Skeleton Code Breakdown

Static Methods

Identifier / Data		Description
CheckIfUserInputEvaluationIsATarget		
Parameters	**Targets** : Integer List **UserInputInRPN** : String List **Score** : Int	This method checks if the evaluation of the expression entered by the user matches any of the values in the **Targets** list and awards points appropriately. The method firstly calls the **EvaluateRPN** method, passing in the parameter **UserInputInRPN**. This evaluates the user inputted expression, represented in RPN which is stored in the variable **UserInputEvaluation**. The method then sets the **UserInputEvaluationIsATarget** variable to False so that the return variable has a default of False. The method tests if the **UserInputEvaluation** variable has the value of -1. This value represents that **UserInputInRPN** could not be evaluated. If **UserInputEvaluation** contains any other value, the method performs a count-controlled loop to iterate through the **Targets** list looking for matches with targets. The loop compares the **UserInputEvaluation** with each element of the **Targets** list. If a match is found the **Score** is incremented by 2 and the value at the index where the target was matched is set to -1 and the **UserInputEvaluationIsATarget** variable is set to True. Once the loop is complete, the current state of the **UserInputEvaluationIsATarget** variable and the **Score** are returned.
Return values	**UserInputEvaluationIsATarget** : Bool **Score** : Int	
CheckIfUserInputValid		
Parameters	**UserInput** : String	This method uses a Regular Expression to test if the **UserInput** parameter matches the pattern of an infix expression. The Regular Expression does not test for brackets. The Regular Expression used is: ^([0-9]+[\+\-*\/])+[0-9]+$ To match, the **UserInput** parameter must start with one or many digits 0 to 9 followed by a single mathematical operator which can only be either + - * or / (these are escaped with backslashes to be treated as literal characters). This entire group of digit(s) followed by an operator can be repeated one or many times. The string must end with one or many digits 0 to 9. If the **UserInput** parameter matches the Regular Expression pattern the method returns True, otherwise it returns False.
Return values	Bool	

<table>
<tr><th colspan="3">CheckNumbersUsedAreAllInNumbersAllowed</th></tr>
<tr><td>Parameters</td><td>NumbersAllowed : Integer List
UserInputInRPN : String List
MaxNumber : Int</td><td rowspan="2">This method is used to test if the numbers entered by the user are present in their NumbersAllowed list.

The method firstly creates a temporary integer list (Temp) to compare against. It then iterates through the NumbersAllowed list assigning copies of its values to the new temporary list. This is because lists are, by default, passed as references not by value. The method removes numbers from a comparison list when it finds them to prevent multiple operands matching the same value in the NumbersAllowed list. If the method removed values directly from the NumbersAllowed list, this would impact the application elsewhere.

The method then iterates through the UserInputInRPN list. It firstly checks each element using the CheckValidNumber to confirm the element is a valid number which is within range. This is required to ensure that only operands are compared with the NumbersAllowed list. If True, the method subsequently checks if the operand is contained in the Temp list and if it is, the operand is removed from the Temp list. If the operand is NOT contained in the Temp list, the method returns False because it has found an operand which cannot be in the NumbersAllowed list.

The CheckValidNumber check does not have an else condition, meaning that if an Item in UserInputInRPN does not meet with the requirements of the check, for example the number is greater than MaxNumber, the method doesn’t acknowledge it.</td></tr>
<tr><td>Return values</td><td>Bool</td></tr>
<tr><th colspan="3">CheckValidNumber</th></tr>
<tr><td>Parameters</td><td>Item : String
MaxNumber : Int</td><td rowspan="2">This method checks if a value passed to it is an integer which is within the range set by the game.

This method uses a Regular Expression to test if the Item parameter matches the pattern of an integer number.

The Regular Expression used is: ^[0-9]+$

To match, the Item parameter must be one or many digits 0 to 9. If the Item parameter matches the Regular Expression pattern, the method casts the parameter to a local integer variable ItemAsInteger. The method then tests to see if ItemAsInteger is greater than zero and less than or equal to the MaxNumber parameter. If it is, the method returns True. If neither of these conditions is met, the method returns False.</td></tr>
<tr><td>Return values</td><td>Bool</td></tr>
</table>

ConvertToRPN		
Parameters	**UserInput** : String	This method converts the infix expression entered by the user into postfix notation. This method uses a version of the shunting yard algorithm. Initialises the following local variables: • **Position** to 0. This is used to identify indices in the **UserInput**. • **Precedence** to Dictionary of type <string, int>. This stores the main four mathematical operators with an associated value. Multiplication and Division are assigned higher values than Addition and Subtraction. This is used to allow the method to comply with BIDMAS. *The Dictionary does not recognise Brackets or Indices, however.* • **Operand** as an integer. This uses the **GetNumberFromUserInput** method to get the first number in the infix notation. • **UserInputInRPN** as a list of strings. This is immediately populated with the **Operand** variable casted as a string. • **Operators** as a list of strings. This is immediately populated with the first operator in the **UserInput** expression. The method then enters a condition-controlled loop which performs the following actions: **Operand** and **Position** are updated using the **GetNumberFromUserInput** method to get the next number in the infix notation. The updated **Operand** is appended to the **UserInputInRPN** list as a string. The next value in the expression (assuming it is valid) must therefore be either an operator or the end of the expression. If the **Position** variable is less than the length of the **UserInput** string, there must be operators and operands in the expression which have not yet been included in the postfix version. Since the method has just extracted an operand from the expression, it uses the **Position** variable minus 1 to extract the operator prior and stores this in the variable **CurrentOperator**. It then tests this against the other values incrementally in the **Operators** list by popping values from the back of the list and using the **Precedence** dictionary to compare their worth. If the value in the **Operators** list has a higher or equal precedence than the **CurrentOperator**, it is added to the **UserInputInRPN** list and removed from the operators list. The **CurrentOperator** is then added to the operators list. This calculation ensures that Multiplication and Division functions are added to the **UserInputInRPN** list before Addition and Subtraction. If the **Position** variable is not less than the length of the **UserInput** string, all the operands and operators from the string have been extracted so the method iterates through the **Operators** list popping values from the back of the list and adding them to the **UserInputInRPN** list. The method then returns the completed **UserInputInRPN** list.
Return values	**UserInputInRPN**: String List	

CreateTargets		
Parameters	**SizeOfTargets** : Int **MaxTarget** : Int	This method populates the **Targets** list at the start of the game for a standard game. The method initialises the **Targets** integer list and uses a count-controlled loop to populate the first five indices with the value -1. It then uses a second count-controlled loop with an upperbound of the **SizeOfTargets** parameter minus 4 to continue populating the list with a random number from the **GetTarget** method. In the standard pre-release game this will result in a **Targets** list with 20 elements which is returned.
Return values	**Targets** : Integer List	
DisplayNumbersAllowed		
Parameters	**NumbersAllowed** : Integer List	This method is used to display all the values in the **NumbersAllowed** list. The method iterates through the **NumbersAllowed** parameter writing each value to the screen.
Return values	n/a	
DisplayScore		
Parameters	**Score** : Int	This method displays the current game **Score** on the screen using concatenation.
Return values	n/a	
DisplayState		
Parameters	**Targets** : Integer List **NumbersAllowed** : Integer List **Score** : Int	This method displays the current state of the game by calling the following methods: • **DisplayTargets** – to display the contents of the **Targets** list • **DisplayNumbersAllowed** – to display the contents of the **NumbersAllowed** list • **DisplayScore** – to dis • play the current game **Score**
Return values	n/a	
DisplayTargets		
Parameters	**Targets** : Integer List	This method is used to display all the values in the **Targets** list with each element surrounded by the pipe symbol | The method iterates through the **Targets** parameter. If an element contains -1 the method prints a blank space onto the screen, otherwise it prints the element.
Return values	n/a	

<table>
<tr><th colspan="3">EvaluateRPN</th></tr>
<tr><td>Parameters</td><td>UserInputInRPN : String List</td><td rowspan="3">This method evaluates the RPN version of the expression entered by the user. If the expression evaluates to an integer (positive or negative) the integer is returned, otherwise the method returns -1.

This method initialises a string list S. The method uses this list as a stack. It then uses a condition-controlled loop to iterate through the UserInputInRPN parameter. This method uses this list as a queue.

The method iterates through the UserInputInRPN list comparing the first element with the string list “+-*/” and adding elements which are not part of this string to the list S (essentially dequeuing them from UserInputInRPN and pushing them onto the stack S). This allows the method to extract all the number values from the start of the postfix expression. When an operator is found at position 0 in the UserInputInRPN list, the loop stops, and the two most recent values added to the list S are assigned to the variables Num2 and Num1 (essentially popping them from the stack). These are cast as doubles to allow float division to be performed on them. The method uses selection to peek at the operator at the start of the UserInputInRPN list and perform the correct associated mathematical operation. The result of the operation is stored in the variable Result. The operator at the start of the UserInputInRPN is removed (essentially dequeued) and the Result is pushed onto the list S ready for the next evaluation.

This process is repeated until the UserInputInRPN list is empty which means all the expression has been evaluated and the list S only now contains the final result.

The method then subtracts a truncated version of the final result from the final result itself. If this evaluates to 0.0, then the result must have been a whole number and the truncated version of the final result cast as an integer is returned. If not, the expression must have included some division which has evaluated to a decimal and therefore cannot be a target in the Targets list; therefore -1 is returned.</td></tr>
<tr><td>Return values</td><td>Int</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><th colspan="3">FillNumbers</th></tr>
<tr><td>Parameters</td><td>NumbersAllowed : Integer List
TrainingGame : Bool
MaxNumber : Int</td><td rowspan="3">This method repopulates the NumbersAllowed list during the game.

If the TrainingGame parameter is True, the user is in a training game and the method simply returns a pre-populated list with the values 2, 3, 2, 8, 512. This ensures that a training game has the same values in the NumbersAllowed list on each turn.

If the TrainingGame parameter is False, the user is in a standard game and the method uses a condition-controlled loop to append values to the NumbersAllowed list using the GetTarget method to get a new in-range target until the list has five elements.</td></tr>
<tr><td>Return values</td><td>NumbersAllowed : Integer List</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th colspan="3">GetNumber</th></tr>
<tr><td>Parameters</td><td>MaxNumber : Int</td><td rowspan="3">This method returns a random number between 1 and the MaxNumber parameter (inclusive).</td></tr>
<tr><td>Return values</td><td>Int</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th colspan="3">GetNumberFromUserInput</th></tr>
<tr><td>Parameters</td><td>UserInput : String
Position : Int</td><td rowspan="3">This method is used to extract numbers from the infix expression entered by the user as it is being converted into postfix.

The method initially instantiates an empty String “Number”.

The method iterates through the UserInput parameter using a condition-controlled loop and the Position parameter to set the index of where to start iterating. Each character is checked using a Regular Expression to confirm if it is a number from 0 to 9. If it is, it is concatenated onto the Number variable. The Position variable is then incremented to move through the UserInput parameter. This technique allows the iteration to find multiple digit numbers without a delimiter. If a character found does not match the Regular Expression, it must be an operator which sets the MoreDigits variable to False, exiting the loop. The loop also exits if the Position variable equals the length of the UserInput string, meaning that it has iterated to the end of the string.

If the Number variable is an empty string, the UserInput parameter was not a number, and the method returns -1 together with the updated Position variable. If the Number variable is not empty, it is cast as an integer and returned together with the updated Position variable.</td></tr>
<tr><td>Return values</td><td>Int
Position : Int</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th colspan="3">GetTarget</th></tr>
<tr><td>Parameters</td><td>MaxTarget : Int</td><td rowspan="3">This method returns a random number between 1 and the MaxTarget parameter (inclusive).</td></tr>
<tr><td>Return values</td><td>Int</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><th colspan="3">GetNumber</th></tr>
<tr><td>Parameters</td><td>MaxNumber : Int</td><td rowspan="3">This method returns a random number between 1 and the MaxNumber parameter (inclusive).</td></tr>
<tr><td>Return values</td><td>Int</td></tr>
<tr><td></td><td></td></tr>
<tr><th colspan="3">Main</th></tr>
<tr><td>Parameters</td><td>default</td><td rowspan="3">This is the main entrance point for the application. It is used to determine if the application is going to use a standard game with a randomly generated Targets list and NumbersAllowed list or the training game with fixed content lists.

It initialises the following variables with default values:
• NumbersAllowed as an integer list.
• Targets as an integer list.
• MaxNumberOfTargets as an integer with an initial value of 20.
• MaxTarget as an integer.
• MaxNumber as an integer.
• TrainingGame as a Boolean.

The method asks the user if they would like to play the training game or a standard random game.

If the user selects a training game, these values are assigned to the following variables for use later in the game:
• MaxTarget = 1000
• MaxNumber = 1000
• TrainingGame = True
• The Targets list is populated with 20 numbers.

If the user does not select a training game, these values are assigned to the following variables for use later in the game:
• MaxTarget = 10
• MaxNumber = 50
• TrainingGame = False
• The Targets list is populated with 20 random values which are between 1 and 10 (the MaxTarget) inclusive.

The method calls the FillNumbers method to populate the NumbersAllowed list and then calls the main PlayGame method to start the game.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><th colspan="3">PlayGame</th></tr>
<tr><td>Parameters</td><td>Targets : Integer List
NumbersAllowed : Integer List
TrainingGame : Bool
MaxTarget : Int
MaxNumber : Int</td><td rowspan="3">Initialises the following local variables with default values:
Score to 0
GameOver to False
UserInput as a string
UserInputInRPN as a list of strings
These variables are then used and populated in the main application loop.

The method then enters into the main application loop. It is held in that loop by the value of the GameOver variable. The loop operates using the following steps:
Call the DisplayState method passing the Targets, NumbersAllowed and Score variables to display the current values in these variables onto the screen.
Prompt the user to enter an infix mathematical expression. The input is stored in the UserInput variable.
Call the CheckIfUserInputValid method, passing the UserInput variable.
If the input is valid, the ConvertToRPN method is called, passing in the UserInput variable. This converts the infix UserInput into reverse polish notation which is stored in the list UserInputInRPN.
Call the CheckNumbersUsedAreAllInNumbersAllowed method passing the NumbersAllowed list, UserInputInRPN list and the MaxNumber variable.
If all the values in the UserInputInRPN list are present in the NumbersAllowed list the CheckIfUserInputEvaluationIsATarget method is called passing in the Targets list, UserInputInRPN list and the Score variable.
If UserInputInRPN evaluates to one or more of the targets in the Targets list the Score is appropriately incremented. The RemoveNumbersUsed method is then called, passing in the UserInput variable, MaxNumber variable and NumbersAllowed list to remove the numbers used in a successful target match expression from the NumbersAllowed list. The FillNumbers method is then called, passing in the NumbersAllowed list together with the TrainingGame and MaxNumber variables to backfill the NumbersAllowed list with values.
The Score variable is then decremented. This occurs regardless of whether the user has successfully identified a target.
The method then tests to see if the first element in the Targets list is NOT -1. If it is, the GameOver variable is set to True which will exit the main application loop. If the first element of the Targets list is not -1, the UpdateTargets method is called, passing in the Targets list together with the TrainingGame and MaxTarget variables to shunt the elements in the Targets list one index to the left.
If the GameOver variable has been set to True and the main application loop has exited, “Game over!” and the final Score are displayed on the screen.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
</table>

<table>
<tr><th colspan="3">RemoveNumbersUsed</th></tr>
<tr><td>Parameters</td><td>UserInput : String
MaxNumber : Int
NumbersAllowed : Integer List</td><td rowspan="2">This method removes any numbers from the NumbersAllowed list which were used in a successful evaluation match with a target.

The method firstly calls the ConvertToRPN method, passing in the UserInput string which is the infix version of the expression. Although when the RemoveNumbersUsed method was called in the PlayGame method the UserInputInRPN list had been assigned values and confirmed valid, lists are, by default, passed as references not by value. Therefore, the EvaluateRPN method (called by the CheckIfUserInputEvaluationIsATarget method previously) had removed all the values from the UserInputInRPN list, consequently RemovedNumbersUsed needs to go back to the original infix expression from the user to rebuild a new postfix version of the expression.

The method then iterates through the UserInputInRPN list. It firstly checks each element using the CheckValidNumber to confirm the element is a valid number which is within range. This is required to ensure that only operands are compared with the NumbersAllowed list. If True, the method then checks if the operand is contained in the NumbersAllowed list and if it is, the operand is removed from the NumbersAllowed list.

Finally the method returns the NumbersAllowed list.</td></tr>
<tr><td>Return values</td><td>NumbersAllowed : Integer List</td></tr>
<tr><th colspan="3">UpdateTargets</th></tr>
<tr><td>Parameters</td><td>Targets : Integer List
TrainingGame : Bool
MaxTarget : Int</td><td rowspan="2">This method uses a count-controlled loop to shunt values in the Targets list one index to the left and backfill the list with a new value. This represents the functionality of a queue.

The method firstly iterates through the Targets list assigning each element the value at index + 1. This has the effect of moving each value one index to the left in the list.

The method then removes the last element in the list.

The method then uses selection on the TrainingGame variable. If True, the user has selected a training game and therefore the value at the last element in the Targets list (which is 119) is added to the end of the list. If False, the user has selected a normal game, so the GetTarget method is called, passing in the parameter MaxTarget. This selects a new random number between 1 and the MaxTarget (inclusive) and adds it to the end of the Targets list.

Finally the method returns the Targets list.</td></tr>
<tr><td>Return values</td><td>Targets : Integer List</td></tr>
</table>

