Файл model.cs

Там два таймера

1)

/// <summary>

/// Таймер отвечающий за движение текущей фигуры вниз

/// </summary>

public void TimerTickMoveDown(Object source, ElapsedEventArgs e)

2)

/// <summary>

/// Таймер отвечающий за движение фигуры по командам от пользователя

/// </summary>

public void TimerTickUserCommand(Object source, ElapsedEventArgs e)

Один таймер отвечает за падение фигуры вниз, он просто спускает фигуру вниз на одну клетку ч/з своё время.

Другой таймер отвечает за команды пользователя: влево, вправо, повороты. Этот таймер работает «быстрее», т.е. чаще срабатывает, поэтому за время падения фигуры на одну клетку вниз, пользователь может успеть несколько раз сдвинуть/повернуть фигуру

код в каждом из них выполняется в своём потоке.

Эти два таймера работают параллельно.

Чтобы не было конфликта доступу к Action(ref currentFigure, GameBox, action);

используем  
/// <summary>

/// Объект используемый для синхронизации потоков

/// </summary>

private readonly object toDoLock = new object();

т.е. вот так:

lock (toDoLock)

{

return Action(ref currentFigure, GameBox, action);

}

**------------------------------ ------------------------------------------------------------------------------**

Чтобы в Viewer выводилась информация без конфликтов,  
а они получаются из-за того, что  
код вызванный в Таймерах выполняется в своих потоках, (а отрисовка в Windows Form должна быть только в потоке создания Windows Form)

и именно в этом же потоке (таймера) заставялем сработать событиям (event) и вызывается код View-ра и выполняется   
а на самом деле информация во view должна выводится только в потоке в котором создано окно и элементы на нём ( например в TextBox.Text=”привет”,)  
поэтому используем Контекст Синхронизации

public SynchronizationContext SyncContext { get; set; }

// используем, чтобы события вызывались в том же потоке,

// в котором создано окно Windows

**Паттерны**

1)Для создания фигур используется паттерн Фабричный метод

2) Паттерн Одиночка /Singelton  
используем для class GameBox

public static GameBox GetInstance()

{

if (instance == null)

instance = new GameBox();

return instance;

}

**пока написан только один тест nUnit**

TestFigureCloneMethod

**Координаты фигуры**

Вращение происходит вокруг «цента» фигуры. Пример

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | |  | x |  | | 0 |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  | x |  | |  | 0 |  | | |  |  |  | | --- | --- | --- | |  |  |  | | 0 | x |  | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  | x |  | | 0 |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  | x |  | | 0 |  |  | |

X – центр вращения фигуры

0 – (x,y) координата фигуры = левого нижнего угла **прямоугольника** фигуры. Эти координаты при повороте пересчитываются

Пример кода

public FigureJ(int x) : base(x, 3, ConsoleColor.Blue, new int[2, 3] { { 1, 0, 0 }, { 1, 1, 1 } })

{

functionsForRotateLeft[0] = (ref int x, ref int y) => { x ++; y+=0; };

functionsForRotateLeft[1] = (ref int x, ref int y) => { x --; y --; };

functionsForRotateLeft[2] = (ref int x, ref int y) => { x+=0; y ++; };

functionsForRotateLeft[3] = (ref int x, ref int y) => { x+=0; y+=0; };

}