CAB202 Assignment 2

Due Date: 23:59:59 (AEST) Friday 26th October 2018

Marks: 40 (40% of your final mark)

For this assignment, you will be writing a TeensyPewPew program that emulates a player moving through a series of static and moving blocks. The aim of the player is to move/jump between blocks while gaining points and lives. This assignment builds on Assignment 1 and requires that you demonstrate knowledge of the Teensy microcontroller.

Please join the [Assignment 2 Slack channel](https://cab202-18s2.slack.com/messages/CCUD8SKHQ/) to ask your questions.

Assignment Specifications

This assignment is divided into different sections, however you may attempt the sections in any order you see fit. You may attempt other sections even if you have not fully completed the Basic Functionality.

You will submit documentation as per the instructions given below and you will be marked on a combination of your documentation and your program. You will need to do both well to get the full marks. If you submit code without the specified documentation you will be marked down, conversely, if you submit documentation for aspects you have not implemented or documentation that does not accurately reflect your program, you will be marked down.

If your program crashes, you will be marked for the sections your marker was able to test before the program crashed. Please test your code regularly to minimise your chance of this happening. You can make multiple submissions to AMS and you are encouraged to do so.

Basic Functionality (12 Marks)

1. Your program begins with an intro screen showing your name and student number. When SW2 is pressed, the game begins.
2. When joystick centre is pressed once, the game pauses (all sprites stop moving and are unable to move by using the normal controls), the game screen is cleared, and game information is displayed. The game is resumed when joystick centre is pressed again. The game information displayed is:
   1. Lives remaining
   2. Current score
   3. Game time in mm:ss format
3. The player initially appears on a 'starting block' in the top row. The player’s sprite is at least 3 pixels high and 3 pixels wide.
4. All blocks are:
   1. At least 2 pixels high,
   2. At least 10 pixels wide and are clearly distinguished from each other (through spacing between sprites or other visual design),
   3. Always at least **player sprite height + 2** pixels vertically separated from other blocks.
   4. There are at least 7 safe blocks on the screen at one time.
   5. There are at least 2 forbidden blocks on the screen at one time.
   6. Safe and forbidden blocks are easily distinguished from each other.
5. Blocks have no consistent observable pattern and each block:
   1. Appears in a randomly selected row,
   2. Appears in a randomly selected column,
   3. Does not overlap other blocks.
6. Pressing the joystick LEFT or RIGHT has the following effect:
   1. Pressing the joystick left or joystick right moves the character one step left or right, respectively,
   2. If the player is not supported by a block, it will move vertically toward the bottom of the screen. *For Basic Functionality, realistic physics such as acceleration and horizontal momentum, are not required.*
   3. The joystick has no effect on the player's motion when the player is not supported by a block.
   4. If the falling player collides with a safe block from above, then the player lands on the block. The player will then immediately stop moving.
   5. Under no circumstances may the visible pixels of the player sprite overlap with the visible pixels of a block.
7. There is a treasure sprite that:
   1. Is no larger than the player's sprite and has a clearly distinct image.
   2. Does not overlap any of the blocks.
   3. Spawns in the bottom half of the screen and moves back and forward, changing horizontal direction when it reaches the edges of the screen.
   4. Stops moving when SW3 (right button) is pressed and starts moving again if SW3 (right button) is pressed again.
   5. Disappears when the player collides with it and gives the player **2** more lives and returns the player to the 'starting block'.
   6. Does not reappear again unless the game is restarted.
8. The following game mechanics are present:
   1. The player starts with **10** lives.
   2. A point is scored every time the player lands on a safe block.
   3. The player dies if any part of the player sprite moves off the screen in any direction or manner, or if it collides with a forbidden block.
   4. On death, the player respawns on the 'starting block'.
   5. When the player loses all their lives, the game over screen is displayed which displays a game over message, total score, and game play time.
   6. The **game over screen** allows the player to restart by pressing SW3 (right button) and score, lives, time, and player position all reset or end the game by pressing SW2 (left button) which clears everything and just displays student number on the screen.

Advanced Functionality (10 Marks)

1. In addition to the functionality of points 4 and 5 above, the blocks move.
   1. All blocks move at a constant horizontal speed.
   2. Each row of blocks must move in the opposite direction to the row above it.
2. The functionality defined in point 6 above is **redefined** as follows:
   1. Pressing the joystick either LEFT or RIGHT while the player is supported by a block sets the player in continuous horizontal motion at a constant speed (relative to the block) in the appropriate direction. When in horizontal motion, the players horizontal velocity must be greater than that of the block, so that it is possible for the player to make progress against the direction of the block.
   2. If the player is moving horizontally on a supporting block, and the joystick is pressed in the opposite direction, then the player stops moving relative to the supporting block, and is carried by the block.
   3. If the player is moving horizontally on a supporting block, and the joystick is pressed in the same direction as current movement, then the player continues to move at the same speed in the current direction.
   4. If the player is moving horizontally on a supporting block, and the joystick is not pressed, then the player continues to move at the same speed in the current direction.
   5. If the player is not supported by a block, then it will commence to accelerate downwards. If the player is moving horizontally before leaving the support of a block, then the player must continue to move horizontally at the same speed while accelerating downwards, so that a parabolic flight path will be observed.
   6. If the player is moving without support of a block (ie, flying, jumping, or falling) and the player lands on a safe block, it will then immediately begin to move horizontally in the same speed and direction as the block, and all vertical motion will cease. That is, the player will ride along the block, and player movement block.
   7. If the player is moving without support of a block, and its Sprite collides with the end of a safe block (from the side), then its horizontal motion becomes zero, and it continues to accelerate downward.
   8. All other collisions between the player and safe blocks are treated in a manner consistent with the combined motion.
   9. Any collison with an unsafe block results in death.
3. Pressing the joystick UP while the player is supported by a block causes the player to jump.
   1. After UP is pressed, the player should commence to move upwards. Any horizontal motion should continue, and the acceleration provision of 10(e) will take effect.
   2. Once UP is pressed, the joystick has no effect until the player lands on a block or dies.
   3. Immediately upon landing on a block, the player's velocity changes to match that of the block so it is carried along.
   4. If the player jumps off screen, the player dies.
   5. The initial vertical velocity should be sufficient to allow the player to jump through gaps between blocks and land on blocks on the row above.
4. Player has an inventory which stores Food. Food occurs such that:
   1. At the start of the game, the player has five (5) Food in their inventory.
   2. The Food sprite's area (i.e. sprite width \* height) is no larger than the player sprites's area.
   3. When the player is supported by a block and joystick DOWN is pressed, Food appears. The Food must be supported by the block, and overlap the player sprite. The number of Food in the inventory decreases by one.
   4. The total Food count (inventory + screen) must always equal 5.
   5. Food dropped on a block is carried by the block.
   6. Food may overlap.
5. Five (5) Zombies appear from the top of the screen, and fall straight down without overlapping each other, after the program has been running for approximately 3 seconds. The Zombies behave as follows:
   1. Zombies stop falling when they land on any block.
   2. If a Zombie falls all the way to the bottom of the screen, it passes from view and does not reappear immediately. The player doesn't get any points for this.
   3. After landing on a block, Zombies prowl left and right along the blocks at fixed constant speed *relative to the blocks*. Zombies can cross over narrow gaps (just how narrow depends on the span of the Zombies' "feet"). Whenever a Zombie reaches a gap that it cannot cross, it reverses and moves in the opposite direction.
   4. Zombies may overlap each other if they are moving in opposite directions on the same span of blocks.
   5. If a Zombie is on a block that leaves the screen, the Zombie is carried off the screen; it continues to move along the blocks as if nothing untoward had occurred.
   6. When a Zombie collides with Food:
      1. the Zombie disappears,
      2. the player gains 10 points,
      3. the Food disappears, and
      4. the Food inventory increases by 1.
   7. If a Zombie collides simultaneously with multiple Food, then the Zombie consumes one Food, leaving the others unaffected.
   8. Zombies do not disappear when the player collides with them, but instead the player loses a life and respawns.
   9. Three seconds after the last of the five Zombies disappears, either by feeding or falling out of view, the full complement of five Zombies respawn in the manner noted above.
6. In addition to the items listed under point 2, the pause screen shows:
   1. Number of Zombies on screen.
   2. Number of Food in inventory.

Specialised Teensy Functionality (18 Marks)

1. ADC (your program uses one potentiometer) to control the movement of blocks, such at:
   1. All rows of blocks move at the same speed which is proportional to the potentiometer value.
   2. Block movement speed should range from 0 to a speed where the blocks are still clearly visible.
2. All switches that are used in the program are debounced efficiently, using the algorithm developed in AMS Topic 9, Question 3.
3. Both LED0 and LED1 flash at approximately 4Hz (synchronously or asynchronously) before the zombies spawn and continue to flash until all the zombies have landed or fallen off screen.
4. Direct control of the LCD using lcd\_write (without using show screen or any equivalent function) is used to animate a death animation when the player dies the final time (i.e. when lives reach 0).
5. Multiple timers and interrupts are used to implement elapsed game time and zombie spawning and animation.
6. Program (flash) memory is used in an appropriate manner (other than the ASCII font already supplied in the library code) to store large data objects which would normally occupy static memory.
7. PWM is used to control the backlight when the player loses a life (but not the last time, when lives reach 0) and respawns and contrast changes as well so that the following effects are observable:
   1. When the player dies, the backlight fades off and the screen fades out so nothing is visible,
   2. Then the screen gradually goes back to normal contrast, the backlight fades on and the player respawns at the top row of blocks.
8. Pixel-level collision is testable and sprite images are chosen for their ability to clearly demonstrate this.
   1. The method of testing pixel level collision versus what would occur if box collision was implemented is clearly documented.
   2. If the player's sprite is humanoid, the player could collide with the block so that its hand lands on top of the platform and it gets pulled along hanging by its hand.
9. Serial communication is used to send information to a computer. The events and information are:
   1. Game starts - name of event, player x position, player y position.
   2. Player dies - name of event, reason for death, lives after death, score, game time.
   3. Player respawns - name of event, player x, player y.
   4. Zombies appear - name of event, number of zombies, game time, player lives, player score.
   5. Zombies collide with Food - name of event, number of zombies on screen after collision, number of Food in inventory after collision, game time.
   6. Player collides with treasure - name of event, score, lives, game time, position of player after returning to the top row,
   7. Pause button pressed - name of event, player lives, player score, game time, number of zombies on screen, number of Food in inventory.
   8. Game over - name of event, player lives, player score, game time, total number of zombies fed.
10. Your program uses USB serial communication to control the Teensy Game. All requirements specifying joystick and buttons now also apply to the corresponding keyboard keys specified below:
    1. 's' starts the game from the intro screen,
    2. 'a' moves the character left,
    3. 'd' moves the character right,
    4. 'w' lets the character jump,
    5. 't' stops and starts the treasure movement,
    6. 's' drops the Food where the player stands,
    7. 'p' pauses the game and shows the game information,
    8. 'r' restarts the game after game over,
    9. 'q' takes the game to the student number screen after game over.