The University of Queensland
School of Information Technology and Electrical Engineering
Semester 2, 2011

CSSE1000 / CSSE7035

PROJECT

Due: **5pm Saturday October 29, 2011**
Weighting: **15% (100 marks)**

Objective

As part of the assessment for this course, students are required to complete 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 AVR Studio environment, the AVR GCC compiler and associated tools.

You are required to modify an existing game program in order to implement additional features. The program is a simple version of Space Invaders / Asteroids. The AVR device runs the program and receives input from a joystick PMOD and outputs a display to an LED display board. The game involves firing projectiles at a field of asteroids from a base station that can be moved left and right at the bottom of the display.

The version of the game provided to you has 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, sound effects, increasing the speed of play over time, asteroids that descend towards the base over time etc. The features are grouped into levels which are an indication of their difficulty, level 1 features are the easiest, and most are required to achieve a passing mark. Level 2 features are more challenging and level 3 features are advanced. Note that some features cannot be implemented (and won't be marked) unless certain prerequisite features are implemented.

You are reminded that it is **mandatory to submit** and receive at least 10 marks on the project (before any late-penalty applies) in order to have a chance of passing CSSE1000/CSSE7035 (i.e. grade of 4). You must receive at least 50 marks (after any late penalty) in order to have a chance of getting a grade of 6 or 7. Consult the course profile for further details.

Individual or Group of Two

You may complete this project individually or as part of a group of two. You are required to tell us, via the link on the Course Blackboard site, by **12 noon Friday October 21, 2011** whether you are completing the project individually or as part of a group of two students. If you are completing it in a group, you must tell us who your partner is and they must also enter your details via the course Blackboard site. Failure to complete this form (by both partners) means that you will be assumed to be completing this project individually.

A group of two will be required to do additional work to get the same mark as an individual; however the amount of work is less than twice that required of an individual. Both members of a group will receive the same mark for the project. Certain features are intended mainly for groups, i.e., groups will need to implement these features while for individuals they are optional and worth fewer marks.

Don't Panic!

You have been provided with a large amount of code to start with. 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 joystick 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-libc documentation to understand some of the functions used. You may also need to consult Cerebot, PMOD or other documentation.

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.

- project.c – this is the main file that contains the event loop. You should read and understand this file
- led_display.c – this contains the code that drives the 7x15 LED display. You can change the contents of the display by updating the "display" global variable. (Note that the code makes certain assumptions about the ports used to connect the display to the AVR board. You should not change these.) More information about the LED display will be provided on the course Blackboard site. The game uses the display rotated 90 degrees anti-clockwise.
- scrolling_char_display.c – this contains code which provides a scrolling message display on the LED display board. (Orientation of the display when doing this is 90 degrees to the orientation used for game display.)
- joystick.c – contains the code which interacts with the joystick – being able to turn the LEDs on the joystick PMOD on and off and sampling the joystick position and button status.
- game.c – this encapsulates the state of the game field (where the base station, projectiles and asteroids are located) along with functions to do things to the game field (e.g. initialise it, move the base station, fire a projectiles, output the game field state to the LED display etc). You should read this file and understand how the game stores information etc.
- score.c – a module for keeping track of and adding to the score (initially unused).
- timer2.c – implements a clock tick count which we can use for timing.

Program Features

It is reasonably easy to get a passing mark by understanding and taking advantage of code provided or worked on in learning labs. You may modify any of the code you have been provided with and you may freely use code provided in learning lab sessions. You will need to specify which PMODS need to be connected to which ports on your AVR board (see the Feature Summary page at the end of this document).

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).

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 allow the base station to move left when the joystick is moved left. No marks can be earned for other features unless this requirement is met. Failure to meet this requirement

will mean a failure of the course as a whole, due to the 10% minimum project mark requirement (before any late penalty applies). Resubmissions may be possible to meet the pass requirement, but a late penalty would apply for grade calculation purposes.

Splash Screen **(Level 1)**

Modify the program so that when it starts (i.e. the AVR microcontroller is reset) it scrolls a message on the LED display that includes the student number(s) of the student(s) whose project this is. Do this by modifying the function `splash_screen()` in file *project.c*.

Move Base Station Right **(Level 1)**

The provided program only moves the base station to the left (in response to the left or right cursor keys being pressed). You must complete the `move_base()` function in file *game.c* and update the joystick checking code in *project*.c in order to enable pieces to be moved to the right also.

Base Station Limits **(Level 1)**

The provided program allows the base station to move off the display (which can cause the program to crash). 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)**

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)**

(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 `init_game_field()` in *game.c* for ideas on how to randomly place asteroids.

Scoring **(Level 1)**

Add support for a seven-segment display PMOD which displays the current score. The score should be a two digit number (0 to 99) with both digits displayed (although you may choose not to display the left digit for scores in the range 0 to 9 if you wish). The score should start at 0 and be incremented by 1 every time a projectile hits an asteroid. The score should be capped at 99.

Start New Game **(Level 1 – group feature)**

Add support for a button PMOD where one of the buttons (specify which on your feature summary) causes a new game to be started (at any time, even if the current game is in progress).

High Score **(Level 1 – group feature)**

Keep track of the high score across several games. (This requires that you can play the game multiple times without a reset – see the "Start New Game" description above.) Choose a different button on the button PMOD (the PMOD added for the new game functionality) and when that button is pressed and held down the seven segment display should show the high score rather than the current score. The game should continue to be playable during this time. If the current game's score is the high score then this should update as the game is being played. You need only store scores in RAM (i.e. they will be reset when the microcontroller is reset) – you do not need to store scores persistently (EEPROM) – this is a more advanced feature – see below.

Random Asteroid Positions **(Level 1 – group feature)**

Add a method for seeding the random number generator so that the game is not always the same after Reset. (See the srandom() function in stdlib.h) Seeding is often done based on a timer value – e.g. wait for a user to press a button to start the game and use the time that happens as the seed value.

Falling Asteroids **(Level 2)**

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

Base Station Hit Detection **(Level 2)**

(Assumes that "Falling Asteroids" are implemented as well as "Scoring" or "Health Bar".) 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. score reduced, or "health reduces" (see below)).

Acceleration **(Level 2)**

(Requires implementation of "Scoring" 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 one minute, but the speed-up must be noticeable within one minute.)

Health Bar **(Level 2)**

(Requires that "Falling Asteroids" and "Base Station Hit Detection" are implemented.) Modify the program to support an LED PMOD that implements a "health bar" – e.g. the number of lives remaining. Initially all 4 LEDs should be on. Every time the base station is hit by a falling asteroid, the player's health should reduce (an LED should be turned off). If the health falls to 0 (i.e. base station hit four times), then the game is over.

Game Pause **(Level 2 – group feature)**

Modify the program so that if a particular button is pressed on the button PMOD (the one added for the "New Game" feature) then the game is paused at the current position. The game does not recommence until the same button is pressed and released again. (All other button presses and joystick movements should be discarded whilst the game is paused.)

Scoring #2 **(Level 2 – group feature)**

(Requires "Scoring" and "Health Bar" features to be implemented). Add at least three more scoring/health features, e.g. choose two of:

- scores greater than 99 supported by scrolling the seven segment display
- health can be restored when scores reach certain levels
- penalties applied if any asteroids reach the bottom
- scoring values increase as the game progresses (e.g. related to acceleration) – requires support for scores greater than 99

Joystick Auto-repeat **(Level 3)**

In the initially provided game, after moving the joystick you must return it to the centre position before you can move the base station again. Implement a suitable auto-repeat function so that after a suitable delay, if the joystick is held to one side then the base station continues to move to that side. You must move the base station faster if the joystick is held further to the side. (Note that the joystick position is available on a five point scale: -2 (far left), -1 (partially left), 0 (centre), 1 (partially right), 2 (far right).) 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.

Sound Effects **(Level 3)**

Add sound effects to the program using the PMOD-AMP and speaker. Different sound effects (tones or combinations or tones) should be implemented for at least three events. Choose at least three of:

- projectile hitting asteroid
- asteroids moving down by one position
- base station moving
- asteroid hitting base station

Do not make the tones too annoying!

EEPROM Storage of Game **(Level 3)**

Implement storage of the game state (game field, asteroids, base station position, score, high score) into on-board EEPROM when a particular button is pressed. If that button is held down when the board is reset or powered-on then the game should restart at the same point. (Consider the situation of the EEPROM initially containing data other than that written by your program.)

Visual Effects on the LED display **(Level 3)**

Implement visual effects on the LED display – e.g. an explosion effect when a projectile hits an asteroid

Variable Speed Asteroids **(Level 3)**

(Requires "Falling Asteroids".) Implement asteroids which move at different speeds, i.e., some descend faster than others.

Health Packs **(Level 3)**

Implement "health packs" – special blinking "asteroids" which appear on the display for a limited time and if hit by a projectile result in additional health being granted to the player.

Advanced Feature(s) of Your Choice **(Level 3)**

Feel free to implement other advanced features. The number of marks which may be awarded will vary depending on the judged level of difficulty or creativity involved – up to a maximum of 5 marks.

Assessment of Program Improvements

The program improvements will be worth the number of marks shown on the mark sheet at the end of this document. 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.

Features are shown grouped in their levels of approximate difficulty (level 1, level 2, and level 3). Groups of two students must implement additional functionality to achieve the same marks at each level. 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 16 marks for advanced features even if you successfully add all the suggested advanced features.

Submission Details

The due date for the project is **5pm Saturday October 29, 2011**. The project must be submitted via the ITEE online submission system at http://submit.itee.uq.edu.au with supporting material (see below) to be submitted on paper. You must electronically submit a single .zip file containing:

- **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 Flash memory of the AVR microcontroller)

You may submit numerous times, but only the last submission will be marked. Each submission must be complete. (Do not submit (or include in your .zip file) any other files. We do **not** want .o, .lst, .map, etc files.)

You must **also submit** the feature summary form shown on the last page of this document. This will specify which features you have implemented as an aid to marking. If you have not specified that you have implemented a particular feature, we will not test for it. This form must be submitted by email, as described on the top of the form.

For those students working in a group, only one student should submit the files and only one copy of the feature summary should be submitted. Students working in a group must also both complete the acknowledgement of originality declaration (which will be generated as part of the online submission process). This declaration should be signed, scanned and emailed with the feature summary form. For students working individually, the online declaration of originality required during the submission process is sufficient. Failure to submit the feature summary by the time of marking may mean some of your features are missed during marking (and we will NOT remark your submission). Failure of a group to submit the declaration form signed by both students will result in a mark of 0 for the project. (This form will be accepted up until the time of the final examination.)

Assessment Process

Your project will be assessed during the revision period (the week beginning 31 October). 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 the feature summary). Arrangements for the assessment process will be publicised closer to the time.

Late Submissions

Late submission will result in a penalty of 10% plus 10% per working day or part thereof, i.e. a submission less than one working day late (i.e. submitted by 5pm Monday October 31) will be penalised 20%, less than two working days late 30% and so on. (The penalty is a percentage of the mark you earn, not of the total available marks.) Requests for extensions should be made to the course coordinator (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.

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/group under <u>any</u> circumstances. You must not look at or copy code from any other student/group. 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.

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 2, 2011 - CSSE1000/CSSE7035 Project - Feature Summary

	Student Number	Last Name	Given Names
Student #1			
Student #2 (if group)			

An electronic version of this form will be provided. You must complete the form electronically and submit it to csse1000@itee.uq.edu.au by 6pm on Saturday October 29 (i.e. up to one hour later than the submission deadline).

Specify which PMODs should be connected to which connectors on the Cerebot II board when testing your project.

Cerebot II Connector	Upper pins (1 to 4)	Lower pins (7 to 10)
JA	**LED Display Board – columns 0 to 7**	
JB	**LED Display Board – columns 8 to 14**	
JC		**LED Display Board – row select**
JD	**Joystick PMOD**	
JE		Not applicable
JF	*SSEG Display*	Not applicable
JG		Not applicable
JH	*Reset Button*	

Feature (For Groups)	**✓ if attempted**	**Student comment (Anything you want the marker to consider or know?)**	**Marker comment**	**Mark (out of indiv/group)**	
Splash screen				5/3	
Move Right				11/7	
Move Limits				11/7	
Hit Detection				11/7	
Replace Asteroids				11/7	
Scoring				11/7	
New Game		Which button? *Btn 0*		1/7	
High Score		Which button?		1/8	
Random Positions				1/7	/60
Acceleration				6/4	
Base hit detection				6/4	
Acceleration				6/4	
Health bar				6/4	
Game Pause		Which button?		1/4	
Scoring #2				1/4	/24
Auto-repeat				6/4	
Sound Effects				6/4	
EEPROM		Which button?		6/4	
Visual Effects				6/4	
Variable Speeds				6/4	
Health Pack				6/4	
Other Advanced				max 5/5	/16 max
				TOTAL: (out of 100, max 100)	