The University of Queensland
School of Information Technology and Electrical Engineering
Semester 2, 2021

CSSE2010 / CSSE7201 Assignment 2 (project)

Due: **4:00pm Monday November 1, 2021**
Weighting: **20% (100 marks)**

Objective

As part of the assessment for this course, you are required to undertake a project which will test you against some of the more practical learning objectives of the course. The project will enable you to demonstrate your understanding of

- C programming
- C programming for the AVR
- The Microchip Studio environment.

You are required to modify a program in order to implement additional features. The program is a basic template of a game called "Diamond Miners" (a description is given on page 3).
For internal (IN) students: the AVR ATmega324A microcontroller runs the program and receives input from a number of sources and outputs a display to an LED display board, with additional information being output to a serial terminal and – to be implemented as part of this project – a seven segment display and other LEDs.
For external (EX) students: the AVR ATmega328P microcontroller runs the program and receives input from a number of sources and outputs a display to a serial terminal and – to be implemented as part of this project – a seven segment display and other LEDs.

The version of Diamond Miners provided to you has very basic functionality – it will present a start screen upon launch, respond to button presses or a terminal input 's' to start the game, then display the starting screen for the game with a cursor that flashes on and off. You can add features such as moving the player, inspecting walls, placing bombs, collecting diamonds, pausing, sound effects, etc. The different features have different levels of difficulty and will be worth different numbers of marks.

Don't Panic!

You have been provided with approximately 2000 lines of code to start with – many of which are comments. Whilst this code may seem confusing, you don't need to understand all of it. The code provided does a lot of the hard work for you, e.g., interacting with the serial port and the LED display. To start with, you should read the header (.h) files provided along with game.c and project.c. You may need to look at the AVR C Library documentation to understand some of the functions used.

Academic Merit, Plagiarism, Collusion and Other Misconduct

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 <u>any</u> circumstances. You must not post your code to public discussion forums or save your code in publicly accessible repositories. You must not look at or copy code from any other student. All submitted files will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected.** The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you copy code, you will be caught.

Grading Note

As described in the course profile, if you do not score at least 10% on this project (before any late 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 project (before any late penalty), then your course grade will be capped at a 5. Your project mark (after any late penalty) will count 20% towards your final course grade. Resubmissions prior to grade finalization are possible to meet the 10% requirement in order to pass the course, but a late penalty will be applied to the mark for final grade calculation purposes.

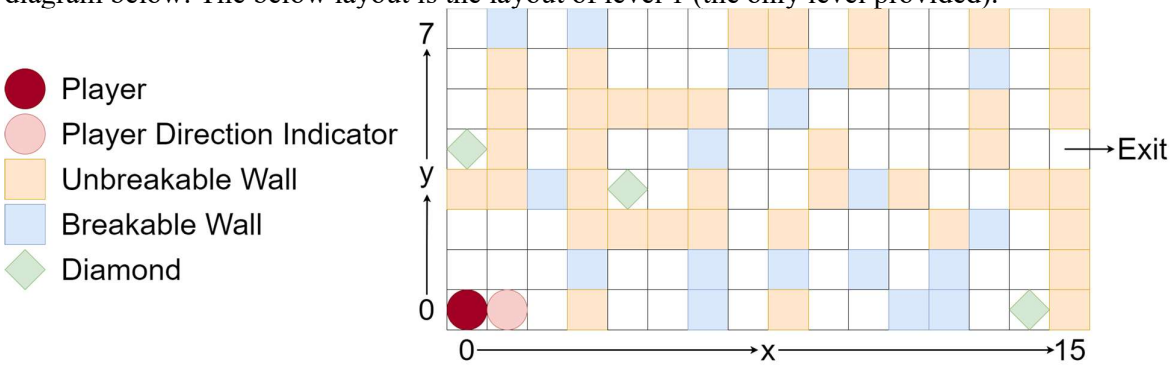
Program Description

The program you will be provided with has several C files which contain groups of related functions. The files provided are described below. The corresponding .h files (except for project.c) list the functions that are intended to be accessible from other files. You may modify any of the provided files. You must submit ALL files used to build your project, even if you have not modified some provided files. Many files make assumptions about which AVR ports are used to connect to various IO devices. You are encouraged not to change these.

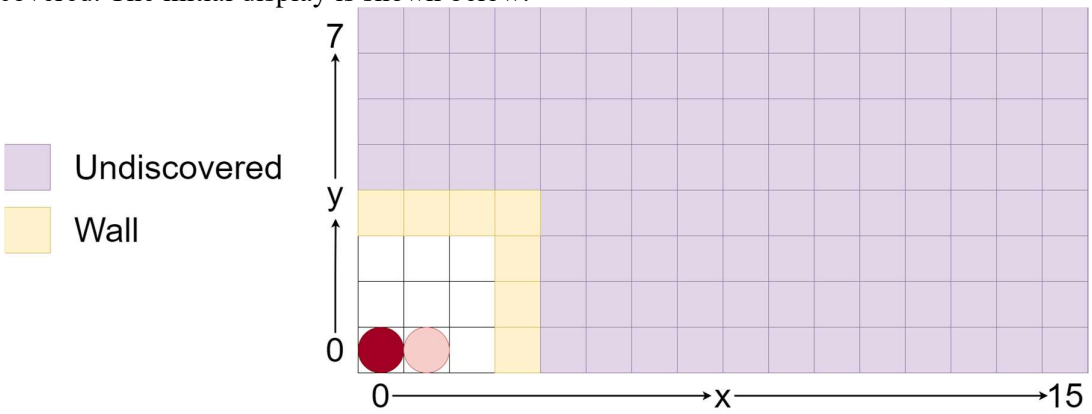
- **project.c** – this is the main file that contains the event loop and examples of how time-based events are implemented. You should read and understand this file.
- **game.h/game.c** – this file contains the implementation of the state of the game, including the playing field and the player's location. You should read this file and understand what representation is used for the playing field and player location. You will need to modify this file to add required functionality.
- **display.h/display.c** – this file contains the implementation for displaying the current state of the board. This file contains useful functions for displaying the board to the LED matrix (internal students) or the terminal display (external students). This file contains the same functions for IN and EX students but with significantly different implementations.
- **buttons.h/buttons.c** – this contains the code which deals with the IO board push buttons. It sets up pin change interrupts on those pins and records rising edges (buttons being pushed). For EX students, this includes a basic implementation of software debouncing.
- **ledmatrix.h/ledmatrix.c** (IN students only) – this contains functions which give easier access to the services provided by the LED matrix. It makes use of the SPI routines implemented in spi.c
- **pixel_colour.h** (IN students only) – this file contains definitions of some useful colours.
- **serialio.h/serialio.c** – this file is responsible for handling serial input and output using interrupts. It also maps the C standard IO routines (e.g. printf() and fgetc()) to use the serial interface so you are able to use printf() etc for debugging purposes if you wish. You should not need to look in this file, but you may be interested in how it works and the buffer sizes used for input and output (and what happens when the buffers fill up).
- **spi.h/spi.c** (IN Students only) – this file encapsulates all SPI communication. Note that by default, all SPI communication uses busy waiting (i.e. polling) – the "send" routine returns only when the data is sent. If you need the CPU cycles for other activities, you may wish to consider converting this to interrupt based IO, similar to the way that serial IO is handled.
- **terminalio.h/terminalio.c** – this encapsulates the sending of various escape sequences which enable some control over terminal appearance and text placement – you can call these functions (declared in terminalio.h) instead of remembering various escape sequences. Additional information about terminal IO will be provided on the course Blackboard site.
- **timer0.h/timer0.c** – sets up a timer that is used to generate an interrupt every millisecond and update a global time value.

Diamond Miners Description

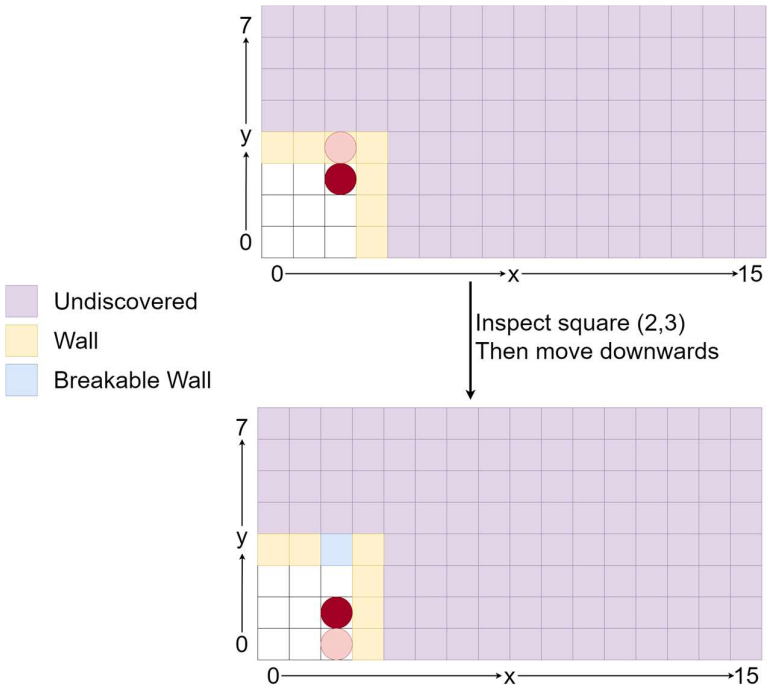
This project involves creating a game called 'Diamond Miners'. Diamond miners is played in an 8×16 grid shown below. Coordinates will be listed as (x,y) pairs with x and y shown on the diagram below. The below layout is the layout of level 1 (the only level provided).



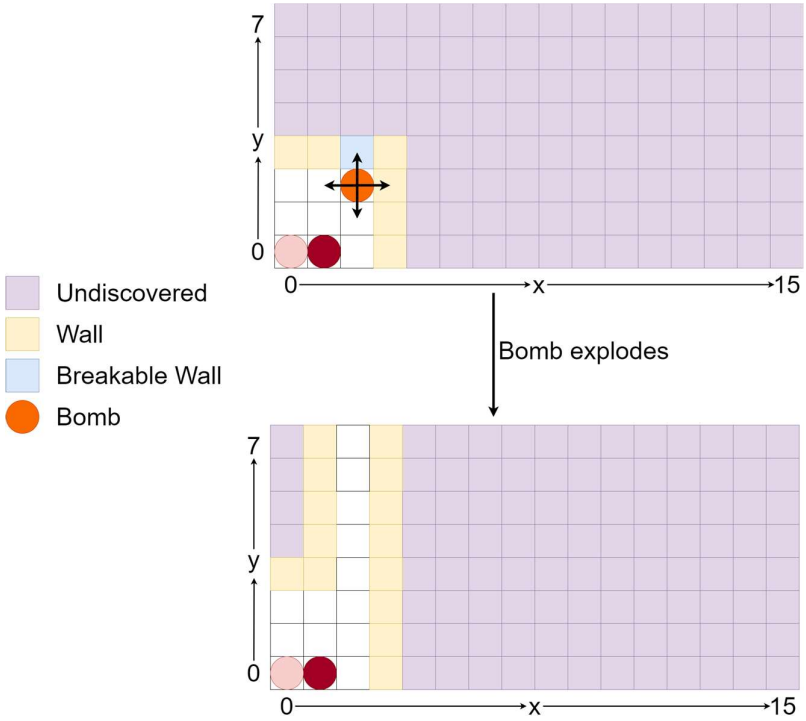
The goal of the game is to collect all of the diamonds and leave through the exit. The player is able to move around the playing field but cannot walk through any walls. The player direction indicator is used to display which direction the player is facing. The player can only view squares that have been 'discovered'. An empty square is discovered if the player can walk to that square from their current location, a wall is discovered if an adjacent empty square is discovered. The initial display is shown below.



All walls (breakable and unbreakable) appear the same initially. The player can 'inspect' any wall they are facing (the location of the player direction indicator). If they inspect a breakable wall, the display of the wall will change to indicate that it is breakable as shown below.



The player may also place bombs at their current location, which will detonate after a short delay. The bomb will explode the square it is on and directly adjacent squares (not diagonal). If the player is on an exploded square, the game will be over. If a breakable wall (whether inspected or not) is exploded, it will be removed from the game and made an empty square. When this happens, new squares may become discoverable. An example bomb explosion is shown below. This is after the player places a bomb at (2,2) then moves to (1,0).



The player can use bombs to discover new areas and find the hidden diamonds. When the player has collected all diamonds, they can exit the level through the exit on the right-hand side of the playing field. When they exit the level, they will go to the next level.

Initial Operation

The provided program has very limited functionality. A start screen is displayed, which detects the terminal character 's' and any rising edge on the pins connected to buttons 0-3 (pins B0-B3 for IN students, pins C0-C3 for EX students). Pressing of any of these will start a game with the player at position (0,0) and a flashing direction indicator at position (1,0).
Once started, the program detects a rising edge on the pin connected to button 0, but no action is taken on this input.
The design of the first level is provided in the code, but some features will need to be implemented before being able to see it.

For internal students, the Baud rate being used is 19200, for external students, the Baud rate being used is 38400.

For EX Students: The game display is partially positioned outside the PuTTY screen with the default PuTTY window size. You will need to enlarge the PuTTY display in order to get the entire game to display. In PuTTY, you can save and load sessions with the COM port and baud rate. You can also change the window size under Window → Rows. We will be using a window size with 80 columns and 35 rows to mark your assignment.

Wiring Advice

When completing this assignment, you will need to make additional connections to the ATmega324A/ ATMega328P. To do this, you will need to choose which pins to make these connections to. For IN students, there are multiple ways to do this, so the exact wiring configuration will be left up to you, and you should communicate this using your submitted feature summary form.

Due to the reduced number of external pins available with the Arduino Uno for EX students, all pins have to be used to add all features and working out a valid configuration could be quite tricky, as a result a configuration is provided below which you are encouraged to use to ensure everything fits. Note that pins D1 and D0 are the TX/RX pins, which are being used to communicate to the serial terminal and no connections should be made to them.

For EX Students Only

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
B			SSD G	LEDs		Piezo	SSD CC1	SSD CC0
C			ADC connections		Button 3	Button 2	Button 1	Button 0
D	SSD A-F						Reserved for RX/TX Baud rate: 38400	

Program Features

Marks will be awarded for features as described below. Part marks will be awarded if part of the specified functionality is met. 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, unless the feature can be demonstrated. You may implement higher-level features without implementing all lower level features if you like (subject to prerequisite requirements). The number of marks is **not** an indication of difficulty. It is much easier to earn the first 50% of marks than the second 50%.

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. For some of the easier features, the description below tells you which code to modify or there may be comments in the code which help you.

Minimum Performance (Level 0 – Pass/Fail)

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

Start Screen (Level 1 – 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. Do this by modifying the function `start_screen()` in file *project.c*.

Move Player with Buttons (Level 1 – 10 marks)

The provided program does not allow the player to move. Modify the program so that button 0 (connected to pin B0 for IN students, connected to pin C0 for EX students) moves the player to the right, button 1 moves the player downwards, button 2 moves the player up and button 3 moves the player left. The player should not be able to move through walls or off the edge of the game area. Note that the player direction indicator does not need to be moved as part of this feature, that is part of "Player Direction Indicator" below.

In the `play_game()` function in the file *project.c*, when button 0 is pressed, the function `move_player(dx,dy)` in the file *game.c* is called. This function is currently empty, start by filling in the `move_player` function, there are some hints to get you started. Then update `play_game` to detect buttons 1, 2 and 3 as well.

Move Player with Terminal Input (Level 1 – 6 marks)

The provided program does not register any terminal inputs once the game has started. Modify the program such that pressing 'W' moves the player upwards, 'A' moves the player to the left, 'S' moves the player downwards and 'D' moves the player to the right. Both the lower case and upper case of each letter should move the cursor.

On the start screen, the game can be started by pressing 's' or 'S', looking at the function `start_screen()` should give you an idea of how to read serial input from the terminal.

Note that if you press the directional arrows, they might also move your cursor in an unexpected direction, this is because they use escape characters that contain 'A' and 'D'. We will not be considering escape characters when testing and will only use the letters of the alphabet, digits and spacebar as potential inputs.

Player Direction Indicator **(Level 1 – 8 marks)**

(This assumes that you have implemented "Move Cursor with Buttons" above.) When the game begins, there is a cursor flashing at position (0,1). This cursor is used to indicate which direction the player is currently facing. Whenever a player move is attempted (button or one of 'WASD' pressed) whether successful or not, the player direction indicator should be updated as well. If the player attempts to move to the right, the player direction indicator should appear one square to the right of the player etc.

The player direction indicator should restart its flashing cycle every time a move is attempted. If the player direction indicator would appear off the screen (e.g if the player is at the (0,0) and facing downwards) then there should be no display.

Inspect Square **(Level 1 – 8 marks)**

(Assumes that "Player Direction Indicator" is implemented.) Some of the walls are breakable as described in the "Diamond Miners Description". Initially, all walls are displayed the same colour whether or not they are breakable. When 'e' is pressed, the player should 'inspect' the square they are currently facing (i.e the square where the player direction indicator is). If the square being inspected is a breakable wall, the colour of the wall should be changed to a new colour to indicate that it is breakable. This new colour should be different to any colour already used on the display. You may want to add a new 'object' to those defined in display.h.

Cheat Mode **(Level 1 – 11 marks)**

(Assumes that "Inspect Square" is implemented.) Pressing 'c' should enable/disable 'cheat mode', cheat mode should begin disabled. Whether cheat mode is enabled or disabled should be displayed on the terminal. When in cheat mode, if a breakable square is inspected, it should be immediately removed from the display. When a breakable wall is removed from the playing field, the discoverable area (defined in "Diamond Miners Description") will be changed.

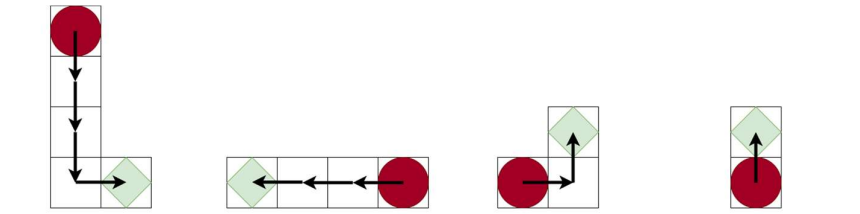
The function `discoverable_dfs(x,y)` is provided, which uses a depth first search to update the display of all discoverable squares. This function may be called with the (x,y) coordinate of the broken wall and should update the display for you.

Diamond Collection **(Level 1 – 7 marks)**

(This assumes that you have implemented "Cheat Mode" above.) When the player walks over a diamond, they should collect it. When a diamond is collected, it should be removed from the playing field. The number of diamonds which have been collected should be displayed to the terminal in a fixed location. This number should be displayed as zero when the game begins.

Diamond Detector **(Level 2 – 7 marks)**

(Assumes that "Cheat Mode" is implemented.) Your player has been equipped with a diamond detector. The diamond detector can tell how close the player is to a diamond, regardless of if there are walls in the way or if the diamond has been discovered. This distance will be communicated using LED L2 (green) by toggling the LED on and off at different speeds. Distance should be measured using Manhattan distance and the distance to the nearest diamond should be considered. The speed of the flash is outlined below with examples of how Manhattan distance is calculated. When the Manhattan distance to the nearest diamond is greater than 4, the LED should be off.



For external students, if you do not have a green LED, that is okay. Just light up any LED and include on your feature summary form which LED is used for the diamond detector.

Bombs **(Level 2 – 7 marks)**

Allow the player to place a bomb by pressing spacebar. The bomb should be placed at the player's current location. Only one bomb can be in play at any time. The bomb must be a different colour to any of the existing colours on the display. The bomb should not be visible if the player is standing on top of it. 2 seconds after the bomb was placed, it should detonate, exploding the bomb's square and adjacent squares as described in "Diamond Miners Description". Any breakable walls should be removed (whether inspected or not), and new discoverable areas made visible. If the player is on one of the exploded squares, the game should end. A new game should be able to be started using the existing `handle_game_over()` function.

Visual effects are not required for this feature and are covered in "Bomb Visuals".

Step Counter **(Level 2 – 7 marks)**

(Assumes that "Move Player with Buttons" is implemented.) The number of steps taken by the player should be displayed to the seven segment display. If 100 or more steps have been taken, '99' should be displayed. For step counts from 0 to 9 inclusive the left seven segment digit should be blank. No display flickering should be apparent. Both digits (when displayed) should be of equal brightness. Include any connections required on your submitted feature summary form.

Game Pause **(Level 2 – 7 marks)**

Modify the program so that if the 'p' or 'P' key on the serial terminal is pressed then the game will pause. When the button is pressed again, the game recommences. (All other button/key presses/inputs should be discarded whilst the game is paused – i.e. the player cannot be moved, no bombs can be exploded etc.) All timed displays must be unaffected – e.g. if the pause happens 200ms before the player direction indicator is going to flash off, then the player direction indicator should not flash off until 200ms after the game is resumed, not immediately upon resume.

Next Level **(Level 2 – 7 marks)**

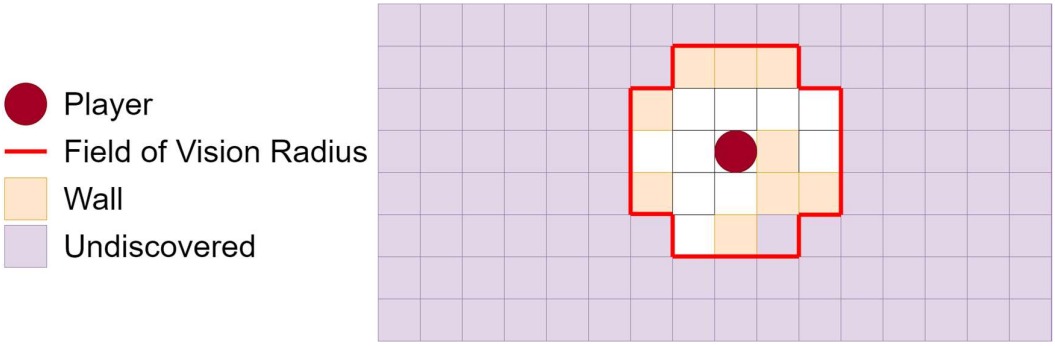
(Assumes that "Diamond Collection" is implemented.) The provided game has the layout for one level. Once all diamonds are collected, the player should be able to complete the level by exiting through the gap on the right-hand side of the playing field. The player should not be able to leave if any diamonds have not been collected.

The next level should have a different level design. An indefinite number of levels should be able to be played, but there only needs to be two distinct level designs (i.e. the level designs can alternate).

Field of Vision **(Level 3 – 6 marks)**

Pressing 'f' should enable/disable 'field of vision', field of vision should begin disabled. When field of vision is enabled, it will alter the display of the game. When field of vision is enabled, only a radius around the current player will be visible as shown below. Any squares inside the

vision radius that would be visible normally (i.e. discoverable) should be visible. Any squares outside this vision radius should not be visible, these squares should be the 'undiscovered' colour.



This field of vision radius should move with the player as they move and should not introduce lag into gameplay.

Joystick **(Level 3 – 6 marks)**

Add support to use the joystick to move the player left, right, up and down. This must include support for auto-repeat – if the joystick is held in one position then, after a short delay, the code should act as if the joystick was repeatedly moved in that direction. Your movement must be reasonable – i.e a reasonable player could stop at whichever square they desired. Your player must be shown to move through all positions and be able to be stopped at that position if the joystick is released. 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. For external students, this will be marked more leniently as your joystick likely differs more substantially.

Bomb Visuals **(Level 3 – 6 marks)**

(Assumes that "Bombs #1" is implemented.) This feature involves adding visual effects for the bombs being placed. When the bomb is placed, it should flash progressively faster before exploding. There should be an animation when the bomb explodes that covers at least the bomb's location and directly adjacent squares. The animation should not interfere with the display after the animation is finished.

Led L1 (red) should be lit whenever the player is "in danger". The player is in danger if they are at the same square as the bomb or on an adjacent square.

Sound Effects **(Level 3 – 6 marks)**

Add sound effects to the program which are to be output using the piezo buzzer. Different sound effects (tones or combinations or tones) should be implemented for at least four events. (At least two of these must be sequences of tones for full marks.) For example, choose four of:

- Player movement
- Successful inspection
- Unsuccessful inspection
- Bomb countdown
- Bomb explosion
- Game start-up
- Constant background tune

Do not make the tones too annoying! Pressing the 'm' or 'M' key on the serial terminal must be used to toggle sound on and off, sound should be on when the game starts. You must specify

which AVR pin the piezo buzzer must be connected to. (The piezo buzzer will be connected from there to ground.) Your feature summary form must indicate which events have different sound effects. Sounds must be tones (not clicks) in the range 20Hz to 5kHz. Sound effects must not interfere with game play, e.g. the speed of play should be the same whether sound effects are on or off. Sound must turn off if the game is paused.

Assessment of Program Improvements

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, for example, if you implement game pausing then sound effects should pause.

Features are shown grouped in their levels of approximate difficulty (level 1, level 2, and level 3). Some degree of choice exists at levels 2 and 3, but the number of marks to be awarded is capped. i.e., you can't gain more than 18 marks for level 3 features even if you successfully implement all level 3 features.

Submission Details

The due date for the project is **4:00pm Monday November 1 2021**. 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);
- Your final .hex file (suitable for downloading to the ATmega324A/ATmega328P AVR microcontroller program memory); and
- A PDF feature summary form (see below).

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 file – do not use folders/directories or other zip/rar files inside the zip file.

If you make more than one submission, each submission must be complete – the single zip file must contain the feature summary form, the hex file 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.

The feature summary forms are on the last pages of this document, there is a separate feature summary form for IN and EX students. A separate electronically-fillable PDF form will be provided to you also. This form can be used to specify which features you have implemented and how to connect the ATmega324A/ATmega328P to peripherals 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 we will NOT remark your submission). 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.

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 obviously no changes you made to 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 mark deduction will apply – **1 mark penalty per warning up to a maximum of 10 marks.** 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.

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.

Late Submissions

Late submission will result in a penalty of 10% plus 10% per calendar day or part thereof, i.e. a submission less than one day late (i.e. submitted by 4:00pm Tuesday November 2, 2021) will be penalised 20%, less than two days late 30% and so on. (The penalty is a percentage of the mark you earn (after any of the other penalties described above), not of the total available marks.) Requests for extensions should be made via the process described in the course profile (before the due date) and be accompanied by documentary evidence of extenuating circumstances (e.g. medical certificate). The application of any late penalty will be based on your latest submission time.

Notification of Results

Students will be notified of their results via Blackboard's "My Grades".

The University of Queensland – School of Information Technology and Electrical Engineering
Semester 2, 2021 – CSSE2010 / CSSE7201 Project – Feature Summary EXTERNAL

Student Number	Family Name	Given Names

An electronic version of this form will be provided. You must complete the form and include it (as a PDF) in your submission. You must specify which IO devices you've used and how they are connected to your ATmega328P.

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
B								
C					Button 3	Button 2	Button 1	Button 0
D							Reserved for RX/TX	
							Baud rate: 38400	

Feature	✓ if attempted	Comment (Anything you want the marker to consider or know?)	Mark	
Start screen			/4	
Move Player with Buttons			/10	
Move Player with Terminal Input			/6	
Player Direction Indicator			/8	
Inspect Square			/8	
Cheat Mode			/11	
Diamond Collection			/7	/54
Diamond Detector			/7	
Bombs			/7	
Step Counter			/7	
Game Pause			/7	
Next Level			/7	/28 max
Field of Vision			/6	
Joystick			/6	
Bomb Visuals			/6	
Sound Effects			/6	/18 max
			Total: (out of 100)	
			General deductions: (errors in the program that do not fall into any above category, e.g. general lag in gameplay)	
			Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty)	
			Final Mark: (excluding any late penalty which will be calculated separately)	

The University of Queensland – School of Information Technology and Electrical Engineering
Semester 2, 2021 – CSSE2010 / CSSE7201 Project – Feature Summary INTERNAL

Student Number	Family Name	Given Names

An electronic version of this form will be provided. You must complete the form and include it (as a PDF) in your submission. You must specify which IO devices you've used and how they are connected to your ATmega324A.

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 3	Button 2	Button 1	Button 0
C								
D							Serial RX	Serial TX
							Baud rate: 19200	

Feature	✓ if attempted	Comment (Anything you want the marker to consider or know?)	Mark	
Start screen			/4	
Move Player with Buttons			/10	
Move Player with Terminal Input			/6	
Player Direction Indicator			/8	
Inspect Square			/8	
Cheat Mode			/11	
Diamond Collection			/7	/54
Diamond Detector			/7	
Bombs			/7	
Step Counter			/7	
Game Pause			/7	
Next Level			/7	/28 max
Field of Vision			/6	
Joystick			/6	
Bomb Visuals			/6	
Sound Effects			/6	/18 max
			Total: (out of 100)	
			General deductions: (errors in the program that do not fall into any above category, e.g. general lag in gameplay)	
			Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty)	
			Final Mark: (excluding any late penalty which will be calculated separately)	