Contents

[Define the problem 2](#_Toc36210241)

[Module Identification – Control Structures 3](#_Toc36210242)

[IPO Chart 5](#_Toc36210243)

[Data Dictionary 8](#_Toc36210244)

[Algorithm 9](#_Toc36210245)

[Storyboard / Screen Design 10](#_Toc36210246)

# Define the problem

# Module Identification – Control Structures

1. User chooses first case – At the start of the game the user chooses a case that they keep throughout the game until the end or an offer is accepted.

The button that indicates this case will be changed to “selected”, and the value will be stored into selected. Then the value in the list Cases will be set to 0, which indicates an selected case.

Appropriate control structures would be:

* Sequence – has to happen in order for game to begin. Values stored into variables or array
* Binary Selection: Checks if the selection is in the range [1, 26]

1. Select other cases – User selects to open the all other cases except the last one:

The button that indicates this case will be locked and reveal value. Corresponding label will be locked.

Control Structures:

* Binary selection: Check if opened case number in BankTime, and if yes, go to Bank(), otherwise open case. Binary selection is also used to check if this case is already opened, or already selected.
* Sequence: Set the value to negative, which indicates that case is opened. Then call updateButtonsAndLabelStatus()

1. updateButtonsAndLabelStatus() – Updates the status of buttons and labels

This function checks the list Cases, and decide to either lock or unlock/hide or unhide a button/label.

Control Structures:

* Counted loop: Loop through every value in Cases, and:
* Binary Selection: Check if the value is less than 0, equals to 0 or greater than 0. If the value is greater than 0, don’t reveal value. If the value is equal to 0, show selected. If the value is greater than 0, show the value and lock the button, and hide corresponding label.

1. Bank Offer [ bank() ] – Asks if the user is going to accept the bank offer.

Control Structures:

* Sequence: Call calcBankOffer(), and get the value. Pops up a message box, and then:
* Binary Selection: Determine if the user clicks yes or no. If yes, quit game; else update the state and continue.

1. Bank Offer Calculation [ calcBankOffer() ] – Calculates the bank offer and return.

Control Structures

* Counted loop: Loop through every value in Cases, and:
* Binary Selection: Check if the value is greater than 0. If the value is greater or equal to 0, adds the number of the cases by 1 and adds its value to a temporary variables.
* Sequence: Then, adds the value of the selected Case and do the math to calculate the offer. (value / number \* 0.85). Return the value.

1. Final Cases – User choose one from the last two (including his case that is selected), and keep the value.

Control Structures:

* Binary Selection: check if value is in range and if the case is opened.
* Sequence: Reveal the value in the case. Then show everything.

1. showEverything() – Show every value in the case.

Control Structures:

* Counted loop: For everything in Cases,
* Binary Selection: Determine if its value is greater than 0. If true, set its value to negative.
* Sequence: Then put the selected case value back in. After that, call updateButtonsAndLabelStatus().

# IPO Chart

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| <Form load>;  List Cases;  List Labels;  List Buttons;  List LabelAscending;  List LabelsInit;  List ValuesInit; | Generate labels and buttons by looping through everything in ValuesInit, which is a set of values predefined. This then gets stored in variable Buttons, LabelsInit and LabelAscending.  Then, it randomizes values by randomly select an integer i from 0 to ValuesInit.Count() – 1, then choose LabelsInit(i), add it to Labels, and choose ValueInit(i) and add it to Cases. This ensures that the value represented in Cases(i) is consistent to the Lables(i). | Updated Cases, Labels, Buttons and LabelAscending. |
| Select one case (when handleSelection() is called);  List Cases; | Locks case by storing the value in the case to a new global variable “Selected” and then set the value of the case in the list Cases to 0. | Global Variable Selected; |
| <Button press> (Selecting other cases);  List Cases;  List BankTime; | If the status is selection, then call handleSelection();  If the status is open, then call handleOpen();  If the status is finish, do nothing;  If the status is final, call handleFinal(); | Output depends on each functions. |
| Function bank() is called;  List BankTime;  Input from message box | Determine the value of bank offer by calling function calcOffer(), and then show up a message box with yes and no.  If yes is pressed, then the game finishes and the user gets that money.  If no is pressed, then the game continues. | Output varies:  YES – The value that the user can get  NO – Nothing, redirection to updateStatus() |
| Function calcOffer() is called;  List Cases; | Loop through everything in the Cases, determine the number of non-negative values and the sum of all non-negative values, then find the bank offer by the average (sum / number) multiplied by 0.85. | Float bankOffer; |
| Function updateState() is called;  List Cases;  Variable status; | Check the status and sets the new status accordingly.  Before all those, call updateButtonsAndLabelStatus().  If status is load, then instruct user to select one to keep and update state to selection;  If status is selection, then instruct user to select ones to open and update state to open;  If status is open, then loop through the list Cases and determine the number of unopened, unselected case. If that number is in list BankTime, then switch to bank(). Otherwise, do nothing;  If status is bankfinish, then update status to open;  If status is finish, display a message box that tells the user the game is already finished;  If status is final, update status to finish. | Updated status; |
| Choose one to reveal value (Function handleOpen() is called) | Check the value in the case.  If it is a negative value, show up a message box that indicates that case is already open.  If it is 0, show up a message box that the participant is already eliminated, and end the game immediately.  If it is a positive value, set its value to negative (by multiplying it by -1), then update the label for newest value.  After all those, call function updateState() | Revealed value |
| Last round, choose one to open and keep (Function handleFinal() is called);  List Cases; | Check if the value is valid.  Open the case, sets its value to negative.  Call updateState() | Revealed value |
| updateButtonsAndLabelStatus() is called;  List Cases; | Updates buttons according to list Cases, which then updates labels that indicates the value of the cases. | Updated button values and updated label status (hidden or visible) |
| showEverything() is called;  List Cases; | Updates list Cases, substitute the value of selected case back in, and set all values to negative. This then calls updateButtonsAndLabelStatus(), which makes all of it visible. | Updated list Cases. |
| Update\_form() is called;  <Form 1 On Resize> | Recalculate elements, and adjust the form ( this is not a part of the logic part ) | Updated form |

# Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Variable Name | Data Type | Description |
| [Logic] Global  A variable that stores any all values in cases. | Cases | List of Double | This variable stores the values in cases. It changed when a case is opened, selected, or a bank offer is accepted.  A positive number means that the case is not opened, and that number is the cash value.  A negative number means that the case is opened, and the ABSOLUTE VALUE of the number is the cash value.  A zero means that case is selected, and the value can not be retrieved from this list. The value must be retrieved by accessing variable selected.  When the box is opened, selected, or when the game is finished, this value will be altered correspondingly to the operation. |
| [Logic] Global  A variable that stores all label objects, in CASE NUMBER (CASE INDEX) randomized order. | Labels | List of Label | This variable stores all label objects in CASE NUMBER (CASE INDEX) randomized order.  This ensures Cases(x) always equal to the corresponding label Values(x), so when a case is opened, it can easily get the corresponding label that represents the value.  When a case is opened, it will get the specific Label object by index (Labels(i)) and it will perform Labels(i).Hide().  This value is set in Init(). |
| [Logic] Global  A variable that stores all button objects, in CASE NUMBER (CASE INDEX) ascending order. | Buttons | List of Button | This variable stores all button objects in CASE NUMBER ascending order.  When a case is opened, it will get the specific Button object by index (Buttons(i)) and it will alter the text on the button by performing Buttons(i).Text=”Something” and it will also disable / enable button correspondingly, by using Buttons(i).Enabled=False.  This value is set in Init(). |
| [Logic] Global  A variable that stores the value of the selected case. | Selected | Double | This variable stores the value of the selected case. This is only updated when a case is selected. |
| [Logic] Global  A variable that stores all labels objects in ascending order, and is used temporary to initialize / randomize the labels. | LabelsInit | List of Label  Generated automatically by the number of the possible values, refer to ValuesInit | A variable that stores all labels objects in ascending order, and is used temporary to initialize / randomize the labels.  This variable is only used in init(), and it will be cleared after being used. |
| [Logic] Global  A variable that stores all values in ascending order, and is used temporary to initialize / randomize the values.  This variable is set manually. | ValuesInit | List of Double,  Set manually by hand to specify all possible values in the case. | A variable that stores all values in ascending order, and is used temporary to initialize / randomize the values.  This variable is only used in init(), and it will be cleared after being used. |
| [Logic] Global  A variable set manually to indicate when the bank offer should step in | BankTime | List of Integer  Set manually to specify the cases | A variable set manually to indicate when the bank offer should step in.  This variable is set manually, and it is not written (only read). |
| [Logic] Global  A control variable that controls the whole flow of the program.  This variable is very important. | State |  | The control variable that controls the whole flow of the program.  This variable is very important as a fundamental part of finite state machine.  This variable have those possible values:  Load – When the form is loaded.  Selection – When the user selects the first case  Open – When the user opens the case  Finish – When a game is already finished  Final – When the user chooses one case from the last two.  Bankfinish – When the user did not accept the bank offer, and goes back to main loop.  UpdateState and HandleSubmission use this variable to redirect the input to different functions that can handle it. For example, when state = selection, handleSubmission will redirect the selected number to handleSelection, and after a selection is handled, it will call back to updateState, which the state is updated to open (of the selection is done correctly.). |
| [Layout] Global  A variable that stores all labels, in VALUE ascending order. | LabelAscending | List of Label | This variable is used in the layout. It sets the automatic layout of the page. |
| [Layout] Global | Button\_width | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Button\_height | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Button\_up | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Button\_left | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Label\_width | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Label\_height | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Label\_up | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | Label\_left | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | reservedWidthLeft | Constant Int | Set the automatic layout of the page. |
| [Layout] Global | reservedWidthRight | Constant Int | Set the automatic layout of the page. |

# Algorithm

START

# Storyboard / Screen Design