Assignment Submissions

This project will be submitted in two parts. Both parts of the assessment are equally important for the successful completion of your project, so it is essential that you understand the requirements of both parts before you start.

- **Assignment 2 :: Mastermind (Part A: Project Plan)**

  Due Date: Week 6 on Saturday 7th September 2019

  Marks: This assignment will be marked out of 30 points.

  Weighting: 15% of your final mark for the unit.

  This assignment is the first part of a larger project, which you will complete in Assignment 3. This task consists of your project planning documentation. It will include details of the requirements & analysis of your program, including UML Class diagrams.

  The purpose of this assignment is to get you comfortable with planning a C++ programming project for Assignment 3. The task is detailed later in this assignment specification, as are the specific marks allocation.

- **Assignment 3 :: Mastermind (Part B: C++ Project Implementation)**

  Due Date: TBA (Week 12)

  Marks: This assignment will be marked out of 100 points.

  Weighting: 25% of your final mark for the unit.

  This assignment consists of your implementation of your project, as outlined in your Project Planning document (Assignment 2).

  Your project must follow your project plan and must be submitted as a Visual Studio project, including all header and cpp files, and any appropriate text files to ensure the program compiles and runs.

  This assignment consists of one Application file and associated custom Class files. The purpose of this assignment is to get you comfortable with designing and implementing basic multi-class C++ programs. The task is detailed later in this assignment specification, as are the specific marks allocation.

NOTE! Your submitted files must be correctly identified and submitted (as described above). Any submission that does not comply will receive and automatic **10% penalty** (applied after marking).

NOTE! Your submitted program MUST compile and run. Any submission that does not compile will automatically be awarded a **ZERO**. This means it is your responsibility to continually compile and test your code as you build it.

Submission Instructions:

This project will be submitted in two parts:

- **Assignment 2 – Part A:** consists of your project planning documentation.

  This document will include an outline of your program structure and UML Class diagrams.

  The assignment must be created and submitted as a single Word or PDF document to the Moodle site. This document must clearly identify both your Name and Student ID to facilitate ease of assessment and feedback.

  Your document file **MUST** be named as follows:

  "***A2-YourFistNameLastName***.docx" or "***A2-YourFistNameLastName***.pdf".

  This file must be submitted via the Moodle assignment submission page.

  The document should contain the project plan and the UML diagrams. You can use Microsoft Visio to draw the UML diagrams or you can use any other software, provided that the diagrams are included in your submitted document.

  Explicit assessment criteria are provided, however please note you will also be assessed on the following broad criteria:

  ✓ Detail of a proposed project plan for the overall project.
  ✓ Creating accurate and complete UML diagrams.
  ✓ Applying a solid Object-Oriented Design (OOD) for the overall project
  ✓ Using appropriate naming conventions, following the unit Programming Style Guide.

- **Assignment 3 – Part B:** consists of your implementation of your game project.

  Your project must follow your project plan and must be submitted as a Visual Studio project, including all header and code files, and any appropriate text files to ensure the program compiles and runs.

  You may complete the tasks in your preferred IDE, however you **MUST** create a Visual Studio project in order to submit. Your project folder must be identified by using your name and assignment number, such as ***YourNameA3***.

  The entire project folder must then be zipped up into one zip file for submission. The zip file **MUST** be named "**A3-*YourFistNameLastName***.zip". This zip file must be submitted via the Moodle assignment submission page.

  Explicit assessment criteria are provided, however please note you will also be assessed on the following broad criteria:

  ✓ Meeting functional requirements as described in the assignment description
  ✓ Demonstrating a solid understanding of C++ concepts, including good practice
  ✓ Demonstrating an understanding of specific C++ concepts relating to the assignment tasks, including object-oriented design and implementation and the use of Pointers
  ✓ Following the unit Programming Style Guide
  ✓ Creating solutions that are as efficient and extensible as possible
  ✓ Reflecting on the appropriateness of your implemented design and meeting functional requirements as described in the assignment description

Please ask your tutor for clarification of any aspects of the assessments you do not understand as soon as possible because this project is worth 40% of your overall marks.

Assignment 2: Master Mind (Part A)

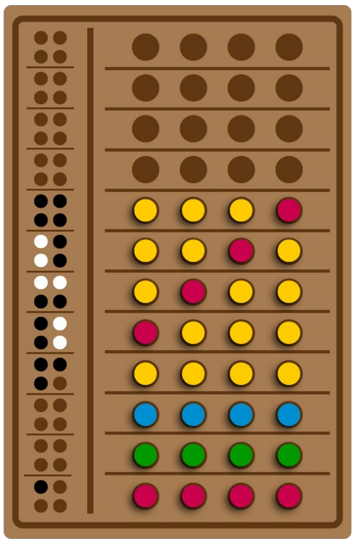
You are to implement a computer-based variant of the board game Master Mind. This is a two-player deduction game played with a specialised game board in which the aim of the game is to break the hidden code in the fewest number of moves.

You can watch a video on how to play here: https://www.youtube.com/watch?v=ZSXpmIH-_d4 or read this article: https://www.wikihow.com/Play-Mastermind or even play an online version here: http://www.webgamesonline.com/mastermind/.

In your version, you will be the code breaker, playing against the computer who is the code maker.

For Part A of the assignment you will focus on the planning of the project. In Part B you will focus on creating the various interactive objects in the game and program the player interactions.

Basic Game Play



First, the **code maker** selects any 4 coloured pegs (from 8 different colours) in any combination to create the hidden code.

The **code breaker** now attempts to determine the hidden code by selecting 4 coloured pegs at a time, and placing them in the order they think matches the hidden code.

After each guess, the code maker gives clues – a black peg for the correct colour in the correct position, a white peg for the correct colour but in the wrong position, leaving empty slots if the colours are not in the code.

The code breaker must guess the code in 10 turns or less, otherwise they lose the game.

In your version you can use words, letters, numbers or symbols for the code.

Human Player:

You, as the human player, must be able to input your choices for the hidden code as required by the program. For example: if the program uses numbers for the code then you must enter 4 numbers as your guess. After you enter your guess, the board will update, showing you the accuracy of your guess. You continue until you have cracked the code (you win) or you have run out of turns (you lose).

Computer Player:

Obviously, you as a player can make strategic choices as to which colours to use when guessing the hidden code. However, programming the computer breaking the code is beyond the scope of this assignment. Therefore, the computer will be assigned the role of code maker and the player will be the code breaker (see Assignment 3 brief for more details on this).

Project Plan

Having a clear plan for your project before you begin coding is essential for finishing a successful project on time and with minimal stress. So part of this assignment is defining what will be in your project and what you need to do to actually develop it.

> ***Important:*** *You must also read the requirements for Assignment 3 in order to be able to complete the documentation required for Assignment 2.*

The documentation you must submit will include the following:

✓ **A description of "How to Play"**

This will be the information you display on the opening screen of your game to inform the player what they need to do in order to play and win the game. Do not just copy the game play description above, make your own description.

Note: This description must be saved as a text file and read into your game when the program begins. You can also use this as a convenient way to display "help" if the player asks for it.

✓ **A development outline of your game**

Using a simple outline format (numbered or bullet points) state the main actions that the program will have and then, as sub-points, state the things you will need to do to make that happen.

The outline structure should contain all the elements of your game, as this is a high level description of your approach to the development of your program. You should include at least the following headings and provide examples of happens under each section.

- The game setup (everything that happens before the game starts)
- The player's turn (the sequence of events that happen during a turn)
- Processing player input (include each of the commands your player can use)
- Providing feedback to the player (in response to the player's interactions)
- The end game conditions (include all win and lose conditions)
- Additional Features included, if any – see Assignment 3
- Outline the functionality of all your game classes – see Assignment 3

Here is an example to get you started with your project outline:

- The Game Setup
  - Display an overview of the game rule so the player knows what to do to win.
    - read this information from a text file
  - Initialise the game elements:
    - add the player – ask for the player's name, set default variables
    - all the other things that will happen during initialisation including
      - creating the game board
      - initialising other game variables (list them here)

As you can see, you only have to describe the actions the program will take, not the code, for the outline. The idea here is to give you a starting point for when you start writing your code as you can use this as a checklist of the things you need to include.

✓ **UML *Diagrams***

UML diagrams are designed to make structuring your program easier. How to create them will be covered in class, but the general structure is shown here – see Assignment 3 for more details about classes.

You will need UML diagrams for each of the classes you include in your game – at least a Player, Board and Application (main) class.

ClassName
list of attributes (variables)
list of behaviours (functions)

Assignment 2: Marking Criteria [30 marks in total]

Project Plan [25]

- Description of the rules (the introduction to your game) **[2]**
- Outline includes all game functionality (from Part B) **[3]**
- Each section is broken into logical tasks **[5]**
- Tasks are performed in a logical order **[5]**
- Task descriptions given provide sufficient detail **[10]**

UML Diagrams [5]

- Correct structure used (Name, Attributes, Behaviours) **[1]**
- Included the correct designations for public (+) and private (-) data **[2]**
- All variables and functions have meaningful names **[2]**

Assignment 3: Master Mind (Part B)

You are to implement the Master Mind game you started in Assignment 2 by creating a Visual Studio Project using your project plan as described in your previous submission.

Your completed Master Mind game must demonstrate the following:

- ✓ You MUST implement your program using the following classes, as a minimum, you may include more (as appropriate for your game design):
  - **Player class:** holds a player's details including their name, score, wins and the number of games played.
  - **Board class:** holds the current game's details including each of the player's previous moves and associated clues, the number of rows and columns used in the game.
  - **Application file:** holds the main() function and controls the overall flow of the game.

  You may include other relevant attributes and behaviours to these classes, as identified in your project plan.
- ✓ The **Player** must be able to do the following:
  - assign a name which is requested at the start of the game and used in the feedback given
  - see all their previous moves at the start of their turn so they can make an informed choice
  - type in a code, using either letters or numbers and seeing appropriate feedback on the accuracy of their input
  - continue typing in codes and seeing feedback until they either "crack the code" or run out of turns
  - quit the game at any time – during or after a game
- ✓ The **Board** in the game should have the following characteristics:
  - a defined number of columns (code slots) and rows (the maximum turns a player has)
  - be able to display each code entered and given feedback as a specific row
  - be able to reset the board to empty at the end of a game
  - not be able to add any more rows beyond the specified number
- ✓ The **Game Application** must do the following:
  - display the "how to play" information at the start of the game

- create or select a secret code – using words/letters, symbols and/or numbers
- display an appropriate and uncluttered user interface providing relevant information to the player at all times
- ask for and allow the player enter a code
- compare the player's code to the secret code and provide appropriate feedback to the player (a symbol for a correct letter/number in the correct position and a different symbol for a correct letter/number but in the wrong position, it is optional to use another symbol if the letter/number is not in the code)
- display the updated game board after each code entered by the player
- terminate the game (player wins) when the player has guessed the code
- terminate the game (player loses) when the player has run out of turns
- provide player stats at the end of the game (wins, loses and score)
- the player should be able to QUIT the game at any time

Program Reflection

You must also provide a 300-word written reflection of your object-oriented design and how well you believe it was to implement. You should cover the following areas:

- Discuss why you designed it the way you did.
  - Why did you create your classes that way?
  - How does this reflect an OO design or approach?
- Discuss how well you were able to code it
  - Highlight any issues you found once you tried to implement your design.
  - How did you resolve these issues?
- If you were to do this project again, discuss how you might change your design to make your solution easier to implement, more efficient, or better for code reuse.

This must be a Word or PDF document which must be included in the same folder as your **.sln* file. Your document file **MUST** be named as follows:

"***A3-YourFistNameLastName***.docx" or "***A3-YourFistNameLastName***.pdf".

Extra Functionality

The marking criteria indicates that you should make some individual additions to this in order to achieve the final 20% of the mark.

Following is a list of additional features you can include, with the maximum number of marks [x] you can earn for each one. You may implement one or more features from the list, but you will only score up to the maximum of 20% of the total assignment marks or 20 marks.

You should aim to add some additional creative elements to the gameplay, as well as advanced object-oriented design elements or advanced use of pointers.

- Add a theme to the game which is incorporated into the game play through the story and the feedback given (for example: Word Detective, Number Cruncher, etc.) **[2]**
- The player can select a skill level (eg: Novice, Thinker and Master Mind) which modifies the game parameters – the number of elements in the code and the number of rows (turns) the player has to solve it. For example: easy has 4 elements out of 6 with 10 rows, tricky has 5 elements out of 8 with 12 rows and hard has 6 elements out of 10 with 14 rows. **[2]**
- Implement an appropriate scoring system based on the game parameters – number of turns taken, a bonus for unused rows, +/- point based on a win or a loss, etc. **[3]**

- Read in word and/or element lists from a file (based on word length, such as 4-, 5- and 6-letter words, numbers or symbols) and store appropriately. Word lists should have at least 20+ words of each length. The list used in the game could be linked to a player skill level and/or randomly selected at the start of each game. **[3]**
- Give the player additional options apart from just typing the code or quitting the game. This could include options to: ask for help (displays an information screen loaded from a file), ask for a hint (displays a random element from the secret code and must be a limited option), ask to resign from the current game but not quit the entire game (ends the current game as if the player ran out of turns and displays the secret code). **[4]**
- Display the board using ASCII art. **[4]**
- Allow the game to be saved and restored at the player's request. **[4]**
- Allow the player to select what elements will be used in the code (numbers, symbols, letters, words). Words should range from 4- to 6-letter words, while a specific range of numbers or set of letters/symbols should be clearly identified and displayed to the player. The feedback symbols can remain the same. **[5]**
- Ask the player if they want to be promoted to the next skill level after winning 5 games. If they accept their skill level is increased by one (to the maximum level permitted in your game). Also, if the player loses 5 games in a row they are automatically demoted one skill level (to the minimum level permitted in your game). **[5]**
- Implement a more sophisticated computer player. The computer player should be able to have its own Player object (including name and any other player stats as required for your game), and score the points the human player would lose plus earn a suitable bonus for the win. It should also make intelligent choices for words/letters, symbols or numbers including using doubles or triples of the same letter/symbol/number. It should also have the ability to give random comments to the player based on how well (or not) the player is doing. **[10]**

You certainly do not have to implement all of the above to earn marks for extra functionality. Just remember the maximum number of marks you can earn are given in [x]. It is up to you!

Assignment 3: Marking Criteria [up to 100 marks in total]

Does the program compile and run? Yes or No

- **Zero marks will be awarded for a non-compiling program.**

Class Design [15]

- Player Class **[5]**
  - Has an appropriate header file [2]
  - Required data members and member functions using meaningful names [1]
  - Contains only aspects that relate to a "player" (has no data members or member functions that are not directly related to a Player) [2]
- Board Class **[5]**
  - Has an appropriate header file [2]
  - Required data members and member functions using meaningful names [1]
  - Contains only aspects that relate to a "board" (has no data members or member functions that are not directly related to the board) [2]
- Game Application **[5]**
  - Has an appropriate header file [2]
  - Has appropriate variables and functions using meaningful names [2]
  - The main() function has appropriate function calls to keep it uncluttered [1]

Functionality [35]

- Game set up including: displaying "how to play" information, initialising the player and game variables, creating the secret code using appropriate data, etc. **[5]**
- Implementation of a clear and uncluttered User Interface display **[5]**
- Successful implementation of action processes **[5]**
- Successful implementation of calculating and displaying the correct clues **[5]**
- Appropriate feedback to player's interactions **[5]**
- Appropriate feedback displayed to the player **[5]**
- Appropriate end game conditions triggered **[5]**

Quality of Solution and Code [20]

- Does the program perform the functionality in an efficient and extensible manner? **[12]**
  - Appropriate use of functions and function calls [1]
  - Appropriate use of data types [1]
  - Appropriate use of decisions, loops and other programming techniques [2]
  - Appropriate use of references and/or pointers [5]
  - Appropriate use of good programming practices and techniques [2]
  - Extraneous and redundant code removed [1]
- Has a well-designed OO program been implemented? **[4]**
  - Contains classes appropriate to the assignment brief [3]
  - Program structures support and OO design [1]
- Has the Programming Style Guide been followed appropriately? **[4]**
  - Appropriate commenting and code documentation [2]
  - Correct formatting of code within *.h and *.cpp files [2]

Extra Functionality [maximum 20]

- Incorporate a theme into the story and feedback given to the player **[2]**
- Implement difficulty levels which may be selected by the player **[2]**
- Implement an appropriate scoring system based on game parameters **[3]**
- Read word lists from a file and store appropriately **[3]**
- Player has additional command choices **[4]**
- Display the board using ASCII art **[4]**
- Allow the game to be saved and restored at the player's request **[4]**
- Allow player to select which elements to use **[5]**
- Player promotion/demotion every 5 games won/lost **[5]**
- Implement a more sophisticated computer player **[10]**

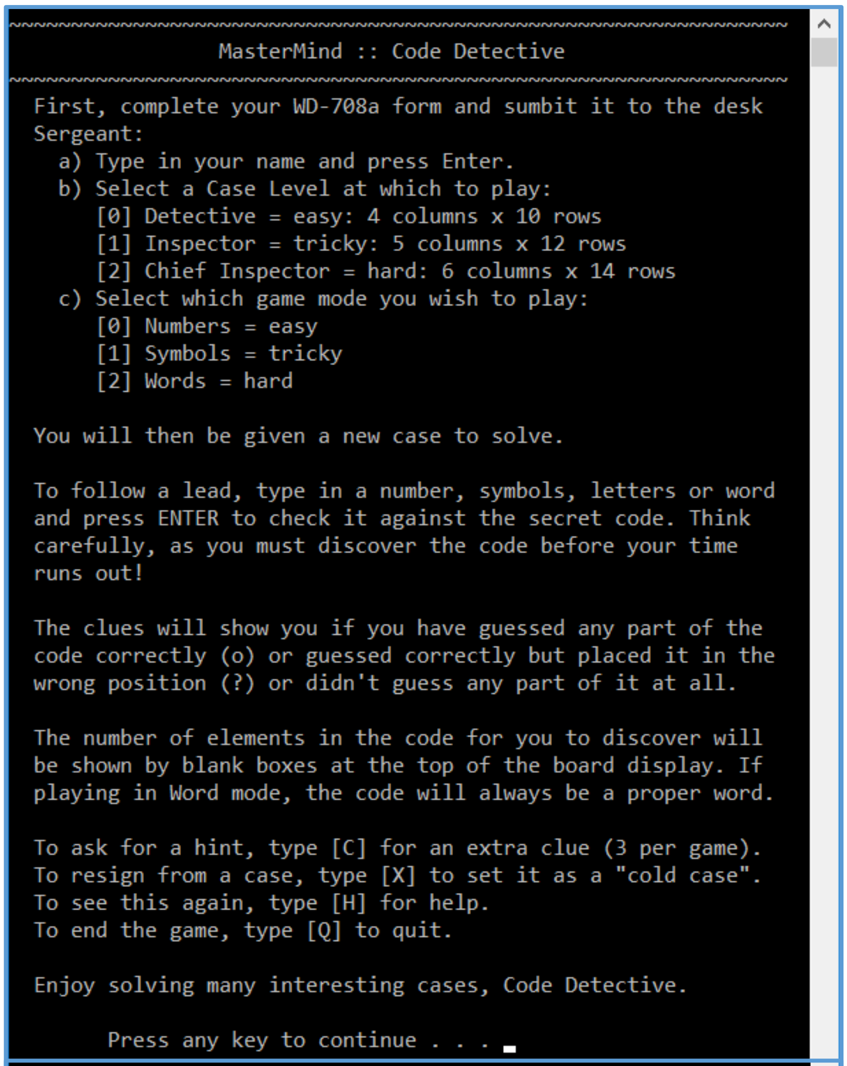
Reflection [10]

- Discussion of motivations for the program design [3]
- Discussion of how well the design was to implement [3]
- Discussion of what they would do differently if they were to start it again [4]

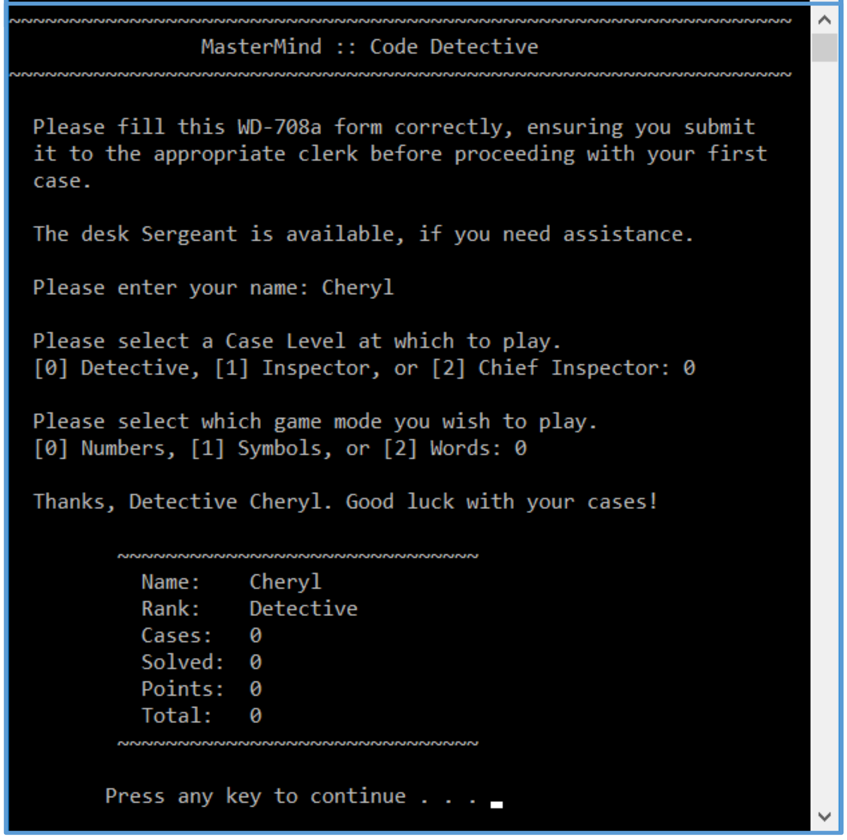
Assignment Notes:

It is your responsibility to know what is expected of you. If you are unsure about anything, ask your tutor sooner rather than later. Write any questions and/or notes here to help you clarify what you have to do.

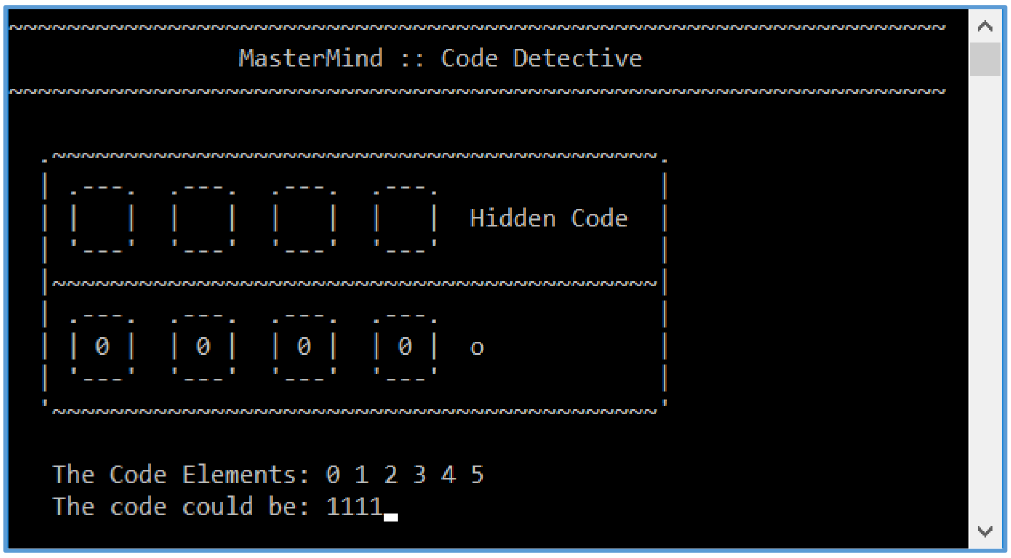
Here are some sample screen shots to help you develop your user interface:



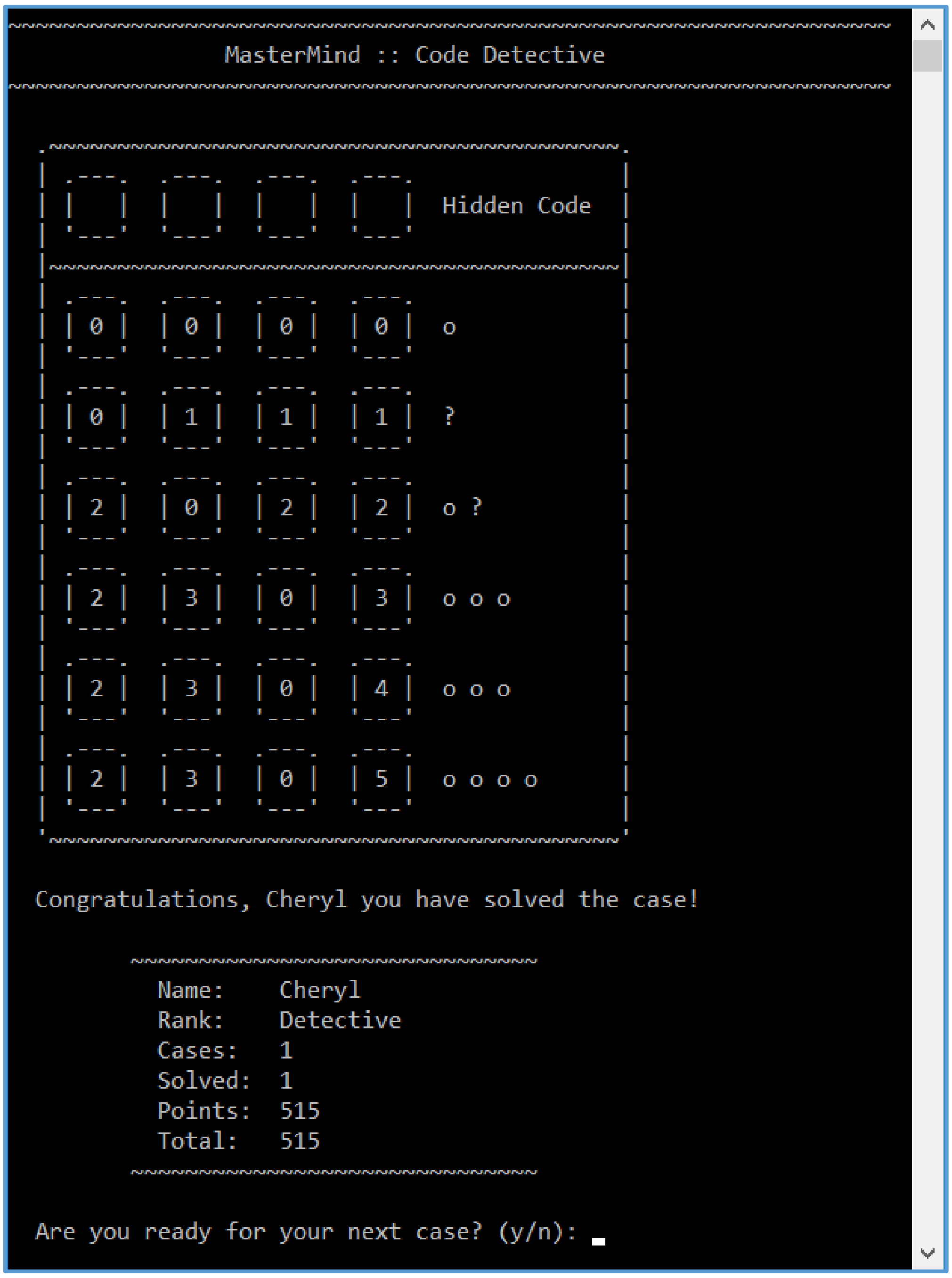
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                    MasterMind :: Code Detective
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  First, complete your WD-708a form and sumbit it to the desk
  Sergeant:
    a) Type in your name and press Enter.
    b) Select a Case Level at which to play:
       [0] Detective = easy: 4 columns x 10 rows
       [1] Inspector = tricky: 5 columns x 12 rows
       [2] Chief Inspector = hard: 6 columns x 14 rows
    c) Select which game mode you wish to play:
       [0] Numbers = easy
       [1] Symbols = tricky
       [2] Words = hard

  You will then be given a new case to solve.

  To follow a lead, type in a number, symbols, letters or word
  and press ENTER to check it against the secret code. Think
  carefully, as you must discover the code before your time
  runs out!

  The clues will show you if you have guessed any part of the
  code correctly (o) or guessed correctly but placed it in the
  wrong position (?) or didn't guess any part of it at all.

  The number of elements in the code for you to discover will
  be shown by blank boxes at the top of the board display. If
  playing in Word mode, the code will always be a proper word.

  To ask for a hint, type [C] for an extra clue (3 per game).
  To resign from a case, type [X] to set it as a "cold case".
  To see this again, type [H] for help.
  To end the game, type [Q] to quit.

  Enjoy solving many interesting cases, Code Detective.

        Press any key to continue . . .
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                  MasterMind :: Code Detective
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  Please fill this WD-708a form correctly, ensuring you submit
  it to the appropriate clerk before proceeding with your first
  case.

  The desk Sergeant is available, if you need assistance.

  Please enter your name: Cheryl

  Please select a Case Level at which to play.
  [0] Detective, [1] Inspector, or [2] Chief Inspector: 0

  Please select which game mode you wish to play.
  [0] Numbers, [1] Symbols, or [2] Words: 0

  Thanks, Detective Cheryl. Good luck with your cases!

         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
           Name:    Cheryl
           Rank:    Detective
           Cases:   0
           Solved:  0
           Points:  0
           Total:   0
         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        Press any key to continue . . .
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                MasterMind :: Code Detective
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  .~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~.
  | .---.  .---.  .---.  .---.               |
  | |   |  |   |  |   |  |   |  Hidden Code  |
  | '---'  '---'  '---'  '---'               |
  |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
  | .---.  .---.  .---.  .---.               |
  | | 0 |  | 0 |  | 0 |  | 0 |  o            |
  | '---'  '---'  '---'  '---'               |
  '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'


   The Code Elements: 0 1 2 3 4 5
   The code could be: 1111
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                MasterMind :: Code Detective
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  .~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~.
  | .---.  .---.  .---.  .---.               |
  | |   |  |   |  |   |  |   |  Hidden Code  |
  | '---'  '---'  '---'  '---'               |
  |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
  | .---.  .---.  .---.  .---.               |
  | | 0 |  | 0 |  | 0 |  | 0 |  o            |
  | '---'  '---'  '---'  '---'               |
  | .---.  .---.  .---.  .---.               |
  | | 0 |  | 1 |  | 1 |  | 1 |  ?            |
  | '---'  '---'  '---'  '---'               |
  | .---.  .---.  .---.  .---.               |
  | | 2 |  | 0 |  | 2 |  | 2 |  o ?          |
  | '---'  '---'  '---'  '---'               |
  | .---.  .---.  .---.  .---.               |
  | | 2 |  | 3 |  | 0 |  | 3 |  o o o        |
  | '---'  '---'  '---'  '---'               |
  | .---.  .---.  .---.  .---.               |
  | | 2 |  | 3 |  | 0 |  | 4 |  o o o        |
  | '---'  '---'  '---'  '---'               |
  | .---.  .---.  .---.  .---.               |
  | | 2 |  | 3 |  | 0 |  | 5 |  o o o o      |
  | '---'  '---'  '---'  '---'               |
  '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'

  Congratulations, Cheryl you have solved the case!

         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
           Name:    Cheryl
           Rank:    Detective
           Cases:   1
           Solved:  1
           Points:  515
           Total:   515
         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  Are you ready for your next case? (y/n):
```