Game tutorials made easy

Game tutorials for smart beginners and meek geeks

About

Game Engines #1: IndieLib, 2.5d engine using c++ for rapid game development

Follow me on Twitter

Tags

editor game logic indie IndieLib SDL tetris tools tutorials

Categories

advanced tuts

beginners tuts

engines

indie

Uncategorized

Recent Posts

Please, vote for Erasmusu.com!

Indie Game Development Vanguard

In game c++ map editor tutorial with IndieLib engine that dosen't use tiles but pieced images like in Braid or Aquaria games

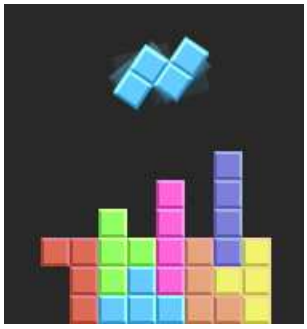
Game Engines #1: IndieLib, 2.5d engine using c++ for rapid game development

Tetris tutorial in C++ platform independent focused in game logic for beginners

Recent Comments

lyzen on Tetris tutorial in C++ platform independent

Tetris tutorial in C++ platform independent focused in game logic for beginners



We are going to learn how to create a Tetris clone from scratch using simple and clean C++. And this will take you less than an hour! This is the perfect tutorial for beginners. Just enjoy it and leave a comment if you want me to explain something better. I know my English sucks, so if you see some mistakes, please, tell me. Don't forget to follow me on Twitter. Let's go!

Download sourcecode

Here it is the complete sourcecode.

Windows platforms

The sourcecode comes with SDL includes and libs ready to compile in Visual C++ Express Edition 2008. In "Release" folder there is also an executable file just in case you want to try it directly.

Other platforms (for advanced users)

Thanks to lmelior and to Javier Santana, there is a Linux version of this tutorial. The sourcecode is platform independent and comes with a "makefile". However, under Linux, you need **libsdl-gfx1.2-dev** and **libsdl1.2-dev** (If you are using Ubuntu you can get them this way: *sudo apt-get install libsdl1.2-dev libsdl-gfx1.2-dev*)

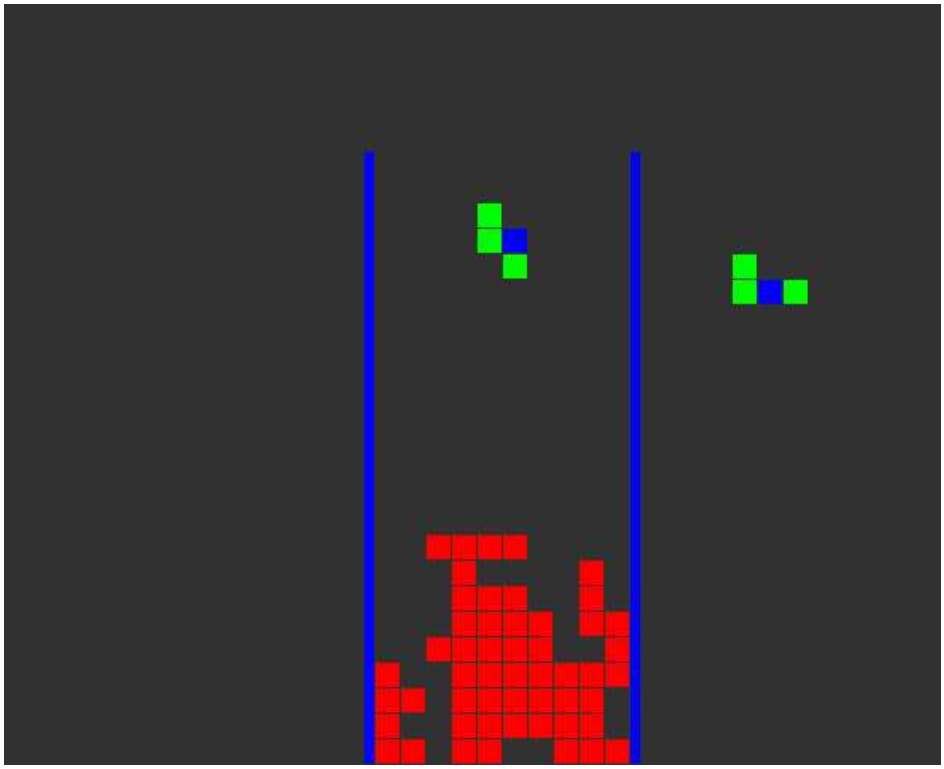
Keys:

- ESC = Quit the game
- z = Rotate the piece
- x = Drop piece
- Left, Right, Down = I will not offend your intelligence

Step 0: Introduction

We are going to focus on the game logic, using only rectangle primitives (SDL) for the rendering. All the game logic is isolated from the drawing, so you can expand the tutorial easily. I'm planning making a second tutorial of how to improve this Tetris clone using sprites, background, effects, etc. But right now, let's focus on the game logic. This is how your prototype will look after you finish the tutorial:

Using the Simplifi theme by Jason, and easternwest.



In this tutorial you will learn:

- How to store the pieces and board using matrices (multidimensional arrays).
- How to solve the rotation problem in Tetris, in a really easy way, without using complex maths or anything difficult, just using an intelligent hack.
- How to check collisions between the pieces and the board.
- How the main loop of a Tetris game works.

What you are supposed to already know:

- C++
- A little bit of graphical programming if you want expand the tutorial with improved graphics. Don't worry about that if you just want to learn the Tetris game logic.

What do you need?

- A compiler or programming IDE. I've used Visual C++ Express Edition for this tutorial, that is a free C++ IDE. But you can use the one of your choice, of course.
- Desire to learn 😀

What is the license of the sourcecode?

The sourcecode is under the "Creative Commons - Attribution 3.0 Unported". That means you can copy, distribute and transmit the work and to adapt it. But you must attribute the work (but not in any way that suggests that they endorse you or your use of the work). The manner of attribution is up to you. You can just mention me (Javier López). A backlink would be also appreciated.

Step 1: The pieces

First, we are going to create a class for storing all the pieces. There are 7 different types of pieces: square, I, L, L-mirrored, N, N-mirrored and T. But, how can we define each piece? Just check out the figure:

0	0	0	0	0
0	0	0	1	0
0	0	2	1	0
0	0	1	0	0
0	0	0	0	0

As you can see, this piece is defined in a matrix of 5×5 cells. 0 means "no block", 1 means "normal block" and 2 means "pivot block". The pivot block is the rotation point: yes, the

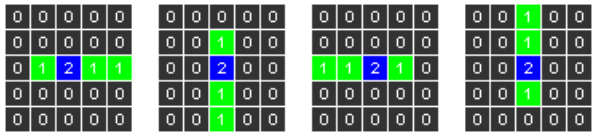
original Tetris game has a rotation point for each piece

And how can we store that using C++? Easy: using a bidimensional array of 5×5 ints (or bytes, if you are a fanatic of optimization). The previous piece is stored like that:

view plain copy to clipboard print ?

```
01. {0, 0, 0, 0, 0},
02. {0, 0, 0, 1, 0},
03. {0, 0, 2, 1, 0},
04. {0, 0, 1, 0, 0},
05. {0, 0, 0, 0, 0}
```

Now that we already now how to store each piece let's think about rotations. We can solv the rotation problem in a lot of different ways. In other tutorials, I've seen them use complex rotation algebra in order to rotate the piece… but we can solve this problem easily. If we can store each piece… why don't we just store each piece rotated too? There are four possible rotations for each piece:



As you can see, the longer piece is only 4 block widht. But we are using 5 blocks matrices ir order to be able to store all the rotations respeting the pivot block. In a previous version o this tutorial, I was using 4-block matrices, but then it was necessary to store translations of the pivot to the origin. This way, we are using some bytes more but the sourcecode is cleaner. In total we only use 448 bytes to store all the pieces. That's nothing

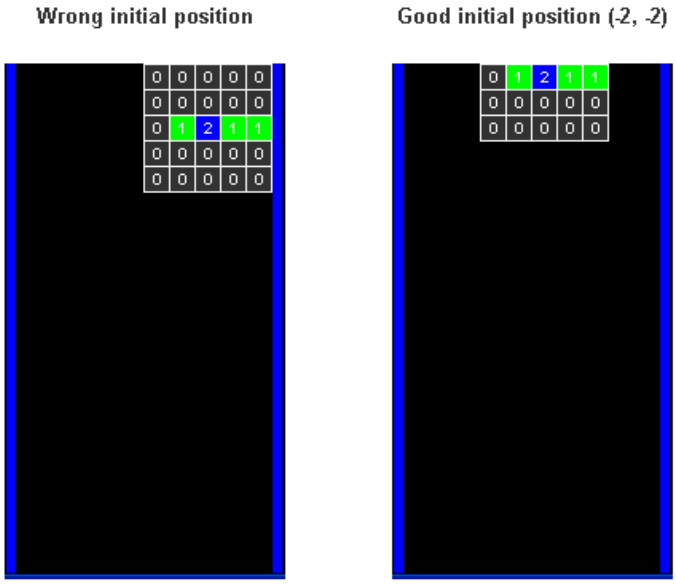
So, in order to store all this information we need a 4-dimensional array (wow!), in order to store the 4 possible rotations (matrices of 5×5) of each piece:

view plain copy to clipbaord print ?

```
// Pieces definition
char mPieces [7 /*kind */ ][4 /* rotation */ ][5 /* horizontal blocks */ ][5 /*
{
// Square
  {
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 2, 1, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 2, 1, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 2, 1, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 2, 1, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    }
   },

// I
  {
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 1, 2, 1, 1},
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 1, 0, 0},
    {0, 0, 2, 0, 0},
    {0, 0, 1, 0, 0},
    {0, 0, 1, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {1, 1, 2, 1, 0},
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 1, 0, 0},
    {0, 0, 1, 0, 0},
    {0, 0, 2, 0, 0},
    {0, 0, 1, 0, 0},
    {0, 0, 0, 0, 0}
    }
   }
  ,
// L
  {
   {
    {0, 0, 0, 0, 0},
    {0, 0, 1, 0, 0},
    {0, 0, 2, 0, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {0, 1, 2, 1, 0},
    {0, 1, 0, 0, 0},
    {0, 0, 0, 0, 0}
    },
   {
    {0, 0, 0, 0, 0},
    {0, 1, 1, 0, 0},
```

Great! Now, in order to rotate a piece we just have to choose the following stored rotated piece.

There is something important that we have to take in count. Each different piece must be correctly positioned every time it is created on the top of the screen. In other words, it needs to be translated to the correct position (in order to show ONLY one row of blocks in the board and to be centered, upper blocks should be OUTSIDE the board). Like each piece is different (some are lower or smaller than others in the matrices), each one needs a different translation every time it is created. We will store these translations in another array, one translation per rotated piece. Take your time to understand this.



The translation are two numbers (horizontal tranlastion, vertical translation) that we have to store for each piece. We will use these numbers later in “Game” class when creating the pieces each time a new piece appears, so it will be initialized in the correct position. This is the array that stores these displacements:

view plain copy to clipboard print ?

```
// Displacement of the piece to the position where it is first drawn in the boar
int mPiecesInitialPosition  [7 /*kind */ ][4 /* r2otation */ ][2 /* position */]
{
/* Square */
  {
    {-2, -3},
    {-2, -3},
    {-2, -3},
    {-2, -3}
   },
/* I */
  {
    {-2, -2},
    {-2, -3},
    {-2, -2},
    {-2, -3}
   },
/* L */
  {
    {-2, -3},
    {-2, -3},
    {-2, -3},
    {-2, -2}
   },
/* L mirrored */
  {
    {-2, -3},
    {-2, -2},
    {-2, -3},
    {-2, -3}
   },
/* N */
  {
    {-2, -3},
    {-2, -3},
    {-2, -3},
    {-2, -2}
   },
/* N mirrored */
  {
    {-2, -3},
    {-2, -3},
    {-2, -3},
    {-2, -2}
   },
/* T */
  {
    {-2, -3},
    {-2, -3},
    {-2, -3},
    {-2, -2}
   },
};
```

And with that we have solved one of the most tricky parts of this tutorial.

We can now create our Pieces class, this file is called “Pieces.h”:

view plain copy to clipboard print ?

```
#ifndef _PIECES_
#define _PIECES_

// ---------------------------------------------------------------------------
//                                    Pieces
// ---------------------------------------------------------------------------

class Pieces
{
public:

    int GetBlockType        (int pPiece, int pRotation, int pX, int pY);
    int GetXInitialPosition (int pPiece, int pRotation);
    int GetYInitialPosition (int pPiece, int pRotation);
};

#endif // _PIECES_
```

The 3 methods that you can see in the header returns some information that we will need later. Their implementation is trivial:

view plain copy to clipboard print ?

```
/*
=====================================
Return the type of a block (0 = no-block, 1 = normal block, 2 = pivot block)

Parameters:

>> pPiece:        Piece to draw
>> pRotation: 1 of the 4 possible rotations
>> pX:            Horizontal position in blocks
>> pY:            Vertical position in blocks
=====================================
*/
int Pieces::GetBlockType (int pPiece, int pRotation, int pX, int pY)
{
    return mPieces [pPiece][pRotation][pX][pY];
}

/*
=====================================
Returns the horizontal displacement of the piece that has to be applied in order
correct position.

Parameters:

>> pPiece:    Piece to draw
>> pRotation: 1 of the 4 possible rotations
=====================================
*/
int Pieces::GetXInitialPosition (int pPiece, int pRotation)
{
    return mPiecesInitialPosition [pPiece][pRotation][0];
}

/*
=====================================
Returns the vertical displacement of the piece that has to be applied in order t
correct position.

Parameters:

>> pPiece:    Piece to draw
>> pRotation: 1 of the 4 possible rotations
=====================================
*/
int Pieces::GetYInitialPosition (int pPiece, int pRotation)
{
    return mPiecesInitialPosition [pPiece][pRotation][1];
}
```

Step 2: The board

Now we are going to learn how to store the pieces in the board and check collisions. This class stores a bidimensional array of N x N blocks that are initialized to POS_FREE. The pieces will be stored by filling these blocks when they fall down updating the block to POS_FILLED. In this class we need to implement methods in order to store a piece, check if movement is possible, delete lines, etc. Our board is going to be very flexible, we will be able to choose the amount of horizontal and vertical blocks and the size of each block.

This is the header of the class (”Board.h”):

view plain copy to clipboard print ?

```
01. #ifndef _BOARD_
02. #define _BOARD_
03.
04. // ------ Includes -----
05.
06. #include "Pieces.h"
07.
08. // ------ Defines -----
09.
10. #define BOARD_LINE_WIDTH 6          // Width of each of the two lines that delim
11. #define BLOCK_SIZE 16               // Width and Height of each block of a piece
12. #define BOARD_POSITION 320          // Center position of the board from the lef
13. #define BOARD_WIDTH 10              // Board width in blocks
14. #define BOARD_HEIGHT 20             // Board height in blocks
15. #define MIN_VERTICAL_MARGIN 20      // Minimum vertical margin for the board lim
16. #define MIN_HORIZONTAL_MARGIN 20    // Minimum horizontal margin for the board l
17. #define PIECE_BLOCKS 5              // Number of horizontal and vertical blocks
18.
19. // ----------------------------------------------------------------------------
20. //                                   Board
21. // ----------------------------------------------------------------------------
22.
23. class Board
24. {
25. public:
26.
27.     Board                       (Pieces *pPieces, int pScreenHeight);
28.
29.     int GetXPosInPixels         (int pPos);
30.     int GetYPosInPixels         (int pPos);
31.     bool IsFreeBlock            (int pX, int pY);
32.     bool IsPossibleMovement     (int pX, int pY, int pPiece, int pRotation);
33.     void StorePiece             (int pX, int pY, int pPiece, int pRotation);
34.     void DeletePossibleLines    ();
35.     bool IsGameOver             ();
36.
37. private:
38.
39.     enum { POS_FREE, POS_FILLED };          // POS_FREE = free position of the b
40.     int mBoard [BOARD_WIDTH][BOARD_HEIGHT]; // Board that contains the pieces
41.     Pieces *mPieces;
42.     int mScreenHeight;
43.
44.     void InitBoard();
45.     void DeleteLine (int pY);
46. };
47.
48. #endif // _BOARD_
```

Now, let's see each different method.

InitBoard method is just a nested loop that initializes all the board blocks to POS_FREE.

view plain copy to clipboard print ?

```
01. /*
02. ======================================
03. Init the board blocks with free positions
04. ======================================
05. */
06. void Board::InitBoard()
07. {
08.     for (int i = 0; i < BOARD_WIDTH; i++)
09.         for (int j = 0; j < BOARD_HEIGHT; j++)
10.             mBoard[i][j] = POS_FREE;
11. }
```

StorePiece method, just stores a piece in the board by filling the appropriate blocks as POS_FILLED. There is a nested loop that iterates through the piece matrix and store the blocks in the board.

view plain copy to clipbord print ?

```
/*

=====================================
Store a piece in the board by filling the blocks

Parameters:

>> pX:        Horizontal position in blocks
>> pY:        Vertical position in blocks
>> pPiece:    Piece to draw
>> pRotation: 1 of the 4 possible rotations
=====================================
*/
void Board::StorePiece (int pX, int pY, int pPiece, int pRotation)
{
    // Store each block of the piece into the board
    for (int i1 = pX, i2 = 0; i1 < pX + PIECE_BLOCKS; i1++, i2++)
    {
        for (int j1 = pY, j2 = 0; j1 < pY + PIECE_BLOCKS; j1++, j2++)
        {
            // Store only the blocks of the piece that are not holes
            if (mPieces->GetBlockType (pPiece, pRotation, j2, i2) != 0)
                mBoard[i1][j1] = POS_FILLED;
        }
    }
}
```

IsGameOver checks if there are blocks in the first row. That means the game is over.

view plain copy to clipbord print ?

```
/*
=====================================
Check if the game is over becase a piece have achived the upper position

Returns true or false
=====================================
*/
bool Board::IsGameOver()
{
    //If the first line has blocks, then, game over
    for (int i = 0; i < BOARD_WIDTH; i++)
    {
        if (mBoard[i][0] == POS_FILLED) return true;
    }

    return false;
}
```

DeleteLine is the method that erases a line and moves all the blocks of upper positions one row down. It just starts from the line that has to be removed, and then, iterating through the board in a nested loop, moves all the blocks of the upper lines one row done.

view plain copy to clipbord print ?

```
/*
=====================================
Delete a line of the board by moving all above lines down

Parameters:

>> pY:        Vertical position in blocks of the line to delete
=====================================
*/
void Board::DeleteLine (int pY)
{
    // Moves all the upper lines one row down
    for (int j = pY; j > 0; j--)
    {
        for (int i = 0; i < BOARD_WIDTH; i++)
        {
            mBoard[i][j] = mBoard[i][j-1];
        }
    }
}
```

DeletePossibleLines is a method that removes all the lines that should be erased from the board. It works by first checking which lines should be removed (the ones that have all the horizontal blocks filled). Then, it uses the **DeleteLine** method in order to erase that line and move all the upper lines one row down.

view plain copy to clipboard print ?

```
/*
=====================================
Delete all the lines that should be removed
=====================================
*/
void Board::DeletePossibleLines ()
{
    for (int j = 0; j < BOARD_HEIGHT; j++)
    {
        int i = 0;
        while (i < BOARD_WIDTH)
        {
            if (mBoard[i][j] != POS_FILLED) break;
            i++;
        }

        if (i == BOARD_WIDTH) DeleteLine (j);
    }
}
```

IsFreeBlock is a trivial method that checks out if a board block is filled or not.

view plain copy to clipboard print ?

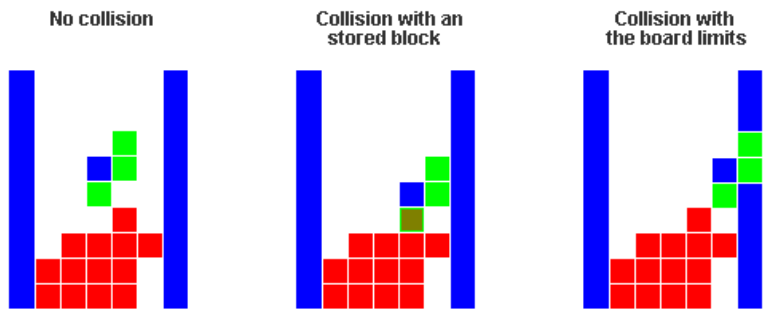
```
/*
=====================================
Returns 1 (true) if the this block of the board is empty, 0 if it is filled

Parameters:

>> pX:        Horizontal position in blocks
>> pY:        Vertical position in blocks
=====================================
*/
bool Board::IsFreeBlock (int pX, int pY)
{
    if (mBoard [pX][pY] == POS_FREE) return true; else return false;
}
```

Until now we have been always talking about “blocks”. But in order to draw them to the screen we need to specify the position in pixels. So, we need two methods (**GetXPosInPixe** and **GetYPosInPixels**) in order to obtain the horizontal and vertical position in pixels of a given block.

view plain copy to clipboard print ?

```
/*
=====================================
Returns the horizontal position (in pixels) of the block given like parameter

Parameters:

>> pPos:  Horizontal position of the block in the board
=====================================
*/
int Board::GetXPosInPixels (int pPos)
{
    return  ( ( BOARD_POSITION - (BLOCK_SIZE * (BOARD_WIDTH / 2)) ) + (pPos * Bl
}

/*
=====================================
Returns the vertical position (in pixels) of the block given like parameter

Parameters:

>> pPos:  Horizontal position of the block in the board
=====================================
*/
int Board::GetYPosInPixels (int pPos)
{
    return ( (mScreenHeight - (BLOCK_SIZE * BOARD_HEIGHT)) + (pPos * BLOCK_SIZE)
}
```

IsPossibleMovement is the last and most complex method of Board class. This method wil be used later in the main loop to check if the movement of a piece is possible or not. The method compares all the blocks of a piece with the blocks already stored in the board and with the board limits. That comparison is made by iterating through the piece matrix and comparing with the appropriate 5×5 area in the board. If there is a collision that means th movement is not possible, so it returns false. If there is no collision, the movement is possible and it returns true.



view plain copy to clipboard print ?

```
/*
=====================================
Check if the piece can be stored at this position without any collision
Returns true if the movement is  possible, false if it not possible

Parameters:

>> pX:        Horizontal position in blocks
>> pY:        Vertical position in blocks
>> pPiece:    Piece to draw
>> pRotation: 1 of the 4 possible rotations
=====================================
*/
bool Board::IsPossibleMovement (int pX, int pY, int pPiece, int pRotation)
{
    // Checks collision with pieces already stored in the board or the board lim
    // This is just to check the 5x5 blocks of a piece with the appropriate area
    for (int i1 = pX, i2 = 0; i1 < pX + PIECE_BLOCKS; i1++, i2++)
    {
        for (int j1 = pY, j2 = 0; j1 < pY + PIECE_BLOCKS; j1++, j2++)
        {
            // Check if the piece is outside the limits of the board
            if (    i1 < 0           ||
                i1 > BOARD_WIDTH  - 1    ||
                j1 > BOARD_HEIGHT - 1)
            {
                if (mPieces->GetBlockType (pPiece, pRotation, j2, i2) != 0)
                    return 0;
            }

            // Check if the piece have collisioned with a block already stored i
            if (j1 >= 0)
            {
                if ((mPieces->GetBlockType (pPiece, pRotation, j2, i2) != 0) &&
                    (!IsFreeBlock (i1, j1)) )
                    return false;
            }
        }
    }

    // No collision
    return true;
}
```

Step 3: The game

Now we are going to implement a general class, called “Game”, that itializes the game, draws the board and pieces by drawing each block as a rectangle (using another class that we will see later called “IO” that uses SDL) and creates new random pieces.

This is the header, “Game.h”:

view plain copy to clipboard print ?

```
#ifndef _GAME_
#define _GAME_

// ------ Includes -----

#include "Board.h"
#include "Pieces.h"
#include "IO.h"
#include <time.h>

// ------ Defines -----

#define WAIT_TIME 700           // Number of milliseconds that the piece remains

// ---------------------------------------------------------------------------
//                                    Game
// ---------------------------------------------------------------------------

class Game
{
public:

    Game            (Board *pBoard, Pieces *pPieces, IO *pIO, int pScreenHeight)

    void DrawScene ();
    void CreateNewPiece ();

    int mPosX, mPosY;               // Position of the piece that is falling dow
    int mPiece, mRotation;          // Kind and rotation the piece that is falli

private:

    int mScreenHeight;              // Screen height in pixels
    int mNextPosX, mNextPosY;       // Position of the next piece
    int mNextPiece, mNextRotation;  // Kind and rotation of the next piece

    Board *mBoard;
    Pieces *mPieces;
    IO *mIO;

    int GetRand (int pA, int pB);
    void InitGame();
    void DrawPiece (int pX, int pY, int pPiece, int pRotation);
    void DrawBoard ();
};

#endif // _GAME_
```

As you can see, the current piece is defined using 4 variables: **mPosX**, **mPosY** (the position of the piece in blocks), **mPiece** (the type of the piece), **mRotation** (the current matrix that defines the piece, as we have seen, each piece has four matrices, one for each rotation).

Let's see the implementation of the methods.

GetRand is a trivial method that returns a random number between two boundaries.

view plain copy to clipboard print ?

```
/*
=====================================
Get a random int between to integers

Parameters:
>> pA: First number
>> pB: Second number
=====================================
*/
int Game::GetRand (int pA, int pB)
{
    return rand () % (pB - pA + 1) + pA;
}
```

InitGame, takes care of the initialization of the game by selecting the first and next piece randomly. The next piece is shown so the player can see which piece will appear next. This method also sets the position in blocks of that pieces. We use two methods that we have seen before in "Pieces" class: **GetXInitialPosition** and **GetYInitialPosition** in order to initialize the piece in the correct position.

view plain copy to clipboard print ?

```
/*
=====================================
Initial parameters of the game
=====================================
*/
void Game::InitGame()
{
    // Init random numbers
    srand ((unsigned int) time(NULL));

    // First piece
    mPiece          = GetRand (0, 6);
    mRotation       = GetRand (0, 3);
    mPosX           = (BOARD_WIDTH / 2) + mPieces->GetXInitialPosition (mPiece,
    mPosY           = mPieces->GetYInitialPosition (mPiece, mRotation);

    //  Next piece
    mNextPiece      = GetRand (0, 6);
    mNextRotation   = GetRand (0, 3);
    mNextPosX       = BOARD_WIDTH + 5;
    mNextPosY       = 5;
}
```

CreateNewPiece method sets the “next piece” as the current one and resets its position, then selects a new “next piece”.

view plain copy to clipboard print ?

```
/*
=====================================
Create a random piece
=====================================
*/
void Game::CreateNewPiece()
{
    // The new piece
    mPiece          = mNextPiece;
    mRotation       = mNextRotation;
    mPosX           = (BOARD_WIDTH / 2) + mPieces->GetXInitialPosition (mPiece,
    mPosY           = mPieces->GetYInitialPosition (mPiece, mRotation);

    // Random next piece
    mNextPiece      = GetRand (0, 6);
    mNextRotation   = GetRand (0, 3);
}
```

DrawPiece is a really easy method that iterates through the piece matrix and draws each block of the piece. It uses green for the normal blocks and blue for the pivot block. For drawing the rectangles it calls to **DrawRectangle** method of the class “IO” that we will see later.

view plain copy to clipboard print ?

```
/*
=====================================
Draw piece

Parameters:

>> pX:        Horizontal position in blocks
>> pY:        Vertical position in blocks
>> pPiece:    Piece to draw
>> pRotation: 1 of the 4 possible rotations
=====================================
*/
void Game::DrawPiece (int pX, int pY, int pPiece, int pRotation)
{
    color mColor;               // Color of the block

    // Obtain the position in pixel in the screen of the block we want to draw
    int mPixelsX = mBoard->GetXPosInPixels (pX);
    int mPixelsY = mBoard->GetYPosInPixels (pY);

    // Travel the matrix of blocks of the piece and draw the blocks that are fil
    for (int i = 0; i < PIECE_BLOCKS; i++)
    {
        for (int j = 0; j < PIECE_BLOCKS; j++)
        {
            // Get the type of the block and draw it with the correct color
            switch (mPieces->GetBlockType (pPiece, pRotation, j, i))
            {
                case 1: mColor = GREEN; break;  // For each block of the piece e
                case 2: mColor = BLUE; break;   // For the pivot
            }

            if (mPieces->GetBlockType (pPiece, pRotation, j, i) != 0)
                mIO->DrawRectangle   (mPixelsX + i * BLOCK_SIZE,
                                      mPixelsY + j * BLOCK_SIZE,
                                      (mPixelsX + i * BLOCK_SIZE) + BLOCK_SIZE - 1
                                      (mPixelsY + j * BLOCK_SIZE) + BLOCK_SIZE - 1
                                      mColor);
        }
    }
}
```

DrawBoard is similiar to the previous method. It draws two blue columns that are used as the limits of the boards. Then draws the board blocks that are flagged as POS_FILLED in a nested loop.

view plain copy to clipboard print ?

```
/*
=====================================
Draw board

Draw the two lines that delimit the board
=====================================
*/
void Game::DrawBoard ()
{

    // Calculate the limits of the board in pixels
    int mX1 = BOARD_POSITION - (BLOCK_SIZE * (BOARD_WIDTH / 2)) - 1;
    int mX2 = BOARD_POSITION + (BLOCK_SIZE * (BOARD_WIDTH / 2));
    int mY = mScreenHeight - (BLOCK_SIZE * BOARD_HEIGHT);

    // Check that the vertical margin is not to small
    //assert (mY > MIN_VERTICAL_MARGIN);

    // Rectangles that delimits the board
    mIO->DrawRectangle (mX1 - BOARD_LINE_WIDTH, mY, mX1, mScreenHeight - 1, BLUE

    mIO->DrawRectangle (mX2, mY, mX2 + BOARD_LINE_WIDTH, mScreenHeight - 1, BLUE

    // Check that the horizontal margin is not to small
    //assert (mX1 > MIN_HORIZONTAL_MARGIN);

    // Drawing the blocks that are already stored in the board
    mX1 += 1;
    for (int i = 0; i < BOARD_WIDTH; i++)
    {
        for (int j = 0; j < BOARD_HEIGHT; j++)
        {
            // Check if the block is filled, if so, draw it
            if (!mBoard->IsFreeBlock(i, j))
                mIO->DrawRectangle ( mX1 + i * BLOCK_SIZE,
                                     mY + j * BLOCK_SIZE,
                                     (mX1 + i * BLOCK_SIZE) + BLOCK_SIZE - 1,
                                     (mY + j * BLOCK_SIZE) + BLOCK_SIZE - 1,
                                     RED);
        }
    }
}
```

DrawScene, just calls the previous methods in order to draw everything.

view plain copy to clipboard print ?

```
/*
=====================================
Draw scene

Draw all the objects of the scene
=====================================
*/
void Game::DrawScene ()
{
    DrawBoard ();                                                    // Draw the
    DrawPiece (mPosX, mPosY, mPiece, mRotation);                     // Draw the
    DrawPiece (mNextPosX, mNextPosY, mNextPiece, mNextRotation);     // Draw the
}
```

Step 4: Easy drawing, window management and keyboard input using SDL, isolated from the game logic

"IO.cpp" and "IO.h" are the files that implement the "IO" class. It uses SDL in order to creat the window, clear it, update the screen and take care of the keyboard input. You can check out "IO.cpp" and "IO.h" files in order to see its implementation. I'm not going to explain th methods that are SDL related. You can change this class in order to use a different rendere (like IndieLib, Allegro, OpenGL, Direct3d, etc).

This is the header ("IO.h"):

view plain copy to clipboard print ?

```
#ifndef _IO_
#define _IO_

// ------ Includes -----

#ifndef LINUX
#include "SDL/include/SDL.h"
#include "SDL/SDL_GfxPrimitives/SDL_gfxPrimitives.h"
#else
#include <SDL/SDL.h>
#include "SDL/SDL_GfxPrimitives/sdl_gfxprimitives.h"
#endif
#pragma comment (lib, "SDL/lib/SDL.lib")
#pragma comment (lib, "SDL/SDL_GfxPrimitives/SDL_GfxPrimitives_Static.lib")

// ------ Enums -----

enum color {BLACK, RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, WHITE, COLOR_MAX}; /

// ---------------------------------------------------------------------------
//                                    IO
// ---------------------------------------------------------------------------

class IO
{
public:

    IO                      ();

    void DrawRectangle      (int pX1, int pY1, int pX2, int pY2, enum color pC);
    void ClearScreen        ();
    int GetScreenHeight     ();
    int InitGraph           ();
    int Pollkey             ();
    int Getkey              ();
    int IsKeyDown           (int pKey);
    void UpdateScreen       ();

};

#endif // _IO_
```

Step 5: The main loop

The main loop is quite simple. In each frame we draw everything. Later, we use keyboard input in order to move the piece. Before each movement, we first check out if it is possible We also measure the time in order to move the piece down every n milliseconds. When th piece fall down one block, we check out if that movement is possible, if not, we store the piece in the board. We also check out if there are blocks in the upper row, if so, the game i over.

Let's see "Main.cpp" step by step:

First, we initialize all the classes. Then, we get the actual milliseconds, which will be used t determine when the piece should move down.

view plain copy to clipboard print ?

```
01. #include "Game.h"
02. #ifndef LINUX
03. #include <windows.h>
04. #endif
05.
06. /*
07. ==================
08. Main
09. ==================
10. */
11. int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLir
12. {
13.     // ----- Vars -----
14.
15.     // Class for drawing staff, it uses SDL for the rendering. Change the methoc
16.     // in order to use a different renderer
17.     IO mIO;
18.     int mScreenHeight = mIO.GetScreenHeight();
19.
20.     // Pieces
21.     Pieces mPieces;
22.
23.     // Board
24.     Board mBoard (&mPieces, mScreenHeight);
25.
26.     // Game
27.     Game mGame (&mBoard, &mPieces, &mIO, mScreenHeight);
28.
29.     // Get the actual clock milliseconds (SDL)
30.     unsigned long mTime1 = SDL_GetTicks();
```

This is the main loop. We can exit by pressing ESC. In each frame we clear and update the screen and draw everything.

view plain copy to clipboard print ?

```
01. // ----- Main Loop -----
02.
03. while (!mIO.IsKeyDown (SDLK_ESCAPE))
04. {
05.     // ----- Draw -----
06.
07.     mIO.ClearScreen ();         // Clear screen
08.     mGame.DrawScene ();         // Draw staff
09.     mIO.UpdateScreen ();        // Put the graphic context in the screen
```

We start with the input. If we press left, down or right we try to move the piece in that directions. We only move the piece if the movement is possible.

view plain copy to clipboard print ?

```
01. // ----- Input -----
02.
03. int mKey = mIO.Pollkey();
04.
05. switch (mKey)
06. {
07.     case (SDLK_RIGHT):
08.     {
09.         if (mBoard.IsPossibleMovement (mGame.mPosX + 1, mGame.mPosY, mGame.mPiec
10.             mGame.mPosX++;
11.             break;
12.     }
13.
14.     case (SDLK_LEFT):
15.     {
16.         if (mBoard.IsPossibleMovement (mGame.mPosX - 1, mGame.mPosY, mGame.mPiec
17.             mGame.mPosX--;
18.         break;
19.     }
20.
21.     case (SDLK_DOWN):
22.     {
23.         if (mBoard.IsPossibleMovement (mGame.mPosX, mGame.mPosY + 1, mGame.mPiec
24.             mGame.mPosY++;
25.         break;
26.     }
```

By pressing “x”, the piece will fall down directly to the ground. This is really easy to implement by trying to move the piece down until the movement is not possible. Then we store the piece, delete possible lines and check out if the game is over, if not, we create a new piece.

view plain copy to clipboard print ?

```
case (SDLK_x):
{
    // Check collision from up to down
    while (mBoard.IsPossibleMovement(mGame.mPosX, mGame.mPosY, mGame.mPiece, mGa

    mBoard.StorePiece (mGame.mPosX, mGame.mPosY - 1, mGame.mPiece, mGame.mRotati

    mBoard.DeletePossibleLines ();

    if (mBoard.IsGameOver())
    {
        mIO.Getkey();
        exit(0);
    }

    mGame.CreateNewPiece();

    break;
}
```

By pressing “z” we rotate the piece. With the methods that we have already implement thi is an easy task. The rotation is in fact to change to the next rotated stored piece. We first should check that the rotated piece will be drawn without colliding, if so, we sets this rotation as the current one.

view plain copy to clipboard print ?

```
    case (SDLK_z):
    {
        if (mBoard.IsPossibleMovement (mGame.mPosX, mGame.mPosY, mGame.mPiece, (
            mGame.mRotation = (mGame.mRotation + 1) % 4;

        break;
    }
}
```

If WAIT_TIME passed, the piece should fall down one block. We have to check out if the movement is possible, if not, the piece should be stored and we have to check if we can delete lines. We also see if the game is over, if not, we create a new piece.

view plain copy to clipboard print ?

```
    // ----- Vertical movement -----

    unsigned long mTime2 = SDL_GetTicks();

    if ((mTime2 - mTime1) > WAIT_TIME)
    {
        if (mBoard.IsPossibleMovement (mGame.mPosX, mGame.mPosY + 1, mGame.mPiec
        {
            mGame.mPosY++;
        }
        else
        {
            mBoard.StorePiece (mGame.mPosX, mGame.mPosY, mGame.mPiece, mGame.mRo

            mBoard.DeletePossibleLines ();

            if (mBoard.IsGameOver())
            {
                mIO.Getkey();
                exit(0);
            }

            mGame.CreateNewPiece();
        }

        mTime1 = SDL_GetTicks();
    }
}

return 0;
```

And that’s all! Please leave a comment if you see some mistakes, language errors or if you have any doubts… or just to say thanks!

Credits

- Javier López López
- Special thanks: Imelior, who fixed English mistakes and compiled the tutorial under

Linux.

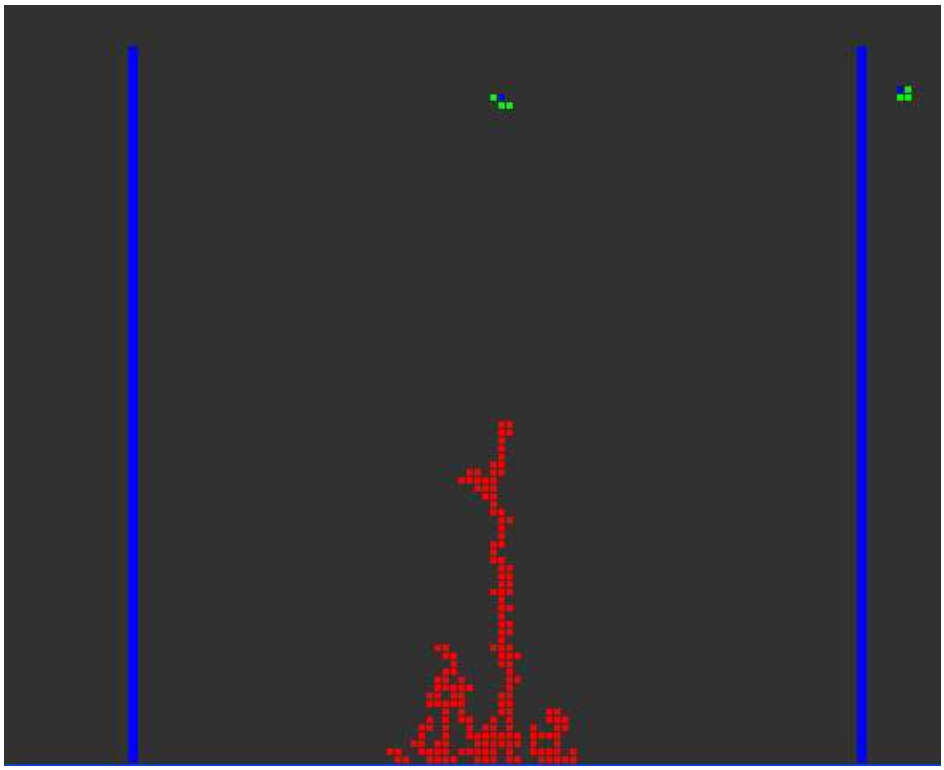
- Special thanks: Javier Santana, who added #ifndef sentences and pointed that was necessary to use libsdl-gfx1.2-dev and libsdl1.2-dev under Linux.

Bonus

Don't forget to play with the "defines". Crazy example:

view plain copy to clipboard print ?

```
01. #define BLOCK_SIZE 5                // Width and Height of each block of a
02. #define BOARD_WIDTH 90              // Board width in blocks
03. #define BOARD_HEIGHT 90             // Board height in blocks
```



How can you help me?

I have spent lot of hours of my free time for doing this tutorial. Some of you asked me how to say thanks for all the tutorials, and here you have the answer! Just keep reading and backlinking gametuto.com tutorials from your blogs! You can also follow me on Twitter. Thank you!

This entry was written by Javier López, posted on December 14, 2008 at 5:33 pm, filed under beginners tuts and tagged game logic, SDL, tetris, tutorials. Bookmark the permalink Follow any comments here with the RSS feed for this post. Post a comment or leave a trackback: Trackback URL.

Game Engines #1: IndieLib, 2.5d engine using c++ for rapid game development

80 Comments

1.  miguelSantirso
   Posted December 14, 2008 at 11:24 pm | Permalink

   Muy bueno el tutorial! La verdad es que estos tutoriales son una currada tremenda, pero también ayudan a mucha gente… ¡Espero que tengas ánimo para seguir con ello!

2.  AdrianMG

Posted December 15, 2008 at 1:27 am | Permalink

Nice tut thanks!

3.  Javier López
Posted December 15, 2008 at 1:55 am | Permalink

Thank you for the appreciative comments, Adrián and Miguel

4.  Joe Larson
Posted December 15, 2008 at 8:22 am | Permalink

I like your solution to having the output be scalable. I also find your rotation solutior interesting. I wouldn't have taken that approach. 16 lines of data per piece! Draws the code out, which is fine in most cases. But personally I prefer one piece declaratic and procedurally rotate it. It makes the code a bit harder to read, tho, but that's the way I did Alleytris.
(Hope you don't mind the plug *insert smiley*)

5.  Javier López
Posted December 15, 2008 at 1:15 pm | Permalink

Thank you taking a look to the tutorial, Joe. I usually prefer to have everything precalculated. It is usually easier to understand and faster. But a procedurally rotation is also interesting. Alleytris is a great example of a good tetris clones. Good work!

6.  Francois MAROT
Posted December 15, 2008 at 3:24 pm | Permalink

Nice tutorial, it reminds me of all the love and effort I put into my first C++ programr that was… Tetris !!! I had just learned what templates and subclasses were and used all that without knowing about pointers etc… But it was fun ! Anyway, as it was my 1st programm, it was in no way as clean as your solution, but what fun ! Thanks for reminding me this good ol' time

7.  Javier López
Posted December 15, 2008 at 3:55 pm | Permalink

Francois, Tetris also make me remember the first games I tried to develope 15 years ago using Basic in my old MSX. Snif! Snif!

8.  nihed
Posted December 15, 2008 at 11:40 pm | Permalink

good job
+1

9. Javier López
P□sted December 16, 2008 at 1:06 am | Permalink

Thanks, nihed.

10. Ivan Guardad□
P□sted December 16, 2008 at 2:21 pm | Permalink

Really is a great tut□rial! I'm h□ping t□ create my first game!

11. Javier López
P□sted December 16, 2008 at 11:30 pm | Permalink

@Ivan, thank y□u! D□n't f□rget t□ p□st y□ur game.

12. Emeka
P□sted December 22, 2008 at 7:39 am | Permalink

This is great, I have been thinking □f h□w t□ great this f□r s□mething n□w, and I have always being intimidated. N□w, y□u have d□ne the hard j□b f□r me. But, I feel like asking s□me stupid questi□ns. C□uld y□u supp□rt me get started in game pr□gramming? Or will y□u be there t□ answer my questi□ns?

13. Javier López
P□sted December 23, 2008 at 8:24 am | Permalink

Welc□me Emeka,

If y□u want t□ get started in game pr□gramming, I think this is the c□rrect place. Feel free t□ ask whatever questi□ns y□u have! I'll try t□ help y□u, □f c□urse!

I think a g□□d start is als□ visiting IndieLib.c□m. IndieLib is a 2d game engine I devel□ped f□r rapid game pr□t□typing under c++. I think is a g□□d engine in □rder t□ start learning game pr□gramming.

See y□u there and here!

S□□n… m□re tut□rials!

14. Emeka
P□sted December 24, 2008 at 8:46 am | Permalink

Javier,
Thanks s□ much, I checked □ut IndieLib game engine tut□rial and I liked what I f□unc
H□wever, the tut□rial is based □n □nly visual C++ 2008 Express Editi□n. I have different c□mpiler. H□pe IndieLib w□uld w□rk in a envir□nment.

Emeka

15. Javier López
P□sted December 24, 2008 at 3:25 pm | Permalink

Don't worry, post your doubts on IndieLib forums. I'm sure we will be able to help you.

16.  Santiago
Posted December 28, 2008 at 11:17 am | Permalink

Muy bueno el tutorial!
Una pregunta, estoy usando la librería IO que hiciste para hacer mi versión del tetris pero me sale un error de "undefined reference to _boxColor" cuando compilo con e CodeBlocks.
Te agradezco si me podés ayudar.
saludos!

17.  Javier López
Posted December 28, 2008 at 12:33 pm | Permalink

Bienvenido, Santiago!

El error posiblemente se deba a que no se ha linkado correctamente SDL_GfxPrimitives_Static.lib. La tienes en \tetris_tutorial_sdl\SDL\SDL_GfxPrimitives

Yo usé vc2008 Express Edition y para linkarla usé la directiva "pragma". Es decir, puse

#pragma comment (lib, "SDL/lib/SDL.lib")
#pragma comment (lib, "SDL/SDL_GfxPrimitives/SDL_GfxPrimitives_Static.lib")

Puede que en Code Blocks los linkados se hagan de manera distinta. No te lo sé deci porque no lo uso. Busca o pregunta cómo se linka una .lib en un proyecto de Code Blocks y supuestamente resolverás el problema.

Mantenme al tanto, espero que lo puedas solucionar

Si decides pasarte a vc20008 express edition, te recuerdo sólo que es gratis.

¡Un saludo y suerte!

18.  Santiago
Posted January 1, 2009 at 11:50 am | Permalink

Me anduvo con el vc++2008 usando el mismo proyecto del tutorial pero con mis archivos, muchas gracias!

19.  Callum
Posted January 8, 2009 at 6:48 pm | Permalink

Thanks for the well commented tutorial! I was wanting to create tetris but was unsu how i could implement the blocks and i certainly like your method

Just a quick question, how does the maths behind the GetXPosInPixels work? i just cant seem to figure out how you manage to convert block size into pixels and would love to find out how it actually does.

Thanks

20.  Javier López

Posted January 9, 2009 at 1:17 am | Permalink

You are welcome! 😀

So, for getting the X coordinate in pixeles we have:

return ((BOARD_POSITION - (BLOCK_SIZE * (BOARD_WIDTH / 2))) + (pPos * BLOCK_SIZE));

Maybe this line seems a bit tricky, so maybe you could try to use pen and paper in order to change the variables by real numbers.

Let's say:
BOARD_POSITION = 320 (in pixels)
BLOCK_SIZE = 16 (in pixels)
BOARD_WIDTH = 10 (in blocks)
pPos = 2 (Horizontal position of the block (in blocks)

So, having that, we now we want to get the x coordinate in pixels of a block that is ir a third column (it is in pPos 2 and we have to start counting by 0).

So, with this line:

((BOARD_POSITION - (BLOCK_SIZE * (BOARD_WIDTH / 2)))

We just one to calculate the x coordinate in pixels of the left corner of the board. Remember that BOARD_POSITION is the center of the board, in pixels. So we have tc substract to that value the amount in pixels of the half of the blocks that fill in a board.

After that we add to this value this line:

(pPos * BLOCK_SIZE)

This is just for getting the width of 2 blocks, that is the same as getting the x coordinate in order to draw a block in position 3.

I hope this was explanatory enough.

21.  Shotaro
Posted January 15, 2009 at 10:55 am | Permalink

This is brilliant! Thank you so much! Really, really helpful.

22.  Johan
Posted January 17, 2009 at 2:39 am | Permalink

Great tutorial, there is only one thing that's unclear for me and that is that in step 1 I'm not sure in what header file I should put the information of the blocks.

23.  Javier López
Posted January 17, 2009 at 9:49 am | Permalink

In fact I put the information not in the header but in the .cpp (check out the sourcecode you can download). But it could be in the header if you wish.

24.  Alexess
Posted January 20, 2009 at 9:45 am | Permalink

very good!! thanks you.

25.  Javier López
Posted January 20, 2009 at 10:16 am | Permalink

Thank to all you for the appreciative comments

26.  arbi
Posted January 23, 2009 at 11:48 pm | Permalink

hey Javier Lopez I am studying game and simulation programming at devry and I got many years to get to programm a game engine. I am just curious what books you rea or what helped you learn c++ to make a game engine by yourself. Wow that is amazing what you have done.. I love your work.

27.  Emeka
Posted January 24, 2009 at 9:40 am | Permalink

I have not been able to get my head around this method mPiecesInitialPosition , please explain it further.

28.  Bob
Posted January 25, 2009 at 2:38 am | Permalink

… I've written a Java Tetris years ago - it's an applet and should run in your browser (Java's installed).

The game does not use matrices or anything - the positions are hard-coded too, but in an even simpler way

29.  Javier López
Posted January 25, 2009 at 5:55 am | Permalink

@Emeka, mPiecesInitialPosition is just an array that has the displacement that each piece (the 4 possible rotations) has to suffer in order to be positioned when it is created. Just check out the two images that shows that in the tutorial:
http://gametuto.com/images/wrong_and_good_tetris_positions.png
http://gametuto.com/images/wrong_and_good_tetris_positions.png

In the first image, we has drawn the piece without applying the translation, so it is not correctly positioned when created. On the second image, we have applied the translation to the piece, so it is correcly positioned when created.

I hope this will clarify your doubt.

30.  Javier López
Posted January 25, 2009 at 6:44 am | Permalink

@arbi, thank you very much for the appreciative words. It took me lot of years to in order to arrive to the 1.0 release. I think I spent more or less 5 years, but only workin in spare time and with breaks of several months.

I'm quite autodidact, I learnt c++ by myself, and before that Pascal, and Basic. I also work using Java and php. And I'm starting to enjoy c#. I've been programming since I was 9 years old and I'm currently 27 years old.

I don't know any book I can tell you to take a look. Anything I needed was is on Google, specially reading source code from other open source engines and talking ir forums. I used to write a lot on stratos-ad.com forums (spanish). Now, I usually write on IndieLib forums: http://www.indielib.com/forum

It would be great if you come to IndieLib forum and join our community. People like you, motivated and well prepared, will give the community a lot of value. Furthermore, if you are interested on engine programming, maybe you want to join the IndieLib development team, what do you think?

See you!

31.  Javier López
Posted January 25, 2009 at 6:46 am | Permalink

@Bob, that seems cool. What about explaining it a bit more? I'm interested!

32.  Vitor Almeida
Posted January 26, 2009 at 4:01 am | Permalink

Very nice tutorial (simple, easy to understand and detailed). Backlinked to my blog. Keep up the good work.

33.  Javier López
Posted January 26, 2009 at 4:47 am | Permalink

@Victor Almeida, thank you very much neighbour (iberic) 😃

34.  arbi
Posted February 6, 2009 at 9:01 pm | Permalink

Javier,
Thank you for your reply sorry I am replying so late. Been busy programming. Taking courses now. I am not so much a self learner but I am starting to be. Anyways the book that I use right now are C++ primer plus, Problem solving with C++, game programming with C++, and i look online for http://www.cplusplus.com/tutorial. It will take me some time just like you to really get a grasp of this language. One question I had did you first truely understand one language before you jumped into another or did you learn two at the same time. I am studying c++ but I know Java is easier since it has a garbage collector. Also C++ is ideal for game development but java is still used.

I will for sure joing your forum. Thank you again for the reply. Sorry for the late reply

35.  Javier López
Posted February 8, 2009 at 5:13 pm | Permalink

@arbi

Hello again! That books you are talking about seems really interesting.

Answering your question, sometimes, like right now, I have to deal with learning different languages at the same time. For example, currently I'm learning Php, because I need it for my day work. But I'm still learning C++… because you know: you never knows everything about nothing

The best way to go is to have a good knowledge of POO programming and the different programming metodologies. Then is just a matter of getting some new concepts and keywords when learning a new language.

36. Veiko
Posted February 10, 2009 at 3:13 pm | Permalink

Hi Javier.

This is really good tutorial. Its clean, its simple and it explains really good whats goin on. The big bonus is pictures. I have searched such Tetris tutorial for a long time.

I have bookmarked your site and this tutorial.

Thanks alot Javier.

37. Javier López
Posted February 10, 2009 at 4:20 pm | Permalink

@Veiko, thanks alot to you for reading!

38. BERNARD
Posted February 15, 2009 at 7:42 am | Permalink

This is good man. Do some of these code lines change when using borland c++. I am beginner in c++ programming.

39. vkson
Posted February 15, 2009 at 8:49 pm | Permalink

Thanks Javier López.
Tutor is detail. Easy to understand. But Could you add rotation solution into project very good
Thanks

40. Niko
Posted February 16, 2009 at 3:15 am | Permalink

Javier thank you so much for very good tutorial/lesson. Keep up good work!

41. Dave
Posted February 20, 2009 at 4:37 pm | Permalink

Thanks that was a really great tutorial!

I want to get into programming games and have found this really useful. Please write

more soon!

I was motivated to build tetris by this great lecture about an academic course that looks inspiring.
http://video.google.com/videoplay?docid=7654043762021156507

http://www1.idc.ac.il/tecs/

Thought you might enjoy these fun tetris vids:

http://www.youtube.com/watch?v=SYRLTF71Sow&feature=related

http://www.notsonoisy.com/tetris/index.html

42. Bobic
Posted February 24, 2009 at 6:07 am | Permalink

Very Good!
Thanks.

43. Nick
Posted March 2, 2009 at 5:18 am | Permalink

Thanks for writing a good tutorial. I've found it to be in-depth and very easy to understand.

Just one question - why the use of WinMain and windows.h in main.cpp? From what have done in the past using SDL on Windows, you can just use the normal int main(int argc, char** argv) entry point.

44. chris
Posted March 15, 2009 at 7:45 pm | Permalink

Thank you so much for this tuitorial. I have some questions, How can i execute this game? Right now i'm figuring it out but failed to execute it. Do i need to debug it in C++? and what file? or how can i make an exe DOS file out of it, to make it work?

45. mya
Posted April 18, 2009 at 12:06 pm | Permalink

nice ,awesome !!!!!!!!
i expect more such ideas in upcoming game

46. Roque Terrani
Posted April 19, 2009 at 5:13 pm | Permalink

Muchísimas gracias por el código y el tutorial, me ayudo bastante a terminar de darme la idea de como afrontar la mecánica del juego. Te comento que yo lo estoy programando en C para *PIC (microcontrolador de la empresa Microchip) y ya tenia bastante pensado los diagramas de la lógica del juego pero tu código me ayuda en la cuestión técnica de la programación. Obviamente ni bien lo tenga terminado paso a dejar fotos del bicho funcionando asi todos nos sentimos orgullosos.

pd: Para los amantes de la electrónica dejo una breve explicación de lo que estoy

haciendo. Pic 16f877a, 4 registros de desplazamiento para las filas, 6 matrices de 7×5 leds(2,5 cm x 5,3 cm en total), 3 display de 7 segmentos para las lineas y 4 pulsadore que hacen de teclas (Luego serán remplazados con una interfaz a teclado ps/2)

47. Wai Kam
Posted May 5, 2009 at 7:34 pm | Permalink

Love it, and thx!!!

48. Battie
Posted May 18, 2009 at 4:03 am | Permalink

Thank you very much for this wonderfull tutorial… However i want to implement something that it computes what the highest point of the POS_FILLED is.

The time to drop a block should always be 6 no matter how much there POS_FILLED there are.

in addition to that there should be 4 seconds inbetween a block is dropped and a new block appears in the air.

Is this possible or not?

49. Soo
Posted May 29, 2009 at 12:34 pm | Permalink

Nice tutorial, thanks you a lot for that.
I just have one question, i got 1 error when i tried to compile this project. It is about an undefined reference to 'boxColor'.
Can anyone help me with this plz ?

50. Sam L.
Posted June 13, 2009 at 5:49 am | Permalink

Hi Javier, I translate your tutorial to Brazilian Portuguese.
Download:
http://www.4shared.com/file/95111285/e93a8160/Tetris_tutorial_para_iniciantes.html

vlw!

51. sshow
Posted June 16, 2009 at 4:23 pm | Permalink

Great tutorial!

I'm using this to write my first game in C#.
Planning on implementing multi-player features like in TetriNET.

Great introduction for me, even though I've never used C++.
The logic was basically what I was looking for, and you are explaining things _well_!

Thanks!

52. Alekh
Posted June 17, 2009 at 1:06 am | Permalink

Hi a really nice tutorial. But I want to add some animations to this like the line complete animation. How do i implement such animations? Pls help me on this.. I really need help..

53. Tom
Posted June 23, 2009 at 8:55 am | Permalink

Hey - thanks alot for this tutorial - it's very enlightening.

However, I am having trouble understanding the StorePiece method in the Board class. I just can't seem to follow it. Could you explain in more detail what this is doing? Also, why are you adding the PIECE_BLOCKS constant to pX and pY?

Thanks

54. Emeka
Posted June 23, 2009 at 11:51 am | Permalink

Hello,
I can't find where this field " mScreenHeight" was assigned value?

Emeka

55. jimmy
Posted June 29, 2009 at 10:06 am | Permalink

Had a lot of fun going through this great tutorial, but I run into the craziest problem when I try to alter the values of the #define's… the compiler still keeps the original #define values!

I've tried changing the #define's to const int's, and I've even completely commented out the #define's… and the compiler still doesn't mind.

How does this happen??

56. roxlu
Posted July 15, 2009 at 1:48 am | Permalink

Wow amazing! What a clear, nice code!
Though I'm wondering, in Board::StorePiece() you make a call to mPieces->GetBlockType(pPiece, pRotation, j2, i2) .. should this be: mPieces->GetBlockType(pPiece, pRotation, i2, j2)
.. for clearity sake?

57. Heather
Posted July 19, 2009 at 2:19 pm | Permalink

You know, you can very easily, without complex code, rotate the tetris pieces and cu down on your piece array.

In a nested loop where the array is 5 wide by 5 wide:

No rotation: newarray[j][i] = oldarray[j][i]
1 step counterclockwise: newarray[4 - i][j] = oldarray[j][i]
2 step counterclockwise: newarray[4 - j][4 - i] = oldarray[j][i]
3 step counterclockwise: newarray[i][4-j] = oldarray[j][i]

58. ~MoZzY~
Posted July 28, 2009 at 8:00 pm | Permalink

Very good tutorial…2 thumbs up…

59. FFUUU
Posted July 31, 2009 at 2:49 am | Permalink

Not another Tetris clone!

60. Joe
Posted August 23, 2009 at 2:33 pm | Permalink

Greetings,

Has anyone managed to run this in MacOSX? I managed to build it but when I tried t
run it I got the following error message: Terminating app due to uncaught exception
'NSInternalInconsistencyException', reason: 'Error (1002) creating CGSWindow'

61. Skaruts
Posted September 16, 2009 at 11:56 pm | Permalink

Just a minor detail I noticed: I remember in the original tetris that the pieces rotatio
only cycled through 2 positions.

Taking the I shape example figures, figures 3 and 4 wouldn't exist. They would only
cycle through 1 and 2.

Despite that, very nice and neat tut.
Thanks.

62. adnoctum
Posted November 11, 2009 at 12:34 pm | Permalink

Thx man, i will help from this because i will make a tetris in VB, thanx again

63. feng
Posted November 13, 2009 at 1:08 am | Permalink

vr nice n well explained tutorial
reli appreciate it!
but i was stuck in here :

```
bool Board::IsGameOver()
{
//If the first line has blocks, then, game over
```

```
for (int i = 0; i < BOARD_WIDTH; i++)
{
if (mBoard[i][0] == POS_FILLED) return true;
}

return false;
}

xxxxxxxxxx
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
1111111111
```

this function is it checking the xxxxxx row?
then i tot suppose the if statement sud look like this
“if (mBoard[i][0] == POS_FILLED) return true;”

anyone pls correct me if im wrong

64.  feng
Posted November 13, 2009 at 1:10 am | Permalink

(sorry the upper comment is wrong, pls refer to this)

vr nice n well explained tutorial
reli appreciate it!
but i was stuck in here :

```
bool Board::IsGameOver()
{
//If the first line has blocks, then, game over
for (int i = 0; i < BOARD_WIDTH; i++)
{
if (mBoard[i][0] == POS_FILLED) return true;
}

return false;
}

xxxxxxxxxx
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
1111111111
```

this function is it checking the xxxxxx row?
then i tot suppose the if statement sud look like this
“if (mBoard[0][i] == POS_FILLED) return true;”

anyone pls correct me if im wrong

65. feng
Posted November 13, 2009 at 1:20 am | Permalink

oohh.. i understand d
mBoard[x(width)][y(height)]

66. supreme aryal
Posted November 14, 2009 at 12:52 pm | Permalink

Thank you for the tutorial. I implemented it in C. The code looks much the same, but it is in one large file. You can see it in action at:

http://www.youtube.com/watch?v=WD52EoZHNJA

67. soitsthateasy
Posted November 28, 2009 at 2:26 pm | Permalink

great tut,

just one difficulty i'm having with it.
forgive me if it's obvious, i'm new to game programming!!!

I created a new piece (a cross) and i'm not sure how to make the random piece function thingy include it in it's selections…

if someone could post and say all the places i would have to change, it would be greatly appreciated

68. afshin
Posted November 30, 2009 at 11:06 am | Permalink

thanks javier.you're my hero

69. feng
Posted December 3, 2009 at 7:30 pm | Permalink

very nice, neat and easy understanding tetris tutorial!
appreciate this so much !

70. Pilot Boy
Posted December 22, 2009 at 3:09 am | Permalink

Nice tut! Just what I needed!

71. amir yazdanbakhsh
Posted December 26, 2009 at 8:00 am | Permalink

Dear López,
Thx for your brilliant game and tutorial.
I'm going to write a simple game that are somehow similar with Tetris. I want captur
mouse clicks. If player click on the one of these blocks, that blocks disapper, and if th

blocks reach bottom of the board the user get a negative point.
I'd appreciate if you could help me.

Best Regards

72. Hyoungseok
Posted January 13, 2010 at 7:54 am | Permalink

I was looking for a good tutorial making TETRIS!
so here I am

I'm not good in english but I'd like to say "thank you very much"

from south korea

73. zhunie chen
Posted February 8, 2010 at 5:33 pm | Permalink

heLLo!!

ammhpp…anyone cud help me how to make a bingo chess game in Turbo C????

pLz……

74. Mahi
Posted March 15, 2010 at 10:59 pm | Permalink

Hi,

I went through the whole code and i really really appriciate it, though i am a rookie i
programming. So can anyone help me getting this code with direct3d please? i will
really appreciate it

Thnx,

Mahi

75. Eko
Posted March 26, 2010 at 12:48 am | Permalink

Hello going through your code, but stopped at

```
int Pieces::GetXInitialPosition (int pPiece, int pRotation)
{
return mPiecesInitialPosition [pPiece [pRotation][0];
}
```

the method only have 2 parameter, but return 3? What is the third doing "[0]" ? Anc
why doesnt it say anything about it in description above? Ty for help.

76. fina
Posted April 3, 2010 at 4:51 pm | Permalink

tahnk u so much for your knowledge…:)

77. hordi
Posted April 8, 2010 at 2:59 am | Permalink

This tutorial shows Tetris logic very nicely. I started to make a Tetris game with C# an this really helps.

One thing though, I fail to see the point of the pivot points since they're not even used to rotate the piece.

78. Mariano
Posted April 21, 2010 at 5:22 am | Permalink

¿Que compilador o programa usas?(Compiler or what program do you use?)

79. Douglas
Posted April 21, 2010 at 10:52 am | Permalink

estou começando agora a programar jogos, você tem algum tutorial de como fazer engine pra principiantes ou algum livro para me recomendar

80. lyzen
Posted May 21, 2010 at 10:19 am | Permalink

i want a program c++ games or ATM
before thanks,,,

5 Trackbacks

1. By Tutorial: Tetris en c++ para novatos on December 14, 2008 at 6:11 pm

   [...] Tetris tutorial in c++ render independent in one hour [...]

2. By pixelame.net on December 14, 2008 at 7:10 pm

   Tutorial: Tetris en c++ para novatos, independiente del render [ING]…

   He echado unas cuantas horas este fin de semana y he hecho un tutorial que explica paso a paso la creación de un clon del Tetris usando c++. El render está separado de la lógica del juego y usé simplemente primitivas de SDL (rectángulos). La idea …

3. By meneame.net on December 16, 2008 at 2:29 am

   Tutorial para programar un clon del Tetris en c++ en una hora para linux y windows [ING]…

   Tutorial, paso a paso, sobre como programar un clon del Tetris independiente de la plataforma usando SDL. Se centra en la programación de la lógica del juego y no en los gráficos. Recomendado sobre todo para principiantes, pero curioso de ver para l

4. By   Game Programming Tutorial: Tetris in C++ » GBGames - Thoughts on Indie Game Development on December 16, 2008 at 6:01 am

[...] Lopez wrote a beginner's tutorial to write Tetris in C++. The tutorial is platform-independent, which I like. While some people have complained that the [...]

5. By rascunho » Blog Archive » links for 2008-12-16 on December 16, 2008 at 12:12 pm

   [...] Tetris tutorial in C++ platform independent focused in game logic for beginners We are going to learn how to create a Tetris clone from scratch using simple and clea C . (tags: gametuto.com 2008 mes11 dia16 Tetris C++ tutorial OOP blog_post) [...]

Post a Comment

Your email is *never* published nor shared. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment