The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2017

CSSE2010 / CSSE7201

PROJECT

Due: **5pm Friday June 2, 2017**
Weighting: **20% (100 marks)**

Objective

As part of the assessment for this course, you are required to undertake a project that 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 version of Snake. (If you are unfamiliar with Snake, there are numerous websites that describe the game and have versions playable within your web-browser.) The AVR ATmega324A microcontroller runs the program and receives input from a number of sources (e.g. serial port input, push buttons, etc.) and outputs information to various devices (e.g. LED matrix).

The version of Snake provided to you will implement simple movement (the snake can only move to the right or up and wrap around the display) and food consumption. You can add features such as scoring, special types of food, increasing the speed of play, sound effects, use of the joystick, 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 over 2300 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 project.c, game.c and snake.c. You may need to look at the AVR C Library documentation to understand some of the functions used.

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 a form linked from the course Blackboard site, by **12 noon Thursday May 25, 2017** 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.

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 any 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/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.

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 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 that contain groups of related functions. The files provided are described below. The corresponding .h files (except for project.c) list the functions and constants that are intended to be accessible from other files. You may modify any of the provided files and add files if you wish. 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.

The 27 files you are provided with are:

- **project.c** – this is the main file that contains the event loop and examples of how time-based events (e.g. moving the snake) are implemented. You should read and understand this file.
- **game.h/game.c** – game.c contains the implementation of the basic operations in the game – initially setting up the game and attempting to move the snake forward. You should read this file and understand how and when the display is updated.
- **board.h** – board.h defines the size of the game board.
- **snake.h/snake.c** – maintains details of the snake (direction, length and positions of the head, tail and elements in between). You should read and understand these files.
- **food.h/food.c** – maintains details of the food items (number, positions).
- **position.h/position.c** – a type for recording an (x,y) position and associated functions
- **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) in a queue that can then be queried.
- **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 module 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 way 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 is available 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.

Initial Operation

The provided program responds to the following inputs:

- Rising edge on the button connected to pin B0 (moves snake right)
- Rising edge on the button connected to pin B2 (moves snake up)
- Serial input escape sequence corresponding to the right cursor key (moves snake right)
- Serial input escape sequence corresponding to the up cursor key (moves snake up)

The snake will move and grow when it eats food. The initially provided code will allow the snake to collide with itself but the display may not show this appropriately.

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

- Rising edge on the button connected to pin B1 (intended to move snake down);
- Rising edge on the button connected to pin B3 (intended to move snake left);
- Serial input escape sequence corresponding to the left cursor key (intended to move snake left).
- Serial input escape sequence corresponding to the down cursor key (intended to move snake down).
- Serial input characters 'p' and 'P' (intended to be the pause/unpause key)

Program Features

Marks will be awarded for the features described below. (The marks available for individuals and groups are shown on the feature summary page.) 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 that 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 the snake must move and the game must allow the snake direction to be changed to up or right. 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)**

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 the student number(s) of the student(s) whose project this is. You must also change the message output to the serial terminal to include your name(s). Do this by modifying the function `splash_screen()` in file *project.c*.

Snake Movement **(Level 1)**

The provided program will only allow the snake to move up or right. You must update function `play_game()` in *project.c* and `advance_snake_head()` in *snake.c* to allow the snake to also move left and down. You must also modify the program so that attempts to move the snake in a direction opposite to the current direction of travel are ignored. Modify the `set_snake_dirn()` function in file *snake.c*.

Collision Detection **(Level 1)**

Modify the program so that collisions of the snake with itself are detected and will finish the game. Modify the `advance_snake_head()` function in file *snake.c*.

Scoring **(Level 1)**

Add a scoring method to the program as follows:

- 1 is added to the score each time the snake moves
- 3 is added to the score each time a food item is eaten

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.) The score should remain displayed on game-over (until a new game commences when the score should be reset).

Game Pause **(Level 1)**

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 should be discarded whilst the game is paused except the 'n' or 'N' keys if "New Game" is implemented as described below.) The snake movement rate must be unaffected – e.g. if the pause happens 450ms before a snake movement is due, then the snake 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.

Snake Length Display **(Level 1)**

Use the seven segment display to show the length of the snake. This must work for all possible lengths (and must be able to be demonstrated for lengths of 10 or higher, i.e. two digits displayed). Single digit lengths must show a "blank" left digit. There must be no perceived flicker in the display. You should indicate on the feature summary form how the seven segment display is connected to your ATmega324A. If game pause is implemented, the length must remain displayed when the game is paused.

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

Add a feature so that if the 'n' or 'N' key is pressed, a new game is started (at any time – including game over or if the current game is in progress but not if leaderboard initials are being entered for the level 3 feature described below). Game play should behave as it does for the first game after power-on. (The splash screen must not be displayed.)

Game Randomisation **(Level 1 – group feature)**

The provided program always initially places the snake and food at the same locations. Modify the program so that it places the snake in a random position with a random direction. There must be at least one space ahead of the snake before it will hit the edge. The initial and subsequent food items should be placed at random locations. The random positions must vary from between board RESETs. HINT: Consider the or `random()` function – see how it is used within the splash screen code in *project.c*. You will need to seed the random number generator appropriately, e.g., with the timer value when the user clears the splash screen.

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

Keep track of and display (via the terminal) the high score across several games. (This implies your program can play the game multiple times without having been reset.) 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). The current high score must be displayed on each game-over. (If the high score is displayed continuously then it must be updated whilst the game is being played if the current score is the high score.)

Acceleration **(Level 2)**

Make the game speed up as the score gets higher. A suggested approach is that every time a food item is eaten the game speeds up *slightly*, i.e. the snake moves faster. (Do not speed-up play too quickly. An average player should be able to play for at least one minute, but the speed-up must be noticeable within 30 seconds. The average player should be able to grow the snake to its maximum length.)

Super-food **(Level 2)**

Every 15 seconds a "super-food" item should be displayed in a random position (in a different colour so that it can be identified). (The random position must never overlap the snake or an existing food item.) The item should be displayed for exactly 5 seconds and disappear at the end of this time if it is not eaten. If the item is eaten then 10 must be added to the score.

Rats **(Level 2)**

Make <u>one</u> of the food items wander around the game field at slow speed. The moving food item (rat) should change direction randomly and frequently to give the appearance that it is disoriented. The rat should move significantly slower than the snake itself (when the snake is moving at the default speed). A rat should never move into the snake or off the edge of the game field or into other food or rats, i.e. if a movement is obstructed by the snake and the edge of the field, then a different movement direction (or no movement direction) is chosen. If the rat is eaten then 5 must be added to the score and a new rat should be added to the game.

Moving Food Items **(Level 2 – group feature)**

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. Then implement one of:
<u>Partial requirement (half marks)</u>: Modify the program so that a randomly selected food item jumps to another (random) empty spot every 5 seconds of game play.
<u>Full requirement</u>: Make every food item (other than a super-food item or a rat) jump to another location after it has been present on the display for five seconds. Each item should blink rapidly

for the last second before it jumps. (Initially these jumps will be synchronized for all food items but as food items are eaten/replaced this will not be the case.)

Wrap-around Toggle Switch (Level 2 – group feature)

Switch 0 on the IO board must be able to be used to toggle whether the snake can wrap around or not. If the switch is 1, then wrap-around is permitted (as per the initially supplied game). If the switch is 0, then wrap-around is not permitted and the game will finish if the snake tries to move off the game field. The terminal display must show the current mode. Any change takes effect immediately. Your feature summary form must indicate which port/pin the switch is connected to.

EEPROM Storage of High Score Leader Board (Level 3)

Implement storage of a leader board (top 5 scores and associated names or initials) 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 or initials. The score and name/initials must be stored in EEPROM and must be displayed on the serial terminal on program startup and at each game-over. (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/initial entry must be resilient to arbitrary responses (i.e. invalid characters / keypresses / button presses etc. should not cause unexpected behaviour). Backspaces must be supported. The minimum accepted length of the name/initial entry is 2 characters.

Sound Effects (Level 3)

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

- snake moving
- snake eating food
- snake changing direction
- 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.

Joystick Support (Level 3)

Add support to use the joystick to change the direction of the snake in the same way that the serial terminal cursor keys work:

- joystick left – moves the snake left (if possible)
- joystick right – moves the snake right (if possible)
- joystick up – moves the snake up (if possible)
- joystick down – moves the snake down (if possible)

The joystick should be sampled at the time of movement of the snake. Joystick movements between snake movement are ignored – the snake movement will only depend on the position of the joystick at the time of snake movement. The joystick position will override any button presses or cursor key presses unless the joystick is "near-centre" (i.e. unused) at the time of snake movement.

EEPROM Storage of Game (Level 3)

Implement storage of the complete game state in EEPROM. If the "s" or "S" key is sent from the serial terminal then the whole game state should be saved. If the "o" or "O" key (for "Open") is later sent (possibly many power cycles later), then the saved game should be retrieved and play should continue on in that game. Note the comment about "signature" requirements above (EEPROM High Score Leader Board) – it should not be possible to "open" a saved game if none was saved from your game.

Game Display on Terminal Screen (Level 3)

Display a copy of the LED matrix display on the serial terminal using block graphics 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.

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

Feel free to implement other advanced features. The number of marks that may be awarded will vary depending on the judged level of difficulty or creativity involved – up to a maximum of 7 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 (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). 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 20 marks for advanced features even if you successfully add all the suggested advanced features. If an individual implements a group feature then this counts as 1 mark towards level 3.

Submission Details

The due date for the project is **5pm Friday 2 June 2017**. 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.

For those students working in a pair, only one student should submit the files.

Assessment Process

Your project will be assessed during the revision period (from Tuesday 6 June 2017). 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.

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 5pm Saturday 3 June, 2017) 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 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 Blackboard's "My Grades".

The University of Queensland - School of Information Technology and Electrical Engineering
Semester 1, 2017 – CSSE2010 / CSSE7201 Project – Feature Summary

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

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	
Notes for Marker e.g. compile/link options								

Feature (For Groups)	**✓ if attempted**	**Comment (Anything you want the marker to consider or know?)**	**Marks (indiv/grp)**	
Splash screen			4/3	
Snake Movement			8/5	
Collision Detect			6/4	
Scoring			10/7	
Game Pause			10/7	
Length Display			15/10	
New Game			(1)/5	
Randomisation			(1)/7	
High Score			(1)/5	/53
Acceleration			9/6	
Super-food			9/6	
Rats			9/6	
Moving Food			(1)/5	
Wrapping Toggle			(1)/4	/27
EEPROM Leaders			7/5	
Sound Effects			7/5	
Joystick			7/5	
EEPROM game			7/5	
Terminal Display			7/5	
Other Advanced			max 7/7	/20 max

(Penalties apply for compile warnings, code corrections, etc.) **Total:** (out of 100, max 100)