CSSE2010/CSSE7201 AVR Project

Semester 2, 2024 - Version 1.00 (30/09/2024)

Due: 4:00pm, Friday 25th October

Weighting: 20% (100 marks)

Objective

As part of the assessment for this course, you are required to undertake an AVR project which will test you against some of the more practical learning objectives of the course, namely your C programming skills applied to the ATmega324A microcontroller.

You are required to modify a program to implement additional features. The program is a basic template of the game "Sokoban" (described in detail on page 3) and has very basic functionality. It will present a start screen upon launch, respond to push button presses or a terminal input "s"/"S" to start the game, then display the first level which contains a map of game objects and a flashing player icon. You can add features such as moving the player, game logic, pausing, audio etc. The various features have different levels of difficulty and will each contribute a certain number of marks.

Don't Panic

While this is a long project specification, there's not actually that much code to write! You've been provided with approximately 2000 lines of code to start with! You are not expected to fully understand all of the provided code, and several intro/getting started videos are available on Blackboard, which contains a demonstration of some of the expected functionality to be implemented and walks through setting the project up with the provided base code, as well as how to submit.

Note

The requirements in this document take priority over anything shown in the feature demonstration videos.

Academic Integrity

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. **You must not show your code to or share your code with any other student under any**

circumstances. You must not post your code to public discussion forums or save your code in publicly accessible repositories, even after the completion of this course. You must not look at or copy code from any other student. All submitted files may be subject to electronic plagiarism detection and the electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. **Your submission will be compared against the submissions of all other students, including submissions from previous semesters and 20 submissions generated by 5 different large language models (generative AIs).** Furthermore, a random sample of 50 submissions will be spot checked and manually reviewed for academic integrity by the staff team. **Formal misconduct proceedings will be instituted against students where plagiarism or collusion is suspected.**

Grading Note

As described in the course profile, if you do not score at least 10% on this AVR project (before any penalty) then your course grade will be capped at a 3 (i.e., you will fail the course). If you do not obtain at least 50% on this AVR project (before any penalty), then your course grade will be capped at a 5. Your AVR project mark (after any penalty) will count 20% towards your final course grade.

This project has 17 features, broken into three tiers, and marks are distributed as follows:

	Available Marks	**Max Mark**
Tier A	54	50
Tier B	34	30
Tier C	24	20
	112	100

This means if you implement all features of tier A and score full marks for each feature, you will receive 54 marks, however it will be capped at 50 when the grade of your project is calculated. This also means that you do not need to score full marks for each feature to be able to achieve 100% for the project.

Code Style and Restrictions

There is no restriction on your code, as long as your code **compiles with Microchip Studio installed on the lab computers**. No marks are awarded nor deducted in the marking criteria for code style, however you are advised to follow a good code style for your own benefit and to make it easier for course staff to assist you. You may freely create and add `.c`/`.h` files, as well as include additional header files (eg: `string.h`, `ctype.h`) from the C Standard Library to any source file using `#include`.

You may use any C language feature (e.g., switch statements, structures, pointers), provided that they are C99 compliant and supported by the GCC compiler used by Microchip Studio.

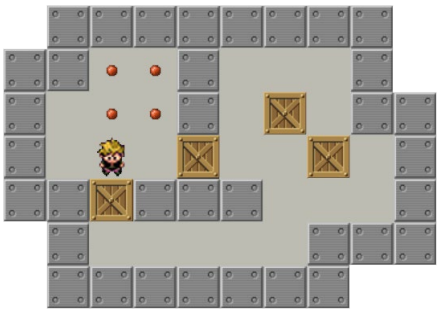
Provided Code

You have been provided with some base code as a starting point for this project. You should import the base code files into a project, and you will be working on adding features to the provided base code. There is a documentation of the base code on Blackboard which you should read prior to implementing the features and refer to while working on the project. A setting up video on Blackboard shows how to create a new project and get the base code up and running.

Note

When you create a new project in Microchip Studio, a `main.c` file will automatically be created, containing an empty `main()` function. `project.c` also contains a `main()` function, but Microchip Studio will preferentially look for the `main()` function in the `main.c` file, if it exists. Please ensure that you **delete the `main.c` file** so that Microchip Studio will look for the `main()` function in `project.c`.

Sokoban Description



This AVR project involves creating a replica of the classic game "Sokoban". Sokoban is a classic puzzle video game in which the player pushes boxes around in a warehouse, aiming to get them to designated storage locations.

There are 4 main object types in a game of Sokoban – player, wall, box and target. The player and boxes are movable, while walls and targets are not. In the original game, the player may push a box in 4 directions – left, right, up and down, provided that there is no wall or other boxes behind the box. 2 boxes cannot be pushed together. The player can move to empty or target squares, but not to walls or boxes (boxes can be pushed, but player and box can never coexist on the same square). The level is complete when all boxes have been pushed onto the targets by the player. The number of boxes is always equal to the number of targets.

Note

You may play the Sokoban game online at https://sokoban.info/ to gain a better understanding of the game. Please note however the version of Sokoban described in this project specification has differences compared to online version, and you **must implement the game according to this specification.**

For this AVR implementation, the warehouse is represented using the 16x8 LED matrix:

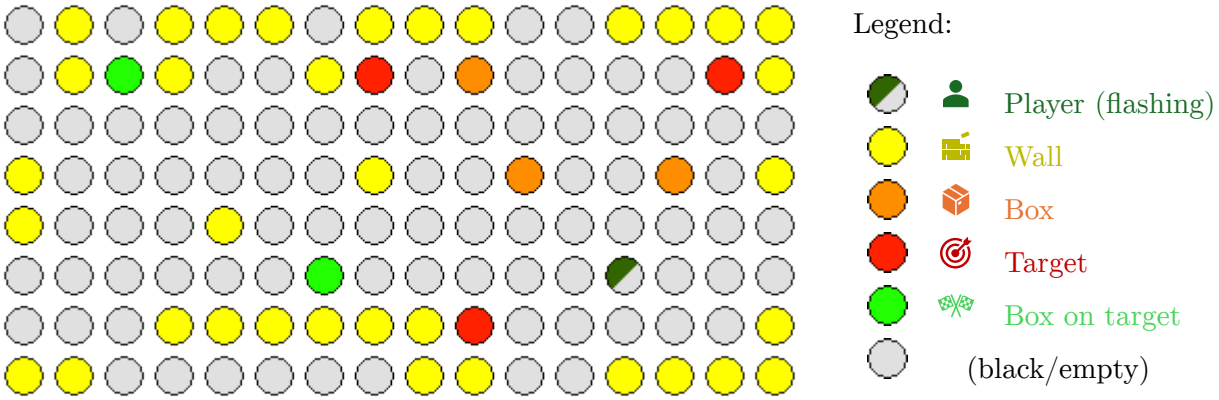


Figure 1: Snapshot of a sample game, showing all game objects in action.

The flashing player icon is rendered in this document as , meaning that particular pixel of the LED matrix flashes between dark green and black. Likewise, means a pixel which flashes between dark green and red (i.e., player on a target).

Initial Operation

The provided program has very limited functionality. It will display a start screen (see Figure 2) which detects the rising edge on the push buttons B0, B1, B2 and B3, as well as the input terminal character "s"/"S". Pressing any of these will start a game of Sokoban and take you to the initial game board as illustrated in Figure 3.

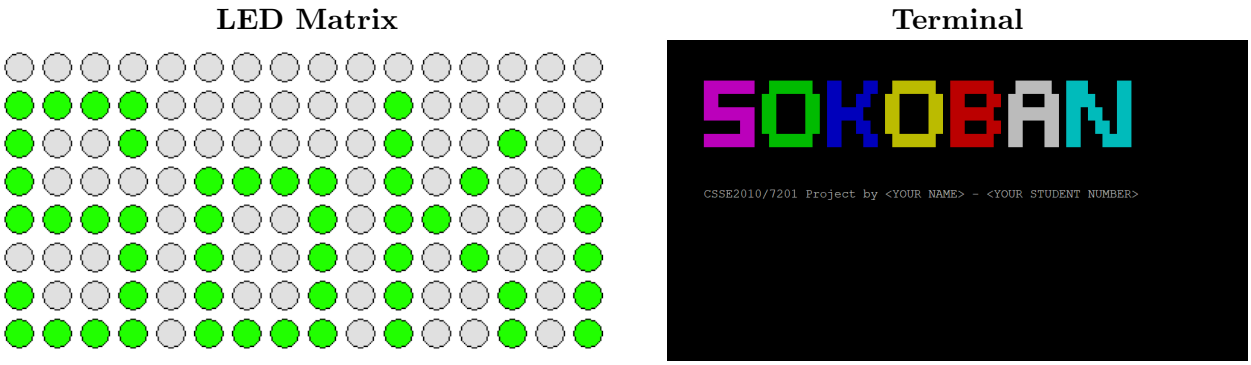


Figure 2: Start screen on the LED matrix (scrolling) and on the terminal.

Once started, the provided program is capable of detecting a rising edge on the push button B0, but no action when the button is pressed (this will need to be implemented as part of the Move Player with Push Buttons feature).

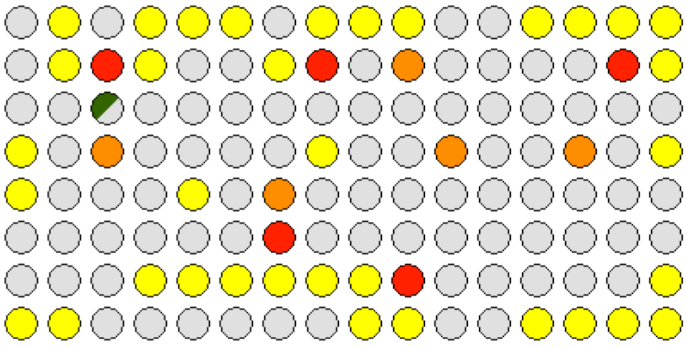


Figure 3: Initial game board after starting the game.

Terminal Layout

The terminal used for marking will have 80 columns and 24 rows, and you are advised to keep this in mind when implementing the project. During gameplay, a message area one row in height is defined on the terminal, for displaying messages.

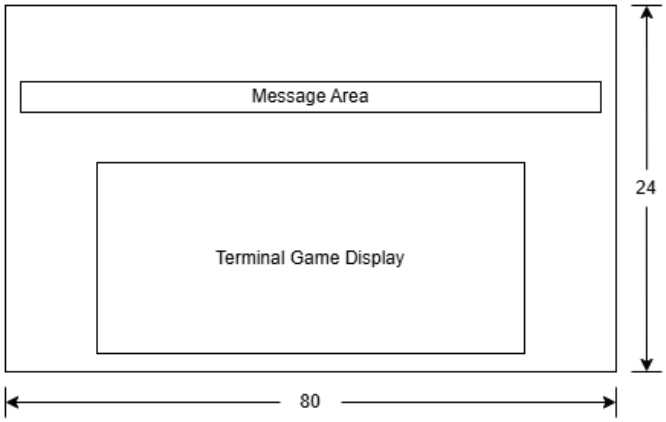


Figure 4: Recommended in-game terminal layout.

Wiring Advice

When completing this AVR project, you will need to make additional connections to the ATmega324A microcontroller to implement particular features. To do this, you will need to choose which pins to make these connections to. There are multiple ways to do this, so the exact wiring configuration will be left up to you, **and you must communicate this using your submitted feature summary form** (on Blackboard).

Hint

Before implementing any features, read through them all, and consider what peripherals each feature requires and any pin limitations this imposes. If you do not do this, you may find yourself in a situation where the pins that must be used for a peripheral for a later feature are already connected to another peripheral, requiring you to rewire your breadboard and update your code before you can use the new peripheral.

Some connections are defined for you in the provided base code and are shown in grey in the following table.

Wiring Table

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
A								
B	SPI connection to LED matrix				Button B3	Button B2	Button B1	Button B0
C								
D							Serial RX	Serial TX
							Baud rate: 19200	

Program Features

Marks will be awarded for features as described below. Part marks will be awarded if part of the specified functionality is demonstrated. Marks are awarded only on demonstrated functionality in the final submission – no marks are awarded for attempting to implement the functionality, no matter how much effort has gone into it, if there is no evidence of functionality when the program is run. You may implement higher-tier features without implementing all lower-tier features if you like (subject to prerequisite requirements). The number of marks is not an indication of difficulty. Marks may be deducted if features interact negatively with one another or affect gameplay experience.

You may modify any of the code provided and use any of the code from learning lab sessions and/or posted on the course Blackboard site.

Minimum Performance — Pass/Fail

Your program must have at least the features present in the code supplied to you, i.e., it must build on Microchip Studio and run, show the start screen, and display the initial game when a push button or "s"/"S" is pressed. No marks can be earned for other features unless this requirement is met, i.e., your AVR project mark will be zero.

Start Screen — Tier A: 4 marks

Modify the program so that when it starts (i.e., the AVR microcontroller is reset) it outputs your name and student number to the serial terminal, in the indicated location (remove the placeholder text, including the chevrons <>). Do this by modifying the function `start_screen` in file `project.c`.

Move Player with Push Buttons — Tier A: 6 marks

The provided program does not allow moving the player. Modify the program so that when push button B0 (connected to port B pin 0) is pressed, the player moves right. Similarly, pressing push button B1 (connected to port B pin 1) should move the player down, push button B2 (connected to port B pin 2) should move the player up, and push button B3 (connected to port B pin 3) should move the player left.

If the player would be moved off the LED matrix display, it must wrap around to the other side of the display. For example, if the player is on the far-right side of the LED matrix, then pressing B0 must move the player to the far-left side of the LED matrix (same is true for up/down). Whenever the player is moved, the flashing cycle of the player icon must be reset, such that the player icon instantly displays dark green and remains dark green for 200ms. The player should move when the button is pressed; no behaviour is expected when the button is released, nor if the button is held down.

If the player is moved onto a target, the target must flash between dark green and red (). When the player is moved off the target, the target must revert to displaying solid red. You do not have to consider moving onto walls or boxes until the later Game Logic – Walls and Game Logic – Boxes features, respectively.

Hint

The function `play_game` in `project.c` calls `move_player(0, 1)` in `game.c` when push button B0 is pressed. The `move_player` function is currently empty; start by filling it in according to the comments provided.

Move Player with Terminal Input — Tier A: 6 marks

The provided program does not register any terminal inputs once the game has started. Modify the program so that pressing "w"/"W" moves the player up, and "a"/"A", "s"/"S", and "d"/"D" move the player left, down, and right, respectively, in a similar manner to the previous task. Note that both the lowercase and uppercase versions of each letter must execute these movements as described. Also, note that the inbuilt serial functionality handles keyboard inputs that are held down for you.

Just like in the previous task, the player must wrap around the edges, and the flashing must be reset whenever the player moves. Holding down a key will usually send multiple instances of that key to the terminal. Unlike the push buttons, this means holding down a key will result in the player repeatedly moving, which is fine and to be expected.

Hint

On the start screen, the game can be started by pressing "s"/"S"; look at the function `start_screen()` to get an idea of how to read serial inputs from the terminal.

Game Logic – Walls — Tier A: 6 marks

Requires Move Player with Push Buttons or Move Player with Terminal.

When the player attempts to move onto a wall, nothing should happen, and a message must be printed to the message area of the terminal telling the player that they've hit a wall. Below is an example of moving to the right onto a wall.

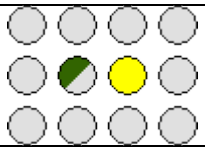
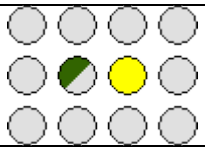
	Before	Action	After
Display		→ Terminal: "d"/"D" Buttons: B0	
Message	[Previous Message]		I've hit the wall.*[1]

Figure 5: Example of player moving to the right and hitting a wall. This is only one example; you must consider the player hitting walls after moving up, left and down as well.

Given how often people run into walls, the message displayed must be randomly chosen from at least three different messages. You may decide on the messages to use; however, they must clearly indicate that the player has hit a wall, and the order in which the messages are displayed must be non-deterministic (random).

As with the player movements, if the player is on the far-right side of the LED matrix, and there is a wall on the far-left side of the LED matrix, and the player moves to the right, they would hit the wall on the far-left side, and the move would be considered invalid. The same goes for all the other edges.

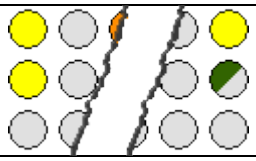
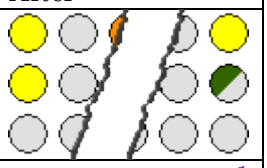
	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		I've hit the wall.*[1]

Figure 6: Example of player moving to the right and hitting a wall after wrapping around. This is only one example; you must consider hitting walls after wrapping around in other directions too.

A valid move is defined as a move that changes the player's location, and whenever a valid move is made, the message area of the terminal must be cleared.

Hint

The C Standard Library includes a function `rand()` for returning pseudo-random numbers. The numbers returned by the `rand()` function are deterministic (i.e. the first time you call it each time you start your program, it'll always give you the same number), unless you seed the random number generator with a custom, pseudo-random seed. The random number generator can be seeded by calling the function `srand()`, with the appropriate seed as argument. Current time is often used by programmers as the random seed, and you are recommended to do so as well. You should only seed the random number generator once.

Hint

You will not be able to use the `time()` function for retrieving the current time, because it does not work by default on the AVR. You should instead search for functions given to you in the base code.

Hint

The time when, for example, `initialise_hardware()`, is called is always the same (e.g., 40 milliseconds after the microcontroller is reset), as it always takes a specific number of instructions to arrive at that particular line of code. The time when the player presses “s”/“S” or a push button to start the game on the other hand is completely dependent on the player - a good source of randomness.

[1] Does not have to be this exact message and must be one out of 3 random messages that carries the same meaning.

Game Logic – Boxes — Tier A: 8 marks

Requires Move Player with Push Buttons or Move Player with Terminal.

When the player attempts to move onto a box:

- If there is nothing behind the box the player is trying to move onto, the box must be pushed by one square in the direction of the player's move, and the player must also be moved one square in the same direction, such that the player icon ends up at the original location of the box. If there was a message displayed in the message area of the terminal, it must be cleared.

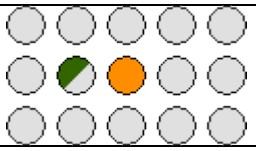
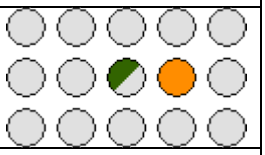
	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		[No Message]

Figure 7: Example of player moving the box to the right. This is only one example; you must consider the player moving boxes in other directions too.

- If there is a wall behind the box they are trying to move onto, nothing should happen, and the player must be notified via a message in the message area of the terminal saying that a box cannot be pushed onto the wall.

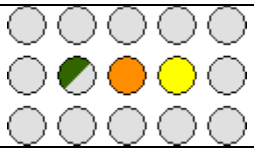
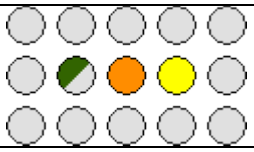
	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		Cannot push box onto wall.[*2]

Figure 8: Example of player moving a box to the right onto a wall. This is only one example; you must consider the player moving boxes onto walls in other directions too.

- If there is another box behind the box they are trying to move onto, nothing should happen, and the player must be notified via a message in the message area of the terminal saying that a box cannot be stacked on top of another box.

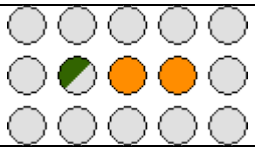
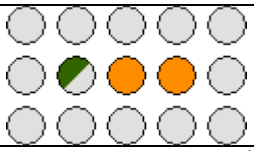
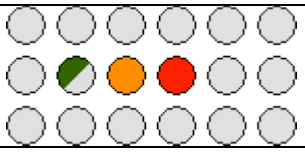
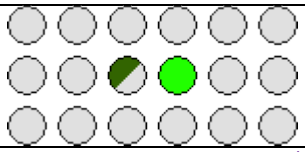
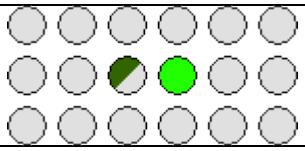
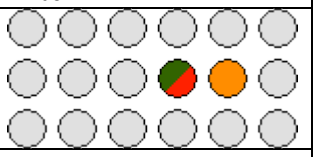
	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		Cannot stack boxes.[*2]

Figure 9: Example of player moving a box to the right onto another box. This is only one example; you must consider the player moving boxes onto boxes in other directions too.

When a box is moved onto a target square, that square must be rendered bright green, and a message indicating that a box has been moved onto a target must be displayed in the message area of the terminal. Boxes can also be pushed off target squares, and in that case, the target square must revert back to red. However, since the player would be on top of the target square immediately following the move, it should flash between dark green and red. If the player then moves the player icon off the target square, it should revert back to displaying solid red.

	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		Box moved onto target.*2

	Before	Action	After
Display		→ Terminal: “d”/”D” Buttons: B0	
Message	[Previous Message]		[No Message]

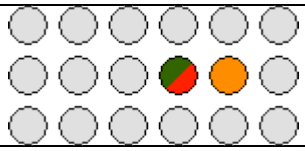
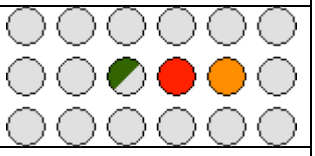
	Before	Action	After
Display		← Terminal: “a”/”A” Buttons: B3	
Message	[Previous Message]		[No Message]

Figure 10: Example of player pushing a box onto and off a target square.

As with the previous features, wrapping around the edges must be considered, meaning that if a box is at the far-left side and there’s nothing on the far-right side, and the player pushes that box further to the left, it would show up on the far-right side. Messages printed after invalid moves must also be cleared whenever a valid move is made, as before.

*2 *Does not have to be this exact message; you can come up with your own.*

Level Time — Tier A: 4 marks

Keep track of the amount of time the player has spent on the level by displaying the number of seconds elapsed since the start of the level on the terminal. The amount of time elapsed must be displayed in seconds and must not overflow (i.e., back to 0 or become negative) during a reasonable gameplay. When the level starts, the level time must be 0 (seconds), and it should increment to 1 after a whole second has passed. This feature must not spam the terminal.

Note

Terminal spamming happens when you send more data than needed. For example, you only need to update the time once per second, however if you update it 5 times per second with the same value, that is considered as terminal spamming and marks will be deducted.

Seven-Segment Display Step Count — Tier A: 6 marks

Requires Move Player with Push Buttons or Move Player with Terminal.

Dependencies	Reason	Max Mark if Not Implemented
Game Logic – Walls or Game Logic – Boxes	Needed for verifying that invalid moves do not cause the step count to increase.	5

Keep track of the number of steps that the player has taken since the beginning of the level and display the last two decimal digits of that count on the seven-segment display. When a level starts, the step count must be 0 and should increment by one whenever a **valid** move is made. When the step count reaches 99, it should overflow back to 0 on the seven-segment display when another valid move is made. If the value to be displayed on the seven-segment display is a single-digit number (i.e., 0-9), it must be displayed on the right digit and the left digit could either be turned off or display 0.

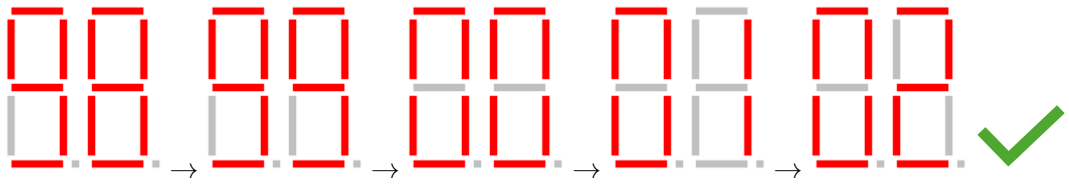


Figure 11: Seven-segment display overflowing correctly.

The step count must not overflow at any other number.

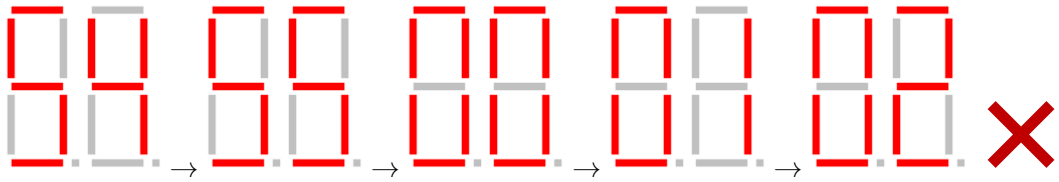


Figure 12: Seven-segment display overflowing incorrectly.

Both digits of the seven-segment display must be on at the same time, without flickering or ghosting, and the display must be appropriately bright. The seven-segment display must be off (display nothing) at the start screen and should display the step count only during a game (and in the future, at level victory).

Hint

Check out the **Lab 15** tasks and **Lab 13 Task 2** for how to display 2 digits simultaneously on the seven-segment display. You may copy code from lab exercise solutions.

Terminal Game Display — Tier A: 6 marks

Dependencies	Reason	Max Mark if Not Implemented
Move Player with Push Buttons or Move Player with Terminal	Used for testing.	4.5
Game Logic – Boxes	Used for testing.	5

Display a copy of the LED matrix display on the terminal (may be displayed at any reasonable location but see Terminal Layout section and Figure 4 for recommendations) using block characters (e.g., spaces) of various colours.

Hint

Normally, a space looks like a solid black character, because it has a black background and no foreground. If you set the background colour to something else, for example red, then a space would look solid red. You can set background colour to, for example, red by including `terminalio.h` and calling `set_display_attribute(BG_RED)`.

This should allow the game to be played either by looking at the LED matrix or at the terminal. The game display on the terminal must be in sync with the LED matrix display (i.e., no apparent difference between the displays), with the exception that the play icon does not have to flash (it can if you want to; but extra marks will not be awarded). The speed of the gameplay must not be adversely affected by the presence of this feature (it is okay if the very start of a game lags a bit due to having to paint the entire map), and this feature must not spam the terminal.

The colours displayed on the LED matrix must match the colours displayed on the terminal, with the exception that you may substitute colours that are used by the game but not defined in the terminal I/O code (dark green and orange) with other colours of your choice (e.g., blue, magenta, cyan). **Please document the colours you have used in your feature summary.**

Note

Since the LED matrix pixels are square/round while terminal characters are not, it is fine for the terminal display of the game to look vertically squashed. You may improve the aesthetics of the terminal game display by using multiple characters for one pixel; however, that is not required.

Level Victory — Tier A: 8 marks

Requires Game Logic – Boxes.

Dependencies	Reason	Max Mark if Not Implemented
Seven-Segment Display Step Count	Used for testing.	7
Level Time	Used for testing.	7

When all targets have boxes on them, the game is over and the player wins. Implement logic to detect game over.

Hint

Do this in the `is_game_over()` function of `game.c`. Return `true` if game over is detected, otherwise return `false`. The base code will break out of the event loop and return from the `play_game()` function if game over is detected, and you will be dropped into the `handle_game_over()` function of `project.c`, for the next part of this feature.

Upon game over, the player must be notified on the terminal that they have completed the level and be shown the number of steps and the amount of time in seconds (rounded down to the nearest second) they took. In addition, a score should be displayed. The score is calculated by:

$$Score = max(200 - S, 0) \times 20 + max(1200 - T, 0)$$

where S is the number of steps it took the player to complete the level (the actual number, not just the last two digits displayed on the seven-segment display), T is the time in seconds (rounded down to the nearest second) it took, and *max* is a function which returns the largest of its two parameters.

The player must be able to restart the current level (at this stage, there is only one level) by entering "r"/"R" at the terminal or exit back to the start screen by entering "e"/"E". You do not have to indicate these on the terminal, but you may do so if you wish (the base code prints it by default).

When the level is restarted, all game objects (boxes, player) and states (step count, level time) must be reset. When the game is exited, the player should be returned to the start screen, and it must be possible to start a new game by pressing "s"/"S" or any of the push buttons.

When the level is complete, the LED matrix should show the last box being on top of the last target (i.e., all target squares being bright green and no orange boxes left), and the player icon should be hidden.

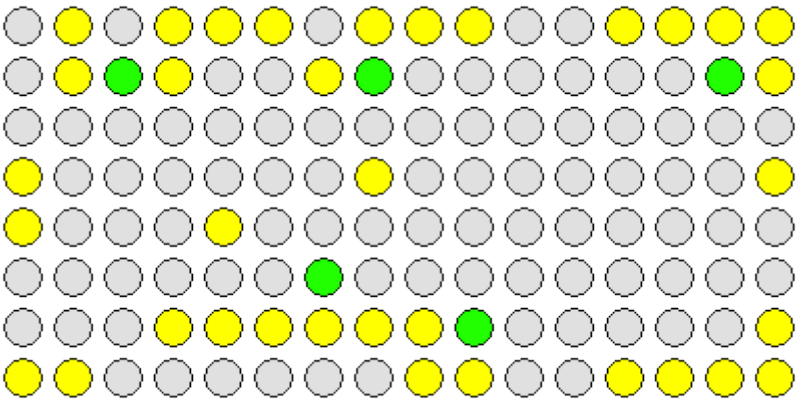


Figure 13: LED matrix display after solving level 1 (the only level at this stage).

If the Seven-Segment Display Step Count feature is implemented, the step count must remain displayed and include the final move which triggered the level victory. If the Level Time feature is implemented, the level time must stop incrementing on the terminal.

If the Terminal Game Display feature is implemented, after game over, the terminal game display does not have to be in sync with the LED matrix anymore and can be removed from the terminal to make room for displaying the level statistics.

Game Pause — Tier B: 6 marks

Dependencies	Reason	Max Mark if Not Implemented
Move Player with Push Buttons	Used for testing.	5
Move Player with Terminal	Used for testing.	5.5
Level Time	Used for testing.	5
Seven-Segment Display Step Count	Used for testing.	5

When the "p"/"P" key is pressed during a game (excluding the start screen or game over screen), the game should pause and remain so until "p"/"P" is pressed again, and a message indicating that the game is paused must be displayed on the terminal (not in the message area). While paused, (at this stage), all inputs other than "p"/"P" should be ignored (this includes button presses too). In addition, the player icon should stop flashing while paused, but when resumed, its progress through its flash cycle should continue. For example, if the game is paused 120ms after the player icon flashed on, then it should flash off 80ms after the game is resumed. It should neither flash off immediately after unpausing nor reset its cycle so that it flashes off 200ms after resuming.

Hint

200ms is very short, so it can be rather difficult to test. What you could do is hold down the "p"/"P" key, and see if the icon flashes at half the normal rate. By holding down the "p"/"P" key, the terminal app repeatedly tells the game to pause and resume, which causes the game to be running at 50% of the time and to be paused for the other 50%. Since it runs 50% of the time, if done correctly, everything should be half as fast (including the level time, if implemented).

If the Level Time feature is implemented, the time displayed on the terminal must stop incrementing immediately when paused and resume incrementing when the game is unpaused. If the game is paused 13.5 seconds after starting the level, then the level time on the terminal should remain 13 until 0.5 seconds after the game is resumed, when it becomes 14.

If the Seven-Segment Display Step Count feature is implemented, both digits must still be displayed simultaneously when the game is paused.

2 Level Two — Tier B: 6 marks

Requires Level Victory.

Extend the Level Victory feature to advance to the next level when the "n"/"N" key is pressed after game over. For example, after finishing level 1, the player should be able to start level 2 by pressing

"n"/"N". You must also allow level 2 to be started directly from the start screen, by entering the number "2" at the terminal, for ease of testing.

The second level must have the following initial layout:

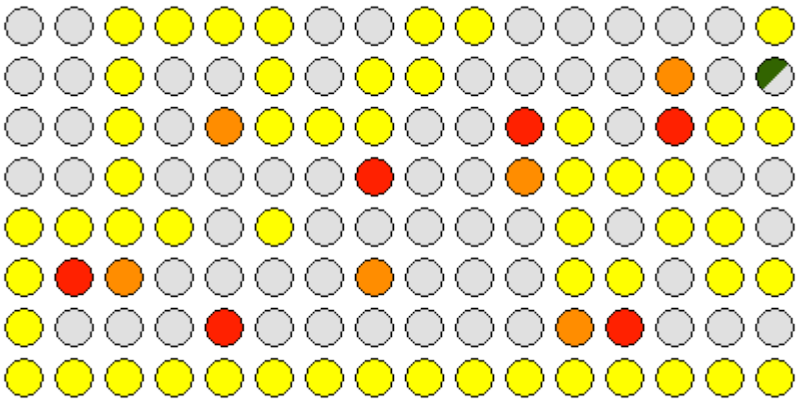


Figure 14: Initial level 2 layout.

This level should behave exactly like the first level. If you have implemented the level restart part of the Level Victory feature, restarting the level after solving level 2 must restart level 2 (rather than restarting level 1).

If there is more than one level (i.e., you implement this feature), then the current level (e.g., level 1 or level 2) which the player is playing must be indicated on the terminal.

You may decide on what to do if the player chooses to advance to the next level when the very last level (i.e., this level) is finished, however your decision must be reasonable. Crashing the game will not be accepted as a solution.

Level Solution

Sound Effects — Tier B: 6 marks

Dependencies	Reason	Max Mark if Not Implemented
Game Pause	Used for testing.	5.5

Add sound effects to the program which are to be output using the piezo buzzer. Different sound effects (tones or combinations of tones) should be implemented for at least three events. At least one of them must be a sequence of at least 4 different tones. For example, choose three of the following events:

- ~~Player being moved~~
- ~~Invalid move~~
- ~~Box pushed~~
- ~~Box moved onto target~~
- ~~Game startup~~
- ~~Game over~~
- ~~Background music~~

Note

Do not make the sounds too annoying! We do not want to go deaf after marking 500 or so AVR projects :).

Pressing the "q"/"Q" key on the terminal should toggle the mute state of the game, the game must be unmuted by default when the AVR microcontroller is powered on. If background music is to be implemented, it can either be played only during gameplay, or be played at all times (i.e., at start screen and game over too).

Your feature summary form must indicate which events have different sound effects and specify which pin the piezo buzzer is to be connected to. Sounds must be tones (not clicks) in the range 20Hz to 5kHz. Sound effects must not interfere with gameplay, e.g., the speed of play must be the same whether sound effects are on or off.

If the Game Pause feature is implemented, no sound should play while the game is paused, however playing sounds must resume where they were left off when unpaused. You may choose to implement sound effects for more than 3 events, however no extra marks will be awarded.

Visual Effects — Tier B: 6 marks

Dependencies	Reason	Max Mark if Not Implemented
Game Logic – Boxes	Animation requires the player to be able to move boxes.	2
Game Pause	Used for testing.	5

Make the target squares flash between red and black at the rate of once per second (red for 500ms, black for 500ms). The flash should work exactly like the player icon, except being slower. When a box is on top of a target, it should remain solid green. When the player icon is on top of a target, it should still flash between dark green and red, at the rate of 200ms for each colour, like before.

When a box is pushed onto a target, display an animation that highlights the location of the target. The animation must be at least 500ms long, and all game inputs must still work while the animation plays. If a second box is moved onto a second target while the animation is playing, you could either finish the playing animation prematurely and start playing the new animation, or you could make them overlap. If the box is moved off the target while the animation is playing, the animation must finish prematurely. It is optional to display the animation for pushing the last box onto the last target.

Visual effects do not have to be reflected on the terminal if the Terminal Game Display feature is implemented (you can if you wish, but no extra marks are awarded). Visual effects must pause if the Game Pause feature is implemented and the player pauses the game, and resume when the player resumes the game.

Hint

The flashing player icon implemented for you in the base code is a form of animation, check out how it handles waits in a non-blocking fashion. Any use of `_delay_ms()` or `_delay_us()` results in the inability to respond to inputs while waiting/animating.

Move Player with Joystick — Tier B: 10 marks

Dependencies	Reason	Max Mark if Not Implemented
Seven-Segment Display Step Count	Needed for verifying that each diagonal move count as 2 steps.	9
Game Logic – Walls and Game Logic – Boxes	Some marks are allocated to the interaction with these features.	8

The joystick can also be used to move the player. Extend your existing code to make it possible to control the player movements using the 2-axis joystick. The joystick must be able to move the player in 8 directions – up, down, left, right, north-east (↗), north-west (↖), south-east (↘) and south-west (↙). The horizontal and vertical movements must behave exactly like moving the player with the push buttons or terminal.

The diagonal moves are, however, slightly more complicated. A diagonal move is 2 steps in a single move, with the first one being an intermediate step (described later) not displayed to the player. The player must perceive the player icon as being moved diagonally in a single move, and the step count should increment by 2. When a diagonal move is made, if the target location contains:

- Wall – treat it as an invalid move and display the corresponding message. The message can either be randomly picked out of 3 different ones like the Game Logic – Walls feature, or it can be a single hard coded message.
- Box – treat it as an invalid move and display a message saying that boxes cannot be pushed diagonally.
- Nothing or target – take 2 steps, one horizontal and one vertical, such that after the steps, the player lands at the target location. There are two ways of achieving this – horizontal step followed by a vertical step, vertical step followed by a horizontal step. Below is an example for moving in the north-east direction:

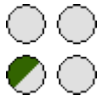
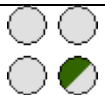
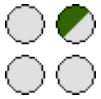
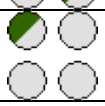
Before	Action	Intermediate	After	Comment
	↗ Northeast			Horizontal then vertical.
				Vertical then horizontal.

Figure 15: Moving northwest when there is nothing on the target square. This is only one diagonal direction; you must consider the other ones too.

It is up to you which way to go, provided that there are no obstacles (boxes or walls) along the way. If there are obstacles along the way, adjacent to the original player location in the vertical or horizontal direction, you must make the move such that the player does not end up passing through the wall or pushing the box.

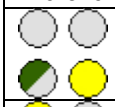
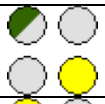
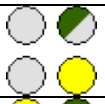
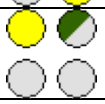
Before	Action	Intermediate	After	Comment
	↗ Northeast			Can only move vertically then horizontally.
				Can only move horizontally then vertically.

Figure 16: Moving northwest when there is an obstacle on one side. This is only one diagonal direction; you must consider the other ones too, and boxes are also considered as obstacles.

If there are obstacles on either side preventing you from moving the player to the target location in either direction, then nothing should happen and the move must be treated as an invalid move, with a message stating that the diagonal move cannot be made printed to the message area of the terminal.

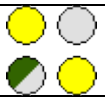
Before	Action	Intermediate	After	Comment
	↗ Northeast			Cannot make the move at all.

Figure 17: Moving northwest when there are obstacles on either side. This is only one diagonal direction; you must consider the other ones too, and boxes are also considered as obstacles.

If the target location is not empty and there are obstacles on both sides preventing the player from making the move, it is up to you which message to display.

If the joystick is held down, the moves should be repeated automatically. You can choose the speed of the automatic repeat yourself, but it must not be too fast that the game becomes unplayable or too slow that the player becomes frustrated. While holding down the joystick for automatic repeating, all other inputs (e.g., push buttons and terminal) must still work.

Note

Be sure to specify which AVR pins the U/D and L/R outputs on the joystick are connected to. Be aware that different joysticks may have different min/max/resting output voltages, and you should allow for this in your implementation – your code will be marked with a different joystick to the one you test with.

Undo Moves

Tier C: 8 marks

Requires Game Logic – Boxes.

Dependencies	Reason	Max Mark if Not Implemented
Game Logic – Boxes	Used for testing.	7
Seven-Segment Display Step Count	Used for testing.	7
Terminal Game Display	Used for testing.	7.5
Move Player with Joystick	Used for testing.	7

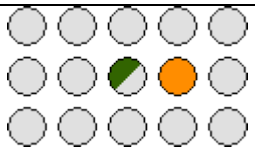
Make it possible for the player to undo up to 6 moves by keeping track of the recent moves and an undo capacity (the number of moves that can be undone). At the beginning of the level, the undo capacity is 0, which means no moves can be undone, because the player hasn't moved yet. Each time a valid move is made, the undo capacity is incremented, and the move is remembered, until the capacity reaches 6. If the undo capacity is 6 and another valid move is made, the oldest remembered move is discarded and the newest move is added, keeping the overall capacity unchanged. Once valid moves are made, pressing the "z"/"Z" key undoes the most recent move and decreases the undo capacity, until the undo capacity becomes 0, which means no more moves can be undone. The table below contains a few examples.

Action	Move History						Undo Capacity
Initial (no moves yet)	-	-	-	-	-	-	0
Up	-	-	-	-	-	↑	1
Undo Move	-	-	-	-	-	-	0
Down	-	-	-	-	-	↓	1
Left	-	-	-	-	↓	←	2
Down	-	-	-	↓	←	↓	3
Down	-	-	↓	←	↓	↓	4
Right	-	↓	←	↓	↓	→	5
Up	↓	←	↓	↓	→	↑	6
Up	←	↓	↓	→	↑	↑	6
Undo Move	-	←	↓	↓	→	↑	5
Undo Move	-	-	←	↓	↓	→	4
Left	-	←	↓	↓	→	←	5

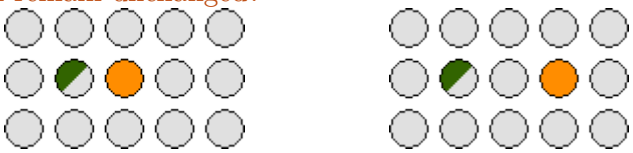
Figure 18: High level representation of possible actions and their associated move history and undo capacity. Each move here is represented with an arrow. The actions listed here are just examples, you are required to handle all possible player moves and undos.

Hint

In the above table, each move is represented with an arrow, however when you are implementing this feature, keeping track of the move direction alone may not be enough, as moves may cause boxes to change location. Consider that you have just moved the player to the right, and your map looks like this:



Do you know the location of the box before the move? If the player undoes the move, where should the box be? Should the box be shifted to the left with the player, or should its location remain unchanged?



Now, if you have also kept track of whether the box was pushed, then you can say that if the box was pushed by the move, then the box should be shifted to the left along with the player, otherwise the box's location should remain unchanged.

When a move is undone, everything changed as the result of the move must be also be undone. Everything on the LED matrix, seven-segment display and terminal must revert to what they have looked like before the move was made, with the exception of the level time and the message area of the terminal, which should remain unchanged and be cleared, respectively.

- The player should go back to its last location (i.e., the location before the move was made) when a move is undone. It does not matter if the flash cycle is reset or not.
- If the Game Logic – Boxes feature is implemented, and the move being undone caused movement of boxes, the box moved should go back to its last location (i.e., the location before the move was made).
- If the Seven-Segment Display Step Count feature is implemented, the step count should be decremented by one when a move is undone. If the current step count is 200 and the display is 00, after undoing the last move, the step count should become 199 and the number 99 should be displayed on the seven-segment display.
- If the Move Player with Joystick feature is implemented, you can either chose to have the diagonal moves being treated as 1 move and be undone completely at once (i.e., take up 1 slot in the move history, decrease step count by 2), or have each diagonal move count as 2 moves and require 2 separate undos to fully undo (i.e., take up 2 slots in the move history, and the player would end up at the intermediate location if they choose to only undo once, and decrease step count by 1 per undo).

The undo capacity is indicated by the LEDs L0 to L5. If the undo capacity is 0, none of those LEDs should be lit, likewise, if the undo capacity is 6, all of them should be lit. When the undo capacity decreases, the leftmost lit LED should turn off. When undo capacity increases, the rightmost unlit LED should turn on.

Undo Capacity	LEDs
0	
1	
2	
3	
4	

5	
6	

Figure 19: Undo capacities and their associated LED displays.

If the player attempts to undo a move when the undo capacity is 0, nothing should happen. You may optionally display a message in the message area of the terminal, however that is not required.

Redo Moves — Tier C: 8 marks

Requires Undo Moves.

Dependencies	Reason	Max Mark if Not Implemented
Game Logic – Boxes	Used for testing.	7
Seven-Segment Display Step Count	Used for testing.	7
Terminal Game Display	Used for testing.	7.5
Move Player with Joystick	Used for testing.	7

Make it possible for the player to redo up to all of the recently undone moves, provided that the player has not made new valid moves since the last undo. Each time "y"/"Y" is pressed, the last undone move is redone. Redone moves must also be undoable. The LEDs must still display the undo capacity.

Hint

This redo feature combined with the undo feature works exactly like the undo/redo of most text editors. If you're confused, try opening up a text editor, typing in some text and play around with its undo/redo functionality.

Much like the undo feature, when a move is redone, everything changed by undoing the move must be redone, with the exception of the level time and the message area of the terminal, which should again remain unchanged and be cleared, respectively.

If the player has not undone any moves yet, or if valid moves have been made since the last undo, nothing should happen, and you may optionally display a message in the message area.

Like the undo feature, you should keep track of the undone moves, and the latest undone move should be redone first. The remembered undone moves should be discarded whenever a valid move is made by the player (via the push buttons, terminal or joystick). You may also find keeping track of a redo capacity useful, however unlike the undo capacity, it does not need to be displayed anywhere. The table below contains a few examples.

Action	Move History						Undone Moves						Undo Capacity
Initial (no moves yet)	-	-	-	-	-	-	-	-	-	-	-	-	0
Up	-	-	-	-	-	↑	-	-	-	-	-	-	1
Undo Move	-	-	-	-	-	-	↑	-	-	-	-	-	0
Down	-	-	-	-	-	↓	-	-	-	-	-	-	1
Right	-	-	-	-	↓	→	-	-	-	-	-	-	2
Down	-	-	-	↓	→	↓	-	-	-	-	-	-	3
Undo Move	-	-	-	-	↓	→	↓	-	-	-	-	-	2
Redo Move	-	-	-	↓	→	↓	-	-	-	-	-	-	3
Undo Move	-	-	-	-	↓	→	↓	-	-	-	-	-	2
Down	-	-	-	↓	→	↓	-	-	-	-	-	-	3
Down	-	-	↓	→	↓	↓	-	-	-	-	-	-	4
Left	-	↓	→	↓	↓	←	-	-	-	-	-	-	5
Undo Move	-	-	↓	→	↓	↓	←	-	-	-	-	-	4
Undo Move	-	-	-	↓	→	↓	↓	←	-	-	-	-	3
Undo Move	-	-	-	-	↓	→	↓	↓	←	-	-	-	2
Undo Move	-	-	-	-	-	↓	→	↓	↓	←	-	-	1
Undo Move	-	-	-	-	-	-	↓	→	↓	↓	←	-	0
Redo Move	-	-	-	-	-	↓	→	↓	↓	←	-	-	1
Redo Move	-	-	-	-	↓	→	↓	↓	←	-	-	-	2
Left	-	-	-	↓	→	←	-	-	-	-	-	-	3

Figure 20: High level representation of possible actions and their associated move and undo histories and undo capacity. Each move here is represented with an arrow. The actions listed here are just examples, you are required to handle all possible player moves/undos/redos.

Game Progress Save — Tier C: 8 marks

Requires Game Pause.

Dependencies	Reason	Max Mark if Not Implemented
Move Player with Push Buttons or Move Player with Terminal or Move Player with Joystick	Testings require the player to be able to make moves.	2
Game Logic – Boxes	Used for testing.	8
Level Time	Used for testing.	9.5
Seven-Segment Display Step Count	Used for testing.	9.5
Level Two	Used for testing.	9.5

Extend the Game Pause feature to be able to exit back to the start screen when the game is paused, and optionally save the current game progress. When the game is paused, pressing the "x"/"X" key must allow the player to exit back to the start screen without having to complete the level. When the "x"/"X" key is pressed, the game pause message on the terminal is replaced with a question asking if the player would like to save the current game progress. If the player responds with "y"/"Y", then the

current game progress is saved and the player is returned to the start screen. If the player responds with "n"/"N", then the player is returned to the start screen immediately and the current game progress is discarded. Other responses are ignored.

If the game progress is to be saved, and a previous progress save exists, the previous progress save is discarded and replaced with the new progress save. On the start screen, if a progress save is present, a message indicating so must be displayed, and the player must be able to restore the progress save by entering "r"/"R". Once the progress save is restored, it is deleted. The player should also be able to delete the progress save by entering "d"/"D" at the start screen (which must cause the message saying there's a progress save present to be removed).

When a progress save is restored, the entire state of the game before the progress save was made is restored. If the Level Time feature is implemented, the level time must be restored too, however you are only expected to be able to remember the level time in terms of seconds. For example, if the progress save was made 13.5 seconds into the game, then when the progress is restored, the time displayed on the terminal should be 13, and it could either be incremented to 14 after 500ms, or 1 second.

The player icon flash cycle does not need to be remembered. You are not expected to remember or restore the message in the message area of the terminal. If the Undo Moves or Redo Moves feature is implemented, it does not matter if undos and redos are remembered. If the Visual Effects feature is implemented, and the progress is saved while an animation is playing, the animation does not need to be remembered. If the Sound Effects feature is implemented, whether sounds are on and the current sound playing does not need to be remembered.

Progress saves should be stored on the EEPROM (Electrically Erasable Programmable Read-Only Memory) of your AVR microcontroller, rather than in the RAM (Random Access Memory), which allows them to survive resets and power cycles. You must handle the situation of the EEPROM initially containing data other than that written by your program. You may need to use a "signature" value to indicate whether your program has initialized the EEPROM for use – it must not say progress save exists if none was saved from your program.

Note

The EEPROM aspect of this feature goes beyond the content that is taught in the labs. You will be expected to gain the necessary knowledge to complete this feature yourself by **consulting the datasheet (pages 32 to 36)**. Course staff will only provide limited assistance.

Assessment of Feature Implementation

The program improvements will be worth the number of marks shown above. You will be awarded marks for each feature up to the maximum mark for that feature. Part marks will be awarded for a feature if only some part of the feature has been implemented or if there are bugs/problems with your implementation (which may include issues such as incorrect data direction registers). Your additions to the game must not negatively impact the playability or visual appearance of the game. Note also that the features you implement must appropriately work together.

Submission Details

The deadline for the project is 4:00pm AEST Friday 25 October 2024. The project must be submitted via Blackboard. You must electronically submit a single `.zip` file containing ONLY the following:

- **All** of the C source files (`.c` and `.h`) necessary to build the project (including any that were provided to you – even if you haven't changed them);
- A PDF feature summary form (separate document provided on Blackboard).

Do not submit `.rar` or other archive formats – the single file you submit must be a `.zip` format file. All files must be at the top level within the ZIP archive – do not have folders/directories within the archive.

If you make more than one submission, each submission must be complete – the single `.zip` file must contain the feature summary form, and all source files needed to build your work. We will only mark your last submission, and we will consider your submission time (for late penalty purposes) to be the time of submission of your last submission.

It is the responsibility of the student to ensure that their submission is correctly uploaded to the Blackboard submission portal with all of the files they intend to submit. This can be ensured by downloading your ZIP file after submission is made, un-zipping the submission to check all files are correctly included and checking whether you are able to compile the project successfully.

An electronically fillable PDF feature summary form will be provided on Blackboard. This form can be used to specify which features you have implemented and how to connect the ATmega324A to other devices so that your work can be marked. **If you have not specified that you have implemented a particular feature, we will not test for it. Failure to submit the feature summary with your files may mean some of your features are missed during marking (and your submission will NOT be remarked).** You can electronically complete this form or you can print, complete and scan the form. Whichever method you choose, you must submit a PDF file with your other files. If you have any assumptions or comments to convey to the marker, include these on the feature summary form.

Incomplete or Invalid Code

If your submission is missing files (i.e., won't compile and/or link due to missing files) then we will substitute the original files as provided to you. No penalty will apply for this, but no changes you made to the missing files will be considered in marking.

If your submission does not compile and/or link in Microchip Studio for other reasons, then the marker will make reasonable attempts to get your code to compile and link by fixing a small number of simple syntax errors and/or commenting out code which does not compile. A penalty of between 10% and 50% of your mark will apply depending on the number of corrections required. If it is not possible for the marker to get your submission to compile and/or link by these methods, then you will receive 0 for the project (and will have to resubmit if you wish to have a chance of passing the course). A minimum 10% penalty will apply, even if only one character needs to be fixed.

Compilation Warnings

If there are compilation warnings when building your code (in Microchip Studio, with default compiler warning options) then a 0.5-mark penalty per warning up to a maximum of 5 marks will apply. To check for warnings, rebuild ALL of your source code (choose "Rebuild Solution" from the "Build" menu in Microchip Studio) and check for warnings in the "Error List" tab.

Note

PlatformIO may not report all the warnings that Microchip Studio does, and the deductions will be calculated based on the number of warnings reported by Microchip Studio. All students are advised to test compile their code using Microchip Studio installed on the lab computers.

General Deductions

If there are problems in your submitted project that do not fit into any of the above feature categories, then some marks may be deducted for these problems. This could include problems such as general lag, errors introduced to the original program etc. The number of marks deducted will depend on the severity of the issues.

Late Submission

A penalty of 10% of the maximum possible mark will be deducted per 24 hours from the time submission is due for up to 7 days. After 7 days, you will receive a mark of 0.