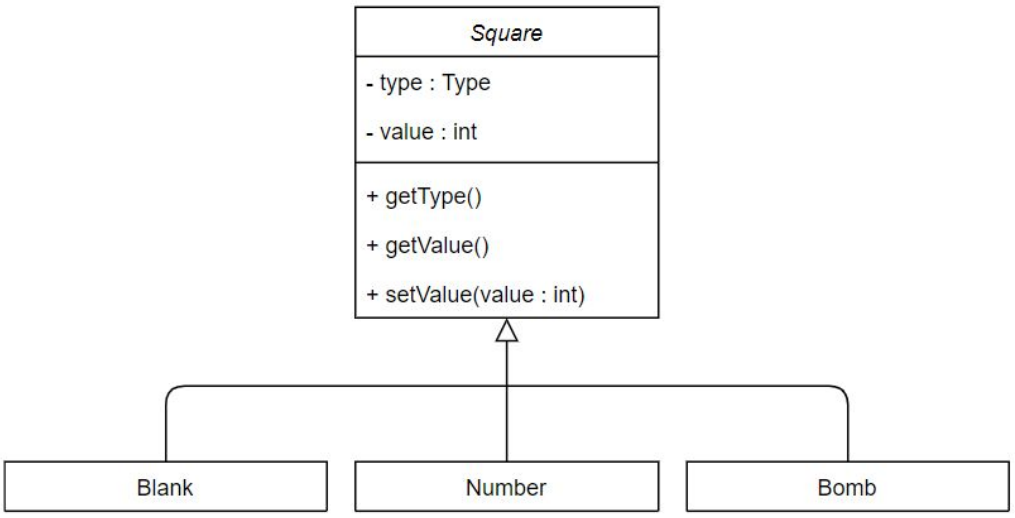
CCE2111 Assignment 15/1/2021
2048: Enhanced Luca Ruggier

Requirements: To recreate the popular game “2048” in C++, implementing object oriented programming principles. The game should include features of loading, saving, a “Hall of Fame”, standard gameplay with a new “Bomb” tile.

UML Diagram:



Description:

Classes: A class was made that acted as a base class for each tile, this class is called Square. 3 derived classes were made, Number, Blank, and Bomb, denoting the three different types of tiles that could be made. A new enumerated type called Type was made, to help differentiate the type of tile, and each derived class made use of it’s own type.

Game State Storage: The game was stored in a 4x4 array of Square objects, all initialized to Blank objects, as an array allows for good random read speeds, which is frequent in this design.

Game Flow: The game runs in a permanent while loop, unless the user quits. Each iteration displays the game menu, and the game grid, and waits for a user input, which is validated, and then triggers the correct event. All events that destroy the current game are confirmed with a yes/no input from the user.

Saving and Loading: A game is saved in the given format, with the score first as “Score:[score]” and then a recollection of all tiles in the grid, and their position. This is done using a simple for loop iterating through all tiles and writes them to the specified save file with the use of an ofstream. The x/y coordinate system is converted to a single value coordinate system with the equation `position = (4 * x) + y;`

Similarly, a game is loaded using an ifstream, with the input of the specified file. The file is analysed line by line, and is delimited with ":" using the string function .find(":"). The position and value of each tile is written to the grid, and the position is converted back to an x/y coordinate system with the equations: `x = pos / 4; y = pos - (4 * x);`

Hall of Fame: The hall of fame is stored in a sorted text file. This is done so that the hall of fame does not need to be sorted after being read. The hall of fame is loaded into memory (in an array) at game start. When it is being updated, the new entry is sorted into the correct position, and then the old text file is overwritten with the new array of Hall of Fame players. The file is overwritten by opening the original hall of fame file as truncated. Another variant would be to store the hall of fame users in the text file unsorted, and sort the array after being loaded, however the previous method was chosen as it allowed for easier reading of the external hall of fame file.

Slide Functions: Using a for loop, the program scans for tiles that will combine or be destroyed, using three cases, Number on Number, Bomb on Number, Number on Bomb. The program then either joins the numbers, or destroys the bomb and the number, depending on the case. After this, all non-blank tiles are shifted to the corresponding location. The program updates the score every time a number is combined or destroyed.
In order to invalidate turns in which no tile moves, a boolean is initialised to false. The slide function is run normally, and whenever a tile is moved, combined or destroyed, this boolean is set to true. After the slide function is complete, the boolean is checked, and if it is true, a new tile is generated using the random function, if not, the user is warned that this is an invalid move.
All slide functions work similarly, except that the direction of the for loops changes for each direction.
Every time two tiles are joined, their new value is checked whether or not the value is equal to 2048

Randomising Function: The randomising function is used to generate a new tile. The function stores all possible new locations in an array. If there are no new locations, the game is lost. The function randomly selects a location from the location array, and then generates a number between 0 and 100. If the number is between 0 and 40, a 2 will be placed, if between 40 and 80, a 4 will be placed, else a bomb will be placed.

Debug Mode: The user may also enter 'Debug Mode' by typing a 'P' as an input. This displays what item was generated, and the location, as it might be useful to have this information.

Winning and Losing: When the win function is called, the hall of fame is updated and then displayed, and a new blank grid is generated. When the lose function is called, the hall of fame is displayed and a new blank grid is generated.

Overall Class Structure Choices: The main OOP principles used in the running of this game are inheritance and polymorphism. The number, blank and bomb classes are all derived classes of the 'Square' class. This inheritance in turn enables polymorphism, where the 3 derived classes could all be switched out and changed in the main gameplay grid, where an object of type Square was expected.

Testing of Functionality: The program was tested in multiple ways.
The input menu was tested by entering invalid inputs not accepted or suggested by the ingame menu, which were caught correctly, and the user is told that this is an invalid input.

Saving and loading were also tested by trying to save or load the Hall of fame text file, which was correctly caught, and the loading function was also tested by entering a save file that did not exist, which was also caught correctly, and the user is told the file does not exist.

When loading, quitting or starting a new game, the user is prompted “Are you sure?” in order to not lose their current progress. The user should enter Y or N to confirm or deny their choice, however this was tested with a different input, which is rejected and the user is told that this is an invalid input, and asked again to enter Y or N.

While the game is in runtime, it does not accept an invalid move (a slide that would result in no tiles to move, combine or be destroyed). This was tested by attempting an invalid move, which was rejected and the user was warned.

To aid testing, a debug mode was created which can be enabled with the user input of ‘P’, which prints what new tile is generated, and where the tile was generated.