Project for “Programmazione ad Oggetti” for the Winter 2020/21 Session

By: Bram Steenbergen (Nr Matricola: 282839)

Problem description:

The game of connect4 is famous for being both very easy to get into as well as complex enough to be enjoyable for more experienced players. The game works as follows: A board contains a matrix of tiles (with 6 rows and 7 columns) on which a player can change the color of 1 tile to the color corresponding to their player (player 1 is red, player 2 is yellow). The player chooses a column and the lowest white tile changes to the player's color, then the other player does the same. This goes on until one of the players creates a line of 4 tiles directly adjacent to each other horizontally, vertically or diagonally. The goal of this project is to create a version of the connect4 game which includes a graphical interface.

A standard game of connect4 is played with a board containing 7 columns and 6 rows, to make the program more versatile and more interesting to develop, I've chosen to allow the players to set these numbers to different amounts. The number of rows and columns are inserted before the game starts as well as both players' names.

Problem Analysis:

The goal of the project is to construct a program that allows 2 players to play one or more games of connect4 via mouse input. There are several things that need to be done to make sure the game works as it should.

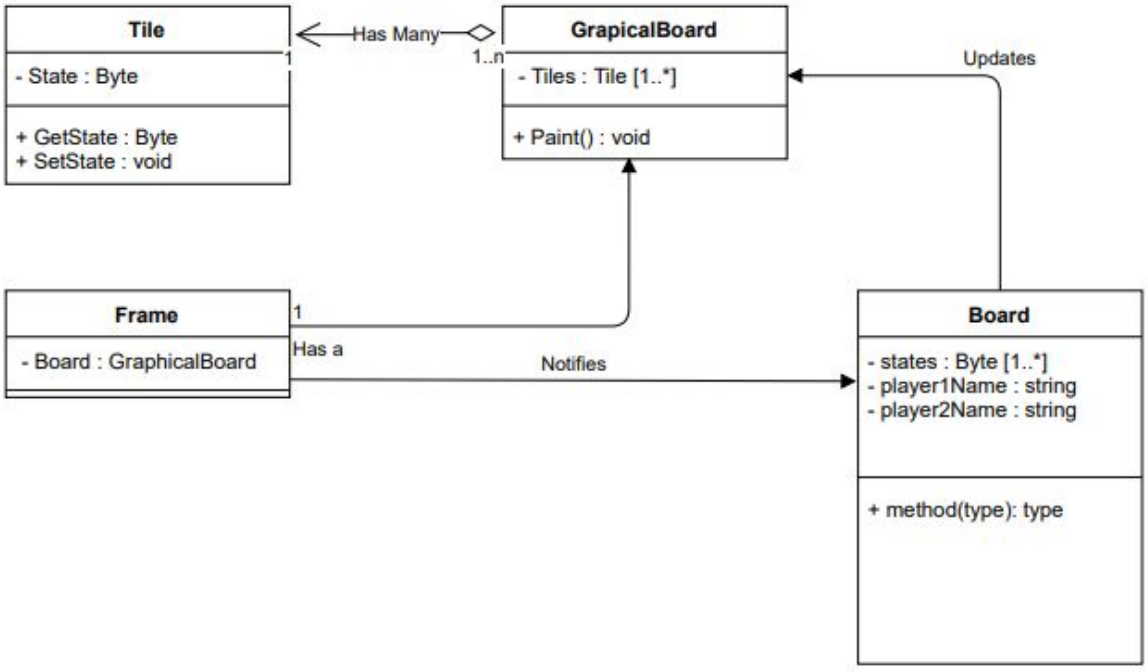
First of all we need a way to make sure that whenever a player wants to make a move, he is told where the move is going to be made. For that the interface needs to provide some kind of indication of where the move is going to be made.

Secondly, we need to make sure it's always clear which player is expected to make a move. The interface needs to provide an indication of which player's turn it is.

Finally, whenever a player makes a move, the program needs to check whether that player was able to win the game using the move he made. It is not necessary to check the whole board instead the program only needs to check for the possible wins which include the last made move. This is true because if there hasn't been a win up until that move was made, the only way to win is to do so with a configuration which includes the final move that was made.

Architectural Choises:

The main difficulty in designing the connect4 game is separating the interface from the actual game. It's very appealing to just program all the logic of the game directly into the interface as it saves on time as well as the number of files needed, however this would lead to very hard to understand code. Therefore we want to separate the interface from the game logic. Here is the first iteration of a design idea for this project:

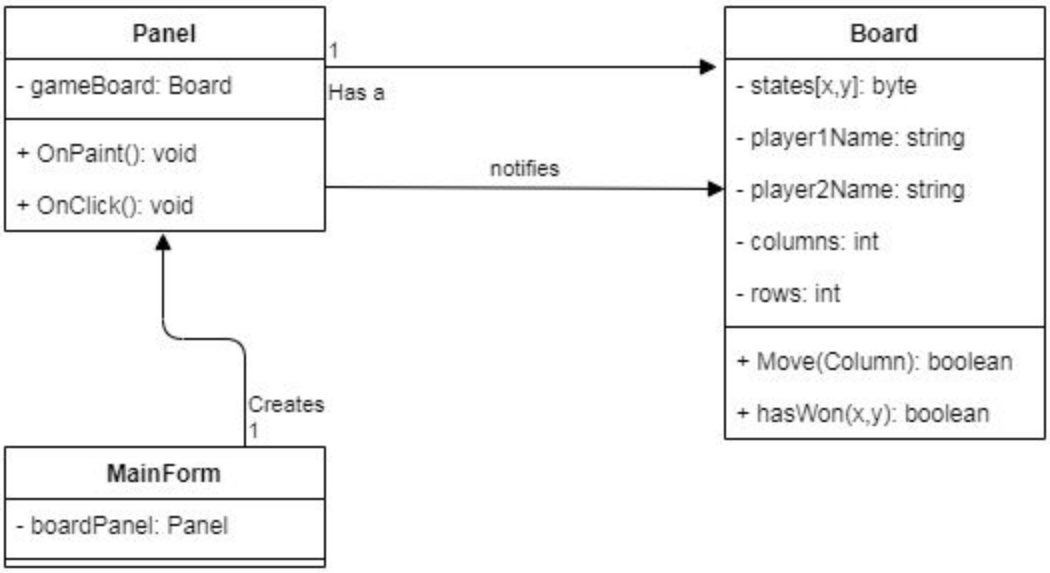


This was the very first sketch. The idea was to have a class containing the data of the game (Board), one class updating the data via method calls (Frame) and one class for showing the interface to the user (GraphicalBoard). The idea was that the GraphicalBoard class would obtain updates from the Board class when a move was made so that the GrapicalBoard could display them.

There is a bit of a problem with this approach though. The GraphicalBoard class is a panel, which is useful because we can paint on it. But it's smaller than the frame class as the frame would be the whole form in which the panel exists, this means we would need to convert mouse events from the frame coordinates to panel coordinates to find out where on the board the user clicked. This is unnecessarily complicated and it would be better to obtain mouse events directly from the panel.

Something more to consider is the reason for including the TIle class, I originally added this to the diagram as I thought it would allow me to store the information directly in the panel and allow me to paint the panel even when not receiving an update from the board, for example when the user resizes the program, the panel needs to be redrawn to fit the form but there has been no update to the board.

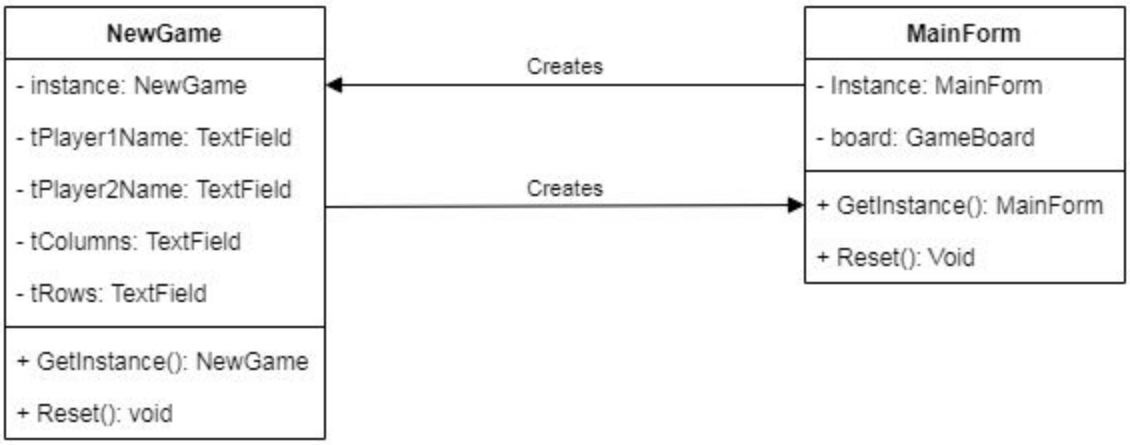
Because I decided there was a better way than to link the GUI and the actual game data earlier, it might be better to replace the Tile class too or even just remove it and allow the panel itself to verify when it needs to be updated and then directly contact the Board class to ask for the states of the game. Here is the second revision of the design of the interface and game connections:



The goal of MainForm at this point is to create the Panel and pass it the GameBoard to use for the game, This GameBoard contains the states of the game as well as both player names and the number of rows/columns to play with. This information needs to be obtained before the game starts, which can be done through another form which, when completed, opens a MainForm which hosts the game.

It’s easy to get confused when creating several forms in a single program. For example you don’t want to have 2 versions of MainForm this would result in undefined behavior as both MainForm’s panels can modify the game in different ways. To make sure this can’t be done we can use the singleton design pattern, which makes sure each class is limited to a single instance.

Given this knowledge I've created the following UML to show the relation between the 2 forms:



As you can see both NewGame and MainForm create an instance of the other. For NewGame this is done whenever the user has inserted all required information and wants the game to start. For MainForm this is done whenever the player is done with the current game and wants a new game to start, A new version of NewGame is then shown for the players to input information in.

User Instructions:

After compilation, the program can be by executing the executable(.exe) file. You will be prompted to input both players' names and the amount of rows and columns the board should have. Defaults are given for all those fields. Once you're happy with the settings for the game press *start* or if you wish to quit press *cancel* which will exit the program.

Now a new window opens with the game itself. Player1 is the first player who can make a move. Player1 can make a move by clicking on the board on the specific column on which they want to make a move, the column is highlighted by a bounding box. Once a move is made, it's player2's turn to make a move. This goes back and forth until one of the 2 players gets 4 tiles in a row either horizontally, vertically or diagonally.

Once a player has won the game it's shown on the game board and no more moves can be made. You can choose to quit the game by pressing *quit* or create a new game by pressing the banner or the *New Game* button. You can also start a new game or quit during the game.

Use Cases:

Here is a diagram showing the use cases through UML: