Final Documentation

Product introduction:

Our team made a simple Tetris game application. In this application, you can move, iterate or change the shape of the moving block. Besides, you can also change the state of a block. This means that you can change a block from a path to an obstacle. This application will count the scores you get, the lines you delete, the hints you use and the scores you lose in this game. Every time you place a block, you will get 10 scores. If you delete a line, you will get 100 scores. When you want to use hints, you must have enough scores. Every time you use a hint, it will cost you 20 scores. If you don't have enough scores, you will see an alert on your screen which tells you that this operation can't be executed because you don't have enough scores. You can pause the game and restart the game. When you exit, you can choose to play again or end this game. You can see your final score on the screen. Below we will explain how we implement this application.

Implementation introduction:

Our project is assembled by four different parts: Block, Item, Board and the main function. Below we will explain each part features one by one.

Block:

This is the fundamental part of our project. All other parts are built based on this class. This class represents a small block on the screen and stores all the information we need. This class has five private instances. We use these instances to store the location, size, color and the state of this block.

In our project, a block has two state. One is filled the other is unfilled. When a block is filled, it will be considered as an obstacle. Other blocks can't go through it. When we draw a filled block, we need to draw the we need to draw the outline and interior of this block. When a block is unfilled, it will be considered as a path, other blocks can go through it and we only need to draw the outline of this block.

This class has a non-argument constructor. In this constructor we set default values to all the instances of this class. For every instance, there is a set and get value function. Using these functions, we can get the value of a specific instance or set it to be another value. Finally, there is a draw function. This function can draw a block on the screen. If a block is filled, we will draw a blue rectangle. If a block is unfilled, we will draw a blue rectangle inside which is filled with red.

Board:

We use this class to build our main play area. Using this class, we can divide a whole place into several different blocks. In this way, we can check whether we need to delete a line, whether a moving block can be placed, whether the game is end and change the state of a block.

This class has six private instances. They represent the size of this area, the total number of columns and rows in this area, all the blocks in this area and the size of these blocks.

To initialize a board, you need to pass the length and width of the whole area into its constructor. Also, you need to give the size of blocks. Using these data, the constructor will calculate the number of blocks we need to fill this area. All these blocks are stored in an array, so we can find and change them. There are three get value function in this class. Using these functions, you can get the number of rows and columns of this area. You can have access to all the blocks in this area, too.

In the update function, we check whether we need to delete a line or not. We first iterate through all the blocks in this area and for every row we check whether all the blocks in this row are filled or not. If any of these blocks are unfilled, then we start to check the next row. If all the blocks are filled, then it means we need to delete this line. So, we simply copy all the blocks from the above row to this row and do the same thing to the above row until we reach the first row. In this way, we can delete a row. After we delete a line, we will continue to do next iteration until we reach the first row.

In the filled function, we change the state of a specific block. In this function the user needs to provide the x and y position of a point. We first check whether this point is in the play area or not. If the answer is no, we simply end this function. Otherwise, we use iteration to find out which block contains this point. If this block is filled, we won't do anything. Otherwise, we will change this block from unfilled to filled.

In the whetherEnd function, we check whether we need to end this game. According to the rule, we simply need to check whether there are blocks in the first row. If the answer is yes, the function will return true to true to indicate that the game is over. Otherwise, the function will return false.

In the draw function, we draw all the blocks in this play area. We simply call the draw function of each block.

Item:

We use this class to represent a moving block. Using this class, we can randomly generate five different type of blocks with different colors, change their places, rotate them or change their shapes.

An Item is made up of four small blocks. By change the place of these blocks, we can generate five different type of items. In this class, we have four private instances. Using them we can check whether an item is alive, the size of blocks in this item, the type of this item and have access to all the blocks in this item.

The constructor of this class has two inputs. One is the size of the blocks, the other is the address of the board this item will be placed in. In the constructor, we first generate a random number from 0 – 4 to represent the type of this item. Then for different types, we use different ways to assemble blocks so that we can get five different shapes.

The function horizontialMove and goDown are used to move an item. Every time we call these functions, they will first check whether the item can be moved to the desired place or not. If the answer is yes, all the blocks of this item will be update. If the answer is no, this item will be regard as a dead item and four blocks of this item will be copied into the board. Now this item will never move.

The function rotation is used to rotate the item. We use the left upper block of an item as the rotate center. We find out that when a point (x1, y1) is rotate 90 degrees around the point (x0, y0), the final point (x2, y2) can be calculated by $x2 = x0 + y0 - y1$ and $y2 = y0 + x1 - x0$. In this way we can calculate the final position of all the blocks after the rotation. After we get the position, we need to check whether this position is in the play area or not. If it is in the play area, we simply update all the blocks. Otherwise, we won’t do anything.

The changeShape function is used to change the shape of an item. When this function is called , we first randomly generate a new type for this item (it must be different from its original type). According to the new type, we will update all the blocks in this item to build a new item.

The draw function is used to draw the item. In this function we simply call each block’s draw function.

Main:

In this main function, we design the GUI for our application and add some pictures and music to it. Since we have four different interfaces, we design four while loops to control each interface. We put all the information on the main screen so that the users don’t need to switch windows when they are playing. In order to make the operation easier, we add some buttons to our interfaces so that the users can use these buttons to do operations. More information can be found in the User Guide file.