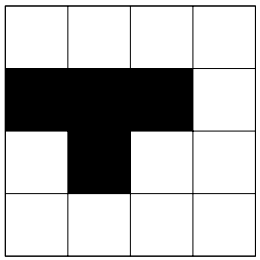
Tetris

Partea a doua

30 noiembrie 2010

Ne amintim: reprezentarea pieselor

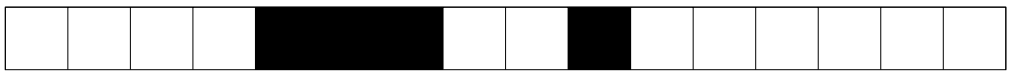
Codificare pe biți



0	0	0	0
1	1	1	0
0	1	0	0
0	0	0	0

Avem 4x4=16 biți

- Încap exact pe un **unsigned short**



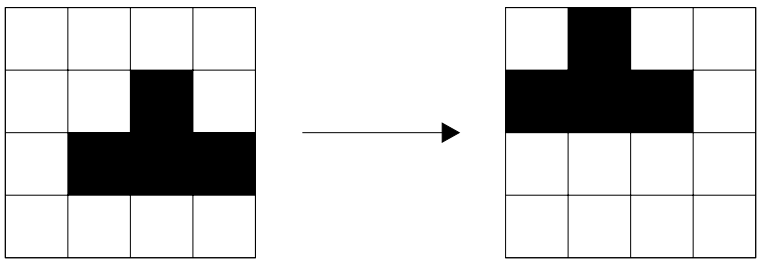
0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0

Valoarea 0x0E40 (în hexa)

Normalizarea unei piese

Dorim să

- Încadrăm orice piesă în spațiul minim posibil
- Calculăm lățimea și înălțimea minime necesare

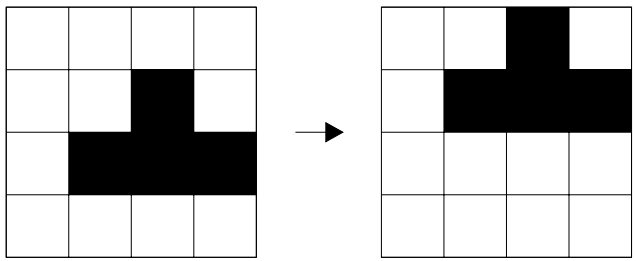


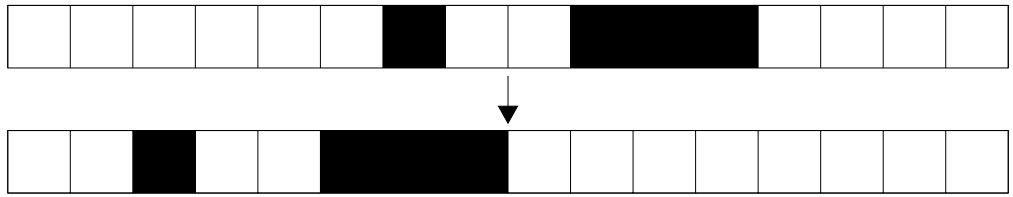
Pentru exemplul de mai sus

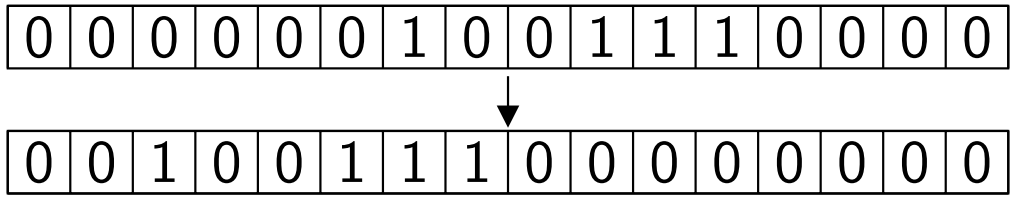
- Inițial: lățime 4, înălțime 3
- Final: lățime 3, înălțime 2

Normalizarea unei piese

Eliminarea liniilor goale







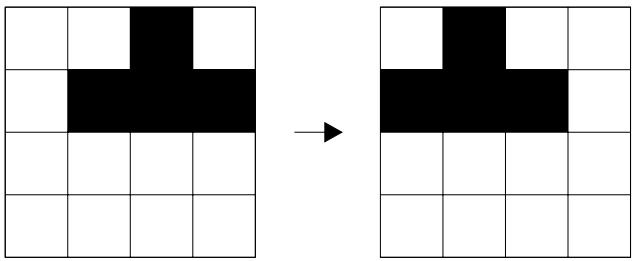
Deplasare cu 4 biți la stânga

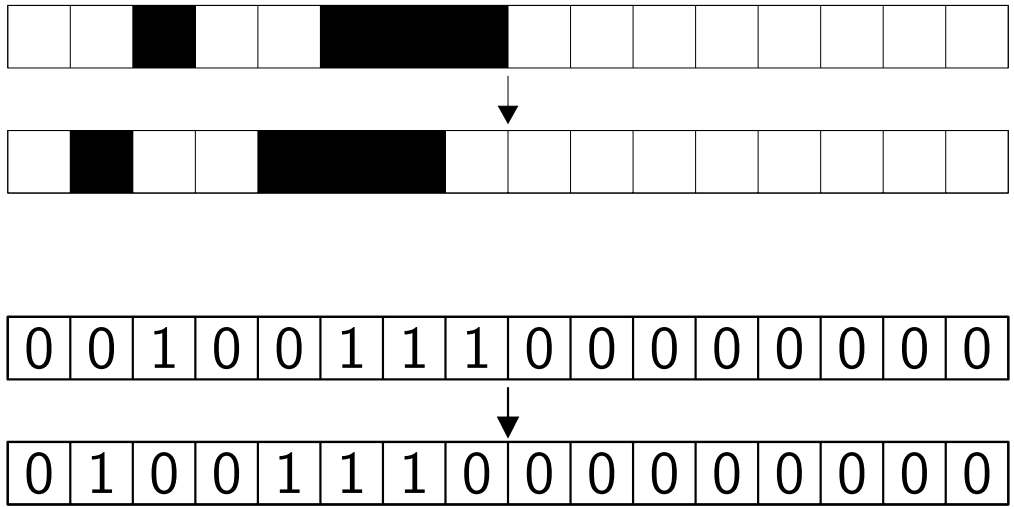
- Atât timp cât primii 4 biți sunt 0

```
while (!(piesa & 0xF000))
  piesa <<= 4;
```

Normalizarea unei piese

Eliminarea coloanelor goale





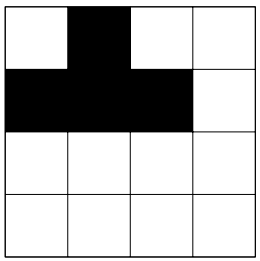
Deplasare cu 1 bit la stânga

- Atât timp cât biții 3, 7, 11 și 15 sunt 0

```
while (!(piesa & 0x8888))
  piesa <<= 1;
```

Normalizarea unei piese

Calculul înălțimii minime



0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0

&

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Scădem din 4 numărul liniilor goale

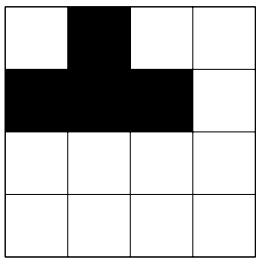
Folosim o mască formată de forma 0000000000001111

- Pe care o deplasăm cu câte 4 poziții spre stânga

```
inaltime = 4;
masca = 0x000F;
while (!(piesa & masca)) {
   inaltime--;
   masca <<= 4;
}
```

Normalizarea unei piese

Calculul lățimii minime



0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0

&

0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

Scădem din 4 numărul coloanelor goale

Folosim o mască formată de forma 0001000100010001

- Pe care o deplasăm cu câte o poziție spre stânga

```
latime = 4;
masca = 0x1111;
while (!(piesa & masca)) {
   latime--;
   masca <<= 1;
}
```

Normalizarea unei piese

```
unsigned short normalizeaza(unsigned short cod,
                            int *inaltime, int *latime) {
  unsigned short masca, cod_nou = cod;
  while (!(cod_nou & 0xF000))
    cod_nou <<= 4;
  while (!(cod_nou & 0x8888))
    cod_nou <<= 1;
  *inaltime = *latime = DIM_PIESA;
  masca = 0x000F;
  while (!(cod_nou & masca)) {
    (*inaltime)--;
    masca <<= 4;
  }
  masca = 0x1111;
  while (!(cod_nou & masca)) {
    (*latime)--;
    masca <<= 1;
  }
  return cod_nou;
}
```

Tabla de joc

Matrice de dimensiune prestabilită

```
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

#define BLOCK_SIZE 32

#define BOARD_HEIGHT SCREEN_HEIGHT/BLOCK_SIZE
#define BOARD_WIDTH BOARD_HEIGHT/2
```

Fiecare element al matricii păstrează o culoare

- Culoarea piesei care a rămas imobilizată în zona respectivă

```
Uint32 tabla[BOARD_HEIGHT][BOARD_WIDTH];
```

Inițial toată tabla este neagră

```
negru = SDL_MapRGB(screen->format, 0, 0, 0);
for (i = 0; i < BOARD_HEIGHT; i++)
  for (j = 0; j < BOARD_WIDTH; j++)
    tabla[i][j] = negru;
```

Piesa curentă

Este piesa care cade la un moment dat

Se caracterizează prin

- Formă: folosim codificarea pe biți
- Dimensiune: lățime și înălțime
  - Rezultă din operația de normalizare
- Poziția curentă: linie și coloană
- Culoare: aleasă aleator

```
unsigned short cod;
int latime, inaltime;
int linie, coloana;
Uint32 culoare;
```

Generarea unei piese noi

Ne amintim: avem un tablou cu codurile pieselor disponibile

```
unsigned short piese[] = { 0x0E80, 0x0E20, 0x0E40,
  0x0660, 0x4444, 0x06C0, 0x0C60
};
```

Pentru a genera o piesă nouă care urmează să cadă

- Alegem aleator un cod de piesă din tablou
- Normalizăm codul ales
  - Ca urmare avem înălțimea și lățimea
- Plasăm piesa în afara tablei
  - În partea de sus a tablei, pe mijloc
- Alegem aleator o culoare pentru piesă

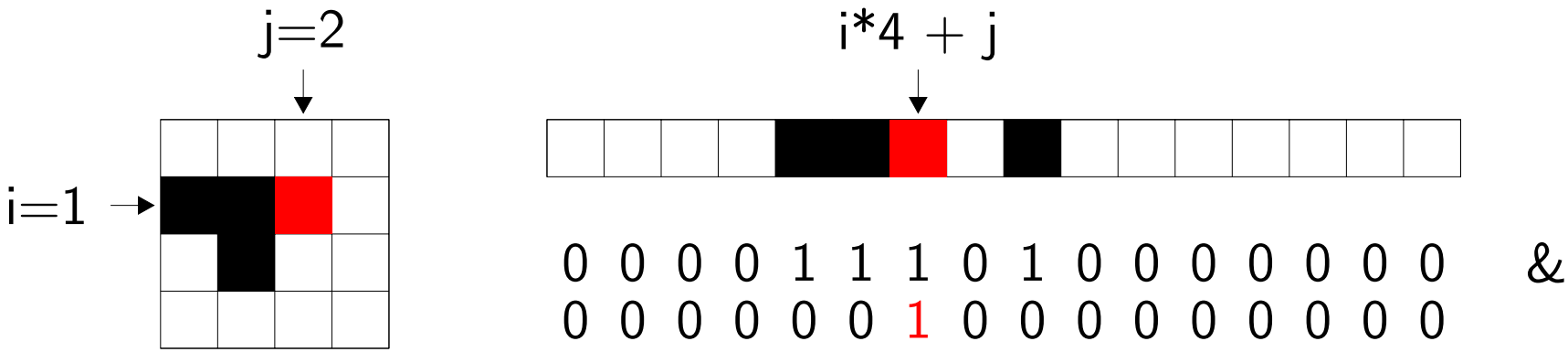
Generarea unei piese noi (2)

```
void piesa_noua(SDL_Surface * screen) {
  int idx, rosu, verde, albastru;
  int npiese = sizeof(piese) / sizeof(piese[0]);
  idx = rand() % npiese;
  cod = normalizeaza(piese[idx], &inaltime, &latime);
  linie = -inaltime;
  coloana = (BOARD_WIDTH - latime) / 2;
  rosu = 100 + rand() % 156;
  verde = 100 + rand() % 156;
  albastru = 100 + rand() % 156;
  culoare =
      SDL_MapRGB(screen->format, rosu, verde, albastru);
}
```

Desenarea piesei curente

Scriem o funcție ajutătoare

- Pentru a determina dacă la coordonatele *(i, j)* e bloc sau spațiu




```
int are_bloc(unsigned short piesa, int i, int j) {
  unsigned short masca = 0x8000;
  masca >>= i * 4 + j;
  return (piesa & masca) != 0;
}
```

Desenarea piesei curente (2)

```
void deseneaza_piesa(SDL_Surface * screen) {
  int i, j;
  SDL_Rect r;
  Uint32 c;
  r.w = r.h = BLOCK_SIZE;
  for (i = 0; i < inaltime; i++) {
    for (j = 0; j < latime; j++) {
      if (pe_tabla(linie + i, coloana + j)) {
        r.x = (coloana + j) * r.w;
        r.y = (linie + i) * r.h;
        if (are_bloc(cod, i, j))
          c = culoare;
        else
          c = tabla[linie + i]
                [coloana + j];
        SDL_FillRect(screen, &r, c);
      }
    }
  }
}
```

Desenarea tablei

Întâi desenăm tabla propriu-zisă

Pe urmă desenăm piesa curentă

- În poziția în care se află în acest moment

```
void deseneaza_tabla(SDL_Surface * screen) {
  int i, j;
  SDL_Rect r;
  r.w = r.h = BLOCK_SIZE;
  for (i = 0; i < BOARD_HEIGHT; i++) {
    for (j = 0; j < BOARD_WIDTH; j++) {
      r.x = j * r.w;
      r.y = i * r.h;
      SDL_FillRect(screen, &r, tabla[i][j]);
    }
  }
  deseneaza_piesa(screen);
  SDL_Flip(screen);
}
```

Alte operații necesare

Alte operații de care avem nevoie:

- Verificare dacă piesa curentă se poate deplasa
  - În jos (piesa cade)
  - La stânga/dreapta (jucătorul mută piesa)
- "Contopirea" piesei cu tabla
  - Când ea nu mai poate să cadă
- Eliminarea liniilor pline de pe tablă
- Efectul de cădere a piesei curente

În funcția principală toate aceste operații se repetă

- Până când utilizatorul iese din joc
- Până când nu se mai pot pune piese pe tablă

Pentru detalii urmăriți codul sursă atașat