|  |  |  |
| --- | --- | --- |
| **Software Implementation Checklist** | **Faculty of Computing, Engineering and the Built Environment** | New Logo Tiny |

|  |  |
| --- | --- |
| Please fill in your name and student ID in the table below. | |
| **Student Name** | Mohammad Moghadaszadeh Behbahani |
| **Student Number** | 21174353 |
| **Course and Year** | MSc. Computer Science 2021-22 |
| **Module Code** | CMP7244 |
| **Module Title** | Software Development |
| **Module Leader** | Dr Aliyuda Ali |
| **Assessment item:** | 1.2 A game program development in Python |

|  |  |  |  |
| --- | --- | --- | --- |
| Students are required to complete this software implementation checklist for their Python game. First, you need to tick alongside the name of the game chosen for the assessment 1.2. Then, select only the features that were implemented in your code. You can select all/some features from any marking range as long as they have been implemented in the code submitted for this assessment. For example, you can select all/some features from the marking range 40%-49% and all/some features under the range 70%-79% etc. This implementation checklist should be submitted with the test documentation for the assessment 1.2.  **Important notice**: This checklist will assist the tutors when marking your code, hence, you should only select the feature requirements that are implemented in your code. Even if some features are not working correctly, you can select them as long as there is evidence in your code showing the implementation attempt. However, it is not acceptable for a student to claim the implementation of features that were not attempted/implemented in the game. False claims are a clear indication that the student does not understand the submitted code, hence, the submission will be investigated further for plagiarism, and the tutor marking the assessment may invite the student to explain all/parts of the submitted code. | | | |
| 1. **Marking criteria for the implementation of the Battleship game** | **🞎** | | |
| **Achieving a mark of 40% to maximum of 49%** | | | |
| The game **must** implement **all** the following: | | | |
| * Two players have two grids. | | | **🞎** |
| * Each player can secretly position the ships on its primary grid | | | **🞎** |
| * Each ship can position 5 types of ships either horizontally or vertically on the grid | | | **🞎** |
| * Players take turns to target opponent's grid. If hit the grid changes its colour to red, otherwise it turns to white for miss. | | | **🞎** |
| * When a player hits a hidden ship, the player is given an extra turn until it is a miss. | | | **🞎** |
| * Text based user interface is used to play the game. | | | **🞎** |
| * Computer player plays randomly. | | | **🞎** |
| **Achieving a mark of 50% to maximum of 59%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Development of Basic Graphical User Interface (GUI) using Python Tkinter. | | | **🞎** |
| * The game should announces the losing player and type of ship that was sunk. | | | **🞎** |
| * The game should announce the winning player when all opponent’s ships have sunk. | | | **🞎** |
| * Computer player should allow some time before it targets opponent’s grid. 2 seconds, emulating its thinking period even if the next move is random. | | | **🞎** |
| **Achieving a mark of 60% to maximum of 69%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Players can select the size of the grid at the beginning of the game. | | | **🞎** |
| * The game must validate the positions of the ships. For example, the ships cannot overlap (i.e., only one ship can occupy any given square in the grid). | | | **🞎** |
| * Ships should only be placed within the game grid. | | | **🞎** |
| * The game can be saved and loaded into and from a text file. | | | **🞎** |
| **Achieving a mark of 70% to maximum of 79%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Computer player uses random moves, but when there is a hit, it should continue hitting the ship by hitting adjacent grids laid horizontally or vertically on the gird. | | | **🞎** |
| * Computer player should consider the type of ship and its size when implementing the above functionalities. | | | **🞎** |
| **Achieving a mark of 80% and over** | | | |
| The game **must** implement **all** the above and the following: | | | **🞎** |
| * Computer player should exhibit some intelligence. For example, learning from its previous moves when targeting new grids. | | | **🞎** |
| 1. **Marking criteria for the implementation of the Connect4 game** | **🞎** | | |
| **Achieving a mark of 40% to maximum of 49%** | | | |
| The game **must** implement **all** the following: | | | |
| * Two players play on a single seven-column, six-row vertically suspended grid. | | | **🞎** |
| * Players should select a colour before the game starts. | | | **🞎** |
| * Players take turns to select the column in which the player wants to drop its coloured/symbol disc. The pieces fall straight down occupying the next available space inside the column. | | | **🞎** |
| * The game must validate the positions of the discs so they do not overlap. | | | **🞎** |
| * The game should announce the winning player when a plyer connects four of its own discs of the same colour next to each other *vertically* or *horizontally*. | | | **🞎** |
| * Text based user interface is used to play the game. | | | **🞎** |
| * Computer player plays randomly. | | | **🞎** |
| **Achieving a mark of 50% to maximum of 59%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Development of Basic Graphical User Interface (GUI) using Python Tkinter. | | | **🞎** |
| * The game can be restarted at any stage of the game. | | | **🞎** |
| * The game should announce the winning player who connects four of its own discs next to each other *diagonally*. | | | **🞎** |
| * Computer player should allow some time before it insert its disc e.g. 2 seconds, emulating its thinking period even if the subsequent move is random. | | | **🞎** |
| **Achieving a mark of 60% to maximum of 69%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| 1. Players can select the size of the grid at the beginning of the game. | | | **🞎** |
| 1. The game can be saved and loaded into and from a text file. | | | **🞎** |
| 1. Visualise the motion of the disc falling until it occupies the next available space in the game column. | | | **🞎** |
| 1. Two human players can play against each other taking turns. Also, two computer players should play against each other. | | | **🞎** |
| **Achieving a mark of 70% to maximum of 79%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| 1. Computer player associate random moves with some kind of techniques trying to connect 4 discs in any direction. | | | **🞎** |
| 1. Computer player should implement strategies to prevent human player from winning. | | | **🞎** |
| **Achieving a mark of 80% and over** | | | |
| The game **must** implement **all** the above and the following: | | | |
| 1. The game should have 3 levels of difficulty, the human player can chose from. | | | **🞎** |
| * 1. Easy – completely random | | | **🞎** |
| * 1. Medium – Computer player associate random moves with some kind of techniques trying to connect4 discs in any direction. | | | **🞎** |
| * 1. Advanced – Computer player implement some intelligence. For example, before it makes the move, it will look ahead at the possible sequences of moves the computer and the opponent could make, based on all those sequences of move it will determine which first move is best. | | | **🞎** |
| 1. **Marking criteria for the implementation of the Mastermind game** | | **🞎** | |
| **Achieving a mark of 40% to maximum of 49%** | | | |
| The game **must** implement **all** the following: | | | |
| * One player plays on a single four-column, ten-row vertically suspended grid. | | | **🞎** |
| * The computer (the code maker) should randomly select four colours before the game starts, and these should be kept hidden from the player. Do not allow duplication of colours. | | | **🞎** |
| * When player complete each guess, by filling four columns of a single row, the computer checks if the guessed colours matches hidden code. If the colours matched, the player wins, otherwise the player is given another chance only if the guessed row is located before the final row of the Mastermind board. | | | **🞎** |
| * The game should announce the “Game is over” and reveals the secret code when a player reaches the final row without guessing the correct code. | | | **🞎** |
| * Text based user interface is used to play the game. | | | **🞎** |
| * Computer player plays randomly. | | | **🞎** |
| **Achieving a mark of 50% to maximum of 59%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Development of Basic Graphical User Interface (GUI) using Python Tkinter. | | | **🞎** |
| * The game can be restarted at any stage of the game. | | | **🞎** |
| * After each guess, the computer responds by placing 0, 1, 2, 3, and 4 Key Pegs in the Key Peg holes located beside the guessed row, and these Pegs should follow the following colour convention: | | | **🞎** |
| * + A red Key peg for correctly guessed colour and in the right position (without indication of which Code Peg it corresponds to). | | | **🞎** |
| * + A white Key Peg for correctly guessed colour but in the wrong position. | | | **🞎** |
| * + No Key Peg to indicate a wrong colour that does not match any colour in the secret code. | | | **🞎** |
| **Achieving a mark of 60% to maximum of 69%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * The game can be saved and loaded into and from a text file. | | | **🞎** |
| * Random computer player should play against the computer code maker. | | | **🞎** |
| * Time delay should be introduce in-between computer guesses to allow easy observation of the computer moves. | | | **🞎** |
| **Achieving a mark of 70% to maximum of 79%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Column length is configurable, hence the number of columns can be between 4, 6 and 8. Of course the longer the length of the row, the more difficult the guess it becomes. Note, the number of Key Pegs should correspond to the number of columns. | | | **🞎** |
| * Computer code breaker should associate random moves with some kind of rules to increase its chances of winning. | | | **🞎** |
| **Achieving a mark of 80% and over** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * The computer player should have 3 levels of intelligence. | | | **🞎** |
| * 1. Simple - completely random | | | **🞎** |
| * 1. Medium - Computer code breaker should associate random moves with some kind of rules to increase its chances of winning. | | | **🞎** |
| * 1. Advanced – Computer player implement AI algorithms such as using history data and/or scoring system. For example, before the computer player makes a guess, it will look at the Key Pegs and previous guesses to determine future guesses. | | | **🞎** |
| 1. **Marking criteria for the implementation of the Minesweeper game** | | **🞎** | |
| **Achieving a mark of 40% to maximum of 49%** | | | |
| The game **must** implement **all** the following: | | | |
| * One player plays on a single 9 \* 9 grid. | | | **\*** |
| * The computer should randomly select 10 grids, and secretly insert mines inside these grids. | | | **\*** |
| * The player should navigate the grid by selecting the grid to reveal its hidden content. | | | **\*** |
| * If a square contains a mine the player loses the game, and the rest of the grid is revealed showing the locations of the rest of the mines. But, if the grid contains no mine, the grid is tagged with a colour or character to indicate a clear field. This grid cannot be selected again throughout the game. | | | **\*** |
| * The game should display the number of hidden mines, and count the number of clear grids discovered by the player. | | | **\*** |
| * The game should announce if the player wins the game that is revealing all clear grids. | | | **\*** |
| * Text based user interface is used to play the game. | | | **🞎** |
| * Computer player plays randomly. | | | **\*** |
| **Achieving a mark of 50% to maximum of 59%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Development of Basic Graphical User Interface (GUI) using Python Tkinter. | | | **\*** |
| * The game can be restarted at any stage of the game. | | | **\*** |
| * While navigating the game, if the selected grid contains no mine, a digit is displayed in the square, indicating how many adjacent squares contain mines. On the other hand, if no mines are adjacent, the square becomes blank, and all adjacent squares will be recursively revealed. | | | **\*** |
| **Achieving a mark of 60% to maximum of 69%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * The game can be saved and loaded into and from a text file. | | | **\*** |
| * To help avoid hitting a mine, the grid of a suspected mine can be marked by flagging it. The grid can also be unmarked. | | | **\*** |
| * The grid size should be configurable, i.e. the player should be able to select the number of columns, rows and mines before the start of the game. | | | **\*** |
| * Random computer player should play the game by navigating the grid. | | | **\*** |
| * Time delay should be introduced between each computer guess to observe computer moves. | | | **\*** |
| * Time delay should be introduced in-between computer guesses to allow easy observation of the computer moves. | | | **\*** |
| **Achieving a mark of 70% to maximum of 79%** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * Computer player should associate random moves with some kind of rules to increase its chances of winning. | | | **\*** |
| **Achieving a mark of 80% and over** | | | |
| The game **must** implement **all** the above and the following: | | | |
| * The computer player should have 3 levels of intelligence. | | | **\*** |
| * + Simple – completely random | | | **\*** |
| * + Medium - Computer player should associate random moves with some kind of rules to increase its chances of winning. | | | **\*** |
| * + Advanced – Computer player should implement AI algorithms. For example, using history data and/or scoring system. For example, before the computer player selects a grid, it will look at the digits to determine selection. | | | **\*** |