**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

# Suggestion 3: A Chocolate Chilli Game Machine Learning Program

Write a **procedural program** that learns to win at the game of Chocolate Chilli. Players take it in turns to eat 1,2 or 3 chocolates from a pile. If there are none left they lose (eat the chilli). To see how the learning works on a similar game read https://cs4fn.blog/2025/05/19/the-sweet-learning-computer-learning-ladder/

- ❑ **Your code MUST do the right thing as specified above AND**
- ❑ **Your code MUST use the specified constructs at each level AND**
- ❑ **Your code MUST be well written as specified for each level (see the style guide)**

## Level 1 – G minus: Getting started: Input-Output

- ❑ Is a **literate program**, that you can explain, with all methods separately documented and tested as well as a final tested full version at end
- ❑ Includes **Screen output, Keyboard input**
- ❑ Defines methods doing a well-defined task and uses a sequence of **method calls**
- ❑ Comment gives at least author's name and student number at start of program.

**Example (one way to achieve the above)**: The program explains the rules and asks the person their name and how many chocolates they wish to start with. Whatever the answer is it repeats it saying that is how many virtual chocolates there are (eg "Thanks, Paul! There are 11 chocolates on the table. I will go first.").

## Level 2 – G: Making progress: Assignments and Expressions

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Defines and uses **at least one method that returns a result**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but the program now takes chocolates from the table for its turn (say 3 every time it is run), remembering the initial value but also keeping track of the current number left (so uses 2 variables for this). The main method is structured as a sequence of calls to methods: 1) call a method to describe the rules 2) call a method to ask for the number of chocolates (this method returns their answer) 3) repeat their answer "There are 11 chocolates …" 4) Call a method to say the move (take 3).

## Level 3 – F: Getting there: Decisions

- ❑ All the constructs, style and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **a method that take argument(s)**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ **Each method has a comment** stating what it does.
- ❑ All variables are defined inside methods

As above, but the program now allows one move each and checks if there are no chocolates left after each move and says who won (if none left the last person to move wins - the loser is told to eat a chilli). The machine now makes a random move taking either 1, 2 or 3 chocolates based on the "roll" of a virtual dice. (HINT: see the module workbook part 1 for how to roll a virtual dice!) Write a separate method for the dice roll. If it is not the end of the game then the human player is asked to make a move. A separate 'check for the end of the game' method is passed the current number of chocolates as argument and prints the appropriate next thing to say.

## Level 4 - D: Bare Pass: Loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Loops (no using while (true) )**
- ❑ **Well-structured** in to multiple **methods** that both take arguments and return results
- ❑ Comments included are helpful, and **each method has a comment** stating what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ All variables are defined inside methods
- ❑ Final variables are used for literal constants.
- ❑ Indentation is good making structure of program clear.

**Example (one way to achieve the above)**: As above, but now players take it in turns to eat chocolate(s) repeatedly until it is the end of the game (no chocolates left). A separate method is called for each player's turn. The new total is returned and passed to the check end method each time. The program also checks a move doesn't take the number of chocolates below 0.

**Continued overleaf**

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Level 5 - C: Pass: Arrays and Loops inside loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Arrays, accessed with a loop**
- ❑ Includes **Loops within loops**.
- ❑ Defines and uses **methods** including at least one that is passed and uses **array arguments**
- ❑ Methods individually commented about what they do and all clearly indented.
- ❑ Some variable declarations are within blocks to reduce scope to a minimum.
- ❑ Consistent use of well chosen variable names that give a clear indication of their use (perhaps making comments about them redundant). Consistent use of final variables for literal constants.

**Example (one way to achieve the above)**: As above but now the program allows the game to be played repeatedly with an array recording the winner each time. Statistics about who won most are printed out. Every game is played with the same starting number of chocolates. The arrays should be initialised by an initialisation method with the player being asked how many games they wish to play. The arrays are passed as argument to method(s) that need the information.

## Level 6 - B: Satisfactory: Records

- ❑ All the constructs, style and features above AND
- ❑ Includes **records (defined as a special class with no methods just field definitions**).
- ❑ Excellent style over comments, indentation, variable usage, final variables  etc.
- ❑ Clearly decomposed into multiple small methods doing distinct jobs that take arguments / return results

**Example (one way to achieve the above)**: As above but the program also keeps an array of records. Each array entry corresponds to a number of chocolates left so possible state of the game. Each record has three boolean fields standing for “eat one chocolate”, “eat two chocolates” or “eat three chocolates’. All are set to true initially except for at array position one in which only the value for “eat one ..” is true; and array position two in which only “eat one…” and “eat two …” are true. The program then only makes moves on its turn corresponding to the record being true for that move. Array position 0 is not used. If the program makes a move and then immediately loses then the entry for that move is set to false. The program now resigns immediately if all booleans in the record for the current position are false (as all lead to losses). It then changes the boolean flag for its last move (to false so it won’t make that move again if in the same position).

## Level 7 - A: Merit: File I/O

- ❑ All the constructs, style and features above AND
- ❑ **BOTH significant file input AND significant file output**
- ❑ Excellent style over comments, indentation, variable usage, final variables, etc.
- ❑ Program is fully composed into methods so excellent use of methods throughout
- ❑ All variables are defined in methods and have minimum scope.

**Example (one way to achieve the above)**: As above but now an option is to save the array as a file (a different file depending on the starting number of chocolates). This allows the program to be quit and restarted with the file loaded in and sequence of games continued, remembering which moves led to losses. The file I/O in your program must be at least as complex as this suggestion so you are strongly advised to follow the example not simplify it or it likely will not pass the level. ASK before doing anything different!

## Level 8 – A+: Distinction: ADTs

- ❑ All the constructs, style and features above AND
- ❑ Includes procedural programming style **abstract data type** hiding the record implementation.
- ❑ ADT has a clearly specified set of operations defined as **accessor methods** including a **create method** and **primitives for allowed operations** defining the data structure
- ❑ ADT is **commented in a single comment block**.
- ❑ All **accessor methods** of the ADT are defined in the main class.

**Example (one way to achieve the above)**: As above, but now, the records for questions are ONLY accessed via accessor methods including a create method. The only use of the dot notation is in a small number of simple, primitive accessor methods. The accessor methods are part of an Abstract Data Type with clearly specified **primitive** operations: only operations that are legal on a question record should be implemented. The ADT is clearly documented in a comment with the ADT methods. Remember there should be no OOP features used including no constructors (see the style guide). What is a winning strategy playing first?

## And then …

Keep Learning! Now carry on and make your program really **something special**, using other more advanced constructs or algorithms from the course, or something you have read up on yourself. For example, use recursive methods  in sensible ways or add search and sorting methods. Alter the learning algorithm. Restructure it to use more abstract data types. Make it a program someone would really want to use! Implement different learning rules…Use it as an excuse to learn more. Make it something special and include it in your CV / job portfolio.