Tetris Under FreeRTOS

University of Canterbury
ENCE463 - Embedded Software Engineering - 2016
Simon Jones
15169024

Introduction

Tetris is a tile-matching video game developed by the Russian Programmer Alexey Pajitnov during the cold-war era. By 2010 the game had sold over 170 million copies across all platforms [1].

The objective of this project was to engineer the classic game Tetris on the Stellaris EKS-LM3S1968 development kit and to use a real-time operating system to manage the system operation. FreeRTOS [2] was selected as the real-time operating system to be used for this project.

Compliance to Specification

The main objective of the project was to implement Tetris on the stellaris EKS-LM3S1968. This was achieved, but the problem was more complex than initially speculated. Time constraints have lead to some non-critical parts of the specification being omitted.

The inputs and outputs are implemented as specified.

The functionality is implemented as specified with the exception of the difficult selection. This was omitted due to the aforementioned time limitations.

System timing compliance was determined with 10 trial games.

Trial	Average Input Delay (ms)	Maximum Input Delay (ms)	Average Refresh Rate Error (%)	Maximum Refresh Rate Error (%)	Level Reached
1	3	14	3	116	18
2	3	14	1	15	11
3	3	11	0	12	5
4	3	23	3	116	18
5	3	14	0	7	5
6	3	26	1	14	14
7	3	26	0	14	6
8	3	14	1	14	12
9	3	17	1	17	14
10	3	26	2	38	16

Table 1, Timing Tests

The average input delay and the average refresh rate error are within the specification. However, the maximum input delay and the maximum refresh rate error are outside the specification. This is not a major problem for the system as it is well within specification on average, so the occasional spike does not affect the stability or playability of the game.

Modules

Sixteen Modules are used by this project. Since many of them share a great number of functions they will only be listed here with a brief description of what their purpose. See the appendix for the function prototype descriptions.

Header File	**Description**
blocks.h*	Prototype block type definitions
board.h	Defines the board type and operations for accessing and modifying a board
bool.h*	Defines TRUE and FALSE
buttons.h	Defines button types and functions for reading and processing buttons.
display.h	Defines display types and functions for accessing the OLED display
game_engine.h	Defines the game type and game engine functionality
game_over_screen.h*	Defines the game over screen data array
images.h*	Defines the image type and all constant images
math.h	Defines several simple mathematical functions
screen_draw.h	Defines a set of functions for screen context changes
strings.h*	Defines constant strings
tests.h	Defines several simple tasks for testing system components
tetris_background.h*	Defines the tetris background data array
tetris_start_screen.h*	Defines the start screen data array
tetrominoe.h	Defines tetrominoe types and functions for tetrominoe manipulation
text_images.h*	Defines font (numbers and letters) data arrays
timer.h	Defines timer types and functions for manipulating timer types

*Modules that defined constant data/types and have no functions.

Table 2, Module List

Testing

Basic test task are implemented in a separate file, “tests.c”. These tests were written to interact with or replace the three tetris tasks.

void tetronimoe_drop_test(void* parameters);
This task was created to test basic tetrominoe alignment, dropping and rotation. It simply drops a tetronimoe from the top of the grid to the bottom, rotating it once per drop. This task utalizes the display queue to display the tetrominoes.

void tetronimoe_drop_test2(void* parameters);
This task is a more sophisticated version of drop_test. It generates a tetrominoe and incrementally lowers it until it is at rest. The task then places the tetrominoe and spawns a random new tetrominoe at a different x-position. This task tests collision detection and tetrominoe placement.

void vDisplayRunning(void* parameters);
This task is designed to test the display by periodically generating display tasks and enqueueing them onto the display queue. The effect of these tasks is a small spinning animation in the bottom right corner of the screen. This animation is used to indicate that the display task is running. It can also be used to show whether or not the system is completely frozen or not.

void vButtonTest(void* parameters);
This task tests the game engine by generating button event tasks and enqueueing them onto the button event queue.

The timing tests are directly integrated into the game engine. Statistics from these timing tests are displayed on the debug screen which can be viewed after exiting the score screen if the DEBUG_SCREEN_ENABLE constant was defined in game_engine.c when the program was built.

Tasks

The system runs using three tasks which are always running concurrently and never exit. The three tasks are used to decouple the input, processing and output. Queues are used to pass data between the tasks.

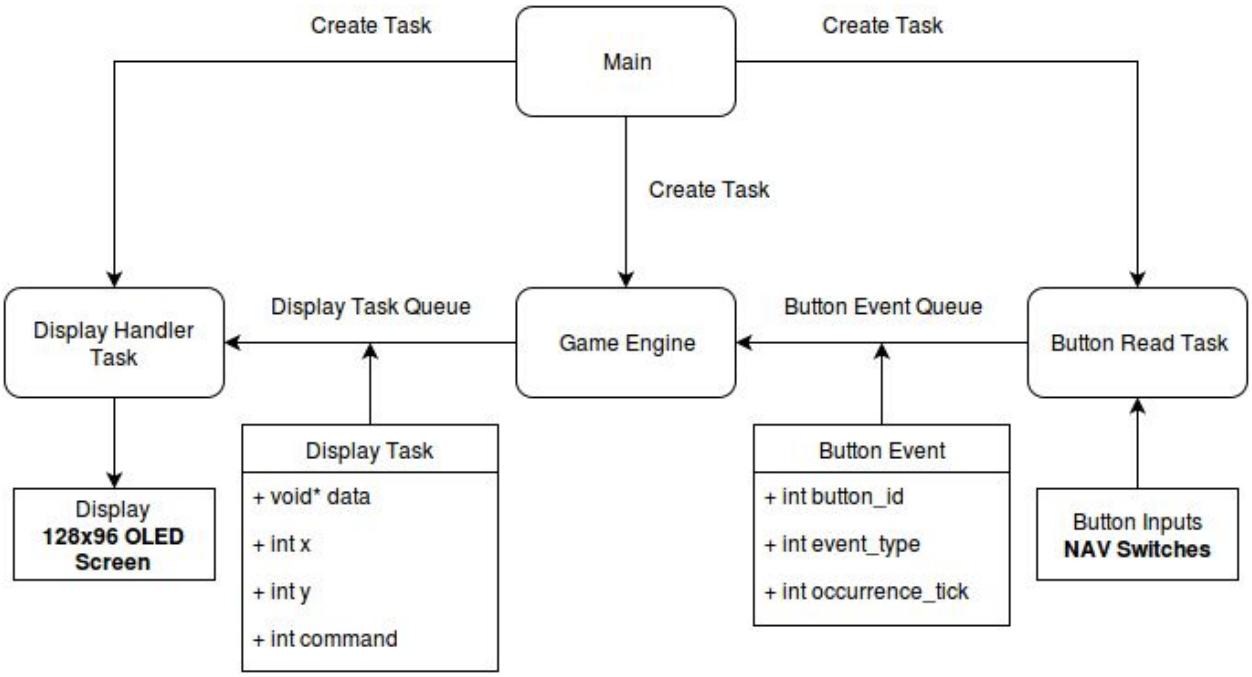

Figure 1, Tetris task system

Display Task

The first task is the display task xDisplayTask(). This task handles output to the 128x96 OLED display. This is performed by constantly polling a queue for “display tasks”. Received display tasks are executed immediately. Refer to figure 1 in the appendix for a state diagram.

Direct calls to the OLED display are treated as critical sections using the taskENTERCRITICAL() and taskEXITCRITICAL() functions. This is done to prevent unwanted visual artefacts from occurring due to preemption during a write to the screen. This means that display tasks that write large images are undesirable as they block other tasks for long periods of time. For this reason the a frame-buffer approach was not applied.

Instead the system sends many small display tasks while the game is running. Large display tasks are only sent when the screen display needs to be overwritten such as when the game ends and the score screen is displayed. The main disadvantage of this approach is that the game engine needs to “delete” unwanted drawings from the screen because the screen is not being constantly refreshed.

Game Engine Task

The second task that will be discussed is the game engine task. This is the most complex task as it handles all the behind-the-scenes data processing needed to run the game and switches

between the start screen, game running and score screen modes. Refer to figure 2 in the appendix for a state diagram of the overall system.

The game engine task reads button tasks from a queue, processes the tasks and uses information on the game's state to generate display tasks which are pushed onto the display queue.

Button Input Task

The third and final task is the button input task xButtonReadTask(). This task handles all button input and generates button event tasks which it pushes onto a queue. This is done by constantly polling the buttons. Refer to figure 3 in the appendix for a state diagram.

When a button is pressed or released the task will detect a state change in the button. If the button is not currently in timeout mode then the state change will generate a button event. The button event contains information regarding which button the event occurred on and whether the event was a push or release. After all the buttons have been polled and any tasks enqueued, the active timeouts are updated.

The philosophy with this task was to decouple the inputs from the game engine. This was useful for simplifying the input processing done in the game engine task as all the information regarding a button input is readily available within a button event structure.

Game Engine - Tetris Game Loop

Within the game engine there is a loop which runs the tetris game. Refer to figure 4 in the appendix for a state diagram of this loop. The game engine inherits code from most other modules in order to produce the game engine needed to drive tetris. Refer to figure 5 for a class diagram of the dependencies of the structures in the code. Refer to figure 6 for a timing diagram for the game engine.

References

[1] SoMa Play Connects With Ubisoft® To Bring Tetris® To Next-Gen Consoles. Accessed [2/8/2016]. [Online], Available: http://www.ireachcontent.com/news-releases/soma-play-connects-with-ubisoft-to-bring-tetris-to-next-gen-consoles-250340431.html

[2] FreeRTOS™. Accessed [11/8/2016]. [Online], Available: http://www.freertos.org/

Appendix

Header File Prototypes

blocks.h
No Functions. Defines prototype block structures.

board.h
Defines structures and functions for an array which holds information regarding the state of the game.

void clear_board(Board* board);
//clears a board (sets all cells to zero)

int get_board_value(Board* board, int x, int y);
//accesses the board at some x,y coordinates (board is actually a 1d array)

void set_board_value(Board* board, int x, int y, int value);
//sets the value of a board cell at some x,y coordinates

int get_next_complete_row(Board* board);
//returns the row index of the next row that is complete (all 1s)

int num_complete_rows(Board* board);
//returns the number of completed rows on the board

int is_off_board(Board* board, int x, int y);
//calculates whether an x,y position is on the board

int is_occupied(Board* board, int x, int y);
//calculates whether an x,y position is occupied

void clear_row(Board* board, int row);
//clears a row (sets it all to zero)

void trickle_down_column(Board* board, int base_row, int column);
//lowers all cells in a column from a given row index and upwards

int highest_occupied_cell_in_column(Board* board, int column);
//returns the row index of the highest occupied cell in column

void copy_board(Board* source, Board* dest);
//copies a board from source to dest

void board_difference(Board* board_a, Board* board_b, Board* output);
//copies the differences between board A and board B into output

bool.h
No Functions. Defines TRUE and FALSE.

display.h
Defines structures and functions for driving the OLED display.

void write_string(char* string, int x, int y, int brightness);
//writes a string to the display

void write_image(const Image* image, int x, int y);
//writes an image to the display

void write_tetris_number(int number, int x, int y);
//writes a tetris-number to the display (uses tetris font)

void write_tetris_string(char* string, int x, int y);
//writes a tetris-string to the display (uses the tetris font)

void initalise_display(void);
//initialises the OLED display

void clear_display(void);
//clears the OLED display

void xDisplayTask(void* parameters);
//Display task

void execute_display_task(DisplayTask* task);
//executes a dispay task

void update_latency(DisplayTask* task);
//update the screen update latency recordings

void initalise_display_queue(xQueueHandle* queue);
//initialize the display queue

void enqueue_display_task(xQueueHandle* queue, DisplayTask* task);
//enqueue a display task on the display queue

void quick_initalise_screen(xQueueHandle* queue);
//initialize the screen

void quick_send_image(xQueueHandle* queue, int x, int y, const Image* image);
//send an image to the screen

void quick_clear_screen(xQueueHandle* queue);
//clear the screen

void quick_send_number(xQueueHandle* queue, int x, int y, int number);
//sends a number to the screen

void quick_send_string(xQueueHandle* queue, int x, int y, char* string);
//send a string to the screen

void quick_send_string_rit(xQueueHandle* queue, int x, int y, char* string);
//send a "normal" string to the screen

Game_engine.h
Defines structures, constants and functions that implement the game logic needed to drive tetris.

void xGameEngineTask(void* parameters);
//game engine task. runs the game engine

void reset_game(Game* game);
//resets a game structure

Tetrominoe* get_current_tetrominoe(Game* game);
//gets the currently active tetrominoe from the game

void draw_current_tetrominoe(Game* game);
//draws a game's currently active tetrominoe

void erase_current_tetrominoe(Game* game);
//erases the currently active tetrominoe from a game

int spawn_new_tetrominoe(Game* game);
//attempts to place the next tetrominoe on the game grid. returns a TRUE/FALSE value to indicate success and failure

void get_next_tetrominoe(Game* game);
//shifts the next tetrominoe into the current tetrominoe and randomly selects a new next tetrominoe

void initalise_game(Game* game);
//initializes a game structure

void place_current_tetrominoe(Game* game);
//places the currently active tetrominoe in the game grid array

int can_drop_current_tetrominoe(Game* game);
//indicates whether or not the currently active tetrominoe can drop down.

game_over_screen.h
No functions. Defines an array of data used to construct the game over screen image.

images.h
No Functions. Defines the Image type and all constant images including, the font, numbers, tetrominoe cells, tetris background.

math.h
Defines some basic mathematical operations that are needed in the game's logic.

unsigned int urand(void);
//generates a random unsigned integer

int rand_range(int min, int max);
//generates a random number between min and max

void initalise_random_number_generator(unsigned long seed);
//initializes the random number generator with the specified seed

int abs_modulo(int number, int divisor);
//calculates the absolute modulus for a given number

int symbols_in_number(int number);
//calculates the number of symbols needed to represent a number

screen_draw.h
Defines constants and functions needed to draws the static portion of the screen for different game states.

void draw_start_screen(Game* game);
//draws the start screen

void draw_tetris_background(Game* game);
//draws the background for the tetris game

void draw_score_screen(Game* game);
//draws the score screen

void draw_debug_screen(Game* game);
//draws the debug screen

strings.h
No Functions. Defines all the constant strings used in the game.

tests.h
Defines functions for testing several of the tasks. There is a test for the inputs, ouputs and some aspects of the game engine.

void tetronimoe_drop_test(void* parameters);
//tests if the tetrominoe drop is working

void tetronimoe_drop_test2(void* parameters);
//tests if the tetrominoe drop is working and switches pieces

void vDisplayRunning(void* parameters);
//plays a small animation in the bottom right corner of the screen to show that the system is running. Also tests the display task's functionality

void vButtonTest(void* parameters);
//tests the button queue by periodically writing button events

tetris_background.h
No Functions. Defines the array data for the tetris game background image.

tetris_start_screen.h
No Functions. Defines the array data for the tetris start screen image.

tetrominoe.h
Defines structures, constants and functions that define tetrominoes and the operations that can be performed upon them.

void reset_tetromineo(Tetrominoe* block);
//resets a tetrominoe

const int* get_tetromineo_grid(Tetrominoe* block);
//accesses the array defining the tetrominoe's current pose

void rotate_tetromineo(Tetrominoe* block);
//rotates the tetrominoe 90deg clockwise

void shift_tetrominoe(Tetrominoe* block, int x, int y);
//translates the tetrominoe by a given x and y

text_images.h
No Functions. Defines the array data for the font and number images.

timer.h
Defines structures, constants and functions that are needed to perform timing related tasks.

int has_timed_out(Timeout* timer);
//returns true or false depending on whether a timeout has occurred

void start_timeout(Timeout* timer, int timeout_ms);
//starts a timeout

portTickType tick_difference(portTickType start_time, portTickType end_time);
//calculates the difference between two freeRTOS systick values. Overflow safe.

void timer_start(Timer* timer);
//starts a timer

void timer_update(Timer* timer);
//updates a timer (should be called periodically while a timer is operating)

void timer_stop(Timer* timer);
//stops a timer

int elapsed_time_ms(Timer* timer);
//gets a timer's elapsed time in ms

State Diagrams

Figure 1. Display Task State Diagram.

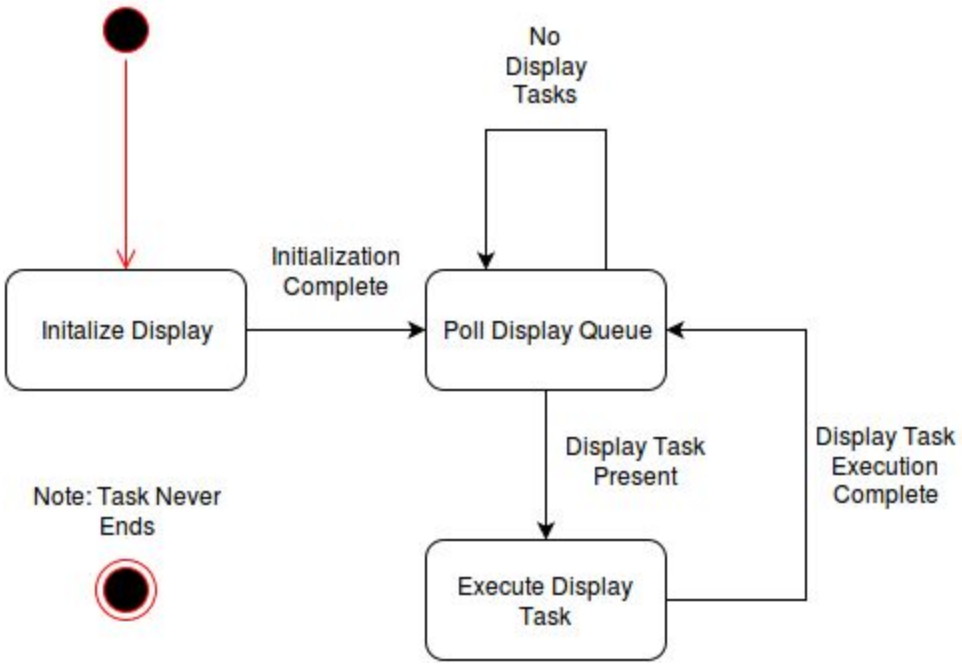

Figure 2. Game Engine Task State Diagram (Overview).

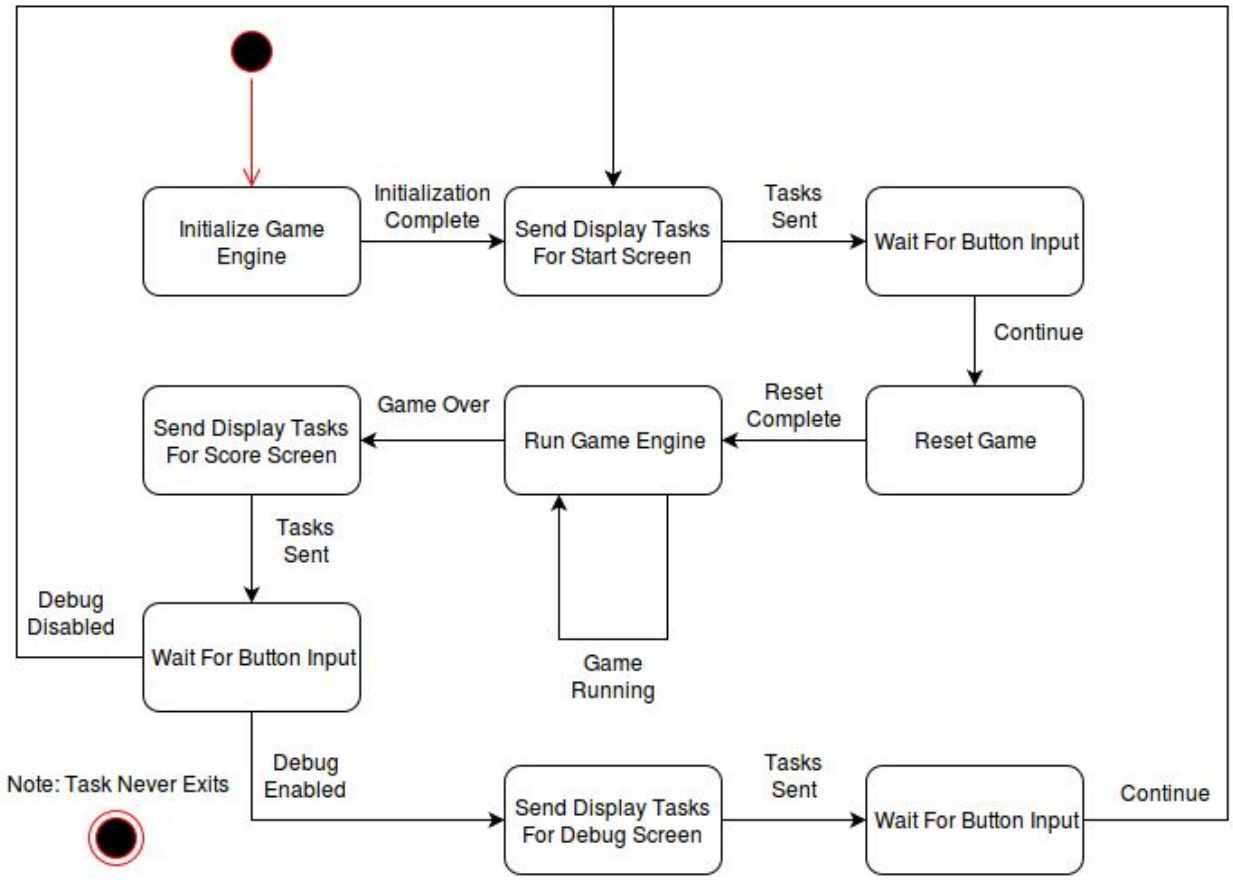

Figure 3. Button Input Task State Diagram.

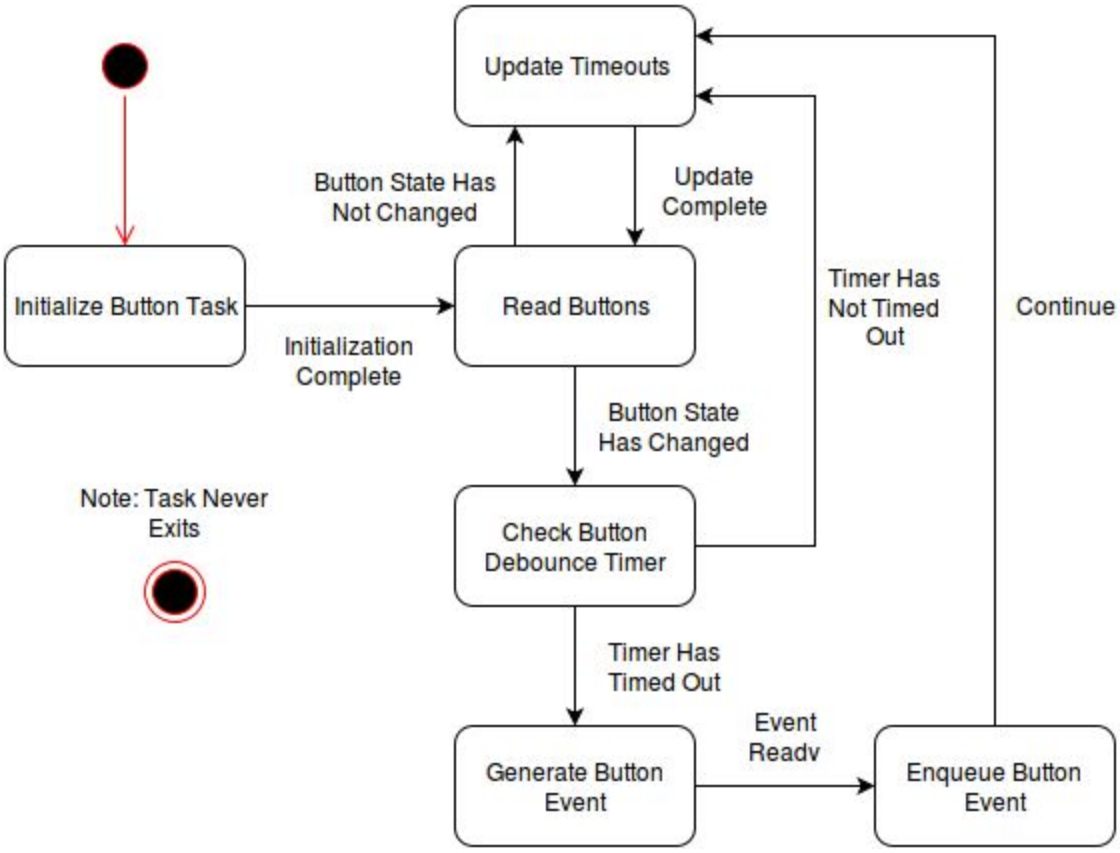

Figure 4. Game Loop State Diagram

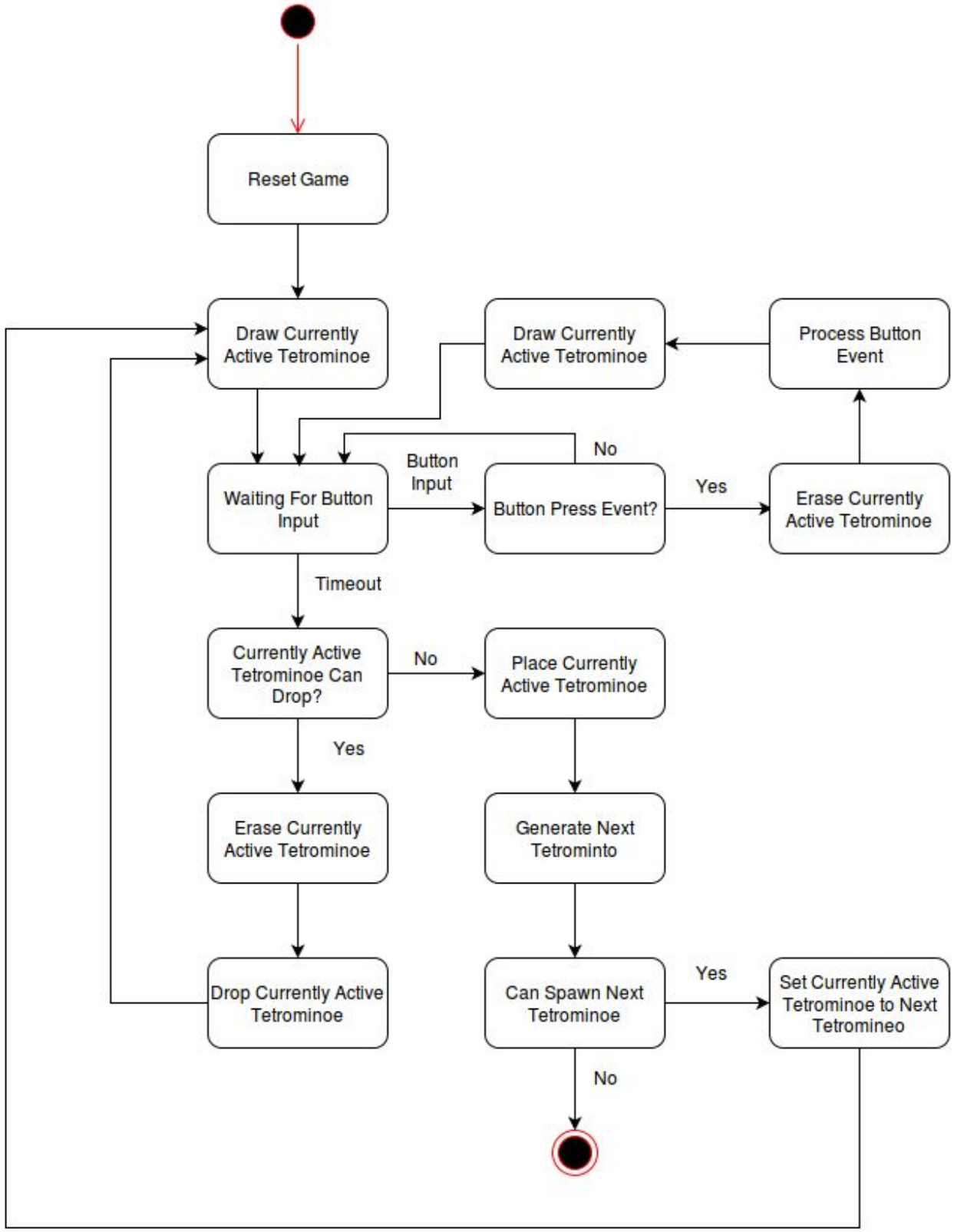

Figure 5, Class Dependencies for Tetris

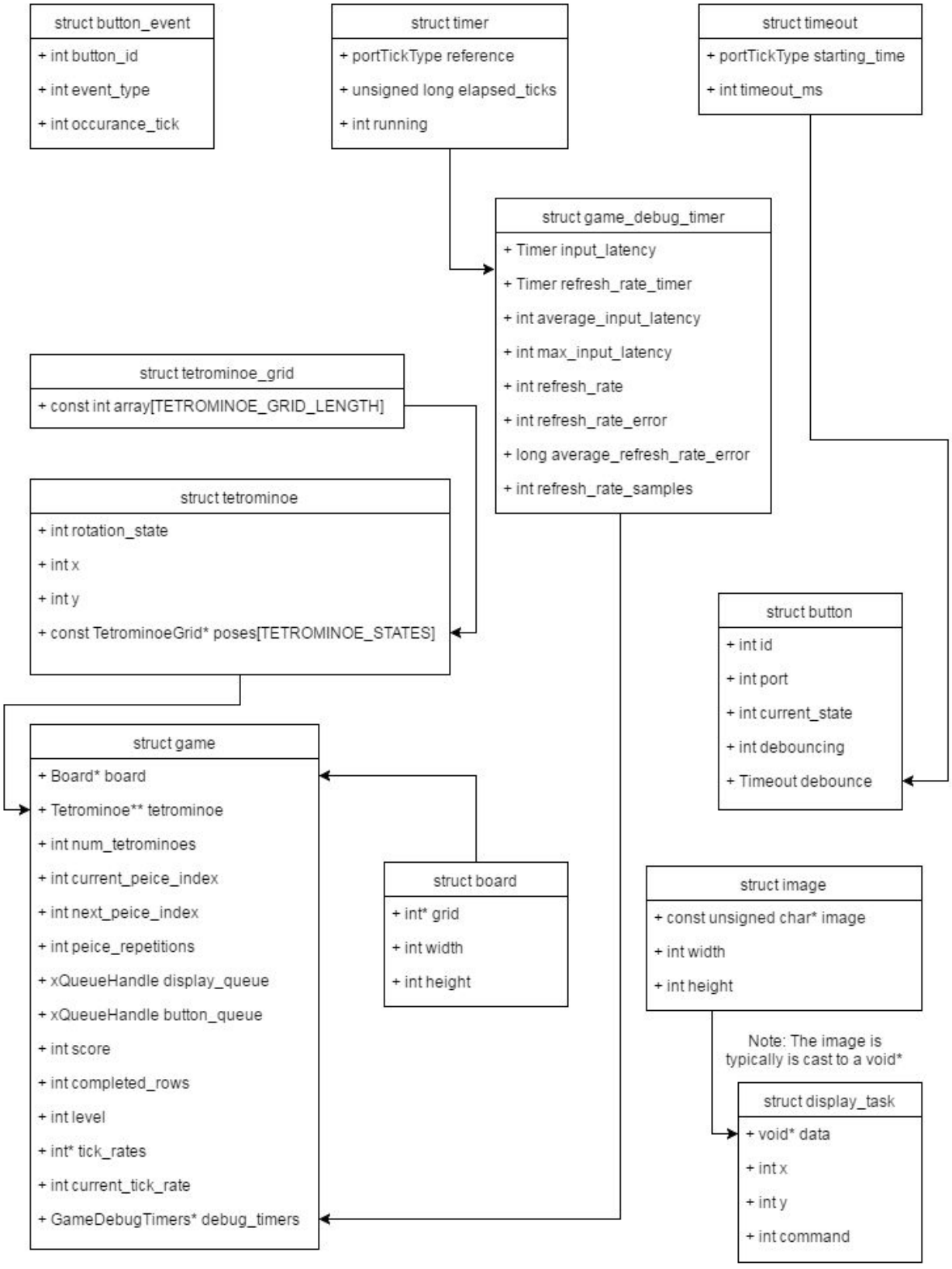

Figure 6, Timing Diagram for Tetris Game Engine