The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2018

CSSE2010 / CSSE7201

PROJECT

with clarifications as at 27 May 2018 (shown in red)

Due: **5pm Friday June 1, 2018**
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 version of Frogger – see www.happyhopper.org for a web-based version of Frogger if you are unfamiliar with the game. 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 Frogger provided to you will implement basic functionality – the frog can only move in one direction (forward / up) and the game ends immediately when a frog "dies". You can add features such as scoring, other directions of movement, 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 over 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 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. 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 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 allow the frog to move in other directions.
- **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.

Initial Operation

The provided program responds to the following inputs:

- Rising edge on the button connected to pin B2
- Serial input character 'u' or 'U'
- Serial input escape sequence corresponding to the cursor-up key

All of these move the frog forward (up) by one row.

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

- Rising edge on buttons connected to B3, B1 and B0 (intended to be left move, down move, and right move respectively).
- Serial input characters 'd', 'D', 'l' (lower-case L), 'L', 'r', 'R' (intended to be move down, move down, move left, move left, move right and move right respectively)
- Serial input escape sequences corresponding to the cursor left, cursor down and cursor right keys.
- 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 moving lanes of traffic and logs and allow the frog to be moved forward. 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*.

Scoring (Level 1 – 10 marks)

Add a scoring method to the program as follows:

- 1 is added to the score each time the frog moves forward without dying. (You can add 1 for every move forward – even if the frog has moved backwards – OR you can ignore such moves if you wish and only add to the score if the frog moves beyond where it has previously moved.)
- 10 is added to the score each time the frog successfully reaches the other side (i.e. the riverbank). (The +1 from the above point can also be added in this case if you wish but does not have to be.)

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).

Move Frog Left, Right, Down **(Level 1 – 13 marks)**

The provided program will only move the frog up. You must complete the `move_frog_to_left()`, `move_frog_to_right()` and `move_frog_backward()` functions in file *game.c*. The frog position should be updated and redrawn. It should be checked whether the frog will be alive or not. The frog will die if it jumps into a vehicle or jumps into the river. If an attempt is made to jump off the game field then the frog dies in its current position (i.e. the position just before the jump).

Multiple Lives **(Level 1 – 13 marks)**

Allow the player to have multiple "lives" (chances), i.e. 3 frogs must die before the game is over. Indicate the number of lives remaining by using individual LEDs on the IO board (e.g. LD0 to LD2). If there is one life remaining (last frog) then there should be one LED illuminated. Be sure to indicate which AVR pins the LEDs should be connected to. There must be sufficient LED connections specified to support the maximum number of lives (see Game Levels below). When a frog dies, it should be clear to the player that the frog has died (by looking at the LED matrix display) – e.g. the dead frog could be shown in the position it died for a short period (e.g. one second) before the game restarts with the next life, or the dead frog could be shown until a button is pushed to start the next life. In all cases, the dead frog must be removed from the display when the game continues, including if the frog died on the riverbank.

Scrolling Speeds **(Level 1 – 13 marks)**

The provided program scrolls traffic lanes and logs one pixel per second. Modify the program so that each of the five lanes/channels scrolls at slightly different speeds. Speeds should initially (i.e. on level one) range between one pixel per 750ms to one pixel per 1.3s. If multiple levels are implemented (see below) then speeds will increase on each level.

Game Pause **(Level 2 – 8 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 should be discarded whilst the game is paused – i.e. will not affect the movement of the frog after the game resumes.) The vehicle/log movement rate must be unaffected – e.g. if the pause happens 450ms before a vehicle movement is due, then the vehicle 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.

Game Levels **(Level 2 – 8 marks)**

The provided game finishes if all four "holes" on the far riverbank are filled with frogs. Modify the program so that the game will continue if this happens. The current level (1 to …) must be shown on the terminal display. When a level is finished (the fourth frog reaches home), the current "image" on the LED matrix should shift off to the left of the display – over about one second – and then the next level should appear and play start. The speed of play (i.e. the rate at which the vehicles/logs scroll) increases on each succeeding level. (Do not speed up play too quickly. An average player should be able to play for at least 120 seconds, but the speed-up

must be noticeable on each level change.) Different game levels should use different vehicle/log/riverbank patterns and colours. At least three different level patterns should be evident but it is expected that more than three levels should be supported (i.e. the game should not stop after three levels but patterns can repeat from the fourth level if desired). "Different colours" doesn't mean new colours – it can be reuse of previously defined colours in different ways. If the "multiple lives" feature has been implemented (see above) then successful completion of a level should restore a life (up to some maximum, e.g. 5). Please specify the maximum number of lives on the feature summary form.

Time Limit **(Level 2 – 8 marks)**

Implement a time limit for each frog (e.g. 15 seconds, though the time may depend on the difficulty of play in your game). If the frog does not make it to the other side before the time runs out, then the life is lost. Use the seven-segment display to show the number of seconds remaining. (The time limit must be greater than 10 seconds so that the use of two digits is demonstrated. For single digit numbers there should be no leading 0 – show a blank. When the time falls below one second, show tenths of seconds with a decimal point, i.e. 0.9, 0.8 etc.) The remaining time must continue to be displayed while the game is paused (if pause is implemented).

Auto-repeat **(Level 2 – 8 marks)**

Add auto-repeat support to the push buttons which move the frog – i.e., if they are held down then, after a short delay, they should act as if they are being repeatedly pressed. The speed of auto-repeat must be such that the game remains playable by the average user. You must appropriately handle multiple buttons being held down. "Appropriately handle" means the game shouldn't crash or hang – a sensible choice should be made. For example, this could mean that conflicting movement directions (e.g. up and down) could be ignored, or only the latest button is accepted, or only the first button or …. Perpendicular moves could also be ignored, or could result in diagonal movement, or the latest button could be accepted or the first button or …

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. 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 / backspaces / button presses etc. should not cause unexpected behaviour). The only characters supported must be letters, spaces and the left cursor key, with name entry terminated by the return key. The left cursor key must result in the cursor being moved one position to the left (if not already in the leftmost position) so that that character can be overwritten. If desired (not required), the backspace character may also be accepted and processed as expected. Other key presses (including other cursor keys) must be ignored. Names up to 10 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 three events. (At least one sequence of tones must be present for full marks.) For example, choose three of:

- frog moving
- frog successfully reaching the far side
- frog dying

- game start-up
- start/end of level
- 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). Switch 6 on the IOboard must control the volume when sound is being output: 0 on this switch must result in quieter output than 1. You must specify which AVR pin these switches are 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 (and resume when the game is unpaused if sound was being produced at the time of pausing).

Joystick **(Level 3 – 5 marks)**

Add support to use the joystick to move the frog up, left, right and down (corresponding to joystick U, L, R, D). This must include support for diagonal movement and auto-repeat support – 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. (Diagonal movement should be implemented as a direct move to that diagonal position – not as a horizontal move followed by a vertical move or vice versa – i.e. the frog can't die "part way" through a diagonal move.) 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.

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

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. (Block graphics means the whole pixel is a solid colour. Other characters can be used in addition to block graphics (e.g. for the frog) but there must be block graphics used on the display using at least two different colours.) 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. Note that the markers will be using default PuTTY terminal settings and your implementation should work with those.

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 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 15 marks for advanced features even if you successfully add all the suggested advanced features.

Submission Details

The due date for the project is **5pm Friday 1 June 2018**. 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); (You may add your own .c and .h files if you wish.)
- 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 4 June 2018). 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. Your code must not use #pragmas – each #pragma will result in a 1 mark penalty.

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 2 June, 2018) 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, 2018 – 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	
Scoring			/10	
Moving L/R/D			/13	
Multiple Lives			/13	
Scrolling Speeds			/13	/53
Game Pause			/8	
Game Levels			/8	
Time Limit			/8	
Auto-repeat			/8	/32
EEPROM Leaders			/5	
Sound Effects			/5	
Joystick			/5	
Terminal Display			/5	
Other Advanced			/5 max	/15 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)	