Chapter

0

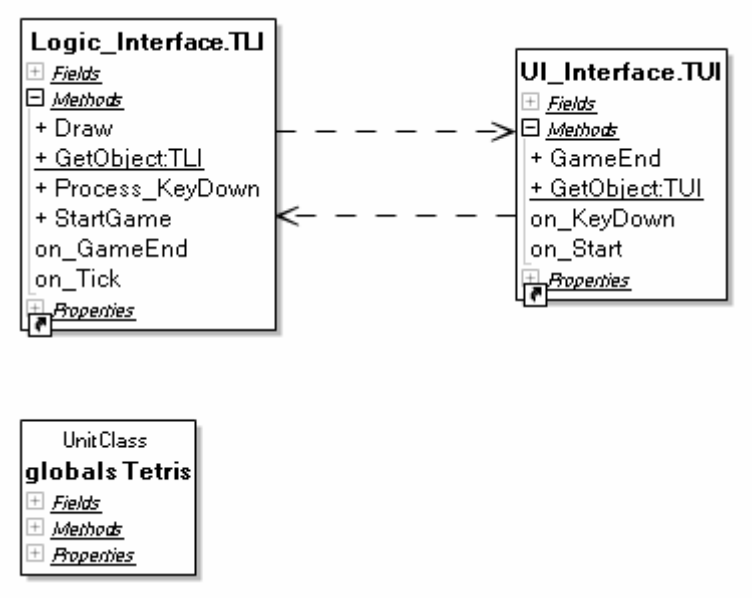
테트리스 게임 만들기

본 강좌의 목적은 테트리스 게임 자체보다는 사용자 인터페이스와 논리구조를 분리하여 분석/설계하고 코딩하는 과정을 설명하기 위한 것이다. 특히 BDS 2006에서 새로 추가된 Together for Delphi를 이용하여 보다 쉽게 객체지향적 설계를 실제 업무에 도입하는 과정을 설명하고자 한다.

1단계 - UI와 Logic의 분리

이번 강좌에서는 필자가 원래 사용하던 문서작성법이 아닌 클래스 다이어그램만으로 기능설계와 구조설계를 병행하도록 하겠다.

User-Interface의 기능분석



[그림 1] UI와 Logic의 분리

[그림 1]에서 우선 우측의 UI_Interface.TUI 클래스를 주목하자. 해당 클래스의 멤버들에서 사용된 작명규칙을 보면, Public 메쏘드 앞에는 아무런 접두어가 붙지 않지만, 이벤트에 해당하는 메소드 앞에는 “on_”을 붙여놨다. 이것은 필자만의 습관으로, 이름만으로 해당 메쏘드의 역할을 쉽게 구별하기 위해서이다. 추후 내부처리용으로 사용되는 Private 메쏘드는 “do_”를 앞에 붙이게 된다.

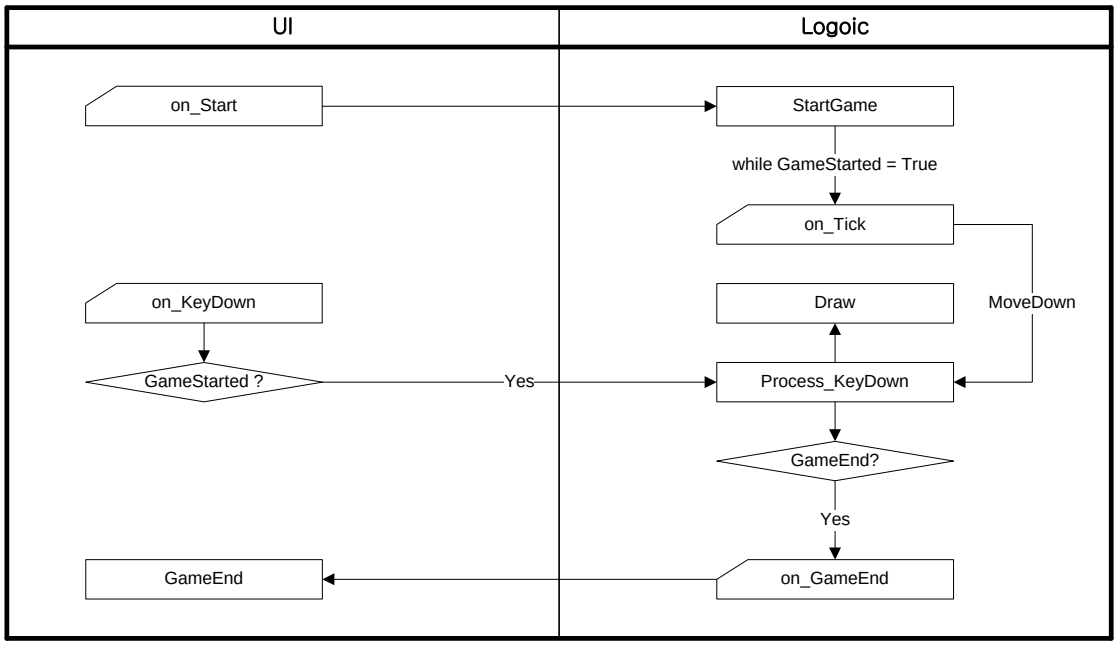
[그림 1]의 UI_Interface.TUI 클래스는 아래와 같은 기능을 가져야 한다.

- 기능
  - GameEnd : 게임 종료 처리
- 이벤트
  - on_Start : 사용자가 게임을 시작하도록 하였음
  - on_KeyDown : 사용자가 키보드를 클릭하였음

Logic-Interface의 기능분석

이어서 [그림 1]의 좌측 Logic_Interface.TLI 클래스는 테트리스 게임의 논리계층의 최상의 클래스이며, 해당 클래스는 다음과 같은 기능을 가져야 한다.

- 기능
  - StartGame
  - Process_KeyDown
  - Draw
- 이벤트
  - on_Tick
  - on_GameEnd



[그림 2] Job Flow

[그림 2]에서는 UI와 Logic이 서로 호출하는 순서와 이벤트의 흐름을 설명하고 있다.

on_Start이벤트가 발생하면 Logic의 GameStarted 속성을 True로 변경한다. GameStarted 속성이 True로 변경되면 내부 타이머가 작동하면서 주기적으로 on_Tick 이벤트를 발생한다.

On_Tick 이벤트는 블록을 한 칸씩 밑으로 이동하는 메소드를 호출한다. 이동을 담당하고 있는 Process_KeyDown 메소드는 더 이상 블록을 내려놓을 수 없을 때, GameStarted 속성을 False로 변경시키면서 on_GameEnd 이벤트를 발생시킨다.

사용자가 키를 누르게 되면 on_KeyDown 이벤트가 발생하게 되고, 게임 중일 경우에만 Logic에게 이 메시지를 전달하게 된다.

사실 기능분석이 분석의 가장 기본이 되기는 하지만, 업무분석에는 동적분석이 더욱 효율적일 때가 많다. 어느 분석을 먼저 하더라도 분석은 단일 프로세스로 끝나는 경우는 거의 없다. 즉, 기능분석을 마치고 동적분석을 하는 동안 필요한 기능이 분석되지 않은 것을 발견했을 경우 다시 기능분석을 변경하고, 동적분석 후에도 기능분석을 통해서 동적분석의 오류를 발견할 수도 있다.

기능분석과 구조분석 그리고 동적분석은 진행하는 동안 서로 보완될 경우가 많다. 그리고, 반드시 “어느 것을 먼저 해야 한다”라는 규칙은 없다.

[소스 1] UI_Interface Unit의 소스

```
 1 : unit UI_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils;
 7 :
 8 : type
 9 :   TUI = class
10 :   protected
11 :     procedure on_KeyDown(Key:Word);
12 :     procedure on_Start;
13 :   public
14 :     class function GetObject:TUI;
15 :     procedure GameEnd;
16 :   end;
17 :
18 : implementation
19 :
20 : uses
21 :   Logic_Interface;
22 :
23 : var
24 :   MyObject : TUI = Nil;
25 :
26 : { TUI }
27 :
28 : class function TUI.GetObject: TUI;
29 : begin
30 :   if MyObject = Nil then MyObject:= TUI.Create;
31 :   Result:= MyObject;
32 : end;
33 :
```

```
34 : procedure TUI.on_Start;
35 : begin
36 :   TLI.GetObject.StartGame;
37 : end;
38 :
39 : procedure TUI.on_KeyDown(Key:Word);
40 : begin
41 :   if TLI.GetObject.GameStarted = True then TLI.GetObject.Process_KeyDown(Key);
42 : end;
43 :
44 : procedure TUI.GameEnd;
45 : begin
46 :   {ToDo : }
47 : end;
48 :
49 : end.
```

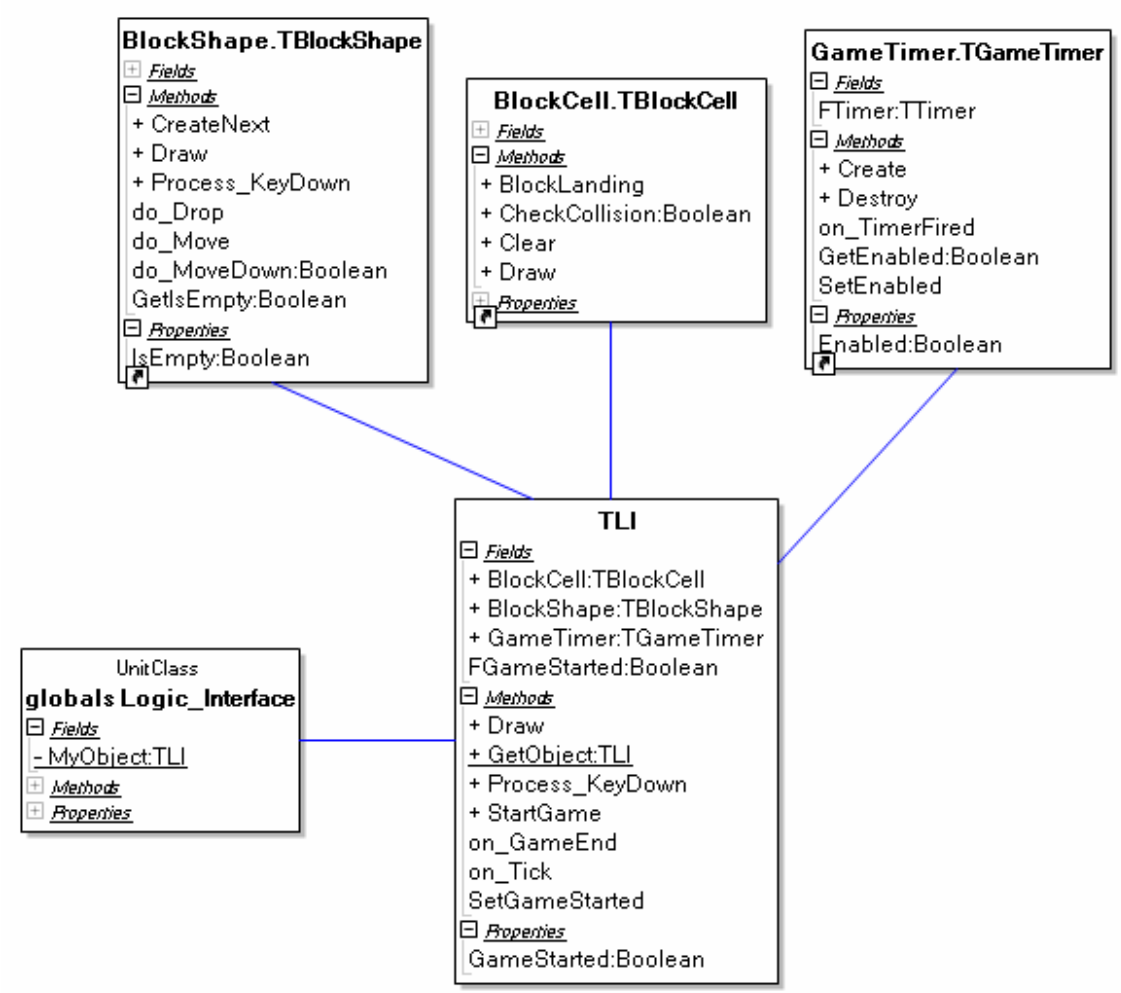
[소스 2] Logic_Interface Unit의 소스

```
 1 : unit Logic_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   BlockShape, BlockCell, GameTimer, Windows, Classes, SysUtils;
 7 :
 8 : type
 9 :   TLI = class
10 :   private
11 :     FGameStarted: Boolean;
12 :     procedure SetGameStarted(const Value: Boolean);
13 :   protected
14 :     procedure on_GameEnd;
15 :     procedure on_Tick;
16 :   public
17 :     GameTimer : TGameTimer;
18 :     BlockCell : TBlockCell;
19 :     BlockShape : TBlockShape;
20 :     class function GetObject:TLI;
21 :     procedure StartGame;
22 :     procedure Draw;
23 :     procedure Process_KeyDown(Key:Word);
24 :   published
25 :     property GameStarted : Boolean read FGameStarted write SetGameStarted;
26 :   end;
27 :
28 : implementation
29 :
30 : uses
31 :   UI_Interface;
32 :
33 : var
```

```
34 :   MyObject : TLI = Nil;
35 :
36 : { TLI }
37 :
38 : class function TLI.GetObject: TLI;
39 : begin
40 :   if MyObject = Nil then MyObject:= TLI.Create;
41 :   Result:= MyObject;
42 : end;
43 :
44 : procedure TLI.SetGameStarted(const Value: Boolean);
45 : begin
46 :   FGameStarted := Value;
47 :
48 :   if Value = True then Self.StartGame
49 :   else Self.on_GameEnd;
50 : end;
51 :
52 : procedure TLI.StartGame;
53 : begin
54 :   {ToDo : }
55 : end;
56 :
57 : procedure TLI.on_GameEnd;
58 : begin
59 :   TUI.GetObject.GameEnd;
60 : end;
61 :
62 : procedure TLI.on_Tick;
63 : begin
64 :   Self.Process_KeyDown(VK_Down);
65 : end;
66 :
67 : procedure TLI.Draw;
68 : begin
69 :   {ToDo : }
```

```
70 : end;
71 :
72 : procedure TLI.Process_KeyDown(Key:Word);
73 : begin
74 :   case Key of
75 :       VK_Left :  {ToDo : };
76 :       VK_Right : {ToDo : };
77 :       VK_Up :    {ToDo : };
78 :       VK_Down :  {ToDo : if 게임종료 then GameStarted:= False};
79 :       VK_Space : {ToDo : 바닥까지 VK_Down 처리};
80 :   end;
81 :
82 :   Self.Draw;
83 : end;
84 :
85 : end.
```

기능분석

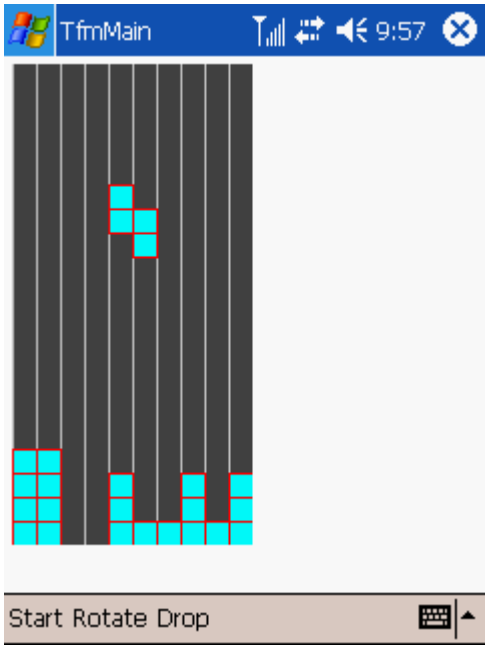


[그림 3] Logic Interface Class Diagram

우선 Logic을 구성하는 각 클래스의 역할분담에 대해서 설명하도록 하겠다. TGameTimer는 주기적으로 이벤트를 발생하여 테트리스의 블록들이 바닥으로 떨어지게 하는 역할을 담당한다.

TBlockCell은 테트리스의 블록이 쌓여있는 정보를 간직하기 위해서 작성되었다. (복수형태인 TBlockCells라고 이름을 지어야 할 것을 나중에 발견했다. 일단 넘어가자.)

TBlockShape는 현재 떨어지고 있는 블록들의 묶음에 대한 정보를 관리한다.



[화면 1] PDA용으로 작성된 프로그램의 실행 화면

[화면 1]은 이 강좌를 진행하기 위해 만들어진 데모 프로그램의 실행화면이다. [화면 1]에서 보듯이 현재 떨어지고 있는 블록모양(TBlcokShape)은 총 4개의 단위 블록으로 구성되어 있다. 이것을 표현하기 위해서 TShapeBlock이라는 클래스를 생성하였다. 또한 바닥에 이미 쌓여있는 블록은 TBlockCell 클래스가 그 정보를 관리하게 된다.

TGameTimer의 기능목록

- 기능
  - 없음
- 이벤트
  - on_TimerFired

TBlockShape의 기능목록

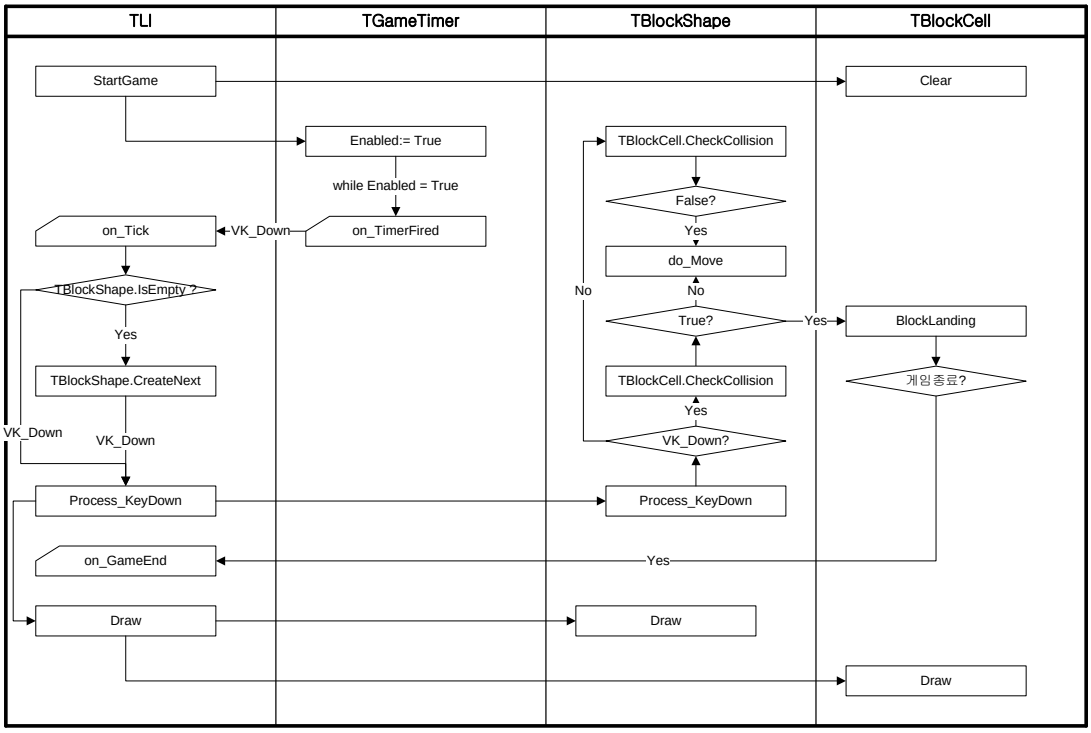
- 기능
  - CreateNext
  - Draw
  - Process_KeyDown

- 이벤트
  - 없음

TBlockCell의 기능목록

- 기능
  - Clear
  - Draw
  - BlockLanding
  - Checkcollision
- 이벤트
  - 없음

동적분석



[그림 4] Job Flow

[그림 4]에서는 동적분석을 통해서 Logic 부분의 전반적인 흐름을 표현하고 있다. 여기서는 기능 분석에서 발견하지 못했던 “do_Move” 메소드가 필요하다는 것을 발견하였다. 이제 기능분석을 통해 작성한 클래스 다이어그램을 수정하여야 한다.

이것은 소스상에서만 반영하도록 하겠다. 현재 필자가 사용하고 있는 BDS 2006에서는 소스를 변경하면 클래스 다이어그램이 자동으로 변경되기 때문에 변경작업에 의한 문서작성에 대해서는 스트레스 받을 필요가 없다.

또한, “do_Move”는 메쏘드 이름에서 알 수 있듯이 Private 메쏘드이다. 따라서, 기능분석 동안에 찾아내기는 쉽지 않다.

[소스 3] Logic_Interface Unit의 수정

```
 1 : unit Logic_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   BlockShape, BlockCell, GameTimer, Windows, Classes, SysUtils;
 7 :
 8 : type
 9 :   TLI = class
10 :   private
11 :     function GetGameStarted: Boolean;
12 :   protected
13 :     procedure on_GameEnd;
14 :     procedure on_Tick;
15 :   public
16 :     GameTimer : TGameTimer;
17 :     BlockCell : TBlockCell;
18 :     BlockShape : TBlockShape;
19 :     constructor Create;
20 :     destructor Destroy; override;
21 :     class function GetObject:TLI;
22 :     procedure StartGame;
23 :     procedure Draw;
24 :     procedure Process_KeyDown(Key:Word);
25 :   published
26 :     property GameStarted : Boolean read GetGameStarted;
27 :   end;
28 :
29 : implementation
30 :
31 : uses
32 :   UI_Interface;
33 :
```

```
34 : var
35 :   MyObject : TLI = Nil;
36 :
37 : { TLI }
38 :
39 : function TLI.GetGameStarted: Boolean;
40 : begin
41 :   Result:= GameTimer.Enabled;
42 : end;
43 :
44 : class function TLI.GetObject: TLI;
45 : begin
46 :   if MyObject = Nil then MyObject:= TLI.Create;
47 :   Result:= MyObject;
48 : end;
49 :
50 : procedure TLI.StartGame;
51 : begin
52 :   GameTimer.Enabled:= True;
53 : end;
54 :
55 : procedure TLI.on_GameEnd;
56 : begin
57 :   GameTimer.Enabled:= False;
58 :   TUI.GetObject.GameEnd;
59 : end;
60 :
61 : procedure TLI.on_Tick;
62 : begin
63 :   if BlockShape.IsEmpty = True then BlockShape.CreateNext;
64 :   Self.Process_KeyDown(VK_Down);
65 : end;
66 :
67 : constructor TLI.Create;
68 : begin
69 :   inherited;
```

```
70 :
71 :   GameTimer:=  TGameTimer.Create;
72 :   BlockCell:=  TBlockCell.Create;
73 :   BlockShape:= TBlockShape.Create;
74 : end;
75 :
76 : destructor TLI.Destroy;
77 : begin
78 :   GameTimer.Free;
79 :   BlockCell.Free;
80 :   BlockShape.Free;
81 :
82 :   inherited;
83 : end;
84 :
85 : procedure TLI.Draw;
86 : begin
87 :   TUI.GetObject._fmMain.InitBoardCanvas;
88 :   BlockCell.Draw;
89 :   BlockShape.Draw;
90 : end;
91 :
92 : procedure TLI.Process_KeyDown(Key:Word);
93 : begin
94 :   BlockShape.Process_KeyDown(Key);
95 :   Self.Draw;
96 : end;
97 :
98 : end.
```

[소스 4] GameTimer Unit의 소스

```
 1 : unit GameTimer;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, ExtCtrls;
 7 :
 8 : type
 9 :   TGameTimer = class
10 :   private
11 :     FTimer : TTimer;
12 :     function GetEnabled: Boolean;
13 :     procedure SetEnabled(const Value: Boolean);
14 :   protected
15 :     procedure on_TimerFired(Sender:TObject);
16 :   public
17 :     constructor Create;
18 :     Destructor Destroy; override;
19 :   published
20 :     property Enabled : Boolean read GetEnabled write SetEnabled;
21 :   end;
22 :
23 : implementation
24 :
25 : uses
26 :   Logic_Interface;
27 :
28 : type
29 :   TFLI = class(TLI)
30 :   end;
31 :
32 : { TGameTimer }
33 :
```

```
34 : constructor TGameTimer.Create;
35 : begin
36 :   inherited;
37 :
38 :   FTimer:= TTimer.Create(Nil);
39 :   FTimer.Interval:= 50;
40 :   FTimer.OnTimer:= Self.on_TimerFired;
41 :   FTimer.Enabled:= False;
42 : end;
43 :
44 : destructor TGameTimer.Destroy;
45 : begin
46 :   FTimer.Free;
47 :
48 :   inherited;
49 : end;
50 :
51 : procedure TGameTimer.on_TimerFired(Sender: TObject);
52 : begin
53 :   TFLI(TFLI.GetObject).on_Tick;
54 : end;
55 :
56 : function TGameTimer.GetEnabled: Boolean;
57 : begin
58 :   Result:= FTimer.Enabled;
59 : end;
60 :
61 : procedure TGameTimer.SetEnabled(const Value: Boolean);
62 : begin
63 :   FTimer.Enabled:= Value;
64 : end;
65 :
66 : end.
```

[소스 5] BlockShape Unit의 소스

```
 1 : unit BlockShape;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Windows;
 7 :
 8 : type
 9 :   TBlockShape = class
10 :   public
11 :     procedure Draw;
12 :     procedure Process_KeyDown(Key:Word);
13 :     procedure CreateNext;
14 :   private
15 :     function GetIsEmpty: Boolean;
16 :     function do_MoveDown:Boolean;
17 :     procedure do_Drop;
18 :     procedure do_Move(Key:Word);
19 :   published
20 :     property IsEmpty : Boolean read GetIsEmpty;
21 :   end;
22 :
23 : implementation
24 :
25 : uses
26 :   Logic_Interface;
27 :
28 : procedure TBlockShape.Draw;
29 : begin
30 :   {Todo : }
31 : end;
32 :
33 : function TBlockShape.GetIsEmpty: Boolean;
```

```
34 : begin
35 :   {Todo : }
36 : end;
37 :
38 : procedure TBlockShape.Process_KeyDown(Key:Word);
39 : begin
40 :   if Key = VK_Down then do_MoveDown
41 :   else do_Move(Key);
42 : end;
43 :
44 : function TBlockShape.do_MoveDown:Boolean;
45 : begin
46 :   Result:= not TLI.GetObject.BlockCell.CheckCollision(VK_Down);
47 :
48 :   if Result = True then
49 :     {Todo : }
50 :   else
51 :     TLI.GetObject.BlockCell.BlockLanding;
52 : end;
53 :
54 : procedure TBlockShape.do_Drop;
55 : var
56 :   bMoved : Boolean;
57 : begin
58 :   Repeat
59 :     bMoved:= do_MoveDown;
60 :   until bMoved = False;
61 : end;
62 :
63 : procedure TBlockShape.do_Move(Key:Word);
64 : begin
65 :   if TLI.GetObject.BlockCell.CheckCollision(Key) = True then Exit;
66 :
67 :   case Key of
68 :       VK_Left :  {ToDo : };
69 :       VK_Right : {ToDo : };
```

```
70 :       VK_Up :    {ToDo : };
71 :       VK_Space : do_Drop;
72 :   end;
73 : end;
74 :
75 : procedure TBlockShape.CreateNext;
76 : begin
77 :    {Todo : }
78 : end;
79 :
80 : end.
```

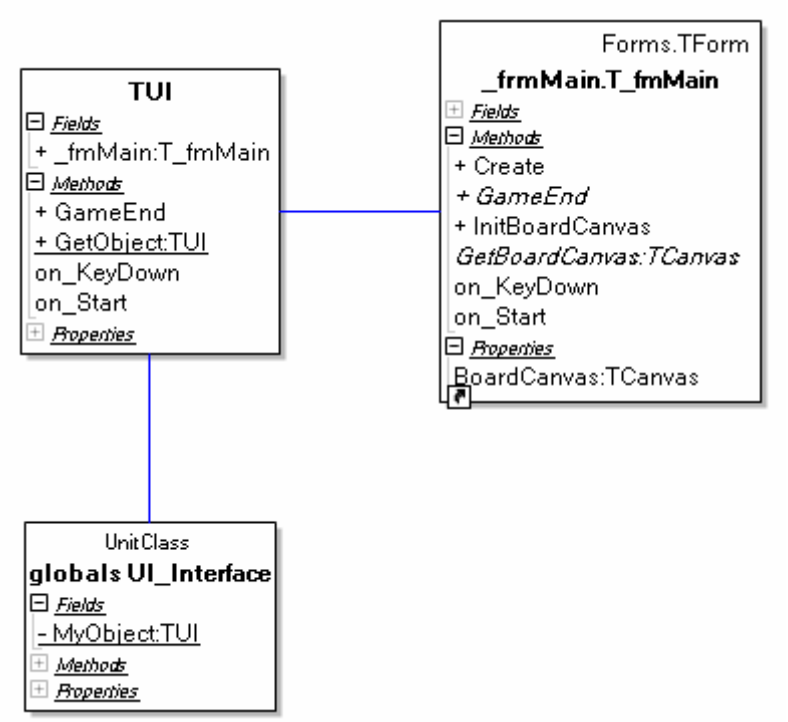
[그림 4]의 Job Flow와는 달리 좀더 로직을 쉽게 접근할 수 있도록 Private 메쏘드 몇 개가 추가되었다. Flow 자체는 동일하다.

[소스 6] TBlockCell Unit의 소스

```
 1 : unit BlockCell;
 2 :
 3 : interface
 4 :
 5 : type
 6 :   TBlockCell = class
 7 :   public
 8 :     procedure Draw;
 9 :     procedure Clear;
10 :     procedure BlockLanding;
11 :     function CheckCollision(Key:Word):Boolean;
12 :   end;
13 :
14 : implementation
15 :
16 : uses
17 :   Logic_Interface;
18 :
19 : procedure TBlockCell.Draw;
20 : begin
21 :   {Todo : }
22 : end;
23 :
24 : procedure TBlockCell.Clear;
25 : begin
26 :   {Todo : }
27 : end;
28 :
29 : procedure TBlockCell.BlockLanding;
30 : begin
31 :   {Todo : }
32 :
33 :   {Todo : 종료조건 처리}
```

```
34 : end;
35 :
36 : function TBlockCell.CheckCollision(Key:Word):Boolean;
37 : begin
38 :   {Todo : }
39 : end;
40 :
41 : end.
```

기능분석



[그림 5] User Interface 클래스 다이어그램

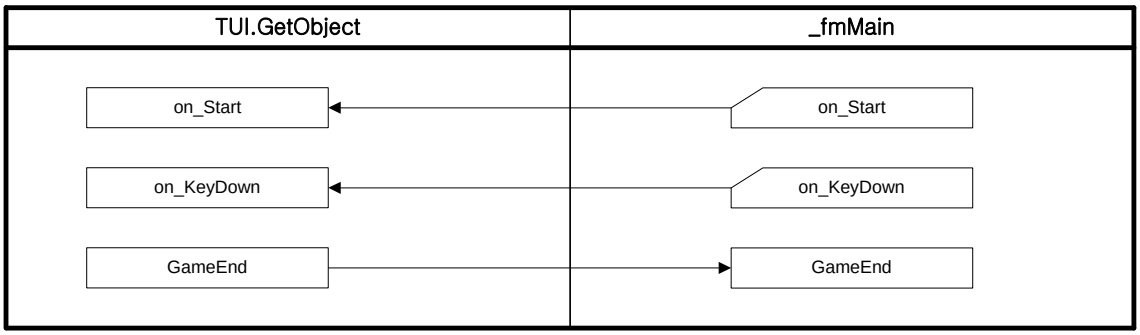
TUI 클래스는 변경된 사항이 없으니, _fmMain 클래스의 기능목록만 살펴보자.

- 기능
  - GameEnd
  - InitBoardCanvas : 실제로는 구현과정 중에 추가된 메쏘드이다.
- 이벤트
  - on_KeyDown
  - on_Start

_fmMain은 메인 폼에 대한 부모클래스이다. TUI는 직접 메인 폼에 대하여 의존하지 않도록 중간에 _fmMain 클래스를 생성하고, 메인 폼은 이것을 상속받아서 사용하도록 한다. 상속받아서 처리하는 이유는 TUI 자체의 논리계층과 구현계층을 분리하여 변경사항이 발생하더라도 변경사항에 대한 쇼크를 줄이기 위해서이다.

예를 들어 TLI에서 게임이 종료되었음을 알리면 TUI.GameEnd가 호출된다. 이때 TUI는 _fmMain.GameEnd라는 **abstract** 메쏘드를 실행하게 된다. 추후 메인 폼이 _fmMain에서 상속을 받아서 해당 메쏘드를 **override** 하여 게임 종료처리를 완료하게 되는 것이다.

메인 폼이 게임 종료처리를 어떠한 방식으로 진행하던지 이제 TUI는 알 필요가 없는 것이다. 그것은 TUI가 메인 폼이 아닌 _fmMain에만 의존관계를 가지고 있기 때문이다.



[그림 6] Job Flow

현재의 설계는 _fmMain에서 on_Start 등의 이벤트가 발생하면 TUI의 같은 on_Start가 호출되어 중복된 메쏘드가 나오게 된다. 이것은 Demeter 법칙을 준수하기 위한 것이다. TLI가 TUI이 너무 깊숙한 곳까지 알게 되면 소스코드의 결합도가 높아지기 때문이다. 하지만, 필자는 TUI가 너무 많은 메쏘드를 처리하게 되는 시점에서 모든 메쏘드는 TUI 하위 클래스에게 위임한다. 하나의 클래스에 너무 많은 메쏘드를 포함시키면 가독성이 떨어지기 때문이다.

일반적으로는 다른 클래스의 내장을 후벼 파는 것은 소스의 유연성을 떨어트리게 된다. 따라서, 우선적으로 Demeter 법칙을 준수하다가 어느 일정 수준이 넘어서 가독성이 문제가 된다고 판단이 될 때, 필자는 메쏘드를 하위 클래스에서만 정의하는 방식을 채택했다.

(객체지향적)설계에서는 때에 따라서 그 적용방법을 달리해야 할 때가 발생한다. “반드시”라는 법칙은 없다. (어쩌면 그것이 다행일지도 모르겠다. 만약 “반드시”라는 법칙이 통한다면 사람이 아닌 기계가 대신 설계를 도맡아 할지도 모르겠다.)

[소스 7] UI_Interface Unit의 수정

```
 1 : unit UI_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, _frmMain;
 7 :
 8 : type
 9 :   TUI = class
10 :   protected
11 :     procedure on_KeyDown(Key:Word);
12 :     procedure on_Start;
13 :   public
14 :     _fmMain : T_fmMain;
15 :     class function GetObject:TUI;
16 :     procedure GameEnd;
17 :   end;
18 :
19 : implementation
20 :
21 : uses
22 :   Logic_Interface;
23 :
24 : var
25 :   MyObject : TUI = Nil;
26 :
27 : { TUI }
28 :
29 : class function TUI.GetObject: TUI;
30 : begin
31 :   if MyObject = Nil then MyObject:= TUI.Create;
32 :   Result:= MyObject;
33 : end;
```

```
34 :
35 : procedure TUI.on_Start;
36 : begin
37 :   TLI.GetObject.StartGame;
38 : end;
39 :
40 : procedure TUI.on_KeyDown(Key:Word);
41 : begin
42 :   if TLI.GetObject.GameStarted = True then TLI.GetObject.Process_KeyDown(Key);
43 : end;
44 :
45 : procedure TUI.GameEnd;
46 : begin
47 :   Self._fmMain.GameEnd;
48 : end;
49 :
50 : end.
```

[소스 8] _fmMain Unit의 소스

```
 1 : unit _frmMain;
 2 :
 3 : interface
 4 : uses
 5 :   Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms;
 6 :
 7 : type
 8 :   T_fmMain = class(TForm)
 9 :   private
10 :   protected
11 :     function GetBoardCanvas: TCanvas; virtual; abstract;
12 :     procedure on_KeyDown(Key:Word);
13 :     procedure on_Start;
14 :   public
15 :     constructor Create(AOwner:TComponent); override;
16 :     procedure InitBoardCanvas;
17 :     procedure GameEnd; virtual; abstract;
18 :   published
19 :     property BoardCanvas : TCanvas read GetBoardCanvas;
20 :   end;
21 :
22 : implementation
23 :
24 : uses
25 :   UI_Interface;
26 :
27 : type
28 :   TFUI = class(TUI)
29 :   end;
30 :
31 : { T_fmMain }
32 :
33 : constructor T_fmMain.Create(AOwner: TComponent);
```

```
34 : begin
35 :   inherited;
36 :
37 :   TUI.GetObject._fmMain:= Self;
38 : end;
39 :
40 : procedure T_fmMain.on_KeyDown(Key: Word);
41 : begin
42 :   TFUI(TUI.GetObject).on_KeyDown(Key);
43 : end;
44 :
45 : procedure T_fmMain.on_Start;
46 : begin
47 :   TFUI(TUI.GetObject).on_Start;
48 : end;
49 :
50 : procedure T_fmMain.InitBoardCanvas;
51 : var
52 :   Loop: Integer;
53 : begin
54 :   BoardCanvas.Brush.Color:= $004F4F4F;
55 :   BoardCanvas.FillRect(Rect(0, 0, 120, 240));
56 :
57 :   // 배경에 수직선 그리기
58 :   for Loop := 1 to 10 do begin
59 :     BoardCanvas.Pen.Color:= clWhite;
60 :     BoardCanvas.MoveTo((Loop-1)*12,   0);
61 :     BoardCanvas.LineTo((Loop-1)*12, 240);
62 :   end;
63 :
64 : end;
65 :
66 : end.
```

중간점검

필자는 하나의 프로젝트(델파이에서는 *.dpr) 파일에 관련된 범위를 프로세스 모듈이라고 부른다. 그리고, 여기까지가 프로세스 모듈에 대한 논리계층의 설계과정이다. 이제부터는 구현계층에 대해서 설명하고자 한다.

만약 프로세스 모듈에 대한 논리계층이 변경되어야 한다면, 그것은 업그레이드가 아닌 버전업의 시점이라고 필자는 생각한다. 구현계층이 이루어져 있는 상황에서 논리계층을 변경하는 것은 건물을 거의 지어놓은 상태에서 철골구조를 새로 하려는 것과 같다. (실제 프로젝트에서는 그것보다 더 위험하다.)

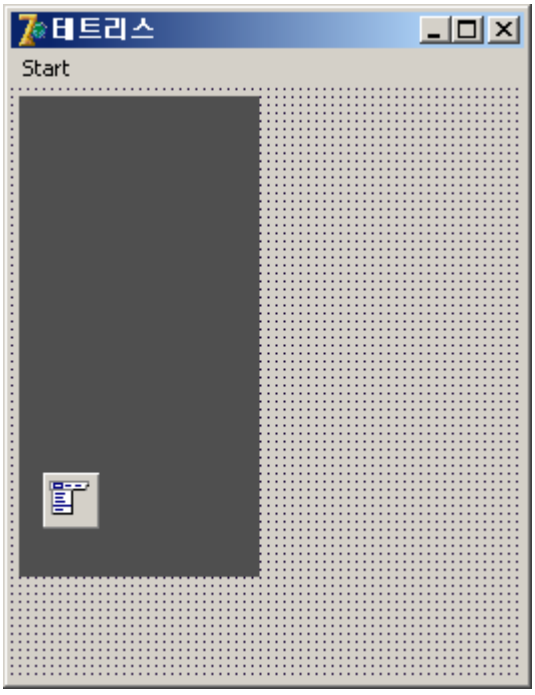
4단계 - Logic에 대한 구현계층 설계

필자는 구현계층에 대한 분석과 설계 과정은 다소 복잡한 부분을 제외하고는 대부분 생략하는 편이다.

User Interface 준비

사용자 인터페이스는 개발과정에서 최대한 뒤로 미루는 것이 좋다. 모든 것이 그러하듯이 우선 골격을 먼저 구성하고 살을 붙여가는 것이 실수와 비용을 줄이는 지름길이다.

소프트웨어 개발에서는 Logic이 바로 그 골격에 해당하므로, 이에 대한 설계와 테스트가 완료되고 난 뒤에 사용자 인터페이스를 완성해 나가는 것이 유리하다. 하지만, 테스트를 위해서 최소한의 사용자 인터페이스는 미리 분석과 설계를 거쳐서 작성하는 편이 좋다. 필자는 지금이 바로 그 시점이라고 생각한다.



[화면 2] 메인 폼 디자인

테스트를 위해서 [화면 2]처럼 콤포넌트를 내려놓고 프로퍼티를 아래와 같이 수정한다. TPanel의 경우에는 Canvas 속성이 Protected로 되어 있어 접근이 불가능하여 약간의 트릭을 이용하였다. 이점은 소스를 참고하기 바란다.

```
 1 : object fmMain: TfmMain
 2 :   Left = 0
 3 :   Top = 0
 4 :   Caption = 테트리스
 5 :   ClientHeight = 296
 6 :   ClientWidth = 240
 7 :   OnKeyDown = FormKeyDown
 8 :   object plGameBoard: TPanel
 9 :     Left = 4
10 :     Top = 4
11 :     Width = 120
12 :     Height = 240
13 :     BevelOuter = bvNone
14 :     Color = 5197647
15 :   end
16 :   object MainMenu: TMainMenu
17 :     object miStart: TMenuItem
18 :       Caption = 'Start'
19 :       OnClick = miStartClick
20 :     end
21 :   end
22 : end
```

[소스 9] frmMain Unit의 소스

```
 1 : unit frmMain;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 7 :   Dialogs, Menus, _frmMain, ExtCtrls, CanvasPanel;
 8 :
 9 : type
10 :   TfmMain = class(T_fmMain)
11 :     MainMenu: TMainMenu;
12 :     miStart: TMenuItem;
13 :     plGameBoard: TPanel;
14 :     procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
15 :     procedure miStartClick(Sender: TObject);
16 :   private
17 :     { Private declarations }
18 :     function GetBoardCanvas: TCanvas; override;
19 :   public
20 :     { Public declarations }
21 :     procedure GameEnd; override;
22 :   end;
23 :
24 : var
25 :   fmMain: TfmMain;
26 :
27 : implementation
28 :
29 : {$R *.dfm}
30 :
31 : procedure TfmMain.FormKeyDown(Sender: TObject; var Key: Word;
32 :   Shift: TShiftState);
33 : begin
```

```
34 :   Self.on_KeyDown(Key);
35 : end;
36 :
37 : procedure TfmMain.GameEnd;
38 : begin
39 :   ShowMessage('Game Over!!!');
40 : end;
41 :
42 : function TfmMain.GetBoardCanvas: TCanvas;
43 : begin
44 :   Result:= plGameBoard.Canvas;
45 : end;
46 :
47 : procedure TfmMain.miStartClick(Sender: TObject);
48 : begin
49 :   Self.on_Start;
50 : end;
51 :
52 : end.
```

테스트 시나리오 작성

테스트를 위해서 사용자가 접근할 수 있는 모든 가능성에 대한 시나리오를 찾아내고 이를 하나씩 수행하여 본다. 본 강좌에서는 사용자가 게임을 시작하기 위해서 Start 메뉴를 실행하는 초기 단계부터 코딩과 테스트를 병행하면서 설명을 하도록 하겠다.

시나리오 Step 1 - 사용자가 Start 메뉴를 실행하였음

사용자가 Start 메뉴를 실행하면 우선 [소스 9]의 49: 라인이 실행된다. 이것은 "TLI.GetObject.GameTimer.Enabled:= True"를 실행하게 되고, 결과적으로 TLI.on_Tick 이벤트가 주기적으로 실행된다.

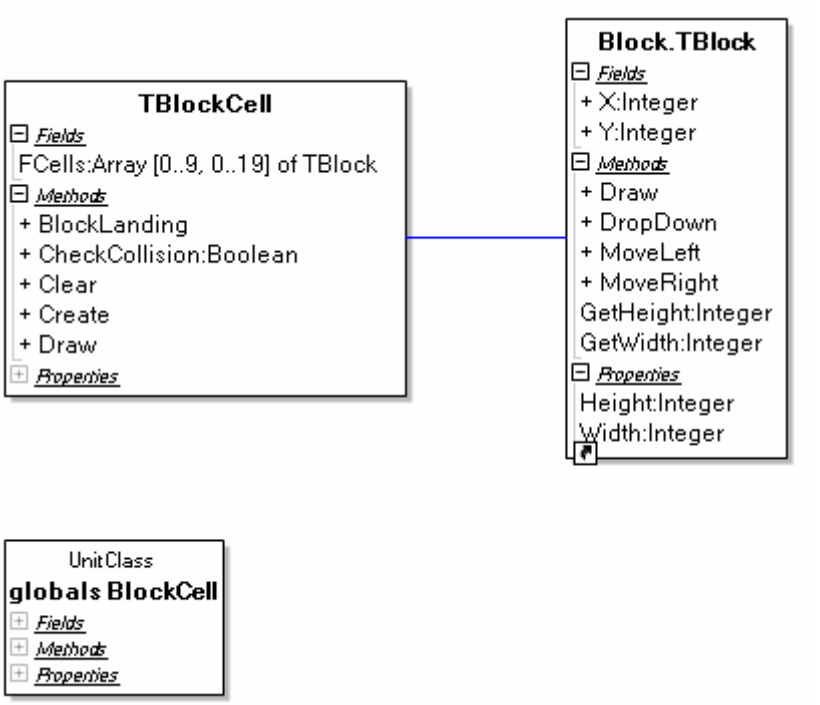
그리고 해당 이벤트가 실행될 때 마다, TLI.Draw 메쏘드가 호출된다. 아래는 해당 메쏘드의 변경내용이다.

```
1 : procedure TLI.Draw;
2 : begin
3 :   TUI.GetObject._fmMain.InitBoardCanvas;
4 :   BlockCell.Draw;
5 :   BlockShape.Draw;
6 : end;
```

3: 라인에서 게임보드를 초기화하는 소스는 이미 작성되어 있으니, 4: 라인의 BLockCel.Draw에 대해서 코딩을 하겠다.

TBlock 클래스 추가

[화면 1]에서 보았듯이 테트리스에는 최소단위의 블록을 관리하는 무엇인가가 필요하다. 필자는 TBlock 클래스를 추가하여 이를 해결하도록 할 것이다.



[그림 7] TBlockCell의 클래스 다이어그램

[그림 7]에서 보듯이 TBlockCell은 TBlock 클래스를 배열로 관리하고 있다. 배열의 사이즈는 10X20으로 현재 작성하려고 하는 테트리스의 셀 크기이다.

TBlock의 기능목록은 아래와 같다.

- 기능
  - Draw : 자기 자신을 그린다.
  - DropDown : 바닥으로 한 칸씩 떨어진다.
- 이벤트
  - 없음

TBlock은 단위 사이즈를 나타내기 위해 Read-Only 속성 Width와 Height를 가지고 있다.

[소스 10] Block Unit의 소스

```
 1 : unit Block;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Graphics;
 7 :
 8 : type
 9 :   TBlock = class
10 :   private
11 :     function GetHeight: Integer;
12 :     function GetWidth: Integer;
13 :   public
14 :     X, Y : Integer;
15 :     procedure Draw;
16 :     procedure DropDown;
17 :     procedure MoveLeft;
18 :     procedure MoveRight;
19 :   published
20 :     property Width : Integer read GetWidth;
21 :     property Height : Integer read GetHeight;
22 :   end;
23 :
24 : implementation
25 :
26 : uses
27 :   UI_Interface;
28 :
29 : { TBlock }
30 :
31 : procedure TBlock.Draw;
32 : begin
33 :   TUI.GetObject._fmMain.BoardCanvas.Brush.Color:= clLime;
```

```
  34 :   TUI.GetObject._fmMain.BoardCanvas.FillRect(Rect(X*Width, Y*Height, (X+1)*Width,
(Y+1)  35 : *Height));
  36 :   TUI.GetObject._fmMain.BoardCanvas.Pen.Color:= clRed;
  37 :   TUI.GetObject._fmMain.BoardCanvas.Rectangle(Rect(X*Width, Y*Height,
(X+1)*Width, (Y+1)  38 : *Height));
  39 : end;
  40 :
  41 : procedure TBlock.DropDown;
  42 : begin
  43 :   Y:= Y + 1;
  44 : end;
  45 :
  46 : function TBlock.GetHeight: Integer;
  47 : begin
  48 :   Result:= 12;
  49 : end;
  50 :
  51 : function TBlock.GetWidth: Integer;
  52 : begin
  53 :   Result:= 12;
  54 : end;
  55 :
  56 : procedure TBlock.MoveLeft;
  57 : begin
  58 :   X:= X -1;
  59 : end;
  60 :
  61 : procedure TBlock.MoveRight;
  62 : begin
  63 :   X:= X + 1;
  64 : end;
  65 :
end.
```

위의 소스는 메인 폼에 버턴을 올려두고 아래와 같은 소스로 제대로 블록이 그려지는 지 테스트를 진행해보았다. 강좌의 특성으로 인해 자잘한 과정을 생략하고 넘어갈 수 밖에 없

음이 아쉽다.

```
 1 : implementation
 2 :
 3 : uses
 4 :   Block;
 5 :
 6 : {$R *.dfm}
 7 :
 8 : procedure TfmMain.Button1Click(Sender: TObject);
 9 : var
10 :   Block : TBlock;
11 : begin
12 :   Block:= TBlock.Create;
13 :   Block.Draw;
14 : end;
```

[소스 11] BlockCell Unit의 수정

```
 1 : unit BlockCell;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Graphics, Block;
 7 :
 8 : type
 9 :   TBlockCell = class
10 :   private
11 :     FCells : Array [0..9, 0..19] of TBlock;
12 :   public
13 :     constructor Create;
14 :     procedure Draw;
15 :     procedure Clear;
16 :     procedure BlockLanding;
17 :     function CheckCollision(Key:Word):Boolean;
18 :   end;
19 :
20 : implementation
21 :
22 : uses
23 :   Logic_Interface;
24 :
25 : procedure TBlockCell.Draw;
26 : var
27 :   LoopX: Integer;
28 :   LoopY: Integer;
29 : begin
30 :   for LoopX := 0 to 10 - 1 do
31 :   for LoopY := 0 to 20 - 1 do
32 :     if FCells[LoopX, LoopY] <> Nil then FCells[LoopX, LoopY].Draw;
33 : end;
```

```
34 :
35 : procedure TBlockCell.Clear;
36 : var
37 :   LoopX: Integer;
38 :   LoopY: Integer;
39 : begin
40 :   for LoopX := 0 to 10 - 1 do
41 :   for LoopY := 0 to 20 - 1 do
42 :     if FCells[LoopX, LoopY] <> Nil then begin
43 :       FCells[LoopX, LoopY].Free;
44 :       FCells[LoopX, LoopY]:= Nil;
45 :     end;
46 : end;
47 :
48 : constructor TBlockCell.Create;
49 : var
50 :   LoopX: Integer;
51 :   LoopY: Integer;
52 : begin
53 :   inherited;
54 :
55 :   for LoopX := 0 to 10 - 1 do
56 :   for LoopY := 0 to 20 - 1 do
57 :     FCells[LoopX, LoopY]:= Nil;
58 : end;
59 :
60 : procedure TBlockCell.BlockLanding;
61 : begin
62 :   {Todo : }
63 :
64 :   {Todo : 종료조건 처리}
65 :   if false then TLI.GetObject.GameStarted:= False;
66 : end;
67 :
68 : function TBlockCell.CheckCollision(Key:Word):Boolean;
69 : begin
```

```
70 :   {Todo : }
71 : end;
72 :
73 : end.
```

[소스 12] BlockShape Unit의 수정

```
 1 : unit BlockShape;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Windows, Block;
 7 :
 8 : type
 9 :   TBlockShape = class
10 :   private
11 :     FBlockList : TList;
12 :     function GetIsEmpty: Boolean;
13 :     procedure do_Drop;
14 :     procedure do_Move(Key:Word);
15 :     function do_MoveDown:Boolean;
16 :     procedure do_MoveLeft;
17 :     procedure do_MoveRight;
18 :     procedure do_CreateBlockShape1;
19 :     procedure do_CreateBlockShape2;
20 :     procedure do_CreateBlockShape3;
21 :     procedure do_CreateBlockShape4;
22 :     procedure do_CreateBlockShape5;
23 :     procedure do_CreateBlockShape6;
24 :     procedure do_CreateBlockShape7;
25 :   public
26 :     constructor Create;
27 :     destructor Destroy; override;
28 :     procedure Draw;
29 :     procedure Process_KeyDown(Key:Word);
30 :     procedure CreateNext;
31 :   published
32 :     property IsEmpty : Boolean read GetIsEmpty;
33 :   end;
```

```
34 :
35 : implementation
36 :
37 : uses
38 :   Logic_Interface;
39 :
40 : procedure TBlockShape.Draw;
41 : var
42 :   Loop: Integer;
43 : begin
44 :   for Loop := 0 to FBlockList.Count - 1 do
45 :     TBlock(FBlockList.Items[Loop]).Draw;
46 : end;
47 :
48 : function TBlockShape.GetIsEmpty: Boolean;
49 : begin
50 :   Result:= FBlockList.Count = 0;
51 : end;
52 :
53 : procedure TBlockShape.Process_KeyDown(Key:Word);
54 : begin
55 :   if Key = VK_Down then do_MoveDown
56 :   else do_Move(Key);
57 : end;
58 :
59 : function TBlockShape.do_MoveDown:Boolean;
60 : var
61 :   Loop: Integer;
62 : begin
63 :   Result:= not TLI.GetObject.BlockCell.CheckCollision(VK_Down);
64 :
65 :   if Result = True then begin
66 :     for Loop := 0 to FBlockList.Count - 1 do
67 :       TBlock(FBlockList.Items[Loop]).DropDown;
68 :   end else
69 :     TLI.GetObject.BlockCell.BlockLanding;
```

```
 70 : end;
 71 :
 72 : procedure TBlockShape.do_MoveLeft;
 73 : var
 74 :   Loop: Integer;
 75 : begin
 76 :   for Loop := 0 to FBlockList.Count - 1 do
 77 :     TBlock(FBlockList.Items[Loop]).MoveLeft;
 78 : end;
 79 :
 80 : procedure TBlockShape.do_MoveRight;
 81 : var
 82 :   Loop: Integer;
 83 : begin
 84 :   for Loop := 0 to FBlockList.Count - 1 do
 85 :     TBlock(FBlockList.Items[Loop]).MoveRight;
 86 : end;
 87 :
 88 : destructor TBlockShape.Destroy;
 89 : begin
 90 :   FBlockList.Free;
 91 :
 92 :   inherited;
 93 : end;
 94 :
 95 : procedure TBlockShape.do_CreateBlockShape1;
 96 : var
 97 :   Block : TBlock;
 98 : begin
 99 :   Block:= TBlock.Create;
100 :   Block.X:= 4;
101 :   Block.Y:= -2;
102 :   FBlockList.Add(Block);
103 :
104 :   Block:= TBlock.Create;
105 :   Block.X:= 5;
```

```
106 :   Block.Y:= -2;
107 :   FBlockList.Add(Block);
108 :
109 :   Block:= TBlock.Create;
110 :   Block.X:= 4;
111 :   Block.Y:= -1;
112 :   FBlockList.Add(Block);
113 :
114 :   Block:= TBlock.Create;
115 :   Block.X:= 5;
116 :   Block.Y:= -1;
117 :   FBlockList.Add(Block);
118 : end;
119 :
120 : procedure TBlockShape.do_CreateBlockShape2;
121 : var
122 :   Block : TBlock;
123 : begin
124 :   Block:= TBlock.Create;
125 :   Block.X:= 4;
126 :   Block.Y:= -3;
127 :   FBlockList.Add(Block);
128 :
129 :   Block:= TBlock.Create;
130 :   Block.X:= 4;
131 :   Block.Y:= -2;
132 :   FBlockList.Add(Block);
133 :
134 :   Block:= TBlock.Create;
135 :   Block.X:= 5;
136 :   Block.Y:= -2;
137 :   FBlockList.Add(Block);
138 :
139 :   Block:= TBlock.Create;
140 :   Block.X:= 5;
141 :   Block.Y:= -1;
```

```
142 :   FBlockList.Add(Block);
143 : end;
144 :
145 : procedure TBlockShape.do_CreateBlockShape3;
146 : var
147 :   Block : TBlock;
148 : begin
149 :   Block:= TBlock.Create;
150 :   Block.X:= 5;
151 :   Block.Y:= -3;
152 :   FBlockList.Add(Block);
153 :
154 :   Block:= TBlock.Create;
155 :   Block.X:= 4;
156 :   Block.Y:= -2;
157 :   FBlockList.Add(Block);
158 :
159 :   Block:= TBlock.Create;
160 :   Block.X:= 5;
161 :   Block.Y:= -2;
162 :   FBlockList.Add(Block);
163 :
164 :   Block:= TBlock.Create;
165 :   Block.X:= 4;
166 :   Block.Y:= -1;
167 :   FBlockList.Add(Block);
168 : end;
169 :
170 : procedure TBlockShape.do_CreateBlockShape4;
171 : var
172 :   Block : TBlock;
173 : begin
174 :   Block:= TBlock.Create;
175 :   Block.X:= 4;
176 :   Block.Y:= -3;
177 :   FBlockList.Add(Block);
```

```
178 :
179 :   Block:= TBlock.Create;
180 :   Block.X:= 4;
181 :   Block.Y:= -2;
182 :   FBlockList.Add(Block);
183 :
184 :   Block:= TBlock.Create;
185 :   Block.X:= 5;
186 :   Block.Y:= -2;
187 :   FBlockList.Add(Block);
188 :
189 :   Block:= TBlock.Create;
190 :   Block.X:= 4;
191 :   Block.Y:= -1;
192 :   FBlockList.Add(Block);
193 : end;
194 :
195 : procedure TBlockShape.do_CreateBlockShape5;
196 : var
197 :   Block : TBlock;
198 : begin
199 :   Block:= TBlock.Create;
200 :   Block.X:= 4;
201 :   Block.Y:= -4;
202 :   FBlockList.Add(Block);
203 :
204 :   Block:= TBlock.Create;
205 :   Block.X:= 4;
206 :   Block.Y:= -3;
207 :   FBlockList.Add(Block);
208 :
209 :   Block:= TBlock.Create;
210 :   Block.X:= 4;
211 :   Block.Y:= -2;
212 :   FBlockList.Add(Block);
213 :
```

```
214 :   Block:= TBlock.Create;
215 :   Block.X:= 4;
216 :   Block.Y:= -1;
217 :   FBlockList.Add(Block);
218 : end;
219 :
220 : procedure TBlockShape.do_CreateBlockShape6;
221 : var
222 :   Block : TBlock;
223 : begin
224 :   Block:= TBlock.Create;
225 :   Block.X:= 4;
226 :   Block.Y:= -3;
227 :   FBlockList.Add(Block);
228 :
229 :   Block:= TBlock.Create;
230 :   Block.X:= 4;
231 :   Block.Y:= -2;
232 :   FBlockList.Add(Block);
233 :
234 :   Block:= TBlock.Create;
235 :   Block.X:= 4;
236 :   Block.Y:= -1;
237 :   FBlockList.Add(Block);
238 :
239 :   Block:= TBlock.Create;
240 :   Block.X:= 5;
241 :   Block.Y:= -1;
242 :   FBlockList.Add(Block);
243 : end;
244 :
245 : procedure TBlockShape.do_CreateBlockShape7;
246 : var
247 :   Block : TBlock;
248 : begin
249 :   Block:= TBlock.Create;
```

```
250 :   Block.X:= 5;
251 :   Block.Y:= -3;
252 :   FBlockList.Add(Block);
253 :
254 :   Block:= TBlock.Create;
255 :   Block.X:= 5;
256 :   Block.Y:= -2;
257 :   FBlockList.Add(Block);
258 :
259 :   Block:= TBlock.Create;
260 :   Block.X:= 5;
261 :   Block.Y:= -1;
262 :   FBlockList.Add(Block);
263 :
264 :   Block:= TBlock.Create;
265 :   Block.X:= 4;
266 :   Block.Y:= -1;
267 :   FBlockList.Add(Block);
268 : end;
269 :
270 : procedure TBlockShape.do_Drop;
271 : var
272 :   bMoved : Boolean;
273 : begin
274 :   Repeat
275 :     bMoved:= do_MoveDown;
276 :   until bMoved = False;
277 : end;
278 :
279 : procedure TBlockShape.do_Move(Key:Word);
280 : begin
281 :   if TLI.GetObject.BlockCell.CheckCollision(Key) = True then Exit;
282 :
283 :   case Key of
284 :       VK_Left :  do_MoveLeft;
285 :       VK_Right : do_MoveRight;
```

```
286 :       VK_Up :   {ToDo : };
287 :       VK_Space : do_Drop;
288 :   end;
289 : end;
290 :
291 : constructor TBlockShape.Create;
292 : begin
293 :   inherited;
294 :
295 :   Randomize;
296 :
297 :   FBlockList:= TList.Create;
298 : end;
299 :
300 : procedure TBlockShape.CreateNext;
301 : begin
302 :   FBlockList.Clear;
303 :
304 :   case Round(Random(7)) of
305 :       0 : do_CreateBlockShape1;
306 :       1 : do_CreateBlockShape2;
307 :       2 : do_CreateBlockShape3;
308 :       3 : do_CreateBlockShape4;
309 :       4 : do_CreateBlockShape5;
310 :       5 : do_CreateBlockShape6;
311 :       6 : do_CreateBlockShape7;
312 :   end;
313 : end;
314 :
315 : end.
```

블록 쌓기

이번에는 블록이 바닥에 떨어지면 쌓이도록 소스를 변경하도록 하겠다. 바닥이나 이미 쌓여있는 블록에 닿으면 멈추기 위해서는 우선 충돌테스트를 위한 메쏘드를 완성해야 한다.

해당 메쏘드는 블록을 현재 진행방향으로 한 칸 움직여 봐서 바닥이나 벽 또는 다른 블록에 부디치는 지를 검사하여 결과값을 알려주게 된다. 지금은 밑으로 떨어지는 방향이기 때문에 양쪽 벽에 부딪치는 것은 일단 무시하도록 하겠다.

우선 현재의 BlockShape를 진행방향으로 한 칸 움직이고 이것이 충돌하는 지를 검사하게 되면 화면에 검사하는 과정이 표시될 뿐 아니라, 현재의 위치정보를 모두 저장하고 다시 복원하는 등의 문제가 복잡해질 수 있다. 따라서, 필자는 현재의 BlockShape에 대한 클론을 만들고 클론을 통해서 충돌 테스트를 대신하도록 진행하기로 한다.

클론을 만들기 위해서 TBlockShape 클래스에 **private** 메소드를 하나 더 추가하도록 하겠다. 그리고, 클론의 경우에는 충돌 테스트 없이 이동하여야 하기 때문에 자신이 클론인지의 여부를 확인하는 플래그 FIsClone을 추가하도록 하겠다. 상속을 통해서 삭제할 수도 있으나, 우선은 플래그 변수를 이용하도록 하겠다.

```
1 : type
2 :   TBlockShape = class
3 :   private
4 :     FIsClone : Boolean;
```

```
1 : constructor TBlockShape.Create;
2 : begin
3 :   inherited;
4 :
5 :   FIsClone:= False;
```

```
1 : function TBlockShape.Clone: TBlockShape;
2 : var
3 :   Loop: Integer;
4 :   Block : TBlock;
5 : begin
6 :   Result:= TBlockShape.Create;
```

```
  7 :   Result.FIsClone:= True;
  8 :   for Loop := 0 to FBlockList.Count - 1 do begin
  9 :     Block:= TBlock.Create;
 10 :     Block.X:= TBlock(FBlockList.Items[Loop]).X;
 11 :     Block.Y:= TBlock(FBlockList.Items[Loop]).Y;
 12 :     Result.FBlockList.Add(Block);
 13 :   end;
 14 : end;
```

아래의 소스에서는 BlockShape를 밑으로 떨어트리는 메소드를 변경한 내용이다. BlockShape의 클론을 전달하기 위해서 CheckCollision 메쏘드에 파라메터를 추가하였다. 또한, do_CheckCollision 메쏘드를 추가하여 클론을 통한 충돌 테스트 루틴을 이용하는 것을 간략화하였다.

```
  1 : function TBlockShape.do_CheckCollision(Key:Word):Boolean;
  2 : var
  3 :   CloneShape : TBlockShape;
  4 : begin
  5 :   if FIsClone = False then begin
  6 :     CloneShape:= Self.Clone;
  7 :     try
  8 :       CloneShape.Process_KeyDown(Key);
  9 :       Result:= TLI.GetObject.BlockCell.CheckCollision(CloneShape.FBlockList,
Key);
 10 :     finally
 11 :       CloneShape.Free;
 12 :     end;
 13 :   end else
 14 :     Result:= False;
 15 : end;
 16 :
 17 : function TBlockShape.do_MoveDown:Boolean;
 18 : var
 19 :   Loop: Integer;
 20 : begin
 21 :   Result:= not Self.do_CheckCollision(VK_Down);
```

```
22 :
23 :   if Result = True then begin
24 :     for Loop := 0 to FBlockList.Count - 1 do
25 :       TBlock(FBlockList.Items[Loop]).DropDown;
26 :   end else
27 :     TLI.GetObject.BlockCell.BlockLanding;
28 : end;
```

또한, 내부에 TBlock으로 생성된 객체가 미아가 되면서 메모리 누수가 발생할 수가 있기 때문에, TBlockShape의 소멸자에서 TBlock의 객체들을 메모리에서 삭제하도록 소스를 추가하였다.

```
 1 : destructor TBlockShape.Destroy;
 2 : var
 3 :   Loop : Integer;
 4 : begin
 5 :   for Loop := 0 to FBlockList.Count - 1 do
 6 :     TBlock(FBlockList.Items[Loop]).Free;
 7 :
 8 :   FBlockList.Free;
 9 :
10 :   inherited;
11 : end;
```

```
1 : procedure TBlockShape.do_Move(Key:Word);
2 : begin
3 :   if Self.do_CheckCollision(Key) = True then Exit;
```

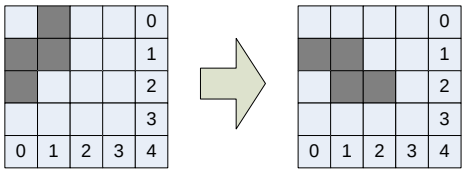
다음으로는 충돌 여부를 점검하는 TBlockCell.CheckCollision 메쏘드의 소스를 아래와 같이 작성한다. 이미 놓여진 블록과의 충돌을 검사하는 do_CheckBlocksInCells 메쏘드가 추가되었다.

```
1 : function TBlockCell.do_CheckBlocksInCells(List: TList): Boolean;
2 : var
3 :   Loop: Integer;
4 :   Block : TBlock;
```

```
 5 : begin
 6 :   Result:= False;
 7 :   for Loop := 0 to List.Count - 1 do begin
 8 :     Block:= TBlock(List.Items[Loop]);
 9 :     if (Block.X in [0..9]) and (Block.Y in [0..19]) then
10 :     if FCells[Block.X, Block.Y] <> Nil then begin
11 :       Result:= True;
12 :       Break;
13 :     end;
14 :   end;
15 : end;
16 :
17 : function TBlockCell.CheckCollision(List:TList; Key:Word):Boolean;
18 : var
19 :   Loop: Integer;
20 :   Block  : TBlock;
21 :   bCondition1, bCondition2, bCondition3 : Boolean;
22 : begin
23 :   Result:= False;
24 :
25 :   for Loop := 0 to List.Count - 1 do begin
26 :     Block:= Pointer(List.Items[Loop]);
27 :
28 :     bCondition1:= (Block.X < 0) or (Block.X > 9);
29 :     bCondition2:= (Block.Y > 19);
30 :     bCondition3:= do_CheckBlocksInCells(List);
31 :     if bCondition1 or bCondition2 or bCondition3 then begin
32 :       Result:= True;
33 :       Break;
34 :     end;
35 :   end;
36 : end;
```

이제는 바닥에 충돌된 이후 블록을 쌓이도록 하는 TBlockCell.BlockLanding 메쏘드를 작성하도록 하겠다.

```
 1 : implementation
 2 :
 3 : uses
 4 :   Logic_Interface;
 5 :
 6 : type
 7 :   TFLI = class(TLI)
 8 :   end;
                                                    …
45 : procedure TBlockCell.BlockLanding(List:TList);
46 : var
47 :   Loop: Integer;
48 :   Block  : TBlock;
49 :   bGameEnded : Boolean;
50 : begin
51 :   bGameEnded:= False;
52 :   for Loop := 0 to List.Count - 1 do begin
53 :     Block:= Pointer(List.Items[Loop]);
54 :     if (Block.X in [0..9]) and (Block.Y in [0..19]) then
55 :       FCells[Block.X, Block.Y]:= Block
56 :     else
57 :       bGameEnded:= True;
58 :   end;
59 :   List.Clear;
60 :
61 :   if bGameEnded = True then TFLI(TLI.GetObject).on_GameEnd;
62 : end;
```

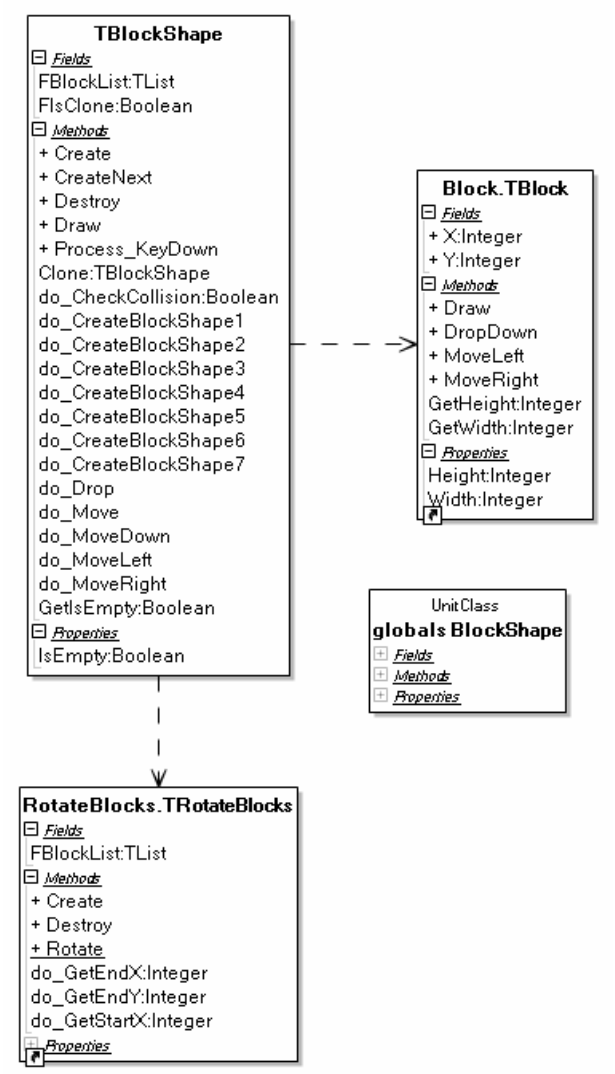


[그림 8] 블록들을 회전하고 난 뒤의 좌표변환

위의 그림을 참고로 해서 각 블록의 좌표(x,y)를 알고 있을 때, 회전 후의 좌표(x',y')를 구하는 식은 다음과 같다. (아래 식은 필자가 전철 안에서 노트를 통해서 구한 것이므로 최선의 방법이라고 할 수 없을 지도 모른다. 여러분들의 보다 기발한 아이디어를 기대하겠다.)

x' = StartX + (EndY - Y)
y' = EndY + (X – EndX)

여기서 (StartX, StartY)는 도형이 차지하고 있는 최소한의 사각형의 좌측 상단 모서리의 좌표이며, (EndX, EndY)는 해당 사각형의 우측 하단 모서리의 좌표이다.



[그림 9] 블록 회전을 위해서 TRotateBlocks 클래스를 추가한 다어이그램

아래 소스는 BlockShape 유닛에서 회전을 위해 변경된 부분이다.

```
1 : procedure TBlockShape.do_Move(Key:Word);
2 : begin
3 :   case Key of
4 :       VK_Left :  do_MoveLeft;
5 :       VK_Right : do_MoveRight;
6 :       VK_Down :  do_MoveDown;
```

```
 7 :       VK_Up :   TRotateBlocks.Rotate(FBlockList);
 8 :       VK_Space : do_Drop;
 9 :   end;
10 : end;
```

[소스 13] RotateBlocks Unit의 소스

```
 1 : unit RotateBlocks;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Block, Dialogs;
 7 :
 8 : type
 9 :   TRotateBlocks = class
10 :   private
11 :     FBlockList : TList;
12 :     function do_GetStartX:Integer;
13 :     function do_GetEndX:Integer;
14 :     function do_GetEndY:Integer;
15 :   public
16 :     constructor Create;
17 :     destructor Destroy; override;
18 :     class procedure Rotate(List:TList);
19 :   end;
20 :
21 : implementation
22 :
23 : { TRotateBlocks }
24 :
25 : constructor TRotateBlocks.Create;
26 : begin
27 :   inherited;
28 :
29 :   FBlockList:= TList.Create;
30 : end;
31 :
32 : destructor TRotateBlocks.Destroy;
33 : begin
```

```
34 :   FBlockList.Free;
35 :
36 :   inherited;
37 : end;
38 :
39 : function TRotateBlocks.do_GetEndX: Integer;
40 : var
41 :   Loop, X : Integer;
42 : begin
43 :   Result:= -100;
44 :
45 :   for Loop := 0 to FBlockList.Count - 1 do begin
46 :     X:= TBlock(FBlockList.Items[Loop]).X;
47 :     if X > Result then Result:= X;
48 :   end;
49 :
50 :   if Result < 0 then Result:= 0;
51 : end;
52 :
53 : function TRotateBlocks.do_GetEndY: Integer;
54 : var
55 :   Loop, Y : Integer;
56 : begin
57 :   Result:= -100;
58 :
59 :   for Loop := 0 to FBlockList.Count - 1 do begin
60 :     Y:= TBlock(FBlockList.Items[Loop]).Y;
61 :     if Y > Result then Result:= Y;
62 :   end;
63 :
64 :   if Result < 0 then Result:= 0;
65 : end;
66 :
67 : function TRotateBlocks.do_GetStartX: Integer;
68 : var
69 :   Loop, X : Integer;
```

```
 70 : begin
 71 :   Result:= 100;
 72 :
 73 :   for Loop := 0 to FBlockList.Count - 1 do begin
 74 :     X:= TBlock(FBlockList.Items[Loop]).X;
 75 :     if X < Result then Result:= X;
 76 :   end;
 77 :
 78 :   if Result > 9 then Result:= 9;
 79 : end;
 80 :
 81 : class procedure TRotateBlocks.Rotate(List: TList);
 82 : var
 83 :   Loop, X, Y, StartX, EndX, EndY : Integer;
 84 :   Block : TBlock;
 85 :   RotateBlocks : TRotateBlocks;
 86 : begin
 87 :   RotateBlocks:= TRotateBlocks.Create;
 88 :   try
 89 :     RotateBlocks.FBlockList.Assign(List);
 90 :
 91 :     StartX:= RotateBlocks.do_GetStartX;
 92 :     EndX:= RotateBlocks.do_GetEndX;
 93 :     EndY:= RotateBlocks.do_GetEndY;
 94 :
 95 :     for Loop := 0 to List.Count - 1 do begin
 96 :       Block:= Pointer(List.Items[Loop]);
 97 :       X:= Block.X;
 98 :       Y:= Block.Y;
 99 :       Block.X:= StartX + (EndY - Y);
100 :       Block.Y:= EndY   + (X - EndX);
101 :     end;
102 :   finally
103 :     RotateBlocks.Free;
104 :   end;
105 : end;
```

```
106 :
107 : end.
```

끝으로

블록이 짜맞춰졌을 때 해당 라인을 삭제하는 것은 숙제로 남겨두겠다.

(참고자료 http://cafe.naver.com/codeway/2410)