Ciaran Robertson 20260084

Software Project Manual

Software Description

This software is designed to interface with a drawing robot using G-code to play and display a game of tic-tac-toe, taking user input throughout the process. A file with a list of shapes and side data is read by the code and acts as a directory of potential shapes that each player can choose and use during the game. This file is read into the structure **ShapeRead** through the use of the function **load_shape**. The list of shapes that can be chosen are then listed using a printf function. The **select_shapes** function is then called to obtain user input for each player's desired playing shape. If an invalid shape name is inputted or both players choose the same shape the players will be prompted to choose again. The players are then prompted to input a board size of size between 30mm and 100mm. Similar to the shape selection stage if the inputted value isn't between this range the users are re-prompted to enter a valid value. The code then scales the shape sides using the **ScaleShapes** function to ensure the shapes are of the correct size and the **DrawBoard** function is then called to send G-code to the drawing robot to draw the correctly sized board.

From this stage the game is played. Each player is prompted to enter a move in the format of grid coordinates, with (1,1) being the bottom left grid square and (3,3) being the top right square. For each move the code checks if the input is valid (between 1 and 3) and whether or not the chosen square is already occupied. If it is taken or an invalid input the user is asked to select again. The taken spaces on the board are kept track of through the use of the 3x3 array called "board" with free spaces containing 0 and taken spaces containing 1 or 2 depending on which player has moved into that square. If the chosen move is valid the "board" array is updated to this move and the **DrawShape** function is called to send G-code to the drawing robot to draw the player's chosen shape in the selected square. This is then repeated for player 2.

After each valid move the **GameState** function is called to check if the move has resulted in the game ending. If the game has been won, the **WinLine** function is called to draw a line through the winning row and a winning message is then displayed informing the users that the game is over and which player has won followed by the termination of the code. If there are no more possible moves available and neither player has won, the game has been draws and the code displays a game draw message followed by the termination of the code. If neither scenario has occurred yet the loop of the player moving is continued until either of these conditions are met.

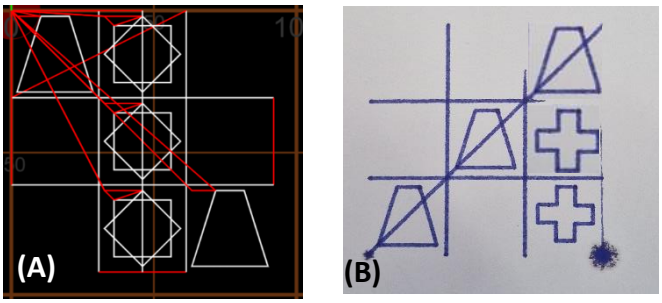

FIGURE 1: RESULTING OUTPUTS OF CODE FROM THE (A) G-CODE EMULATOR (B) PHYSICAL DRAWING ROBOT

Project Files

This program uses rs232.c, rs232.h, serial.c and serial.h files to run the section of code used to send the G-code from the program to the drawing robot. These were provided in the skeleton files as part of the resources provided at the start of the project. The data for the shapes used in the project are

contained in the file ShapeStrokeData.txt and also contains the data for the custom shape. Additionally, the main code used in the project is contained in the file SoftwareProject.c

Key Data Items

Name	Data type	Rationale
ShapeRead	Struct	Multiple sets of data of different sizes that need to be stored for easy access within the code, so structures are the best format to use due to the grouping nature of these data sets.
board_size	Float	Using a float variable for the board size allows for complete flexibility of grid size between the range values (i.e. any number within the grid size range can be selected).
unit	Float	This value of the unit definition of the shape steps for the inputted grid size must be a float to ensure that the symbols are the required size as specified in the brief (2mm from the grid square edges).
square_coords	Int	The square coordinates variable is the inputted square that the player wishes to put their symbol as a 1x2 array of the format (x, y). This coordinate is discrete, with both values in the array being integers between 1 and 3 inclusive as there are 3 grid squares in each row.
Board[3][3]	Int	3x3 array initially populated by zeros that keeps track of the previous moves. This is referenced to ensure players can't move into already taken squares and to check to see if the game has been won or drawn. Integer type has been chosen for this variable as there are only 3 possible values in the array: 0, 1 or 2.
Game	Unsigned int	Value of variable dictates the state of play of the game. Using int values means the states are easily differentiable from each other and making the variable unsigned frees up more memory in the system.

Functions

int num_of_shapes(FILE *fInput)

Function to read first line of the shape data file and output the number of shapes in the file to create a suitably sized structure for the data storage.

Parameters:

fInput – Pointer to the address of the open shape data file for access within the function

Return value – returns the number of shapes with data stored in the file.

int load_shape(FILE *fInput, struct shape_def *Shapes)

Function reads shape data file and inputs data into structure.

Parameters:

fInput – Pointer to the address of the open shape data file for access within the function

Shapes – pointer to return read data into structure for storage and later use.

Return value – returns 0 when function is successful.

int select_shapes(int *pIndex, struct shape_def *Shapes, int n, int Pn)

Parameters:

pIndex – Pointer to outside variable to store the location of the chosen shape for each player.

Shapes - pointer to return read data into structure for storage and later use.

n – number of shapes within the file (returned by num_of_shapes).

Pn – Variable indicating which player is currently choosing their shape.

Return value – returns 1 if successful, 0 if shape name does not exist within the structure.

float ScaleShapes(float board_size)

Function calculates and outputs unit size so that shapes fit into drawn grid with 2mm border from the grid lines.

Parameters:

board_size – size of whole playing board ($30 \leq board_{size} \leq 100$)

Return value – returns calculated unit size for shape drawing.

void DrawBoard(float board_size)

Function draws 3x3 board of overall size board_size by converting inputted board size into G-code instructions.

Parameters:

board_size – size of whole playing board ($30 \leq board_{size} \leq 100$)

Return value – No return value.

int PlayerMove(int Pn, int *x, int *y)

Function prompts moving player to input grid coordinates ensuring the inputted values are within the range 1-3.

Parameters:

Pn – Variable indicating which player is currently moving.

x – pointer to variable storing the x coordinate of the chosen move of the moving player

y – pointer to variable storing the y coordinate of the chosen move of the moving player

Return value – returns 0 when function is successful.

void DrawShape(int pIndex, int x, int y, struct shape_def *Shapes, float board_size, float unit)

Function draws player's chosen symbol into chosen board square.

Parameters:

pIndex – location in structure of moving player's chosen shape.

Pn – Variable indicating which player is currently moving.

x – variable storing the x coordinate of the chosen move of the moving player.

y – variable storing the y coordinate of the chosen move of the moving player.

Shapes – pointer to return read data into structure for storage and later use.

board_size – size of whole playing board ($30 \leq board_{size} \leq 100$)

unit - unit size of steps for shape drawing

Return value – No return value.

unsigned int GameState(int board[3][3], int Pn, int moves)

Function checks each potential winning line for 3-in-a-row and outputs data value corresponding to the win position, tie or ongoing game.

Parameters:

board[3][3] – 3x3 array storing the places each player has moved.

Pn – Variable indicating which player is currently moving.

Moves – Number of moves already made.

Return value - 0 = in progress, 1 = row 1 win, 2 = row 2 win, 3 = row 3 win, 4 = column 1 win, 5 = column 2 win, 6 = column 3 win; 7 = -ve diagonal win; 8 = +ve diagonal win, 9 = draw

void WinLine(unsigned int game, float size)

Takes game state value (returned from GameState function) and draws winning line through the 3-in-a-row according to this value.

game – Game state data value (as defined under GameState function above) indicating winning location.

size – board size used to obtain relevant coordinates for the line.

Return value – No return value.

Testing Information

Function	Test Case	Test Data	Expected Output
ReadShapeFile()	1. Verify the code can open the shape data files, read the data and input it into the defined structure. 2. Verify the failsafe against the potential of an invalid file	1. Reading the existing shape data files: SingleShape.txt ShapeStrokeData.txt 2. Reading invalid file	1. The files are opened without any problems or warnings. The whole of the text file used is read. All of the data is stored in the relevant sections of the designated structure. 2. Warning shown once file cannot be opened, and the function is exited.

	works as intended.		
DrawBoard()	1. Verify that the board grid is drawn correctly and to the correct specified size input by the user within the range specified in the brief.	1. Any board sizes in range $30 \leq x \leq 100$ (including decimal values	1. Board is drawn to the inputted specified size ($x \times x$ square grid split into nine squares)
PlayerMove()	1. Verify that valid moves are accepted and stored into board array, as inputted by the user. 2. Verify that invalid inputs result in an error and a chance to re-input valid value.	1. Valid Integer grid coordinates between (1,1) and (3,3) inclusive 2. Invalid grid coordinates outside range and float values e.g. (0,0), (4,1), (2.5,2) etc.	1. The move is accepted, and move is executed into the correct specified grid square. 2. The function should output an error message and re-prompt the user to input a valid value.
	1. Verify that a winning message is displayed, and the winning line is drawn through the row once a player has won. 2. Verify that if the grid is full with no winner there is a relevant output message, and then terminate the code.	1. Input moves such that one player ends up with three symbols in a row. 2. Fill the grid with moves such that there are no three identical symbols in a row.	1. Code should output a message indicating which player has won, should then draw a line through the 3-in-a-row and then terminate the code. 2. Code should output a message indicating that the game has been drawn and then terminate the code.
main()	1. Verify code will prevent players from selecting the same shape. 2. Verify that invalid inputs result in an error and a	1. Input two of the same shape names e.g. “SQUARE” 2. Input invalid shape names (e.g. “BOX”) or empty field (“”) 3. Any board sizes in range $30 \leq x \leq 100$	1. The function should output an error message and re-prompt the users to select different shapes. 2. The function should output an error message and re-

	chance to re-input valid value. 3. Verify that the board size inside of specified range is accepted. 4. Verify that invalid inputs result in an error and a chance to re-input valid value. 5. Verify that a player cannot select an already used grid square for a move.	(including decimal values 4. Values outside range ($x < 30$ and $x > 100$) 5. Input the same grid square coordinates on subsequent moves.	prompt the user to input a valid value. 3. The calculated unit size when used alongside the shape drawing data results in shapes that when drawn have a 2mm gap from the sides of the grid markings. 4. The function should output an error message and re-prompt the user to input a valid value. 5. Code should output an error informing the user of their invalid input and re-prompt the user to input a valid value.

These tests were run in the G-code emulator and the testing lab, and all outcomes were of those expected.

Flowchart(s)

Included as separate pdf.