ELEC00002C
Maths and C Programming
January 2015

Examination Number: Y3604378

Contents

Requirements

The game must:

- Involve launching an object.
- Take into account the effect of gravity.
- Include some sort of obstacle in the path of the projectile in addition to some other element to increase difficulty.
- Be useable and enjoyable by those not experienced with computers.
- Be both rewarding and challenging.
- Run in Windows
- Utilise the provided graphics/audio libraries

Analysis of Requirements

Assumptions

That “someone not experienced with computers” refers to someone able to run a program from an executable file (.exe) and is able to follow instructions given in the program to use it.

The User

The game should be designed for someone who is likely to be play a computer game, and are also able to account for environmental factors such as wind and gravity. Such a person will likely be aged roughly between 10 and 16. As such the interface should be clear and not include unnecessary information which could become confusing,

Expected Inputs

Mouse Input to control both direction and momentum of projection. The momentum and the direction in which the projectile is fired could be set by clicking and dragging the cursor across the window so that it draws a line in the dame direction as the projectile will be fired.

Mouse Input to select menu options.

Outputs

Graphical output may include the stickman, a target, any obstacles and the projectile.

Numerical outputs may include the player’s current score, and the number of attempts the player has made this round.

Some audio indication that the round is over or that the target has been hit may be included.

Processes

Gravity will be simulated upon the projectile. The Equation $s = ut + \frac{1}{2}at^2$ can be used to calculate the x and y position of the projectile as t changes. This allows both gravity and any horizontal acceleration we wish to add to be accounted for easily.

Specification

The program shall:

- Be simple and intuitive to use.
- Draw a stick figure, a target and under certain circumstances, obstacles on the screen.
- Allow the user to fire a projectile by clicking and dragging the mouse across the screen.
- Simulate gravity on the projectile.
- Have incremental difficulty levels which increase each time a level is completed.
- Difficulty will be increased by adding other elements including:
    - Wind.
    - Obstacles.
    - A Moving Target.

The Program may:

- Give feedback to the player once a level is completed by playing a tune.
- Indicate the target has been hit by playing a tune.
- Include a menu allowing the user to select the difficulty.
- Include an options menu allowing the user to change the screen size and audio volume.

Design

The Theme

The game will be based around archery. The objective will be to score as many points as possible by firing an arrow at a target. Given this theme, the game will be called “Stickman Archery”.

User Interface

The Menu Screen

Upon running the program, a window of size 1260x480 will open. For convenience these will be defined as symbols which can be used from any file (ie not just the main file). Once the window has opened the menu screen will be displayed. The menu screen will display the title of the game "Stickman Archery" in the centre of the window, along with two buttons, "Play" and "Exit".

Button	Action	Dimensions		Position	
		X	Y	X	Y
Play	Displays the select difficulty menu	300px	100px	Centred	1/3 window height
Exit	Closes the window and stops the Program	300px	100px	Centred	2/3 window height

Figure 1 – Buttons to be displayed on the menu screen.

The program will wait until the window is closed or a button is pressed. When a button is clicked it shall run the function corresponding to that button.

While waiting, if the mouse hovers over a button, the width of the button border shall double.

The Level Select Screen

The level select screen will consist of four buttons; one to return to the menu three to select difficulty.

Button	Action	Dimensions		Position	
		X	Y	X	Y
Menu	Displays the main menu	50px	50px	5/6 window width	5/6 window height
1	Runs the game with difficulty set to 1	100px	100px	1/4 window width	1/2 window height
2	Runs the game with difficulty set to 2	100px	100px	2/4 window width	1/2 window height
3	Runs the game with difficulty set to 3	100px	100px	3/4 window width	1/2 window height

Figure 2 – Buttons to be displayed on the difficulty selection screen

The program waits here until a button is clicked or the window is closed.

The Game Screen

The game screen shall always include:

- The stickman holding a bow.
- The target.
- The current attempt number and the total number of attempts allowed,
- The player's current score.
- The wind value and direction.
- A button to return to the initial menu.

If the difficulty level is 3, an obstacle will also be included.

Object	Description	Dimensions	Position	
			X	Y
Stickman & Bow	The stickman	See Figure 5	5	5
Target		See Figure 4		
Attempt number	Text: "Attempt [attempt number] of [max attempts]"		5	5
Score	Text: "Score: [score]"		Centred	5
Wind indicator	Text: "Wind: [wind]" and an arrow indicating direction		9/10 window width	5
Menu button	Stops the game and takes the player to the start menu	75x25px	5	25

Figure 3 – Objects to be displayed while playing the game.

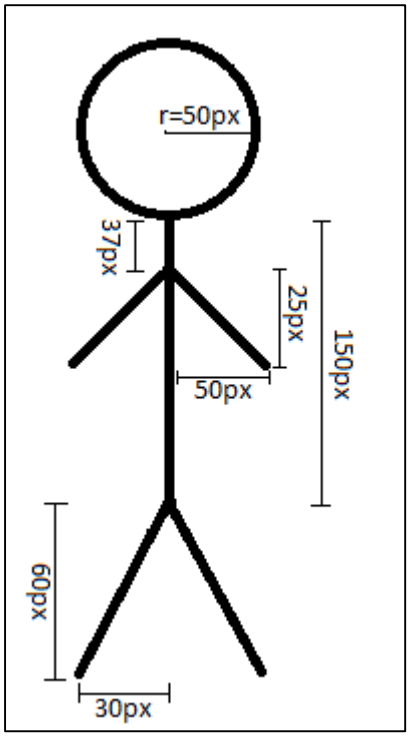


Figure 5 – Stickman Dimensions

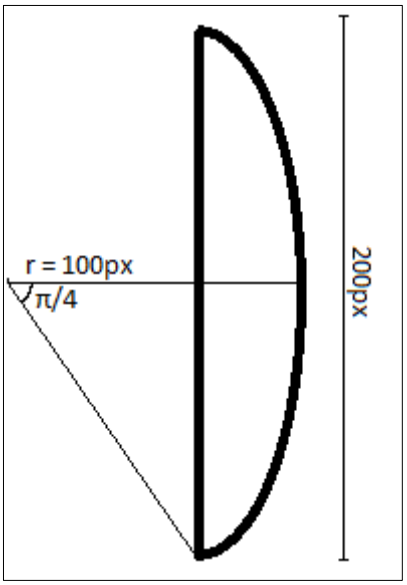


Figure 4 – Bow Dimensions

If the difficulty level is 3, an obstacle will be randomly generated between the stickman and the target. If the difficulty level is greater than 1, a wind will also be generated randomly. The wind may act with or against the motion of the arrow.

To determine the speed and direction in which the arrow is fired the user will click, drag the mouse and release. As the mouse is dragged around the window, the direction the bow is pointing and the amount the string has been pulled back will update to provide a visual indicator to the user as to the direction the arrow will be fired.

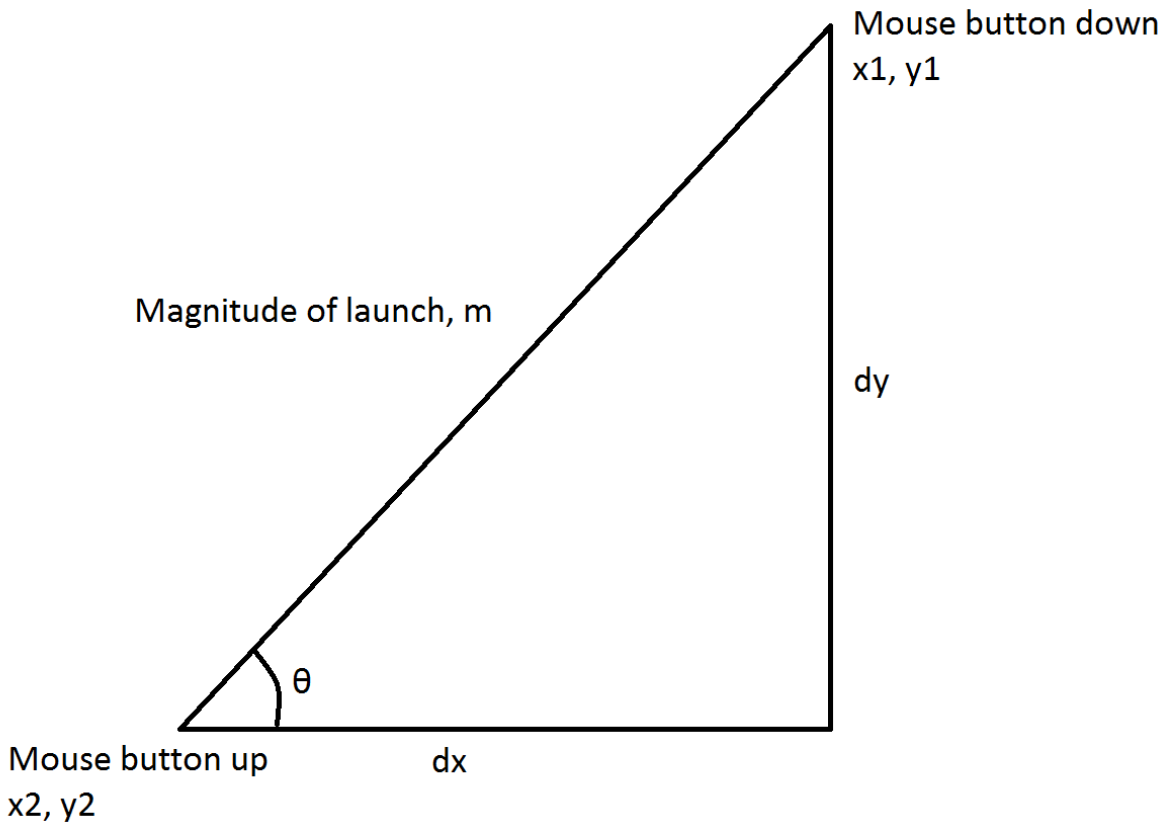


Figure 6 – Calculating angle and magnitude of the mouse drag.

$$m = \sqrt{dy^2 + dx^2}$$

$$\theta = \tan^{-1}\frac{dy}{dx}$$

Upon releasing the mouse button, the arrow will be fired. To calculate the trajectory of the arrow I will use $s = ut + \frac{1}{2}at^2$ to calculate the horizontal and vertical displacement at any given time.

$$x = x_0 + u_x t + \frac{1}{2}a_{wind}t^2$$

$$y = y_0 + u_y t + \frac{1}{2}a_{gravity}t^2$$

Where x_0 and y_0 correspond to the position from which the arrow was fired. To make the effect of gravity significant enough I will increase it by a factor of 10 to give -98.1 m/s^2. Likewise to make the effect of wind significant its possible range should at least include $\pm 100\ m/s^2$. u_x and u_y will be the horizontal and vertical components of the magnitude.

As the arrow travels along its trajectory it will need to rotate such that it is always pointing in the direction of travel. To do this the program will find the gradient of the trajectory and use it to find the angle to the x-axis in which the arrow is travelling.

$$\frac{dx}{dt} = u_x + a_{wind}t$$

$$\frac{dy}{dt} = u_y + a_{gravity}t$$

$$\frac{dy}{dx} = \frac{u_y + a_{gravity}t}{u_x + a_{wind}t}$$

$$\theta = \tan^{-1}\frac{dy}{dx}$$

The positions of the tip and tail of any length arrow can then easily be found using;

$x = l\cos\theta$ and $y = l\sin\theta$.

For every interval of t, the program will also check whether the arrow has hit either the target or the obstacle. This is easily done by checking whether the arrow is within the bounds either of these as both the target and the obstacle will be rectangular.

Constants

Name	Value	Description
WINDOW_WIDTH	1260	Width of the window
WINDOW_HEIGHT	480	Height of the window
TICK_TIME_MILLIS	10	The time in milliseconds between each frame while the arrow is in flight.
MENU_BUTTON_WIDTH	300	Width of a button on the main menu
MENU_BUTTON_HEIGHT	100	Height of a button on the main menu
MENU_BUTTON_THICKNESS	1	Thickness of the menu button box border
LEVEL_SELECT_BUTTON_WIDTH	100	Width of a button on the level select screen
LEVEL_SELECT_BUTTON_HEIGHT	100	Height of a button on the level select screen
BACK_TO_MENU_BUTTON_WIDTH_HEIGHT	50	Height and width of the 'back to menu' button on the level select screen
HEAD_RADIUS	50	Stickman's head radius
ARM_LENGTH	50	Stickman's arm length

STICKMAN_THICKNESS	2	Thickness of line that make stickman
STICKMAN_POSITION_X	100	X position of stickman's head
STICKMAN_POSITION_Y	210	Y position of stickman's head
BOW_BEND	100	Radius of the bow curve
BOW_LENGTH	200	Length of the string of the bow
BOW_MAGNITUDE_SCALE_FACTOR	0.1	Scale factor to keep bowstrings on screen
SHAFT_LENGTH	100	Length of an arrow
SHAFT_THICKNESS	2	Thickness of the arrow shaft
BARB_LENGTH	15	The length of the barb at the tip and the length of the fletchings
BARB_ANGLE	0.5	The angle the barb takes to the shaft in radians
MAX_ATTEMPTS	3	The max number of attempts per round
OBSTACLE_COLOUR	BLUE	The colour of the obstacle
MAX_OBSTACLES	2	Maximum number of obstacles
NUMBER_OF_TARGET_DIVISIONS	10	Number of divisions on the target
TARGET_THICKNESS	2	Thickness of the line composing the target
TARGET_LABEL_SHIFT	5	How far to the right the labels are shifted
TARGET_DEFAULT_X	200	Default x position of the target
TARGET_DEFAULT_Y	9*WINDOW_WIDTH/10	Default y position of the target
TARGET_DEFAULT_SIZE	200	Default height of the target
GROUND_LEVEL	420	Y coordinate of the ground level
SOLIDS_TARGET	1	The obstacle type is a target. Used to find out what the arrow hit.
SOLIDS_OBSTACLE	2	
SOLIDS_MISS	3	
ACCEL_GRAVITY	-9.81*10	The acceleration due to gravity used to calculate the arrow trajectory
MAX_WIND	100	Maximum allowed wind value
MIN_WIND	-100	Minimum allowed wind value

Figure 7 – Table of constants

Structure

The High Level Structure

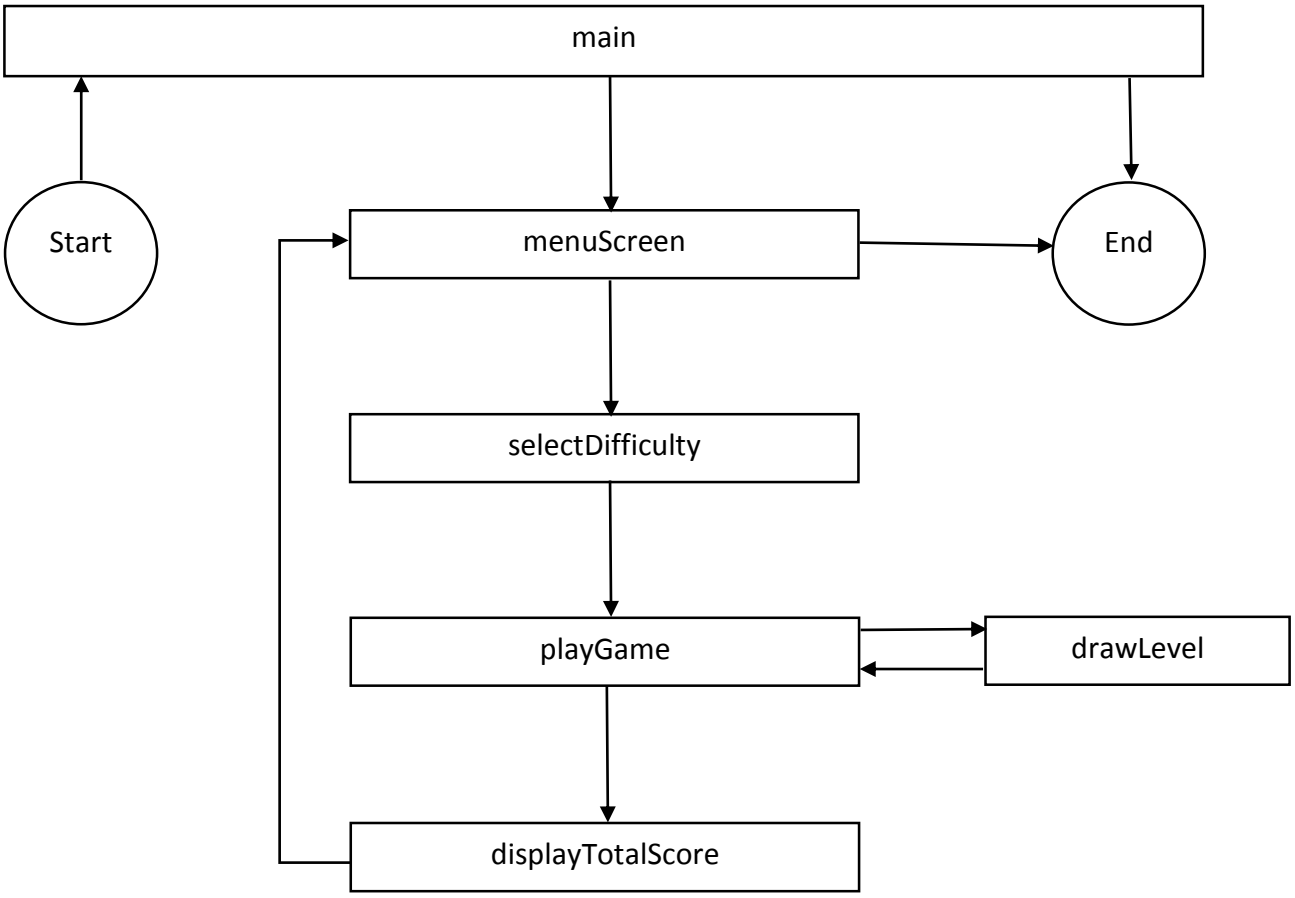


Figure 8 – Flow of the high level functions

Function	Arguments	Returns	Description
main	-	Exit type (int)	The main entry function
menuScreen	-	-	Draws the menu screen and waits for the user to click a button.
selectDifficulty	-	-	Draws the difficulty select screen and waits for user to select a difficulty by clicking a button.
playGame	int difficulty	-	Calls functions required to play the game. ie draws the level, gets the input, calls function to draw trajectory.
drawGame	struct target* t, struct box* obstacle, int* wind,	-	Draws the level. It gets passed the position to draw elements which may have moved eg. The arrow

	int* attemptNumber, int* score, double* bowAngle, double* bowMag		
displayTotalScore	Int* score	-	Displays the overall score for the round and draws a button to take the user back to the menu screen.

Figure 9 – Table of high level functions

Low Level Functions

Function	Arguments	Returns	Description
button	int leftX, int topY, int rightX, int bottomY, int borderThickness, char text[], void (*onClick)()	-	Draws a button and checks whether it has been clicked since its last call. If it has been clicked then *onClick()* is called.
getMouseDrag	double* angle, double* magnitude, target* t, box* obstacle, int* wind, int* attemptNumber, int* score	Int mouseDragged	Gets the distance the mouse was dragged across the screen and the lines angle to the x-axis while updating the game to that the bow updates dynamically.
finish	-	-	Closes the window and mouse then stops the program.
hasCollided	int* x, int* y, box* solids[], int length, int* collidedWith	int collided	Checks to see if **x, *y* are inside any of the boxes defined in *solids[].* If they are, it sets *collidedWith* to *box.type* and returns 1;
findScore	int* x, int* y, target* theTarget	int score	Finds out where on the target **x*, **y* are and returns the corresponding number of points.
fireArrow	double* angle, double* magnitude, int*	-	Calculates the trajectory of the arrow and draws it in its position for time *t*

	horizontalAcceleration, double t, int xStart, int yStart, int* x, int* y		
drawStickman	int xPos, int yPos, float scale, int color, double* angle, double* mag	-	Draws a stickman holding a bow which points in the direction set by **angle* with the string draw back by **mag*
drawBow	double* angle, double* magnitude, int x, int y	-	Draws the bow pointing in a direction set by **angle* and with its string draw back by **mag*
drawArrow	double* angle, int* tipXPos, int* tipYPos, int shaftLength	-	Draws the arrow pointing in a direction set by **angle*
drawTarget	int xPos, int yPos, int length	-	Draws the target.
Flush_event-queue	-	-	Flushes the event queue declared in graphics_lib.h
event_mouse_button_up_and_coords	int* x, int* y	int eventOccured	Returns 1 if the last event in the queue indicated the mouse button being released.
event_mouse_button_down_and_coords	int* x, int* y	int eventOccured	Returns 1 if the last event in the queue indicated the mouse button being pressed.

Figure 10 – Table of low level functions.

Algorithm and Function Structure Design

Main

The primary purpose of main is to initialise variables that will be needed throughout the program.

```
Function main
      Initialise a window of size WINDOW_WIDTH, WINDOW_HEIGHT
      Initialise a font
      Initialise the mouse
      Call create_event_queue
      Call ref_mouse_events
      Call reg_display_events
      Call menuScreen
      Call finish
      Return 0
End function main
```

PlayGame

Variable	C-Type	Range	Description
solids[MAX_OBSTACLES]	struct box*		A list of objects the arrow could collide with
Attempt	Int	1 – 3	Number of attempts user has had
Score	Int	0 – 27	Cumulative Score
Angle	Double	0 – π/2	The current angle of the bow
Mag	Double		The magnitude of the drag
collidedWith	Int	1-3	What the arrow hit
Wind	Int	MIN_WIND – MAX_WIND	The wind value for this round.
theTarget	struct target		The target for this round
obstacleBuilt	int	0-1	Has the obstacle been generated yet?
windGenerated	int	0-1	Has the wind been generated yet?

```
Function playGame
      Call flush_event_queue
      Declare an array to hold anything arrow can collide with
      Declare the values for the target
      Loop while player hasn't used all their attempts
            Clear the screen
            If difficulty == 3
                  Generate obstacle
            End if
```

```
            If difficulty > 1
                  Generate a wind
            End if
            Call drawGame
            If getMouseDrag
                  Loop while not hasCollided
                        Clear the screen
                        Call fireArrow
                        Call drawGame
                        pause for TICK_TIME_MILLIS

                  End loop
                  If arrow hit target
                        Call findScore
                        Add points to score
                  End if
                  Increment the number of attempts player has had
            End if
      Update the display
      Wait for next event
End Function
```

Event_get_moue_button_up_and_coords, Event_get_moue_button_down_and_coords and flush_event_queue

The functions, event_get_moue_button_up_and_coords, Event_get_moue_button_down_and_coords and flush_event_queue will be wrappers for allegro.

Event_mouse_button_down_and_coords :

```
Function event_mouse_button_down_and_coords
      IF last event on queue was ALLEGRO_EVENT_MOUSE_BUTTON_DOWN
            Set *x to mouse X coordinate
            Set *y to mouse Y coordinate
            Return 1
      End if
      Return 0
End function
```

Event_mouse_button_down_and_coords will be the same as event_mouse_button_down_and_coords but will return 1 if the last event on the queue was ALLEGRO_EVENT_MOUSE_BUTTON_UP.

Flush_event_queue:

Will call al_flush_event_queue and pass it *event_queue* as defined in graphics_lib.h to clear the event queue.

Implementation Report

File Structure

main.c

Type	Name
Function	main

assignment_graphics.h

Type	Name
Symbol	NUMBER_OF_TARGET_DIVISIONS
Symbol	TARGET_THICKNESS
Symbol	TARGET_LABEL_SHIFT
Symbol	MENU_BUTTON_WIDTH
Symbol	MENU_BUTTON_HEIGHT
Symbol	MENU_BUTTON_THICKNESS
Symbol	STICKMAN_POSITION_X
Symbol	STICKMAN_POSITION_Y
Symbol	GROUND_LEVEL
Symbol	BOW_BEND
Symbol	BOW_LENGTH
Symbol	BOW_THICKNESS
Symbol	BOW_MAGNITUDE_SCALE_FACTOR
Void Function Pointer (void)	VoidFunctionVoid
Function Prototype	Button
Function Prototype	drawStickman
Function Prototype	drawTarget
Function Prototype	drawBow
Function Prototype	drawArrow

Functions for the prototypes defined in assignment_graphics.h are set in assignment_graphics.c.

displays.h

Type	Name
Symbol	LEVEL_SELECT_BUTTON_WIDTH
Symbol	LEVEL_SELECT_BUTTON_HEIGHT
Symbol	MAX_OBSTACLES
Symbol	TARGET_DEFAULT_X
Symbol	TARGET_DEFAULT_Y
Symbol	TARGET_DEFAULT_SIZE
Symbol	MAX_WIND
Symbol	MIN_WIND
Function Prototype	drawGame
Function Prototype	selectDifficulty
Function Prototype	difficulty1
Function Prototype	difficulty2
Function Prototype	difficulty3
Function Prototype	difficulty4
Function Prototype	menuScreen
Function Prototype	playGame
Function Prototype	displayTotalScore

Functions for the prototypes defined in displays.h are set in displays.c.

processes.h

Type	Name
Function Prototype	finish
Function Prototype	hasCollided
Function Prototype	findScore
Function Prototype	fireArrow
Function Prototype	getMouseDrag

Functions for the prototypes defined in processes.h are set in processes.c.

allegro_wrappers.h

Type	Name
Function Prototype	flush_event_queue
Function Prototype	event_mouse_button_up_and_coords

Function Prototype	event_mouse_button_down_and_coords

Functions for the prototypes defined in allegro_wrappers.h are set in allegro_wrappers.c.

constants.h

Type	Name
symbol	WINDOW_WIDTH
Symbol	WINDOW_HEIGHT
Symbol	TICK_TIME_MILLIS
Symbol	SOLID_TARGET
Symbol	SOLID_OBSTACLE
Symbol	SOLID_MISS
Symbol	MAX_ATTEMPTS
Symbol	ACCEL_GRAVITY
Symbol	SHAFT_LENGTH
Symbol	SHAFT_THICKNESS
Symbol	BARB_LENGTH
Symbol	BARB_ANGLE
Struct	target
Struct	box

Design changes

Very few design changes were made.

1. In function, *playGame():*

   ```
   box* solids[MAX_OBSTACLES];
   ```

   was changed to

   ```
   box** solids = malloc(MAX_OBSTACLES * sizeof(box*));
   ```

   The former caused a runtime error, presumably because the correct amount of memory was not being assigned for the elements of the array. The memory allocated to *solids* is freed later in *playGame()*.

Testing

Testing Inputs and Outputs

Testing the Buttons

Button	Effect of clicking	As Expected?
Play	Displays select difficulty screen	Yes
Exit	Stops the program	Yes
Difficulty 1	Plays the game with difficulty set to 1	Yes
Difficulty 2	Plays the game with difficulty set to 2	Yes
Difficulty 3	Plays the game with difficulty set to 3	Yes
Difficulty 4		
Menu from difficulty screen	Displays the menu screen	Yes
Go to menu from game	Displays the menu screen	Yes

However, flush_event_queue() was not working correctly, so upon clicking a button in a spot that was within a button on the next screen, it would think the second button had been clicked also.

Updates During the mouse drag

Secondly I will check that as the mouse is dragged, the bow and line update and upon releasing the button, the arrow is fired in the correct direction.

Result:

As expected, the bow updates to point in the correct direction as the mouse is dragged around the screen. Likewise the line indicating the line the mouse has been dragged across was also updated correctly. When the mouse button was released the arrow was fired in the correct direction.

Allowed angles

Thirdly I verified the range of angles through which the arrow could be fired. This will be done by dragging the mouse though around a circle and monitoring the direction the bow is pointing.

Result:

Mouse drag angle range (rad)	Range of angles the bow was able to point in (rad)	As Expected?
$0 - \frac{\pi}{2}$	$0 - \frac{\pi}{2}$	Yes
$\frac{\pi}{2} - \pi$	$\frac{\pi}{2} - 0$	Yes
$\pi - \frac{3\pi}{2}$	$0 - \frac{\pi}{2}$	Yes
$\frac{3\pi}{2} - 0$	$\frac{\pi}{2} - 0$	Yes

Collisions and Scoring

Finally I will check the program calculates whether the arrow has collided or gone out of bounds, and that the correct number of points is added to the score in each case.

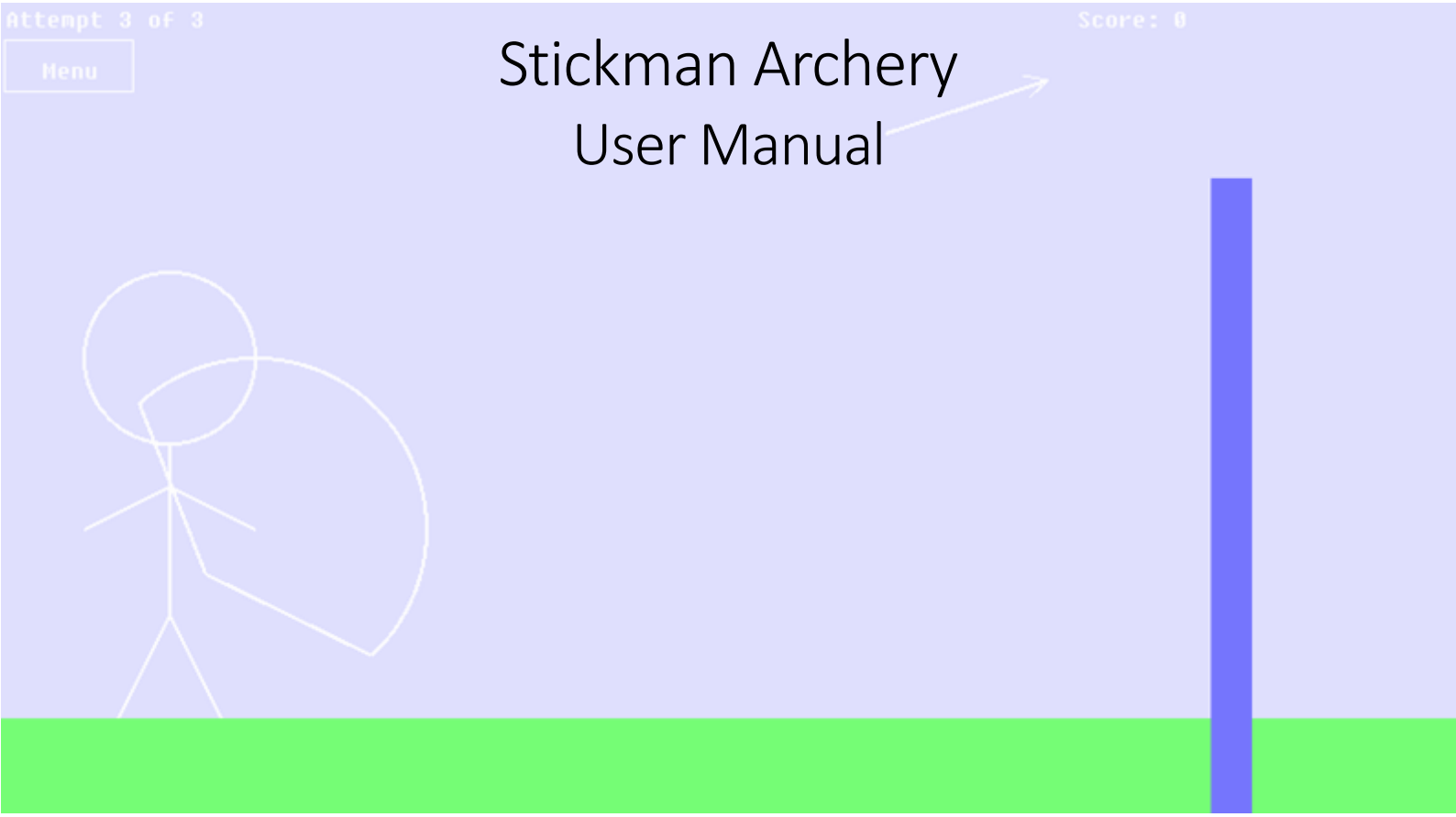
Result:

The program operated as expected by adding no points when the arrow hit an obstacle or went out of bounds. I also tested each region of the target to ensure it calculated the number of points to award correctly.

Changes made to the program after testing

To solve the issue of buttons being clicked automatically in some cases, I moved the wait_for_event() call to the beginning of their respective while loops. This has the slight downside of meaning the screen will not be re-rendered until an event occurs. However as simply moving the mouse is also considered an event, this slight delay is acceptable.

Copy of the User Manual



Stickman Archery

User Manual

Contents

Installation

You can install the program anywhere by copying the “Release” folder found in “bin” and pasting is where you would like to install the game.

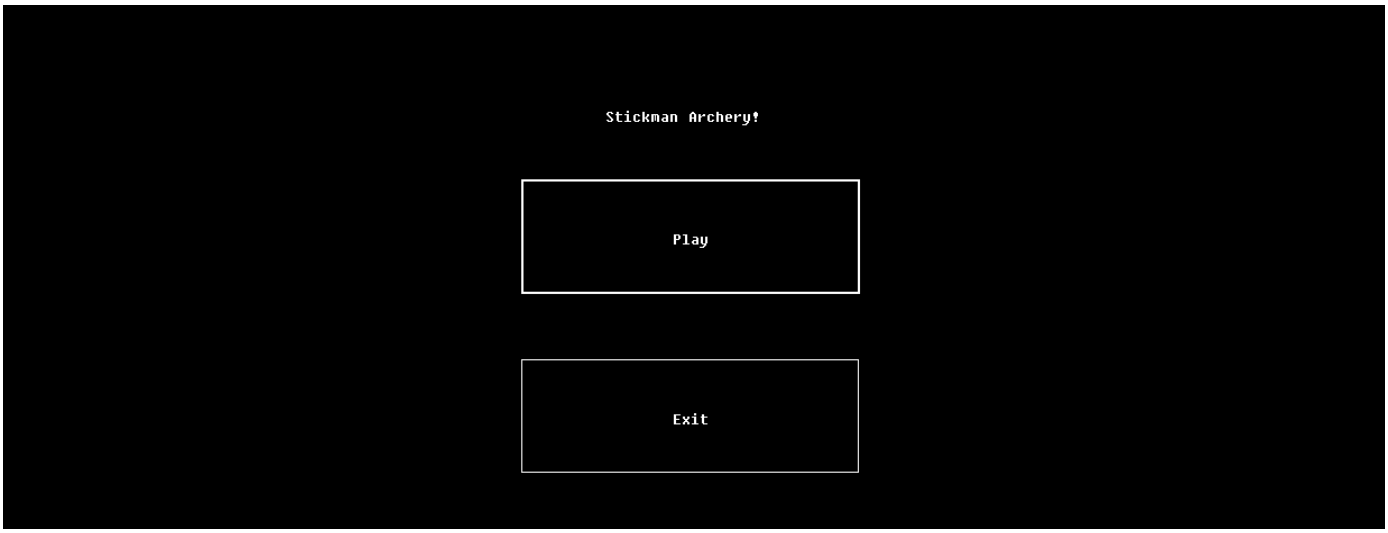
System Requirements

Tested Operating Systems:

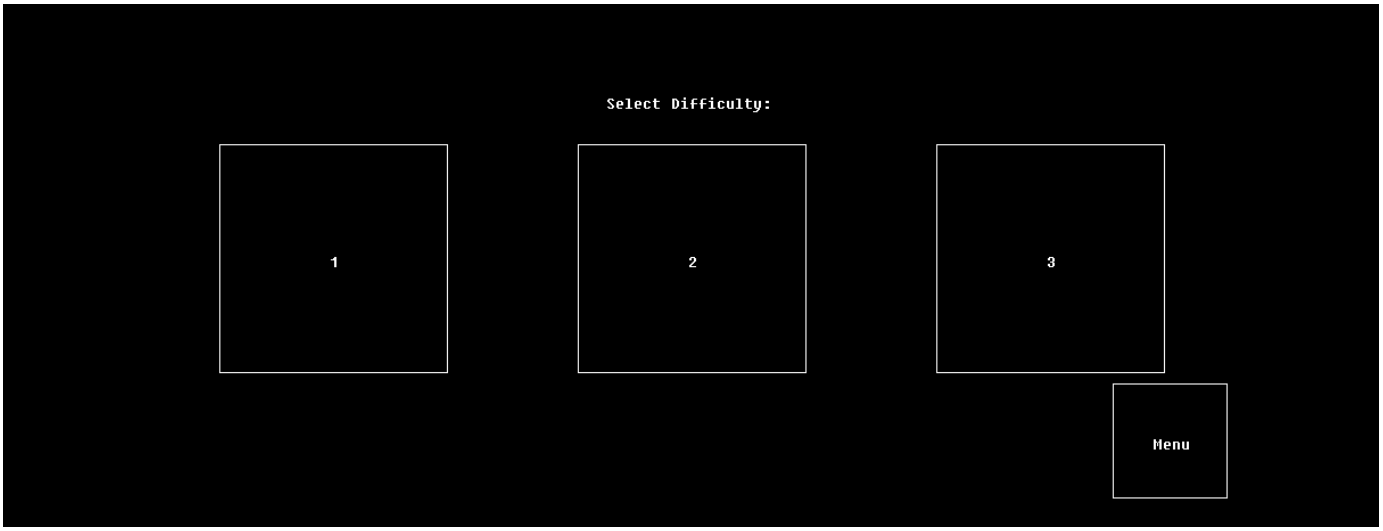
- Windows 7 (64-bit)
- Windows 7 (32-bit)

How to play

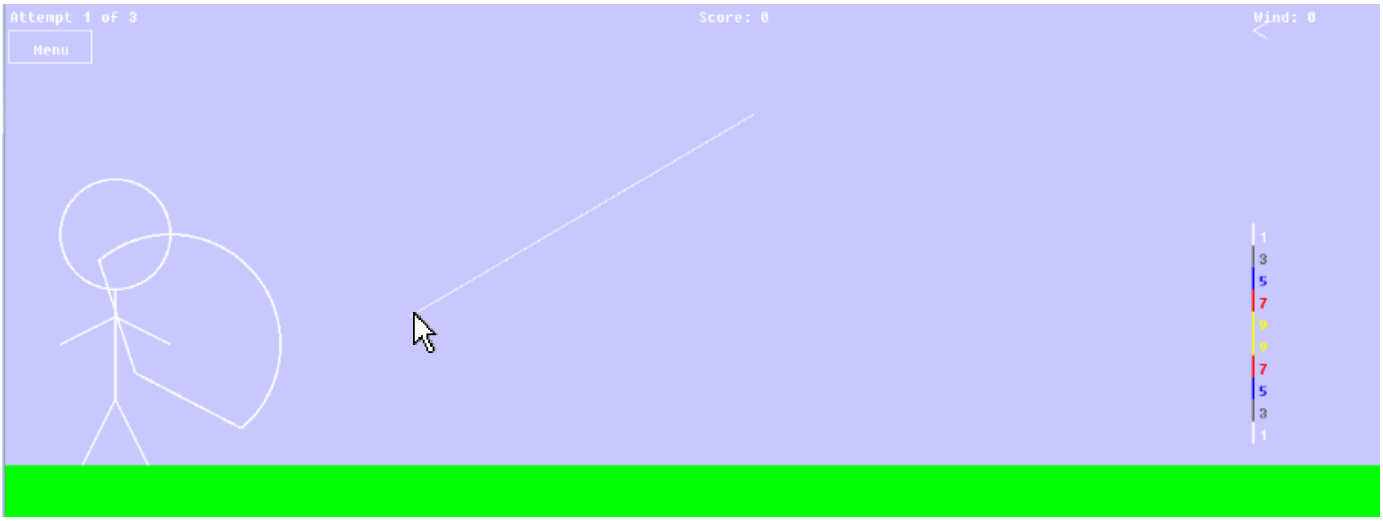
1. On the menu screen click “Play!”.



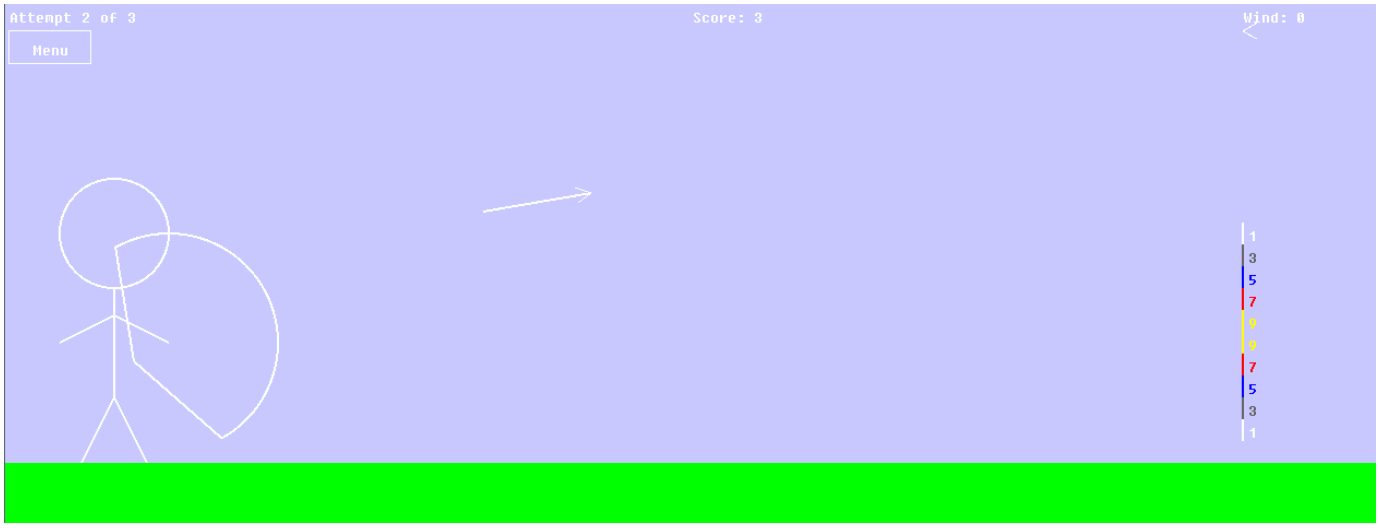
2. Select the difficulty level.



3. To shoot an arrow:
    a. Click and drag the mouse button in the direction you want it to go.
    b. To shoot further, drag the mouse back more.

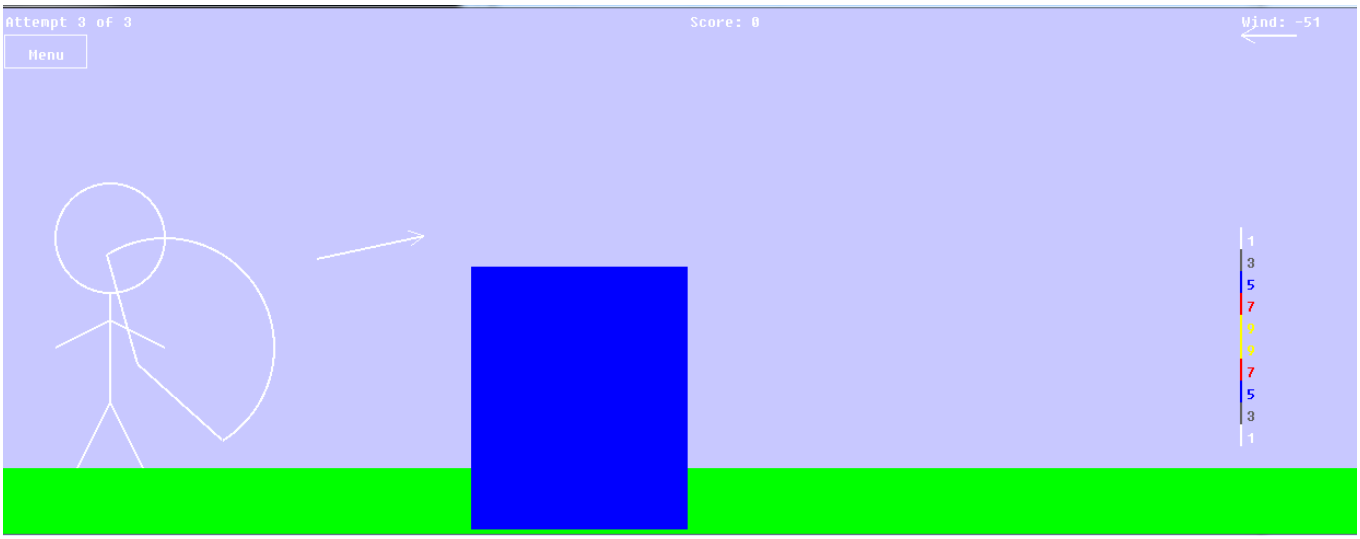


c. Release the mouse button to fire.



4. You get 3 goes per round.
5. You can return to the menu to change difficulty by clicking the menu button in the top left.

6. On harder difficulties, you will need to take into account the wind shown in the top right and avoid obstacles.



Difficulty Options:

1. This is normal mode with no extra things to take into account.

2. On difficulty 2, there will be a wind you must account for.
3. On difficulty 3, there will an object in the way as well as wind.

Frequently Asked Questions

Q. When I try to run the game, the window pops up for a moment, then closes.

A. Make sure you have copied the entire “Release” folder. Make sure it contains another folder called “data”.