The University of Queensland
School of Information Technology and Electrical Engineering
Semester 2, 2009

CSSE1000 / CSSE7035

PROJECT

Due: **5pm Friday October 30**
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 for the AVR.

You are required to implement a version of the popular Snake game for the AVR board. If you are unfamiliar with Snake, there are numerous websites which describe the game. The pages at

- http://www.torontoplace.com/games/ZSnake.htm
- http://javaboutique.internet.com/Snake/

contain Java applet versions of the game which you can run within your web browser.

The AVR is connected to a 15x7 LED display for displaying game field. The AVR is also connected to a PC over the UART and receives input (keypresses) from the terminal (HyperTerminal).

We have provided you with skeleton code that you can base your design on. The skeleton code implements a 3 pixel long Snake's movement in one direction (up) with the ability to turn right and also displays several items of food on the game field. For the project, you will be required to complete this example to implement a fully working snake game. This document details a list of features that you are expected to implement. The different features have different levels of difficulty and will be worth different numbers of marks. (The number of marks does not necessarily equate to the level of difficulty.)

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 share your source files with any other student/group under <u>any</u> circumstances. You must not 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. The likely penalty for a first offence is a mark of 0 for the project.

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 course website, by **5pm Tuesday October 20** 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 web page. Failure to complete this form (by both partners) means that you will both 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.

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 provided header (.h) files show the functions that are intended to be accessible from other files. You may modify any or all of the provided files. The files provided are:

- project.c – contains the main() function
- position.h and position.c – defines the type used for storing board positions, and useful functions
- board.h and board.c – for managing the "game board"
- snake.h and snake.c – for managing aspects of the snake
- food.h and food.c – for managing aspects of the food
- score.h and score.c – for managing the score
- timer.h and timer.c – for managing AVR timer/counter 0 and the software timers
- serialio.h and serialio.c – for managing serial (UART) communication
- terminalio.h and terminalio.c – helper functions for a terminal display
- led_display.h and led_display.c – for driving the LED matrix display

Program Features

The features you are required to implement are grouped into three levels. Level 1 features are the basic functionality of the Snake game and you should complete these first. Implementing all level 1 features completely without errors will achieve a passing mark for the project (max 55%). Hints are provided for first few level one features and, in these cases, comments in the code indicate where changes need to be made. Level 2 and 3 features are more advanced and are required to achieve higher marks. Level 3 features are more advanced than level 2 and it is suggested (not compulsory) that you concentrate your effort on level 2 features before level 3. Some features are only relevant to those students working in a group of two (though individuals may earn minimal marks for implementing these features).

The marks for each feature are shown in the assessment criteria section. In some cases the higher level features rely on lower level features being implemented but in other cases they do not. You can implement any set of features that you like – e.g. you can implement a mixture of base, level 1 and level 2 features and get marks for all of them (provided they work and you've implemented any prerequisite features). You do NOT need to complete all features at a given level to receive marks for higher level features. Marks are only given when features can be demonstrated – marks are not awarded for an attempt which doesn't work.

Movement (Level 1)

Extend the program so that the snake can move freely around the game field in all four directions (up, down, left and right). Pressing an arrow key will cause the snake to turn in that direction. If the snake attempts to move in the same or opposite direction to its current direction nothing happens, i.e, if the snake is moving up it cannot change direction to down, but it can change direction to left or right. You will need to add/modify code in both project.c (main() function) and snake.c (move_snake() function) to implement this feature.

Eating (Level 1)

Extend the program so that if the head of the snake collides with the food item on the game field, the food item is removed (i.e. the snake eats the food). A new food item is then immediately created elsewhere on the game field. The new food item must not be placed on top of the snake or another food item. You will need to modify snake.c (move_snake() function) to implement this feature.

Splash Screen (Level 1)

Modify the program so that when it starts (i.e. the AVR microcontroller is reset or programmed) it displays a message on the terminal giving the name of the game (you can change this if you wish) and your name(s) and student number(s). This should be displayed until the user presses the space bar. The game does not start until the user has pressed the space bar and the splash screen has been cleared. You will need to modify project.c (splash_screen() function) to implement this feature.

Game Over **(Level 1)**

Modify the program so that if the snake either runs into itself (its own tail) or off the edge of the screen the game ends and a new game starts. The splash screen must be displayed again before the new game commences. You will need to add/modify code in both project.c (handle_game_over() function) and snake.c (move_snake() function) to implement this feature.

Scoring **(Level 1)**

Add a scoring system to the game. Every time the snake eats food, 5 points are added to the score. The score is displayed in the terminal window in a fixed position while the game is in play and updated every time food is eaten. If a new game is started, the score is reset back to 0. (No traces of the previous score are left on the display.)

New Game **(Level 1)**

Modify the program so that if the user presses the 'N' key (upper case or lower case) whilst the game is being played, a new game is started. The splash screen is not displayed if the game is restarted this way.

Game Randomisation **(Level 1 - Group Only)**

Modify the program such that the snake's initial position on the screen and the position of all food items is random. Multiple plays of the game after RESET should show different initial snake and food item positions. The snake should not start in a position where it is about to run off the edge of the screen, i.e., the player should have a reasonable length of time to turn the snake around at the start. HINT: Consider the rand() function. You will need to seed the random number generator appropriately, e.g., with the global *time* value when the user hits the space bar.

Moving Food Items **(Level 1 - Group Only)**

Modify the program so that one (randomly selected) food item jumps to another (random) spot every 5 seconds. You must also modify the program so that there are 5 food items in play at any one time (instead of 3). Eating any one food item causes another one to appear elsewhere immediately.

Growth **(Level 2)**

Modify the program so that each time the snake eats, its length increases by one pixel. The snake should be able to grow up to a maximum length of 40 pixels (and no longer).

Pause Game **(Level 2)**

Add a pause key to the game. If the user presses the 'P' key (upper case or lower case) the game is paused. The display is turned off (or cleared) while the game is paused (i.e. all LEDs are off). The terminal screen should indicate that the game is paused. The game is un-paused (resumes from the same point) and the display comes back on when the user presses 'P' a second time. (The terminal screen should no longer indicate that the game is paused.)

High Score **(Level 2)**

Make the game keep track of the high score across multiple games. The high score should be displayed on the splash screen as well as alongside the score for the current game. The high score does not have to be saved through a system reset but should survive a new game – either initiated by button press or game over.

Sound **(Level 2)**

Make the game play a short tone on the piezo speaker (connected to pins D4 and D5) each time the snake eats. The tone should have a frequency of 1kHz and a duration of 0.3 sec. The user can toggle the sound effects on and off by pressing the 'M' key (upper case or lower case) – for "mute".

Multiple Sounds **(Level 2 - Group Only)**

Make the game play a very short tone on the piezo speaker each time the snake changes direction. The tone should be a noticeably different pitch and duration to the eating sound effect (you must complete the eating sound effect to get marks for this). This sound effect should also be mutable by pressing the 'M' key (upper case or lower case).

Acceleration and Decceleration **(Level 2 - Group Only)**

Add a speed control mechanism to the game. Speed up and slow down the movement of the snake when the user presses the '+' and '-' keys respectively. Limits must be placed on how slow or fast the snake can move. The minimum speed for the snake should be 1 pixel/sec. The maximum speed should be 5 pixels/sec. The initial movement speed should be 2 pixels/sec.

Persistent High Score **(Level 3)**

Make the high score keep its value even when the system is reset by saving it to EEPROM. Make it such that the high score can be reset back to 0 while the splash screen is displayed by pressing 'X'.

Save Game **(Level 3)**

Add a save-game functionality to the game which can save the current state of the game to EEPROM. When the game is paused, the user can save the current game by pressing the 'S' key (upper or lower case). The user can load this saved game by pressing the 'L' key on the splash screen. The saved game should be kept saved even when the system is reset.

Rats **(Level 3)**

Make two of the food items wander around the game field at slow speed. The moving food items (rats) should change direction randomly and frequently to give the appearance that they are disoriented. The rats should move significantly slower than the snake itself (when the snake is moving at the default speed). A rat should never move into the snakes tail or off the game field or into other food or rats, i.e. if a movement is obstructed by the snakes tail and the edge of the field, then a different movement direction (or no movement direction) is chosen.

Jump Attack **(Level 3)**

Make the snake jump forward 3 pixels instantly when the user presses the space bar. Any rats or food items in the path are eaten. Normal movement rules apply, i.e. if the snake jumps into its tail or off the edge of the screen its game over. Implement a 5 second cool-down on the jump attack, i.e. once the jump attack is used, it cannot be used again for another 5 seconds.

Tail Cut **(Level 3)**

When the snake collides with itself, instead of ending the game, have the snake cut off its own tail. When colliding, all sections of the snake behind the collision point detach from the snake and form an obstacle on the game field. The obstacle stays on the game field for 20 seconds. If the snake crashes into the obstacle (the detached tail) the game is over. 3 points are deducted from the score for each pixel of tail cut off.

Blinking Food Items **(Level 3 - Group)**

Make the food items (and/or rats) blink on and off at a rate of 5 blinks/sec. This should make them clearly distinguishable from any nearby sections of the snake's tail (or detached tail obstacle).

Terminal Game Field **(Level 3 - Group)**

Replicate the game field on the terminal display. The user should be able to play the game by either looking at the LED display or the terminal, i.e. they should mirror each other. Use a '+' character for the head of the snake, a '*' character for a tail section of snake and a '#' character for a food item. Draw a border around the game field. Make the border, snake and food items all different (and clearly visible) colors. (There is no need to replicate blinking since different colours can be used to identify different things.)

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

Feel free to implement your own extra advanced features as long as they do not impact on the correctness or testability of the above listed features. You may receive marks for your own extra features if you get them pre-approved on the newsgroup or by email. Write a one (short) paragraph proposal for your advanced feature and post it to the newsgroup or email it to Peter Crosthwaite (peterc@itee.uq.edu.au). The number of marks which may be awarded will vary depending on the judged level of difficulty or creativity involved. You can only receive marks for your own advanced features if all level 1-2 functionality is implemented.

Assessment of Program Features

Testable program improvements will be worth the number of marks shown below. 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 can be demonstrated or if there are bugs/problems with your implementation. Note that marks can not be given for a feature unless the feature can be demonstrated. Non-functional code that you have written in an attempt to get a feature working will not be assessed.

Features are shown grouped in to their levels of approximate difficulty (levels 1-3). Groups of two students must implement additional functionality to achieve the same marks at each level.

Feature Group	**Marks**	**Individual Functionality Requirements** (marks for each feature)	**Group of Two Functionality Requirements** (marks for each feature)
Level 1	55	Movement (11) Eating (10) Splash screen (10) Game Over (10) Scoring (9) New Game (5)	Movement (8) Eating (7) Splash screen (7) Game Over (7) Scoring (6) New Game (3) Randomisation (8) Moving Food Items (9)
Level 2 features	20	Growth (5) Sound (6) Pause (5) High Score (4)	Growth (3) Sound (4) Pause (3) High Score (2) Acceleration / Deceleration (4) Multiple Sounds (4)
Level 3 features (Subset)	25	Persistent High Score (3) Save Game (5) Rats (6) Jump Attack (4) Tail Cut (7) Approved advanced feature(s) (max 5 each)	Persistent High Score (2) Save Game (3) Rats (4) Jump Attack (2) Tail Cut (5) Blinking Food Items (3) Terminal Game Field (6) Approved advanced feature(s) (max 5 each)

An example of the criteria needed to be met for a particular overall result is shown below. Note that other combinations are possible.

Equiv Grade	**Mark Range**	**Example Criteria (Other combinations of features possible)**
0	0	No submission, no features implemented or work with no academic merit submitted.
1	1-19	Only a couple of level-1 features implemented
2	20-44	Several level-1 features implemented
3	45-49	All level-1 implemented but some shortcomings
4	50-64	All level-1 features implemented and a partially successful attempt made at some higher level features
5	65-74	All level-1 and level-2 features implemented
6	75-84	All level-1 and level-2 features implemented and a partially successful attempt made at some level 3 features
7	85-100	All level-1 and level-2 features implemented and all/most level-3 features implemented

Submission Details

The due date for the project is 5pm Friday October 30. 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 or by email. You must electronically submit:

- 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 make multiple submissions, but only the last submission will be marked. Each submission must contain ALL of the required files. We are expecting each submission to consist of a single zip file containing at least 20 files (19 supplied source files plus a hex file) – more if you have added source files of your own. (Do not submit (or include in your .zip file) any other files. We do **not** want .o, .lst, .map, .cof etc files.)

You must **also submit** the form shown on the last page of this document – either on paper or by email (a PDF form will be provided which can be completed electronically). This will specify which features you have implemented as an aid to marking. If you submit the form and you have not specified that you have implemented a particular feature, we will not test for it. If you do not submit the form, we may miss some of your features.

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 on paper or by email. Students working in a group must also both print and complete the acknowledgement of originality declaration (which will be generated as part of the online submission process). For students working individually, the online declaration is sufficient – you should only submit the feature summary on paper. The paper submission deadline is the same as that for electronic submission but will not affect the calculation of any late penalty. 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. Groups should use the group ID *studentnum_studentnum*, e.g. 41234567_49876543, using the student numbers of the group members.

Assessment Process

Your project will be assessed during the revision period. 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

Submission after 5pm Friday October 30 will result in a penalty of 10% plus 10% per working day or part thereof. (The penalty is applied to the mark earned.) For example, a submission less than one working day late will be penalised 20% of the mark earned, less than two working days late 30% and so on. 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.

Notification of Results

Students will be notified of their results at the time of project marking (if they are present) or later via their "My Record" course web page.

The University of Queensland
School of Information Technology and Electrical Engineering
Semester 2, 2009

CSSE1000 / CSSE7035 Project
Feature Summary

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

You must complete this form and submit it to the CSSE1000/7035 submission box (on Level 1 of GPSouth) by the due date OR complete and submit electronically. Students working in a group of two should also submit the originality declaration generated during the submission process. No coversheet is necessary.

Feature (Group Features are shaded)	**✓ if attempted**	**Student comment (Anything you want the marker to consider or know?)**	**Marker comment**	**Mark (out of indiv/group)**	
Movement				11/8	
Eating				10/7	
Splash Screen				10/7	
Game Over				10/7	
Scoring				9/6	
New Game				5/3	
Game Randomisation				1/8	75 max
Moving Food Items				1/9	
Growth				5/3	
Sound				6/4	
Pause Game				5/3	
High Score				4/2	
Acceleration / Deceleration				1/4	
Multiple Sounds				1/4	
Persistent High Score				3/2	
Save Game				5/3	
Rats				6/4	
Jump Attack				4/2	
Tail Cut				7/5	25 max
Blinking Food Items				1/3	
Terminal Game Field				1/6	
Approved other advanced feature		(Attach your approval note to your submission).		_/_	
Approved other advanced feature		(Attach your approval note to your submission).		_/_	
			TOTAL: (out of 100)		