CAB202 Assignment 2 Documentation

Jarod Lam
n9625607

October 2018

Contents

1 Assignment implementation summary

Table 1: Assignment 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	Fully implemented
6	Player movement	Fully implemented
7	Treasure	Fully implemented
8	Basic game mechanics	Fully implemented
9	Block movement	Fully implemented
10	Player velocity	Partially implemented
11	Player jumping	Fully implemented
12	Player inventory	Fully implemented
13	Zombies	Partially implemented
14	Pause screen advanced	Fully implemented
15	ADC for block speed	Fully implemented
16	Switch debouncing	Fully implemented
17	LED warning	Fully implemented
18	Direct control of LCD	Fully implemented
19	Multiple timers	Fully implemented
20	Program (flash) memory	Fully implemented
21	PWM controlled visual effects	Fully implemented
22	Pixel level collision	Not implemented
23	Serial communication events	Fully implemented
24	Serial communication game control	Fully implemented

2 Basic functionality test plan

2.1 Intro

Test of specific functionality	Test setup and actions	Expected result	Actual result
Program displays student name and number initially.	Load the program and check that name and student number are displaying as expected.	Student name and student number are displayed.	As expected.
Pressing SW2 starts the game.	After loading game, press SW2.	Intro screen clears and game screen is drawn.	As expected.

2.2 Pause game

Test of specific functionality	Test setup and actions	Expected result	Actual result
When joystick centre is pressed once, the game pauses.	Press the joystick centre while the game is running.	All sprites stop moving and are unable to move by using the normal controls, the game screen is cleared, and information is displayed.	As expected.
Game information is displayed on the pause screen.	Press the joystick centre to view the pause screen.	Lives remaining, current score, and game time in mm:ss format are displayed.	As expected.
The game is resumed when joystick centre is pressed again.	Press the joystick centre while on the pause screen.	Pause screen disappears and game reappears on screen.	As expected.

2.3 Player size

Test of specific functionality	Test setup and actions	Expected result	Actual result
The player initially appears on a 'starting block' in the top row.	Start the game.	The player begins on a stationary safe starting block in the top row.	As expected.
The player's sprite is at least 3 pixels high and 3 pixels wide.	Run the game and observe the player sprite.	The player's sprite is 8 pixels high and 9 pixels wide.	As expected.

2.4 Block size

Test of specific functionality	Test setup and actions	Expected result	Actual result
All blocks are at least 2 pixels high.	Start the game and observe the block sprites.	All blocks are 2 pixels high.	As expected.

All blocks are at least 10 pixels wide.	Start the game and observe the block sprites.	All blocks are 10 pixels wide.	As expected.
Blocks are clearly distinguished from each other.	Start the game and observe the block sprites.	All blocks have visible horizontal spacing between each other.	As expected.
All blocks are always at least `player sprite height + 2` pixels vertically separated from other blocks.	Start the game and observe the block sprites.	All blocks are 10 pixels vertically separated from other blocks.	As expected.
There are at least 7 safe blocks on the screen at one time.	Start the game and observe the block sprites.	There are at least 7 safe blocks on the screen	As expected.
There are at least 2 forbidden blocks on the screen at one time.	Start the game and observe the block sprites.	There are at least 2 safe blocks on the screen.	As expected.

2.5 Random blocks

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
Blocks have no consistent observable pattern.	Start the game and observe the blocks.	Each block appears in a randomly selected row and column.	As expected.
Blocks do not overlap other blocks.	Start the game and observe the blocks.	All blocks stay within their respective rows and columns.	As expected.

2.6 Player movement

Advanced functionality implemented, see "*Player velocity*".

2.7 Treasure

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
The treasure sprite is no larger than the player's sprite.	Start the game and observe the treasure sprite.	The treasure sprite is 8 pixels high and 8 pixels wide.	As expected.
The treasure sprite does not overlap any of the blocks.	Start the game and observe the treasure sprite movement.	The treasure sprite never overlaps a block.	As expected.
The treasure sprite spawns in the bottom half of the screen.	Start the game and observe the treasure sprite.	The treasure sprite spawns above the bottom row of blocks.	As expected.
The treasure sprite moves back and forward, changing horizontal direction when it reaches the edges of the screen.	Start the game and observe the treasure sprite movement.	The treasure sprite moves back and forward horizontally and 'bounces' off the edges of the screen.	As expected.

The treasure sprite stops moving when SW3 is pressed and starts moving again if SW3 is pressed again.	Press SW3 while the treasure sprite is visible, then press it again.	The treasure sprite will stop moving, then start moving again.	As expected.
The treasure sprite disappears when the player collides with it and gives the player 2 more lives and returns the player to the ‘starting block’.	Guide the player to the treasure sprite and collide with it. Press joystick centre to check lives.	The treasure sprite disappears, the player gains 2 more lives, and is returned to the ‘starting block’.	As expected.

2.8 Basic game mechanics

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
The player starts with 10 lives.	Start the game and press joystick centre to view the player’s lives on the pause screen.	The player has 10 lives.	As expected.
A point is scored every time the player lands on a safe block.	Move the player around onto multiple safe blocks, then press joystick centre to check the score.	The player’s score goes up when landing on a safe block.	As expected.
The player dies if any part of the player sprite moves off the screen in any direction or manner.	Guide the player off the sides or bottom of the screen.	The player dies when it hits the edges of the screen.	As expected.
On death, the player respawns on the ‘starting block’	Kill the player using any method imaginable while having 2 lives or more.	The player dies and respawns on the stationary ‘starting block’.	As expected.
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.	Kill the player repeatedly by any means until all lives are gone.	The game over screen is displayed showing a message, total score, and game play time in mm:ss format.	As expected.
The game over screen allows the player to restart by pressing SW3 and score, lives, time, and player position all reset.	Press SW3 on the game over screen.	The game screen disappears and score, lives, time, and player position all reset, and the game restarts.	As expected.
The game over screen allows the player to end the game by pressing SW2 which clears everything and just displays student number on the screen.	Press SW2 on the game over screen.	The screen is cleared and the student number is displayed on the screen.	As expected.

3 Advanced functionality test plan

3.1 Block movement

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
All blocks move at a constant horizontal speed.	Start the game and observe block movement.	Blocks move at the same constant speed and do not accelerate by themselves, except for the starting block.	As expected.
Each row of blocks must move in the opposite direction to the row above it.	Start the game and observe block movement.	Adjacent rows move in opposite directions, except for the starting block.	As expected.

3.2 Player velocity

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
Pressing the joystick 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.	Move the player onto a moving safe block and press joystick left and right.	The player will move left when the left and right when the left and right joysticks are pressed, respectively, at a speed relative to the block.	As expected.
When in horizontal motion, the player's horizontal velocity must be greater than that of the block.	Move the player onto a moving safe block and press the joystick against the direction of motion.	The player makes progress against the direction of the block.	As expected.
If the player is moving horizontally on a supporting block and the joystick is pressed in the opposite direction, the player stops moving relative to the supporting block.	Move the player onto a moving safe block, then press joystick left and then right.	The player begins moving left relative to the block, then stops moving relative to the block and is carried by its motion alone.	As expected.
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.	When the player is stationary, press joystick left or right, then press the joystick the same direction again.	The player moves in the direction of the joystick and does not speed up or slow down.	As expected.
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.	When the player is stationary, press and release either joystick left or right.	The player continues moving in the direction the joystick is pressed after it is released.	As expected.

If the player is not supported by a block, then it will commence to accelerate downwards.	Guide the player off a safe block until it is not supported by any block.	The player will start to accelerate downwards.	As expected.
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.	Move the player off a moving safe block.	The player will keep its horizontal velocity but accelerate downwards according to gravity.	As expected.
If the player is moving without support of a block 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.	Move the player off a moving safe block so that it falls and lands on another moving safe block.	When the player lands, it will immediately begin to move horizontally in the same speed and direction as the block, and all vertical motion will cease.	As expected.
All other collisions between the player and safe blocks are treated in a manner consistent with the combined motion.	From a lower row of blocks, make the player jump into a safe block on the row above.	The player is given an upward velocity passes upward through the block above, keeping the horizontal velocity of its previous block and accelerating vertically downwards.	As expected.
Any collision with an unsafe block results in death.	Guide the player into an unsafe block by standing or jumping into it.	The player dies and a life is deducted.	As expected.

3.3 Player jumping

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
Pressing the joystick up while the player is supported by a block causes the player to jump.	Press the joystick up while the player is supported by a block.	The player is given an upward velocity.	As expected.
After UP is pressed, the player should commence to move upwards. Any horizontal motion should continue, and the acceleration provision when the player is not supported by a block will take effect.	Press joystick up while on a safe block with horizontal motion.	The player will retain the previous horizontal motion of the block while being given a new upwards velocity.	As expected.

Once up is pressed, the joystick has no effect until the player lands on a block or dies.	Press joystick up to jump when the player is standing on a block, then press joystick up again while the player is still in the air.	The player will jump once, then will not be affected by the next press and continue its jump.	As expected.
Immediately upon landing on a block, the player's velocity changes to match that of the block so it is carried along.	Make the player jump off a moving safe block so that it falls and lands on another moving safe block.	When the player lands, it will immediately begin to move horizontally in the same speed and direction as the block, and all vertical motion will cease.	As expected.
If the player jumps off screen, the player dies.	Make the player jump off any side of the screen.	The player dies.	As expected.
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.	From a lower safe block, make the player jump through a gap of blocks in the row above and land.	The player will jump high enough to land on the row of blocks above.	As expected.

3.4 Player inventory

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
At the start of the game, the player has five Food in their inventory.	Start the game and view the food count on the pause screen by pressing joystick centre..	The pause screen shows 5 food in the inventory.	As expected.
The Food sprite's area (i.e. sprite width * height) is no larger than the player sprites's area.	Place a Food by pressing joystick down and observe the food sprite.	The Food sprite is 3 pixels wide and 3 pixels high.	As expected
When the player is supported by a block and joystick down is pressed, Food appears.	Press joystick down when the player is supported by a block.	Food appears where the player is standing.	As expected.
The Food must be supported by the block it is on, and overlap the player sprite.	Place a Food by pressing joystick down and observe the food sprite.	The Food is supported by the block it is placed on, and overlaps the player sprite.	As expected.
When the player is supported by a block and joystick DOWN is pressed, the number of Food in the inventory decreases by one.	With at least 1 Food in the inventory, view the Food count on the pause screen, place Food by pressing joystick Down, then view the pause screen again.	The number of Food in the inventory decreases by one.	As expected.

The total Food count (inventory + screen) must always equal 5.	Place Food by pressing joystick down, view the pause screen, and keep doing this until no Food is left in the inventory.	The total Food count always equals 5.	As expected.
Food dropped on a block is carried by the block.	Stand on a moving safe block and place Food by pressing joystick down, then watch as it moves off screen.	The horizontal velocity of the Food matches the horizontal velocity of the block it is placed on, and Food wraps around the screen.	As expected.
Food may overlap.	Press joystick down twice while standing on a safe block.	Two Food will be placed on top of each other and overlap.	As expected.

3.5 Zombies

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
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.	Start the game and wait approximately 3 seconds.	Five Zombies appear from the top of the screen and fall straight down without overlapping each other.	As expected.
Zombies stop falling when they land on any block.	Start the game and observe the Zombie motion.	The Zombies stop falling when they land on a block.	As expected.
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.	Start the game and wait for a Zombie to fall off the bottom of the screen. Check the score on the pause screen.	When the Zombie falls off the bottom of the screen, it passes from view and does not reappear immediately. The player's score does not change.	As expected.
Zombies may overlap each other if they are moving in opposite directions on the same span of blocks.	Start the game and observe the Zombie motion.	If two zombies are moving in opposite directions on the same span of blocks, they overlap and pass through each other.	As expected.
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.	Start the game and observe the Zombie motion.	Zombies that are on a block that move off-screen are teleported to the other side of the screen where the block reappears.	As expected.
When a Zombie collides with Food, the Zombie disappears, the player gains 10 points, the Food disappears, and the Food inventory increases by 1.	Place Food in the path of a Zombie and wait for them to collide. Check the score and inventory on the pause screen.	When the Zombie collides with food, it disappears, the player gains 10 points, the Food disappears, and the Food inventory increases by 1.	As expected.

If a Zombie collides simultaneously with multiple Food, then the Zombie consumes one Food, leaving the others unaffected.	Place two Food on the same block in the path of a Zombie, and wait for the Zombie to collide with it.	The Zombie and one Food disappears, leaving the other Food still on the block.	As expected.
Zombies do not disappear when the player collides with them, but instead the player loses a life and respawns.	Guide the player to collide with a Zombie.	The Zombie is unaffected, the player loses a life and respawns.	As expected.
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.	Kill all five Zombies by placing Food in their paths or waiting for them to fall off screen, then wait three seconds.	The five Zombies respawn from the top of the screen.	Zombies respawn immediately.

3.6 Pause screen advanced

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
The pause screen shows the number of Zombies on screen.	Start the game and press joystick centre to view the pause screen.	In addition to other statistics, the pause screen shows the current number of Zombies on screen.	As expected.
The pause screen shows the number of Food in inventory.	Start the game and press joystick centre to view the pause screen.	In addition to other statistics, the pause screen shows the current number of Food in inventory.	As expected.

4 Specialised Teensy functionality test plan

4.1 ADC for block speed

Test of specific functionality	Test setup and actions	Expected result	Actual result
All rows of blocks move at the same speed which is proportional to the potentiometer value.	Star the game and turn the left potentiometer Pot0 down to minimum, then turn it slowly up to maximum.	The blocks will stop moving, then their speed will slowly increase proportional to the potentiometer value.	As expected.
Block movement speed should range from 0 to a speed where the blocks are still clearly visible.	Star the game and turn the left potentiometer Pot0 down to minimum, then turn it up to maximum.	Block speed is zero when the potentiometer is at a minimum, and blocks are still clearly distinguishable and playable at max speed.	As expected.

4.2 LED warning

Test of specific functionality	Test setup and actions	Expected result	Actual result
Both LED0 and LED1 flash at approximately 4Hz (synchronously or asynchronously) before the zombies spawn.	Start the game and observe the LEDs for the first approximately 3 seconds.	Both LED0 and LED1 flash at approximately 4Hz synchronously before the zombies spawn.	As expected.
Both LED0 and LED1 flash at approximately 4Hz (synchronously or asynchronously) continue to flash until all the zombies have landed or fallen off screen.	Start the game and observe the LEDs while the zombies spawn.	Both LED0 and LED1 flash at approximately 4Hz synchronously until all the zombies have landed or fallen off screen.	As expected.

4.3 PWM controlled visual effects

Test of specific functionality	Test setup and actions	Expected result	Actual result
When the player dies with more than one life remaining, the backlight fades off and the screen fades out so nothing is visible.	Start the game and kill the player by pressing joystick left or right to collide with a forbidden block.	The backlight gradually fades out via PWM to zero brightness and the screen contrast decreases to zero.	As expected.
After the screen fades out, the screen gradually goes back to normal contrast, the backlight fades on and the player respawns at the top row of blocks.	Start the game and kill the player by pressing joystick left or right to collide with a forbidden block.	The screen fades out, then the backlight gradually turns on via PWM and the screen contrast gradually increases to normal levels.	As expected.

4.4 Serial communication events

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
When the game starts, the name of event, player x position, and player y position are sent over USB serial.	Start the game by pressing SW2 while viewing the Teensy serial output from the computer.	A serial message saying that the game has started, player x position, and player y position are received over USB serial.	As expected.
When the player dies, the name of event, reason for death, lives after death, score, and game time are sent over USB serial.	Start the game by pressing SW2 and kill the player by any means, while viewing the Teensy serial output from the computer.	A serial message saying that the player has died, reason for death, lives after death, score, and game time are received over USB serial.	As expected.
When the player respawns, the name of event, player x, and player y are sent over USB serial.	Start the game by pressing SW2, kill the player by any means, and wait for it to respawn, while viewing the Teensy serial output from the computer.	A serial message saying that the player has respawned, player x, and player y are received over USB serial.	As expected.
When the Zombies appear, the name of event, number of zombies, game time, player lives, and player score are sent over USB serial.	Start the game by pressing SW2 and wait for the Zombies to appear, while viewing the Teensy serial output from the computer.	A serial message saying that the Zombies have appeared, number of zombies, game time, player lives, and player score are received over USB serial.	As expected.
When the Zombies collide with Food, the name of event, number of zombies on screen after collision, number of Food in inventory after collision, and game time are sent over USB serial.	Place a Food in the path of a Zombie and wait for the to collide, while viewing the Teensy serial output from the computer.	A serial message saying that a Zombie has collided with Food, number of zombies on screen after collision, number of Food in inventory after collision, and game time are received over USB serial.	As expected.
When the player collides with the treasure, the name of event, score, lives, game time, and position of player after returning to the top row are sent over USB serial.	Guide the player to collide with the treasure while avoiding Zombies and forbidden blocks, while viewing the Teensy serial output from the computer.	A serial message saying that the player has collided with the treasure, score, lives, game time, and position of player are received over USB serial.	As expected.

When the pause button is pressed, the name of event, player lives, player score, game time, number of zombies on screen, number of Food in inventory are sent over USB serial.	Start the game by pressing SW2 and pause the game by pressing SW3, while viewing the Teensy serial output from the computer.	A serial message saying that the pause button was pressed, player lives, player score, game time, number of zombies on screen, number of Food in inventory are received over USB serial.	As expected.
When the game is over, the name of event, player lives, player score, game time, total number of zombies fed are sent over USB serial.	Kill the player repeatedly by any means until all lives are gone, while viewing the Teensy serial output from the computer.	When all lives are lost, a serial message saying that the game is over, player lives, player score, game time, total number of zombies fed are received over USB serial.	As expected.

4.5 Serial communication game control

Test of specific functionality	**Test setup and actions**	**Expected result**	**Actual result**
When ‘s’ is sent from the intro screen, the game is started.	Turn on the Teensy and send ‘s’ over USB serial.	The game is started.	As expected.
When ‘a’ is sent while the game is running, the character moves left.	Start the game and send ‘a’ over USB serial.	The character moves left according to normal player movement.	As expected.
When ‘d’ is sent while the game is running, the character moves right.	Start the game and send ‘d’ over USB serial.	The character moves right according to normal player movement.	As expected.
When ‘w’ is sent while the game is running, the character jumps	Start the game and send ‘w’ over USB serial.	The character jumps according to normal player movement.	As expected.
When ‘t’ is sent while the game is running, the treasure starts and stops moving.	Start the game and send ‘t’ over USB serial twice with a pause in between when the treasure is visible and moving.	The treasure motion stops then starts again.	As expected.
When ‘s’ is sent while the game is running, Food is dropped where the player stands.	Start the game and send ‘s’ over USB serial while the player is on a safe block and has Food in inventory.	A Food is dropped where the player stands.	As expected.
When ‘p’ is sent while the game is running, the game is paused and game information is displayed.	Start the game and send ‘p’ over USB serial.	The game is paused and game information is displayed.	As expected.

When ‘r’ is sent after game over, the game is restarted.	Kill the player by any means until the player has run out of lives and the game over screen is displayed, then send ‘r’ over USB serial.	The game is reset and restarts from the beginning.	As expected.
When ‘q’ is sent after game over, the student number screen is displayed.	Kill the player by any means until the player has run out of lives and the game over screen is displayed, then send ‘q’ over USB serial.	The screen clears and the student number screen is displayed.	As expected.

5 Specialised Teensy functionality justification

5.1 Switch debouncing

When activating or releasing a hardware switch, transient digital signals can be generated when the electrical contacts transition between open and closed. This can result in a single switch press being detected as multiple presses in quick succession. Switch debouncing prevents this from happening.

Switches are debounced in the game by sampling the switches at a high frequency and storing these values. Timer 0 is set to an overflow frequency of approximately 112 Hz. Every time the timer overflows, an interrupt service routine is run that stores the value of each switch (joystick left, right, up, down, centre, SW2, and SW3) in the rightmost bit of the respective byte in the array `state_count[]`. All the previous values are shifted left, and then all but the 3 most recent values are truncated.

If all three of these bits for a given switch are '`1`', there is enough certainty that this switch is indeed pressed, and the rest of the program now sees this switch as activated. However, if one or more of these bits are '`0`', then this switch may either be open or experiencing transient signals, and the rest of the program sees this switch as open.

Furthermore, leading edge detection was implemented for the switch signals so that a single switch press only activates its function once. This was done by keeping a record of previous switch values, and comparing these to the current values every loop. If the previous and current value is the same, the switch value is ignored. However, if the previous value is a '`0`' and the current value is a '`1`', the game detects this signal and will trigger whatever functions are necessary.

5.2 Direct control of LCD write

For most of the game, pixels are written to a screen buffer, and then this buffer is sent to the LCD once per loop. The player death animation renders a column of Zombie sprites that 'wipes' across the screen from left to right and clears all other images. This could be done by generating the screen buffer for every frame and sending this to the LCD, but a process like that is relatively computationally expensive.

Instead, direct control of the LCD is used in the game to create this animation. The Zombie sprite array is diagonally flipped at the bit level, so that each byte corresponds to a column of the image instead of a row. This is done because the LCD attached to the Teensy receives data in 8-row high buffers. Then, this flipped sprite array is sent to the function `lcd_write` from the `cab202_teensy` library, along with an x and y value.

When the player loses all their lives, for loop is used to display a 6-high column of Zombie sprites on the left of the screen. The game then waits for a fraction of a second and then overwrites this area of the screen using `lcd_write`. This process is repeated with increasing x values until the Zombies reach the right edge of the screen, leaving blank pixels in their wake to signify the end of the game.

5.3 Timers

Four Teensy timers are used in the game to implement various functionality.

Timer 0 is used to debounce switches. Its overflow frequency is set to roughly 112 Hz, or a period of about 0.008 s. Every time Timer 0 overflows, the current value of all switches is stored in the rightmost bit of the storage array `state_count[]`. See '*Switch debouncing*' for details.

Timer 1 is used to flash the LEDs. Its overflow frequency is set to roughly 15 Hz, or a period of about 0.066 s. This timer is required because LEDs need to be flashed at a constant 4 Hz frequency while allowing the CPU to continue running instructions. An overflow counter for Timer 1 is kept that would turn on LEDs when it reaches `2`, then turn off LEDs and reset when it reaches `4`. This has the effect of flashing the LEDs at a period of roughly 4 Hz.

Timer 3 is used to track game time. Its overflow frequency is set to roughly 1.9 Hz, or a period of about 0.524 s. The time in the game is stored as number of overflows of Timer 3, then this is converted to seconds using the formula `elapsed_time = (timer3_overflow * 65536.0 + TCNT3 ) * 64 / 8000000`. The overflow counter for Timer 3 is reset every time a new game starts.

Timer 4 is used for LCD backlight PWM. It is set up so that the output of the timer drives a PWM signal on pin C7, which is connected to the backlight. This timer is required because PWM operates at high and regular frequencies that would be more difficult to sustain in software. The duty cycle of the can be set by changing the number of ticks per overflow, thus changing the percentage of time that pin C7 stays high. This ability to change the duty cycle is used to animate the backlight fading in and out when the player loses a life.

5.4 Program memory

To reduce the amount of memory used at any one time by the Teensy, large blocks of data can be stored in program memory and accessed when required, rather than being all being loaded into RAM at startup.

All sprite bitmaps are marked with the `PROGMEM` keyword so that these are kept in program memory when the game was compiled and run. To access these bitmaps, a pointer to the location of the bitmap data in program memory is passed to the function `load_rom_bitmap` from the `cab202_teensy` library. This function allocates memory in RAM, copies the bytes from program memory to RAM, and then returns a pointer to the data in RAM so that it can be used by the game. This RAM then needs to be freed after the program is done with it.

The downside to using program memory is the computational overhead and added complexity of copying data to RAM, then ensuring that this memory is freed afterwards. Other large data blocks, such as text strings for serial output and screen display, could have also been stored in program memory. However, it was ultimately decided that the added computational overhead and complexity was not worth the benefits, as the Teensy still had sufficient RAM to run the game without these changes.