CMPE 490

Final Report

Jason Kulatunga – jk17@ualberta.ca
Tyler Best - tcbest@ualberta.ca

April 12, 2010

Declaration of original content:

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

matrix rotation code[1]
tetris colors and gameplay rules [2]
tetris scoring formula's [3]
lines/level up [4]
tetris game struct reference [5]
at91 board serial port ready check code [6]
pentomino blocks for twisted gametype [7]
keypad Schematic[8]

Abstract:

Our proposed plan for this project is to design and implement the game of Tetris using the Arm processor. The game of Tetris is a fairly simple game. The object is to make pieces of various shapes and sizes fit together neatly to make complete rows without running out of room on the screen. The main goal will be to have the game running and be playable. This will require the hardware to be interfaced, that is an LCD screen that will display the game play and allow the user to see what is going on. Also a 2 axis joystick will be used as a controller that will allow the user to move and place the pieces where desired. Aside from the hardware, the game software must be coded and then integrated with the hardware. There are several optional components that will be considered, time permitting. These options are color instead of black and white, some more information for the player (high score table, next piece upcoming, the players score and level), music and sound effects and a 2-player Tetris where two players can play at the same time. The hardware for the required part consists of the processor, the LCD screen and a joystick. The hardware for the optional parts consists of a second joystick, color LCD as opposed to black and white and a speaker for the sound. More in depth details are included in the following report.

Contents

Functional Requirements:

All of the following requirements have been met to the fullest degree.

1. The classical rules of Tetris are all intact. The game will drop pieces at random, clearing full rows as they are filled. Only once an incoming piece is unable to enter the playing surface will the game end.

2. System supports multiple levels, with each level having different game piece drop speeds and line clear requirements. The game will start on level 1 and increase after every 10 lines cleared. Drop speeds will depend on the current level.

3. 3 different game modes are playable: A classic Tetris, a twisted Tetris and a strategic Tetris.

4. Twisted Tetris has 17 new game pieces in place of the 7 classic pieces. Also each piece speed will now depend on a random number between 1 and 5, as well as the level.

5. Strategic Tetris has the 7 classic pieces, plus an additional bomb piece. The bomb piece will cause an explosion, taking out each block in a 1 block radius when the line it is in is cleared. The radius will gradually increase as the player levels up.

6. In addition, during strategic Tetris the player is given only a certain amount of moves per level. If this target is not met, the game will end. A health bar, located on the right side of the screen, will slowly disappear as the amount of moves available decreases.

7. The score, the level and the lines cleared this level are displayed next to game grid.

8. User is able to pause and resume the game at anytime. A play/pause button is included on the keypad

9. User is able to reset game at anytime. Hitting the play/pause button will also allow the user to scroll to the option of restarting their game.

10. User is able to quit game at any time. Hitting the play/pause button will also allow the user to scroll to the option of quitting the game.

11. Upon selecting quit in the menu, the user will be taken to a second menu allowing the user to select 1 of the 3 game types.

12. Controls simulate real-time play.

13. System is able to rotate game pieces clockwise. A rotate button is included on the keypad

14. LCD screen supports a 10x20 colored grid with proper resolution.

15. Controller must have 4 arrow keys, as well as a play/pause button and rotate button.

16. Each game piece given a unique color until it is set on the grid, at which time it will become permanently red.

17. User can “slam” blocks down when correctly oriented. Pushing up on the keypad will accomplish this.

18. Next game piece appears next to grid in both classic and strategic mode. 2 next game pieces appears next to grid while playing twisted mode.

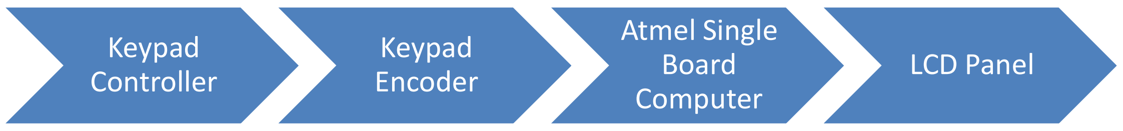
19. A ghost (shadow) is displayed directly below the income pieces. This replicates the exact position the piece will be set to if it remains in its current position.

20. In addition to the keypad controls, the user can tap the LCD to accomplish the “slam” command, or can touch and drag their finger, left or right, to make the piece move in the corresponding direction.

21. A game over graphic will display in the game grid upon a game over.

Design and Description of Operation:

Block Diagram of Data Flow



The Atmel Board runs a function that constantly checks the real-time clock and determines if enough time has passed for a the current game piece to move down 1 row on the grid. A function is then called to calculate the new locations of the blocks in pixels and output this information to the LCD. If a button is pressed, an interrupt is sent from the controller to the Atmel Board where functions are called to pause or resume game play, rotate blocks and move them in specific directions. These functions then perform mathematical calculations which determine the new pixel locations of the blocks, and render them. This information is then written to the LCD, displaying the users actions in real time. The 4 micro switches located on the base of the Joystick are connected to different interrupts on the ARM development board. These interrupts call functions which move the current block left or right, and speed up or drop the current block.

Hardware requirements:

-Arm microprocessor running at 32Mhz

-Minimal RAM requirements, on the order of 30KB

- LCD screen must support a 10x20 block game grid, where each block is at least 8px by 8px.

-LCD screen must have touch capabilities

- keypad with at least 6 buttons.

Parts list:

Part Description	Model Number	Relevant Spec	Supplier	Cost(approx)	Order Status
LCD display	ezLCD -002	260px x 160px	ECE Store	NA	-pre purchased
Keypad		16 Buttons	ECE Store	NA	-pre purchased

I/O:

Signal Type	Signal Name	Function
Input-Required	D0 – D3	Encoded Keypad Up
Input-Required	D0 – D3	Encoded Keypad Down
Input-Required	D0 – D3	Encoded Keypad Left
Input-Required	D0 – D3	Encoded Keypad Right
Input-Required	D0 – D3	Encoded Keypad Rotate Clockwise
Input-Required	D0 – D3	Encoded Keypad Rotate Counter-Clockwise
Input-Required	D0 – D3	Encoded Keypad Play/Pause
Input-Required	D0 – D3	Encoded Keypad Reset
Output-Required	UART2_TX	LCD Display RS232 Transmit Line
Output-Required	UART2_RX	LCD Display RS232 Receive Line

Software Design

Our software will be broken down into different modules. This will allow for easier design and testing to be done. Each file will have a specific purpose pertaining to each peripheral.

Main – Main.c will initialize a new game. This means drawing the initial game grid and other graphics on the screen. It also means creating the initial first blocks. This is where the game loop is that will constantly move the pieces down. Main.c also contains all interrupt handlers. If any keypad presses, or LCD touches occur, main.c will deal with them accordingly.

LCD Driver – The LCD driver has all the functions required to use the LCD. This includes drawing the blocks, the grid and the screen.

Tetris File – Tetris.c has most of the function calls needed for the game play. This includes a function to generate blocks, move blocks, rotate blocks, set blocks, etc... An important component in this file is the function to check boundaries. This must check that once a block is moved or rotated that it does not collide with blocks that are currently set on the grid, or that it does not exit the 10x20 grid. By far the bulk of the program is done here.

Test Plan

Software

Hardware Independent

Software will be tested by mapping the LCD pixels to a 2 dimensional array. The pixels will be mapped such that "NULL" and everything else are off and on respectively. The array will be printed to the console, testing the correct output for the LCD display. Print statements will also be used to display the current state of each function before and after each function is run.

Hardware Dependant

Software that depends on hardware, such as interrupts and speakers will be tested. Interrupts will be tested by writing directly to the interrupt control register, enabling the interrupt in software, and allowing the program flow to be tested, as if the joystick had been moved. Software used to generate the sound information sent to the speakers will be tested using an oscilloscope and printing state information. The oscilloscope will show the information sent to the output pins, before the speaker is physically connected to the system.

Hardware

Keypad

The keypad will be tested by measuring the voltages across the various terminal pins as each button is pressed. Tests will be done to ensure that when a button is pressed only one terminal switch is closed. Similar to the Joystick, tests will also be done to ensure that the keypad supports the data voltages and currents required by the ARM development board.

LCD

The LCD functionality will be ensured by testing the LCD initialization. The LCD data and control lines will be set to clear and fill the display in a loop. This will ensure that the hardware works, and that the performance is satisfactory to run Tetris without any graphical stuttering.

Results of experiments and characterization:

There were a couple of trade-offs in this project. One was in selecting an LCD. When playing a video game, generally the larger the screen the better. However, large LCD's cost a lot of money. Therefore we had to find a balance between a size that would fit for our game play and something that would cost a lot. In our case,luckily, we were provided an LCD.

Another tradeoff was in the keypad. It was not possible to find a joystick at a reasonable price therefore we will use the keypad. The keypad can perform the same basics function as a joystick but at a much lower cost.

Another possibility is we have an old, used controller. This can be used in place of the keypad. The cost, for us, would be none; however the design effort will be greater than the design effort for the keypad since we've used keypads before and know how to interface with it, whereas the controller would take more time to familiarize with it.

Citations :

[1]http://bytes.com/topic/c/answers/779190-c-rotating-square-matrix-90-degrees

[2] http://en.wikipedia.org/wiki/Tetris

[3]http://www.experiencefestival.com/a/Tetris_-_Scoring_formula/id/2076749

[4]http://www.online-flash-games.net/free-tetris.php

[5] http://code.google.com/p/simple-tetris-clone/source/browse/trunk/src/game.h

[6] http://code.google.com/p/tick-tock/source/browse/trunk/ticktock/src/main.c?spec=svn208&r=208

[7] http://en.wikipedia.org/wiki/Pentomino

[8] http://www.ece.ualberta.ca/~cmpe401/fall2008/labs/Schematics/keypad.pdf

Appendix A: Quick Start Manual

The following is a step by step procedure of re-assembling the project. It should be noted that it is assumed that proper flashing of the Atmel board with the provided code has been done.

Step 1

Connect all headers between the Atmel board and the encoder board, making sure to align all numbered pins on the Atmel board to their corresponding numbered pins on the header of the encoder Board (pin 1 on the Atmel board, to pin 1 of the encoder board).

Step 2

Connect serial cable between the Atmel board and the LCD.

Step 3

Connect power and ground lines to encoder board. This will power the keypad and the on/off switch for the LCD. A voltage of about 3.3 V should be applied to the board. Do not go much higher then this otherwise it could overload the circuit.

Step 4

Connect power cord to the LCD. Then switch the LCD on, using the on/off switch. If the LCD is turned on it will display writing across the bottom of the screen.

Step 5

Connect power to Atmel board. The LCD should now be displaying the game and the keypad should be functional.

Appendix B: Future Work

Due to time restraints, not every feature that was envisioned was able to be implemented in the final project. The following is a list of what could have been done if there was longer to work on the project or what future groups could incorporate into a similar project.

Two player Tetris

This would involve either displaying two 10x20 game grids onto the current LCD or having a second LCD. Each player would get their own keypad.

The game play would involve having the players compete against each other by being able to hinder the other persons' game play by sending over lines of blocks, being able to invert the other players' controls for a period of time or some other sort of interaction.

It would also be possible for the two players to work together if they choose. They could work together to level up faster, or pass upcoming blocks to each other.

Sound

A speaker would have to be interfaced to allow for sounds to be outputted.

For basic implications, a tone could be emitter to signify various actions, including a dropped block, an increase in levels, a game over, etc...

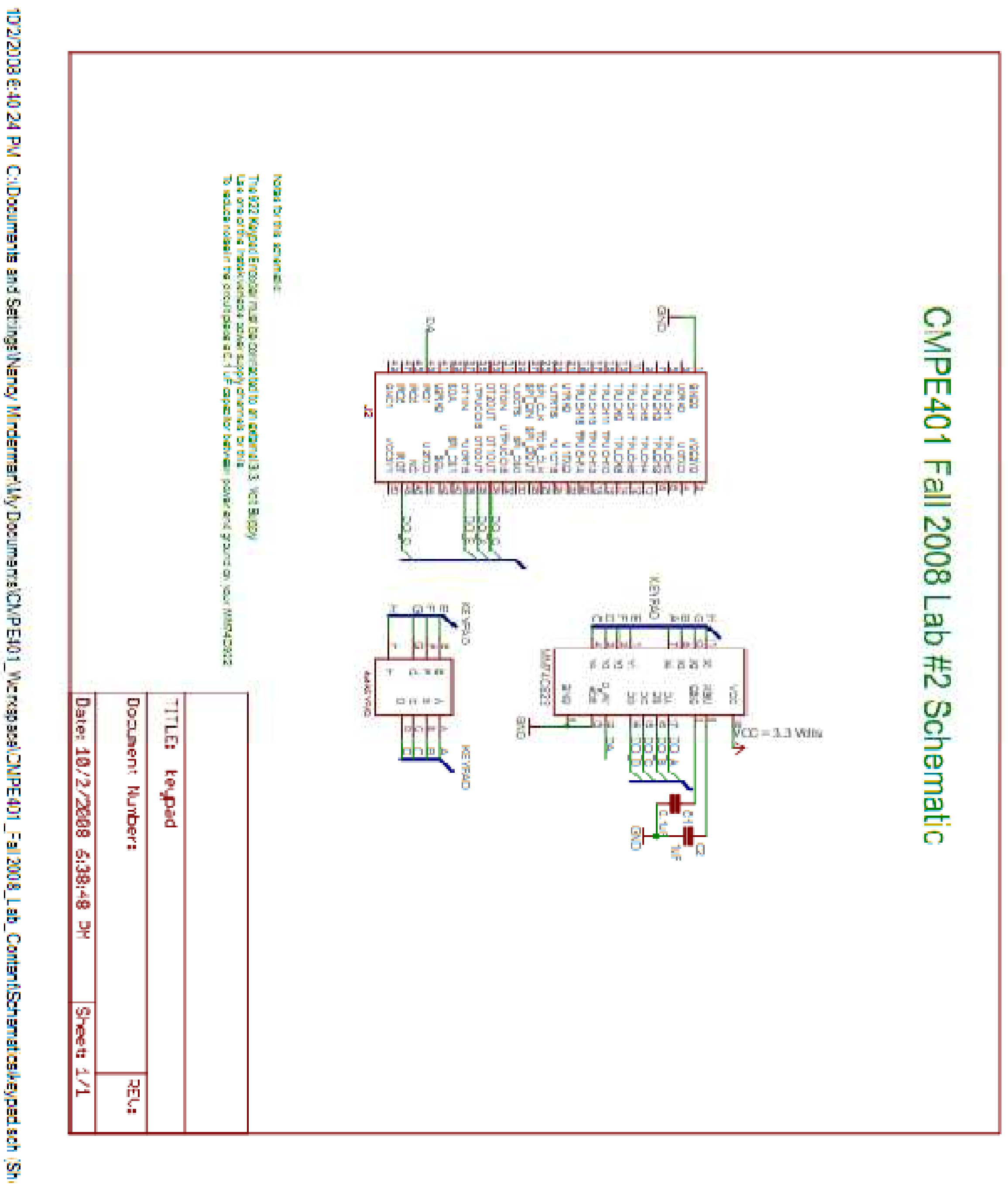
For more advanced implications, a speaker would be used to play music, as well as various sounds throughout the game.

Joystick

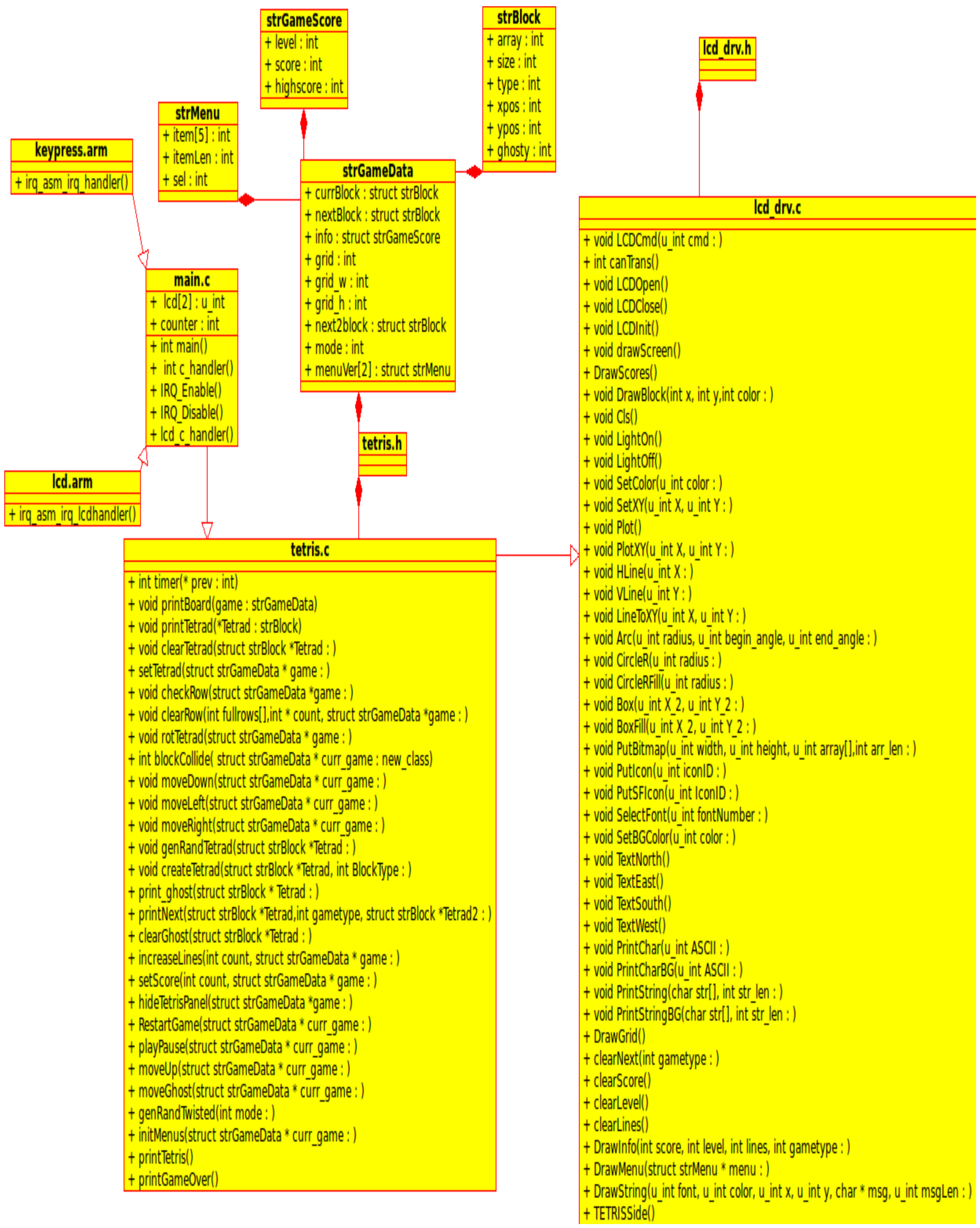
In place of the keypad, a joystick could be used for all controls. The joystick must be 2-axis, four directional, to allow for left, right, up and down controls. It also must have at least 2 buttons, one for rotation of the blocks and one for play/pause/select.

In addition to the previous options, there are many other options possible. Any new variations on the game play could be implemented as there have been countless variations of Tetris in the past 25 years.

Appendix C: Schematics


CMPE401 Fall 2008 Lab #2 Schematic
TITLE: keypad
Document Number:
REV:
Date: 10/2/2008 6:38:48 PM
Sheet: 1/1
MM74C922
KEYPAD
VCC = 3.3 Volts
GND
J2
10/2/2008 6:40:24 PM C:\Documents and Settings\Nancy Minderman\My Documents\CMPE401_Workspace\CMPE401_Fall2008_Lab_Content\Schematics\keypad.sch (Sh

Appendix D: Graphical Hierarchy of Code


strGameScore
+ level : int
+ score : int
+ highscore : int
strBlock
+ array : int
+ size : int
+ type : int
+ xpos : int
+ ypos : int
+ ghosty : int
lcd_drv.h
strMenu
+ item[5] : int
+ itemLen : int
+ sel : int
keypress.arm
+ irq_asm_irq_handler()
strGameData
+ currBlock : struct strBlock
+ nextBlock : struct strBlock
+ info : struct strGameScore
+ grid : int
+ grid_w : int
+ grid_h : int
+ next2block : struct strBlock
+ mode : int
+ menuVer[2] : struct strMenu
main.c
+ lcd[2] : u_int
+ counter : int
+ int main()
+ int c_handler()
+ IRQ_Enable()
+ IRQ_Disable()
+ lcd_c_handler()
tetris.h
lcd.arm
+ irq_asm_irq_lcdhandler()
tetris.c
+ int timer(* prev : int)
+ void printBoard(game : strGameData)
+ void printTetrad(*Tetrad : strBlock)
+ void clearTetrad(struct strBlock *Tetrad :)
+ setTetrad(struct strGameData * game :)
+ void checkRow(struct strGameData *game :)
+ void clearRow(int fullrows[],int * count, struct strGameData *game :)
+ void rotTetrad(struct strGameData * game :)
+ int blockCollide(struct strGameData * curr_game : new_class)
+ void moveDown(struct strGameData * curr_game :)
+ void moveLeft(struct strGameData * curr_game :)
+ void moveRight(struct strGameData * curr_game :)
+ void genRandTetrad(struct strBlock *Tetrad :)
+ void createTetrad(struct strBlock *Tetrad, int BlockType :)
+ print_ghost(struct strBlock * Tetrad :)
+ printNext(struct strBlock *Tetrad,int gametype, struct strBlock *Tetrad2 :)
+ clearGhost(struct strBlock *Tetrad :)
+ increaseLines(int count, struct strGameData * game :)
+ setScore(int count, struct strGameData * game :)
+ hideTetrisPanel(struct strGameData *game :)
+ RestartGame(struct strGameData * curr_game :)
+ playPause(struct strGameData * curr_game :)
+ moveUp(struct strGameData * curr_game :)
+ moveGhost(struct strGameData * curr_game :)
+ genRandTwisted(int mode :)
+ initMenus(struct strGameData * curr_game :)
+ printTetris()
+ printGameOver()
lcd_drv.c
+ void LCDCmd(u_int cmd :)
+ int canTrans()
+ void LCDOpen()
+ void LCDClose()
+ void LCDInit()
+ void drawScreen()
+ DrawScores()
+ void DrawBlock(int x, int y,int color :)
+ void Cls()
+ void LightOn()
+ void LightOff()
+ void SetColor(u_int color :)
+ void SetXY(u_int X, u_int Y :)
+ void Plot()
+ void PlotXY(u_int X, u_int Y :)
+ void HLine(u_int X :)
+ void VLine(u_int Y :)
+ void LineToXY(u_int X, u_int Y :)
+ void Arc(u_int radius, u_int begin_angle, u_int end_angle :)
+ void CircleR(u_int radius :)
+ void CircleRFill(u_int radius :)
+ void Box(u_int X_2, u_int Y_2 :)
+ void BoxFill(u_int X_2, u_int Y_2 :)
+ void PutBitmap(u_int width, u_int height, u_int array[],int arr_len :)
+ void PutIcon(u_int iconID :)
+ void PutSFIcon(u_int IconID :)
+ void SelectFont(u_int fontNumber :)
+ void SetBGColor(u_int color :)
+ void TextNorth()
+ void TextEast()
+ void TextSouth()
+ void TextWest()
+ void PrintChar(u_int ASCII :)
+ void PrintCharBG(u_int ASCII :)
+ void PrintString(char str[], int str_len :)
+ void PrintStringBG(char str[], int str_len :)
+ DrawGrid()
+ clearNext(int gametype :)
+ clearScore()
+ clearLevel()
+ clearLines()
+ DrawInfo(int score, int level, int lines, int gametype :)
+ DrawMenu(struct strMenu * menu :)
+ DrawString(u_int font, u_int color, u_int x, u_int y, char * msg, u_int msgLen :)
+ TETRISSide()