**Project 2 - Axe game – PART 1**

**Line 1 -**

/\*Includes external "Raylib" Library. Uses .h and "" header files as it's an external library - not built into language and not part of c++ standard library.

<> - C++ standard library - "" External library - Extenal library requires .h header file\*/

**Line 2 + 3 -**

// Code entry point - Main function header and function body that = function definition. - Within body create statements (must end in;)

**Line 4 -**

/\*Create a float data type and named it root\_beer

Assigned the float a value of 1.99\*/

**Line 5-**

//Create a double data type and name it cheese\_burger - Assign value using braced initialisation

Line 6 -

//Create a boolean data type and name it shouldhavelunch - Assign value using braced initialisation

**Line 8 -**

/\*Created an integer variable called Width and assigned it a value of 1350. Integer type in C++ is called int and holds an integer value.

Declare value with int and name it. Use assingment operator (=) and assign a valuable to the integer\*/

**Line 9 -**

/\*Passed width into initwindow as the first argument\*/

**SUMMARY -**

/\*

\*Axe\_game.cpp

\*Created a main function and included the Raylib extension library

\*Declared a variable called "width"

\* Assigned the variable a value of 1350

\*Created a popup window using the value of "width" as it's first agument as well as a height and named it "warren's window"

\*Created different data types (float, double and boolean) and assigned them a value on a seperate line similar to how I've done for "width"

\*I then assigned them all on the same line using 2 different methods "assignment" and "Braced inistialisation"

\*I learned how to create breaking points to pause execution at specified points and how to use debugging to find errors and process in code

\*Learned to use step into to advance to next line in the code.

\*Learned to hover mouse over variables and observe their values before and after execution

\*Learned that variables contain garbage data before being initialised.

\*/

**PART 2 –**

**Line 7 –**

Run a Boolean search for equal to – used numbers 4 and 9 – value of false  
**Equal symbol - ==**

**Line 9 –**

Run a Boolean search for not equal to – used numbers 4 and 9 – value of True  
**Not Equal symbol - !=**

**Line 1 –**

Run a Boolean search for Less than – used numbers 4 and 9 – value of True  
**Less than symbol - <**

**Line 13 –**

Run a Boolean search for greater than – used numbers 4 and 9 – value of false  
**Greater than symbol - >**

**Line 15 –**

Run a Boolean search for Less than or equal – used numbers 4 and 9 – value of True  
**Less than equal symbol - <=**

**Line 17 –**

Run a Boolean search for greater than or equal – used numbers 4 and 9 – value of True  
**Greater than equal symbol - >=**

**SUMMARY –**

Learned about different comparison operators - Comparison operators compare 2 values and return a boolean

Learned about expressions - An expression is a combination of operators and operands (the things that operators act on)

An expression will return a value.

**Comparison Operators:**

**Equal symbol - ==**  
**Not Equal symbol - !=**  
**Less than symbol - <**  
**Greater than symbol - >**   
**Less than equal symbol - <=**  
**Greater than equal symbol - >=**

Ran a Boolean search for each comparison operator using a value of 4 – 9

**Part 3 –**

**Line 12, 13, 17 –**

Created an infinite loop – See notes on Loops or summary if you need a refresher.   
Used a “while loop” executes a block of code repeatedly as long as a certain condition is true. – Made the condition for the loop true using the below:

while (true)  
{  
}

**SUMMARY –**

Learned about loops.

Loops repeat code until a condition is met

This is called the loop condition

Loop condition is a boolean expression which are used to repeatedly execute a block of code while a certain condition is true.

There are three main types of loops in C++:

* **for loop**: executes a block of code a fixed number of times, based on a counting variable.
* **while loop:** executes a block of code repeatedly as long as a certain condition is true.
* **do-while loop:** similar to a while loop, but executes the block of code at least once before checking the condition.

Loop condition is check for truth value - if true - Loops body get's to execute so all statements inside the body get to run.

once complete the loop condition is revaluated and runs again - continues over and over again until loop condition evaluates to false

**Part 4 –**

**Lines 14, 15, 16**

Used begin drawing and end drawing functions from raylib to setup canvas so it’s drawn properly  
Used Clearbackground to clear the canvas to the red colour.

**Summary –**

Used begin drawing and end drawing functions from raylib to setup canvas so it’s drawn properly  
Start and end drawing use the setup and teardown of the window.   
**BeginDrawing();  
EndDrawing();**

Clear background clears the canvas to a certain colour.  
Have to use clear background as this avoids flickering which is a result of Raylibs double buffering method to update image in the canvas for the window.  
**ClearBackground(red);**

**Part 5 –**

**Lines 12 –**

Created a function return value if equal to false – loop runs until X key or ESC are pressed.

**Summary –**

Function Return values – Learned a function can return a value to the part of the program that called it. This value can be used for further calculations or actions in the program. In this case I used the windows should close function with () and used the == (equal to) operator to check if windows should close return false. Will return false until the X key or ESC key is entered.

**while (WindowShouldClose() == false)**

**Part 6 -**

**Lines 14, 15, 16, 17**

Added new line below 15 – Added Draw circle functions – gave it an x & Y axis of 960/540 and a radius of 100 – coloured it clue

**SUMMARY –**

Learned I can use **DrawCircle()** to draw a circle on the screen and use coordinates to place the circle.   
I then placed the circle at the centre of the screen.

**Part 7 -**

Lines 16 – 40

Create a circle using an if variant and the **IsKeyDown** which is an int function – added +10 to the x+Y axis of the button press to let the circle move in the pop up window.

**SUMMARY –**

Learned the if statement can be used to check a Boolean expression and run code if the expression evaluates the true.   
Used the Boolean expression to detect whether or not pressing specific keys  
Used **IsKeyDown()** raylib function to check to see if pressing WASD keys using the  
**Key\_W, Key\_A, Key\_S, Key\_D** variables that are built into raylib

**Part 8 -**

Lines 22 – 38

In the if (iskeydown) Sections added **&& circle\_x < 1920** or **&& circle\_y > 0** to create a boundary for the circle on screen on the 1920/1080 diameters.

**SUMMARY –**

Learned about the logical and operator - used to check truth values of the expression to the left and the expression to the right - returning true if both expressions evaluate to true.

Use the AND operator to add boundaries to the circle.

**Part 8 -**

Lines 16 – 53

Set axe coordinates use int axe\_x/Y.   
Drew in a rectangle using those coordinates.   
move it off screen and changed the circle code to use Compound assignment operators

**SUMMARY –**

Learned how to draw a rectangle into the code.   
Set coordinates for the rectangles start point.   
Moves the rectangle using the same method as the circle.   
Learned about Compound assignment operators += and -=   
used these as a short hand when adding values to the variables.

**Example –**   
When moving the axe  
**Axe\_x = Axe\_x + 10;** can be shortened to **Axe\_x += 10;**

And when moving the opposite way **Axe\_x = Axe\_x - 10;** can be shortened to **Axe\_x -= 10;**

**Part 8 -**

Lines 16 – 62

Added an integer for direction and added a value of 10 – Assigned the axe to move up and down depending on if it’s greater mor less than the boundary.

**SUMMARY –**

Learned about the Logical OR operator. - ||  
Learned that logical Or returns true if either the left or the right expressions are true.  
Used the or operator to detect when the app hits the edge of the screen and then changed directions using the sign of the direction variable.

**Part 9 -**

Lines 12 – 71

Added Circle and axe edges  
Created an Int (value) for the circle radius EG – Int Circle\_radius{100};  
applied the circle radius to the circle edge coordinates and Draw circle command  
shorten axe and movement code  
Used the Logical OR operator to loop the axes movement up and down.  
Changed IskeyDown circle boundries to assigned values

**SUMMARY –**

Learned about Collision detection works.   
Learned that we had to use the dimensions of the circle radius to detect if the circles outer edges collide with a particular line.  
Learned that we can use the **dimensions of** the axe length to detect if the axe outer edges collide with a particular line.  
Added circle and axe edges by setting an INT value   
**For example**  
**int L\_axe\_x[axe\_x];  
int r\_axe\_x{axe\_x + axe\_length};  
int u\_axe\_y{axe\_y};  
int b\_axe\_y{axe\_y + axe\_length};**

Used compound operator to shorten axe and movement code and ssed the Logical OR operator to loop the axes movement up and down.  
**For Example**  
**axe\_y += direction;  
if (axe\_y > height || axe\_y < 0)  
{  
 direction = -direction;  
}**  
This means the axe will move by a value of 10 (direction) it's higher than the top, it will reverse direction and go down or if it’s lower than the bottom, it will reverse direction and go up. This is much cleaner than the below example.

**if (axe\_y > 1080)  
{  
Direction = -Direction;  
}  
If (Axe\_y < 0)  
{  
Direction = -Direction;  
}**

I also changed the values of the circles width and height to the assigned value name instead of the actual number. This is to avoid **hard coding (using the number instead of the assigned value)**   
This means now that when I change the int with that value in to a different figure, I don’t have to go back through the code and change the numbers as well.

**Part 10 -**

Lines 35 – 80

Created a Boolean search called collision with axe  
used an IF/ELSE statement  
Wrote a game over screen  
Moved game logic into the Else statement

**SUMMARY –**

Learned about the ELSE statement and how it ALWAYS goes with the IF statement.  
Learned about the draw text function which draw’s text to screen  
Created a Boolean called Collision with axe – when true in the if body we print game over using draw text  
When false it draws the circle and the AXE using our already existing game logic.   
Moved game logic into the else section of the Boolean.

**Part 11 –**

Lines 35 – 103

Updated Boolean initialisation and assigned the collision condition.   
Updated circles edge variables to change with each frame by adding them to the while loop.  
Updated axe edge variables to change with each frame by adding them to the while loop.  
Fixed code issues where I set [] instead of {} for axe\_x

SUMMARY –

Learned to place the set coordinates for axe and circle edges as well as the Boolean inside the while loop so that they can update with each frame.   
Used the && operator to give the collision with axe the correct value.  
created a game over state for the game.