CAB202 Assignment 2 Report

Ollie Pye - n9703977

26 October 2018

Contents

1 Introduction

Serial communications was tested using Putty on Linux Ubuntu.

All functions used from the lecture notes/examples have the following comment preceding them: *// function from lecture notes*

I was unsure about the what to do when zombies were carried off the screen. As such I made them disappear and no score is added.

The specs say to have the side LEDs (0 and 1) to flash at 4Hz 3 seconds before the zombies spawn and while the zombies are on screen. When all of the zombies are off the screen they then respawn after 3 seconds. As such, the LEDs flash continuously unless the game is paused or the game is over or on the start screen.

The player speed is proportional to the block speed. When block speed is 0 the player speed is very slow but is present.

1.1 Bugs

Serial communication does work, however when using the computer keyboard for controls I found that I had to hold the key for 1-2 seconds for it to work on the teensy.

Since the zombies do not move along the blocks (not implemented), to test the food collision I deployed a food on the left of the starting block. When the zombies spawn one of them collides with this food. While this is not ideal it does test the functionality.

Player is able to jump/move through blocks.

I have not implemented my seeding for srand correctly randomness is not apparent unless you recompile the game.

2 Implementation Summary

Item Number	Item Description	Implementation Level
1	Intro	Fully Implemented
2	Pause Game	Fully Implemented
3	Player size	Fully Implemented
4	Block size	Fully Implemented
5	Random Blocks	Partially Implemented
6	Player movement	Mostly Implemented
7	Treasure	Fully Implemented
8	Basic game mechanics	Fully Implemented
9	Player velocity	Mostly Implemented
10	Player jumping	Fully Implemented
11	Block movement	Fully Implemented
12	Player inventory	Fully Implemented
13	Zombies spawn	Fully Implemented
14	Zombie movement	Partially Implemented
15	Zombie cycle	Fully Implemented
16	Pause screen advanced	Fully Implemented
17	ADC for block speed	Fully Implemented
18	Switch debouncing	Fully Implemented
19	LED warning	Fully Implemented
20	Direct control of LCD	Fully Implemented
21	Multiple timers	Fully Implemented
22	Program (flash) memory	Fully Implemented
23	PWM controlled visual effects	Fully Implemented
24	Pixel level collision	Not Implemented
25	Serial communication events	Fully Implemented
26	Serial communication games control	Fully Implemented

3 Specialised Functionality

3.1 Debouncing

Debouncing was implemented to reduce the affects of bouncing from the switches. This is important because the game performance is significantly increased when a precise recognition of the button is implemented.

Non-blocking debouncing was implemented. The button state is repeatedly sampled to determine its state. This is not a perfect solution but it works reasonably well.

3.2 Direct control of LCD write

Writing directly to the LCD is useful if you only want to change a small section of the screen, while leaving the rest unchanged. For my animation however, I clear the screen first and then run the animation.

Direct LCD write was used for the game over animation (a cross). First the screen was cleared by writing nothing (0) to the screen. The cross bitmap was then converted into vertical slices. This is because the y scale of the screen is split into banks of 8 bits. These vertical slices are then iterated through a draw function so they are directly drawn on the screen.

3.3 Timers

Multiple timers were used throughout the program for various functions. One timer was used to keep track of the game time, while another was used to accurately release the zombies. Finally, a third timer was used for debouncing. Multiple timers were needed because various functionality needs to work at various times during the game. For example, when the game is paused the game timer stops, but a timer is still needed so that debouncing will still work and as such you can still resume the game.

Each of the timers were initialised. An Interrupt Services Routine (ISR) was then set to count how many times the timer has overflowed. This overflow value was then used to calculate the time that had passed. Because of rounding, there is some uncertainty.

3.4 Program memory

Program memory was implemented by storing constants using PROGMEM. These constants were then called when they needed to be used. Program memory is used to store the application. Since the constants are not changing, they can also be stored in the program memory. This means that the static memory is not being wasted on storing a constant value.

4 Testing

Test of Specific Functionality	**Test Setup**	**Expected Result**	**Actual Result**
Program displays student number and name. Game starts when SW2 is pressed.	Load game and observe student number and name. Press SW2 and see if game begins.	Student name and number appear. They disappear and game starts when SW2 is pressed.	As expected
Pause screen is displayed when joystick is pressed. Lives, score and time is displayed.	Play the game and press the joystick at various stages/multiple times. Check score, time and lives update appropriately and that all sprites stop.	Pressing the joystick centre pauses the game. The pause screen clears the screen and displays the score, time and lives remaining. The values update as the game progresses. Pressing the joystick again resumes the game.	As expected.
Player respawns on a starting block, and the player is at least 3x3 pixels	Run game and die multiple ways. Also reset the game.	When the player resets from either starting the game, after death or restarting the game, the player always lands on a safe block in the top row. The player is always larger than 3x3.	As expected
Block Setup - all blocks are at 10x2 pixels. They are always player height + 2 vertically spaced and at least 7 safe and 2 forbidden blocks. The forbidden and safe blocks are easily distinguished.	Play the game multiple times so that various random block setups are observed. Ensure that the blocks are easily distinguishable, of the correct spacing and that the correct number are present.	All blocks are 10x2 with spacing of player height +2 vertically. There are at a minimum 7 safe and 2 forbidden blocks at all times. Safe and forbidden blocks are easily distinguishable.	As expected

Block Randomness - blocks do not overlap and appear in randomly selected rows and columns.	Run game multiple times to allow blocks to be set up in various ways. Check the blocks do not overlap and that they have a randomness to their layout each time.	The blocks do not overlap and appear in random rows and columns. This is true for any state in the game and when the game resets. I have not implemented srand properly and as such to see any randomness the game needs to be recompiled.	As expected
Basic Player movement	Advanced player movement has been implemented.	THE PLAYER AND BLOCK SPRITES DO OVERLAP AT TIMES	Advanced player movement has been implemented.
The treasure is correct size (at least 3x3), does not overlap with any blocks, spawns on the bottom half of the screen moving back and forth along the width of the screen and stops and starts when SW3 is pressed. It disappears when the player collides with it, adds 2 more lives and does not reappear until the game has restarted.	Start the game and observe the size and location of the treasure. Observe it move across the screen and pause it at various stages (ie. near the edge of screen, in the middle of screen, moving in either direction). Ensure it starts and stops appropriately. Collide player with the treasure, ensure 2 lives are added, the treasure disappears. The player respawns. Play until player loses all lives. Ensure treasure does not reappear until game restarts after the loss of all lives.	The treasure is always at least 3x3, does not collide with any blocks and moves backwards and forwards along the bottom half of the screen. When SW3 is pressed the treasure stops/starts. 2 lives are added and the treasure disappears when the player collides with it. The player respawns. The treasure reappears when the game resets (though you have to Press SW3 for it to start moving).	As expected

Basic Game Mechanics - Player starts with 10 lives, scores a point for each safe block landing, loses a life if it moves off screen or lands on forbidden block and respawns on a starting block. When all lives are lost end game screen displays message, total score and play time. SW2 clears screen and student number is displayed, SW3 restarts the game where score, lives, time and player position all reset.	Start the game. Check initial time, score and life values. Play the game and die by means of landing on forbidden blocks, moving off screen in all directions, and check that lives update accordingly. Ensure time updates throughout game. Land on safe blocks and check score increments appropriately. Lose all lives and observe game over screen. Ensure values displayed are consistent with the values observed during testing. SW3 restarts the game. Again, check initial values and player position are correct. Run again to ensure that SW2 displays only the student number.	Player starts with 0 score and 10 lives. Player dies in appropriate scenarios (landing of forbidden blocks and moving off screen). The lives are reduced each time. The time updates throughout the game. The score increments when the player lands on a safe block. The game over screen is displayed when the player runs out of lives. The correct lives, score and time are shown. SW3 restarts the game and resets the lives, score and time. SW2 displays the student number.	As expected
Block Movement - All blocks except starting block have a horizontal speed and that each row of blocks move in the opposite direction to the one above/below.	Start game. Run various scenarios (death by forbidden blocks/moving off screen, collision with treasure) and observe block movement is maintained in correct direction. Restart game after game over screen to ensure the blocks maintain motion.	The blocks in each row move in the opposite direction to the blocks in the row above/below. This is maintained regardless of game situation. When the game is reset or the player respawns, the motion is maintained.	As expected

Advanced Player Movement - Pressing the joystick left and right gives the player lateral velocity. This velocity is greater than that of the block so the player moves along the block. When a player lands on a block it moves with the block. The player accelerates downwards when not supported by a block. Horizontal motion is maintained (parabolic motion). If a player collides with the side of a block it will fall down. Collision with forbidden block results in death.	Start the game. Land on various blocks and observe their motion. While on different blocks, press the joystick and observe the motion of the player. Collide with a forbidden block. The player accelerates downwards when not supported by a block and maintains a parabolic path if the player has a horizontal velocity.	The player gains the blocks horizontal velocity upon landing. Pressing the joystick gives the player additional velocity to move along the blocks. When not supported by the block the player accelerates downwards and maintains any horizontal velocity. Colliding with a forbidden block results in death. Collision with the end of a block and overlapping with block sprite has not been implemented. As such, the player will not fall when colliding with the end of a block.	As expected
Player Jump - Pressing UP causes player to move up and gravity acceleration occurs. Horizontal motion remains. The joystick has no affect until player lands. The players velocity changes to that of the block it lands on after jumping. The player dies when it jumps off screen and is able to jump through gaps between and land on blocks on the row above.	Start game. Press UP at various stages of the game. Press UP when the player has a horizontal velocity. When performing the jump, use the joystick. Jump onto various blocks at different stages of the game, and jump through gaps to land on the row above/below. Attempt to jump off the screen in any direction.	The player moves up and is immediately affected by gravity when the UP button is pressed. Horizontal velocity is maintained. It is possible to jump on rows above/below. The player loses a life when it jumps off the screen.	As expected

Food inventory - The player starts with 5 food and they are no larger than the player. Food can be deployed when player is supported by the block, the food is supported by the block and overlaps the player. Food inventory decreases by one and food may overlap each other.	Start the game. Land on various blocks and press the down key to deploy the food. Observe the movement of the food on the block. Deploy multiple food to see if they overlap. Pause to ensure the number of food in the inventory changes accordingly.	The food is deployed and the block supports the food. Food does not deploy when player is not supported by a block. The food overlaps the player and other food. The food inventory count decrease each time a food is released. When the food is deployed it is carried by the block that is supporting it.	As expected

Zombies - 5 zombies appear after 3 seconds at the top of the screen, falling straight down. Zombies are supported by safe blocks they land on. If a zombie collides with a food they both disappear, zombies decrease by 1 while food inventory increases by 1 and score increases by 10. If the zombie falls off the screen the player receives no points. Zombies prowl left and right on the block they are supported by. Zombies may overlap and if they are carried off screen by the block reappear with the block on the other side of the screen. Player dies when it collides with a zombie and 3 seconds after the last zombie disappears the zombies respawn.	Start the game. Wait approximately 3 seconds and observe the behaviour of the zombies. Collide a food with a zombie and observe the score values. Observe the zombies land on a safe block. Check score as zombie falls off screen. Time how long it takes for zombies to respawn after all have left the screen or have eaten food.	The zombies appear after 3 seconds both when game starts and when there are 0 zombies left. When a zombie collides with a food they both disappear. The zombie count decreases while the food inventory increases, and the score increases by 10. The score does not change when a zombie moves/falls off the screen. The zombie does not prowl left or right as this has not been implemented, instead the zombie obtains the block speed.	As expected

Updated pause screen - pause screen shows number of zombies on screen and number of food inventory	Start game and pause instantly. Observe the values. Play the game and test various functions (deploying food and death). Pause throughout these events and observe the values displayed.	Initially zombies is 0 and food is 5. As the zombies are deployed and appear on screen the zombie value changes to 5. As zombies disappear the value decreases and as food is used the food inventory decreases. If a food is eaten by the zombie then the food inventory increases while the zombies still decrease.	As expected
ADC - Using the Potentiometer to change the block speed	Start the game. Change the potentiometer and observe the speed of the blocks. Set the potentiometer in its maximum and minimum position and observe the block speed.	As the potentiometer is changed, all moving blocks change speed. When the maximum position is set the blocks are still visible and when the minimum is set the block speed is 0. The player speed also changes relative to the potentiometer value.	As expected
Both LED0 and 1 flash at approximately 4Hz when zombies spawn.	Start the game. Observe the LEDs and compare their flashing rate to a stopwatch. Count the number of flashes. Run through various scenarios like death or pausing and observe the lights behaviour.	When the game starts the lights will begin to flash. They will flash approximately 4 times per second. When the player dies or the game is paused the flashing stops. The flashing restarts when the game resumes.	As expected

PWM used to control the backlight when the player loses a life (except when lives is 0).	Start the game. Cause the player to die via multiple methods (falling off screen, bad blocks, zombies), and observe the screen. Let number of lives reach 0 multiple times and via various methods and observe the screen.	When the player dies and the lives left is not 0, the back light and contrast dims until nothing is visible. The screen then gradually goes back to normal contrast and the back lights go to their normal brightness. The player then respawns on the starting block. This does not occur when lives reaches 0, instead the direct LCD animation is played.	As expected

Serial Communications - Information is sent to the computer after various events occur during the game.	Set up serial communication. Start the game and run through various scenarios (death, feeding zombie, treasure collision, pause). Observe the output on the computer terminal (tested on putty).	When any event occurs the name of the event is displayed. When the game starts or the player respawns the player x and y location are listed. When the player dies the reason for death, lives after death, score and game time are listed. When the zombies appear, the number of zombies, game time, lives and score are listed. When a zombie collides with food, the number of zombies on screen after collision, the number of food and the game time are listed. If the player collides with the treasure the score, lives, game time and player respawn location are listed. When the game is paused the lives, score, game time, number of zombies on screen and food inventory are listed. Finally, when the game is over the lives, score, game time and total zombies fed is listed.	As expected

USB Serial communication - the computer keyboard is used to control the game.	Set up serial communication. Start the game. Use the keyboard inputs to control the game. Test all scenarios (for example, jumping, moving left or right, pause both the game and the treasure).	The game can be started by pressing ‘s’. The keys ‘a’, ‘d’ and ‘w’ cause the player to move left, right and jump respectively. ‘t’ stops and starts the treasure. Food is deployed during play using the ‘s’ key. ‘p’ pauses the game. When the game over screen is displayed ‘r’ restarts the game while ‘q’ end the game and displays the student number.	As expected