The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2019

CSSE2010 / CSSE7201 PROJECT

Due: **7:50pm Friday May 31, 2019**
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 Atmel Studio environment.

You are required to modify a program in order to implement additional features. The program is a simple version of Asteroids. 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.

The version of Asteroids provided to you will implement very basic functionality – it will present a fixed asteroid field, allow the base station to move only in one direction, and allow projectiles to be fired, but not detect when they hit asteroids. You can add features such as scoring, detecting hits, appropriate movement of the base station, increasing the speed of play, 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 15% 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 15% requirement in order to pass the course, but a late penalty will be applied to the mark for final grade calculation purposes.

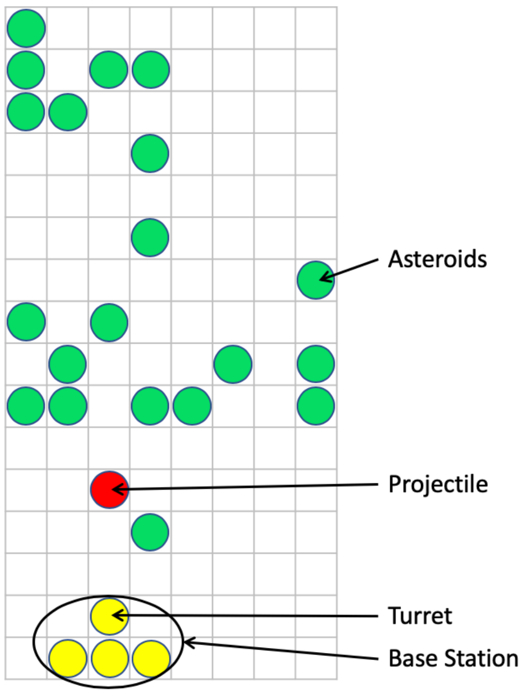
Program Description

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 operations (e.g. movements) in the game. You should read this file and understand what representation is used for the game. You will need to modify this file to add required functionality.
- **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).
- **ledmatrix.h/ledmatrix.c** – 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** – this file contains definitions of some useful colours
- **score.h/score.c** – a module for keeping track of and adding to the score. This module is not used in the provided code.
- **scrolling_char_display.h/scrolling_char_display.c** – this contains code which provides a scrolling message display on the LED matrix board.
- **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** – this file encapsulates all SPI communication. Note that by default, all SPI communication uses busy waiting – 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.

Terminology

The figure below illustrates the terminology used in the game and the initial positions of the asteroids, and a projectile after it has been fired from the base station.



Initial Operation

The provided program responds to the following inputs:

- Rising edge on the button connected to pin B3
- Serial input character 'L' or 'l' (lower case L)
- Serial input escape sequence corresponding to the cursor-left key

All of these move the base station left by one position.

The program also responds to:

- Rising edge on the button connected to pin B2
- Serial input character ' ' (i.e. space)
- Serial input escape sequence corresponding to the cursor-up key

All of these cause a projectile to be fired from the base station (provided there aren't too many projectiles in flight already).

The program also responds to:

- Rising edge on the button connected to pin B0
- Serial input character 'R' or 'r'
- Serial input escape sequence corresponding to the cursor-right key

All of these are intended to move the base station right by one position but currently move the base station left by one position.

Code is present to detect the following, but no actions are taken on these inputs:

- Rising edge on button connected to pin B1
- Serial input escape sequences corresponding to the cursor down key
- Serial input characters 'p' and 'P' (intended to be the pause/unpause key)

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 and show an asteroid field, allow the base station to be moved and allow projectiles to be fired. No marks can be earned for other features unless this requirement is met, i.e., your project mark will be zero.

Splash Screen (Level 1 – 4 marks)

Modify the program so that when it starts (i.e. the AVR microcontroller is reset) it scrolls a message to the LED display that includes your student number. You should also change the message output to the serial terminal to include your name and student number. Do this by modifying the function `splash_screen()` in file *project.c*.

Move Base Station Right (Level 1 – 4 marks)

The provided program only moves the base station to the left (in response to the buttons and keypresses described above). You must complete the `move_base()` function in file *game.c* in order to enable the base station to be moved to the right also.

Base Station Limits (Level 1 – 6 marks)

The provided program allows the base station to move off the display. You must modify the `move_base()` function in *game.*c in order to prevent this. (The turret should be permitted to move to the edge columns – i.e. the edge of the base station is off the display, but the base station is able to fire projectiles up any of the columns.)

Hit Detection (Level 1 – 10 marks)

Modify the program so that it detects when a projectile hits an asteroid and removes both the projectile and the asteroid from the game field. You will need to modify the `advance_projectiles()` function in the file *game.c* and make use of other functions in *game.c*.

Replacement Asteroids (Level 1 – 10 marks)

(This assumes that you have implemented “Hit Detection” above and are removing asteroids when hit.) Modify the program so that any asteroid removed is replaced by a new asteroid in a random position, not already occupied and not in the lowest three rows. Consult the code in `initialise_game()` in *game.c* for ideas on how to randomly place asteroids. (If “Falling Asteroids” is implemented (see below) then replacement asteroids should appear in a random column in the top row – not already occupied by an asteroid or projectile.)

Scoring #1 (Level 1 – 10 marks)

(This assumes that you have implemented "Hit Detection" above.) Add a scoring method to the program so that 1 is added to the score each time a projectile hits an asteroid. You should make use of the function `add_to_score(uint16_t value)` declared in *score.h*. You should call this function (with an appropriate argument) from any other function where you want to increase the score. If a .c file does not already include *score.h*, you may need to #include it. You must also add code to display the score (to the serial terminal in a fixed position) and update the score display only when it changes. The displayed score must be right-aligned – i.e. the right-most digit in the score must be in a fixed position. (The score need not be on the right hand edge of the terminal display – it just must be right-aligned within a given field position – i.e. the least significant digit must always be in the same location.) A score of 0 must be displayed when the game starts. Add appropriate text (e.g. "Score:") to the terminal display so it is clear where the score is being displayed. If game-over is possible (see functionality below) then the score must remain displayed on game-over (until a new game commences when the score should be reset).

Scoring #2 (Level 1 – 10 marks)

(Assumes that Scoring #1 is implemented.) Display the score on the seven segment display. The score should start at 0. For scores from 0 to 9 inclusive the left seven segment digit should be blank. If scores greater than 99 are possible in your game then make a reasonable assumption about what should be displayed. (This is unlikely to be tested.) No display flickering should be apparent. Both digits (when displayed) should be of equal brightness.

Falling Asteroids (Level 2 – 6 marks)

Add a feature so that the asteroids descend down the display (e.g. every half a second, all the asteroids are moved down by one position). Asteroids should be removed when they reach the bottom. Removed asteroids should be replaced by a new asteroid in the top row (in a random column not already occupied). You may need to vary the speed of projectiles to ensure a playable game. The base station should always remain visible.

Base Station Hit Detection (Level 2 – 6 marks)

(Assumes that "Falling Asteroids" are implemented as well as "Scoring #1".) Modify the program so that collisions between falling asteroids and the base station are detected (and the asteroid removed) and some action is taken e.g. game over or score reduced or the number of lives is reduced (see below). (If you implement a score reduction rather than reducing the number of lives then you must deal with the possibility of the current score being zero, e.g., "game over".) On game-over, the score must remain displayed on the terminal display (at least) and the game must be able to be restarted by pushing a button on the IO board. If the game is not over upon a "hit" then the removed asteroid should be replaced at the top as described in "Falling Asteroids".

Multiple Lives – Health Bar (Level 2 – 6 marks)

(Requires that "Falling Asteroids" and "Base Station Hit Detection" are implemented.) Modify the program to support multiple lives, i.e., a life is lost every time the base station is hit by an asteroid. There are four lives initially. If all lives are lost then the game is over (and the same game-over requirement as described in "Base Station Hit Detection" applies). The number of lives remaining must be indicated using 4 LEDs as a "health bar" (L5, L4, L3 and L2 on the IO Board – which are green, red, orange and green respectively). When one life is lost, a green LED should be switched off. When the second is lost, the other green LED is switched off. When the third is lost, the orange LED is switched off (leaving only the red). When the last life is lost, all four LEDs should be off. The number of lives remaining must also be shown on the terminal display. Note that the connection to the 4 LEDs must be able to be made with a single 4-wire jumper cable from 4 adjacent IO pins on the AVR microcontroller board.

Acceleration (Level 2 – 6 marks)

(Requires implementation of “Scoring #1” and “Falling Asteroids”.) Make the game speed up as the score gets higher. This can be gradual or in steps (e.g. as certain scores are reached). (Do not speed up play too quickly. An average player should be able to play your game for at least 90 seconds, but the speed-up must be noticeable within 45 seconds.)

Game Pause (Level 2 – 6 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. will not affect the movement of the base station or projectile firing when the game is unpaused.) The asteroid/projectile movement rate must be unaffected – e.g. if the pause happens 450ms before an asteroid movement is due, then the asteroid should not move until 450ms after the game is resumed, not immediately upon resume. The check for this key press is implemented in the supplied code, but does nothing.

EEPROM Storage of High Score Leader Board (Level 3 – 5 marks)

Implement storage of a leader board (top 5 scores and associated names) in EEPROM so that values are preserved when the power is off. If a player achieves a top-5 score then they should be prompted (via serial terminal) for their name. The score and name must be stored in EEPROM and must be displayed on the serial terminal on program startup and at each game-over – in decreasing order of score. If there are fewer than 5 high scores then only show those that have been saved so far. (You must handle the situation of the EEPROM initially containing data other than that written by your program. You will need to use a “signature” value to indicate whether your program has initialized the EEPROM for use.) Name entry must be resilient to arbitrary responses (i.e. invalid characters / escape sequences / button presses etc. should not cause unexpected behaviour). The only characters supported must be letters, spaces and the backspace (Ctrl-H) and delete (Ctrl-? – ASCII 127) keys (with both of the latter having the effect of a backspace). Name entry is terminated by the return/enter key. Names up to 12 characters long must be supported.

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 four events. (At least two of these must be sequences of tones for full marks.) For example, choose four of:

- Base station moving
- Projectile being fired
- Projectile hitting asteroid
- Base station hit by asteroid
- game start-up
- constant background tune

Do not make the tones too annoying! Switch 7 on the IOboard must be used to toggle sound on and off (1 is on, 0 is off). You must specify which AVR pin this switch is connected to and 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.

Joystick (Level 3 – 5 marks)

Add support to use the joystick to move the base station left and right (by moving the joystick left and right) and to fire projectiles (by moving the joystick up or 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

– e.g. do not immediately jump or appear to immediately jump to one side if the joystick is held to that side. Your base station 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 marked with a different joystick to the one you test with.

Game Display on Terminal Screen (Level 3 – 5 marks)

Display a copy of the LED matrix display on the serial terminal using block graphics and characters of various colours – possibly different colours to those used on the LED matrix. This should allow the game to be played either by looking at the LED matrix or at the serial terminal. (The serial terminal display must keep up with the LED matrix display, i.e. must be no more than about 100ms behind the LED matrix display.) The baud rate **must** remain at 19200. You can assume that the terminal display will be at least 80 columns in width and 24 rows in height (i.e. the default size in PuTTY). You will need to draw an appropriate border to indicate the game field.

Visual Effects on the LED display (Level 3 – 5 marks)

Implement visual effects on the LED display – i.e. multi-pixel animations in response to at least two events, e.g. an explosion effect when a projectile hits an asteroid. Select two of the following events:

- projectile hitting asteroid
- asteroid hitting base station
- game over

Variable Speed Asteroids (Level 3 – 5 marks)

(Requires "Falling Asteroids".) Implement asteroids which move at different speeds, i.e., some descend faster than others (but each asteroid descends at a constant rate which is randomly chosen when the asteroid appears). It must not be possible for a faster asteroid to pass through a slower one. If a faster asteroid (or group of asteroids) "catches up" with a slower one then it will slow down to the speed of the asteroid ahead of it and occupy the position immediately behind it. This functionality must be able to be demonstrated in normal game play.

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 level 3, but the number of marks to be awarded here is capped, i.e., you can't gain more than 20 marks for advanced features even if you successfully add all the suggested advanced features. You can't receive more than 100 marks for the project as a whole.

Submission Details

The due date for the project is **7:50pm Friday 31 May 2019**. 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 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 and 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 form is on the last page of this document. 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 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.

Assessment Process

Your project will be assessed during the revision period (the week beginning Monday 3 June 2019). You have the option of being present when this assessment is taking place, but whether you are present or not should not affect your mark (provided you have submitted an accurate feature summary form). Arrangements for the assessment process will be publicised closer to the time.

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 Atmel Studio 7 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 Atmel Studio 7, 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 Atmel Studio) and check for warnings in the "Error List" tab. Note that the code supplied to you has one compilation warning – the `remove_asteroid()` function in *game*.c is defined but not used. You should remove this function if you do not use it in your submission to avoid this warning.

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 7:50pm Saturday 1 June, 2019) 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 at the time of project marking (if they are present) or later via Blackboard's "My Grades".

The University of Queensland – School of Information Technology and Electrical Engineering
Semester 1, 2019 – CSSE2010 / CSSE7201 Project – Feature Summary

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**	
Splash screen			/4	
Move Right			/4	
Base Station Limits			/6	
Hit Detection			/10	
Replacement Asteroids			/10	
Scoring #1			/10	
Scoring #2			/10	/54
Falling Asteroids			/6	
Base Station Hit Detection			/6	
Multiple Lives			/6	
Acceleration			/6	
Game Pause			/6	/30
EEPROM Leaderboard			/5	
Sound Effects			/5	
Joystick			/5	
Terminal Game Display			/5	
Visual Effects			/5	
Variable Speed Asteroids			/5	/20 max

Total: (out of 100, max 100)	
Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty)	
Final Mark: (excluding any late penalty which will be calculated separately)	