Practical Software Engineering I.

Exercises for the 2nd assignments

Common requirements:

- Your program should be user friendly, and easy to use. Apply object oriented concepts if it is possible. Multilayer architecture is not required.
- Use controls to make the GUI of the game. Some assignments require to use several board sizes. Make sure that the window size fits everytime to the board of the game.
- The documentation should contain:
    - the description of the exercise,
    - the class diagram,
    - the short description of each methods,
    - and the connections between the events and event handlers.

1. Connect Four

Connect Four is a two-player game. The discs of the first player are marked with X, and the discs of the second player are marked with O. The players take turns dropping their disc from the top into a *n*-column, *m*-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. If the grid becomes full, the result is draw.

Implement this game, and let the grid size be selectable (8x5, 10x6, 12x7). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with draw), and automatically begin a new game.

2. Pebble

Pebble is a two-player game, played on a board consists of *n* x *n* fields. Initially, *n* white and *n* black pebbles are placed on the board randomly. Each color belongs to only one player. The players take turns choosing one of its pebble, and then move it horizontally or vertically. The movement also affects the neighbouring pebbles in the direction (the pebble on the edge falls off). The objective of the game is to push out as much pebbles of the opponent from the board as we can, within a given number of turns (*5n*). A player wins, if he has more pebbles on the board at the end than his opponent. The game is draw, if they have the same number of pebbles on the board.

Implement this game, and let the board size be selectable (3x3, 4x4, 6x6 → turns are 15, 20, 30). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with draw), and automatically begin a new game.

3. Hunting

Hunting is a two-player game, played on a board consists of *n* x *n* fields, where the first player (call him fugitive) tries to run away, while the second player (the hunter) tries to capture him/her. Initially, the character of the fugitive is at the center of the board, while the hunter has four characters (one at each corner). The players take turns moving their character (hunter can choose from 4) 1 step on the board (they cannot step on each others character). The objective of the hunter is to surround the fugitive in at most *4n* steps, so it won't be able to move.

Implement this game, and let the board size be selectable (3x3, 5x5, 7x7 → turns are 12, 20, 28). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with draw), and automatically begin a new game.

4. Tricky five-in-a-row

Create a game, which is a variant of the well-known five-in-a-row game. The two players can play on a board consists of *n* x *n* fields. Players put their signs alternately (X and O) on the board. A sign can be put only onto a free field. The game ends, when the board is full, or a player won by having five adjacent signs in a row, column or diagonal. The program should show during the game who turns.

The trick in this variant is that if a player makes 3 adjacent signs (in a row, column or diagonal), then one of his signs is removed randomly (not necessary from this 3 signs). Similar happens, when the player makes 4 adjacent signs, but in this case two of his signs are removed.

Implement this game, and let the board size be selectable (6x6, 10x10, 14x14). The game should recognize if it is ended, and it has to show in a message box which player won (if the game is not ended with draw), and automatically begin a new game.

5. Black hole

Hunting is a two-player game, played on a board consists of *n* x *n* fields, which has a black hole at its center. Each player has *n-1* spaceships, which are placed initially in the lower (upper) diagonal of the board (so the same colored spaceships placed on the same side).

The players take turns moving one of their own spaceships. The black hole interferes with the navigation system of the spaceships, so they cannot move only one place, but they move until they reach the edge of the board, the black hole, or an other spaceship. A spaceship cannot jump over an other one. A player wins, if he manages to move half of his spaceships into the black hole.

Implement this game, and let the board size be selectable (5x5, 7x7, 9x9). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

6. Four game

This a two-player game is played on a board consists of *n* x *n* fields, where each field contains a value between 0 and 4. Initially, all the fields contain the value of 0. If a player chooses a field, then the value of the field and its neighbours incremented by one (if the value is less than 4). The player's score represents how many fields did he make to have the value of 4. If a value of a field reaches 4, then the field is colorized with the color of the actual player (red or blue). The game ends, when all fields have the value of 4. The player having the higher score wins.

Implement this game, and let the board size be selectable (3x3, 5x5, 7x7). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

7. Break-through

Break-through is a two-player game, played on a board consists of *n* x *n* fields. Each player has *2n* dolls in two rows, placed on at the player's side initially (similarly to the chess game, but now every dolls of a player look like the same). A player can move his doll one step forward or one step diagonally forward (can't step backward). A player can beat a doll of his opponent by stepping diagonally forward onto it. A player wins when his doll reaches the opposite edge of the board.

Implement this game, and let the board size be selectable (6x6, 8x8, 10x10). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

8. Knight tournament

This two-player game is played on a board consists of *n* x *n* fields. Initially, two white and two black knights are placed at the corners of the board (knights of the same color are placed at the opposite corners).

Players step alternately, and knights can move only in an L shape, like in chess. The board is initially grey, but after each step, the visited places are colored with the color of the visitor knight (the previous color of the field doesn't matter). A player wins, if there are 4 adjacent fields (horizontally, vertically, or diagonally) which colored to player's color. The game ends, when there are no more grey fields.

Implement this game, and let the board size be selectable (4x4, 6x6, 8x8). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

9. Rubik board

This game is a two dimension variant of the Rubik cube. The board of the game consists of *n* x *n* fields. There are *n* different colors, and exactly *n* fields have the same color. The fields are shuffled initially. A player can move cyclically the colors of a row or column (e.g. moving a row to the right, the color of the first field will be the color the last field) to make homogenous rows and columns. The game ends, when each row (or column) contains one color.

Implement this game, and let the board size be selectable (2x2, 4x4, 6x6). The game should recognize if it is ended, and it has to show in a message box how much steps did it take to solve the game. After this, a new game should be started automatically.

10. Rubic clock

Create a game, which implements the Rubik clock. In this game there are 9 clocks. Each clock can show a time between 1 and 12 (hour only). Clocks are placed in a 3x3 grid, and initially they set randomly. Each four clocks on a corner has a button placed between them, so we have four buttons in total. Pressing a button increase the hour on the four adjacent clocks by one. The player wins, if all the clocks show 12.

Implement the game, and let the player restart it. The game should recognize if it is ended, and it has to show in a message box how much steps did it take to solve the game. After this, a new game should be started automatically.