Omida

Generat de Doxygen 1.8.13

Contents

Chapter 1

Omida

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

Chapter 2

Cum se joaca?

- Meniu start
- Jocul efectiv
- Pauza
- Final

Chapter 3

Cum functioneaza?

Chapter 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 corpuli 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.

Chapter 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.

Chapter 6

Indexul Modulelor

6.1 Module

Lista tuturor modulelor:

Chapter 7

Indexul Namespace-ului

7.1 Lista de Namespace-uri

Lista tuturor namespace-urilor documentate , cu scurte descrieri:

Chapter 8

Index Ierarhic

8.1 Ierarhia Claselor

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

Chapter 9

Indexul Claselor

9.1 Lista Claselor

Lista claselor, structurilor, uniunilor şi interfeţelor, cu scurte descrieri:

Chapter 10

Indexul Fişierelor

10.1 Lista fişierelor

Lista tuturor fişierelor documentate, cu scurte descrieri:

Chapter 11

Documentaţia 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::STA↩NGA, 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 schimbari 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 propiul 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

In primul rand pentru a putea desena orice pe ecran in c++(si nu numai) trebuie sa creem o fereastra. Procesul de a crea o fereastra este specific fiecraui sistem de operare. In windows avem la dispozitie functia CreateWindowEx, in lumea Linux poate fi creata cu ajutorul Xlib/XCB sau Wayland, iar pe Mac OS o varianta ar fi Cocoa(desi nu este nativ c++ poate fi creat un wrapper cu ajutorul llvm-clang). Jocul meu suporta toate aceste platforme(poate chiar si android si ios) si totusi nu trebuie sa contina cod pentru fiecare platforma in parte. Asta datorita faptului ca foloseste o librarie numita SDL care a facut deja asa ceva. Libraria este inclusa in proiect. In plus imi ofera si functii pentru a desena imagini si figuri geometrice simple. Daca nu m-as fi folosit de o astfel de librarie ar fi trebuit sa folosesc opengl care este suportat pe multe platforme(dar de ceva timp nu mai este suportat pe ios is Mac OS) sau sa recurg la mai mult cod care depinde de platforme(direct3d - windows, Metal/Quartz - MacOs/ios etc.).

Codul care deschide o fereastra se gaseste in functia initializeaza. Prima data chem functia SDL_Init care va returna o valoarea negativa daca va esua, apoi creez efectiv fereastra cu ajutorul SDL_CreateWindow. Primul parametru este titlul ferestrei, al doilea si al treilea reprezinta coordonatele unde va fi afisata fereastra, al patrulea si al cincilea reprezinta lungime si inaltimea ferestrei, iar ultimul nu este atat de important.

Odata ce am deschis fereastra trebuie sa "activam" posibilitatea de a desena pe ea. Variabila global::desenator este un pointer catre un obiect de tip SDL_Renderer, iar pentru a crea efectiv desenatorul chem functia SDL_↩ CreateRenderer care ia ca parametrii fereastra pe care vreau sa desenez si anumite "steaguri" pe care vreau sa le setez. In cazul meu setez SDL_RENDERER_ACCELERATED care va folosi accelerarea hardware daca va putea, ceea ce va imbunatati performanta considerabil.

Mai jos chem TTF_Init si Mix_OpenAudio care imi ofera posibilitatea de a desena text pe ecran si de e 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 pozitiil este Punct. De remarcat este faptul ca retin pozitiile 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 folosescu 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 redata. 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.

Chapter 12

Documentaţia Namespace-ului

12.1 Referinţă 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 Documentaţia 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.

Definiţia în linia 387 a fişierului main.cpp.

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

12.1.2.2 fereastra_inchisa

```
bool global::fereastra_inchisa = false
```

Devine adevarat cand jucatorul inchide fereastra de tot.

Definiţia în linia 392 a fişierului main.cpp.

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

12.1.2.3 scene

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

Toate scenele sunt pastrate in acest vector. Clasa Scena ofera o interfata comuna pe care orice scena(care e de fapt o alta aplicatie, de exemplu meniul de inceput/pauza) trebuie sa o urmeze. Utilizarea arata cam asa:

```
while(!global::fereastra_inchisa) {
    global::scene.back()->incarca();

    while(global::scene.back()->itereaza()) {}

    global::scene.back()->iesire();
}
```

Definiţia în linia 418 a fişierului main.cpp.

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

12.1.2.4 taste_apasate

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

De fiecare data cand jucatorul apasa o tasta aceasta e retinuta in aceasta mapa. Mapa este un fel de vector dar care poate avea altfel de indexare. In cazul acesta ma pot referi la tasta escape asa:

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

Definiţia în linia 429 a fişierului main.cpp.

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

Chapter 13

Documentaţia Claselor

13.1 Referinţă la clasa Joc

Scena jocului in sine.

Diagrama de relaţii pentru Joc

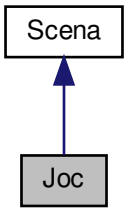
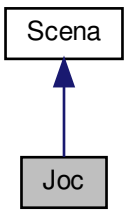


Diagrama de relaţii pentru Joc:



Metode Publice

- virtual void iesire (StadiulJocului &) override

  Paraseste scena.
- virtual void incarca (StadiulJocului &) override
- virtual bool incarcat_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_incarcat_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.

Definiţia în linia 558 a fişierului main.cpp.

13.1.2 Documentaţia Funcţiilor Membre

13.1.2.1 incarca()

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

Initializeza scena.

Implementează Scena.

Definiţia în linia 1128 a fişierului main.cpp.

Referinţe global::desenator şi 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.

Definiţia în linia 1197 a fişierului 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.

Definiţia în linia 1200 a fişierului main.cpp.

Referinţe StadiulJocului::directia_omizii, StadiulJocului::exista_powerup, genereaza_pozitie_noua(), Punct::i, Punct::j, StadiulJocului::pozitie_powerup, StadiulJocului::teren, StadiulJocului::timp_asteptare_maxim şi 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.

Definiţia în linia 992 a fişierului main.cpp.

Referinţe StadiulJocului::directia_omizii, Punct::i, StadiulJocului::inaltime_teren, Punct::j, StadiulJocului::lungime↩
_teren, StadiulJocului::pozitii_omida şi 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.

Definiţia în linia 1047 a fişierului main.cpp.

Referinţe a_apasat(), global::taste_apasate şi 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.

Definiţia în linia 1097 a fişierului main.cpp.

Referinţe a_apasat() şi StadiulJocului::directia_omizii.

13.1.3 Documentaţia 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.

Definiţia în linia 563 a fişierului 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.

Definiţia în linia 571 a fişierului main.cpp.

Documentaţia pentru această clasă a fost generată din fişierul:

- main.cpp

13.2 Referinţă la clasa MeniuFinal

Meniul care va fi afisat dupa terminarea unui joc.

Diagrama de relaţii pentru MeniuFinal

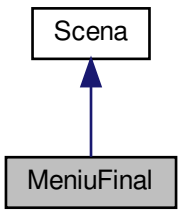
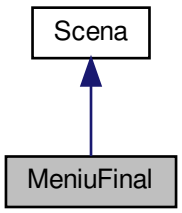


Diagrama de relaţii pentru MeniuFinal:



Metode Publice

- virtual void iesire (StadiulJocului &) override

  Paraseste scena.
- virtual void incarca (StadiulJocului &) override
- virtual bool incarcat_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.

Definiţia în linia 668 a fişierului main.cpp.

13.2.2 Documentaţia Funcţiilor Membre

13.2.2.1 incarca()

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

Initializeza scena.

Implementează Scena.

Definiţia în linia 1436 a fişierului main.cpp.

Referinţe global::desenator şi StadiulJocului::scor.

13.2.2.2 incarcat_deja()

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

Verifica daca scena curenta a fost incarcata deja.

Implementează Scena.

Definiţia în linia 1498 a fişierului 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.

Definiţia în linia 1528 a fişierului main.cpp.

Referinţe a_apasat(), global::scene şi global::taste_apasate.

13.2.3 Documentaţia 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

Definiţia în linia 684 a fişierului main.cpp.

Documentaţia pentru această clasă a fost generată din fişierul:

- main.cpp

13.3 Referinţă la clasa MeniuStart

Defineste meniul afisat la deschiderea jocului.

Diagrama de relaţii pentru MeniuStart

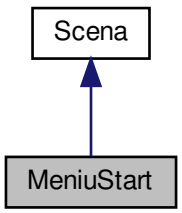
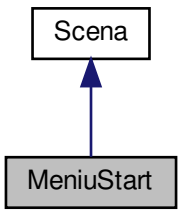


Diagrama de relaţii pentru MeniuStart:



Metode Publice

- virtual void iesire (StadiulJocului &) override

  Paraseste scena.
- virtual void incarca (StadiulJocului &) override
- virtual bool incarcat_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.

Definiţia în linia 631 a fişierului main.cpp.

13.3.2 Documentaţia Funcţiilor Membre

13.3.2.1 incarca()

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

Initializeza scena.

Implementează Scena.

Definiţia în linia 1304 a fişierului main.cpp.

Referinţe global::desenator, Punct::i, global::inaltime şi global::lungime.

13.3.2.2 incarcat_deja()

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

Verifica daca scena curenta a fost incarcata deja.

Implementează Scena.

Definiţia în linia 1335 a fişierului 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.

Definiţia în linia 1338 a fişierului main.cpp.

Referinţe a_apasat(), global::desenator, global::fereastra_inchisa, Punct::i, StadiulJocului::inaltime_textura, StadiulJocului::lungime_textura şi verifica_evenimente().

Documentaţia pentru această clasă a fost generată din fişierul:

- main.cpp

13.4 Referinţă 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.

Definiţia în linia 289 a fişierului main.cpp.

13.4.2 Documentaţia pentru Constructori şi 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
```

Definiţia în linia 319 a fişierului main.cpp.

Referinţe j şi 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
```

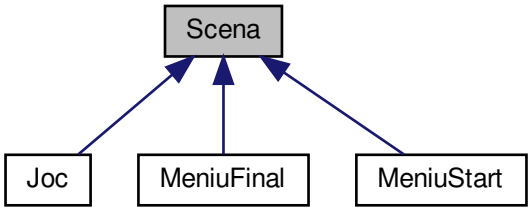
Documentaţia pentru această structură a fost generată din fişierul:

- main.cpp

13.5 Referinţă la clasa Scena

Interfata pentru toate scenele.

Diagrama de relaţii pentru Scena



Metode Publice

- virtual void incarca (StadiulJocului &)=0
- virtual bool incarcat_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.

Definiţia în linia 530 a fişierului main.cpp.

13.5.2 Documentaţia Funcţiilor Membre

13.5.2.1 incarca()

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

Initializeza scena.

Implementat în MeniuFinal, MeniuStart şi 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 şi 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 şi Joc.

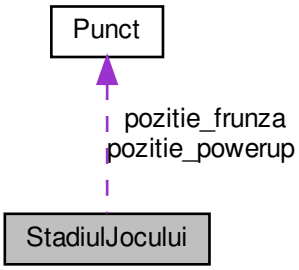
Documentaţia pentru această clasă a fost generată din fişierul:

- main.cpp

13.6 Referinţă la clasa StadiulJocului

Contine toata informatia de care avem nevoie in timpul jocului.

Diagrama de relaţii pentru StadiulJocului:



Atribute Publice

- PosibilitateTeren teren [inaltime_teren][lungime_teren]

  Matricea jocului.
- std::deque< Punct > pozitii_omida

  Coada care contine pozitiile omizii.
- Punct pozitie_frunza

  Unde se afla frunza.
- Punct pozitie_powerup

  Unde se afla powerup-ul.
- bool exista_powerup = false

  Adavarat daca 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

  In ce directie se deplaseaza 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.

Definiţia în linia 435 a fişierului main.cpp.

13.6.2 Documentaţia Datelor Membre

13.6.2.1 directia_omizii

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

In ce directie se deplaseaza omida.

Initial se deplaseaza in sus dar se va schimba in functie de ce taste apasa jucatorul.

Definiţia în linia 515 a fişierului main.cpp.

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

13.6.2.2 lungime_textura

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

Lungimea unei texturi.

In functie de dimensiunea matricii jocului lungimea si inaltimea texturilor se schimba. In cazul nostru o textura nu e altceva decat o imagine incarcata si desenata in fereastra.

Definiţia în linia 454 a fişierului main.cpp.

Semnalat de deseneaza_frunza(), deseneaza_omida(), deseneaza_powerup(), deseneaza_textura(), MeniuStart↩ ::itereaza() şi 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.

Definiţia în linia 487 a fişierului main.cpp.

Semnalat de deseneaza_powerup() şi Joc::itereaza().

13.6.2.4 pozitii_omida

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

Coada care contine pozitiile omizii.

Este o coada in care se poate insera si la inceput si la final. Prin pozitiile omizii se intelege unde se afla cercurile cu care desenez omida. Omida va fi reprezentata cu cercuri, capul este un cerc rosu, restul cercuri albastre. Astfel parcurg coada pentru a determina unde trebuie sa desenez cercurile. pozitii_omida.front() va fi capul omizii.

Definiţia în linia 477 a fişierului main.cpp.

Semnalat de deseneaza_omida(), Joc::muta_omida() şi Joc::pune_scor().

13.6.2.5 scor

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

Scorul jocului.

Pentru fiecare frunza mancate scorul creste cu 1.

Definiţia în linia 497 a fişierului main.cpp.

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

13.6.2.6 teren

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

Matricea jocului.

Matricea reprezinta terenul pe care se deplaseaza omida. Valorile din matrice pot fi una dinre PosibilitateTeren. Omida este reprezentata ca o coada care contine Punct, o coada in care se poate insera si la inceput si la final: pozitii_omida.

Definiţia în linia 467 a fişierului main.cpp.

Semnalat de genereaza_pozitie_noua(), Joc::itereaza(), marcheaza_frunza(), Joc::muta_omida() şi Joc↩ ::reseteaza_teren().

Documentaţia pentru această clasă a fost generată din fişierul:

- main.cpp

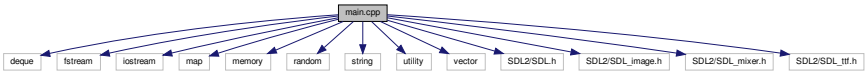
Chapter 14

Documentaţia Fişierelor

14.1 Referinţă la fişierul 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 dependenţelor prin incluziune pentru main.cpp:



Membri

- struct Punct
- class StadiulJocului

  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** }

Funcţii

- 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 Documentaţia enumerărilor

14.1.1.1 PosibilitateTeren

```
enum PosibilitateTeren
```

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

Definiţia în linia 365 a fişierului main.cpp.

14.1.2 Documentaţia funcţiilor

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.

Atenţie

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.

Definiţia în linia 837 a fişierului main.cpp.

Referinţe 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.

Definiţia în linia 853 a fişierului main.cpp.

Referinţe deseneaza_cap(), deseneaza_corp(), Punct::i, StadiulJocului::inaltime_textura, Punct::j, StadiulJocului::lungime_textura şi 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 matricesa nu fie nimic.

Definiţia în linia 892 a fişierului main.cpp.

Referinţe Punct::i, StadiulJocului::inaltime_teren, Punct::j, StadiulJocului::lungime_teren şi StadiulJocului::teren.

Semnalat de Joc::itereaza().

14.1.2.4 initializeaza()

```
void initializeaza ( )
```

Creeaza fereastra si desenatorul.

Definiţia în linia 912 a fişierului main.cpp.

Referinţe global::desenator, global::fereastra, global::inaltime şi global::lungime.

Semnalat de main().

14.1.2.5 main()

```
int main ( )
```

Toata magia se intampla aici.

Definiţia în linia 768 a fişierului main.cpp.

Referinţe global::desenator, global::fereastra_inchisa, initializeaza(), global::scene, global::taste_apasate, StadiulJocului::timp_trecut şi verifica_evenimente().

14.1.2.6 marcheaza_frunza()

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

Pune in matricea jocului pe pozitia StadiulJocului::pozitie_frunza PosibilitateTeren::FRUNZA.

Definiţia în linia 942 a fişierului main.cpp.

Referinţe Punct::i, Punct::j, StadiulJocului::pozitie_frunza şi StadiulJocului::teren.

14.1.3 Documentaţia variabilelor

14.1.3.1 directie

```
Punct directie[]
```

Valoarea iniţială:

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

Pentru a face mutarea omizii mai usoara pozitiile capului si cozii vor fi actualizate cu pozitiile 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;
```

Definiţia în linia 355 a fişierului main.cpp.