Documentation for ThrowLin Videogame

1. Requirements

- Create a video game based on Javelin Throw sport
- The program should be written using C programming language
- The program should run on a PC with windows operating system
- The graphics_lib and amio_lib libraries are expected to be utilized in the program

2. Analysis

- The inputs to the program are from pressing a key on keyboard or clicking on mouse, expected keys are “r”, ”q” and left and right buttons on the mouse.
- The input to program will be translated to parameters such as velocity (from 0 to 68) and angle from (0 to 90 degrees) or just simply will act as commands to change the flow of the game/program
- The output of program will be in form of graphics, text and audio. The visuals from program will be projected on a 1280 by 960-pixel window frame
- The physics of the game will be following Newtonian laws for falling bodies under gravity:

$$Px = P_{x0} + \sum Vx \ * t$$

$$Py = P_{y0} - \sum Vy \ *t + (gt^2)/2$$

And Vector mathematics will be used:

$$V_{x\,=} |V| \times \cos\theta$$

$$V_{y\,=} |V| \times \sin\theta$$

- It’s expected that the user to be computer literate just enough to be able to interact with the computer. The user should know that he/she is running a video game software.

3. Specification

The requirements of the program:

1- draw a movable stick-man on the screen, able to launch a drawn projectile towards a target

2- the flight of the javelin must be subjected to the effects of gravity and wind

3- the distance the javelin is thrown is used to calculate a score, the achieved score should be displayed on the screen

4- The thrower should be within the predetermined runway area when the javelin is thrown – otherwise the throw is invalid;

5- The user can control:

- o the speed at which the thrower runs
- o the angle at which the javelin is thrown

o when to release (throw) the javelin

6- 6 tries will be given to the user in each game and based on that the final score is calculated

7- Sound should get incorporated into the game

8- Both mouse and keyboard should be used to control the game.

It would be desirable if:

9- The game would have a multi-player option;

10- ensuring that the game is not too frustrating to play — for example, give the player a

practice session and give hints on how to improve the throw

11- adding a simulated (computer) competitor.

12- there would be options for selecting different characters with different characteristics (Height, base speed of movement)

4. Design

1. User Interface Design

The program starts with welcoming the user and introducing the name of the game ThrowLin. Afterwards, it explains to user how the game is played. This text should appear on the console:

```
"Welcome to ThrowLin!
 In this game, you should try to throw the javelin as far as you can to score
the best.
"To start first you should try to choose the speed with which the runner will
be moving
First, Try to stop the oscillating slider by holding down the left button on
your mouse, the closer to middle the slider is, the higher your speed will
be.
Stopping the slider at a fast speed will give your athlete more power to
throw the javelin,
Then try to stop the runner before he passes the limit flag in the middle by
removing your finger form the button,
Closer to middle line you stop the runner, your chances for scoring better
will be increased!
At the end you should choose the angle of your throw,
Do so by stopping the oscillating indicator at your desired angle by doing a
left click, consider the wind direction!
After your shot you can watch the result of your throw and then by doing a
right click you can start throwing again.
press any key to start the game”
```

When user presses any button, The graphics window will be launched. The size of window should be 1280 by 960 pixels and defined with symbols.

Window_width_size	1280
Window_height_size	960

The details of opened graphic window will be described as follows:

The opened window will show the track and field area in 2-D. The coordinate line y = 540, will divide the screen to two sections of soil and sky. The background color for soil should be defined with symbol as BROWN and for sky it should be LIGHTBLUE. The sun in the sky will be depicted as a Yellow colored square with the side of 20 pixels.

There should be a stick person drawn at position x =75, The head will be represented with a circle, the center for head will be at (75, 340) and the radius will be 10 pixels. The height of torso will be 125 pixels and the torso will be a vertical line starting from lowest point of head circle. The two hands will be represented by an obliques line. The relative offset with respect to the center of head for start point will be (-37, 25) and for end point (+37, 100). The legs will be represented by two oblique lines. Starting from the end of torso and ending on the ground. The total distance between two legs should be symmetrical with reference to torso and equal to 2*37.

At the end of stickperson's right hand, a Javelin should be drawn. The Javelin will be represented by a straight line. The middle of line should intersect with stickperson's hand.

There should be a Flag at position x= 640, the height of which will be 100 pixels and it should be drawn on Ground line.

There will be an arrow at top left side of the screen which will show the direction of the wind, below that the speed of wind should be printed. User-score should be showing at the top-middle of the screen, with these texts:

```
X out of 6 throw   current score:   total score:           Highest score:

[the achieved score]
```

Symbol Name	Value
Ground_Line_y_pos	540
Sun_side	100
Head_center_x_pos	100
Head_center_y_pos	340
Head_radius	10
Torso_height	138
Start_dx_pos_hand	-50
Start_dy_pos_hand	+50
End_dx_pos_hand	+50
End_dy_pos_hand	+150
End_dx_pos_leg	50
Flag_height	100
Flag_x_position	640
Wind_indicator_x_pos	250
Wind_indicator_y_pos	160
Speed_slider_dx	12
Speed_slider_dy	100
Sky_bkg_color	LIGHTBLUE
Soil_bkg_color	BROWN
Javelin_dx	25

Javelin _dy	75
Angle_dx_pos	12
Angle_dy_pos	100
Stick_person_color	WHITE
Javelin _color	MAGENTA
First_note_pitch	60
Second_note_pitch	64
Third_note_pitch	67
General_channel	1
Flag_color	RED
SCORE_x_position	400
SCORE_y_position	125
splitter_width	5
Speed_slider_color	LIGHTGREEN

Few pixels below the ground line, there will be a slider, the knob of which will be constantly and swiftly sweeping the line from the left to right. This slider will determine the speed of the stick person. Below the slider line, at both ends of the line, the label “`slow`” should appear and below the middle of line the label “`fast`” should be printed. Few pixels above the slider, the “`speed`” label should be printed to inform the user.

At the first stage, user should select the speed for runner by freezing the slider at the desired position. The slider can be frozen by doing a left-click.

After user response, the slider should stop moving at this point and the runner should start moving from left to right. The runner will be stopped when the user releases the left button on the mouse. If the stickperson has reached the flag but there is no second response from the user, the stick person should be stopped and zero score should be given to user and then the user should be sent whether to starting stage or the final scoreboard stage.

if the user clicked at right moment, the following should happen:

At the leftmost side of screen, two perpendicular lines should appear, between those lines an arrow should be constantly and swiftly sweeping from 0 to 90 degrees. After user input (left click on the mouse), the Javelin will be thrown with the corresponding angle that is chosen and the trajectory should be drawn. According to where trampoline lands, the user should be given a score. Then with receiving a right click from user, the game will be guided to first stage again.

After receiving each click from the user, a distinctive note should get played. There are 3 clicks from the user so there should be three distinctive notes playing in total. The notes grow in pitch and velocity to for inducing a feeling of tension in the player.

The scoreboard at the top of screen will display the cumulative score of the user. The user will have 6 tries in each set, at the end of the game the user can choose to play again or not and following text should appear at the bottom of screen in RED color:

```
Press Q to return to main menu          Press R to restart the game
```

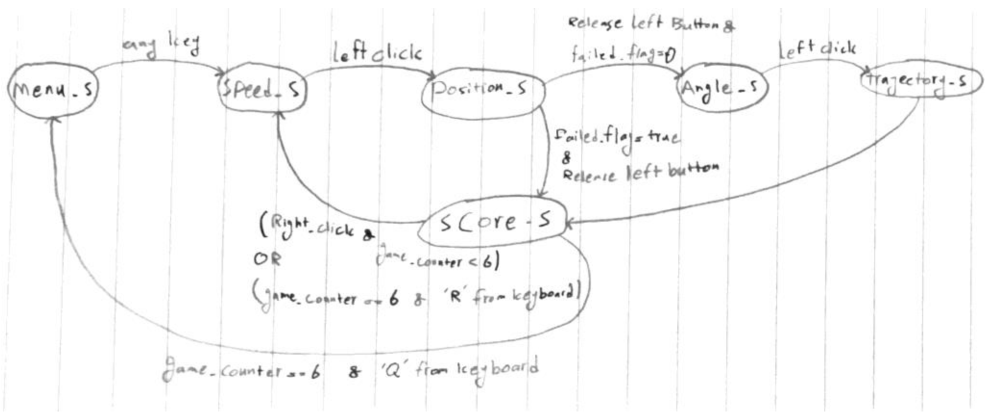
2. Program Structure Design

The whole program is designed with a Moore state machine. The description for each state and the main used functions are brought in below:

State name	State description	Function	Input Parameters	Return Value
Menu_S	In this state welcome script and tutorial will be printed on the console, when user enters any key, the state will transition to next one and general functions of game will get initialized.	Menu_display	(None)	(None)
		initialize_setting	(None)	(None)
Speed_S	In this state the graphic window will be opened, first window is cleared and current score and position of stickperson will get set to default and playing sound from previous stage will be muted. The skeleton for static figures that will be on the window for whole game will be drawn then the dynamic figure will get drawn which at this stage is a speed slider. When user presses the left button, the speed for stickperson will be selected and music note will be played on background then the state will transition to next one.	Play_sound	-Int, the pitch for the played note -Int, the velocity for the played note	(None)
		Draw_SpeedS_static	(None)	(None)
		Draw_SpeedS_dynamic	(None)	(None)
		draw_wind_indicator	(None)	(None)
Position_s	In this state, a moving stickperson will be drawn on screen until whether user releases the left button on mouse or stickperson reaches the flag line. When user releases the button, the second note will get played and depending on the position of stickperson, state can be transitioned to whether angle_s (if stickperson is on the left side of flag) or score_s(if stickperson has reached flag). This is how the correct position of runner will be validated.	Draw_PositionS_dynamic	(None)	(None)

Angle_s	In this state, first the static axes for angle indicator are drawn then dynamic angle indicator which will be moving in that structure is drawn. when user does a left click the angle of throw will be chosen, the third note will get played and state will transition to next one.	Draw_AngleS_static	- Int, the color for angle indicator and structure	(None)
		Draw_AngleS_dynamic	(None)	(None)
Trajectory_s	In this state, the trajectory for thrown javelin will be drawn based on the speed of stickperson, angle of throw and current speed of wind. At the end of this state, the final landing position of Javelin is calculated and automatically state transitions to next one.	Draw_Trajectory	(None)	(None)
Score_s	In this state, the resulting score from throwing will get calculated and updated based on whether the throw was valid or not. Then, results are printed on the screen. Afterwards, the set number is updated, if the game hasn't reached the end of set, it will wait for a right click from the user to get transitioned to speed state. If its at the end, the high score will get updated, and the user will be asked about restarting or quitting the game	Display_scores	(None)	(None)

State machine structure diagram:



Here is a description for the main global variables used in this program:

Variable	Range	C Type	Description
stick_person_x_pos	75-640	int	Determines the x-coordinate for the position of stickperson
speed_indicator_x_pos	12-162	int	Determines the x-coordinate for the position of oscillating speed indicator
stick_person_speed	1-22	int	Determines the how fast the character will run across the screen and also the initial momentum for throwing
initial_projectile_speed	1-68	int	Determines the magnitude of initial speed vector for projectile
angle	0-1.57	double	Determines the angle of angle indicator line, which eventually will translate to angle of throw
traj_pos_x	75-1280	int	Determines the x coordinate for the tip of Javelin, in the end it will indicate at which coordinate javelin is landed
wind_speed	1-6	int	Determines the speed of wind in the game, the value will be generated using a rand function
wind_angle	0-25	double	Determines the direction of wind in the game, the value will be generated using a rand function
failed_throw_flag	0, 1	int	Denotes whether the runner has passed the flag or is in the legal area
highest_score	0-600	int	Denotes the highest score achieved in the game's memory
total_score	0-600	int	Denotes the total score achieved in the current game
current_score	0-100	int	Denotes the current score achieved in the current set
game_counter	1-6	int	Denotes what is the current game set
state	0-5	int	Denotes the state of state machine

The program is designed with global variables for several reasons. In this project, the use of global variables greatly facilitates the writing of the program without introducing the risks associated with them to this particular project. This approach helped me to create a more engaging game with less time. The global variables are defined using a structure to give them a distinctive global tag to mitigate one of the dangers associated with them, so they wouldn't get mistaken with any other variable from other files. Moreover, the program is consisted of only 3 files and there wouldn't be any further files and features to be added, so there aren't many dependencies between different parts of program. Although I acknowledge that global variables have the disadvantage of making the program difficult to understand for other programmers when working in a group, I think the current project is explained with comprehensive details for anyone who wants to learn about the program and edit it.

The pseudo code for main algorithm is as follows:

```
Function main

      Call initialize_global_struct function
      Call run_state_machine  function

End function main

Function initialize_global_struct

      Give initial value to all structure members

End function initialize_global_struct

Function run_state_machine

      Begin loop forever
            If state = Menu_s
                  Go to Menu_display function

            Else if state = Speed_S
                  Go to Speed_state_draw function

            Else if state = Position_s
                  Go to Draw_PositionS_dynamic function

            Else if state = Angle_s
                  Go to Angle_state_draw function

            Else if state = Trajectory_s
                  Go to Draw_Trajectory function

            Else if state = Score_s
                  Go to Display_scores function

      End loop

End function run_state_machine
```

The program is consisted of many algorithms and limited space doesn't allow to bring the pseudo code for all of them here. Only the pseudo code for more complicated ones is brought here. The pseudo code for score state is as follows, it also demonstrates how the keyboard inputs from user are validated:

```
Function Display_scores

    Call Clear pervious score from screen function

    If failed_flag = true
        Reset the failed_flag
        Give user current score of zero
    else
        if traj_pos_x > Flag_pos_x
            give user curent score of (traj_pos_x - Flag_pos_x), and
            normalize it between 0 and 100
        else
            Give user current score of zero

    calculate the total score using the current score and print it on screen
    in red color

    increment the game counter
    if game counter <= 6
        wait till user right clicks
        change the state to speed_s
    else
        if total_score > highest_score
            clear the old score board
            update highest_score
            print new scores

        call the function for printing prompts

        Begin loop
            if there is a response from user
                if the response is from keyboard
                    if key = 'R'
                        game_counter =1
                        total_score = 0
                        state = Speed_s
                        break the loop

                    if key = 'Q'
                        state = Menu_s
                        close graphical display and the midi
                        break the loop
        end loop

End Function Display_scores
```

From other complicated algorithms, the functions that draw dynamic objects can be pointed to. The pseudocode for Draw_SpeedS_dynamic is brought here; it also demonstrates how the mouse inputs from user are validated:

```
Function Draw_SpeedS_dynamic

    intitalize the speed of slider

    loop while true
```

```
        erase speed indicator at previous position

        update the new position for the slider based on current position and
        speed of slider

        call check_pos_slider function to see if the slider will be in its
        specified range and correct it

        Draw the slider at its new position

        call update_display to move contents of buffer to the display

        if there is a response from user
            if it’s a left button-down event
                call play_sound function for the first pitch
                call convert_pos_to_vel function to map the slider position
                                                to speed
                state = Position_s
                break the loop
    end loop

End Function Draw_SpeedS_dynamic
```

5. Implementation: Report

The source code for the program is divided between 3 .c files. The main function is separated in its own file, the functions for speed state which make up a large part of the program is separated in another file and other functions for other states are placed in the final source code. This helps to organize the code base and makes finding them easier.

Source File	Functions	Global variable
main.c	int main(void) void initialize_global_struct(void) void run_state_machine(void)	Globals SYSTEM
Speed_state.c	void Speed_state_draw(void) void Draw_SpeedS_static(void) void Draw_SpeedS_dynamic(void) void draw_wind_indicator(void) int Check_pos_slider(int Slider_V) void convert_pos_to_vel(void) void print_score(int Score_color, int Highscore_color) void draw_stick_person(int x_position, int y_position, int person_color, int jap_color) void draw_speed_slider(void) void Play_sound(int pitch, int velocity)	
Other_functions.c	void Menu_display(void) void initialize_setting(void) void Display_scores(void) void Print_prompts(void) void wait_right_click_response(void) void Draw_Trajectory(void) void Angle_state_draw(void) void Draw_AngleS_dynamic(void) void Check_angle_indicator(double* Angle_speed, double* current_angle)	

	void Draw_AngleS_static(int Boundry_color) void Draw_PositionS_dynamic(void)	

Header File	Prototypes	Global Constants	

Speed_state.h	void Speed_state_draw(void);
	void Draw_SpeedS_static(void);
	void Draw_SpeedS_dynamic(void);
	void draw_wind_indicator(void);
	int Check_pos_slider(int Slider_V);
	void convert_pos_to_vel(void);
	void print_score(int Score_color, int Highscore_color);
	void draw_stick_person(int x_position, int y_position, int person_color, int jap_color);
	void draw_speed_slider(void);
	void Play_sound(int pitch, int velocity);		

Other_functions.h	void Menu_display(void);
	void initialize_setting(void);
	void Display_scores(void);
	void Print_prompts(void);
	void wait_right_click_response(void);
	void Draw_Trajectory(void);
	void Angle_state_draw(void);
	void Draw_AngleS_dynamic(void);
	void Check_angle_indicator(double* Angle_speed, double* current_angle);
	void Draw_AngleS_static(int Boundry_color);
	void Draw_PositionS_dynamic(void);
		Sky_bkg_color
		Soil_bkg_color
		Japoline_color
		Score_text_color
		Speed_slider_color
		Stick_person_color
		Flag_color
		Angle_Boundry_color
		New_Score_color
		Prompt_Color
		Wind_color
		Ground_Line_y_pos
		Window_width_size
		Window_height_size
		Thickness_size
		Sun_side
		Head_center_def_x
		Head_center_def_y
		Head_radius
		Torso_height
		Start_dx_pos_hand
		Start_dy_pos_hand
		End_dx_pos_hand
		End_dy_pos_hand
		Flag_x_position
		SCORE_x_position
		SCORE_y_position
		End_dx_pos_leg
		Trampoline_dx
		Trampoline_dy
		Flag_height
		small_horizontal_dis
		small_vertical_dis
		Speed_slider_dx
			Speed_slider_dy
			Speed_range_width
			splitter_width
			Slider_V_def
			TICK
			channel_music
			pitch_1st
			velocity_1st
			pitch_2nd
			velocity_2nd
			pitch_3rd
			velocity_3rd
			MAX_speed_projectile
			Angle_dx_pos
			Angle_dy_pos
			Angle_side
			Angle_V_def
			Angle_Max
			Angle_Min
			Angle_Extreme_margin
			screen_refresh_rate
			Gravity_Acceleration
			coeff_speed
			High_Score_X_pos
			MAX_current_score
			Total_Game_set
			Wind_Indicator_x_pos
			Wind_Indicator_y_pos
			MAX_wind_speed
			MAX_Wind_angle
			Wind_Indicator_length
			Wind_minor_value
			Wind_mild_value
			Wind_vector_coeff

The Gobals SYSTEM structure is also externed in header file for other_functions.h. These two header files are included in all of the source files. The functions in different files interact with each other using the included prototypes and the SYSTEM struct.

6. Verification and Testing

The program is created by using state machines, so the whole program is divided into states which contain smaller pieces of code. For the purpose of testing the program, each state was tested one by one, first in terms of functionality, to see if the performance of the state meets the required definition for it. Then the state transition rules which are the links between the states were verified.

The specification of the program is met with the functionalities embedded in each state. For example, the specification about "Thrower being in predetermined runaway area" happens in Position state. This specification was tested by bringing the game to the "Position_S" state and moving the thrower over and over to make sure that each time the game stops the thrower before he passes the predetermined area. In addition to verifying the game visually, breakpoints in the program were also used to verify quantitatively that the thrower doesn't pass the flag line. In a similar way, other specifications of the program were also tested.

The transition between states usually happens when we get an input from the user. For testing the transition links, a wide range of inputs beside the desired input were given to the program from user. This made sure that the transitions are executed correctly under (almost) all conditions. For example, when it was needed to receive a right click from the user, beside from right click, a range of inputs from a-z , 0-9 buttons were also given to the program to see if the program still functions correctly. In a similar way, other state transition rules were also verified.

In addition to the method above, it was also asked from several game testers to play the game and see if anything goes wrong or any bugs can be detected in the game.

One of the detected bugs in the game, happened when the stick person stopped in the same position as angle indicator and in early versions of program, this resulted in stickperson getting cleared from screen by the angle indicator. Then, this bug was fixed by redrawing the stickperson at its current position in addition to drawing the indicator at every iteration of the loop inside of angle state. The result of retesting can show that this accident doesn't happen anymore.

7. User Manual

The game can be installed by double clicking on the icon called Assignment in the main folder for the project. The game can be played on any pc with Windows operating system. Your pc should be connected to sound speakers for fully enjoying the game and it's required that you have a keyboard and mouse connected to your pc. Any person who can carry out simple tasks with computers, aged 4-99, can play this game. For learning how to play the game just read the tutorial instructions on the welcome page very carefully and follow them along with the prompts on the screen.