

Omida

Cuprins

1	Omida	1
2	Cum se joaca?	3
3	Cum functioneaza?	5
4	La ce ajuta?	7
5	Extra	9
6	Indexul Modulelor	11
6.1	Module	11
7	Indexul Namespace-ului	13
7.1	Lista de Namespace-uri	13
8	Index Ierarhic	15
8.1	Ierarhia Claselor	15
9	Indexul Claselor	17
9.1	Lista Claselor	17
10	Indexul Fişierelor	19
10.1	Lista fişierelor	19

11 Documentația Modulelor	21
11.1 Meniu start	21
11.2 Jocul efectiv	22
11.3 Pauza	23
11.4 Final	24
11.5 Desenand pe ecran	25
11.6 Cum reprezint obiectele jocului?	26
11.7 Meniuri	27
11.8 Principala repetitie	28
12 Documentația Namespace-ului	29
12.1 Referință la Namespace-ul global	29
12.1.1 Descriere Detaliată	29
12.1.2 Documentația variabilelor	29
12.1.2.1 desenator	29
12.1.2.2 fereastra_inchisa	30
12.1.2.3 scene	30
12.1.2.4 taste_apasate	30
13 Documentația Claselor	31
13.1 Referință la clasa Joc	31
13.1.1 Descriere Detaliată	32
13.1.2 Documentația Funcțiilor Membre	32
13.1.2.1 incarca()	33
13.1.2.2 incarcat_deja()	33
13.1.2.3 itereaza()	33
13.1.2.4 muta_omida()	34
13.1.2.5 pauza()	34
13.1.2.6 verifica_taste_apasate()	34
13.1.3 Documentația Datelor Membre	35
13.1.3.1 m_timp_asteptat	35

13.1.3.2	noua_directie_a_omizii	35
13.2	Referință la clasa MeniuFinal	35
13.2.1	Descriere Detaliată	36
13.2.2	Documentația Funcțiilor Membre	37
13.2.2.1	incarca()	37
13.2.2.2	incarcat_deja()	37
13.2.2.3	itereaza()	37
13.2.3	Documentația Datelor Membre	37
13.2.3.1	m_istoric	38
13.3	Referință la clasa MeniuStart	38
13.3.1	Descriere Detaliată	39
13.3.2	Documentația Funcțiilor Membre	39
13.3.2.1	incarca()	39
13.3.2.2	incarcat_deja()	40
13.3.2.3	itereaza()	40
13.4	Referință la structura Punct	40
13.4.1	Descriere Detaliată	41
13.4.2	Documentația pentru Constructori și Destructori	41
13.4.2.1	Punct() [1/3]	41
13.4.2.2	Punct() [2/3]	41
13.4.2.3	Punct() [3/3]	42
13.5	Referință la clasa Scena	42
13.5.1	Descriere Detaliată	42
13.5.2	Documentația Funcțiilor Membre	43
13.5.2.1	incarca()	43
13.5.2.2	incarcat_deja()	43
13.5.2.3	itereaza()	43
13.6	Referință la clasa StadiulJocului	44
13.6.1	Descriere Detaliată	45
13.6.2	Documentația Datelor Membre	45
13.6.2.1	directia_omizii	45
13.6.2.2	lungime_textura	45
13.6.2.3	pozitie_powerup	46
13.6.2.4	pozitii_omida	46
13.6.2.5	scor	46
13.6.2.6	teren	46

14 Documentația Fișierelor	47
14.1 Referință la fișierul main.cpp	47
14.1.1 Documentația enumerărilor	49
14.1.1.1 PosibilitateTeren	49
14.1.2 Documentația funcțiilor	49
14.1.2.1 deseneaza_cap()	49
14.1.2.2 deseneaza_omida()	49
14.1.2.3 genereaza_pozitie_noua()	50
14.1.2.4 initializeaza()	50
14.1.2.5 main()	50
14.1.2.6 marcheaza_frunza()	51
14.1.3 Documentația variabilelor	51
14.1.3.1 directie	51
Index	53

Capitolul 1

Omida

Omida este un joc care seamana foarte tare cu Snake. Nu l-am putut numi snake pentru ca numele era deja luat :(.

- [Cum se joaca?](#)
- [Cum functioneaza?](#)
- [La ce ajuta?](#)
- [Extra](#)

Capitolul 2

Cum se joaca?

- [Meniu start](#)
- [Jocul efectiv](#)
- [Pauza](#)
- [Final](#)

Capitolul 3

Cum functioneaza?

- Desenand pe ecran
- Cum reprezint obiectele jocului?
- Meniuri
- Principala repetitie

Capitolul 4

La ce ajuta?

Un joc in sine nu are ata de multe aplicatii in lumea reala in afara de entertainment. Totusi consider ca acest proiect reprezinta un foarte bun inceput pentru oricine vrea sa inceapa programarea grafica. Aceasta are mult extrem de multe aplicatii precum vizualizarea organelor corpului uman, simulari, design etc. "Omida" nu este ceva complex dar este o fundatie pentru a dezvolta subiecte mai avansate. Chiar si conceptul scenelor intr-un joc este foarte important deoarece cand vrem sa facem un joc nu ne gandim prima data la meniu ci la joc in sine. Dupa ce am terminat jocul intervine problema de reprezentare a meniurilor, conceptul de scena ajutand enorm. Scenele pot fi extinse si reprezentate ca niste arbori pentru flexibilitate maxima(de exemplu submeniuri). De asemenea poate fi extins si modul in care functioneaza [main](#), in loc sa chemam fiecare [Scena::itereaza](#) cu [StadiulJocului::timp_trecut](#) variabil, putem stabili un timp maxim (de exemplu 60fps) cu care sa actualizam scena, folositor in cazul unei simulari in care legile fizicii depind foarte mult de timpul trecut.

Capitolul 5

Extra

Codul sursa este complet liber, poate fi gasit la <https://gitlab.com/librehead/atestat> si se afla sub licenta Apache 2.0. Libraria SDL este si ea open source sub licenta zlib. Documentatia am generat-o cu ajutorul doxygen, de asemenea proiect open source, sub licenta GNU General Public License.

Capitolul 6

Indexul Modulelor

6.1 Module

Lista tuturor modulelor:

Meniu start	21
Jocul efectiv	22
Pauza	23
Final	24
Desenand pe ecran	25
Cum reprezint obiectele jocului?	26
Meniuri	27
Principala repetitie	28

Capitolul 7

Indexul Namespace-ului

7.1 Lista de Namespace-uri

Lista tuturor namespace-urilor documentate , cu scurte descrieri:

global	29
----------------------------------	--------------------

Capitolul 8

Index Ierarhic

8.1 Ierarhia Claselor

Această listă de moșteniri este sortată în general, dar nu complet, în ordine alfabetică:

Punct	40
Scena	42
Joc	31
MeniuFinal	35
MeniuStart	38
StadiulJocului	44

Capitolul 9

Indexul Claselor

9.1 Lista Claselor

Lista claselor, structurilor, uniunilor și interfețelor, cu scurte descrieri:

Joc		
Scena	jocului in sine	31
MeniuFinal		
Meniul care va fi afisat dupa terminarea unui joc		35
MeniuStart		
Defineste meniul afisat la deschiderea jocului		38
Punct	40
Scena		
Interfata pentru toate scenele		42
StadiuJocului		
Contine toata informatia de care avem nevoie in timpul jocului		44

Capitolul 10

Indexul Fișierelor

10.1 Lista fișierelor

Lista tuturor fișierelor documentate, cu scurte descrieri:

main.cpp	47
--------------------------	----

Capitolul 11

Documentația Modulelor

11.1 Meniu start

La inceput jocul afiseaza un meniu cu 2 optiuni: joaca si iesire.

Meniul selectat are un fundal mov inchis iar pentru a intra in meniul respectiv se apasa tasta ENTER. Pentru a selecta alt meniu fie se apasa click pe meniul pe care il vreti, fie folositi sagetile, fie tastele j si h(j - jos, k - sus).

Pentru a iesi din joc(in meniul de start) in afara de a apasa "iesire" se poate apasa tasta 'x' sau ESCAPE.

Odata ce ati selectat "joaca" va incepe jocul efectiv.

11.2 Jocul efectiv

Jocul va incepe cu o omida mica asezata aproximativ in centrul ferestrei. Pe capul omizii este afisat scorul curent(care incepe cu 0). Omida se misca automat in directia setata de jucator la un interval de 150 de milisecunde, reprezentat de variabila [StadiulJocului::timp_asteptare_maxim](#).

Initial directia in care se va deplasa omida este `TipDirectie::SUS`. Toate directiile sunt in enumeratia [TipDirectie](#). Pentru a schimba directia omizii se folosesc fie sagetile, fie tastele h, j, k, l(inspirate din vim, h - `TipDirectie::STAG`, j - `TipDirectie::JOS`, k - `TipDirectie::SUS`, l - `TipDirectie::DREAPTA`).

De asemenea pe undeva in teren este afisata si o frunza care va incrementa scorul cu 1 daca veti ajunge la ea. Cand "mancati" o frunza veti auzi si sunetul reprezentat de variabile [Joc::m_sunet_frunza](#). Exista si un powerup in joc(adica un mar) care va incrementa scorul cu 10. Marul este generat odata la 20 de secunde(evident, daca nu este deja in teren). Sunetul redat la "mancarea" unui mar este reprezentat de variabila [Joc::m_sunet_powerup](#).

Terenul are dimensiunile 20x20 ceea ce inseamna ca intervalul de 150ms poate fi un pic cam mare. Pentru a misca omida mai repede tineti apasat tastele cu care schimbati directia. Cat tineti apasat tastele respective acele 150ms vor deveni 50ms.

Pentru a face jocul un pic mai complicat daca ajugeti la marginile terenului omida va muri. In multe variante de snake sarpele nu moare daca atinge peretele ci va continua pe partea opusa. Omida noastra poate muri si daca a ajuns la margine si daca s-a lovit de propriul ei corp.

11.3 Pauza

Jocul poate fi oprit temporar, daca apasati tasta ESCAPE. Ca sa reluati jocul apasati din nou tasta ESCAPE. Daca vreti sa iesiti din joc in timp ce sunteti in pauza apasati tasta 'x'.

11.4 Final

Odata ce omida a murit sau ati iesit din joc in timpul unei pauze veti fi intampinat de un nou meniu care va afisa scorul obtinut si scorul maxim de pana acum si cine il detine. Daca ati depasit scorul maxim vi se va cere numele pentru a retine noul maxim. Pentru a reveni la meniul principal apasati enter.

11.5 Desenand pe ecran

În primul rând pentru a putea desena orice pe ecran în C++ (și nu numai) trebuie să creăm o fereastră. Procesul de a crea o fereastră este specific fiecărui sistem de operare. În Windows avem la dispoziție funcția `CreateWindowEx`, în lumea Linux poate fi creată cu ajutorul `Xlib/XCB` sau `Wayland`, iar pe Mac OS o variantă ar fi `Cocoa` (deși nu este nativ C++ poate fi creat un wrapper cu ajutorul `llvm-clang`). Jocul meu suportă toate aceste platforme (poate chiar și Android și iOS) și totuși nu trebuie să conțină cod pentru fiecare platformă în parte. Asta datorită faptului că folosește o bibliotecă numită `SDL` care a făcut deja așa ceva. Biblioteca este inclusă în proiect. În plus îmi oferă și funcții pentru a desena imagini și figuri geometrice simple. Dacă nu m-aș fi folosit de o astfel de bibliotecă ar fi trebuit să folosesc `opengl` care este suportat pe multe platforme (dar de ceva timp nu mai este suportat pe iOS și Mac OS) sau să recurg la mai mult cod care depinde de platforme (`direct3d` - Windows, `Metal/Quartz` - MacOS/iOS etc.).

Codul care deschide o fereastră se găsește în funcția [initializeaza](#). Prima dată chem funcția `SDL_Init` care va returna o valoare negativă dacă va eșua, apoi creez efectiv fereastra cu ajutorul `SDL_CreateWindow`. Primul parametru este titlul ferestrei, al doilea și al treilea reprezintă coordonatele unde va fi afișată fereastra, al patrulea și al cincilea reprezintă lungimea și înălțimea ferestrei, iar ultimul nu este atât de important.

Odată ce am deschis fereastra trebuie să "activăm" posibilitatea de a desena pe ea. Variabila [global::desenator](#) este un pointer către un obiect de tip `SDL_Renderer`, iar pentru a crea efectiv desenatorul chem funcția `SDL_CreateRenderer` care ia ca parametrii fereastra pe care vreau să desenez și anumite "steaguri" pe care vreau să le setez. În cazul meu setez `SDL_RENDERER_ACCELERATED` care va folosi accelerarea hardware dacă va putea, ceea ce va îmbunătăți performanța considerabil.

Mai jos chem `TTF_Init` și `Mix_OpenAudio` care îmi oferă posibilitatea de a desena text pe ecran și de a reda sunet.

11.6 Cum reprezint obiectele jocului?

Prima data vom avea nevoie de un mod de a reprezenta omida. Cel mai usor ar fi sa folosim un vector care tine pozitii ale omizii, primul/ultimul element fiind capul omizii. Eu folosesc ceva similar dar care imi ofera mult mai multa flexibilitate si care este si mai rapid si consuma mai putina memorie, si anume `std::deque`. Deque inseamna "double ended queue" in engleza si care este un fel de coada dar care este foarte buna la insertia la inceput si la sfarsit - exact ce imi trebuie. In spate deque este o lista de vectori cu o marime fixa(de exemplu 5) care vrea sa aiba avantajele unui vector de a avea memorie continua si foarte rapid de accesat dar si avantajele unei liste de a fi foarte eficienta la insertia in mijlocul unui sir.

Omida este stocata in variabila `StadiulJocului::pozitii_omida`. Tipul de date folosit pentru a stoca efectiv pozitii este `Punct`. De remarcat este faptul ca retin pozitile unei matrici(care reprezinta terenul) `StadiulJocului::teren`, nu valorile absolute in fereastra. Acest lucru face mult mai usoara logica jocului.

Cand jocul incepe sunt inserate 3 pozitii in omida. Cand este timpul ca pozitia omizii sa fie actualizata chem `Joc::muta_omida`. La fiecare pas chem `deseneaza_omida`, care parcurge `StadiulJocului::pozitii_omida` si deseneaza fiecare parte a omizii pe ecran.

Pentru a usura munca de a schimba directia omizii folosesc o enumeratie `TipDirectie`. Apoi am un vector `directie` care va tine valorile cu care trebuie sa incrementez i-ul si j-ul omizii pentru a se deplasa. Cand omida se deplaseaza inserez la inceputul cozii noua pozitie a capului in functie de directie. Daca nu a mancat nimic cat s-a deplasat atunci sterg ultima pozitie din coada.

Pozitia frunzei este retinuta in variabila `StadiulJocului::pozitie_frunza`. Cand omida mananca o frunza trebuie generata o noua pozitie, astfel chem functia `genereaza_pozitie_noua`. Aceeasi functie este chemata si atunci cand omida mananca un mar.

Daca jucatorul apasa tasta escape atunci chem functia `Joc::pauza`.

11.7 Meniuri

Jocul in sine nu este prea complicat. Snake este un joc arhipopular care are multe implementari. Totusi, problema intervine cand vrem sa avem si alte "auxiliare" pentru joc, cum ar fi un meniu de start si final. Cum putem tine separat codul pentru a desena meniuri si cel care tine de jocul efectiv?

O metoda naiva ar fi sa verificam pur si simplu prin niste if-uri unde ne aflam. Ceva de genul:

```
if(este_in_meniu_start) { /* ... */ }  
else if(este_in_joc) { /* .. */ }  
/* etc */
```

Metoda naiva functioneaza dar este foarte limitata(si urata). Jocurile mult mai complexe care trebuie sa deseneze un meniu separat numai pentru optiunile jocului in timp ce are alte submeniuri nu poate functiona asa. Atunci am implementat conceptul de scene.

O scena este practic un joc intreg. Meniul de start are propria scena, jocul in sine are propria scena, meniul de final alta scena, etc. Mai exact, fiecare scena este o clasa care trebuie sa mosteneasca [Scena](#). O scena trebuie sa aibe 3 functii implementate:

- incarca
- incarcat_deja
- itereaza

[Scena::incarca](#) face o initializare a scenei, de exemplu in scena [Joc](#) functia [Joc::incarca](#) incarca toate imaginile pentru omida, frunza, mar, initializeaza matricea jocului etc.

[Scena::incarcat_deja](#) este evidenta.

[Scena::itereaza](#) este apelata la fiecare pas in [Principala repetitie](#).

Scenele sunt stocate in [global::scene](#) intr-o coada, la fel ca [StadiulJocului::pozitii_omida](#) sub forma unor pointeri. [Scena](#) este o clasa abstracta deci nu poate fi create direct, decat prin mostenitori, deci trebuie folositi pointeri. Astfel avem un sistem flexibil prin care putem reprezenta scene. Vedeti cum sunt folosite exact in [Principala repetitie](#).

11.8 Principala repetitie

In while-ul din functia [main](#) se intampla practic tot. Inainte de while sunt inserate doua obiecte de tip [Joc](#) si [Meniu↔Start](#). Scenele sunt folosite ca un fel de stiva, ultima scena fiind cea care trebuie redată. De aceea este inserata prima data scena jocului. Cand vom iesi din meniul de start va fi sters si va ramane scena jocului. Orice scena poate iesi complet din aplicatie daca seteaza [global::fereastra_inchisa](#) la fals. O scena va fi stearsa numai daca a fost initializata, si va fi initializata numai daca nu s-a intamplat deja.

Capitolul 12

Documentația Namespace-ului

12.1 Referință la Namespace-ul global

Variabile

- `SDL_Window * fereastra { nullptr }`
Fereastra in care vom desena tot.
- `SDL_Renderer * desenator { nullptr }`
- `bool fereastra_inchisa = false`
- `constexpr int lungime = 600`
Lungimea ferestrei.
- `constexpr int inaltime = 600`
Inaltimea ferestrei.
- `std::deque< std::unique_ptr< Scena > > scene`
- `std::map< SDL_Scancode, bool > taste_apasate`

12.1.1 Descriere Detaliată

Contine variabile de care putem avea nevoie oricand, indiferent de stadiul jocului.

12.1.2 Documentația variabilelor

12.1.2.1 desenator

```
SDL_Renderer* global::desenator { nullptr }
```

Nu putem desena direct in fereastra dar SDL ne ofera posibilitatea sa creem un 'desenator' care va desena efectiv figuri geometrice/texturi.

Definiția în linia 387 a fișierului main.cpp.

Semnalat de `deseneaza_textura()`, `Joc::incarca()`, `MeniuStart::incarca()`, `MeniuFinal::incarca()`, `initializeaza()`, `MeniuStart::itereaza()`, `main()` și `Joc::pune_scor()`.

12.1.2.2 fereastră_inchisa

```
bool global::fereastră_inchisa = false
```

Devine adevărat când jucătorul închide fereastra de tot.

Definiția în linia 392 a fișierului main.cpp.

Semnalat de `Joc::iesire()`, `MeniuStart::itereaza()`, `main()` și `verifica_evenimente()`.

12.1.2.3 scene

```
std::deque<std::unique_ptr<Scena> > global::scene
```

Toate scenele sunt pastrate în acest vector. Clasa `Scena` oferă o interfață comună pe care orice scenă (care e de fapt o altă aplicație, de exemplu meniul de început/pauză) trebuie să o urmeze. Utilizarea arată cam așa:

```
while(!global::fereastră_inchisa) {  
    global::scene.back()->incarca();  
  
    while(global::scene.back()->itereaza()) {}  
  
    global::scene.back()->iesire();  
}
```

Definiția în linia 418 a fișierului main.cpp.

Semnalat de `Joc::iesire()`, `MeniuFinal::itereaza()` și `main()`.

12.1.2.4 taste_apasate

```
std::map<SDL_Scancode, bool> global::taste_apasate
```

De fiecare dată când jucătorul apasă o tastă, aceasta este reținută în această mapă. Mapa este un fel de vector dar care poate avea altfel de indexare. În cazul acesta, ea poate referi la tasta `escape` așa:

```
if(global::taste_apasate[SDL_SCANCODE_ESCAPE]) {  
    // a apasat escape  
}
```

Definiția în linia 429 a fișierului main.cpp.

Semnalat de `a_apasat()`, `MeniuFinal::itereaza()`, `main()`, `Joc::pauza()` și `verifica_evenimente()`.

Capitolul 13

Documentația Claselor

13.1 Referință la clasa Joc

[Scena](#) jocului in sine.

Diagrama de relații pentru Joc

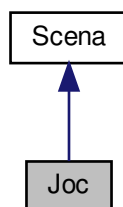
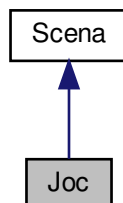


Diagrama de relații pentru Joc:



Metode Publice

- virtual void [iesire](#) ([StadiulJocului](#) &) override
Paraseste scena.
- virtual void [incarca](#) ([StadiulJocului](#) &) override
- virtual bool [incarcata_deja](#) () override
- virtual bool [itereaza](#) ([StadiulJocului](#) &) override
Executa functia la fiecare 'desenare'.

Metode Private

- bool [muta_omida](#) ([StadiulJocului](#) &)
- bool [pauza](#) ([StadiulJocului](#) &)
- void [pune_scor](#) ([StadiulJocului](#) &)
Printeaza scorul curent pe capul omidei.
- void [reseteaza_teren](#) ([StadiulJocului](#) &)
Marcheaza totul PosibilitateTeren::GOL.
- bool [verifica_taste_apasate](#) ([StadiulJocului](#) &)
Schimba StadiulJocului::noua_directie_a_omizii in functie de tastele apasate.

Atribute Private

- long long [m_timp_asteptat](#) = 0LL
- [TipDirectie](#) [noua_directie_a_omizii](#) = TipDirectie::SUS
- bool [m_incarnat_deja](#) = false
- TTF_Font * [m_font](#) = nullptr
Fontul folosit pentru a desena text pe ecran.
- Mix_Chunk * [m_sunet_frunza](#) = nullptr
Sunetul redat de fiecare data cand omida mananca o frunza.
- Mix_Chunk * [m_sunet_powerup](#) = nullptr
Sunetul redat de fiecare data cand omida mananca un powerup.
- long long [m_timp_powerup](#) = 0LL

13.1.1 Descriere Detaliată

[Scena](#) jocului in sine.

Definiția în linia 558 a fișierului main.cpp.

13.1.2 Documentația Funcțiilor Membre

13.1.2.1 incarca()

```
void Joc::incarca (
    StadiulJocului & ) [override], [virtual]
```

Initializeza scena.

Implementează [Scena](#).

Definiția în linia 1128 a fișierului main.cpp.

Referințe `global::desenator` și `StadiulJocului::directia_omizii`.

13.1.2.2 incarcat_deja()

```
bool Joc::incarcat_deja ( ) [override], [virtual]
```

Verifica daca scena curenta a fost incarcata deja.

Implementează [Scena](#).

Definiția în linia 1197 a fișierului main.cpp.

13.1.2.3 itereaza()

```
bool Joc::itereaza (
    StadiulJocului & ) [override], [virtual]
```

Executa functia la fiecare 'desenare'.

Întoarce

Adevarat daca iterarea va continua, fals altfel.

Implementează [Scena](#).

Definiția în linia 1200 a fișierului main.cpp.

Referințe `StadiulJocului::directia_omizii`, `StadiulJocului::exista_powerup`, `genereaza_pozitie_noua()`, `Punct::i`, `Punct::j`, `StadiulJocului::pozitie_powerup`, `StadiulJocului::teren`, `StadiulJocului::timp_asteptare_maxim` și `StadiulJocului::timp_trecut`.

13.1.2.4 muta_omida()

```
bool Joc::muta_omida (
    StadiulJocului & stadiu ) [private]
```

Muta omida in directia setata de jucator.

Întoarce

Adevarat daca omida nu a murit, fals altfel.

Definiția în linia 992 a fișierului main.cpp.

Referințe StadiulJocului::directia_omizii, Punct::i, StadiulJocului::inaltime_teren, Punct::j, StadiulJocului::lungime_↵
_teren, StadiulJocului::pozitii_omida și StadiulJocului::teren.

13.1.2.5 pauza()

```
bool Joc::pauza (
    StadiulJocului & ) [private]
```

Ruleaza cat timp jocul e in pauza.

Întoarce

Fals daca jucatorul a iesit din joc in timpul pauzei sau adevarat altfel.

Definiția în linia 1047 a fișierului main.cpp.

Referințe a_apasat(), global::taste_apasate și verifica_evenimente().

13.1.2.6 verifica_taste_apasate()

```
bool Joc::verifica_taste_apasate (
    StadiulJocului & stadiu ) [private]
```

Schimba StadiulJocului::noua_directie_a_omizii in functie de tastele apasate.

Întoarce

Fals daca jocul nu mai trebuie sa continue.

Definiția în linia 1097 a fișierului main.cpp.

Referințe a_apasat() și StadiulJocului::directia_omizii.

13.1.3 Documentația Datelor Membre

13.1.3.1 m_timp_asteptat

```
long long Joc::m_timp_asteptat = 0LL [private]
```

Cand depaseste `StadiulJocului::timp_asteptare_maxim` omida trebuie mutata.

Definiția în linia 563 a fișierului main.cpp.

13.1.3.2 noua_directie_a_omizii

```
TipDirectie Joc::noua_directie_a_omizii = TipDirectie::SUS [private]
```

Pana cand omida trebuie mutata actualizeaza directia. De exemplu, jucatorul apasa tasta UP, dar dupa apasa D↵OWN fara ca `Joc::m_timp_asteptat` sa fi depasit `StadiulJocului::timp_asteptare_maxim`. Cand omida trebuie mutata actualizeaza `StadiulJocului::directia_omizii` cu directia noua.

Definiția în linia 571 a fișierului main.cpp.

Documentația pentru această clasă a fost generată din fișierul:

- [main.cpp](#)

13.2 Referință la clasa MeniuFinal

Meniul care va fi afisat dupa terminarea unui joc.

Diagrama de relații pentru MeniuFinal

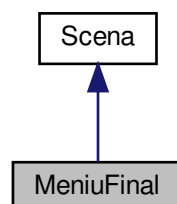
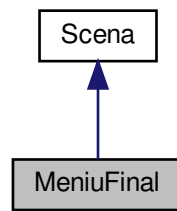


Diagrama de relații pentru MeniuFinal:



Metode Publice

- virtual void **iesire** (**StadiulJocului** &) override
Paraseste scena.
- virtual void **incarca** (**StadiulJocului** &) override
- virtual bool **incarcata_deja** () override
- virtual bool **itereaza** (**StadiulJocului** &) override
Executa functia la fiecare 'desenare'.

Metode Private

- void **citeste_istoric** ()

Atribute Private

- bool **m_incarcat_deja** = false
- TTF_Font * **m_font** = nullptr
- SDL_Texture * **m_text** = nullptr
- SDL_Texture * **m_text_scor_maxim** = nullptr
- std::ifstream **m_istoric** {}
Fisierul in care sunt stocate toate scorurile.
- int **m_scor_maxim** { 0 }
- std::string **m_numele_scorului_maxim** { "" }
- std::string **m_anul_scorului_maxim** { "" }
- std::string **m_luna_scorului_maxim** { "" }
- std::string **m_ziua_scorului_maxim** { "" }
- bool **m_are_scor_maxim** { false }
- std::string **m_nume_curent** { "NUME" }
- SDL_Texture * **m_text_nume_curent** { nullptr }

13.2.1 Descriere Detaliată

Meniul care va fi afisat dupa terminarea unui joc.

Definiția în linia 668 a fișierului main.cpp.

13.2.2 Documentația Funcțiilor Membre

13.2.2.1 incarca()

```
void MeniuFinal::incarca (
    StadiulJocului & ) [override], [virtual]
```

Initializeza scena.

Implementează [Scena](#).

Definiția în linia 1436 a fișierului main.cpp.

Referințe `global::desenator` și `StadiulJocului::scor`.

13.2.2.2 incarcat_deja()

```
bool MeniuFinal::incarcat_deja ( ) [override], [virtual]
```

Verifica daca scena curenta a fost incarcata deja.

Implementează [Scena](#).

Definiția în linia 1498 a fișierului main.cpp.

13.2.2.3 itereaza()

```
bool MeniuFinal::itereaza (
    StadiulJocului & ) [override], [virtual]
```

Executa functia la fiecare 'desenare'.

Întoarce

Adevarat daca iterarea va continua, fals altfel.

Implementează [Scena](#).

Definiția în linia 1528 a fișierului main.cpp.

Referințe `a_apasat()`, `global::scene` și `global::taste_apasate`.

13.2.3 Documentația Datelor Membre

13.2.3.1 m_istoric

```
std::ifstream MeniuFinal::m_istoric {} [private]
```

Fisierul in care sunt stocate toate scorurile.

Istoricul are forma: Nume Jucator Luna zi An Scor

Definiția în linia 684 a fișierului main.cpp.

Documentația pentru această clasă a fost generată din fișierul:

- [main.cpp](#)

13.3 Referință la clasa MeniuStart

Defineste meniul afisat la deschiderea jocului.

Diagrama de relații pentru MeniuStart

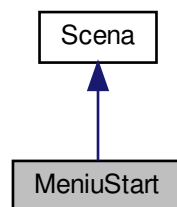
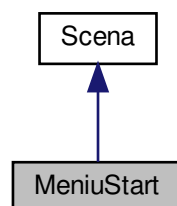


Diagrama de relații pentru MeniuStart:



Metode Publice

- virtual void [iesire](#) ([StadiulJocului](#) &) override
Paraseste scena.
- virtual void [incarca](#) ([StadiulJocului](#) &) override
- virtual bool [incarcata_deja](#) () override
- virtual bool [itereaza](#) ([StadiulJocului](#) &) override
Executa functia la fiecare 'desenare'.

Tipuri Private

- enum **tip_meniu** { **START_JOC** = 0, **IESIRE**, **NR_MENIURI** }

Metode Private

- bool [click_in_meniu](#) ()
- bool [verifica_evenimente](#) ([StadiulJocului](#) &)

Atribute Private

- bool **m_incarcat** = false
- tip_meniu **m_meniu_selectat** = tip_meniu::START_JOC
- SDL_Rect **m_dreptunghi_meniu** [NR_MENIURI]
- SDL_Texture * **m_textura_meniu** [NR_MENIURI]
- TTF_Font * **m_font** = nullptr
- const char * **m_meniu_text** [NR_MENIURI]
- const SDL_Color **m_culoare_background_meniu** = { 153, 51, 153, 255 }

13.3.1 Descriere Detaliată

Defineste meniul afisat la deschiderea jocului.

Definiția în linia 631 a fișierului main.cpp.

13.3.2 Documentația Funcțiilor Membre

13.3.2.1 incarca()

```
void MeniuStart::incarca (
    StadiulJocului & ) [override], [virtual]
```

Initializeza scena.

Implementează [Scena](#).

Definiția în linia 1304 a fișierului main.cpp.

Referințe `global::desenator`, `Punct::i`, `global::inaltime` și `global::lungime`.

13.3.2.2 incarcat_deja()

```
bool MeniuStart::incarcat_deja ( ) [override], [virtual]
```

Verifica daca scena curenta a fost incarcata deja.

Implementează [Scena](#).

Definiția în linia 1335 a fișierului main.cpp.

13.3.2.3 itereaza()

```
bool MeniuStart::itereaza (
    StadiulJocului & ) [override], [virtual]
```

Executa functia la fiecare 'desenare'.

Întoarce

Adevarat daca iterarea va continua, fals altfel.

Implementează [Scena](#).

Definiția în linia 1338 a fișierului main.cpp.

Referințe [a_apasat\(\)](#), [global::desenator](#), [global::fereastra_inchisa](#), [Punct::i](#), [StadiulJocului::inaltime_textura](#), [StadiulJocului::lungime_textura](#) și [verifica_evenimente\(\)](#).

Documentația pentru această clasă a fost generată din fișierul:

- [main.cpp](#)

13.4 Referință la structura Punct

Metode Publice

- [Punct](#) ()=default
- [Punct](#) (int t_i, int t_j)
- [Punct](#) (const [Punct](#) &)=default

Atribute Publice

- int [i](#) { 0 }
Linia in matrice.
 - int [j](#) { 0 }
Coloana in matrice.
-

13.4.1 Descriere Detaliată

O structura ajutatoare pentru a reprezenta o pozitie in matricea jocului.

Definiția în linia 289 a fișierului main.cpp.

13.4.2 Documentația pentru Constructori și Destructori

13.4.2.1 Punct() [1/3]

```
Punct::Punct ( ) [default]
```

Vrem ca o variabila de tipul **Punct** sa aiba valorile initiale $i = 0$ si $j = 0$ asa ca nu ar trebui modificat nimic in constructor.

Exemplu:

```
Punct a;  
a.i == 0; // adevarat  
a.j == 0; // adevarat
```

Semnalat de Punct().

13.4.2.2 Punct() [2/3]

```
Punct::Punct (  
    int t_i,  
    int t_j ) [inline]
```

Putem declara un punct si cu valori customizate pe loc, exemplu:

```
Punct a{ 2, 3 }; // a.i va fi 2 si a.j va fi 3  
Punct b(2, 3); // la fel ca a
```

Definiția în linia 319 a fișierului main.cpp.

Referințe j și Punct().

13.4.2.3 Punct() [3/3]

```
Punct::Punct (
    const Punct & ) [default]
```

Declarat pentru a putea copia valorile unui punct in alta variabila, exemplu:

```
Punct a{ 2, 3 };
Punct b{ 5, 6 };
b = a; // acum b.i = 2 si b.j = 3
```

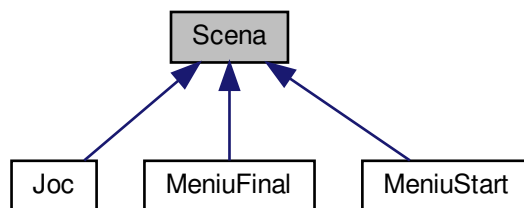
Documentația pentru această structură a fost generată din fișierul:

- [main.cpp](#)

13.5 Referință la clasa Scena

Interfata pentru toate scenele.

Diagrama de relații pentru Scena



Metode Publice

- virtual void [incarca](#) (StadiulJocului &)=0
- virtual bool [incarcata_deja](#) ()=0
- virtual void [iesire](#) (StadiulJocului &)=0
Paraseste scena.
- virtual bool [itereaza](#) (StadiulJocului &)=0
Executa functia la fiecare 'desenare'.

13.5.1 Descriere Detaliată

Interfata pentru toate scenele.

Definiția în linia 530 a fișierului main.cpp.

13.5.2 Documentația Funcțiilor Membre

13.5.2.1 incarca()

```
virtual void Scena::incarca (
    StadiulJocului & ) [pure virtual]
```

Initializeza scena.

Implementat în [MeniuFinal](#), [MeniuStart](#) și [Joc](#).

13.5.2.2 incarcat_deja()

```
virtual bool Scena::incarcat_deja ( ) [pure virtual]
```

Verifica daca scena curenta a fost incarcata deja.

Implementat în [MeniuFinal](#), [MeniuStart](#) și [Joc](#).

13.5.2.3 itereaza()

```
virtual bool Scena::itereaza (
    StadiulJocului & ) [pure virtual]
```

Executa functia la fiecare 'desenare'.

Întoarce

Adevarat daca iterarea va continua, fals altfel.

Implementat în [MeniuFinal](#), [MeniuStart](#) și [Joc](#).

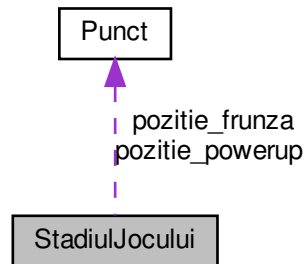
Documentația pentru această clasă a fost generată din fișierul:

- [main.cpp](#)
-

13.6 Referință la clasa StadiulJocului

Contine toata informatia de care avem nevoie in timpul jocului.

Diagrama de relații pentru StadiulJocului:



Atribute Publice

- `PossibilitateTeren teren [inaltime_teren][lungime_teren]`

Matricea jocului.

- `std::deque< Punct > pozitii_omida`

Coadă care conține pozițiile omizii.

- `Punct pozitie_frunza`

Unde se afla frunza.

- `Punct pozitie_powerup`

Unde se afla powerup-ul.

- `bool exista_powerup = false`

Adăvarat dacă am generat deja un powerup.

- `int scor = 0`

Scorul jocului.

- `long long timp_trecut = 0`

Timpul care a trecut de la ultima 'desenare'.

- `TipDirectie directia_omizii = TipDirectie::SUS`

În ce direcție se deplasează omida.

- `SDL_Texture * textura_capului { nullptr }`

- `SDL_Texture * textura_corpului { nullptr }`

- `SDL_Texture * textura_frunzei { nullptr }`

- `SDL_Texture * textura_powerup { nullptr }`

- `SDL_Color culoare_fundal = { 15, 15, 15, 255 }`

Atribute Statice Publice

- static constexpr int `lungime_teren` = 20
Lungimea matricii jocului.
- static constexpr int `inaltime_teren` = 20
Inaltimea matricii jocului.
- static constexpr int `lungime_textura` = `global::lungime` / `lungime_teren`
Lungimea unei texturi.
- static constexpr int `inaltime_textura` = `global::inaltime` / `inaltime_teren`
Inaltimea unei texturi.
- static long long `timp_asteptare_maxim` = 150LL
Timpul maxim pana cand omida se va misca.

13.6.1 Descriere Detaliată

Contine toata informatia de care avem nevoie in timpul jocului.

Definiția în linia 435 a fișierului main.cpp.

13.6.2 Documentația Datelor Membre

13.6.2.1 `directia_omizii`

```
TipDirectie StadiulJocului::directia_omizii = TipDirectie::SUS
```

În ce direcție se deplasează omida.

Inițial se deplasează în sus dar se va schimba în funcție de ce taste apasă jucătorul.

Definiția în linia 515 a fișierului main.cpp.

Semnalat de `Joc::incarca()`, `Joc::itereaza()`, `Joc::muta_omida()` și `Joc::verifica_taste_apasate()`.

13.6.2.2 `lungime_textura`

```
constexpr int StadiulJocului::lungime_textura = global::lungime / lungime_teren [static]
```

Lungimea unei texturi.

În funcție de dimensiunea matricii jocului lungimea și înălțimea texturilor se schimbă. În cazul nostru o textură nu e altceva decât o imagine încărcată și desenată în fereastră.

Definiția în linia 454 a fișierului main.cpp.

Semnalat de `deseneaza_frunza()`, `deseneaza_omida()`, `deseneaza_powerup()`, `deseneaza_textura()`, `MeniuStart::itereaza()` și `Joc::pune_scor()`.

13.6.2.3 pozitie_powerup

```
Punct StadiulJocului::pozitie_powerup
```

Unde se afla powerup-ul.

Un Powerup este un mar, mancare mult mai consistenta pentru omida.

Definiția în linia 487 a fișierului main.cpp.

Semnalat de `deseneaza_powerup()` și `Joc::itereaza()`.

13.6.2.4 pozitii_omida

```
std::deque<Punct> StadiulJocului::pozitii_omida
```

Coadă care conține pozițiile omizii.

Este o coadă în care se poate insera și la început și la final. Prin pozițiile omizii se înțelege unde se afla cercurile cu care desenez omida. Omida va fi reprezentată cu cercuri, capul este un cerc roșu, restul cercuri albastre. Astfel parcurg coada pentru a determina unde trebuie să desenez cercurile. `pozitii_omida.front()` va fi capul omizii.

Definiția în linia 477 a fișierului main.cpp.

Semnalat de `deseneaza_omida()`, `Joc::muta_omida()` și `Joc::pune_scor()`.

13.6.2.5 scor

```
int StadiulJocului::scor = 0
```

Scorul jocului.

Pentru fiecare frunză mâncată scorul crește cu 1.

Definiția în linia 497 a fișierului main.cpp.

Semnalat de `MeniuFinal::iesire()`, `MeniuFinal::incarca()` și `Joc::pune_scor()`.

13.6.2.6 teren

```
PosibilitateTeren StadiulJocului::teren[inaltime_teren][lungime_teren]
```

Matricea jocului.

Matricea reprezintă terenul pe care se deplasează omida. Valorile din matrice pot fi una dinre [PosibilitateTeren](#). Omida este reprezentată ca o coadă care conține [Punct](#), o coadă în care se poate insera și la început și la final: `pozitii_omida`.

Definiția în linia 467 a fișierului main.cpp.

Semnalat de `genereaza_pozitie_noua()`, `Joc::itereaza()`, `marcheaza_frunza()`, `Joc::muta_omida()` și `Joc::reseteaza_teren()`.

Documentația pentru această clasă a fost generată din fișierul:

- [main.cpp](#)

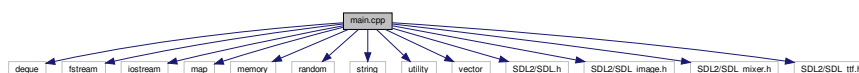
Capitolul 14

Documentația Fișierelor

14.1 Referință la fișierul main.cpp

```
#include <deque>
#include <fstream>
#include <iostream>
#include <map>
#include <memory>
#include <random>
#include <string>
#include <utility>
#include <vector>
#include "SDL2/SDL.h"
#include "SDL2/SDL_image.h"
#include "SDL2/SDL_mixer.h"
#include "SDL2/SDL_ttf.h"
```

Graful dependențelor prin incluziune pentru main.cpp:



Membri

- struct [Punct](#)
- class [StadiuJocului](#)
Contine toata informatia de care avem nevoie in timpul jocului.
- class [Scena](#)
Interfata pentru toate scenele.
- class [Joc](#)
Scena jocului in sine.
- class [MeniuStart](#)
Defineste meniul afisat la deschiderea jocului.
- class [MeniuFinal](#)
Meniul care va fi afisat dupa terminarea unui joc.

Namespace-uri

- `global`

Enumerări

- enum `TipDirectie` { **SUS** = 0, **JOS** = 1, **STANGA** = 2, **DREAPTA** = 3 }
Unde se poate deplasa omida.
- enum `PosibilitateTeren` { **GOL** = 0, **OMIDA**, **FRUNZA**, **POWERUP** }

Funcții

- bool `a_apasat` (SDL_Scancode tasta)
Verifica daca jucatorul a apasat tasta data.
- void `deseneaza_cap` (StadiulJocului &, int x, int y)
Deseneaza capul omizii la pozitia data.
- void `deseneaza_corp` (StadiulJocului &, int, int)
Deseneaza un corpul la pozitia data.
- void `deseneaza_frunza` (StadiulJocului &)
Deseneaza frunza.
- void `deseneaza_omida` (StadiulJocului &)
Deseneaza intreaga omida.
- void `deseneaza_powerup` (StadiulJocului &)
Deseneaza powerup-ul(un mar) pe ecran.
- void `deseneaza_textura` (SDL_Texture *textura, int, int)
Pune textura data la pozitia data.
- `Punct genereaza_pozitie_noua` (StadiulJocului &stadiu)
- void `initializeaza` ()
- void `marcheaza_frunza` (StadiulJocului &)
- void `verifica_evenimente` ()
Vede ce taste a apasat jucatorul si orice alte evenimente.
- int `main` ()
- bool `e_litera` (SDL_Scancode t_tasta)
- bool `e_numar` (SDL_Scancode t_tasta)

Variabile

- `Punct directie` []
- SDL_Window * `global::fereastra` { nullptr }
Fereastra in care vom desena tot.
- SDL_Renderer * `global::desenator` { nullptr }
- bool `global::fereastra_inchisa` = false
- constexpr int `global::lungime` = 600
Lungimea ferestrei.
- constexpr int `global::inaltime` = 600
Inaltimea ferestrei.
- std::deque< std::unique_ptr< `Scena` > > `global::scene`
- std::map< SDL_Scancode, bool > `global::taste_apasate`

14.1.1 Documentația enumerărilor

14.1.1.1 PosibilitateTeren

```
enum PosibilitateTeren
```

Terenul este o matrice care contine una din urmatoarele posibilitati de mai jos.

Definiția în linia 365 a fișierului main.cpp.

14.1.2 Documentația funcțiilor

14.1.2.1 deseneaza_cap()

```
void deseneaza_cap (
    StadiulJocului & stadiu,
    int x,
    int y )
```

Deseneaza capul omizii la pozitia data.

Parametri

in	x	Coordonata lungime ca in plan cartezian.
in	y	Inaltimea, coordonata (0, 0) este in coltul din stanga sus al ferestrei.

Atenție

Pozitia data nu are aceleasi coordonate ca [Punct](#). x si y se refera la pozitia in fereastra, nu in matrice. Se aplica si la [deseneaza_corp](#), [deseneaza_textura](#).

Definiția în linia 837 a fișierului main.cpp.

Referințe [deseneaza_textura\(\)](#).

Semnalat de [deseneaza_omida\(\)](#).

14.1.2.2 deseneaza_omida()

```
void deseneaza_omida (
    StadiulJocului & stadiu )
```

Deseneaza intreaga omida.

Parcurge `StadiulJocului::pozitii_omida` si deseneaza fiecare parte a omizii.

Definiția în linia 853 a fișierului `main.cpp`.

Referințe `deseneaza_cap()`, `deseneaza_corp()`, `Punct::i`, `StadiulJocului::inaltime_textura`, `Punct::j`, `StadiulJocului::lungime_textura` și `StadiulJocului::pozitii_omida`.

14.1.2.3 genereaza_pozitie_noua()

```
Punct genereaza_pozitie_noua (
    StadiulJocului & stadiu )
```

Returneaza un punct la nimereala cu conditia ca in acea pozitie in matricea nu fie nimic.

Definiția în linia 892 a fișierului `main.cpp`.

Referințe `Punct::i`, `StadiulJocului::inaltime_teren`, `Punct::j`, `StadiulJocului::lungime_teren` și `StadiulJocului::teren`.

Semnalat de `Joc::itereaza()`.

14.1.2.4 initializeaza()

```
void initializeaza ( )
```

Creeaza fereastra si desenatorul.

Definiția în linia 912 a fișierului `main.cpp`.

Referințe `global::desenator`, `global::fereastra`, `global::inaltime` și `global::lungime`.

Semnalat de `main()`.

14.1.2.5 main()

```
int main ( )
```

Toata magia se intampla aici.

Definiția în linia 768 a fișierului `main.cpp`.

Referințe `global::desenator`, `global::fereastra_inchisa`, `initializeaza()`, `global::scene`, `global::taste_apasate`, `StadiulJocului::timp_trecut` și `verifica_evenimente()`.

14.1.2.6 marcheaza_frunza()

```
void marcheaza_frunza (
    StadiulJocului & stadiu )
```

Pune în matricea jocului pe poziția `StadiulJocului::pozitie_frunza` PosibilitateTeren::FRUNZA.

Definiția în linia 942 a fișierului main.cpp.

Referințe Punct::i, Punct::j, StadiulJocului::pozitie_frunza și StadiulJocului::teren.

14.1.3 Documentația variabilelor

14.1.3.1 directie

```
Punct directie[]
```

Valoarea inițială:

```
= {
    { -1, 0 },
    { 1, 0 },
    { 0, -1 },
    { 0, 1 }
}
```

Pentru a face mutarea omizii mai ușoară pozițiile capului și cozii vor fi actualizate cu pozițiile lor curente + `directie[STANGA]` sau `DREAPTA` etc.

```
Punct cap{ 2, 3 };
// Pentru a muta la stanga:
cap.i += directie[STANGA].i;
cap.j += directie[STANGA].j;
```

Definiția în linia 355 a fișierului main.cpp.

Glosar

Cum reprezint obiectele jocului?, [26](#)

Desenand pe ecran, [25](#)

desenator

global, [29](#)

deseneaza_cap

main.cpp, [49](#)

deseneaza_omida

main.cpp, [49](#)

directia_omizii

StadiulJocului, [45](#)

directie

main.cpp, [51](#)

fereastra_inchisa

global, [29](#)

Final, [24](#)

genereaza_pozitie_noua

main.cpp, [50](#)

global, [29](#)

desenator, [29](#)

fereastra_inchisa, [29](#)

scene, [30](#)

taste_apasate, [30](#)

incarca

Joc, [32](#)

MeniuFinal, [37](#)

MeniuStart, [39](#)

Scena, [43](#)

incarcata_deja

Joc, [33](#)

MeniuFinal, [37](#)

MeniuStart, [39](#)

Scena, [43](#)

initializeaza

main.cpp, [50](#)

itereaza

Joc, [33](#)

MeniuFinal, [37](#)

MeniuStart, [40](#)

Scena, [43](#)

Joc, [31](#)

incarca, [32](#)

incarcata_deja, [33](#)

itereaza, [33](#)

m_timp_asteptat, [35](#)

muta_omida, [33](#)

noua_directie_a_omizii, [35](#)

pauza, [34](#)

verifica_taste_apasate, [34](#)

Jocul efectiv, [22](#)

lungime_textura

StadiulJocului, [45](#)

m_istoric

MeniuFinal, [37](#)

m_timp_asteptat

Joc, [35](#)

main

main.cpp, [50](#)

main.cpp, [47](#)

deseneaza_cap, [49](#)

deseneaza_omida, [49](#)

directie, [51](#)

genereaza_pozitie_noua, [50](#)

initializeaza, [50](#)

main, [50](#)

marcheaza_frunza, [50](#)

PosibilitateTeren, [49](#)

marcheaza_frunza

main.cpp, [50](#)

Meniu start, [21](#)

MeniuFinal, [35](#)

incarca, [37](#)

incarcata_deja, [37](#)

itereaza, [37](#)

m_istoric, [37](#)

MeniuStart, [38](#)

incarca, [39](#)

incarcata_deja, [39](#)

itereaza, [40](#)

Meniuri, [27](#)

muta_omida

Joc, [33](#)

noua_directie_a_omizii

Joc, [35](#)

Pauza, [23](#)

pauza

Joc, [34](#)

PosibilitateTeren

main.cpp, [49](#)

pozitie_powerup

StadiulJocului, [45](#)

pozitii_omida

StadiulJocului, [46](#)
Principala repetitie, [28](#)
Punct, [40](#)
 Punct, [41](#)

Scena, [42](#)
 incarca, [43](#)
 incarcata_deja, [43](#)
 itereaza, [43](#)
scene
 global, [30](#)
scor
 StadiulJocului, [46](#)
StadiulJocului, [44](#)
 directia_omizii, [45](#)
 lungime_textura, [45](#)
 pozitie_powerup, [45](#)
 pozitii_omida, [46](#)
 scor, [46](#)
 teren, [46](#)

taste_apasate
 global, [30](#)
teren
 StadiulJocului, [46](#)

verifica_taste_apasate
 Joc, [34](#)
