TETRIS

11.01.2024

Îndrumător:

dr. ing. Daniel Morariu

Student:

Lupean Bianca-Ioana

(223/2)

Istoric Versiuni

Data	Versiune	Descriere	Autor
05.01.2024	1.0D	aGrid.setIndex(tiles[index].Row,tiles[index].Column, PIESA_MISCARE);	Daniel Morariu
08.01.2024	1.1D(finala)	aGrid.setIndex(tiles[index].Row,tiles[index].Column, ID);	Lupean Bianca-Ioana

Aceste două versiuni se referă la versiunile metodei Draw din clasa Block. Această metodă nu desenează propriu-zis, ci completează matricea jocului cu valori numerice în funcție de care se va face desenarea propriu-zisă de către metoda Draw din clasa Game.

În versiunea scrisă de domnul profesor (1.0D), dânsul a ales ca în matrice să fie reținute valorile în felul următor: 0 pentru celulă goală, 1 pentru celulele blocului aflat în mișcare și 2 pentru piesa așezată.

În versiunea scrisă de mine(1.1D), am ales să mă folosesc de variabila membru ID a fiecărui bloc, deoarece acest lucru a făcut posibilă desenarea fiecărui bloc cu o culoare diferită față de celelalte. Aceste modificări au avut loc și în metodele Lock și IsBlockOutside din clasa Game, întrucat și acestea lucrează cu matricea jocului.

Cuprins

1 Specificarea cerințelor software

1.1 Introducere

Tetris este un joc video de tip puzzle creat de catre inginerul Alexei Pajitnov în anul 1984. Este un joc autetic, îndrăgit de mulți oameni, acesta fiind motivul pentru care am ales această temă de proiect. În jocul Tetris, participanții umplu rândurile prin mutarea pieselor variate care coboară în zona de joc. Atunci când se formează rânduri complete, acestea dispar, aducând puncte jucătorului și oferindu-i posibilitatea de a elibera mai mult spațiu. Partida se încheie atunci când ecranul este complet ocupat. Cu cât jucătorul reușește să amâne umplerea ecranului, cu atât va obține un scor mai mare.

1.1.1 Obiective

1. **Configurarea unui „game-loop”** (cu ajutorul unei componente de tip timer)
2. **Crearea grid-ului**, o matrice de 20 de linii și 10 coloane care reține matematic starea celulelor din DrawGrid, iar în funcție de această stare se va realiza desenarea blocurilor
3. **Crearea blocurilor**
4. **Mutarea blocurilor**
5. **Rotirea blocurilor**
6. **Verificarea coliziunilor** (spre exemplu, daca un bloc a cazut peste altul asezat anterior)
7. **Verificarea rândurilor complete, dizolvarea acestora și mutarea în jos a celor rămase**
8. **Game Over**
9. **Counter pentru scor**
10. **Incrementarea nivelelor**

1.1.2 Definiții, Acronime și Abrevieri

Din clasa Block:
std::map<int, std::vector<Position> > tiles; - reține poziția blocului în fiecare stadiu de rotație
Std::vector<Position> GetCellPositions(); -această metodă returnează poziția celulelor ocupate de blocuri după aplicarea offset-ului

Din clasa Game:
std::vector<Block> blocks; -variabilă membru care va reține toate tipurile de blocuri
std::vector<Block> GetAllBlocks();- metodă care va inițializează variabila membru blocks de mai sus în cadrul constructorului (blocks=GetAllBlocks();)

1.1.3 Tehnologiile utilizate

#include <map>
Am folosit o mapă cu chei de tip int și conținut de tip vector cu elemente de tip Position (clasă creată de mine) pentru a putea stoca organizat setul de coordonate ale blocului în fiecare stadiu de rotație. De exemplu, blocul L în stadiul 1 de rotație are coordonatele (0,2), (1,0), (1,1), (1,2) (cheia-1, conținutul-vectorul de perechi de coordonate).

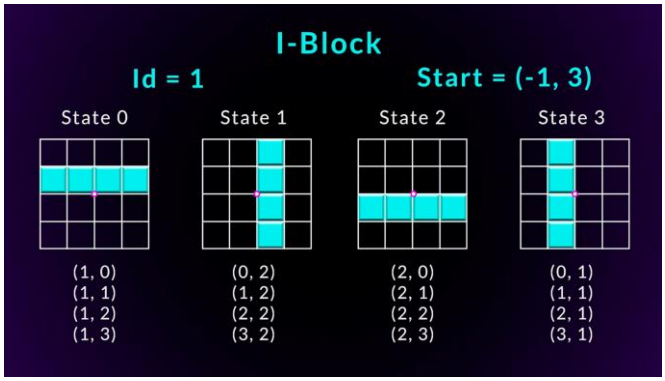
#include <vector>
Am folosit elemente de tip vector atât ca elemente auxiliare, pentru umplerea mapei menționate mai sus, cât și ca tip returnat pentru variabile mebru și metode din cadrul claselor (vezi 1.1.2).

1.2 Cerințe specifice

1. **Configurarea unui „game-loop”:**
   Acest lucru l-am realizat cu ajutorul unei componente TTimer pe care am denumit-o tTime. Când apăsăm butonul START proprietatea Enabled a componentei tTime devine true, adică ceasul începe să „ticaie”. Am creat un eveniment pe care l-am asociat timer-ului în cadrul căruia, atâta timp cât timer-ul este pornit, se efectuează metodele HandleInput si Draw care deplasează în grid obiectul ob de clasa Game. Generarea unui obiect nou imediat dupa așezarea celui anterior se face datorita metodei HandleInput în cadrul căreia, atunci când blocul se deplasează în jos, cînd nu mai poate coborî, se apelează metoda Lock. Aceasta completează celulele ocupate de blocul cazut în grid cu valoarea ID-ului blocului și generează un bloc nou.

2. **Crearea grid-ului:**
   Când instanțiem un obiect ob de clasă Game, în constructorul acestuia apelăm constructorul clasei GameGrid care inițializează variabila membru numită grid a obiectului ob. Acest grid este defapt o matrice de douăzeci de linii și zece coloane, care este practic „oglinda” celulelor componentei DrawGrid pe care se desfășoară jocul nostru.
   În clasa GameGrid avem, pe lângă constructorul menționat mai sus, metode precum Initialize, getIndex si setIndex. Metoda Initialize este apelată in constructor și setează valorile întregii matrici cu 0 (asta va însemna mai tarziu că acele celule sunt goale, că nu s-a așezat încă niciun bloc acolo). Metodele getIndex si setIndex sunt apelate de către alte metode ale clasei Game pentru a facilita desenarea blocurilor.

3. **Crearea blocurilor:**
   Am creat clasa Block pentru care am avut nevoie de elemente încuibărite de tip Position care să rețina offset-ul de start al blocului (acesta fiind linia 0 și coloana 0) si offset-ul după mutarea sau rotirea blocului. Mapa tiles reține poziția blocului în fiecare stadiu de rotație, mai precis coordonatele pe care le-ar ocupa blocul dacă l-am încadra într-o matrice de 3 pe 3.



   Din această clasă am derivat alte 7 clase corespunzatoare celor 7 „tetrominoes”. Pentru fiecare bloc în parte am completat mapa tiles cu valorile corespunzătoare.
   Pe baza acestor coordonate, mai precis, pe baza coordonatelor din stadiul de rotatie 0 (deaorece vorbim despre crearea blocului) metoda Draw din clasa Block va completa pozițiile necesare în grid ajutându-se de metoda GetCellPositions, iar metoda Draw din clasa Game va desena propriu-zis în drawGrid pătrățelele corespunzătoare. Pentru ca acestea să se spauneze în mijlocul grid-ului, am folosit metoda Init(int rows, int cols).

4. **Mutarea blocurilor**:
   Pentru mutarea blocurilor m-am folosit de variabila membru direction pe care am setat-o în constructor să fie la început JOS, pentru ca blocul să coboare. În clasa Game am creat o metodă numită Directie care primește ca parametru o tastă, iar în funcție de tasta primită, modifică valoarea variabilei membru direcție. Mai departe, am implementat metoda HandleInput care se apelează în urma celei de mai sus. HandleInput verifică, în primul rând, dacă jocul nu s-a terminat (dacă blocurile nu au ajuns în varful grid-ului) iar mai apoi se asigură ca blocul să fie în interiorul gridului și îl mută cu ajutorul metodei Move(GameGrid &aGrid, int, int) din clasa Block. Metoda Move șterge blocul de pe poziția curentă (setează coordonatele cu 0) și îl redesenează după incrementarea offset-ului (setează coordonatele rezultate în urma aplicării offset-ului cu valoarea ID-ului blocului). Repet, desenarea propriu-zisă se face mai apoi cu metoda Draw a clasei Game.

5. **Rotirea blocurilor**:
   Similar mutării blocurilor, pentru a le roti avem nevoie de mapa tiles. Rotirea se face în momentul apăsării tasteti UP.
   Când tasta UP este apăsată, dacă jocul nu s-a terminat se apelează metoda RotateBlock din clasa Game. Aceasta verifică daca blocul este patrat (acest lucru este necesar pentru că pătratul are aceleași coordonate chiar dacă îl rotim). Dacă blocul nu este pătrat, acesta este șters și se apelează metoda Rotate din clasa Block care nu face altceva decât că modifică variabila membru numită rotationState, sau o resetează la 0. Mai apoi, dacă în urma rotirii blocul nu riscăm ca acesta să iese înafara grid-ului, se actualizează starea celulelor gridului și se redesenează blocul.

6. **Verificarea coliziunilor:**
   Acest aspect trebuie luat în calcul înainte de fiecare mutare, rotire sau așezare a unui bloc. Metodele IsBlockOutside și CanSettle se ocupă de acest lucru. IsBlockOutside funcționează în felul următor: cu ajutorul metodei IsCellOutside din clasa GameGrid se verifică dacă fiecare celulă a blocului este sau nu în interiorul grid-ului. Această metodă putem spune că este multifuncțională deoarece pe lângă această verificare, ea și modifică conținutul grid-ului în funcție de răspunsul acelei verificări. Metoda CanSettle se ajută de motada IsCellEmpty din clasa GameGrid care se apelează cu coordonatele rândului de sub blocul aflat în cădere (Row+1). Dacă celulele nu sunt goale (figurează în grid cu o valoare numerică diferita de 0) nu se poate trece peste ele, iar blocul rămâne deasupra.

7. **Verificarea rândurilor complete, dizolvarea acestora și mutarea în jos a celor rămase:**
   Toate aceste lucruri sunt asigurate de metoda ClearFullRows din clasa GameGrid, care însumează mai multe metode. Întâi se face verificarea dacă un rand este plin sau nu. Dacă una dintre celulele de pe acel rând este zero, acel rând nu este plin. În cazul în care rândul este plin, incrementăm un counter. După acest lucru, dacă valoarea counteru-ului este mai mare decat zero, se apelează pentru acel rând si acea valoare a counterului metoda MoveRowDown(int row, int counter), care face modificări în grid pentre redesenarea ulterioară. Mai precis, celulele rândului plin iau valoarea celulelor rândului de deasupra, iar rândul care a fost inițial deasupra devine gol, pentru a se putea așeza alte piese acolo. Odată modificate corespunzător valorile din grid, metoda Draw a clasei Game se va ocupa de partea vizibilă a acestui proces.

8. Game Over

În clasa Game am adăugat o variabilă membru de tip bool numită gameOver, pe care am setat-o în constructor să fie inițial falsă, din motive evidente. Această variabilă devine true în momentul în care blocurile au ajuns până sus și nu mai este posibil să cadă altul, moment în care pe ecran se afișează mesajul „Game Over!” (chiar dacă nu aș fi optat pentru afișarea unui mesaj, jocul oricum s-ar fi oprit din cauza variabilei gameOver care a devenit adevarată). Verificarea stării de adevar a variabilei gameOver se verifică înainte de fiecare mișcare.

2 Mutarea blocurilor

2.1 Descriere

Consider că această funcționalitate este relevantă pentru a fi detaliată deoarece prezintă multe condiții și este unul ditre evenimentele pe care le putem controla ca și jucatori de la tastatură, de accea ar fi interesant să înțelegem mai bine ce se întamplă „în spate” atunci când apăsăm una dintre taste.

2.2 Fluxul de evenimente

2.2.1 Fluxul de bază

Mutarea blocurilor se face atât automat, pe baza timer-ului (coborârea lor în jos), dar și de la tastatură. Iată metodele care se ocupă de mutarea blocurilor:

```
void Game::Directie(WORD &Key){
	switch(Key){
		case VK_UP: if(!gameOver){RotateBlock();} break;
		case VK_DOWN: if(!gameOver){if(IsBlockOutside()) direction=JOS;}
else{gameOver=true; ShowMessage("Game Over");} break;
		case VK_LEFT: if(!gameOver){if(IsBlockOutside()) direction=STANGA;} break;
		case VK_RIGHT: if(!gameOver){if(IsBlockOutside()) direction=DREAPTA;} break;
	}
}
```

Moteda Directie primește ca parametru o tastă apasată de către utilizator, iar în funcție de aceasta, modifică componenta direction a obiectului de clasă Game. Componenta direction este inițial setată în constructor ca fiind JOS, pentru ca blocul să poată coborî automat în jos pe baza timer-ului, fara a necesita apăsarea tastei DOWN.

```
void Game::HandleInput(){
currentBlock.Delete(grid);
switch(direction){
		case JOS: if(!gameOver){grid.ClearFullRows(); if(IsBlockOutside() && (CanSettle()==true))
							currentBlock.Move(grid, 1,0); else {Lock();}} break;
		 case STANGA: if(!gameOver){if(!IsBlockOutside())currentBlock.Move(grid,0,1);
				else {currentBlock.Move(grid, 0,-1); } ; direction=JOS; }  break;
	    case DREAPTA: if(!gameOver){if(!IsBlockOutside()) currentBlock.Move(grid,0,-1);
						else {currentBlock.Move(grid, 0,1);} ; direction=JOS;} break;
}}
```

Aceasta este motoda care face propriu zis mutările, ajutându-se de funcția Move din clasa Block:

```
void Block::Move(GameGrid &aGrid, int rows, int columns){
	Delete(aGrid);	//sterge blocul de pe pozitia curenta
	offset.Row+=rows;		//actualizeaza
	offset.Column+=columns;	//offset-ul
	Draw(aGrid);	//redeseneaza blocul in pozitia obtinuta dupa actualizarea offset-ului
}
```

Metodele Delete și Draw de mai sus sunt următoarele:

```
void Block::Draw(GameGrid &aGrid){
        std::vector<Position> tiles=GetCellPositions();
       for (int index = 0; index < tiles.size(); ++index) {
        aGrid.setIndex( tiles[index].Row,tiles[index].Column, ID);
   }
}
void Block::Delete(GameGrid &aGrid){
      std::vector<Position> tiles=GetCellPositions();
       for (int index = 0; index < tiles.size(); ++index) {
       aGrid.setIndex(tiles[index].Row,tiles[index].Column,GOL);}
}
```

Aceste metode nu șterg și desenează propriu-zis, ci actualizează starea grid-ului, pe baza căruia se va desena cu ajutorul metodei Draw din clasa Game, care este următoare:

```
void Game::Draw(){
      for (int row = 0; row < grid.Rows; ++row) {
       for (int col = 0; col < grid.Columns; ++col){

          TRect d = FereastraTetris->dgGameGrid->CellRect(col, row);
           switch (grid.getIndex(row, col))
           {
             case GOL:
                    FereastraTetris->dgGameGrid->Canvas->Brush->Color = cl3DDkShadow;
                    break;
             case I:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clAqua;
                   break;

             case L:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clBackground;
                   break;
             case J:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clBlue;
                   break;
             case T:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clFuchsia;
                   break;
             case S:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clLime;
                   break;
             case Z:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clRed;
                   break;
             case O:
                   FereastraTetris->dgGameGrid->Canvas->Brush->Color = clYellow;
                   break;
           }
           FereastraTetris->dgGameGrid->Canvas->Rectangle(d);
          }
```

```
    }
}
```

Această metodă este mai benefică decît cea care va fi prezentată în capitolul Fluxuri alternative deoarece permite desenarea cu o culoare diferită pentru fiecare tip de bloc.

2.2.2 Fluxuri alternative

2.2.2.1 < Primul flux alternativ >

Există un flux alternativ pentru metoda Draw a clasei Game, dar are un dezavantaj. Blocurile nu pot avea culori diferite.

```
void Game::Draw(){
    for (int row = 0; row < grid.Rows; ++row) {
     for (int col = 0; col < grid.Columns; ++col){

        TRect d = FereastraTetris->dgGameGrid->CellRect(col, row);
         switch (grid.getIndex(row, col))
         {
           case GOL:
                FereastraTetris->dgGameGrid->Canvas->Brush->Color = cl3DDkShadow;
                break;
           /*case PIESA_MISCARE:
                FereastraTetris->dgGameGrid->Canvas->Brush->Color = clYellow;    //piesele aflate în
                                                                                  //mișcare ar avea culoarea galben
                break;*/

           /*case PIESA_ASEZATA:
                    FereastraTetris->dgGameGrid->Canvas->Brush->Color = clBlue;  //iar dupa ce ar fi
                                                                                  //așezate ar deveni albastre
                break;*/
}}}
```

2.2.3 Pre-condiții

Inainte de a muta un bloc, trebuie să ne asigurăm că acesta este și rămâne în interiorul grid-ului. Acest lucru îl facem prin metoda urmatoare:

```
bool Game::IsBlockOutside(){
    std::vector<Position> tiles=currentBlock.GetCellPositions();
    for (int index = 0; index < tiles.size(); ++index){
        if(grid.IsCellOutside(tiles[index].Row, tiles[index].Column)){
            for (int index = 0; index < tiles.size(); ++index){
                                        //am ajuns jos trebuie sa scriu piesa in grid si sa generez alta
             grid.setIndex(tiles[index].Row,tiles[index].Column, currentBlock.ID);
            }
             currentBlock=nextBlock;
             nextBlock=GetRandomBlock();
            return false;
        }     } return true; }
```

Pentru a înțelege mai bine ce face metoda, trebuie să vedem, și metoda IsCellOutside care face parte din clasa GameGrid:

```
bool GameGrid::IsCellOutside(int r, int c){
	if(r>=0 && r<(Rows-1) && c>=0 && c<(Columns)){return false;}
	return true;
}
```

Practic, dacă coordonatele unei celule ale blocului iese înafara grid-ului, blocul se consideră înafara grid-ului.

Erori: metoda IsCellOutside am scris-o așa ca să meargă doar în cazul în care nu apăsăm tasta stanga sau dreapta până când ieșim afară, deoarece blocurile încă pot ieși în extremitatea stangă și în cea dreaptă.

2.2.4 Post-condiții

După rularea acestei funcționalități utilizatorul poate observa cum piesele coboară automat în jos, având totodată posibilitatea să le miște stânga-dreapta prin intermediul tastelor.

3 Verificarea rândurilor complete, dizolvarea acestora și mutarea în jos a celor rămase

3.1 Descriere

Consider că această funcționalitate este importantă deoarece aceasta face deosebirea între un joc Tetris clasic și unul de tipul Block Clash, în care trebuie să completezi atât rânduri cât și coloane.

3.2 Fluxul de evenimente

3.2.1 Fluxul de bază

Acest proces este în totalitate dependent de matricea grid, ca tot jocul în sine. Pentru ca aceste lucruri să se întample, se apelează metoda ClearFullRows în metoda HandleInput în case: JOS

```
case JOS: if(!gameOver){grid.ClearFullRows(); if(IsBlockOutside() && (CanSettle()==true))
                                                currentBlock.Move(grid, 1,0); else {Lock();}} break;
```

Să aruncăm o privire asupra implementării acestei metode:

```
int GameGrid::ClearFullRows(){
      int cleared=0;     //variabilă auxiliară cu ajutorul căreia ținem minte câte rânduri trebuie să mutăm
     for(int r=(this->Rows)-1; r>=0; --r){    //facem verificarea începând de la ultimul rând, în sus
            if(IsRowFull(r)){   //parcurgem rândul iar dacă chiar și o singură celulă este goală (figurează
                                  // cu valoarea 0 în grid) acest rând nu este în concluzie plin
                  ClearRow(r);  //setăm valorile celulelor rândului cu zero în matricea grid
                  cleared++;  //incrementam variabila auxiliară
            }
             else if(cleared>0){
                  MoveRowDown(r, cleared);  //o detaliel amplu mai jos
            }
      }
      return cleared;  //această linie de cod ar fi trebuit să ajute la implementarea scorului
}

void GameGrid::MoveRowDown(int r, int numRows){
      for(int c=0; c<Columns; c++){
            grid[r+numRows][c]=grid[r][c];
            grid[r][c]=GOL;
      }
}
```

Aici observăm cum se face o „pasare” a valorilor rândului de deasupra celui/celor pline către ultimul rând, dar asta numai după ce randurile pline au fost golite .
Desigur că de partea vizuală a acestui proces se ocupă tot metodelele detaliate la funcționalitatea Mutarea blocurilor. Matodele ClearFullRows si MoveRowDown doar pregătesc matricea grid în funcție de care se va redesena toată tabla jocului.

3.2.2 Pre-condiţii

Pentru ca un rând să fie plin metoda IsRowFull apelată lui trebuie să întoarcă valoarea true.

```
bool GameGrid::IsRowFull(int r) {
   for (int c = 0; c < Columns; c++) {
       if (grid[r][c]==0) {
          return false;
       }
   }
   return true;
}
```

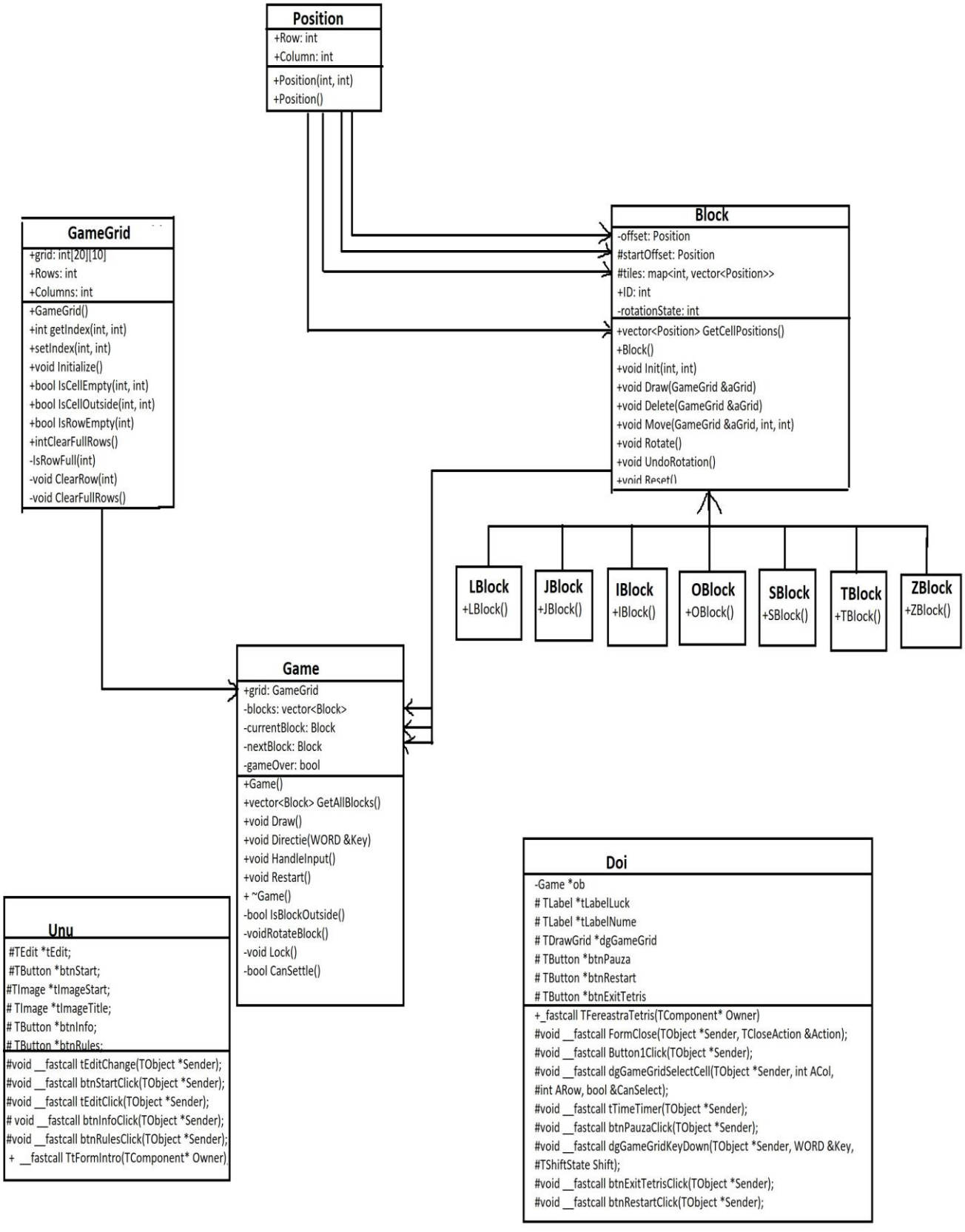
Un rând este plin atunci când în matricea grid, pe acel rând, toate celulele de pe toate coloanele coțin o valoare diferită de zero.

3.2.3 Post-condiţii

În urma rulării acestei funcționalități, utilizatorul poate observa cum un rând se dizolvă, dipare, în momentul îin care acesta s-a umplut, iar celelalte de deasupra lui coboară în jos.

4 Implementare

4.1 Diagrama de clase



5 Bibliografie

Tutorial Tetris in C++ with raylib: https://youtu.be/wVYKG_ch4yM?si=PkIFVbVKO_ZoGu5-

Tutorial Tetris in C#: https://youtu.be/jcUctrLC-7M?si=IYzVCie0il4Cpoic

Realizare joc Snake C++ Builder 6, dr. ing. Daniel Morariu:
https://drive.google.com/file/d/14EayOduIXx3mQldL63nsJx896aH-UDn0/view

http://www.functionx.com/cppbuilder/