consent.

7.1 Human Participants

The author has requested the help of volunteers to help with test the LMC product.

The first is a teacher, called Mr Jacob, from the nearby Canterbury Academy school. He is a computer science teacher who has taught the concepts of LMC to students many times before. This makes him the ideal volunteer as he can organize his class's numerous students to also test the LMC simulator game – giving feedback from the 2 main audiences (teachers and students). By making Mr Jacob to organise the students' testing, as a middleman to the third party, is also highly beneficial as all the legal and ethical responsibilities are handled by Mr Jacob.

The second volunteer participant is the author's supervisor, called Tina Eager, of this dissertation/project. Not only she is also a teacher but also teaches LMC to the first-year university (level 4 in higher education) students in computer science hardware-based modules. While Mr Jacob teaches students from middle school (Key Stage 3) to A-Levels (Key Stage 5), she teaches university students. This means that those have the expertise, knowledge, and connection to the majority of the age range of the target audience of 13-18 years old.

There is also a third participant is a lecturer, called Mr Warner, from the EKC Group Canterbury College. He was suggested (and connected to) to the author as a tester by the author's supervisor and helped with organising at least 4 students as seen below. He beings the same advantages of effect feedback and middleman (for ethics) as Mr Jacob.

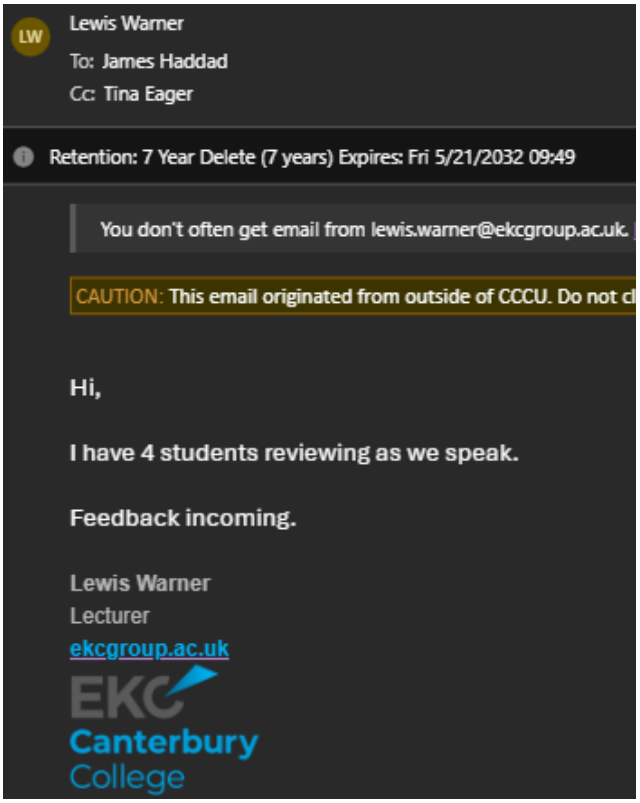


Figure 7.1—1 - email proof of Mr Warner acting as a middleman for at least 4 students reviewing the LMC educational game.

With those three highly valuable, the feedback from them will be top notch and any suggestions, not matter how minor, will be taken seriously and changes implemented.

Feedback from the participants can be seen in the **feedback** of **Appendix N: Agile Development: Timebox 3**.

8 Conclusion Chapter

8.1 Development and implementation conclusion

Due to the authors use of the Agile methodology, the conclusion of the development and implementation of the LMC educational game product is split up into sprint specific in the "review" section of **Appendix L: Agile Development: Timebox 1**, **Appendix M: Agile Development: Timebox 2**, **Appendix N: Agile Development: Timebox 3** (**11.12.2.4**, **11.13.3**, and **11.14.1.4** respectively).

8.2 Post-project conclusions

Post-project conclusion for after completing all 3 sprints can be found in the **Review** section of sprint #3.

To know what happen with feedback implementation, please see the **Review** part of the **feedback gathering and implementation (post-sprints)** section in Appendix N.

To see how the **Porting to offline application (post-sprints)** went, please see its **Review** section.

8.2.1 Overall conclusion

The author achieved what he wanted to do. He has done all that in the "should have" overall priority (**Table 5.2—2**) and even some of the "could have" and "would have" priorities, including (but not limited to): different styles catering to disabilities, 20 total levels, and ability to save and load current campaign progress (by the URL).

Its effectiveness has been proven by the volunteer participation (by both teachers and students), enforcing the achievement of the project's original goal.

Note that this is just the overall conclusion (what went well, improvement for next projects, et cetera), the author already mention this but what to reemphasize that the more detailed post-conclusion is in the **Review** section of sprint #3 (of **Appendix N: Agile Development: Timebox 3**) and other sections mentioned in the **Post-project conclusion**.

9 References

- Android SVG sourced from *Openmoji.org* from *SVG Repo* LLC (Openmoji.org, 2020).
- "*UI Button Heavy Button Press Metallic*" sound effect sourced from *Epic_Stock_Media* on *Pixabay* (Epic_Stock_Media, 2017).
- "*Simple Gantt Chart*" template from *Vertex42* (Vertex42, 2021).

10 Bibliography

Below lists all article referenced in Havard academic style (automatically updated by RefWorks Citation Manager):

101 Computing (2024a) *OCR – H446 – Computer Science A-Level.* Available at: https://www.101computing.net/ocr-h446-computer-science-a-level/ (Accessed: May 30, 2025).

101 Computing (2024b) *OCR – J277 – Computer Science GCSE.* Available at: https://www.101computing.net/ocr-j277-computer-science-gcse/ (Accessed: May 30, 2025).

101 Computing (2019a) *Little Man Computer (LMC) CPU Simulator.* Available at: www.101computing.net/LMC/ (Accessed: May 30, 2025).

101 Computing (2019b) *LMC Simulator.* Available at: https://www.101computing.net/lmc-simulator/ (Accessed: May 30, 2025).

Abramov, &., Aaron, Bekkhus, S., Hanlon, R.I., openjs-operations and cpojer (2023) *jest.* Available at: https://www.npmjs.com/package/jest (Accessed: May 30, 2025).

Aladdin, A.M., Bakir, Y.N. and Saeed, S.I. (2018) 'The Effects to Trend the Suitable OS Platform', *JOURNAL OF ADVANCES IN NATURAL SCIENCES,* 5(1), pp. 342–351 Available at: 10.24297/jns.v5i1.7528.

Arikpo, I.I., Ogban, F.U. and Eteng, I.E. (2007) 'Von Neumann architecture and modern computers', Available at: 10.4314/gjmas.v6i2.21415.

Betts, P., Quintana, &., Michelle, Lee, M. and Hess, C. (2020) *electron-prebuilt-compile.* Available at: https://www.npmjs.com/package/electron-prebuilt-compile (Accessed: May 30, 2025).

Bianchi, &. and Fabrizio (2015) *Coolors - The super fast color palettes generator!* Available at: https://coolors.co/generate (Accessed: May 30, 2025).

Bootstrap (2018) *Bootstrap V3.4.1.* Available at: https://getbootstrap.com/docs/3.4/ (Accessed: May 30, 2025).

Brieger, E., Arghode, V. and McLean, G. (2020) 'Connecting theory and practice: reviewing six learning theories to inform online instruction', Available at: 10.1108/EJTD-07-2019-0116.

Brinkmeier, P. (2017a) *LMC assembler & emulator.* Available at: https://pbrinkmeier.github.io/lmc-emulator/ (Accessed: May 30, 2025).

Brinkmeier, P. (2017b) *LMC emulator.* Available at: https://github.com/pbrinkmeier/lmc-emulator/blob/master/README.md (Accessed: May 30, 2025).

Brinkmeier, P. (2017c) *lmc-emulator.* Available at: https://github.com/pbrinkmeier/lmc-emulator/ (Accessed: May 30, 2025).

Burhanuddin, N.A.N., Ahmad, N.A., Said, R.R. and Asimiran, S. (2021) 'Learning Theories: Views from Behaviourism Theory and Constructivism Theory', *HRMARS,* Available at: 10.6007/IJARPED/v10-i1/8590.

Bussa, M., Millett, S. and Blankenship, J. (2011) 'Sprint 0: Generating the Product Backlog', *Pro Agile . NET Development with SCRUM,* , pp. 53–86 Available at: 10.1007/978-1-4302-3534-7_4.

C. Girard, J. Ecalle and A. Magnan (2012) 'Serious games as new educational tools: how effective are they? A meta-analysis of recent studies', Available at: 10.1111/j.1365-2729.2012.00489.x.

Caldwell, B., Cooper, M., Guarino Reid, L., Vanderheiden, G., Chisholm, W., Slatin, J. and White, J. (2008) *Web Content Accessibility Guidelines (WCAG) 2.0.* Available at: https://www.w3.org/TR/WCAG20/ (Accessed: May 30, 2025).

Chrome Enterprise and Education (2016) *Chrome browser system requirements.* Available at: https://support.google.com/chrome/a/answer/7100626 (Accessed: May 30, 2025).

Coolmath *Coomath Games.* Available at: https://www.coolmathgames.com/ (Accessed: May 30, 2025).

Correia da Silva, T. (2021) 'Mascots Vs Celebrities : Attitudes Towards the Brand', .

Dweck, C.S. (2016) *The New Psychology of Success.* Available at: https://adrvantage.com/wp-content/uploads/2023/02/Mindset-The-New-Psychology-of-Success-Dweck.pdf (Accessed: May 30, 2025).

Dyer, J.N. (2010) 'Desk-Top Software Development Using HTML Applications', *Journal of Business, Industry and Economics,* 15.

Education Host (2025) *Client Area.* Available at: https://clientarea.educationhost.co.uk (Accessed: May 30, 2025).

Eigenmann, R. and Lilja, J.D. (1998) *Von Neumann Computers.* Available at: https://public.callutheran.edu/~reinhart/CSC521/Week3/VonNeumann.pdf (Accessed: May 30, 2025).

Electron (2025) *Why Electron.* Available at: https://www.electronjs.org/docs/latest/why-electron (Accessed: May 30, 2025).

Electron Forge (2025) *Handling startup events.* Available at: https://www.electronforge.io/config/makers/squirrel.windows#handling-startup-events (Accessed: May 30, 2025).

Electron HQ and Electron Nightly (2025) *electron.* Available at: https://www.npmjs.com/package/electron (Accessed: May 30, 2025).

Embrey, B. and Bradley, A. (2024) *ts-node.* Available at: https://www.npmjs.com/package/ts-node (Accessed: May 30, 2025).

Epic_Stock_Media (2017) *UI Button Heavy Button Press Metallic.* Available at: https://pixabay.com/sound-effects/ui-button-heavy-button-press-metallic-333826/ (Accessed: May 30, 2025).

Firmwehr (2022) *gentle.* Available at: https://github.com/Firmwehr/gentle (Accessed: May 30, 2025).

Gamble, M. (2024) *LMC.* Available at: https://github.com/Wellingborough/LMC (Accessed: May 30, 2025).

Gamble, M. (2022a) *Little Man Computer Simulation.* Available at: https://wellingborough.github.io/LMC/LMC0.3.html (Accessed: May 30, 2025).

Gamble, M. (2022b) *Wellingborough School - Little Man Computer Simulator.* Available at: https://wellingborough.github.io/LMC/ (Accessed: May 30, 2025).

Google Partner Marketing Hub (2020) *Legal requirements.* Available at: https://partnermarketinghub.withgoogle.com/brands/android/legal-and-trademarks/legal-requirements/#requirements (Accessed: May 30, 2025).

Google Search Central (2025) *Define a favicon to show in search results.* Available at: https://developers.google.com/search/docs/appearance/favicon-in-search (Accessed: May 30, 2025).

GOV.UK (2024) *'Subjects taught' for Art & Design, Biology, Business Studies, Careers and Key Skills, Chemistry and 29 other filters in England for 2023/24.* Available at: https://explore-education-statistics.service.gov.uk/data-tables/fast-track/ac2977f7-92fa-47f8-b0e8-08dc6f35f09f (Accessed: May 30, 2025).

Haddad, J.R. (2025a) *Little Man Computer Educational Game Gantt Chart.* Available at: https://cccu-my.sharepoint.com/:x:/g/personal/jh1662_canterbury_ac_uk/EZXMZy_ybpdGsnDqv5LyYg8BjlmibTE86VsBHVUxzkS_-Q?e=YueD2y (Accessed: May 30, 2025).

Haddad, J.R. (2025b) *LMC Sim Game.* Available at: https://jh1662.github.io/LMC_simulator_game/ (Accessed: May 30, 2025).

Haddad, J.R. (2025c) *LMC_simulator_game.* Available at: https://github.com/jh1662/LMC_simulator_game (Accessed: May 30, 2025).

Hankin, P. (2022) *Paul's Blog.* Available at: https://blog.paulhankin.net (Accessed: May 30, 2025).

Hankin, P. (2016a) *Little Man Computer.* Available at: https://blog.paulhankin.net/littlemancomputer/ (Accessed: May 30, 2025).

Hankin, P. (2016b) *Little Man Computer Emulator.* Available at: https://blog.paulhankin.net/lmc/lmc.html (Accessed: May 30, 2025).

Hankin, P. (2016c) *Shift left.* Available at: https://paulhankin.github.io/lmc/lmc.html?program=ICAgICBJTlAKICAgICBTVEEgUjAKICAgICBJTlAKICAgICBTVEEgUjEKTE9PUCBMREEgUjEKICAgICBCUlogRU5ECiAgICAgU1VCIE9ORQogICAgIFNUQSBSMQogICAgIExEQSBSMAogICAgIEFERCBSMAogICAgIFNUQSBSMAogICAgIEJSQSBMT09QCkVORCAgTERBIFIwCiAgICAgT1VUCgpSMSAgIERBVApSMCAgIERBVApPTkUgIERBVCAx&input=MTQKMw== (Accessed: May 30, 2025).

Higginson, P.L. (2024a) *ARM Lite.* Available at: https://www.peterhigginson.co.uk/ARMlite/ (Accessed: May 30, 2025).

Higginson, P.L. (2024b) *LMC V1.5 help file.* Available at: https://www.peterhigginson.co.uk/lmc/help_new.html (Accessed: May 30, 2025).

Higginson, P.L. (2021) *peterhigginson / peterhigginson.co.uk / LMC /.* Available at: https://github.com/FrBrGeorge/peterhigginson/tree/main/peterhigginson.co.uk/LMC (Accessed: May 30, 2025).

Higginson, P.L. (2018) *AQA Assembly Language Simulator.* Available at: www.peterhigginson.co.uk/AQA/ (Accessed: May 30, 2025).

Higginson, P.L. (2016) *RISC Simulator in JavaScript.* Available at: https://www.peterhigginson.co.uk/RISC/ (Accessed: May 30, 2025).

Higginson, P.L. (2014) *Little Man Computer.* Available at: https://peterhigginson.co.uk/lmc/ (Accessed: May 30, 2025).

Howard Gardner (1987) *The Theory of Multiple Intelligences.* Available at: https://www.jstor.org/stable/pdf/23769277.pdf?refreqid=fastly-default%3A77453d850c9d11af88de631d15d84fab&ab_segments=&initiator=&acceptTC=1 (Accessed: May 30, 2025).

Illinois State University (2014) *Simulation of Little Man Computer.* Available at: https://web.archive.org/web/20090227005033/http://www.acs.ilstu.edu/faculty/javila/lmc/ (Accessed: May 30, 2025).

Javed, N. and Zeeshan, F. (2022) 'LMC + Scratch: A recipe to construct a mental model of program execution', Available at: 10.1145/3498343.3498353.

jebbs (2024) *PlantUML.* Available at: https://marketplace.visualstudio.com/items/?itemName=jebbs.plantuml (Accessed: May 30, 2025).

Kabra, K., Ahn and tsjest (2025) *ts-jest.* Available at: https://www.npmjs.com/package/ts-jest (Accessed: May 30, 2025).

Lave, J. and Wenger, E. (1991) *Situated learning : legitimate peripheral participation.* Available at: https://bibliotecadigital.mineduc.cl/bitstream/handle/20.500.12365/17387/cb419d882cd5bb5286069675b449da38.pdf?sequence=1 (Accessed: May 30, 2025).

Lee, M., Attard, S., Zhao, E., Xu, G. and vertedinde (2025a) *@electron-forge/cli.* Available at: https://www.npmjs.com/package/@electron-forge/cli (Accessed: May 30, 2025).

Lee, M., Attard, S., Zhao, E., Xu, G. and vertedinde (2025b) *@electron-forge/maker-squirrel.* Available at: https://www.npmjs.com/package/@electron-forge/maker-squirrel (Accessed: May 30, 2025).

LINGsCARS (2025) *LINGsCARS.* Available at: https://www.lingscars.com (Accessed: May 30, 2025).

Maslow, A.H. (1943) *A Theory of Human Motivation.* Available at: https://www.un-forum.org/books/library/(digimob)%20Student%20of%20the%20Occult%20Mega-Torrent%20%232.3%20(L%20-%20Z)/Philosophy/Maslow,%20A.H.%20-%20A%20Theory%20of%20Human%20Motivation.pdf (Accessed: May 30, 2025).

Mauliya, I., Relianisa, R. and Rokhyati, U. (2020) *LACK OF MOTIVATION FACTORS CREATING POOR ACADEMIC PERFORMANCE IN THE CONTEXT OF GRADUATE ENGLISH DEPARTMENT STUDENTS.* Available at: https://ejournal.uinfasbengkulu.ac.id/index.php/linguists/article/view/3956/3175 (Accessed: May 30, 2025).

mdn (2025) *Window - Web APIs.* Available at: https://developer.mozilla.org/en-US/docs/Web/API/Window (Accessed: May 30, 2025).

mdr (2025) *File origins.* Available at: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#file_origins (Accessed: May 30, 2025).

Millett, S., Blankenship, J. and Bussa, M. (2011) 'Pro Agile .NET Development with Scrum', Available at: 10.1007/978-1-4302-3534-7.

Molnar, D. (2013) 'Cracking the colour code:
A case study of red', *Jezikoslovlje,* 14.

Muhajirah, M. (2020) 'Basic of Learning Theory: (Behaviorism, Cognitivism, Constructivism, and Humanism)', Available at: 10.46966/ijae.v1i1.23.

Nakatsu, R., Tosa, N., Rauterberg, M. and Xuan, W. (2017) *Entertainment, Culture, and Media Art.* Available at: https://www.researchgate.net/profile/Matthias-Rauterberg/publication/312005441_Entertainment_Culture_and_Media_Art/links/5f07977045851550509 88f08/Entertainment-Culture-and-Media-Art.pdf (Accessed: May 30, 2025).

Openmoji.org (2020) *Android SVG Vector.* Available at: https://www.svgrepo.com/svg/311690/android (Accessed: May 30, 2025).

Osborne, H. and Yurcik, W. (2002) 'The educational range of visual simulations of the Little Man Computer architecture paradigm', Available at: 10.1109/FIE.2002.1158742.

Osman, K., Bakar, A. and Nurul (2012) 'Educational Computer Games for Malaysian Classrooms: Issues and Challenges', Available at: 10.5539/ass.v8n11p75.

Pawson, R. and Higginson, P. (2020) *Assembly Language Programming.* Available at: https://metalup.org/al/AL_Student.pdf (Accessed: May 30, 2025).

Piaget, J. (1955) *The construction of reality in the child.* Available at: https://web.archive.org/web/20170808105211id_/http://pages.uoregon.edu/rosem/Timeline_files/The%20Construction%20of%20Reality%20in%20the%20Child.pdf (Accessed: May 30, 2025).

Piaget, J. and Warden, M. (1927) 'The Language and Thought of the Child', Available at: 10.2307/1415214.

Possler, D., Kümpel, A.S. and Unkel, J. (2020) *Entertainment Motivations and Gaming-Specific Gratifications as Antecedents of Digital Game Enjoyment and Appreciation.* Available at: https://anna-kuempel.de/publication/possler-entertainment-2020/possler-et-al_2020.pdf (Accessed: May 30, 2025).

Rueckstiess, T., Jordan, D., Schroll, A., Broadstone, M., Wolff, H., Sinha, S., Ratcliffe, M., Allen, J., Balsano, J., Weir, J., Casimir, M., Stefano, K., Jarjee, [.f.n., Shaket, B., Henningsen, A., Sup, G., Krish, M., Irinchev, N., Pears, B. and Holmquist, L. (2024) *electron-squirrel-startup.* Available at: https://www.npmjs.com/package/electron-squirrel-startup (Accessed: May 30, 2025).

Scratch Foundation (2024a) *Our Story.* Available at: https://www.scratchfoundation.org/our-story (Accessed: May 30, 2025).

Scratch Foundation (2024b) *Scratch Foundation.* Available at: https://www.scratchfoundation.org/our-story (Accessed: May 30, 2025).

Sharp, R. (2025) *nodemon.* Available at: https://www.npmjs.com/package/nodemon (Accessed: May 30, 2025).

Shepherd, A. and L., A. (2021) *Extending global types (e.g. "Window") inside a typescript module.* Available at: https://stackoverflow.com/questions/47130406/extending-global-types-e-g-window-inside-a-typescript-module/ (Accessed: May 30, 2025).

SVG Repo (2023) *Licensing.* Available at: https://www.svgrepo.com/page/licensing/ (Accessed: May 30, 2025).

Tauri (2025) *Tauri 2.0.* Available at: https://v2.tauri.app (Accessed: May 30, 2025).

The Chromium Project (2023) *Chromium.* Available at: https://github.com/chromium/chromium?tab=readme-ov-file#-chromium (Accessed: May 30, 2025).

Tiiny.host (2019) *Home.* Available at: https://tiiny.host (Accessed: May 30, 2025).

Tomorrow Corporation (2015) *Human Resource Machine.* Available at: https://tomorrowcorporation.com/humanresourcemachine (Accessed: May 30, 2025).

trincot (2025) *trincot.github.io.* Available at: https://github.com/trincot/trincot.github.io (Accessed: May 30, 2025).

trincot (2021a) *lmc.* Available at: https://trincot.github.io/lmc.html (Accessed: May 30, 2025).

trincot (2021b) *lmc.* Available at: https://github.com/trincot/lmc (Accessed: May 30, 2025).

trincot (2020) *Trincot.* Available at: https://trincot.github.io (Accessed: May 30, 2025).

types (2024) *@types/jest.* Available at: https://www.npmjs.com/package/@types/jest (Accessed: May 30, 2025).

TypeScript (2023) *Optional Parameters.* Available at: https://www.typescriptlang.org/docs/handbook/2/functions.html?form=MG0AV3#optional-parameters (Accessed: May 30, 2025).

typescript-bot, Wigham, &., Wesley, Shively-Sanders, &., Nathan, Branch, &., Andrew, Nandi, &., Sheetal, typescript-deploys, Bailey, &. and Jake (2025) *typescript.* Available at: https://www.npmjs.com/package/typescript (Accessed: May 30, 2025).

Vertex42 (2021) *Simple Gantt Chart.* Available at: https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html (Accessed: May 30, 2025).

Visual Studio Code (2025) *Search and replace.* Available at: https://code.visualstudio.com/docs/editor/codebasics?form=MG0AV3#_search-and-replace (Accessed: May 30, 2025).

Visual Studio Code (2018) *Extensions for Visual Studio Code.* Available at: https://marketplace.visualstudio.com/vscode (Accessed: May 30, 2025).

W3Schools (2024) *HTML DOM Audio play() Method.* Available at: https://www.w3schools.com/jsref/met_audio_play.asp (Accessed: May 30, 2025).

W3Schools (2023) *Bootstrap 5 Dark Mode.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_dark_mode.php (Accessed: May 30, 2025).

W3Schools (2021a) *Bootstrap 5 Button Groups.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_button_groups.php (Accessed: May 30, 2025).

W3Schools (2021b) *Bootstrap 5 Buttons.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_buttons.php (Accessed: May 30, 2025).

W3Schools (2021c) *Bootstrap 5 Containers.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_containers.php (Accessed: May 30, 2025).

W3Schools (2021d) *Bootstrap 5 Input Groups.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_form_input_group.php (Accessed: May 30, 2025).

W3Schools (2021e) *Bootstrap 5 Tables.* Available at: https://www.w3schools.com/bootstrap5/bootstrap_tables.php (Accessed: May 30, 2025).

W3Schools (2021f) *Bootstrap CSS Images Reference.* Available at: https://www.w3schools.com/bootstrap/bootstrap_ref_css_images.asp (Accessed: May 30, 2025).

W3Schools (2021g) *CSS The !important Rule.* Available at: https://www.w3schools.com/css/css_important.asp (Accessed: May 30, 2025).

W3Schools (2018) *CSS Variables - The var() Function.* Available at: https://www.w3schools.com/css/css3_variables.asp (Accessed: May 30, 2025).

W3Schools (2015a) *Bootstrap Input Sizing.* Available at: https://www.w3schools.com/bootstrap/bootstrap_forms_sizing.asp (Accessed: May 30, 2025).

W3Schools (2015b) *Bootstrap Wells.* Available at: https://www.w3schools.com/bootstrap/bootstrap_wells.asp (Accessed: May 30, 2025).

W3Schools (2014a) *Click the buttons to play or pause the audio.* Available at: https://www.w3schools.com/JSREF/tryit.asp?filename=tryjsref_audio_play (Accessed: May 30, 2025).

W3Schools (2014b) *Location search Property.* Available at: https://www.w3schools.com/jsref/prop_loc_search.asp (Accessed: May 30, 2025).

W3Schools (2010) *Location hash Property.* Available at: https://www.w3schools.com/jsref/prop_loc_hash.asp (Accessed: May 30, 2025).

Watson, J.B. (1913) *Psychology as the behaviorist views it.* Available at: https://pure.mpg.de/rest/items/item_2404108/component/file_2404107/content (Accessed: May 30, 2025).

Wellingborough School (2025) *Wellingborough School.* Available at: https://www.wellingboroughschool.org (Accessed: May 30, 2025).

Wikipedia contributors (2025) *Little man computer.* Available at: https://en.wikipedia.org/w/index.php?title=Little_man_computer&oldid=1269373119 (Accessed: May 30, 2025).

Yurcik, W. and Brumbaugh, L. (2001) 'A web-based little man computer simulator', Available at: 10.1145/364447.364585.

Yurcik, W. and Osborne, H. (2001) 'A crowd of Little Man Computers: visual computer simulator teaching tools', Available at: 10.1109/WSC.2001.977496.

Zachtronics (2020) *Exapunks.* Available at: https://www.zachtronics.com/exapunks/ (Accessed: May 30, 2025).

Zachtronics (2016) *About SHENZHEN I/O.* Available at: https://www.zachtronics.com/shenzhen-io/ (Accessed: May 30, 2025).

Zachtronics (2015) *About TIS-100.* Available at: https://www.zachtronics.com/tis-100/ (Accessed: May 30, 2025).

Zhao, Z., Zhou, Y., Tan, G. and Li, J. (2018) 'Research progress about the effect and prevention of blue light on eyes', *International Journal of Ophthalmology,* 11(12), pp. 1999–2003 Available at: 10.18240/ijo.2018.12.20.

11 Appendices

11.1 Appendix A: Glossary

11.1.1 Abbreviations

Abbreviations	Meaning
CPU	Central Processing Unit
PC	Program Counter
CU	Control Unit
ALU	Arithmetic Logical Unit
RAM	Random Access Memory
MAR	Memory Address Register
MDR	Memory Data Register
CS	Computer Science
JS	JavaScript
TS	TypeScript
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
OS	Operating System
VS	Visual Studio
OOP	Object-Oriented Programming
IPOD	Input, Process, Output, and Decision
IPO	Input, Process, and Output
HCI	Human-Computer Interaction
JSON	JavaScript Object Notation
URL	Universal Resource locator
SVG	Sector Vector graphics
IDE	Integrated Development Environment
GUI	Graphical User Interface
CLI	Command-Line Interface
API	Application Programming Interface
UML	Unified Modelling Language
PWA	Progressive Web Application
CDN	Content Delivery Network
NPM	Node Package Manager
GDPR	General Data Protection Regulation
GCSE	General Certificate of Secondary Education
A-Level	Advanced Level (UK pre-university qualification)
AQA	Assessment and Qualifications Alliance
OCR	Oxford, Cambridge & RSA Examinations
PERT	Program Evaluation Review Technique
MIR	Memory Instruction Register
CIR	Current Instruction Register
DOM	Document Object Model
UI	User Interface
STEM	Science, Technology, Engineering, and Mathematics
PNG	Portable Network Graphics
GIF	Graphics Interchange Format
IO	Input Output
PERT	Program Evaluation and Review Technique

Table 11.1—1 - abbreviations and thier corrosponding full forms.

Despite GANTT being widely typed/written in upper-case, in the development industry, it is in fact not an abbreviation.

11.2 Appendix B: Marking Scheme

IP40 Default Marking Scheme – THESIS

11.2.1 Part A: Development

Part A: Development (15% of the total marks available)	Weight %
Methodical approach to work/developing a plan of work.	5
Working independently generally and independently of the supervisor. Is the work substantially the candidate's own work?	5
Using initiative and/or imagination (e.g. in developing the scope of the Individual Study), extending familiar concepts and a general understanding of the work.	5
Total %	15

Table 11.2—1 - IP40 default marking scheme for development.

11.2.2 Part B: Report (IP Submission Only)

Part B: Report (IP Submission Only) (85% of the total marks available)	Weight %
Introduction: background to the problem, justification for the chosen task, etc., which maybe contained in more than just the Introduction chapter.	5
Systems Analysis: methods and procedures employed; quality of products; a clear and complete statement of (the current system, if appropriate, and) and the user requirements; data/processing models; perhaps user interface specifications; non-functional requirements; etc.	10
Systems Design: methods & procedures employed; consistency with systems analysis; data/processing models; user interface models (e.g. dialogues, menus, control structures, information displays); etc.	10
Code quality: consistency with systems design; comments, maintainability, extendibility, efficiency, portability, reusability, etc.	5
Unit & system testing: clear testing strategies; complete/comprehensive/well structured test plan including expected results, actual results and a management summary.	10
Quality of implementation: usability, value and usefulness. Does it actually work?	15
HCI considerations as implemented.	5
User Guide(s).	5
Project management procedures: schedules; monitoring of progress; etc.	5
Presentation, organisation and style: quality of English (with few editorial, grammatical or spelling errors); appropriate writing style with due emphasis on formality; relevant use of jargon, technical terms and glossary; physical and logical layout of the report; formatting; headings/sub-headings; page numbering; binding; table of contents; margins and white space; tables; diagrams; sequencing of chapters/topics; appropriate use of appendices; cross referencing; ease of auditing; appropriate abstract; acknowledgments; quoting and in-text referencing of sources; departmental standards for report layout & presentation; references and bibliographies; etc.	5
Achievement: Generally, and more specifically with respect to meeting the requirements specified in the Requirements Specification and the (amended) PID.	5
Conclusions, critical perspective, evaluation and suggestions for further work.	5
Total %	85

Table 11.2—2 - IP40 default marking scheme for report.

11.3 Appendix C: Changes to the Project Initiation Document

Deliberately Left Blank

11.4 Appendix D: Current Environment Investigation Report

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.5 Appendix E: Requirements Specification

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.6 Appendix F: Design Report

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.7 Appendix G: Implementation

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.8 Appendix H: Testing

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.9 Appendix I: User Guide

Not Applicable, see Appendices L, M and possibly N for Agile development method timeboxes.

11.10 Appendix J: Project Management

11.10.1 Overview

The overview of the project management can easily be summed up in the project presentation poster, as seen below.

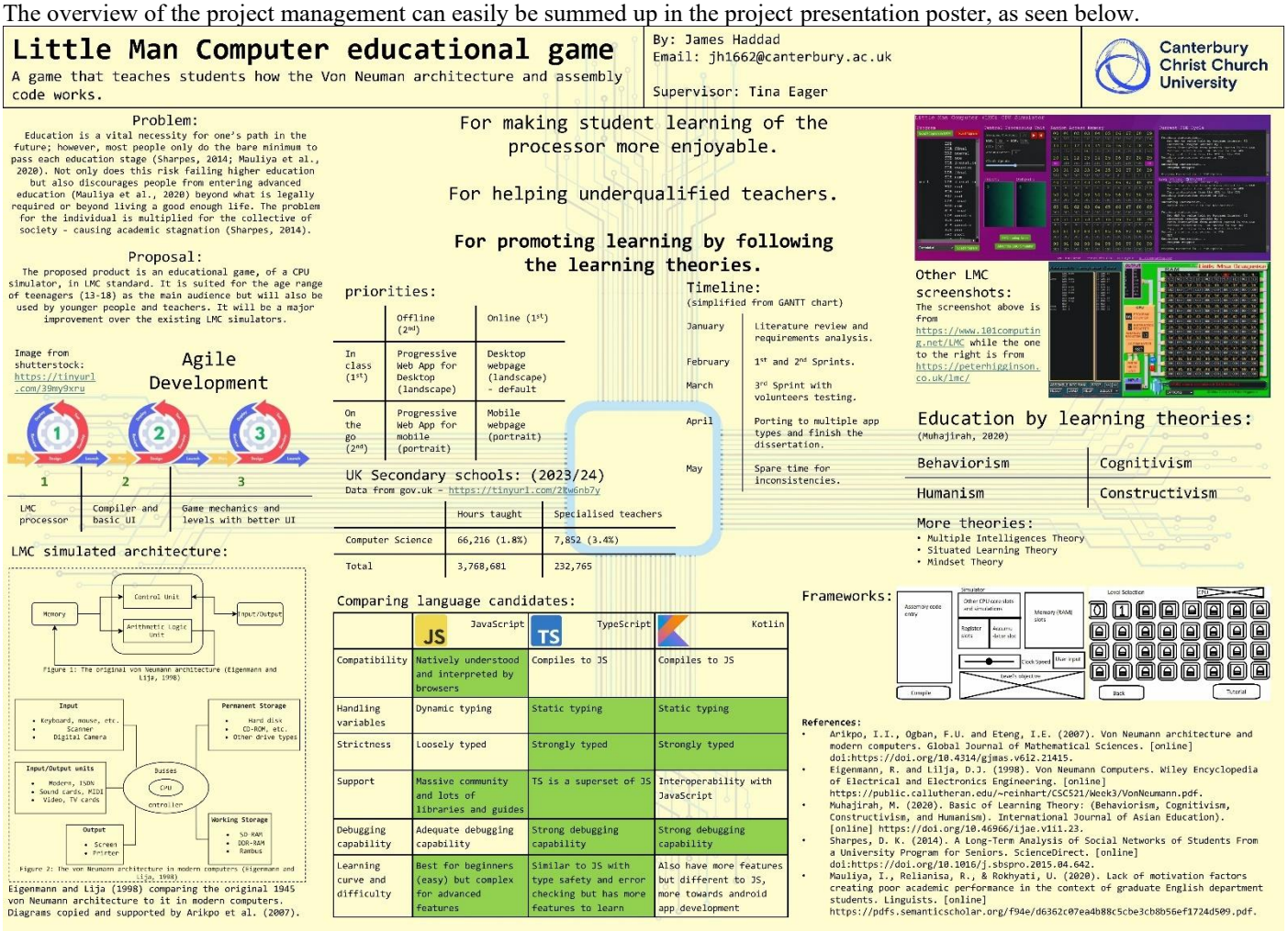


Figure 11.10—1 - screenshot of the presented project poster used as an outline.

11.10.2 Critical Path Analysis

As per the agile sprint, each sprint of the Agile development methodology will have a plan before implementation, and each plan includes the sprint-specific critical path analysis. To see the sprint-specific critical path analyses, see Appendices L, M, and N.

11.10.2.1 Project flow - PERT chart (time tree)

Before making a time-constrained plan, like a GANTT chart, the project needs to know exactly what tasks need to be done, in what order, and their relationship with each other (dependency, concurrency, et cetera). For that reason, it is decided that a PERT chart (time tree) will be done first, as seen below:

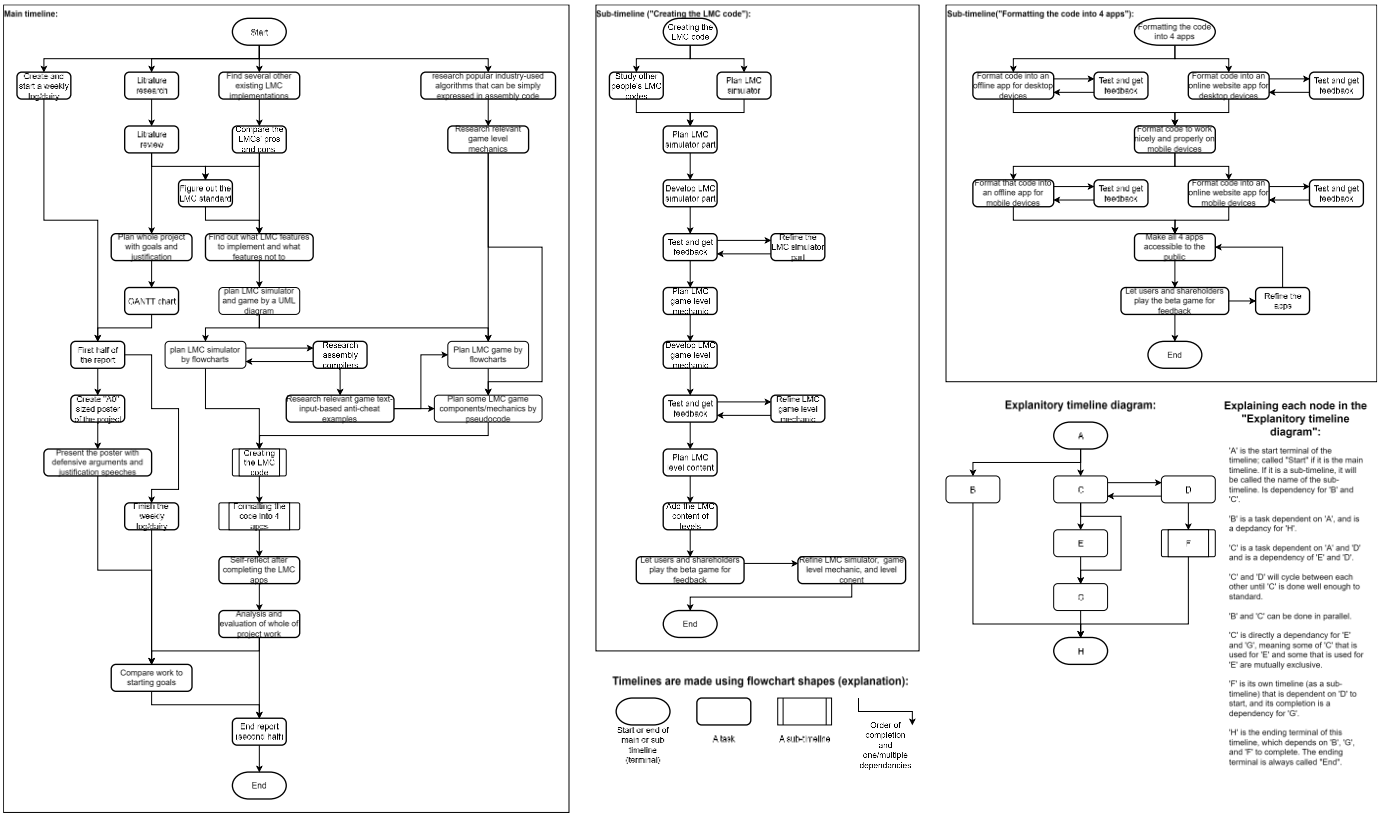


Figure 11.10—2 - a SVG image of a PERT chart (time tree) of the project.

Note that the proposal part is not included here because it was done before creating and starting to follow the time tree.

The PERT chart is in SVG format, meaning if the reader has difficulty reading, the reader can zoom in without the figure losing quality.

11.10.2.2 GANTT chart (timeline chart)

With the lot of ordered relational tasks ascertained, we can now add the time constraints to create a GANTT chart using a GANTT chart template (Vertex42, 2021).

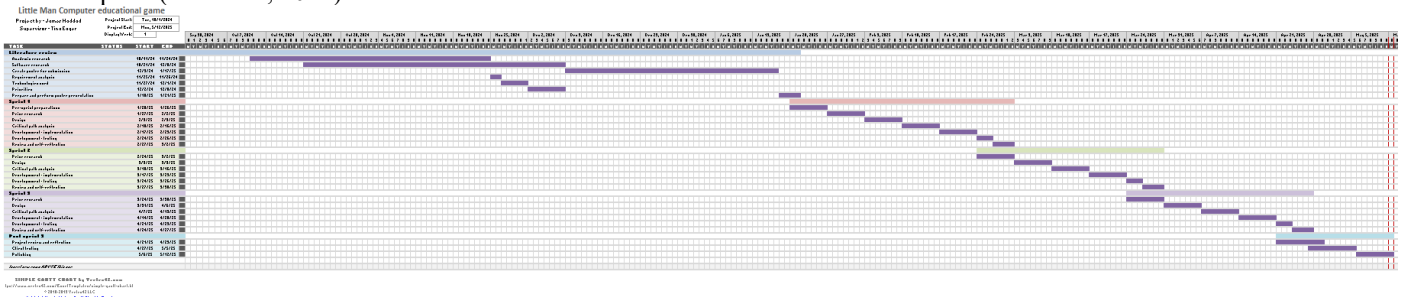


Figure 11.10—3 - overview screenshot of the GANTT.

Due to the sheer size of the GANTT chart, it cannot be fully viewed in this dissertation. Please refer to the full version (Haddad, 2025a) to see it in full detail.

Every task has been planned in a timeline with extra time to take unexpected minor delays into account, such as sickness. However, this does not include major unexpected delays into account, such as hospitalisation for a bone fracture. In that case, lower priorities will have to be skipped. For a task to be deemed low priority, it must not be a dependency for any major task that heavily depends on it, and those tasks do not heavily affect the end product. For example, the research "*anti-cheat*" task is preferred to help with planning the LMC game, but only slightly hinders the LMC game creation without any notice of consideration when playing the game; therefore, it is a low priority. Meanwhile for a high-priority example will be the research "*LMC standard*" because if the game is not up to LMC standard, then students will not learn the relevant information for later education, defeating a huge portion of the reasoning behind the LMC game in the first place.

The allocated times are also determined by the risk matrix table below

	negligible	minor	moderate	significant	severe
very unlikely				TS failing to compile	
unlikely		Difficulty with setting up a UI in the first place.	TS-compiled JS not working for Node.js and the browser.		
possible		Porting to other platforms failing.			
likely		No or not as much feedback expected			
very likely		Difficulty with UI styling			

Table 11.10—1 - risk management table.

11.10.2.3 Planning and risk management

11.10.3 Priorities

To read about how the author deals with differing levels of priority, please see the last paragraph of **GANTT chart (timeline chart)**.

11.10.3.1 Product platform priorities

As seen in the PERT chart, the author plans to make 4 versions of the LMC game – one per platform. Those are desktop online, desktop offline, mobile online and mobile offline.

The priorities of each platform can easily be visualised and explained using **Figure 11.10—4** as seen below.

	Offline (2nd priority)	**Online (1st priority)**
In class (1st priority)	Progressive Web App for Desktop (landscape)	Desktop webpage (landscape) - default
On the go (2nd priority)	Progressive Web App for mobile (portrait)	Mobile webpage (portrait)

Figure 11.10—4 - priority display of platforms.

Because our target audiences are students learning Computer Science (or a similar subject) in school, we prioritise them using the product in class. We assume that the class will be in a computer classroom (with one desktop per student) – therefore, this is considered the same as somebody using a desktop at home or any other compatible computer with a landscape screen and mouse and keyboard peripherals (such as laptops). This demotes the "*on the go*" to second priority in what computer type platform, as the author expects students are less likely to play the LMC game on their smartphones. This demotion is also backed up by the fact that changing a program to work properly in both landscape and portrait is no easy task.

Besides the physical aspect of the platforms, it is also important to consider whether these platforms are online or not. This is also an important aspect of differing platforms, as both online and offline applications are widely used with highly considerable advantages and disadvantages. To be more clear, online applications mean they run in the computer's browser and are accessed using a URL (not stored in permanent storage), while offline applications are their own software and are stored in the computer's permanent storage. There are many pros and cons of each one, but the author has decided to only cover the ones that are relevant to the main audience (teenage school students). These pros and cons can be seen in the table below. Because online and offline are polar opposites, so are their pros and cons. Hence, only the pros of each one are shown as the con of one is the pro of the other.

Pros of online applications	**Pros of offline applications**
No installation on permanent storage required.	No internet connection required.
More secure as security is handled by the browser.	No browser required.
People are more willing to try out online than offline due to convenience – no need to uninstall if not liked.	No need to download application assets every time.
No computer administrator permissions are needed to run the applications.	Exact style and how it runs consistently unlike using one of multiple browsers' engines.
Easier and quicker to set up in a classroom of computers (just give the application URL).	Performance is not limited by browser-imposed restrictions.

Table 11.10—2 - comparing the pros of online against offline applications

As seen, the online application pros are a lot more significant in the school workload. For example, classroom computers have a small chance of momentarily losing internet access but a far larger chance of not having administrator permission to install an application for security reasons (preventing students from installing malicious software).

Because of this, being online is a priority compared to being offline.

Figure 11.10—5 shows these priorities used in the project presentation.

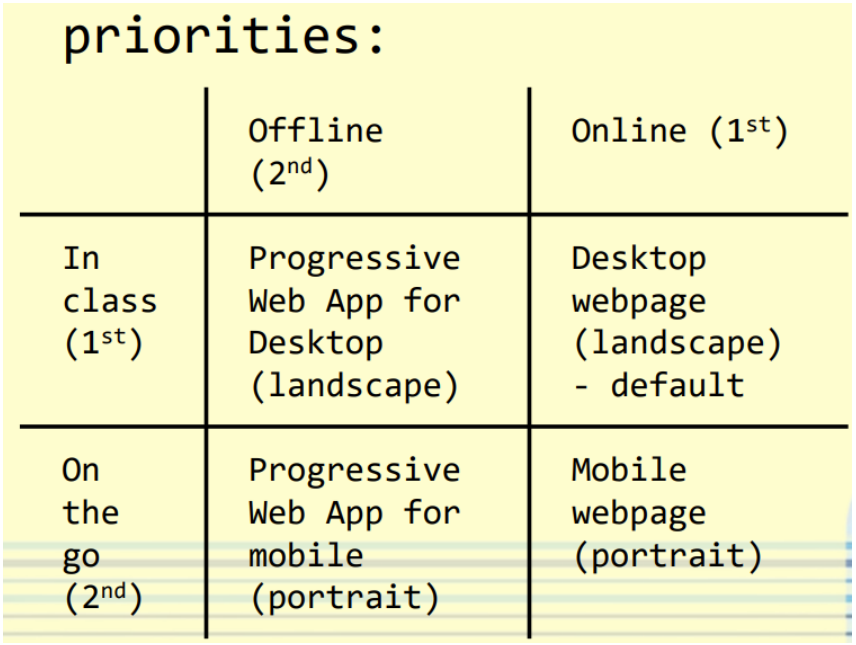


Figure 11.10—5 - snapshot of the poster showing the priorities of the platform for the product.

LMC + Scratch: A recipe to construct a mental model of program execution 3

Code	Name	Description
0	HLT	Stop execution
1xx	ADD	Add the contents of the memory address to the Accumulator
2xx	SUB	Subtract the contents of the memory address from the Accumulator
3xx	STA or STO	Store the value in the Accumulator in the memory address given.
4		Not in use
5xx	LDA	Load the Accumulator with the contents of the memory address given
6xx	BRA	Branch - use the address given as the address of the next instruction
7xx	BRZ	Branch to the address given if the Accumulator is zero
8xx	BRP	Branch to the address given if the Accumulator is zero or positive
9xx	INP or OUT	Input or Output. Take from Input if address is 1, copy to Output if address is 2.
	DAT	Used to indicate a location that contains data.

Table 1. LMC - Instruction Set

Table 11.10—3 - snapshot of the modern LMC instruction set table by Javed and Zeeshan (2022).

11.10.3.2 Product feature priorities – MoSCoW

As mentioned in **Overall priorities - MoSCoW**, MoSCoW tables are used to determine the priority of each planned feature/functionality of the specific sprint alongside a vague draft MoSCoW table of the overall project. To see the more detailed and specialised MoSCoW tables, please see them inside the planning sections of the sprints, in **Appendix L: Agile Development: Timebox 1**, **Appendix M: Agile Development: Timebox 2**, and **Appendix N: Agile Development: Timebox 3**.

11.11 Appendix K: Meetings with Supervisor

Below is a detailed list of all the meetings with my supervisor:

11.11.1 Prior to project presentation

11.11.1.1 26th October 2024 09:35-10:30

11.11.1.1.1 Purpose and description

Talking about first steps and what the poster, presentation, and dissertation should have.

11.11.1.1.2 Notes from meeting

- Posters:
    - Key things shown in poster example #1: justification, methods, conclusion, identifying the problem, explaining how the project could work, introduction, and deliverables.
    - Key things shown in poster example #2: introduction, aims, design, approach and methodology, screenshots, progress so far, next steps, framework diagrams, and references.
    - Key things shown in poster example #3: references, methods, time plan, improvements, objectives, and demonstration of product in the middle of the poster.
    - Key things that my post should have:
        - Be pretty and simple
        - Include bits of the literature review
        - Diagrams, screenshots, and other pictures of the project
        - More tables and fewer words
        - Maybe include screenshots of other people's LMC programs
    - The poster can be stored with the supervisor before the presentation day.
    - Poster must be printed by the author's self (not the university's responsibility).
    - In contrast to the first Individual Project lecture, the poster does not have to be A2 size. For example, it can be A3 or A1.

- My poster and presentation will be confronted and marked by my supervisor and a CCCU staff member called Vijay Sahota. There will be other confronters, but they won't be my markers.
- The lead of the Individual Project is Konstantinos Sirlantzis.
- I must keep track of:
    - Weekly logs.
    - Communication with relevant people.
    - Audits of supervisor meetings.

- Generic questions that will most likely be asked of the author at the poster presentation:
    - What are your plans?
    - In technical terms, how will you implement?
    - What is your product lifecycle?
    - How to test your product?
    - How is the literature review?
    - Who is the target audience and why?

- My Literature review should have:
    - Education theory
    - Learning theory
    - Games in education/learning
    - Review software
    - Not only literature!
    - Human computer interface for everyone with disabilities, such as colour-blindness
    - The acronym "MoSCoW" (must have, should have, could have, won't have)
    - Framework diagrams
    - Analysis and design methods

11.11.1.1.3 Actions

Set up the different sections in my dissertation, start a log document and apply the discussed points to my poster.

11.11.1.1.4 Evidence

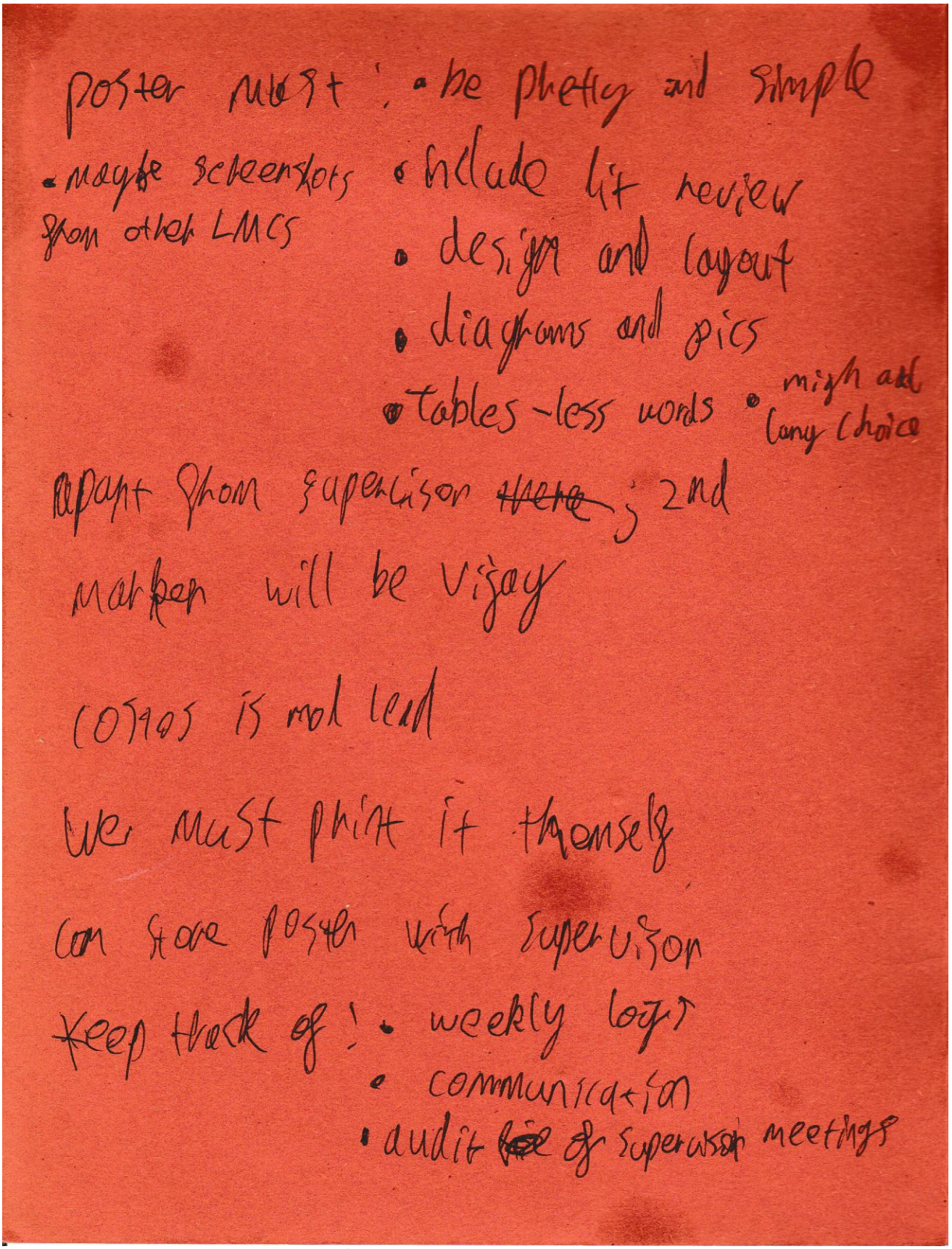


Figure 11.11—1 - first page of notes from the 26th October 2024 meeting with the supervisor.

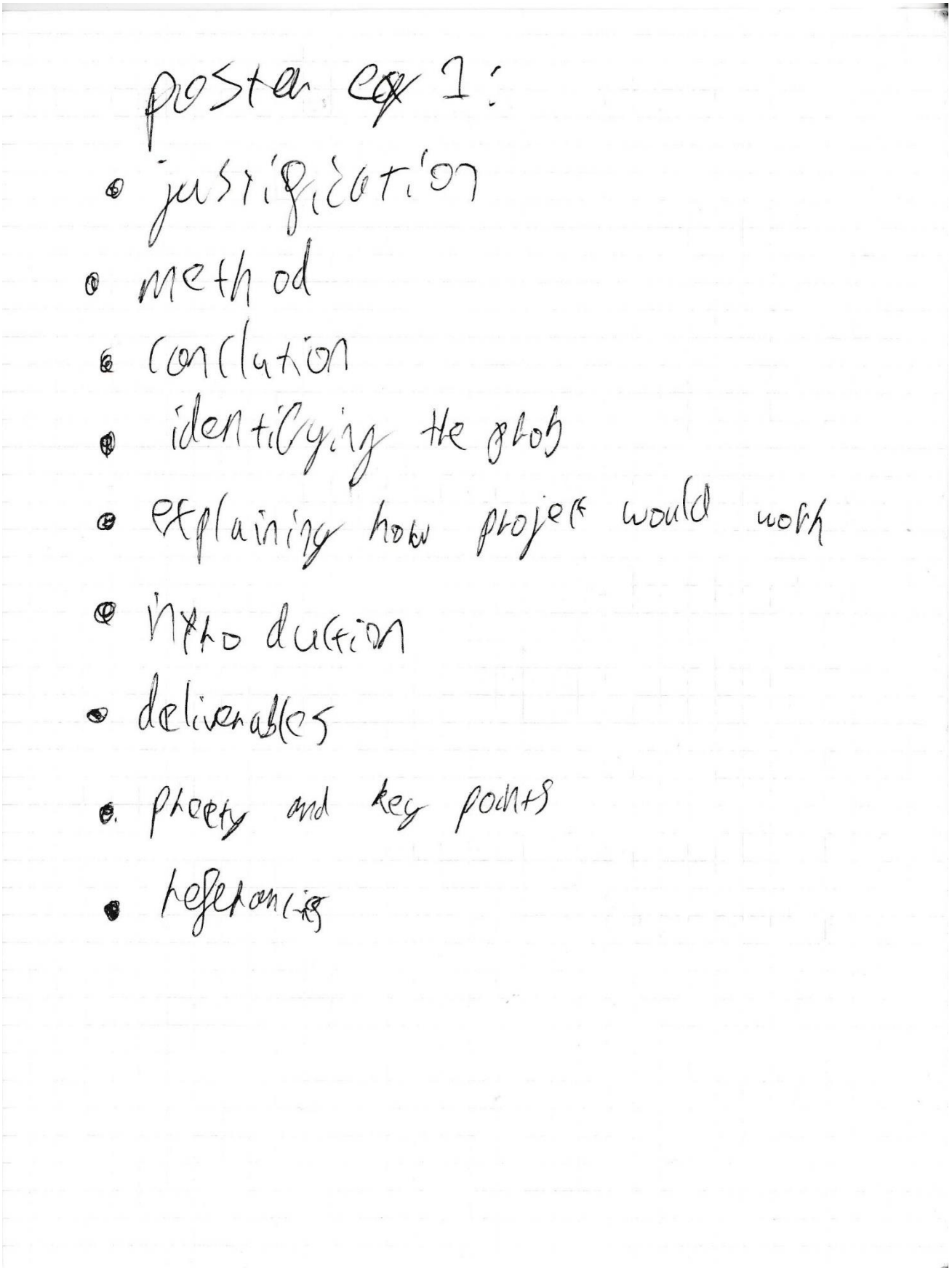


Figure 11.11—2 - second page of notes from the 26th October 2024 meeting with the supervisor.

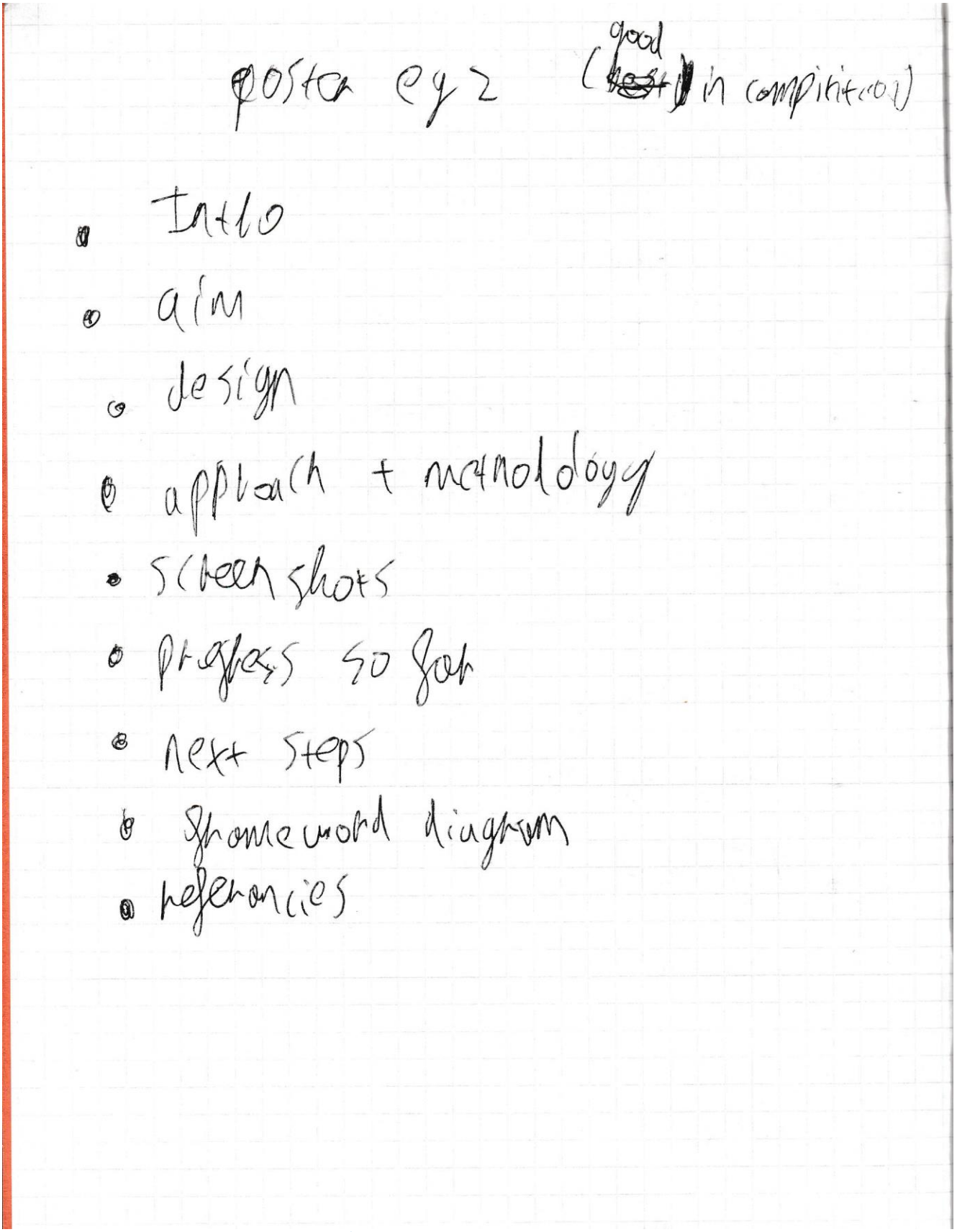


Figure 11.11—3 - third page of notes from the 26th October 2024 meeting with the supervisor.

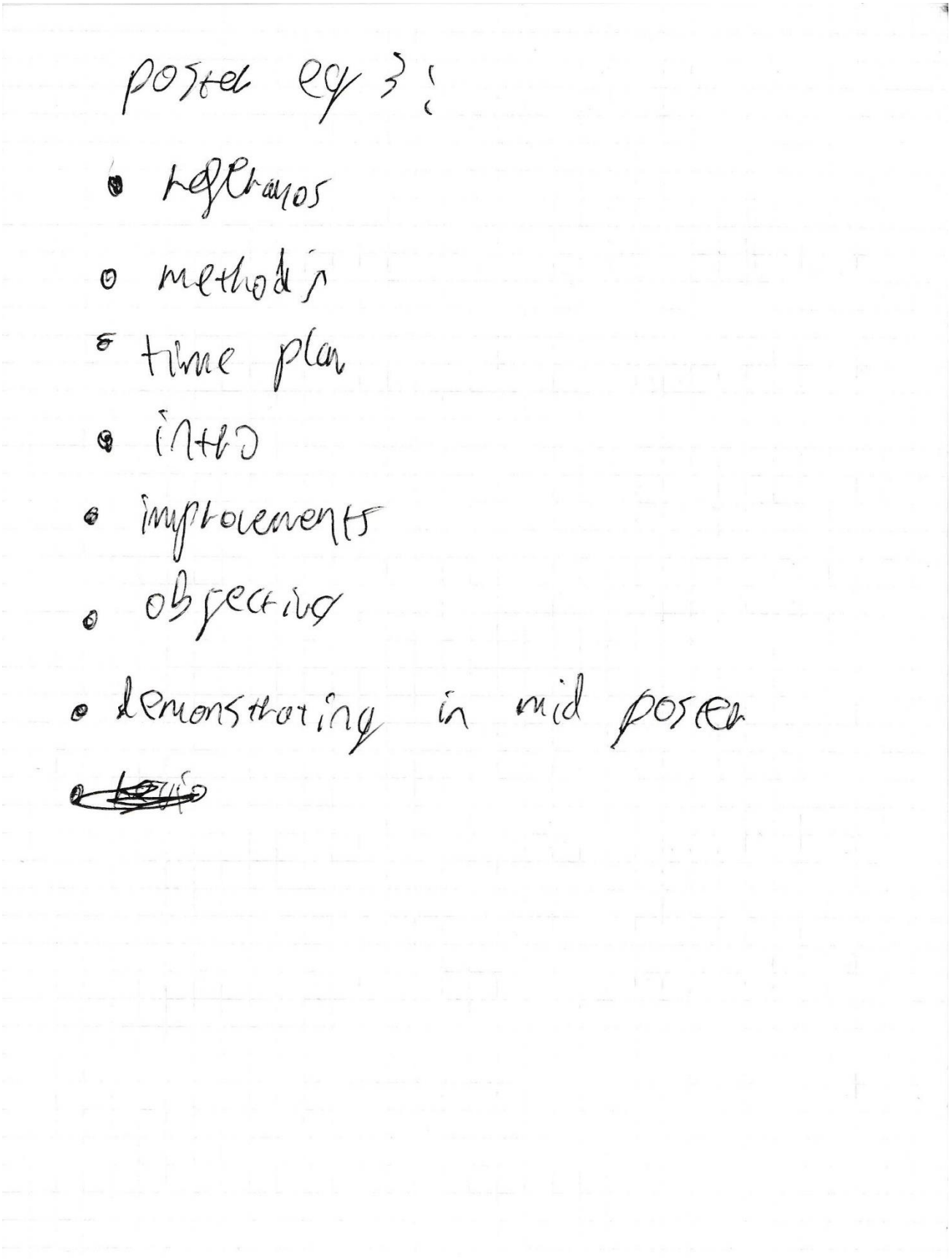


Figure 11.11—4 - fourth page of notes from the 26th October 2024 meeting with the supervisor.

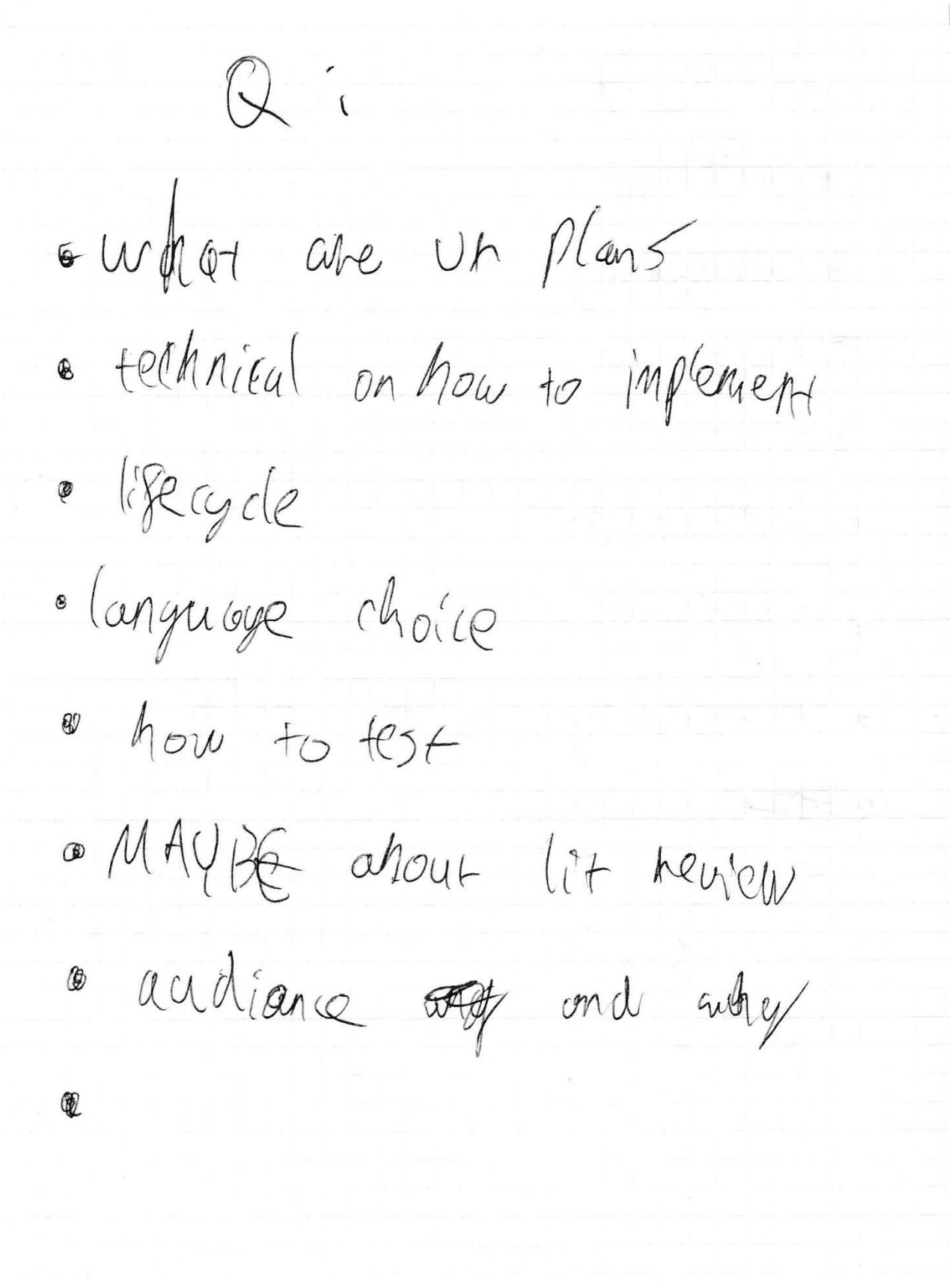


Figure 11.11—5 – fifth page of notes from the 26th October 2024 meeting with the supervisor.

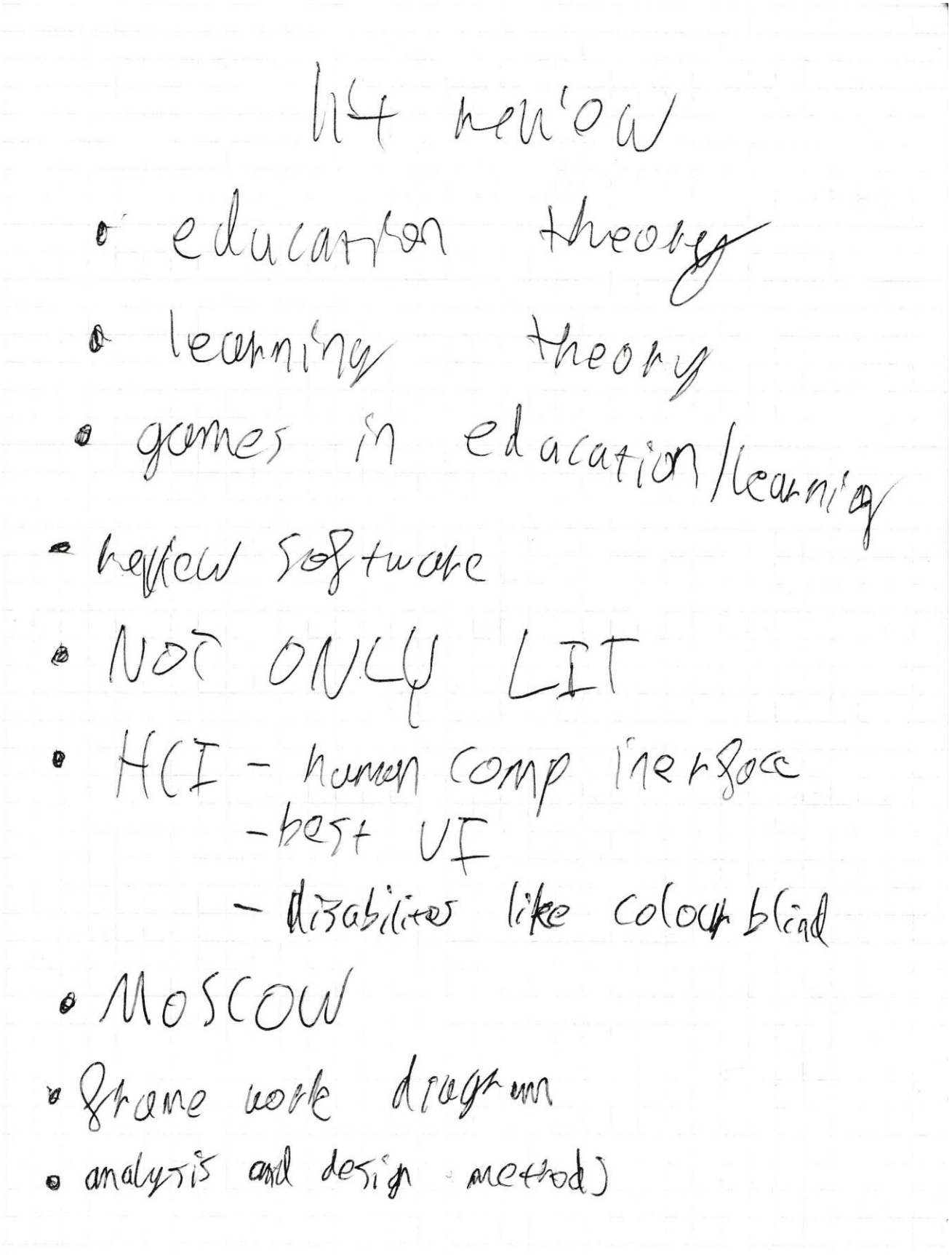


Figure 11.11—6 - fifth page of notes from the 26th October 2024 meeting with the supervisor.

11.11.1.2 12th December 2024 12:30-13:30

11.11.1.2.1 Purpose and description

Together with the supervisor's other supervisees (totalling 4 of us), to discuss regarding the poster presentation.

11.11.1.2.2 Notes from meeting

Relevant notes for the semi-social meeting:

- Because A2 paper is 4x the size of A4, so would the font size – for example, font size 6 on A4 will be size 24 on A2.
- For presentation, the post should not have a white background.
- As mentioned before, the poster must not have much text.
- There is no due date for the ethics form (for volunteer testing of the project product).
- Poster should be colourful but not too colourful like “Ling’s cars” website (LINGsCARS, 2025).
- Confirmed personal meeting on the 17th.
- Need to do the GANTT chart.

11.11.1.2.3 Actions

Changed poster style, layout, and content as seen in **Figure 11.11—7** and **Figure 11.11—8**.

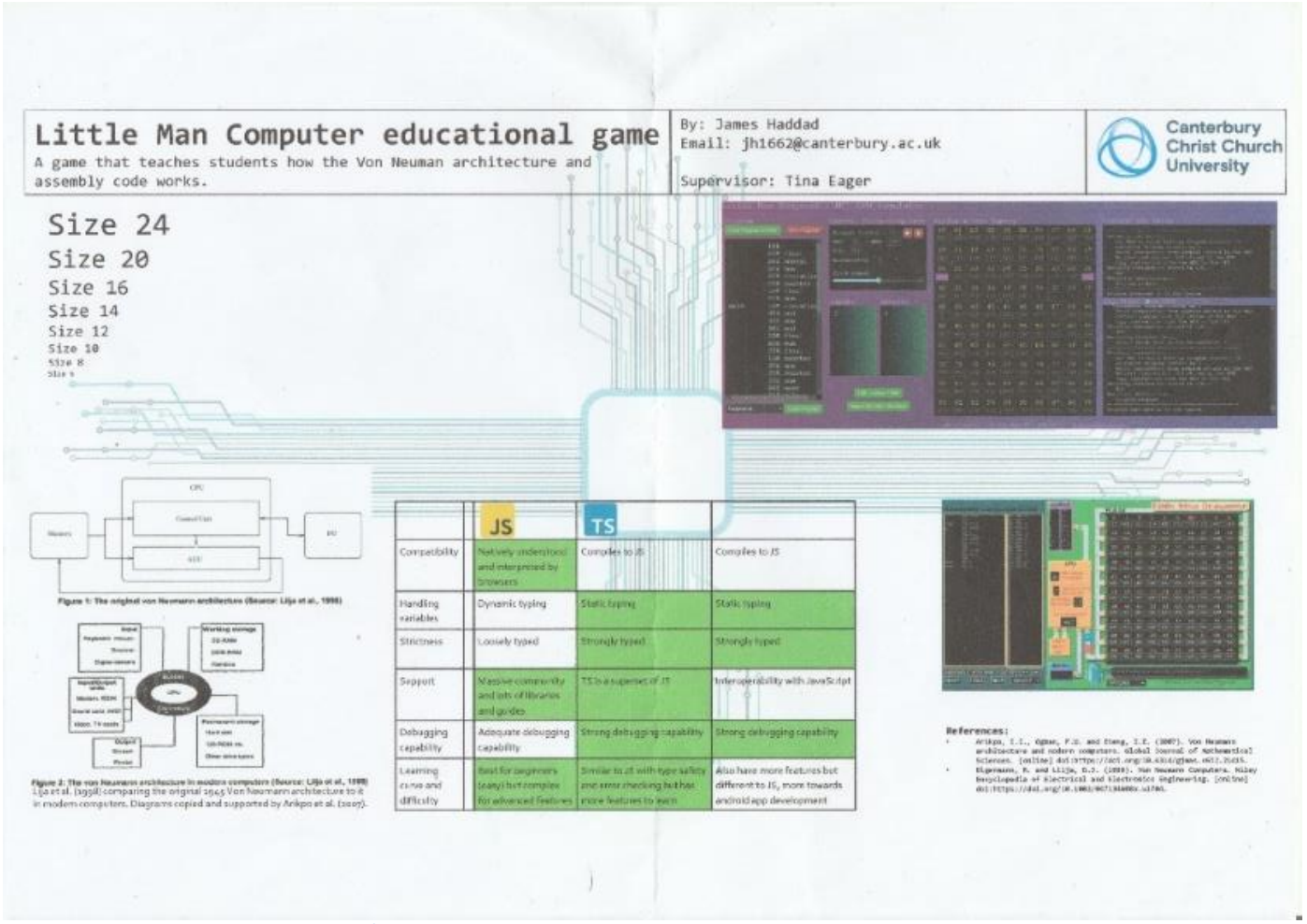


Figure 11.11—7 – poster prior to the 24th December 2025 supervisor meeting.

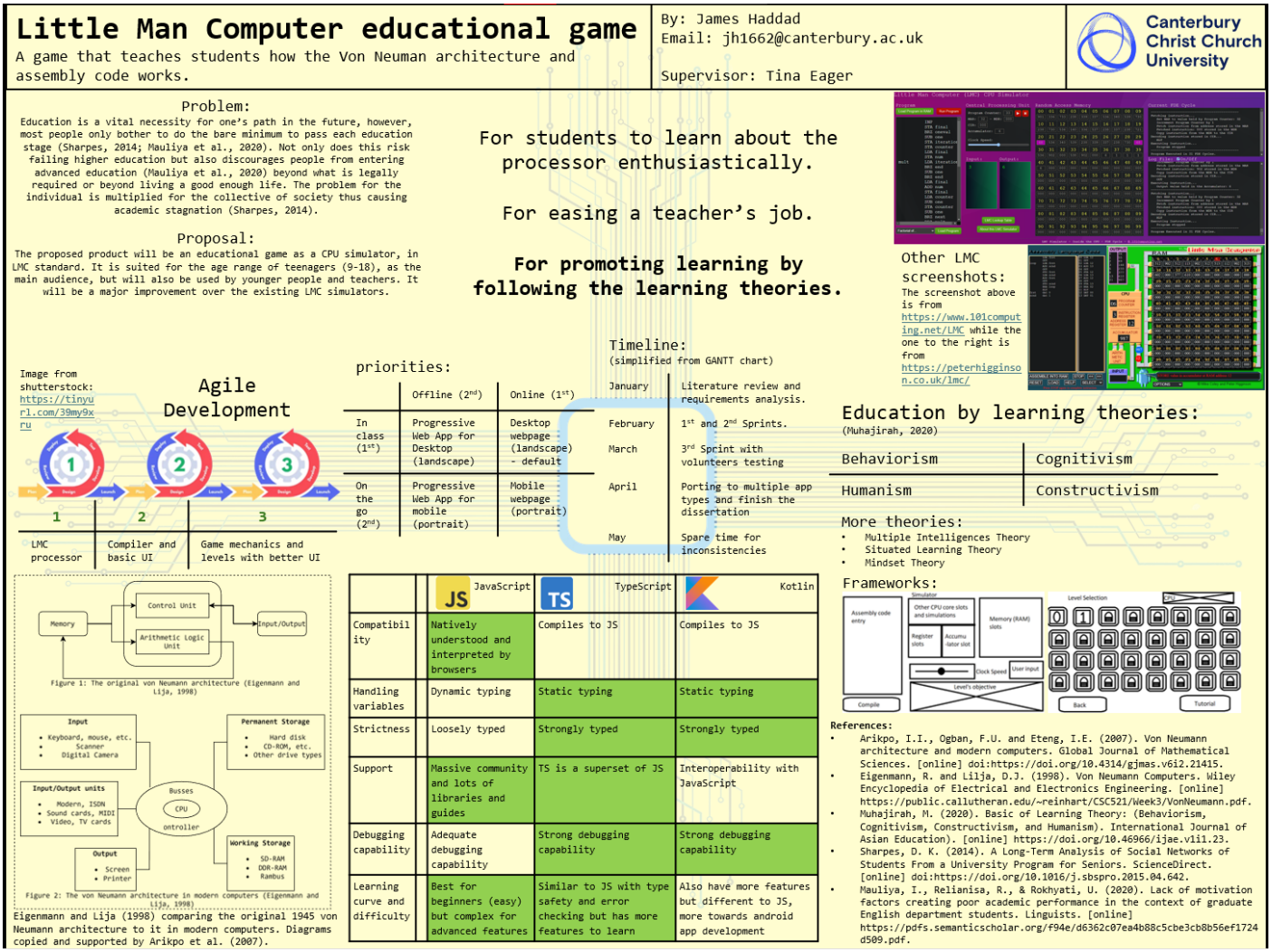


Figure 11.11—8 – poster after the 24th of December 2025 supervisor meeting.

Contents

Figure 11.11—9 - list of contents for the dissertation before 17th December 2025.

11.11.1.2.4 Evidence

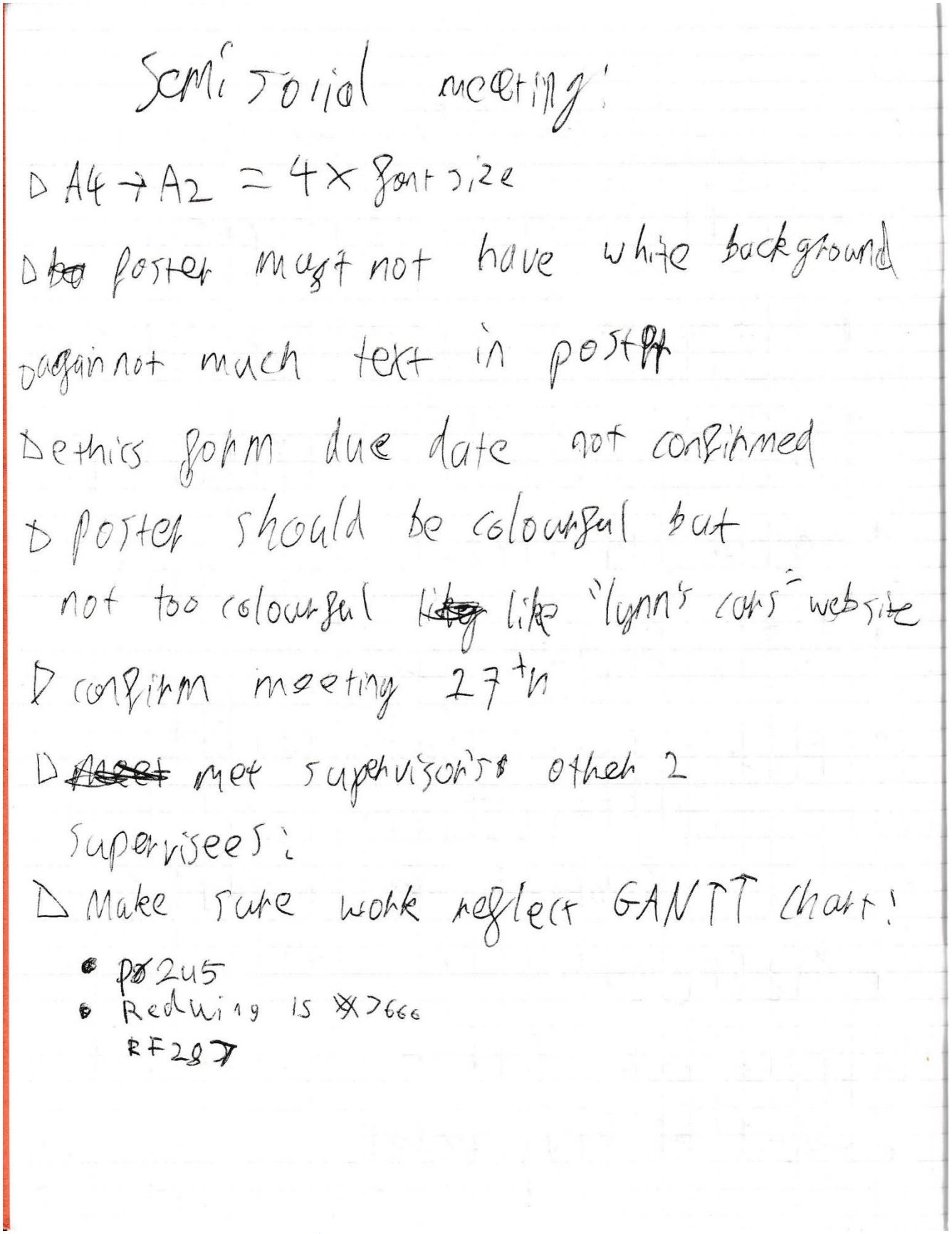


Figure 11.11—10 - page of notes from the 12th December 2024 semi-social meeting.

11.11.1.3 17th December 2025 09:35-10:30

11.11.1.3.1 Purpose and description

To discuss and go over the poster regarding the presentation on the 21st, and the questions raised while doing the dissertation.

11.11.1.3.2 Notes from meeting

On the 17th, we went over two main parts. One was about the final poster (for the presentation on the 20th) in regards what the author will be asked about, what the author needs to explain as an introduction, what the author should know, et cetera. The second part was us discussing the 11 questions the author prepared. the author has written notes about the conclusions/answers of those, so the author does not forget, as seen:

1. Better to have original and modern, for how popular it was, for a long time, and its reliability, with it still being used in modern times, respectively.
2. Can still paraphrase the review but also link back to the first conclusion/answer.
3. N/A – it was just the author telling my supervisor about it.
4. CCCU markers will have similar or the same level of access as the students, so however it was referenced, it must be accessible to the markers as well. Can put both DOI and a different URL, but say which one was used to access the article. Side note from discussion: URLS should lead to the PDFs rather than the metadata webpage.
5. Will probably be a grey area, but will be fine if it was found in Google Scholar. However, must make sure that the article is 100% not fake!
6. Need to screenshot how RefWorks displays the cites, each in Harvard Style, and send to the supervisor for checking.
7. If the reference, in Harvard format, is pasted into Google Scholar and results in only that result (of the article), then it is fine. Although the author still needs to screenshot and send it to the supervisor to be sure. Side note: Format of authors (for example ‘Haddad, J’ or ‘James R. Haddad’) in Harvard reference must depend on what format is used in the article that it is from instances of all capitalisations, of names, from one article and its publisher need to be checked by the supervisor.
8. Should try to put all links/websites as Harvard references. Side note: Should add access dates (does not matter if all the same date) for all non-academic papers and academic papers from not-so-popular publishers to show that it was at least working/accessible at that time.
9. The only questions asked will be to explain what the author put on the poster, why the author did so, and the project plans.
10. No method is objectively better than the other, so it entirely depends on the specific situations.
11. No need to explain academic paper research methodology on this level.

Extra discussed point: Because the author was discussing articles about CSS frameworks, the author should reference them, despite them being just blogs.

11.11.1.3.3 Actions

Continued with the individual project where the author:

- Continued the literature review:
  - Introduced, explained and explored the 4 primary learning theories’ relevancies.
  - Introduced, explained and explored the 3 secondary learning theories’ relevancies.
  - Incorporated the 7 theories’ satisfaction into my product requirements, features, and justification in the “Software Research” part of the literature review.
  - Introduced, compared, and analysed whatever games the author can find where the player plays by coding in assembly code.
  - Moved the project plan after the literature review
  - Moved the part describing the problem (that the project is to solve) to the literature review.
  - Included screenshots of the other LMC simulators that the author is analysing in the “Software Research” part of the literature review. As well as adding a bit more justification for the analysis of those LMC simulators.
  - Added a few notes to tables for more context.
  - Updated references section with more entries
- Done an abstract for the dissertation.
- Continued with the poster creation:
  - Added background colour (creamy colour).
  - Recreated the von Neumann diagrams as SVGs to scale nicely to the poster’s paper size.
  - Added more reference entries to the references part.
  - Added a development methodology part (3 sprint agile development)
  - Added and filled a paragraph for the title “Problem:”
  - Added and filled a paragraph for the title “Proposal:”
  - Added a part that shows the 4 primary and 3 secondary learning theories
  - Added text that describes where the author got the other LMC simulator screenshots from.
  - Added a timeline with entries for each month – taken and simplified from my GANTT chart.
  - Added an intro with 2 sentences and another one in bold. They are short but simple and describe the project’s overall product.
  - Added a priorities table that describes in order of priorities what platform the author will port my products to.
  - Changed slide dimensions from 16:9 to 4:3 so it will fully fit on paper (like A2 paper).

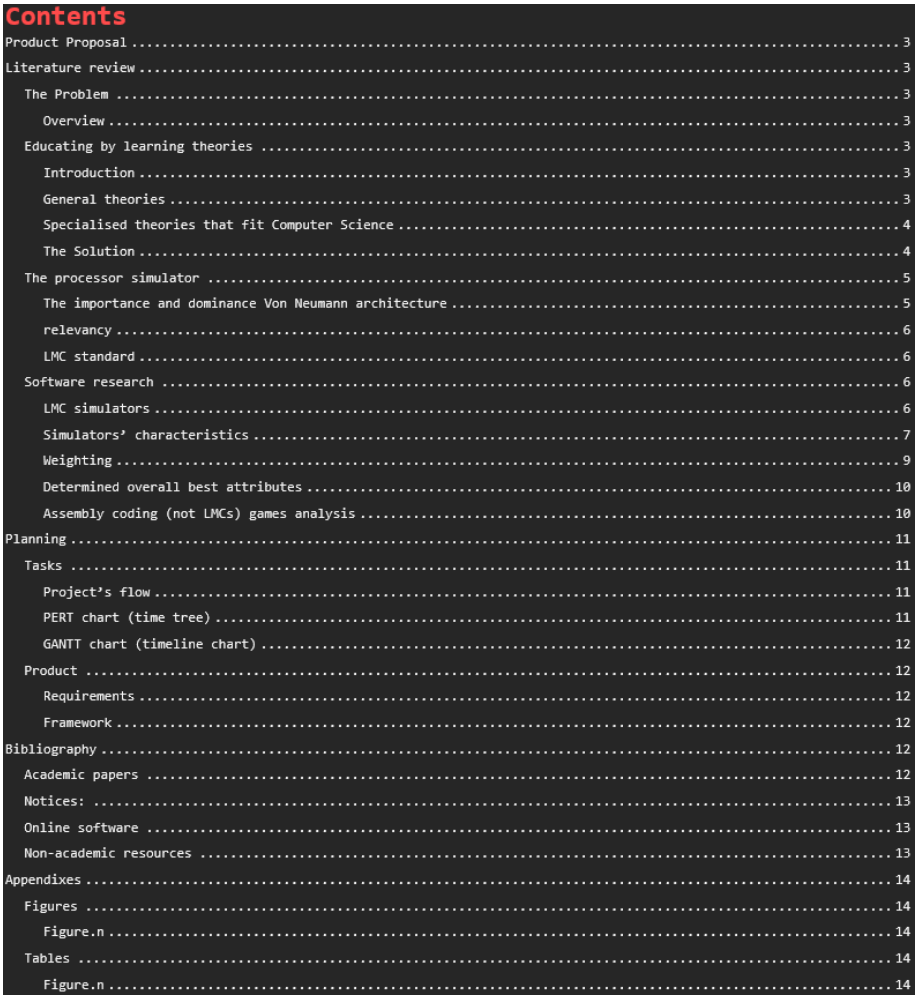
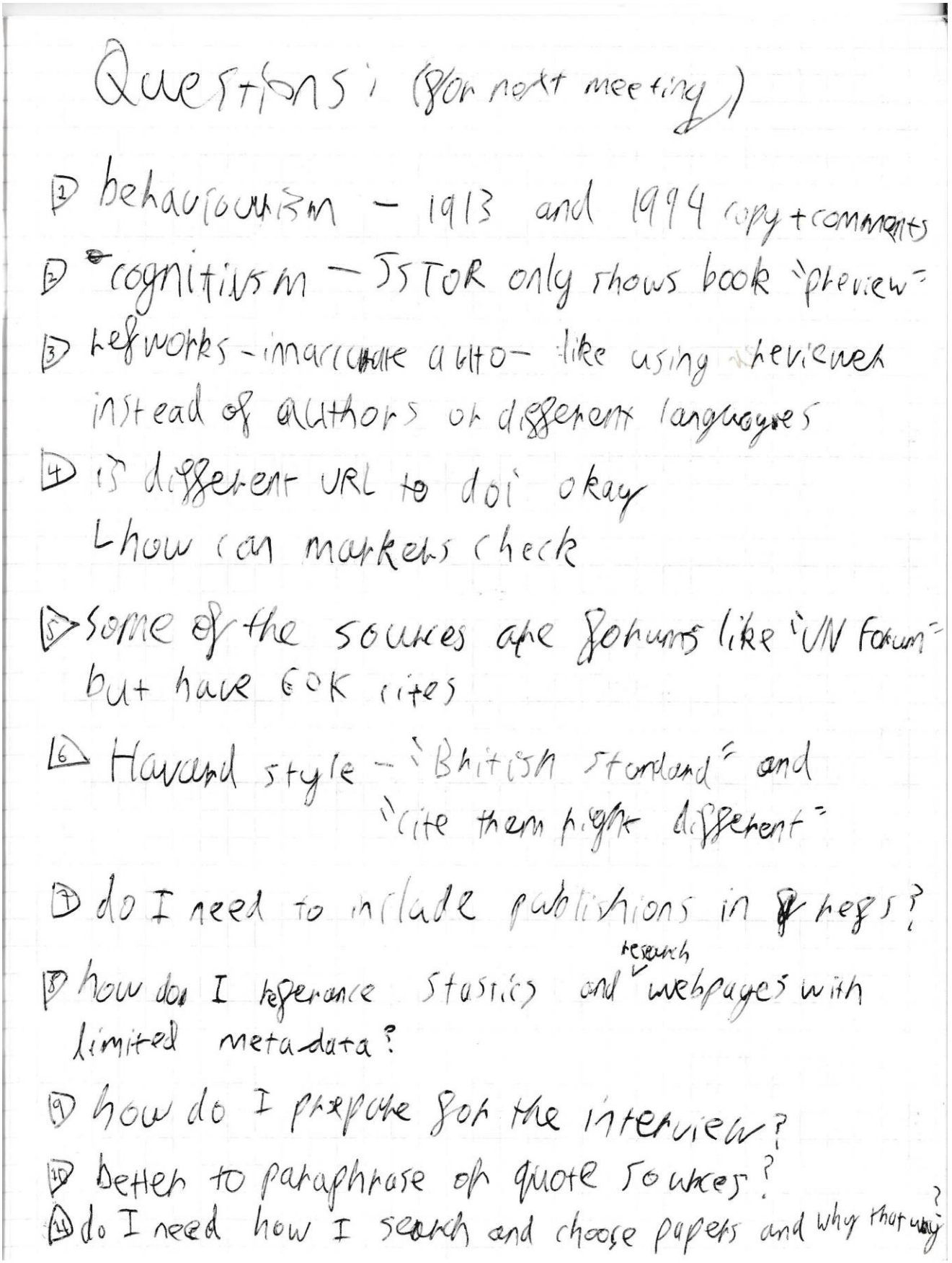


Figure 11.11—11 - list of contents for the dissertation before 20th December 2025.

11.11.1.3.4 Evidence


Questions: (for next meeting)

1) behaviourism - 1913 and 1994 copy + comments
2) "cognitivism - JSTOR only shows book "preview"
3) refworks - inaccurate auto- like using reviewer instead of authors or different languages
4) is different URL to doi okay
   L how can markers check
5) some of the sources are forums like "UN Forum" but have GOK rites
6) Havard style - "British standard" and "cite them right different"
7) do I need to include publishions in refs?
8) how do I reference statics and research webpages with limited metadata?
9) how do I prepare for the interview?
10) better to paraphrase or quote sources?
11) do I need how I search and choose papers and why that way?

Figure 11.11—12 - first page of notes from the 17th December 2024 meeting with the supervisor.

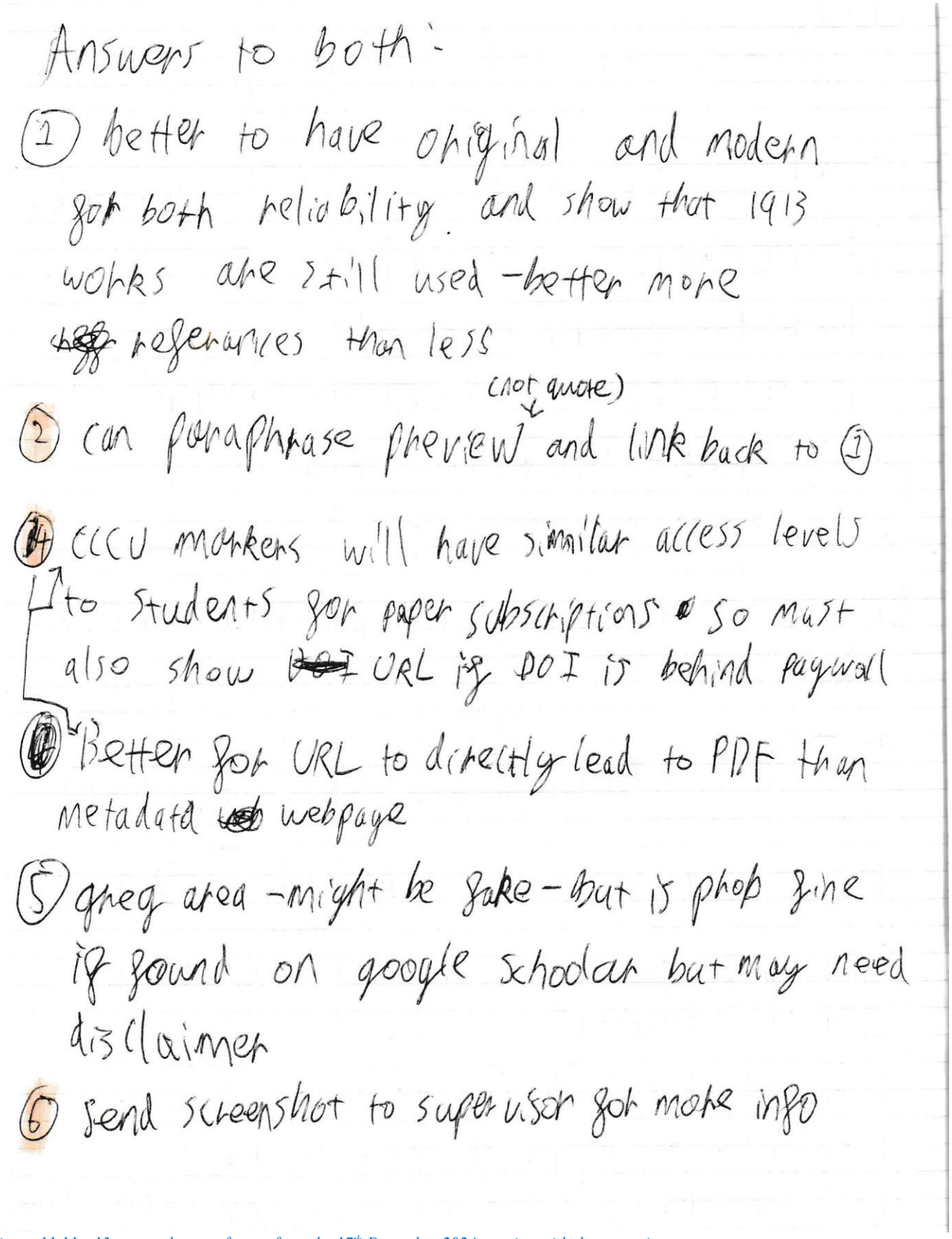


Figure 11.11—13 - second page of notes from the 17th December 2024 meeting with the supervisor.

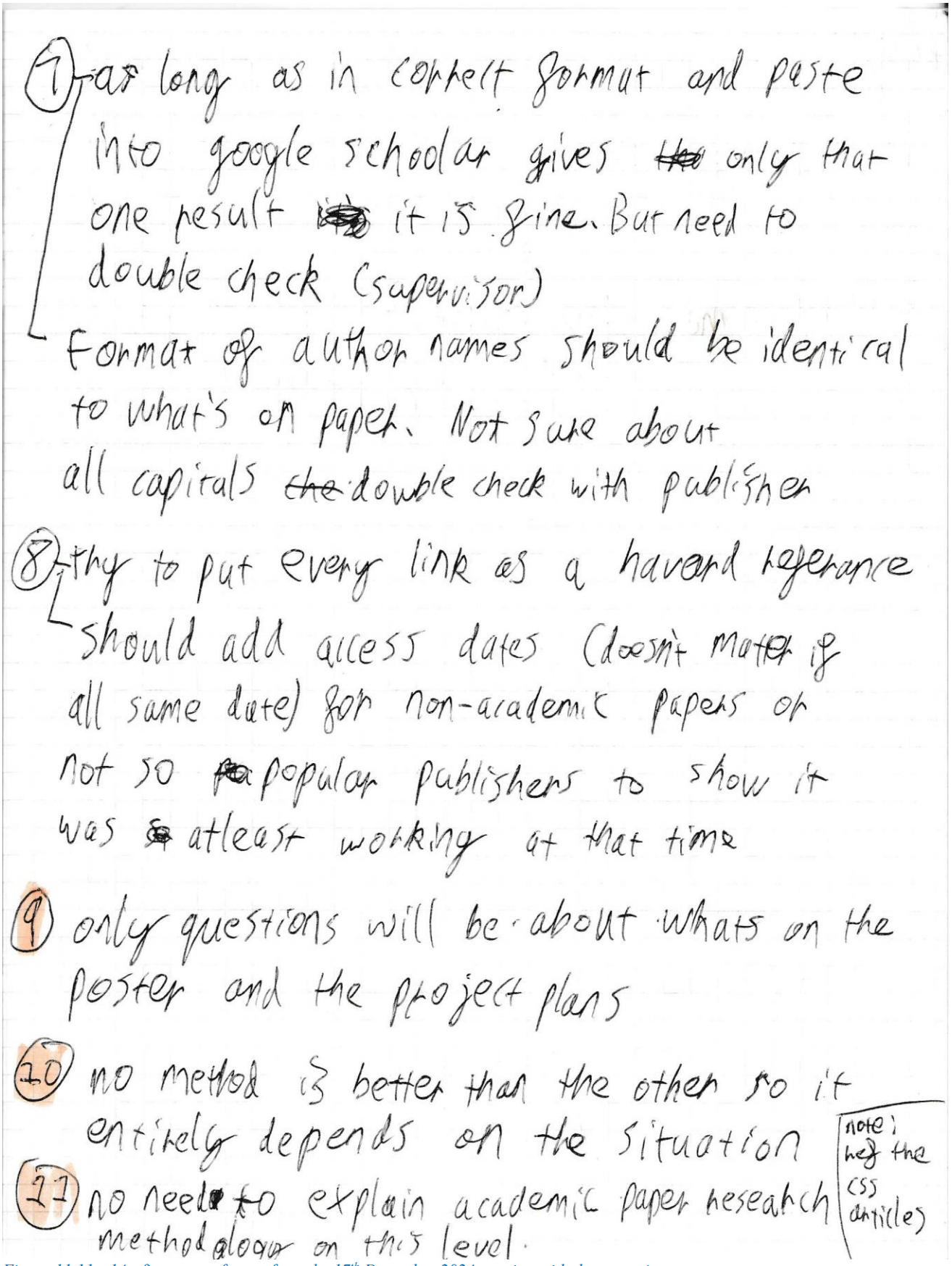


Figure 11.11—14 - first page of notes from the 17th December 2024 meeting with the supervisor.

11.11.1.4 20th December 2025 12:30-13:30

11.11.1.4.1 Purpose and description

To mostly ask about the presentation, such as what the author should work on, what part needs improvement, et cetera.

11.11.1.4.2 Notes from meeting

In the evidence part.

11.11.1.4.3 Actions

As presentation day (21st January 2025) was getting near, the author decided to revise and practice presenting the poster until then.

11.11.1.4.4 Evidence

20th dec 2024

poster intro sentence #1 change → replace enthusiastically with enjoyable or other positive

diss → not only justify why language is used but also why language is not used

poster intro sentence #2 → when asked, explain how alot of CS teachers are underqualified

poster von Neumann → label the diagram to why its there ... actually label everything (von neumann and long table

poster table → remove the empty column, try to get every word not disconnected

poster theories → will definitately be asked about the theories

blackboard → given access to resources about learning theories

poster framework → update it as its from last academic year

Figure 11.11—15 - first page of notes from the 20th December 2024 meeting with the supervisor.

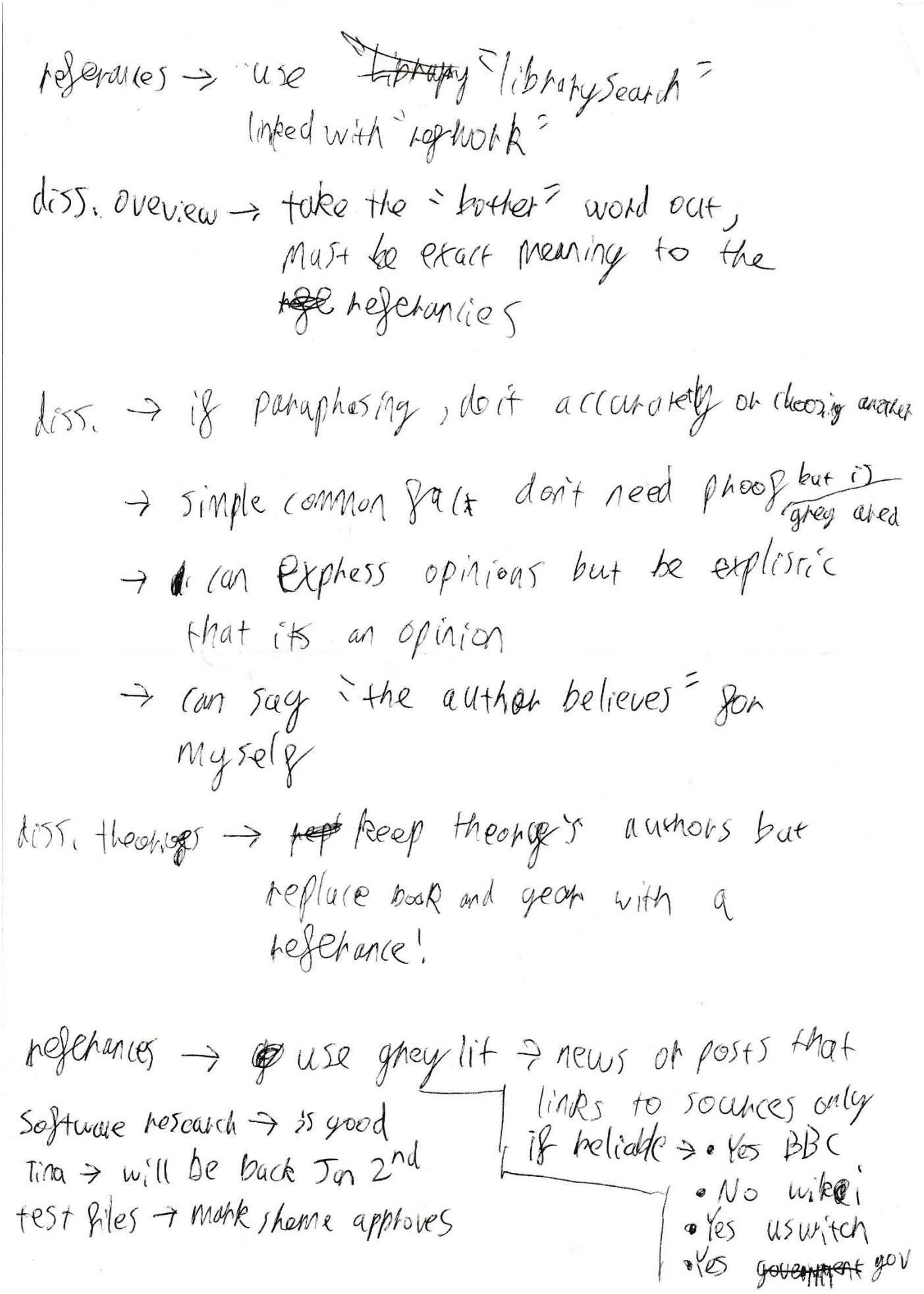


Figure 11.11—16 - second page of notes from the 20th December 2024 meeting with the supervisor.

11.11.2 After project presentation

11.11.2.1 26th January 2025 09:35-10:30

11.11.2.1.1 Purpose and description

Now that the presentation has finished last week, the author wants to go over the current state of the dissertation, going over parts that are good and others that need improvement. To keep things organised and simple, the author brought in a printed version of the dissertation to write over.

11.11.2.1.2 Notes from meeting

In the evidence part as the point of the meeting was to mark and write improvements on the printed current dissertation with suggestions from the discussion between the author and supervisor.

11.11.2.1.3 Actions

Implementation of the notes from the photocopied dissertation.

11.11.2.1.4 Evidence

order: want to do → lit review analysis → program solution

James Haddad (jh1662)	Individual Project - Module U10834
Product Specification Document	2024/25 (Advent Easter)

Project Proposal ← not part of lit rev

Product Description

The proposed product will be an educational game as a CPU simulator, in LMC standard. It is suited for the age range of teenagers (13-18), as the main audience, but will also be used by younger people and teachers. It will be a major improvement over the existing LMC simulators.

The game is based on 2 modes. One will have levels that give the user tasks to proceed while the other is a sandbox where users can advance their skill after completing the levels. Both the levels and sandbox trigger critical thinking and strategies (referring/basing on previous work or using a logical thought pattern) to solve logical problems. Younger teenagers will have these concepts introduced to them smoothly while older teenagers will appreciate and solidify them for more advanced uses.

The Problem ← lit review

Education is a vital necessity for one's path in the future, however, most people only bother to do the bare minimum to pass each education stage (Sharpes, 2014) (Mauliya, Relianisa, & Rokhyati, 2020). Not only does this risk failing higher education but also discourages people from entering advanced education (Mauliya, Relianisa, & Rokhyati, 2020) beyond what is legally required or beyond living a good enough life. The problem for the individual is multiplied for the collective of society thus causing academic stagnation (Sharpes, 2014).

The Solution ← 50/50 lit

Utilising the popularity of gaming

One crucial (and most popular) thing people dedicate their leisure time to is entertainment media (Nakatsu, Tosa, Rauterberg, & Xuan, 2017), and one big part of it is gaming (Nakatsu, Tosa, Rauterberg, & Xuan, 2017) (Possler, Kümpel, & Unkel, 2020). Gaming can vary from relaxing to interactive and intense. Games can be any genre (Sci-fi, fantasy, visual novel, etc.) but the one relevant here is the education genre. Educational games let people educate themselves, to a degree, when happily and willingly spending time and effort playing the game (Possler, Kümpel, & Unkel, 2020). This different approach has been met successfully many times (Sharpes, 2014). So such, that schools use them the encourage their teenage students to learn to the best of their ability (Mauliya, Relianisa, & Rokhyati, 2020), mostly using a media platform at https://www.coolmathgames.com (a very popular website containing hundreds, if not thousands of inspiring educational games).

Game's effect and speciality

This phenomenon has inspired this project to be a simple educational game to help people learn for enjoyment. By being a progression (by levels) and sandbox game, students of wither genre will all be attracted to the game.

As this will be only one game, it has been decided to specialise this game to teach the users programming (specifically assembly programming). This is so as I will use my own experience and skills in programming will be of essential use to make the game without much practice.

Not only it in my expertise but educational programming is very popular, take Scratch for example (https://scratch.mit.edu/) with them claiming "Since 2007, more than 130 million children in every country worldwide have created more than a billion Scratch projects" (https://www.scratchfoundation.org/our-story). This project, however, narrows the programming down to a LMC simulator. LMC simulator is a simplistic Von Neumann architecture CPU, programmed by assembly code(). Students playing this game get to learn how the dominant CPU architecture works and the skills of coding in assembly code.

Plan

PERT chart (time tree)

ideas → research → practical ideas

Figure 11.11—17 - first page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

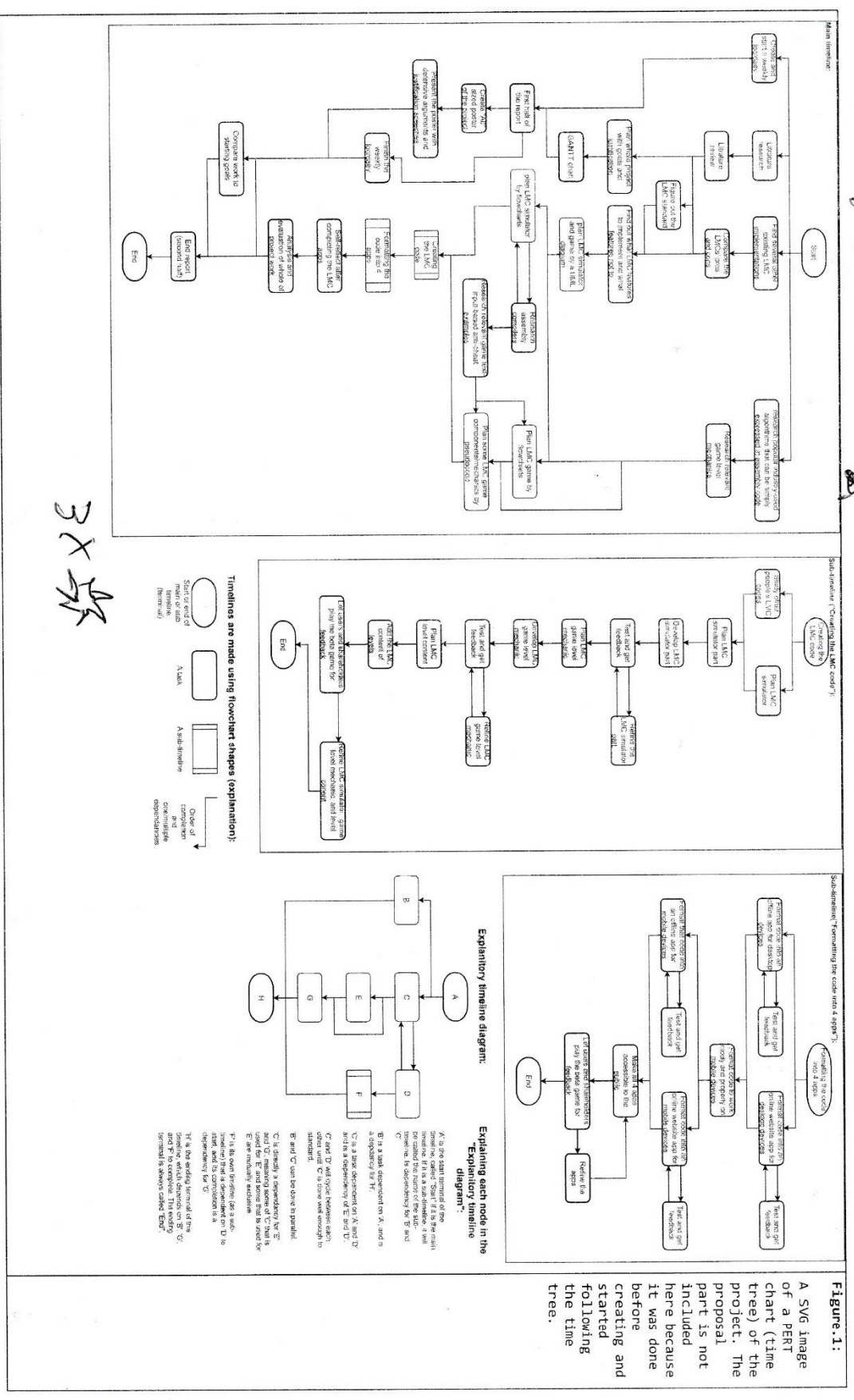


Figure 11.11—18 - second page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

Literature review

Literature research

[note: explain how LMC must simulate this

The dominant Von Neumann architecture

Despite the Von Neumann architecture's immense age, the architecture of all LMC CPUs are implementations of the Von Neumann architecture as seen in figure.1 (Eigenmann and Lija, 1998). This is so because the Von Neumann architecture is the dominant CPU architecture idea (Arikpo et al., 2017; Eigenmann and Lija, 1998). All popular modern CPU architectures (x86, ARM, RISC-V, etc.) are all fundamentally based on the Von Neumann architecture. It beats other proposed CPU architecture ideas, the popular beaten one being the Harvard architecture. Harvard architecture differs by have a discrete data memory and instruction memory (Eigenmann and Lija, 1998), whereas Von Neumann has them unified as one memory (Arikpo et al., 2017; Eigenmann and Lija, 1998)(also known as RAM). Harvard architecture excels at simultaneous performance by not using a shared bus for both memories and security as the data memory can be manipulated without changing the instruction memory. Despite this Harvard easily falls out of favour as its increased complexity severely makes its designs much more difficult to implement and being far more expensive to manufacture. Von Neumann preference over Harvard architecture is amplified by Harvard architecture's performance advantage, of faster communication from having 2 buses, by simply increasing the buses' bandwidth in Von Neumann architecture as mentioned by Eigenmann and Lija (1998).

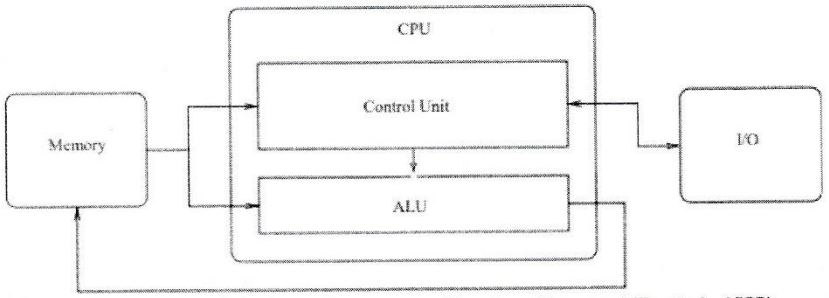


Figure 1: The original von Neumann architecture (Source: Lilja et al., 1998)

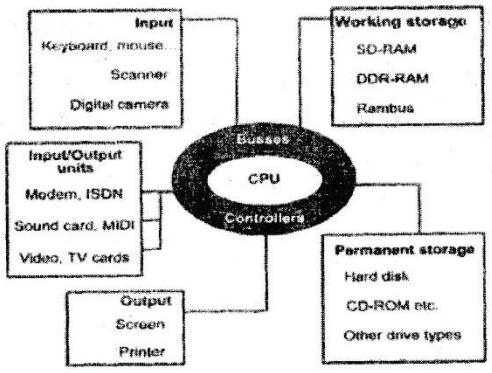


Figure 2: The von Neumann architecture in modern computers (Source: Lilja et al., 1998)

Figure.2:

Lija et al. (1998) comparing the original 1945 Von Neumann architecture to it in modern computers. Diagrams copied and supported by Arikpo et al. (2007).

CPU simulator

Assembly code

LMC academic articles

LMC

↳ justify effectiveness of postman

Software research

LMC simulators

As the proposed project is an improvement/upgrade of existing software, software research is im

CPU simulators based on Little Man Computers (LMC) in **table.1:**

Figure 11.11—19 - third page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

• include screen shots of other LMC

Links (for simulator)	Reference number	Introduction page	Source code	Introduction has manual?
https://peterhigginson.co.uk/lmc/	1	https://www.peterhigginson.co.uk/lmc/help_new.html	https://github.com/FrBrGeorge/peterhigginson/tree/main/peterhigginson.co.uk/LMC	Yes
https://www.101computing.net/LMC/	2	https://www.101computing.net/lmc-simulator/	Not found	Yes
https://pbrinkmeier.github.io/lmc-emulator/	3	https://github.com/pbrinkmeier/lmc-emulator/blob/master/README.md	https://github.com/pbrinkmeier/lmc-emulator/	Yes
https://wellingborough.github.io/LMC/LMC0.3.html	4	https://wellingborough.github.io/LMC/	https://github.com/Wellingborough/LMC?tab=GPL-3.0-1-ov-file	Yes
https://trincot.github.io/lmc.html	5	https://trincot.github.io/	https://github.com/trincot/lmc	No
https://blog.paulhankin.net/lmc/lmc.html	6	https://blog.paulhankin.net/littlemancomputer/	Not found	No

Notes:

- LMCs 1 and 2 are far more popular and regularly used for education purposes while the others are more like side projects on GitHub.
- LMCs 1,2,3, and 4 all refer to the Wikipedia page of LMC as a source of further information and learning about LMC.
- Points like the previous 2 bullet points will be account for the weighting of which LMC is more reliable and hence preferred.
- Other CPU simulator that don't base themselves on LMC principles will be ignored such as the AQA exam board CPU simulator aligned to the exam board's syllabus (https://www.peterhigginson.co.uk/AQA/) or based more complicated architecture (https://peterhigginson.co.uk/RISC/V2.php)
- LMC #5's introduction page (https://trincot.github.io/) shows the differences between LMC 1,2 and 5. These will be used as a reference but different when used. For example, testing the input range (and valid contents like decimal and letters) in LMC #1:

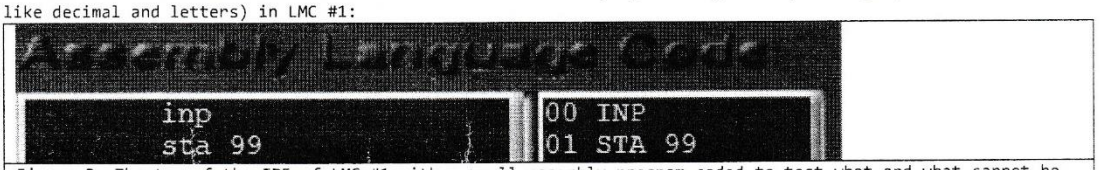


Figure.3: The top of the IDE of LMC #1 with a small assembly program coded to test what and what cannot be accept as input.

- While it may be possible to get the source code of all simulators by inspecting any JS file received by the browser, we do not know if the authors permit the use of their code that way. Ignoring author's licensees may potentially lead to legal issues.

LMC attributes in **table.2:** Note, not everything in manual so have to test but may not be 100% accurate

LMC #	RAM size	Valid inputs	Instruction set as mnemonics	Base	Shown registers	Compiler
1	100	integers -999 to 999	HLT, ADD, SUB, STA/STO*, LDA, BRA, BRZ, BRP, INP, OUT, OCT, DAT *'STA' and 'STO' are the same instruction – interchangeable **'OCT'	10 – decimal	PC, MIR, MAR, Accumulator	Detects syntax errors and prevents compilation if so. Uses double slashes (//) to initiate comments. Explains error when detected, for example "unknown instruction at line 3 LDO"
2	100	integers -999 to 999 and "NaN' for invalid inputs. However, text can be stored (as an address) if the STA instruction is called	INP, OUT, LDA, STA, ADD, SUB, BRP, BRZ, BRA, HLT, DAT	10 - decimal	PC, MIR, MAR, CIR, Accumulator	Detects syntax errors and compiles as if error lines never existed.
3	100	integers -999 to 999	ADD, SUB STA, LDA, BRA, BRZ, BRP**, INP, OUT, COB*	10 - decimal	PC, Accumulator,	Detects syntax error and prevents execution but does not say the error's location.

Figure 11.11—20 - fourth page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

			*'COB' and 'HLT' does the same thing. ** 'BRP' here branches if carry register is '1' instead of the accumulator being positive.		Carry, Inbox (input), Outbox (output),	Instructions must be uppercase and labels in lowercase.
4	100	integers -999 to 999, input box doesn't allow other characters to be inputted	LDA, STA, ADD, SUB, INP, OUT, BRA, BRP, BRZ, HLT, DAT	10 - decimal	PC, MAR, MDR*, CIR *'MDR' stores whatever is stored to or retrieved from the RAM	Far simpler, and hence easier, as each line is split into 3 text boxes (labelled as "Label", "Operator", and "Operand"). Detects syntax errors and prevents compilation if so. Explains error when detected, for example "Error, line 0: machine code instruction inp should not have an operand".
5	?	integers 0 to 999	?	10 - decimal	Accumulator, Negative	Minimal UI and lack of manual makes it very difficult to use. Uses the semi-colon symbol (;) to initiate comments.
6	100	integers 0 to 999,	?	10 - decimal	Accumulator, PC, Neg, Ins* *'Ins' appears to be counting how many "clock cycles" passed since execution (increases indefinitely after an error)	Small width to bigger height ratio fits the LMC assembly language well (having a max of 3 syntax per line). Detect only some errors before compilation. Doesn't detect errors in inputs despite being compulsorily entered before compilation.

LMC weighting calculations in **table.3:**

	LMC #1	LMC #2	LMC #3	LMC #4	LMC #5	LMC #6
Weight based on manual/ introduction (0-5)	4 - Explains instructions and registers. LMC has multiple assembly example programs, but not in the manual.	5 - Explains instructions, registers, and multiple assembly example programs.	4 - Explains instructions, registers, and an example assembly program.	4 - Explains instructions, registers, an example assembly program.	2 - Instead of a manual, it has a feature comparison table to other LMCs and has multiple example assembly programs.	0 - A blog that doesn't explain much with the most informative being the LMC Wikipedia page but has multiple example assembly programs.
Weight based on Source code availability (0 or 5)	5	0	5	5	5	0
Weight based on how many attributes found in table.2 (0-6)	6	6	6	6	4	5
Weight based on author/company's background, such as: other projects, experience, and relevancy (0-5)	5 - Author made 3 other CPU simulators ([1,2,3]) and assisted with a book at [4]	4 - Company makes computer science books and lessons for GCSE [5] and A-level [6] students. So LMC must be made with the students in mind.	1 - Only thing of note is the developer's contribution to a "compiler for a Java subset" at [7]. Not much experience nor relevancy.	3 - Company is literally a private school for children 3-18 years old ([8]). So LMC must be made with the students in mind.	0 - developer's only other GitHub repository [9] is a HTML page of the a table of comparison between developer's	2 - developer's blogs ([11]) explores mostly mathematical concepts using programming. The much mathematical experience is

Figure 11.11—21 - fifth page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

					LMC and LMC #1 and #2 [10].	only slightly relevant.
Total	20	14	16	18	12	7
%						

Links for table.3:

1. https://www.peterhigginson.co.uk/RISC/
2. https://www.peterhigginson.co.uk/ARMlite/
3. https://www.peterhigginson.co.uk/AQA/
4. https://metalup.org/
5. https://www.101computing.net/ocr-j277-computer-science-gcse/
6. https://www.101computing.net/ocr-h446-computer-science-a-level/
7. https://github.com/Firmwehr/gentle
8. https://www.wellingboroughschool.org
9. https://github.com/trincot/trincot.github.io
10. https://trincot.github.io
11. https://blog.paulhankin.net

Note: '?' means that the subjected LMC program does not enough information to be worth mentioning.

Note: the term "label" refers to establishing a refer points for looping in the assembly program.

Overall attributes based on weights of the LMCs and unique characteristics from one LMC where is blank for others (exclusive characteristics explained later) in **table.4:**

	RAM size	Valid inputs	Instruction set as mnemonics	Base	Shown registers	Compiler
Overall (based on weighting)	100	integers - 999 to 999 and 'NaN' if invalid value	HLT, ADD, SUB, STA, LDA, BRA, BRZ, BRP, INP, OUT, OCT, DAT	10 - decimal	PC, MIR, MAR, Accumulator	*Due to relative sizing, compiler traits are instead in the "Compiler traits:" list!*

Compiler traits:

- Detects syntax errors (pointing out the error location), explains its nature, and prevents compilation if so.
- 3 text boxes per line (for the parts: label, operator, and operand). The right-hand-side textbox can be selected by click or the tab key/button and the left text box of next line can be selected by click or the enter key/button.
- Uses double slashes (//) to initiate comments.
- Entire compiler is not case sensitive.
- Minimal UI.

Unique characteristics against weighting and their reasoning:

- 'NaN' value - User should learn how to detect error instead of being spoon-fed to get them used to and potentially prepare them for logic-errors.
- OCT instruction - Barely changes the core functionality of LMC as it CPU simulator games as it only changes the output from integer to any ASCII character. However, it also opens a lot more imaginative programs in sandbox and be used for making a lot more levels; exponentially expanding students' creativity and practicing critical thinking respectfully.
- Comments done by double slashes - used in popular languages, such as JS, and for more accessible then other widely used comment notation such as the hashtag (#) for comments in Python.
- 3 text boxes per line - Lets the student understand the structure of assembly code quickly enough to not lose interest before understanding how to code in assembly code and get on with the level's content.
- Non-case sensitive and minimal UI - helps with students with small screen sizes such as a small laptop or even a smartphone.

While there isn't any games involving LMC simulators there are some that simulate a variation of the CPU architecture through various ways.

Bibliography

Academic papers

Arikpo, I.I., Ogban, F.U. and Eteng, I.E. (2007). Von Neumann architecture and modern computers. Global Journal of Mathematical Sciences. [online] AJOL doi:https://doi.org/10.4314/gjmas.v6i2.21415.

Eigenmann, R. and Lilja, D.J. (1999). Von Neumann Computers. Wiley Encyclopedia of Electrical and Electronics Engineering. [online] Wiley doi:https://doi.org/10.1002/047134608x.w1704.

Bogner, J. and Merkel, M. (2022). To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub. Institute of Electrical and Electronics Engineers. [online] IEEE Xplore. doi:https://doi.org/10.1145/3524842.3528454.

Figure 11.11—22 - sixth page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

Zhdanov, V. (2023). Methodology and application of full-stack software production improvement. Electronic Institutional Repository of the National Aviation University of Ukraine. [online] erNAU https://er.nau.edu.ua/bitstream/NAU/63360/1/%d0%a4%d0%9a%d0%9f%d0%86_2023_221%d0%b0_%d0%96%d0%b4%d0%b0%d0%bd%d0%be%d0%b2%20%d0%92.%d0%9e..pdf.

Yurcik, W. and Osborne, H. (2001). A crowd of Little Man Computers: visual computer simulator teaching tools. Institute of Electrical and Electronics Engineers. [online] IEEE Xplore. doi:https://doi.org/10.1109/WSC.2001.977496.

Osborne, H. and Yurcik, W. (2003). The educational range of visual simulations of the little man computer architecture paradigm. Institute of Electrical and Electronics Engineers. [online] IEEE Xplore. doi:https://doi.org/10.1109/FIE.2002.1158742.

Yurcik, W. and Brumbaugh, L. (2001). A web-based little man computer simulator. Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education. Reserch Gate. [online] Research Gate doi:https://doi.org/10.1145/364447.364585.

Hasan, R. and Mahmood, S. (2012). Survey and evaluation of simulators suitable for teaching for computer architecture and organization Supporting undergraduate students at Sir Syed University of Engineering & Technology. Institute of Electrical and Electronics Engineers. [online] IEEE xplore doi:https://doi.org/10.1109/control.2012.6334776.

Nikolic, B., Radivojevic, Z., Djordjevic, J. and Milutinovic, V. (2009). A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization. Institute of Electrical and Electronics Engineers. [online] IEEE xplore doi:https://doi.org/10.1109/te.2008.930097.

Liang, B., Wang, T., Bai, X. and Zhao, H. (2023). Sparrow: A Teaching CPU Simulator Based on Windows with Graphical User Interface. Institute of Electrical and Electronics Engineers. [online] IEEE xplore doi:https://doi.org/10.1109/iscsic60498.2023.00084.

Jensen, S., Møller, A., & Thiemann, P. (2009). Type analysis for JavaScript. Static Analysis: 16th International Symposium, 238-255.

Mauliya, I., Relianisa, R., & Rokhyati, U. (2020). Lack of motivation factors creating poor academic performance in the context of graduate English department students. Linguists: Journal Of Linguistics and Language Teaching, 73-85.

Nakatsu, R., Tosa, N., Rauterberg, M., & Xuan, W. (2017). Entertainment, culture, and media art. Handbook of Digital Games and Entertainment Technologies, vol 1 pages 725-776.

Possler, D., Kümpel, A., & Unkel, J. (2020). Entertainment motivations and gaming-specific gratifications as antecedents of digital game enjoyment and appreciation. Psychology of Popular Media, 541-580.

Richards, G., Lebresne, S., Burg, B., & Vitek, J. (2010). An Analysis of the Dynamic Behavior of JavaScript Programs. Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, 1-12.

Sharpes, D. K. (2014). A Long-Term Analysis of Social Networks of Students From a University Program for Seniors. American Economic Decline: A Consequence of Lower Educational Standards, 56-62.

Online software

Appendixes
Figures
Tables

DOIs are great

either put links in to relevent content or both in content and bibliography

+ language justification

may be able to make appointment on ~~17th~~, ~~18th~~, ~~19th~~

~~17th~~ 17th (afternoon) afternoon

good mark scheme: agile spreadsheet of thesis marking scheme in thesis subject

Figure 11.11—23 – seventh page of dissertation annotations from the 26th January 2025 meeting with the supervisor.

11.11.2.2 19th February 2025 13:05-14:00

11.11.2.2.1 Purpose and description

Now that it has been a meeting since the presentation is over, the author booked another meeting with his supervisor regarding how to now proceed with the dissertation now after the first post-presentation meeting.

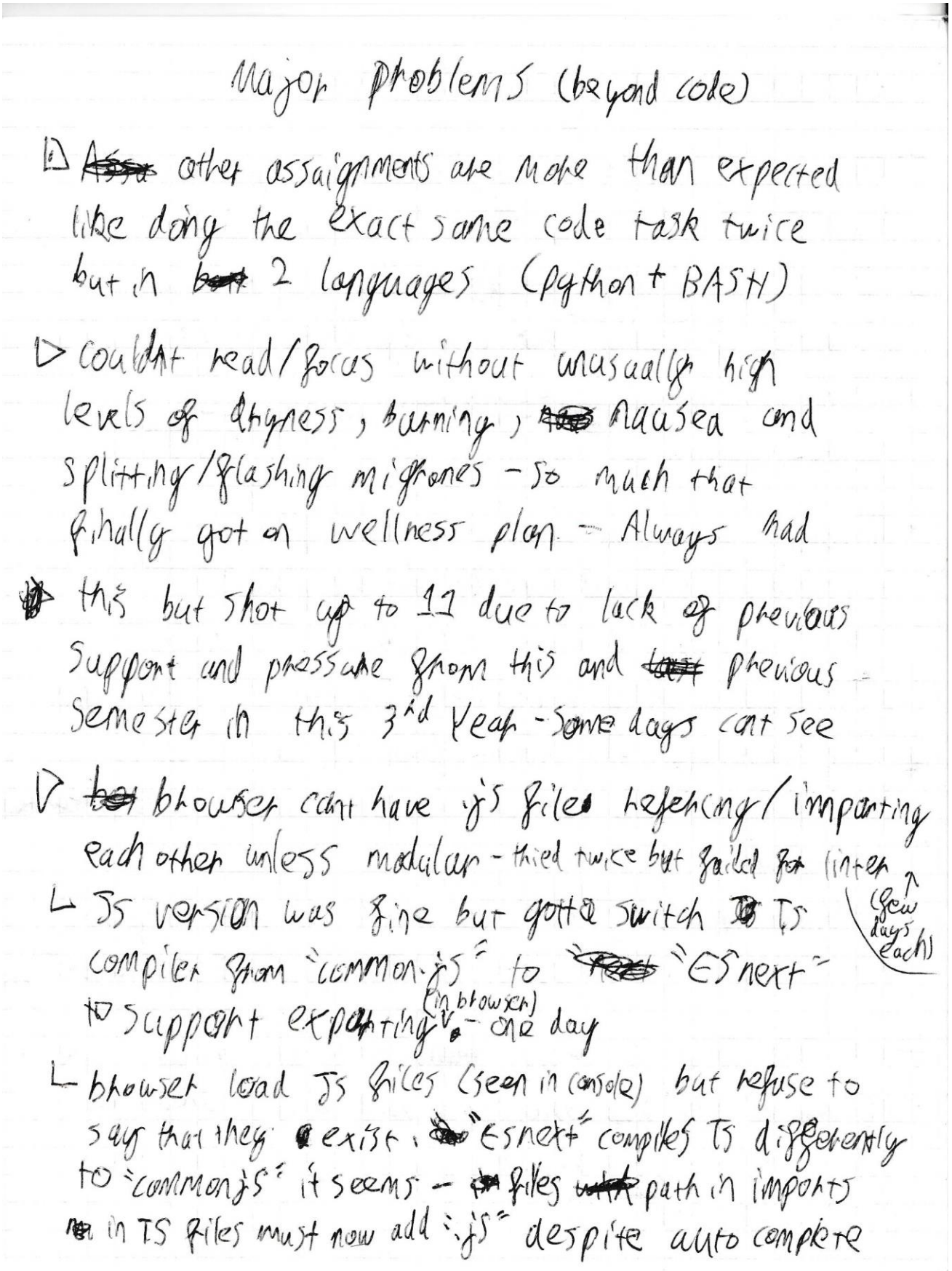
11.11.2.2.2 Notes from meeting

Had a meeting with the supervisor (19th Feb) and discussed and concluded the following:

- Doing implementation, dissertation, and log at the same time causes stagnation – just, for now, put informal notes in logs, while doing the project/product development.
- If an academic paper is only available on Google and not Google Scholar, it is okay to include it if those kinds of academic papers are only a small proportion of the bibliography.
- Some instances of unprofessionalism in the dissertation due to using certain words too common.
- It is very good that the author noticed the mission LMC instruction and its related research and justification.
- The author has kind of done his "critical path analysis" but needs to explicitly mention it and link back to it in other sections.
- When the author is not 100% certain about something, the author must say so explicitly, such as "assume as".
- It is better/safer to use ".HTA" instead of the Electron framework because it is quicker and easier, and the markers will mark the assignment using a Windows PC. Only use Electron if the author has extra time to do it as a lower priority.
- Dissertation has a word limit of 8500, but close to the due date, so author can just move a bunch of different parts of the dissertation can be moved to the appendix because text in the appendix does not count.
- Going over the vague mark scheme:
    - Explain the sprint methodology and how it leads to my final goal. Be clear and obvious about it and link it with other parts of the project.
    - Creativity and uniqueness – comprehensive, thoughtful, and main audience testers.
    - Already good at analysis – just do reflections.
    - System designs – plans of designs and show that the author knows what the author is doing.
    - Understanding – introductions and abstract.
    - Code quality – best practice does not mean commenting on every line of code.
    - Unit/system testing – Just continue with the Jest testing, but also use client/audience testing.
- Tina (my supervisor) mentions that LMC is used in the university's year 1's introduction to computer hardware and mathematics.
- Need to show understanding and implementation of satisfying HCI principles.
- Should book 2 more meetings:
    - Middle of March
    - At least a few days before Easter holidays

11.11.2.2.3 Actions

Proceeded to implement the notes from the meeting, but also had problems with TS compiling, which needed a lot of attention, as seen by the notes below.



Major problems (beyond code)

- other assaignments are more than expected like doing the exact same code task twice but in 2 languages (Python + BASH)
- couldn't read/focus without unusually high levels of dryness, burning, nausea and splitting/flashing migranes - so much that finally got on wellness plan - Always had this but shot up to 11 due to lack of previous support and pressure from this and previous semester in this 3rd year - some days can't see
- browser can't have .js files referencing/importing each other unless modular - tried twice but failed for (interesting ↑ few days each)
  - JS version was fine but gotta switch TS compiler from "common.js" to "ESnext" to support exporting (in browser) - one day
  - browser load JS files (seen in console) but refuse to say that they exist. "ESnext" compiles TS differently to "commonJS" it seems - files path in imports in TS files must now add ".js" despite auto complete

Figure 11.11—24 - first page of notes after the 19th February 2025 meeting.

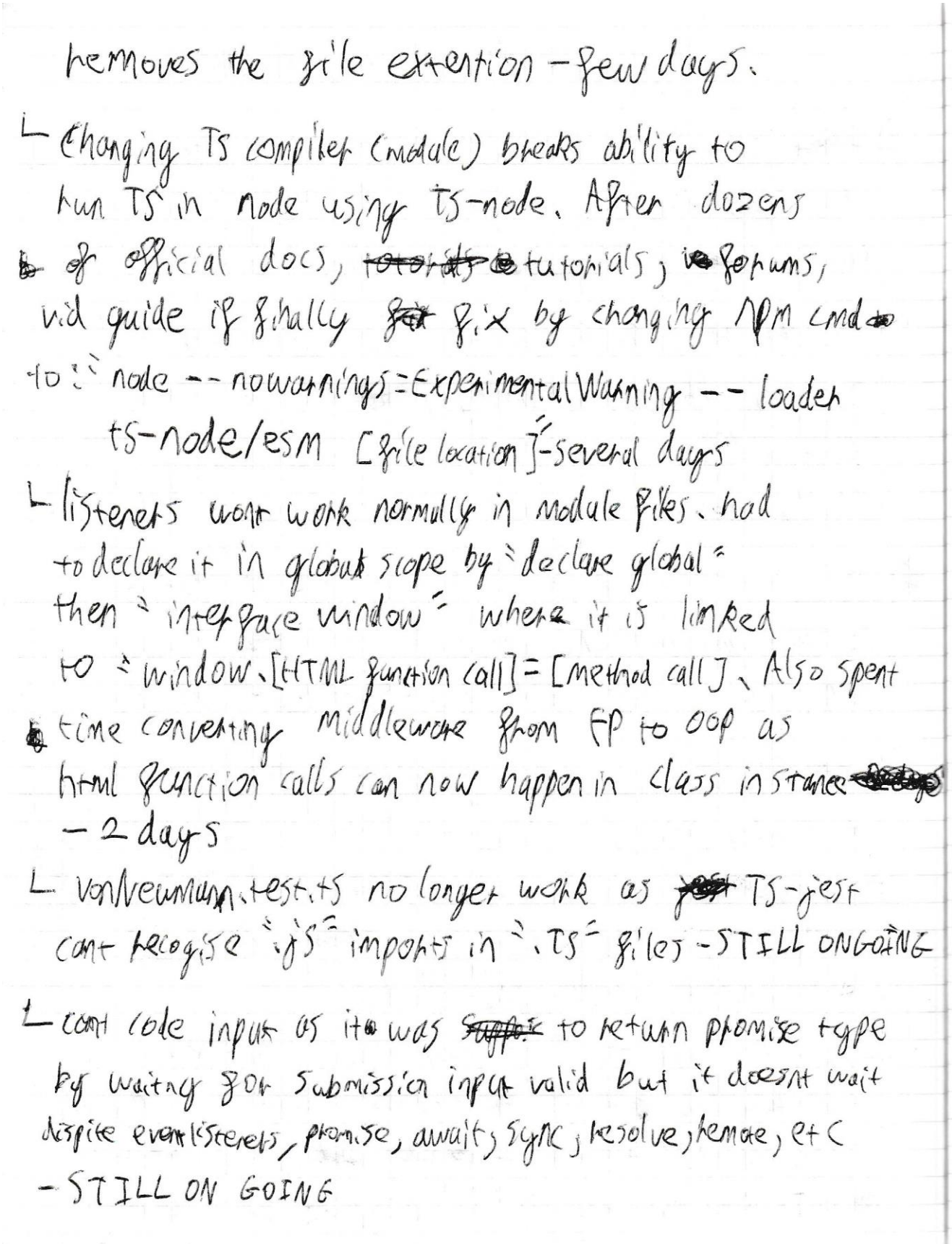


Figure 11.11—25 - second page of notes after the 19th February 2025 meeting.

11.11.2.2.4 Evidence

probs/Questions for next session:

① high-fi: generates are either limited by paywalls or limited styles

② markscheme: very vague - splits into 5% bits regarding very general things

④

Figure 11.11—26 - first page of notes from the 19th February 2025 meeting with the supervisor.

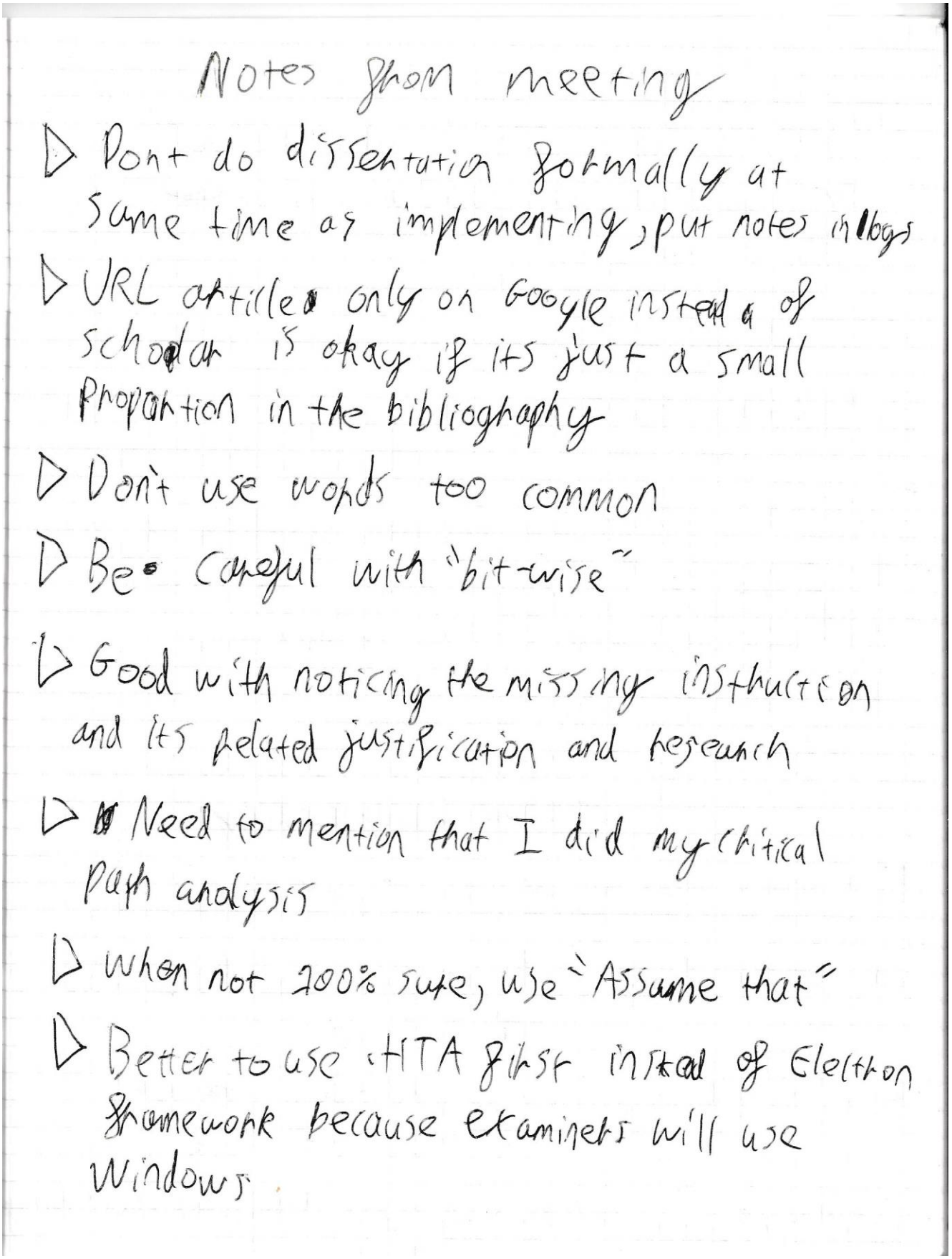


Figure 11.11—27 - second page of notes from the 19th February 2025 meeting with the supervisor.

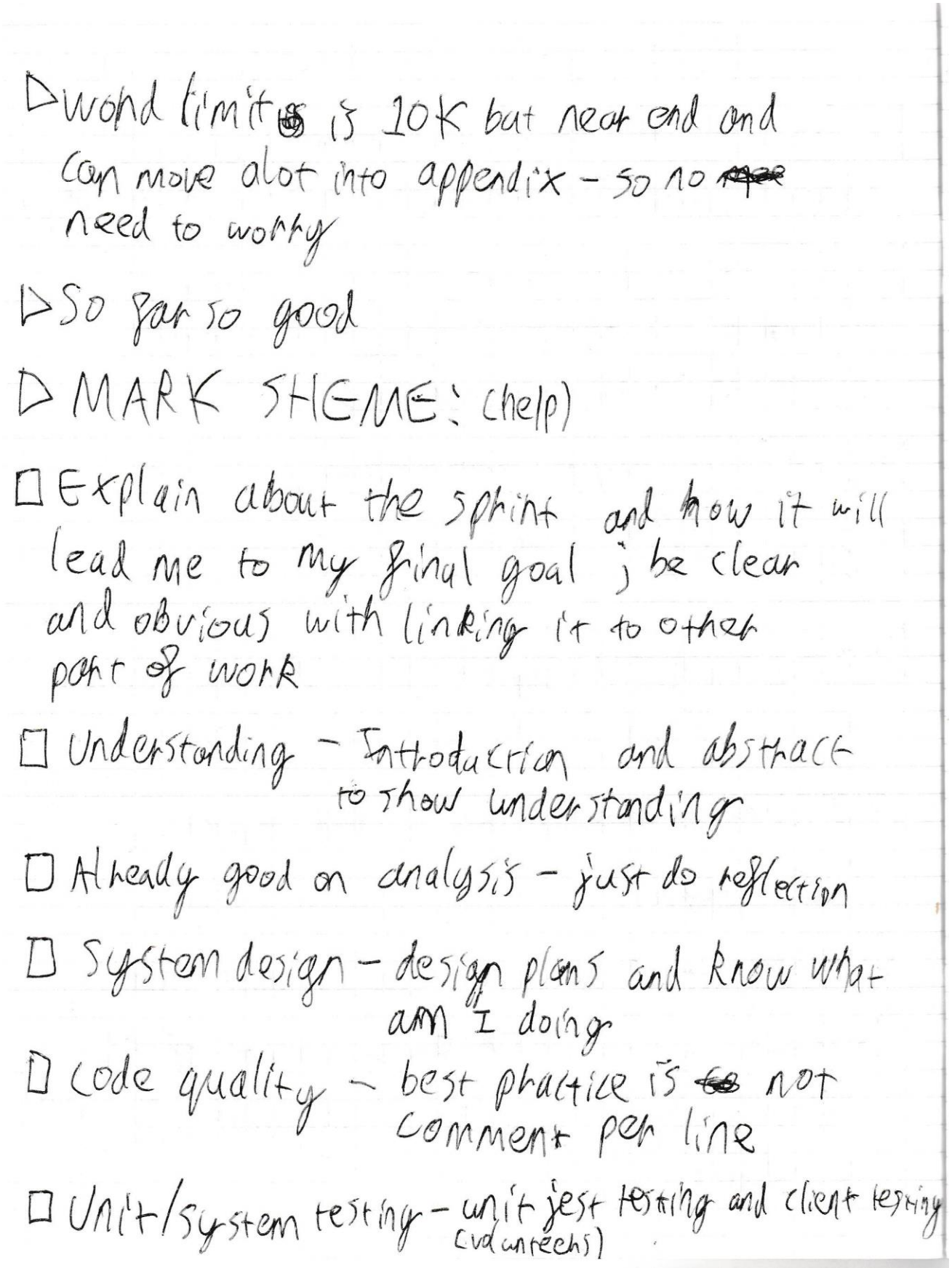


Figure 11.11—28 - third page of notes from the 19th February 2025 meeting with the supervisor.

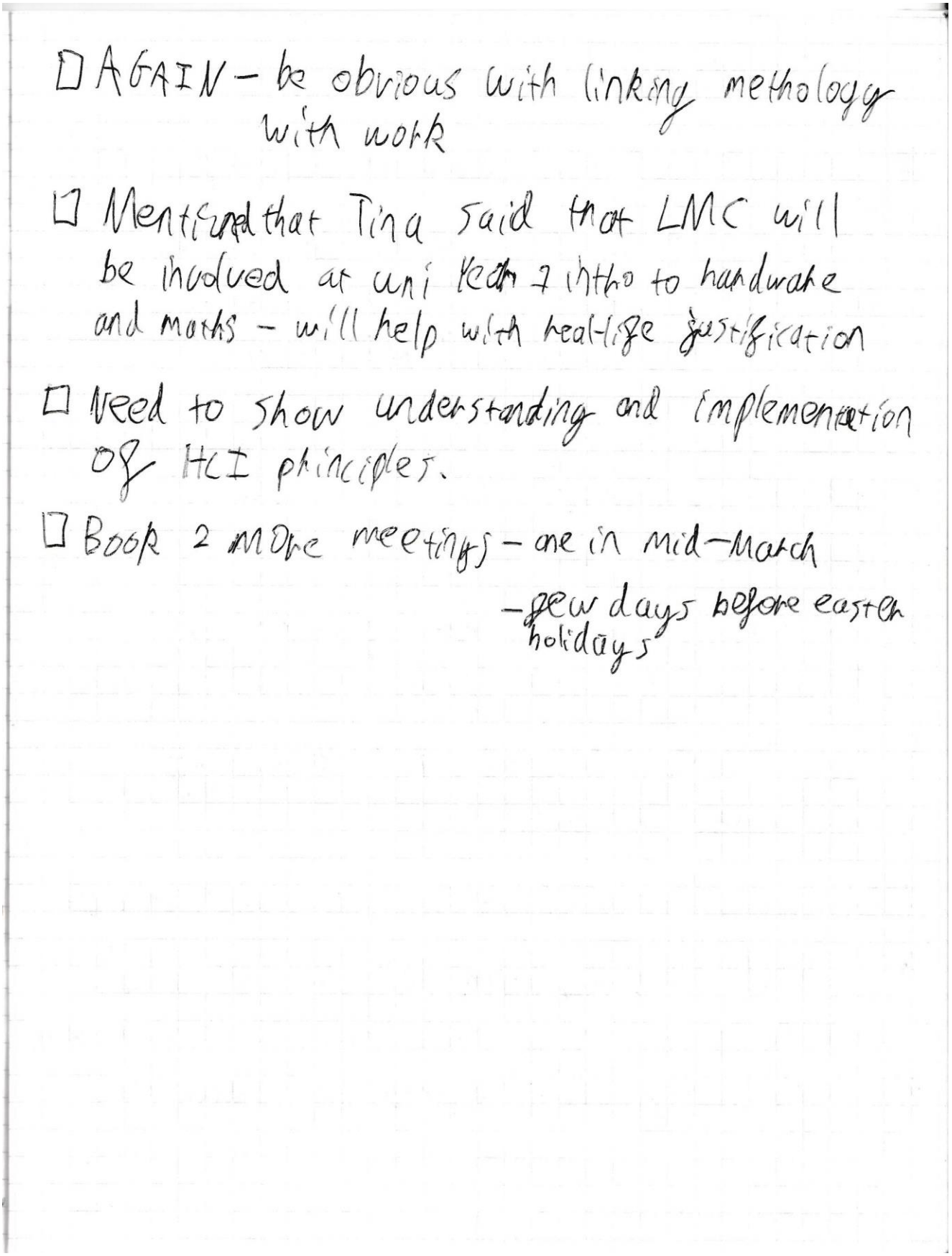


Figure 11.11—29 - fourth page of notes from the 19th February 2025 meeting with the supervisor.

11.11.2.3 28th March 2025 9:35-10:45

11.11.2.3.1 Purpose and description

Go over what to put in each chapter and appendix of the template of the dissertation.

11.11.2.3.2 Notes from meeting

28th March 9:35 - 10:45

Core chapters:
Abstract <300 words
Acknowledgements - Supervisor, Mr Jacob, any other testers/helpers
Contents - table of contents
Introduction - Project's background and justification for doing it. Make sure to set a good first impression
Legal Considerations - Data Protection Law, Computer Misuse Law, copyright assets, and organising the testers. Stuff that does not apply must be mentioned.
Ethical considerations - Make sure that public release is not wrong to prevent misinformation. Vulnerability and product disclaimers. research and human participants (mention use of third party).
Conclusion chapter - Talk about early points that became important in the long run. Critical evaluation of the project. Things the author would do next time. does it meet the target? Do not mention PID, but do mention about requirement specification. What did the author learnt during this project, and how could this help in future similar projects? Potential further improvements and use of the project without constraint. Directions of further development. Identifying unresolved problems or inconveniences.
References - Content taken (not original work) such as images, tables, intellectual properties, etc.
Bibliography - Harvard referencing.

Can be put in the appendices:
A> Glossary.
B> Mark scheme table copied.
C> Still have it, but leave it black.
D> Research (academic and software). <-- literature review
E> Requirement specification, but justification must be in the core report. <-- literature review
F> say "not applicable".
G> say "not applicable".
H> say "not applicable".
I> say "not applicable".
J> Explain and show the GANTT chart and explain deviances between GANTT and actual progress.
K> All supervisor meetings: date, about, description, actions, and time. <-- logs
Includes: Planning, actions, and reflection - the whole thing. <-- Planning,
L> Sprint 1 and before.
M> Sprint 2.
N> Sprint 3 and after.

The project proposal goes into the report core.

Main chapters can be anything, but make sure to include the rest.

Main sections must follow template format, but can still use the author’s format for the sub-chapters (Even if they are a bit out of place).

Tina will get the author the meeting dates and the author will get what's about.

make sure to link and make the introduction, abstract, and conclusion all about the same.

To be on the safe side, have Harvard reference per individual web page instead of the websites themselves.

Can adjust the document's borders.

The supervisor can proofread per request.

Supervisor will be available most of the Easer holidays but not the weekend after.

11.11.2.3.3 Actions

Implemented the notes.

11.11.2.3.4 Evidence

Due to forgetting to bring anything to write on, the meeting notes were digitally typed (in the **Notes from meeting**).

11.11.2.4 2nd May 2025 12:35-13:30

11.11.2.4.1 Purpose and description

Talks primarily regards the template's appendices and codebase.

11.11.2.4.2 Notes from meeting

In the evidence part.

11.11.2.4.3 Actions

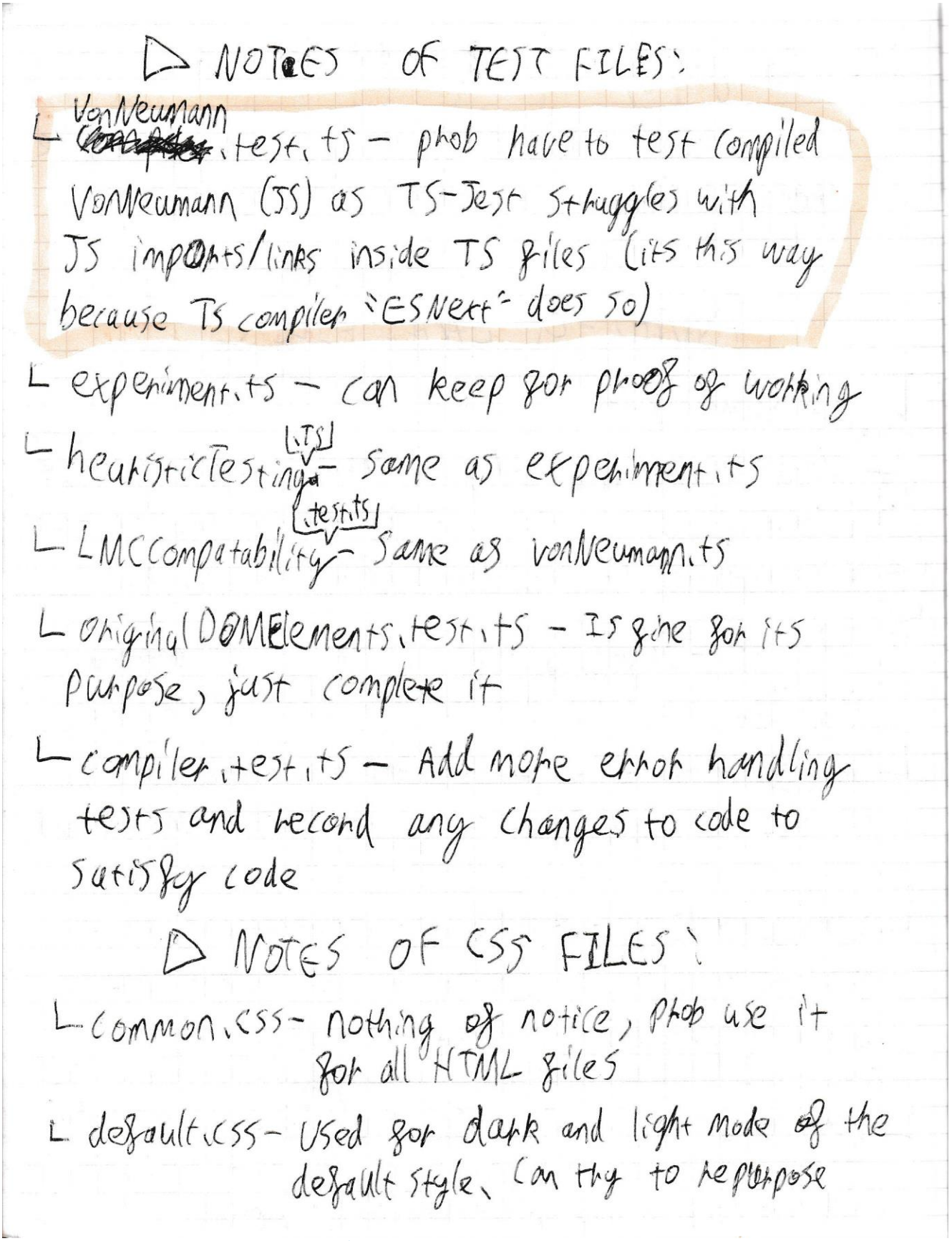


Figure 11.11—30 - first page of notes after the 2nd May 2025 meeting.

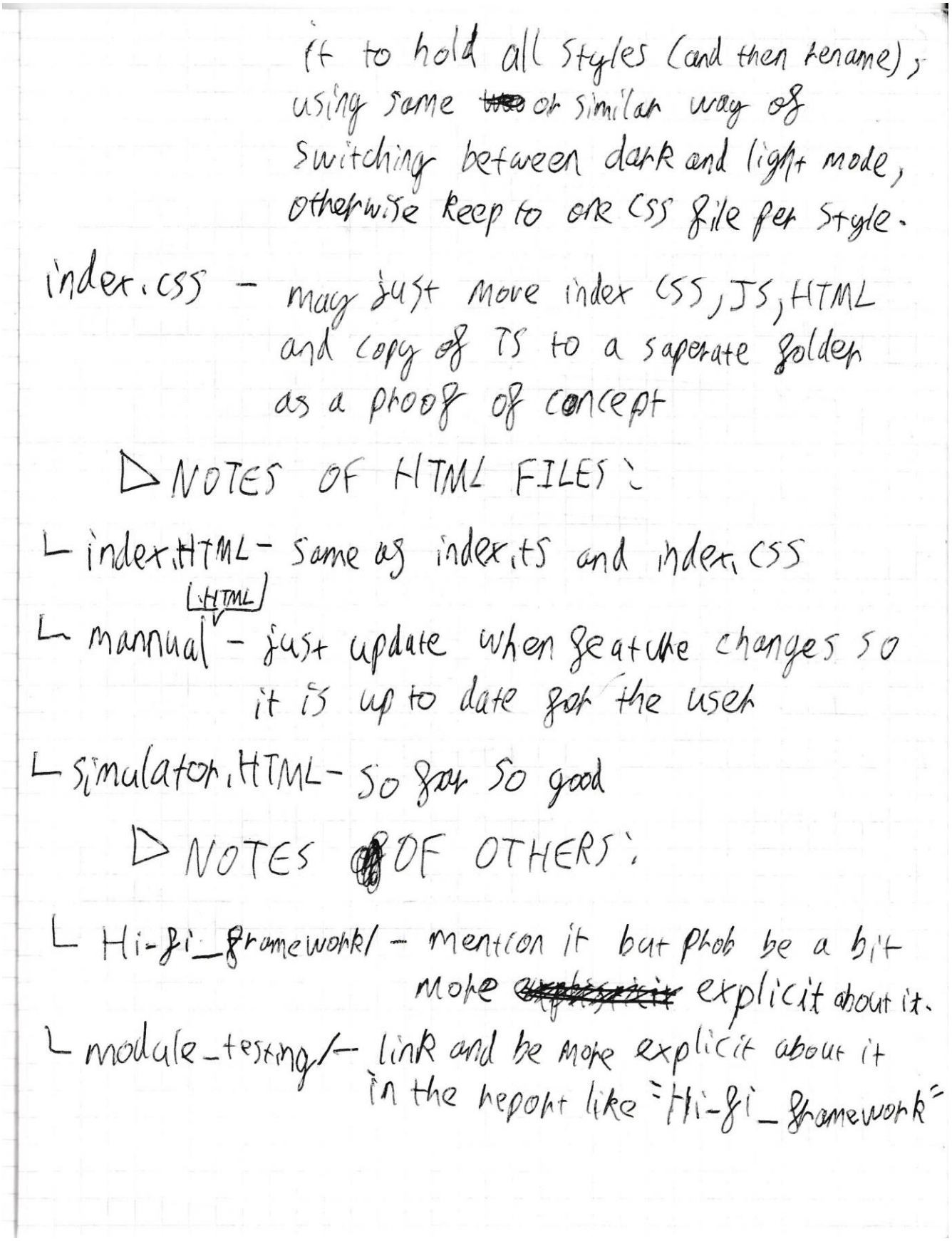


Figure 11.11—31 - second page of notes after the 2nd May 2025 meeting.

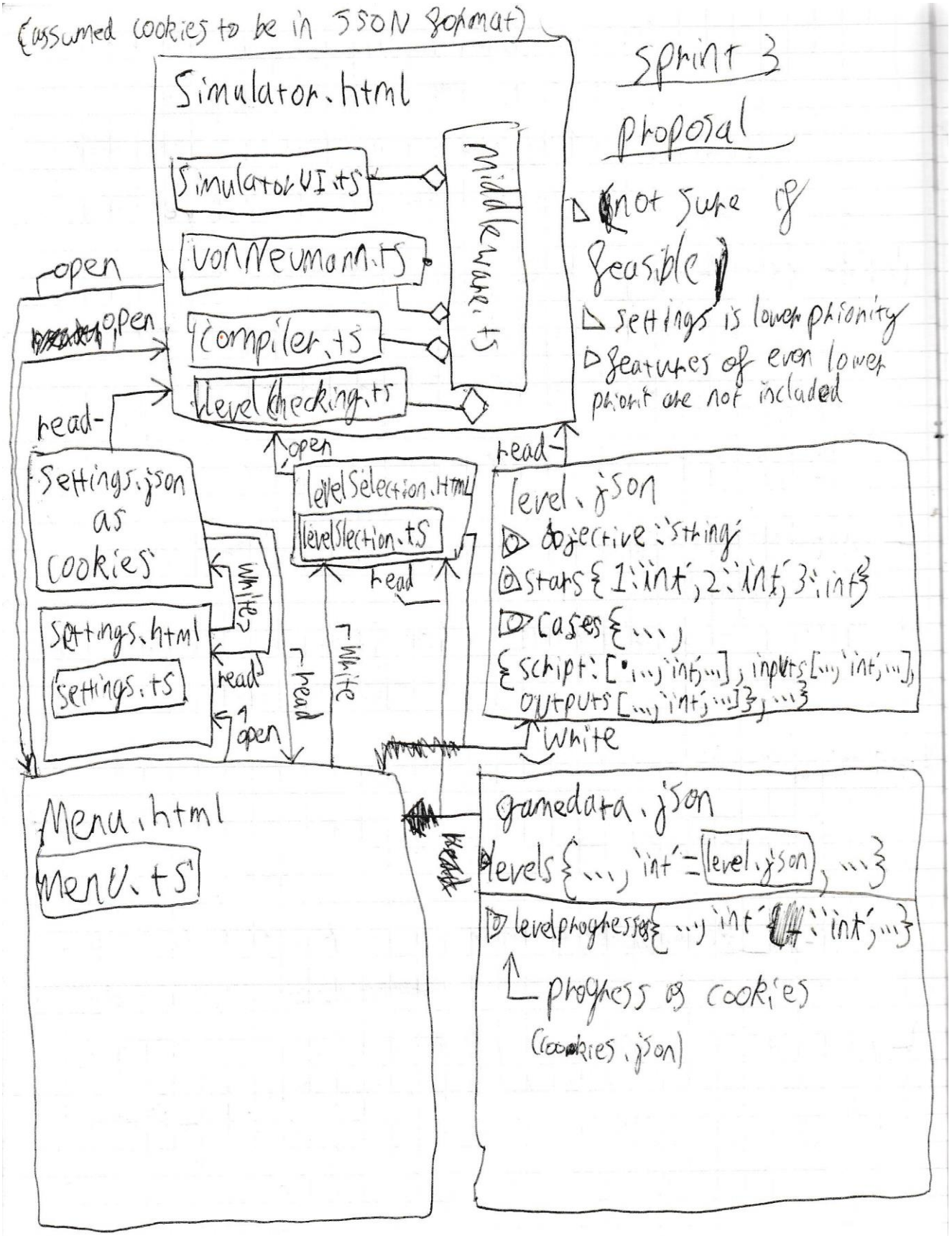


Figure 11.11—32 - third page of notes after the 2nd May 2025 meeting.

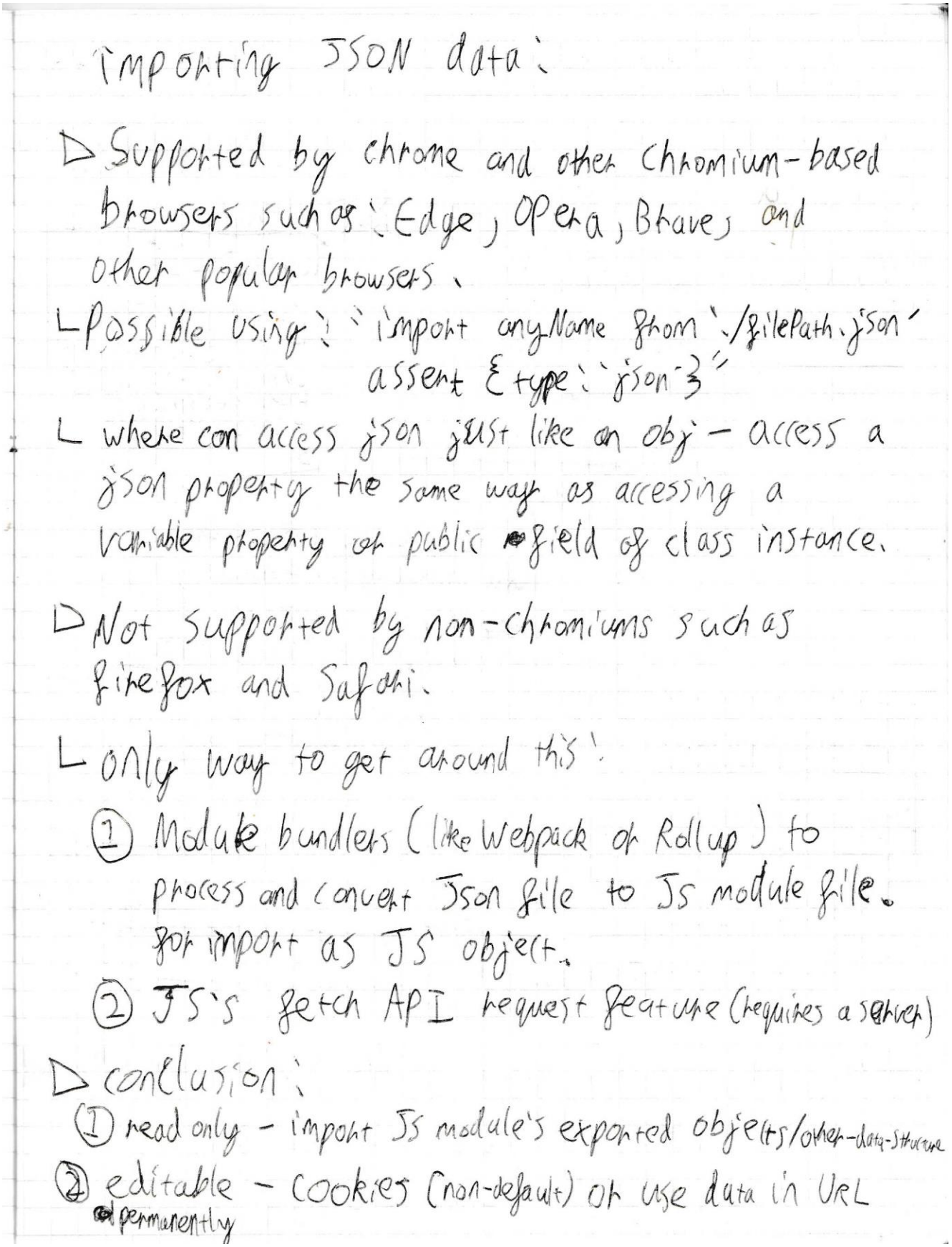


Figure 11.11—33 - fourth page of notes after the 2nd May 2025 meeting.

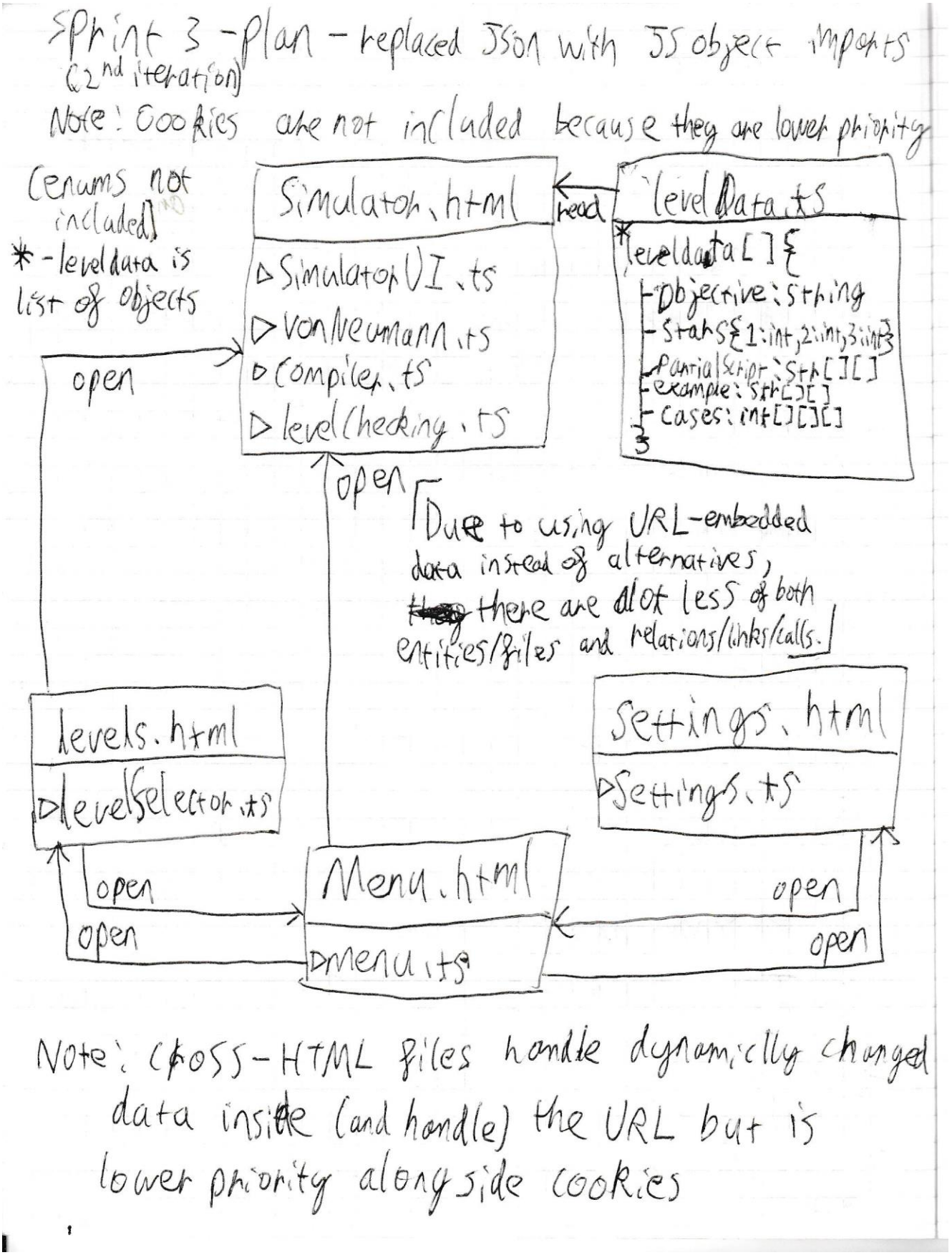
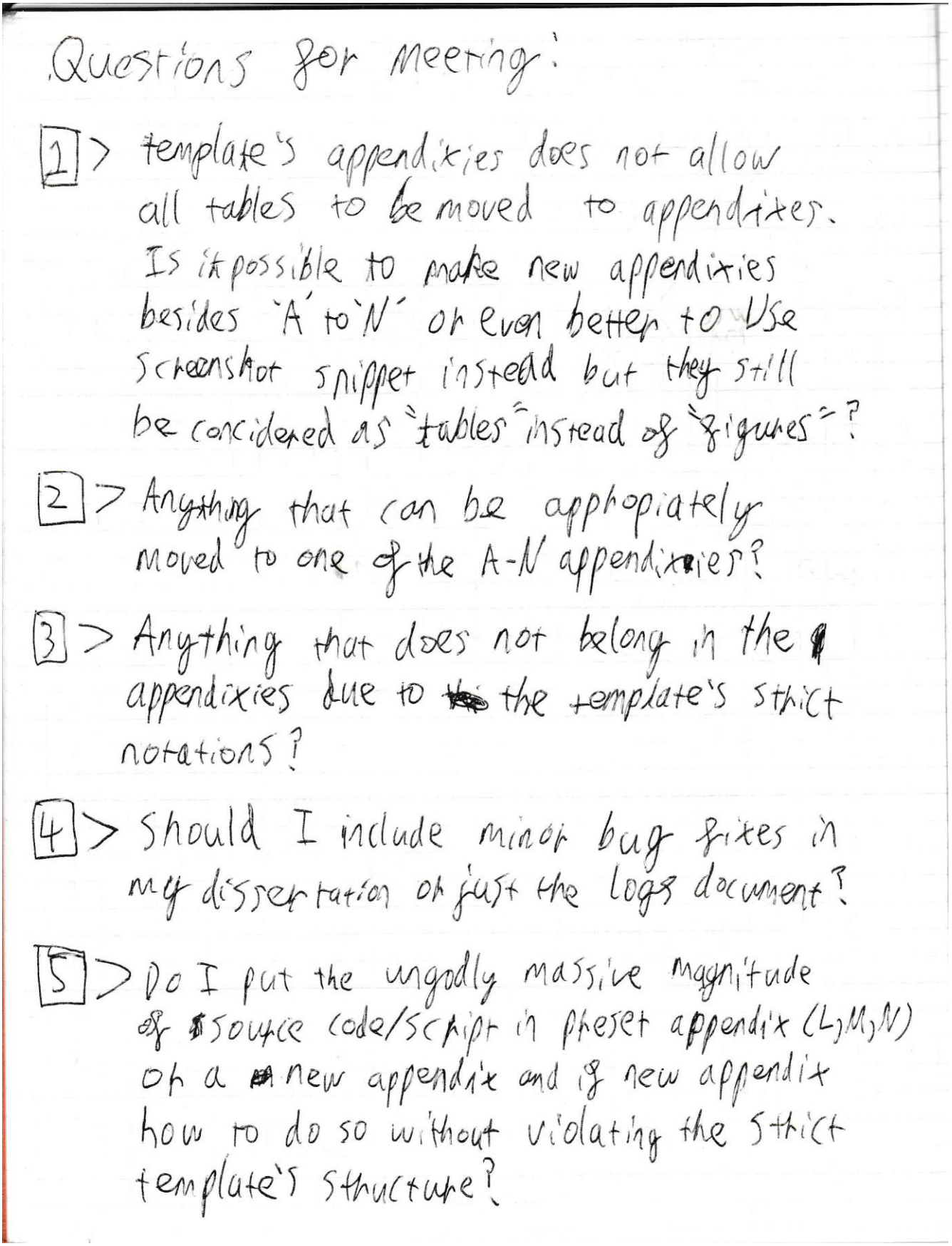


Figure 11.11—34 - fifth page of notes after the 2nd May 2025 meeting.

11.11.2.4.4 Evidence



Questions for meeting:

1> template's appendixies does not allow all tables to be moved to appendixes. Is it possible to make new appendixies besides 'A' to 'N' or even better to use screenshot snippet instead but they still be concidered as "tables" instead of "figures"?

2> Anything that can be apphopiately moved to one of the A-N appendixies?

3> Anything that does not belong in the appendixies due to ~~the~~ the ~~template~~'s strict notations?

4> should I include minor bug fixes in my dissertation or just the logs document?

5> Do I put the ungodly massive magnitude of source code/script in preset appendix (L,M,N) or a new appendix and if new appendix how to do so without violating the strict template's structure?

Figure 11.11—35 - first page of notes from the 2nd May 2025 meeting with the supervisor.

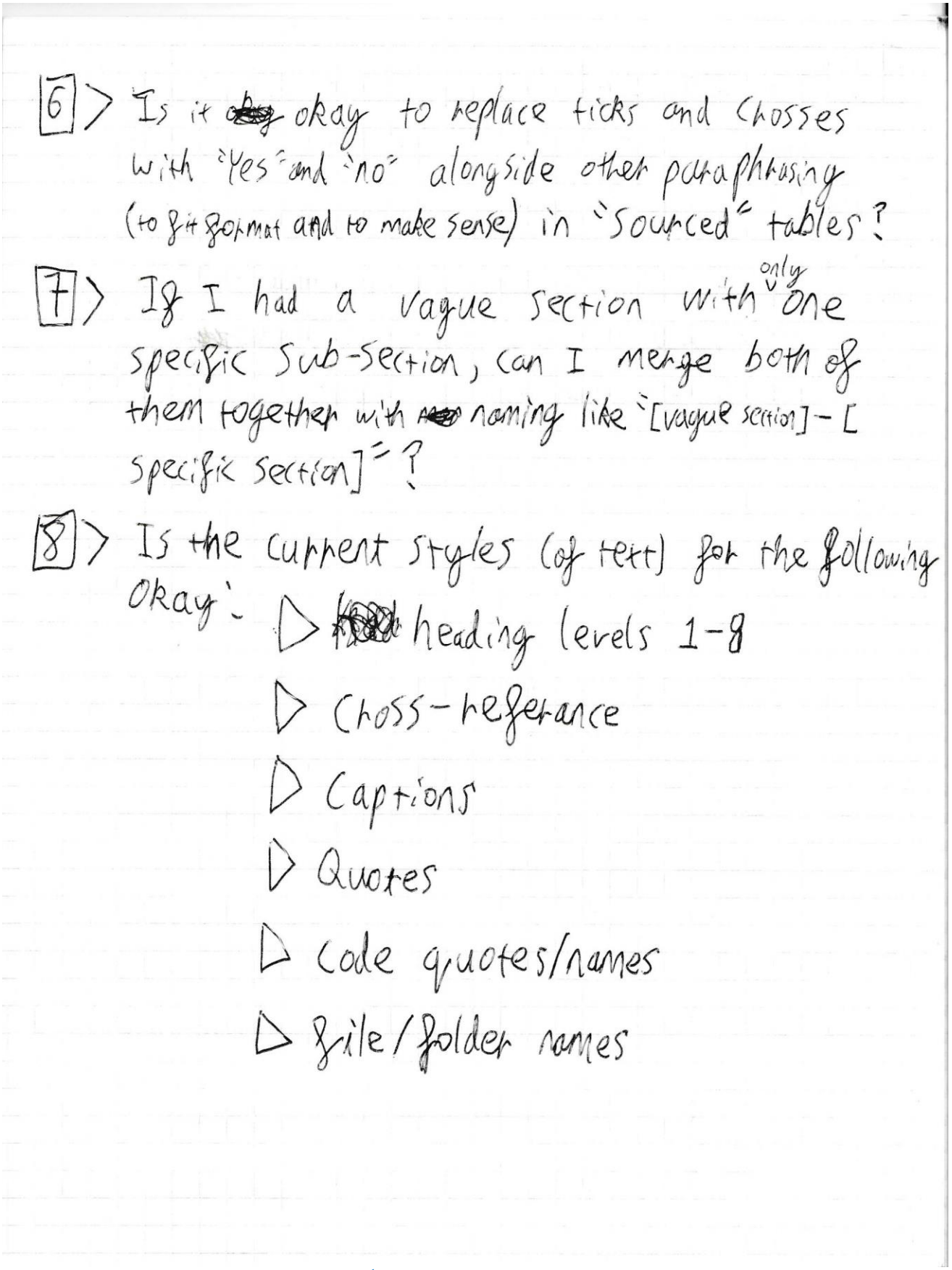


Figure 11.11—36 - second page of notes from the 2nd May 2025 meeting with the supervisor.

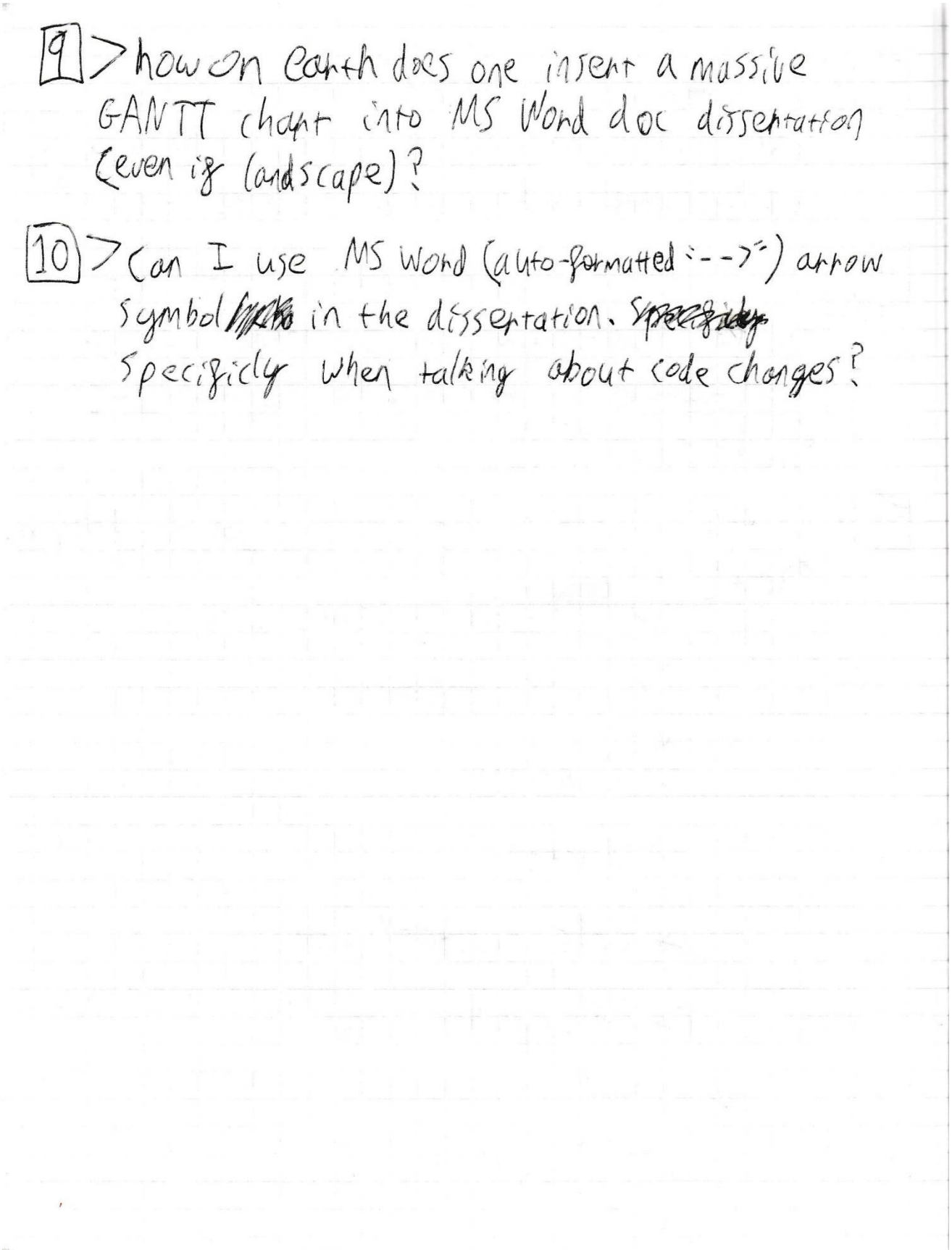


Figure 11.11—37 - third page of notes from the 2nd May 2025 meeting with the supervisor.

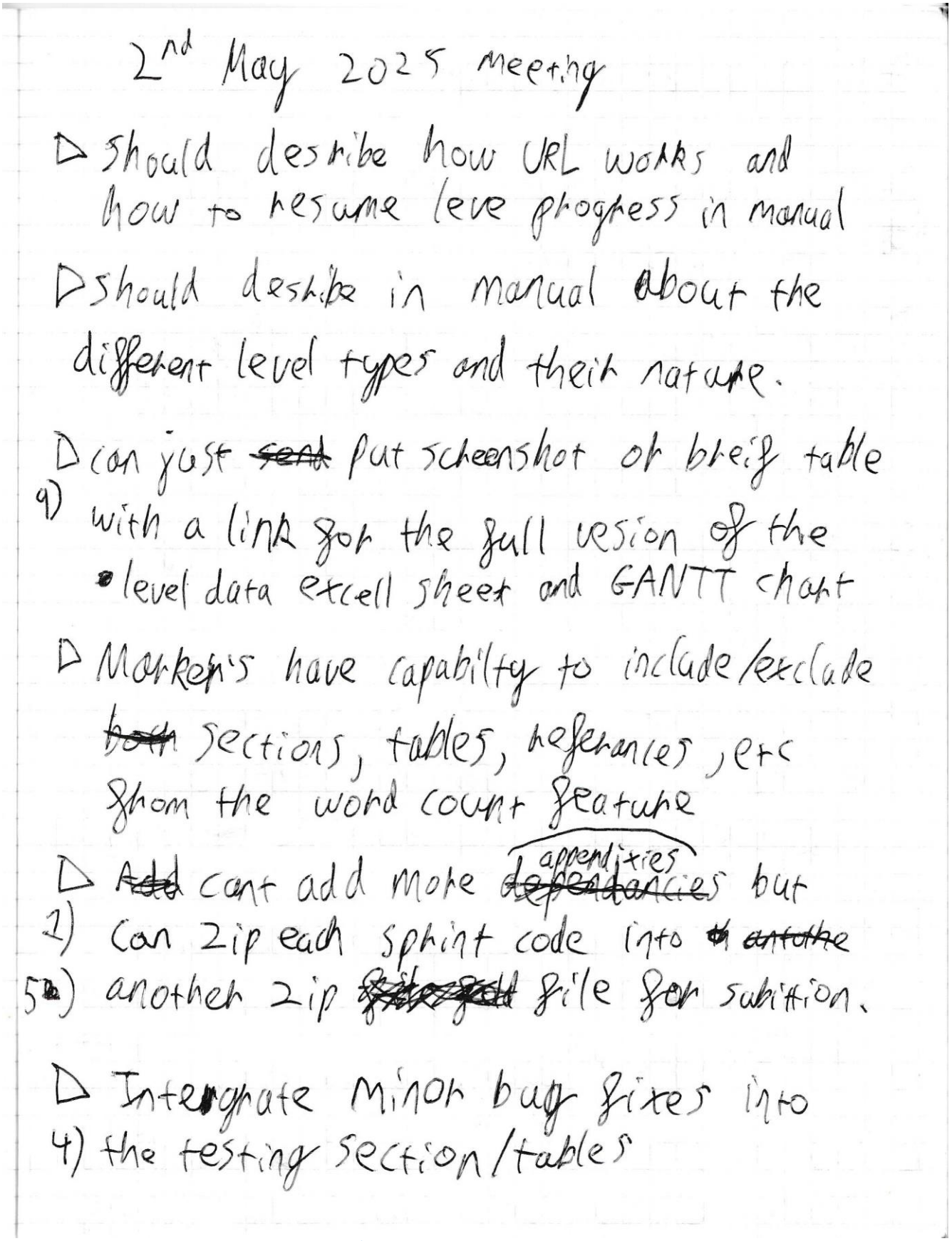


Figure 11.11—38 - fourth page of notes from the 2nd May 2025 meeting with the supervisor.

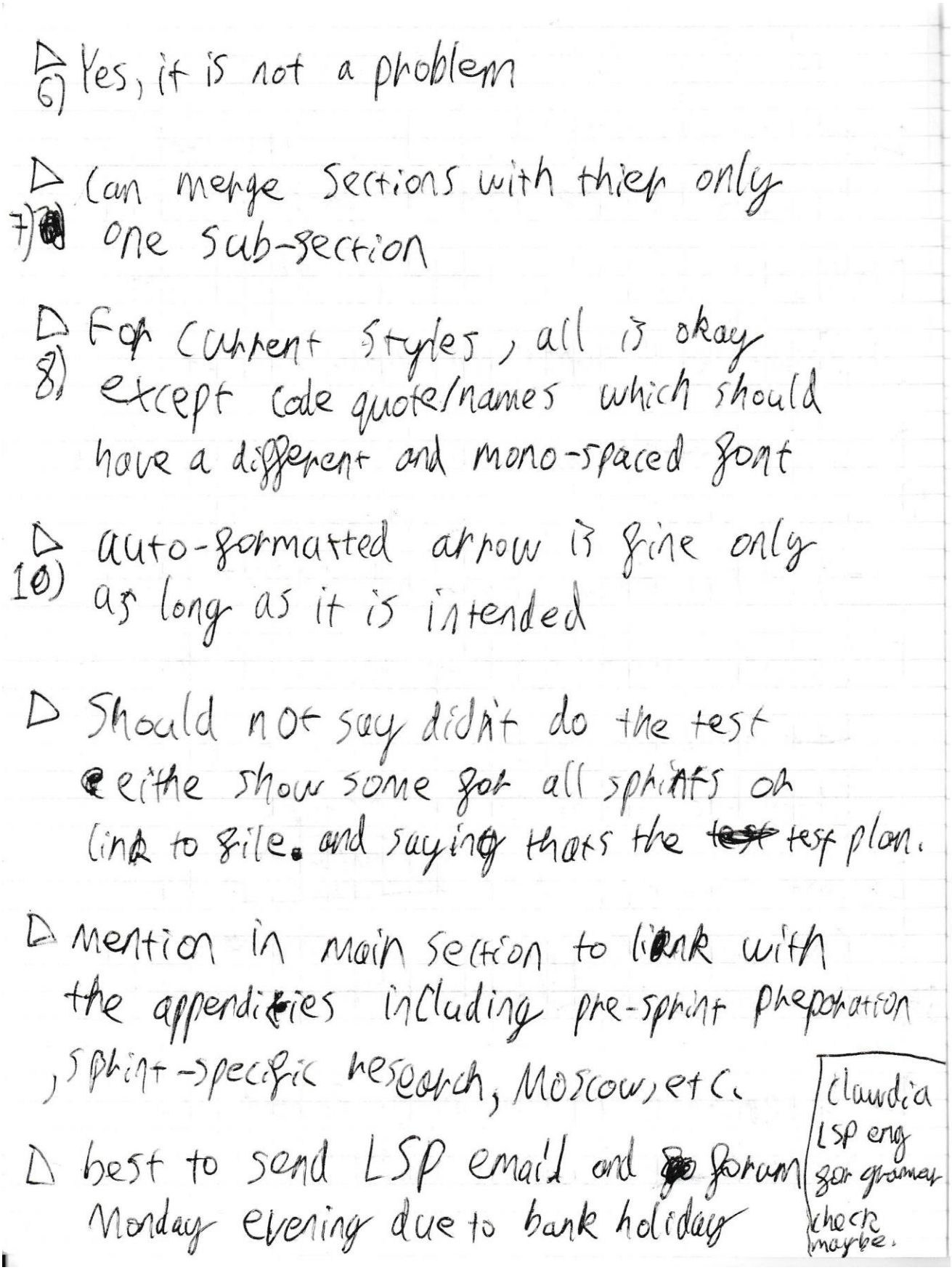


Figure 11.11—39 - fifth page of notes from the 2nd May 2025 meeting with the supervisor.

11.11.2.5 20th May 2025 13:05-14:00

11.11.2.5.1 Purpose and description

11.11.2.5.2 Notes from meeting

In the evidence part.

11.11.2.5.3 Actions

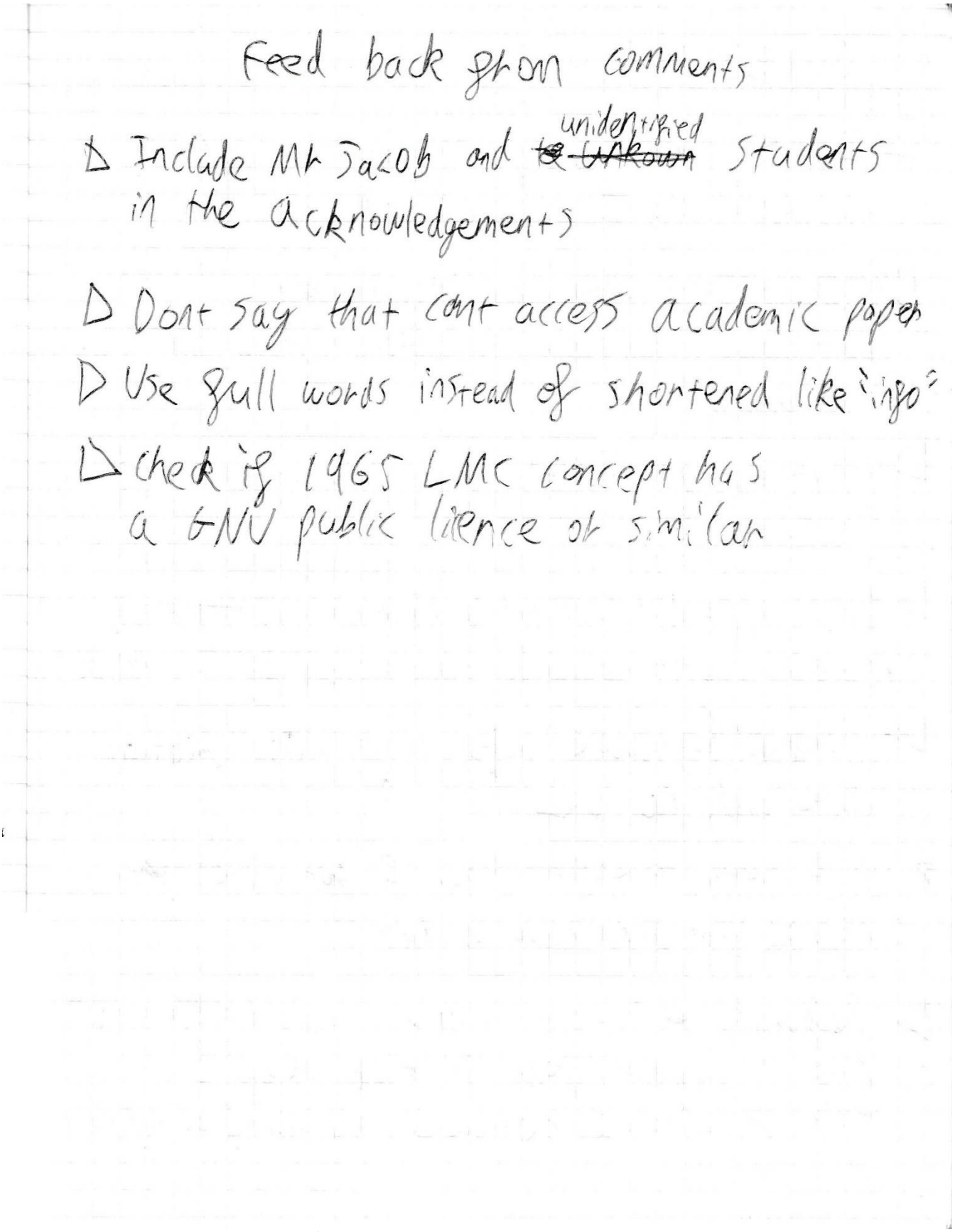


Figure 11.11—40 - page of notes after the 20th May 2025 meeting.

11.11.2.5.4 Evidence

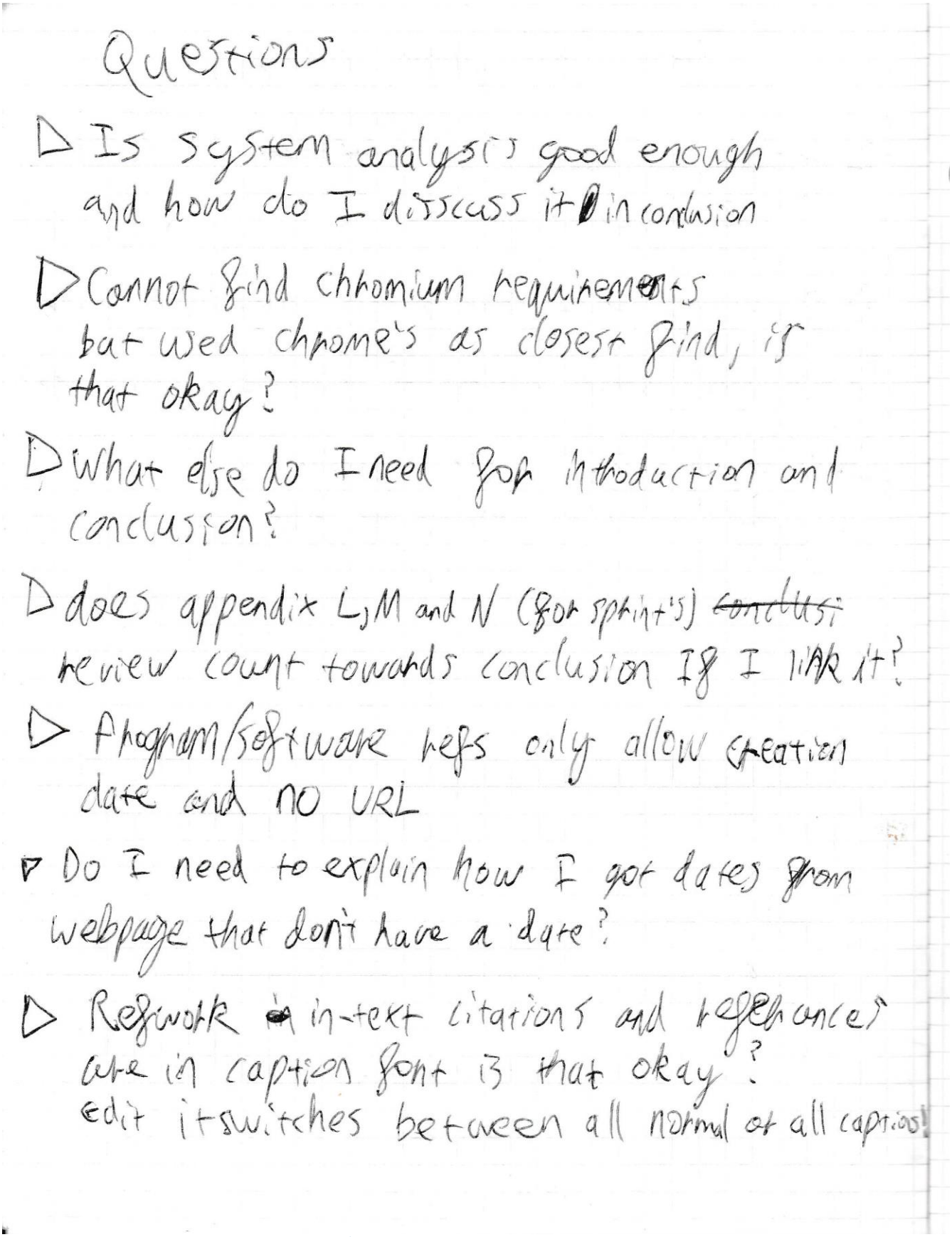


Figure 11.11—41 - first page of notes from the 20th May 2025 meeting with the supervisor.

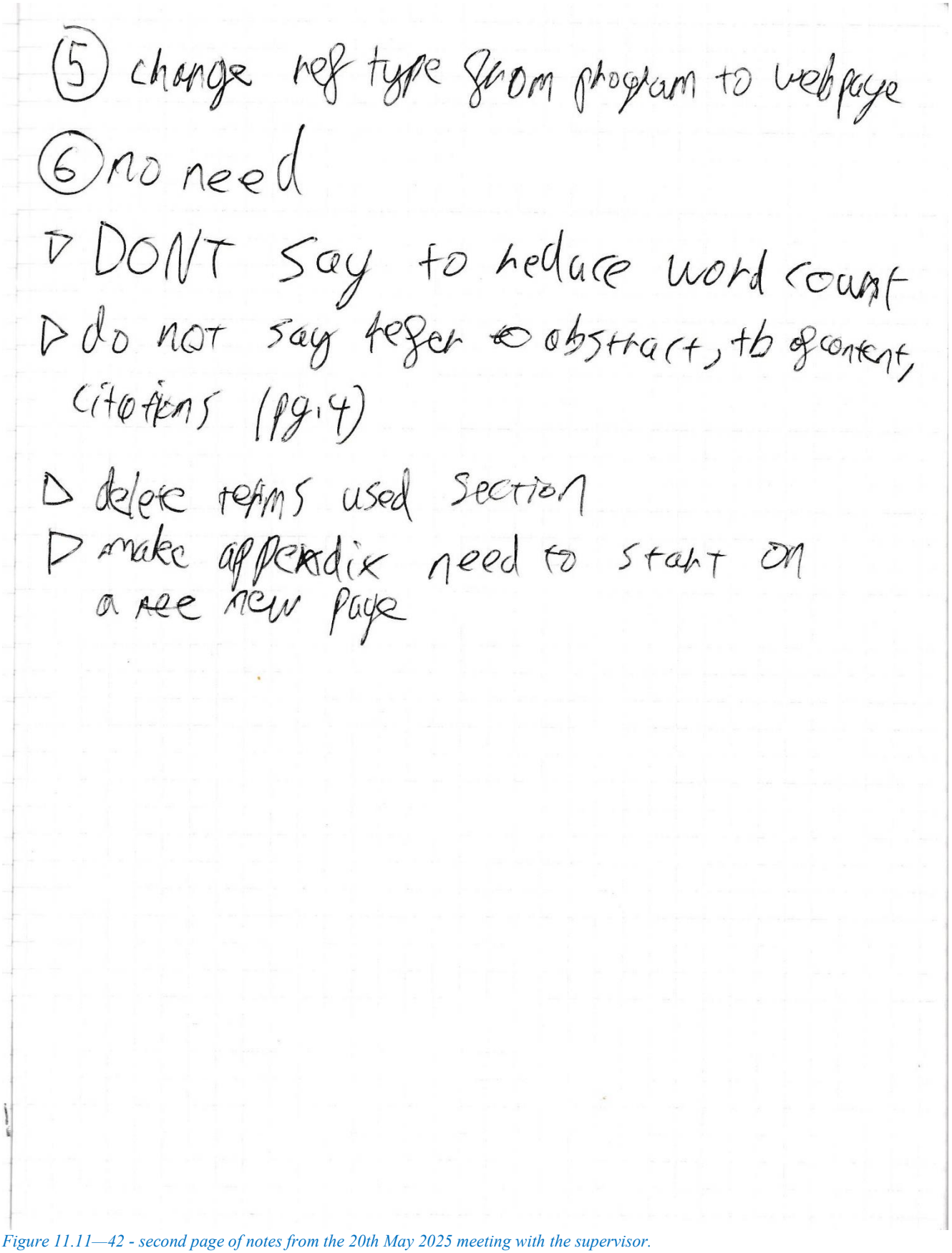


Figure 11.11—42 - second page of notes from the 20th May 2025 meeting with the supervisor.

11.12 Appendix L: Agile Development: Timebox 1

11.12.1 Before sprint 1

11.12.1.1 Setting up the code-space

Pre-sprint activities primarily consist of performing proof of concepts and vague, but all-sprint overview, draft plan. It is a very straight-forward process and not much needs to be mentioned here. Pre-sprint #1 activities are also known as sprint #0 in the industry (Bussa, Millett and Blankenship, 2011; Millett, Blankenship and Bussa, 2011).

In a more detailed description, the author leverages sprint #0 to validate critical technologies through fast proofs of concept and define a first product backlog, thereby creating technical unknowns before official development. Early validation minimises the chance of costly rework, while having a high-level roadmap sets scope and aligns objectives. Along the way, Sprint 0 provides a laser-like, predictable foundation for improved execution of subsequent sprints.

11.12.1.2 Testing the run environment

11.12.1.2.1 Mark-up

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../styles/index.css">
    <title>Document</title>
</head>
<body>
    <h1 id = 'message'>This text means the script has not yet executed or failed</h1>
    <script src="../compiled/index.js"></script>
</body>
</html>
```

11.12.1.2.2 Style

```
/* Not grey-scale incase darkmode is used to view*/
body {
    background-color: green;
    color: yellow;
}
```

11.12.1.2.3 Script

11.12.1.2.3.1 index.ts

```
document.addEventListener('DOMContentLoaded', main)

function main(){
    const testMsg: string = "Hello world!"
    const message: HTMLElement | null = document.getElementById('message');
    if (message != null) {message.textContent = testMsg;}
}
```

11.12.1.2.3.2 index.js

```
"use strict";
```

```
document.addEventListener('DOMContentLoaded', main);
function main() {
    const testMsg = "Hello world!";
    const message = document.getElementById('message');
    if (message != null) {
        message.textContent = testMsg;
    }
}
```

11.12.1.3 Planning for the whole project

These plans are made as a guide to ensure that the author stays on the right path. These will have more deviation than sprint-specific plans however, this does not mean that a plan from one sprint cannot apply to another sprint's development. Hence, these all-sprint plans tend to have a lower level of detail.

11.12.1.3.1 Use cases

Sprint 1 only covers the backend simulator, but having a use case diagram will be useful here to make sure that the authors do not deviate from the program's original purpose in the long run.

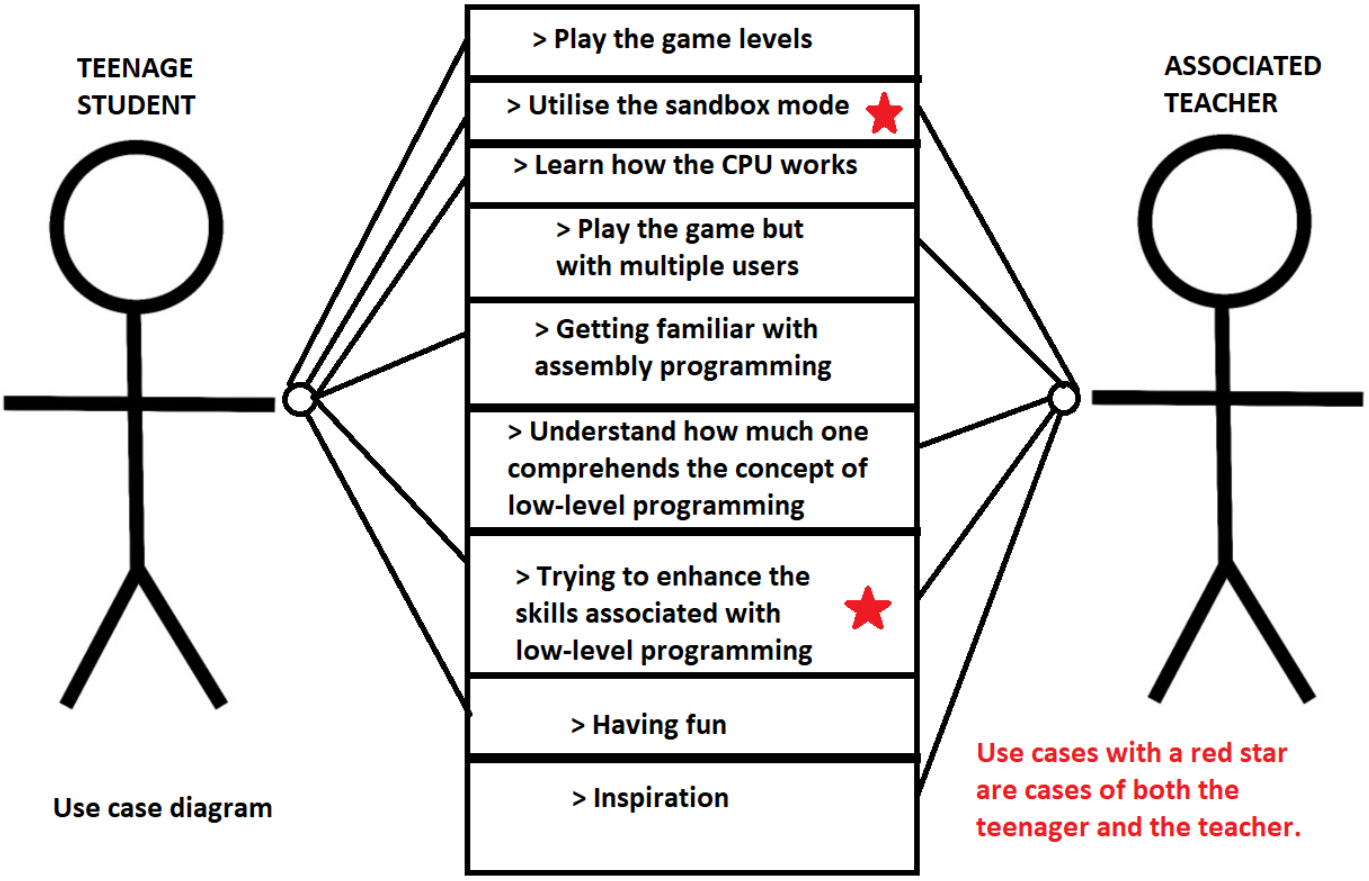


Figure 11.12—1 - use case diagram showing and comparing the use cases of the student (13-18) and their teachers.

11.12.1.3.2 Wireframe Mock-ups

Only the first sprint does not involve implementing UI (focusing on the simulator's workings instead) – making it suitable for only sprints #2 and #3. However, a quick set of low-fi draft wireframes is made before doing the first sprint to help guide the author in the right direction without tangents.

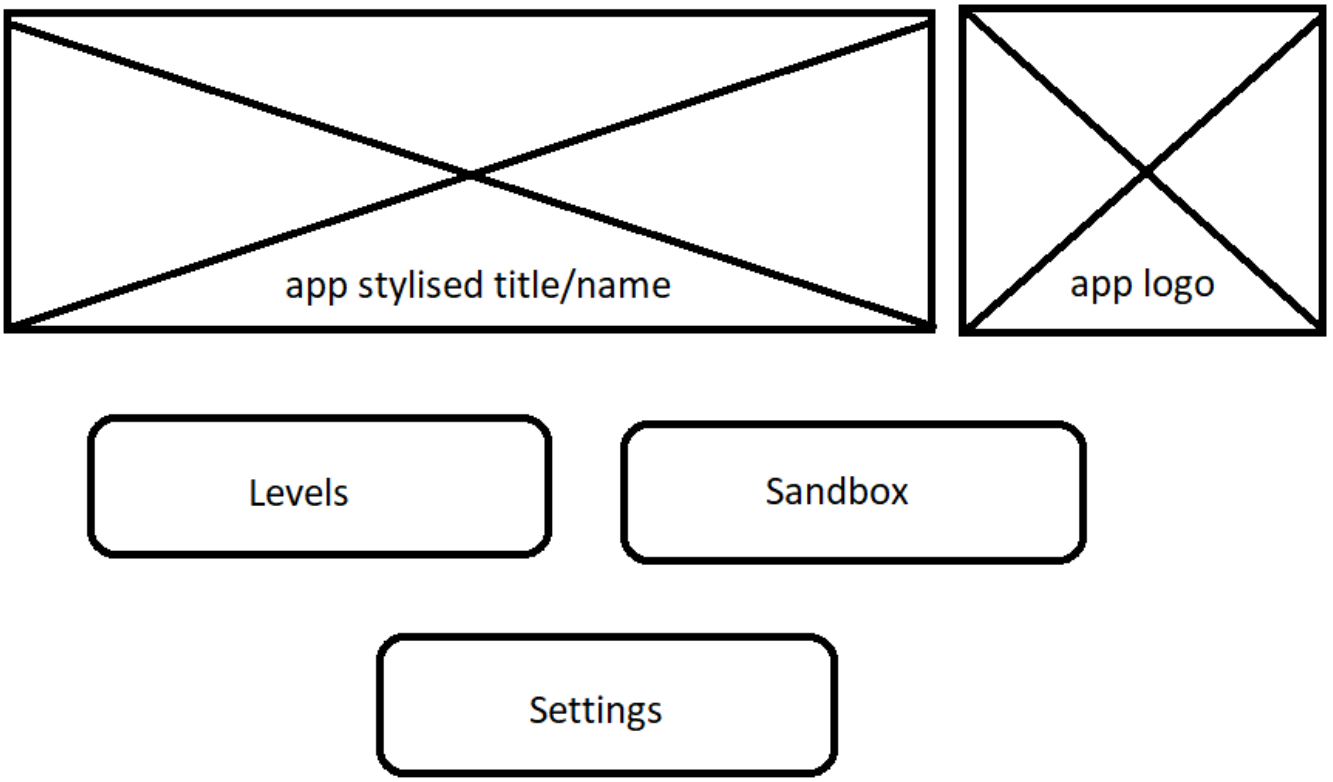


Figure 11.12—2 - wireframe mock-up of the main menu page.

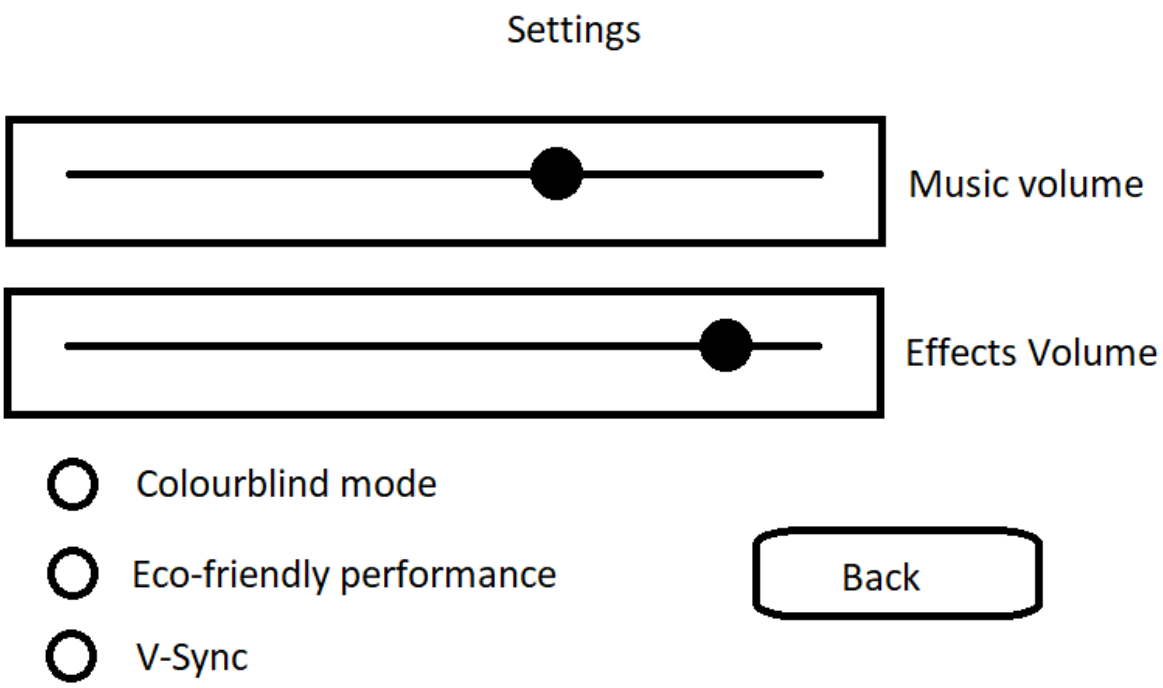


Figure 11.12—3 - wireframe mock-up of the settings page.

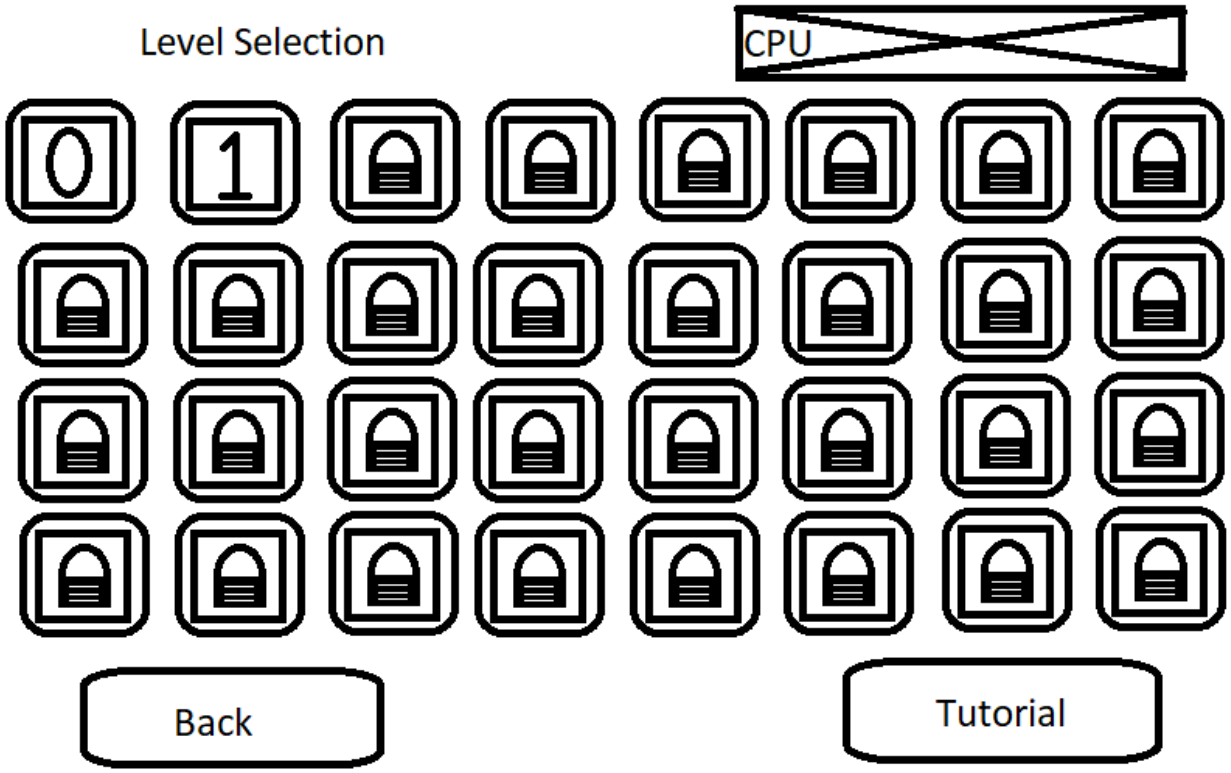


Figure 11.12—4 - wireframe mock-up of the level selection page.

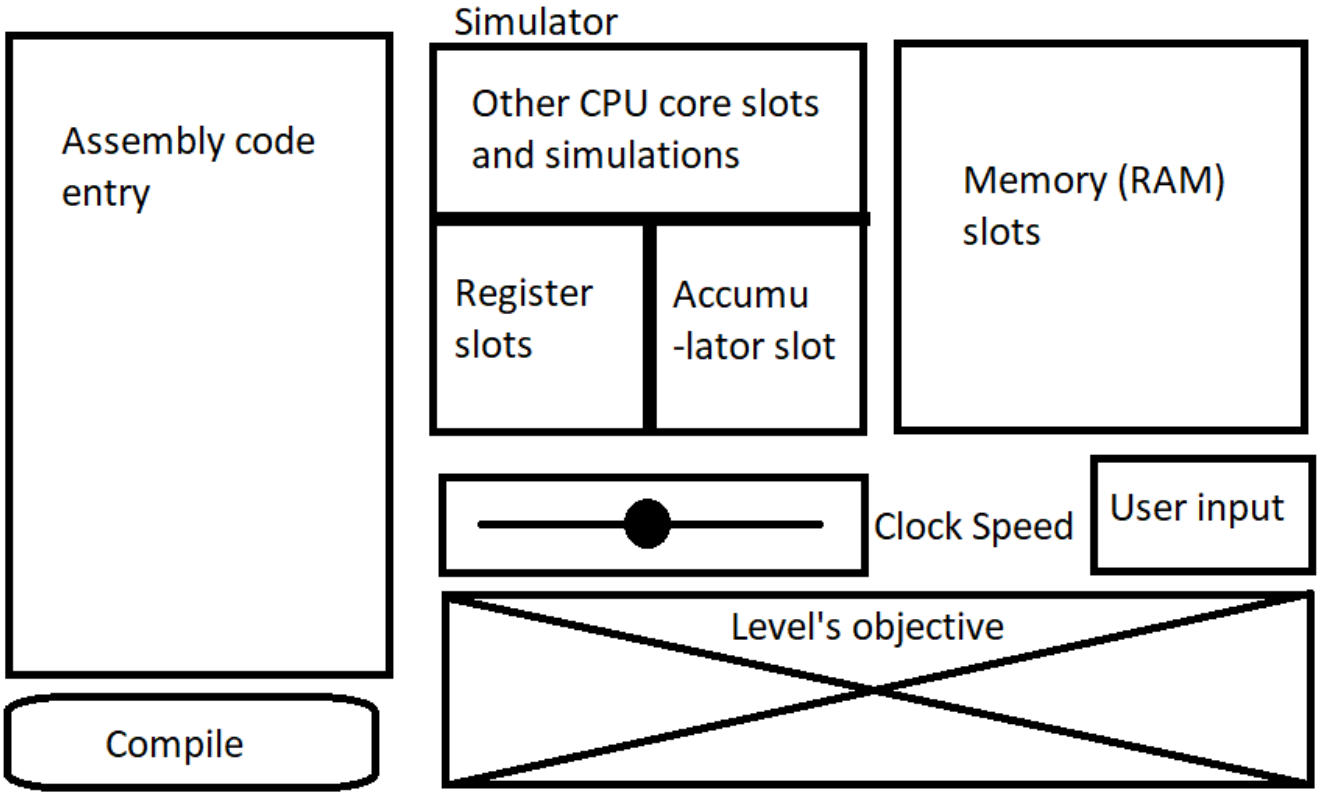


Figure 11.12—5 - wireframe mock-up of the simulator page.

11.12.1.4 Current PERT chart completion

As PERT chart does task order and task dependencies, instead of sprint completion, it gives another perspective regarding the current project progression. Because of this PERT charts are only updated when appropriate: sprint #0, post-sprint #1, feedback implementation and platform porting.

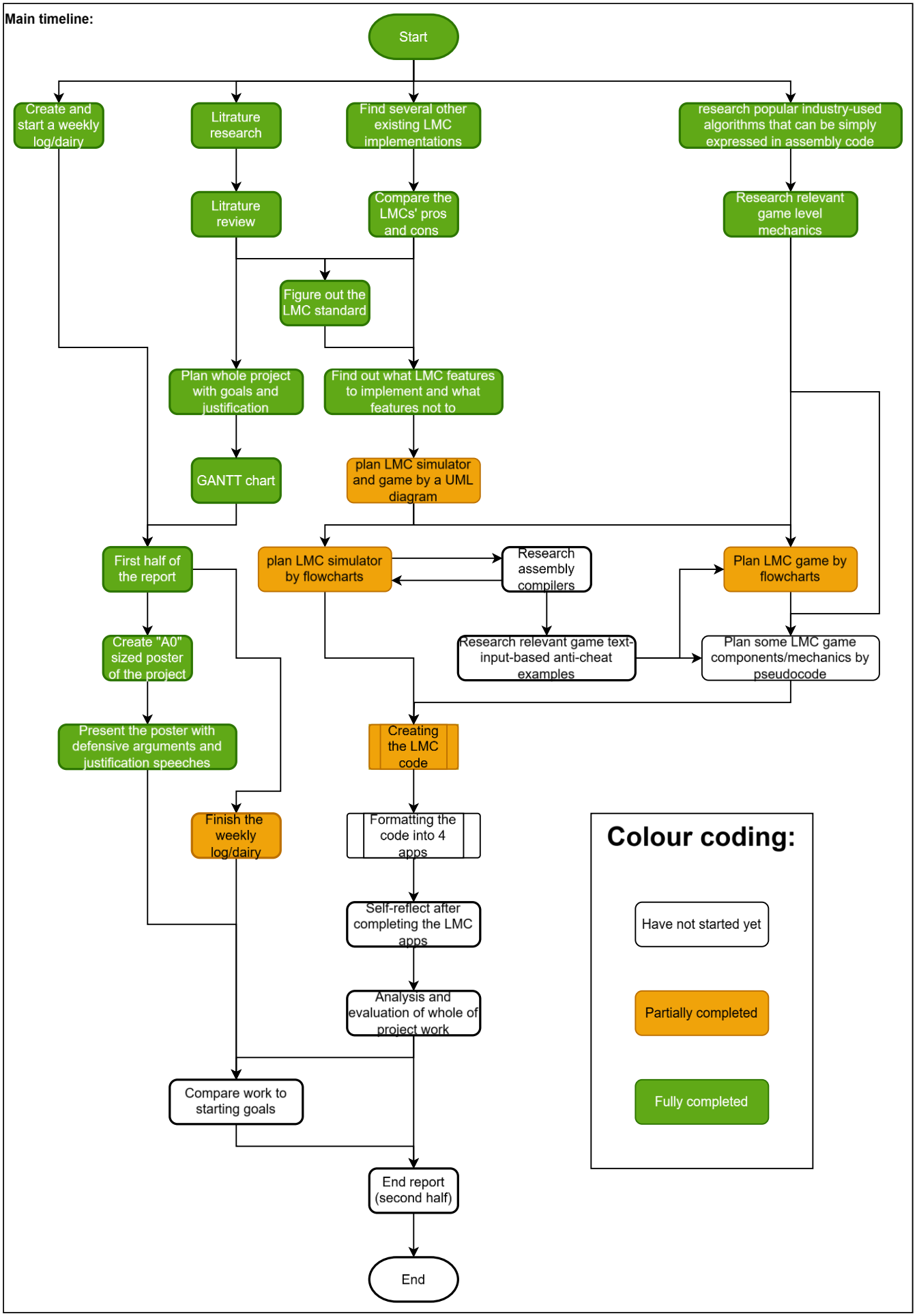


Figure 11.12—6 - current PERT chart progress at the time of after completing sprint #0.

11.12.2 Sprint 1

11.12.2.1 Planning

Spirit #1 will be implemented in accordance with the plan. The sprint 1 plan is split between the: use cases, IPOD table, instruction set, flowchart, and frameworks.

11.12.2.1.1 Prior Research

The majority of the research relevant for the effective development of the first sprint is done in the **Software research**.

11.12.2.1.1.1 Development dependencies

11.12.2.1.1.1.1 NPM developer dependencies

While not required to run the application, they are a must for critical assistance with the development of the LMC simulator game. Downloaded and managed from Node.js's *NPM* package manager.

Sub-sections ordered by name are used in the installation command.

11.12.2.1.1.1.1.1 @types/jest

- Used download command – `npm install --save-dev @types/jest`.
- Used version – 29.5.14 (types, 2024)
- What is it – A type definitions library dependency.
- What it does – Provides TS type definitions to allow Jest functionality to be written in TS.
- Why use it – Without it, TS will not recognise the Jest-provided functions and objects.

11.12.2.1.1.1.1.2 jest

- Used download command – `npm install --save-dev jest`.
- Version – 29.7.0 (Abramov *et al.,* 2023)
- What is it – A Testing framework dependency.
- What it does – Allows automated testing of the project's features and functionality.
- Why use it – For rapid prototyping and mass testing across all features to ensure that not a single logic error nor run-time error remains. The other main reason is that it has very high code reproducibility, due to its standard as a framework, making it easily replicable by other developers and tested on other hardware and browsers.

11.12.2.1.1.1.1.3 ts-jest

- Used download command – `npm install --save-dev ts-jest`.
- Version – 29.2.5 (Kabra, Ahn and tsjest, 2025)
- What is it – A package that allows Jest test files to directly run TS scripts instead of JS scripts.
- What it does – Allows Node to directly run TS files instead of JS files.
- Why use it – Automatic tests are updated very often and usually run after each update to make sure both the test files and the program are working as intended. Not having to manually compile the multiple test files to JS will save both a lot of time in the long run and potentially mistakes, such as believing that the test file does not work as intended, when instead the developer forgot to compile the test file. There is an option to include the compilation in a node script, but that will take some time each time the script needs to get updated when a new test file is made; also, it will make the process far more complicated.

11.12.2.1.1.1.1.4 ts-node

- Used download command – `npm install --save-dev ts-node`.
- Version – 10.9.2 (Embrey and Bradley, 2024)
- What is it – Runtime environment dependency.
- What it does – Allows Node to directly run TS files instead of JS files.
- Why use it – Has the same advantage as ts-jest, of saving time from manually compiling or extra complexity editing node run scripts each time a new TS file is made or an existing change's name or location. One may have a concern that potentially, when TS is compiled to JS, it does not work fully for all browsers and is too big to mend. This is not of concern to the author because, as long as it works on Chromium as a baseline, it should be fine.

11.12.2.1.1.1.1.5 typescript

- Used download command – *`npm install --save-dev typescript`*.
- Version – 5.7.3 (typescript-bot *et al.,* 2025)
- What is it – Compiler dependency.
- What it does – Allows TS files and syntax to be compiled into JS.
- Why use it – TS is the language chosen by the author for the reasons mentioned in the "Technologies used" part of this section. Compiling to JS allows the program to be natively run in browsers, allowing online access.

11.12.2.1.1.1.2 IDE extensions

All IDE extensions searched and downloaded from the VS marketplace, accessible from either the extensions tab in VS Code or online (Visual Studio Code, 2018).

These are not necessary for the development, deployment, or maintenance of the project's program, but they make the development smoother.

Downloads and ratings are not key in choosing the VS extensions, but they help show their reliability by their popularity. Hence, the inclusion of screenshot snippets in the extensions' overviews. However, the number of ratings is usually too low to comment on.

To view these extensions yourself, simply enter the extension's unique extension identifier into the search bar of the VS marketplace (Visual Studio Code, 2018).

11.12.2.1.1.1.2.1 Better Comments by Aaron Bond

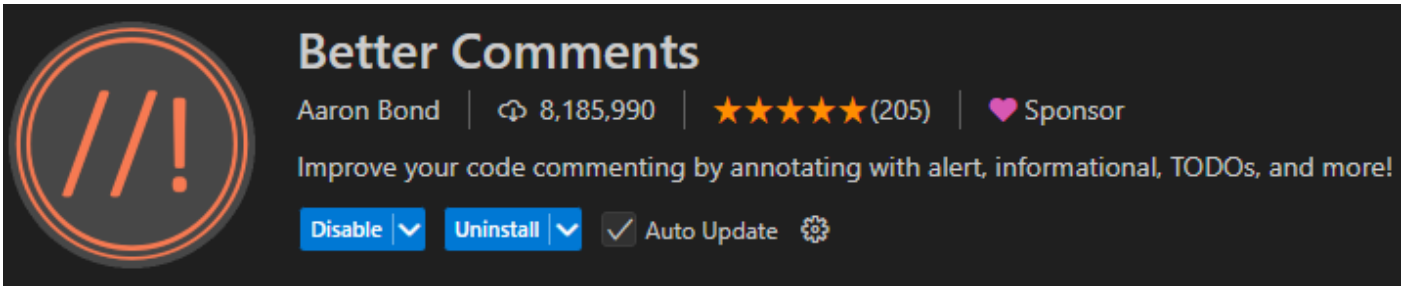


Figure 11.12—1 - shows the overview of the "Better Comments" VS extension.

Version – *3.0.2*.
VS unique extension identifier - *aaron-bond.better-comments*.

Changes the colour and format of the text of each comment depending on the first character (after the comment notation). Allow to differentiate between different kinds of comments, such as: disable code, single line comments under, commenting an entire code block, to-do comments, alerts, points of interest, et cetera.

```
TS commentTypes.ts
1   //  Generic.
2   //^ Describing or explaining the code line above.
3   //: Describing or explaining all code lines below until first emptry line.
4   //* Describing or explaining all code lines below in the same scope - until code line with lower indentation than comment's.
5   //< Describing or explaining the code in the same line.
6   //! Noting any logic errors or other significant code that needs fixing (for developer) ASAP.
7   //? Noting any confustion for user to solve later.
8   /// Disabled code line
9   //# Notes for the current scope it is in (not as strict as '//*') but does not affect region start nor end declarations.
10  //x Note that is not neccessarily related to the code - such as talking about the greater project (can also be done with '//X').
11
12  /* showing its use in multiline comments:
13      ^
14      :
15      *
16      <
17      !
18      ?
19      ≠
20      #
21      //x
22  */
23
24  //x Due to '/**/' restrictions and limitations, some multiline comments are done with multiple adjecent single-line comments.
25  //x If each of those comment line is its own sentence (start capitalised, unless not a letter, and ends with full stop '.') then it is its own saperate comment.
26
27  //x otherwise you concider the multiply comment lines
28  //x as the one comment
29
```

Figure 11.12—2 - shows and explains the different comment formats used by the "Better Comments" VS extension.

It also has a reasonable number of downloads – good enough to be considered reliable. The number of ratings is too low to make a comment on.

11.12.2.1.1.1.2.2 Live Server by Ritwick Dey

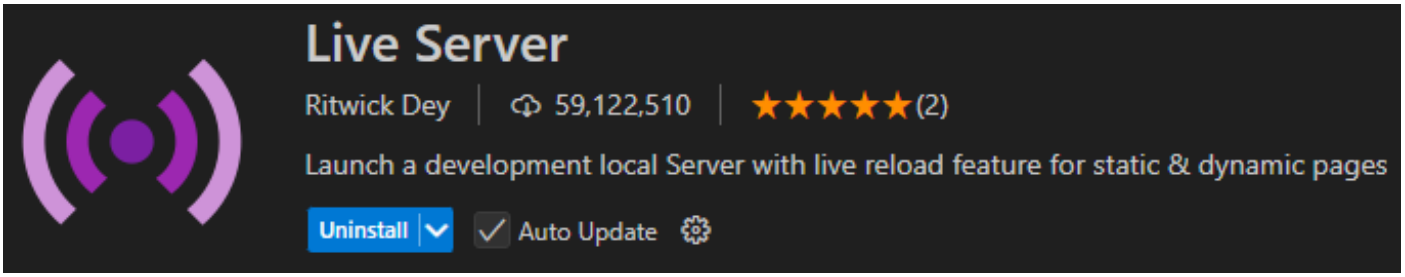


Figure 11.12—3 - shows the overview of the "Live Server" VS extension.

Version – *5.7.9*.
VS unique extension identifier - *ritwickdey.liveserver*.

Allows a local-hosted server to run whatever HTML file is selected and links to other necessary files (CSS, JS, other HTML, PNG assets, et cetera) that auto-update every time a saved change is made to the files.

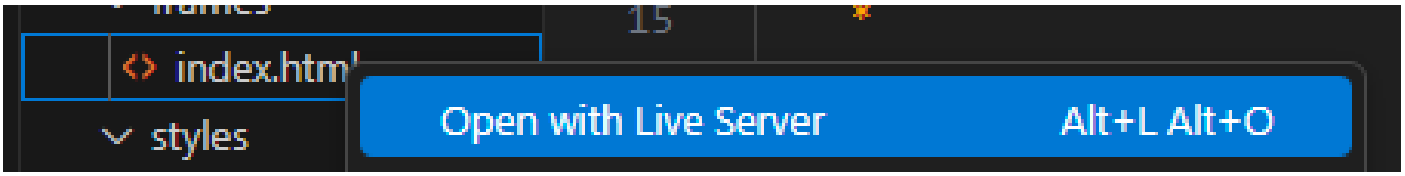


Figure 11.12—7 - shows how a HTML file can be easily opened/hosted with the "Live Server" VS extension.

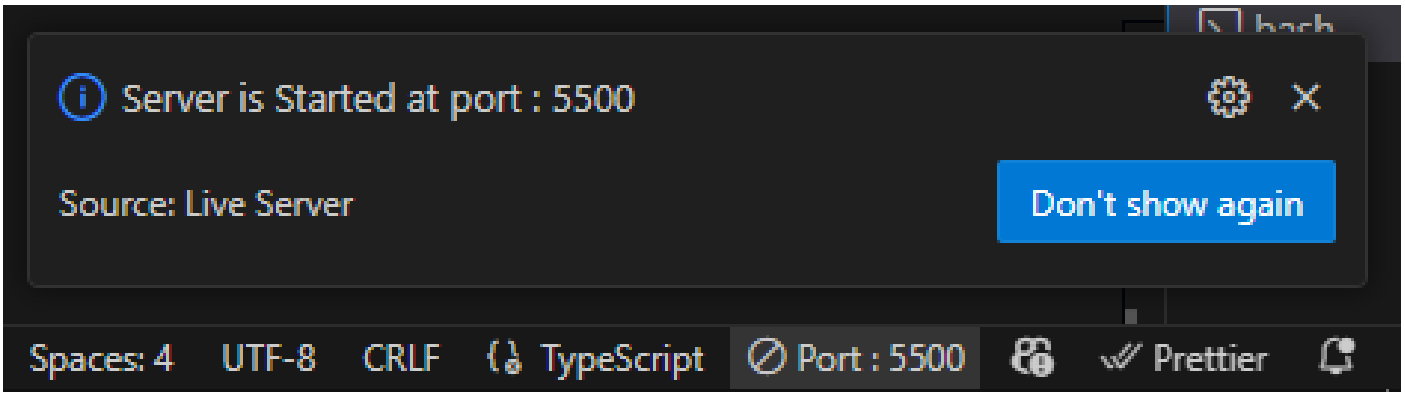


Figure 11.12—8 - Shows the confirmation and details for the local host from the "Live Server" VS extension.

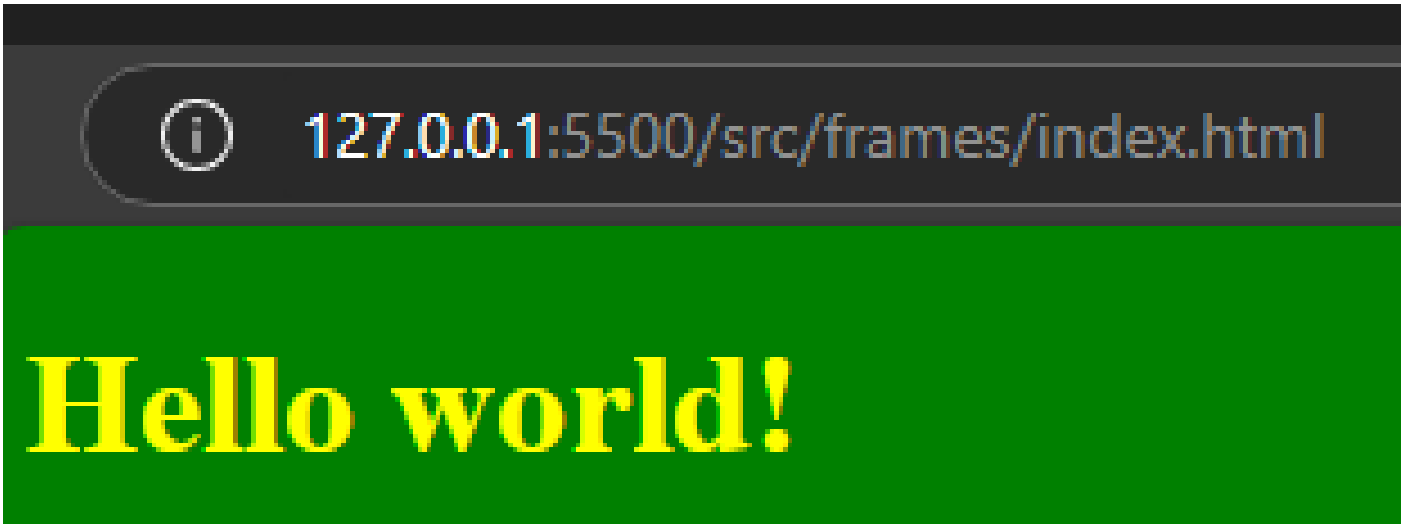


Figure 11.12—9 - shows the viewing of a HTML file using the server host from "Live Server" VS extension.

Save so much time rather than reopening HTML files every time the author updates the code, and despite it being a server, it does not require server code to run (just code that a client node would need), unlike other auto-updating server hosts like the *nodemon* NPM package (Sharp, 2025). This is important because, even in the online website platform/version of the LMC project, the project will not involve server hosting code as it takes a long time to set up and even more to plan and debug.

It also has an astounding number of downloads – popular on an industrial scale makes it extremely reliable.

11.12.2.1.1.1.2.3 Region Marker by Przemysław Orłowski

Version – *0.3.6*.
VS unique extension identifier - *awwsky.region-marker*.

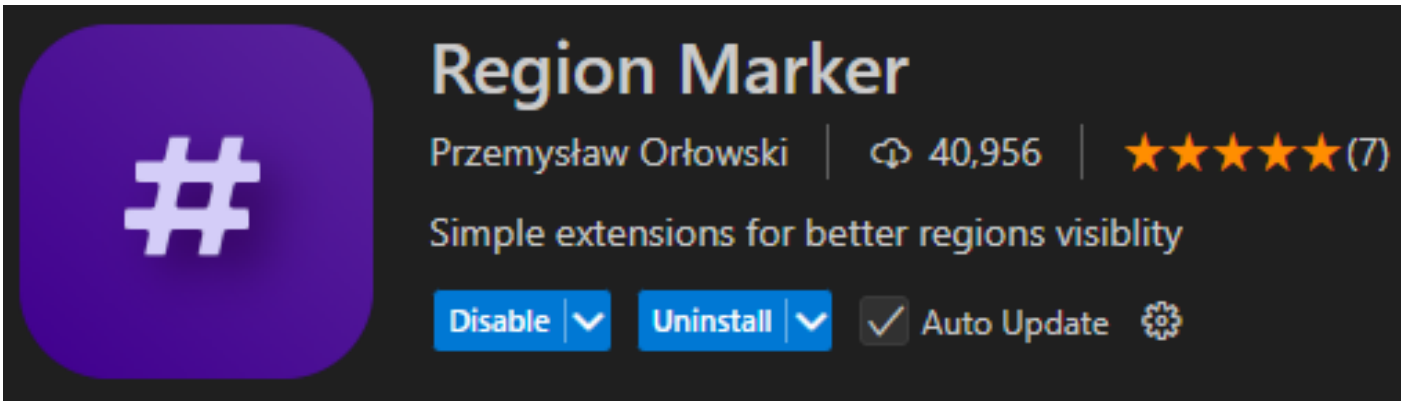


Figure 11.12—10 - shows the overview of the "Region Marker" VS extension.

Light extension that changes the format of region declarations. This makes them more visible to the developer, which is very useful for organising code files.

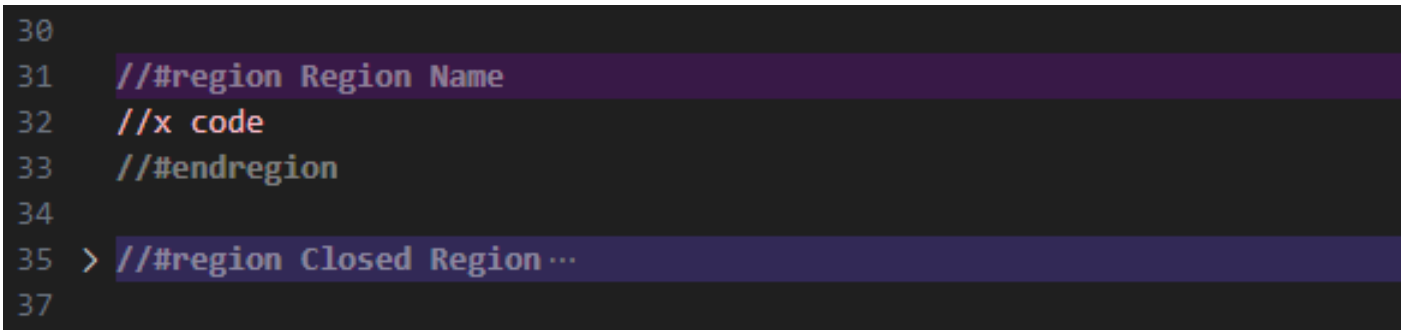


Figure 11.12—11 - shows the change of format, of the region declarations, from the "Region Marker" VS extension.

Not many downloads, but it is a light extension that only makes regions more visible, so reliability does not really matter here.

11.12.2.1.1.2.4 Region Viewer by SantaCodes

Version – *0.1.4*.
VS unique extension identifier - *santacodes.santacodes-region-viewer*.

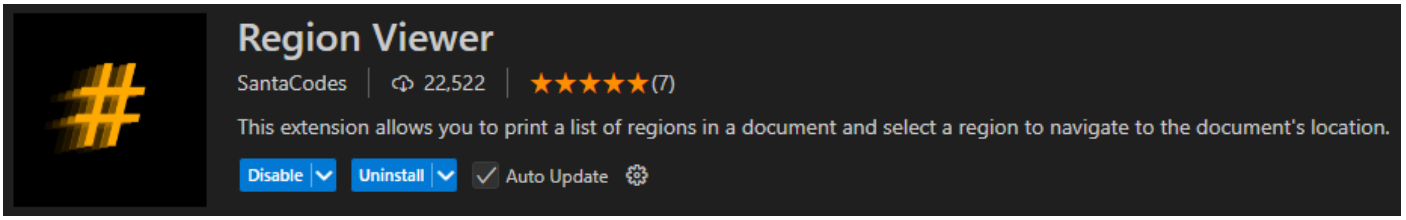


Figure 11.12—12 - shows the overview of the "Region Viewer" VS extension.

Light extension that adds a section in the explorer panel to easily navigate between regions in a code file (includes sub-regions).

This is essential for finding regions in large code files, as it saves a lot of time scrolling and searching.

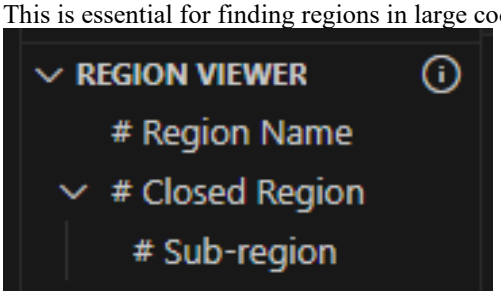


Figure 11.12—13 - shows the section of VS's 'explorer' panel that the "Region Viewer" VS extension adds.

Not many downloads, but it is just an extension that only adds an 'explorer' section, so reliability does not really matter here.

11.12.2.1.2 Design

11.12.2.1.2.1 Instruction set

The instruction set serves two purposes. The first one is for planning, which sets instructions the simulator is expected to execute and how to do so. This plan will prevent the author from going on a tangent and deviating from the LMC standard compatibility. The second one is to be used as part of the user manual so the user knows what opcodes/instructions to use and how to use them.

Name	Code	Mnemonic	Description
Addition	1XX	ADD	Add the memory cell address's value to the accumulator's value.
Subtraction	2XX	SUB	Subtract the memory cell address's value from the accumulator's value.
Store from Accumulator	3XX	STA	Store the accumulator's value in the memory cell address. Mnemonic can also stand for "Store to Accumulator".
Left Shift	401	LSH	Shift the accumulator value's base-10 digits of significance to the left by one digit.
Right Shift	402	RSH	Shift the accumulator value's base-10 digits of significance to the right by one digit.
Load to Accumulator	5XX	LDA	Load the memory address's value into the accumulator (becomes the new accumulator's value).
Branch Always	6XX	BRA	Branch – change PC's value to – the address value (regardless of the accumulator's value).
Branch if Zero	7XX	BRZ	Branch – change PC's value to – the address value if the accumulator's value is zero.
Branch if Positive	8XX	BRP	Branch – change PC's value to – the address value if the accumulator's value is positive or zero (not negative).
Input	901	INP	Takes the user's or predefined input and stores it as the accumulator's value.
Output	902	OUT	Outputs from the accumulator's value (as an integer).
Output as Character	903	OCT	Outputs from accumulator's value as an ASCII character.
Halt	0	HLT	Stops program.
Data location	N/A	DAT	Compile data into memory before the program starts (can be used as variables).

Table 11.12—4 - instruction set plan for sprint #1

11.12.2.1.2.2 IPOD Table

IPOD tables are a less popular variant of the widely used IPO model, where it serves and is structured the same way, but IPOD includes an extra column for decision making (hence to extra "*D*" in the name). IPOD is used instead of IPO due to that extra decision column, especially in sprint 1, as the LMC simulator's fetching, decoding and executing cycles will involve a lot of decision making, as simulators tend to do.

IPOD tables are usually used for UI because they describe how a user interacts with the program. Sprint one, however, does not have a UI. Because of this, sprint #1 would not be included. But the author sees use in making the sprint 1 IPOD for the current code (in CU) – a more detailed improvement, for sprint 1, over the instruction set table.

Inputs	Process	Outputs	Decisions
1XX	Adds the specified memory address (XX) value to the accumulator value.		
2XX	Subtracts the specified memory address (XX) value from accumulator value.		
3XX	Store the accumulator value into the specified memory address (XX) value.		
401	Shift the accumulator value's digits, by one digit, to the left (new digits are zero).		
402	Shift the accumulator value's digits, by one digit, to the right (new digits are zero).		
5XX	Load from the specified memory address (XX) value to the accumulator value.		
6XX	Change PC's value to as specified (XX).		
7XX	Change PC's value to as specified (XX).		Does the accumulator's value equate to zero?
8XX	Change PC's value to as specified (XX).		Is the accumulator's value either positive or zero?

901, predefined input	Stores predefined input to the accumulator value.		
902			Accumulator's value
903	Match the accumulator value to the corresponding visible ASCII character.		Is the ASCII character visible to output (the space whitespace is considered visible)
0	End program		
any value (-999 to 999)	Stores input in the accumulator		

Figure 11.12—14 - IPOD table plan for sprint 1

Note that because sprint 1 does not have a UI, the IPOD table only applies to numerical arguments and returns.

11.12.2.1.2.3 UML diagram

Because the author decided to use OOP, plans suited to this style of programming should also be used. One of the most suited plans for OOP programming is a UML diagram.

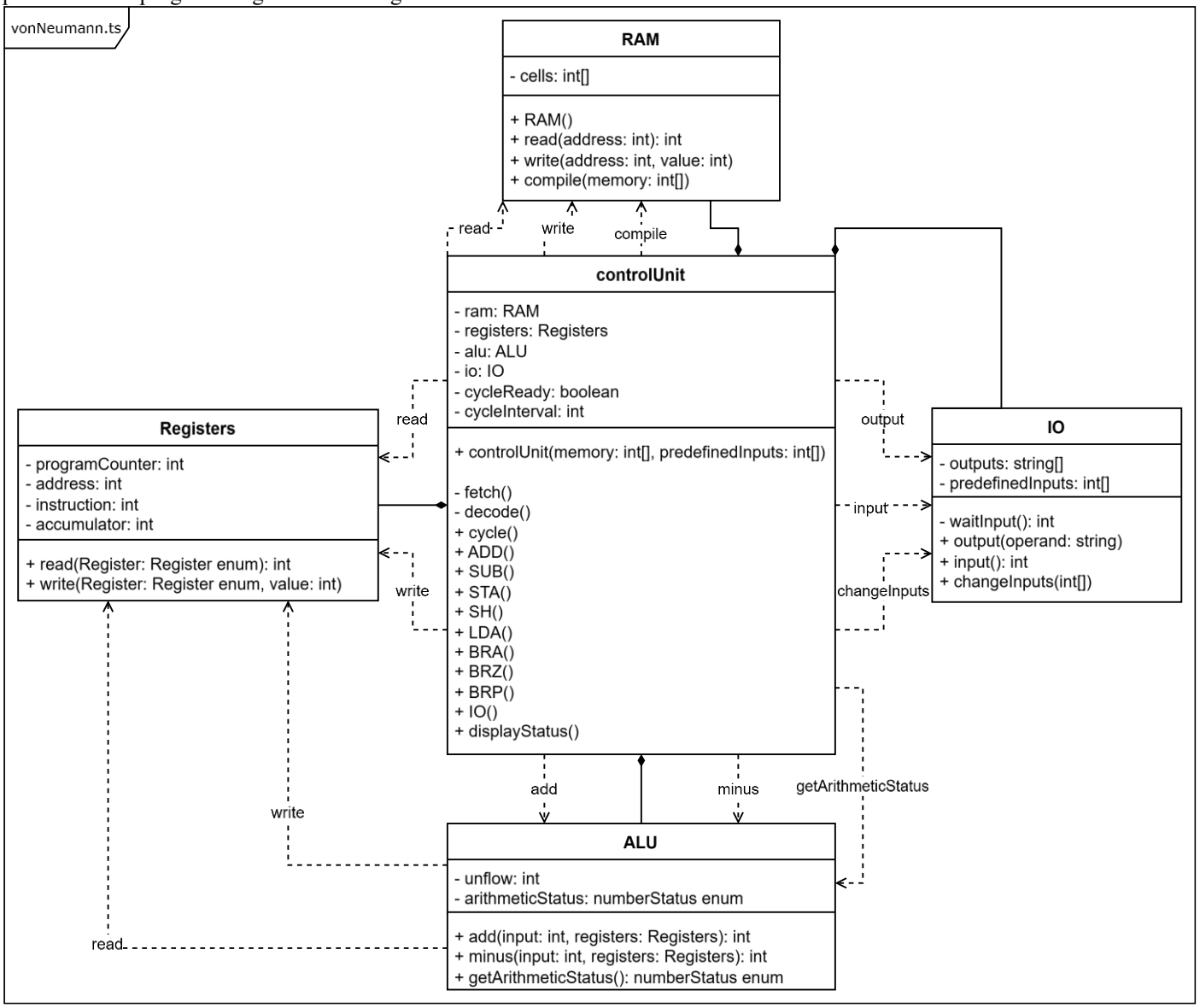


Figure 11.12—15 - UML diagram for `vonNeuman.ts` code of sprint 1.

`controlUnit` originally was not planned to have the instruction methods (`ADD()`, `IO()`, et cetera), but changed when the author was doing the **Flowchart** plan for this sprint.

11.12.2.1.2.4 Flowchart

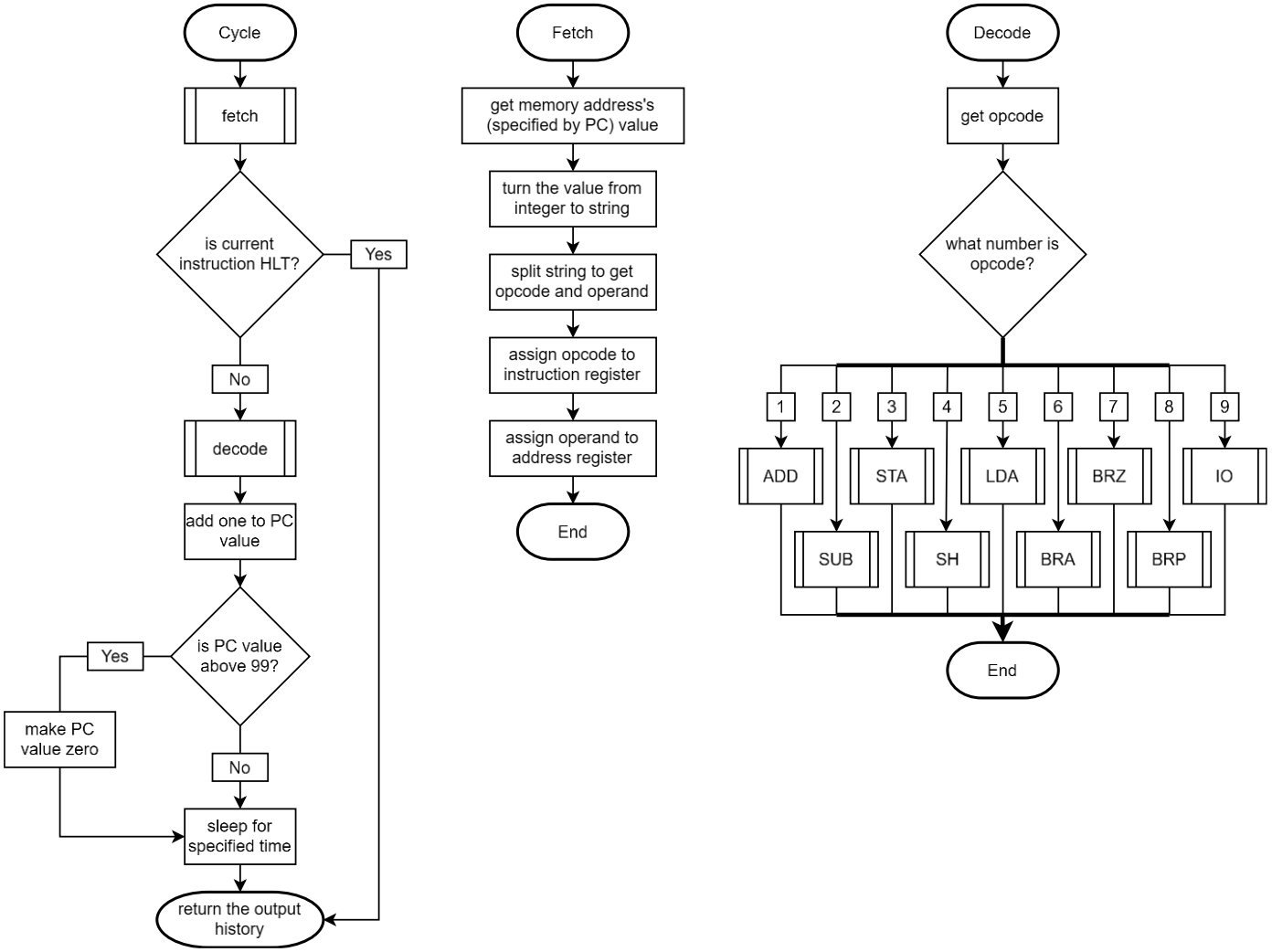


Figure 11.12—16 - flowchart for the cycle method of sprint 1.

When making the flowchart, it was the author's original intent to do all execution of instructions inside a switch-statement of the decode method, but seeing how complex it would make the flowchart, and thus also the later code, the author decided to move each instruction execution to their own method. Each of those methods is named after the mnemonic, except "*IO*" and "*SH*" as they cover a group of similar instructions that are differentiated by the operand/address.

A more imposing problem that using a switch-statement is the different natures of each instruction case and the problems it would cause in code, such as having to declare every commonly used variable, by 2 or more cases, beforehand or having different names for near identical variables.

Please note that the reason for that lack of an execution method is that the instruction sub-routines themselves are the executions.

11.12.2.1.2.5 Styling

Only sprint 1 does not have a UI, thus, this only applies to sprints #2, #3 and beyond.

11.12.2.1.2.6 Wireframe Mock-up

Only the first sprint does not involve implementing UI (focusing on the simulator's workings instead) – making it suitable for only sprints #2, #3 and beyond.

11.12.2.1.2.7 Level-Checking Mechanics

Since level checking mechanics is solely covered in the third sprint, it makes sense that the level table is only for that sprint.

11.12.2.1.2.8 Level Contents

Similar to the level-checking mechanics.

11.12.2.1.2.9 Automatic Testing Structure

The author believed that creating automated (*jest*) test files would be simple enough to do without a dedicated plan because the test files can be easily based on whatever features are coded in the current sprint.

11.12.2.1.2.10 Modification Mapping Table

The modification mapping table serves different purposes depending on the sprint. In the first sprint, the modification mapping provides the developer with a deeper and more detailed plan on how exactly each instruction behaves in the simulator, whereas the IPOD table merely gives an idea of what inputs, processes, outputs, and decisions occur. This allows the author to code more in alignment with the plan.

The table below shows what each marking in the modification mapping table means:

Marking	Meaning
empty	Is not modified.
X	Will be modified – either to a different value or overwritten by the same value.
? [#number]	Conditional modification - property only modified on condition.

Table 11.12—5 - explanation for each cell marking of a modification mapping table.

	Memory cells	Program Counter	Memory Instruction Register	Memory Address Register	Accumulator	Outputs	Pre-defined Inputs
ADD					X		
SUB					X		
STA	X						
LSH					X		
RSH					X		
LDA					X		
BRA		X					
BRZ		? #1					
BRP		? #2					
INP					X		? #3
OUT						X	
OCT						X	
HLT							
DAT							
a F.D.E. cycle		X	X	X			

Table 11.12—6 - modification mapping table for sprint 1.

Note: keep in mind that these properties are only related directly to the simulator and not any property that may be used by the other sprints (UI/middleware) in future sprints, such as `ALU.arithmeticStatus` or `ControlUnit.displayStatus`.
Elaboration on conditional modifications:

Conditional modification #	Condition for modification
#1	The accumulator value is zero.
#2	The accumulator value is zero or positive.
#3	Pre-defined inputs have had at least one available input/element.

Table 11.12—7 - explaining each conditional modification of the modification mapping table.

11.12.2.1.3 Whitebox test case planning

Below is a set tables listing every extremity and aspect/feature of the first sprint that are planned to be tested. These are where the actual test cases will be based on. Justification for each set of tests are also explained in this section. Their implementation is done in the **Test cases** section.

Using Jest, the author has created two automatic testing files.

If the reader wants more information regarding how the testing is done and context regarding the tested data – mnemonic uncompiled script versions – view the `vonNeumann.test.ts` and `LMCCompatability.test.ts` files' content. Explanations and justification for why some tests are included and others are shown as code comments in the mentioned test files.

11.12.2.1.3.1 Testing each instruction

The first one made was the regular testing. It tested each instruction and the extremities, such as overflow and underflow. Other errors, such as invalid input assembly and invalid/out-of-bounds addresses, are covered by the compiler in the second sprint. Thus, those are not covered in this sprint's testing. That test file is called `vonNeumann.test.ts`.

Below are groups of tests, all from `vonNeumann.test.ts`. Due to the high number of tests, and not wanting to have too many tables, each table represents testing on a group of similar-nature instructions (including valid and purposely invalid tests).

What is tested?	How is it tested?	Expected result
Adding 2 and 3 using labels	Takes first input and store it, then get second input. Add stored input to second input then output result. Tested compiled script: `901, 306, 901, 506, 902, 0, 0`. Corresponding pre-defined inputs: `2, 3`.	5
Subtracting 5 from 2	Takes first input and store it, then get second input. Subtract first input from second input then output result. Tested compiled script: `901, 306, 901, 206, 902, 0, 0`. Corresponding pre-defined inputs: `5, 2`.	-3
Adding 2 and 3 without using labels	Takes first input and store it, then get second input. Add stored input to second input then output result. The difference to the other *ADD* (2+3) is that the stored input is in an address beyond the compiled script array – variable is in memory cell address 9 but compiled script array only has 5 elements. Tested compiled script: `901, 309, 901, 109, 902`. Corresponding pre-defined inputs: `2, 3`.	5

Table 11.12—8 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing arithmetic instructions - addition and subtraction (*ADD *and* SUB*).*

What is tested?	How is it tested?	Expected result
Left shifting a 3-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `123`.	230
Right shifting a 3-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `123`.	12
Left shifting a 3-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-123`.	-230
Right shifting a 3-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-123`.	-12
Left shifting a 2-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `45`.	450
Right shifting a 2-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `45`.	4
Left shifting a 2-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-45`.	-450
Right shifting a 2-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-45`.	-4

Left shifting a single digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `6`.	60
Right shifting a single digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `6`.	0
Left shifting a single digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-6`.	-60
Right shifting a single digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-6`.	0

Table 11.12—9 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`***, testing shifting instructions – left shift and right shift (*`LSH` *and* `RSH`*).*

What is tested?	How is it tested?	Expected result
`BRA` (branch always) instruction.	Take input, uses `BRA` instruction to skip the first `HLT` instruction (`0`), allowing the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRA` instruction (`603`) did not work, then nothing would be outputted. Tested compiled script: `901, 603, 0, 902, 0`. Corresponding pre-defined input: `6`.	6
`BRZ` (branch if zero) instruction using value `0`.	Take input, `BRZ` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`0`) being zero. This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRZ` instruction (`703`) was not satisfied, then nothing would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `0`.	0
`BRZ` (branch if zero) instruction using value `1`.	Take input, `BRZ` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`1`) not being zero. This should not allow the output accumulator instruction (`902`) to happen. If the `BRZ` instruction (`703`) was satisfied, then input would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `1`.	Nothing – simulator is expected to not output anything.
`BRZ` (branch if zero) instruction using value `-1`.	Take input, `BRZ` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`-1`) not being zero. This should not allow the output accumulator instruction (`902`) to happen. If the `BRZ` instruction (`703`) was satisfied, then input would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `-1`.	Nothing – simulator is expected to not output anything.
`BRP` (branch if positive) instruction using value `0`.	Take input, `BRP` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`0`) being zero or positive (in this case, it is zero). This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRP` instruction (`803`) was not satisfied, then nothing would be outputted. Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `0`.	0

`BRP` (branch if positive) instruction using value `1`.	Take input, `BRP` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`1`) being zero or positive (in this case, it is positive). This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRP` instruction (`803`) was not satisfied, then nothing would be outputted. Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `0`.	0
`BRP` (branch if zero) instruction using value `-1`.	Take input, `BRP` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`-1`) not being zero nor positive. This should not allow the output accumulator instruction (`902`) to happen. If the `BRP` instruction (`803`) was satisfied, then input would be outputted. Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `-1`.	Nothing – simulator is expected to not output anything.

Table 11.12—10 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing branching instructions – branch always, branch if positive, and branch if zero (*BRA, BRP, *and* BRZ*).*

What is tested?	How is it tested?	Expected result
Inputting the lowest possible number (-999).	Take input (`-999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `-999`.	-999
Inputting the middle number (0).	Take input (`-999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `0`.	0
Inputting the highest possible number (999).	Take input (`999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `999`.	999
Inputting a single digit number.	Take input (`5`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `5`.	5
Inputting a 2-digit number.	Take input (`50`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `50`.	50
Inputting a 3-digit number.	Take input (`500`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `500`.	500
Output positive number	Load variable (`10`), then output it before halting. Tested compiled script: `503, 902, 0, 10`.	10
Output negative number	Load variable (`-10`), then output it before halting. Tested compiled script: `503, 902, 0, -10`.	-10
Outputting an ASCII character	Load variable (`65`), then output it as an ASCII character before halting. Tested compiled script: `503, 903, 0, 65`.	A
Outputting the first visible ASCII character (after the whitespace character).	Load variable (`65`), then output it as an ASCII character before halting. Tested compiled script: `503, 903, 0, 33`.	!

Outputting the last visible ASCII character.	Load variable (`255`), then output it as an ASCII character before halting. Tested compiled script: `503, 903, 0, 255`.	ÿ
Attempting to output an ASCII character with decimal value below the 32 minimum – non-visible/command only.	Attempt to load a non-existent ASCII (not extended) character with the decimal value one above the limit (31). Tested compiled script: `503, 903, 0, 31`.	The 3 characters: `[?]`
Attempting to output an ASCII character with decimal value above the 255 maximum – In the extended ASCII territory.	Attempt to load a non-existent ASCII (not extended) character with the decimal value one above the limit (`256`). Tested compiled script: `503, 903, 0, 256`.	The 3 characters: `[?]`
Attempting to output an ASCII character with decimal value in the negative.	Attempt to load a non-existent ASCII (not extended) character with the decimal value in the negative (`-1`). Tested compiled script: `503, 903, 0, 256`.	The 3 characters: `[?]`

Table 11.12—11 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing IO instructions – input, output and output as character (*`INP`, `OUT`, *and* `OTC`*).*

What is tested?	How is it tested?	Expected result
`STA` (load from accumulator) instruction.	Stores the first of two inputs and load the stored first to overwrite the accumulator's current value of the second input before outputting accumulator's value. Tested compiled script: `901, 306, 901, 506, 902, 0, 0`. Corresponding pre-defined inputs: `10, 20`.	10
`LDA` (load to accumulator) instruction.	Simple loads value from stored variable (in memory) the output result. Tested compiled script: `503, 902, 0, 6`.	6

Table 11.12—12 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing memory instructions – storing and loading between accumulator and memory – (*`STA` *and* `LDA`*).*

The author is aware that the STA test also tests LDA, to be outputted, but having both tests will help the author to pinpoint the reason of any failure and add making testing more rigid.

What is tested?	How is it tested?	Expected result
ALU Underflow	Subtraction operation that causes underflow by subtracting 1 from -999. Tested compiled script: `504, 205, 902, 0, -999, 1`.	999
ALU Overflow	Addition operation that causes overflow by added 1 to 999. Tested compiled script: `504, 105, 902, 0, 999, 1`.	-999
Doubling the highest possible number	Adding the highest value to itself i.e. 999 plus 999. Tested compiled script: `504, 105, 902, 0, 999, 999`.	-1

Table 11.12—13 - plan for test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing how the accumulator deals with extremities (can also be viewed as extremity tests of the arithmetic instruction testing).*

11.12.2.1.3.2 Testing compatibility by example scripts

The second test file focused on making sure that the LMC simulator was up to LMC standard. If it is compatible with other standard LMC simulators, then it is up to standard. This is done by taking the compiled assembly code from example programs, provided by the LMC simulators, to be tested. The other LMC simulators are those that were gone over in the software research part of the dissertation. However, not all example programs are included for various reasons. LMC #6's "*Sort input*" program was not included because it utilised a trait of the simulator that conflicts with the LMC standard (stored values range is 0 to 999 instead of *-999* to *999* - giving different results). Another LMC #6's example program could not be included ("Shift left" program) because it was inaccessible, as seen in below. Meanwhile, LMC #5 was not included at all because the assembly code never compiles and compiling it manually will not make the result authentic. There was one thing that was tolerated, which was the compiled digits. This is because Typescript number lists can only store numbers in the correct format (no zeros before), and having a zero digit before a number does not change its value unless it is a string (which it is not a string). That test file is called **`LMCCompatibility.test.ts`**.

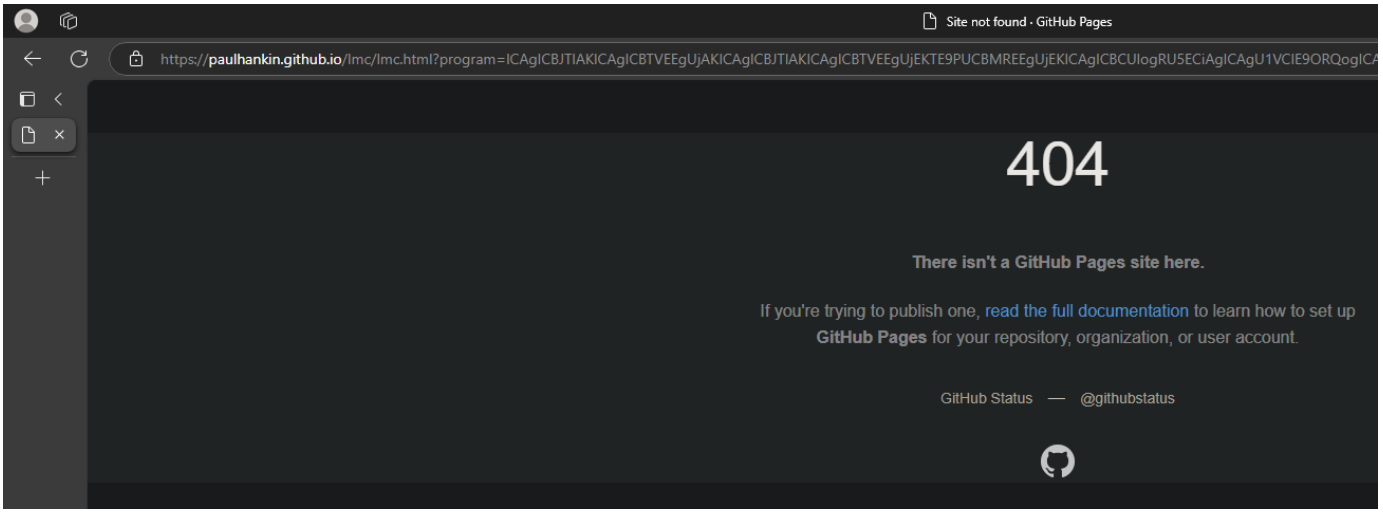


Figure 11.12—17 - 404 error page when attempting to get the "Shift left" example of LMC #5 (Hankin, 2016c).

What is tested?	How is it tested?	Expected result
LMC #1's `"add" program`.	Using compiled script of program: `901, 399, 901, 199, 902, 0`. Add two numbers and output result. With pre-defined inputs: `20, 300`.	320
LMC #1's `"add/subtr" program`.	Using compiled script of program: `901, 309, 901, 109, 902, 901, 209, 902, 0, 0`. Output sum of first 2 inputs and output the result of third input minus the first. With pre-defined inputs: `20, 300, 41`.	320, 21
LMC #1's `"ascii" program`.	Using compiled script of program: `510, 313, 513, 903, 111, 313, 212, 709, 602, 0, 32, 1, 127, 0`. To output visable ASCII characters (including whitespace) from decimal 32 to 127, corrosponding ASCII decimal, and a whitespace (ASCII decimal 32) between the number and ASCII cahracter per set.	" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[", "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~"
LMC #1's `"ascii table" program`.	Using compiled script of program: `514, 317, 517, 902, 514, 903, 517, 903, 115, 317, 216, 713, 602, 0, 32, 1, 97, 0`. To output visable ASCII characters (including whitespace) from decimal 32 to 127.	"32", " ", " ", "33", " ", "!", "34", " ", "\"", "35", " ", "#", "36", " ", "$", "37", " ", "%", "38", " ", "&", "39", " ", "'", "40", " ", "(", "41", " ", ")", "42", " ", "*", "43", " ", "+", "44", " ", ",", "45", " ", "-", "46", " ", ".", "47", " ", "/", "48", " ", "0", "49", " ", "1", "50", " ", "2", "51", " ", "3", "52", " ", "4", "53", " ", "5", "54", " ", "6", "55", " ", "7", "56", " ", "8", "57", " ", "9", "58", " ", ":", "59", " ", ";", "60", " ", "<", "61", " ", "=", "62", " ", ">", "63", " ", "?", "64", " ", "@", "65", " ", "A", "66", " ", "B", "67", " ", "C", "68", " ", "D", "69", " ", "E", "70", " ", "F", "71", " ", "G", "72", " ", "H", "73", " ", "I", "74", " ", "J", "75", " ", "K", "76", " ", "L", "77", " ", "M", "78", " ", "N", "79", " ", "O", "80", " ", "P", "81", " ", "Q", "82", " ", "R", "83", " ", "S", "84", " ", "T", "85", " ", "U", "86", " ", "V", "87", " ", "W", "88", " ", "X", "89", " ", "Y", "90", " ", "Z", "91", " ", "[", "92", " ", "\\", "93", " ", "]", "94", " ", "^", "95", " ", "_", "96", " ", "`"

Table 11.12—14 - plan for test cases, of sprint #1 from `LMCCompatibility.test.ts`, testing how the simulator deals with example compiled scripts from LMC #1 (Peter Higginson's LMC as mentioned in the software research).

What is tested?	How is it tested?	Expected result
LMC #2's `"add" program`.	Using compiled script of program: `901, 306, 901, 106, 902, 0, 0`.	320

	Add two numbers and output result. With pre-defined inputs: `20, 300`.	
LMC #2's `"add/subtr" program`.	Using compiled script of program: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0`. To find which of the two inputs are higher. With pre-defined inputs: `20, 300`.	300
LMC #2's `"ascii" program`.	Using compiled script of program: `901, 902, 308, 207, 308, 801, 0, 1, 0`. To count down from input's value to zero and output each decrement. With pre-defined input: `10`.	10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
LMC #2's `"ascii table" program`.	Using compiled script of program: `901, 316, 901, 317, 519, 116, 319, 517, 218, 317, 804, 519, 216, 319, 902, 0, 0, 0, 1, 0`. To multiply two inputs together. With pre-defined input: `10, 11`.	110
LMC #2's `"add" program`.	Using compiled script of program: `512, 111, 902, 312, 511, 113, 311, 514, 211, 80, 0, 1, 0, 1, 10`. Generates the first 10 terms of the Triangular number pattern outputs each term.	1, 3, 6, 10, 15, 21, 28, 36, 45, 55
LMC #2's `"add/subtr" program`.	Using compiled script of program: `901, 336, 733, 238, 339, 337, 536, 340, 539, 730, 238, 730, 536, 140, 336, 537, 238, 337, 238, 721, 608, 536, 340, 539, 238, 339, 337, 238, 730, 608, 536, 902, 0, 538, 902, 0, 0, 0, 1, 0, 0`. To calculate and output the factorial of given input value. With pre-defined input: `5`.	120

*Table 11.12—15 - plan for test cases, of sprint #1 from **`LMCCompatibility.test.ts`**, testing how the simulator deals with example compiled scripts from LMC #2 (101 Computing's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result
LMC #3's `"takes two inputs and puts their difference in the outbox" program`.	Using compiled script of program: `901, 308, 901, 309, 508, 209, 902, 0, 0, 0`. Subtract the second input from the first input and output result. With pre-defined input: `300, 20`.	280

*Table 11.12—16 - plan for test cases, of sprint #1 from **`LMCCompatibility.test.ts`**, testing how the simulator deals with example compiled scripts from LMC #3 (pbrinkmeier's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result
LMC #4's `"Example 1 - Add two numbers together" program`.	Using compiled script of program: `901, 320, 901, 120, 902, 0, 0`. Adds 2 inputs and output result. With pre-defined input: `600, 40`.	640
LMC #4's `"Example 2 - Output a pattern of 1s and 0s" program`.	Using compiled script of program: `520, 902, 521, 902, 522, 220, 322, 800, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3`. Generate and output a binary pattern (1,0) by 4 times.	1, 0, 1, 0, 1, 0, 1, 0
LMC #4's `"Example 3 - Calculate the square of a number program`.	Using compiled script of program: `901, 330, 533, 331, 332, 531, 130, 331, 532, 134, 332, 230, 814, 605, 531, 902, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1`. Calculate the square of input and output result. With pre-defined input: `9`.	81
LMC #4's `"Example 4 - Integer division`.	Using compiled script of program: `901, 323, 901, 324, 520, 322, 523, 224, 323, 813, 522, 902, 0, 522, 121, 322, 523, 607, 0, 0, 0, 1, 0, 0, 0`.	4

	Performs floor division on first input by second input and outputs result. With pre-defined input: `19, 4`.	

*Table 11.12—17 - plan for test cases, of sprint #1 from **`LMCCompatibility.test.ts`**, testing how the simulator deals with example compiled scripts from LMC #4 (Wellingborough School's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result
LMC #6's `"Max of 3 numbers" program`.	Using compiled script of program: `901, 318, 901, 319, 901, 320, 219, 810, 519, 320, 520, 218, 815, 518, 320, 520, 902, 0, 0, 0, 0`. Takes in 3 inputs then output the highest of them. With pre-defined input: `213, 987, 88`.	987
LMC #6's `"Multiply 2 numbers" program`.	Using compiled script of program: `901, 315, 901, 314, 514, 712, 217, 314, 516, 115, 316, 604, 516, 902, 0, 0, 0, 1`. Takes in 2 inputs then multiply them before outputting the answer. With pre-defined input: `4, 5`.	20
LMC #6's `""Copy N inputs to an array" (self-modifying) program`.	Using compiled script of program: `901, 314, 514, 714, 215, 314, 517, 115, 317, 116, 312, 901, 0, 602, 0, 1, 300, 49`. Takes in inputs and store them in memory as a simulated array. With pre-defined input: `5, 10, 101, 14, 998, 8`.	Nothing. Does not output as it just changes values inside memory. Still included to see if script runs as intended.
LMC #6's `""Sieve of Erastothenes" (self-modifying) program`.	Using compiled script of program: `530, 131, 330, 227, 826, 530, 128, 308, 0, 711, 600, 530, 902, 332, 532, 227, 800, 532, 129, 321, 530, 0, 532, 130, 332, 614, 0, 69, 531, 331, 1, 1, 0`. Uses the Sieve of Eratosthenes algorithm to generate and output all prime numbers up to 67.	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67

*Table 11.12—18 - plan for test cases, of sprint #1 from **`LMCCompatibility.test.ts`**, testing how the simulator deals with example compiled scripts from LMC #6 (Paul's blog's LMC as mentioned in the software research).*

11.12.2.1.4 Critical path analysis

Little Man Computer educational game

Project by - James Haddad

Supervisor - Tina Eager

Project Start:	Tue, 10/1/2024
Project End:	Mon, 5/12/2025
Display Week:	1

TASK	STATUS	START	END
Literature review	Fully done		
Academic research	Fully done	10/11/24	11/24/24
Software research	Fully done	10/21/24	12/8/24
Create poster for submission	Fully done	12/9/24	1/17/25
Requirement analysis	Fully done	11/25/24	11/26/24
Technologies used	Fully done	11/27/24	12/1/24
Priorities	Fully done	12/2/24	12/8/24
Prepare and perform poster presentation	Fully done	1/18/25	1/21/25
Sprint 1	Currently doing		
Pre-sprint preparations	Fully done	1/20/25	1/26/25
Prior research	Fully done	1/27/25	2/2/25
Design	Fully done	2/3/25	2/9/25
Critical path analysis	Currently doing	2/10/25	2/16/25
Developmment - implementation	haven't started	2/17/25	2/23/25
Developmment - testing	haven't started	2/24/25	2/26/25
Review and self-reflection	haven't started	2/27/25	3/2/25
Sprint 2			
Prior research	haven't started	2/24/25	3/2/25
Design	haven't started	3/3/25	3/9/25
Critical path analysis	haven't started	3/10/25	3/16/25
Developmment - implementation	haven't started	3/17/25	3/23/25
Developmment - testing	haven't started	3/24/25	3/26/25
Review and self-reflection	haven't started	3/27/25	3/30/25
Sprint 3			
Prior research	haven't started	3/24/25	3/30/25
Design	haven't started	3/31/25	4/6/25
Critical path analysis	haven't started	4/7/25	4/13/25
Developmment - implementation	haven't started	4/14/25	4/20/25
Developmment - testing	haven't started	4/21/25	4/23/25
Review and self-reflection	haven't started	4/24/25	4/27/25
Post sprint 3			
Project review and reflection	haven't started	4/21/25	4/29/25
Client testing	haven't started	4/27/25	5/5/25
Polishing	haven't started	5/6/25	5/12/25

Figure 11.12—18 - current state of the GANTT chart tasks at this moment in sprint #1.

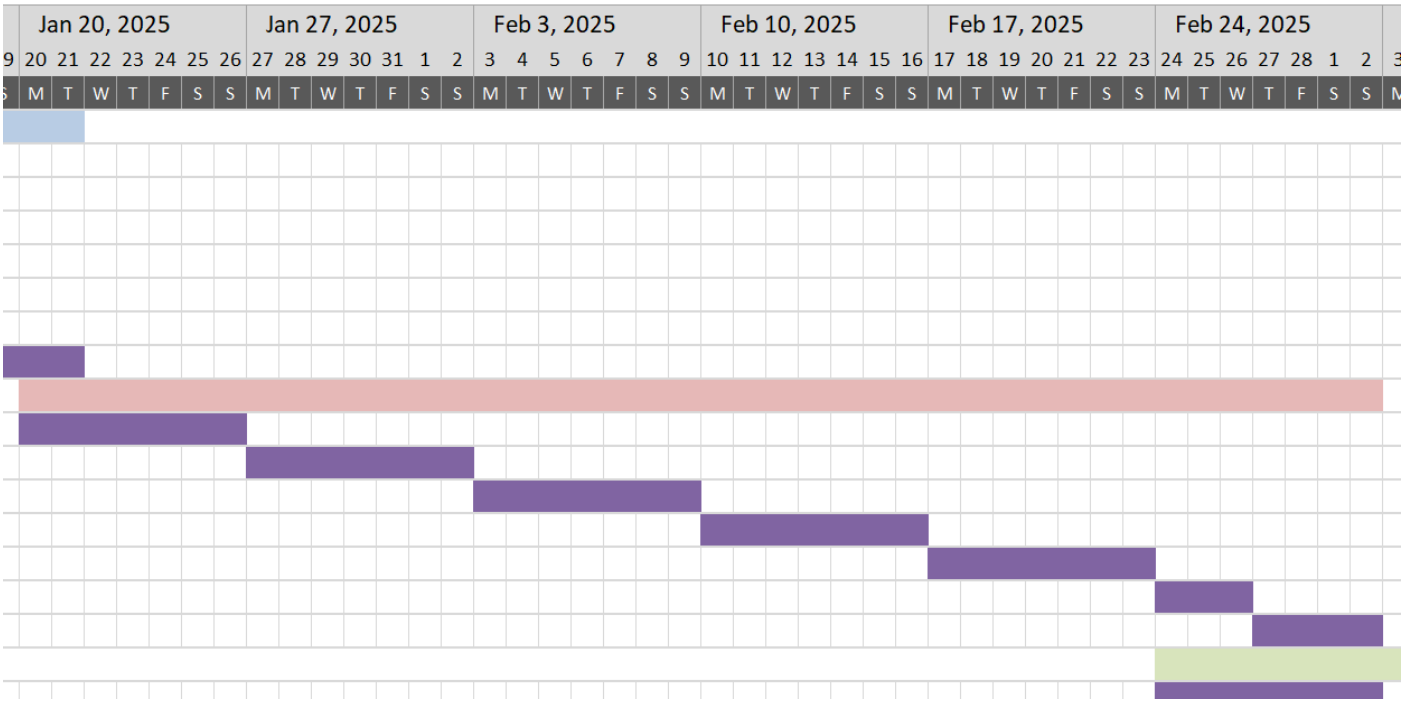


*Figure 11.12—19 - view of GANTT tasks duration and deadlines corresponding to sprint #1 GANTT tasks in **Figure 11.12—19**.*

With all previous tasks fully completed and a bit earlier than predicted time, there is currently no problems and will likely continue into sprint #1. Not much can be said here because this is only the first sprint and so far, no obstacles. Because of this, sprint #1 is expected to finish by (or before) the deadline and start the second sprint on time, both sets of deadlines listed on **Figure 11.12—18**.

11.12.2.2 Development

11.12.2.2.1 Main features

- LMC simulator that can fetch, decode, and execute compiled assembly code (`vonNeumann.ts`).
- Automatic (jest) file that makes sure that LMC is up to LMC standard (`LMCCompatability.test.ts`).
- Automatic (jest) file that makes sure that LMC does not encounter any logic nor run-time errors. (`LMCCompatability.test.ts`).
- More configuration files.

11.12.2.2.2 Current project directory

Currently project directory is visually shown below:

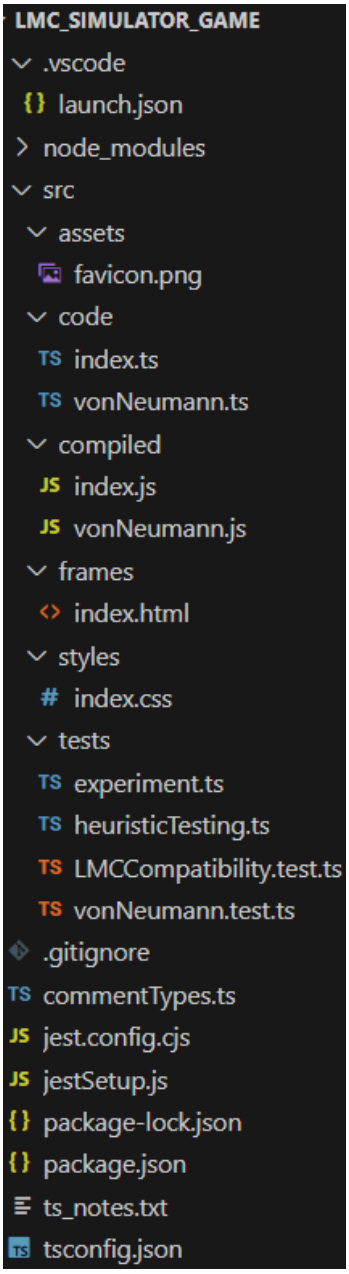


Figure 11.12—20 screenshot snippet of the project's directory of sprint 1 (showing every file except those in `node_modules` as they are just node.js dependencies).

This primarily consists of:

- Configuration files
- Test files
- Simulator files
- Compiled files
- Frontend technology files (from pre-sprint testing)

Note that the test TS script (`.test.ts`) is not compiled because the author uses the ts-node and ts-jest VS extensions. The other TS files are compiled for the sole purpose of checking if the TS program compiles to JS files properly.

11.12.2.2.3 Test cases

Test cases done based upon the test plan from sprint #1's **Whitebox test case planning** section, justification and explanation are also done there.

11.12.2.2.3.1 Testing each instruction

What is tested?	How is it tested?	Expected result	Actual result	Changes made

Adding 2 and 3 using labels	Takes first input and store it, then get second input. Add stored input to second input then output result. Tested compiled script: `901, 306, 901, 506, 902, 0, 0`. Corresponding pre-defined inputs: `2, 3`.	5	5	None. (actual is expected)
Subtracting 5 from 2	Takes first input and store it, then get second input. Subtract first input from second input then output result. Tested compiled script: `901, 306, 901, 206, 902, 0, 0`. Corresponding pre-defined inputs: `5, 2`.	-3	-3	None. (actual is expected)
Adding 2 and 3 without using labels	Takes first input and store it, then get second input. Add stored input to second input then output result. The difference to the other *ADD* (2+3) is that the stored input is in an address beyond the compiled script array – variable is in memory cell address 9 but compiled script array only has 5 elements. Tested compiled script: `901, 309, 901, 109, 902`. Corresponding pre-defined inputs: `2, 3`.	5	5	None. (actual is expected)

Table 11.12—19 - test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing arithmetic instructions - addition and subtraction (*ADD *and* SUB*).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
Left shifting a 3-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `123`.	230	230	None. (actual is expected)
Right shifting a 3-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `123`.	12	12	None. (actual is expected)
Left shifting a 3-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-123`.	-230	-230	None. (actual is expected)
Right shifting a 3-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-123`.	-12	-12	None. (actual is expected)
Left shifting a 2-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `45`.	450	450	None. (actual is expected)
Right shifting a 2-digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `45`.	4	4	None. (actual is expected)
Left shifting a 2-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-45`.	-450	-450	None. (actual is expected)
Right shifting a 2-digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-45`.	-4	-4	None. (actual is expected)

Left shifting a single digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `6`.	60	60	None. (actual is expected)
Right shifting a single digit positive number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `6`.	0	0	None. (actual is expected)
Left shifting a single digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-6`.	-60	-60	None. (actual is expected)
Right shifting a single digit negative number.	Take input, left shift it by one digit then output the result. Tested compiled script: `901, 401, 902`. Corresponding pre-defined input: `-6`.	0	0	None. (actual is expected)

*Table 11.12—20 - test cases, of sprint #1 from **vonNeumann.test.ts**, testing shifting instructions – left shift and right shift (LSH and RSH).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
`BRA` (branch always) instruction.	Take input, uses `BRA` instruction to skip the first `HLT` instruction (`0`), allowing the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRA` instruction (`603`) did not work, then nothing would be outputted. Tested compiled script: `901, 603, 0, 902, 0`. Corresponding pre-defined input: `6`.	6	6	None. (actual is expected)
`BRZ` (branch if zero) instruction using value `0`.	Take input, `BRZ` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`0`) being zero. This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRZ` instruction (`703`) was not satisfied, then nothing would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `0`.	0	0	None. (actual is expected)
`BRZ` (branch if zero) instruction using value `1`.	Take input, `BRZ` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`1`) not being zero. This should not allow the output accumulator instruction (`902`) to happen. If the `BRZ` instruction (`703`) was satisfied, then input would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `1`.	Nothing – simulator is expected to not output anything.	Same as expected result.	None. (actual is expected)
`BRZ` (branch if zero) instruction using value `-1`.	Take input, `BRZ` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`-1`) not being zero. This should not allow the output accumulator instruction (`902`) to happen. If the `BRZ` instruction (`703`) was satisfied, then input would be outputted. Tested compiled script: `901, 703, 0, 902, 0`. Corresponding pre-defined input: `-1`.	Nothing – simulator is expected to not output anything.	Same as expected result.	None. (actual is expected)
`BRP` (branch if positive) instruction using value `0`.	Take input, `BRP` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`0`) being zero or positive (in this case, it is zero). This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRP` instruction (`803`) was not satisfied, then nothing would be outputted.	0	0	None. (actual is expected)

	Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `0`.			
`BRP` (branch if positive) instruction using value `1`.	Take input, `BRP` instruction skips the first `HLT` instruction (`0`) due to accumulator being the input value (`1`) being zero or positive (in this case, it is positive). This should allow the output accumulator instruction (`902`) to happen before executing the other `HLT` instruction. If the `BRP` instruction (`803`) was not satisfied, then nothing would be outputted. Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `0`.	0	0	None. (actual is expected)
`BRP` (branch if zero) instruction using value `-1`.	Take input, `BRP` instruction does not skip the first `HLT` instruction (`0`) due to accumulator being the input value (`-1`) not being zero nor positive. This should not allow the output accumulator instruction (`902`) to happen. If the `BRP` instruction (`803`) was satisfied, then input would be outputted. Tested compiled script: `901, 803, 0, 902, 0`. Corresponding pre-defined input: `-1`.	Nothing – simulator is expected to not output anything.	Same as expected result.	None. (actual is expected)

Table 11.12—21 - test cases, of sprint #1 from **`vonNeumann.test.ts`**, *testing branching instructions – branch always, branch if positive, and branch if zero (*BRA, BRP, *and* BRZ*).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
Inputting the lowest possible number (-999).	Take input (`-999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `-999`.	-999	-999	None. (actual is expected)
Inputting the middle number (0).	Take input (`-999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `0`.	0	0	None. (actual is expected)
Inputting the highest possible number (999).	Take input (`999`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `999`.	999	999	None. (actual is expected)
Inputting a single digit number.	Take input (`5`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `5`.	5	5	None. (actual is expected)
Inputting a 2-digit number.	Take input (`50`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `50`.	50	50	None. (actual is expected)
Inputting a 3-digit number.	Take input (`500`), then output it before halting. Tested compiled script: `901, 902, 0`. Corresponding pre-defined input: `500`.	500	500	None. (actual is expected)
Output positive number	Load variable (`10`), then output it before halting. Tested compiled script: `503, 902, 0, 10`.	10	10	None. (actual is expected)
Output negative number	Load variable (`-10`), then output it before halting. Tested compiled script: `503, 902, 0, -10`.	-10	-10	None. (actual is expected)
Outputting an ASCII character	Load variable (`65`), then output it as an ASCII character before halting.	A	A	None. (actual is expected)

	Tested compiled script: `503, 903, 0, 65`.			
Outputting the first visible ASCII character (after the whitespace character).	Load variable (`65`), then output it as an ASCII character before halting. Tested compiled script: `503, 903, 0, 33`.	!	!	None. (actual is expected)
Outputting the last visible ASCII character.	Load variable (`255`), then output it as an ASCII character before halting. Tested compiled script: `503, 903, 0, 255`.	ÿ	ÿ	None. (actual is expected)
Attempting to output an ASCII character with decimal value below the 32 minimum – non-visible/command only.	Attempt to load a non-existent ASCII (not extended) character with the decimal value one above the limit (31). Tested compiled script: `503, 903, 0, 31`.	The 3 characters: `[?]`	`[?]`	None. (actual is expected)
Attempting to output an ASCII character with decimal value above the 255 maximum – In the extended ASCII territory.	Attempt to load a non-existent ASCII (not extended) character with the decimal value one above the limit (`256`). Tested compiled script: `503, 903, 0, 256`.	The 3 characters: `[?]`	`[?]`	None. (actual is expected)
Attempting to output an ASCII character with decimal value in the negative.	Attempt to load a non-existent ASCII (not extended) character with the decimal value in the negative (`-1`). Tested compiled script: `503, 903, 0, 256`.	The 3 characters: `[?]`	`[?]`	None. (actual is expected)

*Table 11.12—22 - test cases, of sprint #1 from **vonNeumann.test.ts**, testing IO instructions – input, output and output as character (INP, OUT, and OTC).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
`STA` (load from accumulator) instruction.	Stores the first of two inputs and load the stored first to overwrite the accumulator's current value of the second input before outputting accumulator's value. Tested compiled script: `901, 306, 901, 506, 902, 0, 0`. Corresponding pre-defined inputs: `10, 20`.	10	10	None. (actual is expected)
`LDA` (load to accumulator) instruction.	Simple loads value from stored variable (in memory) the output result. Tested compiled script: `503, 902, 0, 6`.	6	6	None. (actual is expected)

*Table 11.12—23 - test cases, of sprint #1 from **vonNeumann.test.ts**, testing memory instructions – storing and loading between accumulator and memory – (STA and LDA).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
ALU Underflow	Subtraction operation that causes underflow by subtracting 1 from -999. Tested compiled script: `504, 205, 902, 0, -999, 1`.	999	999	None. (actual is expected)
ALU Overflow	Addition operation that causes overflow by added 1 to 999. Tested compiled script: `504, 105, 902, 0, 999, 1`.	-999	-999	None. (actual is expected)
Doubling the highest possible number	Adding the highest value to itself i.e. 999 plus 999. Tested compiled script: `504, 105, 902, 0, 999, 999`.	-1	-1	None. (actual is expected)

*Table 11.12—24 - test cases, of sprint #1 from **vonNeumann.test.ts**, testing how the accumulator deals with extremities (can also be viewed as extremity tests of the arithmetic instruction testing).*

11.12.2.2.3.2 Testing compatibility by example scripts

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #1's *"add" program*.	Using compiled script of program: *901, 399, 901, 199, 902, 0*. Add two numbers and output result. With pre-defined inputs: *20, 300*.	320	320	None. (actual is expected)
LMC #1's *"add/subtr" program*.	Using compiled script of program: *901, 309, 901, 109, 902, 901, 209, 902, 0, 0*. Output sum of first 2 inputs and output the result of third input minus the first. With pre-defined inputs: *20, 300, 41*.	320, 21	320, 21	None. (actual is expected)
LMC #1's *"ascii" program*.	Using compiled script of program: *510, 313, 513, 903, 111, 313, 212, 709, 602, 0, 32, 1, 127, 0*. To output visable ASCII characters (including whitespace) from decimal 32 to 127, corrosponding ASCII decimal, and a whitespace (ASCII decimal 32) between the number and ASCII cahracter per set.	" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[", "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~"	Same as expected result. (too large to repeat result here)	None. (actual is expected)
LMC #1's *"ascii table" program*.	Using compiled script of program: *514, 317, 517, 902, 514, 903, 517, 903, 115, 317, 216, 713, 602, 0, 32, 1, 97, 0*. To output visable ASCII characters (including whitespace) from decimal 32 to 127.	"32", " ","' ", "33", " ", "!", "34", " ", "\"", "35", " ", "#", "36", " ", "$", "37", " ", "%", "38", " ", "&", "39", " ", "'", "40", " ", "(", "41", " ", ")", "42", " ", "*", "43", " ", "+", "44", " ", ",", "45", " ", "-", "46", " ", ".", "47", " ", "/", "48", " ", "0", "49", " ", "1", "50", " ", "2", "51", " ", "3", "52", " ", "4", "53", " ", "5", "54", " ", "6", "55", " ", "7", "56", " ", "8", "57", " ", "9", "58", " ", ":", "59", " ", ";", "60", " ", "<", "61", " ", "=", "62", " ", ">", "63", " ", "?", "64", " ", "@", "65", " ", "A", "66", " ", "B", "67", " ", "C", "68", " ", "D", "69", " ", "E", "70", " ", "F", "71", " ", "G", "72", " ", "H", "73", " ", "I", "74", " ", "J", "75", " ", "K", "76", " ", "L", "77", " ", "M", "78", " ", "N", "79", " ", "O", "80", " ", "P", "81", " ", "Q", "82", " ", "R", "83", " ", "S", "84", " ", "T", "85", " ", "U", "86", " ", "V", "87", " ", "W", "88", " ", "X", "89", " ", "Y", "90", " ", "Z", "91", " ", "[", "92", " ", "\\", "93", " ", "]", "94", " ", "^", "95", " ", "_", "96", " ", "`"	Same as expected result. (too large to repeat result here)	None. (actual is expected)

*Table 11.12—25 - test cases, of sprint #1 from **LMCCompatibility.test.ts**, testing how the simulator deals with example compiled scripts from LMC #1 (Peter Higginson's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #2's *"add" program*.	Using compiled script of program: *901, 306, 901, 106, 902, 0, 0*. Add two numbers and output result. With pre-defined inputs: *20, 300*.	320	320	None. (actual is expected)

LMC #2's *"add/subtr" program*.	Using compiled script of program: *901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0*. To find which of the two inputs are higher. With pre-defined inputs: *20, 300*.	300	300	None. (actual is expected)
LMC #2's *"ascii" program*.	Using compiled script of program: *901, 902, 308, 207, 308, 801, 0, 1, 0*. To count down from input's value to zero and output each decrement. With pre-defined input: *10*.	10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0	10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0	None. (actual is expected)
LMC #2's *"ascii table" program*.	Using compiled script of program: *901, 316, 901, 317, 519, 116, 319, 517, 218, 317, 804, 519, 216, 319, 902, 0, 0, 0, 1, 0*. To multiply two inputs together. With pre-defined input: *10, 11*.	110	110	None. (actual is expected)
LMC #2's *"add" program*.	Using compiled script of program: *512, 111, 902, 312, 511, 113, 311, 514, 211, 80, 0, 1, 0, 1, 10*. Generates the first 10 terms of the Triangular number pattern outputs each term.	1, 3, 6, 10, 15, 21, 28, 36, 45, 55	1, 3, 6, 10, 15, 21, 28, 36, 45, 55	None. (actual is expected)
LMC #2's *"add/subtr" program*.	Using compiled script of program: *901, 336, 733, 238, 339, 337, 536, 340, 539, 730, 238, 730, 536, 140, 336, 537, 238, 337, 238, 721, 608, 536, 340, 539, 238, 339, 337, 238, 730, 608, 536, 902, 0, 538, 902, 0, 0, 0, 1, 0, 0*. To calculate and output the factorial of given input value. With pre-defined input: *5*.	120	120	None. (actual is expected)

*Table 11.12—26 - test cases, of sprint #1 from **LMCCompatibility.test.ts**, testing how the simulator deals with example compiled scripts from LMC #2 (101 Computing's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #3's *"takes two inputs and puts their difference in the outbox" program*.	Using compiled script of program: *901, 308, 901, 309, 508, 209, 902, 0, 0, 0*. Subtract the second input from the first input and output result. With pre-defined input: *300, 20*.	280	280	None. (actual is expected)

*Table 11.12—27 - test cases, of sprint #1 from **LMCCompatibility.test.ts**, testing how the simulator deals with example compiled scripts from LMC #3 (pbrinkmeier's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #4's *"Example 1 - Add two numbers together" program*.	Using compiled script of program: *901, 320, 901, 120, 902, 0, 0*. Adds 2 inputs and output result. With pre-defined input: *600, 40*.	640	640	None. (actual is expected)
LMC #4's *"Example 2 - Output a pattern of 1s and 0s" program*.	Using compiled script of program: *520, 902, 521, 902, 522, 220, 322, 800, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3*. Generate and output a binary pattern (1,0) by 4 times.	1, 0, 1, 0, 1, 0, 1, 0	1, 0, 1, 0, 1, 0, 1, 0	None. (actual is expected)
LMC #4's *"Example 3 - Calculate the square of a*	Using compiled script of program: *901, 330, 533, 331, 332, 531, 130, 331, 532, 134, 332, 230, 814, 605, 531, 902, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1*.	81	81	None. (actual is expected)

number program.	Calculate the square of input and output result. With pre-defined input: *9*.			
LMC #4's *"Example 4 - Integer division*.	Using compiled script of program: *901, 323, 901, 324, 520, 322, 523, 224, 323, 813, 522, 902, 0, 522, 121, 322, 523, 607, 0, 0, 0, 1, 0, 0, 0*. Performs floor division on first input by second input and outputs result. With pre-defined input: *19, 4*.	4	4	None. (actual is expected)

Table 11.12—28 - test cases, of sprint #1 from ***LMCCompatibility.test.ts****, testing how the simulator deals with example compiled scripts from LMC #4 (Wellingborough School's LMC as mentioned in the software research).*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #6's *"Max of 3 numbers" program*.	Using compiled script of program: *901, 318, 901, 319, 901, 320, 219, 810, 519, 320, 520, 218, 815, 518, 320, 520, 902, 0, 0, 0, 0*. Takes in 3 inputs then output the highest of them. With pre-defined input: *213, 987, 88*.	987	987	None. (actual is expected)
LMC #6's *"Multiply 2 numbers" program*.	Using compiled script of program: *901, 315, 901, 314, 514, 712, 217, 314, 516, 115, 316, 604, 516, 902, 0, 0, 0, 1*. Takes in 2 inputs then multiply them before outputting the answer. With pre-defined input: *4, 5*.	20	20	None. (actual is expected)
LMC #6's *""Copy N inputs to an array" (self-modifying) program*.	Using compiled script of program: *901, 314, 514, 714, 215, 314, 517, 115, 317, 116, 312, 901, 0, 602, 0, 1, 300, 49*. Takes in inputs and store them in memory as a simulated array. With pre-defined input: *5, 10, 101, 14, 998, 8*.	Nothing. Does not output as it just changes values inside memory. Still included to see if script runs as intended.	Same as expected result.	None. (actual is expected)
LMC #6's *""Sieve of Erastothenes" (self-modifying) program*.	Using compiled script of program: *530, 131, 330, 227, 826, 530, 128, 308, 0, 711, 600, 530, 902, 332, 532, 227, 800, 532, 129, 321, 530, 0, 532, 130, 332, 614, 0, 69, 531, 331, 1, 1, 0*. Uses the Sieve of Eratosthenes algorithm to generate and output all prime numbers up to 67.	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67	2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67	None. (actual is expected)

Table 11.12—29 - test cases, of sprint #1 from ***LMCCompatibility.test.ts****, testing how the simulator deals with example compiled scripts from LMC #6 (Paul's blog's LMC as mentioned in the software research).*

11.12.2.2.3.3 Current state of tests

Even in the first sprint, where it does not have a UI, there is still a large number of tests. So much that the author can only display the test groups and some individual tests in the IDE, as seen in **Figure 11.12—21** below.

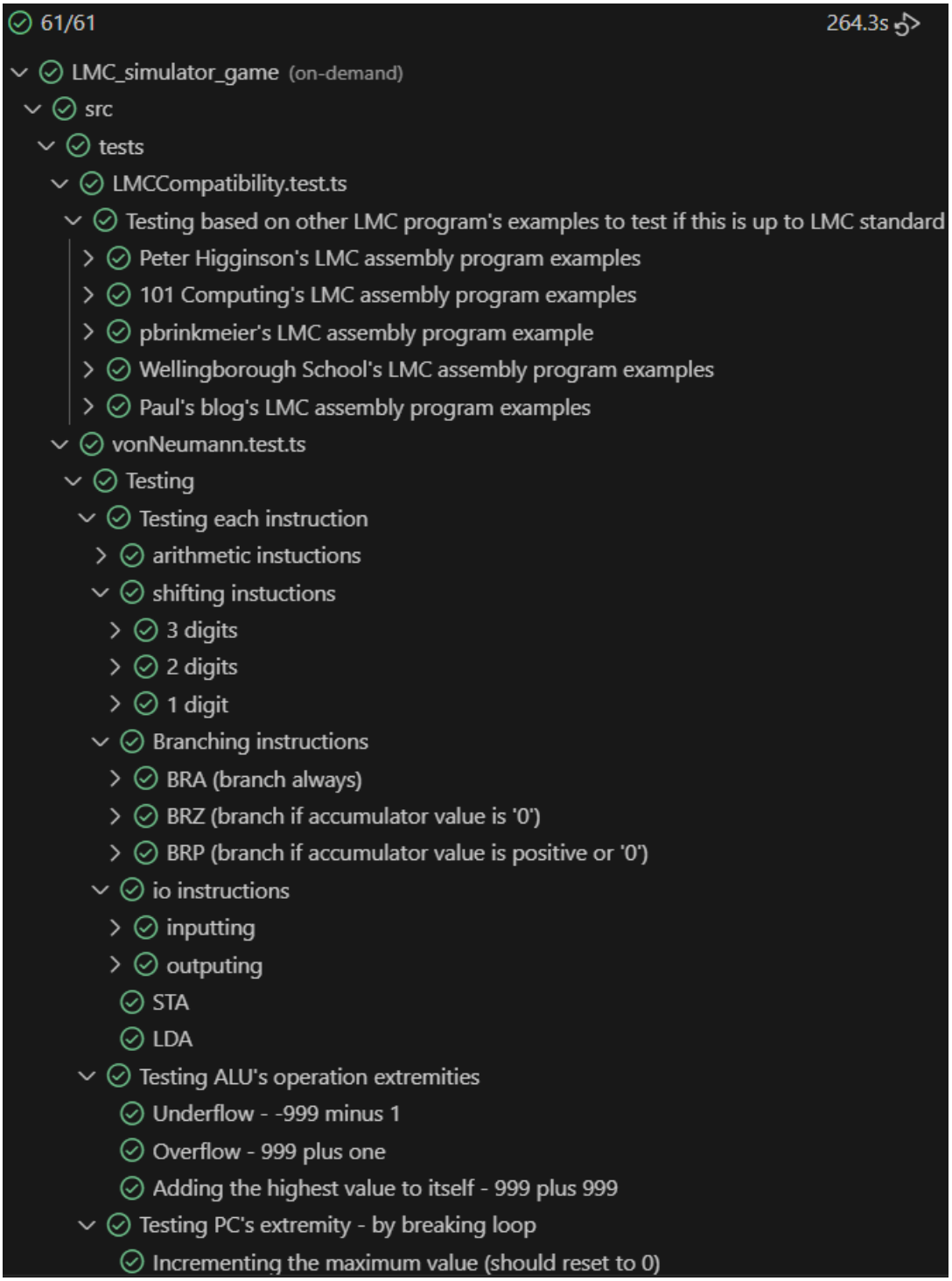


Figure 11.12—21 - all group tests in first sprint (and a few individual tests) after applying solutions to the testing "problems".

The reader may be concerned about the "*264.3*" seconds completion time, but rest assured that is to be expected as there are a total of 62 times the compiled assembly program gets executed, in addition to some compiled assembly programs being far more complex than others. To demonstrate the author's point, the automatic tests are done via the terminal and screenshotted as **Figure 11.12—22** and **Figure 11.12—23** seen below, to prove the author's point.

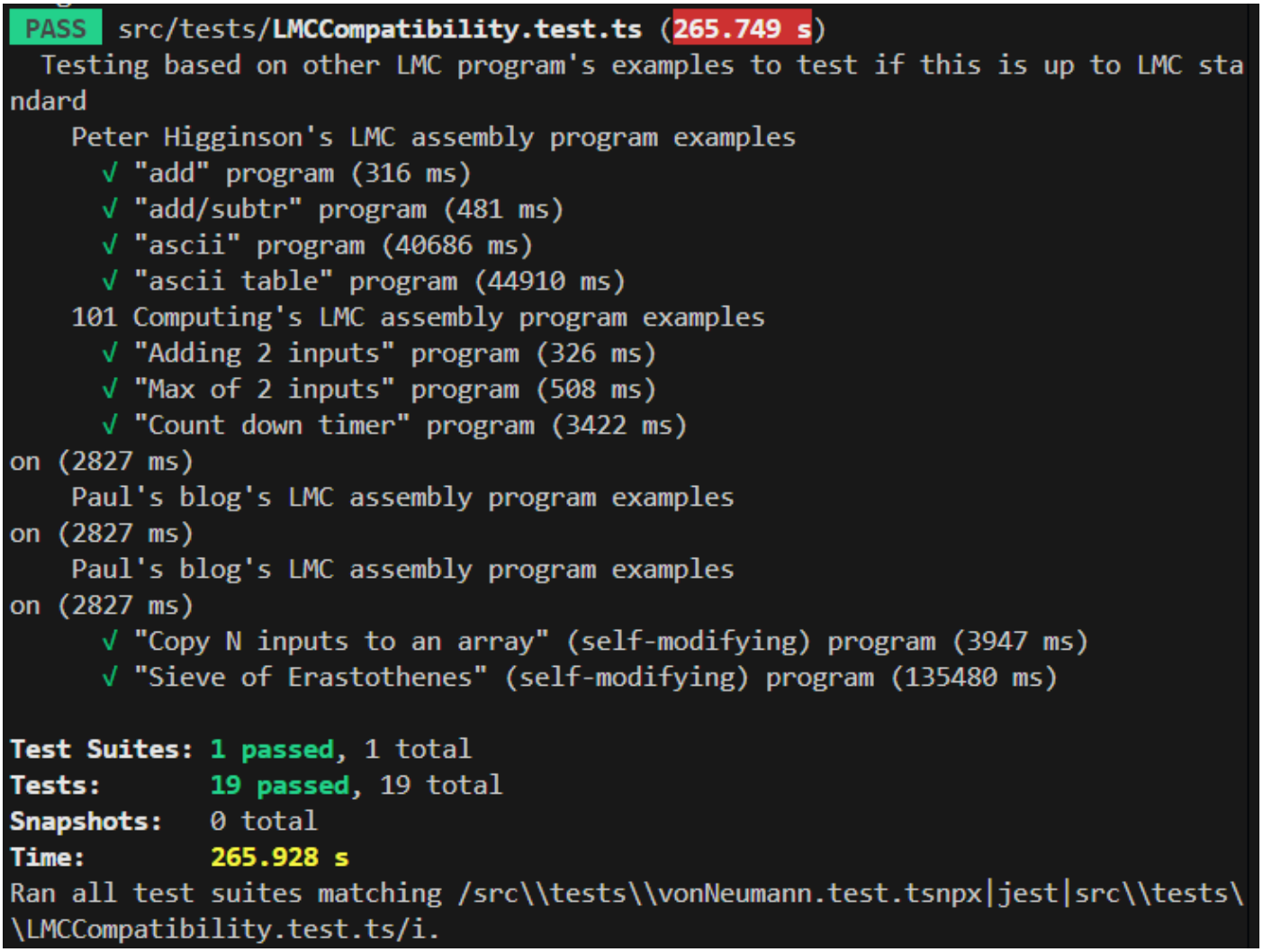


Figure 11.12—22 - screenshot snipped of all individual tests executed in the ***`LMCCompatibility.test.ts`*** *automatic test file. This was done using the command* `npx jest src/tests/LMCCompatibility.test.ts`.

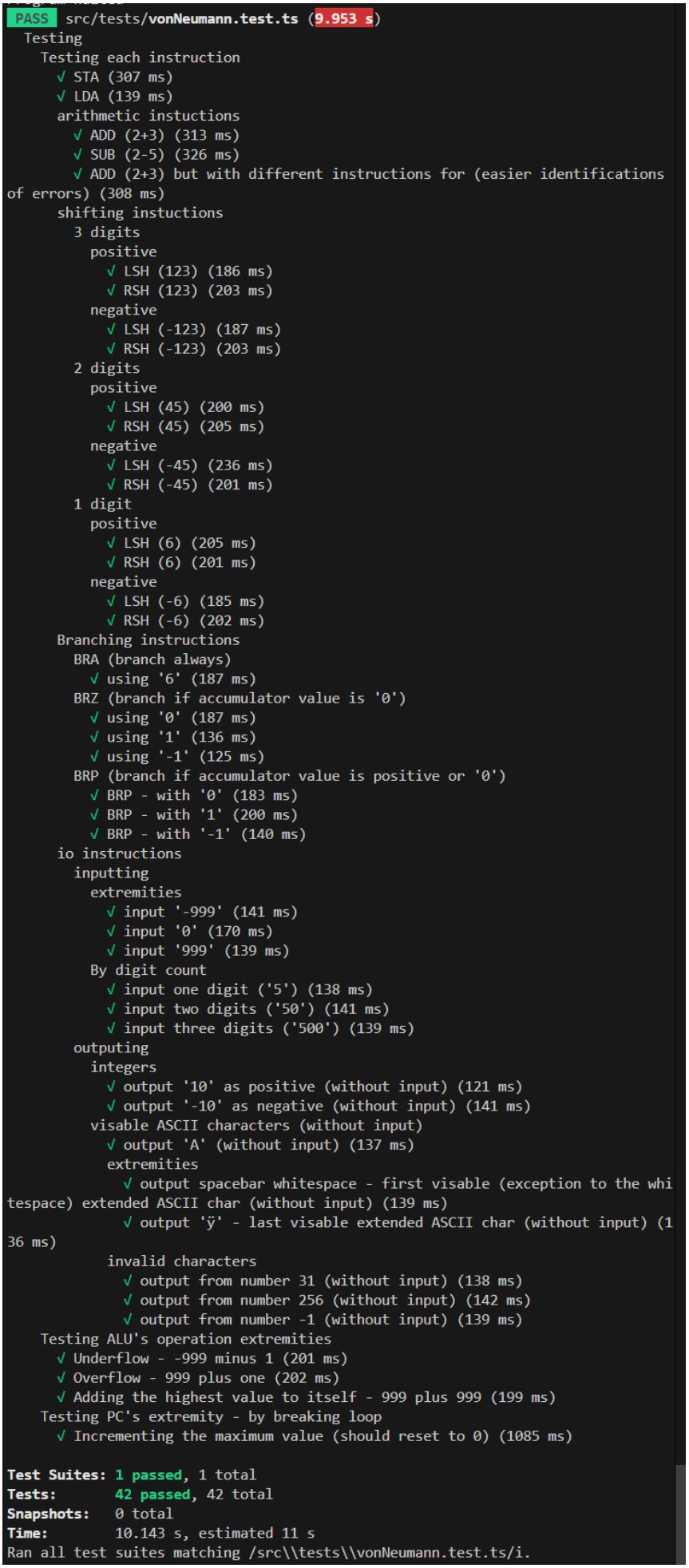


Figure 11.12—23 - Screenshot snipped of all individual tests executed in the ***vonNeumann.test.ts*** *automatic test file. This was done using the command* `npx jest src/tests/vonNeumann.test.ts`.

11.12.2.2.3.4 Testing problems and solutions

Using Jest features, the tests are grouped into described sections, which can have nested sections. These sections and descriptions allow the author and any other developer to easily understand and know what features and extremities are tested.
During the creation and debugging of the test, there were some inconveniences. First was finding exact details of specific tests, such as expected output vs actual output, and the second was testing specific tests without having to spend much time chaining the `.skip` method to every other group of tests and individual tests inside the group of that specific test.
To solve these problems, two VS extensions have been added.

- The first one is Jest by Orta (unique ID - *orta.vscode-jest*, version – 6.4.0).

Figure 11.12—24 - Shows the overview of the Jest VS extension by "orta".

This extension helps with two things. The first one has a VS side tab, displaying every test (in the project) by name and its status (passed, failed, testing, or has not been tested yet). This is much better than having to navigate through multiple test files. The Second thing it helps with is displaying details regarding why a test has failed, when hovering (the cursor) over a test declaration, which is referred to by the list of tests. This helps significantly with instantly knowing which tests failed and why, allowing rapid prototyping. This deals with the first problem.

Figure 11.12—25, below, shows the list of tests (on the left) and the viewing of specified tests (on the right).

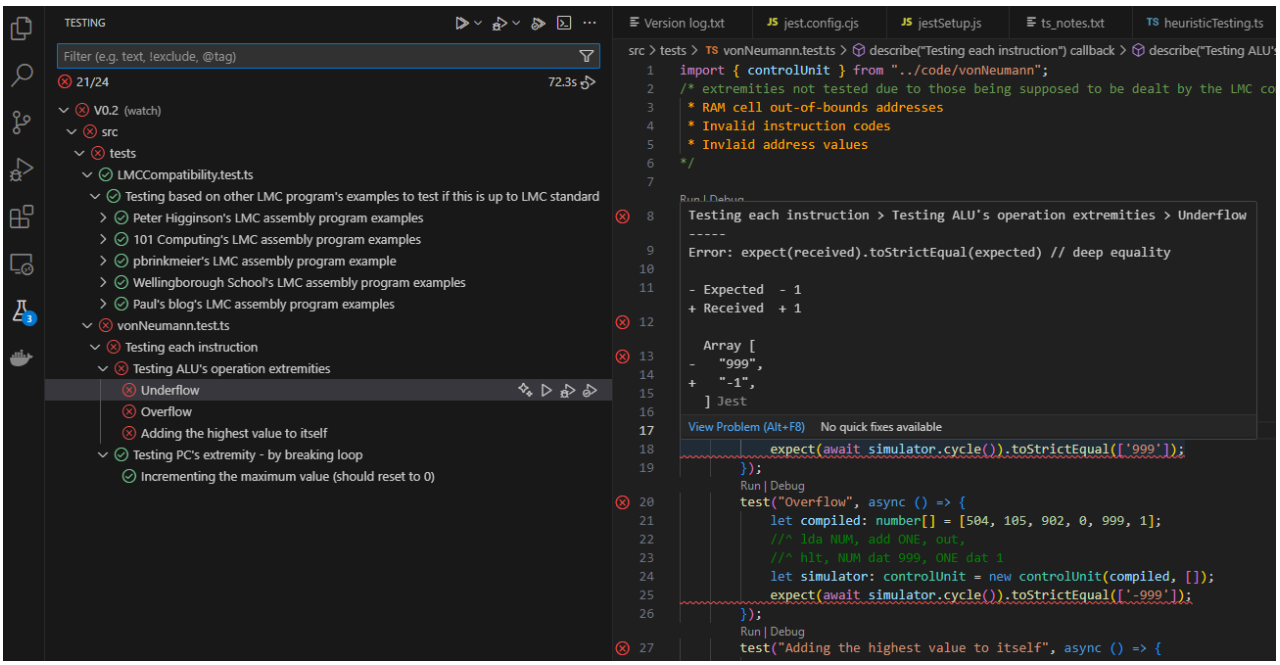


Figure 11.12—25 - shows the use of the Jest VS extension - the list of tests (on the left) and the viewing of specified tests (on the right).

- The second VS extension is also one for the Jest framework called Jest Runner by firsttris (unique ID - *firsttris.vscode-jest-runner*, version – 0.4.74).

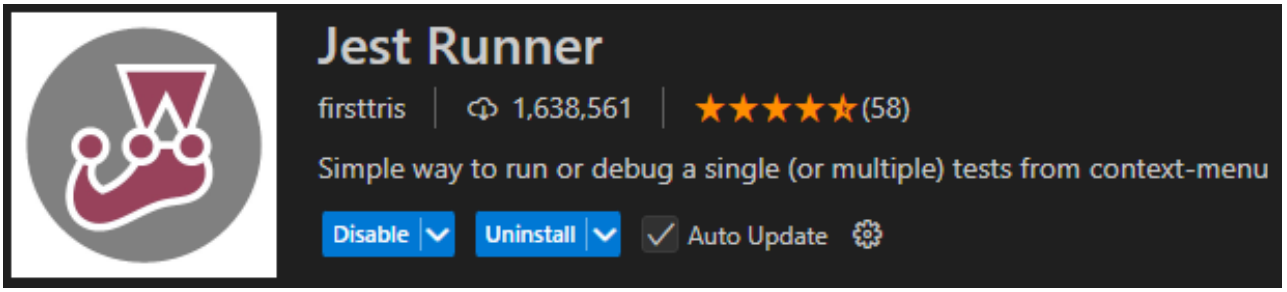


Figure 11.12—26 - Shows the overview of the Jest Runner VS extension by "firsttris".

This extension does seem like it has overlapping features with the other VS extension, such as testing individual tests, but is more compatible with the project. This is because it allows the debugging of tests and the code it tests, which are in TS, without error which the other VS extension struggles to do.

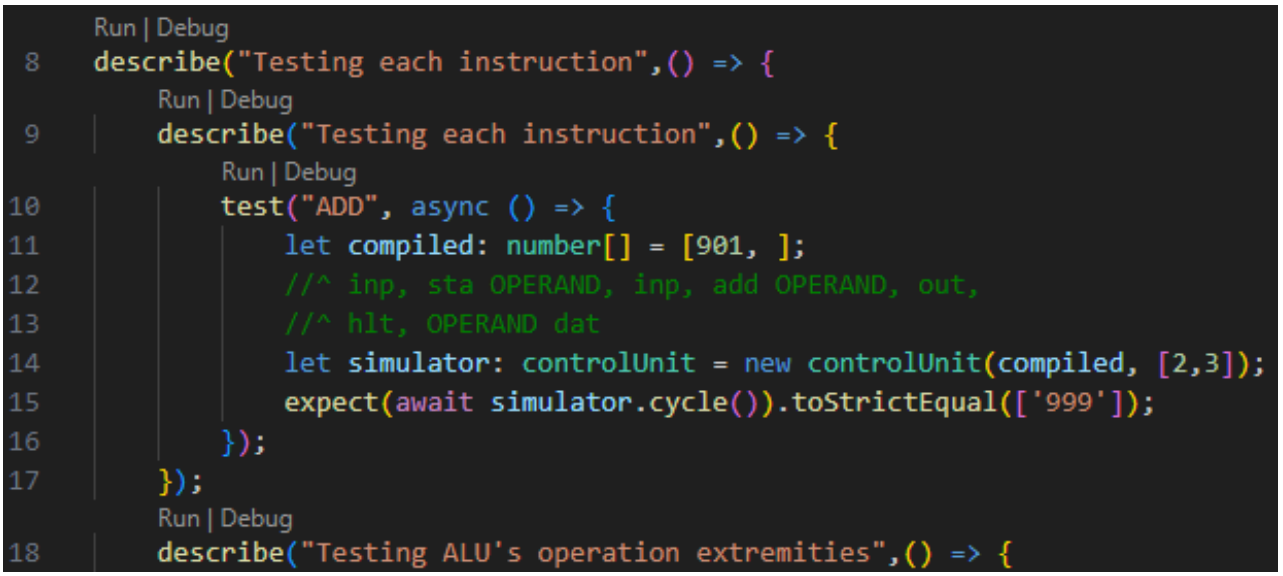


Figure 11.12—27 - shows the use of the Jest Runner VS extension by having the run and debug options "Run | Debug" options directly above each test and group of tests.

11.12.2.3 Deployment

Due to this being the first sprint, it can only be deployed using either heuristic testing or automatic testing. It is possible to allow user input, using dependencies, but the intent of the first sprint is to make the simulator engine itself working – not be usable by users. To make a CLI interface for the first sprint is to significantly deviate from the plan for future sprints, which should be avoided. Such heuristic use is `heuristicTesting.ts`.

11.12.2.3.1 Justifying deviations from plan and disruptions

There were quite a few expected and unexpected variations between the code and the plan.

11.12.2.3.1.1 Expected

Due to this only being the first sprint, there is not many expected deviations.

11.12.2.3.1.1.1 UML

The first expected variation was the UML diagram. When making the UML diagrams, the author has used pseudocode terms such as "integer" datatype and used the class name as the constructor method's name. Meanwhile, TS uses the "number" data type instead of integer and uses the name "constructor" as the constructor method's name.

This is purposely done because, due to the author's experience in TS, the author can easily migrate between TS terms pseudocode terms without effort, and this benefits the readers of the UML plan who are not familiar with TS syntax.

11.12.2.3.1.2 Unexpected

11.12.2.3.1.2.1 UML

Despite expected differences, there is also an unexpected difference in the UML plan. This regards the construction and lifespan nature of the class instances. The classes of the first sprint were originally made to be instantiated once, like static classes – hence the use of methods such as the memory (*`RAM`* class) *`controlUnit.compile`* method and the interface (*`IO`* class) *`controlUnit.changeInputs`* method as this is supposed to be the method to set up for each assembly program. However, as seen

in the UML diagrams, the classes were not stated to be static nor singleton classes. This is because the author has forgotten to put much deep thought into it until coding the classes themselves. When this realisation hit the author, the author analysed what has been done so far and the current direction of the product, concluding that it is best not to make the classes static-like but to make an instance each time when setting up to execute an assembly program. This time, considerable thought was put in.

The reason is the scale of complexity for the current and future state of the program. The code complexity and computational time needed by the class are predicted to be much less when classes are instantiated again instead of being reset or overwritten. Not to mention the time needed to consider and make sure everything is ready for the next assembly code, instead of just simply coding to delete and create another set of instances. The user will almost certainly create/edit an assembly program and run it more than once; influenced to many by the number of levels and the game's engagement, which scales and emphasises the point of computational time. So, for that reason, the set-up/resetting methods (`controlUnit.changeInputs` and `controlUnit.compile`) are removed and any code to setting up the classes themselves (like assigning arguments to fields) are instead handles by the classes' constructor – hence the use of constructors in some classes that were not shown in the UML.

11.12.2.3.1.2.2 Jest

This was already mentioned, explained and justified previously, but the author has put this note here to indicate that this was also considered an unexpected (but also good) variation to the plan.

11.12.2.3.1.2.3 Multi-lined comments

At first, the author has used multiline comments to describe the beginning line of a scope. An example has been presented below.

```
19      //: property methods
20      public read(address:number):number/* getter */{
```

Figure 11.12—28 - example of using a multi-line comment to label a code scope at its beginning.

This becomes a problem as, sometimes in debugging, entire scopes of code must be commented out due to how TS (with the strict config) mandates that anything declared must also be called. Two ways to comment out an entire scope of code it either to comment out line by line, which makes the code more disorganised and cluttered even with the VS shortcuts, or there is the use of multi-line notation of simply putting `/*` behind and `*/` in front. The multi-lined one is far more likely to have a problem of messing the multi-line comment's coverage when there is another multi-lined comment inside, as demonstrated below.

```
18  /*
19      //: property methods
20      public read(address:number):number/* getter */{
21      */
```

Figure 11.12—29 - the mess caused by having a multi-line comment in another.

This is easily solved by changing what comment type the author uses as seen below.

```
18      //: property methods
19      public read(address:number):number{ //< getter
20          //* still a setter
```

Figure 11.12—30 - example of using single-line comments to label a code scope at its beginning.

And as seen below, it fixes the problem.

```
18  /*
19      //: property methods
20      public read(address:number):number{ //< getter
21          //* still a setter
22          */
```

Figure 11.12—31 - the nested multi-lined problem was solved by using single-lined comments instead.

11.12.2.4 Review

11.12.2.4.1 Current state of code

src > tests > TS heuristicTesting.ts > ...

```ts
import { controlUnit } from "../code/vonNeumann";


async function test(){
    //* addition (before async was added)
    const compiled: number[] = [901,399,901,199,902];
    const predefinedInputs: number[] = [20,300];
    const simulator: controlUnit = new controlUnit(compiled,predefinedInputs);
    console.log(await simulator.cycle());
}
test();
async function testAsync(){
    //* bit more complex program that incorporates 'promise' return data and "async" and "await" functionality
    const compiled: number[] = [901, 309, 901, 109, 902, 901, 209, 902, 0];
    const predefinedInputs: number[] = [20,300,41];
    const simulator: controlUnit = new controlUnit(compiled,predefinedInputs);
    console.log(await simulator.cycle());
}
testAsync();
async function testLong(){
    //* time-consuming program
    //const compiled: number[] = [510, 313, 513, 902, 111, 313, 212, 709, 602, 0, 32, 1, 35];
    const compiled: number[] = [514, 317, 517, 902, 514, 922, 517, 922, 115, 317, 216, 713, 602, 0, 32, 1, 97];
    const predefinedInputs: number[] = [];
    const simulator: controlUnit = new controlUnit(compiled,predefinedInputs);
    console.log(await simulator.cycle());
}
testLong();
async function testDebugging(){
    //* solely for debugging failed jest tests (code changes depending on what jest test I am currently dubugging)
    const compiled: number[] = [512, 1, 111, 902, 312, 511, 113, 311, 514, 211, 80, 0, 1, 0, 1, 10];
    const predefinedInputs: number[] = [];
    const simulator: controlUnit = new controlUnit(compiled,predefinedInputs);
    console.log(await simulator.cycle());
}
testDebugging();
```

Figure 11.12—32 – using/instructing the LMC simulator, without GUI nor CLI, using heuristic testing.

```
[ '320' ]
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man connects to the outside world
Little man connects to the outside world

Little man loads mail address 11's value to the calculator
Little man checks the mail counter
Little man checks the mail counter
Little man checks the mail counter
PC - 6
Instruction - 99
Little man fetches next instruction
PC - 6
Instruction - 99
Little man fetches next instruction
PC - 6
Instruction - 55
Little man fetches next instruction
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man subtracts mail address 9's value from calculator.
Little man loads mail address 17's value to the calculator
Little man adds mail address 13's value to calculator.
Little man checks the mail counter
Little man checks the mail counter
Little man checks the mail counter
PC - 7
Instruction - 22
Little man fetches next instruction
PC - 7
Instruction - 55
Little man fetches next instruction
PC - 7
Instruction - 11
Little man fetches next instruction
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man decodes then do the instruction
Little man connects to the outside world
21
Little man connects to the outside world

Little man stores calculator's value to mail address 11
Little man checks the mail counter
Little man checks the mail counter
Little man checks the mail counter
PC - 8
Instruction - 99
Little man fetches next instruction
PC - 8
Instruction - 99
Little man fetches next instruction
PC - 8
Instruction - 33
Little man fetches next instruction
Little man takes a coffee break
Program halted
[ '320', '21' ]
```

Figure 11.12—33 – output of the end of one heuristic test and the whole of another (some lines are repeated to ensure all parts of the simulator runs correctly).

Reasoning for lack of a proper interface with the LMC program can be read in the **Purposeful lack of interface** section.

11.12.2.4.2 Purposeful lack of interface

The author has purposely used a lack of interface and used a heuristic code to interact with it rather than a user-friendly interface. This is obviously because the author's aim for sprint #1 was to simply implement the frontend of the LMC simulator, thus not having any GUI to interface with.

The author could get away with implementing UI, without GUI, by using CLI – interfacing by reading and typing in VS Code's integrated terminal using *Node.js*. However, the author was against this, mainly due to its lack of necessity as the author wants to add frontend in sprint #2, making the CLI code redundant. Developing CLI will be going on a tangent from the plan. Also,

implementing CLI will take user time to research how to implement it, due to it not being well-known which can be spent doing the lower priority points instead; which the author did.

11.12.2.4.3 What went well

The author has managed to create the LMC simulator quickly, completing all planned objectives, without the aid of the source code of the other LMC simulators. This is mostly important for the first sprint because this is the foundation of the LMC educational game, and the influence of other source code will influence the end-product and go on a tangent away from the plan.

11.12.2.4.4 What would be better and used for the next sprint's plan

The author has planned good about the classes' methods and fields as well as the relationship between the classes (like composition), but not so much about the nature of the classes themselves. For the plan of the next sprint (sprint two), the author must focus more on the classes' nature – how will they exist (static, singleton, abstract, interface, child, et cetera). Considering inheritance was not a concern for the first sprint because of the use of a few classes and their distinct purposes from one another. This will be slightly less be the case for the second sprints with the increased number of classed and will be of full concern for the third sprint for the sheer relative size and its objective of checking the assembly code in each level – one class of checking assembling code per campaign level of the LMC educational game.

11.12.2.4.5 Changes to previous sprint(s)

There were no changes to anything previous because this was the first sprint.

11.13 Appendix M: Agile Development: Timebox 2

Unlike sprint 1, there is not intermediate between sprint 2 and the other 2 main sprints.

11.13.1 Planning

11.13.1.1 Prior Research

Alongside using all sprint #2 related plans from the planning section, the author also did much research to be able and ready for sprint 2's development.

Most of the relevant research, for sprint 2's development, is still based on the **Prior Research** of **Sprint 1**.

11.13.1.1.1 Running Dependencies

Using running dependencies will be very difficult for a product that will have both an offline and online version. For that reason, it will best keep the running dependencies to a minimum. This means downloading packages as development dependencies (by using the `--save-dev` *npm* flag) instead of running dependencies.

11.13.1.1.1.1 CSS frameworks

The only running dependency that is viable is CSS libraries. They can be downloaded and accessed in the offline versions' product itself and accessed by a link reference (such as `<link rel="stylesheet" href="https://example.com/pathToCssFile.css">`) for the online versions. That way, both offline and online keep their advantages (not requiring internet and convenient accessibility, respectively) by not sharing their disadvantages. Using a CSS dependency will be a big time-saver as it will take time to create a style for a game, especially for multiple styles for account for disabilities and preferences.

For better reliability and community support, the author wanted to choose one of the more popular and widely used CSS frameworks. From reading upon multiple reviews and lists, "Bootstrap" and "Tailwind CSS" are always mentioned in them and in larger lists, they are always interchangeable in the top two. Both seem very good, but for the sake of consistency, only one can be used.

For choosing a suitable CSS framework, the following factors will be considered:

1. How expansive their style is (how many HTML tags/elements do they cover).
2. How many styles do they offer.
3. The nature/looks of the style itself.
4. How many of the styles cater to people with disabilities, most notably: colour vision deficiency, dyslexia, photophobia with one style for each. Other disabilities, such as macular degeneration, glaucoma, cataracts, diabetic retinopathy, amblyopia, retinitis pigmentosa, nystagmus, convergence insufficiency, et cetera, only require a style that has large fonts and high contrasts.
5. How is the delay for linking to the URL of the CSS framework (online version).
6. Ability to import only modules of the CSS framework instead of the whole thing (offline version).

Comparison of the two frameworks can be seen below.

List point #	Bootstrap	Tailwind CSS
1	Covers a lot (in the thousands).	Also covers a lot (in the hundreds), but not as much as Bootstrap
2	Focuses more on offering multiple predefined styles for each component, such as: such as buttons, forms, navigation bars, modals, tooltips, carousels, et cetera.	It is a "utility-first approach", meaning if focuses more on utility classes parts such as padding, margin, colour, font size, et cetera.
3	Styles are clean and modern-looking.	Focuses more on customising their own styles via its configuration file.
4	Does not have styles made directly for disabilities but it has features that can help, such as options for brightness, contrast, focus states, font sizes, et cetera.	Does not have styles made directly for disabilities but it has features that can help, such as options for brightness, contrast, focus states, font sizes, et cetera.
5	Minimal due to using CDN for fast delivery	Minimal due to using CDN for fast delivery
6	Have the ability to import only specific modules instead of the whole framework	Have the ability to import only specific modules instead of the whole framework

Table 11.13—1 - compares characteristics and traits of both CSS frameworks.

Note: CDN stand for "Content Delivery Network", which means that it connects to the closest geographic server for faster communication – making it load faster.

From this, the author believes that Bootstrap is a much better option. This is because not only does Bootstrap offer more of a variety, but it also focuses more on predefined styles than customisable styles, unlike Tailwind CSS, which saves a lot of time and

time-saving is one of the main advantages of using a framework. Frameworks also give the advantage of less complexity from less code, which is also satisfied more with Bootstrap because less customisation means less code for that customisation.

11.13.1.1.2 Learning

11.13.1.1.2.1 Learning the CSS framework

As mentioned, the author will be using the Bootstrap CSS framework but as the author have not used CSS frameworks before. Despite that the author still decided to use it because CSS is simplistic in nature and the only requirement to use CSS frameworks is to know what CSS relevant classes they offer and what they do.

The author still researched and learned some relevant Bootstrap CSS features and syntax to be prepared for the development of the second sprint.

Topic leant	Link of tutorial
split the page into 3 dynamically sized vertical sections	(W3Schools, 2021c)
style the tables (IDE/editor and RAM/memory)	(W3Schools, 2021e)
display images of Little Man without stretch	(W3Schools, 2021f)
sub-section boxes	(W3Schools, 2015b)
individual buttons	(W3Schools, 2021b)
grouping the buttons	(W3Schools, 2021a)
labelled textboxes and their sizes	(W3Schools, 2021d)
width of individual textboxes	(W3Schools, 2015a)
easy dark/light mode switching	(W3Schools, 2023)

Figure 11.13—1 - showing some of the Bootstrap (CSS Framework) concepts learned.

In the learning, the author initially researched Bootstrap V3, which is “end of life” (Bootstrap, 2018), instead of V5. To amend this, the author switched those (that had a tutorial in V5) to the V5 edition before implementation. Appears that lower version features are still fully supported in V5, and not all Bootstrap features were explained in V5, thus some tutorials are still from V3. The author preferred the better-explained and easier tutorials on W3Schools than the official documentation. The main reason for this is that the official documentation seems more like a revision; meanwhile for the W3School tutorials, the author feels that he is actually learning it in the first place, and thus knows how to implement those Bootstrap features.

11.13.1.2 Design

11.13.1.2.1 IPOD Tables

Sprint two will have multiple IPOD tables. The first one is for the controls for navigating through the compiler input text boxes, while the second one is for the rest of the simulator which are mostly buttons.

11.13.1.2.1.1 Keyboard keys in script editing

Inputs	Process	Outputs	Decisions
Tab key			Go down a line if in an operand box.
Space key			Go down a line if in an operand box.
Shift key + right arrow key			Go down a line if in an operand box.
Enter key			Go back to the top line if on the bottom line.
Downwards key			Go back to the top line if on the bottom line.
Upwards key			Go to the bottom line if on the top line.

Shift key + enter key			Go to the bottom line if on the top line.
Shift key + left key			Go to the line like above if in a label box.
Shift key + space key			Go to the line like above if in a label box.

Table 11.13—2 - IPOD, in sprint 2, for script editing using keyboard keys.

11.13.1.2.1.2 Other simulator controls

Inputs	Process	Outputs	Decisions
Control (Ctrl) Key + enter key	Attempt to compile the current assembly code.		
"Assemble" button	Attempt to compile the current assembly code.		
"Start or Stop" button	Either starts, stop/pause, or none of those.	"Program started". "Program stopped" (stop can also mean pause). "Assembly code must be compiled first".	Has the assembly program already started running? Has the assembly code compiled successfully?
"Time or Step" button	Toggle program executing mode between timed and steps	"Execution mode set to timed" "Execution mode set to steps"	Is program executing mode timed or steps?
"Back to menu" button		Load the main menu (Sprint 2 does not cover this so it will just be a blank but different page)	
"Reset" buttons	Clear memory, registers, and stop current assembly code execution if running.	"Program reset"	Is the program running?
"Run Settings" – step execution mode	Cycle (the simulator) once.	"Step cycle"	
"Run Settings" – timed execution mode	Change the speed of the simulator's execution.	"Execution speed set to [set speed]ms per cycle"	Can input not be parsed into an integer? Is the specified speed too high, too low, or acceptable?
"Switch to Little Man or Level objective" button	Toggle between displaying Little Man and the level objective.		Is it currently displaying Little Man or the level objective?

Table 11.13—3 - IPOD, in sprint 2, for the rest of the simulator besides keyboard keys.

11.13.1.2.2 UML

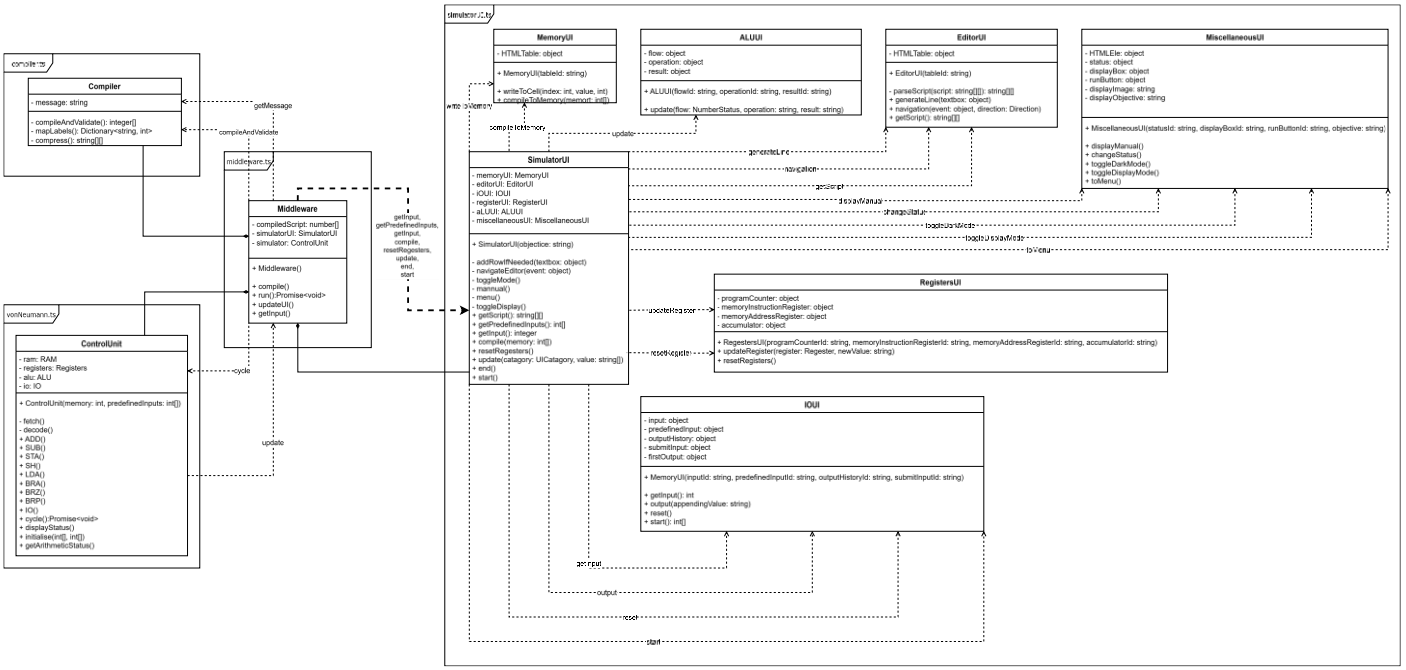


Figure 11.13—2 - UML diagram of all relevant TS files for sprint 2.

11.13.1.2.3 Flowchart

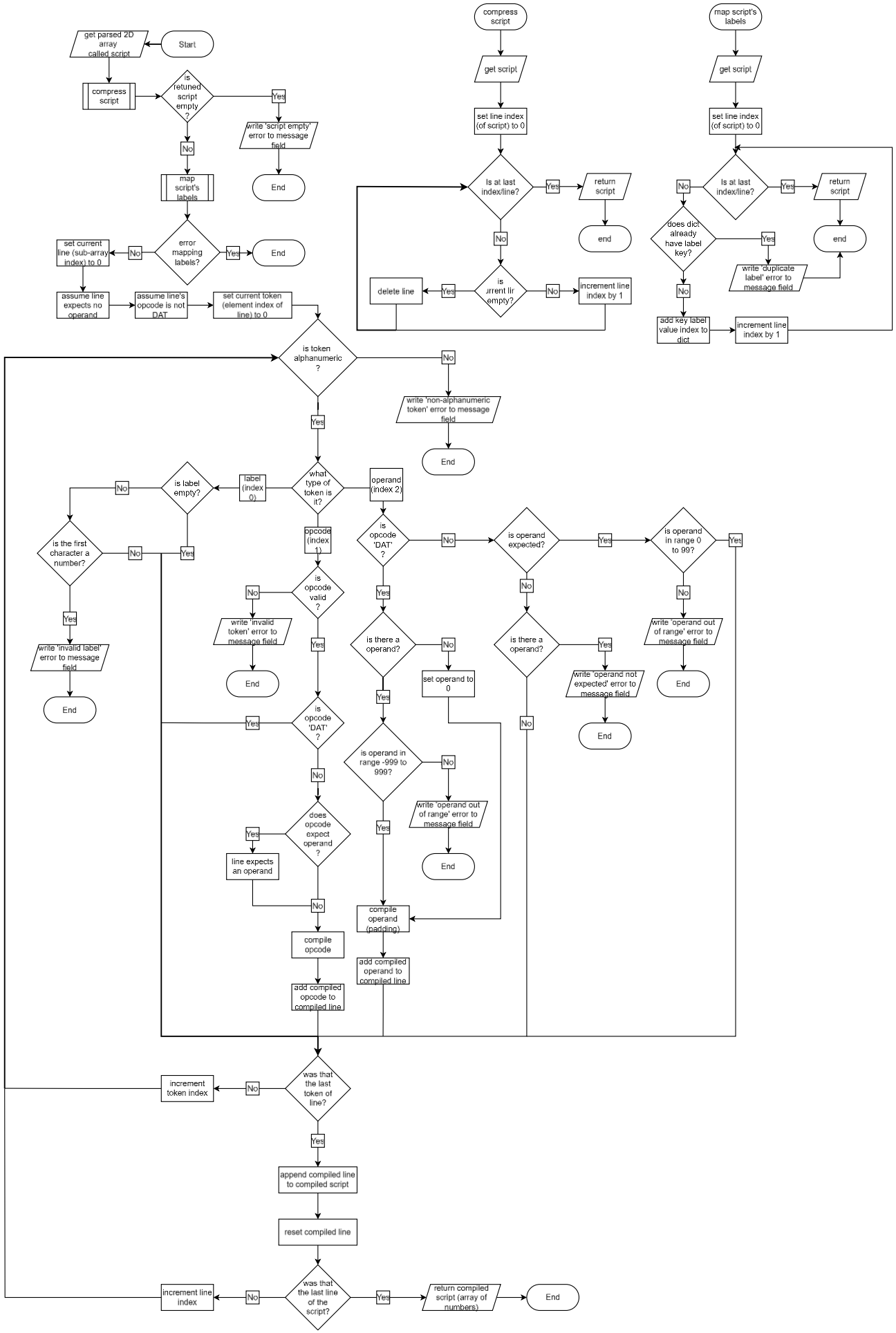


Figure 11.13—3 - flowchart, of the validation and compilation of the assembly script, for sprint 2.

11.13.1.2.3.1 Justification for flowchart

Unlike sprint 1, the main method `validateAndCompile` method's switch-statement cases will not be separated into their own methods. Despite the immense size, the author has good reasoning for splitting the `validateAndCompile` it.

11.13.1.2.3.2 Not splitting switch-statement cases into separate methods

The reason for not splitting the method's switch-statement cases is due to too much cross communication. Unlike the instruction methods in `controlUnit`, switch-statement cases share the same data which is needed. It is needed because of the shared data. Some of these shared variables range from simple Booleans, such as one for indicating if the opcode expects an operand and if the opcode is the 'DAT' instruction. Those are assigned in the opcode case and read in the operand case, as the validation criteria of operands depend on the corresponding opcodes. For examples: if the opcode is 'OUT' then do not expect an operand from the user, if it a 'ADD' then expect an operand between 0 and 99, and if 'DAT' then expect the operand to be between -999 and 999. Other variable types also needed to be shared, such as: 2D lists script and compiled script, string compiled instruction (contains opcode and sometimes operand compiled and padded), index of for-loop containing the switch-statement, et cetera.
With all these commonly written-to and read-from variables, it makes the use of separate methods very inconvenient. Inconvenient because each method will use too many parameters and/or class fields to the point that most of the planning and debugging will be spent dealing with those.

11.13.1.2.3.3 Not splitting the validation and compilation

However, that is not the only reason for its size. Another factor for it is the fact that it does both validation and computation. Those two parts are purposely put together for the sake of computational efficiency. Using big O notation, the method takes *O(5n)* time, where n is the number of lines in the user's inputted script. Breaking this down: there is *O(2n)* from calling methods (*O(n)* for compressing script and *O(n)* for mapping and validating labels), and *O(3n)* from a nested for-loop that iterates over each token (label, opcode and operand) of each line.

If validation were to be separate a method from compilation, it would take far longer to compute. This is because the token-iterating nested for-loop will become two, one for validating the tokens and one for compiling them – extending the time by another *O(3n)*. Meanwhile, the called method for mapping (where mapping is the compilation process and contains validation because it needs to check for undeclared label references) the labels will also be split into 2 methods in accordance with the separation logic – extending by another *O(n)*. This totals the longer time to *O(9n),* which is nearly double the time and since TS is not a fast language (due to JS being an interpretated language), this makes it far more significant and noticeable. For that reason, they are done together.

11.13.1.2.4 Styling

Sprint 2 only focuses on making the LMC simulator usable by a user, hence there being only one style. As sprint 2 style priorities functionality over being stylish, style traits will be basic just for usability and being a basic template for making more styles in the third sprint. To make sure each class/component can be seen, they are all made in different colours.

11.13.1.2.4.1 Default (basic) style

Class/component	Colour	Hex code
Page background	Creamy	#FFB (#FFFFBB)
Text colour	Black	#000 (#000000)
Textbox/label background colour	White	#FFF (#FFFFFF)
Primary colour	Green	#0F0 (#00FF00)
Secondary colour	Blue	#00F (#0000FF)

Table 11.13—4 - describing the colours, and corresponding hex codes, of the UI style for sprint 2.

Please note that hex codes will be used in 3-digit form (#RGB) instead of 6-digit (#RRGGBB) due to its better simplicity with negligible visual differences. For example: 3-digit colour #F2A equates to 6-digit #FF22AA.

11.13.1.2.4.2 Palette of the basic template style

*Table 11.13—5 - colour palette corresponding to **Table 11.13—4** and is exported from Coolors (Bianchi and Fabrizio, 2015).*

11.13.1.2.5 Wireframe Mock-ups

Sprint 2 objectives dictate that only the simulator page is needed for its entire UI. An outline wireframe has already been made in Figure 11.12—5 but it was made before starting the sprints. Therefore, a more detailed (and colour-coded) one is made as the plan for the sprint 2 UI.

11.13.1.2.5.1 Low-fi

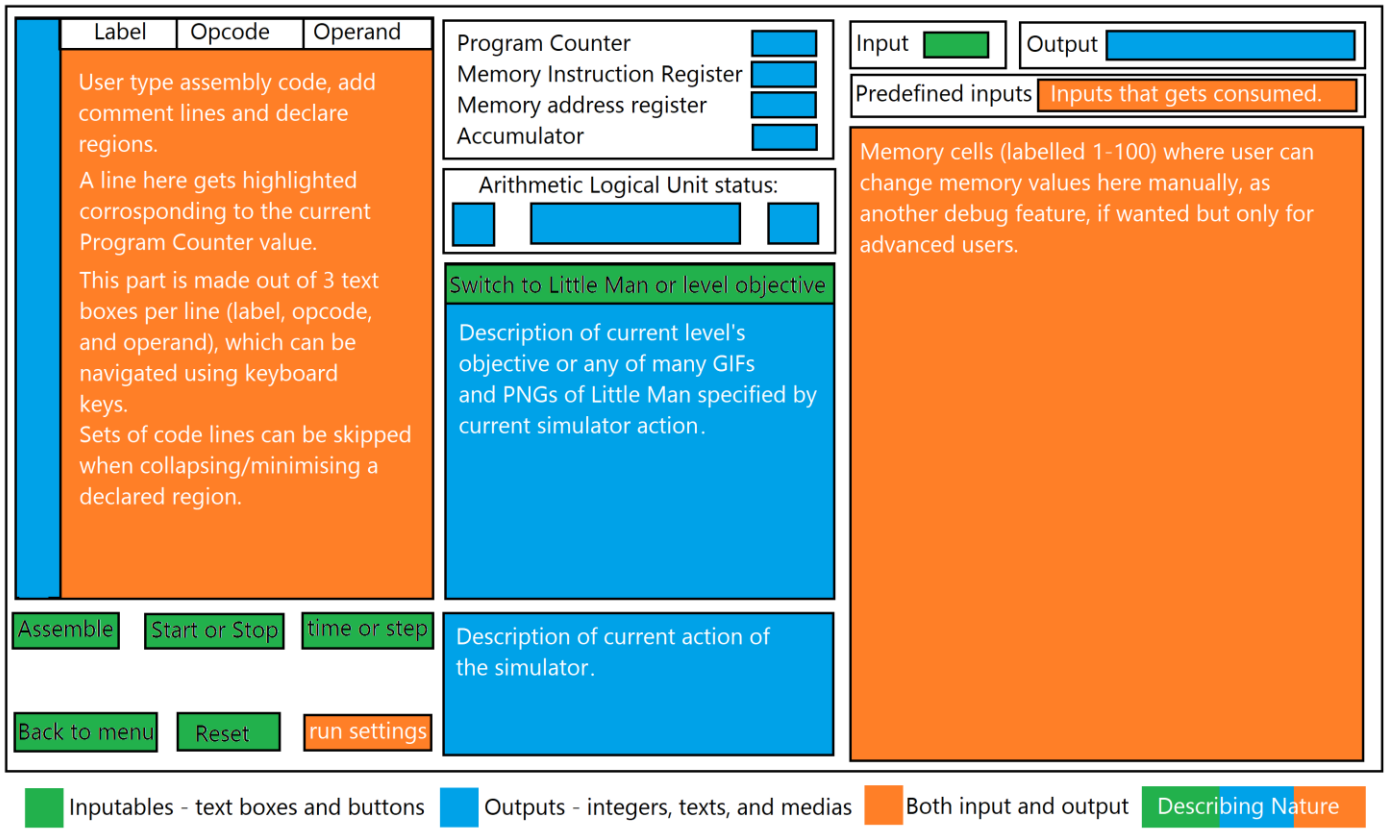


Figure 11.13—4 - a colour-coded low-fi wireframe for sprint 2 (of the simulator page), made in Microsoft Paint.

11.13.1.2.5.1.1 Why use Microsoft Paint?

The author has decided to use Microsoft Paint. This is because unlike other related software, Paint has a negligible learning curve, making it the perfect tool for rapid development of low-fi diagrams. This is unlike other more specialised software, with either a much higher learning curve or is only partially free-to-use, just to have very similar results to Paint.

11.13.1.2.5.2 High-fi

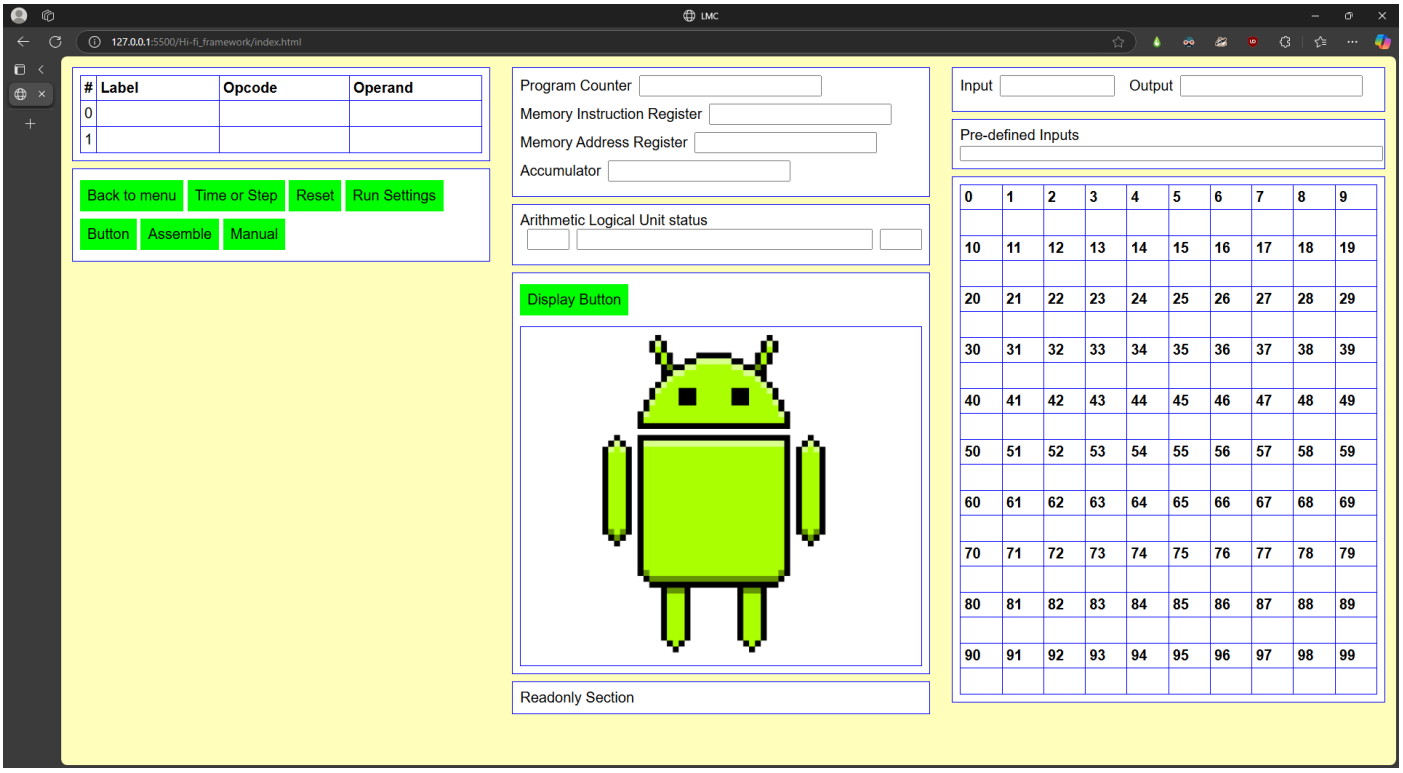


Figure 11.13—5 - a detailed high-fi wireframe made using AI-generated HTML code.

11.13.1.2.5.2.1 Why is high-fi needed with low-fi?

The low-fi wireframe will help make the frame/outline of the UI, such as: elements, descriptions of elements, label texts, and categorising input, output, both or none of them (denoted by colour-coding). Meanwhile high-fi wireframe is made to cover more of the details such as images, how it looks like in a browser, more precise positioning and ratios/measurements, use of a rough draft or example style (colours, shapes, et cetera), and is based on the low-fi's string points and having solutions to the low-fi's problems/weaknesses.

Despite high-fi's more detail, its high-level of concept makes it critical for it to be based on a lower-level concept to prevent the creation of the high-fi wireframe from going on tangents. High-fi's increased level of detail makes it difficult for the author to not overlook the core concepts of the wireframe, which is unavoidable with the low-fi wireframe. For these reasons, the author deemed to mandatory to base the development on both the low-fi and high-fi wireframes.

11.13.1.2.5.2.2 Use of AI to generate high-fi

Just like with the low-fi wireframe, the author has also looked at a variety of tools that fit the job but are still either paid or only partially free and have an even steeper learning curve (due to the elevated level of detail for high-fi). High-fi also have the added requirement of requiring at least a somewhat similar style of the wireframe's elements, which is an extreme scarcity as most wireframe builders only have one style.

Because of this, the author decided to use raw HTML, as it precisely reflects the required level of detail. However, this leads to another problem as doing the wireframe in HTML is nothing less than doing the sprint development itself and thus neglecting the point of a wireframe. To get around this, the author has decided to make use of the AI GPT model using Co-Pilot.

Please note that hi-fi wireframes are the only thing assisted by AI generation in this project!

11.13.1.2.6 Modification Mapping Table

In the second sprint, the modification mapping table gives the developer the full idea towards specifically what UI elements (text values, specific images, et cetera) is affected by each instruction to make the simulator completely visual to the users so they know how LMC simulator is working – based on the modification mapping table of the first sprint (in Appendix L).

	Memory cells	Program Counter	Memory Instruction Register	Memory Address Register	Accumulator	Status	Display image	ALU displays	Outputs	Pre-defined Inputs
ADD					X	X	X	X		
SUB					X	X	X	X		
STA	X					X	X			
LSH					X	X	X	X		
RSH					X	X	X	X		
LDA					X	X	X			
BRA		X				X	X			
BRZ		? #1				X	X			
BRP		? #2				X	X			

INP					X	X	X			? #3
OUT						X	X		X	
OCT						X	X		X	
HLT						X	X			
DAT										
a F.D.E. cycle		X	X	X		X	X			

Table 11.13—6 - modification mapping table for sprint 1.

Note: “ALU displays” are called so instead of separately because all the ALU UI elements are either all updated together or not at all.

Elaboration on conditional modifications is the same as that in sprint #1.

11.13.1.3 Whitebox test case planning

Just like of sprint #1, below is a set tables listing every extremity and aspect/feature of the first sprint that are planned to be tested in sprint #2. These are where the actual test cases will be based on. Justification for each set of tests are also explained in this section. Their implementation is done in the sprint #2’s **Test cases** section.

There are 3 new jest test files, those being:

- `compiler.test.ts` – Tests the compiler by testing each label, opcode, and operand.
- `compilerCompatibility.test.ts` – Test if the compiler is up to LMC standard by testing its compatibility with the other LMC’s, from the software research, example programs.
- `orginalDOMElements.test.ts` – Testing all static HTML elements that are loaded alongside the original DOM.

11.13.1.3.1 Backend tests

Note that the “actual result” columns in the tables of this section refer to the compiler’s output, not the simulator’s output.

11.13.1.3.1.1 Compiler’s functionality

What is tested?	How is it tested?	Expected result

Storing and adding using numeric addresses as operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "99"],`
	`["", "INP", ""],`
	`["", "ADD", "99"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 399, 901, 199, 902, 0

Branching (`BRZ`, `BRP`, `BRA`) using numeric addresses as operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "BRZ", "5"],`
	`["", "BRP", "6"],`
	`["", "BRA", "7"],`
	`["", "OUT", ""],`
	`["", "OUT", ""],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 902, 705, 806, 607, 902, 902, 902, 0

Loading (`LDA`) stored variable (`DAT`) using numeric addresses as operands.	Compiling mnemonic script:
	`["", "LDA", "3"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["", "DAT", "10"]`
		503, 902, 0, 10

Loading (`LDA`) stored variable (`DAT`) using numeric addresses as operands, but stored value is `-999` instead of `10`.	Compiling mnemonic script:
	`["", "LDA", "3"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["", "DAT", "-999"]`
		503, 902, 0, -999

Attempting to use an address too high to use (`1000`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "100"],`
	`["", "HLT", ""]`
		Error message regarding the `100` token.

Attempting to use an address too low to use (`-1`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "-1"],`
	`["", "HLT", ""]`
		Error message regarding the `-1` token.

Attempting to use an address that is a decimal (`5.5`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "5.5"],`
	`["", "HLT", ""]`
		Error message regarding the `5.5` token.

*Table 11.13—7 – plan for test cases, of sprint #2 from **`compiler.test.ts`**, testing use of addresses (instead of labels) in example scripts.*

What is tested?	How is it tested?	Expected result

Storing and adding using labels as operands but `DAT` is at the first instruction line instead of the last.	Compiling mnemonic script:
	`["BUFFER", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		0, 901, 300, 901, 100, 902, 0

Storing and adding using labels as operands but `DAT` is at the lase instruction line.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["BUFFER", "DAT", ""]`
		901, 306, 901, 106, 902, 0, 0

Using 3 different labels.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "BRZ", "ONE"],`
	`["", "BRP", "TWO"],`
	`["", "BRA", "THREE"],`
		901, 902, 706, 807, 608, 902, 902, 902, 0

	`["", "OUT", ""],`
	`["ONE", "OUT", ""],`
	`["TWO", "OUT", ""],`
	`["THREE", "HLT", ""]`	

Program without any operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "SHL", ""],`
	`["", "SHR", ""],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 401, 402, 902, 0

Declared labels that never got referenced in the script.	Compiling mnemonic script:
	`["ONE", "INP", ""],`
	`["", "STA", "99"],`
	`["", "OUT", ""],`
	`["THREE", "OUT", ""],`
	`["", "HLT", ""]`
		901, 399, 902, 902, 0

labels have number digits inside them but not as first character.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "BRA", "E1ND2"],`
	`["", "OUT", ""],`
	`["E1ND2", "HLT", ""]`
		901, 603, 902, 0

Attempting to reference label that was not declared.	Compiling mnemonic script:
	`["", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		Error message regarding the `BUFFER` token.

Attempting to use label that begins with number digit than letters.	Compiling mnemonic script:
	`["1BUFFER", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "1BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "1BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		Error message regarding the `1BUFFER` token.

*Table 11.13—8 - plan for test cases, of sprint #2 from **`compiler.test.ts`**, testing use of labels (instead of addresses) in example scripts.*

What is tested?	How is it tested?	Expected result

Attempting to compile a non-existent/invalid opcode.	Compiling mnemonic script: `["", "STO", ""]`	Error regarding the `STO` not being a valid opcode.

All opcodes that require a corresponding operand, with an operand, in a valid manner.	Compiling mnemonic script:
	`["", "ADD", "99"],`
	`["", "SUB", "99"],`
	`["", "STA", "99"],`
	`["", "LDA", "99"],`
	`["", "BRA", "99"],`
	`["", "BRZ", "99"],`
	`["", "BRP", "99"],`
	`["", "DAT", "99"]`
		199, 299, 399, 599, 699, 799, 899, 99

Attempting to compile an `ADD` instruction without an operand.	Compiling mnemonic script: `["", "ADD", ""]`	Error regarding the `ADD` not having a corresponding operand.
Attempting to compile a `SUB` instruction without an operand.	Compiling mnemonic script: `["", "SUB", ""]`	Error regarding the `SUB` not having a corresponding operand.
Attempting to compile a `STA` instruction without an operand.	Compiling mnemonic script: `["", "STA", ""]`	Error regarding the `STA` not having a corresponding operand.
Attempting to compile a `LDA` instruction without an operand.	Compiling mnemonic script: `["", "LDA", ""]`	Error regarding the `LDA` not having a corresponding operand.
Attempting to compile a `BRA` instruction without an operand.	Compiling mnemonic script: `["", "BRA", ""]`	Error regarding the `BRA` not having a corresponding operand.
Attempting to compile a `BRZ` instruction without an operand.	Compiling mnemonic script: `["", "BRZ", ""]`	Error regarding the `BRZ` not having a corresponding operand.
Attempting to compile a `BRP` instruction without an operand.	Compiling mnemonic script: `["", "BRP", ""]`	Error regarding the `BRP` not having a corresponding operand.

Table 11.13—9 - plan for test cases, of sprint #2 from **`compiler.test.ts`**, *testing use opcodes that require corresponding operands in example scripts.*

What is tested?	How is it tested?	Expected result

All opcodes that require a corresponding operand, with an operand, in a valid manner.	Compiling mnemonic script:
	`["", "SHL", ""],`
	`["", "SHR", ""],`
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "OTC", ""],`
	`["", "HLT", ""]`
		401, 402, 901, 902, 903, 0

Attempting to compile a `SHL` instruction without an operand.	Compiling mnemonic script: `["", "SHL", "invalidOperand"]`	Error regarding the `SHL` not having a corresponding operand.

Attempting to compile a `SHR` instruction without an operand.	Compiling mnemonic script: `["", "SHR", "invalidOperand"]`	Error regarding the `SHR` not having a corresponding operand.
Attempting to compile an `INP` instruction without an operand.	Compiling mnemonic script: `["", "INP", "invalidOperand"]`	Error regarding the `INP` not having a corresponding operand.
Attempting to compile an `OUT` instruction without an operand.	Compiling mnemonic script: `["", "OUT", "invalidOperand"]`	Error regarding the `OUT` not having a corresponding operand.
Attempting to compile an `OTC` instruction without an operand.	Compiling mnemonic script: `["", "OTC", "invalidOperand"]`	Error regarding the `OTC` not having a corresponding operand.
Attempting to compile a `HLT` instruction without an operand.	Compiling mnemonic script: `["", "HLT", "invalidOperand"]`	Error regarding the `HLT` not having a corresponding operand.

*Table 11.13—10 - plan for test cases, of sprint #2 from **`compiler.test.ts`**, testing use opcodes that does not require corresponding operands in example scripts.*

What is tested?	How is it tested?	Expected result
just `DAT` by itself	Compiling mnemonic script: `["", "DAT", ""]`	0
just `DAT` and a label	Compiling mnemonic script: `["LABEL", "DAT", ""]`	0
just `DAT` and a valid numerical operand	Compiling mnemonic script: `["", "DAT", "68"]`	68
`DAT` with a label and valid numerical operand.	Compiling mnemonic script: `["LABEL", "DAT", "68"]`	68
using `DAT` to store the lowest possible integer	Compiling mnemonic script: `["", "DAT", "-999"]`	-999
using `DAT` to store the highest possible integer	Compiling mnemonic script: `["", "DAT", "999"]`	999
using `DAT` to store zero	Compiling mnemonic script: `["", "DAT", "0"]`	0
Attempt: `DAT` referring a non-existent label	Compiling mnemonic script: `["", "DAT", "LABEL"]`	Error message regarding the `LABEL` token.
Attempt: `DAT` with the highest under-range number	Compiling mnemonic script: `["", "DAT", "-1000"]`	Error message regarding the `-1000` token.
Attempt: `DAT` with the lowest over-range number	Compiling mnemonic script: `["", "DAT", "1000"]`	Error message regarding the `1000` token.
Attempt: `DAT` with a decimal instead of an integer	Compiling mnemonic script: `["", "DAT", "5.5"]`	Error message regarding the `5.5` token.

*Table 11.13—11 - plan for test cases, of sprint #2 from **`compiler.test.ts`**, testing Data Location (DAT) operand in example scripts.*

11.13.1.3.1.2 Compiler's compatibility

The author must stress that the mnemonic script used in the tests, for the compiler's compatibility, tends to be too big to fit in these test tables, so please refer to **`compilerCompatability.test.ts`** for the corresponding scripts.

What is tested?	How is it tested?	Expected result
LMC #1's `"add" program`.	Attempts to compile the mnemonic script of the example program.	901, 399, 901, 199, 902, 0
LMC #1's `"add/subtr" program`.	Attempts to compile the mnemonic script of the example program.	901, 309, 901, 109, 902, 901, 209, 902, 0, 0
LMC #1's `"ascii" program`.	Attempts to compile the mnemonic script of the example program.	510, 313, 513, 903, 111, 313, 212, 709, 602, 0, 32, 1, 127, 0

LMC #1's *"ascii table" program*.	Attempts to compile the mnemonic script of the example program.	514, 317, 517, 902, 514, 903, 517, 903, 115, 317, 216, 713, 602, 0, 32, 1, 97, 0

Table 11.13—12 - plan for test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, *testing mnemonic (initially non-compiled) scripts of LMC #1 (Peter Higginson's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result
LMC #2's *"add" program*.	Attempts to compile the mnemonic script of the example program.	901, 306, 901, 106, 902, 0, 0
LMC #2's *"add/subtr" program*.	Attempts to compile the mnemonic script of the example program.	901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0
LMC #2's *"ascii" program*.	Attempts to compile the mnemonic script of the example program.	901, 902, 308, 207, 308, 801, 0, 1, 0
LMC #2's *"ascii table" program*.	Attempts to compile the mnemonic script of the example program.	901, 316, 901, 317, 519, 116, 319, 517, 218, 317, 804, 519, 216, 319, 902, 0, 0, 0, 1, 0
LMC #2's *"add" program*.	Attempts to compile the mnemonic script of the example program.	512, 111, 902, 312, 511, 113, 311, 514, 211, 800, 0, 1, 0, 1, 10
LMC #2's *"add/subtr" program*.	Attempts to compile the mnemonic script of the example program.	901, 336, 733, 238, 339, 337, 536, 340, 539, 730, 238, 730, 536, 140, 336, 537, 238, 337, 238, 721, 608, 536, 340, 539, 238, 339, 337, 238, 730, 608, 536, 902, 0, 538, 902, 0, 0, 0, 1, 0, 0

Table 11.13—13 - plan for test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, *testing mnemonic (initially non-compiled) scripts of LMC #2 (101 Computing's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result
LMC #3's *"takes two inputs and puts their difference in the outbox" program*.	Attempts to compile the mnemonic script of the example program.	901, 308, 901, 309, 508, 209, 902, 0, 0, 0

Table 11.13—14 - plan for test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, *testing mnemonic (initially non-compiled) scripts of LMC #3 (pbrinkmeier's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result
LMC #4's *"Example 1 - Add two numbers together" program*.	Attempts to compile the mnemonic script of the example program.	901, 320, 901, 120, 902, 0, 0
LMC #4's *"Example 2 - Output a pattern of 1s and 0s" program*.	Attempts to compile the mnemonic script of the example program.	520, 902, 521, 902, 522, 220, 322, 800, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3
LMC #4's *"Example 3 - Calculate the square of a number program*.	Attempts to compile the mnemonic script of the example program.	901, 330, 533, 331, 332, 531, 130, 331, 532, 134, 332, 230, 814, 605, 531, 902, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
LMC #4's *"Example 4 - Integer division*.	Attempts to compile the mnemonic script of the example program.	901, 323, 901, 324, 520, 322, 523, 224, 323, 813, 522, 902, 0, 522, 121, 322, 523, 607, 0, 0, 0, 1, 0, 0, 0

Table 11.13—15 - plan for test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, *testing mnemonic (initially non-compiled) scripts of LMC #4 (Wellingborough School's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result
LMC #6's *"Max of 3 numbers" program*.	Attempts to compile the mnemonic script of the example program.	901, 318, 901, 319, 901, 320, 219, 810, 519, 320, 520, 218, 815, 518, 320, 520, 902, 0, 0, 0, 0
LMC #6's *"Multiply 2 numbers" program*.	Attempts to compile the mnemonic script of the example program.	901, 315, 901, 314, 514, 712, 217, 314, 516, 115, 316, 604, 516, 902, 0, 0, 0, 1
LMC #6's *""Copy N inputs to an array" (self-modifying)" program*.	Attempts to compile the mnemonic script of the example program.	901, 314, 514, 714, 215, 314, 517, 115, 317, 116, 312, 901, 0, 602, 0, 1, 300, 49

LMC #6's `""Sieve of Erastothenes" (self-modifying)" program`.	Attempts to compile the mnemonic script of the example program.	530, 131, 330, 227, 826, 530, 128, 308, 0, 711, 600, 530, 902, 332, 532, 227, 800, 532, 129, 321, 530, 0, 532, 130, 332, 614, 0, 69, 531, 331, 1, 1, 0

*Table 11.13—16 - plan for test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, testing mnemonic (initially non-compiled) scripts of LMC #6 (Paul's blog's) to ensure LMC compatibility.*

`compiler.test.ts` is, obviously from the name, the testing file for the compiler. This is a vital test file because if the compiler struggles to do its job properly, then the program cannot translate the user's mnemonic script into the number-based compiled script for the simulator to understand the user's intentions.

11.13.1.3.2 Frontend testing

Frontend testing was done in **`orginalDOMElements.test.ts`** which tested every non-dynamic HTML element of the simulator test.

What is tested?	Tested element ID	Expected result
Fetching an element by a non-existent ID.	`doesNotExist`	Fetched element is null. Fetched element is not instance of `HTMLElement`.
Fetching an element but treating it as a different HTML element type.	`menu`	Fetched element is not null. Fetched element is not instance of `HTMLSpanElement`.
Fetching an element but accessing a property that does not exist in that element type.	`menu`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`. Fetched element is not instance of `HTMLInputElement`. Property placeholder is undefined.

*Table 11.13—17 - plan for purposely invalid tests to make sure that the tests actually work from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result
Program counter register	`registerProgramCounter`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Program counter register	`registerInstruction`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Address/operand register	`registerAddress`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Accumulator register	`registerAccumulator`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.

*Table 11.13—18 - plan for testing register textbox elements, of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result
Flow	`flow`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.

		Property `readOnly` is true.
Operation	`operation`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Result	`result`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.

*Table 11.13—19 - plan for testing ALU elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result
Input	`input`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Outputs	`output`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.
Pre-defined inputs	`predefinedInputs`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is false.

*Table 11.13—20 - plan for testing ALU elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result
To menu	`menu`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
execution method	`timeOrStep`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
reset	`reset`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
compile	`compile`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
run	`run`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
manual	`manual`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
submit input	`submitInput`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
toggle mode (light/dark)	`toggleMode`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.
toggle display	`toggleDisplay`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.

*Table 11.13—21 - plan for testing input/output (IO) elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result

testing the table containing the token textboxes (label, opcode, and operand)	`editorTable`	Fetched element is not null. Fetched element is instance of `HTMLTableElement`.
label (row 0)	`input-0-0`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.
opcode (row 0)	`input-0-1`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.
operand (row 0)	`input-0-2`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.
label (row 1)	`input-1-0`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.
opcode (row 1)	`input-1-1`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.
operand (row 1)	`input-1-2`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.

Table 11.13—22 - plan for the first two lines of textboxes in the script editor, of the simulator page, from **`orginalDOMElements.test.ts`** *of sprint #2.*

What is tested?	Tested element ID	Expected result
current status	`status`	Fetched element is not null. Fetched element is instance of `HTMLSpanElement`.
display box	`displayBox`	Fetched element is not null. Fetched element is instance of `HTMLDivElement`.

Table 11.13—23 - plan for simulator's status testing, of the simulator page, from **`orginalDOMElements.test.ts`** *of sprint #2.*

11.13.1.4 Critical Path Analysis

Little Man Computer educational game

Project by - James Haddad

Supervisor - Tina Eager

Project Start:	Tue, 10/1/2024
Project End:	Mon, 5/12/2025
Display Week:	1

TASK	STATUS	START	END
Literature review	Fully done		
Academic research	Fully done	10/11/24	11/24/24
Software research	Fully done	10/21/24	12/8/24
Create poster for submission	Fully done	12/9/24	1/17/25
Requirement analysis	Fully done	11/25/24	11/26/24
Technologies used	Fully done	11/27/24	12/1/24
Priorities	Fully done	12/2/24	12/8/24
Prepare and perform poster presentation	Fully done	1/18/25	1/21/25
Sprint 1	Fully done		
Pre-sprint preparations	Fully done	1/20/25	1/26/25
Prior research	Fully done	1/27/25	2/2/25
Design	Fully done	2/3/25	2/9/25
Critical path analysis	Fully done	2/10/25	2/16/25
Developmment - implementation	Fully done	2/17/25	2/23/25
Developmment - testing	Fully done	2/24/25	2/26/25
Review and self-reflection	Fully done	2/27/25	3/2/25
Sprint 2	Currently doing		
Prior research	Fully done	2/24/25	3/2/25
Design	Fully done	3/3/25	3/9/25
Critical path analysis	Currently doing	3/10/25	3/16/25
Developmment - implementation	haven't started	3/17/25	3/23/25
Developmment - testing	haven't started	3/24/25	3/26/25
Review and self-reflection	haven't started	3/27/25	3/30/25
Sprint 3			
Prior research	haven't started	3/24/25	3/30/25
Design	haven't started	3/31/25	4/6/25
Critical path analysis	haven't started	4/7/25	4/13/25
Developmment - implementation	haven't started	4/14/25	4/20/25
Developmment - testing	haven't started	4/21/25	4/23/25
Review and self-reflection	haven't started	4/24/25	4/27/25
Post sprint 3			
Project review and reflection	haven't started	4/21/25	4/29/25
Client testing	haven't started	4/27/25	5/5/25
Polishing	haven't started	5/6/25	5/12/25

Figure 11.13—6 - current state of the GANTT chart tasks at this moment in sprint #2.

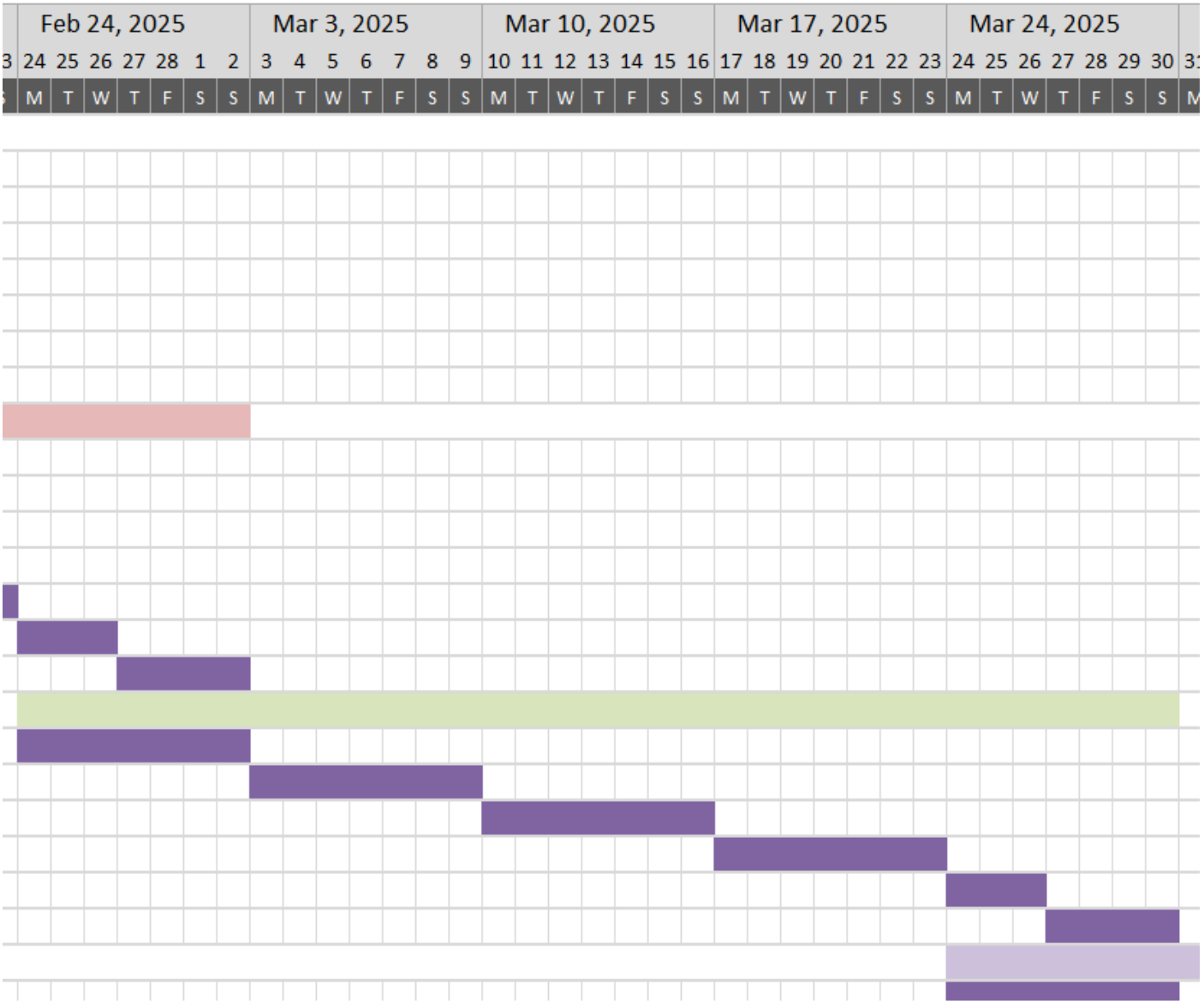


Figure 11.13—7 - view of GANTT tasks duration and deadlines corresponding to sprint #2 GANTT tasks in Figure 11.13—6.

Due to the sprint #1 being the start of creating a program, rather than improving and expanding over existing code, the author was able to easily finish the sprint in time (even with the extra "Pre-sprint preparations" task). This leads to starting from the sprint #2's "Development - implementation" task on time (17th March).

Sprint #2 will be the first sprint of improving and expanding an existing codebase, the author believes that there will likely be delays and unexpected obstacles along the way of sprint #3 development. Delays may be magnified depending on how the user goes on with the author's other university module assignments.

11.13.2 Development

11.13.2.1 Main features

- Now has UI for use to use the simulator (in sandbox mode) using `simulator.html` for display and `simulatorUI.ts` for frontend functionality.
- Use of a compiler, in `compiler.ts`, to convert the user's text script into a list of compiled simulator-compliant instructions.
- Use of middleware (`middleware.ts`) serves as an API between the backend classes, which are `compiler.ts` and `vonNeumann.ts`, and the frontend (`simulatorUI.ts`)
- Placeholder menu page using `menu.html`.
- Start on the manual page (`manual.html`), teaching the user basic concepts of the LMC simulator.
- Addition of light mode and dark mode of the default theme defined in `default.css`.
- Slight customisation of the Bootstrap CSS, such as `font-size`, in `commons.css`.

11.13.2.2 Current project directory

Currently project directory is visually shown below:

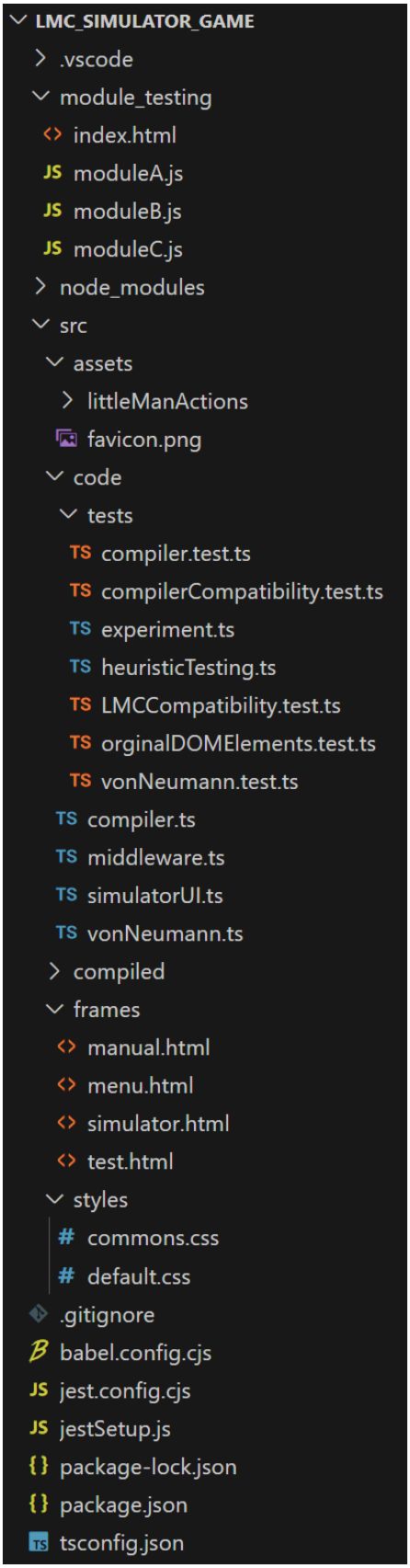


Figure 11.13—8 - a screenshot snippet of the project's directory of sprint #2 displaying the primary files.

Due to the increased number of files, compared to sprint #1, not all can be displayed in **Figure 11.13—8**. Those being the JS files compiled from the TS files and the image assets of the Little Man's actions, as both are seen below.

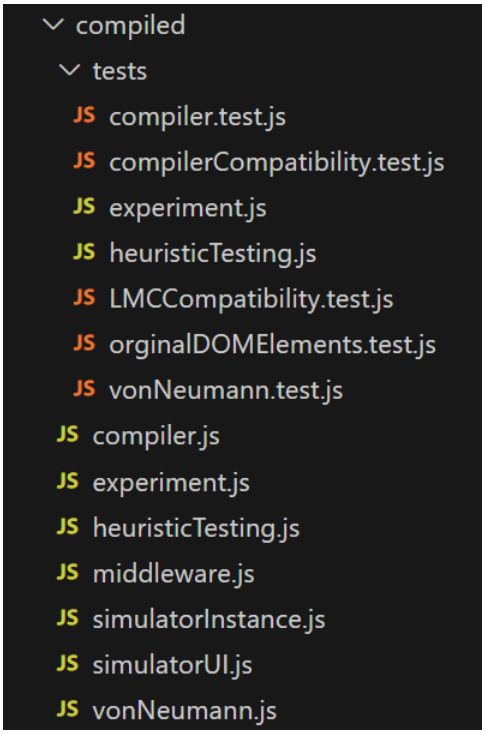


Figure 11.13—9 - folder of JS files compiled from the TS files of sprint #2.

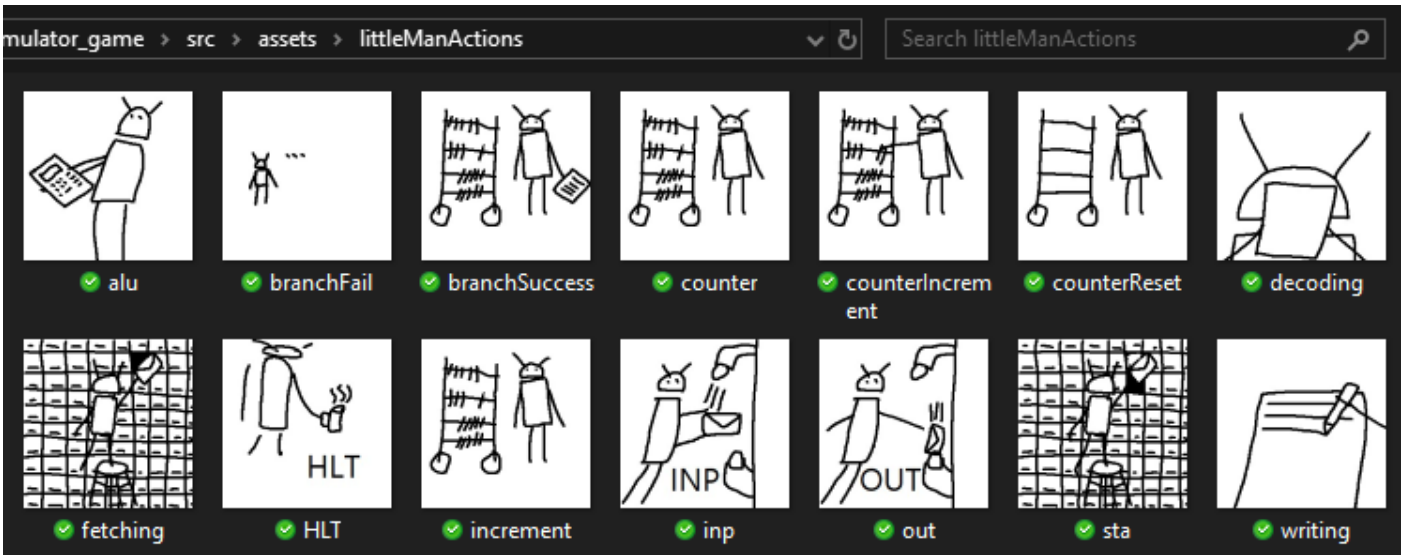


Figure 11.13—10 - displaying the placeholder images of the Little Man's actions.

Compared to sprint #1, there are far more source and compiled files and assets. Another difference is that the full set of web technologies is used for the simulator instead of a practical proof of concept. A keen eye, upon the readers, may have caught the attention of the `module_testing` folder. That is not directly related to the simulator product but was part of a major debugging session, which involves attempting to get modular-type JS files to work in the web browser and will be discussed in detail later in this review.

11.13.2.3 Test cases

Test cases done based upon the test plan from sprint #2's **Whitebox test case planning** section, justification and explanation are also done there.

11.13.2.3.1 Backend tests

Just as mentioned, in the **Whitebox test case planning**, the "actual result" columns in the tables of this section refer to the compiler's output, not the simulator's output.

11.13.2.3.1.1 Compiler's functionality

What is tested?	How is it tested?	Expected result	Actual result	Changes made

Storing and adding using numeric addresses as operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "99"],`
	`["", "INP", ""],`
	`["", "ADD", "99"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 399, 901, 199, 902, 0	901, 399, 901, 199, 902, 0	None. (actual result was expected)

Branching (`BRZ`, `BRP`, `BRA`) using numeric addresses as operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "BRZ", "5"],`
	`["", "BRP", "6"],`
	`["", "BRA", "7"],`
	`["", "OUT", ""],`
	`["", "OUT", ""],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 902, 705, 806, 607, 902, 902, 902, 0	901, 902, 705, 806, 607, 902, 902, 902, 0	None. (actual result was expected)

Loading (`LDA`) stored variable (`DAT`) using numeric addresses as operands.	Compiling mnemonic script:
	`["", "LDA", "3"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["", "DAT", "10"]`
		503, 902, 0, 10	503, 902, 0, 10	None. (actual result was expected)

Loading (`LDA`) stored variable (`DAT`) using numeric addresses as operands, but stored value is `-999` instead of `10`.	Compiling mnemonic script:
	`["", "LDA", "3"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["", "DAT", "-999"]`
		503, 902, 0, -999	503, 902, 0, -999	None. (actual result was expected)

Attempting to use an address too high to use (`1000`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "100"],`
	`["", "HLT", ""]`
		Error message regarding the `100` token.	Same as expected result	None. (actual result was expected)

Attempting to use an address too low to use (`-1`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "-1"],`
	`["", "HLT", ""]`
		Error message regarding the `-1` token.	Same as expected result	None. (actual result was expected)

Attempting to use an address that is a decimal (`5.5`).	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "5.5"],`
	`["", "HLT", ""]`
		Error message regarding the `5.5` token.	Same as expected result	None. (actual result was expected)

*Table 11.13—24 - test cases, of sprint #2 from **`compiler.test.ts`**, testing use of addresses (instead of labels) in example scripts.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made

Storing and adding using labels as operands but `DAT` is at the first instruction line instead of the last.	Compiling mnemonic script:
	`["BUFFER", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		0, 901, 300, 901, 100, 902, 0	0, 901, 300, 901, 100, 902, 0	None. (actual result was expected)

Storing and adding using labels as operands but `DAT` is at the lase instruction line.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""],`
	`["BUFFER", "DAT", ""]`
		901, 306, 901, 106, 902, 0, 0	901, 306, 901, 106, 902, 0, 0	None. (actual result was expected)

Using 3 different labels.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "BRZ", "ONE"],`
	`["", "BRP", "TWO"],`
	`["", "BRA", "THREE"],`
	`["", "OUT", ""],`
	`["ONE", "OUT", ""],`
	`["TWO", "OUT", ""],`
	`["THREE", "HLT", ""]`
		901, 902, 706, 807, 608, 902, 902, 902, 0	901, 902, 706, 807, 608, 902, 902, 902, 0	None. (actual result was expected)

Program without any operands.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "SHL", ""],`
	`["", "SHR", ""],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		901, 401, 402, 902, 0	901, 401, 402, 902, 0	None. (actual result was expected)

Declared labels that never got referenced in the script.	Compiling mnemonic script:
	`["ONE", "INP", ""],`
	`["", "STA", "99"],`
	`["", "OUT", ""],`
	`["THREE", "OUT", ""],`
	`["", "HLT", ""]`
		901, 399, 902, 902, 0	901, 399, 902, 902, 0	None. (actual result was expected)

labels have number digits inside them but not as first character.	Compiling mnemonic script:
	`["", "INP", ""],`
	`["", "BRA", "E1ND2"],`
	`["", "OUT", ""],`
	`["E1ND2", "HLT", ""]`
		901, 603, 902, 0	901, 603, 902, 0	None. (actual result was expected)

Attempting to reference label that was not declared.	Compiling mnemonic script:
	`["", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		Error message regarding the `BUFFER` token.	Same as expected result.	None. (actual result was expected)

Attempting to use label that begins with number digit than letters.	Compiling mnemonic script:
	`["1BUFFER", "DAT", "0"],`
	`["", "INP", ""],`
	`["", "STA", "1BUFFER"],`
	`["", "INP", ""],`
	`["", "ADD", "1BUFFER"],`
	`["", "OUT", ""],`
	`["", "HLT", ""]`
		Error message regarding the `1BUFFER` token.	Same as expected result.	None. (actual result was expected)

*Table 11.13—25 - test cases, of sprint #2 from **`compiler.test.ts`**, testing use of labels (instead of addresses) in example scripts.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
Attempting to compile a non-existent/invalid opcode.	Compiling mnemonic script: `["", "STO", ""]`	Error regarding the `STO` not being a valid opcode.	Same as expected result.	None. (actual result was expected)

All opcodes that require a corresponding operand, with an operand, in a valid manner.	Compiling mnemonic script:
	`["", "ADD", "99"],`
	`["", "SUB", "99"],`
	`["", "STA", "99"],`
	`["", "LDA", "99"],`
	`["", "BRA", "99"],`
	`["", "BRZ", "99"],`
	`["", "BRP", "99"],`
	`["", "DAT", "99"]`
		199, 299, 399, 599, 699, 799, 899, 99	199, 299, 399, 599, 699, 799, 899, 99	None. (actual result was expected)

Attempting to compile an `ADD` instruction without an operand.	Compiling mnemonic script: `["", "ADD", ""]`	Error regarding the `ADD` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)

Attempting to compile a `SUB` instruction without an operand.	Compiling mnemonic script: `["", "SUB", ""]`	Error regarding the `SUB` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `STA` instruction without an operand.	Compiling mnemonic script: `["", "STA", ""]`	Error regarding the `STA` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `LDA` instruction without an operand.	Compiling mnemonic script: `["", "LDA", ""]`	Error regarding the `LDA` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `BRA` instruction without an operand.	Compiling mnemonic script: `["", "BRA", ""]`	Error regarding the `BRA` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `BRZ` instruction without an operand.	Compiling mnemonic script: `["", "BRZ", ""]`	Error regarding the `BRZ` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `BRP` instruction without an operand.	Compiling mnemonic script: `["", "BRP", ""]`	Error regarding the `BRP` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)

*Table 11.13—26 - test cases, of sprint #2 from **`compiler.test.ts`**, testing use opcodes that require corresponding operands in example scripts.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made

All opcodes that require a corresponding operand, with an operand, in a valid manner.	Compiling mnemonic script:
	`["", "SHL", ""],`
	`["", "SHR", ""],`
	`["", "INP", ""],`
	`["", "OUT", ""],`
	`["", "OTC", ""],`
	`["", "HLT", ""]`
		401, 402, 901, 902, 903, 0	401, 402, 901, 902, 903, 0	None. (actual result was expected)

Attempting to compile a `SHL` instruction without an operand.	Compiling mnemonic script: `["", "SHL", "invalidOperand"]`	Error regarding the `SHL` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `SHR` instruction without an operand.	Compiling mnemonic script: `["", "SHR", "invalidOperand"]`	Error regarding the `SHR` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)

Attempting to compile an `INP` instruction without an operand.	Compiling mnemonic script: `["", "INP", "invalidOperand"]`	Error regarding the `INP` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile an `OUT` instruction without an operand.	Compiling mnemonic script: `["", "OUT", "invalidOperand"]`	Error regarding the `OUT` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile an `OTC` instruction without an operand.	Compiling mnemonic script: `["", "OTC", "invalidOperand"]`	Error regarding the `OTC` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)
Attempting to compile a `HLT` instruction without an operand.	Compiling mnemonic script: `["", "HLT", "invalidOperand"]`	Error regarding the `HLT` not having a corresponding operand.	Same as expected result.	None. (actual result was expected)

Table 11.13—27 - test cases, of sprint #2 from **`compiler.test.ts`**, *testing use opcodes that does not require corresponding operands in example scripts.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
just `DAT` by itself	Compiling mnemonic script: `["", "DAT", ""]`	0	0	None. (actual result was expected)
just `DAT` and a label	Compiling mnemonic script: `["LABEL", "DAT", ""]`	0	0	None. (actual result was expected)
just `DAT` and a valid numerical operand	Compiling mnemonic script: `["", "DAT", "68"]`	68	68	None. (actual result was expected)
`DAT` with a label and valid numerical operand.	Compiling mnemonic script: `["LABEL", "DAT", "68"]`	68	68	None. (actual result was expected)
using `DAT` to store the lowest possible integer	Compiling mnemonic script: `["", "DAT", "-999"]`	-999	-999	None. (actual result was expected)
using `DAT` to store the highest possible integer	Compiling mnemonic script: `["", "DAT", "999"]`	999	999	None. (actual result was expected)
using `DAT` to store zero	Compiling mnemonic script: `["", "DAT", "0"]`	0	0	None. (actual result was expected)
Attempt: `DAT` referring a non-existent label	Compiling mnemonic script: `["", "DAT", "LABEL"]`	Error message regarding the `LABEL` token.	Same as expected result - `Missing label error at the operand of line 0 - label "LABEL" is called but cannot find code line with that label.`	None. (actual result was expected)
Attempt: `DAT` with the highest under-range number	Compiling mnemonic script: `["", "DAT", "-1000"]`	Error message regarding the `-1000` token.	Same as expected result - `operand not expected error at the operand of line 0 - label "-1000" is integer but out of bounds (-999 - 999)`	None. (actual result was expected)

Attempt: `DAT` with the lowest over-range number	Compiling mnemonic script: `["", "DAT", "1000"]`	Error message regarding the `1000` token.	Same as expected result - `operand not expected error at the operand of line 0 - label "1000" is integer but out of bounds (-999 - 999)`	None. (actual result was expected)
Attempt: `DAT` with a decimal instead of an integer	Compiling mnemonic script: `["", "DAT", "5.5"]`	Error message regarding the `5.5` token.	Same as expected result - `non-alphanumeric error at the operand of line 0 - token "5.5" is not alphanumeric. Alphanumeric must only comprise of numbers digits (0-9) and letters (A-Z)`	None. (actual result was expected)

*Table 11.13—28 - test cases, of sprint #2 from **`compiler.test.ts`**, testing Data Location (DAT) operand in example scripts.*

11.13.2.3.1.2 Compiler's compatibility

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #1's `"add" program`.	Attempts to compile the mnemonic script of the example program.	901, 399, 901, 199, 902, 0	Same as expected	None. (actual result was expected)
LMC #1's `"add/subtr" program`.	Attempts to compile the mnemonic script of the example program.	901, 309, 901, 109, 902, 901, 209, 902, 0, 0	Same as expected	None. (actual result was expected)
LMC #1's `"ascii" program`.	Attempts to compile the mnemonic script of the example program.	510, 313, 513, 903, 111, 313, 212, 709, 602, 0, 32, 1, 127, 0	Same as expected	None. (actual result was expected)
LMC #1's `"ascii table" program`.	Attempts to compile the mnemonic script of the example program.	514, 317, 517, 902, 514, 903, 517, 903, 115, 317, 216, 713, 602, 0, 32, 1, 97, 0	Same as expected	None. (actual result was expected)

*Table 11.13—29 - test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, testing mnemonic (initially non-compiled) scripts of LMC #1 (Peter Higginson's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #2's `"add" program`.	Attempts to compile the mnemonic script of the example program.	901, 306, 901, 106, 902, 0, 0	Same as expected	None. (actual result was expected)
LMC #2's `"add/subtr" program`.	Attempts to compile the mnemonic script of the example program.	901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0	Same as expected	None. (actual result was expected)
LMC #2's `"ascii" program`.	Attempts to compile the mnemonic script of the example program.	901, 902, 308, 207, 308, 801, 0, 1, 0	Same as expected	None. (actual result was expected)
LMC #2's `"ascii table" program`.	Attempts to compile the mnemonic script of the example program.	901, 316, 901, 317, 519, 116, 319, 517, 218, 317, 804, 519, 216, 319, 902, 0, 0, 0, 1, 0	Same as expected	None. (actual result was expected)
LMC #2's `"add" program`.	Attempts to compile the mnemonic script of the example program.	512, 111, 902, 312, 511, 113, 311, 514, 211, 800, 0, 1, 0, 1, 10	Same as expected	None. (actual result was expected)
LMC #2's `"add/subtr" program`.	Attempts to compile the mnemonic script of the example program.	901, 336, 733, 238, 339, 337, 536, 340, 539, 730, 238, 730, 536, 140, 336, 537, 238, 337, 238, 721, 608, 536, 340, 539, 238, 339, 337, 238, 730, 608, 536, 902, 0, 538, 902, 0, 0, 0, 1, 0, 0	Same as expected	None. (actual result was expected)

*Table 11.13—30 - test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, testing mnemonic (initially non-compiled) scripts of LMC #2 (101 Computing's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made

LMC #3's *"takes two inputs and puts their difference in the outbox" program*.	Attempts to compile the mnemonic script of the example program.	901, 308, 901, 309, 508, 209, 902, 0, 0, 0	Same as expected	None. (actual result was expected)

Table 11.13—31 - test cases, of sprint #2 from ***compilerCompatibility.test.ts****, testing mnemonic (initially non-compiled) scripts of LMC #3 (pbrinkmeier's) to ensure LMC compatibility.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #4's *"Example 1 - Add two numbers together" program*.	Attempts to compile the mnemonic script of the example program.	901, 320, 901, 120, 902, 0, 0	901, 306, 901, 106, 902, 0, 0	Remove the empty lines from the expected compiled script which leads to changing addresses as well.
LMC #4's *"Example 2 - Output a pattern of 1s and 0s" program*.	Attempts to compile the mnemonic script of the example program.	520, 902, 521, 902, 522, 220, 322, 800, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3	509, 902, 510, 902, 511, 209, 311, 800, 0, 1, 0, 3	Remove the empty lines from the expected compiled script which leads to changing addresses as well.
LMC #4's *"Example 3 - Calculate the square of a number program*.	Attempts to compile the mnemonic script of the example program.	901, 330, 533, 331, 332, 531, 130, 331, 532, 134, 332, 230, 814, 605, 531, 902, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1	901, 317, 520, 318, 319, 518, 117, 318, 519, 121, 319, 217, 814, 605, 518, 902, 0, 0, 0, 0, 0, 1	Remove the empty lines from the expected compiled script which leads to changing addresses as well.
LMC #4's *"Example 4 - Integer division*.	Attempts to compile the mnemonic script of the example program.	901, 323, 901, 324, 520, 322, 523, 224, 323, 813, 522, 902, 0, 522, 121, 322, 523, 607, 0, 0, 0, 1, 0, 0, 0	901, 323, 901, 324, 520, 322, 523, 224, 323, 813, 522, 902, 0, 522, 121, 322, 523, 607, 0, 0, 0, 1, 0, 0, 0	Remove the empty lines from the expected compiled script which leads to changing addresses as well.

Table 11.13—32 - test cases, of sprint #2 from ***compilerCompatibility.test.ts****, testing mnemonic (initially non-compiled) scripts of LMC #4 (Wellingborough School's) to ensure LMC compatibility.*

In testing the example programs for LMC #4, all tests initially failed. This is because, unlike the author's LMC and all other LMCs (from the **Software research**), LMC #4 has an outlier feature that accounts for empty lines in the mnemonic script. This is demonstrated in the screenshot below.

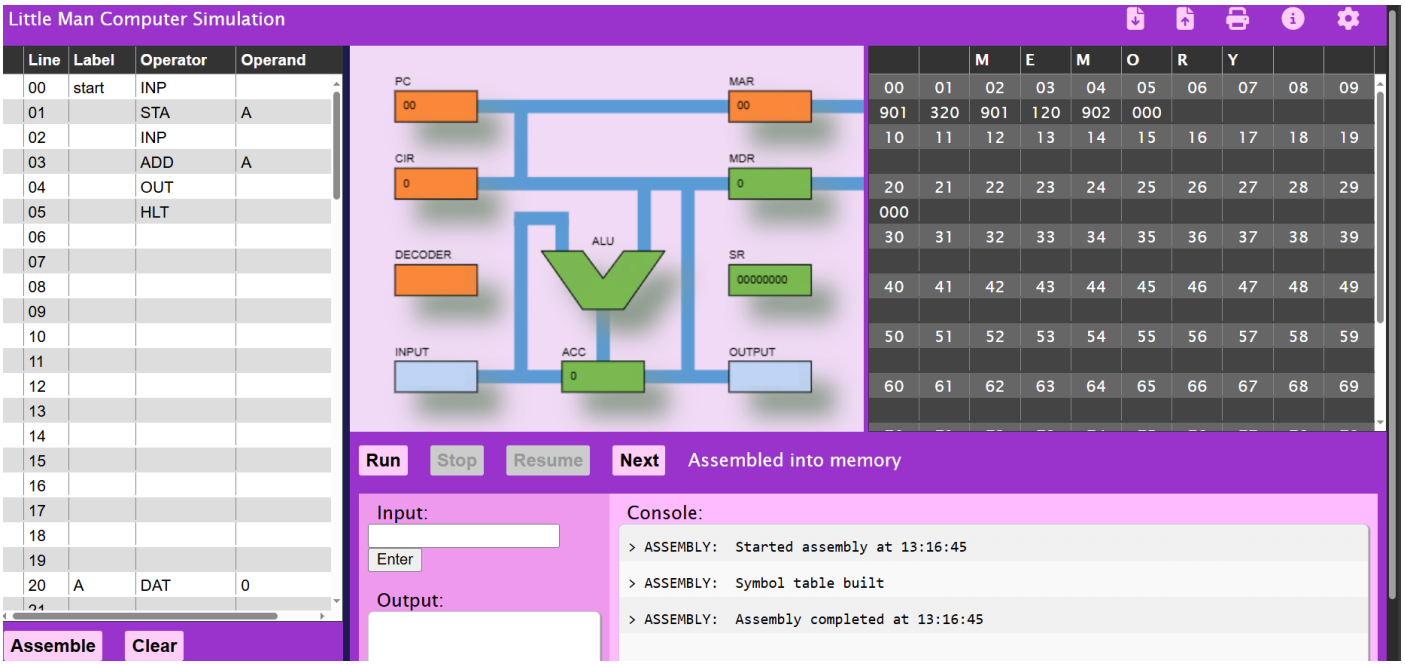


Figure 11.13—11 - screenshot snippet, of LMC #4's first example program, showing that empty lines in the mnemonic script editor also applies to the compiled script in the memory table.

Because of the uncountability of empty lines being a purposeful feature of the author's LMC program, the author had to delete the empty lines of the expected compiled and change affected addresses due to referenced memory cells being at different locations/addresses after removing the lines.

What is tested?	How is it tested?	Expected result	Actual result	Changes made
LMC #6's *"Max of 3 numbers" program*.	Attempts to compile the mnemonic script of the example program.	901, 318, 901, 319, 901, 320, 219, 810, 519, 320, 520, 218, 815, 518, 320, 520, 902, 0, 0, 0, 0	Same as expected result.	None. (actual result was expected)
LMC #6's *"Multiply 2 numbers" program*.	Attempts to compile the mnemonic script of the example program.	901, 315, 901, 314, 514, 712, 217, 314, 516, 115, 316, 604, 516, 902, 0, 0, 0, 1	Same as expected result.	None. (actual result was expected)
LMC #6's *""Copy N inputs to an array" (self-modifying)" program*.	Attempts to compile the mnemonic script of the example program.	901, 314, 514, 714, 215, 314, 517, 115, 317, 116, 312, 901, 0, 602, 0, 1, 300, 49	Same as expected result.	None. (actual result was expected)
LMC #6's *""Sieve of Erastothenes" (self-modifying)" program*.	Attempts to compile the mnemonic script of the example program.	530, 131, 330, 227, 826, 530, 128, 308, 0, 711, 600, 530, 902, 332, 532, 227, 800, 532, 129, 321, 530, 0, 532, 130, 332, 614, 0, 69, 531, 331, 1, 1, 0	Same as expected result.	None. (actual result was expected)

Table 11.13—33 - test cases, of sprint #2 from **`compilerCompatibility.test.ts`**, *testing mnemonic (initially non-compiled) scripts of LMC #6 (Paul's blog's) to ensure LMC compatibility.*

11.13.2.3.2 Frontend testing

Frontend testing was done in **`orginalDOMElements.test.ts`** which tested every non-dynamic HTML element of the simulator test.

What is tested?	Tested element ID	Expected result	Actual result	Changes made
Fetching an element by a non-existent ID.	`doesNotExist`	Fetched element is null. Fetched element is not instance of `HTMLElement`.	Same as expected result.	None. (actual result was expected)
Fetching an element but treating it as a different HTML element type.	`menu`	Fetched element is not null. Fetched element is not instance of `HTMLSpanElement`.	Same as expected result.	None. (actual result was expected)
Fetching an element but accessing a property that does not exist in that element type.	`menu`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`. Fetched element is not instance of `HTMLInputElement`. Property placeholder is undefined.	Same as expected result.	None. (actual result was expected)

Table 11.13—34 - purposely invalid tests to make sure that the tests actually work from **`orginalDOMElements.test.ts`** *of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
Program counter register	`registerProgramCounter`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	`readOnly` was false because HTML uses `readonly` (without capitalisation) while TS/JS uses `readOnly`.	Changed `readOnly` to `readonly` in the HTML.
Program counter register	`registerInstruction`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)

		Property `readOnly` is true.		
Address/operand register	`registerAddress`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)
Accumulator register	`registerAccumulator`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)

Table 11.13—35 - Testing register textbox elements, of the simulator page, from **`orginalDOMElements.test.ts`** *of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
Flow	`flow`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)
Operation	`operation`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)
Result	`result`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)

Table 11.13—36 - Testing ALU elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** *of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
Input	`input`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)
Outputs	`output`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is true.	Same as expected result.	None. (actual result was expected)
Pre-defined inputs	`predefinedInputs`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`. Property `readOnly` is false.	Same as expected result.	None. (actual result was expected)

Table 11.13—37 - Testing ALU elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** *of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
To menu	`menu`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)

execution method	`timeOrStep`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
reset	`reset`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
compile	`compile`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
run	`run`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
manual	`manual`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
submit input	`submitInput`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
toggle mode (light/dark)	`toggleMode`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)
toggle display	`toggleDisplay`	Fetched element is not null. Fetched element is instance of `HTMLButtonElement`.	Same as expected result.	None. (actual result was expected)

*Table 11.13—38 - Testing input/output (IO) elements (all textboxes), of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
testing the table containing the token textboxes (label, opcode, and operand)	`editorTable`	Fetched element is not null. Fetched element is instance of `HTMLTableElement`.	Same as expected result.	None. (actual result was expected)
label (row 0)	`input-0-0`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)
opcode (row 0)	`input-0-1`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)
operand (row 0)	`input-0-2`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)

label (row 1)	`input-1-0`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)
opcode (row 1)	`input-1-1`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)
operand (row 1)	`input-1-2`	Fetched element is not null. Fetched element is instance of `HTMLInputElement`.	Same as expected result.	None. (actual result was expected)

*Table 11.13—39 - The first two lines of textboxes in the script editor, of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

What is tested?	Tested element ID	Expected result	Actual result	Changes made
current status	`status`	Fetched element is not null. Fetched element is instance of `HTMLSpanElement`.	Same as expected result.	None. (actual result was expected)
display box	`displayBox`	Fetched element is not null. Fetched element is instance of `HTMLDivElement`.	Same as expected result.	None. (actual result was expected)

*Table 11.13—40 - Simulator's status, of the simulator page, from **`orginalDOMElements.test.ts`** of sprint #2.*

During amending errors found in the testing, the author discovered that in HTML, the author can just use the syntax `readonly` instead of the full `readonly="true"` in the HTML text boxes, such as the following code line from **`simulator.html`**:

```
<input type="text" class="form-control text-end" id="registerProgramCounter" readonly>
```

11.13.2.4 Effects on other tests

Because sprint #2 only concerns itself with making the frontend code (**`simulatorUI.ts`**) and connecting it to the backend using **`middleware.ts`**, there are no significant changes made to the backend code from sprint #1. Sprint #1 tests were checked anyway, alongside sprint #2 tests, and every test succeeded.

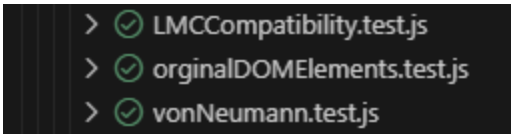


*Figure 11.13—12 - showing test result of jest files **`LMCCompatability.test.js`** and **`vonNeumann.test.js`** that were made in sprint #1 and tested at the end of sprint #2 (UI provided by one of the VS jest extensions).*

11.13.2.5 Testing problems and solutions

11.13.2.5.1.1 Losing direct TS support

Due to the **module format** problem, it had to be changed which caused the ts-jest package to stop working. This is because while jest can utilise other module formats, under the right configurations, the ts-jest extension solely depends on `CommonJS`. When using different modular format ts-jest keep giving errors such as not finding files that are clearly there, invalid configuration, et cetera.

Due to an unexpectedly massive and disruptive amount of time fixing the TS modular format configuration to allow compiled TS to run on the web browser, the author decided to scrap the ts-jest extension and just run the TS-compiled JS jest files instead to implement the testing.

11.13.2.5.1.2 Experimental features

While *jest* has been proven to be compatible with multiple module formats, it struggles to work with `ESNext` despite, Node.js's claimed compatibility with ECMAScript.

11.13.2.6 Justifying deviations from plan and disruptions

11.13.2.6.1 Expected

11.13.2.6.1.1 module format

In sprint 1, the LMC program was just the simulator (`vonNeumann.ts`), so the author knows in the second sprint to change/configure the script files to be modular and to be able to communicate with each other. When using the compiler and simulator files together, they worked together fine because the author was using Node.js to run them. However, when running those with the middleware and frontend in the browser, none of the TS-compiled scripts worked.

The author knew this because he had attempted to change the module format to `ESNext` (a module format the browser is supposed to be compatible with), to use a linter; however, after a few attempts for a few days each, the author got nowhere. Now that the author must make it work now. After much perseverance and a week's worth of constant debugging and research, the author finally fixed this problem. The fix itself was unexpected to the author. In `CommonJS`, the file path in the import statement does not need to have a file name extension, but for `ESNext` it was different. For `ESNext` it was required to add "`.js`" to the file name in the paths. This was very interesting in frustrating because firstly *VS code* automatically completes the line with the extension so it was assumed that it was not needed and secondly adding "`.js`" (not "`.ts`") must also apply to TS files, which conflicts logic of TS syntax compiling to JS syntax.

While this may fix the problem – allowing the modular TS compiled JS files to be modular and work together in the browsers by the calling HTML file – it did cause many other problems, ranging from (jest) automatic test files no longer working to event listeners appearing to not be doing anything. Those problems will be addressed elsewhere.

As the risk management table (**Table 11.10—1**) points out, "TS-compiled JS" code compatibility is dictated to cause significant disruption the project flow which appears to be indeed the case by this and other TS-related deviations of sprint #2.

11.13.2.6.1.2 Getting ts-node to work

Just as mentioned regarding *ts-jest*, the change in module format and the lack of time, which cannot be spared by the author, has been dropped. This is not really a problem because ts-node is primarily used for running in *Node.js*'s runtime environment. Node.js runtime environment was used to run the LMC simulator program, without the need for a UI, in **Sprint 1** development. However, now with UI and the sprint #2's requirement to run in the browser, ts-node has become redundant in the current sprint and hence *ts-node*'s drop is not of any concern to the author.

11.13.2.6.1.3 Von Neumann backend simulator

This part is covered in the **Expected** plan variations of sprint 2, but it is also mentioned here because it is also an expected change.

11.13.2.6.1.4 Event listeners

The author was initially scared because the author was struggling to link HTML buttons and other JS-calling elements to class methods instead of functions. To try to find out how to do so, the author first coded `simulatorUI.ts` both in functions and class to get it working first and then fully to OOP. During this, the author discovered the need to assign event listeners the `interface Window` in the `global` declaration. This was learned from two sources (mdn, 2025; Shepherd and L., 2021).

11.13.2.6.1.5 Commenting

Previously, the author planned to add general comment lines and region name/comment lines. However, due to external interruptions, the author is under a time constraint and thus must skip some lower priorities and substitute other lower priorities. For commenting, the author initially wanted to use the double slashes and hashtag and notations ("*//*" and "*#*") for compiler to know not to compiles those for the script editor to combine all 3 textboxes of that line (label, opcode, and operand) into one to accommodate the comment. This lower priority has been scrapped but the user can still add comments to the label and operand textboxes, without the need for notations, if the corresponding opcode textbox is empty. This is because the compiler ignores any line with a missing opcode for validation purposes (as no instruction or variable exists without an opcode). This way, the user still has a way to comment without spending precious time developing it. Commenting is lower priority due to its complexity (collapsing region blocks and converting 3 textboxes into one based on content) and because script lines can easily be described by the label column – where the user can have unreferenced labels declarations.

11.13.2.6.1.6 Default switch cases

In some switch-statements, the author has included the default case while in others it is not. This was the case then because it was guaranteed that all subjected values match one of the cases within the code-space. However, after discussion with the author's supervisor, the author decided to make all switch-statements include a default case. This was decided because the supervisor pointed out that unexpected errors and scenarios are not only caused by source code but also "undefined behaviour". Undefined behaviour is seemingly random unexpected errors and scenarios where they are caused issues independent of the code, such as hardware issues (such as memory corruption, non-volatile storage failures, CPU malfunction, et cetera), OS bugs disturbing process management for the program, et cetera. However, undefined behaviours tend to happen very rarely so there will not be

code to directly address undefined behaviours but it is still indirectly addressed by the default case as whatever happens as long as one of the cases is satisfied, the application will not encounter any errors.

11.13.2.6.1.7 Variation from UML

Due to UML is made with generic and vague pseudocode syntax, structure and execution, instead in JS or in TS, there are bound to be some variations. The list below shows the variations from the sprint 2 variation plan:

- `Compiler`
  - Due to the sheer number of kinds of error messages, a separate method is dedicated to generating the errors based on error type and relevant data and storing it in the `message` field.
  - Initially, only used `TokenType` enumeration for simpler communication, but now also have `ErrorType` enumeration to tell the error generation method what kind of error. Decided to use it because of the sheer number of error kinds.
  - Now has a constructor to assign a success message to the 'message' fields because the compiler assumes that the script is valid until proven otherwise, for a more simplistic code structure.
- `ControlUnit`
  - Constructor now has an optional parameter for the parent class to update the frontend UI.
  - `.getArithmeticStatus` is redundant due to `.displayStatus` taking its purpose but is still kept because of the sprint 1 tests.
- `MemoryUI`
  - Code for generating the memory table, and its contents (textboxes and labels), has been moved from the constructor to its own method `.generateTable` due to its code complexity (code does not have to be big to be complex).
- `MiscellaneousUI`
  - Added methods `.disableDuringRun` and `.enableAfterRun` methods to deal keep certain features enabled only when not executing a LMC script, such as the run button.

11.13.2.6.1.8 Others

There are many other minor expected deviations for the plan that are too small to be their own sub-section but are still worth mentioning, hence will be displayed in the following list:

- `ControlUnit` in **`vonNeumann.ts`** – Changed default interval speed from `2000`ms to `4000`ms. This is to give user time if they want to switch execution modes before the next cycle.
- **`middleware.ts`** and **`simulatorUI.ts`** – Adding body code that performs a `global` augmentation of the built-in `Window` interface. This allows dynamically generated HTML elements, after original DOM load, to call the modular JS (TS compiled) code when conditions meet.
- `SimulatorUI` in **`simulatorUI.ts`** – Moved methods `newCycle`, `switchCycleModes`, and `changeSpeed` to `Middleware` class in **`middleware.ts`** because methods need to directly call methods in `ControlUnit` in **`vonNeumann.ts`**.
- `ControlUnit` in **`vonNeumann.ts`** – Added a 1ms sleep when in execution mode is manual. This is because when the user has not unpaused for the next cycle, the program still iterates via the while-loop of the `cycle` method but only partially and without the `sleep` call, there will not be any reached `sleep` calls to stop busy-waiting. This is critical because busy-waiting causes the browser tab, that contains the LMC web application, to freeze indefinitely as seen below this list in **Figure 11.13—13**.

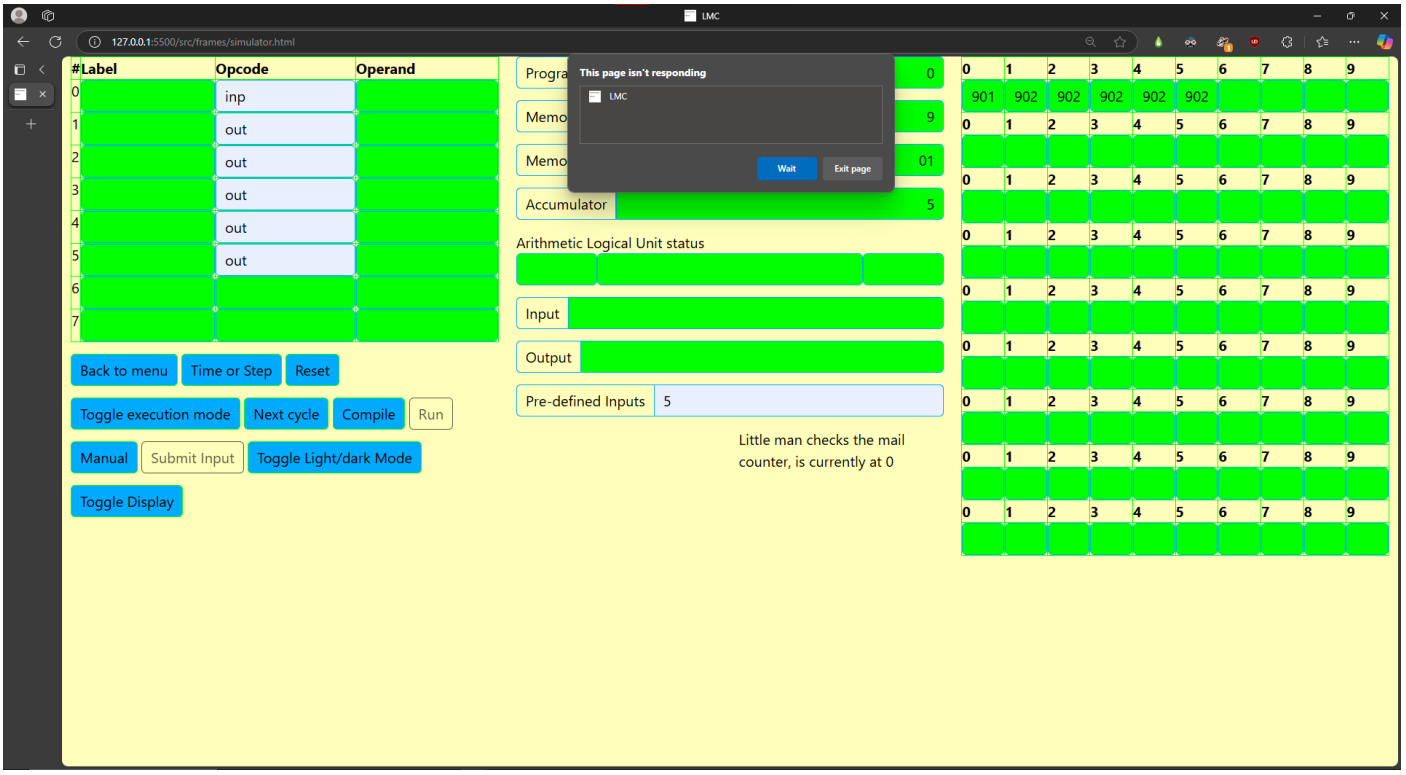


Figure 11.13—13 - browser tab freezing resulting from busy-waiting caused code of sprint 2.

11.13.2.6.2 Unexpected

11.13.2.6.2.1 Advanced CSS features

The author has previously assumed the functionality of the CSS to be highly simplistic because every time the author searches for any advanced thing related to CSS (like direct inheritance, importing, built-in functions, et cetera), sources always point towards using a CSS superset such as SASS or SCSS. However, while using CSS with the Bootstrap CSS framework, the author has discovered more functional features of CSS (sources of learning the features as in-text citations):

- "!important" declaration (W3Schools, 2021g)

  Explanation: It allows overriding all relevant rules/properties no matter their priority.

  Reason for usage: Allows the configuration of properties of pre-made elements from Bootstrap CSS, where otherwise it would not be possible to do. One such example is changing the background of a Bootstrap CSS-styled textbox (named "input"). Without the "!important", the only other way is to use a CSS superset for its direct CSS file inheritance feature.

```
7   * {
8       border-width: 0.1rem !important;
9       /*^ "!important" overwite bootstrap class "input-group-text"*/
10      font-size: 1.25rem !important;
11      /*^ "!important" overwrites bootstrap classes (such as "input-group-text") which are not possible to overwite without */
12  }
```

Figure 11.13—14 - some example uses of the `!important` syntax.

- CSS variables (also called "custom properties") (W3Schools, 2018)

  Explanation: allows assigning a custom property to a variable to be used in place of properties of elements/classes.

  Reason for usage: Allows CSS code for both light mode and dark mode to take around half the code and complexity than without – makes the CSS file look more like one theme than two. This is a very good idea because the author wanted multiple style themes with both light and dark modes for each of them. Using variables, the developer can halve the required work as light and dark mode will be included in each theme rather than being separate themes as well as having double the number of styles than just themes. Currently, in sprint 2, light and dark modes are used in the CSS file. But because the only difference between the themes is colour schemes, it may also be possible the fit all themes in one CSS file but still maintain the same organisation/readability as if only style is used. The only difference is that each style being declared in the same file, but it will be highly simplistic.

```css
:root {
    --text-color: #000;
    --background-color: #FFB;
    --secondary-color: #0AF;
    --primary-color: #0F0;
}

[data-theme="dark"] {
    --text-color: #FFF;
    --background-color: #886;
    --secondary-color: #058;
    --primary-color: #080;
}
* {
    color: var(--text-color)!important;
}
body {
    background-color: var(--background-color)
}
span {
    background-color: var(--background-color) !important;
    border-color: var(--secondary-color) !important;
}
```

Figure 11.13—15 - code snippet showing use of custom properties in CSS.

One of the main reasons for not discovering these earlier is probably because these features are relatively newly added into the CSS system, while CSS supersets had those time ago – meaning any internet source prior will direct to the CSS superset. The other main reason is that most developers who need to create complex CSS files, use a CSS superset instead to achieve the same result in less, but more readable and scalable, code – making it a favourite upon forums, blogs, and tutorials.

For proof on concept, the author edited the CSS file for the simulator's default theme to have not only light mode but also dark mode in accordance with the following **Table 11.13—41**.

Class/component	Colour (light mode)	Hex code – light mode	Hex code – dark mode
Page background	Creamy	#FFB (#FFFFBB)	#886 (#888866)
Text colour	Black	#000 (#000000)	#FFF (#FFFFFF)
Primary colour	Green	#0F0 (#00FF00)	#058 (#005588)
Secondary colour	Blue	#00F (#0000FF)	#080 (#008800)

Table 11.13—41 - Colour style for the default style in both light and dark mode, implemented in sprint #2. The idea is that it originally is light mode but lower the hex values by around half for an optional dark mode, hence only colour names only for light mode.

11.13.2.6.2.2 Messy source code

The author intended to learn and copy ideas from the source code of the other available LMC simulators (mentioned in the software research) – LMC#1, LMC#3, LMC#4, and LMC#5. The author does not mind if the source code is not in TS or even another language besides JS as long as the idea and methodology behind the workings of the script compiler, of the LMC simulators, is understood. However, all the LMCs are difficult to understand for the following reasons:

- LMC#1 – Both frontend and backend are all in the same JS file – making it extremely difficult to comprehend what is going on in the file. The fact that the JS file is done in function programming (not separated into specialised classes or groups of functions), and the complete lack of comments makes it nearly impossible to understand the idea behind the code.
- LMC#3 – Source code is in the Elm language, which would be fine if understandable, but the syntax and code/token structure of that language is so alien-like compared to the languages the author worked with before. To the author, Elm to JS seems to have the same comparison as Bash to Python which emphasises the difference. Obviously, the author does

not believe it to be worth to learn and getting used to a differently structured language just for this in a very limited time span. Elm does compiles to JS, but that is even more difficult to understand in the following screenshots:

```
65  function F8(fun)
66  {
67    function wrapper(a) { return function(b) { return function(c) {
68      return function(d) { return function(e) { return function(f) {
69      return function(g) { return function(h) {
70      return fun(a, b, c, d, e, f, g, h); }; }; }; }; }; }; }; };
71    }
72    wrapper.arity = 8;
73    wrapper.func = fun;
74    return wrapper;
75  }
76
77  function F9(fun)
78  {
79    function wrapper(a) { return function(b) { return function(c) {
80      return function(d) { return function(e) { return function(f) {
81      return function(g) { return function(h) { return function(i) {
82      return fun(a, b, c, d, e, f, g, h, i); }; }; }; }; }; }; }; }; };
83    }
84    wrapper.arity = 9;
85    wrapper.func = fun;
86    return wrapper;
87  }
```

Figure 11.13—16 – LMC #3's JavaScript showing a large amount of recursive calling and parameters in the `fun` function

```
125  function A8(fun, a, b, c, d, e, f, g, h)
126  {
127    return fun.arity === 8
128      ? fun.func(a, b, c, d, e, f, g, h)
129      : fun(a)(b)(c)(d)(e)(f)(g)(h);
130  }
131  function A9(fun, a, b, c, d, e, f, g, h, i)
132  {
133    return fun.arity === 9
134      ? fun.func(a, b, c, d, e, f, g, h, i)
135      : fun(a)(b)(c)(d)(e)(f)(g)(h)(i);
136  }
```

Figure 11.13—17 - LMC #3's JavaScript showing a very unorthodox way of coding functions

```
5245    var _elm_lang$core$Dict$partition = F2(
5246            function (predicate, dict) {
5247                    var add = F3(
5248                            function (key, value, _p59) {
5249                                    var _p60 = _p59;
5250                                    var _p62 = _p60._1;
5251                                    var _p61 = _p60._0;
5252                                    return A2(predicate, key, value) ? {
5253                                            ctor: '_Tuple2',
5254                                            _0: A3(_elm_lang$core$Dict$insert, key, value, _p61),
5255                                            _1: _p62
5256                                    } : {
5257                                            ctor: '_Tuple2',
5258                                            _0: _p61,
5259                                            _1: A3(_elm_lang$core$Dict$insert, key, value, _p62)
5260                                    };
5261                            });
5262                    return A3(
5263                            _elm_lang$core$Dict$foldl,
5264                            add,
5265                            {ctor: '_Tuple2', _0: _elm_lang$core$Dict$empty, _1: _elm_lang$core$Dict$empty},
5266                            dict);
5267            });
```

Figure 11.13—18 - LMC #3's JavaScript showing another very unorthodox way of coding functions

```
9253                                                                    _1: {
9254                                                                            ctor: '::',
9255                                                                            _0: {
9256                                                                                    ctor: '_Tuple2',
9257                                                                                    _0: 'DAT',
9258                                                                                    _1: _pbrinkmeier$lmc_emulator$Lmc_Parser$OneArgument(_pbrinkmeier$lmc_emulator$Lmc_Parser$Data)
9259                                                                            },
9260                                                                            _1: {ctor: '[]'}
9261                                                                    }
9262                                                            }
9263                                                    }
9264                                            }
9265                                    }
9266                            }
9267                    }
9268            }
9269    }
9270 }
9271 });
```

Figure 11.13—19 - LMC #3's JavaScript showing terrible programming practice by the sheer amount of indentation.

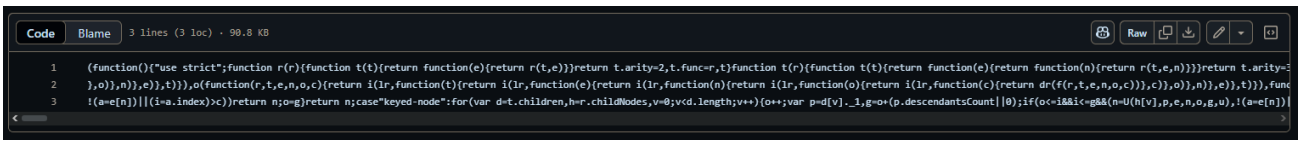


Figure 11.13—20 – one of LMC #3's JavaScript files without format nor readable spacing.

- LMC#4 – Both backend and frontend are all in the same JS files, where the different JS files are simply different variations of each other. The JS has a bit of commenting but is in functional programming (instead of OOP), and the file is 3000+ lines due to much of the code being repetitive and hardcoding.
- LMC#5 – Also has both backend and frontend all in the same JS files. It does have some commenting and is OOP but the mix of backend and frontend in the classes, not much commenting in the 'Editor' class, and many of the methods returning entire objects which makes it very hard to follow.

Note: LMC#2 and LMC#6 are not mentioned because they do not have their source code available to the public, as mentioned back in **Table 5.1—2** of the software research.

This left the author alone and somewhat misguided in both the planning, development, and implementation of the second sprint.

11.13.2.6.2.3 Inputting

One major unexpected problem was that the author was unable to get the input box to work. Difficulty in making it work most likely stems from how, when input is needed, a number promise is returned where it adds an event listener to take in the input. It is done that way to prevent the user from inputting numbers when they are not needed. The author believes that the concept of combining event listeners with returning promises concepts into one is what causes this misguidance, even with many referred guides and tutorials. This is so because the author has successfully implemented promise-returning methods, in `vonNeumann.ts`, and event listeners, in `simulatorUI.ts`, separately.

Since the author has time limited more than expected, he decided to classify the input box concept as a lower priority. It was able to be lower priority because the user can still input using the pre-defined input box for inputs. For now, the input box, and its corresponding buttons, are disabled and calling them will only return 0.

If the third sprint gets done early, then the input feature can be retried, but it will probably involve drastically changing how the LMC program works to get it working properly.

11.13.2.6.2.4 Others

There are many other minor unexpected deviations from the plan that are too small to be their own sub-section but are still worth mentioning, hence will be displayed in the following list:

- `ControlUnit` in `vonNeumann.ts` and `Middlware` in `middlware.ts` – Default cycle interval speed and maximum cycle interval speed (4000ms and 10,000ms respectively) has an extra 1ms added to them (4001ms and 10,001ms respectively) so speed can increment by 2000ms between 1ms and 10,001ms with far simpler logic then if was using previous values – no need to take making sure speed is either multiples of 2000 or is just 1 in range.
- `ControlUnit` in `vonNeumann.ts` – Fields `highestSpeed` and `slowestSpeed` have been renamed to `lowestSpeed` and `fastestSpeed`, respectively, to make better sense. Changes have also been mirrored in `Middleware` of `middleware.ts`.
- `ControlUnit` in `vonNeumann.ts` – Changed `newCycle` and `cycleInterval` to private to make all fields private for better programming practice.
-

11.13.3 Review

11.13.3.1 Current state of code

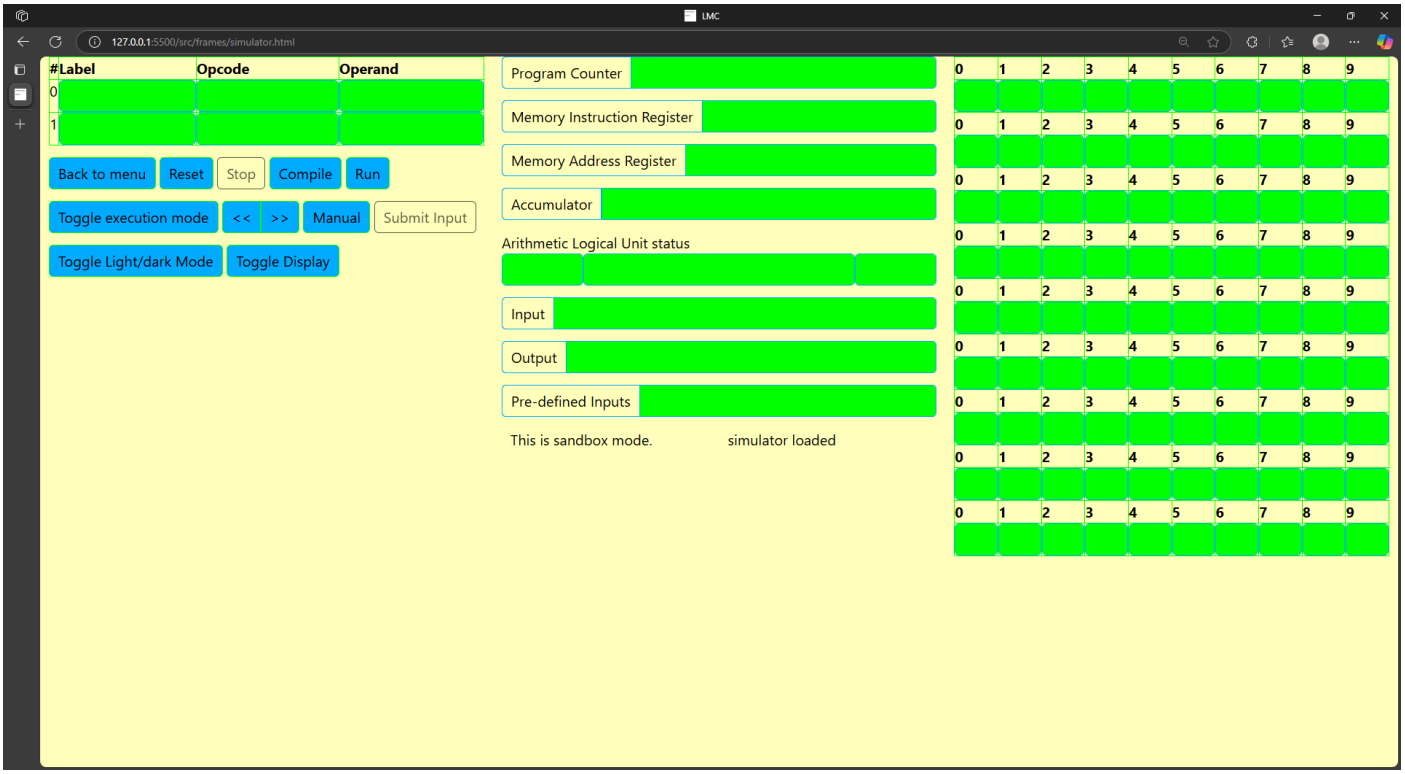


Figure 11.13—21 - showcase of the simulator page, in sprint #2, at first glance.

Figure 11.13—22 - showcase of the simulator page, in sprint #2, in dark mode.

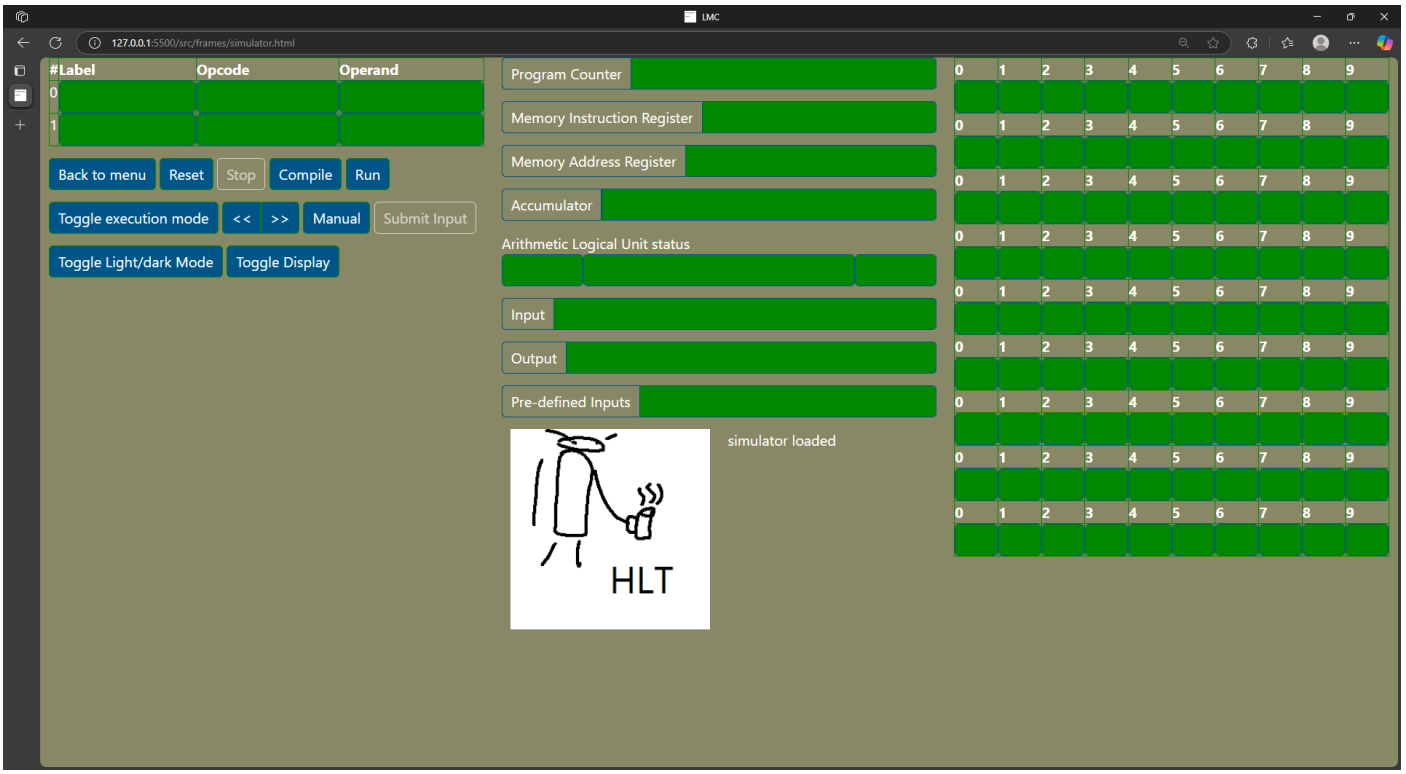


Figure 11.13—23 - showcase of the simulator page, in sprint #2, in dark mode and toggled display.

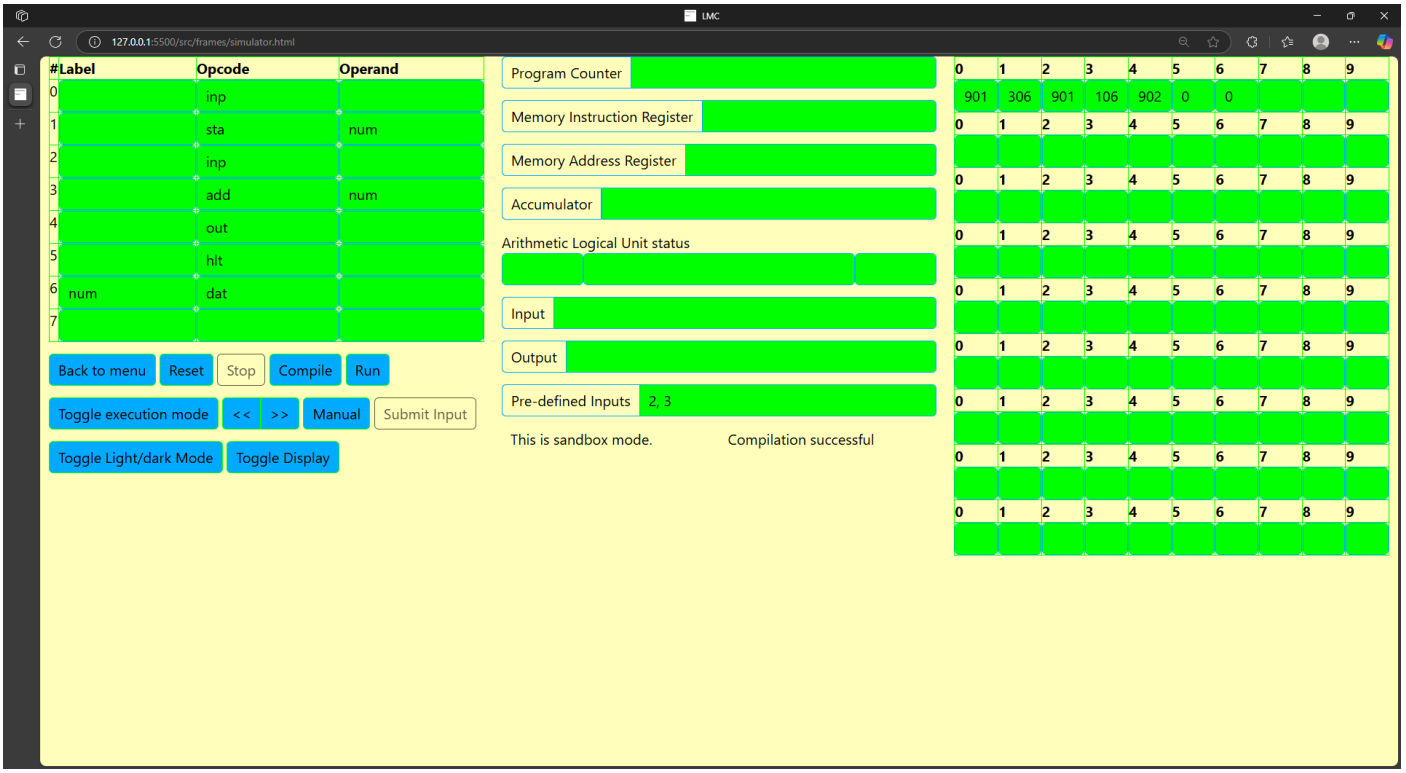


Figure 11.13—24 - showcase of the simulator page, in sprint #2, with a compiled script.

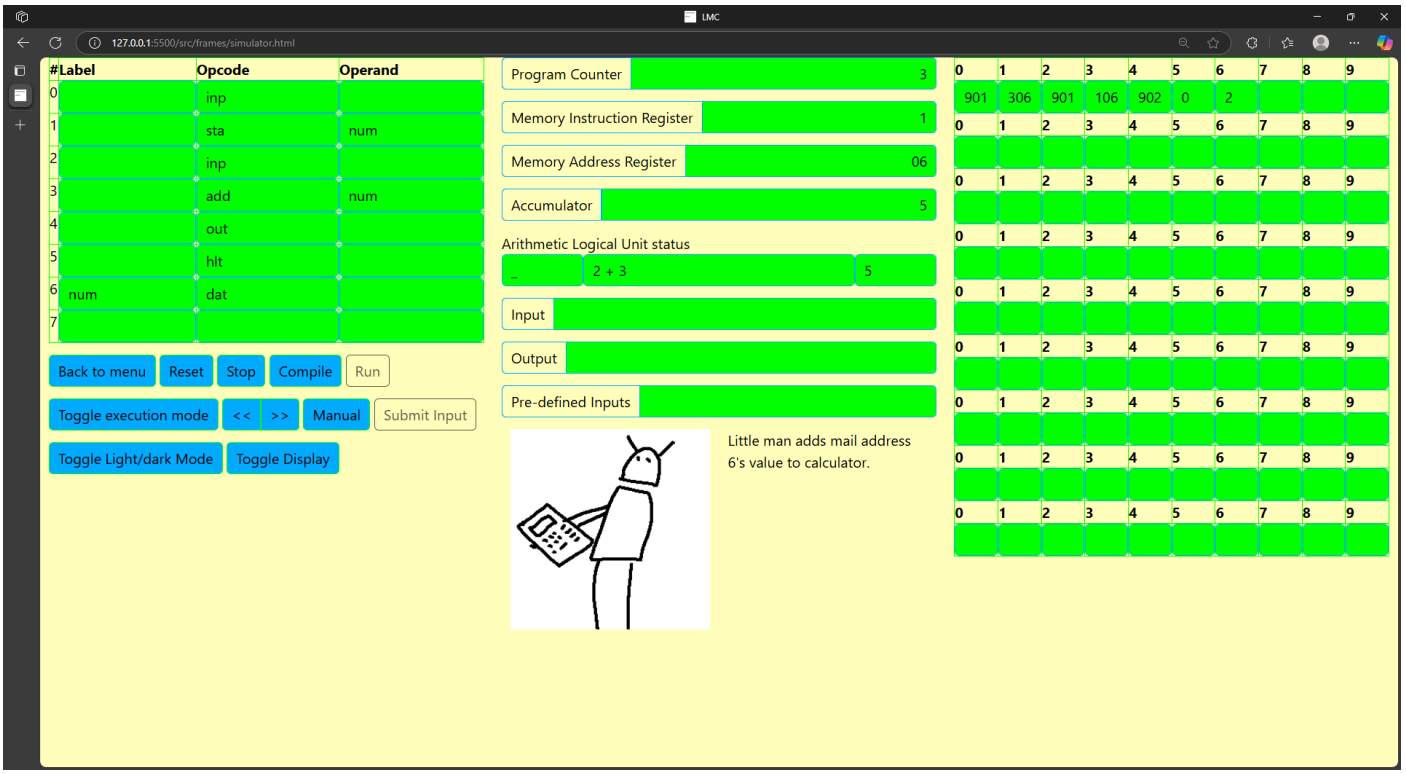


Figure 11.13—25 - showcase of the simulator page, in sprint #2, running an assembly script.

Figure 11.13—26 - showcase of the simulator page, in sprint #2, changing execution speed.

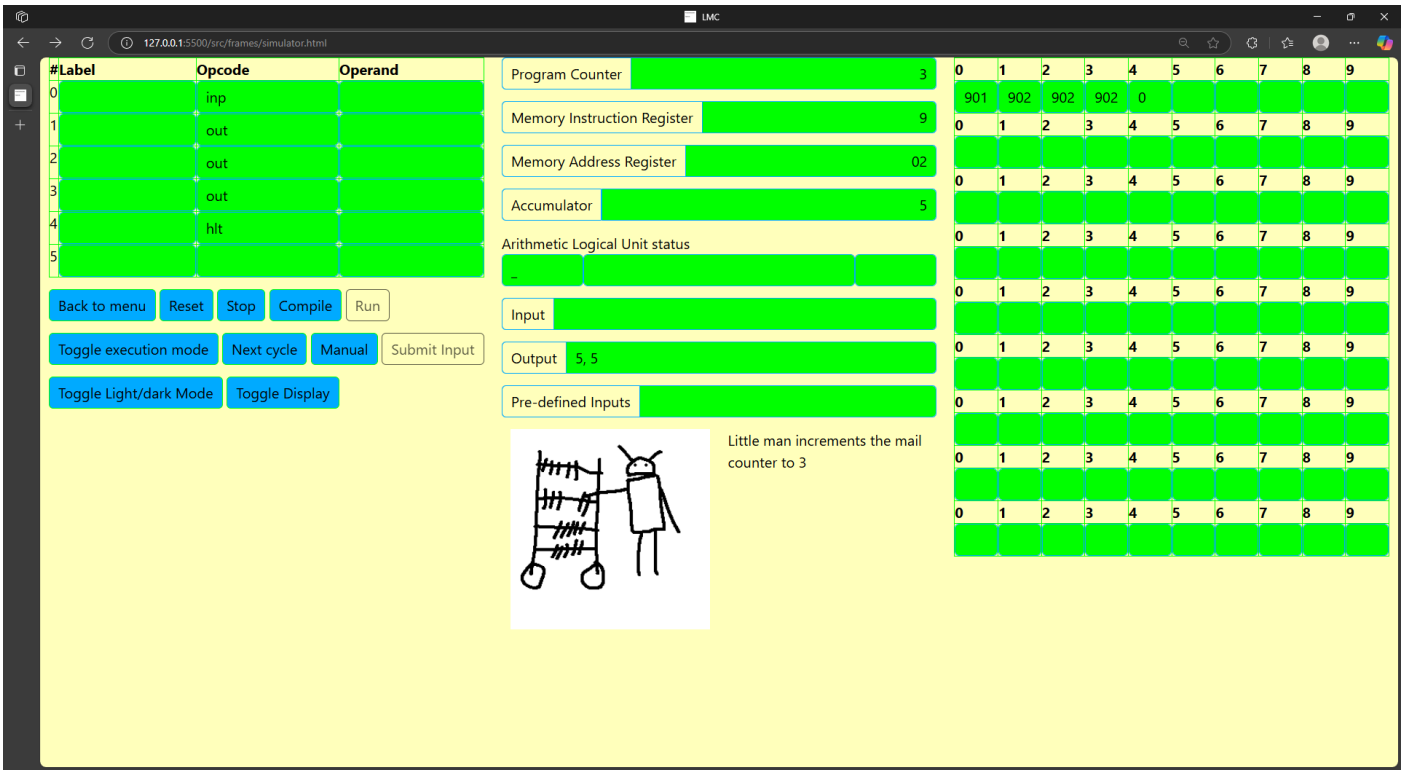


Figure 11.13—27 - showcase of the simulator page, in sprint #2, with execution mode toggled to manual.

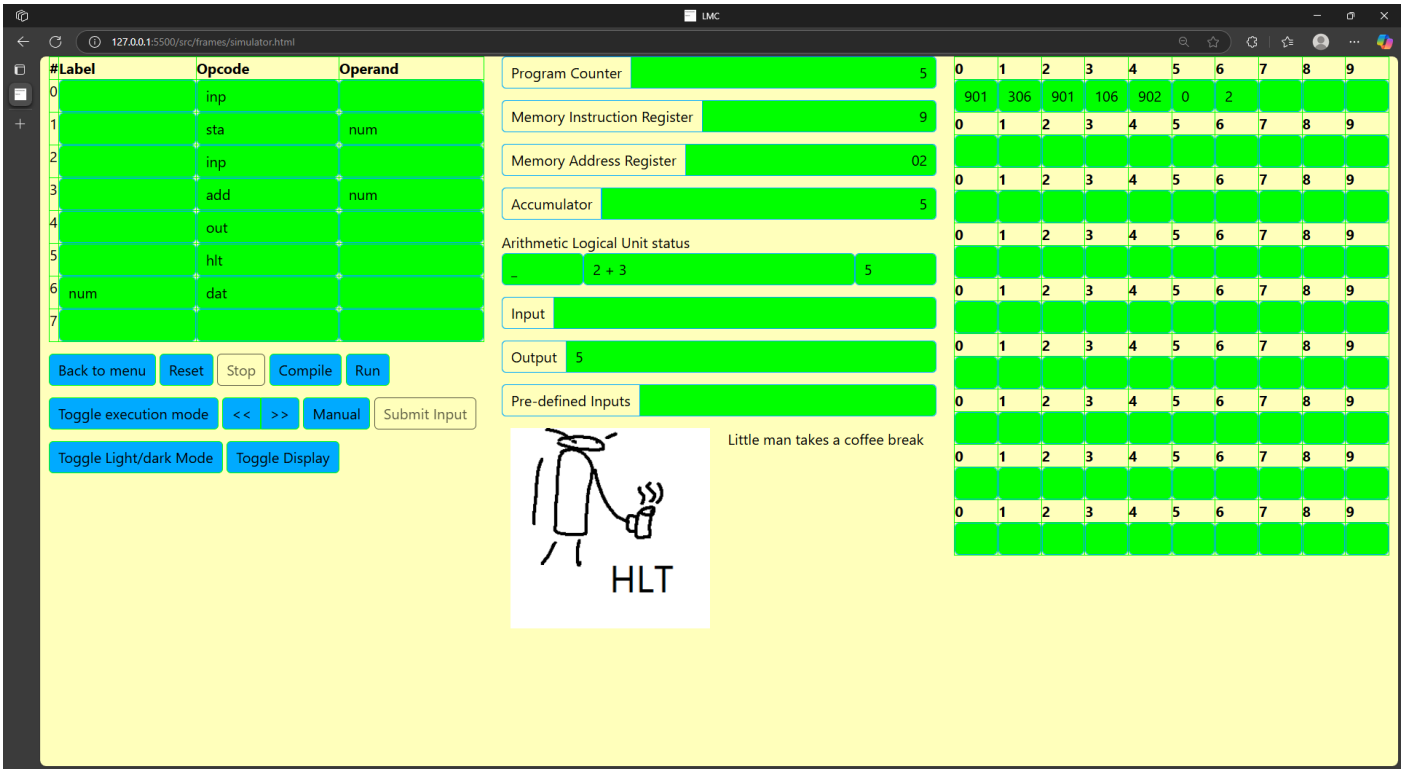


Figure 11.13—28 - showcase of the simulator page, in sprint #2, finished executing an assembly script.

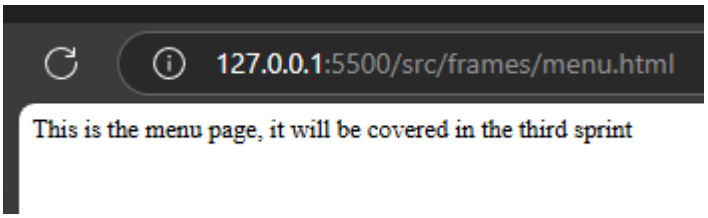
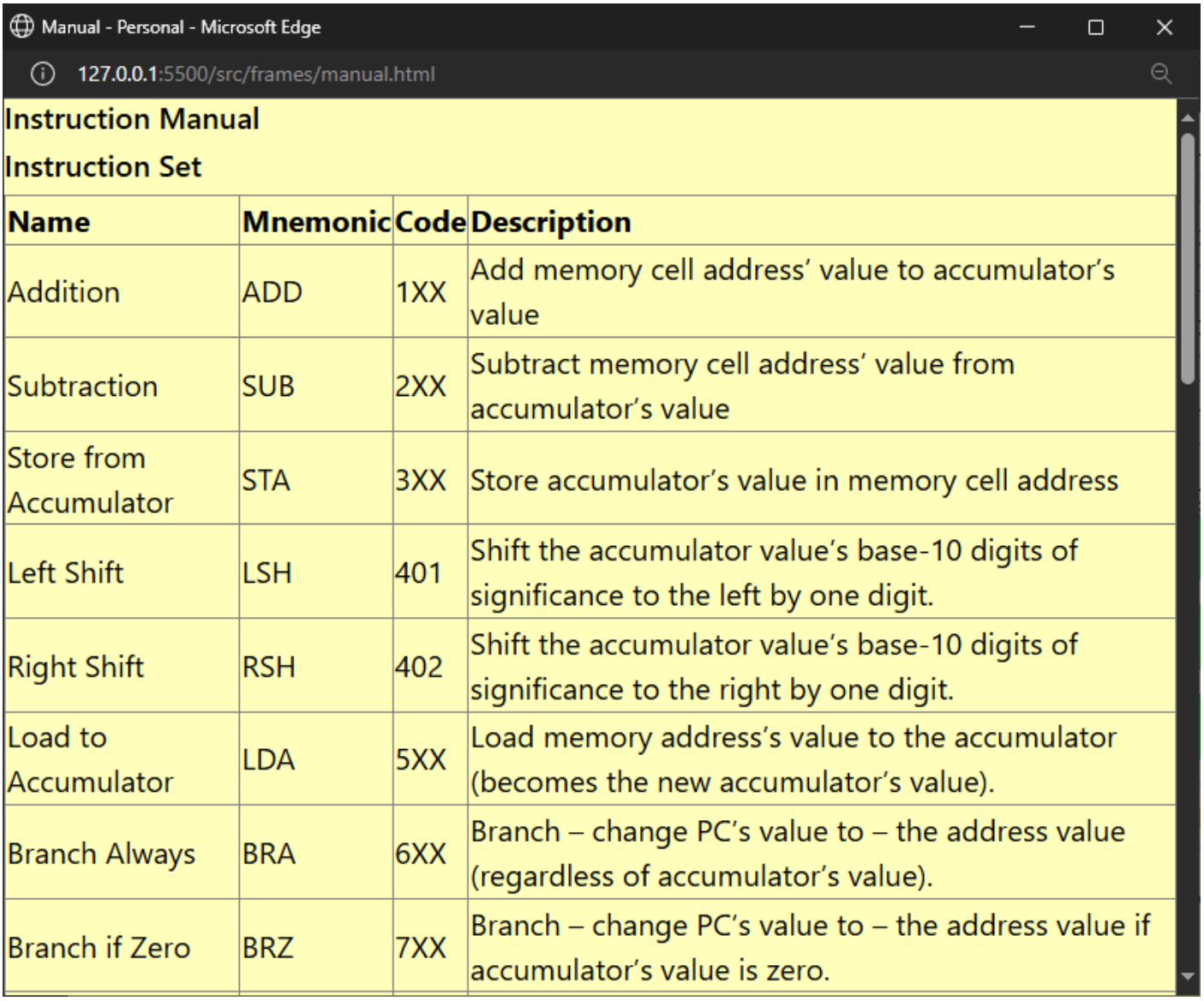


Figure 11.13—29 - showcase of the menu page, in sprint #2, redirected from the "Back to menu" button.



Instruction Manual

Instruction Set

Name	Mnemonic	Code	Description
Addition	ADD	1XX	Add memory cell address' value to accumulator's value
Subtraction	SUB	2XX	Subtract memory cell address' value from accumulator's value
Store from Accumulator	STA	3XX	Store accumulator's value in memory cell address
Left Shift	LSH	401	Shift the accumulator value's base-10 digits of significance to the left by one digit.
Right Shift	RSH	402	Shift the accumulator value's base-10 digits of significance to the right by one digit.
Load to Accumulator	LDA	5XX	Load memory address's value to the accumulator (becomes the new accumulator's value).
Branch Always	BRA	6XX	Branch – change PC's value to – the address value (regardless of accumulator's value).
Branch if Zero	BRZ	7XX	Branch – change PC's value to – the address value if accumulator's value is zero.

Figure 11.13—30 - showcase of the manual page, in sprint #2, as a popup from clicking the "Manual" button.

Instruction Manual

Instruction Set

Name	Mnemonic	Code	Description
Addition	ADD	1XX	Add memory cell address' value to accumulator's value
Subtraction	SUB	2XX	Subtract memory cell address' value from accumulator's value
Store from Accumulator	STA	3XX	Store accumulator's value in memory cell address
Left Shift	LSH	401	Shift the accumulator value's base-10 digits of significance to the left by one digit.
Right Shift	RSH	402	Shift the accumulator value's base-10 digits of significance to the right by one digit.
Load to Accumulator	LDA	5XX	Load memory address's value to the accumulator (becomes the new accumulator's value).
Branch Always	BRA	6XX	Branch – change PC's value to – the address value (regardless of accumulator's value).
Branch if Zero	BRZ	7XX	Branch – change PC's value to – the address value if accumulator's value is zero.
Branch if Positive	BRP	8XX	Branch – change PC's value to – the address value if accumulator's value is positive or zero (not negative).
Input	INP	901	Takes user's or predefined input and store it as accumulator's value.
Output	OUT	902	Outputs from accumulator's value (as integer).
Output as Character	OCT	903	Outputs from accumulator's value as an ASCII character
Halt	HLT	0	Stops program
Data location	DAT	N/A	Compile data into memory before program starts (can be used as variables)

Script Editor

Use the following keys to navigate:

- Tab key - Go down a line if in an operand box.
- Space key - Go down a line if in an operand box.
- Enter key - Go back to the top line if on the bottom line.
- Downwards key - Go back to the top line if on the bottom line.
- Upwards key - Go to the bottom line if on the top line.

Typing on the bottom code line/row will generate a new empty one underneath.

Registers

Name	Description	Valid range
Program Counter (PC)	Dictates where the simulator fetches its next instruction (which RAM cell).	0 to 99
Memory Instruction Register (MIR)	Gathered from the first digit of the fetched instruction from memory.	0 to 9
Memory Address Register (MAR)	Gathered from the last two digits of the fetched instruction from memory.	0 to 99
Accumulator	A form of immediate storage for processor operations such as adding, outputing, inputing, etc.	-999 to 999

Simulator Controls

Controls relavent to simulator

- Run - Runs the compiled program as seen in the RAM table (requires valid compilation first)
- Compile - Attempts to compile script and store in memory.
- Stop - Stops currently executing assembly program.
- Reset - Reloads page to rid off unexpected errors (and current script, so think twice before reloading).
- Toggle Display - Switches between Little Man action display and the current campain level objective (does not really matter if in sandbox mode).
- Toggle Execution Mode - Switches between continuously running each cycle, but with configurable speeds, or only execute next cycle when "New Cycle" button is pressed.

Figure 11.13—31 - showcase of the manual page, in sprint #2, in its entirety.

11.13.3.2 What went well

The author believed he had done well in dealing with solving problems that would otherwise make part of the sprint unable to do. Whether it was researching an alternative or critical thinking.

11.13.3.3 What would be better and used for the next sprints

There were still some problems that were not fully solved or still have side effects, such as removing *ts-node* which costed a lot of time due to debugging and editing two different files for the same set of tests.

The author's research indicated that ESNext is needed for modular TS compiled JS to work in the browser but in fact it is a beta-version for the latest version of ECMAScript. It would save a lot of time and unnecessary changes to use a fixed non-prerelease version of ECMAScript instead of ESNext. The author would know if he did much more research and study of these configuration topics before development, but the author has too little time to do so.

This leads to the next thing that would be better, which is more time. Not only the unexpected changes and the unexpected time needed to do the expected changes costed a lot of time but the author's health has degraded a bit where his eye's dryness, burning, and nausea and migraines from using the eyes has caused many instances of the author not able to focus on anything for several days without immense pain. Even with the recently acquired Learning Support Plan, the author cannot extend the project's deadline much due to its proximity to the end of the academic year and the university degree. The best the author can do is to make the most of what is currently available.

There really is not much the author can do to overcome these problems except for scraping features of lower priority at first, before the LMC educational game is finished. The original plan was to only do a few refinements and polishing on the code after the third sprint but with the number of lower priorities postponed, because of time being potentially not enough, a fourth sprint may be needed to address some low priority features if cannot fit with in sprint #3's timeframe.

As of now, the current low priorities that are postponed (from the second to the third sprint) are:

- Fixing the inputs.
- Cookies to save game progress and preferences.
- Refining improvised code (from both the expected and unexpected changes).
- Adding sound effects to the simulator.
- Having the Little Man action images animated.
- Remodelling HTML tables.

Higher priorities that are postponed:

The reader may assume that “refining improvised code” is a vague and redundant saying, but the author has specific meanings behind it. One such is code dealing with semantic errors. In sprint 1, the author just used in-statements to deal with them, but in sprint #2, type assertions were discovered and used by the author as an alternative for HTML elements or other complex data structures as mentioned before. However, this method of choosing which solution for which specific instance of error is improvised and needs to be reworked because of that. If the author does get to the point of the fourth sprint in time, this is how the author would redo and make a planned and thought-out approach to choosing which solution will be used in the table below.

Scenario	Use an if-statement or a type assertion solution	Why?
Assigning a HTML element by using `document.getElementById`.	Depends on the HTML element in question.	If HTML element to loaded alongside the original DOM then use type assertion as their existence is absolute and even tested in the automatic test files. Otherwise, if dynamically generated, then use if-statements as they should exist, but more likely for the unexpected case of assigning `null` instead due to their dynamic nature.
Elements from a list (no matter the dimensions)	Type assertions	TS uses dynamic and mutable lists instead of arrays, meaning all elements are added values. This makes unexpected undefined values far less likely than getting an out-of-bound error – with negligible chances.
Fetching class fields	Type assertions	Fields will not be `null` nor `undefined` as setter methods and constructor methods will make sure of that.
Variables with 2 potential data types (such as `private middleware:Middleware|undefined`)	If-statement	Variables/fields made this way assume that it really can be one of the two, so and type assertion assumes that there can only be one possible type.

Table 11.13—42 - which errors should be solved using if-statements or type assertions, done with reasoning.

Note that identifications of the semantic error types, as discussed before, are not included in this table. This is because it is more about the nature of the storing variables themselves rather than of the error.

While the HTML tables work perfectly fine, it would still be better to remodel them. This is because the author has not included the table header section tags `<thead>`. The author did not use them because he was simply not aware of their existence, which is reasonable because the normal row `<tbody>` does functionally the exact same thing besides visual presentation, but that can easily be compensated with HTML custom classes.

11.13.3.4 Changes to previous sprint

11.13.3.4.1 Von Neumann backend simulator

11.13.3.4.1.1 Communicating with middleware

As mentioned, Von Neumann will be updated to update and output to the frontend UI (`simulatorUI.ts` via the middleware) in the second sprint. For the plan, the author decided to update the `ControlUnit` class to communicate with the middleware by taking in the parent class’s reference so it can call its methods to update the frontend UI. However, there is the problem where changing the simulator (specifically the `ControlUnit` class) may affect how the simulator behaves and make all sprint 1 tests invalid.

For a solution (for the simulator updating the UI but still having the same behaviour), the author has researched and learned about optional arguments (TypeScript, 2023). This helps because taking in the parent class as an optional constructor argument (and making the `ControlUnit` class work whether the argument has the parent class reference or undefined) allows the class to work and not the sprint 1 tests without changing them.

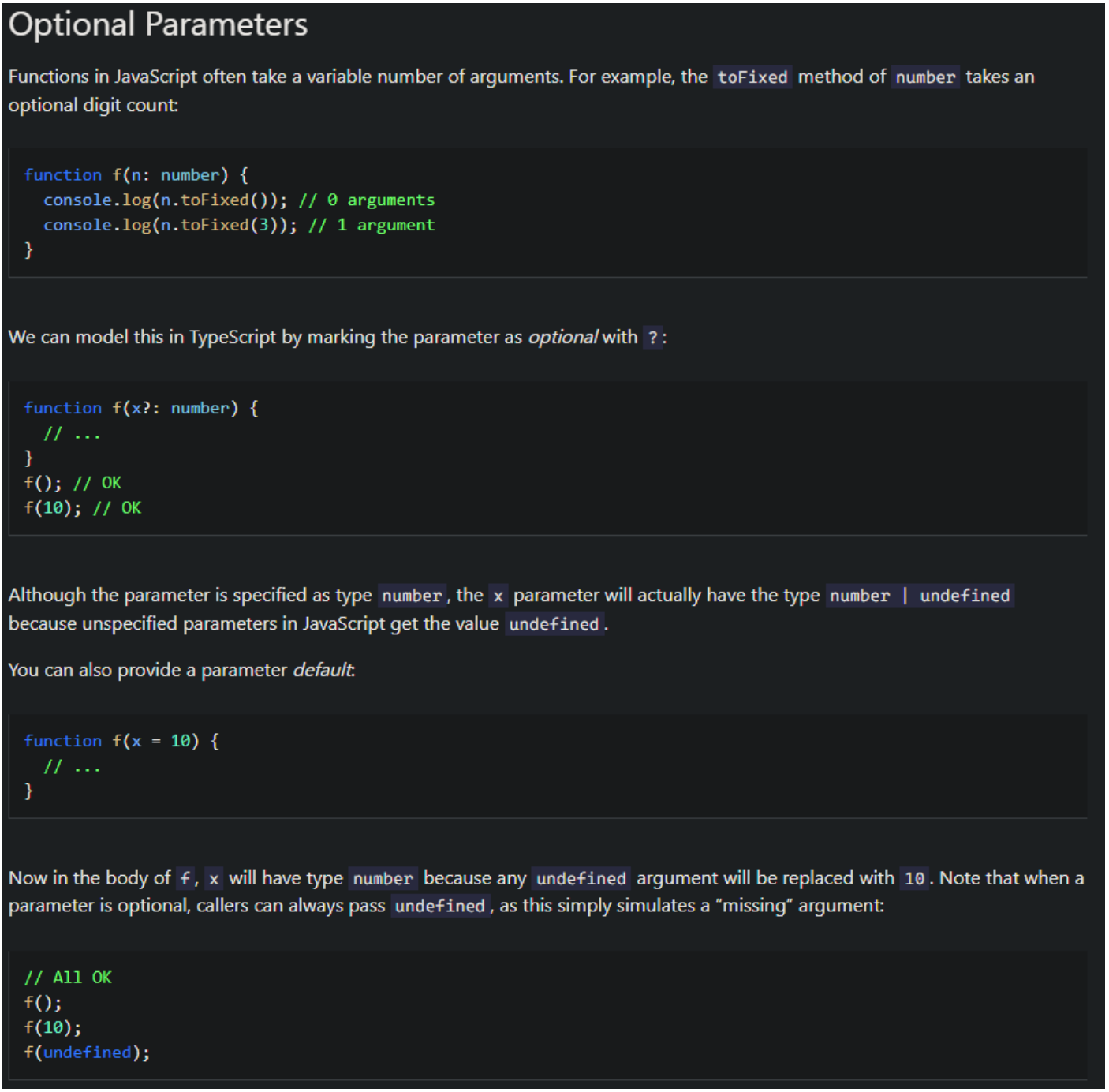


Figure 11.13—32 - Explanation on how optional parameters work in TS (TypeScript, 2023).

This also enforces the author's coding style of modularity and reusability as it allows the simulator backend to continue to have the ability to work in other projects that do not have a frontend UI to connect to.

11.13.3.4.1.2 Pascal base

The author also enforced the Pascal Case (also known as Upper Camel Case) naming convention by changing the original class name from `controlUnit` to `ControlUnit`. The author did originally want to use `ControlUnit` in sprint 1, but did not due to a careless naming convention violation mistake. This was first delayed in the second sprint due to the number of instances that the class was called in the project, but after researching a time-saving way, the author found a VS-code feature that allows all instances of a text to be replaced with another per file (Visual Studio Code, 2025). Very convenient due to the sheer number of `controlUnit` instances in the project, especially in the relevant automatic Jest files.

11.13.3.4.1.3 Completions

Many things were left unfinished in sprint #1 because they require the frontend and middleware of the sprint 2 development to complete. Below is a list of the previously empty, or partially complete, parts of the `ControlUnit` class that got completed in sprint 2:

- Execution modes – added *cycleModeAutomatic* field so the simulator knows if it is to pause for a set time or wait for the user to unpause (clicking “Next Cycle” button).
- Output communication method – Completed the *displayStatus* method, allowing the frontend to display all relevant information to the user, such as: output instruction result, current values of registers, current status and progress of current cycle, et cetera.

Other changes relating to *ControlUnit*, of **vonNeumann.ts**, are listed in the **Justifying deviations from plan and disruptions** section.

11.13.3.4.1.4 Auto-timeout

When doing Blackbox testing the simulator (testing as a user), the author has found a critical vulnerability with the simulator. The vulnerability was when the user creates an infinite branching loop in the assembly code and runs it, which lasts indefinitely. This was not necessarily critical in sprint #2 directly because the user can simply stop the program’s execution by clicking on the “Stop” button. However, sprint #3 will introduce level checking, where *LevelChecker* (from **levelChecker.ts**) checks the user’s script by running it in the background. This means if it checks a script that has an infinite loop, it will be stuck in an infinite loop where the only way to stop is for the user to click on the “Reset” button which refreshes the page. Refreshing the page will clear the user’s own script. This resets the user's progress and effort (for both campaign mode and sandbox mode), which must be avoided at all costs.

To counter this problem, auto-timeout functionality has been merged with the manual stop functionality of the simulator (*ControlUnit*) of **vonNeumann.ts**. This has been done by replacing the Boolean in *ControlUnit*, used to indicate if script execution has been manually stopped or not, with a counter. The counter increments per cycle and the while-loop of *ControlUnit.cycle()* will end once the counter equals or exceeds the counter limit of *300*. This merges with the user manual stop because when manually stopping, the counter limit (**ContolUnit.cycleCountLimit**) changes to zero – guaranteeing the counter is equal or higher than the limit.

```
cycleCount++;
if (cycleCount >= this.cycleCountLimit){ this.displayStatus(UICatagory.status,["User has manually stopped the LMC assembly program or it has reached the timeout limit (300 cycles)."]); break; }
```

Figure 11.13—33 - part of the timeout counter functionality of the simulator - both incrementor and counter checker.

```
534     public timeout(){ this.cycleCountLimit = 0; }
535     //^ Infinite loop counter.
536     //^ Makes cycle count limit lower (or same) as the current cycle count which causes the execution to stop.
```

Figure 11.13—34 - part of the timeout counter functionality of the simulator – the manual stop part.

For the user’s curiosity, the author decided to change **ContolUnit.cycleCountLimit** to *0* instead of *cycleCount* to *300* because it is much easier to change the timeout limit (like changing it to 500 instead of 300) if the author or other developer wants to change it – only need to change one number in the code.

11.14 Appendix N: Agile Development: Timebox 3

11.14.1 Third sprint

11.14.1.1 Planning

11.14.1.1.1 Prior research

Two of the main obstacles for this sprint are the level checking mechanics and cross-file data communication.

11.14.1.1.1.1 Sound effects

The author needs a good sound effect that helps with both HCI and Humanism. To this, the author has found one at Pixabay (Epic_Stock_Media, 2017). Related legalities are covered in the **Legal Considerations Chapter** of the main part of the dissertation.

11.14.1.1.1.2 Level data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Level	Index	Type	What program does? - more info	Objective explaination	Objective example	Teaches / based on	Copied?	3-star max	2-star max	Example of a solution	Cases	Partial/incorrect	whats wrong?
2	1	0	Introduction	relay - input then output	One of the core concepts	Inputting "8" to get "8"	INP,OUT,HLT	N/A	N/A	N/A	input INP, output OUT,hal	N/A	N/A	N/A
3	2	1	Introduction	adder calculator - store input ther	The other core concept is	Inputting "2" and "3" t	STA,ADD,1	N/A	N/A	N/A	19, INP, ADD 19, OUT, HLT	N/A	N/A	N/A
4	3	2	Introduction	withdrawal - load then subtract	There are many ways to p	Inputting "2" and "3" t	LDA,SUB,1,2	N/A	N/A	N/A	INP, STA 19, INP, STA 20, L	N/A	N/A	N/A
5	4	3	Introduction	hundred rounding - Shift right twic	Choosing method proces	Inputting "123" to get "	SHR,SHL,1	N/A	N/A	N/A	INP, SHR, SHR, SHL, SHL, C	N/A	N/A	N/A
6	5	4	Introduction	character set - number to ASCII co	Computers do not unders	Inputting "121" to get "	OTC,1	N/A	N/A	N/A	INP, toChar OTC, HLT	N/A	N/A	N/A
7	6	5	Introduction	varaibles - add 10 to input	Not everything needs to b	Inputting "2" to get "12	DAT,1,2	N/A	N/A	N/A	INP, ADD ten, OUT, HLR, te	N/A	N/A	N/A
8	7	6	Introduction	selection (if-statements) - branch	Programs executes sequ	Inputting "0" to get "0"	BRP,BRZ,1	N/A	N/A	N/A	INP, BRP positive, BRZ zer	N/A	N/A	N/A
9	8	7	Introduction	iteration - using counter to exit inf	Branching also allows ite	Inputting "1" to get "1"	BRA,1,3,4,6	N/A	N/A	N/A	INP, loop OUT, BRP end, E	N/A	N/A	N/A
10	9	8	Correcting	comparison - which input is high	This program compares	Inputting "2" and "3" t	1,2,3,6,7,8	2.2	13	18	INP, STA num1, INP, STA r	[2,3;3],	INP, STA num1, IN	swapped "LDA
11	10	9	Big formula	linear graph equation - y=2x+3	Make a program reflect th	Inputting "4" to get "11	1,2,6	No	8	13	INP, STA input, ADD input	[4;11],[	N/A	N/A
12	11	10	Partial	even - is positive number even?	Knowing if a positive inte	Inputting "8" to get "y"	1,3,5,6,7	No	13	18	INP, BRZ isEven, loop SUB	[8;y],[0	INP, BRZ isEven, L	N/A
13	12	11	Correcting	positive multiplication - calculato	Multiplication is fundem	Inputting "2" and "3" t	1,2,3,6,7	2.4	20	25	INP, STA num1, INP, STA r	[2,3;6],	INP, STA num1, IN	"SUB num1" sh
14	13	12	Contexual	unit converter - add 273 for C to K	Scientists from different	Inputting "25" to get "2	1,2,5,7	No	5	10	INP, ADD const, OUT, HLT	[25;298	N/A	N/A
15	14	13	Contexual	squaring - input ^ 2	A simple calculator has l	Inputting "3" to get "9"	1,2,3,6,7,8	4.3	22	27	start INP, STA number, LD	[3;9],[2	N/A	N/A
16	15	14	Partial	overflow - addition over the limit?	This calculators add 2 pa	Inputting "999" and "1	1,2,3,5,6,7	No	12	17	INP, STA num, INP, ADD n	[999,1;	INP, STA num, IN	N/A
17	16	15	Contexual	digit significance - least significa	Depending on the situati	Inputting "123" to get "	1,4	No	7	12	INP, SHL, SHL, SHR, SHR, C	[123;3]	N/A	N/A
18	17	16	Contexual	floor division - calculator	Another key calculator op	Inputting "25" and "8"	1,2,3,6,7,8	4.4	23	28	start INP, STA dividend, IN	[25,8;3	N/A	N/A
19	18	17	Partial	FizzBuzz - classic programming	FizzBuzz classic program	Inputting "15" to get "1	1,2,3,5,6,7,8	No	65	70	INP, STO input, LDA input	[15;F,I,	INP, STO input, LD	N/A
20	19	18	Correcting	bank balance - adding/subtractin	This program acts as a b	Inputting "100" for ba	1,2,3,5,6,7	No	24	29	INP, STO total, loop OUT,	[100,-9	INP, STO total, loo	"OUT" in wrong
21	20	19	Big formula	fibonacci sequence - 1st to 17th te	Fibonacci is a number se	Outputting (without in	1,2,3,6,7,8	No	18	23	LDA first, OUT, LDA secon	[;0, 1, 1	N/A	N/A
22	21	20	Partial											N/A

Figure 11.14—1 - Screenshot snippet of the level content/data plan (using Microsoft Excel).

Creating levels was much more difficult for the author than expected due to the following requirements for level creation:

- Level topic must be with the student's learning in mind – if it does not, then it is not fit.
- Each level must have an example solution which is easy to understand and tested to ensure its correctness.
- Each level must comply with the nature of the level type (introduction/tutorial, correcting, partial, big formula, and contextual).
- Each consecutive level must get slightly harder than the previous level.
- Make sure that each level's knowledge and skills requirements are the knowledge and skills of any of the levels before it.
- Balance the levels' challenge to be both learning and approachable instead of only one of them.

There are other considerations, but those are the main ones. For this reason, the author has decided to do 20 levels instead of 30 levels to save time. When the author believes he will be able to complete the development and testing of sprint #3 earlier than expected, then the author will plan, develop and implement levels 21-30.

Must have	Implementing levels 0-10
Should have	Implementing levels 0-20
Could have	Implementing levels 0-30
Would have	Implementing levels 0-40

Table 11.14—1 - MoSCoW table for implemented campaign levels

11.14.1.1.1.3 Cross-file data communication

There are two kinds of data, used across different HTML pages, that the LMC simulator game needs:

- Non-volatile immutable data – used for information of each campaign level, such as: level objective, example solutions, conditions for star counts of level, multiple sets of pre-defined inputs and their corresponding expected outputs to test the user's own script, et cetera. The magnitude of this immutable data is not small and is relatively massive compared to the mutable data.
- Volatile mutable data – used for applying the user's settings for the application as a whole (sound effects toggle, dark mode toggle, and selected theme). Very small data size.

The author first assumed that such a thing could be achieved with reading and editing from JSON files, for both kinds of data, as JS/TS supports JSON without the need for dependencies (such as node.js) and can store data as an object instead of just text like a

plain text file. However, after research, this does not appear to be fully the case. This is because the LMC educational game product is a client-only web application and not a server-dependent one. JS (compiled TS) in the browser environment cannot read local JSON files and can only read the JSONs sent over HTTP request. This local exclusion seems to be a purposely imposed restriction in the same-origin policy for the sake of protecting the client from malicious activity (mdr, 2025). More information regarding this can be read in the online documentation (mdr, 2025).

With this revelation, the author needs to think of other ways to do the job. non-volatile immutable data can simply be achieved with normal TS objects, as JSON is just a notation of JS objects (hence the full form of the JSON name being JavaScript Object Notation). The only problem is that this data cannot work across HTML, nor is it non-volatile.

There is one way the author can use to get over this, which is the use of URL queries in combination with the object instances. Level data is planned to be stored as one object per level – making the entire thing a list of objects. If the objects' index corresponds to their index (such as index 0 for level 1 and index 29 for level 30), then all you need is to send over a single number rather than an entire object. There are other ways to do this, such as cookies, but due to the seemingly far simpler option of using URL queries, the author decided to use that. Another reason for using URL queries is that they can also be used as volatile mutable data, as the settings can easily be stored as simple numbers. This makes the volatile mutable data also responsible for specifying what level the user selected for the simulator (if in campaign mode) to know and act accordingly – loading the level's data (from the mentioned immutable data). The author has read many sources regarding using URL queries (W3Schools, 2014b).

Please note that cookies are still part of the plan but are of low priority. If cookies are implemented, they will be for storing how many levels the user has completed and the star count of each completed level.

11.14.1.1.1.4 New UML and flowchart diagram editor

Due to the massive amount of time consumed moving and resizing shapes and arrows in the massive flowchart of sprint 2, the author has looked to looking for a text-to-flowchart maker tool.

From research, the author has found the following popular tools:

- Mermaid Live Editor
- PlantUML Online / PlantText
- WebGraphviz
- Code2Flow
- Flowchart.fun

PlantText and Code2Flow seem most suitable because syntax is more like actual programming languages (like TS), such as them using if-statements and while-loops and sequencing to link together instead of arrows ('->'). This familiarity gives them a far less steep learning curve – making it faster to use that instead of manually drawing the diagram. In the end, the author decided to choose PlantText as it supports both UML and flowcharts, unlike Code2Flow which supports only flowcharts.

The author decided to use the VS Code extension (jebbs, 2024) instead of the online version, because the VS Code extension is better displayed and it updates in real time with a much faster refresh rate. However, will still need to use the online version to export the flowchart because only the online website has the option to export as SVG instead of PNG.

Please note that some UML diagrams will still be done in draw.io if it is based on an existing draw.io UML diagram file.

It should also be stated that later on the author discovered that Code2Flow is not fully free by having a 25-node limit paywall.

11.14.1.1.1.5 Webhosting

As mentioned, due to being in sprint #3, the website needs to be hosted on a web server as the top priority of platforming when discussing porting platforms for the LMC educational game.

Initially, the author intended to use either the CCCU-provided web-hosting servers (Education Host, 2025) otherwise to use other free web-hosting servers such as Tiiny (Tiiny.host, 2019). However, the author realised that GitHub had a feature to allow developers to turn code repositories (in which the LMC product was made) into a web server.

Armed with the knowledge, the author decided to use GitHub to host the LMC web application for one major reason: prototyping. By using a GitHub repository as the website, the author does not need to spend any time exporting and organising and separating the hosted web-technology files (HTML, CSS, JS, media assets et cetera) from the source files (TS files, configuration files, wireframes and test files, et cetera). With one of the main time-consuming processes avoided, the author was able to find the time to do other things, such as fixing the real-time user input problem from sprint #2.

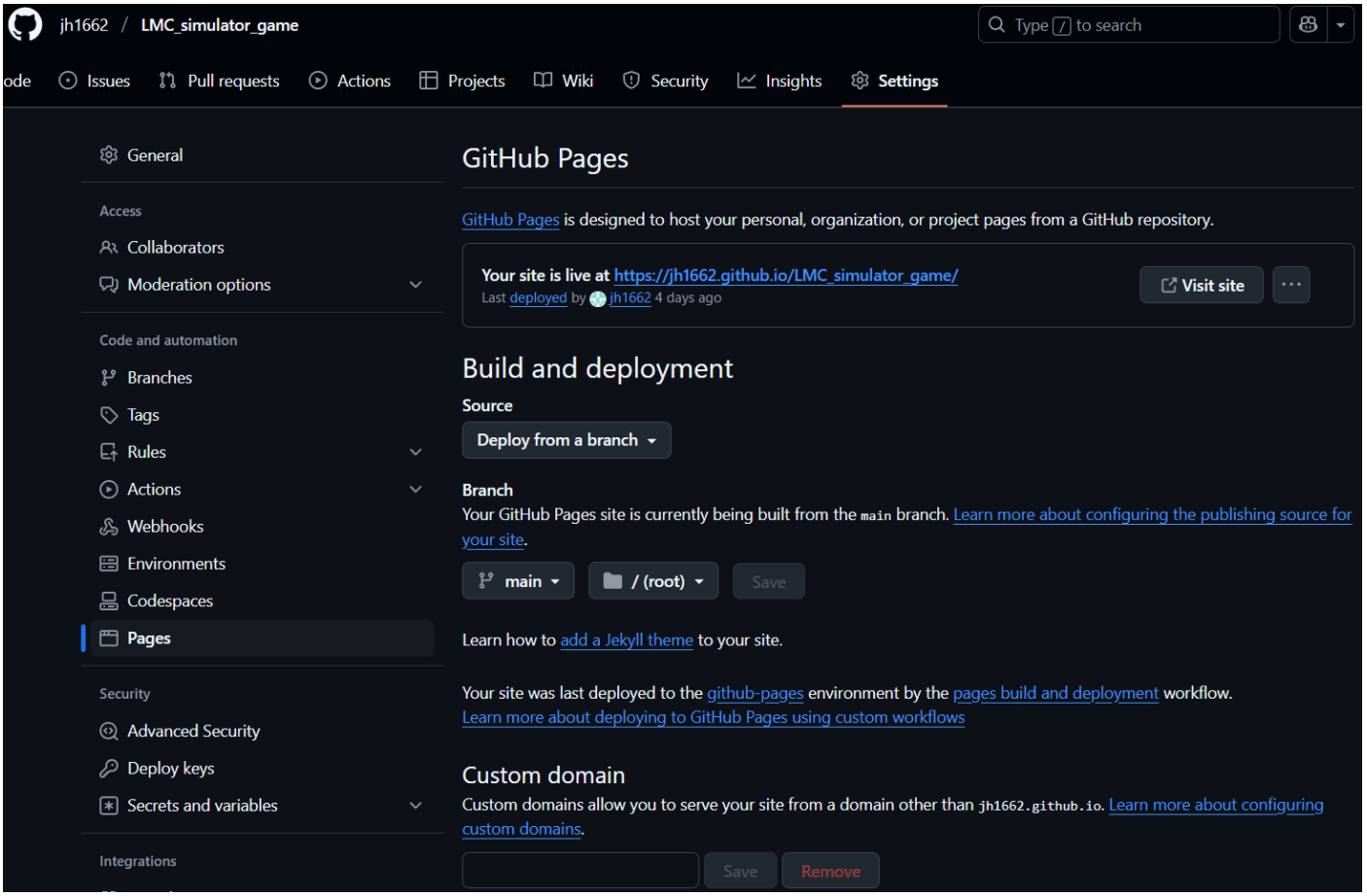


Figure 11.14—2 - configuration regarding setting up a web server for the LMC simulator game's source repository

11.14.1.1.1.6 HCI Principles

Sprint #2 was more of getting the baseline of the UI working, sprint #3 is to make the UI both functional and adhere to the HCI principles. The Web Content Accessibility Guidelines (WCAG) was used as a guideline for this (Caldwell *et al.,* 2008), such as: large text, controls and inputs, “time-based media”, decoration, formatting, et cetera (Caldwell *et al.,* 2008).

11.14.1.1.1.6.1 Theory

HCI principles will be incorporated into the third sprint to further stratify the educational theories. HCI makes the usage of programs more usable, accessible, efficient, and engaging.

For a more detailed explanation:

- Usable – allow the user to easily understand and use the program without struggles nor feeling uncomfortable.
- Accessible – make the program usable to anyone, despite any difficulty and/or disability.
- Efficient – as much of the user’s cognitive work should be done by the program instead of saving time and effort.
- Engaging – the user must always be doing something or experiencing stimuli.

Incorporating HCI is used as a practical method for satisfying educational theories, as its end goal is intertwined with the educational theories. This is especially with Humanism and Behaviourism as both Humanism and HCI principles want programs to be as human-friendly and accessible as possible, while Behaviourism and HCI principles focus on constantly receiving user input or giving stimuli to the user.

Due to the striving to satisfy the educational theories in sprint #2. The LMC educational game is already mostly HCI compliant.

List of incorporated HCI factors:

- Usable – use of the manual to teach the user how to effectively use the program. Also, buttons change colours both when hovered over and when clicked; this responsiveness helps prevent confusion in the user's thinking if they actually clicked

the button or not (one can argue that responsiveness is more related to the efficiency and accessibility factors of HCI rather than the usability factor).
- Accessible – use of both dark/light mode and different themes to cater to different visual difficulties.
- Efficient – uses of displaying statuses of how the assembly script, executes, notifying of user actions, explaining why a running or compilation error happens, et cetera.
- Engaging – When using the simulator (in sandbox or campaign mode), the user is engaged in either building the script or seeing how the simulator simulates the compiled script (change in memory and registers, Little Man action in display box, status, et cetera).

The author believes that for HCI to be effective is to use it in multiple directions. Another main direction to take HCI into is audio HCI. This was chosen because audio-based HCI is one of the 2 HCI types that can be achieved on a webpage in the typical computer without any unusual hardware (such as VR headsets, vibrating feedback, et cetera). Use of audio-based HCI also assists with both a Use of audio assistance helps with people who have more critical visual difficulties (satisfying Humanism) and emphasises the responsiveness of the program to the user.

Must Have	An audio (like a beep) plays every time a button is pressed.
Should Have	The 'must have' and the ability to toggle it on or off for accordance to the user's preference, such as if the user prefers quiet.
Could Have	The 'could have' but have different sound effects for the simulator's workings.
Would Have	A text-to-speech player that reads out every button and newly displayed statuses/text to help visually impaired users.

Table 11.14—2 - MoSCoW table for the sound effects of sprint #3.

11.14.1.1.1.6.2 Method of implementation

Audio files can be imported into the webpage using the HTML 'source' tag (W3Schools, 2014a). This tag can then be fetched and stored as a `HTMLAudioElement` object and can simply be played using the `.play()` method call (W3Schools, 2024).

11.14.1.1.2 Design

11.14.1.1.2.1 UML

11.14.1.1.2.1.1 Diagram

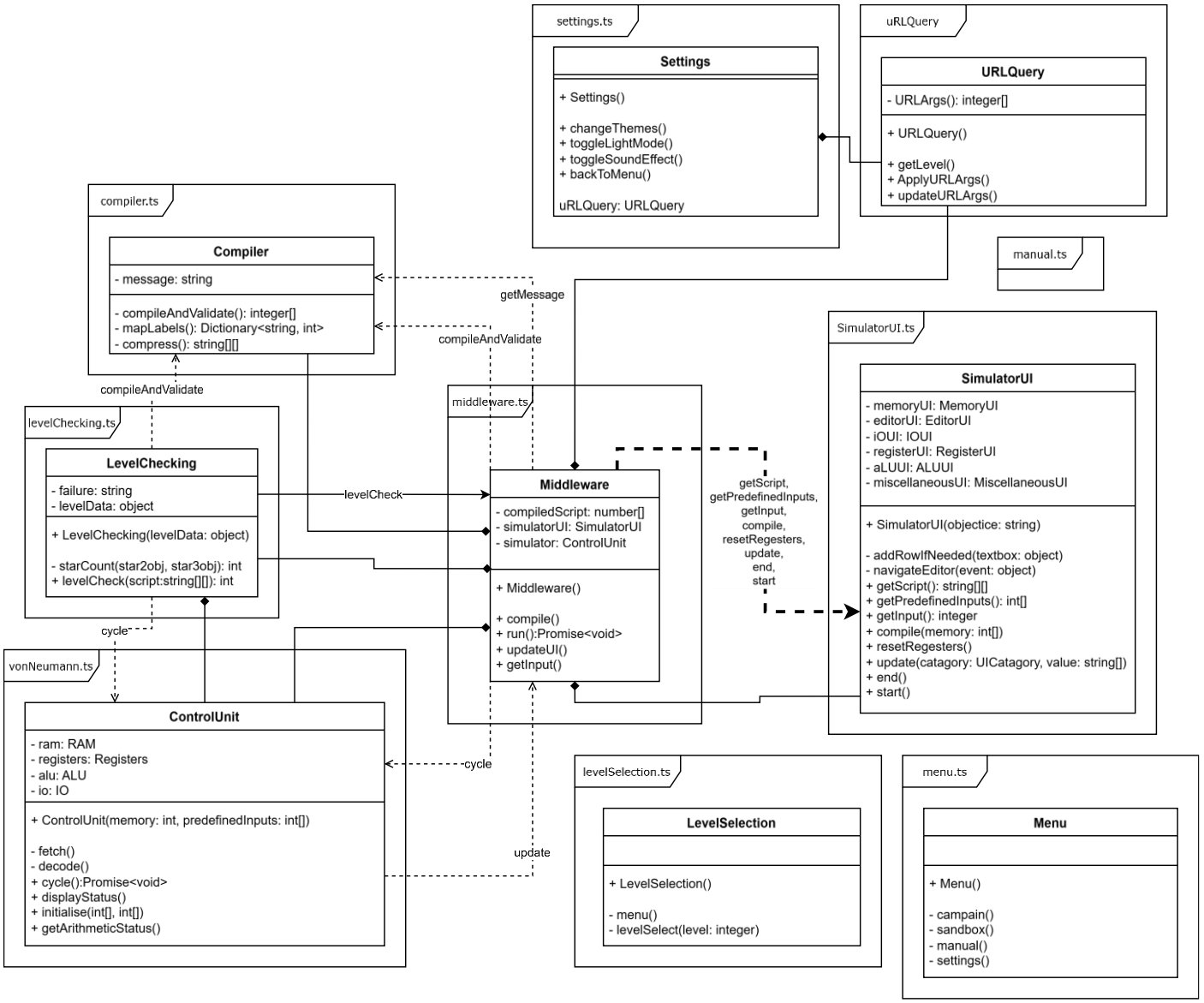


Figure 11.14—3 - UML diagram (SVG format) for the LMC educational game in sprint #3.

11.14.1.1.2.1.2 Abnormal traits

As the LMC product grows more complex, from a simulator page with a simple manual and placeholder menu to a full-blown simulator game it causes some plans, such as the sprint #3 UML, to have some unusual traits.

Those traits of **Figure 11.14—3** are explained and justified:

- Isolated TS classes – classes (such as `Menu`) are isolated because their sole purpose is to set up the pages such as adding event listeners to HTML buttons. Despite all pages having at least two instantiated classes (such as `Menu` and `URLQuery` for the menu page), those classes sometimes have no relation between each other when a relation is not necessary. This is enforced by the OOP programming practice of keeping the different modules of the program as isolated as ideally possible for better organisation and reusability.
- Empty `manual.ts` – To explicitly state that the manual page has no dedicated code (considering `URLQuery` to be generally used code); this is simply because the manual has no HCI besides simply reading and scrolling via the browser page (no buttons or other HTML input elements to add event listeners).
- Lack of fields in some classes – Lack of fields is in the dedicated classes due to their simplistic purposes - mostly just adding event listeners to HTML elements and URL referrals.
- No level data – The all-level data is stored in an exported constant object 1-dimensional list; this makes it not a class, thus not displayed in the UML. Use of objects for level data storage was discussed in the **Cross-file data communication** section of the sprint #3 **Prior research**. The only reference to the all-level data is the `LevelData` object in the `LevelChecker`, which stores one level data object corresponding to which specific level that the user chose to do. It would be ideal if the author were able to use a custom data type for the level data sets – of slightly low priority.

11.14.1.1.2.2 Flowchart

Due to the big difference in the height-to-width ratio, it must be split into 3 separate diagrams.

11.14.1.1.2.2.1 Main routine

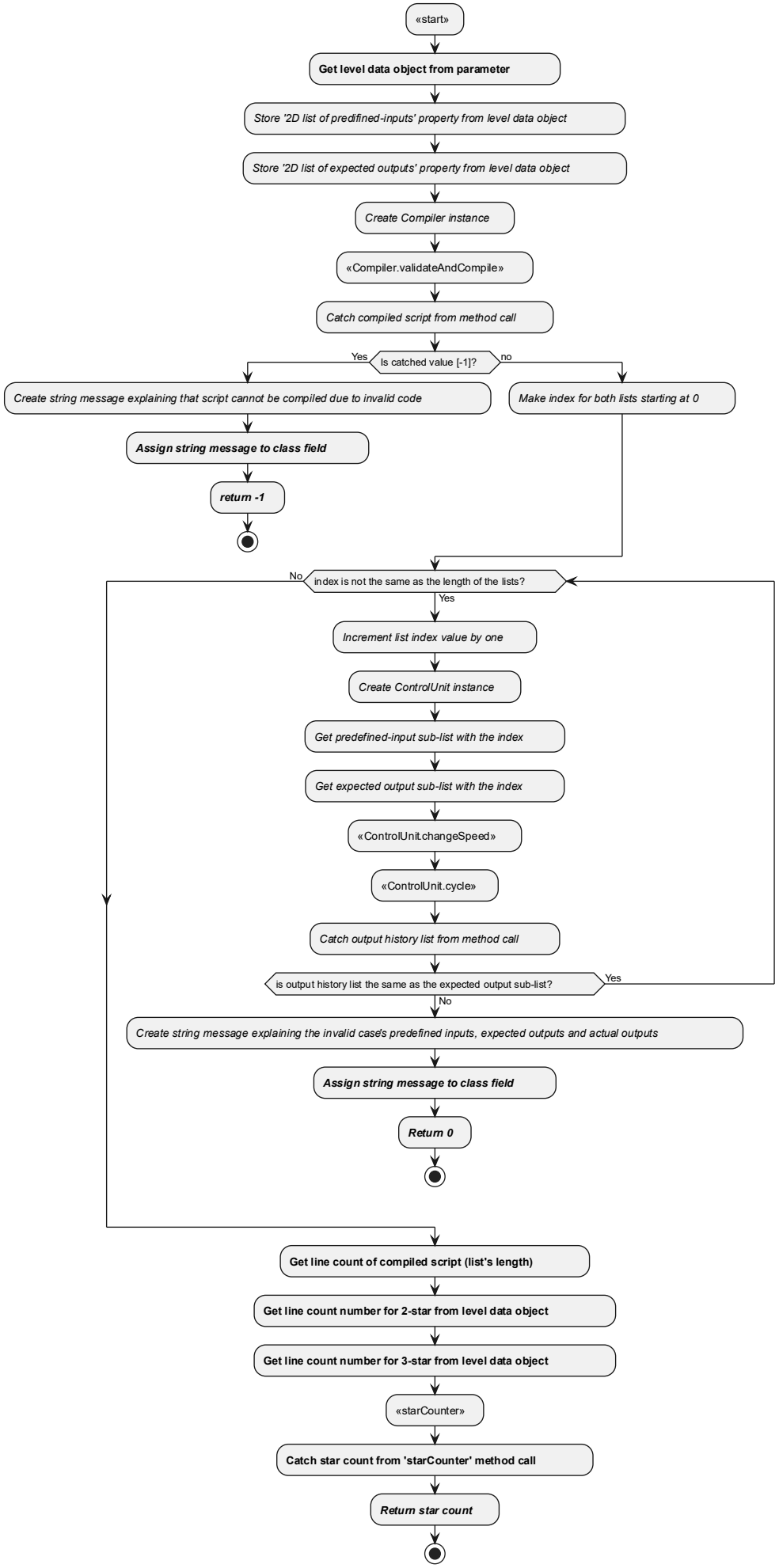


Figure 11.14—4 - first third of the sprint #3 flowchart (SVG format) – showing the `levelChecker.testCases` *public method plan.*

11.14.1.1.2.2.2 Star counter routine

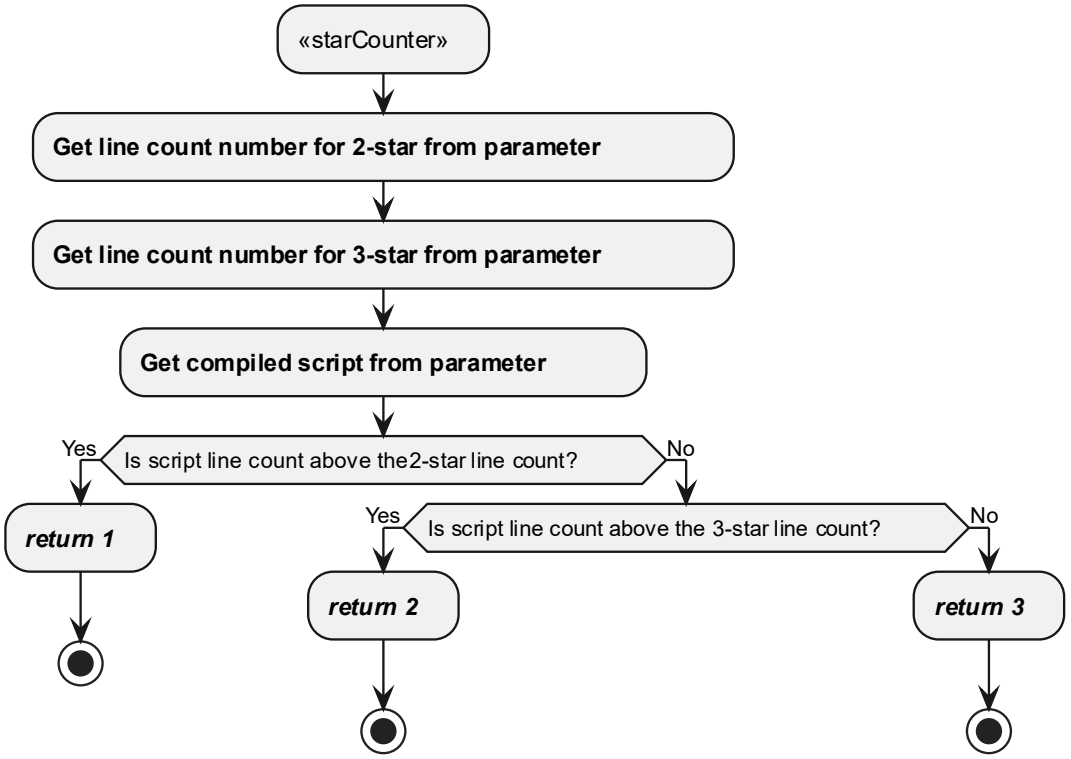


Figure 11.14—5 - second third of the sprint #3 flowchart (SVG format) – showing the `levelChecker.starCounter` *private method plan (called in* `levelChecker.testCases`*).*

11.14.1.1.2.2.3 External calls

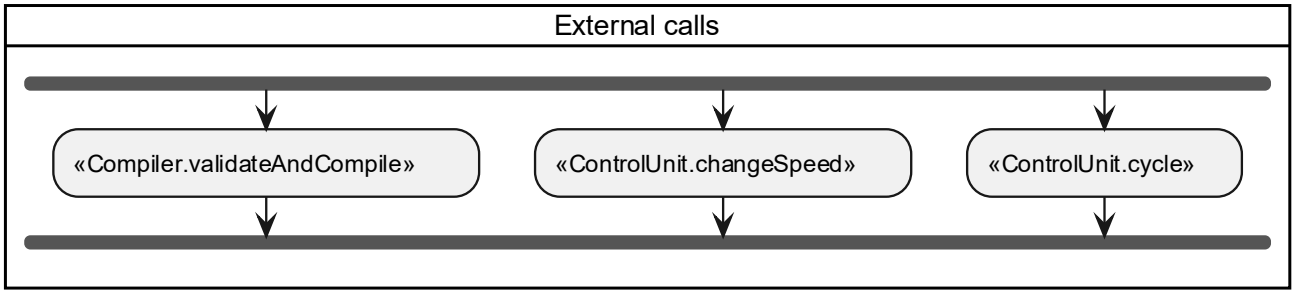


Figure 11.14—6 - last third of the sprint #3 flowchart (SVG format) – showing the external calls in `levelChecker.starCounter` *public method plan.*

11.14.1.1.2.2.4 domain-specific script

Below is the script used to generate the sprint #3 flowchart.

```
@startuml Level Checker
'* There is no point for closing tags (like </b>) because opening tags only apply thier
own line and lack of closing tag does not give error/warnings.
/'
> has << and >> - start of routine or sub-routine call.
> has <b> (is bold) - IO (including from parameters).
> Has <i> (is italic) - process.
> In hexagon shape - if-statements.
> Has <i><b> (is both bold and italic) - return statement or class field assaignment.
> Circle - termination
'/
:<<starCounter>>;
    ': set-up
    :<b>Get line count number for 2-star from parameter;
    :<b>Get line count number for 3-star from parameter;
```

```
    :<b>Get compiled script from parameter;

    ': determining star count (less code = higher star count)
    if (Is script line count above the 2-star line count?) then (Yes)
        :<i><b>return 1;
        stop
    else (No)
        if (Is script line count above the 3-star line count?) then (Yes)
            :<i><b>return 2;
            stop
        else (No)
            :<i><b>return 3;
            stop
        endif
    endif

:<<start>>;
    ': set-up
    :<b>Get level data object from parameter;
    :<i>Store '2D list of predifined-inputs' property from level data object;
    :<i>Store '2D list of expected outputs' property from level data object;
    :<i>Create Compiler instance;

    ': compile for script execution and for 'starCounter' if it gets to it without fail
    :<<Compiler.validateAndCompile>>;
    :<i>Catch compiled script from method call;
    if (Is catched value [-1]?) then (Yes)
        :<i>Create string message explaining that script cannot be compiled due to invalid
code;
        :<i><b>Assign string message to class field;
        :<i><b>return -1;
        stop
    else (no)
        :<i>Make index for both lists starting at 0;
    endif

    while (index is not the same as the length of the lists?) is (Yes)
        '* iterate via each case of testing if compiled script takes in inputs and give
expected outputs
        :<i>Increment list index value by one;
        :<i>Create ControlUnit instance;
        :<i>Get predefined-input sub-list with the index;
        :<i>Get expected output sub-list with the index;
        :<<ControlUnit.changeSpeed>>;
        :<<ControlUnit.cycle>>;
        :<i>Catch output history list from method call;
        if (is output history list the same as the expected output sub-list?) then (No)
            '* outputs of case does match case's expected output meaning user's script
does not satisfies level's criteria
            :<i>Create string message explaining the invalid case's predefined inputs,
expected outputs and actual outputs;
            :<i><b>Assign string message to class field;
            :<i><b>Return 0;
            stop
```

```
        else (Yes)
            endif
    endwhile (No)

    ': user's script satisfies level criteria and now to see how well it does so by using
the 3-star system
    :<b>Get line count of compiled script (list's length);
    :<b>Get line count number for 2-star from level data object;
    :<b>Get line count number for 3-star from level data object;
    :<<starCounter>>;
    :<b>Catch star count from 'starCounter' method call;
    :<i><b>Return star count;
    stop

card External calls{
    '* Noting external calls from different files.
    fork
        :<<Compiler.validateAndCompile>>;
    fork again
        :<<ControlUnit.changeSpeed>>;
    fork again
        :<<ControlUnit.cycle>>;
    end fork
}
@enduml
```

11.14.1.1.2.3 Wireframe mock-up

Due to many variations made in sprint 2 (Appendix M), the “all-sprint” wireframe (originally done before sprint 1) mock-ups will be redone here. This ensures that everything is more accurate and relevant, but mainly for that same reason from sprint 2 – for more detailed wireframes.

11.14.1.1.2.3.1 Low-fi

11.14.1.1.2.3.1.1 Menu

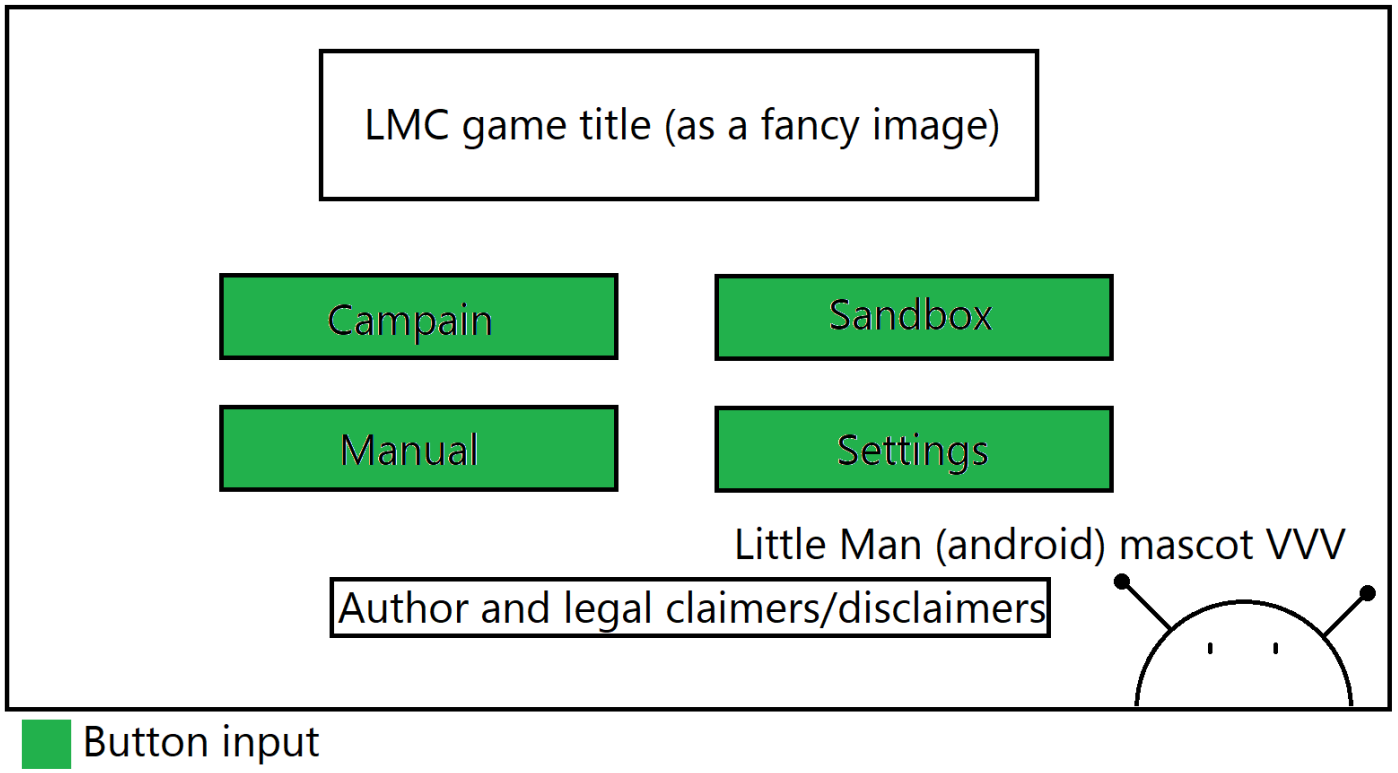


Figure 11.14—7 - a colour-coded low-fi wireframe, for sprint #3, of the menu page.

11.14.1.1.2.3.1.2 Settings

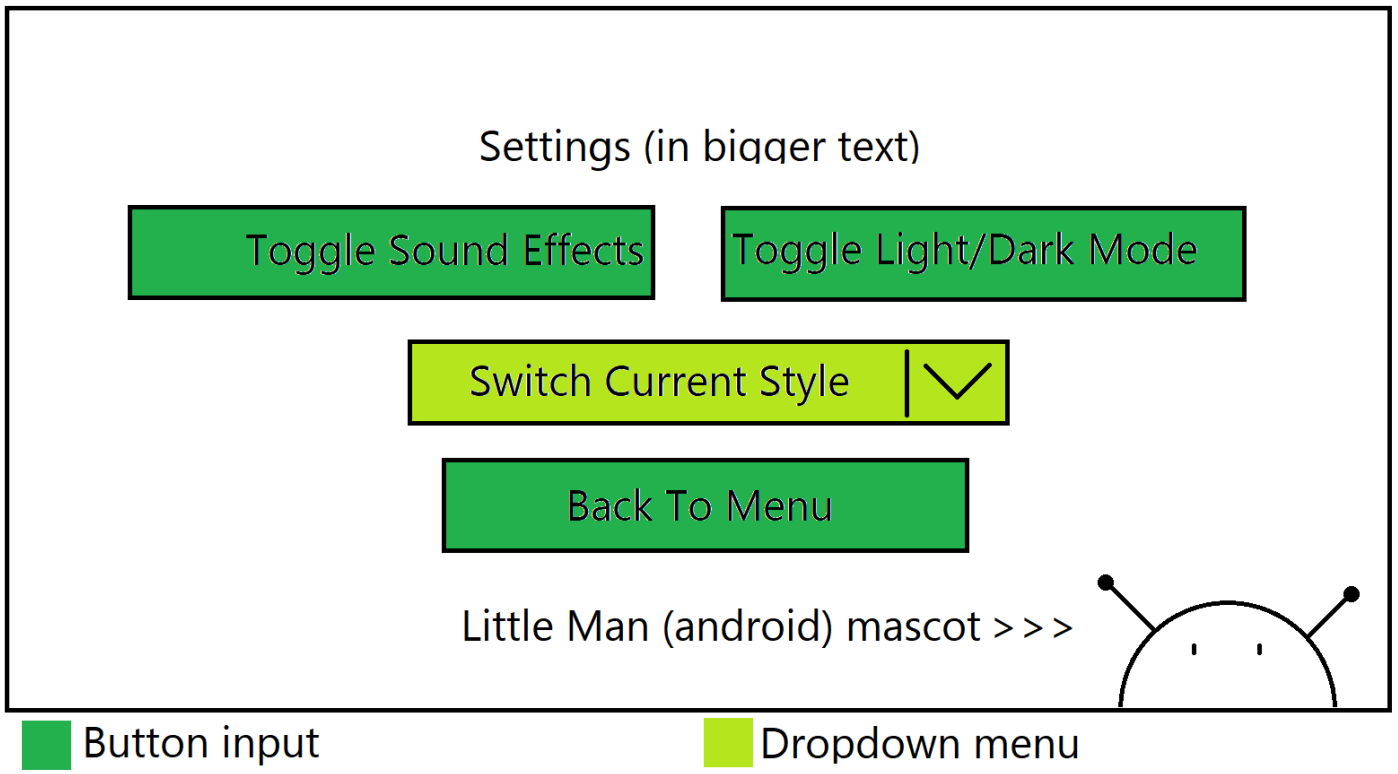
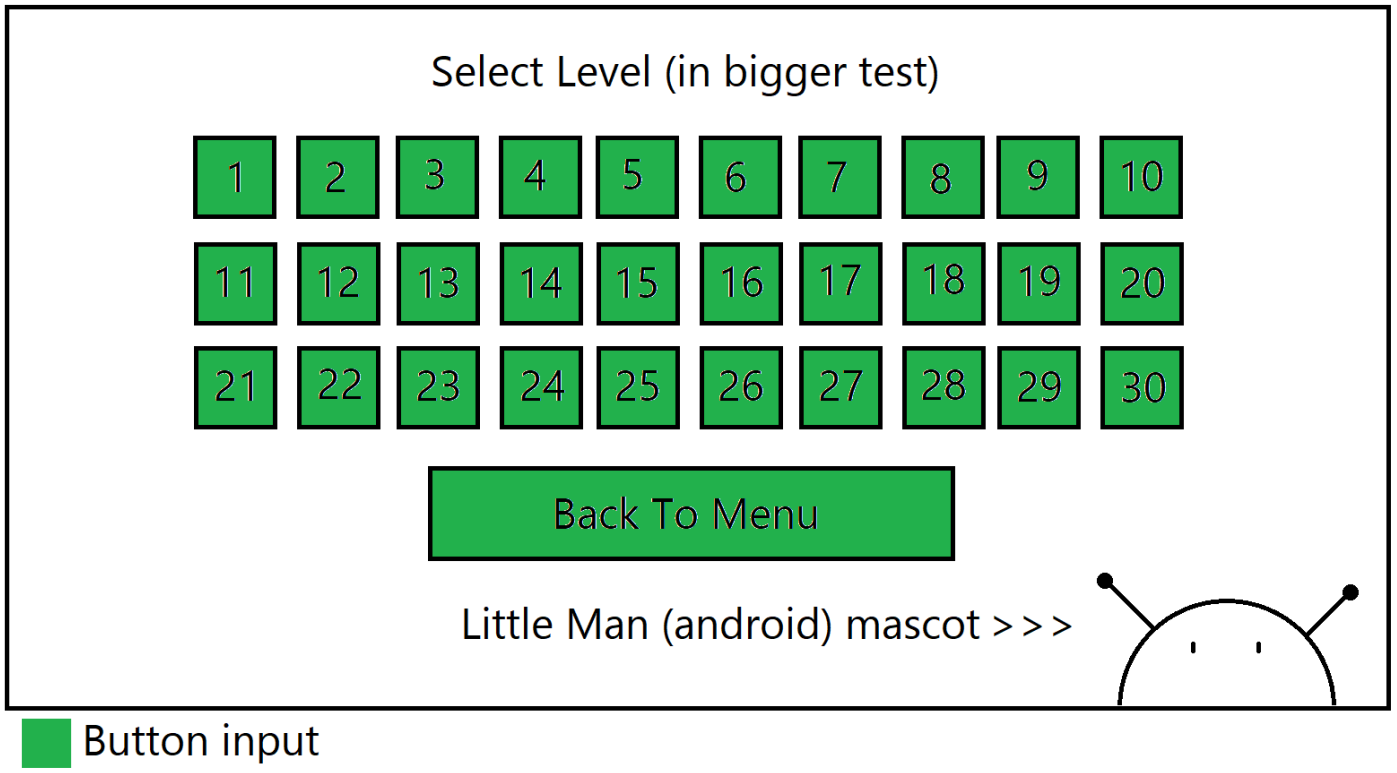


Figure 11.14—8 - a colour-coded low-fi wireframe, for sprint #3, of the settings page.

11.14.1.1.2.3.1.3 Level Selection



Button input

Figure 11.14—9 - a colour-coded low-fi wireframe, for sprint #3, of the level selection page.

11.14.1.1.2.3.1.4 Simulator

The wireframe of the simulator page is already done in sprint 2 critical path analysis (in Appendix M), and since no changes are planned for the simulator page's frontend, a sprint 3 version was not needed. This was initially the case until the author noticed some improvements.

The author did think maybe changing the order of display buttons would save vertical space, but when testing that theory, it surprisingly does not seem to make any noticeable difference – same vertical space even at different zoom levels.

Only differences that will applied is making used of capitalisation is consistent across the UI (such as all buttons with have capitalisation at either the first letter of the first word, first letter of every word, no capitalisation, or all letters to be capitalised) and make sure the RAM cell labels are 0-99 instead of 0-9 ten times which the author failed to notice until that point in time of creating this planning.

11.14.1.1.2.3.1.5 Manual Page

The manual page has been partially done in sprint 2, but this is so without a sprint 2 plan because it was a placeholder for the third sprint and its core information assists with frontend testing. Because of this, the wireframe for the manual is done here instead of sprint 2, as seen below.

Instruction Manual (title)

Quick explaination of the order of compiling to memory then running fetch-decode-esecute cycles of the compiled script.

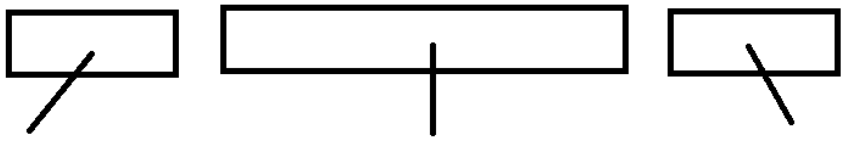
Name	Mnemonic	Code	Description
Instruction set table			

List describing navigation keys of the script editor.

Name (+ menonic)	Description	Range
Table of of the 4 registers.		

List discribing the functionality of each button relevant to the simulator (run, compile, reset, etc).

Screen snipped of the ALU simulator conpoment:



Explainations and examples of each part.

Author and legal claimers/disclaimers

Little Man (android) mascot VVV

Figure 11.14—10 - a low-fi wireframe, for sprint #3, of the manual page.

The contents should be descriptive enough to help an initially clueless user to get the hang of the web application, but balanced by being compact enough not to deter the user away by being too long to be worth reading. This balance is important because the main audience being some from Generation Z and most from Generation Alpha, who tends to be inclined to engage with conscience abstract and bullet-point like summaries – much more likely to read and comprehend than long lengthy paragraphs of text*. Taking these things into consideration is part of satisfying Humanism, which is previously mentioned as one of the 4 primary learning theories for achieving the LMC application goal of being a highly effective learning tool (about Von Neumann CPU architecture and assembly programming) for the users.

By use of multiple lists, tables, and a diagram in an alternating order will help achieve a concisely structured, compact manner and engagement.

11.14.1.1.2.3.2 High-fi

11.14.1.1.2.3.2.1 Menu

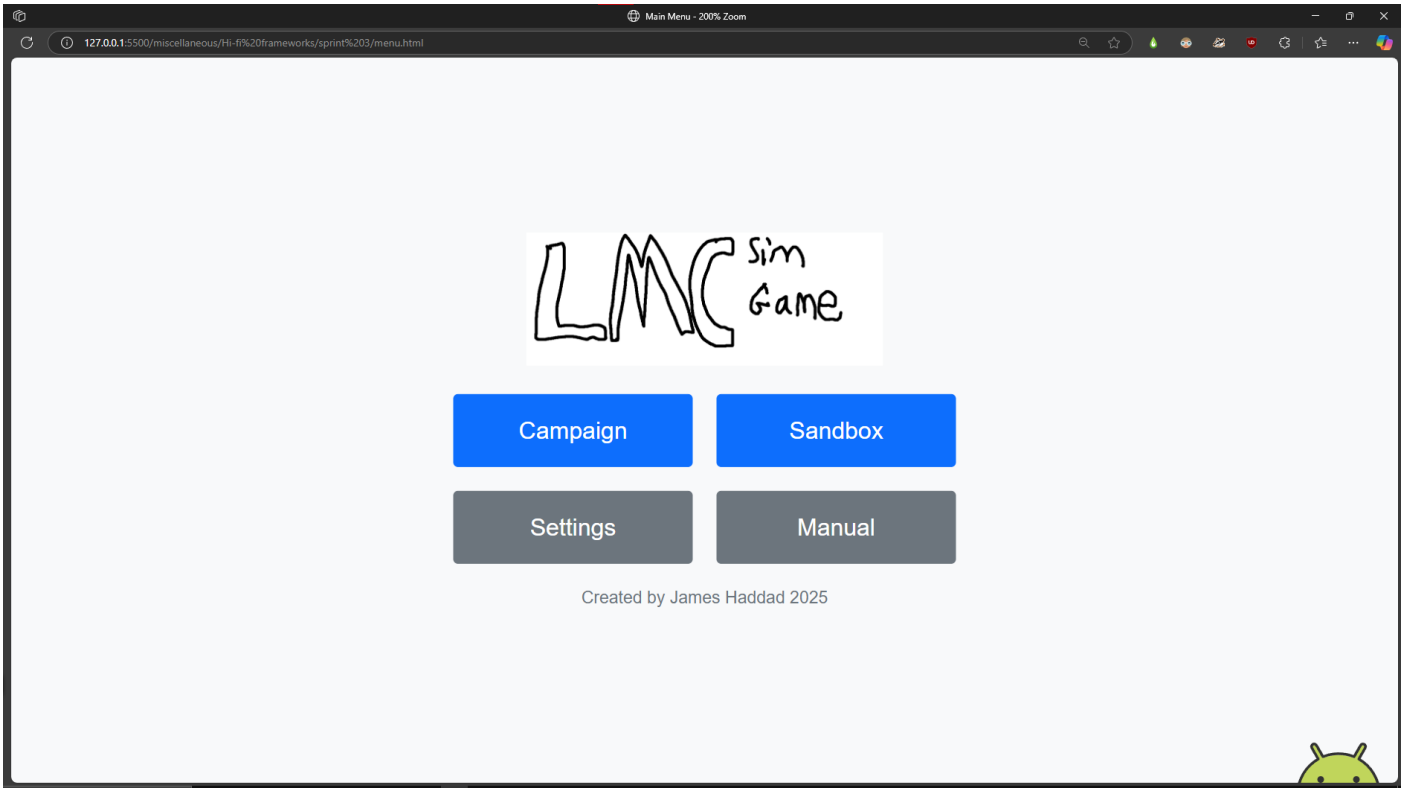


Figure 11.14—11 - a high-fi wireframe, for sprint #3, of the menu page.

11.14.1.1.2.3.2.2 Settings

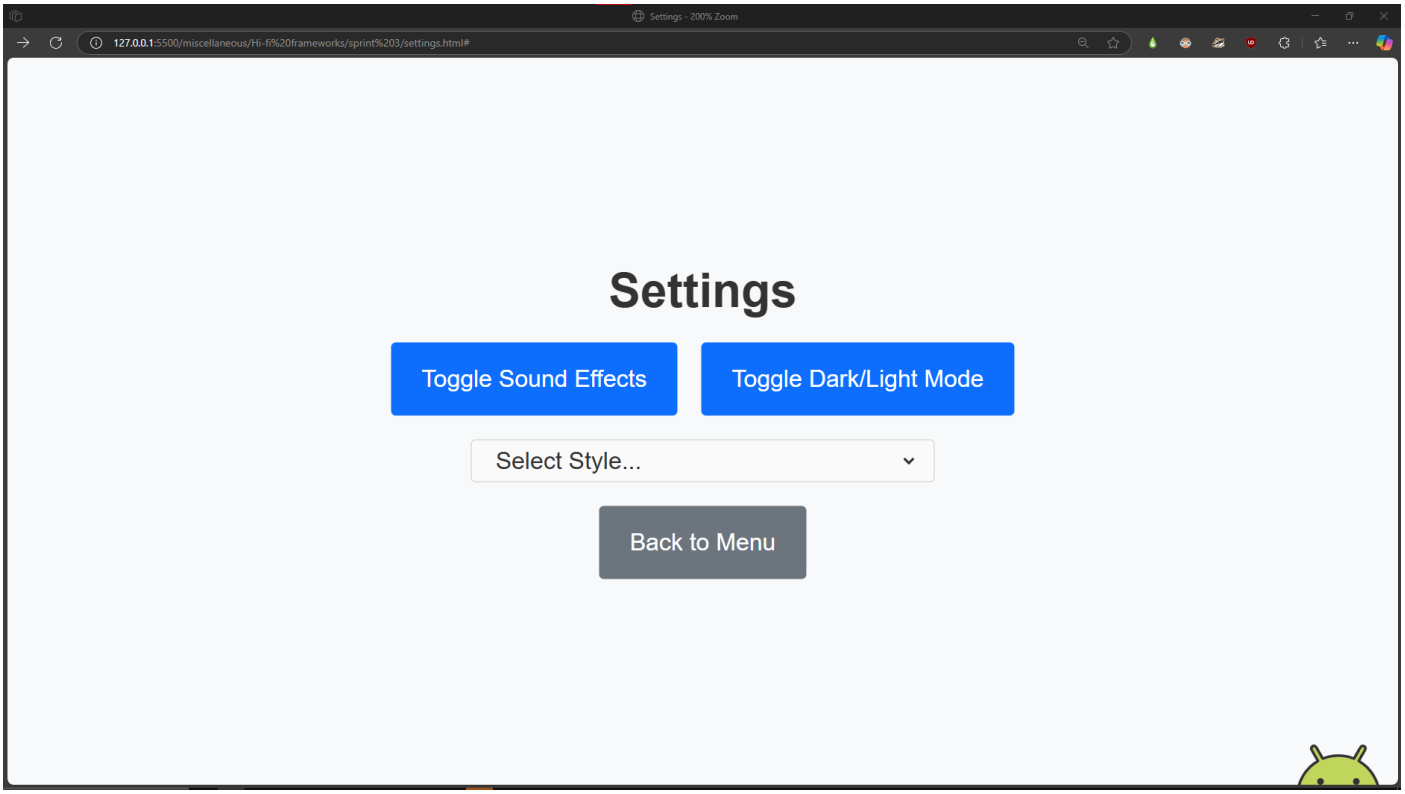


Figure 11.14—12 - a high-fi wireframe, for sprint #3, of the settings page.

11.14.1.1.2.3.2.3 Level selector

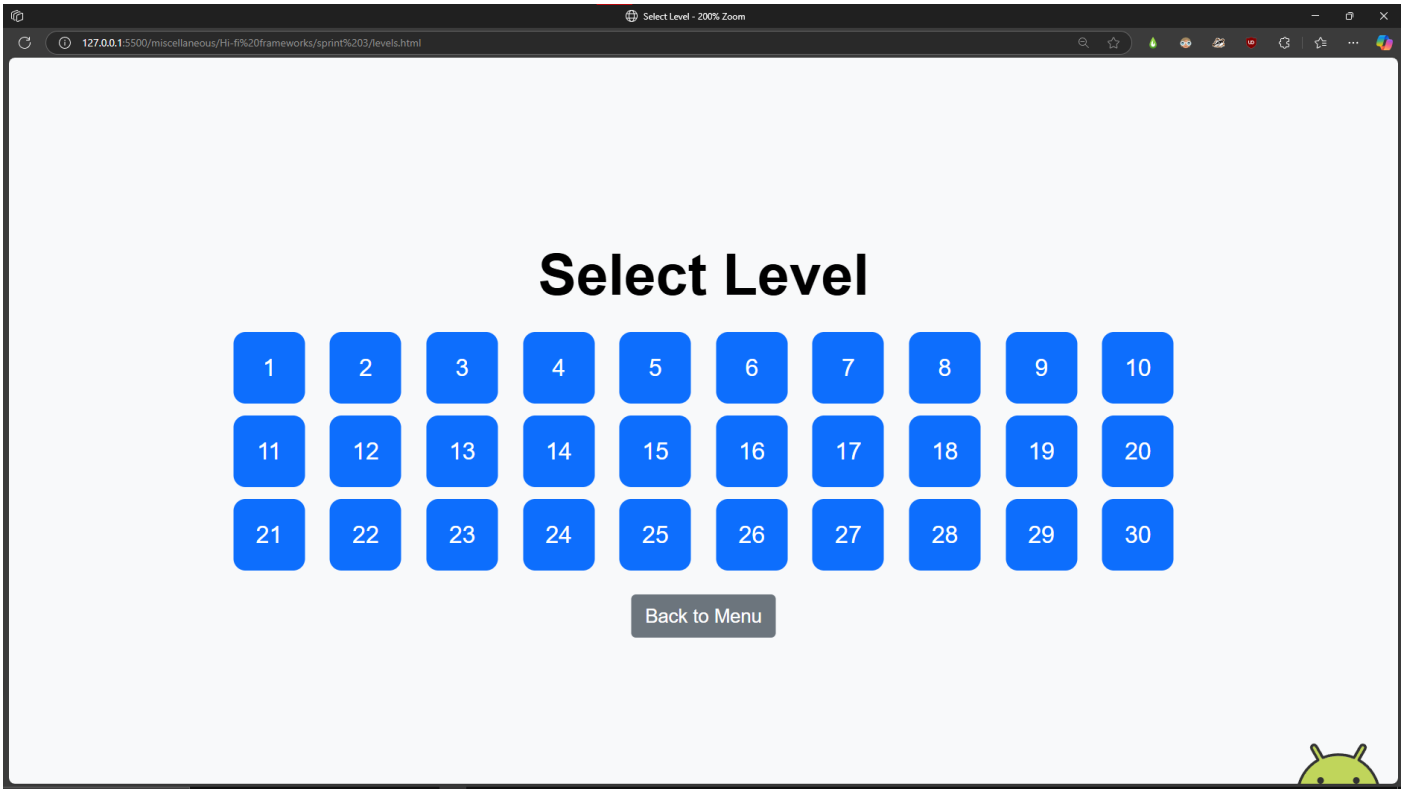


Figure 11.14—13 - a high-fi wireframe, for sprint #3, of the level selector page.

11.14.1.1.2.3.2.4 Simulator

No hi-fi wireframe mock-up for the simulator for the same reason as not showing the low-fi version of it.

11.14.1.1.2.3.2.5 Manual page

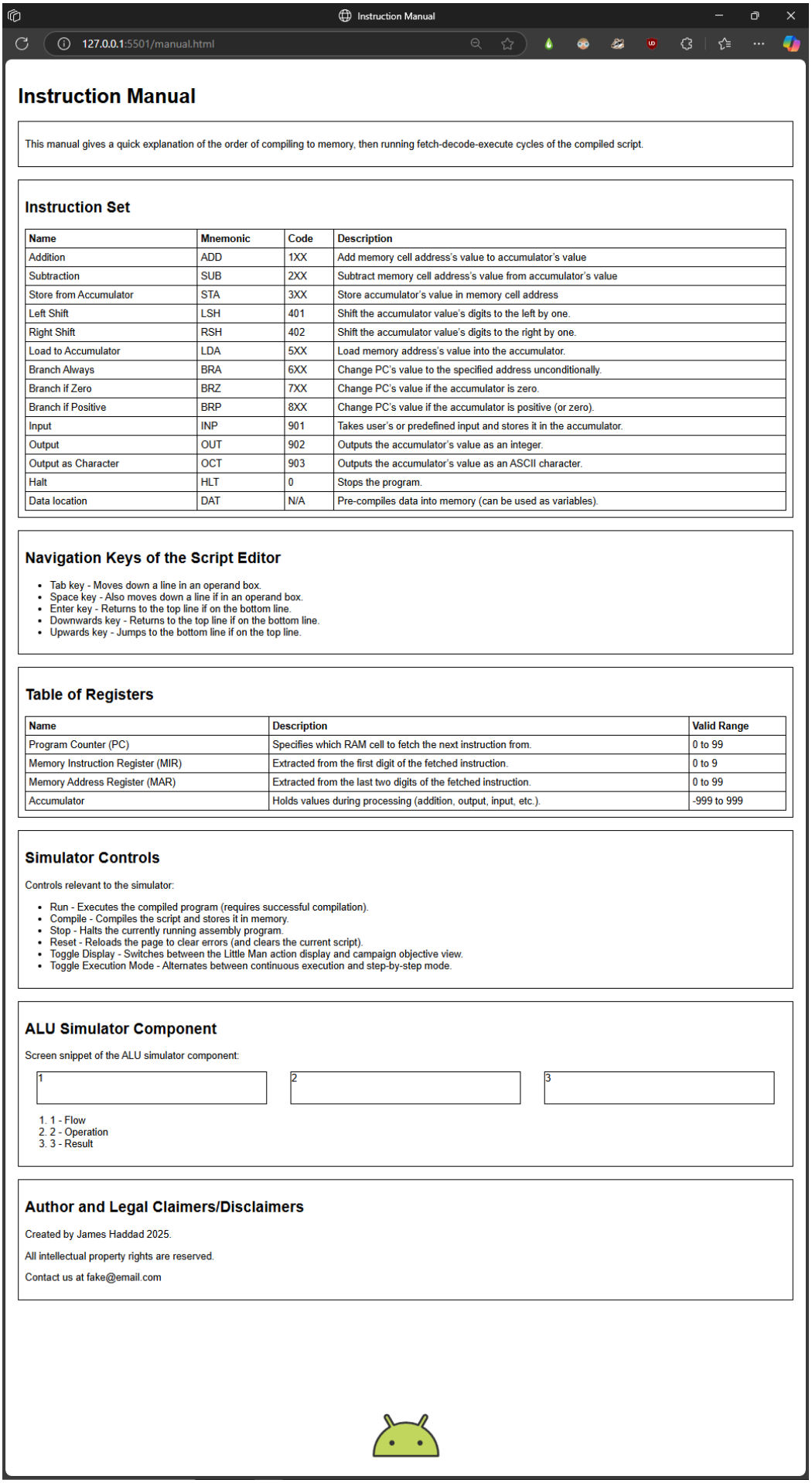


Figure 11.14—14 - a high-fi wireframe, for sprint #3, of the manual page.

As the reader can see, there are not any colours besides the Little Man mascot as there are no interactable elements (like buttons). This means that there is no way to navigate to other pages from there. This is purposely done because the manual was made to be opened as either a new tab or a new browser window. Opening the manual page this way allows the user.

Also, the mascot appears to be floating above the bottom in the hi-fi wireframe; This will not be the case in the actual implementation.

11.14.1.1.2.3.2.6 Use of Android head SVG

The author has decided to add an SVG to all shown sprint #3 wireframes. This is unlike the other placeholder images made by the author. The purpose of the SVG's inclusion is simple to just make the application seem more friendly and attractive with the use of a "cute" head of the popular Android mascot. This idea is the same as using the android figure to represent the Little Man in the simulator action display. These ideas were taken from LMC #1 as it also uses an Android mascot figure (the only difference is that its one static picture of it in the colour blue instead of the original green colour). Mascots add personality and human-like qualities to whatever it is tied to – satisfying Humanism*. This was already achieved in the simulator page, but adding this mascot to other pages makes sure that Humanism is not limited to one page. Having elements of Humanism, even to a limited extent, in every corner of the application will make the whole thing more engaging and encourage exploration outside of just the simulator page, such as engaging with reading the manual.

The reader may be confused as to why, specifically, the Android mascot and not another mascot. This is because the use of relevancy and recognition of the mascot is very important. A well-known and established mascot uses cultural familiarity to deliver a consistent and clear message across any context, as if it is common sense (Correia da Silva, 2021). This allows the mascot to give the user an immediate idea of what the LMC educational game is about. The Android mascot is from the Android company, which often associates itself with computers. For those reasons, the specific Android mascot helps inform the user that the simulator is about a computer processor without much effort as a mental shortcut.

The legality of distributing and modifying the Android mascot is covered in the **Legal Considerations Chapter**.

11.14.1.1.2.4 Styling

Sprint 3 not only makes the style more attractive and comfortable but also has multiple styles (selectable by user) that cater to each considered medical disability or other kind of difficulty, satisfying Humanism.
Red is frowned upon by society because it is often associated with violence (blood) (Molnar, 2013) thus will not be used. Blue light is also known to tire the eyes (Zhao *et al.,* 2018) and disturb sleep patterns (Zhao *et al.,* 2018) but because the author assumes that the classes will not use them at night so it is fine to use, but will try not to use blue as the primary colour.

All styles have the text size increased, helping a big proportion of the audience with their poor vision or dyslexia. They each have a dark mode to help people who have photophobia (sensitivity to light).

11.14.1.1.2.4.1 Default

The default style already has high contrast, so it is also suited for people with Colour Vision Deficiency (colour blindness).

Class/component	Colour (light mode)	Hex code – light mode	Hex code – dark mode
Page background	Creamy	#FFB (#FFFFBB)	#886 (#888866)
Text colour	Black	#000 (#000000)	#FFF (#FFFFFF)
Primary colour	Green	#0F0 (#00FF00)	#058 (#005588)
Secondary colour	Blue	#00F (#0000FF)	#080 (#008800)

Table 11.14—3 - describing the colours, and corresponding hex codes, of the default UI style for sprint 3.

describing the colours, and corresponding hex codes, of the UI style for sprint 2.

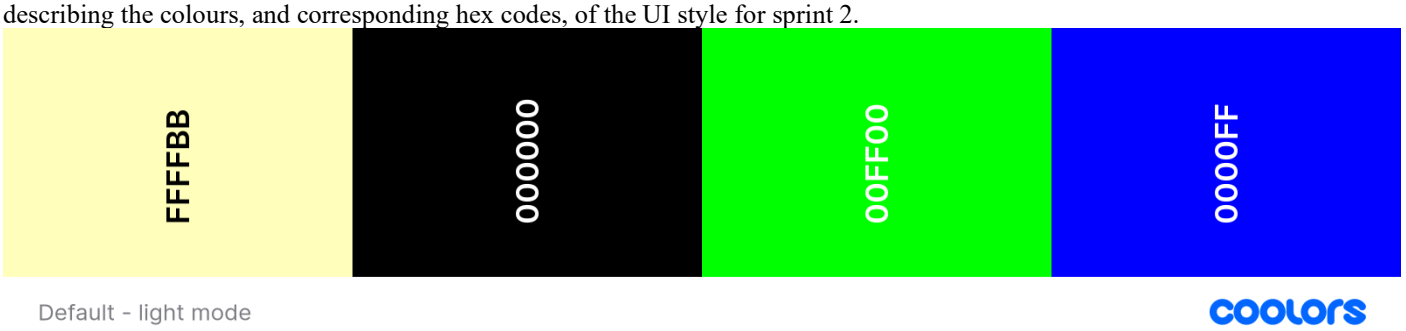


Figure 11.14—15 - colour palette corresponding to the light mode of the default UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

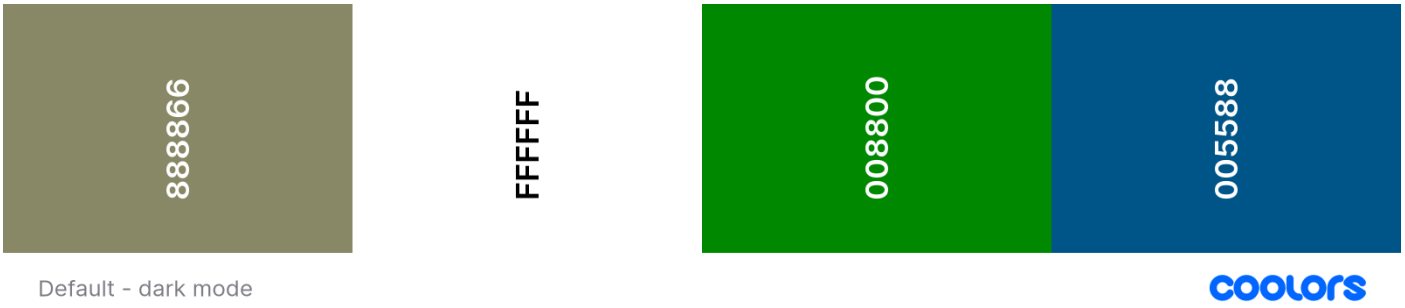


Figure 11.14—16 - colour palette corresponding to the dark mode of the default UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

11.14.1.1.2.4.2 Low Contrast

To help deal with Irlen Syndrome (visual stress). The colour green is used; it is better not to use red nor blue instead due to the reasons that were mentioned previously. The way the entire colour scheme is green-tinted makes the style also suitable with people with Scotopic Sensitivity Syndrome.

Class/component	Colour (light mode)	Hex code – light mode	Hex code – dark mode
Page background	Green	#0F0 (#FFFFBB)	#886 (#888866)
Text colour	Pakistan green	#030 (#003300)	#FFF (#FFFFFF)
Primary colour	Aquamarine	#0FB (#00FFBB)	#058 (#005588)
Secondary colour	Lime	#BF0 (#BBFF00)	#080 (#008800)

Table 11.14—4 - describing the colours, and corresponding hex codes, of the low contrast UI style for sprint 3.

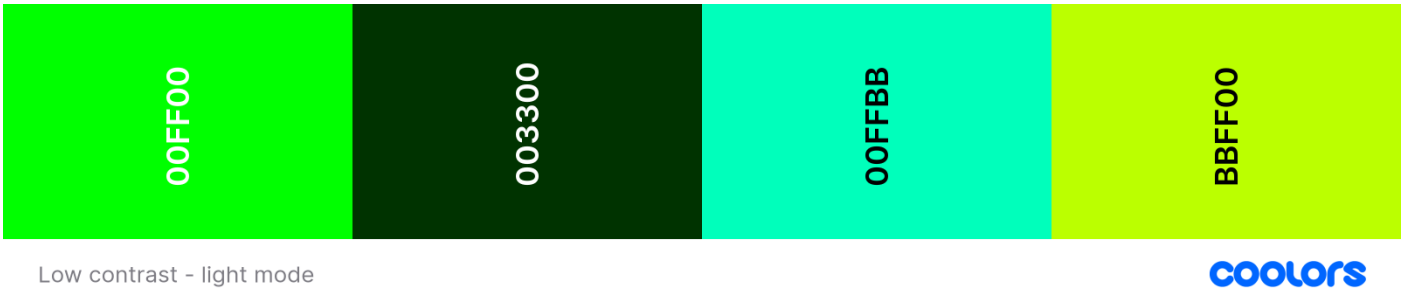


Figure 11.14—17 - colour palette corresponding to the light mode of the low contrast UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

Figure 11.14—18 - colour palette corresponding to the dark mode of the low contrast UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

11.14.1.1.2.4.3 Greyscale

Reduces eye strain in general, can help with more mild cases (Irlen Syndrome, photophobia, et cetera), and especially with colour-blindness.

Class/component	Colour (light mode)	Hex code – light mode	Hex code – dark mode
Page background	White	#FFF (#FFFFFF)	#000 (#000000)
Text colour	Black	#000 (#000000)	#FFF (#FFFFFF)
Primary colour	Silver	#CCC (#CCCCCC)	#666 (#666666)
Secondary colour	Davy’s grey	#333 (#333333)	#555 (#55555)

Table 11.14—5 - describing the colours, and corresponding hex codes, of the greyscale UI style for sprint 3.

According to coolors.co, #CCC and #AAA have the same colour name (silver).

Figure 11.14—19 - colour palette corresponding to the light mode of the greyscale UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

Figure 11.14—20 - colour palette corresponding to the dark mode of the greyscale UI style, of sprint 3, and is exported from Coolors (Bianchi and Fabrizio, 2015).

11.14.1.1.2.4.4 Priorities

Due to the planned implementation of multiple themes, with dark and light modes each, the author decided to make a MoSCoW table for the styling part of the plan.

Must have	Implementing light and dark mode for the default (high contrast) theme.
Should have	Implementing all three themes: the default (high contrast) theme, the low contrast theme, and the grey scale theme. Only the default theme has light and dark modes, totalling 4 styles.
Could have	Implementing light and dark mode per theme of all three themes, totalling 6 themes.
Would have	Alongside the 6 styles, allow the user to use custom colours for each of the CSS variables: page background, text colour, primary colour, and secondary colour.

Table 11.14—6 - MoSCoW priorities regarding the UI styling of the sprint #3 planning.

11.14.1.1.2.5 IPOD tables

All entries in sprint #2's IPOD table are included in sprint #3's IPOD table, but is not shown to save space. Also, the sprint #3 IPOD tables will be more detailed than sprint #2 due to sprint #3 having more pages, hence making IPOD tables more impactful.

Inputs	Process	Outputs	Decisions
Clicking the "Campaign" button	Instantiate *LevelSelector* from **LevelSelector.ts** and *URLQuery* from **URLQuery.ts**.	Load the level selector page (**levelSelection.html**).	What is the current URL query configuration?
Clicking the "Sandbox" button	Instantiate: *Middleware* from **middleware.ts** *SimulatorUI* from **simulator.ts** *URLQuery* from **URLQuery.ts** *ControlUnit*, and its aggravated classes from **vonNeumann.ts** *Compiler* from **compiler.ts**	Load the simulator page (**simulator.html**) in sandbox mode.	What is the current URL query configuration?

	Levelchecker from **levelchecker.ts**.		
Clicking the “Settings” button	Instantiate *Settings* from **settings.ts** and *URLQuery* from **URLQuery.ts**.	Load the settings page (**settings.html**).	What is the current URL query configuration?
Clicking the “Manual” button from the menu	Instantiate *URLQuery* from **URLQuery.ts**. (There is no dedicated class in instantiate for the manual page)	Load the manual page (**manual.html**) in a new browser tab.	What is the current URL query configuration?

Table 11.14—7 - IPOD table for the menu page of sprint #3.

Inputs	Process	Outputs	Decisions
Clicking the “Back To Menu” button	Instantiate *Menu* from **menu.ts** and *URLQuery* from **URLQuery.ts**	Goes back to the menu page (**menu.html**).	What is the current URL query configuration?
Clicking on a level button in the level selection page	Instantiate: *Middleware* from **middleware.ts** *SimulatorUI* from **simulator.ts** *URLQuery* from **URLQuery.ts** *ControlUnit*, and its aggravated classes from **vonNeumann.ts** *Compiler* from **compiler.ts** *Levelchecker* from **levelchecker.ts**.	Load the simulator page (**simulator.html**) in campaign mode.	Which level button did the user click on? What is the current URL query configuration?

Table 11.14—8 - IPOD table for the level selector page of sprint #3.

Inputs	Process	Outputs	Decisions
Clicking the “Back To Menu” button	Instantiate *Menu* from **menu.ts** and *URLQuery* from **URLQuery.ts**.	Goes back to the menu page (**menu.html**).	What is the current URL query configuration?
Toggle Dark/Light Mode	Toggle between light mode and dark mode of the current theme.	Reload the settings page.	What is the current URL query configuration? Is dark mode currently on? (if so then change to light mode, otherwise change to dark mode)
Toggle Sound Effects	Toggle between on and off for sounds effects.	Reload the settings page.	What is the current URL query configuration? Are sound effects currently on? (if so then turn it off, otherwise turn it on)
Change theme	Change theme to selected option.	Reload the settings page.	What is the current URL query configuration?

Table 11.14—9 - IPOD table for the settings page of sprint #3.

Inputs	Process	Outputs	Decisions
Clicking the “Back To Menu” button	Instantiate *Menu* from **menu.ts** and *URLQuery* from **URLQuery.ts**.	Goes back to the menu page (**menu.html**). With an actual menu unlike sprint #2	What is the current URL query configuration?
Clicking the “Submit Input” button	If valid, store input in a field waiting to be read by the simulator when needed.	Replace input text with “Input Cached” or an invalid input message for a few seconds before clearing.	Is input valid.

Table 11.14—10 - partial IPOD table for the simulator page of sprint #3 based of sprint #2’s.

As mentioned, for entries in **Table 11.14—10**, those that were already done in sprint #2 IPOD tables **Table 11.13—2** and **Table 11.13—3** and are not displayed to prevent repeated information.

Inputs	Process	Outputs	Decisions
Clicking the "Submit " button	Instantiate *Menu* from `menu.ts` and *URLQuery* from `URLQuery.ts`.	Goes back to the menu page (`menu.html`). With an actual menu unlike sprint #2	What is the current URL query configuration?
Clicking the "Submit Input" button	If valid, store input in a field waiting to be read by the simulator when needed.	Replace input text with "Input Cached" or an invalid input message for a few seconds before clearing.	Is input valid.

There is no IPOD table for the manual page because the author does not plan to add any form of user input to that page – purely informative.

11.14.1.1.2.6 Modification Mapping Table

In the third sprint, the simulator and the relevant UI elements will be similar so no different mapping table will be used for sprint #3 – still using sprint #2's.

11.14.1.1.3 Whitebox test case planning

Just like of sprint #1 and #2, below is a set tables listing every extremity and aspect/feature of the first sprint that are planned to be tested in sprint #3. These are where the actual test cases will be based on. Justification for each set of tests are also explained in this section. Their implementation is done in the sprint #3's **Test cases** section.

11.14.1.1.3.1.1 Level data checking

Tests for testing level data and their checking have been done in one file called `levelSolutionCases.test.ts` due to their intertwinement with each other, hence the convince of testing both together.

11.14.1.1.3.1.1.1 Data integrity

11.14.1.1.3.1.1.1.1 Valid tests

What is tested?	How is it tested?	Expected result
If level #1 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *1* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #2 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *2* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #3 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *3* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #4 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *4* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #5 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *5* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #6 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *6* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #7 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *7* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #8 adheres to the structure of a tutorial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *8* and *LevelType.tutorial* respectfully and catching returned string.	Empty string indicating no thrown errors.

Table 11.14—11 – plan for tests dedicated for enforcing data integrity of the tutorial levels (levels #1 - #8).

What is tested?	How is it tested?	Expected result

If level #9 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `9` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #12 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `12` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #19 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `19` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.

Table 11.14—12 - plan for tests dedicated for enforcing data integrity of the correction levels (levels #9, #12, #19).

With a keen eye, one may notice that some level types are treated as other ones, such as correction levels treated as partial levels. This will be addressed later.

What is tested?	How is it tested?	Expected result
If level #11 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `11` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #15 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `15` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #18 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `18` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.

Table 11.14—13 - plan for tests dedicated for enforcing data integrity of the partial levels (levels #11, #15, #18).

What is tested?	How is it tested?	Expected result
If level #10 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `10` and `LevelType.contexual` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #20 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `20` and `LevelType.contexual` respectfully and catching returned string.	Empty string indicating no thrown errors.

Table 11.14—14 - plan for tests dedicated for enforcing data integrity of the big formula levels (levels #10 and #20).

What is tested?	How is it tested?	Expected result
If level #13 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `13` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #14 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `14` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #16 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `16` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.
If level #17 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `17` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.

Table 11.14—15 - plan for tests dedicated for enforcing data integrity of the contextual levels (levels #13, #14, #16, and #17).

11.14.1.1.3.1.1.1.2 Purposely invalid tests

What is tested?	How is it tested?	Expected result
If level #1 adheres to the structure of a partial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `1` and `LevelType.partial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.
If level #1 adheres to the structure of a contextual level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `1` and `LevelType.contextual` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.

Table 11.14—16 – plan for tutorial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` helper function, are not faulty.

What is tested?	How is it tested?	Expected result
If level #9 adheres to the structure of a partial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `9` and `LevelType.tutorial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.

If level #9 adheres to the structure of a contextual level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `9` and `LevelType.contextual` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.

Table 11.14—17 – plan for partial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` helper function, are not faulty.

What is tested?	How is it tested?	Expected result
If level #10 adheres to the structure of a partial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `10` and `LevelType.partial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.
If level #10 adheres to the structure of a contextual level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `10` and `LevelType.tutorial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.

Table 11.14—18 – plan for partial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` helper function, are not faulty.

11.14.1.1.3.1.1.2 Level checking

11.14.1.1.3.1.1.2.1 Using the example solution as the submitted script (basic checking)

What is tested?	How is it tested?	Expected result
Checking submitted script of level #9 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `9`.	Empty string indicating no thrown errors.
Checking submitted script of level #10 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `10`.	Empty string indicating no thrown errors.
Checking submitted script of level #11 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `11`.	Empty string indicating no thrown errors.
Checking submitted script of level #12 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `12`.	Empty string indicating no thrown errors.
Checking submitted script of level #13 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `13`.	Empty string indicating no thrown errors.
Checking submitted script of level #14 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `14`.	Empty string indicating no thrown errors.
Checking submitted script of level #15 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `15`.	Empty string indicating no thrown errors.
Checking submitted script of level #16 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `16`.	Empty string indicating no thrown errors.
Checking submitted script of level #17 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `17`.	Empty string indicating no thrown errors.
Checking submitted script of level #18 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `18`.	Empty string indicating no thrown errors.
Checking submitted script of level #19 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `19`.	Empty string indicating no thrown errors.
Checking submitted script of level #20 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `20`.	Empty string indicating no thrown errors.

*Table 11.14—19 – plan for basic tests for testing the `levelChecker` methods' inner-workings and correctness of information in each level's corresponding data in **`levelData.ts`**, including example solution (`.exampleSolution`) and solution cases (`.cases`).*

11.14.1.1.3.1.1.2.2 More detailed checking

What is tested?	How is it tested?	Expected result

Dealing with script that does not meet level's objective (`0` stars) using different program.	Using level `9` and, instead of the wanted program of outputting the bigger output of the two, a compiled script is submitted that just outputs the second input. The compiled script: `901, 901, 902`.	0
Dealing with script that does not meet level's objective (`0` stars) using an infinite branching loop.	Using level `9` and, instead of the wanted program of outputting the bigger output of the two, a compiled script is submitted that branches/loops infinitely until the simulator's auto-timeout activates (on the 300th cycle). The compiled script: `901, 316, 901, 602`.	0
Making sure tests are accurate.	Using level `9` with compiled of the example solution but it checks that if star count is not `0`, `1`, or `2` instead of just checking if it is 3. The compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0`.	3
Testing the 3-star limit.	Using level `9` with compiled of the example solution. The compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0`.	3
Testing the 2-star lower limit.	Using level `9` with compiled of the example solution but with one extra instruction. The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0`.	2
Testing the 2-star upper limit.	Using level `9` with compiled of the example solution but with five extra instructions. The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0, 0, 0, 0, 0`.	2
Testing the 1-star.	Using level `9` with compiled of the example solution but with six extra instructions. The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0, 0, 0, 0, 0, 0`.	1

Table 11.14—20 - plan for tests for testing the `levelChecker` methods' inner-workings of information in each level's corresponding data in `levelData.ts` including solution cases (`.cases`) and star limits (`.stars`).

11.14.1.1.4 Critical path analysis

Little Man Computer educational game

Project by - James Haddad

Supervisor - Tina Eager

Project Start:	Tue, 10/1/2024
Project End:	Mon, 5/12/2025
Display Week:	1

TASK	STATUS	START	END
Literature review	Fully done		
Academic research	Fully done	10/11/24	11/24/24
Software research	Fully done	10/21/24	12/8/24
Create poster for submission	Fully done	12/9/24	1/17/25
Requirement analysis	Fully done	11/25/24	11/26/24
Technologies used	Fully done	11/27/24	12/1/24
Priorities	Fully done	12/2/24	12/8/24
Prepare and perform poster presentation	Fully done	1/18/25	1/21/25
Sprint 1	Fully done		
Pre-sprint preparations	Fully done	1/20/25	1/26/25
Prior research	Fully done	1/27/25	2/2/25
Design	Fully done	2/3/25	2/9/25
Critical path analysis	Fully done	2/10/25	2/16/25
Developmment - implementation	Fully done	2/17/25	2/23/25
Developmment - testing	Fully done	2/24/25	2/26/25
Review and self-reflection	Fully done	2/27/25	3/2/25
Sprint 2	Mostly done		
Prior research	Fully done	2/24/25	3/2/25
Design	Fully done	3/3/25	3/9/25
Critical path analysis	Fully done	3/10/25	3/16/25
Developmment - implementation	Mostly done	3/17/25	3/23/25
Developmment - testing	Nearly done	3/24/25	3/26/25
Review and self-reflection	Fully done	3/27/25	3/30/25
Sprint 3	Currently doing		
Prior research	Fully done	3/24/25	3/30/25
Design	Fully done	3/31/25	4/6/25
Critical path analysis	Currently doing	4/7/25	4/13/25
Developmment - implementation	haven't started	4/14/25	4/20/25
Developmment - testing	haven't started	4/21/25	4/23/25
Review and self-reflection	haven't started	4/24/25	4/27/25
Post sprint 3			
Project review and reflection	haven't started	4/21/25	4/29/25
Client testing	haven't started	4/27/25	5/5/25
Polishing	haven't started	5/6/25	5/12/25

Figure 11.14—21 - current state of the GANTT chart tasks at this moment in sprint #3.

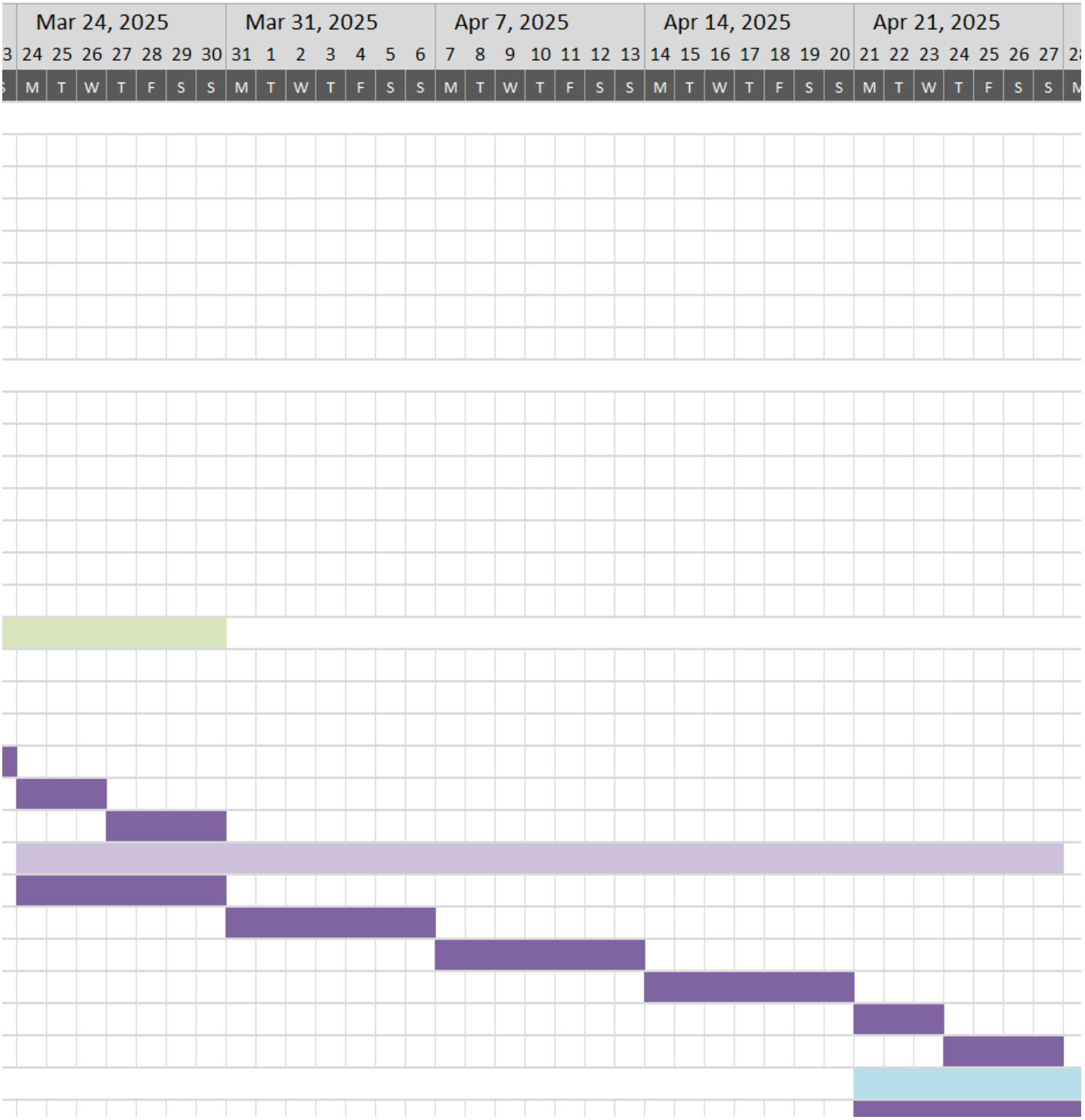


*Figure 11.14—22 - view of GANTT tasks duration and deadlines corresponding to sprint #3 GANTT tasks in **Figure 11.14—21**.*

11.14.1.2 Development

11.14.1.2.1 Test cases

Unlike the tests used in sprint #1 and #2, tests in sprint #3 utilise the use of helper functions. This is mostly because most of the tests are just recurring variations of the same code checkers with different values. Scaling up this repetition with the number of tests used makes the use of testing far more time-consuming than the time saved from fully utilising them for the elimination of bugs and inconsistencies. The reader may initially be alarmed by the use of functional programming in the project, breaking the OOP practice. But let the author reassure the reader, as these functions are only used in testing and are not part of the software product itself. Functional programming is only used in testing due to its advantage of rapid programming for less complex programs.

11.14.1.2.1.1 Whitebox testing

Test cases done based upon the test plan from sprint #3's **Whitebox test case planning** section, justification and explanation are also done there.

11.14.1.2.1.1.1 Level data checking

Tests for testing level data and their checking have been done in one file called **`levelSolutionCases.test.ts`** due to their intertwinement with each other, hence the convince of testing both together.

This file is split between testing the integrity of the level data and testing the level checking mechanics of methods from *`LevelChecker`* (from **`levelChecker.ts`**) with the example solution of each level from the level data (*`LevelData`* object literal list from **`levelData.ts`**).

11.14.1.2.1.1.1.1 Data integrity

For context, data of each level is stored as an object literal defined by the *`Level`* interface.

```
export interface Level {
    //* Object attributes listed in order from simple to complex data structure for easy population and organisation.
    objective:string;
    //^ Explaining the level
    exampleCase:string;
    //^ Example of inputs (except level 20) and corrosponding outputs
    stars?: { 2:number; 3:number; };
    //^ Not using list instead because it is only has a mere 2 elements and is easier to refer this way.
    //^ Purposely listed 3 then 2 instead of 2 then 3 due to being in a decending importance order.
    //^ Must be populated unless tutorial/instruction level.
    patrialScript?:[string, string, string][];
    //^ If level provides partially complete/correct script to display in assemvly code script editor.
    //^ Can be empty.
    //^ elements: label, opcode, operand.
    //^ Must be populated only if level is about correcting, completing an existing script, or a tutorial/instruction level.
    exampleSolution?:[string, string, string][];
    //^ If user have no idea (which they shouldn't), then they can see an example solution and learn from it.
    //^ Must be populated unless tutorial/instruction level.
    cases?:[number[], string[]][];
    //^ Must be populated with atleast one sub-list unless tutorial/instruction level.
    //^ 'number[]' is the pre-defined inputs (optional - can be empty).
    //^ 'string[]' is expected outputs.
}
```

Figure 11.14—23 – the `Level` *interface.*

All *`Level`* objects are stored in the *`Level`* 1-dimensional list *`LevelData`* as seen below.

```
export const levelData:Level[] = [
    //^ Simpler and quicker compute time to referance an object by index then a sub-object by key.
    //* If level does not have 'exampleSolution', 'cases' and 'stars', it is a tutorial/introduction level type.
    //* If level does not have 'patrialScript', it is either a contexual or a big formula level type.
    //* If level does has everything, it is either a correcting or a partial level type.
    /* Template:
    { //< Level :
        //* level.
        objective: '',
        exampleCase: '',
        stars: { 3:, 2: },
        patrialScript:
        exampleSolution:
        cases:
    },
    */
```

Figure 11.14—24 - top part of the `levelData Level` *list declaration.*

This interface stores multiple types of levels as mentioned in the comments inside *`LevelData`* using optional attributes in the *`Level`* interface. This is very useful for keeping the storage and use of level data quick and simple, but may lead to mistakes in using optional attributes for a level type that does not use them or vice versa. To combat it, tests are made to enforce the integrity of the data per level type, as seen in the tables below. Each test table will refer to one group of tests.

As seen in the sub-sections, there are no amendments made in the test analysis tables. This supports the author's point that using a single interface for simplicity is the best choice. The author did double-check the helper function and the purposely invalid tests to make sure results were not fake and there were indeed no fake results.

11.14.1.2.1.1.1.1 Valid tests

What is tested?	How is it tested?	Expected result	Actual result	Changes made

If level #1 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `1` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #2 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `2` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #3 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `3` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #4 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `4` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #5 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `5` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #6 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `6` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #7 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `7` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #8 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `8` and `LevelType.tutorial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

Table 11.14—21 - tests dedicated for enforcing data integrity of the tutorial levels (levels #1 - #8).

What is tested?	How is it tested?	Expected result	Actual result	Changes made
If level #9 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `9` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #12 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `12` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #19 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `19` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

Table 11.14—22 - tests dedicated for enforcing data integrity of the correction levels (levels #9, #12, #19).

What is tested?	How is it tested?	Expected result	Actual result	Changes made
If level #11 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `11` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #15 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `15` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #18 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `18` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

Table 11.14—23 - tests dedicated for enforcing data integrity of the partial levels (levels #11, #15, #18).

What is tested?	How is it tested?	Expected result	Actual result	Changes made
If level #10 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `10` and `LevelType.contexual` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #20 adheres to the	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `20` and	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

structure of a tutorial level.	`LevelType.contexual` respectfully and catching returned string.			

Table 11.14—24 - tests dedicated for enforcing data integrity of the big formula levels (levels #10 and #20).

What is tested?	How is it tested?	Expected result	Actual result	Changes made
If level #13 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `13` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #14 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `14` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #16 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `16` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
If level #17 adheres to the structure of a tutorial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `17` and `LevelType.partial` respectfully and catching returned string.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

Table 11.14—25 - tests dedicated for enforcing data integrity of the contextual levels (levels #13, #14, #16, and #17).

11.14.1.2.1.1.1.2 Purposely invalid tests

What is tested?	How is it tested?	Expected result	Actual result	Changes made

If level #1 adheres to the structure of a partial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `1` and `LevelType.partial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	`expect(received).not.toBe(expected) // Object.is equality`
			
			`Expected: not undefined`
				None. (actual is expected)

If level #1 adheres to the structure of a contextual level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `1` and `LevelType.contextual` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	`expect(received).toBe(expected) // Object.is equality`
			
			`Expected: undefined`
			
			`Received: [["", "INP", ""], ["", "BRP", "positive"], ["", "BRZ", "zero"], ["", "HLT", ""], ["zero", "OUT", ""], ["", "OUT", ""], ["", "HLT", ""], ["positive", "OUT", ""], ["", "HLT", ""]]`
			
			`PASS src/compiled/tests/levelSolution`
				None. (actual is expected)

Table 11.14—26 – tutorial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` helper function, are not faulty.

What is tested?	How is it tested?	Expected result	Actual result	Changes made

If level #9 adheres to the structure of a partial level.	Calling the helper function `integretyHelper` with the arguments `level` and `levelType` as `9` and `LevelType.tutorial` respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	`expect(received).toBe(expected) // Object.is equality`
			
			`Expected: undefined`
			
			`Received: [["", "INP", ""], ["", "STA", "num1"], ["", "INP", ""], ["", "STA", "num2"], ["", "SUB", "num1"], ["", "BRP", "pos"], ["", "LDA", "num1"], ["", "OUT", ""], ["", "BRA", "exit"], ["pos", "LDA", "num2"], …]`
				None. (actual is expected)

If level #9 adheres to the structure of a contextual level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *9* and *LevelType.contextual* respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	*expect(received).toBe(expected) // Object.is equality* *Expected: undefined* *Received: [["", "INP", ""], ["", "STA", "num1"], ["", "INP", ""], ["", "STA", "num2"], ["loop", "LDA", "TOTAL"], ["", "ADD", "num1"], ["", "STA", "total"], ["", "LDA", "num2"], ["", "SUB", "one"], ["", "STA", "num2"], …]*	None. (actual is expected)

Table 11.14—27 – partial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` *helper function, are not faulty.*

What is tested?	How is it tested?	Expected result	Actual result	Changes made
If level #10 adheres to the structure of a partial level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *10* and *LevelType.partial* respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	*expect(received).not.toBe(expected) // Object.is equality* *Expected: not undefined*	None. (actual is expected)
If level #10 adheres to the structure of a contextual level.	Calling the helper function *integretyHelper* with the arguments *level* and *levelType* as *10* and *LevelType.tutorial* respectfully and catching returned string.	Not empty string – the message of the Jest-thrown error.	*expect(received).not.toBe(expected) // Object.is equality* *Expected: not undefined*	None. (actual is expected)

Table 11.14—28 – partial level tests dedicated for ensuring that the tests, and especially the `integretyHelper` *helper function, are not faulty.*

11.14.1.2.1.1.1.3 Helper function

Below shows the first test of the level data integrity tests.

```
127  describe('Testing level data integrety', () => {
         Run | Debug
128      describe('Testing each level if valid to their type', () => {
             Run | Debug
129          describe('Tutorial/introduction levels (level #1-8)', () => {
                 Run | Debug
130              test('level #1', () => {
131                  const result = integretyHelper(1, LevelType.tutorial);
132                  //^ calls the 'integretyHelper' helper function
133                  console.log(result);
134                  //^ prints whatever string was caught from the call
135                  expect(result).toBe("");
136                  //^ if string is empty (""), then helper result experienced no jest-thrown errors
137              });
```

Figure 11.14—25 - Example of one of the level data integrity tests.

As seen with that test, alongside every other level data integrity test, *integretyHelper* gets called as a helper function. This helper function can be seen below.

```
function integretyHelper(level, levelType) {
    try {
        const currentLevel = levelData[level - 1];
        //^ minus 1 because level number is one above corrosponding index (example: level #1 has insex of 0)
        expect(currentLevel).not.toBeUndefined();
        //^ check if data for specified level exists
        if (currentLevel == undefined) {
            return "Custome error - unexpected error happend - first parameter value is undefined!!!";
        }
        //^ Solved TS-18048 but is if-statement is never satisfied because if it would then the above expect
        //^ statement would of fail and end the current test beforehand by throwing an error immediately.
        //: Should be present and populated in all levels.
        //: Not checked for undefined as interface forbids that.
        expect(currentLevel.objective).not.toBe('');
        expect(currentLevel.exampleCase).not.toBe([]);
        switch (levelType) {
            case LevelType.tutorial:
                //: unpopulated
                expect(currentLevel.exampleSolution).toBe(undefined);
                expect(currentLevel.cases).toBe(undefined);
                expect(currentLevel.stars).toBe(undefined);
                //: populated
                expect(currentLevel.patrialScript).not.toBe(undefined);
                expect(currentLevel.patrialScript).not.toBe([]);
                break;
            case LevelType.partial:
                //* All attributes must be populated
                //: populated
                expect(currentLevel.patrialScript).not.toBe(undefined);
                expect(currentLevel.patrialScript).not.toBe([]);
                expect(currentLevel.exampleSolution).not.toBe(undefined);
                expect(currentLevel.exampleSolution).not.toBe([]);
                expect(currentLevel.cases).not.toBe(undefined);
                expect(currentLevel.cases).not.toBe([]);
                expect(currentLevel.stars).not.toBe(undefined);
                if (currentLevel.stars) {
                    //^ Solved TS-18048
                    expect(currentLevel.stars[2]).not.toBe(null);
                    expect(currentLevel.stars[3]).not.toBe(null);
                }
                break;
            default: //< LevelType.contexual
                expect(currentLevel.patrialScript).toBe(undefined);
                //^ unpopulated
                //: populated
                expect(currentLevel.exampleSolution).not.toBe(undefined);
                expect(currentLevel.exampleSolution).not.toBe([]);
                expect(currentLevel.cases).not.toBe(undefined);
                expect(currentLevel.cases).not.toBe([]);
                expect(currentLevel.stars).not.toBe(undefined);
                if (currentLevel.stars) {
                    //^ Solved TS-18048
                    expect(currentLevel.stars[2]).not.toBe(null);
                    expect(currentLevel.stars[3]).not.toBe(null);
                }
        }
    }
    catch (jestError) {
        return jestError.message;
    }
    return "";
}
```

Figure 11.14—26 - the helper function called in every level data integrity test (`integretyHelper`).

As seen, the helper function checks some attributes and the rest depending on the level type.

As the reader already knows, there are 5 types of levels: tutorial, correcting, partial, contextual, and big formula, yet the helper function only accepts 3 types of levels. This is because some levels have the exact same characteristics from the program's

perspective. For further explanation, correcting type levels is for correcting existing scripts while partial type levels are completing existing scripts, but both involve the use of an existing script from the `partialScript` attribute of the `Level` object. Meanwhile, the only difference between contextual and big formulas is that the objective of contextual is more scenario based while the big formula objective is more for making a script that reflects a formula, such as a linear graph, Fibonacci, et cetera.

11.14.1.2.1.1.1.2 Level checking

11.14.1.2.1.1.1.2.1 Using the example solution as the submitted script (basic checking)

Unlike the level data integrity, many issues and corresponding amendments persist, ranging from mistyped data to unaccounted for factors in the script compilation process.

What is tested?	How is it tested?	Expected result	Actual result	Changes made
Checking submitted script of level #9 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `9`.	Empty string indicating no thrown errors.	Expecting 3 stars but got 2 stars instead because it was using level 10's example solution for level 9's level check. Expecting 3 stars but got 0 stars instead because example solution script was not made for overflows. Expecting 3 stars but got 2 stars instead because accidently used 13 instead of 14 as the 2-star limit.	Deeper change #1 Because solution script is from LMC #2, so author changed object and cases attributes to not take overflow into consideration. Changed cases: `[5,5;5], [999,999;999], [999,-999;999]` → `[998,999;999], [-499,498;498], [498,-499;498]`. During debugging, discovered that the pre-defined input list, of the solution cases, becomes empty overtime which changes that the check message that relies on this, this is addressed and fixed in deeper change #2. incremented value of the 2-star (`levelData[10].stars[2]`) from 13 to 14. Also did that same as the 3-star value as it was relative to the 2-star value.
Checking submitted script of level #10 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `10`.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
Checking submitted script of level #11 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `11`.	Empty string indicating no thrown errors.	Expecting 3 stars but got 0 stars instead. Tested script in frontend which worked fine. Turns out author forgot to save previous changes to the example solution.	Applied changes to both `exampleSolution` and `partialCases` attributes: `["", "INP", ""], ["loop", "SUB", "two"], ["", "BRZ", "isEven"]` → `["", "INP", ""], ["loop", "BRZ", "isEven"], ["", "SUB", "two"]` (the first 3 lines/sub-lists shown).
Checking submitted script of level #12 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `12`.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
Checking submitted script of level #13 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `13`.	Empty string indicating no thrown errors.	Compilation out was `[-1]` – indicating failure. Error was "*Out-of-bounds address error at the operand of line 4 - label "273" is integer but out of bounds (0-99)*" despite `273` not intended of being an address. This was due to use of incorrect corresponding opcode.	Replaced the accidental `LDA` with the originally intended `DAT`: `["const", "LDA", "273"]` → `["const", "DAT", "273"]`. Replaced the accidental `258` with the originally intended `248` (3rd case): `[[-25], ["258"]]` → `[[-25], ["248"]]`.

			Expecting 3 stars but got 0 stars instead because of a mistype in the cases attribute.	
Checking submitted script of level #14 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `14`.	Empty string indicating no thrown errors.	Expecting 3 stars but got 0 stars instead because of a mistype in the cases attribute.	Replaced the accidental `21` with the originally intended `31` (6th case): `[[21], ["961"]]` → `[[31], ["961"]]`.
Checking submitted script of level #15 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `15`.	Empty string indicating no thrown errors.	Expecting 3 stars but got 0 stars instead because of a mistype in the cases attribute.	Replaced the accidental `0` with the originally intended `999` (4th case): `[[999, 0], ["0"]]` → `[[999, 0], ["999"]]`.
Checking submitted script of level #16 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `16`.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
Checking submitted script of level #17 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `17`.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)
Checking submitted script of level #18 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `18`.	Empty string indicating no thrown errors.	Compilation out was `[-1]` – indicating failure. Error was "*Invalid token error at the opcode of line 2 - "STO" is not a valid opcode token. Valid opcodes: ADD, SUB, STA, LDA, SHL, SHR, BRA, BRZ, BRP, INP, OUT, HLT, DAT*". This was a simple mistype which was also found in levels #19 and #20. Compilation out was `[-1]` – indicating failure. Error was "*operand not expected error at the operand of line 51 - label "000" is integer but out of bounds (-999 - 999)*". Which indicates that the compiler does not except zero-character ("0") padding. Compilation out was `[-1]` – indicating failure. Error was "*Missing label error at the operand of line 18 - label "F" is called but cannot find code line with that label.*" Which turns out was because the label declarations were in the opcode token slot instead of the label slot.	Replaced all instances of `STO` with `STA`, at once using Ctrl+h shortcut, which also fixed the same problem in levels #19 and #20 as well. Deeper change #3 Moved label declarations from opcode token slot to label token slot: `["", "F", "DAT 070"], ["", "I", "DAT 073"], ["", "Z", "DAT 090"], ["", "B", "DAT 066"], ["", "U", "DAT 085"]` → `["f ", "DAT", "070"], ["i", "DAT", "073"], ["z", "DAT", "090"], ["b", "DAT", "066"], ["u", "DAT ", "085"]`. Also, make label letters lowercase to fit the scripting style of the example solutions and partial scripts of the other levels (to not confuse the user with the capital opcodes). Changed most of the ASCII codes (operands) to be lowercase: `["", "f", "DAT 070"], ["", "i", "DAT 073"], ["", "z", "DAT 090"], ["", "b", "DAT 066"], ["", "u", "DAT 085"]` → `["f", "DAT", "070"], ["i", "DAT", "105"], ["z", "DAT", "122"], ["b", "DAT", "066"], ["u", "DAT", "117"]`.

			Expecting 3 stars but got 0 stars instead. Accidently used ASCII codes for all capital letters instead of lower-case letters (except `F` and `B`).	
Checking submitted script of level #19 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `19`.	Empty string indicating no thrown errors.	Expecting 3 stars but got 0 stars instead. Program timed out due to infinite assembly loop. Because accidently used `15` instead of `-15` preventing accumulator from being below 0.	Replaced the accidental `15` with the originally intended `-15` (5th case): `[[10, 0, 15], ["10", "10", "e", "m", "p", "t", "y"]]` → `[[10, 0, -15], ["10", "10", "e", "m", "p", "t", "y"]]`. Also moved added `return;` syntax to `.constructResultMessage()` method (in **`levelChecker.ts`**) so it will not overwrite message with error message when level objective is satisfied. before testing again, realised that 6th case had opcode out of range (1000>999) so author fixed that: `[[999, 1000], ["999", "e", "m", "p", "t", "y"]]` → `[[999, -999, -999], ["999", "0", "e", "m", "p", "t", "y"]]`.
Checking submitted script of level #20 using its corresponding example solution	Calling the helper function `checkerHelper3Star` with the argument `level` as `20`.	Empty string indicating no thrown errors.	Same as expected result.	None. (actual is expected)

Table 11.14—29 – basic tests for testing the `levelChecker` *methods' inner-workings and correctness of information in each level's corresponding data in* **`levelData.ts`**, *including example solution (*`.exampleSolution`*) and solution cases (*`.cases`*).*

Below is a table of deeper changes which manipulated code outside of the test file **`levelSolutionCases.test.ts`** and the `levelData` in **`levelData.ts`**.

Deeper change #	File's code changed	The change
1	**`levelChecker.ts`**	Fixed in the constructor of `LevelChecker` class where the author forgot to account for decrementing level number to account for index starting from 0 instead of 1: `this.currentLevel = levelData[levelId] as Level;` → `this.currentLevel = levelData[levelId-1] as Level;`.
2	**`levelChecker.ts`**	In `LevelChecker.testCases()` method, the list of pre-defined inputs becomes empty over time, which makes the message field incomplete when constructing an error message (from `this.constructResultMessage(casesResult);` method call). This is because the pre-defined inputs list `currentCase[0]` is passed by reference into `ControlUnit` due to the object nature of a JS/TS list. To combat this is to simply make a deep copy of the list and insert that into the `ControlUnit` constructor parameter as seen: `const simulator:ControlUnit = new ControlUnit( this.userCompiled, JSON.parse(JSON.stringify(currentCase[0])) );`. This was chosen over shallow copy because the author is very familiar with the concept of deep copy, and knows how to practically do it in contrast to shallow copy. JSON methods were used instead, for the deep copy, of the `structuredClone` call because it only works in modern browsers, thus not having much back-compatibility with older browsers, making `JSON.parse` and `JSON.stringify` methods the better choice.
3	**`compiler.ts`**	The compiler's "*operand not expected error at the operand of line 51 - label "000" is integer but out of bounds (-999 - 999)*" custom error means that the Compiler class does not consider DAT corresponding operand valid if it has zero-character ("0") padding such as "000", "-011", "009", et cetera. The author immediately knew it must be because of the corresponding regex statement. The regex statement `/^-?(?:[1-9]?\d{0,2}|0)$/` only accepts from -999 to 999 without leading zeros, except zero by itself. The author decided to use more simpler regex rules to fix it which leads to changing the line `if (!(/^-?(?:[1-9]?\d{0,2}|0)$/).test(token)) {` to `if (!(/^-?\d{1,3}$/).test(token)) {`. Where `/^-?\d{1,3}$/` accepts any string consisting of only 1 to 3 digits.

		This change made the author realise that this problem will also occur with the compiler's check on address operands (non-`DAT` corresponding operands). Hence the author also changed the address operand corresponding regex-contained line from `if (!(/^(?:[1-9]?\d|0)$/).test(token)) {` to `if (!(/^\d{1,2}$/).test(token)) {`.

Table 11.14—30 – Deeper changes, in other files, referenced by Table 11.14—29.

11.14.1.2.1.1.1.2.2 More detailed checking

What is tested?	How is it tested?	Expected result	Actual result	Changes made
Dealing with script that does not meet level's objective (`0` stars) using different program.	Using level `9` and, instead of the wanted program of outputting the bigger output of the two, a compiled script is submitted that just outputs the second input. The compiled script: `901, 901, 902`.	0	0	None. (actual result was expected)
Dealing with script that does not meet level's objective (`0` stars) using an infinite branching loop.	Using level `9` and, instead of the wanted program of outputting the bigger output of the two, a compiled script is submitted that branches/loops infinitely until the simulator's auto-timeout activates (on the 300th cycle). The compiled script: `901, 316, 901, 602`.	0	0	None. (actual result was expected)
Making sure tests are accurate.	Using level `9` with compiled of the example solution but it checks that if star count is not `0`, `1`, or `2` instead of just checking if it is 3. The compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0`.	3	3	None. (actual result was expected)
Testing the 3-star limit.	Using level `9` with compiled of the example solution. The compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0`.	3	2	This was due to the use of 'greater than or equal' instead of just 'greater than' so replace `=>` with `>` in `LevelChecker.starcount`: `lineCount => this.currentLevel.stars` → `lineCount > this.currentLevel.stars`.
Testing the 2-star lower limit.	Using level `9` with compiled of the example solution but with one extra instruction. The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0`.	2	2	None. (actual result was expected)
Testing the 2-star upper limit.	Using level `9` with compiled of the example solution but with five extra instructions. The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0, 0, 0, 0, 0`.	2	2	None. (actual result was expected)
Testing the 1-star.	Using level `9` with compiled of the example solution but with six extra instructions.	1	1	None. (actual result was expected)

	The modified compiled script: `901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0, 0, 0, 0, 0, 0.`			

Table 11.14—31 - tests for testing the `levelChecker` methods' inner-workings of information in each level's corresponding data in `levelData.ts` including solution cases (`.cases`) and star limits (`.stars`).

11.14.1.2.1.1.1.2.3 Helper functions

Below shows the first test of the level data integrity tests.

```
Run | Debug
describe('Testing level checking', () => {
    Run | Debug
    describe('Successful checks testing', () => {
        Run | Debug
        describe('Levels #9-10', () => {
            Run | Debug
            test('Level 9', async () => {
                const result:string = await checkerHelper3Star(9);
                console.log(result);
                expect(result).toEqual("");
            });
```

Figure 11.14—27 - Example of one of the level checking tests calling the `checkerHelper3Star` helper function.

As the reader can see, the *`checkerHelper3Star`* is a helper function dedicated to the basic testing of both the *`LevelChecker`* functionality, correctness of each level's set of cases (except tutorial levels), and correctness of each level's example solution mnemonic scripts (also except tutorial levels). The *`checkerHelper3Star`* helper function can be seen below.

```
17  //#region helper functions
18  async function checkerHelper3Star(level:number):Promise<string>{
19      try{
20          const compiler:Compiler = new Compiler();
21          //^ To compile the solution into accepcted compiled form and check example solution's validity.
22          const checker:LevelChecker = new LevelChecker(level);
23          //^ To check submission script by checking if its execution matches the 'cases' (class calls 'ControlUnit' during testing).
24          const scriptSolution:string[][] = checker.getExample();
25          //^ Assume that user submits the same as the example solution allowing testing of both submittng scripts and the example solution's validity.
26          const uppercaseScriptSolution:string[][] = scriptSolution.map( (line:string[]):string[] => line.map((token:string):string => token.toUpperCase()) );
27          //^ To simulate SimulatorUI's capitalisation feature.
28          //^ Sources - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map
29          //^ and https://stackoverflow.com/questions/29719329/convert-array-into-upper-case .
30          const compiledSolution:number[] = compiler.validateAndCompile(uppercaseScriptSolution);
31          //^ Attempt to compile script.
32          console.log(compiledSolution);
33          //^ For debugging purposes - has example solution compiled properly?
34          console.log(compiler.getMessage);
35          //^ For debugging purposes - explaination of compiler's result.
36          expect(compiledSolution).not.toBe([-1]);
37          //^ If [-1], then solution script is not valid.
38          checker.setUserCompiled(compiledSolution);
39          //^ Submitt successfully compiled script to level checker.
40          const levelStatus:number = await checker.assessScript();
41          //^ Assess submitted script
42          console.log(checker.getMessage());
43          //^ For debugging purposes - explaination of checker's level script checking result.
44          console.log(`Level #${level} star count:${levelStatus}`);
45          //^ Also for debugging purposes - to see what was actual star count.
46          expect(levelStatus).toBe(3);
47          //^ Expected to get '3' (3-star) as example solution is always within 3-star criteria/limits.
48      }
49      catch(jestError){ return (jestError as Error).message; }
50      return "";
51  }
```

Figure 11.14—28 - the `checkerHelper3Star` test helper function used for the basic level checking tests.

For the detailed checking test, a different helper function gets called, as seen below.

```
                Run | Debug
374             test('2-star message - lower limit', async () => {
375                 const stars:number = await checkerHelperCustom(9,[901, 312, 901, 313, 212, 809, 512, 902, 611, 513, 902, 0, 0, 0, 0]);
376                 //^ Same script but has an extra HLT instruction appended to the end to be at the lower limit of being concidered 2-star.
377                 expect(stars).toBe(2);
```

Figure 11.14—29 - Example of one of the level checking tests calling the `checkerHelperCustom` helper function.

The difference between the helper functions is that *checkerHelper3Star* returns any jest-thrown error when expecting a 3-star count result while checkerHelperCustom simply returns the star count – allowing testing of fail (0 stars) and different levels of success (1 star, 2 stars, and 3 stars).

```
52  async function checkerHelperCustom(level:number,scriptCompiled:number[]):Promise<number>{
53      //* Less automatic and more manual as an invalid-checking alternative to 'checkerHelper3Star'.
54      //* Ommited the compiler part because it compiler has its own test file ans was used in 'checkerHelper3Star' because the example solution was not compiled.
55      //* Does not have try-block like other helper function because no there are no 'expect' statements to possibly throw invalid/jest error.
56      const checker:LevelChecker = new LevelChecker(level);
57      checker.setUserCompiled(scriptCompiled);
58      const levelStatus:number = await checker.assessScript();
59      //^ Get either 0 (fail), 1, 2, or 3 stars.
60      return levelStatus;
61  }
```

Figure 11.14—30 - the checkerHelperCustom test helper function that is used for the detailed level checking tests.

As the reader may notice, despite checkerHelperCustom giving the star count, it is smaller than *checkerHelper3Star*. This is because all the tests regarding each level's example solution correctness have already been tested, as well as the detailed tests needing a custom script to get different star counts.

11.14.1.2.1.2 Blackbox testing

In Blackbox testing, the author uses the LMC program as a user. Blackbox testing is great for finding beyond critical errors – it helps fix not only errors but also subtle UI errors that cannot be picked up by the rigorous Whitebox testing.

The author utilises every functionality and feature available on the LMC program's interface to the extreme, attempting to find any function exploit, visual issues, or inconsistencies to log and fix. This is done in every case in the entries of the Blackbox testing tables, as seen below.

Rigorously tested part of the page	Issues encountered	Fixes
Each of the 4 buttons	None	None as there are no encountered issues.
Zooming inwards and outwards	None	None as there are no encountered issues.
Changing the URL configuration/data to valid and invalid characters.	None	None as there are no encountered issues.
Reading all console outputs to see if any previously unnoticed error occurred.	None	None as there are no encountered issues.

Table 11.14—32 - Blackbox testing of the menu page of sprint #3.

Rigorously tested part of the page	Issues encountered	Fixes
Zooming inwards and outwards	None	None as there are no encountered issues.
Changing the URL configuration/data to valid and invalid characters.	None	None as there are no encountered issues.
Reading all console outputs to see if any previously unnoticed error occurred.	None	None as there are no encountered issues.

Table 11.14—33 - Blackbox testing of the manual page of sprint #3.

Rigorously tested part of the page	Issues encountered	Fixes
Toggling the sound effects	None	None as there are no encountered issues.
Toggling light/dark mode	When reloading or changing pages, it takes a while for the style to update to that specified by URL config when not using the dark theme.	Instantiate uRLQuery before any other class to minimise the time spent before changing the style.
Changing theme	When reloading or changing pages, it takes a while for the style to update to that specified by URL config when not using the default theme.	Instantiate uRLQuery before any other class to minimise the time spent before changing the style.
Going back to menu	None	None as there are no encountered issues.
Zooming inwards and outwards	None	None as there are no encountered issues.
Changing the URL configuration/data to valid and invalid characters.	None	None as there are no encountered issues.
Reading all console outputs to see if any previously unnoticed error occurred.	None	None as there are no encountered issues.

Table 11.14—34 - Blackbox testing of the menu page of sprint #3.

Rigorously tested part of the page	Issues encountered	Fixes
Unlocked levels	None	None as there are no encountered issues.
Locked levels	None	None as there are no encountered issues.
Going back to menu	None	None as there are no encountered issues.
Zooming inwards and outwards	None	None as there are no encountered issues.

Changing the URL configuration/data to valid and invalid characters	None	None as there are no encountered issues.
Reading all console outputs to see if any previously unnoticed error occurred.	None	None as there are no encountered issues.

Table 11.14—35 - Blackbox testing of the level selector page of sprint #3.

Rigorously tested part of the page	**Issues encountered**	**Fixes**
Inputting pre-defined and real-time inputs.	Displayed input error `Invalid input - only accepts integers from -999 to 999.` is too big that it is not all displayed in the (real-time) input textbox.	Shortened error message to `Invalid: -999 to 999 only`.
Loading and passing each level	None	None as there are no encountered issues.
Typing scripts and compiling them	None	None as there are no encountered issues.
Executing compiled scripts while using the execution speed control modes	None	None as there are no encountered issues.
Submitting scripts in attempt to satisfy level's objective	None	None as there are no encountered issues.
Reading all console outputs to see if any previously unnoticed error occurred.	`Middleware.prepareLevel` method executes regardless of if campaign or sandbox (should only execute in campaign mode) as seen in **Figure 11.14—31**.	Simple change of comparisons: if `(levelNum == 0){ return; } this.prepareLevel();` → `if (levelNum > 0) { this.prepareLevel(); }` because author forgot to consider the value `-1`.

Table 11.14—36 - Blackbox testing of the simulator page (in both sandbox mode and campaign mode) of sprint #3.

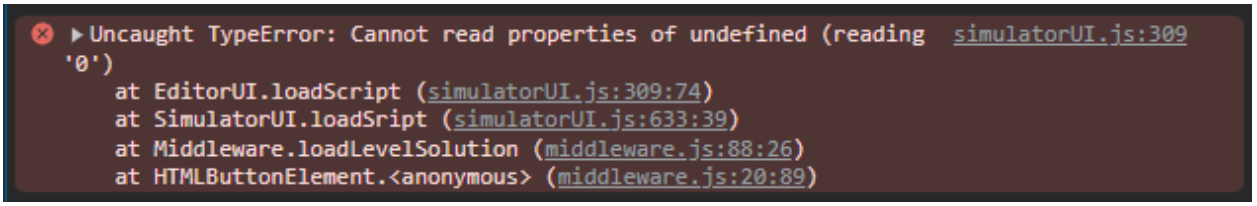


Figure 11.14—31 - previously undetected error, displayed in browser console, found while performing Blackbox testing.

11.14.1.2.1.3 Problems encountered

11.14.1.2.1.3.1 Helper functions

Helper functions were very useful in the prospect of time saving in developing and debugging the tests but did have their fair share of problems.

The author initially called the helper functions, which some can purposely throw *jest* errors using *jest*-supplied `expect` function call. This was to be detected and caught by chaining the jest supplied `.toThrow()` method as seen in example test: `test('as partial', () => { expect(() => { integretyHelper(10, LevelType.partial); }).toThrow; });`. However, when running the tests, all relevant tests (that called the helper functions) passed which means that most likely something is wrong with the helper function.

It turns out that the helper function expects calls do work, but the jest-thrown errors are not picked up in the test code blocks, even with `.toThrow()` or use of try blocks in the test blocks. This means that the errors must be caught in the helper functions directly, hence the use of try blocks in some of the helper functions. Those helper functions return either an empty string if no error is caught, otherwise returning the `.message` attribute of the caught error as a string. Strings are returned instead of error objects to make the testing code as simple as possible.

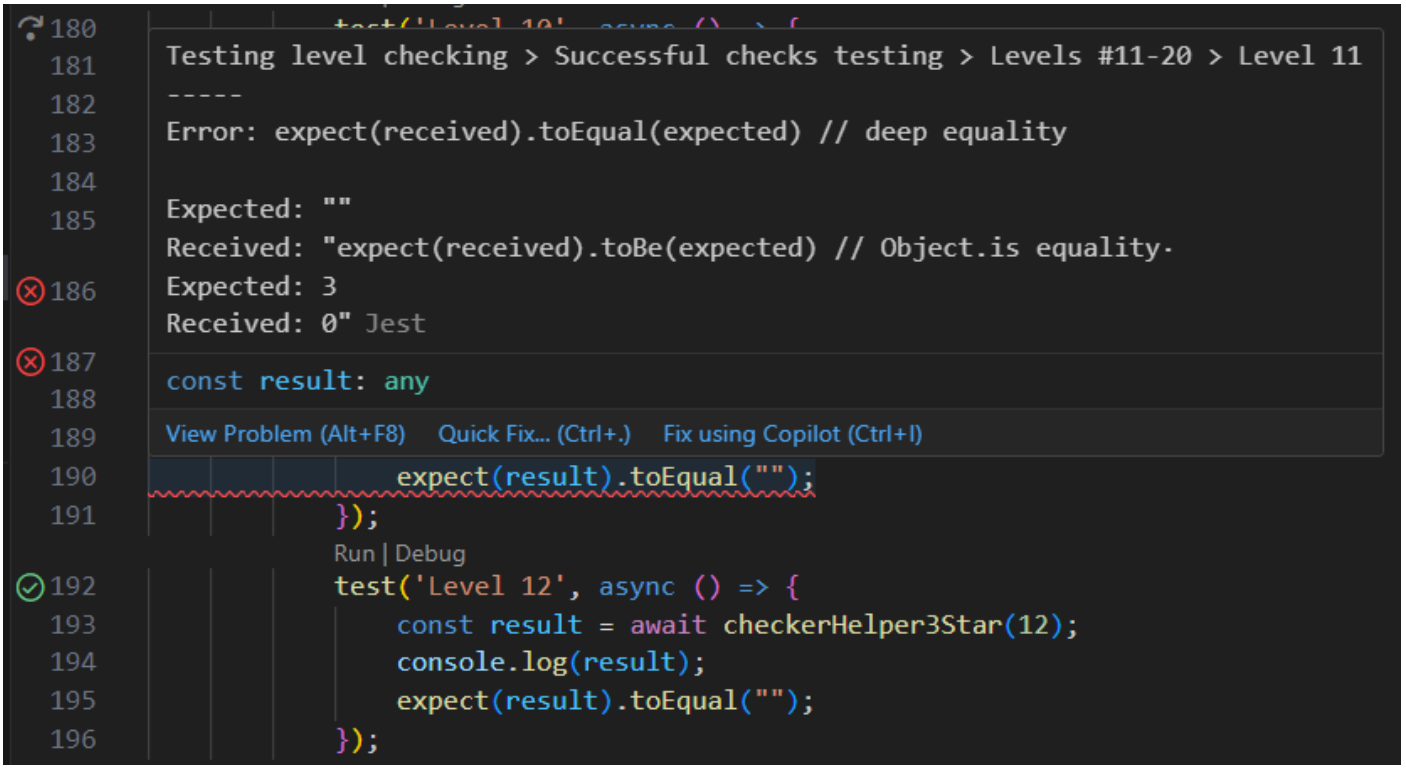


Figure 11.14—32 - example of helper function returning the error message, in TS-compiled ***`levelSolutionCases.test.js`****, above a satisfied test.*

11.14.1.2.1.3.2 Test cases magnitude

It is good to have a big enough number of tests to cover every feature and functionality of the LMC program, but test case files are getting so big that there are more lines of jest testing code than code lines of the actual LMC program itself, despite best efforts to minimise the test cases without compromise. Huge tests take a long time to do in every aspect: test planning, test coding, running and fixing tests, test logging, et cetera.

There is nothing much the author can really do about this, as ensuring the program's stability is a top priority. But the author did try to make some of the tests Blackbox testing instead of Whitebox testing, as this will remove the time required for the test coding part as the author will simply test the LMC program by being a user.

11.14.1.2.1.4 Effects on other tests

Obviously, any new changes to the code may change the outcome of tests of the previous sprints, hence the author running and doing any changes to now-unsatisfied tests.

Which test?	Why did it fail now?	Amendments made
"Sieve of Erastothenes" (self-modifying) program (4th test of 5th test group) of LMC #6 in **`LMCCompatibility.test.ts`**.	Assembly code's execution was too long that it timed out which was proven by adding *`console.log("CYCLE TIMEOUT!")`* to **`vonNeumann.ts`** and *`CYCLE TIMEOUT!`* being printed out in terminal.	Changed expect statement to only expect the first output because that is the most the program does before hitting the cycle count limit: *`expect(await simulator.cycle()).toStrictEqual(["2", "3", "5", "7", "11", "13", "17", "19", "23", "29", "31", "37", "41", "43", "47", "53", "59", "61", "67"]);`* → *`expect(await simulator.cycle()).toStrictEqual(["2"]);`*.
"ascii" program and "ascii table" program (3rd and 4th test, respectively, of 1st test group) of LMC #1 in **`LMCCompatibility.test.ts`**.	The author initially thought it was a problem with the OTC opcode, but it turns out it was the exact same timed-out problem.	Same nature of fix, shortened expected values: *`[" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[", "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~"]`* and *`["32", " ", " ", "33", " ", "!", "34", " ", "\"", "35", " ", "#", "36", " ", "$", "37", " ", "%", "38", " ", "&", "39", " ", "'", "40", " ", "(", "41", " ", ")", "42", " ", "*", "43", " ", "+", "44", " ", ",", "45", " ", "-", "46", " ", ".", "47", " ", "/", "48", " ", "0",`*

		`"49", " ", "1", "50", " ", "2", "51", " ", "3", "52", " ", "4", "53", " ", "5", "54", " ", "6", "55", " ", "7", "56", " ", "8", "57", " ", "9", "58", " ", ":", "59", " ", ";", "60", " ", "<", "61", " ", "=", "62", " ", ">", "63", " ", "?", "64", " ", "@", "65", " ", "A", "66", " ", "B", "67", " ", "C", "68", " ", "D", "69", " ", "E", "70", " ", "F", "71", " ", "G", "72", " ", "H", "73", " ", "I", "74", " ", "J", "75", " ", "K", "76", " ", "L", "77", " ", "M", "78", " ", "N", "79", " ", "O", "80", " ", "P", "81", " ", "Q", "82", " ", "R", "83", " ", "S", "84", " ", "T", "85", " ", "U", "86", " ", "V", "87", " ", "W", "88", " ", "X", "89", " ", "Y", "90", " ", "Z", "91", " ", "[", "92", " ", "\\", "93", " ", "]", "94", " ", "^", "95", " ", "_", "96", " ", "`"]` → `[" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]` and `["32", " ", " ", "33", " ", "!", "34", " ", "\"", "35", " ", "#", "36", " ", "$", "37", " ", "%", "38", " ", "&", "39", " ", "'", "40", " ", "(", "41", " ", ")", "42", " ", "*", "43", " ", "+", "44", " ", ",", "45", " ", "-", "46", " ", ".", "47", " ", "/", "48", " ", "0", "49", " ", "1", "50", " ", "2", "51", " ", "3", "52", " ", "4", "53", " ", "5", "54", " ", "6", "55", " ", "7", "56", " ", "8", "57", " ", "9", "58", " ", ":"]` respectively.

Table 11.14—37 - showing and explaining amendments to previous sprint tests that were made unsatisfied by sprint 3 changes.

11.14.1.3 Justifying deviations from plan and disruptions

11.14.1.3.1 Expected

11.14.1.3.1.1 Level/campaign progression – locked levels

As seen in **Figure 11.14—13**, the author planned to display to the user which levels are locked by replacing the level number with a padlock symbol. Initially, the author thought a SVG, or an image, would have to be used and somehow inserted inside the button's display text (`.innerHTML`) or on top of it. However, when getting towards developing the level selector, the author thought of a much simpler method of achieving the same outcome by taking advantage of the Unicode character set. By using the padlock emoji (*U+1F512*), the author can display a padlock and handle it easily as text instead of media (media would require element grouping and complex CSS code). Demonstration of this can be seen in **Figure 11.14—46** inside sprint #3's **Current state of code**.

```
34              if (levelNum > levelCountProgress) { rowHTML += '<td><button class="square btn mt-3 me-3">&#128274;</button></td>'; }
35              //^ locked levels (hence using padlock emoji "&#128274;")
```

Figure 11.14—33 - TS if-statement code for generating a lock level button, without the unlocked level's functionality, containing the padlock emoji using `&#128274;`.

11.14.1.3.1.2 Deep copying

As mentioned in sprint #3's **Test cases**, a bit of extra code was added in the `LevelChecker` to make `ControlUnit` take a deep copy of the pre-defined inputs list (using `JSON.parse` and `JSON.stringify`) instead of the original. Reasoning and details for this can be found in the second entry of **Table 11.14—30**. This is expected because, besides primitive types, every other data type is passed by reference.

```
86              const simulator:ControlUnit = new ControlUnit( this.userCompiled, JSON.parse(JSON.stringify(currentCase[0])) );
87              //^ JSON methods used for deep copy because is compatible with older browsers unlike 'structuredClone'.
```

Figure 11.14—34 - TS code in `LevelChecker.ts` instantiates the simulator class for testing/examining the user's (compiled) script with `currentcase[0]` as the predefined input argument.

11.14.1.3.2 Unexpected

11.14.1.3.2.1 Wireframe variance

There are some slight variations between the **High-fi** wireframes and the implemented UI in the **Current state of code**. For example, in the high-fi wireframe (specifically **Figure 11.14—12**), the label of the theme selector is inside the drop-down HTML

`select` element (as the default entry/option); meanwhile in the actual LMC product, the label is as a separate HTML `label` element next to the `select` element in a `div` element.

11.14.1.3.2.1.1 Drop-down selector

This change was made because the drop-down's default (first) `option` entry needs to be the default theme to convey to the user the current theme, or otherwise confusion with the user will arise. By doing this, the author helps satisfy HCI principles as the interface will be made more useful than confusing.

```
19          <tr>
20              <td colspan="2">
21              <div class="mt-3 d-flex align-items-center">
22                  <!--^ 'd-flex' lets label be to the dropdown's (select tag's) left instead of above-->
23                  <!--^ 'align-items-center' stops dropdown's height from matching its parent 'div'.-->
24                  <label for="options" class="form-label">Change theme:</label>
25                  <select id="options" class="form-select">
26                      <option value="default">Default (high contrast)</option>
27                      <option value="lowContrast">Low Contrast</option>
28                      <option value="greyscale">Greyscale</option>
29                  </select>
30              </div>
31              </td>
32          </tr>
```

*Figure 11.14—35 - HTML code for the drop-down theme/style selector of the settings page (**`settings.html`**) in sprint #3.*

11.14.1.3.2.1.2 Level buttons

Another such change is the levels of the level selector page. The hi-fi wireframe (**Figure 11.14—13**), displays buttons for levels 1 to 30. However, in the actual implementation, there are two differences.

First, and more visible difference, was that there are only levels 1 to 20 instead of 1 to 30. This is primarily due to the configuration-based problems of sprint #2 (as seen in the sprint's **Justifying deviations from plan and disruptions** section) leading to some of that sprint's aspects and bugs (such as fixing the real-time user inputting) to being postponed to sprint #3. This results in lower priority targets, such as levels 21-30 in this case, to be out of scope and not implemented/develop.

The second, and not as visible difference, was the code behind the buttons. It was first planned to make HTML code per level and put it in **`levelSelector.html`**; however, this contradicts good programming practice by using repetitive code/scripting. This can simply be avoided by using TS code to generate all 20 pieces of HTML for each level with only one piece of HTML as the template for each level button.

```
29      private generateLevelButtons(levelCount:number, levelCountProgress:number){
30          let rowHTML:string = "";
31          for (let levelNum = 1; levelNum < levelCount+1; levelNum++){
32              //: No need for ids for buttons at they are not called at all.
33              //: Adds HTML of one cell of button for a level.
34              if (levelNum > levelCountProgress) { rowHTML += '<td><button class="square btn mt-3 me-3">&#128274;</button></td>'; }
35              //^ locked levels (hence using padlock emoji "&#128274;")
36              else { rowHTML += `<td><button class="square btn mt-3 me-3" onclick="selectLevel(${levelNum})">${levelNum}</button></td>`; }
37              //^ completed levels and next availiable level
38
39              if (levelNum % 10 == 0) {
40                  //* Magnitudes simplier than using a nested loop.
41                  //* Creates ond populates row when fully populated.
42                  const newRow = this.HTMLTable.insertRow(this.HTMLTable.rows.length-1);
43                  //^ inserts row adjacently above bottom row
44                  newRow.innerHTML = rowHTML;
45                  //^ populates said row
46                  rowHTML = "";
47                  //^ clear for next row's inner-HTML
48              }
49          }
50      }
```

*Figure 11.14—36 – screenshot snipped showing the `LevelSelecion.generateLevelButtons` method, from **`levelSelecion.ts`**, in sprint #3.*

By using good programming practice, the author or any other developer can very easily change how the level buttons are displayed without editing the same HTML 20 times.

`LevelSelecion.generateLevelButtons` has a `levelCount` parameter that allows easy configuration of how many level buttons are to be displayed.

11.14.1.3.2.2 Level/campaign progression – current level

The feature of progression through the campaign, by completing incrementing levels for learning in the right order and having a sense of accomplishment, was considered a low priority. This is because it was planned to be done using cookies, which not only requires time to learn and implement, but even more for the legal claimers/disclaimers and ethical considerations as cookies can be used for malicious or other illegal actions.

However, during development, the author realised that cookies were not even necessary in the first place. The level progress can be stored in the URL query (alongside the LMC game's configuration) and the star count does not need to be stored if it gets displayed to the user when completing a level. The only problem is that having both the level progress and the current level identifier (what level is currently played) in the same part of the URL (URL query (W3Schools, 2014b)) will be confusing to both the author (as developer) and the user. Fortunately, the author was easily able to get around this by moving the current level identifier from the URL query to the recently discovered URL fragment (W3Schools, 2010). This is more fitting as URL fragments are usually for pinpointing/anchoring certain parts of the page for the browser to focus on (W3Schools, 2010) just like the current level identifier pinpointing what part of the campaign the user is currently on.

https://jh1662.github.io/LMC_simulator_game/src/frames/simulator.html?0/0/0/13#10

In the quote above, the URL query starts at the question mark and ends at the hashtag where the fragment identifier resides - after the hashtag.

```
84      private campainLevel():number{
85          //* Mere method is not worth being its own class.
86          //* -1 for sandbox, 0 for invalid, 1-30 for level number.
87          let fragmentId:string = window.location.hash.slice(1);
88          //^ the '.slice(1)' removes the hastag denoter from the string
```

Figure 11.14—37 - TS code for fetching the fragment identifier using the `.hash` property

11.14.1.4 Review

11.14.1.4.1 Current state of code

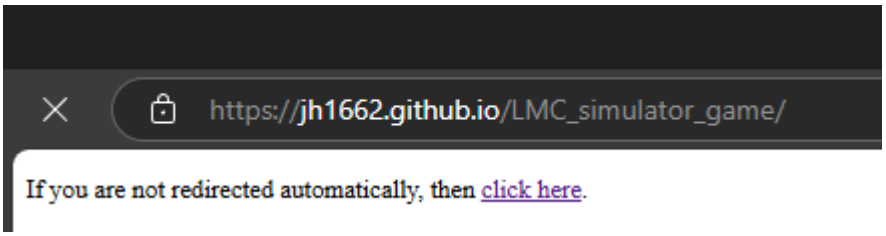


Figure 11.14—38 - When the user only type the website URL without a path.

Note that there is still a path displayed in the URL of **Figure 11.14—38** because the LMC simulator is based on the GitHub repository web hosting and requires the first path "*LMC_simulator_game*" to identify which GitHub repository is wanted to be accessed.

Figure 11.14—39 - showcasing main menu page of sprint #3 in default theme light mode.

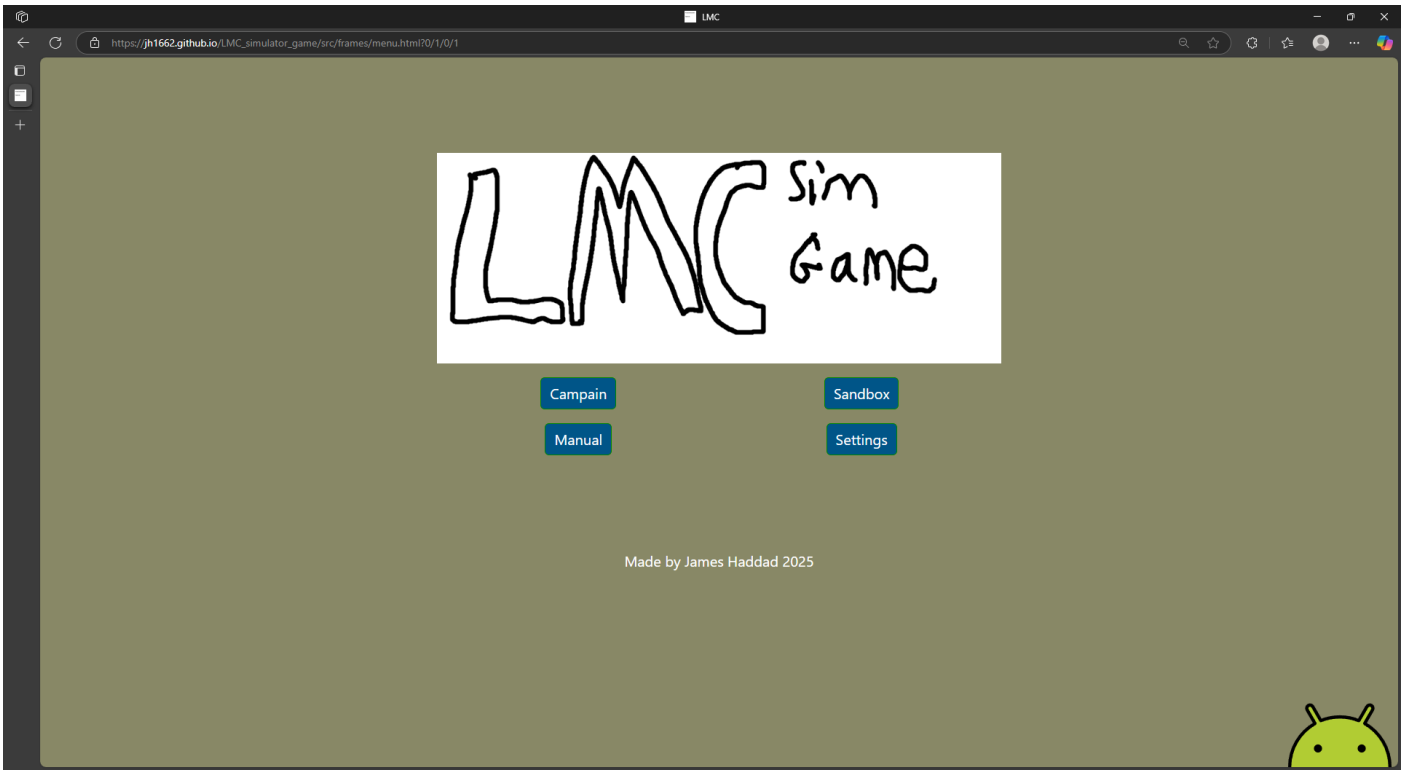


Figure 11.14—40 - showcasing main menu page of sprint #3 in default theme dark mode

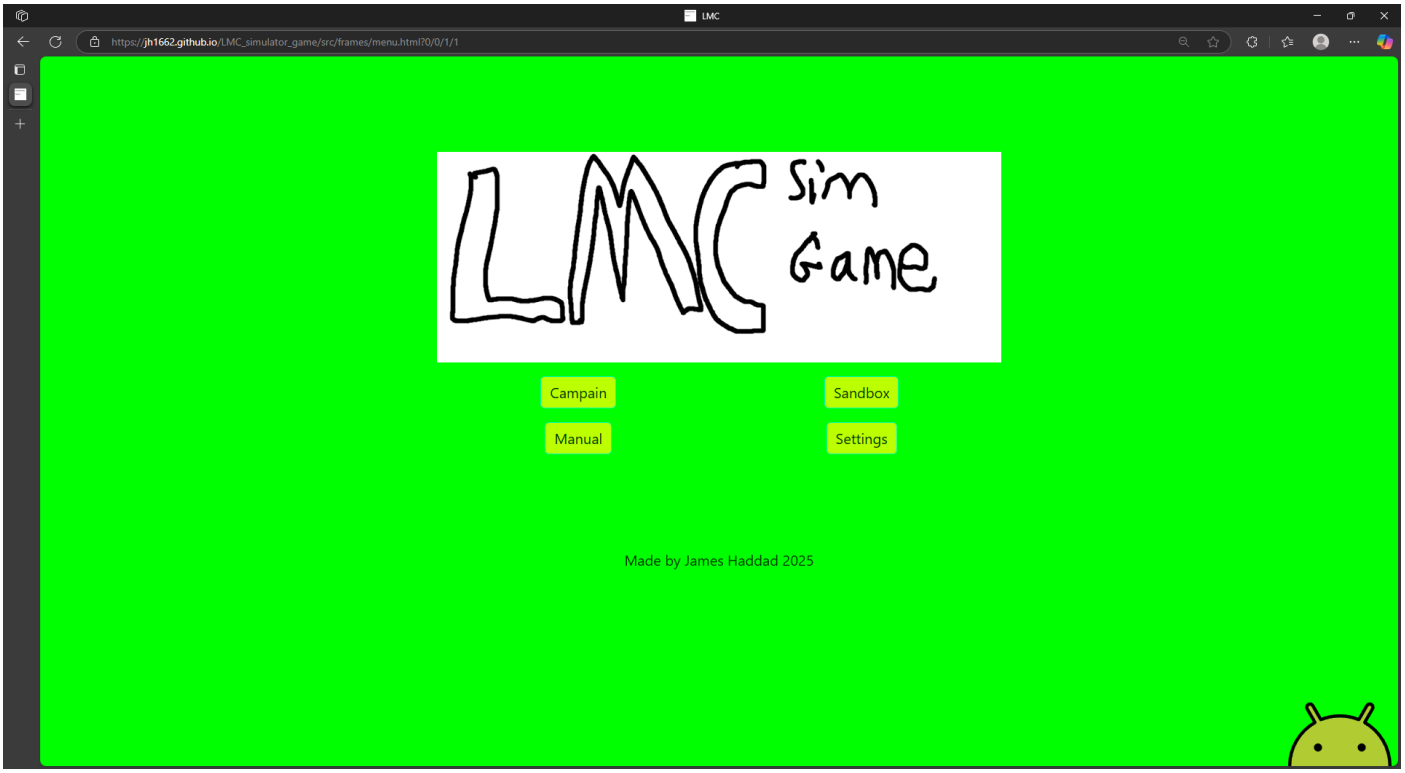


Figure 11.14—41 - showcasing main menu page of sprint #3 in high contrast theme light mode

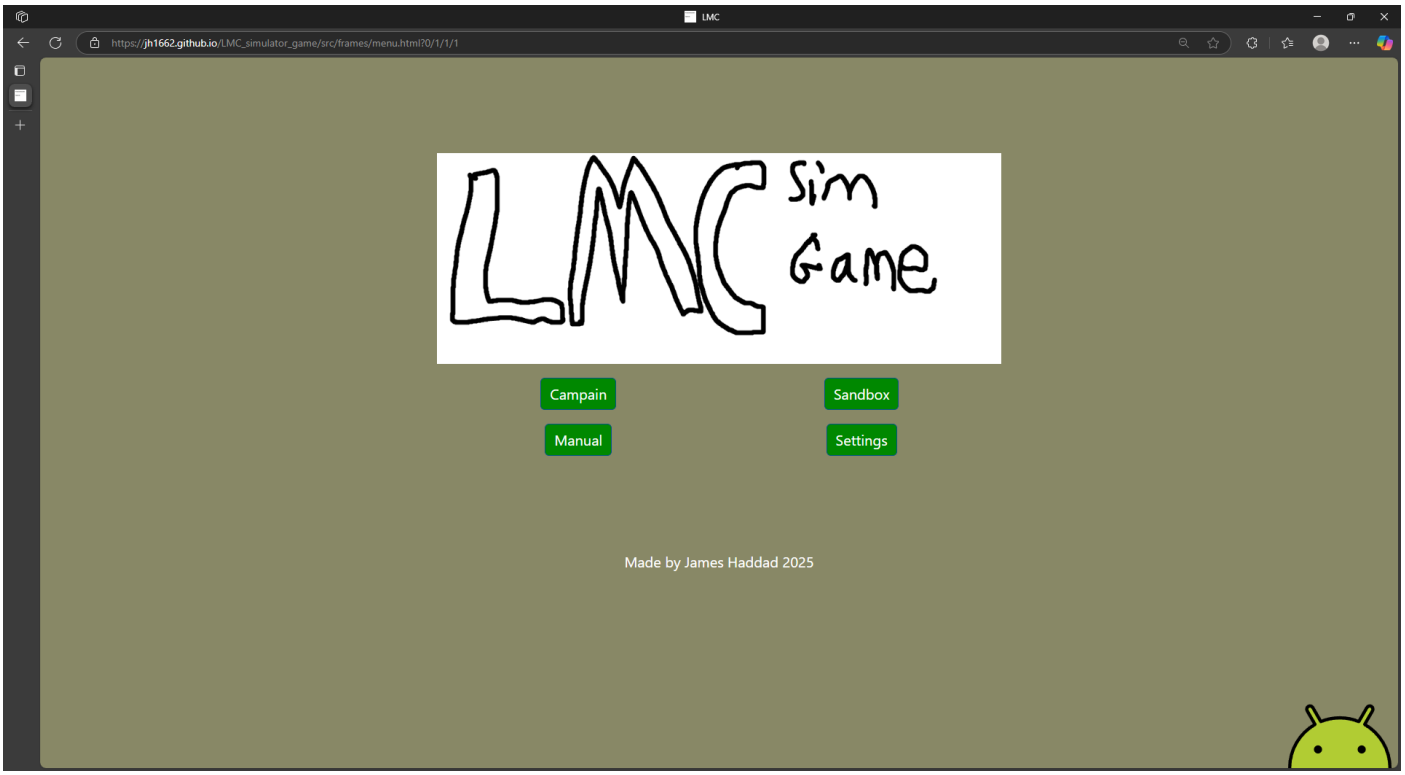


Figure 11.14—42 - showcasing main menu page of sprint #3 in high contrast theme dark mode

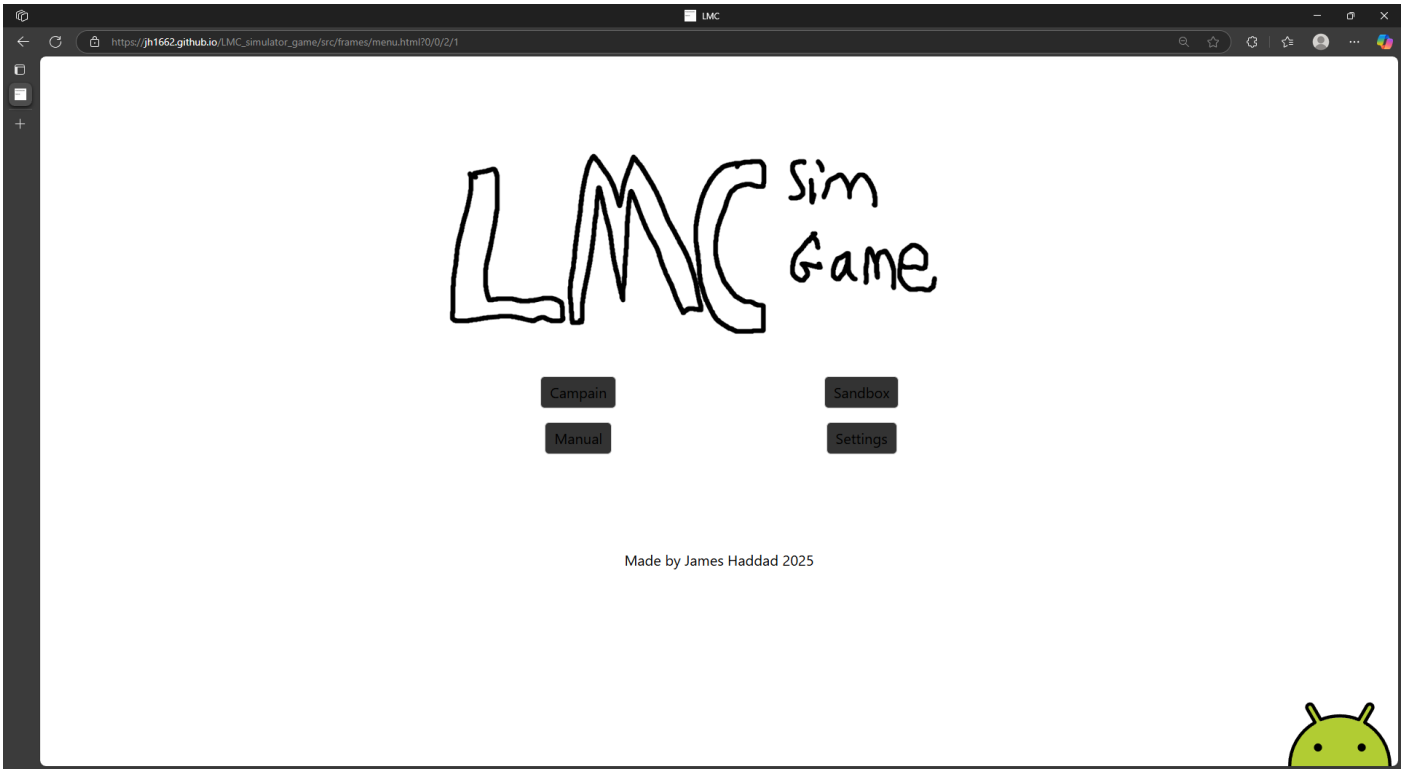


Figure 11.14—43 - showcasing main menu page of sprint #3 in greyscale theme light mode

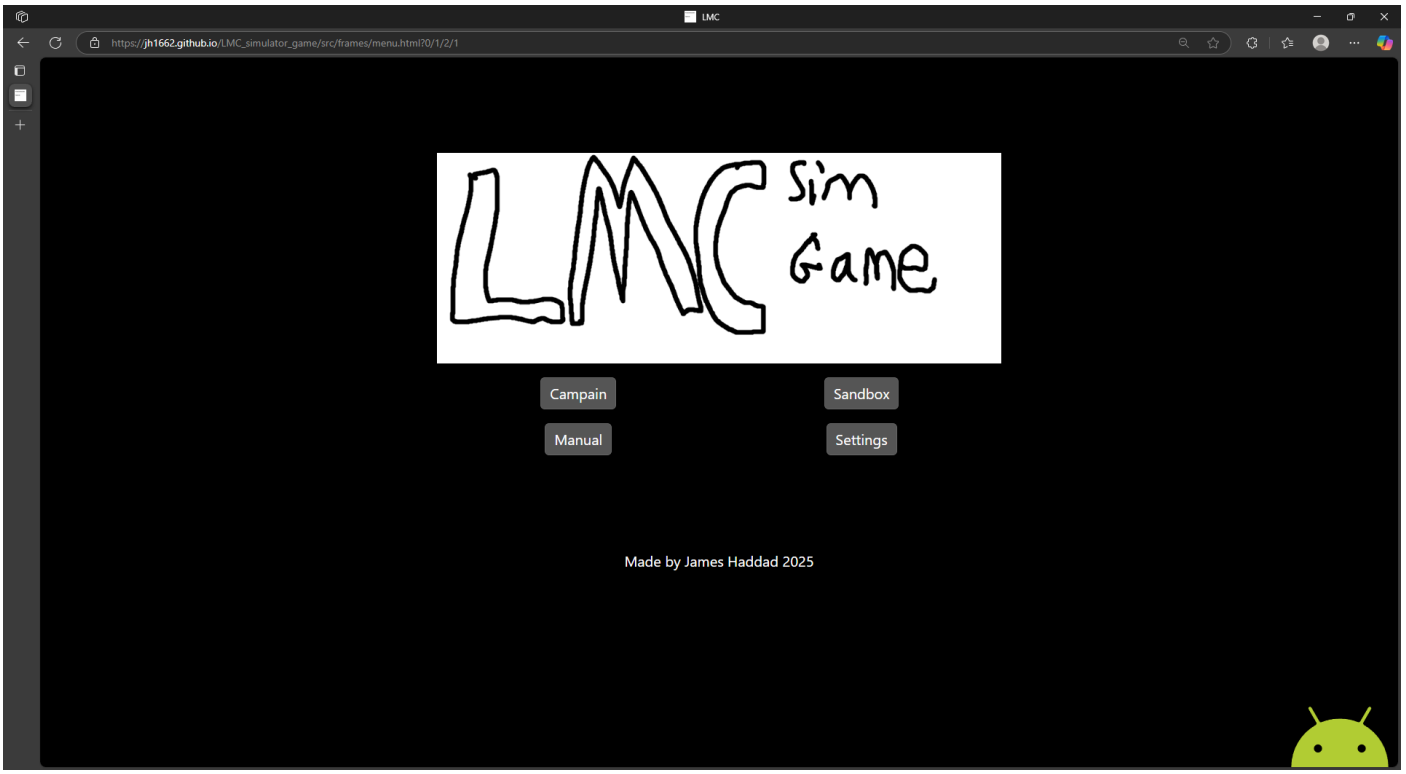


Figure 11.14—44 - showcasing main menu page of sprint #3 in greyscale theme dark mode

Instruction Manual

Little Man Computer (LMC) is a visual simulator of a simplied version of the Von Neumann architecture CPU (computer processing). Below is a quick guide that will assist helping inexperienced users whether they are stuck on a campain level or just want to play around in the sandbox mode.

Instruction Set

Name	Mnemonic	Code	Description
Addition	ADD	1XX	Add memory cell address' value to accumulator's value
Subtraction	SUB	2XX	Subtract memory cell address' value from accumulator's value
Store from Accumulator	STA	3XX	Store accumulator's value in memory cell address
Left Shift	LSH	401	Shift the accumulator value's base-10 digits of significance to the left by one digit.
Right Shift	RSH	402	Shift the accumulator value's base-10 digits of significance to the right by one digit.
Load to Accumulator	LDA	5XX	Load memory address's value to the accumulator (becomes the new accumulator's value).
Branch Always	BRA	6XX	Branch – change PC's value to – the address value (regardless of accumulator's value).
Branch if Zero	BRZ	7XX	Branch – change PC's value to – the address value if accumulator's value is zero.
Branch if Positive	BRP	8XX	Branch – change PC's value to – the address value if accumulator's value is positive or zero (not negative).
Input	INP	901	Takes user's or predefined input and store it as accumulator's value.
Output	OUT	902	Outputs from accumulator's value (as integer).
Output as Character	OCT	903	Outputs from accumulator's value as an ASCII character
Halt	HLT	0	Stops program
Data location	DAT	N/A	Compile data into memory before program starts (can be used as variables)

Script Editor

Use the following keys to navigate:

- Tab key - Go down a line if in an operand box.
- Space key - Go down a line if in an operand box.
- Enter key - Go back to the top line if on the bottom line.
- Downwards key - Go back to the top line if on the bottom line.
- Upwards key - Go to the bottom line if on the top line.

Typing on the bottom code line/row will generate a new empty one underneath.

Registers

Name	Description	Valid range
Program Counter (PC)	Dictates where the simulator fetches its next instruction (which RAM cell).	0 to 99
Memory Instruction Register (MIR)	Gathered from the first digit of the fetched instruction from memory.	0 to 9
Memory Address Register (MAR)	Gathered from the last two digits of the fetched instruction from memory.	0 to 99
Accumulator	A form of immediate storage for processor operations such as adding, outputing, inputing, etc.	-999 to 999

Simulator Controls

Controls relavent to simulator

- Run - Runs the compiled program as seen in the RAM table (requires valid compilation first)
- Compile - Attempts to compile script and store in memory.
- Stop - Stops currently executing assembly program.
- Reset - Reloads page to rid off unexpected errors (and current script, so think twice before reloading).
- Toggle Display - Switches between Little Man action display and the current campain level objective (does not really matter if in sandbox mode).
- Toggle Execution Mode - Switches between continuously running each cycle, but with configurable speeds, or only execute next cycle when "New Cycle" button is pressed.

Arithmetic Logical Unit status

[1] [2] [3]

1. Flow:

Symbol	Meaning	Explaination
_	Normal	Initial result in range (-999 to 999).
^	Overflow	Result of operation exceeded 999 and thus resets back from -999.
V	Underflow	Result of operation exceeded -999 and thus resets back from 999.

   None of these indicate error with the simulator - fully informative.

2. Operation - Last operation performed in the Arithmetic Logical Unit.
3. Result - Result of the last operation performed in the Arithmetic Logical Unit.

Legal claimers

Created by James Haddad 2025

The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Vectors and icons by SVG Repo

Figure 11.14—45 - showcasing the manual page of sprint #3.

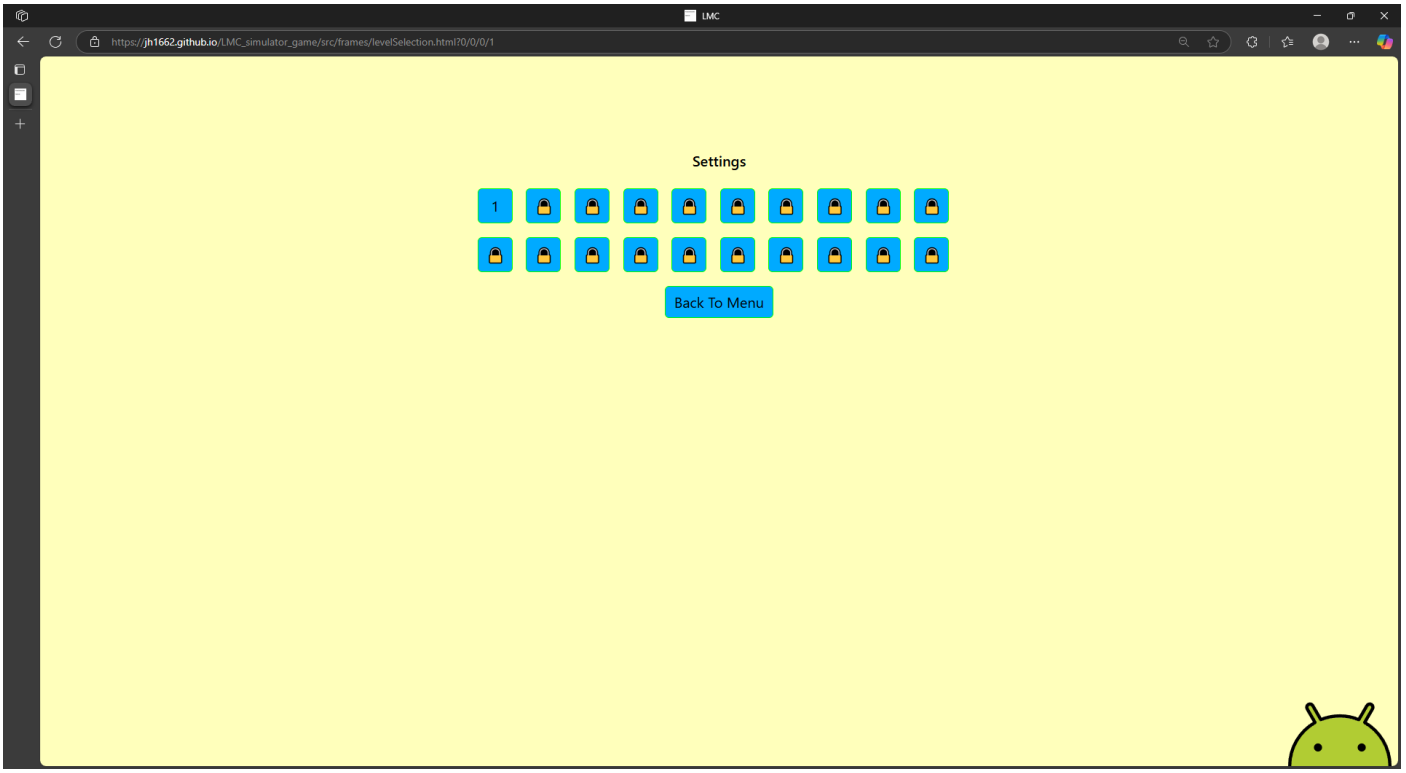


Figure 11.14—46 - showcasing the level selector page of sprint #3.

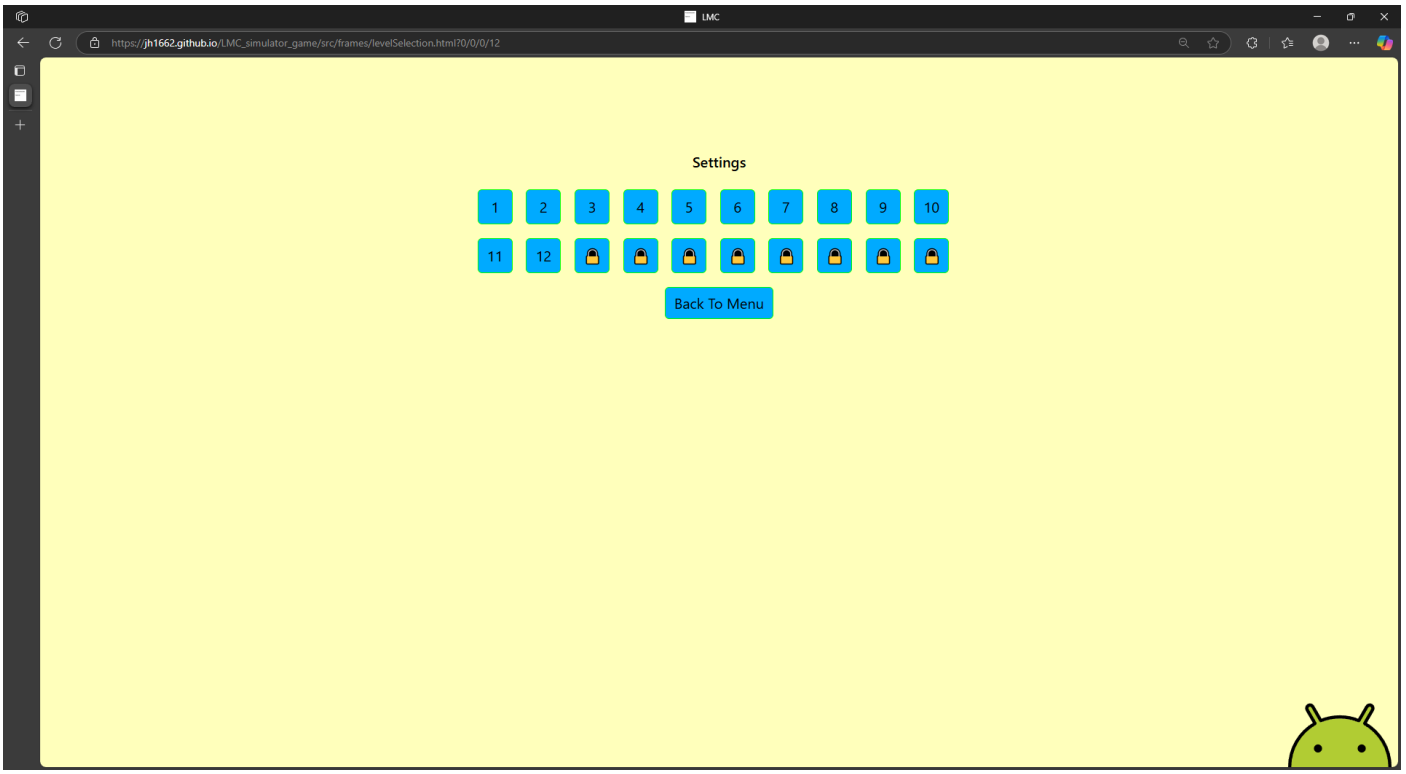


Figure 11.14—47 - showcasing the level selector page of sprint #3 but some levels have been completed (specifically 11),

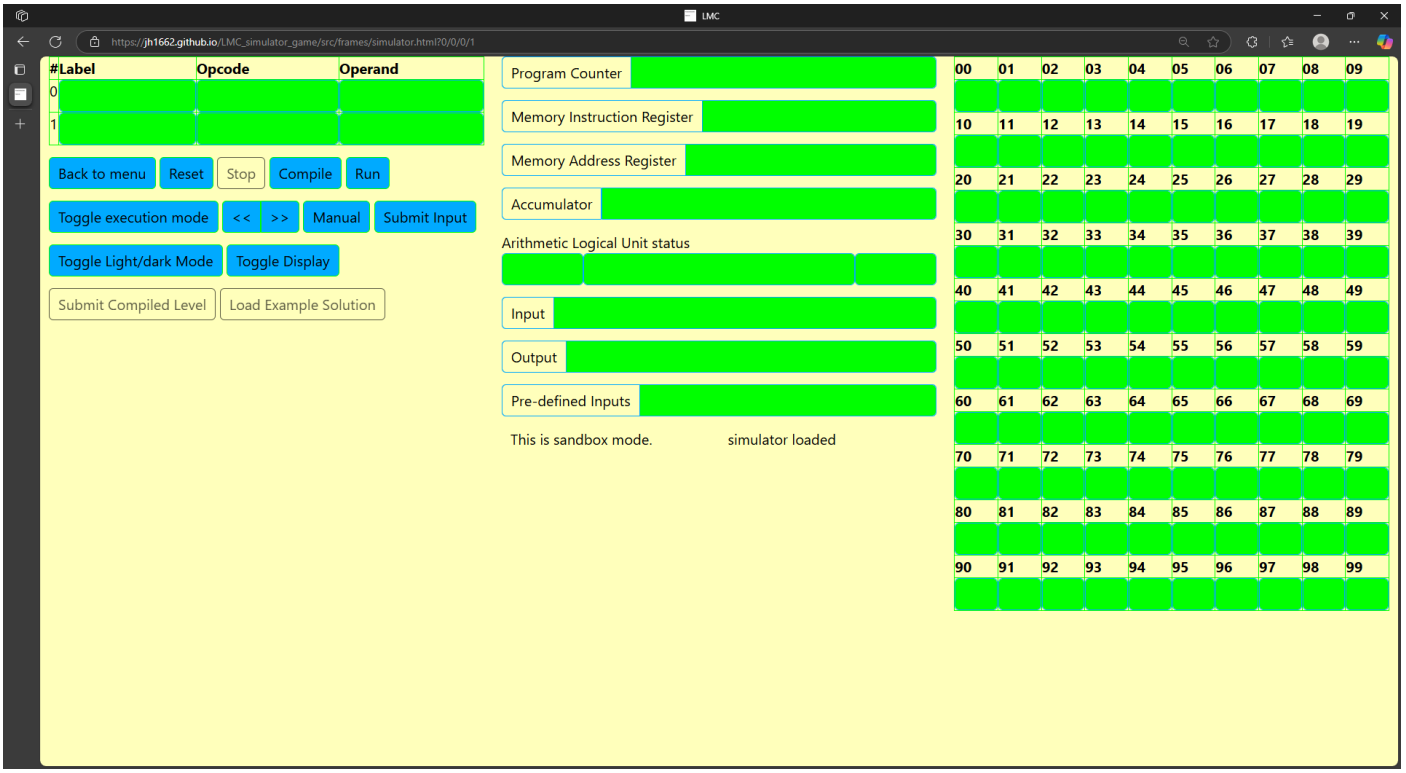


Figure 11.14—48 - showcasing the simulator, in sandbox mode, in sprint #3.

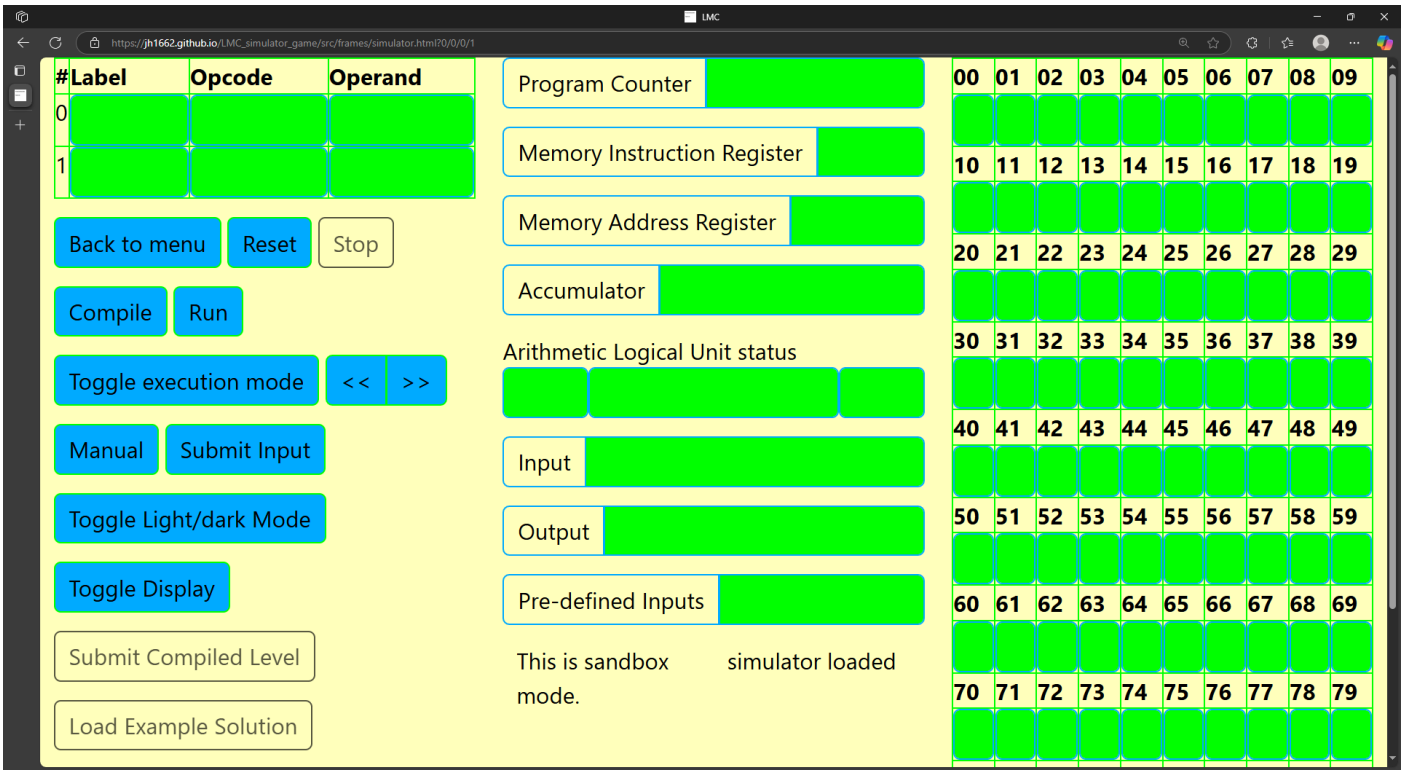


Figure 11.14—49 - showcasing the simulator page, in sandbox mode, in sprint #3 but at 125% zoom.

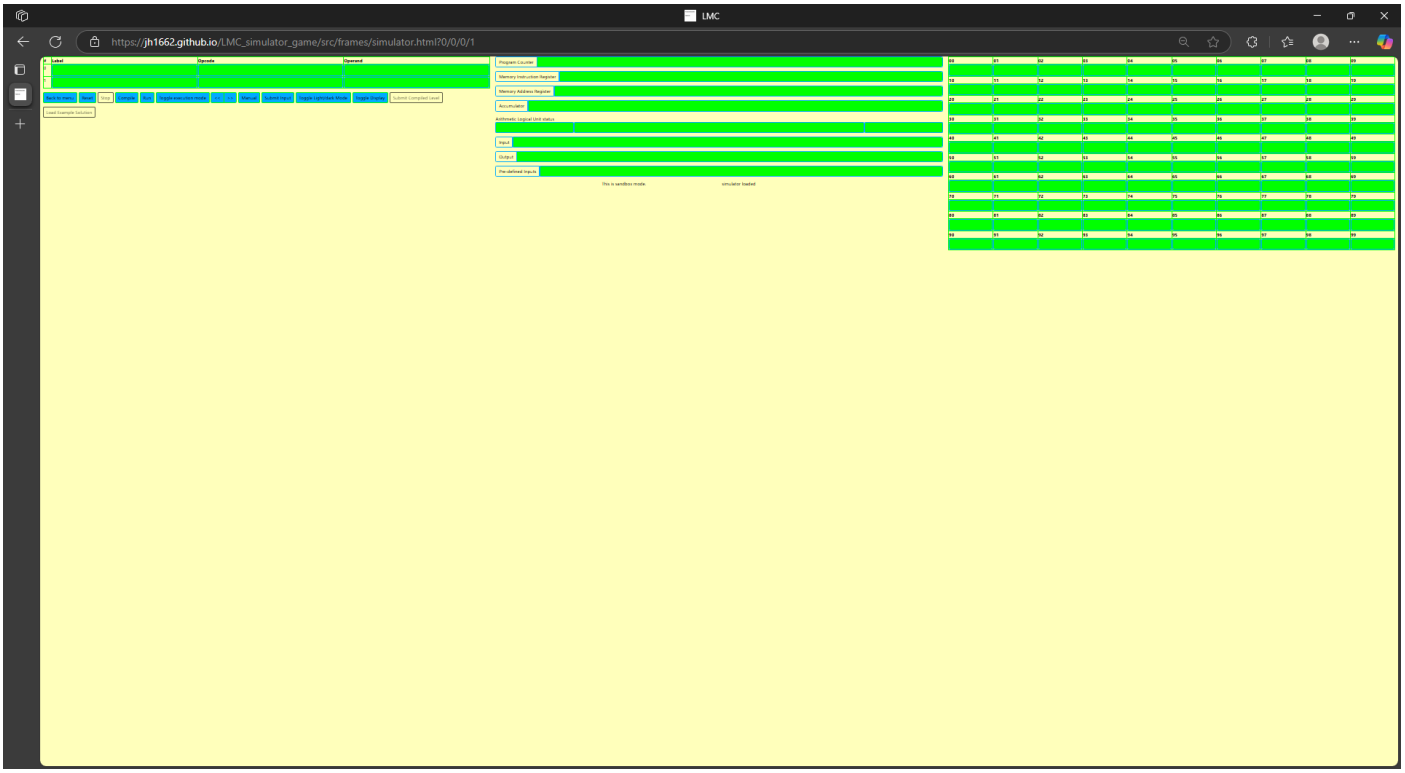


Figure 11.14—50 - showcasing the simulator, in sandbox mode, in sprint #3 but at 25% zoom.

Figure 11.14—51 - showcasing the simulator, in campaign mode at the first level (tutorial level), in sprint #3.

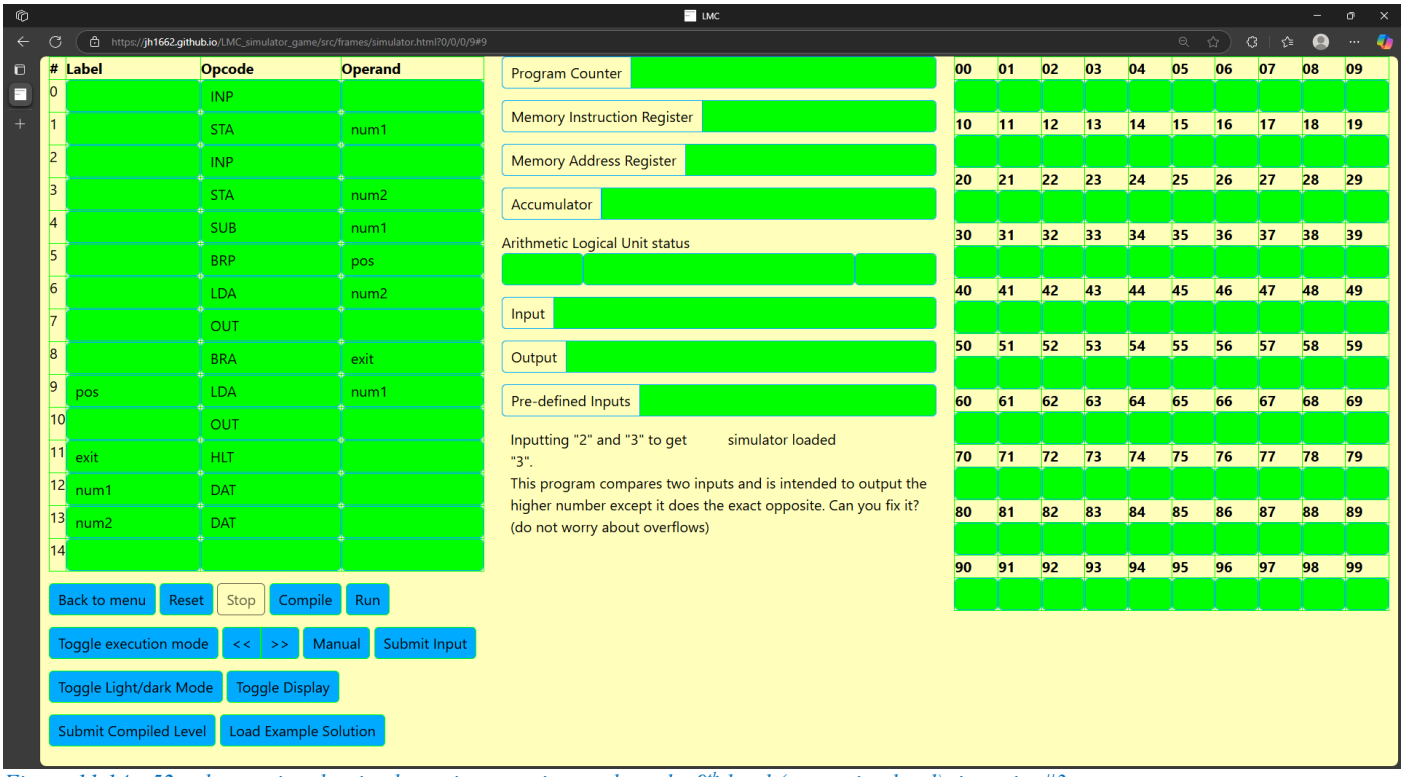


Figure 11.14—52 - showcasing the simulator, in campaign mode at the 9th level (correcting level), in sprint #3.

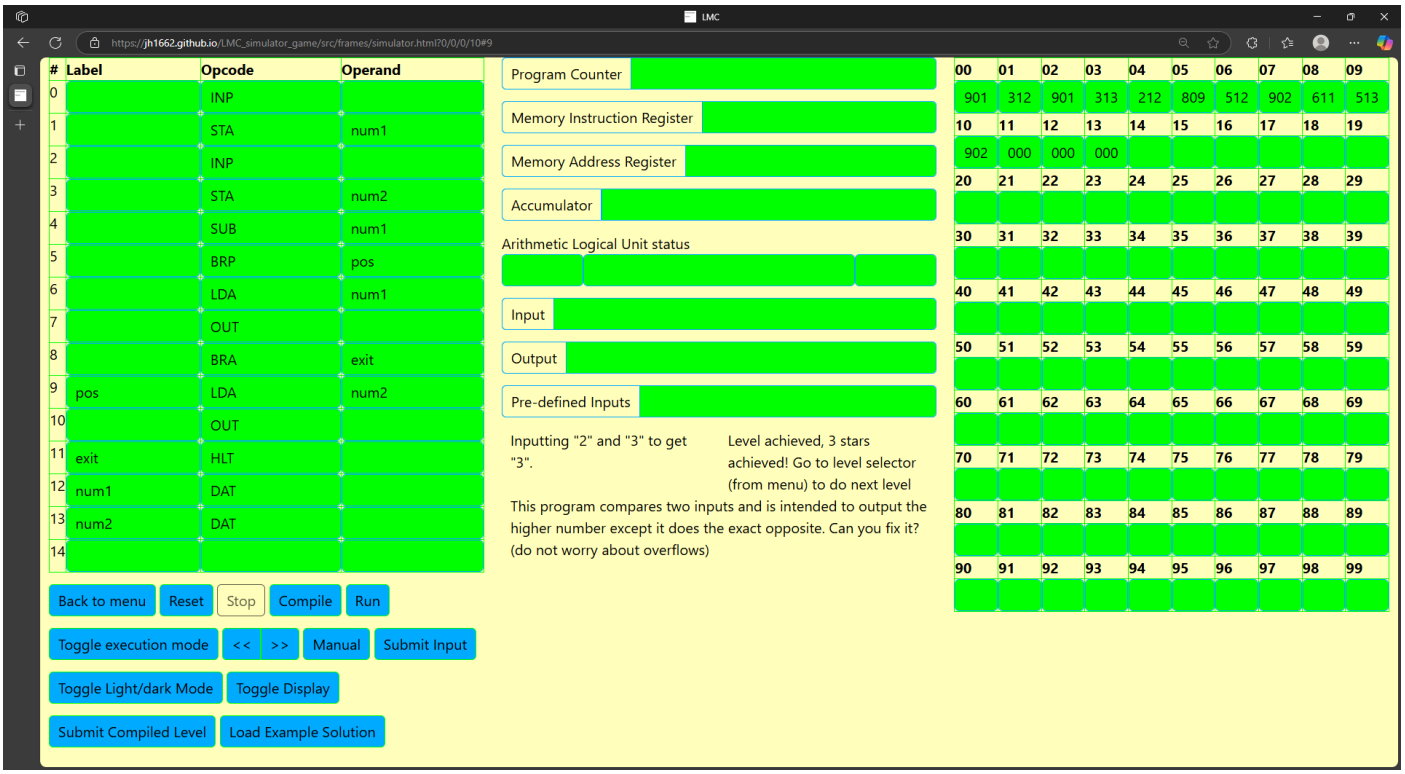


Figure 11.14—53 - showcasing the simulator, in campaign mode at the 9th level (correcting level), in sprint #3 but level is completed at 3 stars.

Figure 11.14—54 - showcasing the simulator, in campaign mode at the 9th level (correcting level), in sprint #3 but level is completed at 2 stars.

Figure 11.14—55 - showcasing the simulator, in campaign mode at the 9th level (correcting level), in sprint #3 but level is completed at 1 star.

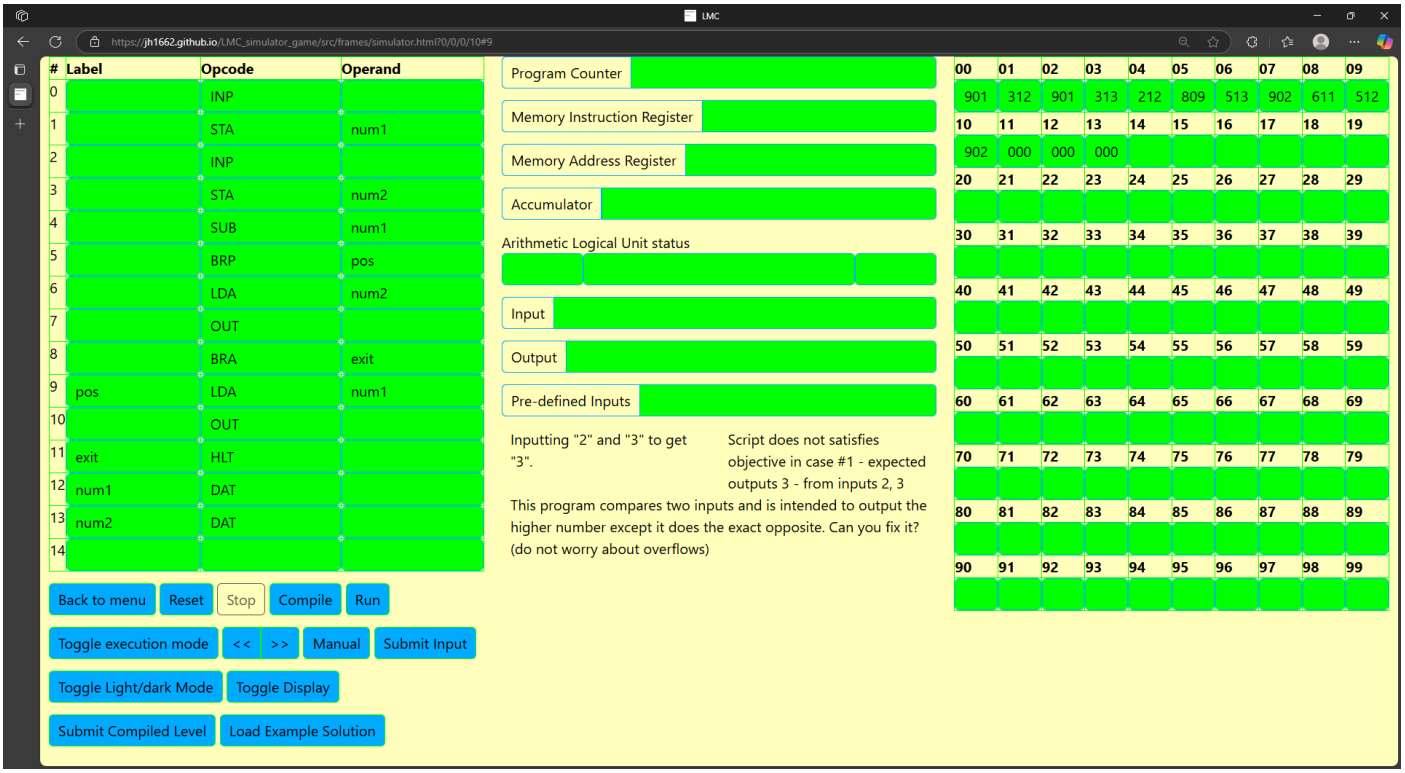


Figure 11.14—56 - showcasing the simulator, in campaign mode at the 9th level (correcting level), in sprint #3 but level submission did not satisfy level's objective.

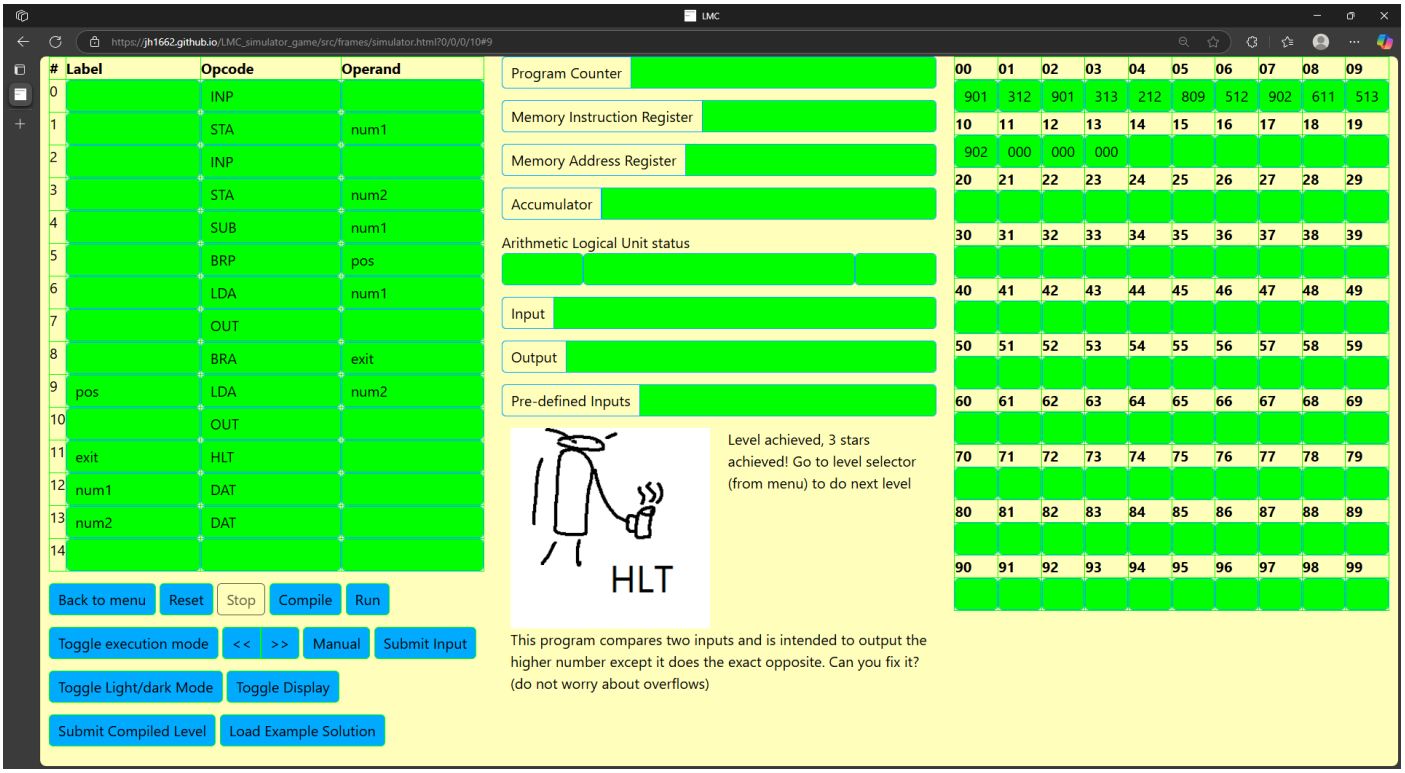


Figure 11.14—57 - showcasing simulator page, in sprint #3, but display has been toggled ('Toggle Display' button pressed).

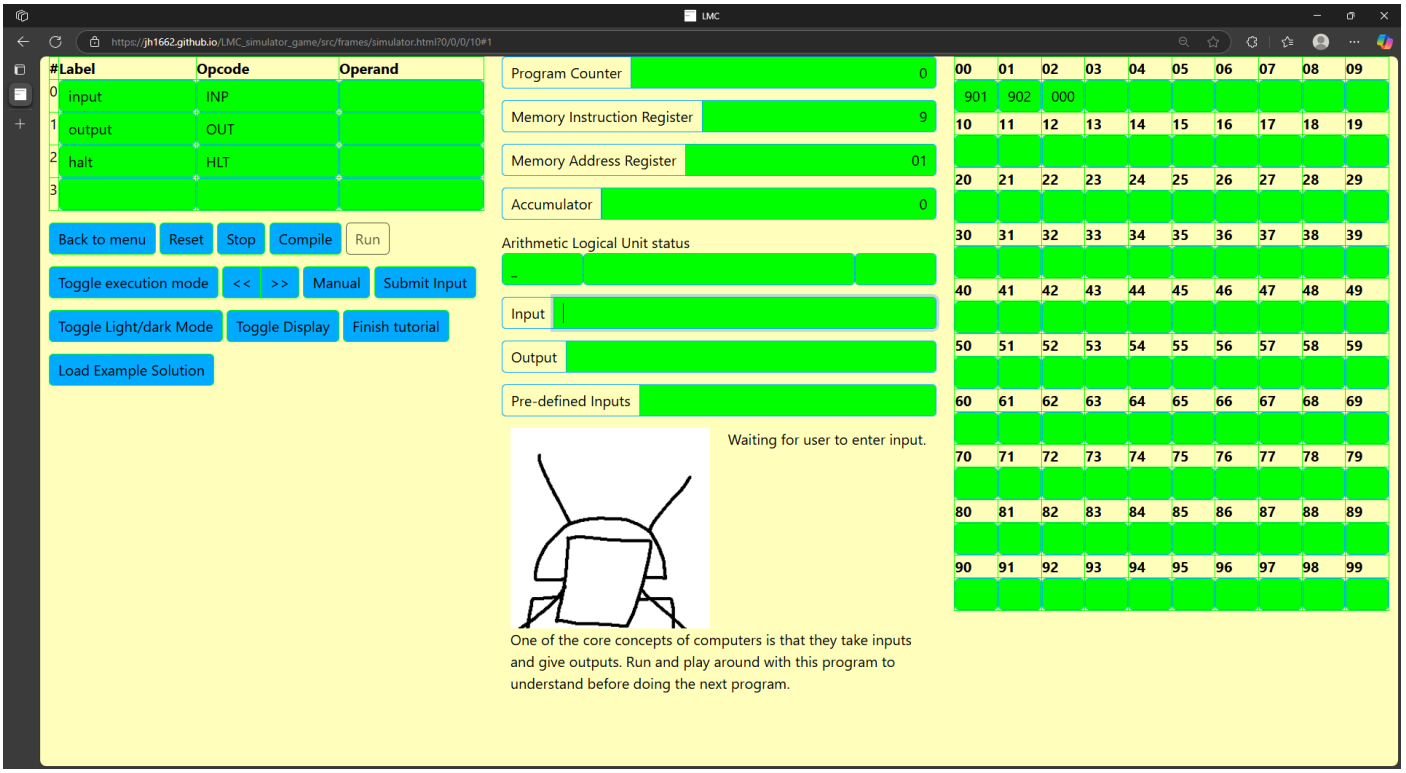


Figure 11.14—58 - showcasing simulator page, in sprint #3, but simulator is waiting for user to enter input.

11.14.1.4.2 What went well in this project (all sprints)

Sprint #3 went much better than expected, especially when managing to both implement at least all "Should have" priorities (as seen in **Table 11.14—1**, **Table 11.14—2**, **Table 11.14—6**, et cetera) and fix errors and incomplete sections from sprint #2 (such as fixing the real-time user inputting).

The author has demonstrated useful critical thinking and improvisation skills in finding simpler alternatives to similar results, such as with the cookies alternative (in sprint #3's **Justifying deviations from plan and disruptions** section).

11.14.1.4.3 What would be better and used for a next project (all sprints)

Because this is the last of the last of the 3 sprints, this section will talk (of the 3 sprints) about what would be better and used if the author were to continue upon the project (without a time constraint) or develop projects of a similar nature – being linked in the **Conclusion Chapter**. Each following sub-sections represents a point of improvement.

11.14.1.4.3.1 TS support

Especially in sprint #2, a lot of things went wrong due to different TS supporting tools being limited to certain parts, such as (*ts-node*). The author has underestimated the complexity of the TS configuration needed to get it working on both Node.js (for jest testing) and the browser. Such an example was being unaware of the fact that there were different JavaScript language specifications (also known as module systems), as the author assumed that only the different browsers' way of JS interpretation was worthy of consideration. More information can be found in the **Review** section of sprint #2.

This lack of information is mostly because while the author has experience in simple TS applications running exclusively in Node.js runtime environment, he has not created a TS-based full-stack (backend, middleware, and frontend) website application. The author also attributes this difficulty to never encountering a programming language that requires so much configuration (relative to actual coding the program) than any other languages, including but not limited to: C#, C++, Bash, Python, JS, Prolog, PostgreSQL, MongoDB, et cetera.

However, the author did manage to persevere in the end and heavily benefited by learning new knowledge and skills such as full-stack TS development, deep configuration, type declarations, use of frameworks, etc.

11.14.1.4.3.2 Testing

One of the most time-consuming parts for the project is the testing of each sprint. The author did try to use Blackbox testing more to help with this, as explained in the **Test cases** section, but still remained one of the most time-consuming activities.

Because of this, the author believes the only way to reduce the time intensity of the testing as a whole is to rethink the methodology of the testing. In the project, the author dedicated a group of tests for applicable functionality and features. For example, the figure below shows some rigorous tests for the extremity of invalid DAT opcodes' corresponding operand.

```
                Run | Debug
516             describe("Invalids",() => {
                    Run | Debug
517                 test("DAT refering an non-existant label",() => {
518                     const preCompiled:string[][] = [["", "DAT", "LABEL"]];
519
520                     const compiler:Compiler = new Compiler();
521                     const compiled:number[]|string = compiler.validateAndCompile(preCompiled);
522
523                     expect(compiled).toStrictEqual([-1]);
524                     expect(compiler.getMessage()).toStrictEqual('Missing label error at the oper
525                 });
                    Run | Debug
526                 test("DAT with the highest under-range number",() => {
527                     const preCompiled:string[][] = [["", "DAT", "-1000"]];
528
529                     const compiler:Compiler = new Compiler();
530                     const compiled:number[]|string = compiler.validateAndCompile(preCompiled);
531
532                     expect(compiled).toStrictEqual([-1]);
533                     expect(compiler.getMessage()).toStrictEqual('operand not expected error at t
534                 });
                    Run | Debug
535                 test("DAT with the lowest over-range number",() => {
536                     const preCompiled:string[][] = [["", "DAT", "1000"]];
537
538                     const compiler:Compiler = new Compiler();
539                     const compiled:number[]|string = compiler.validateAndCompile(preCompiled);
540
541                     expect(compiled).toStrictEqual([-1]);
542                     expect(compiler.getMessage()).toStrictEqual('operand not expected error at t
543                 });
                    Run | Debug
544                 test("DAT with a decimal instead of an integer",() => {
545                     const preCompiled:string[][] = [["", "DAT", "5.5"]];
546
547                     const compiler:Compiler = new Compiler();
548                     const compiled:number[]|string = compiler.validateAndCompile(preCompiled);
549
550                     expect(compiled).toStrictEqual([-1]);
551                     expect(compiler.getMessage()).toStrictEqual('non-alphanumeric error at the c
552                 });
553             });
```

Figure 11.14—59 – testing the extremities of invalid opcodes corresponding to the DAT *opcode from* **`compiler.test.ts`**.

Figure 11.14—59 only shows invalid operand tests. Combine those with the valid *`DAT`* operand tests, and that totals 11 tests for testing one single opcode. The code of the 11 tests can be found in the *`Specifically the Data Location opcode (DAT)`* describe-statement of **`compiler.test.ts`** and details, regarding any corresponding amendments and (expected and actual) results, can be seen in **Table 11.13—28** of sprint #2.

In situations, such as in **`levelSolutionCases.test.ts`**, helper functions would help a lot, but not for test files that have many kinds of testing which would result in several helper functions with their added complexity.

For next time the author does another project, the author would try to minimise the different kind of tests as much as the feature/functionality coverage remains the same. That way, the usability and effectiveness of helper functions will rise significantly.

11.14.1.4.3.3 Planning tools

Besides testing and prior research, time consumed besides the sprint codebase development is planned. As described in **11.14.1.1.1.4**, the author described and used more effective planning tools to make the plans faster, such as using scripting to create a flowchart then manually adding the shapes and arrows and moving them all every time an addition in needed.

11.14.1.4.3.4 Programming practice

To ensure code quality, as planned, the author has stuck to OOP with code having traits of: modularity, reusability, loose coupling, high cohesion, easy-to-understand, configurable, et cetera.

These traits have been achieved as specified in the table below.

Programming practice/traits	How was it achieved
OOP	Using classes, instead of functions, in the codebase of the product, except for the jest test files but with good reasoning explained in sprint #3's **Test cases** section.
modularity and reusability	Achieved by doing the other traits, especially loose coupling.
loose coupling	Due to the nature of loose coupling, it is impossible to achieve 100% loose coupling (or none of the classes will communicate with each other at all) but the author did minimise it by use of: middleware, aggregation over composition where possible, the ability to test classes individually (as seen in the jest `.test.ts` files) thanks to optional constructor parameters, resilience to change, et cetera.
high cohesion	Achieved as seen using multiple classes in some TS files. Such TS file is `vonNeumann.ts` with `ControlUnit` class aggravating same-file classes for different parts of the von Neumann architecture, such as `RAM` for the memory cells functionality and storage, `Registers` for handling register values (like the accumulator) functionality and storage, `ALU`, and `IO`. Another such example is in `simulatorUI` with its same-file aggravated classes.
easy-to-understand	Code comments (as seen mainly in the TS and CSS files) help describe purpose and intentions of code lines and blocks. Meanwhile, regions (by using the `#region` and `#endregion` syntax) help organise large TS and HTML files into manageable and distinguishable pieces. Because of this, other developers will easily be able to comprehend the code and understand the author's original intentions.
configurable	Tied with modularity and reusability, the author has made code to be easily updated/configured the code, such as with the use of default parameters and fields with descriptive names like `ControlUnit.cycleCountLimit`.

Table 11.14—38 - describing how the planned programming practice/traits were achieved at project completion.

11.14.1.4.4 Changes to previous sprints

11.14.1.4.4.1 To sprint #1 code

Some methods in ControlUnit, of vonNeumann.ts, have been made asynchronous (using the `async` syntax) to allow the simulation execution of the compiled script to wait until it gets its needed real-time input from the user.

There was no other significant change in the sprint #1's code, as sprint #3 is more towards improving on sprint #2 directly.

11.14.1.4.4.2 To sprint #2 code

11.14.1.4.4.2.1 Handling CSS

Firstly, `default.css` has been improved on from declaring colouring for light and dark modes (`:root` and `[data-theme="dark"]` respectively) for 3 style themes and light/dark modes for each one of them, totalling to 6.

```
:root {
    --text-color: #000;
    --background-color: #FFB;
    --secondary-color: #0AF;
    --primary-color: #0F0;
}

[data-theme="dark"] {
    --text-color: #FFF;
    --background-color: #886;
    --secondary-color: #058;
    --primary-color: #080;
}
```

Figure 11.14—60 - variable declarations in `default.css` of sprint #2.

```css
/*#region Theme Configuration*/
[data-theme="default-light"] {
    /** originally the light mode in sprint 2*/
    --background-color: #FFB;
    --text-color: #000;
    --primary-color: #0F0;
    --secondary-color: #0AF;
}
[data-theme="default-dark"] {
    /** originally the dark mode in sprint 2*/
    --background-color: #886;
    --text-color: #FFF;
    --primary-color: #080;
    --secondary-color: #058;
}
[data-theme="lowContrast-light"] {
    --background-color: #0F0;
    --text-color: #030;
    --primary-color: #0FB;
    --secondary-color: #BF0;
}
[data-theme="lowContrast-dark"] {
    --background-color: #886;
    --text-color: #FFF;
    --primary-color: #058;
    --secondary-color: #080;
}
[data-theme="greyscale-light"] {
    --background-color: #FFF;
    --text-color: #000;
    --primary-color: #CCC;
    --secondary-color: #333;
}
[data-theme="greyscale-dark"] {
    --background-color: #000;
    --text-color: #FFF;
    --primary-color: #666;
    --secondary-color: #555;
}
/*#endregion*/
```

*Figure 11.14—61 - variable declarations in **`default.css`** of sprint #3.*

Secondly, how CSS styles are interpreted is completely different in sprint #3. In sprint #2, light mode is used by default and can be toggled to dark mode until either toggled back or the page is refreshed. In sprint #3, CSS styles is interpreted by the URL, more specifically the query part as seen after the question mark in the example URL:

https://jh1662.github.io/LMC_simulator_game/src/frames/menu.html?0/0/0/1

The *`URLQuery`* class, from **`URLQuery.ts`**, interprets the string “0/0/0/1”. It deduces the CSS theme (*`default`*, *`lowContrast`*, or *`greyscale`*) from the third number, of the query string, and whether to use dark mode from the second number.

Instead of just a single button in the simulator page to change how CSS is interpreted. An entire page (**`settings.html`**) is dedicated to this, alongside toggling sound effects as seen below:

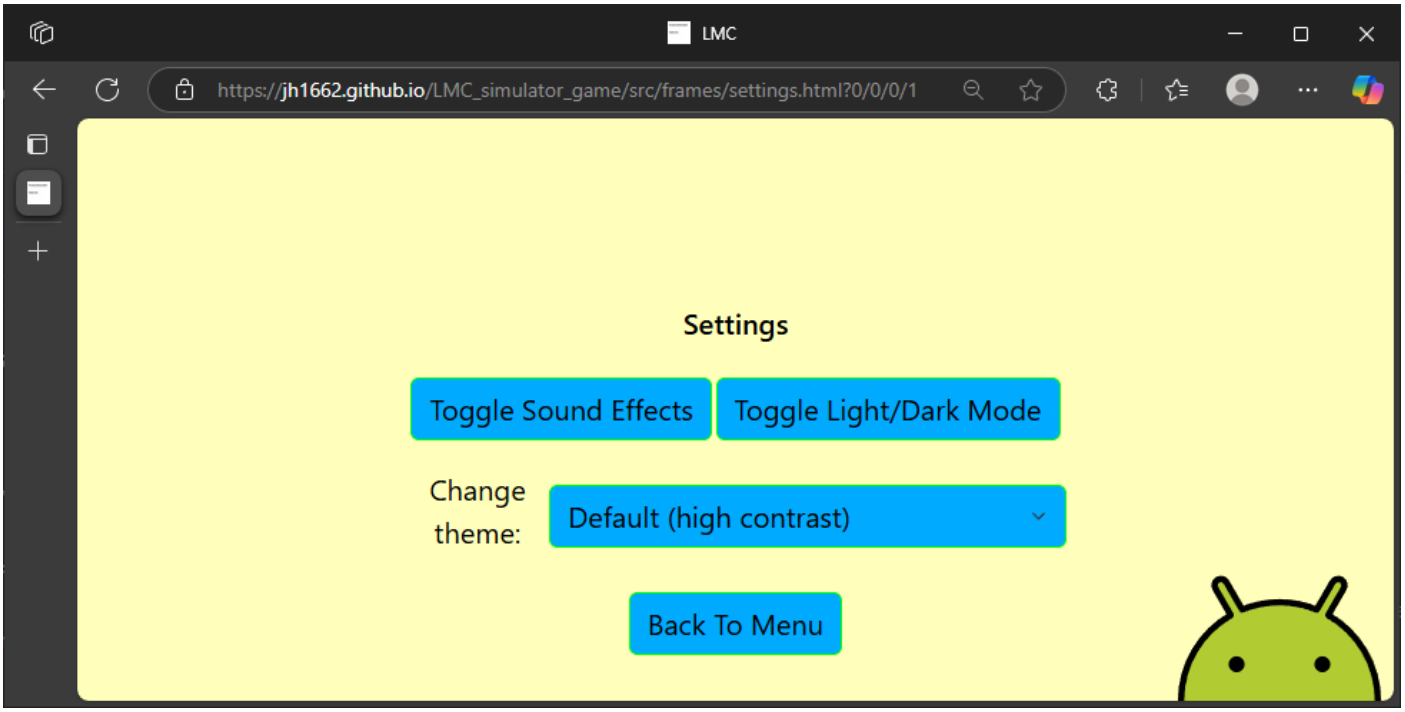


Figure 11.14—62 – Screenshot of the settings page, with the example URL with the query string, showing its options to change how CSS is interpreted (window size changed to fit the screenshot).

11.14.1.4.4.2.2 Middleware

It should be obvious that `Middleware`, from **`middleware.ts`**, had to be changed to not only accommodate simulator sandbox mode but also simulator campaign mode. This resulted in new `Middleware` methods dedicated to doing a campaign level. Methods are listed in the table below.

Method	Parameters and used class fields	When is it used/called?	What it does
`campainLevel`	None but does use the `window.location.hash` property.	When the simulator page loads (in either mode).	Checks if the level identifier value is valid and returns it as a number if so, otherwise notify the user before returning `0`. If page loads in sandbox mode, then return `-1` instead.
`prepareLevel`	Uses the `levelChecker` field to get the `partialScript` of the current level. Uses `simulatorUI` to update UI.	When simulator page loads in campaign mode.	Attempts to get the partial script of the current level. If the partial script is empty (which is expected for some levels – tutorial, contextual, and big formulas), then stop; otherwise, display the partial script into the script editor of the simulator UI.
`loadLevelSolution`	Uses the `levelChecker` field to get the solution script of the current level. Uses `simulatorUI` to update UI.	When the user wants to load the solution script of the current campaign level.	Loads the solution script into the simulator UI. If the current level is tutorial, load the partial script as the solution script instead.
`levelCompleted`	None but does use the `window.location.search` property.	When the user satisfies the current campaign level's objective.	Increments the level progress configuration in the URL query section.
`levelCheck`	`levelChecker` for checks and processes.	When the user submits a (compiled) script to attempt to satisfy the current campaign level's objective	If the level is a tutorial, then deem the level completed. Otherwise, check if there is a compiled script to submit; if not, then tell the user. If there is a compiled script, then let `levelChecker` examine/test it and display the results.

Table 11.14—39 - overview of additional campaign-related methods to the `Middleware` class.

11.14.1.4.5 Current PERT chart completion

The completion of sprint #3 has completed the LMC development, but (as seen below) the "*creating LMC code*" task is not fully completed. This is because feedback has not been taken and implemented yet as seen with the *creating LMC code* sub-timeline in **Figure 11.10—2**.

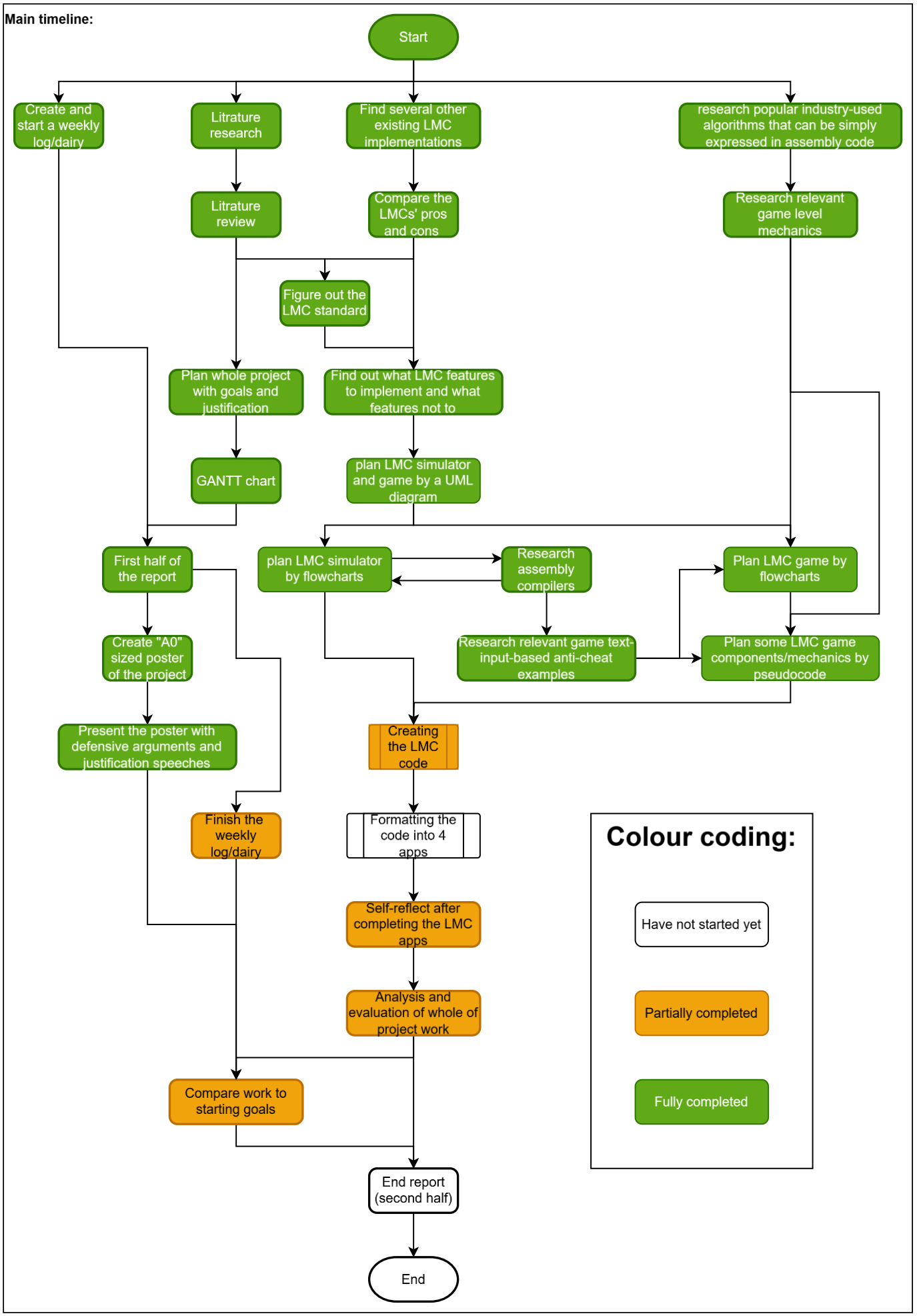


Figure 11.14—63 - current PERT chart progress at the time of after completing sprint #3.

11.14.2 feedback gathering and implementation (post-sprints)

Volunteer participation, their feedback, and value are explained in the **Human Participants** section of the **Ethical Considerations Chapter**. Please refer to that for background context and ethical explanation.

11.14.2.1 Overview

11.14.2.1.1 First volunteer – Tina Eager

11.14.2.1.1.1 Positive feedback

- Like the use of levels to help teach the students, especially the tutorial, as a non-steep learning curve.
- Like the URL data storing instead of cookies to be more transparent (no worries of privacy invasion).

This positive feedback proves that the LMC educational game satisfies Humanism principles by its non-steep learning curve.

11.14.2.1.1.2 Improvement feedback

- Need to correct the spelling of campaign (was `campain`) in the main menu page.
- Should describe how different types of campaign levels work in the manual.
- Maybe an overview of the nature of all levels in the manual.
- If the URL is what loads progress, then the manual must state it so the user keeps the URL for next time instead of starting from scratch with a new URL.

11.14.2.1.2 Second volunteer – Mr Jacob (and corresponding students)

11.14.2.1.2.1 Positive feedback

- Use of the manual was useful for the students to get started with the simulator.
- Liked the "use of gamification" on LMC, says this will help with the importance of making assembly code interesting for his Year 10 students to learn.
- Compares the LMC simulator game to be far better than LMC#2 (101 Computing, 2019a). This
- Likes the use of tutorial levels before the other level tasks for "facilities self learning at different paces".
- Loves the added instructions of shifting and ASCII outputting (`LSH`, `RSH`, and `OTC`).
- Likes the use of the 3-star system.
- Mentions that the "manual is clear".
- Likes the use of the sound effect – calls it "a good addition".

This positive feedback proves the achievement of many of the author's original goals/aims:

- The manual is successful in being the user guide for the LMC educational game
- Making LMC educational game attractive and engaging to students has been met.
- The LMC educational game is being much better and highly preferred when compared to other LMC games, especially LMC #2 from the **Software research**, being the golden standard.
- Same with the author supervisor's feedback – satisfying Humanism principles by its non-steep learning curve. More emphasis on the self-learning prospect.
- Self-learning is also an achieved goal that the author wants the LMC educational game to do, due to the evident lack of qualified Computer Science teachers (GOV.UK, 2024).
- Extra features being highly looked upon instead of being considered as minor/important parts: shifting instructions, ASCII outputting instruction, use of a 3-star level-checking system, sound effects.

11.14.2.1.2.2 Improvement feedback

- Some students got confused about which text box is the register – need to group all 4 register textboxes in a box called registers.
- It will be even better if it can insert a new row in-between lines/rows in the script editor.
- The manual has the `OTC` spelt as OCT instead - typo.

11.14.2.1.3 Third volunteer – Mr Warner (and corresponding students)

11.14.2.1.3.1 Positive feedback

- Likes the light/dark mode, says that it is “a positive for user experience and satisfaction.” And calling it “uncommon”.
- Use of toggle sound effects spares the user the bother and inconvenience of having to mute using the OS’s “volume mixer”.
- Use of accessibility-based themes is a “huge positive for user accessibility and experience”.
- The manual has successfully taught the students of the von Neumann architecture, evident with mentions of it in the students’ feedback.
- Considers the fact that the manual page “displays well regardless of window size too” to be a nice addition.
- Finds the manual very informative regarding campaign mode and its levels.
- Likes the “hand drawn visuals” of Little Man actions (the ones that the author said that he made with Microsoft Paint). Says it is good with helping with visualising that the simulator is doing.
- Finds that the use of displayed Arithmetic Logical Unit, in the simulators, helps a lot with teaching how “a computer goes through the FDE cycle” (FDE – Fetch Decode Execute).

This positive feedback proves the achievement of many of the author’s original goals/aims:

- The fact that a student said that toggle dark mode was an “uncommon”, yet highly appreciated, feature meaning that it has better points that other LMCs do not.
- The lack of inconveniences (like sound effects being toggleable) makes LMC simulator HCI and Humanism complacent.
- Testers have testified for the LMC simulator educational game to being extremely accessible, which fulfils another one of the author’s aims – catering to people with disabilities, which fulfils Humanism.
- Manual does its job of being informative of both the simulator workings and how to proceed with the game.
- Finds the simulation, especially the Arithmetic Logical Unit,
- Likes the Little Man action display as a nice touch, saying that it “help the user to visualise what the computer is doing”.
- The effectiveness of the Little Man actions display is evident, alongside other simulator parts (especially the visualised Arithmetic Logical Unit), to the test users. This demonstrates how the LMC succeeds in teaching how the von Neumann architecture – fully embracing Behaviourism overall, while specific parts cater to different intelligences (Theory of Multiple Intelligences). Such intelligences are musical intelligence catered by the toggle sound effects, logical-mathematical intelligence catered to by the Arithmetic Logical Unit, interpersonal intelligence catered by the Little Man action display, and linguistic intelligence catered by the simulator’s status.

11.14.2.1.3.2 Improvement feedback

- The word *important* is misspelt in the simulator page. Was spelt as `information`.
- Add a contrast changing button along side the dark/light mode toggle button on the simulator page.
- The manual could be a little more in-depth for new users.
- Make the tutorial levels require completing or 3 tried attempts compulsory to proceed to the next level.
- Adding padding to the whole manual page so the text is not nearly touching the edge of the window. Suggested a padding of “25-30px”.

11.14.2.2 Implementation for feedback

11.14.2.2.1 Plan

11.14.2.2.1.1 Author’s opinion of the feedback

As mentioned in **Human Participants** section, the volunteers

11.14.2.2.1.2 MoSCoW table

Because all feedback, from the volunteers, is highly valued, most MoSCoW entries are in the top 2 priority levels.

Must have	Correct spelling of campaign; Group up the register textboxes; Disable all `console.log` code; correct the `OTC` instruction misspelling in the manual page; Fix the spelling of the word *important* in the simulator page.
Should have	Explain the different level types in the manual; explain the nature of URL data and how to load and save with it; Overview of each level’s nature in the manual page; Overview of each level type in the manual page.
Could have	
Would have	Ability to insert new lines in-between existing lines of the script editor

Table 11.14—40 - MoSCoW priorities for improvement based on feedback.

Note that aside from the feedback, the author also plans to disable all `console.log` code as the product will no longer be in initial development (except maybe updates), and it does nothing for the user but slows down the LMC simulator game by a slight and noticeable difference in speed.

The reason why one feedback is in the “Would have” priority is because the author wanted to do that in sprint #2, but due to being a low priority and the lack of time (due to many unexpected obstacles), the author never had enough time to do such a thing.

A few suggestions have been purposely ignored for obvious reasons. One such is one of the students saying that the game should include a contrast setting but that is already covered by the theme changer, on the settings page, which switches between the default high contrast, low contrast, and greyscale. Another such ignored suggestion is one student suggesting making the user attempt 3 times to complete the tutorial level instead of giving the option to proceed to the next level. While that will allow the user to complete more task. The 3-attempt suggestion will heavily disrupt the comfortable yet still with critical thinking, learning curve of self-learning as well as preventing advanced users from skipping the tutorial unless they edit the URL's query data. Also, the learning curve of self-learning is heavily supported by the author's supervisor and Mr Jacob (with their corresponding students), which overshadows the 3-attempt suggestion by magnitudes.

11.14.2.2.2 Development

11.14.2.2.2.1 Disabling console log code

They are disabled instead of deleted because, in the case of a potential update in the future, the author or another developer can easily debug by re-enabling console log lines. One example of disabled console log code is shown below.

```
452         //: degugging purposes
453         ///console.log("PC - "+this.registers.read(Register.programCounter));
454         ///console.log("Instruction - "+String(this.registers.read(Register.instruction)));
455         ///console.log("Accumulator - "+this.registers.read(Register.accumulator));
456         ///console.log("Cycle count - "+cycleCount);
```

*Figure 11.14—64 - example of disabled console log lines in **vonNeumann.ts**.*

11.14.2.2.2.2 Manual page changes (HTML file)

During development, the author realised that there were a few instances of incorrect campaign spelling in the manual page, so the author amended those too.

Information on level types and natures are inserted near the bottom of the manual page, between the *ALU labelling* region and the *legalities region*, because the author wants to order the regions from the most likely wanted to the least. For example, the instruction explanation table will be the most sought after for because users will refer to them in sandbox and campaign mode, meanwhile the campaign level regions are lower because they are only needed to be referred when user is in campaign mode, and finally legalities at the bottom because the user does not expect any student user to use it nor most of the teacher users.

When prototyping, the author included the labelled URL so users not only know the URL's nature but also what each segment stores for transparency's sake.

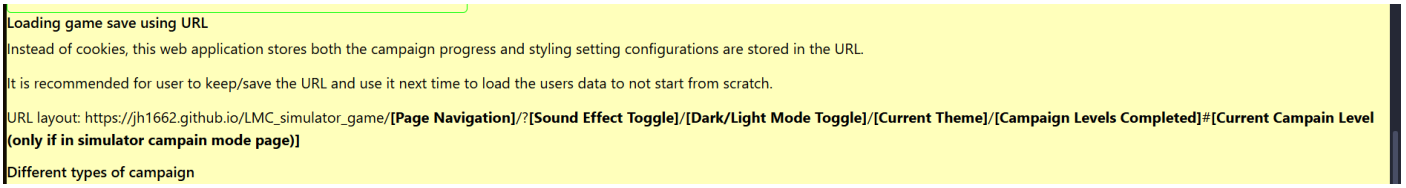


Figure 11.14—65 - Long label URL in the manual page during the development of the feedback implementation.

However, as seen in the figure above, the URL does get big, and its complexity may confuse some users. To simplify this, the author removed the fragment identifier (starting with #), which got rid of a big part of both length and complexity. Since the fragment identifier only tells the simulator what level to load, and it is only in the URL when in the campaign mode of the simulator, it does not affect transparency. Besides, the author believes that the user can easily find out what it is for without guidance.

The improved URL labelling can be seen in **Figure 11.14—79** of the **Showcase**.

```
218     <!--#region URL saving-->
219     <h2>Loading game save using URL</h2>
220     <p>Instead of cookies, this web application stores both the campaign progress and styling setting configurations are stored in the URL.</p>
221     <p>It is recommended for user to keep/save the URL and use it next time to load the users data to not start from scratch.</p>
222     <p>URL layout: https://jh1662.github.io/LMC_simulator_game/<b>[Page Navigation]</b>/?<b>[Sound Effect Toggle]</b>/<b>[Dark/Light Mode Toggle]</b>/<b>[Current Theme]</b>/<b>[Campaign Levels Completed]</b></p>
223     <!--#endregion-->
```

*Figure 11.14—66 – screenshot snippet of change to HTML, in **manual.html**, based on the URL load/save explanation feedback.*

```
<!--#region Campaign level type-->
<h2>Different types of campaign</h2>
<table class="table">
    <header><tr><td>Level type</td><td>Nature</td><td>Affected Levels</td></tr></header>
    <tr>
        <td>Tutorial</td>
        <td>Simple teaches how to code in the LMC - only objective is to learn. Just click "Finish tutorial" to proceed to the next level.</td>
        <td>1, 2, 3, 4, 5, 6, 7, 8</td>
    </tr>
    <tr>
        <td>Contexual</td>
        <td>Level objective is contexualised, no other traits.</td>
        <td>13, 14, 16, 17</td>
    </tr>
    <tr>
        <td>Big formula</td>
        <td>Level objective is based making a program for a mathematical equation.</td>
        <td>10, 20</td>
    </tr>
    <tr>
        <td>Correcting</td>
        <td>An assembly script is already loaded but is inncorrect, user must find mistakes and correct them.</td>
        <td>9, 12, 19</td>
    </tr>
    <tr>
        <td>Partial</td>
        <td>Part of an assembly script is already loaded, user must complete the script to satisfy the level's objective.</td>
        <td>11, 15, 18</td>
    </tr>
</table>
<!--#endregion-->
```

*Figure 11.14—67 - screenshot snippet of change to HTML, in **manual.html**, based on the level type explanation feedback.*

```
<!--#region Campaign level nature-->
<h2>Each campaign's level's nature</h2>
<table class="table">
    <header><tr><td>Level #</td><td>Type</td><td>3-star line count limit</td><td>2-star line count limit</td><td>Program</td></tr></header>
    <tr>
        <td>1</td><td>Tutorial</td><td>N/A</td><td>N/A</td>
        <!--^ all in one line to save space as innerHTMLs are short and concistent -->
        <td>relay - input then output</td>
    </tr>
    <tr>
        <td>2</td><td>Tutorial</td><td>N/A</td><td>N/A</td>
        <td>adder calculator - store input then add</td>
    </tr>
    <tr>
        <td>3</td><td>Tutorial</td><td>N/A</td><td>N/A</td>
        <td>withdrawal - load then subtract</td>
    </tr>
    <tr>
```

*Figure 11.14—68 - screenshot snippet of (the top part of) change to HTML, in **manual.html**, based on each level's nature explanation feedback.*

Due to the size of the level nature table code, only the top part of the code is shown in Figure 11.14—68.

There are also instances of campaign spelling that were corrected, but those codes are not shown here because those instances are spread across `manual.html`.

11.14.2.2.2.3 Main menu page change (HTML file) – spelling

```
        <tr>
            <td><button class="btn mt-3" id="campain">Campaign</button></td>
```

*Figure 11.14—69 - screenshot snippet of change to HTML, in **menu.html**, based on the incorrect campaign spelling feedback.*

11.14.2.2.2.4 GUI

Because the author finished the feedback implementation quicker than expected, he decided to do a bit more of the lower-priority tasks.

One such is the graphics.

Before, the author uses sketched, Microsoft Paint, images instead of refined GIFs and SVGs. While making assets can be time-consuming, especially with the Little Man Actions (as seen in Figure 11.13—10), the foremost images can at least be done. The author deems the favicon and menu title as the most foremost assets. This is because the favicon is displayed on every LMC simulator game page, due to its nature, while the menu title is what gives a first impression to the user and thus must be the more attractive part of the simulator game.

11.14.2.2.2.4.1 Favicon

Favicon must be of 1:1 ratio with dimensions (Google Search Central, 2025). While different web browsers (including Android apps) have different variations of the maximum and minimum dimensions*, a safe bet that should satisfy all (if not then definitely nearly all) should be between the range of 8 and 512 pixels long. Not much research was done with the favicon because it is of lower priority, and sometimes the user may not even notice it if missing. The author decided to go with 16 by 16 pixels because it is a good balance between storage size and is able to convey/depict the LMC simulator game.

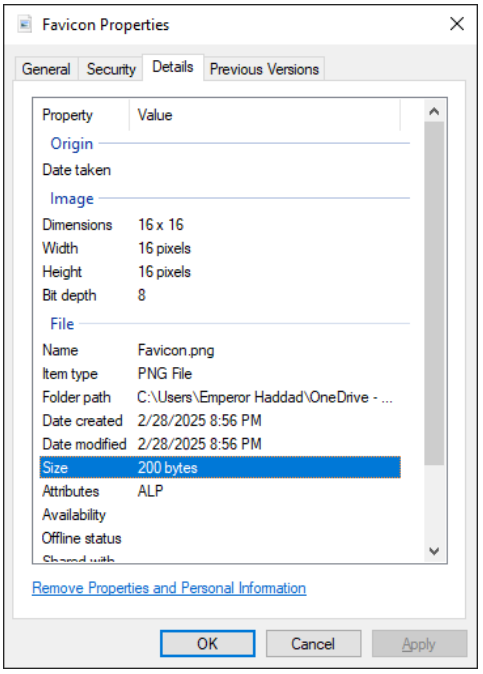


Figure 11.14—70 - showing the feedback implementation favicon's size and, more importantly, the file storage size of 200 bytes.

By using smaller dimension assets, the author ensures the LMC simulator game's trait of being lightweight remains that way.

The small size also makes the favicon much faster and easier to create, which is ideal because adding a favicon is not a necessity.

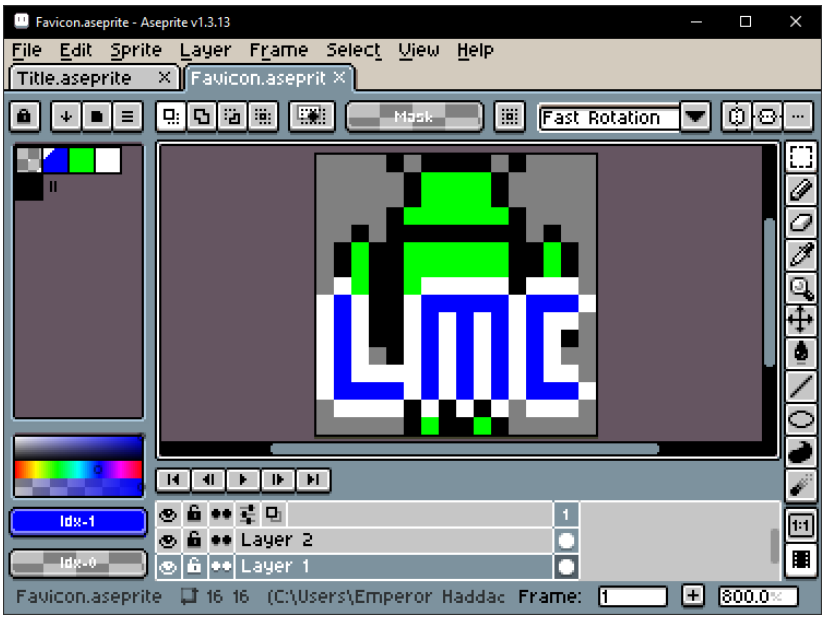


Figure 11.14—71 - a view of the favicon as it is made in the Aseprite editor tool, from the feedback implementation development.

11.14.2.2.2.4.2 Menu Tile

Taking points from the favicon, the menu title's dimension should be as small as possible without diminishing the attractive effect of the menu title asset. To get around that, the author decided to make the asset in a "pixel art" style, using custom CSS (there is no Bootstrap CSS that does it) to prevent blurring as seen in the code screenshot snippet below.

```
/*#endregion*/
/*: From feedback implementation*/
.pixelArt{
    /** inherits from 'img' class*/
    width: 512px;
    /*^ Enlarged size also serves to not mess up the layout of the menu buttons. */
    /*^ Overwrites property of 'img'.*/
    /*^ Pixel unit is used to share the same non-scalable characteristic with other menu page elements, such as paragraphs and buttons.*/
    image-rendering: pixelated;
    /*^ supported in the Chromium-based browsers, such as: Chrome, Edge, Opera, et cetera */
    image-rendering: crisp-edges;
    /*^ supported in the other browsers, such as Firefox and Safari */
}
```

Figure 11.14—72 - custom CSS used for pixel art (main menu title).

However, the author did make the title asset a GIF instead of a PNG – satisfying Humanism, due to being graphically engaging, as part of one of the lower priorities. This makes the "first impression" even better and its constant state of animation satisfies HCI principles by ensuring the user that the page is not frozen (alongside buttons changing colours when hovered on).

Figure 11.14—73 - a view of the first frame of the menu title GIF, made with the Aseprite editor tool, from the feedback implementation development.

Figure 11.14—74 - a view of the second frame of the menu title GIF, made with the Aseprite editor tool, from the feedback implementation development.

Despite GIF being an animated asset, the use of only 2 frames (with a duration of 2 seconds per frame) only makes its file size 1.81KB, which is fully within the acceptance of being very lightweight.

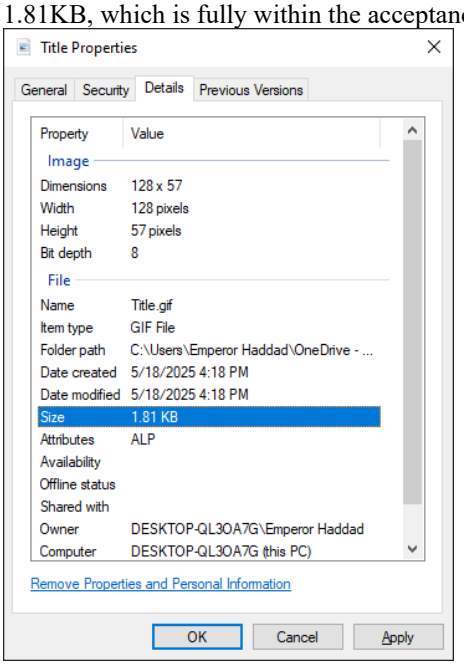


Figure 11.14—75 - Showing the feedback implementation menu title's size and, more importantly, the file storage size of 1.81KB.

11.14.2.2.5 Simulator page change (HTML file)

```
71              <div class="col-sm-4">
72                  <span>Registers</span>
73                  <div class="well">
74                      <div class="list-item">
```

Figure 11.14—76 - adding the title Registers to the sub-section containing the 4 registers in the simulator.

When the author was doing this change, he decided to add titles to all other sub-sections of the simulator to further prevent potential confusion with the different sub-sections of the simulator by grouping them up.

11.14.2.2.6 Testing

All Blackbox and Whitebox tests were performed again, and no problems/failures were encountered there.

11.14.2.2.3 Review

All priorities (below "would have"), from **Table 11.14—40**, have been met – feedback fully accounted for and implemented as improvements to the LMC simulator game.

The author is more satisfied with the quantity and quality of these feedback reports with their compliments and suggestions for improvement.

11.14.2.2.3.1 Showcase

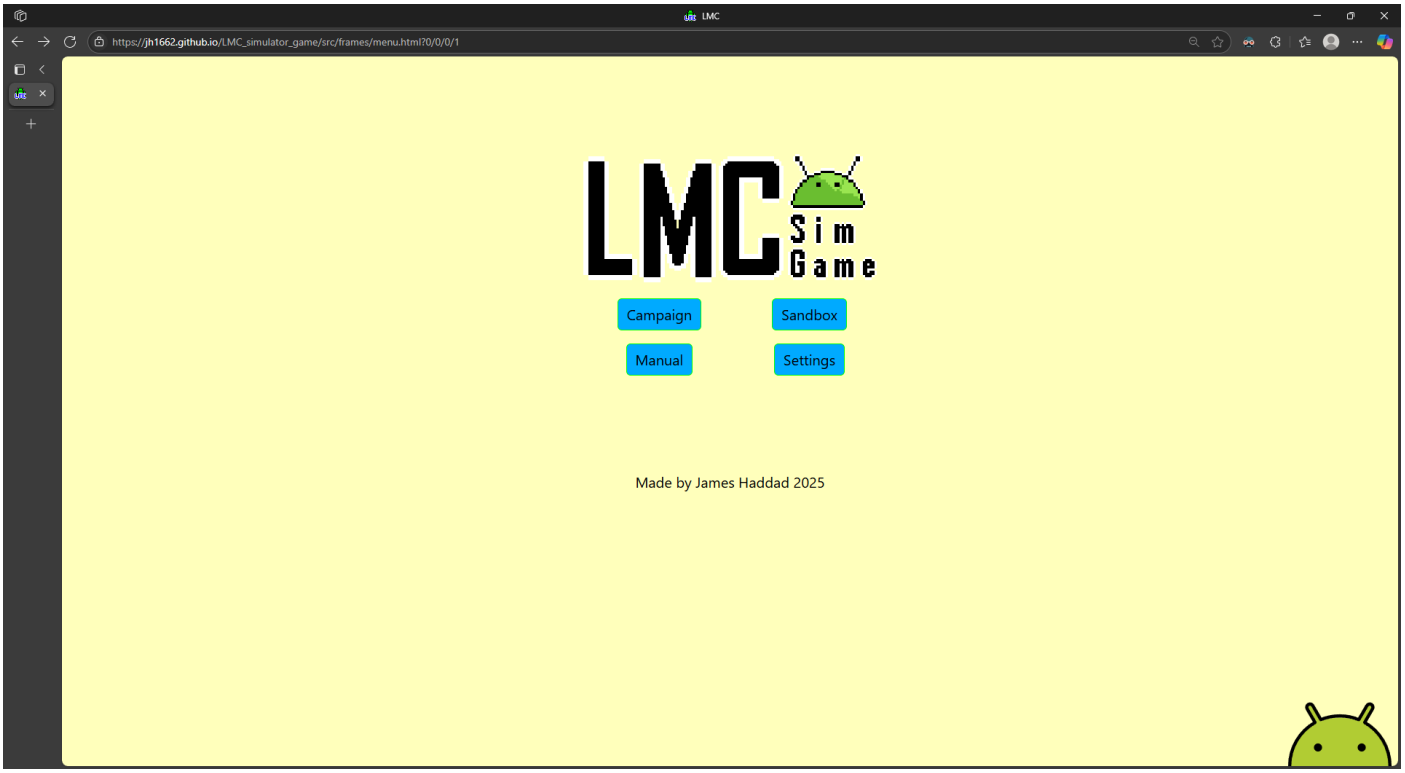


Figure 11.14—77 – screenshot of the main menu, of the improvement feedback implementation, where the spelling of the campaign word is corrected and title logo has been improved.

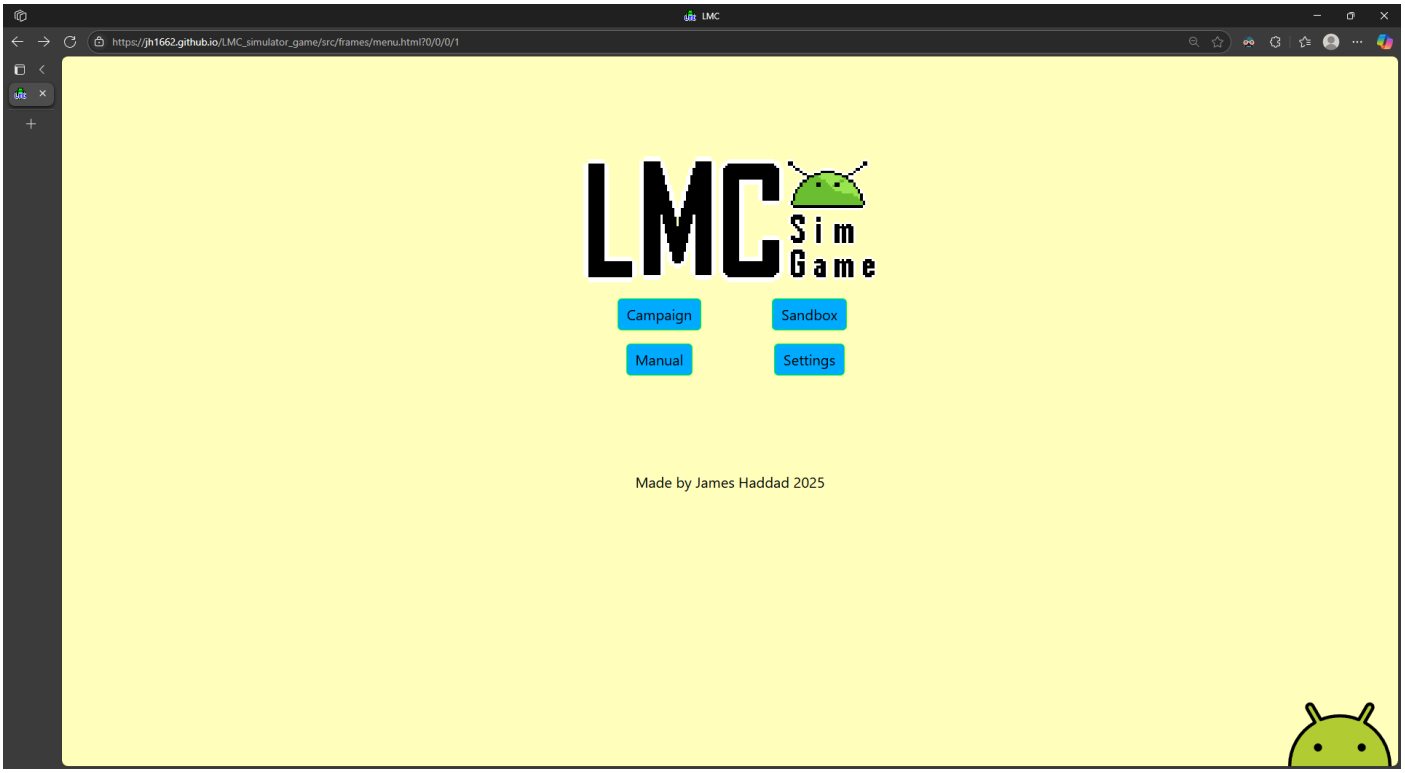


Figure 11.14—78 – screenshot of the main menu, of the improvement feedback implementation, showing the second frame of the title logo (it is a GIF).

Instruction Manual

Little Man Computer (LMC) is a visual simulator of a simplied version of the Von Neumann architecture CPU (computer processing). Below is a quick guide that will assist helping inexperienced users whether they are stuck on a campaign level or just want to play around in the sandbox mode.

Instruction Set

Name	Mnemonic	Code	Description
Addition	ADD	1XX	Add memory cell address' value to accumulator's value
Subtraction	SUB	2XX	Subtract memory cell address' value from accumulator's value
Store from Accumulator	STA	3XX	Store accumulator's value in memory cell address
Left Shift	LSH	401	Shift the accumulator value's base-10 digits of significance to the left by one digit.
Right Shift	RSH	402	Shift the accumulator value's base-10 digits of significance to the right by one digit.
Load to Accumulator	LDA	5XX	Load memory address's value to the accumulator (becomes the new accumulator's value).
Branch Always	BRA	6XX	Branch – change PC's value to – the address value (regardless of accumulator's value).
Branch if Zero	BRZ	7XX	Branch – change PC's value to – the address value if accumulator's value is zero.
Branch if Positive	BRP	8XX	Branch – change PC's value to – the address value if accumulator's value is positive or zero (not negative).
Input	INP	901	Takes user's or predefined input and store it as accumulator's value.
Output	OUT	902	Outputs from accumulator's value (as integer).
Output as Character	OTC	903	Outputs from accumulator's value as an ASCII character
Halt	HLT	0	Stops program
Data location	DAT	N/A	Compile data into memory before program starts (can be used as variables)

Script Editor

Use the following keys to navigate:

- Tab key - Go down a line if in an operand box.
- Space key - Go down a line if in an operand box.
- Enter key - Go back to the top line if on the bottom line.
- Downwards key - Go back to the top line if on the bottom line.
- Upwards key - Go to the bottom line if on the top line.

Typing on the bottom code line/row will generate a new empty one underneath.

Registers

Name	Description	Valid range
Program Counter (PC)	Dictates where the simulator fetches its next instruction (which RAM cell).	0 to 99
Memory Instruction Register (MIR)	Gathered from the first digit of the fetched instruction from memory.	0 to 9
Memory Address Register (MAR)	Gathered from the last two digits of the fetched instruction from memory.	0 to 99
Accumulator	A form of immediate storage for processor operations such as adding, outputing, inputing, etc.	-999 to 999

Simulator Controls

Controls relavent to simulator

- Run - Runs the compiled program as seen in the RAM table (requires valid compilation first)
- Compile - Attempts to compile script and store in memory.
- Stop - Stops currently executing assembly program.
- Reset - Reloads page to rid off unexpected errors (and current script, so think twice before reloading).
- Toggle Display - Switches between Little Man action display and the current campaign level objective (does not really matter if in sandbox mode).
- Toggle Execution Mode - Switches between continuously running each cycle, but with configurable speeds, or only execute next cycle when "New Cycle" button is pressed.

Arithmetic Logical Unit

Arithmetic Logical Unit status

[1] [2] [3]

1. Flow:

Symbol	Meaning	Explaination
_	Normal	Initial result in range (-999 to 999).
^	Overflow	Result of operation exceeded 999 and thus resets back from -999.
V	Underflow	Result of operation exceeded -999 and thus resets back from 999.

None of these indicate error with the simulator - fully informative.

2. Operation - Last operation performed in the Arithmetic Logical Unit.
3. Result - Result of the last operation performed in the Arithmetic Logical Unit.

Loading game save using URL

Instead of cookies, this web application stores both the campaign progress and styling setting configurations are stored in the URL.

It is recommended for user to keep/save the URL and use it next time to load the users data to not start from scratch.

URL layout: https://jh1662.github.io/LMC_simulator_game/**[Page Navigation]**/?**[Sound Effect Toggle]**/**[Dark/Light Mode Toggle]**/**[Current Theme]**/**[Campaign Levels Completed]**

Different types of campaign levels

Level type	Nature	Affected Levels
Tutorial	Simple teaches how to code in the LMC - only objective is to learn. Just click "Finish tutorial" to proceed to the next level.	1, 2, 3, 4, 5, 6, 7, 8
Contexual	Level objective is contexualised, no other traits.	13, 14, 16, 17
Big formula	Level objective is based making a program for a mathematical equation.	10, 20
Correcting	An assembly script is already loaded but is inncorrect, user must find mistakes and correct them.	9, 12, 19
Partial	Part of an assembly script is already loaded, user must complete the script to satisfy the level's objective.	11, 15, 18

Each campaign level's nature

Level #	Type	3-star line count limit	2-star line count limit	Program
1	Tutorial	N/A	N/A	relay - input then output
2	Tutorial	N/A	N/A	adder calculator - store input then add
3	Tutorial	N/A	N/A	withdrawal - load then subtract
4	Tutorial	N/A	N/A	hundred rounding - Shift right twice then left twice
5	Tutorial	N/A	N/A	character set - number to ASCII converter
6	Tutorial	N/A	N/A	varaibles - add 10 to input
7	Tutorial	N/A	N/A	selection (if-statements) - branching if zero then if zero/positive
8	Tutorial	N/A	N/A	iteration - using counter to exit infinite loop
9	Correcting	14	19	comparison - which input is higher
10	Big formula	8	13	linear graph equation - y=2x+3
11	Partial	13	18	even - is positive number even?
12	Correcting	20	25	positive multiplication - calculator
13	Contexual	5	10	unit converter - add 273 for C to K
14	Contexual	22	27	squaring - input ^ 2
15	Partial	12	17	overflow - addition over the limit?
16	Contexual	7	12	digit significance - least significant digit
17	Contexual	23	18	floor division - calculator
18	Partial	65	70	FizzBuzz - classic programming
19	Correcting	24	29	bank balance - adding/subtracting until negative
20	Big formula	18	23	fibonacci sequence - 1st to 17th term (0-987)

Legal claimers

Created by James Haddad 2025

The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Vectors and icons by SVG Repo

Figure 11.14—79 - screenshot of the manual, of the improvement feedback implementation, where the sections `Loading game save using URL`, `Different types of campaign levels`, *and* `Each campaign level's nature` *are added and padding added to the entire body.*

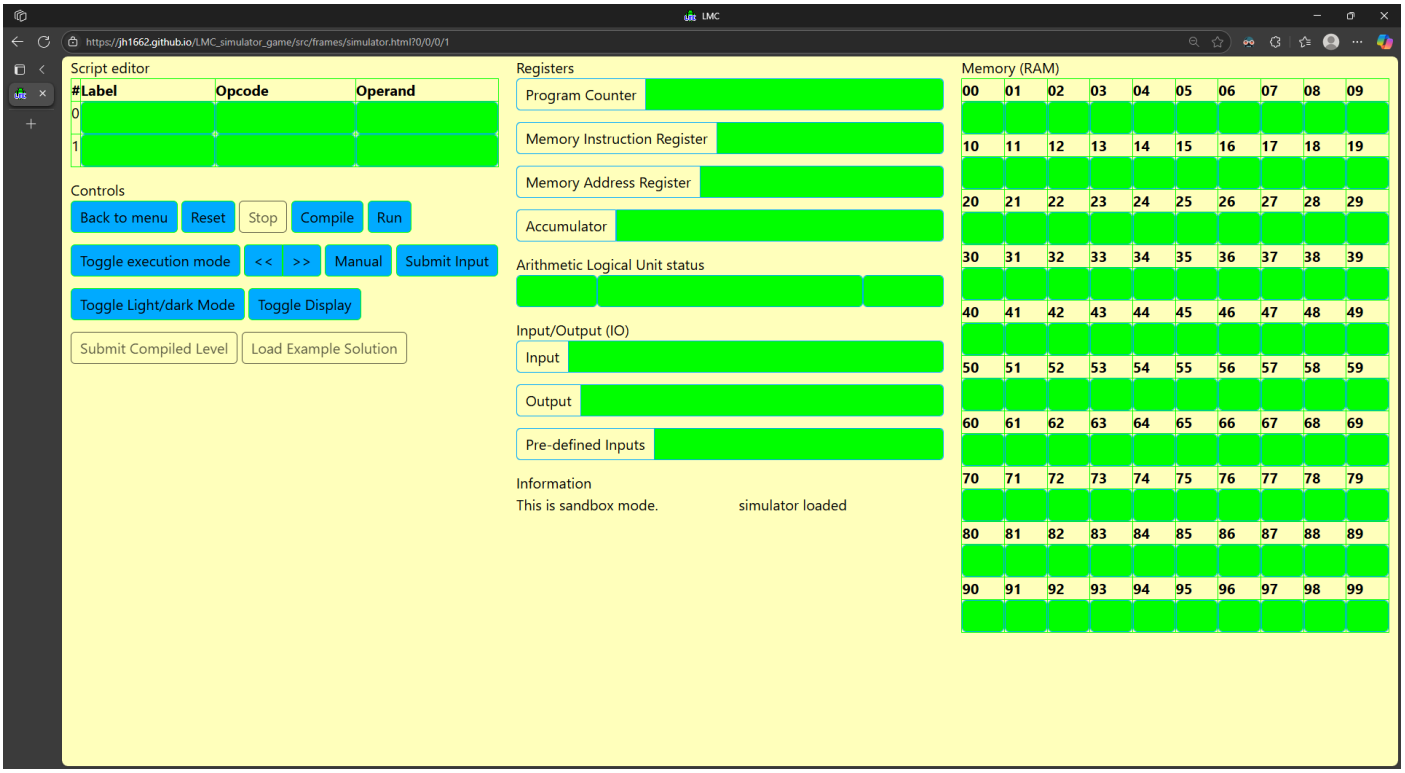


Figure 11.14—80 – screenshot of the simulator (in sandbox mode), of the improvement feedback implementation, where each of every sub-section is labelled, those being: `Script editor`, `Controls`, `Registers`, `Arithmetic Logical Unit status`, `Input/Output (IO)`, `Information`, *and* `Memory (RAM)` *as well as the spelling correction of the word important.*

11.14.2.2.3.2 Unplanned changes to the Agile development product

The only unplanned change was the **GUI** which was justified in the **Development** section.

11.14.2.3 Evidence of feedback

In the case that the reader wants to know the exact evident feedback (from participants), for whatever reason, the reader can simply see it below.

11.14.2.3.1 Feedback from Mr Jacob

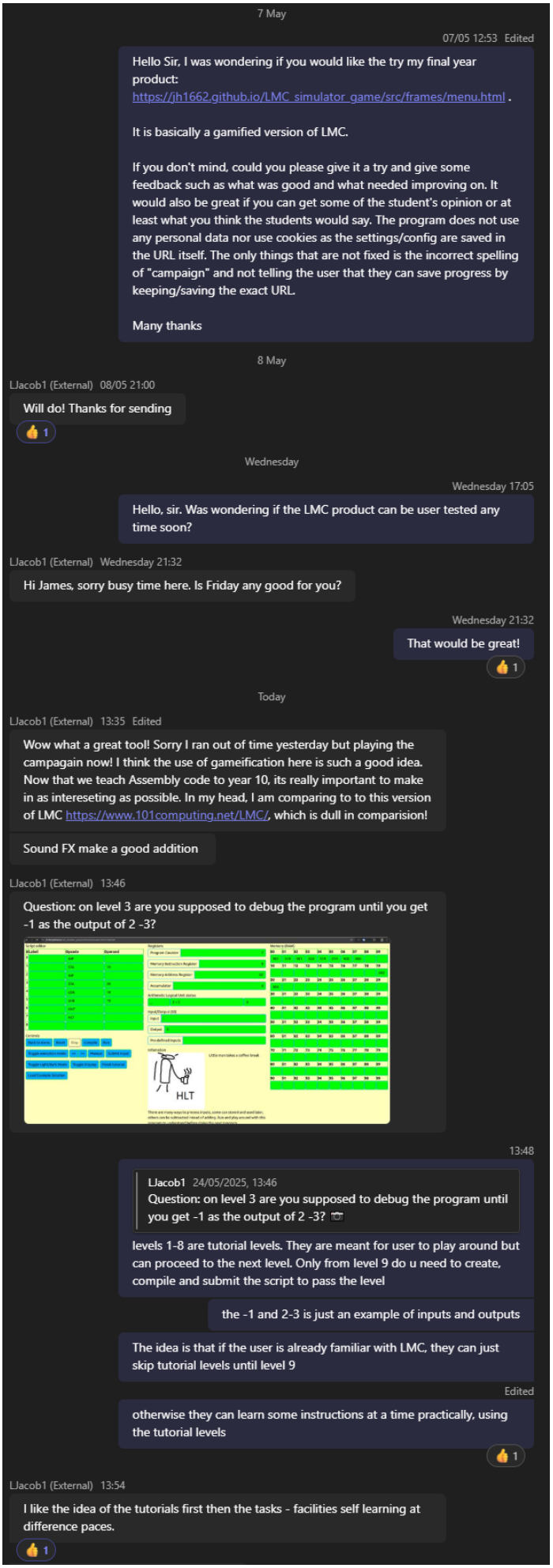


Figure 11.14—81 - part 1/3 of feedback discussion with Mr Jacob.

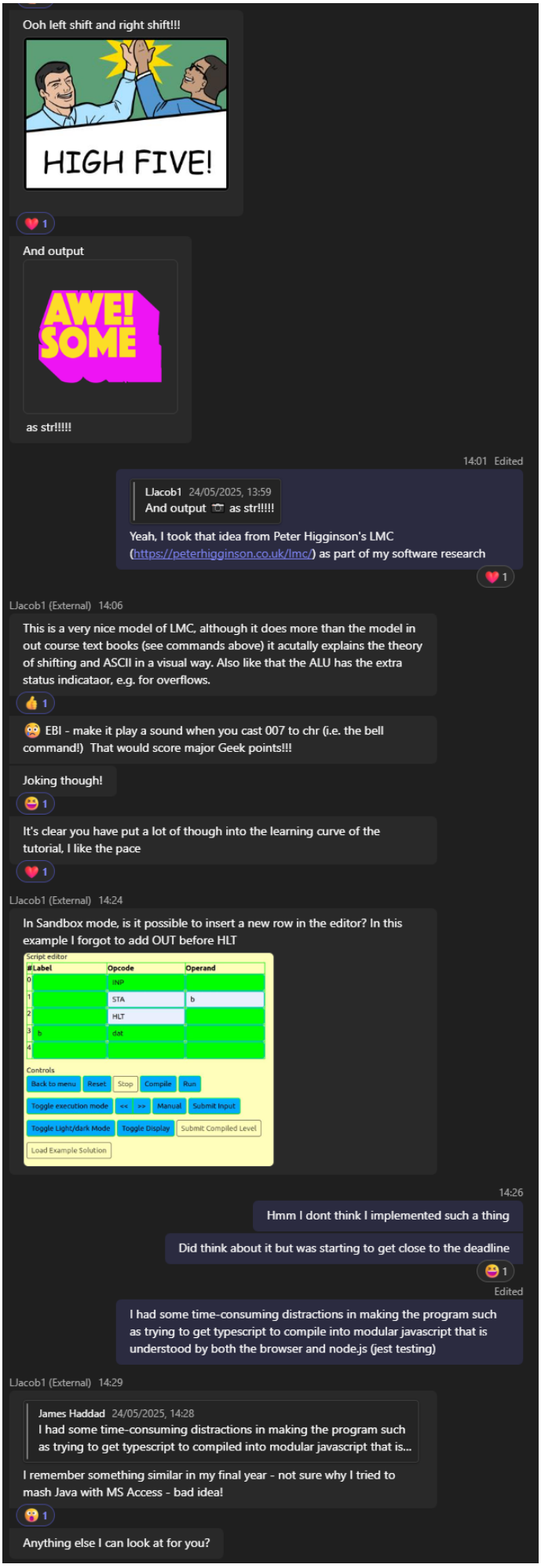


Figure 11.14—82 - part 2/3 of feedback discussion with Mr Jacob.

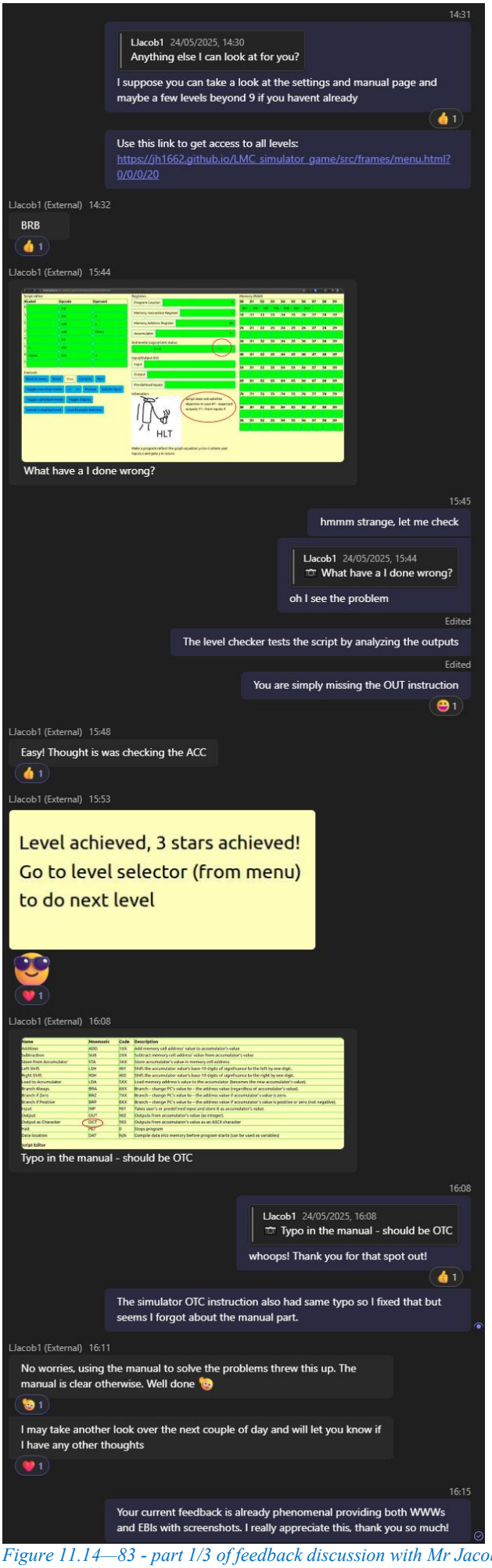
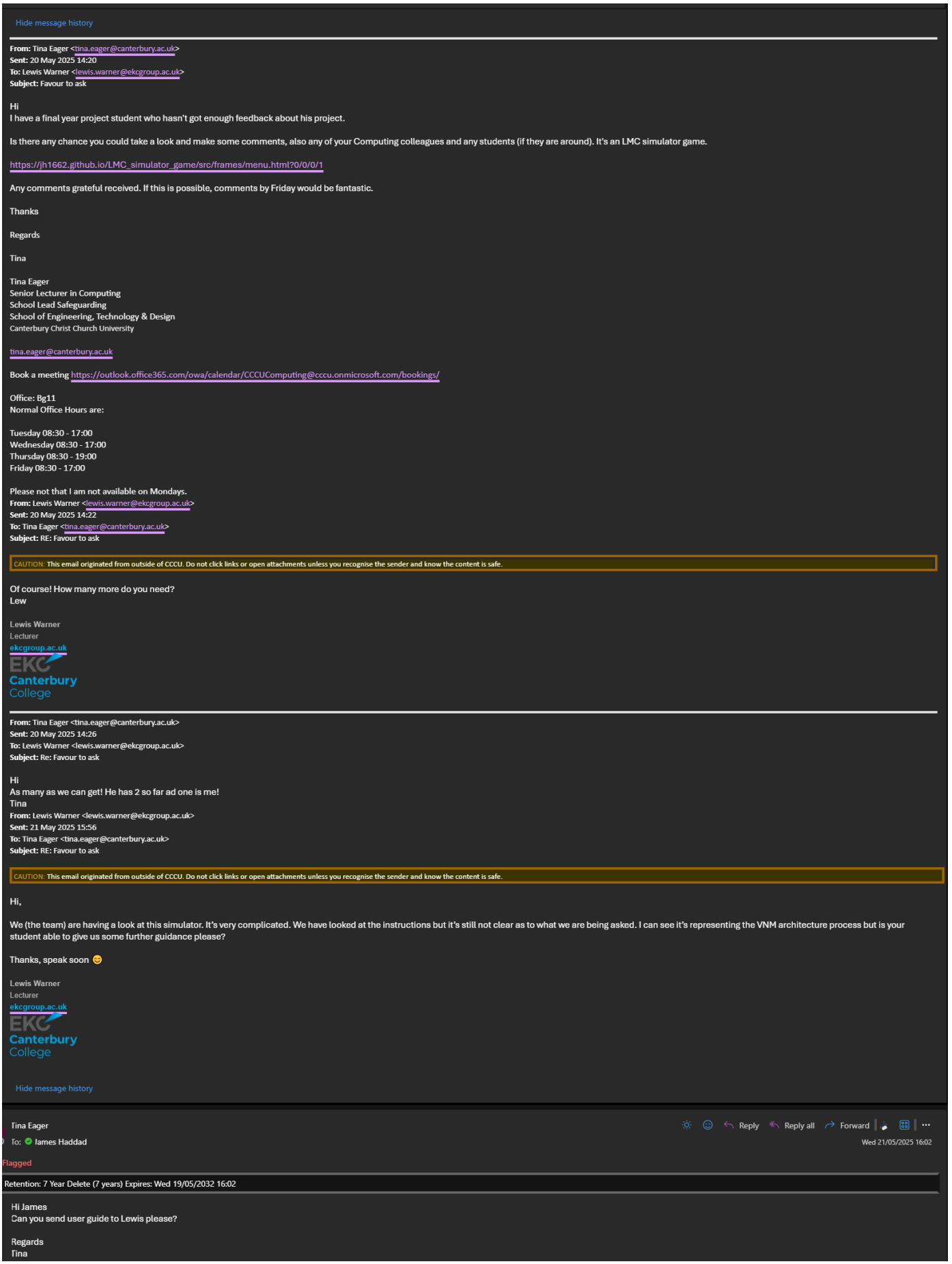


Figure 11.14—83 - part 1/3 of feedback discussion with Mr Jacob.

11.14.2.3.2 Feedback from Mr Warner



Hide message history

From: Tina Eager <tina.eager@canterbury.ac.uk>
Sent: 20 May 2025 14:20
To: Lewis Warner <lewis.warner@ekcgroup.ac.uk>
Subject: Favour to ask

Hi
I have a final year project student who hasn't got enough feedback about his project.

Is there any chance you could take a look and make some comments, also any of your Computing colleagues and any students (if they are around). It's an LMC simulator game.

https://jh1662.github.io/LMC_simulator_game/src/frames/menu.html?0/0/0/1

Any comments grateful received. If this is possible, comments by Friday would be fantastic.

Thanks

Regards

Tina

Tina Eager
Senior Lecturer in Computing
School Lead Safeguarding
School of Engineering, Technology & Design
Canterbury Christ Church University

tina.eager@canterbury.ac.uk

Book a meeting https://outlook.office365.com/owa/calendar/CCCUComputing@cccu.onmicrosoft.com/bookings/

Office: Bg11
Normal Office Hours are:

Tuesday 08:30 - 17:00
Wednesday 08:30 - 17:00
Thursday 08:30 - 19:00
Friday 08:30 - 17:00

Please not that I am not available on Mondays.
From: Lewis Warner <lewis.warner@ekcgroup.ac.uk>
Sent: 20 May 2025 14:22
To: Tina Eager <tina.eager@canterbury.ac.uk>
Subject: RE: Favour to ask

CAUTION: This email originated from outside of CCCU. Do not click links or open attachments unless you recognise the sender and know the content is safe.

Of course! How many more do you need?
Lew

Lewis Warner
Lecturer
ekcgroup.ac.uk
EKC Canterbury College

From: Tina Eager <tina.eager@canterbury.ac.uk>
Sent: 20 May 2025 14:26
To: Lewis Warner <lewis.warner@ekcgroup.ac.uk>
Subject: Re: Favour to ask

Hi
As many as we can get! He has 2 so far ad one is me!
Tina
From: Lewis Warner <lewis.warner@ekcgroup.ac.uk>
Sent: 21 May 2025 15:56
To: Tina Eager <tina.eager@canterbury.ac.uk>
Subject: RE: Favour to ask

CAUTION: This email originated from outside of CCCU. Do not click links or open attachments unless you recognise the sender and know the content is safe.

Hi,

We (the team) are having a look at this simulator. It's very complicated. We have looked at the instructions but it's still not clear as to what we are being asked. I can see it's representing the VNM architecture process but is your student able to give us some further guidance please?

Thanks, speak soon 😊

Lewis Warner
Lecturer
ekcgroup.ac.uk
EKC Canterbury College

Hide message history

Tina Eager
To: James Haddad
Reply Reply all Forward
Wed 21/05/2025 16:02

Flagged

Retention: 7 Year Delete (7 years) Expires: Wed 19/05/2032 16:02

Hi James
Can you send user guide to Lewis please?

Regards
Tina

Figure 11.14—84 - email conversation between author's supervisor and Mr Lewis regarding testing the LMC educational with students from EKO Canterbury Collage.

From: James Haddad <j.haddad1662@canterbury.ac.uk>
Sent: Wednesday, May 21, 2025 3:58:14 PM
To: Lewis Warner <lewis.warner@ekcgroup.ac.uk>
Cc: Tina Eager <tina.eager@canterbury.ac.uk>
Subject: LMC User guide

> You don't often get email from j.haddad1662@canterbury.ac.uk. Learn why this is important

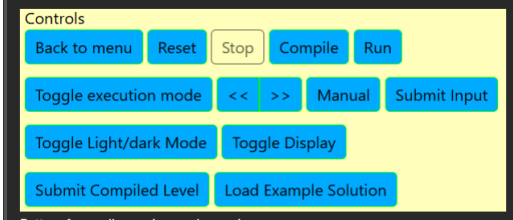
Hello Lewis,

Thank you very much for agreeing at try out my LMC. From the email, it seems u have read the manual page (https://jh1662.github.io/LMC_simulator_game/src/frames/manual.html?) but still confused on what to do.

There are two ways you can use the simulator - sandbox and campaign mode.

Sandbox is where you create any assembly program you like and experiment with assembly programming while campaign mode starts with the first 8 levels as tutorial/introduction levels. The Idea is that the user learns how to use it in those tutorial levels.

The buttons were not clearly covered in the manual so I will explain them here.



Buttons for sandbox and campaign mode:

- Back to menu - Changes page to the menu page. User can change levels by going back to menu then back to campaign to see the level selector.
- Reset - Simply refreshes the page. Used only in unexpected errors but should not happen.
- Stop - Stop the current execution of the compiled user script. To use when user wants to stop for whatever reason, including if the compiled program gets stuck in an infinite loop.
- Compile - Attempts to compiler user's assembly script into the memory (right of the simulator UI) as the level checker and executor only understands compiled code.
- Run - Simply runs the last compiled script
- Toggle execution mode - Switches between execution modes of constant execution by speed or executing one instruction cycle every time the user presses "next cycle".
- <</>> - Decreases and increases execution speed. Changes to "next cycle" if execution mode changes.
- Manual - Opens up the manual in a new window.
- Submit input - Submits input for when the program requests it. If there are entries in the pre-defined inputs, then those will be used until they are all used up.
- Toggle Light/Dark mode - Switched between light and dark mode for user's convivence but preference is not saved. light/dark mode preference is only saved when changing it in the settings page.
- Toggle display - Switches between showing "Little Man" actions and current level's case example

Buttons that are only enabled in campaign mode:

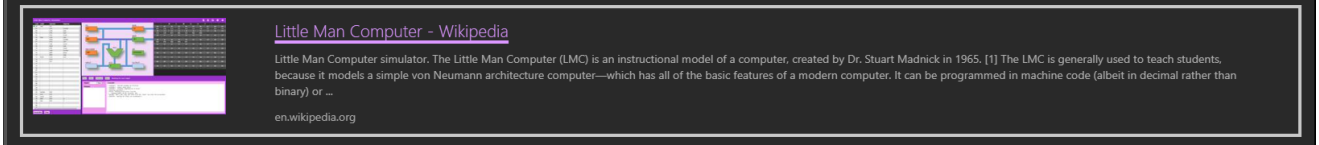
- Submit compiled level - Submit the last compiled assembly script for the level checker to decide if the user passes the level or not.
- Load example solution - Loads a solution of a level if user struggles to do it so user can learn from it (can be compiled and submitted to pass the level if user really wants to skip it).

For more information regarding LMC as a concept please refer to:

- Wikipedia for a general explanation - https://en.wikipedia.org/wiki/Little_Man_Computer
- YouTube for a video guide of using a similar LMC simulator - https://www.youtube.com/watch?v=kozwnTF1vS8

I hope this helps.

If there is still confusion, please do not hesitate to reach out.
Many thanks,
James



From: Lewis Warner
Sent: 22 May 2025 14:41
To: James Haddad <j.haddad1662@canterbury.ac.uk>
Subject: RE: LMC User guide

> **Lewis Warner** reacted to your message:

From: Lewis Warner
Sent: 23 May 2025 09:49
To: James Haddad <j.haddad1662@canterbury.ac.uk>
Cc: Tina Eager <tina.eager@canterbury.ac.uk>
Subject: RE: LMC User guide

Hi,

I have 4 students reviewing as we speak.

Feedback incoming.

Lewis Warner
Lecturer
ekcgroup.ac.uk
EKC
Canterbury
College

Figure 11.14—85 – initial (part 1 of 2) email conversation between author and Mr Lewis regarding testing the LMC educational with students from EKO Canterbury Collage.

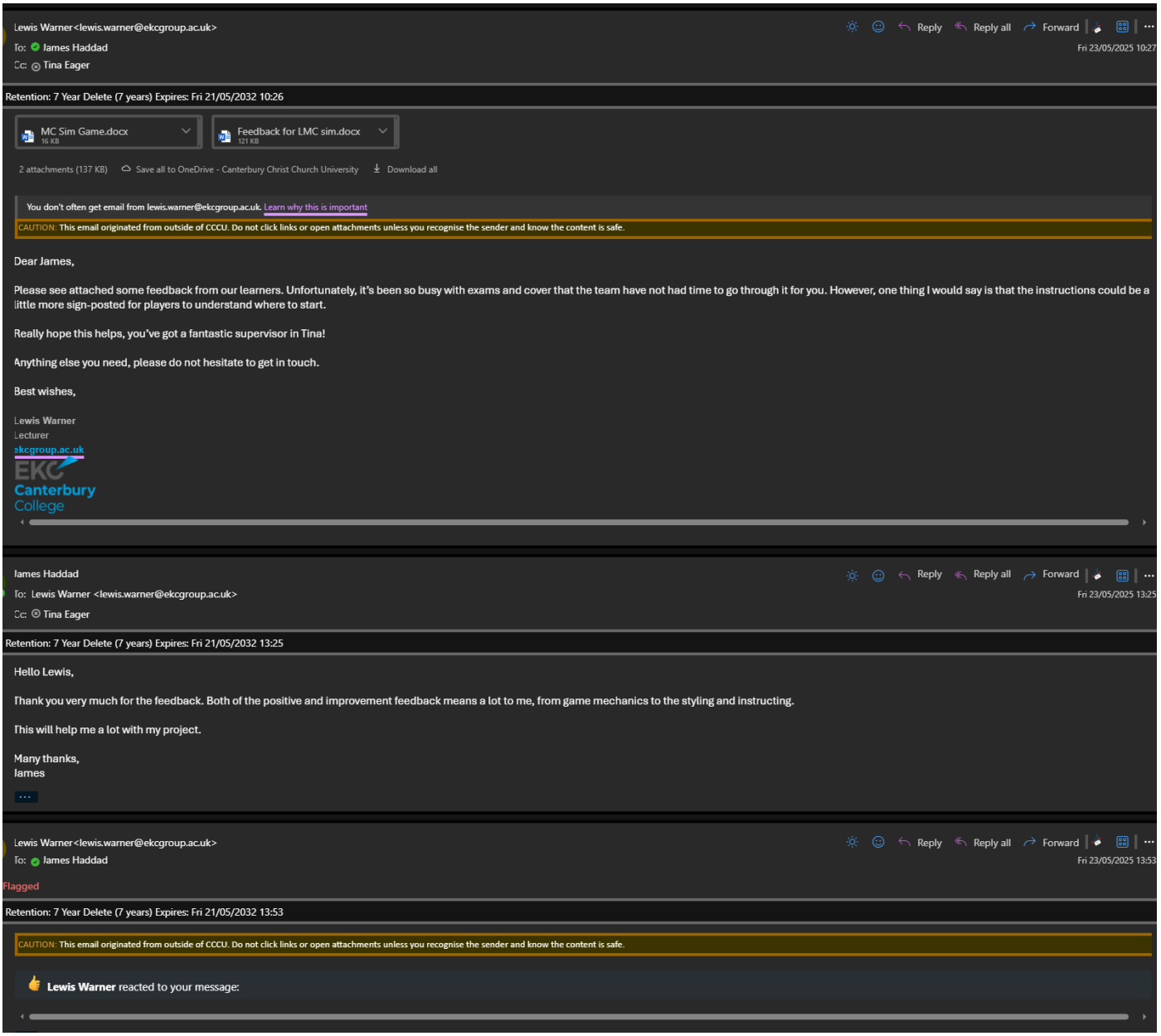


Figure 11.14—86 – initial (part 2 of 2) email conversation between author and Mr Lewis regarding testing the LMC educational with students from EKO Canterbury Collage.

MC Sim Game – Feedback

Settings

Having a toggle menu for light and dark mode is a good feature to have and is uncommon in games so is a positive for user experience and satisfaction.

The same goes for being able to toggle the sound effects – as it allows other tabs to keep their volume instead of the whole browser window having to be muted in the volume mixer.

In terms of accessibility, the theme changes for high contrast, low contrast and greyscale are a huge positive for user accessibility and experience.

Manual

The manual is understandably long considering the amount of content surrounding the Von Neumann architecture and the different instruction sets, and their definitions and how they are represented. The manual also displays well regardless of window size too, which is a bonus.

I personally like that the nature of each campaign level is in the manual too, as this appears to be a good educational resource, so having all of the information in one place to refer to gives the user the best possible chance to be able to understand the game.

Campaign

Thanks to the manual, the campaign is understandable and was easy to complete the first level. The layout of the CPU is well thought out and easy on the eye to look at, with everything in an organised place. The information is a useful touch to understand the process of the input being outputted with all the steps in between.

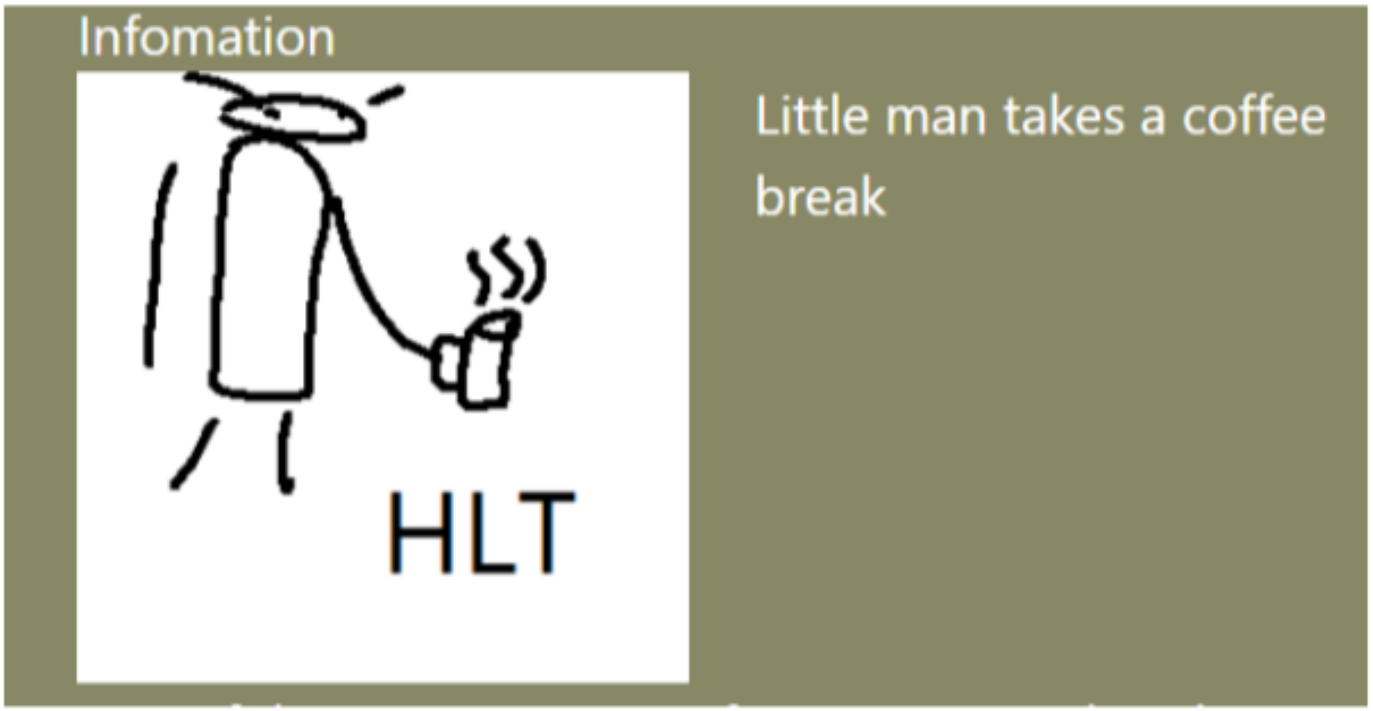
The light and dark mode is also a good feature to have while in the game too, if people immediately enter the game without checking the settings, it removes the necessity of the user having to return to the main menu, then enter settings, return to the main menu and then enter the game again.

Having an example solution feature for people that are struggling to understand or get the process to start is a good user experience-related feature. This will hopefully help users to understand the later levels.

Tips for improvement

- Having the contrast settings in the game as well or instead of the dark mode button would be a good touch, even if you need to sacrifice the button size.
- The colors for the low contrast background isn't visually appealing, although functional so it's 50/50 whether to focus on functionality only or the visual appeal of the game

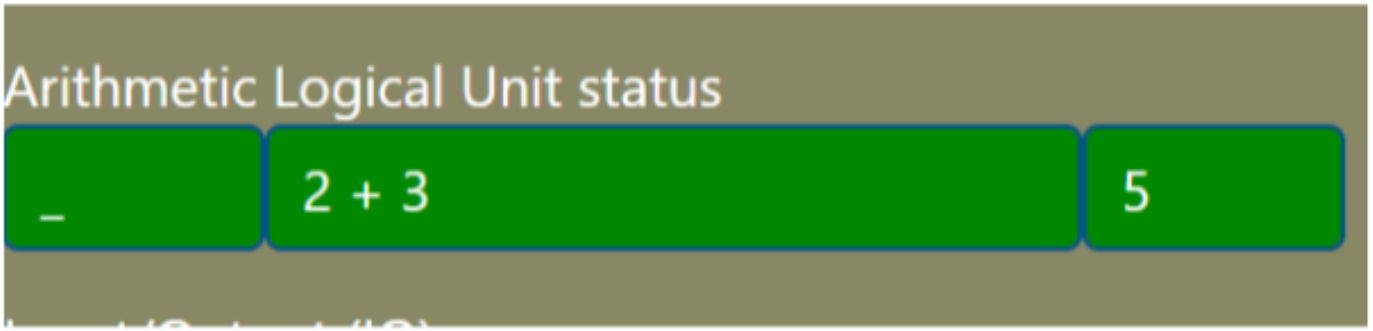
Figure 11.14—87 - first of the student feedback documents.



From a design perspective, the colour scheme and general interface is unintuitive and contrasts poorly. I think the hand drawn visuals of the "little man" are a nice touch, and help the user to visualise what the computer is doing. One comment on this would be that the "Information" text above the image is spelt incorrectly, reading "Infomation".

I think for new users to LMC, a more in-depth tutorial would be suitable, with walkthroughs on how to input data and understanding how each increment means or what the mess of numbers across the screen.

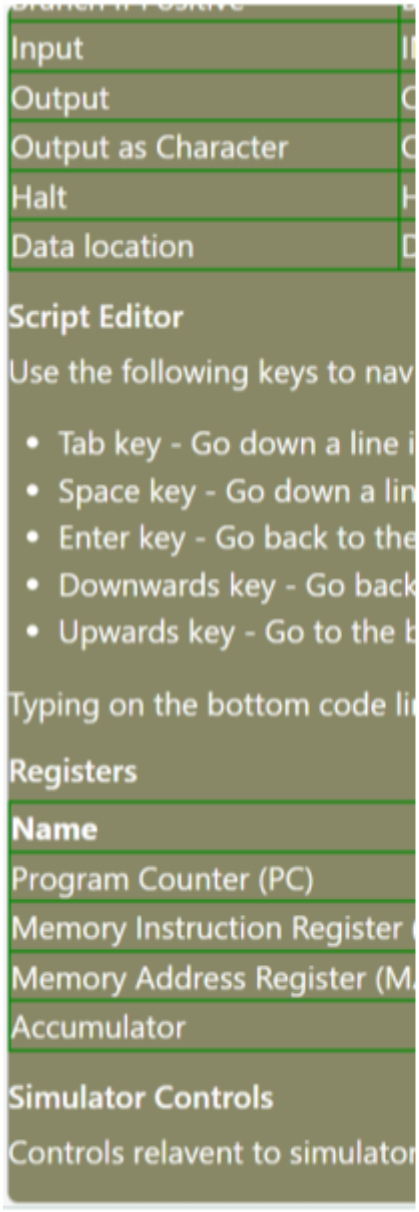
Another comment is that you can skip through the tutorials without completing them successfully. To improve, I think a failure counter would be suitable, which then gives the user additional information once they have made e.g. 3 failed attempts.



The general arithmetic of the site works properly and could help learners to understand how a computer goes through the FDE cycle.

Logically, the site works fine but would benefit from a new colour scheme (as I think it is inaccessible), better browser responsiveness, and spell checks. The padding on the manual page is hard to read on the screen used to test the sim and could benefit from a left indent of about 25-30px.

Figure 11.14—88 - first half of the second student feedback document.



^Screenshot taken from the very edge of the monitor, hard to read.

Figure 11.14—89 - second half of the second student feedback document.

11.14.2.4 Current PERT chart completion

Because the shareholder/main audience have tested and given feedback which were implemented, the “Creating the LMC code” is complete and now moves on the porting task.

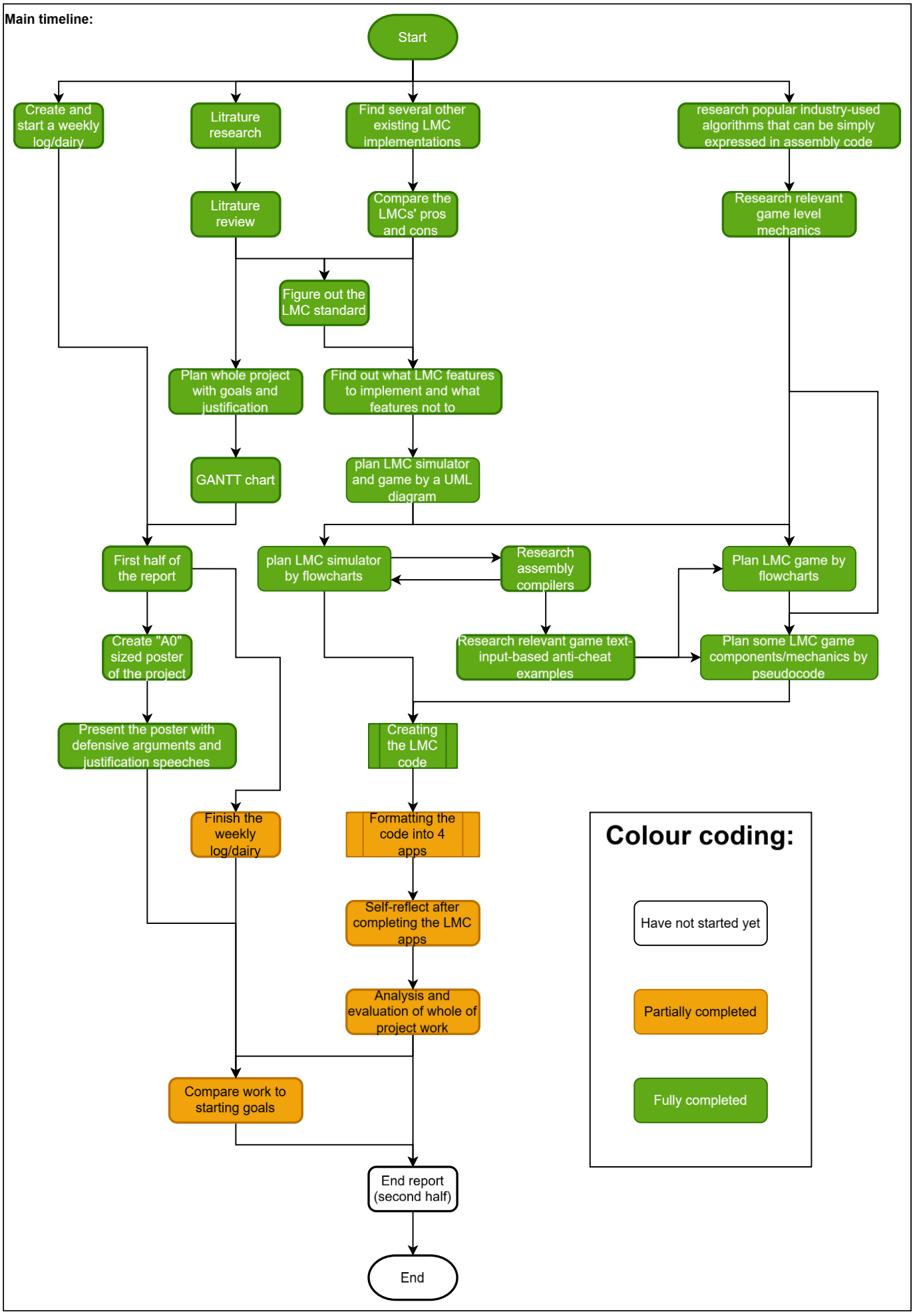


Figure 11.14—90 - current PERT chart progress at the time of after completing the feedback implementation.

11.14.3 Porting to offline application (post-sprints)

11.14.3.1 Planning and researching

11.14.3.1.1 Initial planning and research

Web applications use web technologies (CSS, HTML, and JS). In the industry of office and utility non-browser offline applications, most of them still use web technologies despite being downloaded applications. These include many big names such as: Discord, Visual Studio Code, Spotify, WhatsApp, Microsoft Teams, etc. All of these applications also have an online browser version. With both offline and online applications made using web technologies, it brings one of the most important advantages to have in projects – consistency. These advantages also apply to the LMC educational game project.

The consistency allows both online and offline applications to be made in the same codebase. Halving the required code while doubling development efficiency. In the LMC educational game, which will also be using web technologies - with TS compiling into JS.

Another main reason (which is two in itself) is the dependency ecosystem. Using the same technologies also benefits greatly for dependencies as you do not need to worry about finding a different package in each codebase/technology that does the same job, such as *jest*. As the author is using web technologies for both platforms, this will indeed not be a problem.

For the LMC educational game, despite web technologies being chosen mostly for the nature of the audience, JS is the most widely used language, and with it the largest ecosystems of dependencies and packages including package suppliers such as NPM. This is relevant as JS (as compiled TS) will be used for both platforms.

Alongside codebase consistency is the other advantage of a unified user experience – as the same web technologies are used, the backend and frontend are both identical across the offline and online platforms. This is very convenient for the user base as they can easily use one platform and get used to another platform without any difficulty and the author does not need to put much effort into doing this.

To compile an offline application with web technologies will need to *use a* cross-platform framework. When researching suitable frameworks, the author came across 2 very popular frameworks with their own way/architecture, each having advantages; those frameworks being Electron (Electron, 2025) and Tauri (Tauri, 2025). However, they do bring the significant downside of requiring platform-specific code.

The author decided to attempt to research and test ways to get around it. One way the author found in his research was to use a "HTML Application" (shortened to HTA) file (Dyer, 2010) instead of a normal HTML file. The author has made a testing folder with the HTA file with a linked CSS and JS file. When opening it, it does fully work. However, there are two problems. First is that the HTA has a slightly stricter interpretation than normal web files, but that can easily be fixed without any considerable changes. The second one is more significant. During deeper research, after the testing, the author discovered that HTAs are only runnable on Windows as an executable (*.exe*) (Dyer, 2010). Specifically, they require Microsoft HTML Application Host (identified as *mshta.exe*). There may possibly be another way for Linux and Mac, but they will just increase the number of platforms needing porting. 83% of computers uses Windows (from 2018) (Aladdin, Bakir and Saeed, 2018). Due to this and the assumption that the computers in the classroom will all be Windows, the author has decided to use HTA for porting as it is far simpler to implement. If the author has time, a framework can be used to port to other OSs (Linux and MacOS) and mobile app compatibility, but only as a lower priority.

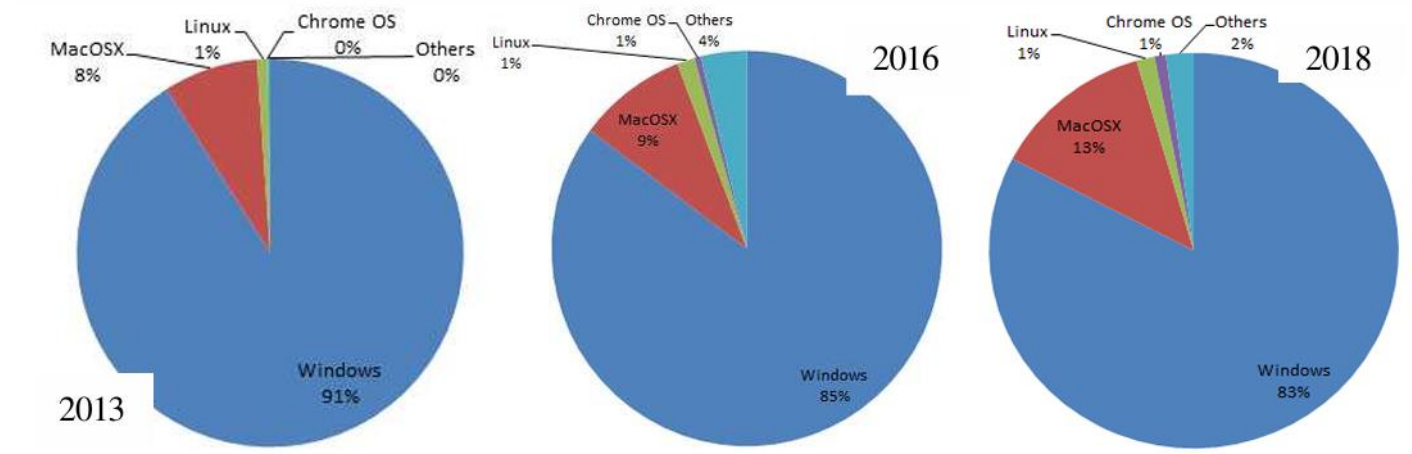


Figure 11.14—91 - pie chart comparisons of different kinds of OSs percentage-wise in 2013, 2016 and 2018 (Aladdin, Bakir and Saeed, 2018).

Going back to the frameworks. If the author was to use a framework, the author firmly believes that Electron is a far better option because it keeps true to the advantage of only using web technologies and no other, unlike Tauri which requires Rust, which is a

completely different technology and one that the author has absolutely zero experience in. For this reason, it alone beats all other comparisons, such as architecture and performance.

11.14.3.1.2 Changed planning and research

When planning to port the LMC educational game to offline using Windows HTML Application (HTA), the author realised that it only accepts only legacy versions of HTML, CSS and JS. Such legacy requirements include JS files not containing `let` nor `const` syntax but the old `var` syntax instead. To port the LMC educational game to revamp the entire codebase, which must be avoided at all costs to maintain the project's manageability. To get over the porting problem, the author decided to use Electron instead, porting the LMC educational game to a Windows executable.

11.14.3.2 Method and changes to codebase and file structure

11.14.3.2.1 Starting file

Since Electron allows a path to be specified for the starting displayed HTML file instead of just `index.html`, unlike GitHub webhosting, therefore `index.html` was not needed in the application at all (`home.html` gets loaded first).

11.14.3.2.2 File structure

In the app source folder (`electron_app`), only the assets, compiled, frames, and styles folders were included, which excludes the code folder. This is because, despite it being possible to use TS in Electron (using the *npm* `electron-prebuilt-compile` package), the program was originally made to execute JS to be browser compatible for the online version (Haddad, 2025b). The author insists on as few differences as possible in the code files (JS and CSS) between the online and offline platforms for a much simpler and faster development and maintenance, as well as minimising any platform-specific errors that would disrupt the consistency.

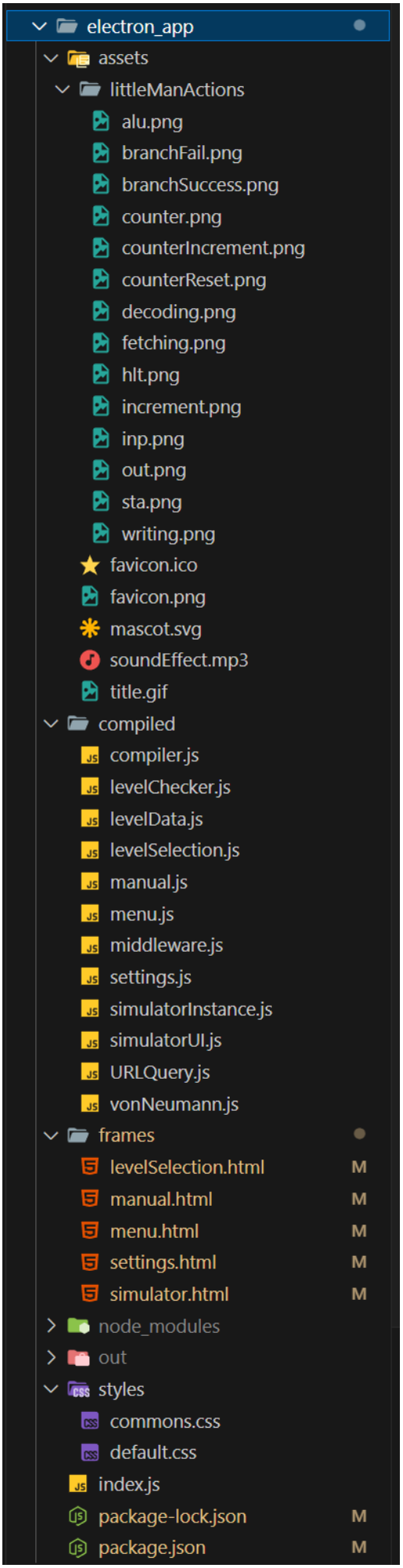


Figure 11.14—92 - file structure of `electron_app`.

11.14.3.2.3 Exclusive (additional) code

The only extra code, exclusive to the offline part, is **index.js** (not to be confused with **index.html** nor related). However, the file only mostly describes the nature of the application's windows, such as window dimensions and application icon. The other part is only 2 logic lines (which were done out of necessity), where one forces an 80% zoom and the other one ensures only one instance of the program is launched. More information regarding these logics is explained in **Default 80% zoom** and **Double launching** parts of the **Unexpected and solved problems** section.

11.14.3.2.4 Edited/changed code

There is only one section of code, across the HTML files, that was changed (instead of just added), which was the Bootstrap hyperlink reference. This was a necessity as one of the main points of having offline software was the ability to use it without an internet connection. To make the Bootstrap CSS framework work offline, it had to be installed in the **electron_app** using the command *npm install bootstrap@5.3.3*. After that, every instance of the online HTTPS reference (in **electron_app**) was replaced by the local path reference, as seen below.

```
6     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" >
```

Figure 11.14—93 - original HTML code for importing the Bootstrap CSS framework via HTTPS request (over the internet).

```
6     <link rel="stylesheet" href="../node_modules/bootstrap/dist/css/bootstrap.min.css">
```

*Figure 11.14—94 - HTML code for importing the Bootstrap CSS framework, in **electron_app**, via using the locally installed npm package in **node_modules**.*

Because the Bootstrap package (*bootstrap*) is part of the Electron-generated executable app, it (*alongside electron-squirrel-startup*) is not downloaded in the developer dependencies (just dependencies).

This process happened completely smoothly without any problems, thanks to the author's prior research.

11.14.3.2.5 Management of offline platform

To keep the offline and online platform dependencies and development tools separate, **electron_app** has its own **packet.json** for managing and configuring the offline platform packages and scripts.

Such a file can be seen below:

```
{
  "name": "lmc_offline_edition",
  "version": "1.1.4",
  "description": "LMC educational simulator game ported to Windows executable",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "electron .",
    "package": "electron-forge package",
    "make": "electron-forge make"
  },
  "author": "James R. Haddad",
  "license": "ISC",
  "devDependencies": {
    "@electron-forge/cli": "^7.8.1",
    "@electron-forge/maker-squirrel": "^7.8.1",
    "electron": "^36.3.1",
    "electron-forge": "^5.2.4",
    "electron-prebuilt-compile": "^8.2.0"
  },
  "config": {
    "forge": {
      "packagerConfig": {
        "icon": "./assets/favicon.ico"
      },
      "makers": [
        {
```

```
          "name": "@electron-forge/maker-squirrel",
          "config": {
            "name": "lmc_offline_edition",
            "authors": "James R. Haddad",
            "setupIcon": "./assets/favicon.ico",
            "setupExe": "LMC Sim Game.exe"
          }
        }
      ]
    }
  },
  "dependencies": {
    "bootstrap": "^5.3.3",
    "electron-squirrel-startup": "^1.0.1"
  }
}
```

Due to JSON's non-allowance of comments, none can be inserted.

11.14.3.2.6 index.js

To know the exact extra code, for specifying how the Electron application will load, it will be displayed here alongside code commenting:

```
//: importing dependencies
const { app, BrowserWindow, screen } = require('electron');
const path = require('path');

if (require('electron-squirrel-startup')){ app.quit(); };
//^ prevents multiple instances of the offline application from starting up.

app.whenReady().then(() => {
    const { width, height } = screen.getPrimaryDisplay().workAreaSize;
    //^ to set app window's dimensions near size of user's screen (excluding the taskbar)
when not maximised
    const win = new BrowserWindow({
        width,
        height,
        icon: path.join(__dirname, 'assets', 'favicon.ico'),
        fullscreen: false,
        //^ very inconvenient for user if fullscreen and want to exit application
        autoHideMenuBar: true,
        //^ the program does not use the menu bar hence it being hidden
        frame: true
        });
    const menuPath = path.join(__dirname, 'frames', 'menu.html');
    //^ first loaded page (can skip index.html as how user types the URL is not a conern
for the offline port)
    win.loadFile(menuPath);
    win.webContents.on('did-finish-load', () => { win.webContents.setZoomFactor(0.8); });
    //^ specified here because the CSS ".zoom-out { transform: scale(0.8); }" works on
local and GitHub webhosting but not on the electron Chromium-based app
    win.maximize();
    //^ maximise for full utilization of window space
    ///win.webContents.openDevTools({ mode: 'right' });
```

```
    //^ for debugging purposes only
});
```

11.14.3.3 Showcase

```
  Emperor Haddad@DESKTOP-QL3OA7G MINGW64 ~/Desktop/LMC_simulator_game (main)
● $ cd electron_app/

  Emperor Haddad@DESKTOP-QL3OA7G MINGW64 ~/Desktop/LMC_simulator_game/electron_app (main)
● $ npm run make

  > lmc_offline_edition@1.1.1 make
  > electron-forge make

  ✓ Checking your system
  ✓ Loading configuration
  ✓ Resolving make targets
    › Making for the following targets:
  ✓ Running package command
    ✓ Preparing to package application
    ✓ Running packaging hooks
      ✓ Running generateAssets hook
      ✓ Running prePackage hook
    ✓ Packaging application
      ✓ Packaging for x64 on win32 [5s]
    ✓ Running postPackage hook
  ✓ Running preMake hook
  ✓ Making distributables
    ✓ Making a squirrel distributable for win32/x64 [46s]
  ✓ Running postMake hook
    › Artifacts available at: C:\Users\Emperor Haddad\Desktop\LMC_simulator_game\electron_app\out\make
```

Figure 11.14—95 - commands used to successfully port the LMC educational game to a Windows executable offline application.

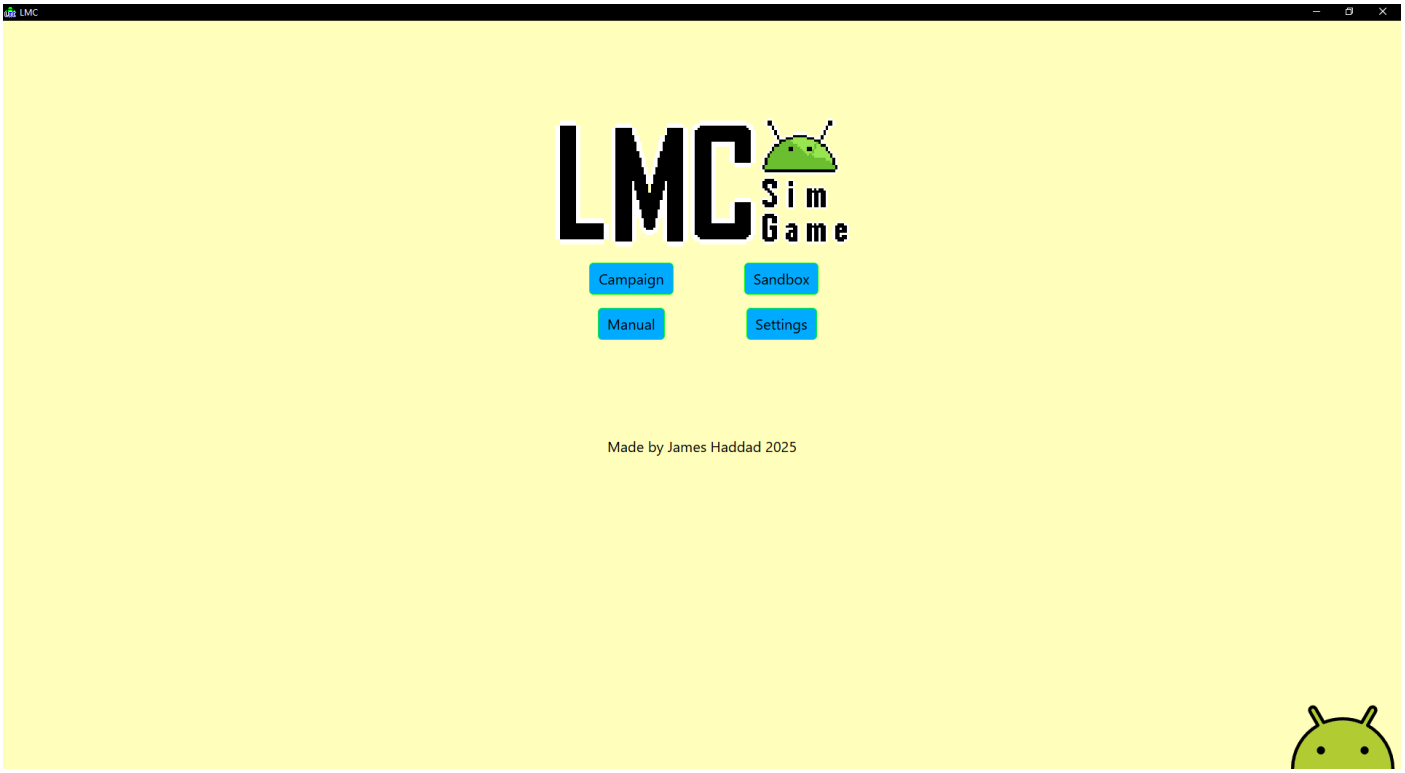


Figure 11.14—96 - offline part application when opened (displaying the menu).

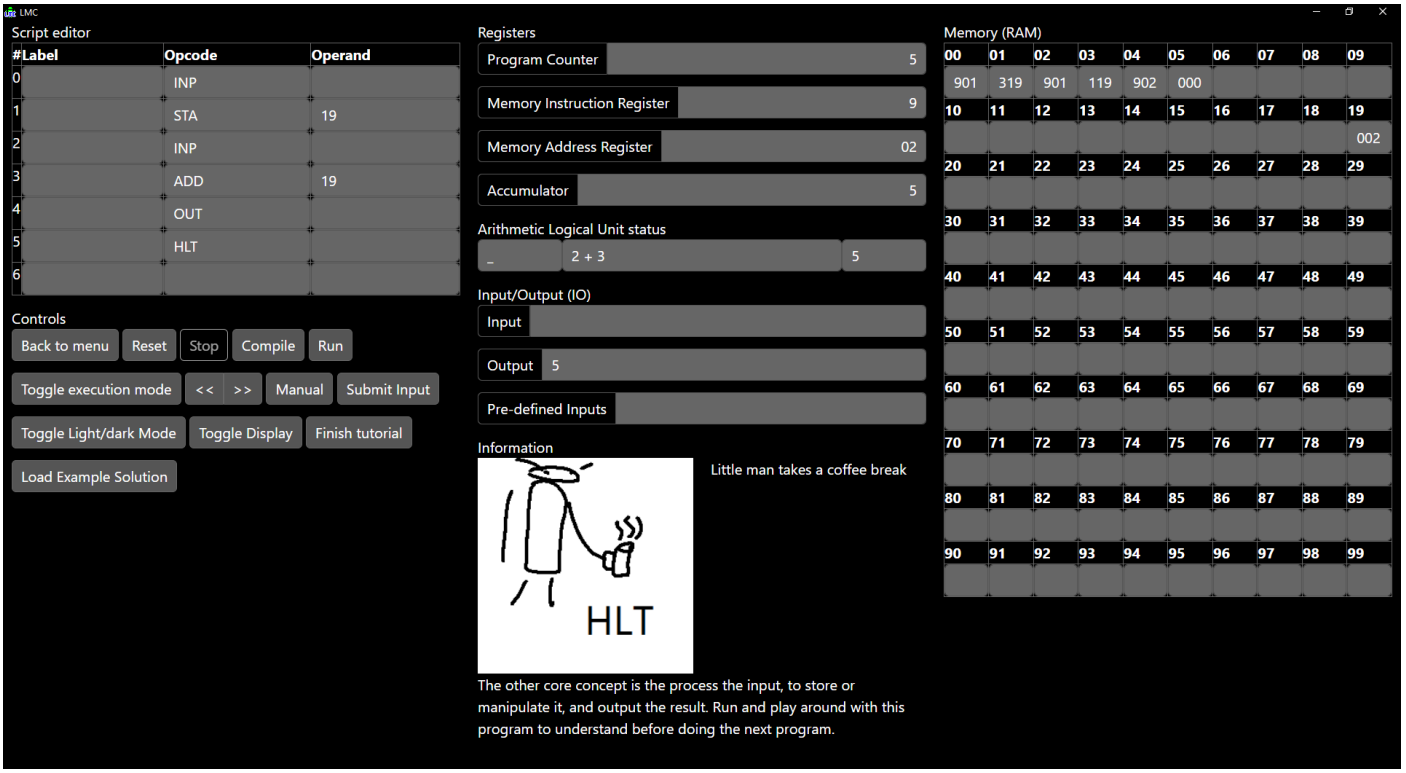


Figure 11.14—97 - offline port application demonstrating executing a campaign level script with settings set to grayscale dark mode.

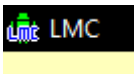


Figure 11.14—98 - offline port application displaying name and logo in its window bar.

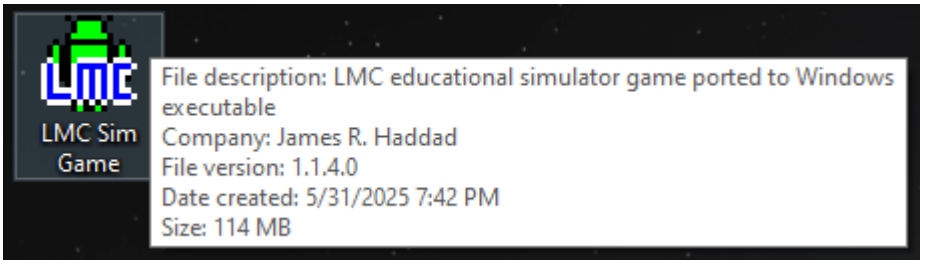


Figure 11.14—99 - the Windows executable file that is the offline port application.

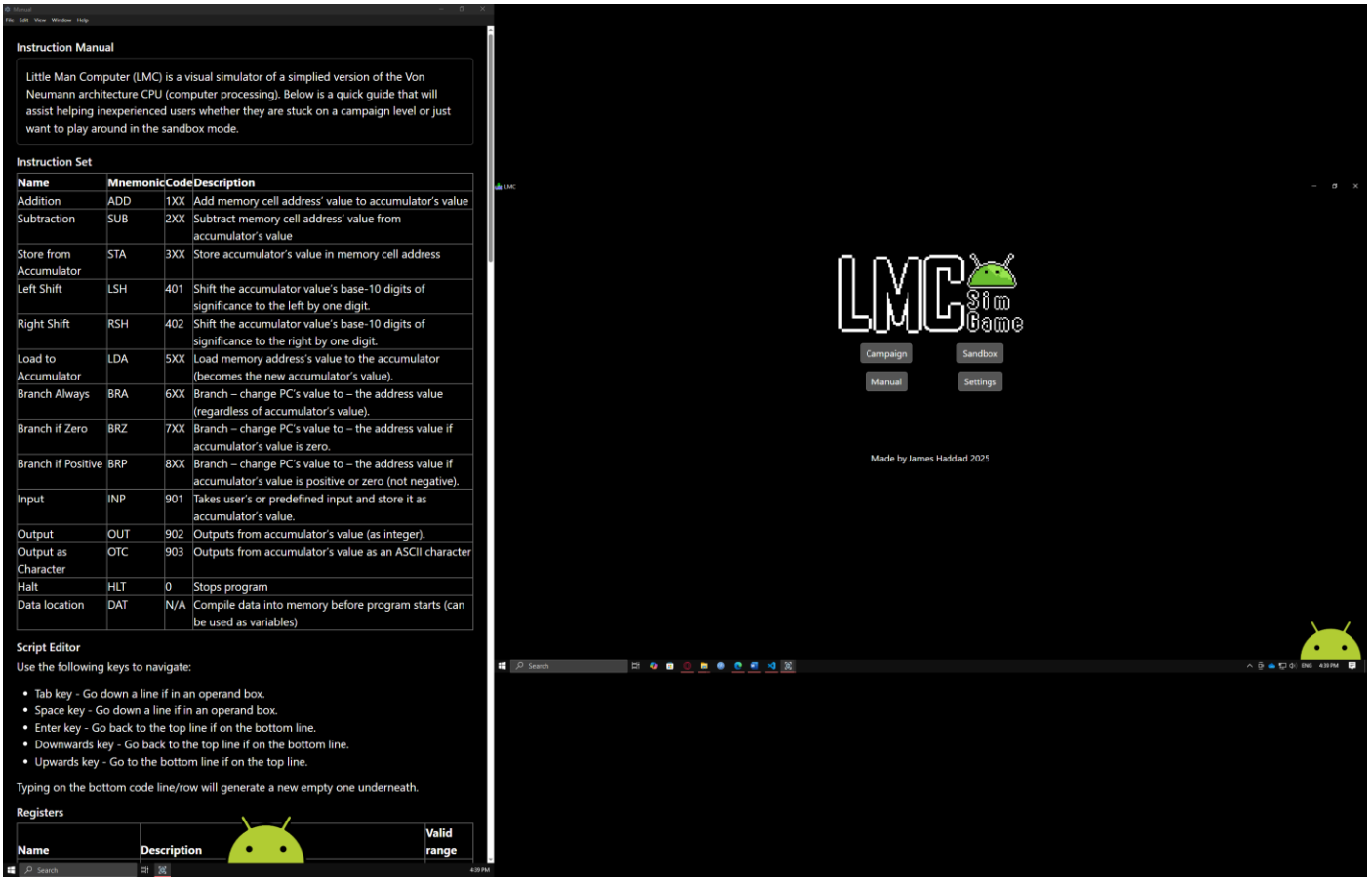


Figure 11.14—100 - opening the manual page in new window with same settings as the main window (each displayed in a different monitor for better view).

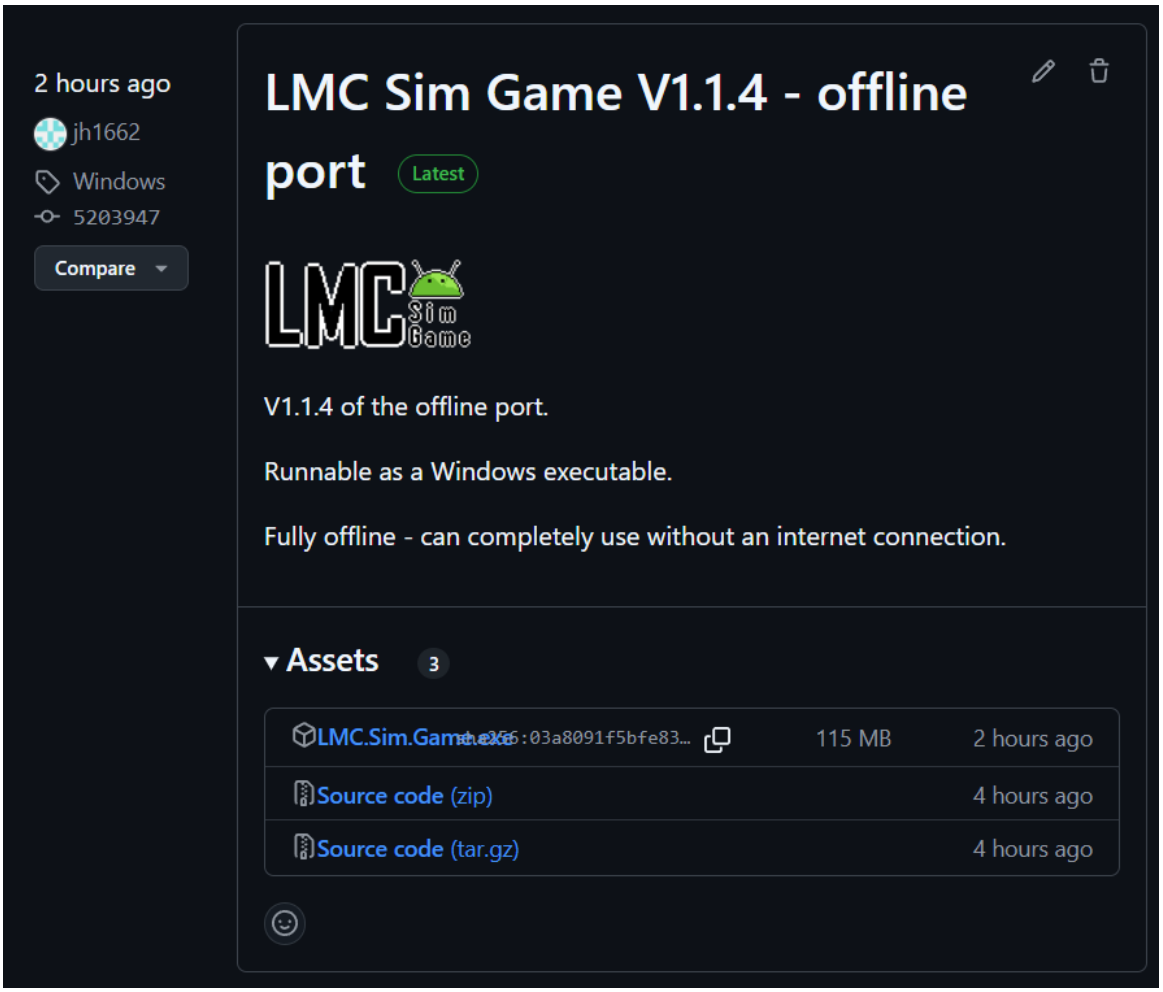


Figure 11.14—101 - upload of offline port to GitHub as a release of the LMC Education Game repository (Haddad, 2025c).

11.14.3.4 Unexpected dependencies

From research, the author thought that only Electron (runtime) (Electron HQ and Electron Nightly, 2025) and Electron-Forge (porting) packages are needed to port to an offline application, but there are other required dependencies:

- `@electron-forge/cli` (version 7.8.1) (Lee *et al.,* 2025a) – the provided CLI allows running the npm scripts. Such scripts as `"start": "electron ."`, for runtime testing, and `"make": "electron-forge make"` to port to an offline application.
- `@electron-forge/maker-squirrel` (version 7.8.1) (Lee *et al.,* 2025b) – required for making the offline application a Windows distributable (`.exe`) instead of just a zip file.
- `electron-prebuilt-compile` (version 8.2.0) (Betts *et al.,* 2020) – provides hassle-free compatibility with modern JS syntax/functionality (such as `import` and `export`), including *ES2020,* which the author uses.
- `electron-squirrel-startup` (version 1.0.1) (Rueckstiess *et al.,* 2024) – not necessary for offline Windows executable app to work but helps it do so properly, such as only opening one app instance instead of two (one displayed, the other one installing in background). This is further explained in the **Double launching** section. Unlike the other Electron-related dependencies, this one cannot be installed as a developer dependency (`devDependencies`) as the runtime makes active use of it .

```
✓ Making distributables
  ✓ Making a squirrel distributable for win32/x64 [52s]
✓ Running postMake hook
  › Artifacts available at: C:\Users\Emperor Haddad\Desktop\LMC_simulator_game\electron_app\out\make
```

Figure 11.14—102 – required use of "squirrel" to make a Windows distributable using `@electron-forge/maker-squirrel`.

```
 8      "start": "electron .",
 9      "package": "electron-forge package",
10      "make": "electron-forge make"
```

Figure 11.14—103 - npm scripts that requires `@electron-forge/cli` *to work*

These packages were not automatically downloaded alongside Electron or Electron-Forge which was highly unusual for downloading *npm* packages. This included having to download Electron separately after Electron-Forge as the author noticed in his research.

11.14.3.5 Unexpected and solved problems

11.14.3.5.1 Default 80% zoom

`.zoom-out { transform: scale(0.8); }` works on local (using the *live server* VS extension) and GitHub webhosting (for zooming out to 80%) but not on the built electron Chromium-based app, this was fixed by adding the zoom specification to **`index.js`** by adding `win.webContents.on('did-finish-load', () => { win.webContents.setZoomFactor(0.8); });`.

11.14.3.5.2 Lack of window controls

Setting the app to full screen by default (using `fullscreen: true` in **`index.js`**) allows more screen space for the problem but has proven to be very difficult to exit unless knowing the F11 key bind.

11.14.3.5.3 Unable to package due to unsettable icon

Unlike local and GitHub webhosting, Electron packages only accept icon file (`.ico`) format, instead of PNG (`.png`), for the offline ported application's icon, as seen below.

```
Emperor Haddad@DESKTOP-QL3OA7G MINGW64 ~/Desktop/LMC_simulator_game/electron_app (main)
$ npm run make

> lmc_offline_edition@1.1.1 make
> electron-forge make

✓ Checking your system
✓ Loading configuration
✓ Resolving make targets
  › Making for the following targets:
✓ Running package command
  ✓ Preparing to package application
  ✓ Running packaging hooks
    ✓ Running generateAssets hook
    ✓ Running prePackage hook
  ✓ Packaging application
    ✓ Packaging for x64 on win32 [13s]
  ✓ Running postPackage hook
✓ Running preMake hook
❯ Making distributables
  ✖ Making a squirrel distributable for win32/x64
    › Failed with exit code: 1
    Output:
    Fatal error: Unable to set icon
◼ Running postMake hook

(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.
(node:17896) [DEP0174] DeprecationWarning: Calling promisify on a function that returns a Promise is likely a mistake.

An unhandled rejection has occurred inside Forge:
Error: Failed with exit code: 1
Output:
Fatal error: Unable to set icon
at ChildProcess.<anonymous> (C:\Users\Emperor Haddad\Desktop\LMC_simulator_game\electron_app\node_modules\electron-winstaller\lib\spawn-promise.js:48:24)
    at ChildProcess.emit (node:events:519:28)
    at ChildProcess.emit (node:domain:488:12)
    at maybeClose (node:internal/child_process:1105:16)
    at ChildProcess._handle.onexit (node:internal/child_process:305:5)

Emperor Haddad@DESKTOP-QL3OA7G MINGW64 ~/Desktop/LMC_simulator_game/electron_app (main)
```

Figure 11.14—104 - Electron-forge error where it was "Unable to set icon".

Because of this, the author simply converted the PNG to an icon file.

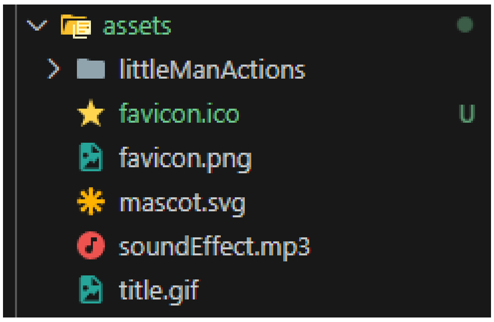


Figure 11.14—105 - icon file alongside the PNG of the icon.

The PNG file was kept to maintain code consistency as the HTML files refer to the PNG for the favicon.

11.14.3.5.4 Double launching

For some reason, when opening the offline application, two instances are launched (which can be seen below). One opens almost immediately, while the other loads when the first one is manually closed or closed automatically after a while.

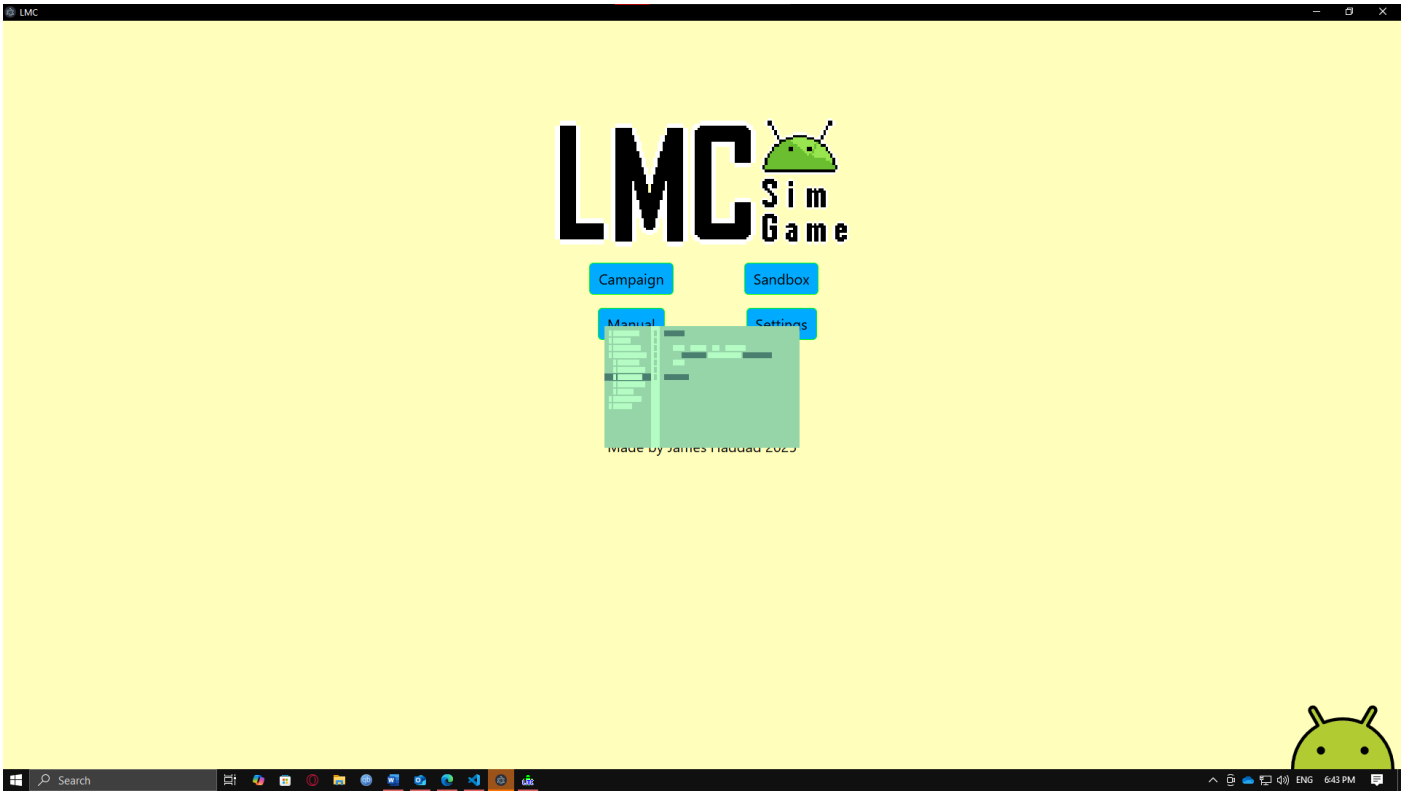
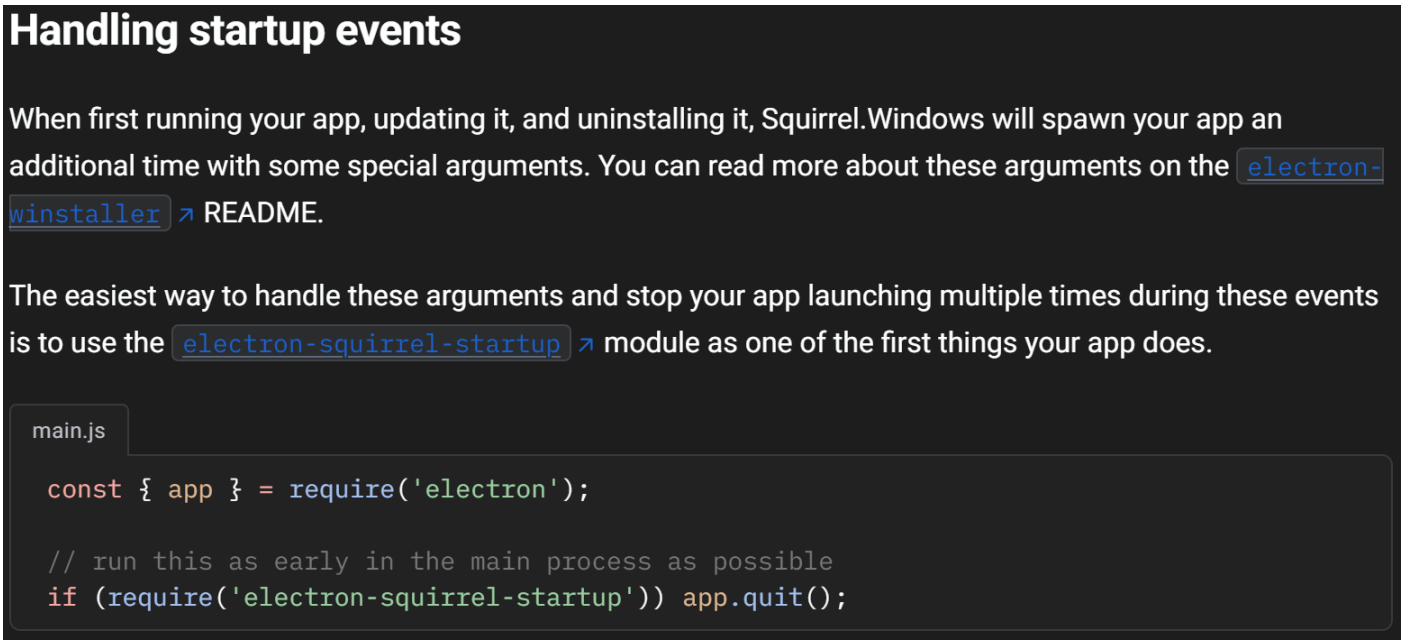


Figure 11.14—106 - one instance of the program is displayed while the other is loading, as dictated by the green box in the middle of the screenshot.

According to the author's research, this is because the application launches and gets installed in the background. This is not ideal because the application must be convenient for the user, which means quick opening of the app and portability by no installation set-ups – just one executable to keep the app portable – as per its planned lightweight characteristic. Fortunately, the same research did also mentioned how to prevent such a scenario as seen below – adding the if-statement `if (require('electron-squirrel-startup')) app.quit();` (Electron Forge, 2025) to **`index.js`**.



Handling startup events

When first running your app, updating it, and uninstalling it, Squirrel.Windows will spawn your app an additional time with some special arguments. You can read more about these arguments on the `electron-winstaller` ↗ README.

The easiest way to handle these arguments and stop your app launching multiple times during these events is to use the `electron-squirrel-startup` ↗ module as one of the first things your app does.

main.js

```
const { app } = require('electron');

// run this as early in the main process as possible
if (require('electron-squirrel-startup')) app.quit();
```

Figure 11.14—107 – the research showing explanation and solution for preventing multiple application instance start-ups from the official Electron Forge documentation (Electron Forge, 2025).

At first, this seems not to work due to being unable to deploy at prototyping runtime (npm run start), and when opening the application, it loads then immediately closes. After deeper research, the author found that it was due to not having a dependency called `electron-prebuilt-compile`. After that, not only did it worked (only running one instance) but does so at an unexpectedly fast speed.

The use of `electron-prebuilt-compile` did stop the favicon from being displayed as the application's, and its window's icon. This was overcomed by simply adding `icon: path.join(__dirname, 'assets', 'favicon.ico')` to **`index.js`**.

11.14.3.5.5 Audio file

Unlike the online version, the audio file in the offline port only sometimes works. After investigation, it appears that the sound effect only works when the button does not refer to another page nor refreshes the current page. This difference is simply because of how Electron handles resource loading differently from GitHub's web hosting. As its primary purpose (of the sound effect is for audio feedback when using the simulator) is not affected, it is not really a concern to the author due to the change in user experience being negligible.

11.14.3.6 Review and conclusion

This review and conclusion are exclusive to **Porting to offline application (post-sprints)**. To see the general conclusion, see the **Conclusion Chapter**.

Everything went smoothly, the author has successfully ported the LMC education game to the offline desktop platform without any considerable disruptions using Electron.

However, due to using a cross-platform desktop application framework, the author did spend considerable time researching and learning as the author previously had no experience with a cross-platform application framework, as seen with the many parts in **Unexpected dependencies** and **Unexpected and solved problems**.

Because of this, the author did not have time to learn and perform on other platforms of mobile online and mobile offline, let alone the other desktop OSs. Electron can cross-platform to other OSs but requires different dependencies, different methods, and different configurations. One such difficulty is that, unlike Windows, Linux requires other packages and dependencies to be downloaded externally – not using *npm* – and cannot use VS Code's integrated terminal.

Despite that, the author has met the preferred expectations that were planned as "could have"; thus, it went well.

11.14.3.6.1 Current PERT chart completion

With the porting done, the author is done with the project. At first the reader may be surprised by not all tasks being completed, as seen below, but the reader should not worry. This is because this PERT chart takes every aim into account, even the lower priorities, such as the porting to mobile offline "would have" MoSCoW entry – meaning that this incompleteness is very much expected.

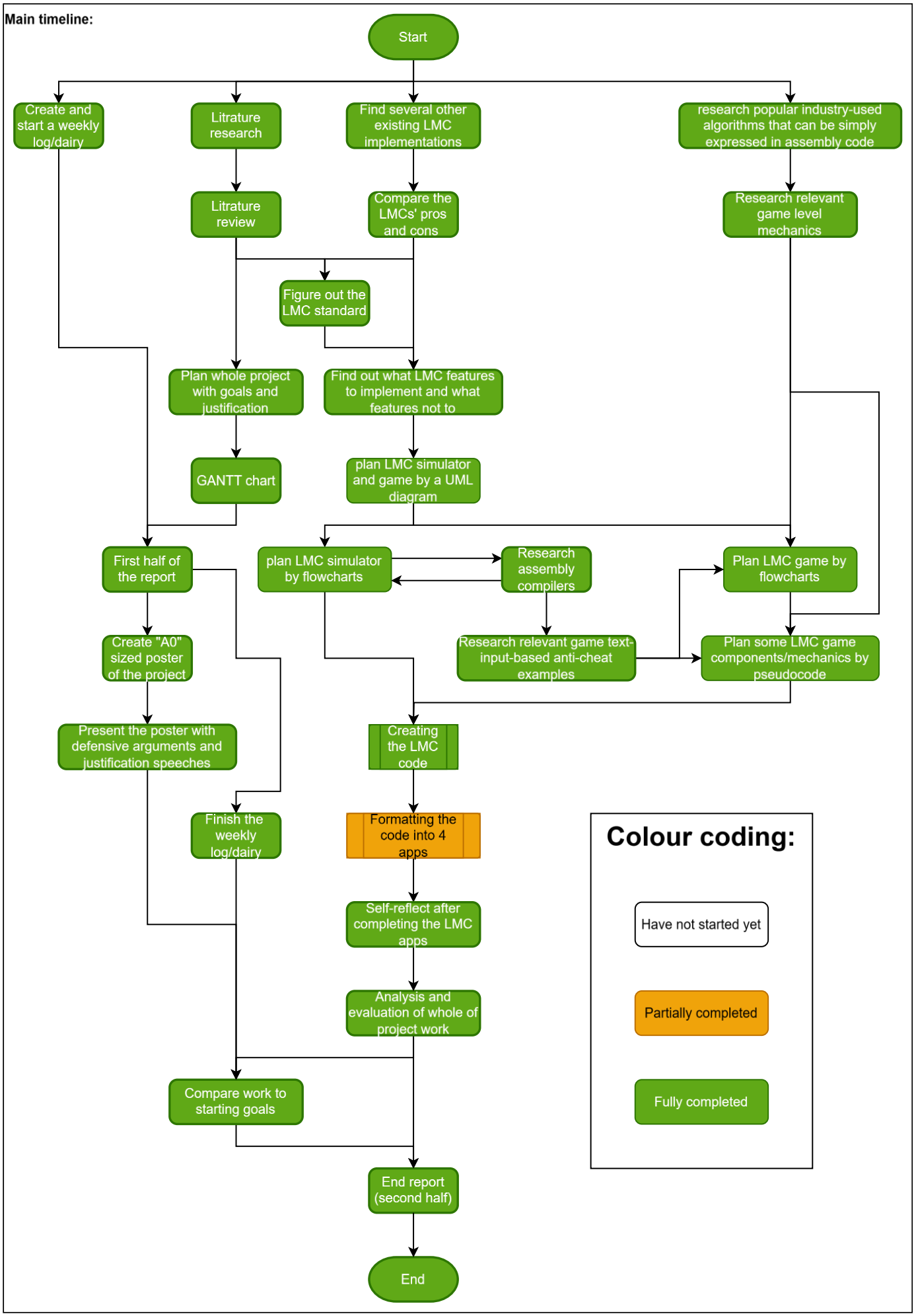


Figure 11.14—108 - current PERT chart progress at the time of after completing the offline Windows porting.