VENDING MACHINE IN C#

Sergiusz Pieszak

30111714@students.southwales.ac.uk

Contents

Introduction

This report details the development and testing of a Visual Studio C# program designed to simulate a vending machine. It covers both the design and development processes, utilizing pseudocode and flowcharts to guide the implementation of the application. Additionally, the report includes comprehensive testing phases that revealed vulnerabilities and flaws in the code, ensuring that the program not only functions correctly but also meet all specified requirements, ultimately providing a reliable user experience.

Design

A basic graphical user interface (GUI) was designed (*see figure 1*) with a strong emphasis on enhancing user experience while aiding the development of a flowchart and pseudocode. An "About Us" section was incorporated to provide new customers with essential background information about the application and its purpose. To avoid copyright issues, an AI-generated background image, DeepAI (2024), was selected, ensuring a unique and visually appealing interface. The chosen theme for the GUI is a gothic Halloween style, which creates an engaging and immersive atmosphere for users, further enhancing their overall experience. The layout is intentionally designed for ease of use:

- A money panel is positioned at the bottom of the screen, allowing users to easily view their balance and available funds.
- The payment box is located close to the money panel, facilitating a convenient drag-and-drop functionality for users to quickly make transactions.
- The drink selection area is centred on the screen, simulating the layout of items inside a vending machine, making it intuitive for users to navigate their options.
- An order list box is situated on the right side of the machine, clearly displaying the items that the user has selected, allowing for easy review and confirmation before payment.

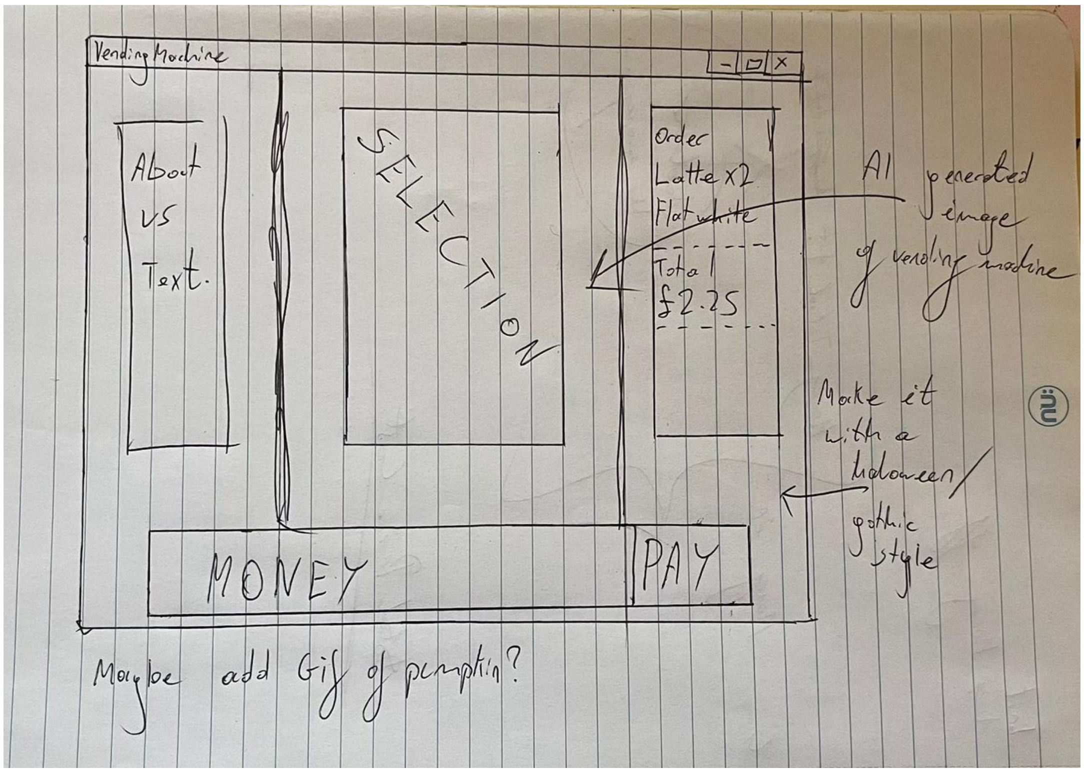

Vending Machine
About us Text.
SELECTION
Order
Latte x2
Flat white
Total
£2.25
AI generated image of vending machine
Make it with a halloween/gothic style
MONEY
PAY
Maybe add Gif of pumpkin?

Figure 1

Flowchart

A flowchart was used to provide a top-down view of the program, breaking it into manageable parts and clarifying the flow of control and function interactions. By visualising the entire process, the flowchart reduced obfuscation, ensuring that each step and relationship in the program was easy to understand and follow. This clarity allowed for more straightforward development and debugging, especially when writing more in-depth pseudocode making it easier to identify issues early and improve code readability and maintainability. As you can see in *figure 2* a rough concept for the vending machine was drafted that helped design the initial flowchart.

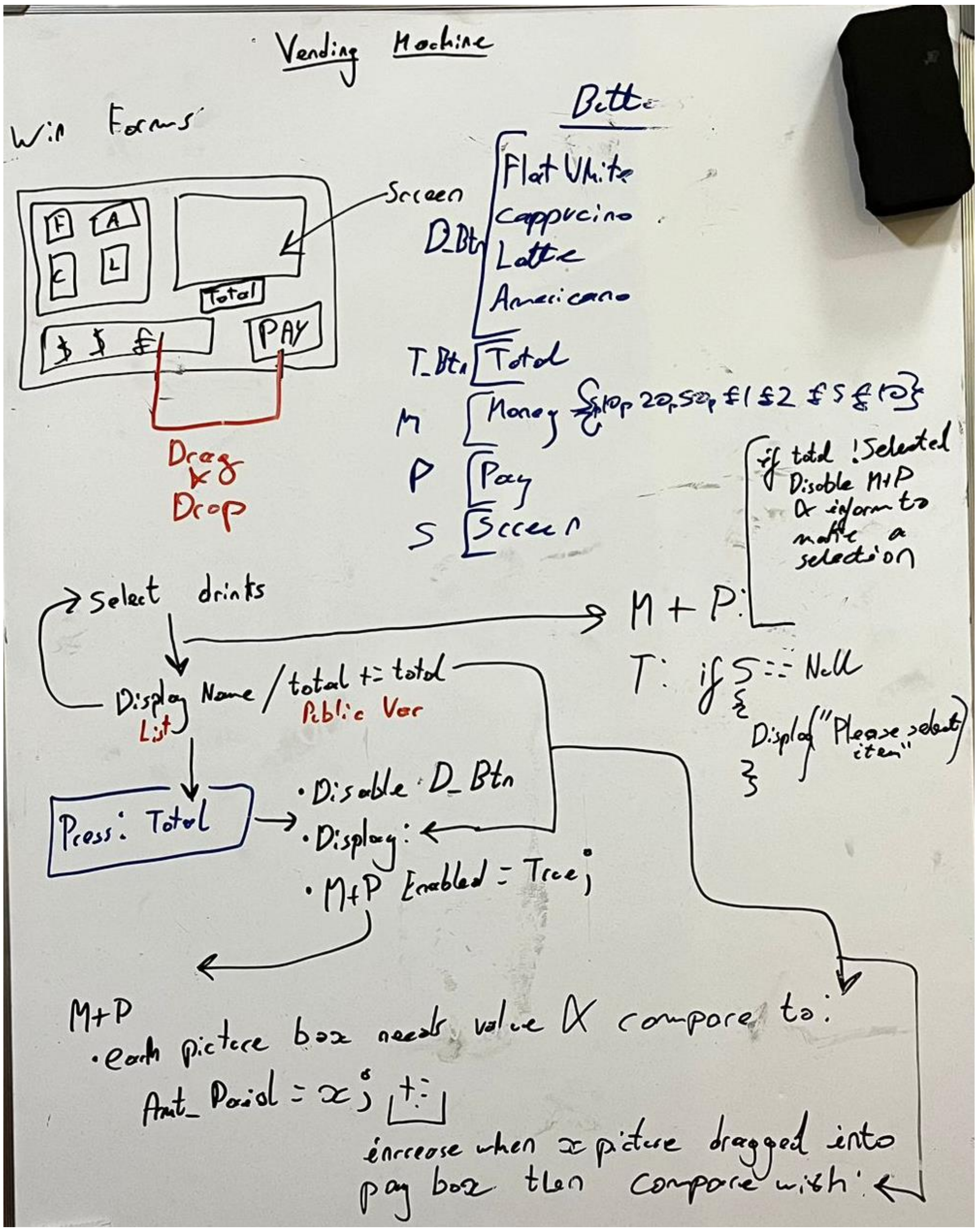

Vending Machine
Win Forms
Butt
F
A
C
L
Screen
Total
$ $ £
PAY
Drag & Drop
D_Btn
Flat White
Cappucino
Latte
Americano
T_Btn Total
M Money {10p 20p 50p £1 £2 £5 £10}
P Pay
S Screen
if total !Selected Disable M+P & inform to make a selection
M + P:
T: if S == Null { Display("Please select item") }
Select drinks
Display Name / total += total
List
Public Var
Press: Total
• Disable D_Btn
• Display:
• M+P Enabled = True;
M+P
• each picture box needs value & compare to:
Amt_Paid = x; +:
increase when x picture dragged into pay box then compare with:

Figure 2 – the initial concept

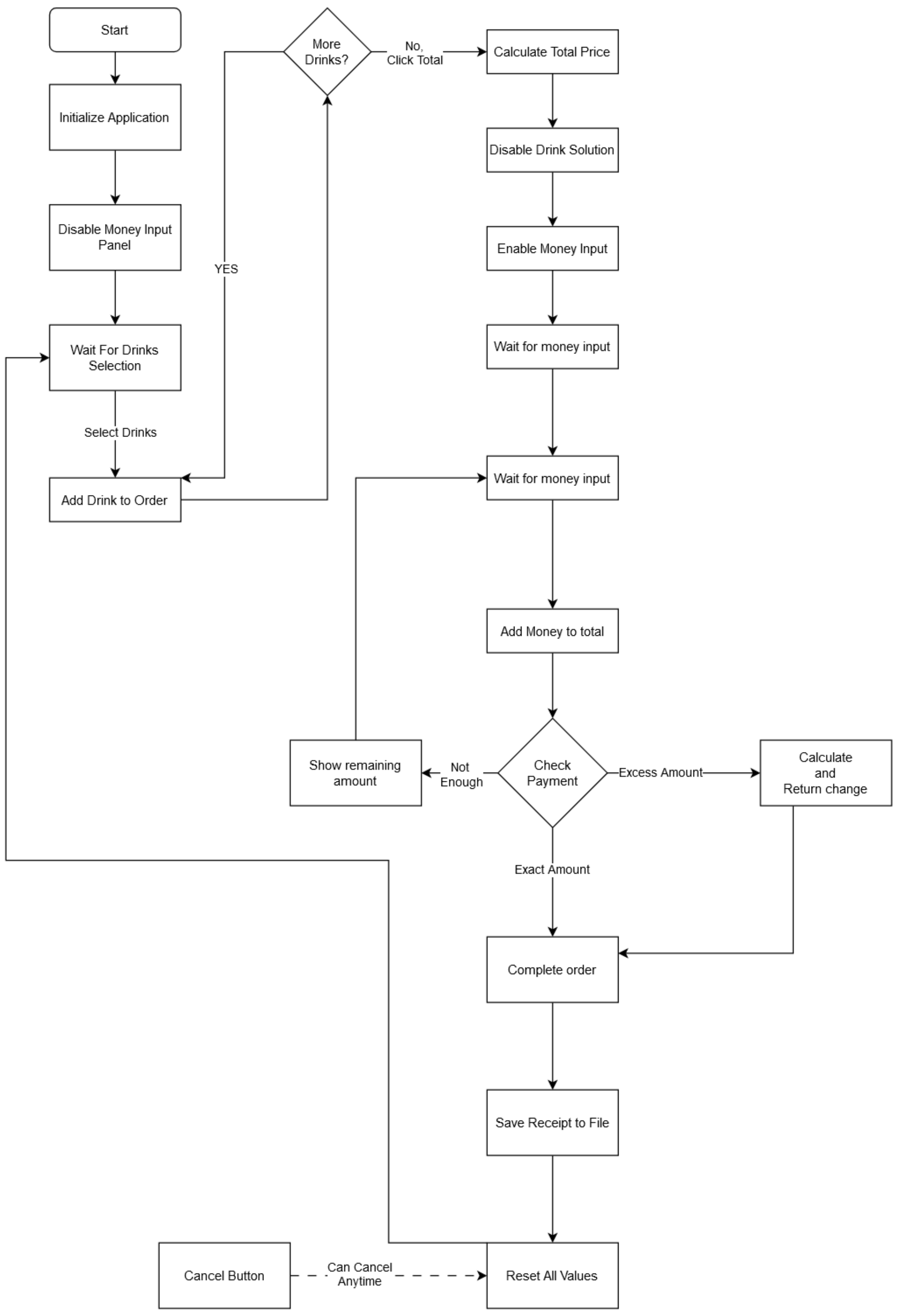

Start
Initialize Application
Disable Money Input Panel
Wait For Drinks Selection
Select Drinks
Add Drink to Order
YES
More Drinks?
No, Click Total
Calculate Total Price
Disable Drink Solution
Enable Money Input
Wait for money input
Wait for money input
Add Money to total
Check Payment
Not Enough
Show remaining amount
Excess Amount
Calculate and Return change
Exact Amount
Complete order
Save Receipt to File
Cancel Button
Can Cancel Anytime
Reset All Values

Figure 3

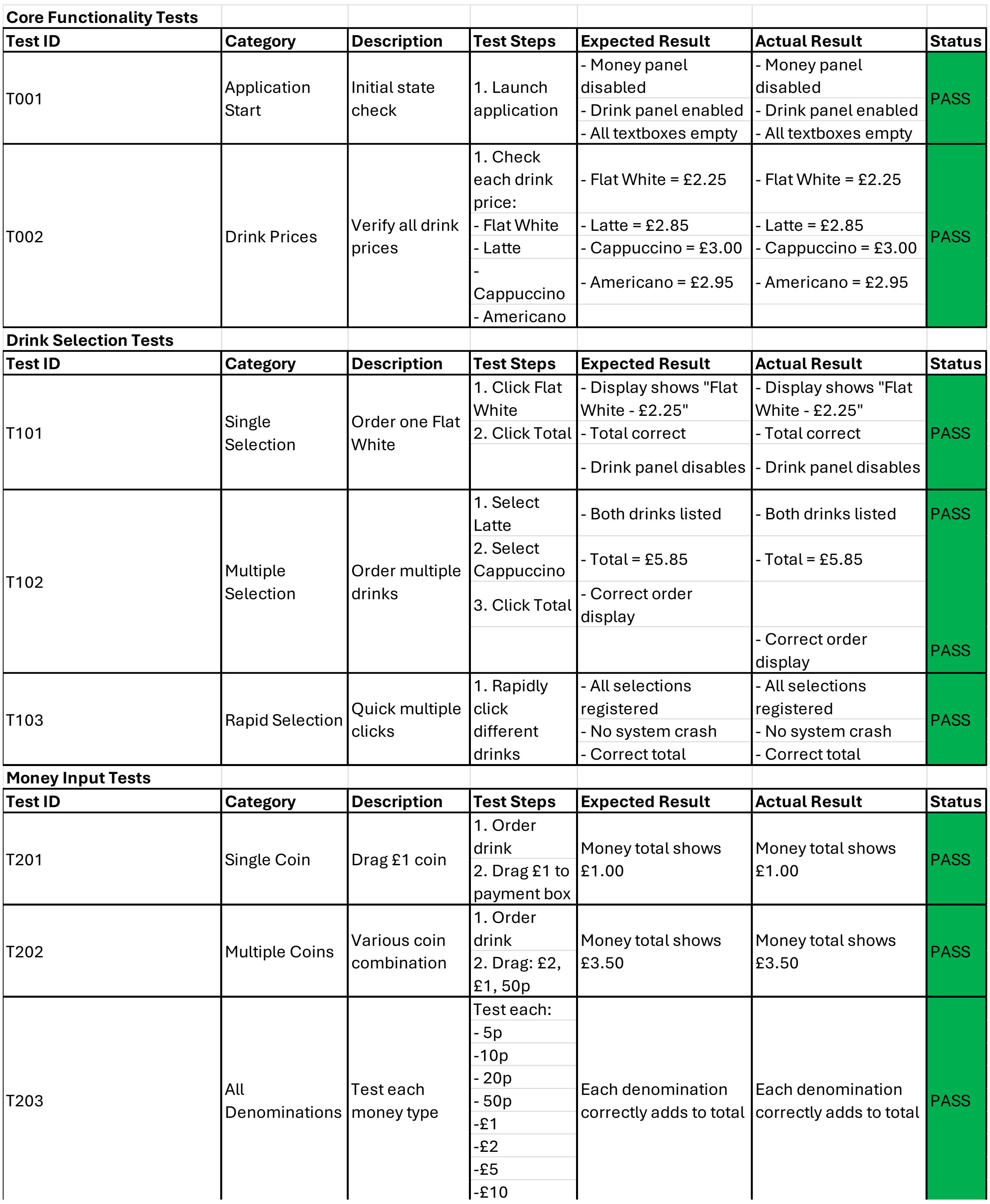
Pseudocode

After creating a basic flowchart, all necessary variables were identified and outlined then outlined program logic in pseudocode. This step provided a clear structure, helping to organise the variables and their interactions before moving to code implementation.

```
Initialize Application:
    - Setup form components
    - Enable drag-and-drop for paymentBox
    - Disable moneyPanel controls initially
    - Set initial quantities in drinkQuantities to 0
    - Call UpdateListBox() to initialize list display

Global Variables:
    decimal totalPriceCost
    decimal totalMoneyInput
    decimal totalChange
    Dictionary<string, int> drinkQuantities
    Dictionary<string, decimal> drinkPrices = {
        "Flat White": 2.25,
        "Latte": 2.85,
        "Cappuccino": 3.00,
        "Americano": 2.95
    }
    string lastAction = "Welcome! Please select your drinks."

Error Handling:
    Function errorMessage():
        Show error message dialog with "Please restart the machine."

Control Panel Function:
    Function buttonControlEnabled(int i, bool selection):
        If i == 0:
            Enable or disable drinkSelectionPanel based on selection
            Enable sum_btn
        Else if i == 1:
            Enable or disable moneyPanel based on selection
        Else:
            Call errorMessage()

Reset Function:
    Function resetValues():
        Set totalMoneyInput, totalPriceCost, totalChange to 0
        Clear main_listBox and moneyIn_txtBox
        Set all values in drinkQuantities to 0
        Enable drink selection, disable money panel
        Set lastAction to empty

Drag-and-Drop Operations:
    Function money_MouseDown(object sender, MouseEventArgs e):
        Start drag-and-drop for moneyBox

    Function paymentBox_DragEnter(object sender, DragEventArgs e):
        Set drag effect to copy

    Function paymentBox_DragDrop(object sender, DragEventArgs e):
        Get droppedCoin from e.Data
        If droppedCoin is money_5p:
            totalMoneyInput += 0.05
        Else if droppedCoin is money_10p:
            totalMoneyInput += 0.10
        ...
        Else:
            Call errorMessage()
        Update moneyIn_txtBox with totalMoneyInput formatted as currency
        Update lastAction with amount added

List Display Function:
    Function UpdateListBox():
        Clear main_listBox
        For each drink with quantity > 0 in drinkQuantities:
            Calculate itemTotal as price * quantity
            Add formatted line to main_listBox with drink name, quantity, and itemTotal
        Add separator line
        Add total cost to main_listBox
        Add lastAction message to main_listBox

Selection Construction:
    Function SelectionConstructor(string selectionText, int foo):
        If foo == 0 and selectionText is not empty:
            Increment quantity in drinkQuantities for selectionText
            Increase totalPriceCost by drinkPrices[selectionText]
            Set lastAction to "Added 1 {selectionText} to your order."
            Call UpdateListBox()
        Else if foo == 1:
            Disable drink selection panel
            Enable money panel
            Set lastAction to "Order totaled. Please insert payment."

Event Handlers:
    Function btn_Latte_Click():
        Call SelectionConstructor("Latte", 0)

    Function btn_FlatWhite_Click():
        Call SelectionConstructor("Flat White", 0)

    Function btn_Cap_Click():
        Call SelectionConstructor("Cappuccino", 0)

    Function btn_amer_Click():
        Call SelectionConstructor("Americano", 0)

    Function sum_btn_Click():
        Call SelectionConstructor(null, 1)
        Disable sum_btn

    Function cancel_but_Click():
        Show confirmation dialog to cancel order
        If confirmed:
            Call resetValues()


    Function payment_but_Click():
        If totalPriceCost > totalMoneyInput:
            Calculate remaining amount
            Show message to insert remaining amount
            Update lastAction with remaining amount
        Else:
            Calculate totalChange as totalMoneyInput - totalPriceCost
            Show message with total change and thank user
            Add final details to main_listBox (total paid and change)
            Call receiptPrinter(main_listBox.Items)
            Set lastAction to "Payment complete."
            Call resetValues()


Receipt Printing:
    Function receiptPrinter(content):
        Try:
            Define file path in receipt directory
            Write content to file
            Log success message
        Catch exception:
            Log error message
```

Testing

Testing is essential to ensure that an application runs smoothly and cohesively. During the testing process, a developer will not only check for code safety but also evaluate resource usage and overall interaction with the hardware. Following the creation of pseudocode based on the project specifications, a testing table was developed to organise the functionalities that required testing. To manage complexity and avoid encountering errors in later stages, each function was tested incrementally. This approach prevented overwhelming issues as the code expanded in complexity. Functions were initially tested in a separate environment; once they performed correctly, they were integrated into the main program through references and appropriate function calls.



Core Functionality Tests

Test ID	Category	Description	Test Steps	Expected Result	Actual Result	Status
T001	Application Start	Initial state check	1. Launch application	- Money panel disabled - Drink panel enabled - All textboxes empty	- Money panel disabled - Drink panel enabled - All textboxes empty	PASS
T002	Drink Prices	Verify all drink prices	1. Check each drink price: - Flat White - Latte - Cappuccino - Americano	- Flat White = £2.25 - Latte = £2.85 - Cappuccino = £3.00 - Americano = £2.95	- Flat White = £2.25 - Latte = £2.85 - Cappuccino = £3.00 - Americano = £2.95	PASS

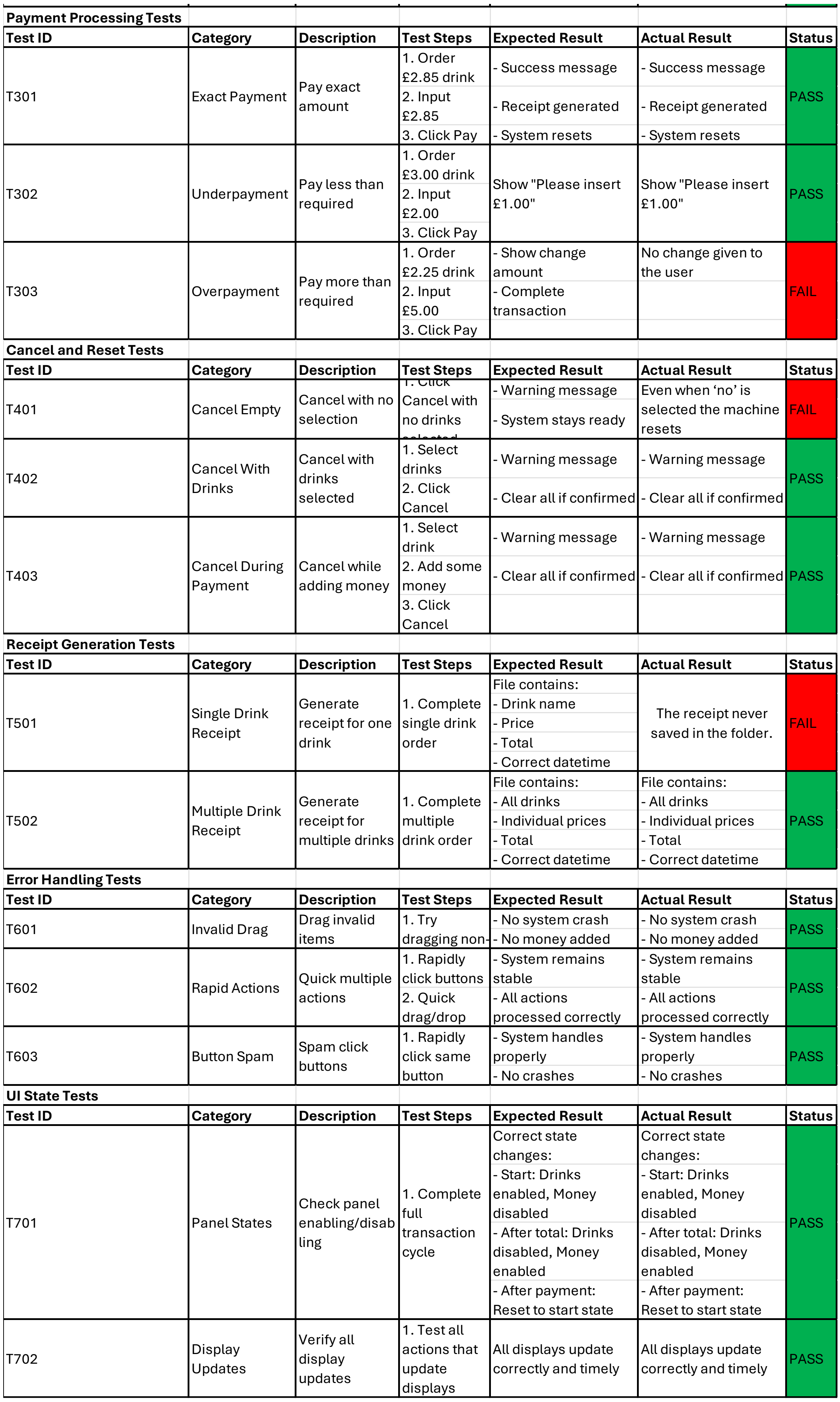
Drink Selection Tests

Test ID	Category	Description	Test Steps	Expected Result	Actual Result	Status
T101	Single Selection	Order one Flat White	1. Click Flat White 2. Click Total	- Display shows "Flat White - £2.25" - Total correct - Drink panel disables	- Display shows "Flat White - £2.25" - Total correct - Drink panel disables	PASS
T102	Multiple Selection	Order multiple drinks	1. Select Latte 2. Select Cappuccino 3. Click Total	- Both drinks listed - Total = £5.85 - Correct order display	- Both drinks listed - Total = £5.85 - Correct order display	PASS PASS
T103	Rapid Selection	Quick multiple clicks	1. Rapidly click different drinks	- All selections registered - No system crash - Correct total	- All selections registered - No system crash - Correct total	PASS

Money Input Tests

Test ID	Category	Description	Test Steps	Expected Result	Actual Result	Status
T201	Single Coin	Drag £1 coin	1. Order drink 2. Drag £1 to payment box	Money total shows £1.00	Money total shows £1.00	PASS
T202	Multiple Coins	Various coin combination	1. Order drink 2. Drag: £2, £1, 50p	Money total shows £3.50	Money total shows £3.50	PASS
T203	All Denominations	Test each money type	Test each: - 5p -10p - 20p - 50p -£1 -£2 -£5 -£10	Each denomination correctly adds to total	Each denomination correctly adds to total	PASS

Figure 4



Payment Processing Tests						
Test ID	**Category**	**Description**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
T301	Exact Payment	Pay exact amount	1. Order £2.85 drink 2. Input £2.85 3. Click Pay	- Success message - Receipt generated - System resets	- Success message - Receipt generated - System resets	PASS
T302	Underpayment	Pay less than required	1. Order £3.00 drink 2. Input £2.00 3. Click Pay	Show "Please insert £1.00"	Show "Please insert £1.00"	PASS
T303	Overpayment	Pay more than required	1. Order £2.25 drink 2. Input £5.00 3. Click Pay	- Show change amount - Complete transaction	No change given to the user	FAIL

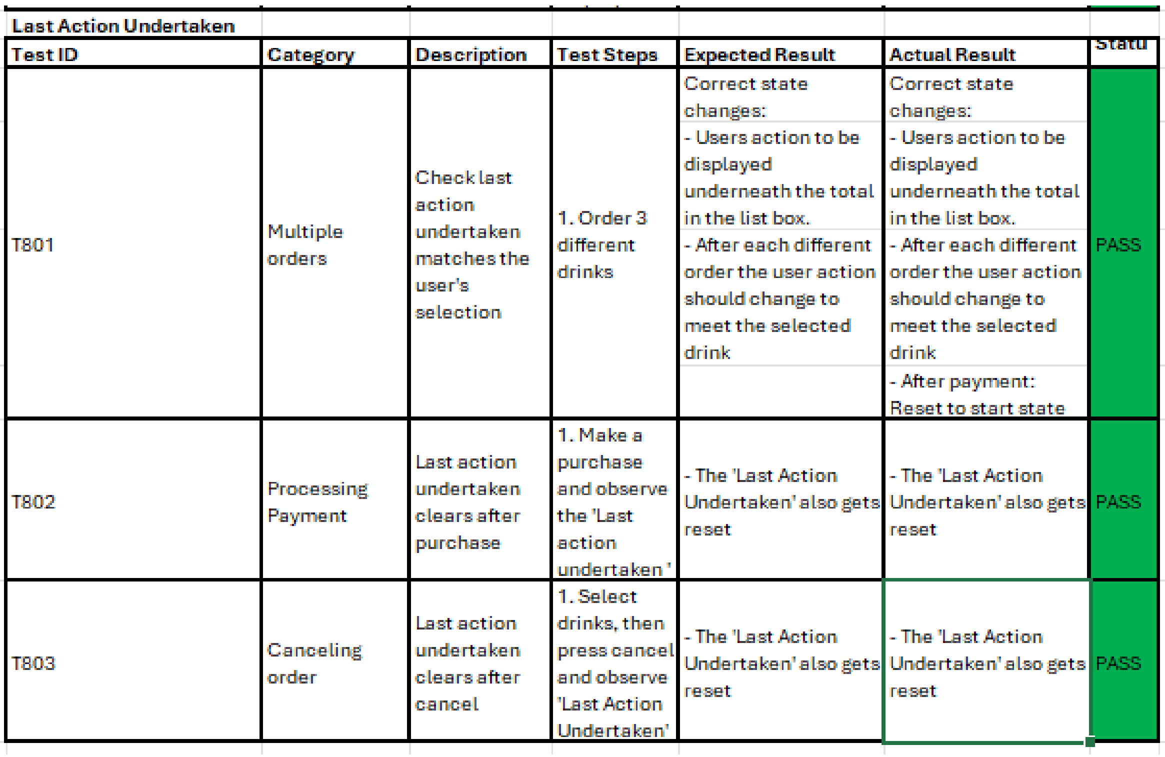
Cancel and Reset Tests						
Test ID	**Category**	**Description**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
T401	Cancel Empty	Cancel with no selection	1. Click Cancel with no drinks selected	- Warning message - System stays ready	Even when 'no' is selected the machine resets	FAIL
T402	Cancel With Drinks	Cancel with drinks selected	1. Select drinks 2. Click Cancel	- Warning message - Clear all if confirmed	- Warning message - Clear all if confirmed	PASS
T403	Cancel During Payment	Cancel while adding money	1. Select drink 2. Add some money 3. Click Cancel	- Warning message - Clear all if confirmed	- Warning message - Clear all if confirmed	PASS

Receipt Generation Tests						
Test ID	**Category**	**Description**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
T501	Single Drink Receipt	Generate receipt for one drink	1. Complete single drink order	File contains: - Drink name - Price - Total - Correct datetime	The receipt never saved in the folder.	FAIL
T502	Multiple Drink Receipt	Generate receipt for multiple drinks	1. Complete multiple drink order	File contains: - All drinks - Individual prices - Total - Correct datetime	File contains: - All drinks - Individual prices - Total - Correct datetime	PASS

Error Handling Tests						
Test ID	**Category**	**Description**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
T601	Invalid Drag	Drag invalid items	1. Try dragging non-	- No system crash - No money added	- No system crash - No money added	PASS
T602	Rapid Actions	Quick multiple actions	1. Rapidly click buttons 2. Quick drag/drop	- System remains stable - All actions processed correctly	- System remains stable - All actions processed correctly	PASS
T603	Button Spam	Spam click buttons	1. Rapidly click same button	- System handles properly - No crashes	- System handles properly - No crashes	PASS

UI State Tests						
Test ID	**Category**	**Description**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
T701	Panel States	Check panel enabling/disabling	1. Complete full transaction cycle	Correct state changes: - Start: Drinks enabled, Money disabled - After total: Drinks disabled, Money enabled - After payment: Reset to start state	Correct state changes: - Start: Drinks enabled, Money disabled - After total: Drinks disabled, Money enabled - After payment: Reset to start state	PASS
T702	Display Updates	Verify all display updates	1. Test all actions that update displays	All displays update correctly and timely	All displays update correctly and timely	PASS

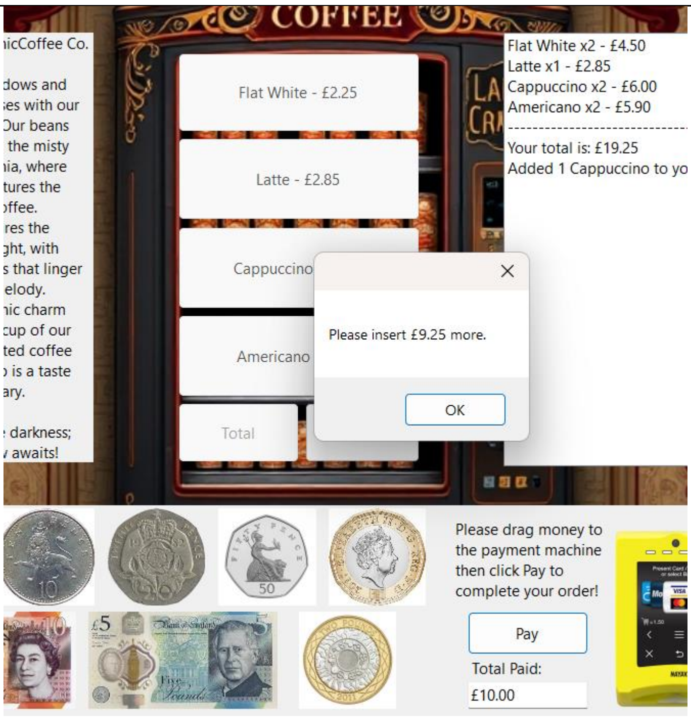
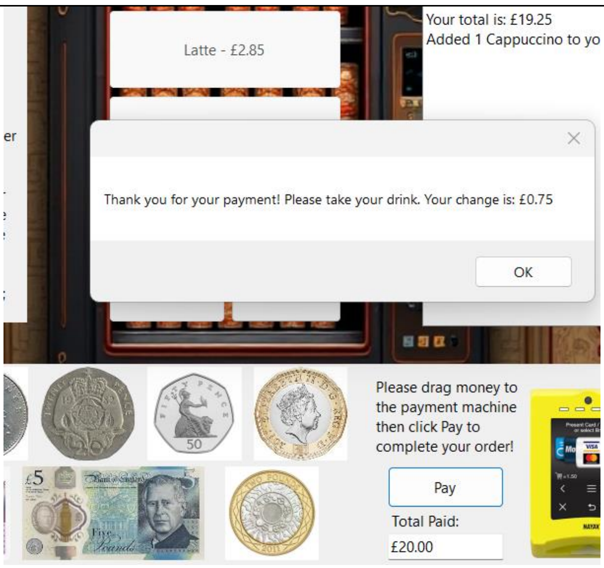
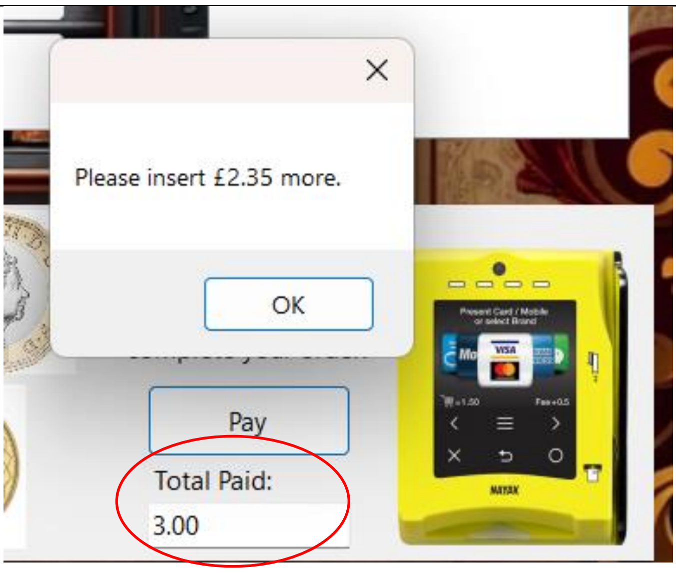
Figure 5



Last Action Undertaken

Test ID	Category	Description	Test Steps	Expected Result	Actual Result	Statu
T801	Multiple orders	Check last action undertaken matches the user's selection	1. Order 3 different drinks	Correct state changes: - Users action to be displayed underneath the total in the list box. - After each different order the user action should change to meet the selected drink	Correct state changes: - Users action to be displayed underneath the total in the list box. - After each different order the user action should change to meet the selected drink - After payment: Reset to start state	PASS
T802	Processing Payment	Last action undertaken clears after purchase	1. Make a purchase and observe the 'Last action undertaken'	- The 'Last Action Undertaken' also gets reset	- The 'Last Action Undertaken' also gets reset	PASS
T803	Canceling order	Last action undertaken clears after cancel	1. Select drinks, then press cancel and observe 'Last Action Undertaken'	- The 'Last Action Undertaken' also gets reset	- The 'Last Action Undertaken' also gets reset	PASS

Figure 6

 icCoffee Co. dows and es with our Our beans the misty ia, where tures the offee. res the ght, with s that linger elody. ic charm cup of our ted coffee is a taste ary. darkness; awaits! Flat White x2 - £4.50 Latte x1 - £2.85 Cappuccino x2 - £6.00 Americano x2 - £5.90 ----------------------------------- Your total is: £19.25 Added 1 Cappuccino to yo Please insert £9.25 more. Please drag money to the payment machine then click Pay to complete your order! Total Paid: £10.00 COFFEE Flat White - £2.25 Latte - £2.85 Cappuccino Americano Total OK Pay *Figure 7*	As shown in *figure 6*, the testing results indicated that the software performed as expected. When the total amount paid was insufficient, a pop-up box appeared to inform the user of the additional amount required. To prevent further selections, the drink selection box behind the pop-up is disabled, ensuring that users cannot choose additional drinks until the correct payment is made. This pop-up will continue to display until the user inserts the necessary amount.
 Your total is: £19.25 Added 1 Cappuccino to yo Thank you for your payment! Please take your drink. Your change is: £0.75 Please drag money to the payment machine then click Pay to complete your order! Total Paid: £20.00 *Figure 7*	The next scenario addresses when too much money is inserted into the payment box. As illustrated in *figure 7*, a pop-up will appear, displaying the amount of change that will be returned to the user. This feature ensures that users are promptly informed about their change, enhancing the overall transaction experience and returning the correct amount of change (*for future reference, if connected within a system this would then trigger the aspect of the machine that would be responsible for returning x amount of change)*.
 Please insert £2.35 more. Total Paid: 3.00 *Figure 8*	During further testing, an amount of £3.00 was inserted into the "Total Paid" box with the expectation that the software would recognise it as money paid and complete the purchase. However, this did not function as predicted, as dragging the money into the payment box only increased the count. Instead, the "Total Paid" box simply displayed the amount without processing it as a completed transaction. This shows that the code has been tested for security vulnerabilities.

Solving Bugs

As you can see in *figure 4* above, 3 bugs were identified. Below is a description and corresponding solution to every error encountered during testing:

T303

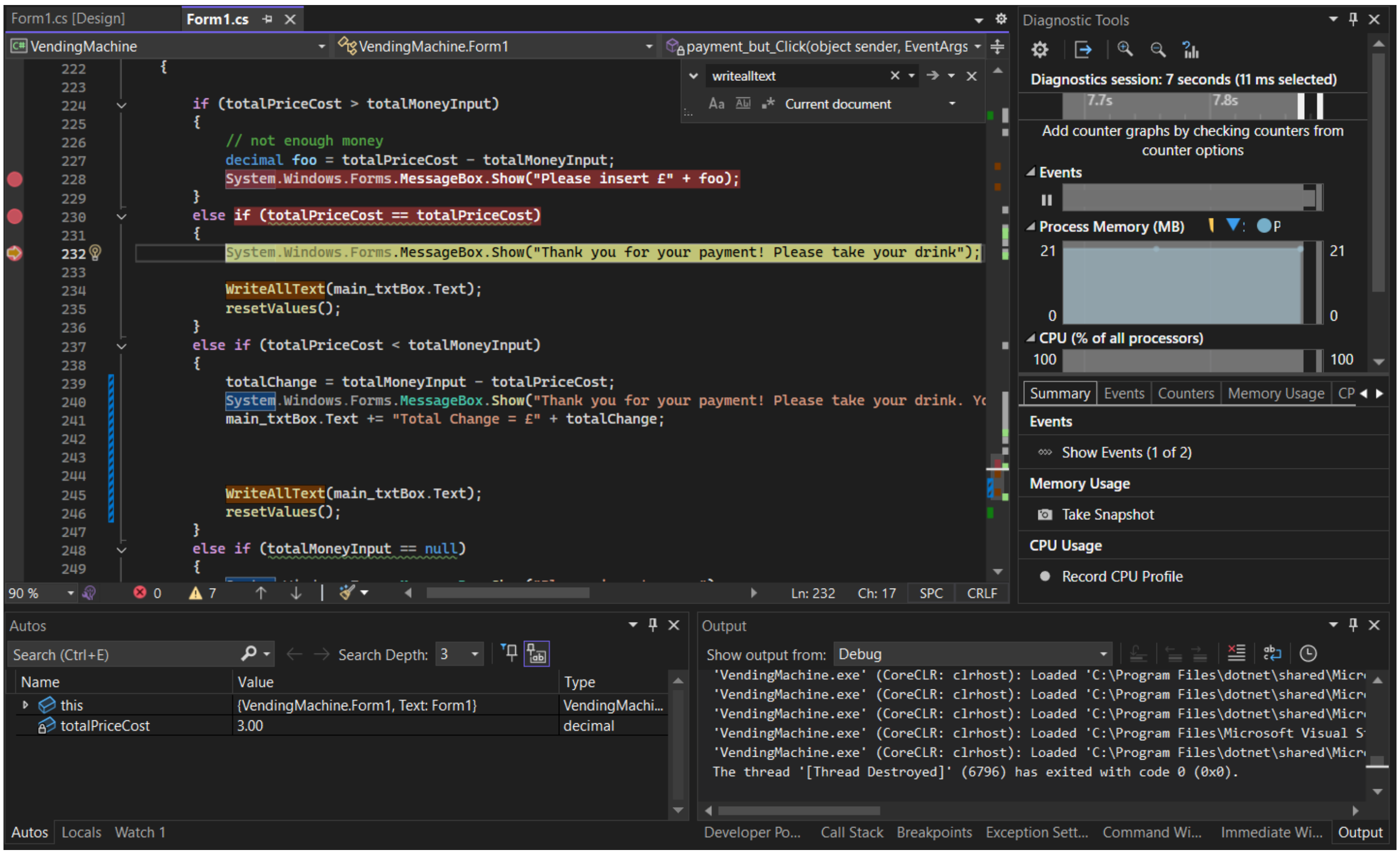
With this bug no change was given to the user if the amount of money inputted was larger than the money owed. The issue was caused by a very simple syntax error as seen in *figure 9*.

The code that was present was:

```
if (totalPriceCost == totalPriceCost)
```

This is because `totalPriceCost` cannot equal itself, if it did, it would always be true. Therefore, a corrective change had to be made to reflect the following:

```
if (totalPriceCost == totalMoneyInput)
```



```
        {

            if (totalPriceCost > totalMoneyInput)
            {
                // not enough money
                decimal foo = totalPriceCost - totalMoneyInput;
                System.Windows.Forms.MessageBox.Show("Please insert £" + foo);
            }
            else if (totalPriceCost == totalPriceCost)
            {
                System.Windows.Forms.MessageBox.Show("Thank you for your payment! Please take your drink");

                WriteAllText(main_txtBox.Text);
                resetValues();
            }
            else if (totalPriceCost < totalMoneyInput)
            {
                totalChange = totalMoneyInput - totalPriceCost;
                System.Windows.Forms.MessageBox.Show("Thank you for your payment! Please take your drink. Yo
                main_txtBox.Text += "Total Change = £" + totalChange;



                WriteAllText(main_txtBox.Text);
                resetValues();
            }
            else if (totalMoneyInput == null)
            {
```

Name	Value	Type
this	{VendingMachine.Form1, Text: Form1}	VendingMachi...
totalPriceCost	3.00	decimal

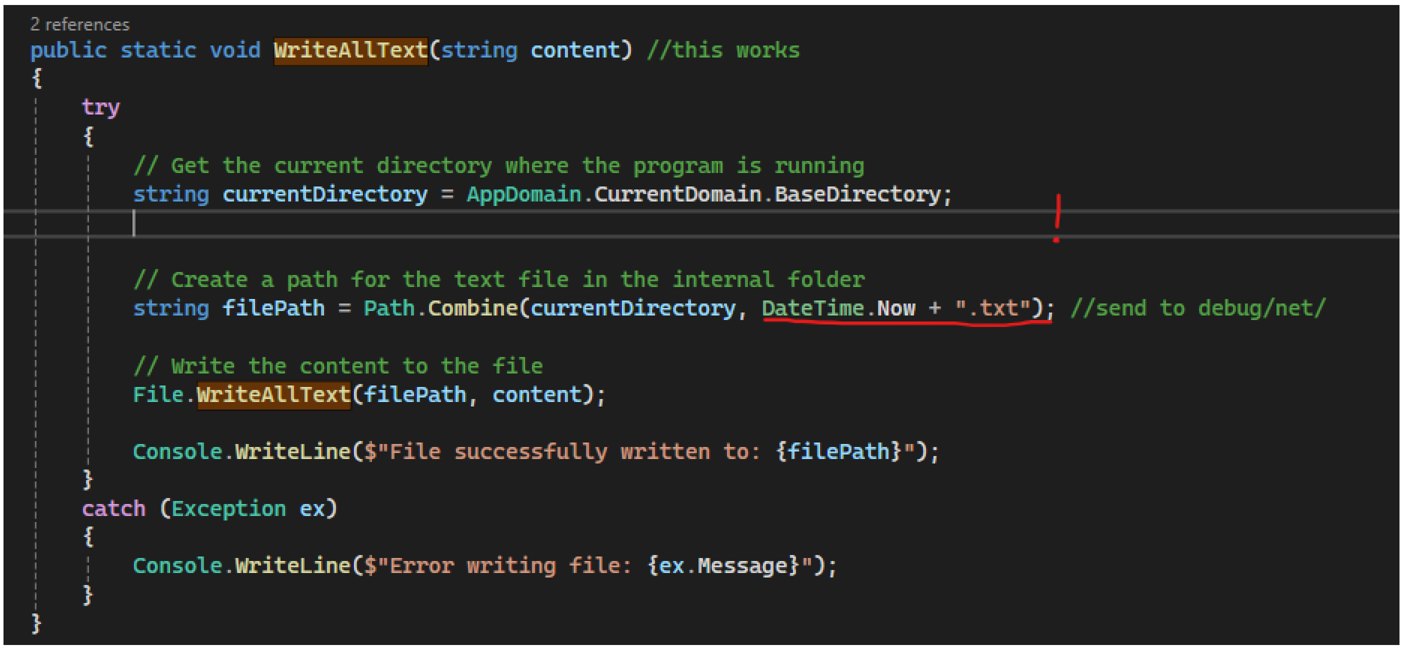
```
'VendingMachine.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Micr
'VendingMachine.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Micr
'VendingMachine.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Micr
'VendingMachine.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\Microsoft Visual S
'VendingMachine.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Micr
The thread '[Thread Destroyed]' (6796) has exited with code 0 (0x0).
```

Figure 9

T501

Whenever a purchase was being made the receipt .txt was never getting saved. The reason this was not happening is highlighted in *figure 10*. The `DateTime.Now + “.txt”` couldn’t be saved as DateTime as thisis it not a string. To solve this problem a string variable called `date` was created that firstly turned DateTime into a string that we could then concatenate it with the .txt string. The updated code looks like this:

```
string date = DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
// Create a path for the text file in the internal folder
string filePath = Path.Combine(currentDirectory, date + ".txt"); //send to debug/net/
```



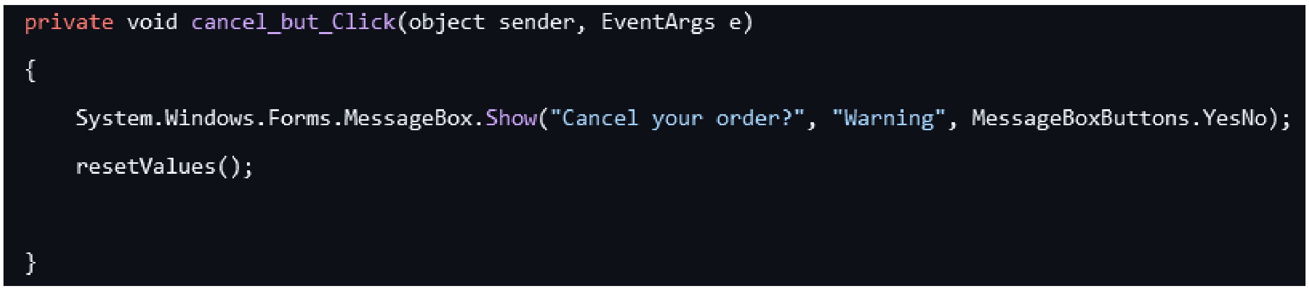
```
2 references
public static void WriteAllText(string content) //this works
{
    try
    {
        // Get the current directory where the program is running
        string currentDirectory = AppDomain.CurrentDomain.BaseDirectory;
                                                                          !
        // Create a path for the text file in the internal folder
        string filePath = Path.Combine(currentDirectory, DateTime.Now + ".txt"); //send to debug/net/

        // Write the content to the file
        File.WriteAllText(filePath, content);

        Console.WriteLine($"File successfully written to: {filePath}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error writing file: {ex.Message}");
    }
}
```

Figure 10

T401

Whenever the cancel button was clicked a message box appeared asking whether they would like to proceed with the cancellation or to go back by pressing ‘no’. The code in *figure 11* would reset the machine no matter what the selection was. The solution was to add an if statement that would only reset the values if ‘yes’ was clicked. This is the corrected code:

```
DialogResult Result = MessageBox.Show("Cancel your order?", "Warning",
MessageBoxButtons.YesNo);
 if (Result == DialogResult.Yes)
 {
     resetValues();
 }
```

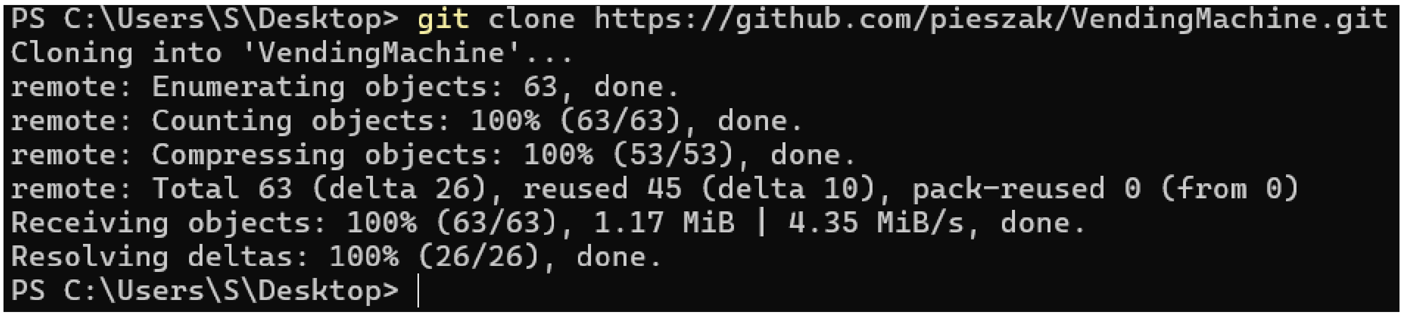


```
private void cancel_but_Click(object sender, EventArgs e)
{
    System.Windows.Forms.MessageBox.Show("Cancel your order?", "Warning", MessageBoxButtons.YesNo);
    resetValues();


}
```

Figure 11

Following the identification of bugs, time was spent optimising and rewriting the code to enhance reusability and eliminate unnecessary code. Cleaner, less cluttered code is not only easier to read but also more robust. Given that this code will be deployed on a vending machine, minimising the program's size is crucial, as hardware limitations must be considered. A streamlined programme will ensure efficient performance on the vending machine's hardware.

For further testing, the code was redownloaded from GitHub to determine if there would be any differences in its operation (*see figure 12*). Upon running the code, it performed flawlessly.



```
PS C:\Users\S\Desktop> git clone https://github.com/pieszak/VendingMachine.git
Cloning into 'VendingMachine'...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 63 (delta 26), reused 45 (delta 10), pack-reused 0 (from 0)
Receiving objects: 100% (63/63), 1.17 MiB | 4.35 MiB/s, done.
Resolving deltas: 100% (26/26), done.
PS C:\Users\S\Desktop>
```

Figure 12

References

DeepAI (2024) ChatGPT [Image Generation] generate a gothic vending machine for coffee, in 2d, with an old library style. 28th October 2024.

Check Your Change (no date) *Other Coins & Banknotes*. Available at: https://www.checkyourchange.co.uk/all-other-decimal-coins/ (Accessed: 29th October 2024).

Check Your Change (no date) *Bank of England Bank Notes*. Available at: https://www.checkyourchange.co.uk/bank-of-england-bank-notes/ (Accessed: 29th October 2024).

Source Code

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;

namespace VendingMachine
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            paymentBox.AllowDrop = true;
            buttonControlEnabled(1, false);

            foreach (string drink in drinkPrices.Keys) // allows us to change if the
drinkPrices gets updated
            {
                drinkQuantities[drink] = 0;
            }
            UpdateListBox();
        }


        #region Public Variables and Data

        // global variable decleration
        private decimal totalPriceCost;
        private decimal totalMoneyInput;
        private decimal totalChange;

        // Dictionary - allows the drinks to stack in the list box
        private Dictionary<string, int> drinkQuantities = new Dictionary<string, int>();


        private string lastAction = "Welcome! Please select your drinks.";


        // Dictionary that has all the components of the drinks offered
        private Dictionary<string, decimal> drinkPrices = new Dictionary<string, decimal>()
        {
            { "Flat White", 2.25m },
            { "Latte", 2.85m },
            { "Cappuccino", 3.00m },
            { "Americano", 2.95m }
        };

        //error message get's printed whenever a critical faul happens in the code.
        private void errorMessage()
        {
            MessageBox.Show("It seems that the vending machine fell into a fatal error. Please
restart the machine.", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
        }

        //controls the money and drinks panel - turns on and off
        private void buttonControlEnabled(int i, bool selection)
        {

            if (i == 0)
            {
                drinkSelectionPanel.Enabled = selection;
                sum_btn.Enabled = true;
            }
            else if (i == 1)
            {
                moneyPanel.Enabled = selection;
            }
            else
            {
                errorMessage();
            }
        }

        // resets values when the program comes to completion
        private void resetValues()
        {
            totalMoneyInput = 0;
            totalPriceCost = 0;
            totalChange = 0;
            main_listBox.Items.Clear();
            moneyIn_txtBox.Clear();
            foreach (var drink in drinkQuantities.Keys)
            {
                drinkQuantities[drink] = 0;
            }
            buttonControlEnabled(1, false);
            buttonControlEnabled(0, true);
            lastAction = "";
        }
        #endregion

        #region Drag and Drop

        //money down pressed
        private void money_MouseDown(object sender, MouseEventArgs e)
        {
            PictureBox moneyBox = sender as PictureBox;
            moneyBox.DoDragDrop(moneyBox, DragDropEffects.Copy);
        }

        private void paymentBox_DragEnter(object sender, DragEventArgs e)
        {
            e.Effect = DragDropEffects.Copy;
        }

        //
        private void paymentBox_DragDrop(object sender, DragEventArgs e)
        {
            PictureBox droppedCoin = e.Data.GetData(typeof(PictureBox)) as PictureBox;

            if (droppedCoin == money_5p) totalMoneyInput += .05m;
            else if (droppedCoin == money_10p) totalMoneyInput += 0.10m;
            else if (droppedCoin == money_20p) totalMoneyInput += 0.20m;
            else if (droppedCoin == money_50p) totalMoneyInput += 0.50m;
            else if (droppedCoin == money_1GBP) totalMoneyInput += 1.00m;
            else if (droppedCoin == money_2GBP) totalMoneyInput += 2.00m;
            else if (droppedCoin == money_5GBP) totalMoneyInput += 5.00m;
            else if (droppedCoin == money_10GBP) totalMoneyInput += 10.00m;
            else
            {
                errorMessage();
            }
            moneyIn_txtBox.Text = "£" + totalMoneyInput.ToString("F2"); //F2 maintains the
output is limited to 2 d.p.
            lastAction = $"Added £{totalMoneyInput:F2}. Total inserted:
£{totalMoneyInput:F2}"; // string interpolation
        }
        #endregion

        #region Selection Construction


        private void UpdateListBox()
        {
            main_listBox.Items.Clear();
            // First, add drink items
            foreach (var drink in drinkQuantities.Where(x => x.Value > 0))
            {
                decimal price = drinkPrices[drink.Key];
                decimal itemTotal = price * drink.Value;
                string line = $"{drink.Key} x{drink.Value} - £{itemTotal:F2}";
                main_listBox.Items.Add(line);
            }

            // Add a separator line
            main_listBox.Items.Add("--------------------------------------------");

            // Add total
            main_listBox.Items.Add($"Your total is: £{totalPriceCost:F2}");

            // Add last action at the very end
            main_listBox.Items.Add(lastAction);
        }

        private void SelectionConstructor(string selectionText, int foo)
        {
            if (foo == 0 && !string.IsNullOrEmpty(selectionText))
            {
                drinkQuantities[selectionText]++;
                totalPriceCost += drinkPrices[selectionText];
                lastAction = $"Added 1 {selectionText} to your order.";
                UpdateListBox();
            }
            else if (foo == 1)
            {
                buttonControlEnabled(0, false);
                buttonControlEnabled(1, true);
                lastAction = "Order totaled. Please insert payment.";
            }
        }
        #endregion

        #region Event Actions
        private void btn_Latte_Click(object sender, EventArgs e)
        {
            SelectionConstructor("Latte", 0);
        }

        private void btn_FlatWhite_Click(object sender, EventArgs e)
        {
            SelectionConstructor("Flat White", 0);
        }

        private void btn_Cap_Click(object sender, EventArgs e)
        {
            SelectionConstructor("Cappuccino", 0);
        }

        private void btn_amer_Click(object sender, EventArgs e)
        {
            SelectionConstructor("Americano", 0);
        }

        private void sum_btn_Click(object sender, EventArgs e)
        {
            SelectionConstructor(null, 1);
            sum_btn.Enabled = false;
        }

        private void cancel_but_Click(object sender, EventArgs e)
        {
            DialogResult Result = MessageBox.Show("Cancel your order?", "Warning", 
MessageBoxButtons.YesNo);
            if (Result == DialogResult.Yes)
            {
                resetValues();
            }
            // else not required
        }

        private void payment_but_Click(object sender, EventArgs e)
        {
            if (totalPriceCost > totalMoneyInput)
            {
                decimal remaining = totalPriceCost - totalMoneyInput;
                MessageBox.Show($"Please insert £{remaining:F2} more.");
                lastAction = $"Insufficient funds. Please insert £{remaining:F2} more.";
            }
            else if (totalPriceCost <= totalMoneyInput)
            {
                //allows us to set the template for the receipt .txt file
                totalChange = totalMoneyInput - totalPriceCost;
                MessageBox.Show($"Thank you for your payment! Please take your drink. Your 
change is: £{totalChange:F2}");
                main_listBox.Items.Add("---------------------------------------------");
                main_listBox.Items.Add($"Total money paid = £{totalMoneyInput:F2}");
                main_listBox.Items.Add("---------------------------------------------");
                main_listBox.Items.Add($"Total Change = £{totalChange:F2}");
                receiptPrinter(main_listBox.Items);
                lastAction = "Payment complete. Thank you for your purchase!";
                resetValues();
            }
            else
            {
                errorMessage();
            }
        }

        private static void receiptPrinter(ListBox.ObjectCollection content) //used for saving 
the above text to a .txt file
        {
            try
            {
                string currentDirectory = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, 
"receipt");
                if (!Directory.Exists(currentDirectory))
                {
                    Directory.CreateDirectory(currentDirectory);
                }

                string date = DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");
                string filePath = Path.Combine(currentDirectory, date + ".txt");

                File.WriteAllLines(filePath, content.Cast<string>());
                Console.WriteLine($"File successfully written to: {filePath}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error writing file: {ex.Message}");
            }
        }
    }
}
#endregion
```