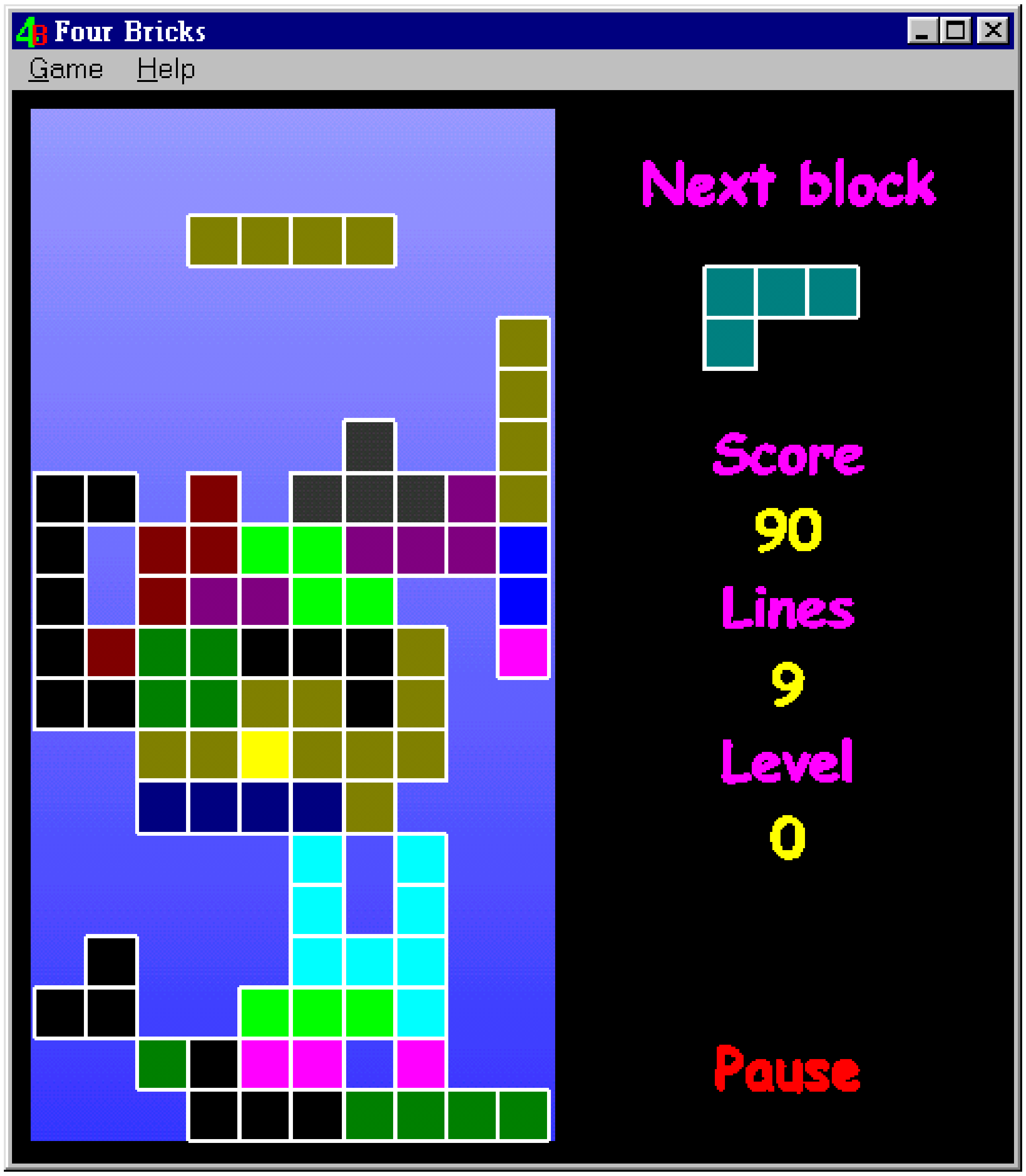
Tetris
150 points
You may work in pairs if you wish



Submit all code necessary to compile and execute (if you use VS.NET, you can include all that stuff in your zip)
Document code as you deem necessary -- basic documentation should be included. Describe anything you deem is worth extra credit and anything that doesn't work properly.
NOTE: Original version of this assignment developed by Paul Schimpf!

In 1987 a company called Spectrum Holobyteä introduced the game Tetrisâ (which is a registered trademark). In order to prove your newly acquired skills in .NET programming, you will create a simplified clone of the game. In doing so, you should get a chance to use the graphics device interface, timers, keyboard handlers, serialization, menus, status bars, and collision detection, among other things.

Following are the specifications for your application, along with some hints on how to proceed. Read the specifications very carefully - each behavior will have some points associated with it. If you are uncertain about any of the requirements, bring your code to me and I will discuss it with you. You will find that I am quite willing to help you get through any problems that you encounter, so I suggest that you get started as soon as possible in order to take maximum advantage of your opportunities to receive my help. Only bad things can happen if you wait until the last week of the quarter to get started.

The specifications represent minimum requirements. Feel free to expand upon the game features if you wish. For example, a preview of the next block to appear would be a nice feature, as well as an ability to store the top 10 scores along with player names. Extra features mean extra credit ;-)

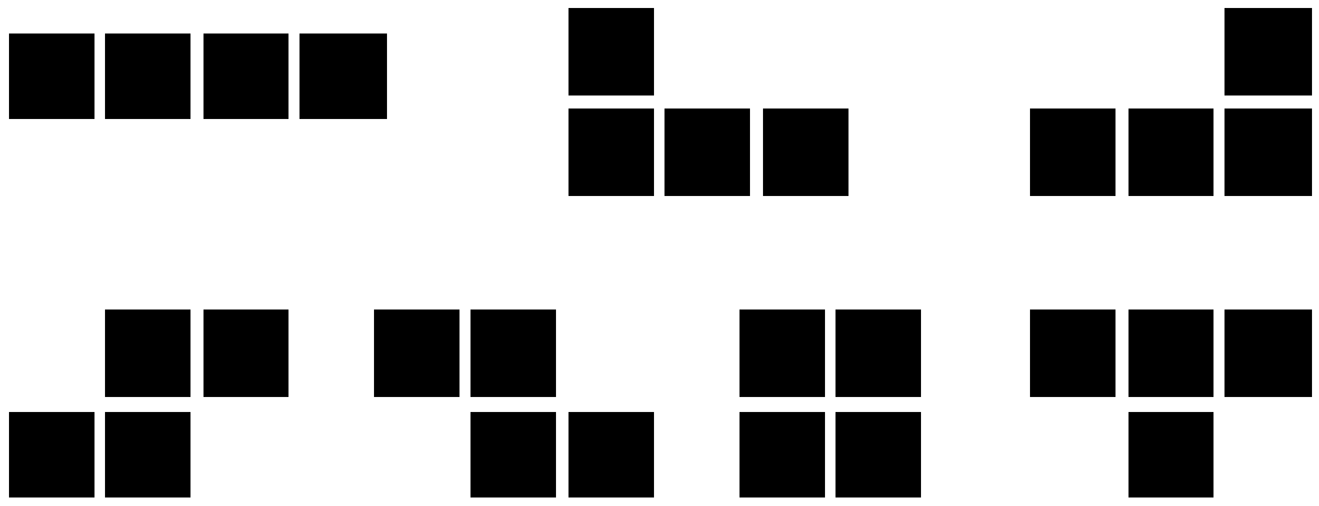
How to get started

Begin by playing with the (C++) executable demo of solution or you can play another version at this link: http://kidrocket.org/game_tetris.php . Here's how the game works: As the blocks fall, you can move them left or right using the cursor left/right keys. You can also rotate the blocks using the cursor up/down keys. You can drop the block by pressing the space bar. The game ends if the blocks stack up to the top row of the screen. Whenever a row across the screen is filled-in, you earn 100 points and that row is removed (and any partially filled rows above it are shifted down). When you complete more than one row at the same time, you receive a bonus of 50 points for each additional row (thus two rows is worth 250 points, three rows is worth 400 points, and four rows is worth 550 points. The game advances to the next level whenever you complete 10 rows, which causes the tiles to fall 25% faster than the previous level (please note that 1/1.25 is NOT equal to 0.75). The points earned for completing rows grow proportional to the level number. Thus at level two, one row is worth 200 points, two rows is worth 500 points, three rows is worth 800 points, and four rows is worth 1100 points.

Specifications

1. Your game must consist of a rectangular playing area that is 10 cells wide and 18 cells tall. The window should size itself around the game grid, and stretching or expanding the window to a larger size should not be allowed.

2. Blocks introduced into the game should be selected randomly from the following 7 patterns:



You should also pick a unique color to associate with each pattern.

3. Blocks should be introduced in a random location with a random rotation at the top of the screen (it is OK if they first appear in the second row). You may end the game when the space at which the new block is to be introduced is already occupied.

4. The player should be able to control the left/right movement and rotation of falling blocks using cursor keys just like my demo version. Note that a block cannot be moved or rotated if that would cause it to overlap with another block or a wall. The space bar should drop the block as far as it will go.

5. The player should be able to Pause and Resume from the Game menu or using Ctrl+P or Ctrl+G, respectively. The Pause menu item should be disabled when the game is already paused, and the Go menu item should be disabled when the game is already in progress. Both should be disabled at the end of a game, so that the player is required to select File/New instead of Game/Go to start a new game.

6. In the first level, the blocks should fall at the rate of approximately 1 row every half second. The player should be advanced to the next level whenever 10 rows are completed. Each time you advance to the next level, the blocks should fall approximately 25% faster, and the point values should increase proportional to the new level. When a group of rows is completed at the same time, then it is OK if you score all of those rows at the current level. For example, if 8 rows were previously completed, and 4 rows are completed at the same time, then you do not have to score 2 of those rows at the current level and 2 at the new level. You may score all 4 at the current level and then increment the level. It is also OK if you do not count those 2 rows towards completion of the new level.

7. The current score and level should be visible during the course of a game. A convenient place for this is on the status bar owned by the frame window. The all-time high score should be available for display. I put this on the About dialog box. Although the high score on my demo solution is persistent between programs runs, you are not required to do that. My demo accomplishes that by recording the high score in the system registry, but PLEASE do not turn in a project that modifies the registry or puts .ini files in the system directory - I would rather not muck up my system with entries from each of your solutions. If you want to make your high score persistent, record it in a file that is in the same directory as the application file.

8. The File menu should have options to start a new game, save a game, load a game, or exit the program. I chose only to include a Save As option, but you may use both a Save and a Save As option if you wish. Menu shortcuts and accelerators should be implemented as in my demo (e.g., Alt-F, N or simply Ctrl+N should start a new game). File save and load options should only be enabled when the game is paused, but the New option should be available at any time. You should not save or restore the currently falling block. After loading a saved game, you should simply introduce a new random block.

9. When the game is over, some visual indication should be provided.

10. Your game should include the following "cheat code": pressing the <Home> key should increment the player to the next level. This should result in the higher point values and more rapid descent rate for the blocks.

11. The title bar of your application should contain a customized icon, and the about box should

contain your name and a customized icon.

<u>**Hints**</u>

1. <u>**Objects:**</u>
   I will tell you how I represented my game state, but there are many other perfectly valid approaches, so use your best judgement. I maintain the state of the playing surface with a 2D array of integers. The content of that array tells me whether each cell is unoccupied (0) or occupied (not zero). When the cell is occupied, the integer value is used as an index into another array of RGB COLORREF values. The game array is encapsulated by my Tetris class, which also encapsulates an object that represents the currently falling block. This object knows how to place itself in the game array (to which it receives a reference) and knows how to draw itself when given a graphics object. Whenever the block is unable to move, it simply leaves a representation of its last position in the game array. The encapsulating class then gives it a randomly selected configuration and tells it to start over at the top of the array.
2. <u>**Block Representation:**</u>

   My class that represents the falling block makes use of a 4-dimensional array. This shouldn't cause you any panic, but if it does, feel free to use another approach. Why 4 dimensions? Each rotation of the block is represented by a 4x4 array of integers (that's 2 dimensions), there are 4 possible orientations (that's the 3rd dimension), and there are 7 different shapes (that's the 4th dimension). My array therefore looks like this: int nShape[7 shapes][4 rotations][4 rows][4 columns].
3. <u>**How to get Blocks to the Edge of the Playing Area:**</u>

   Again, there are several approaches that are workable. I used a brute-force approach. My class that represents the falling block uses another 3-dimensional array that stores the offset of the block from each side of the 4x4 array for each rotation. That array is used, for example, to figure out whether the left side of the 4x4 array can be aligned past the left edge of the playing area. What are the 3-dimensions? Well, this array looks something like this: in m_nOffset[7 shapes][4 orientations][4 sides].
4. <u>**How to get Blocks to fit Together:**</u>

   When a block wants to move, I first test whether the move leaves it within the playing area. I then test whether there is something present in any of the cells that it wants to occupy. This is done by looping through the 4x4 array, and for each cell occupied by the block, testing whether the corresponding cell in the game array is already occupied.
5. <u>**Animation:**</u>

   Use a timer or a thread. The Pause and Go menu handlers should Kill and Start the timer, respectively. When advancing levels you will also need to Kill and restart the timer.
6. <u>**Erasing a Block when it Moves or Rotates**</u>

   The easiest approach is to simply redraw it in the background color. Of course, the cells that it occupies in the game grid should also be zeroed out.
7. <u>**Development Process:**</u>

   Start simple, and add functionality incrementally. Start by modifying the default menu. Add menu handlers as you add other functionality. Add code to initialize the game state. Add code to draw the game grid in response to OnPaint() type events. Add a class to represent the current block, and give it member methods to perform movement and rotations and to draw itself. You will need to develop algorithms to figure out whether various moves and rotations are legal given the current entries in the game grid. Then add a timer event handler that tells the block to move down one position. Add a keyboard handler that tells the block to move, rotate, or drop. Add code that deals with a block that can move no lower (i.e. check for full rows and update the score and the level). Then add handlers for the Pause and Go menu selections. Then add the File I/O. At some point when you need a break from strenuous thinking, modify the about box. Test your code between each change. Backup your directory occasionally.