**Rubric for Assessing 3 Stones in C++**

Name: \_\_John Warren\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Carefully **highlight** **all** the items that **work correctly**. Incorrect entries may be penalized. Not all the entries may be used for grading.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Setup of the Game** | | | | | |
| **Players** | One player is Human | One player is computer | Players alternate |  |  |
| **Setup** | Board consists of 80 pockets | Pockets are laid out in 11 X 11 grid | Center pocket is missing |  |  |
|  | Each player starts with 15 white, 15 black and 6 clear stones |  | Each player starts with score of 0 |  |  |
| **Serialization** | Provides option to stop game after each turn | Game is saved into text file | Correct format used for text file | Game saved correctly into text file | Game quits upon serialization |
|  | Provides option to resume game from text file | Prompts for name of text file | Reads text file |  |  |
|  | Board is correctly restored | Stones of human correctly restored | Stones of computer correctly restored | Scores of players correctly restored | Next player correctly restored |
| **Playing the Game** | | | | | |
| **Picking color** | A coin is tossed | Human is asked to call the toss | If human calls correctly, human is asked to pick the color | If human guesses incorrectly, computer picks color | Other player gets the other color |
| **First player** | Player who picked black plays first |  |  |  |  |
| **Human’s play** | Can place only one stone per turn | Can only play a stone possessed by the player | Can place stone in empty pocket | Cannot place stone in occupied pocket | If first turn, can place stone in any empty pocket |
|  | Can play on the same row as last play | Prevents playing on different row than last play | Can play on the same column as last play | Prevents playing on different column than last play | If no empty pocket on row and column, can place stone in any empty pocket |
|  | Gets 1 point for 3 adjacent stones in a row | Gets 1 point for 3 adjacent stones in a column | Gets 1 point for 3 adjacent stones in descending diagonal | Gets 1 point for 3 adjacent stones in ascending diagonal | Gets no point otherwise |
|  | Gets point only if all three are player’s color |  | Gets point even if up to 2 stones are clear | Gets no point if all 3 stones are clear |  |
|  | A stone can be part of more than one arrangement | If so, player can get more than 1 point when stone is played |  | If stone is clear, both players can get a point from the same play |  |
|  |  | Gets points for only *new* arrangements resulting from a play |  |  | Player’s count of remaining stones is correctly updated |
| **Help Mode** | Has option to ask computer for a recommended play | Computer displays all possible plays for human player | Computer recommends one “best” play: both the color and pocket | Computer uses its own strategy to recommend “best” play for human player | Computer lists the strategy used for the recommendation |
| **Computer’s play** | Will place one stone per play | Will play a stone possessed by the player | Will place stone in empty pocket | Will place stone on the same row as last play | Will place stone on the same column as last play |
|  | Explains the color and location of the stone | Lists the strategy used to place the stone |  |  |  |
|  | Gets 1 point for 3 adjacent stones in a row | Gets 1 point for 3 adjacent stones in a column | Gets 1 point for 3 adjacent stones in descending diagonal | Gets 1 point for 3 adjacent stones in ascending diagonal | Gets no point otherwise |
|  | Gets point only if all three are player’s color |  | Gets point even if up to 2 stones are clear | Gets no point if all 3 stones are clear |  |
|  | A stone can be part of more than one arrangement | If so, player can get more than 1 point when stone is played |  | If stone is clear, both players can get a point from the same play |  |
|  |  | Gets points for only *new* arrangements resulting from a play |  |  | Player’s count of remaining stones is correctly updated |
| **Computer’s Strategy** | Generates all possible plays | Describes the best play it executes | Uses non-trivial strategies to select the next play | Describes the strategy used |  |
| **The color of the stone placed** | *Please enter your strategy* | | | | |
| **The pocket in which the stone is placed** | *Please enter your strategy* | | | | |
| **Round Completion** | Round ends when the last stone is placed | Announces the number of points for both the players | Player with the most points wins the round | Announces the winner of the round | If both players have same number of points, game is called a draw |
| **Tournament Control** | At the end of a round, asks human whether another round should be played | If yes, another round is started |  | Correctly keeps track of the number of rounds won by both players |  |
|  | If no, announces the number of rounds won by both players | Player who won the most rounds is the winner | Announces the winner of the tournament |  | Program exits after announcing winner of the tournament |
| **Game features** | | | | | |
| **Validates input from human player** | Input on calling the coin toss |  | Input on choosing the color to play |  |  |
|  | The color of the stone to be placed | The row on which to place the stone | The column on which to place the stone |  | Asking for play recommendation from the computer |
|  | Input on whether to start a game using a text file | Input of the text file name |  | Input on whether to suspend a game after a turn | Input of the name of the file in which to save the game |
|  | Input on whether to start a new round |  |  |  |  |
| **Output** | Board is correctly displayed | Rows and columns are numbered | Board is correctly updated after each play |  |  |
|  | Stones of each player correctly displayed |  | Score of each player correctly displayed |  | Number of rounds won by each player correctly displayed |
|  | Computer’s play is described in user-friendly format |  | Computer’s recommendation to human player displayed in user-friendly format |  |  |
| **Design** | | | | | |
| **Object-oriented design** | At least 7 classes are included | Each class is complete – self-contains all the necessary functionality | Inheritance is used for player classes: computer and human inherit from player class | Virtual functions used for player classes |  |
| **Code Design – Data flow** | Data: Only independent variables saved, dependent variables saved sparingly, only for efficiency | Data is *not* saved redundantly, no potential fidelity problems in data storage | Data is encapsulated – access to data is controlled | Changes to data always validated |  |
| **Code Design – Control flow** | Overall design is hierarchical, and is evident in main() | Code for repeated execution separated from  code for single execution (e.g., of round, game) | | Display issues separated from problem logic (Model Vs View) |  |
| **Code Reuse** | Code properly factored out of if-else, loops | Functions defined for any code executed more than once | Each function in charge of only one logical task |  |  |
| **Implementation** | | | | | |
| **Board Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **BoardView Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Player Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Human Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Computer Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Round Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Tournament Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation | Destructor releases resources |
| **Identifiers** | All variables have names corresponding to nouns in the problem description | All classes have names corresponding to nouns in the problem description | All client functions have names corresponding to verbs in the problem description | Any abbreviations in the names are readable |  |
| **Coding style** | No global variables used | Symbolic constants are used whenever possible | All literal constants are explained at *each* occurrence | Principle of least privilege used for parameter passing |  |
| **Courtesy Programming** | | | | | |
| **Listing** | Code is indented properly |  | Client functions listed in the order in which they are called | Classes are listed from basic to composite and derived | Each class listed in the following order: public, protected and private |
| **Documentation** | Every function has a complete header | Within each function, code is properly commented – steps in the algorithm are listed | Comments in the code describe semantics, not syntax | Comments in the code do not have spelling/ grammatical errors. |  |
| **Submission** | | | | | |
| **Screen shots of:** | First player of the round being determined | Computer’s move being explained | Computer providing help |  | Winner of the tournament being announced |
| **Manual includes:** | Bug report | Missing features report |  | Project log | Help from Generative AI |
|  | Description of data structures | Description of classes |  | Source, documentation and rubric are placed in a directory with your full name and the directory is zipped | |
| **Milestones uploaded?** | First |  | Second | | |

*Do not delete these pages*