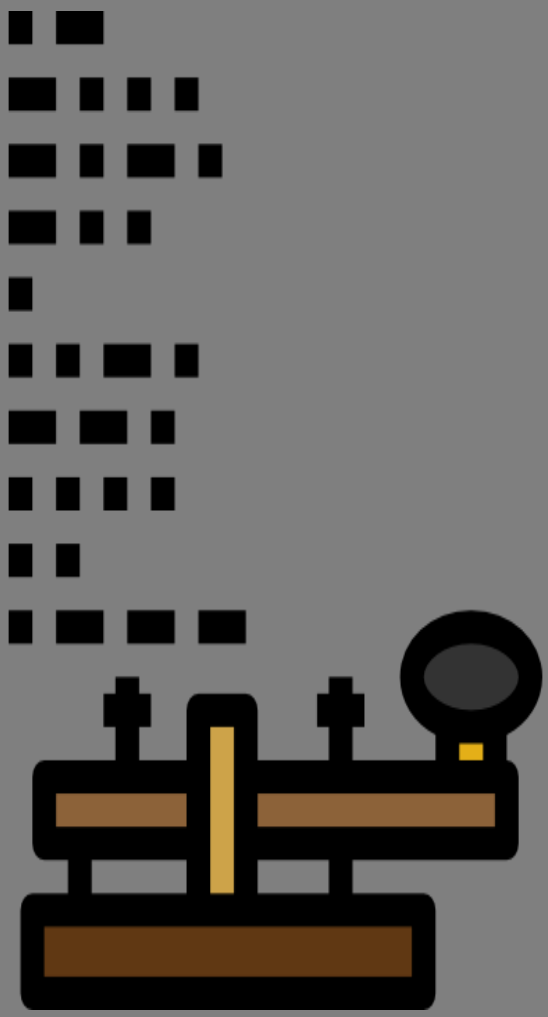
THE MORSE MASTER

~ LEARN TO SAVE THE WORLD ONE DOT/DASH AT A TIME



A MORSE GAME DEVELOPED BY GROUP 12 …

PALAK AGGARWAL

DYLAN GALLAGHER

FAREEDAH HAFIS-OMOTAYO

STEPHEN KOMOLAFE

MATTHEW POWER

INTRODUCTION AND BACKGROUND

About 200 years ago, the very first electrical telegraph system was invented by Alfred Vail, Samuel Morse and Joseph Henry. The problem with the system was there was no way to communicate messages, all it could send were electrical pulses. Samuel Morse found a way to interpret these electrical pulses that were being sent over a long distance to messages that could be understood at the receiving end. This is what we now know as the Morse code. Morse code is an encoding system used to represent alphanumeric characters or symbols using a series of dots, dashes and space. So the length of the electrical pulse determined if the input was a dot or a dash. No electrical pulse was assumed to be a space.

A sequence of dots and dashes separated by spaces is assigned to each number and letter. A chart containing all the sequences and their corresponding numbers or letters was agreed upon and is now known as the international Morse code. The sequences were developed based on how frequently the letters or numbers were being used. The most used letters such as the letters 'E' and 'I' were assigned short and easy sequences. Letters that are used less frequently were assigned longer sequences. There are many versions of the Morse code but the most widely used and recognized version is the international Morse code.

Figure 1

The way it works is each dot acts as the smallest or most basic unit of time in the system and one dash is equivalent to the time it would take to input 3 dots. A space is also equivalent to the same time as a dot.

For example, if sending an electrical pulse for one second generates a dot, then sending the pulse for 3 seconds would generate a dash and no pulse for one second would generate a space. Since the timing is relative to one another, it is possible to speed up or slow down the morse message. Morse code is still used to this day although not nearly as much as before. It is used by amateur radio operators and the U.S. Navy

PROJECT OVERVIEW

The main objective of this project is to develop a game that tests its players on their knowledge of the Morse Code. The game is meant to be a fun and interactive way for the players to learn Morse code. The game was built entirely using C programming and ARM assembly. The user inputs for the game are obtained via the GP21 button on the MAKER-PI-PICO board. The game should have two basic levels that involve inputting alphanumeric characters and two harder levels that involve inputting words. There are additional features that should be added to the game such as a life counter and RGB LED controls. The project serves as a way to combine knowledge of low-level programming with hardware interaction. As well as focusing on the development of the game, the project also encourages teamwork and collaboration. For this reason, various team roles that handle different parts of the project were assigned. The project also required comprehensive documentation and demonstration of the final game which is all covered in this report. Lastly, a proficient knowledge of Git for version control and collaboration was required.

TEAM ROLES AND CONTRIBUTION

The following individual project roles were allocated amongst the members of the group. The team roles are as follows

THE PROJECT WORKFLOW OWNER (STEPHEN KOMOLAFE)

As the project workflow owner, held the responsibility of managing and overseeing the workflow of the project. We completed our assigned tasks promptly, so I may not have been as heavily involved in overseeing everyone's progress as anticipated, however, I made sure to contribute by providing guidance and support to my team members as much as I could. Working closely with my peers, I provided insights and suggestions that influenced the approach we took in making our game. Though my role may have been more supportive than managerial, I remained engaged in the project's development, helping where I could and contributing to the overall direction of our efforts.

In addition to being the owner of the project workflow, I also helped write the game's assembly and C code. In C, I assisted in implementing a functional means of level selection, the conditional statements for switching between level difficulty, the output displaying the current level and stage as well as forming a way to receive user input during gameplay. In terms of the ARM assembly side of the program, we used a slightly modified version of my ARM code from assignment 1 as a base, as it already contained the necessary initialisations for the GP21 button and the alarm interrupt. With said ARM code, I also incorporated some basic Morse code functionality—which was continuously developed and improved upon throughout the course of the project—to distinguish between dots, dashes, and spaces as they were read from the board.

THE PROJECT GitLab OWNER (DYLAN GALLAGHER)

As the project GitLab owner, I was responsible for managing and overseeing the shared GitLab project repository used by the team throughout the project. I was responsible for managing the different branches, overseeing the merges to the main and testing each push to the main before deploying. This was vital to ensure that no breaking changes were pushed to the main and that we always had a working version of the code in the main. As a result of this, we never had to roll back to previous versions of the code. This saved us a lot of time.

As well as being the project GitLab owner I also contributed to the code by carrying out specific tasks assigned to me by the project code owner. I created the game logic and skeleton files for each of the main functions. The game logic defined the execution flow of the program and had the logic for managing levels, lives and game over states. I was also responsible for getting the RGB LED working, and the PIO files and CMakeList.txt files to go with it. On top of this, I also assisted Matthew (the code owner) with a lot of the debugging of the code.

THE PROJECT DOCUMENTATION OWNER (FAREEDAH HAFIS-OMOTAYO)

As the project documentation owner, I was responsible for managing and overseeing all documentation that was related to the project. I created and compiled the final project report and made it accessible to all members of the team. I was responsible for setting up the structure and template of the report. This made it easier to ensure that each person was given an equal opportunity to detail their contribution to the entire project via the final report. My role was important as It made it possible for us to compile a report that reads as a single unified document.

As well as being the project documentation owner, I also contributed to the final code by carrying out specific tasks assigned to me by the project code owner. My task involved designing and writing code for printing all the screens used in the game. This included the ‘welcome’ screen, the ‘game over’ screen and the ‘game-winner’ screen. This process was done by using ASCII art to make the preliminary design and then turning the ASCII art into C code. This was important and it improved the user interface and made the game more realistic as a normal game would have all three screens and even more.

THE PROJECT DEMONSTRATION OWNER (PALAK AGGARWAL)

As the project demonstration owner, I was responsible for creating a 90-second video that demonstrated our final Morse Code. I ensured that the demo video showcased the unique learning experience provided by the Morse code game. The demo covered the game’s initial display and setup, the core gameplay where players translated the Morse code and finally the feedback mechanisms that let the player know how well they are doing in the game. I made sure that the demo had clear visuals and a concise voiceover

In addition to my role as the project demonstration owner, I also contributed to the project's codebase. I implemented print statements throughout the game to provide users with clear instructions, feedback, and results of their actions. This task was crucial for user interaction, making the game accessible and easy to understand for beginners. I also developed a function that randomly generates Morse Code combinations for users to input. It helped to ensure that the game remains engaging and tests the player on a wide variety of Morse combinations. I worked alongside the code owner to Implement a timer which added to the competitive element of the game. It encouraged users to improve their speed and accuracy while playing the game. Lastly, I compiled a comprehensive list of Morse Codes for all alphabets and numbers. This list was crucial to the game's operation, as it served as a reference for generating puzzles and verifying user inputs.

THE PROJECT CODE OWNER (MATTHEW POWER)

As the project code owner, I was responsible for overseeing the entire code development. I ensured that the final Morse Code game met all requirements and functioned as intended. I was also responsible for ensuring that everyone else in the team contributed in different ways to the final code. This was mostly done by assigning specific tasks to each member of the team.

It was essential to organise cooperation and documentation as the code owner to ensure everyone was able to contribute fairly to the project. The brunt of the programming was undertaken by myself, although assigning tasks to everyone equally was done. Different people worked on many different aspects while I stitched, debugged and tested the incoming code.

The software's architecture and logic were designed by myself, with strings originally being sent from the arm to the C part of the code. However, this needed to be redesigned as a hard fault was encountered whenever a null terminator was sent. This was remedied by sending integers and inputting them into an array based on those values, a string for a dot, dash or space was matched and then a generated sequence was then created. String compare was then used to check if they were the same.

Overall all members contributed equally to the final project and worked well together.

GAME DESIGN

The design of the game is detailed below in a step-by-step manner.

STARTING THE GAME

- When the game is launched, the welcome screen which contains brief instructions on how to play the game is shown
- The RGB LED is set to blue to show that the game has not started
- The player is given the choice to pick the level they want to enter.
- Once they choose the level that they want to play at, the game begins

PLAYING THE GAME

- At the start of each level, each player is given 3 lives.
- When the game is started, an alphanumeric character is shown on the screen and it is up to the player to input the corresponding Morse code sequence for the character using the GP21 button (short press for a dot, short pause for a space and a long press for a dash)
- Once the game is started, the RGB LED is set to green and shows the number of lives the player has.
- Following the player's input, if the Morse code sequence is input correctly then the player gains a life, otherwise, they lose a life and the RGB LED is updated accordingly.
- If the player has 0 lives left then they automatically lose the game, the game over screen is shown and the RGB LED is set to red.
- If the player completes 5 correct sequences in a row then the player moves on to the next level.
- If the player completes 5 correct sequences in a row and they are on the final level, the player wins the game and the 'game-winner' screen is shown.

GAME LEVELS

The game is designed to have 4 levels.

- Level 1 (Introductory Level): The alphanumeric character as well as its Morse code is displayed on the screen so all the player is required to do is input the Morse code shown on the screen
- Level 2 (Beginner Level): This level is slightly harder than level 1 as the player is only shown the alphanumeric character without the corresponding Morse code.
- Level 3 (Intermediate Level): This level is slightly harder than level 2. Here, rather than a single alphanumeric character, a word is displayed on the screen as well as its Morse code.
- Level 4 (Advanced Level): This level is the hardest as words are displayed without their corresponding Morse code.

GAME DESIGN FLOWCHART

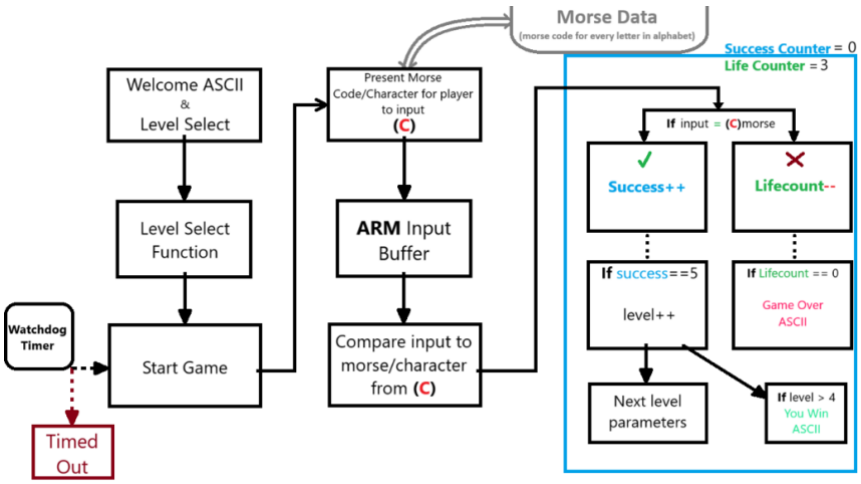


Figure 2: A flowchart describing the C side of the game's design

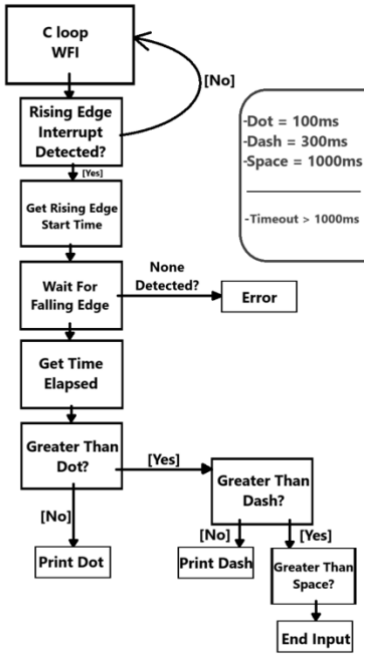


Figure 3: A flowchart describing the ARM Assembly side of the game's design

OPTIONAL FEATURES

An optional feature that was added to the game is the game statistics. It improved the user experience and provided feedback to the player. The statistics printed include the lives lost by the player, the lives gained and the number of sequences generated. These statistics were displayed on the win screen and the game over screen after the game had been completed.

CODE AND IMPLEMENTATION

As has been mentioned previously, the C programming language was used alongside the ARM assembly language. Each code played a specific role and focused on a different aspect of the game's functionality.

C CODE

The C code focused on the higher-level game logic, structure and user interactions. It defined the main structure of the game. This included the game levels, progression and sequence generation. The key functions used in the code and their roles are highlighted below.

- **GPIO (asm_gpio_init, asm_gpio_set_dir, asm_gpio_get & asm_gpio_put):** These functions were used to handle the GPIO pins. **asm_gpio_init** initialized GPIO pins, **asm_gpio_set_dir** set the direction of the pin to input or output, **asm_gpio_get** read the value of the pin and **asm_gpio_put** set the value of the pins. These functions made it possible to interact with the hardware.
- **TIMING (time_check1, time-check2 & time_diff): time_check1** takes note of the time when a button is pressed, **time-check2** takes note of the time when the button is released and **time_diff** finds the difference between both times. These functions make it possible to distinguish between a dot, a dash and a pause based on how long the button was pressed.
- **SCREEN DISPLAY (print_welcome_screen, print_level, print_player_wins_screen & print_game_over_screen):** These functions are used throughout the game to display information to the player. For example, the welcome screen displays a welcome message to the user and gives instructions on how the game is played. These screens make the game more realistic and improve user interaction.
- **GAME FLOW AND LOGIC (level_picker & get_user_input):** These functions improve the flow of the game. The **level_picker** allows the user to select a level of difficulty between 4 different levels. Where level 1 is the easiest and level 4 is the hardest. **get_user_input** allows the users to input their sequences.
- **MORSE PARSING AND FUNCTIONALITY (morse_parse, create_int, sequenceMorse & randomWords): morse_parse** translates the duration of button presses into morse code symbols. **create_int** converts an array of numbers into integers. This creates a format that is easier for the Morse code symbols to be compared against. **sequenceMorse** generates the correct or expected Morse sequence which is compared against the user inputs. **randomWords** generates random words which are presented to the user as a challenge**.** These functions add to the functionality of the game.

ARM ASSEMBLY CODE

The ARM assembly code mostly dealt with hardware interactions. It was used to manage the time-sensitive detection of user inputs. It ensured that the system was able to accurately

recognize and interpret the user inputs via a “button-pressed” system. The key functions used in the code and their roles are highlighted below.

- **GPIO AND BUTTON INITIALIZATION (init_btn):** The purpose of this function was to initialize the GPIO pin responsible for user inputs. The function sets the pin to input mode and enables interrupts to detect when the button is pressed or released.
- **ALARM SETUP (alarm_setup):** This allowed for the configuration of the timer alarm. This made it possible for the system to differentiate between a short pause (that separates characters within a word) and long pauses (that separates words). This made it possible to accurately translate the Morse code the player inputs into words and characters.
- **INSTALLING INTERRUPT HANDLERS (install_alarm & install_gpio):** These functions set up the interrupt vectors for the timer and GPIO. This ensures that the right ISR is called when an interrupt occurs.
- **INTERRUPT SERVICE ROUTINE(alarm_isr & gpio_isr):** These functions handle interrupts. For example, the alarm ISR would handle pauses between the characters while the gpio ISR would handle button releases and button presses.
- **MORSE CODE DETECTION (detect_morse):** This function makes it possible to classify a button press as either a dot or a dash. It does this by measuring the duration for which the button is pressed. With this information, it can classify the button press as either a dot (a short press) or a dash (a long press). If no button is pressed, it is interpreted as a pause or space between dashes and dots.
- **MORSE CODE PARSING (morse_input):** After the dots, dashes and spaces are identified, they are arranged into a sequence that corresponds to a Morse Alphanumerical character.

HOW ARE BOTH CODES LINKED?

Both codes work well together and can interact with each other via function calls. There are specific instances in which the codes are linked. The assembly code directly calls functions within the C code to perform specific tasks. An instance of this would be when the assembly code calls the asm_gpio_init and the asm_gpio_set_dir functions within the C code to handle the initialization of GPIO pins and set the directions. Another instance of this would be when the assembly code calls the morse_parse function within the C code. Overall, the integration of both codes through direct function calls, interrupts service routines and globally accessible variables made it possible to develop a high-level game logic that worked seamlessly.

USER INTERFACE AND HARDWARE

The primary user interface used for this game is the Pi Pico microcontroller board. It made it possible to capture user inputs through buttons pressed on the board. The RGB LED and console display were used as user feedback mechanisms. The RGB LED showed the game status by changing the colour of the LED (e.g. Green means the game is in progress and red means the game is over). These real-time feedback mechanisms enhance the user’s experience. The console display shows the display screens which provide instructions and allows the user to make selections or take actions based on the display.

PROJECT WORKFLOW MANAGEMENT

PROJECT WORKFLOW BY STEPHEN KOMOLAFE

The project workflow was managed by the project workflow owner. All group communications took place on a discord channel created by a member of the group. During the first meeting of the group, the group members decided what task each member would undertake, and a flowchart was developed detailing how the project would be handled and split up. Hence, the project was split up into the following stages.

As a group, we made an effort to meet as frequently as possible, whether it was in person or online. In the event that any members were not available at the agreed-upon time, meetings were rescheduled to a time that suited everyone. During these meetings, we exchanged ideas with each other on how to approach the project.

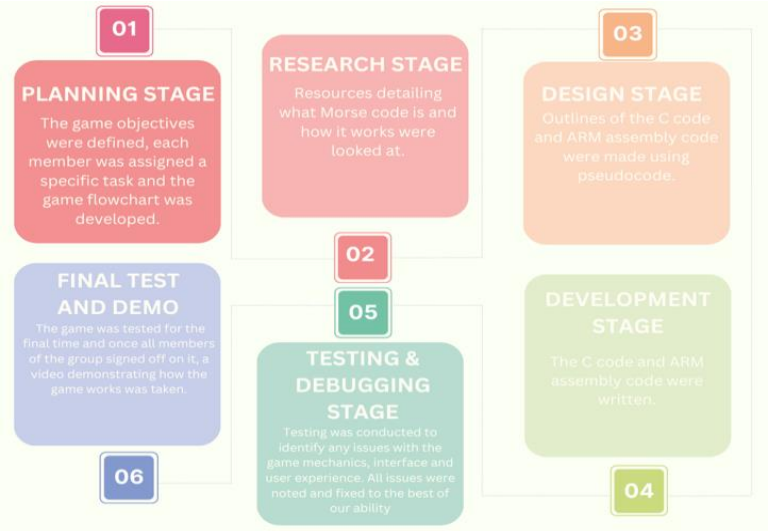


Figure 4: project workflow

GitLab WORKFLOW BY DYLAN GALLAGHER

Throughout the course of this project, we managed our code development via Gitlab. This made it possible for all members of the team to have real-time updates and the latest version of the code. Each member of the code created a separate branch where they worked on their specific tasks. After the tasks were completed, the code was extensively tested and debugged by the code owner before a merge request was created and the code was merged into the main.

Each member of the group also made sure to regularly commit their changes to their branch. This made debugging a lot easier and faster as it was easy to trace the location of the errors in the code.

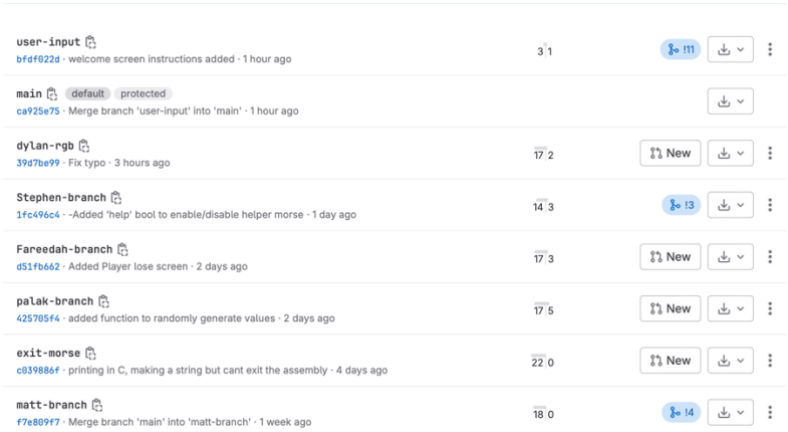


Figure 5: Branches created in GitLab

We also had a review system in place, with main being a protected branch, so we cannot push to main without approval from the GitLab Manager as well as another team member. This allowed us to maintain a well-tested and working version of the project in the main at all times. We utilised both comments in GitLab, as well as messaging in Discord, for communicating changes to code that needed to be made before it was accepted to main.

CONCLUSION

In conclusion, developing this Morse game gave us the opportunity to use our knowledge of both ARM assembly and C language in real-world applications. This project showcased the team's ability to work together to implement a complex game design that implements the low-level hardware functionality of the ARM assembly code and the high-level functionality of the C code. Throughout the project, each member played a fundamental role in bringing the game to life. From the planning done by the workflow owner to the code management done by the code owner, every aspect of the project was handled efficiently by each member of the group. The use of Gitlab for version control and collaboration was also vital to ensuring that each member of the team was able to partake in the project to the best of their ability.

Overall, the project achieved its objective as we were able to develop a Morse code game that works and can test its players on their knowledge of the Morse code equivalent of various alphanumerical characters and words. It also provided an invaluable learning experience for the team members on coding, team collaboration, version control, documentation and project management.