

# Membuat Game Dengan Pascal Yuk! (BAG I:TETRIS)

*Ahmad Priatama*

*ahmad.priatama@gmail.com*

## **Lisensi Dokumen:**

*Copyright © 2003-2006 IlmuKomputer.Com*

*Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.*

## **Pendahuluan**

Ketika anda bermain sebuah game, pernah ga sich terlintas sebuah pertanyaan dalam benak anda ‘bagaimana caranya membuat game ini?’ atau setidaknya ‘apa sih game itu?’. Mungkin sebagian dari pembaca pernah merasa penasaran seperti itu dan sebagian cuek saja dengan masalah tersebut. Untuk itulah saya ingin menunjukkan bagaimana sebuah game sederhana itu dibuat.

Yak, mengapa disebut sederhana karena kita akan menggunakan pascal dan menggantungkan interface pada unit crt. Dan seperti tetris yang biasa ada pada gamebot bentuknya masih dipertahankan seperti itu. Compiler akan lebih baik bila menggunakan free pascal, bila anda belum memilikinya dapat anda download di [freepascal.org](http://freepascal.org).

Sebenarnya tulisan ini merupakan remake dari karya tulis saya, tapi tak apa lah bila dipublikasikan sedikit.

## **Isi**

### **1 Struktur Data**

Tetris merupakan permainan yang menggunakan matriks sebagai struktur data utamanya. Penggunaan matriks lebih ke interpretasi area yang digunakan dalam permainan. Adapun matriks yang digunakan akan memiliki ordo yang cukup besar karena pada dasarnya tetris membutuhkan ruang yang besar agar dapat bekerja. Selain itu ditambah digunakan beberapa tipe data bentukan seperti *record / structure* yang akan memudahkan penulisan kode.

## 2 Uraian Pembuatan Tetris

Penjelasan pembuatan tetris yang diberikan tidak akan urut dari atas kebawah menelusuri kode sumber, namun mengikuti langkah – langkah yang harus dilakukan dalam pembuatan. Untuk kode sumber yang lengkap dapat anda lihat di bagian akhir dari bab ini.

Dalam pembuatan permainan komputer yang paling penting adalah interaksi antara pemain dengan permainan. Tanpa adanya interaksi antara pemain dan permainan, saya rasa permainan tidak akan berguna karena tidak dapat menghibur.

Interaksi yang dapat dilakukan dapat dibantu dengan menggunakan beberapa perangkat masukan maupun keluaran. Perangkat masukan berguna untuk mengirimkan intruksi dari pemain ke permainan yang akan berpengaruh pada kondisi permainan selanjutnya. Sedangkan perangkat keluaran berfungsi untuk menampilkan kondisi permainan yang nantinya akan mempengaruhi permainan dalam menentukan langkah atau intruksi yang akan diberikan selanjutnya.

Dalam pembuatan tetris ini, ternyata bahasa pascal menyediakan sebuah unit yang memiliki interface dasar untuk perangkat keluaran maupun perangkat masukan. Unit ini bernama *CRT* yang akan memegang kendali atas masukan permainan dan keluaran permainan dasar. Perangkat keluaran yang digunakan adalah monitor dengan mode teks atau yang lebih dikenal dengan sebutan '*under dos*'. Sedangkan perangkat masukan yang digunakan adalah papan ketik. Untuk itu kita perlu mendeklarasikan pemakaian unit ini di awal program.

```
Uses crt;
```

Setelah pendeklarasian unit tersebut kita perlu mempersiapkan area monitor yang akan digunakan. Perlu diketahui bahwa monitor dalam komputer menganut system kartesius dengan arah sumbu y yang terbalik dari sistem kartesius pada umumnya. Sumbu y ke arah atas pada sistem kartesius pada umumnya bernilai positif, berlainan pada monitor yang justru bernilai positif pada sumbu y yang mengarah kebawah. Teks mode standar memiliki jangkauan x antara 1 hingga 80 dan jangkauan untuk sumbu y antara 1 hingga 25.

Sebenarnya seluruh area yang monitor setelah deklarasi unit crt sudah dapat digunakan. Namun, kita memerlukan sebuah matriks yang berfungsi sebagai penyimpan keadaan. Misalkan pada beberapa saat yang lalu kita telah mengisi titik A(2,5), beberapa saat kemudian kita akan mengisi titik A tersebut, namun karena titik A tersebut telah terisi maka kita hanya dapat mengisi titik yang tepat berada di atas titik A. penyimpan keadaan ini akan sangat berguna dalam tetris untuk mencegah terjadinya tumpang tindih antar blok.

```
var field : array [1..80,1..25] of boolean;
```

Pendeklarasian matriks pada umumnya dimensi y terlebih dulu, namun untuk mempermudah pembelajaran kode sumber ini maka pendeklarasian mengikuti aturan pada *syntax gotoxy(x,y)*.

Pendeklarasian matriks mewakili semua area pada monitor, mulai dari kiri atas sampai kanan bawah. Penggunaan seluruh area monitor ini akan mempermudah kita dalam proses penggambaran keluaran, karena kita tidak hanya membuat sebuah area untuk seorang pemain, namun dua area untuk dua pemain dalam satu monitor. Dengan penggunaan matriks kondisi untuk seluruh monitor semua penggambaran tetris yang dilakukan hanya membutuhkan sebuah

prosedur saja.

Selain monitor masukan pun menggunakan sebuah papan ketik yang digunakan secara bersama. Sedangkan tipe data yang digunakan adalah Boolean yang akan mewakili titik di monitor tersebut telah terisi atau belum.

Setelah itu kita harus melakukan penggambaran batas area tiap pemain dan inisialisasi nilai matriks pada koordinat tertentu. Inisialisasi dilakukan untuk memudahkan batasan area para pemain sehingga pengaturan gerak blok – blok akan menjadi lebih mudah. Maksud dari gerak blok akan lebih mudah adalah pada saat kita melakukan penggeseran ataupun penurunan, batasan-batasan area sudah jelas ditetapkan melalui prosedur di bawah ini.

```
procedure init_field;
var i : longint;
begin
  clrscr; fillchar(field,sizeof(field),0);
  for i := 3 to 22 do
  begin
    gotoxy(20,i); write(#186); field[20,i] := true;
    gotoxy(36,i); write(#186); field[36,i] := true;
    gotoxy(60,i); write(#186); field[60,i] := true;
    gotoxy(76,i); write(#186); field[76,i] := true;
  end;
  for i := 1 to 15 do
  begin
    gotoxy(20+i,23); write(#205);
    field[19+i,23] := true;
    gotoxy(60+i,23); write(#205);
    field[59+i,23] := true;
  end;
  gotoxy(20,23); write(#200);
  gotoxy(60,23); write(#200);
  gotoxy(36,23); write(#188);
  gotoxy(76,23); write(#188);
  gotoxy(6,4); write('Next:');
  gotoxy(46,4); write('Next:');
end;
```

Kode sumber tersebut akan diletakan pada sebuah prosedur agar mempermudah pemanggilan berulang. *Syntax* `clrscr` akan membersihkan monitor sehingga dari tulisan sehingga keadaan monitor akan menjadi hitam secara keseluruhan. Kemudian, *syntax* `fillchar(...)` akan menginisialisasi keseluruhan isi matrix dengan nilai yang ditentukan. Dalam hal ini kita menginisialisasi matriks field dengan nilai `false`. Matriks field ketika kita deklarasikan sebenarnya seluruh elemennya sudah bernilai false, namun karena tetris kita ini bukan merupakan permainan yang dapat dimainkan berulang kali, maka kita perlu menginisialisasi

kembali nilai dari matriks field, karena nilai matriks sudah berubah oleh permainan sebelumnya. *Syntax* `gotoxy()` akan membawa kursor ke posisi yang ditentukan sehingga apapun yang dihasilkan oleh `write()` akan muncul di monitor pada posisi yang diinginkan. Sedangkan inialisasi dengan nilai `true` pada beberapa titik berarti titik tersebut telah terisi dan akan mencegah keluarnya blok dari daerah pemain.

Setiap iterasi dalam kode sumber penyusun diusahakan menggunakan nama variabel `i` agar pembaca tidak dibingungkan oleh penamaan variabel. Selain itu tipe data yang digunakan adalah `longint`, karena ketika sebuah program berjalan di atas sebuah prosesor bertipe 32 bit, iterasi akan lebih cepat pula bila menggunakan variabel 32 bit.

Setelah area permainan tercipta maka yang diperlukan adalah objek dari permainan itu sendiri yaitu tetris. Tetris, dari kata tetra yang berarti 4, merupakan gabungan dari 4 kotak kotak kecil dengan beberapa variasi bentuk. Dalam permainan nanti tetris akan menjadi satu – satunya objek yang bergerak. Tetris dapat bergeser maupun berputar sesuai dengan keinginan pemain.

Membuat tetris tidaklah sulit karena hanya membutuhkan sebuah array 1 dimensi yang memiliki property `x` dan `y`. Untuk lebih mudahnya kita akan menggabungkan property `x` dan `y` menjadi satu objek yaitu titik, yang kemudian akan menjadi elemen dari array berindex 4.

```
type _points = array [1..4] of record
    x, y : longint;
end;
```

Dengan kode diatas berarti kita telah menciptakan `points`. `Points` merupakan pendeklarasian properti dasar dari sebuah tetris. Sebuah tetris yang akan digunakan tidak hanya memiliki properti `points` saja namun juga dilengkapi dengan beberapa properti lain sehingga tetris tidak langsung dideklarasikan sebagai sebuah `points`, namun diperlukan pembungkusan pembungkusan properti – properti tersebut kedalam sebuah tipe baru. Properti yang lain akan dijelaskan kemudian.

Kita perlu menambahkan kode ini di bagian `type`:

```
type
    ...
    tetris = record
        points : _points;
        cshape, nshape, xmodel, xtetris, ymodel, ytetris: longint;
    end;
var
    ...
    tetris1, tetris2, ttetris : tetris;
```

Titik – titik di atas berarti sebagian kode telah dijelaskan sebelumnya atau akan dijelaskan kemudian.

Kita masih perlu mendeklarasikan beberapa variabel bertipe `tetris` untuk kedua pemain

bernama tetris1, tetris2 dan ttetris sebagai variabel temporer yang selalu digunakan untuk operasi-operasi penggambaran. Selanjutnya kita perlu mendeklarasikan isi dari tetris tersebut. Kita bisa saja memberikan nilai pada tetris secara langsung, namun hal ini menjadi tidak efektif karena akan memakan banyak tempat dan waktu penulisan kode. Sebagai gantinya kita dapat membuat template nilai awal dari tetris. Lebih baik lagi karena template ini nantinya akan menampung sekaligus seluruh variasi bentuk tetris. Template berupa konstanta array dengan tipe data bentukan points.

```
Const template : array [1..8] of _points =  
    ((x:1; y:1),(x:1; y:2),(x:2; y:1),(x:2; y:2)),  
    ((x:2; y:1),(x:1; y:1),(x:3; y:1),(x:4; y:1)),  
    ((x:1; y:2),(x:1; y:1),(x:1; y:3),(x:2; y:3)),  
    ((x:2; y:2),(x:2; y:1),(x:2; y:3),(x:1; y:3)),  
    ((x:1; y:2),(x:1; y:1),(x:2; y:2),(x:2; y:3)),  
    ((x:1; y:2),(x:2; y:1),(x:2; y:2),(x:1; y:3)),  
    ((x:1; y:2),(x:1; y:1),(x:1; y:3),(x:2; y:2)),  
    ((x:1; y:2),(x:2; y:1),(x:2; y:2),(x:2; y:3)));
```

Pendeklarasian template tetris memang cukup rumit. Hal ini dikarenakan oleh karena tetris merupakan array yang bertipe data bentukan. Pada umumnya pendeklarasian setiap element array dipisahkan oleh tanda kurung, ini berarti nilai yang berada dalam tanda kurung tersebut adalah nilai dari elemen array urut dari index 1 hingga 8 urut dari baris kedua hingga baris 9. Sedangkan pendeklarasian dari struktur seperti point membutuhkan nama properti diikuti ‘:’ dan nilai dari properti tersebut.

Template tersebut berisikan 8 variasi bentuk dari gabungan 4 buah blok. Mulai dari linear, t shape, rectangle dsb.

Pada saat program pertama dijalankan kita masih belum menginisialisasi nilai dari tiap – tiap tetris. Oleh karena itu kita perlu menambahkan kode untuk menginisialisasi tetris pada awal permainan.

```
procedure init_tetris(var objek : tetris; id : longint);  
begin  
    objek.nshape := random(8) + 1;  
    if id = 1 then  
    begin  
        objek.xmodel := 7; objek.xtetris := 26;  
    end else begin  
        objek.xmodel := 47; objek.xtetris := 66;  
    end;  
    objek.ymodel := 5; objek.ytetris := 2;  
    dropnew(objek);  
end;
```

Maksud dari menginisialisai nilai dari tetris di atas adalah memberikan nomor bentuk dari tetris yang akan diolah pada prosedur dropnew(dijelaskan kemudian). Kode di atas berfungsi untuk menentukan bentuk awal tetris karena pada awal permainan nilai nshape masih bernilai 0, maka diperlukan pemberian nilai sehingga seolah – olah di tengah – tengah permainan nanti tetris tersebut merupakan tetris yang telah ditentukan pada tetris sebelumnya. Properti xmodel dan ymodel adalah angka yang akan digunakan sebagai penentu posisi penggambaran tetris yang akan muncul ketika tetris sekarang mencapai bawah. Sedangkan properti xtetris dan ytetris digunakan sebagai penentu posisi penggambaran awal tetris yang sedang dimainkan.

Pemilihan bentuk tetris sejak awal permainan selalu diacak. Pengacakan didasarkan pada *syntax* random(). Penggunaan *syntax* random() seperti kode di atas akan selalu memberikan nilai dengan jangkauan antara 1 hingga 8. Nilai yang demikian sesuai dengan jumlah bentuk yang dideklarasikan pada template. Penggunaan *syntax* randomize akan mempengaruhi hasil dari *syntax* random() sehingga hasilnya tidak menimbulkan pola yang sama pada setiap permainan.

Untuk menentukan bentuk dari tetris kita bisa menginisialisasi isinya sesuai dengan yang diinginkan dengan *syntax* tetris1.points := template[index bentuk yang diinginkan];.

Setelah diinisialisasikan kita perlu melakukan penggambaran tetris pada layar monitor. Untuk itu dibutuhkan sebuah prosedur yang sekaligus berfungsi menggambar tetris dan menghapusnya.

```
Procedure drawtetris(objek : tetris; mode : Boolean);
var I : longint;
    c : char;
Begin
    If mode then c := #178 else c := ' ';
    For I := 1 to 4 do
        Begin
            Gotoxy(objek.tpoints[i].x, objek.tpoints[i].y);
            write(c);
            Field[objek.tpoints[i].x,objek.tpoints[i].y]:=mode;
        End;
    Gotoxy(1,1);
End;
```

Prosedur ini penyusun rasa sangat efektif karena dapat melakukan penggambaran di segala daerah pada monitor karena pendeklarasian matriks yang lebih universal untuk monitor. Parameter mode pada prosedur diatas akan berpengaruh pada perilaku yang dilakukan oleh prosedur. Apabila mode bernilai true maka akan dilakukan penggambaran tetris dilayar. Sebaliknya, bila bernilai false maka yang dilakukan oleh prosedur tersebut adalah penghapusan pada layar monitor. Selain dilakukan penggambaran pada layar monitor, perlu dilakukan perubahan kondisi pada matriks field, yakni perubahan apakah pada titik itu di layar monitor terisi atau tidak. Sedangkan *syntax* gotoxy() terakhir pada prosedur tersebut akan membawa kursor ke pojok kiri atas monitor sehingga tidak menjadi gangguan pada area yang digunakan

untuk bermain.

Tetris yang diberi nilai langsung dari template bila digambar di layar akan berada pada pojok kiri atas monitor. Untuk memposisikan tetris untuk pemain pertama dan kedua adalah dengan menambah komponen - komponen x dan y dari properti points dengan nilai tertentu. Contohnya kita perlu memposisikan tetris untuk pemain pertama di posisi A(28,3) maka kita perlu menambahkan nilai 27 pada tiap komponen x dan menambahkan nilai 2 pada tiap komponen y. Begitu halnya dengan tetris yang akan dimainkan selanjutnya, kita pun perlu menambahkan beberapa kode untuk memposisikannya. Kita dapat melakukannya dengan menambahkan kode berikut pada program:

```
procedure dropnew(var objek : tetris);
var i : longint;
begin
  if objek.cshape <> 0 then
  begin
    for i := 1 to 4 do
    begin
      ttetris.points[i].x:=template[objek.nshape,i].x+objek.xmodel;
      ttetris.points[i].y:=template[objek.nshape,i].y+objek.ymodel;
    end;
    drawtetris(ttetris,false);
  end;
  objek.cshape := objek.nshape;
  objek.nshape := random(8) + 1;
  for i := 1 to 4 do
  begin
    objek.points[i].x:=template[objek.cshape,i].x+objek.xtetris;
    objek.points[i].y:=template[objek.cshape,i].y+objek.ytetris;
    ttetris.points[i].x:=template[objek.nshape,i].x+objek.xmodel;
    ttetris.points[i].y:=template[objek.nshape,i].y+objek.ymodel;
    if field[objek.points[i].x,objek.points[i].y] then
    begin
      gameover := true;
      break;
    end;
  end;
  drawtetris(objek,true);
  drawtetris(ttetris,true);
end;
```

Potongan kode di atas awalnya akan melakukan penggambaran tetris selanjutnya, namun untuk mencegah terjadinya bug pada awal permainan dibutuhkan penyeleksian kondisi karena pada saat awal permainan properti c.shape masih bernilai 0, sehingga tetris selanjutnya

tidak akan memiliki bentuk. Karena bernilai 0 pula, bila kita paksakan pemanggilan nilai template berindex 0 akan menyebabkan *memory out of range*. ingat kita deklarasikan template mulai dari index 1.

Prosedur di atas bukan hanya berfungsi mengatur posisi dari tetris tetapi prosedur tersebut akan selalu dipanggil ketika sebuah tetris mencapai dasar dari area permainan. Prosedur ini juga menggambarkan tetris yang akan diturunkan pada bagian berikutnya pada layar tepat di samping kiri dari area permainan, sehingga mempermudah pemain dalam menentukan langkah yang akan diambilnya.

Karena pascal merupakan bahasa yang cenderung prosedural walaupun mendukung pemrograman berbasis objek, maka kita perlu membuat sebuah algoritma yang selalu mengulang – ulang perintahnya selama *runtime*. Kode berulang tersebut biasanya berfungsi menangani input dari pengguna.

Seperti kerja sebuah interpreter program akan menjalankan instruksi mulai dari atas program utama hingga akhir. Di mulai dengan kata begin dan diakhiri dengan kata end. Blok program utama ditandai dengan tanda titik di belakang kata end.

Program utama terdiri dari dua *loop*. *Loop* utama berfungsi untuk mengulang permainan bila permainan mencapai game over. Kemudian, loop kedua tetris ini merupakan runtime algoritma yang akan selalu menurunkan kedua tetris dan memeriksa input yang diberikan pengguna melalui papan ketik. Blok program utama diusahakan selalu memiliki jumlah baris yang sedikit, dengan demikian waktu yang digunakan oleh satu loop akan kecil, kecuali anda memanggil prosedur yang bersifat iteratif dan rekursif. Akan lebih optimal lagi bila kita menyatukan berbagai *syntax* dalam *loop* yang seminim mungkin.

begin

```
randomize;
while not quit do
begin
  init_field;
  init_tetris(tetris1,1);
  init_tetris(tetris2,2);
  gameover := false;
  while not gameover do
  begin
    slide(tetris1);
    slide(tetris2);
    userinput;
  end;
  gotoxy(1,25); write('Play Again[Y/N]? ');
  repeat
    answer := upcase(readkey);
  until answer in ['Y','N',#27];
  if (answer = 'N') or (answer = #27) then break;
```

```
end;  
end.
```

Kita akan membahas dari atas program urut ke bawah, mulai dari bawah kata begin sampai sebelum end. Beberapa prosedur yang telah dibahas akan dilewat begitu saja.

Bisa dikatakan prosedur slide merupakan prosedur mutlak dari program tetris karena tetris merupakan objek yang selalu turun ke bawah baru kemudian digantikan dengan yang baru apabila tetris tersebut telah bertumpu pada tetris lain di bawahnya atau tetris sudah sampai pada dasar dari area permainan. Slide dapat menurunkan tetris satu blok ke bawah setiap kali pemanggilannya. Slide bukanlah prosedur yang cukup sederhana. Slide merupakan penggabungan dari berbagai macam prosedur lain yang secara hierarki berada di bawahnya. Slide dasarnya hanya menurunkan tetris dari waktu ke waktu, baru setelah ada tumbukan dengan tetris di bawahnya ia memanggil prosedur lain dan begitu pula untuk aksi yang lain.

Tepat di bawah loop kedua ada *syntax* yang berfungsi untuk konfirmasi pengulangan permainan. Penggunaan *syntax readkey* untuk memeriksa input pada permainan tetris ini akan lebih dominan daripada *syntax read()* maupun *readln()*. Dominasi disebabkan karena readkey sama sekali tidak mengeluarkan *output* pada pemanggilannya, berbeda dengan kedua prosedur lainnya yang akan menyebabkan *output* pada layar monitor. Contohnya adalah ketika kita menggeser tetris menggunakan papan ketik, kita hanya membutuhkan masukan dari pengguna tanpa harus menuliskan pada monitor masukan apa yang telah dimasukan oleh pengguna sebab masukan itu sendiri telah diwakili oleh respon matriks.

```
procedure slide(var objek : tetris);  
var i : longint;  
    dropped : boolean;  
begin  
    drawtetris(objek,false);  
    ttetris := objek;  
    dropped := false;  
    for i := 1 to 4 do  
        begin  
            inc(ttetris.points[i].y);  
            if field[ttetris.points[i].x,ttetris.points[i].y] then  
                begin  
                    dropped := true;  
                    break;  
                end;  
        end;  
    end;  
    if not dropped then objek := ttetris;  
    drawtetris(objek,true);  
    if dropped then  
        begin  
            dropnew(objek);
```

```
        eliminate(objek,22);  
    end;  
end;
```

Dropnew adalah prosedur yang dipanggil ketika tetris telah mencapai dasar dari area permainan atau tetris sudah bertumbukan dengan tetris di bawahnya. Ditandai dengan penggunaan kata *var* sebelum nama variabel, dropnew memproses variabel objek bertipe tetris secara *by reference*. *By reference* berarti setiap kali dilakukan perubahan pada variabel objek akan berpengaruh pada variabel yang di-*passing* ke prosedur dropnew. *By value* merupakan metode lain yang digunakan pada *passing* parameter. Tanpa adanya *reserved word var* menandakan bahwa parameter akan di-*passing* secara *by value*. Perubahan variabel pada metode *by value* tidak akan mempengaruhi parameter yang di-*passing*.

Slide membutuhkan variabel boolean bernama *dropped* yang berfungsi sebagai representasi keadaan tetris sudah tertumpu pada tetris lain atau sudah mencapai dasar. Kondisi tertumpu atau terjatuh mempengaruhi nilai variabel *dropped* bernilai *true*, sebaliknya bila masih berada dalam keadaan bebas di area permainan akan bernilai *false*. Variabel ini akan menentukan akan dipanggil atau tidak prosedur bernama dropnew.

Karena prosedur ini memproses variabel parameter secara *by reference*, kita perlu melakukan proses pada sebuah variabel temporer, dalam hal ini adalah *tetris*. Penggunaan variabel temporer mencegah hal – hal yang tidak diinginkan, seperti pada saat tetris diturunkan padahal saat itu sudah mencapai dasar, keadaan ini apabila proses diteruskan sehingga menyebabkan perubahan pada tetris tentu tetris akan terlihat saling tumpang tindih.

Kembali ke bagian program utama. Kita sekarang mengamati *syntax* *userinput*. *Userinput* merupakan suatu prosedur yang berguna untuk menerima dan mengolah masukan yang diberikan oleh pemain melalui papan ketik. Prosedur ini hanya akan merespon masukan – masukan yang telah didefinisikan pada saat pembuatan karena pada saat penyeleksian kondisi tidak ada proses yang dilakukan untuk keadaan *default* atau dengan kata lain tidak ada definisi untuk keyword *else* pada bagian akhir seleksi oleh *case of*.

```
procedure userinput;  
var i : longint;  
    c : char;  
begin  
    for i := 1 to 20 do  
        begin  
            if keypressed then  
                begin  
                    c := upcase(readkey);  
                    case c of  
                        'w' : if tetris1.cshape <> 1 then rotate(tetris1);  
                        #72 : if tetris2.cshape <> 1 then rotate(tetris2);  
                        'A' : shift(tetris1,-1,0);  
                        #75 : shift(tetris2,-1,0);
```

```
'S' : shift(tetris1,0,1);  
#80 : shift(tetris2,0,1);  
'D' : shift(tetris1,1,0);  
#77 : shift(tetris2,1,0);  
#27 : gameover := true;  
#32 : repeat delay(10); until keypressed;  
end;  
end;  
delay(20);  
end;  
end;
```

Lagi – lagi digunakan sebuah variabel bertipe char untuk menampung masukan dari pemain melalui fungsi readkey. Nilai yang dikembalikan oleh fungsi readkey berupa tipe char atau bisa dikatakan sebuah nilai dengan jangkauan antara 0 hingga 255. Anda dapat saja mendeklarasikan nilai dari tipe char melalui angka. Deklarasi ini dapat dilakukan dengan dua cara. Cara pertama adalah dengan menggunakan fungsi chr(). Cara kedua adalah dengan menambahkan karakter '#' di depan angka yang dimaksud. Setiap angka akan mewakili karakter yang berbeda. Konvensi dalam penggunaannya berdasarkan pada penomoran ASCII, yaitu standar penomoran karakter yang dikenalkan oleh sebuah organisasi di Amerika.

Karena hasil dari readkey dapat berupa huruf, angka, atau simbol. Huruf yang dihasilkan olehnyapun dapat berupa huruf kapital atau huruf kecil. Ketika kita tidak ingin mempermasalahkan kapitalisasi dari sebuah masukan, kita dapat menggunakan sebuah fungsi bernama upcase(). Upcase() akan mengembalikan nilai kapital dari sebuah huruf apabila huruf tersebut merupakan huruf kecil. Untuk huruf kapital Upcase() tetap mengembalikan huruf kapital. Penggunaan upcase() akan menyebabkan masukan 'w' sama dengan 'W'.

Pada setiap sesi userinput akan dilakukan beberapa kali pengecekan oleh fungsi keypressed. Dengan pengecekan berulang akan membuat pemain dapat menggerakkan tetrisnya beberapa kali sebelum tetris tersebut turun oleh prosedur slide(). Selain itu digunakan pula *syntax* delay() untuk menambahkan selang waktu antara pengecekan yang satu dengan yang lain. Selang waktu menyebabkan penggunaan prosesor tidak terlalu tinggi dan memberikan sedikit waktu bagi pemain untuk memikirkan langkah berikutnya.

Masukan berupa karakter ASCII 27 atau biasa dikenal dengan *escape* akan menyebabkan variabel gameover bernilai true sehingga pada loop kedua pada program utama akan berhenti ketika compiler mencapai batas bawah dari while do. Dengan masukan *escape* ini pemain dapat keluar ketika sedang berada di tengah – tengah permainan. Sedangkan masukan berupa karakter ASCII 32 atau biasa dikenal dengan *space* menyebabkan terjadinya sebuah looping yang dilengkapi dengan delay() yang akan berhenti ketika pemain menekan tombol selain karakter spasi. Dengan adanya looping ini maka permainan seolah berada pada kondisi diam atau *pause* padahal sebenarnya kompilernya sedang sibuk melakukan sebuah looping. Tidak terpikir oleh penyusun bagaimana menimbulkan efek *pause* selain dengan cara selain looping, mungkin pembaca memiliki ide lain yang lebih baik daripada metode yang penyusun pikirkan.

Masukan untuk menggerakkan tetris bagi pemain pertama hanya dapat diberikan dengan menekan tombol A, S, D, dan W. A akan menyebabkan tetris bergerak ke kiri. S akan menyebabkan tetris turun. D akan menyebabkan tetris bergerak ke kanan. Sedangkan W akan

memutar tetris 90 derajat sesuai dengan arah jarum jam. Masukan untuk menggerakkan tetris bagi pemain kedua hanya dapat diberikan dengan menekan tombol panah kiri, bawah, kanan, atas. Panah kiri akan menyebabkan tetris bergerak ke kiri. Panah bawah akan menyebabkan tetris turun. Panah kanan akan menyebabkan tetris bergerak ke kanan. Sedangkan panah atas akan memutar tetris 90 derajat sesuai dengan arah jarum jam.

Masukan yang diberikan akan diseleksi oleh case of sehingga masukan yang berbeda akan menyebabkan pemanggilan prosedur yang berbeda. Masukan W atau panah atas akan memanggil prosedur rotate(). Sedangkan A, S, D, panah kanan, panah bawah, panah kiri akan memanggil prosedur shift(). Untuk masukan pemain pertama, tetris yang *dipassing* sebagai parameter adalah tetris1, sedangkan untuk masukan pemain kedua, tetris yang *dipassing* sebagai parameter adalah tetris2.

```
procedure rotate(var objek : tetris);
var i : longint;
    collide : boolean;
begin
    drawtetris(objek,false);
    ttetris := objek;
    collide := false;
    for i := 2 to 4 do
    begin
        ttetris.points[i].y := objek.points[1].y + objek.points[i].x -
            objek.points[1].x;
        ttetris.points[i].x := objek.points[1].x - objek.points[i].y +
            objek.points[1].y;
        if field[ttetris.points[i].x,ttetris.points[i].y] then
        begin
            collide := true;
            break;
        end;
    end;
    if not collide then objek := ttetris;
    drawtetris(objek,true);
end;
```

Prosedur ini menggunakan prinsip transformasi untuk mengubah posisi dari sebuah tetris.

$$(x,y) \text{ -----}>> (y,-x)$$

Tidak sesederhana pada rotasi transformasi biasanya, rotasi ini tidak menggunakan titik O(0,0) sebagai pusat, namun menggunakan points[1] sebagai pusat transformasi. Hal ini menyebabkan kita akan memperhitungkan kedudukan relatif points[i].x terhadap points[1].x yang kemudian ditambahkan ke points[i].y dan kedudukan relatif points[i].y terhadap

points[1].y yang kemudian ditambahkan ke points[i].x.

Seperti pada prosedur slide, prosedur ini menghapus terlebih dahulu tetris yang dijadikan parameter. Jika tetris ini tidak dihapus terlebih dahulu maka ketika prosedur mengecek keadaan field dari points yang telah diprosesnya akan selalu bernilai true. Setelah pengecekan selesai dan semua field dari points bernilai false maka tetris yang digunakan sebagai parameter perlu dilakukan perubahan. Tetapi ketika field dari points tersebut ada salah satunya yang bernilai true maka tidak perlu dilakukan perubahan pada tetris parameter. Inilah fungsi utama dari ttetris yaitu menyimpan nilai temporer dari sebuah tetris sebelum field dari tetris yang telah diproses. Sebagai dampak dari penghapusan tetris pada saat prosedur dijalankan, kita perlu sekali lagi penggambaran baik tetris telah mengalami perubahan atau tidak sama sekali.

```
procedure shift(var objek : tetris; x, y : longint);
var i : longint;
    collide : boolean;
begin
    drawtetris(objek,false);
    ttetris := objek;
    collide := false;
    for i := 1 to 4 do
    begin
        ttetris.points[i].x := objek.points[i].x + x;
        ttetris.points[i].y := objek.points[i].y + y;
        if field[ttetris.points[i].x,ttetris.points[i].y] then
        begin
            collide := true;
            break;
        end;
    end;
    if not collide then objek := ttetris;
    drawtetris(objek,true);
    if collide then eliminate(objek,22);
end;
```

Shift() merupakan prosedur yang menggeser tetris dari kedudukannya. Yang dilakukan oleh prosedur ini adalah menambahkan properti x dan y dari points tetris dengan nilai yang ada pada parameter. Jadi ketika kita *mempassing* nilai -1 pada parameter x maka tetris tersebut akan bergeser ke kiri, sedangkan ketika kita *mempassing* 1 pada parameter y akan menyebabkan tetris tersebut bergeser ke bawah satu blok.

Setelah sebuah tetris bertubrukan dengan tetris di bawahnya kita perlu melakukan pengecekan terhadap baris – baris yang diisinya, apakah baris tersebut penuh atau tidak. Bila baris tersebut penuh maka kita perlu mengeliminasi baris tersebut dan menurunkan baris – baris

yang ada di atasnya. Maka secara teknis akan dijelaskan seperti ini. Ketika sebuah tetris bertubrukan dengan tetris di bawahnya akan menyebabkan variabel collide bernilai true. Variabel collide yang bernilai true akan menyebabkan pemanggilan prosedur eliminate(). Collide sebelum iterasi perlu diinisialisasi karena variabel ini akan dipakai secara berulang.

```
procedure eliminate(objek : tetris; y : longint);
var i,j,k : longint;
    blank : boolean;
begin
    for i := y downto 3 do
        begin
            blank := false;
            for j := objek.xmodel + 14 to objek.xmodel + 28 do
                if not field[j,i] then
                    begin
                        blank := true;
                        break;
                    end;
            if not blank then
                begin
                    for k := i downto 3 do
                        begin
                            gotoxy(objek.xmodel + 14,k);
                            for j:=(objek.xmodel + 14) to (objek.xmodel + 28) do
                                begin
                                    if field[j,k-1] then write(#178)
                                        else write(#32);
                                    field[j,k] := field[j,k-1];
                                end;
                            end;
                            eliminate(objek,i);
                            break;
                        end;
                    end;
                end;
            end;
        end;
    end;
```

Prosedur eliminate() adalah satu – satunya prosedur yang menerapkan konsep rekursi. Rekursi adalah terminologi yang mengandung definisi dirinya sendiri. Rekursi jarang dipakai

oleh para amatir dalam programnya karena konsep ini membutuhkan pemahaman yang mendalam dan imajinasi yang tinggi.

Prosedur atau fungsi yang bersifat rekursif pada dasarnya terdiri dari basis 0 dan basis 1. Basis 0 adalah bagian dari penyeleksian kondisi yang akan menyebabkan rekursi terhenti sedangkan pada basis 1 rekursi akan terus berjalan dengan memanggil dirinya sendiri. Tetapi dalam program ini rekursi tidak mematuhi konsep basis 1 dan basis 0, karena prosedur `eliminate()` tidak akan berjalan dengan benar mengikuti konsep tersebut. Oleh karena itu diperlukan modifikasi dari konsep yang ada. Perlu diingat bahwa rekursi sangat serakah dalam penggunaan memori, karena setiap kali pemanggilan prosedur akan terjadi penumpukan *chunk* memori. Sebisa mungkin menghindari rekursi dengan cara merubah prosedur tersebut menjadi prosedur linear atau biasa dirubah ke dalam bentuk iterasi.

Seperti yang dikatakan di atas, bahwa prosedur ini akan melakukan pengecekan sebuah baris penuh atau tidak kemudian menurunkan semua baris yang berada di atasnya. Pengecekan dilakukan iterasi mulai dari kolom pertama area pemain sampai kolom terakhir area pemain itu. Ketika baris tersebut terbukti penuh dilakukan eliminasi baris tersebut dan menurunkan baris – baris di atasnya. Setelah itu diperlukan pengulangan pengecekan pada baris tersebut dikarenakan baris yang berada tepat di atasnya bisa saja penuh sehingga perlu dilakukan pengecekan ulang. Pengecekan ulang dilakukan dengan pemanggilan rekursif dirinya sendiri namun tidak perlu dimulai dari baris yang paling bawah namun cukup dimulai dari baris itu sendiri, sehingga akan menghemat penggunaan prosesor untuk hal yang sebenarnya tidak perlu. Pemanggilan `eliminate(tetris,22)` berarti memulai pengecekan dari baris paling bawah. Sedangkan `eliminate(tetris,i)` berarti memulai pengecekan mulai dari baris ke-i.

Selesai sudah uraian pembuatan tetris. Tetris di atas tidak memiliki fungsi penghitungan nilai. Hal ini diharapkan oleh penyusun agar pembaca dapat mengembangkan program tetris ini dan menambahkan fungsi tersebut atau bahkan memodifikasi program ini menjadi lebih baik lagi.

## Biografi Penulis

**Ahmad Priatama**, sekarang masih duduk di kelas tiga SMAN 1 Magelang. Dilahirkan di Bandung 9 september 1990(penting g sich!), lulusan TK sukahaji, SDN Cipadung 2, SMPN 8 bandung, SMAN 1 Magelang dan Ilmu Komputer UGM(doaian aja,Amieen!).

Mengenal komputer ketika duduk pada kelas 1 SMA, ketika itu mengikuti organisasi ekstrakurikuler ICC(informatic clinic community) dan menjabat sebagai ketua(hehe...). Tahun 2007 mengikuti olimpiade komputer sampai pada tingkat pelatnas 1.

Saya berharapa sekali pembaca meluangkan sedikit waktunya untuk mengirim email kepada saya untuk berbagi ilmu atau sekedar korepondensi, karena saya yakin sekali ilmu saya pada saat ini masih bisa dibilang dangkal!Terima Kasih!!