The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2022

CSSE2010/CSSE7201 Assignment 2

Due: **4:00pm (AEST) Friday June 3, 2022**
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 the board game Teeko (a description is given on page 3).
For IN students: the AVR ATmega324A microcontroller runs the program and receives input from a number of sources and outputs a display to an LED matrix, with additional information being output to a serial terminal and – to be implemented as part of this project – a seven segment display and other devices.
For 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 devices.

The version of Teeko 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 board for the game Teeko with a cursor that flashes on and off. You can add features such as moving the cursor, placing pieces, detecting legal moves, timing, 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/serial terminal. 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. A getting started video has been provided on Blackboard.

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.

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 board used to store the state of the game and the position of the cursor. You should read this file and understand what representation is used for the board state and the cursor position. 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 push buttons. It sets up pin change interrupts on those pins and records rising edges (buttons being pushed). For EX students this code also handles button 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. Colours are defined in **terminalio.h** for EX students.
- **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 that is used to time various game events.

Teeko Description

This project involves creating a replica of the board game 'Teeko'. Teeko is a turn-based game played between two players on a 5x5 board. Each player has a set of 4 coloured pieces (green for player 1, red for player 2) they must take turns placing on the board. A player wins the game if they manage to place all of their pieces in a line (which may be a diagonal, vertical or horizontal line). Note that these lines *do not* wrap around the edges of the board.

The game consists of two phases, referred to in this document as game phase 1 and game phase 2.

1. In game phase 1, the two players take turns placing their 4 pieces on the board. These pieces may be placed on any *empty* space on the board. This phase ends when all 8 pieces (4 green pieces + 4 red pieces) have been placed on the board.
2. In game phase 2, players take turns picking up one of their pieces and moving it to one of the adjacent empty spaces. An adjacent space is one of the 8 surrounding spaces but does not wrap around the edges of the board. A piece cannot be placed in the same space it was picked up.

Note: In proper Teeko, players can also win by placing each of their pieces in a square of 4 adjacent spaces, however for this assignment you DO NOT need to consider this.

These rules are also described well on the Teeko Wikipedia page. Another useful resource is teeko.cc, which is an online version of the game you may wish to compare your implementation against. **NOTE:** these two resources allow players to win by placing pieces in a square as in the note above. Once again, you do not need to implement this behaviour.

The following figures illustrate the game layout and some of the expected game functionality.

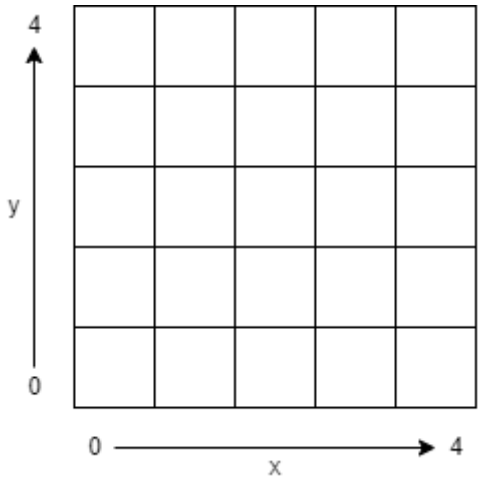

Figure 1: The initial game board. Note the coordinate system used.

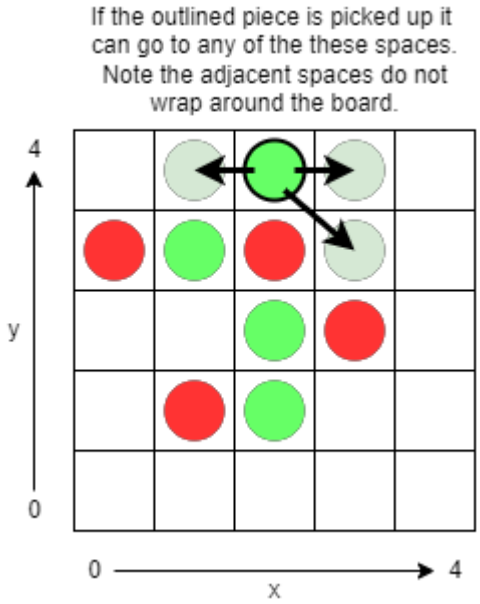

Figure 2: The valid moves when picking up a piece are shown in light green.

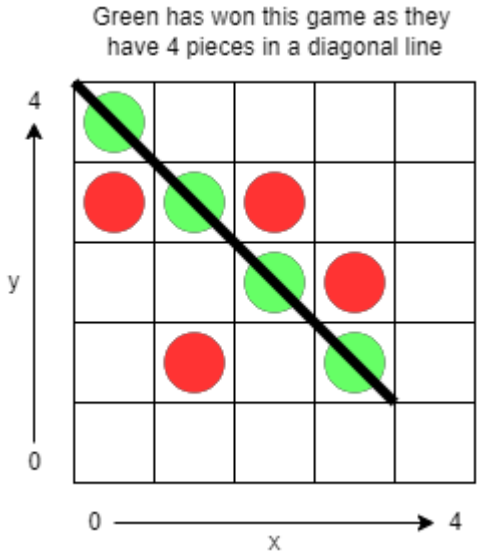

Figure 3: An example of the green player winning the game.

Initial Operation

The provided program has very limited functionality. It will display a start screen which detects the rising edge on the push buttons B0, B1, B2 and B3, and also the input terminal character 's'. Pressing of any of these will start a game with the start configuration and a flashing cursor.

Once started, the program detects a rising edge on the button B3, but no action is taken on this input.

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.

Recommended EX wiring

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
B			SSD DP	SSD G	LED	Piezo Buzzer	SSD CC2	SSD CC1
C			ADC connections		Button B3	Button B2	Button B1	Button B0
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 Cursor with Buttons (Level 1 – 12 marks)

The provided program does not allow for the cursor to be moved. Modify the program so that button B3 (connected to pin B3 for IN students, connected to pin C3 for EX students) moves the cursor to the left, button B2 moves the cursor right, button B1 moves the cursor up and button B0 moves the cursor down. (B3 = left, B2 = right, B1 = up, B0 = down). If the cursor moves off the display, it should wrap around to the opposite side. For example, if the cursor is at space (3,4) and attempts to move upwards, it should wrap to the bottom to (3,0). When the cursor moves, it must be visible immediately and *should restart its flashing cycle* i.e. the cursor should not flash until 500 ms after the cursor has been moved. You may need to modify/relocate some of the existing cursor flashing code seen in **project.c**.

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

Move Cursor with Terminal Input (Level 1 – 5 marks)

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

Game Phase 1 (Level 1 – 8 marks)

(This assumes that you have implemented "Move Cursor with Buttons/terminal input" above.)
Modify the program so that a piece can be placed at the current location of the cursor when the spacebar is pressed. For this level, the piece played does not have to be a legal move as per game phase 2, however you are not allowed to place a piece on top of another piece. When a piece is successfully placed, the current player should switch (it starts as player 1/green). If a player tries to place a piece on top of another piece, this move should be rejected and the current player should not change.

Turn Indicator (Level 1 – 6 marks)

(This assumes that you have implemented "Game Phase 1" above.)
Display the current players turn and the corresponding colour on the terminal in a *fixed position*. Add appropriate text (e.g. "Current player: 1/2, (green/red)") to the terminal display so it is clear which players turn it is. As the first player to place a piece is player 1, the terminal should show "Current player: 1 (green)" (or similar) when the game begins. This text should be updated *only when it changes* i.e. you should not be continuously printing the current player to the terminal.

Game Phase 2 (Level 1 – 10 marks)

(This assumes that you have implemented "Game Phase 1" above.)
After each player places down their 4 pieces, pick up a piece by pressing the space bar while the cursor is above a piece. You should only be able to pick up the current player's pieces. Once a piece has been picked up, you can place it again with the space bar after moving the cursor to a legal space (any one of the 8 space around where the piece was picked up that is empty). Note: *you cannot place a piece at the same location the piece was picked up*. These 8 spaces *do not wrap around the board* like the cursor does (only the cursor wraps around the board). The cursor should flash a different colour while a piece is picked up. Several (unused) colours have been provided in **pixel_colour.h** (IN) **/terminalio.h** (EX).

Valid Move Detection on LED (Level 1 – 7 marks)

(Assumes "Game Phase 1/2" is implemented) Display on an LED whether placing a piece in the current cursor location is a valid move or not. This LED should be on when the move is valid, otherwise the LED should be off. This must work for both game phase 1 and 2. In phase 1, any move is valid as long as it is not on top of another piece. In phase 2, both selecting and placing a held piece must be displayed on the LED. When picking up a piece the LED should be on when the cursor is above one of the current player's pieces. When holding a piece, the LED should be on when the cursor is above one of the 8 empty spaces around the space the piece was picked up. This LED must update *as soon as a move is made*.

Game Over (Level 1 – 12 marks)

(Assumes "Game Phase 1/2" is implemented)
Add functionality to detect if a player has won the game i.e. successfully placed 4 pieces in a row, column or diagonal. If a win is detected then a message should be displayed on the terminal indicating which player has won the game. A new game should be able to be started after the game has ended. Some of this code has already been provided for you in **project.c**.

Hint: To detect a win has occurred, the following steps may be taken:

1. Locate each of the current player's pieces.
2. For each piece, check if a line of 3 *other* pieces exists in each of the 8-possible directions.

Some more advice:

- Ensure you only check positions that are within the game board.
- Loops can be used to check each direction methodically. One possible solution is to define arrays of directions to search. Directions can be defined in terms of the x and y directions as in `move_display_cursor(dx,dy)`.
- You must not only search around the piece just placed. Consider the situation where the final move is placing a piece between the other 3 pieces.

Visual Display of Legal Moves (Level 2 – 7 marks)

In addition to displaying whether a move is valid on the LED, also show all of the legal moves on the display (LED matrix (IN)/terminal (EX)) when holding a piece. You do not need to show legal moves in phase 1 of the game or when no piece has been picked up. The legal moves must be shown in a new colour. Several (unused) colours have been provided in **pixel_colour.h** (IN) **/terminalio.h** (EX).

Longest Line Display (Level 2 – 7 marks)

(This assumes that you have implemented "Game Phase 1" above.)
Display on the seven segment display the current longest line of pieces each player has. The longest line of player 1 (green) should be shown on the left digit and the longest line of player 2 (red) should be shown on the right digit. 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. At the start of the game '00' should be displayed. You should be able to extend the functionality implemented to detect game over for this.

Turn Timing (Level 2 – 7 marks)

(This assumes that you have implemented "Game Phase 1" above.)
Add the ability to toggle between different difficulties with different times for players to make moves. There are 3 difficulties – easy, medium and hard.

- Easy difficulty has no time limit and is the default game behaviour.
- Medium difficulty has a time limit of 20 seconds per move.
- Hard difficulty has a time limit of 10 seconds per move.

When playing one of the timed difficulties, the remaining time is reset when a player makes a move and if either player runs out of time then they forfeit the game and the other player is declared the winner.

During a timed game, the time remaining should be displayed for the current player on the seven segment display and overrides the longest line display (if implemented). The remaining time must *count down* and should show the remaining time in seconds for the current player when the remaining times is 10+ seconds. When the remaining time is less than 10 seconds, the remaining seconds (and the decimal point) should be shown on the leftmost digit and the tenths of seconds shown on the rightmost digit. In easy mode, the seven segment display should show the longest line display (if implemented) or otherwise be blank.

The game difficulty may be changed at any point by pressing:

- 'e'/'E' – select easy difficulty
- 'm'/'M' – select medium difficulty
- 'h'/'H' – select hard difficulty

When the difficulty is changed the timer is reset to the original time i.e. you do not have to remember the remaining time when switching difficulties.

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 'p' or 'P' key on the serial terminal is pressed again, the game recommences. (All other button/key presses/inputs should be discarded whilst the game is paused – i.e. the cursor cannot be moved, and no pieces can be placed.) The cursor flashing must be unaffected – e.g. if the pause happens 200ms before the cursor is going to flash off, then the cursor should not flash off until 200ms after the game is resumed, not immediately upon resume. Other functionality such as the seven segment display should be unaffected i.e. both digits of the seven segment display should still be shown (without ghosting etc.) if implemented.

Visual Effects **(Level 3 – 5 marks)**

(This assumes that you have implemented "Game Phase 2/Game Over" above.)
Implement visual animations to the display when certain events happen. These animations must involve multiple pixels and must not only change the colour of game elements. Implement visual effects for the following events:

- When all 8 pieces have been placed (i.e. game phase 2 has begun).
- When the game is won.

Possible visual effects include flashing the 8 pieces when game phase 2 begins and flashing the winning players pieces when the game is over. These pieces could be flashed simultaneously, randomly, or in some order.

Best of 3 Tournament **(Level 3 – 5 marks)**

Convert the game to a best of 3 tournament to determine the winning player. Instead of the game being over when a player has won, the winner is declared as the first player to reach a total of two wins. The current number of wins for both players should be displayed on the terminal as soon as the tournament begins.

Joystick **(Level 3 – 5 marks)**

Add support to use the joystick to move the cursor 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 space they desired. Your cursor 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.

Sound Effects **(Level 3 – 5 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 three events. (At least one of these must be sequences of tones for full marks.) For example, choose three of:

- Cursor being moved
- Legal move being made
- Illegal move being made
- Game start-up
- Game over
- Constant background tune

Do not make the tones too annoying! Pressing the 'q' or 'Q' 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). Note that maximum marks for level 2 and level 3 features are capped at 21 and 15, respectively, as indicated in the feature summary forms at the end of this document.

Submission Details

The due date for the project is **4:00pm Friday 3 June 2022**. 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 on Blackboard. 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 assignment (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

As per the ECP, **late submissions (i.e. submitted after 4:00pm AEST Friday 3 June 2022) will result in a penalty.** A late penalty of 10% of the maximum possible mark for the assessment item will be deducted per calendar day (or part thereof), up to a maximum of seven (7) days. After seven days, no marks will be awarded for the item. A day is considered to be a 24-hour block from the assessment item due time. Negative marks will not be awarded.

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 or personal statement). 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 1, 2022 – 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 B3	Button B2	Button B1	Button B0
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 Cursor with Buttons			/12	
Move Cursor with Terminal Input			/5	
Game Phase 1			/8	
Turn Indicator			/6	
Game Phase 2			/10	
Valid Move Detection on LED			/7	
Game Over			/12	/64
Visual Display of Legal Moves			/7	
Longest Line Display			/7	
Turn Timing			/7	
Game Pause			/7	/21 max
Visual Effects			/5	
Best of 3 Tournament			/5	
Joystick			/5	
Sound Effects			/5	/15 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 1, 2022 – 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 B3	Button B2	Button B1	Button B0
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 Cursor with Buttons			/12	
Move Cursor with Terminal Input			/5	
Game Phase 1			/8	
Turn Indicator			/6	
Game Phase 2			/10	
Valid Move Detection on LED			/7	
Game Over			/12	/64
Visual Display of Legal Moves			/7	
Longest Line Display			/7	
Turn Timing			/7	
Game Pause			/7	/21 max
Visual Effects			/5	
Best of 3 Tournament			/5	
Joystick			/5	
Sound Effects			/5	/15 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)