COMP2121 S1, 2012

Space Brawls

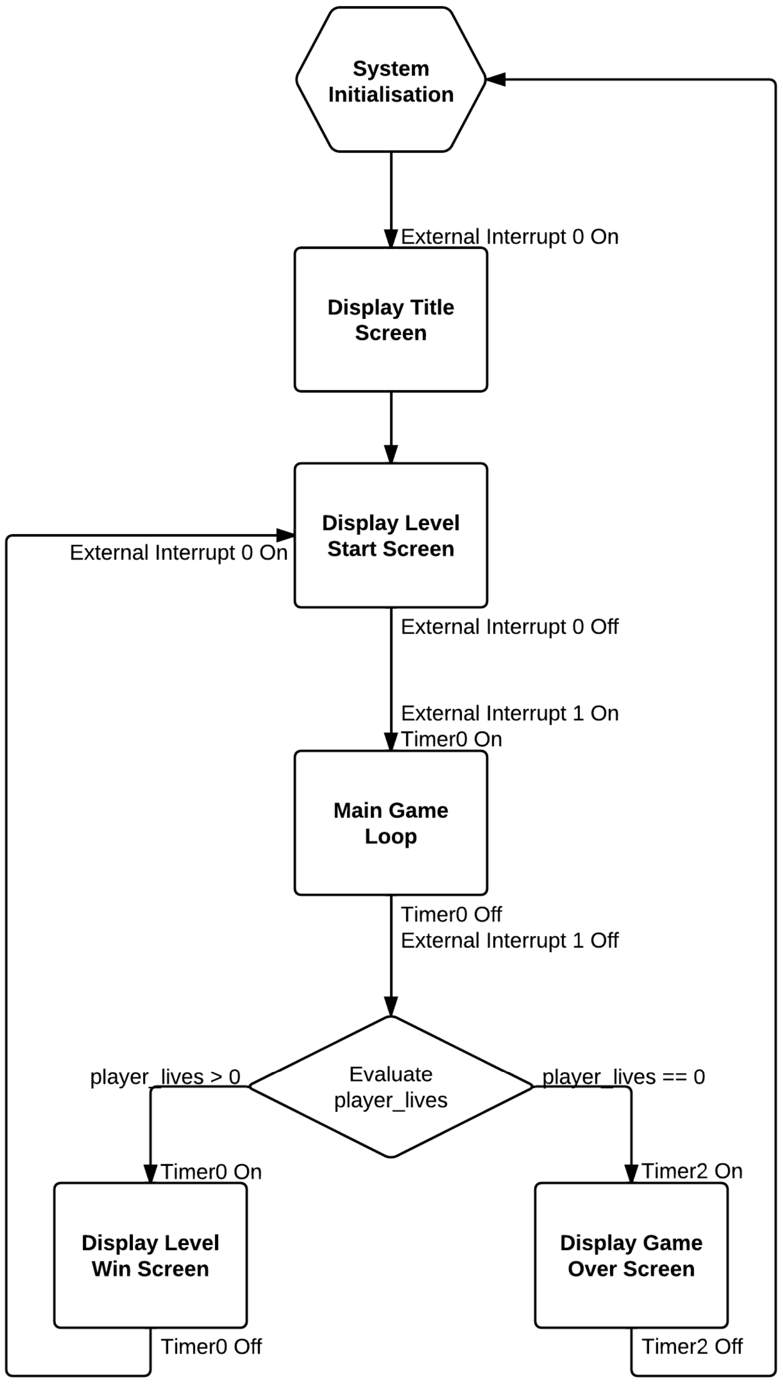
Design Manual

Matthew Moss and Tony Ward

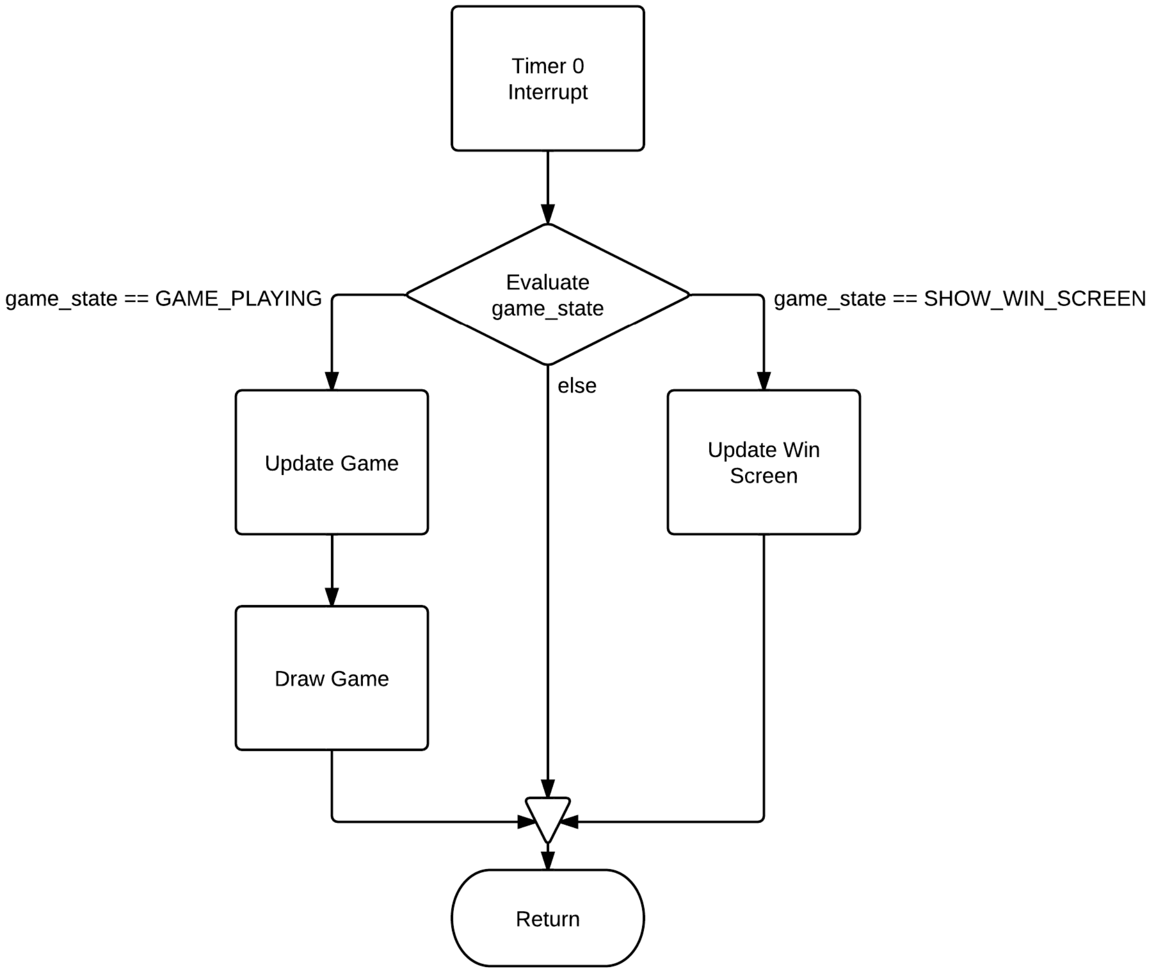
Table of Contents

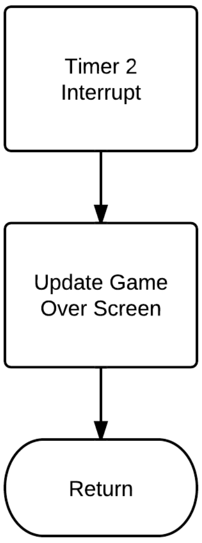
System Control Flow

System Overview

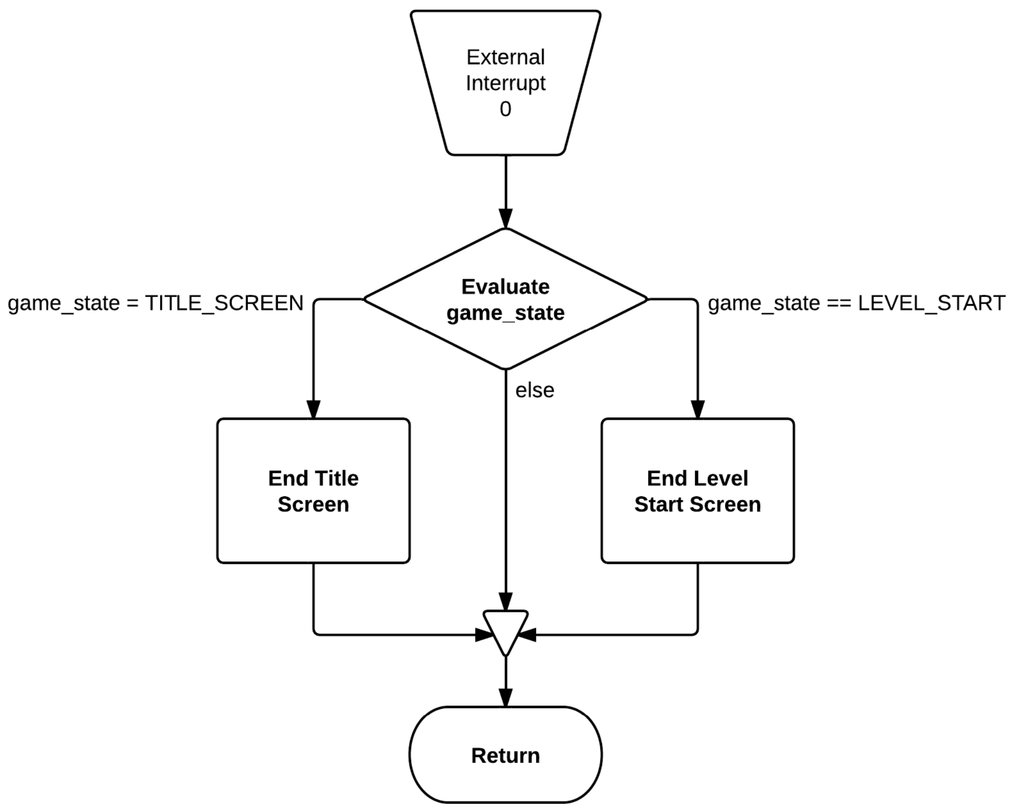


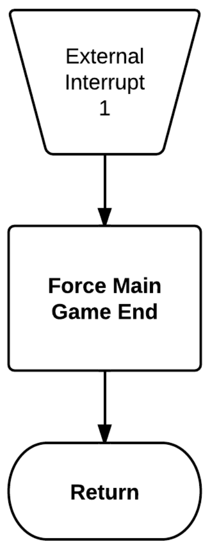
Timers


Timer 0 Interrupt
Evaluate game_state
game_state == GAME_PLAYING
game_state == SHOW_WIN_SCREEN
else
Update Game
Draw Game
Update Win Screen
Return


Timer 2 Interrupt
Update Game Over Screen
Return

External Interrupts





Data Structures

Data Memory

Our game uses a single array in data memory to store the state of the game board. All other information is stored in registers.

game_board: .byte BOARD_SIZE

During game play, this each element in this array is one of four types of value:

- A ' ' character. This represents empty space, and is drawn directly by the screen.
- A character in the range '0' to '9', inclusive. These represent weapons fired by the player
- A character in the range '0' + 10 to '9' + 10, inclusive. These represent weapons fired by the Death Moon
- A '*' character. This represents a collision between two weapons of the same value.

The contents of the array are changed every time step by the board_update function.

Registers

Our game also stores several pieces of information in registers that persist throughout the program.

- r18: game_state. Stores a constant according to the current state of the game. This value is changed when the current module changes.

Legal Values:

GAME_PLAYING

LEVEL_SCREEN

TITLE_SCREEN

WIN_SCREEN

GAME_FINISHED

- r2:r3 score_low:score_high. Stores the current player score. This is changed by several different events.
- r4: current_level. Stores the current game level. Is set to 1 at the start of the program, and increases every time a level is won.
- r5: player_lives. Stores the current player lives. Changed by the game_update function, and at the end of the main game loop.

State specific register data

- r15: cheats_detected. Used to indicate that the player has cheated, ending the current level and advancing. Also indicates their current score should not be increased at the end of the level.

Algorithms

Updating the Game

Every step, a function is called that updates the current state of the game. This is a fairly lengthy, but quite simple operation.

This algorithm relies on several flags. It uses fire_block and dm_fire_block to disable the next round of shots from either the player or the death moon. The algorithm also checks and updates dm_shots_left, which contains the number of shots the death moon is yet to make in the current round.

```
Iterate over the game_board array, replacing any '*' values with ' '
If the player has a weapon at the rightmost position of the board
    Replace it with ' ', and add points as appropriate.
If the Death Moon has a weapon at the leftmost position of the board
    Replace it with ' '.
    Deduct the appropriate number of lives from player_lives.
Iterating right to left over the array, each time a player weapon is found:
    If the position directly to the right contains an enemy weapon, and the weapons
    have the same value
        Replace the enemy weapon with a '*'.
        Replace the player weapon with a ' '.
    If the position to the right contains an enemy weapon, but the weapons have
    different values
        Replace the player weapon with a ' '.
        Deduct the appropriate number of points from the player score
Iterating right to left over the array, move each player weapon a single element to the right.
Iterating left to right over the array, each time an enemy weapon is found:
    If the position directly to the left contains a player weapon, and the weapons have
    the same value
        Replace it with '*'.
        Replace the enemy weapon with a ' '.
        Add points as appropriate.
    If the position to the left contains an player weapon, but the weapons have different values
        Replace the player weapon with a ' '.
        Deduct the appropriate number of points from the player score.
Iterating left to right over the array, move each enemy weapon a single element to the left.
If the fire_block flag is set
        Decrement the fire_block flag.
Else,
        Add the value of next_player_weapon to the leftmost position of the array
        Clear next_player_weapon
If dm_shots_left is not zero, and dm_fire_block is not set
        Generate a random probability if firing
        If the probability is above a certain preset level
            Set the rightmost element to a random value between '0' + 10 and '9' + 10
        Decrement dm_shots_left
Else
        Decrement dm_fire_block
If the board is clear, and dm_shots_left is 0
        Set the finished flag
```

This algorithm is responsible for all gameplay and changes to the game_board array.

Printing Numbers

Our code uses a simple subtraction algorithm to print the value of registers. This pattern can be extended to print registers of any size, but the example given is for an 8 bit register.

```
Assuming the value to be printed is in a register called num
While the value in num is greater or equal to 100
    Increment hundreds
    Subtract 100 from num
    Print (hundreds + '0')
While the value in num is greater or equal to 10
    Increment tens
    Subtract 10 from num
Print (tens + '0')
Print (num + '0')
```

There is some complexity involved when the number is split across multiple registers, but the overall pattern is consistent.

Determining Fire Probability

The probability that the death moon will fire starts at 1/8 and increases by 1/8 per level, capped at a maximum of 6/8. The following algorithm is used to determine the probability of the death moon firing.

```
Set the fire probability to (255/8)
While the level is greater than zero and the probability is less than 6/8
    Increase the fire probability by (255/8)
```

Determining Level Speed

Determining the speed is simple. For the first five levels, the level speed decreases by 1/10. For all levels following, the speed is halved.

```
Set the increments to zero
Set the speed to ONE_SECOND
While the increments is less than five and the level is greater than 1
    Increase the speed by (ONE_SECOND / 10)
    Increment increments
    Decrement level
While the level is greater than 1
    Decrease the speed by (speed / 2)
```

The only other complex algorithm in the game is the function used to cheat and increase the level arbitrarily.

Cheat Algorithm

```
Iterating over the board left to right
        Set each location to ' '
Set dm_shots_left to 0, to prevent the Death Moon from firing
Set ticks_per_step to 1, to force a timer interrupt and end the game quickly
```

Module Specifications

Module Overview

Our code is composed of several modules. These are fairly simple, but they combine to make the game perform as desired.

Throughout our program, the only values that need to persist are those in the current score, player lives and current level. All other information is localised to particular modules, and can be changed as desired. This makes most of the code reasonable simple, and means the connections between modules can be changed at will, making development and modification simple.

Reset/Game Setup

This module is responsible for initialising the stack, clearing the registers, filling the game_board array with ' ' characters, and setting the persistent registers to their starting values. The assumption is that any time this module runs, no previous register contents will persist.

Show Title Screen

This module is responsible for showing the game title screen. It assumes that no registers contain values that need to persist, other than the score, lives and level registers. The module enables timer0, and waits for an interrupt. The interrupt will set ready_to_start to a true value (not zero), breaking the loop and ending the module. The module is not implemented as a function as no register values need to be maintained.

Show Level Screen

This module shows the starting level screen. It again assumes no registers contain important values other than score, lives and level. It clears the ready to start flag, and waits for it to be set to end.

Main Game Loop

This module is responsible for running the game. It can infer from the value of level all the neccesary information to run the game. It sets timer0, and enables external interrupt 1. If external interrupt 1 is triggered, it increases the level and jumps to Show Title Screen, without showing the win screen. Otherwise, it waits until either player lives is zero, or the game board is clear and the death moon shots are zero. While waiting, the module reads the keypad input in a loop, and stores the most recent entry in the next_player_shot. If player lives is zero, the module ends and jumps to the Game Over module. Otherwise, it increases the score and then jumps to the Win Level module.

Win Level

This module displays the win screen at the end of the level. When it is called, it starts PWM on timer 2 in order to play a sound. This sound is then changed at several points by an interrupt on timer0. When a certain time has elapsed, the method jumps to the Show Level Screen.

Game Over

This module displays a game over message, starts PWM on the motorat approximetly 70rpm, and starts timer2. Timer2 sets a flag to indicate a required time has elapsed. When this flag is set, the module ends and jumps to the Reset/Game Setup module.