Asteroid Invasion – Un joc menit să aducă copilăria mai aproape

Numele si prenumele elevului realizator: Dinu Matei-Alexandru

Tipul lucrarii (sectiunea la care se incadreaza): Soft educational – joc interactiv, notiuni avansate de programare

Titlul lucrarii: AsteroidInvasion

Clasa: a XII-a

Școala de proveniență: Colegiul National de Informatica “Tudor Vianu”

Numele profesorilor indrumatori: Minca Carmen, Lupașcu Oana

Realizat de MatthewAlgo (Dinu Matei-Alexandru)

București, martie 2022

https://github.com/MatthewAlgo/AlienInvadersRetroBuild

- **Ce înseamnă un joc de calitate?**
  Cu toții știm că un joc bun are în spate o poveste la fel de interesantă. Acum avem ocazia să readucem copilăria la viață, deoarece are loc lansarea AsteroidInvasion, un joc scris in C++ ce are rolul de a încuraja evoluția tehnologiei, mai ales în contextul dezvoltării astronomiei, care a ajuns sa acapareze interesul publicului larg. Adoptând un design minimalist, este gata să intre în interesul tinerilor (nu atât de tineri).

- **Care este povestea?**
  Este anul 3042, un an in care tehnologia a devenit atât de dezvoltată încât am ajuns să cunoaștem cum să ne apărăm de intemperiile aduse de spațiul cosmic. Banda de asteroizi dintre Jupiter și Marte s-a destabilizat din cauza unui experiment eșuat de modificare a forței gravitaționale ce a afectat direct planeta Marte, iar asteroizii au fost deviați către Pământ! Este misiunea ta să pleci cu naveta spațială în misiune pentru a-i distruge înainte sa treacă de event horizon-ul Pământului – zona unde un impact devine inevitabil. Asteroizii nu sunt obișnuiți! Pentru a se apăra de eventuale atacuri extraterestre, urmașii lui Elon Musk au plasat pe asteroizi lasere distrugătoare, a căror menire e să gonească invadatorii care vor să colonizeze planeta Jupiter – au considerat că plasarea laserelor pe asteroizi este mai eficientă. De asemenea, pentru a se apăra, asteroizii au implementate motoare ce execută mișcări de autoapărare. Momentan nu mai funcționează atât de bine, ele executând mișcări haotice – Sistemul de control a fost decuplat. Rezistă cât mai mult în fața rocilor spațiale și acumulează un scor cât mai mare!

- **Cum se controlează?**

  Pentru a controla naveta spațială, puteți folosi tastele W, A, S, D, iar pentru a trage cu laserul folosiți tasta SPACE. Pentru a vă deplasa în meniu, folosiți mouse-ul.

- **Cum il fac sa mearga?**

  Pentru a clona repository-ul:

```
PS C:\Users\MatthewAlgo\Projects> git clone https://github.com/MatthewAlgo/AlienInvadersRetroBuild.git -b AsteroidInvasion
```

  Pentru a construi executabilul din surse:

  UNIX:

```
PS C:\Users\MatthewAlgo\Projects\AlienInvadersRetroBuild> bash .\build_release.sh
```

  Windows (cu VSCommunity):

```
PS C:\Users\MatthewAlgo\Projects\AlienInvadersRetroBuild> .\build_release.bat
```

  Pentru a rula executabilul, dupa generare:

  Windows (cu VSCommunity):

```
PS C:\Users\MatthewAlgo\Projects\AlienInvadersRetroBuild> .\build\standalone\Release\AsteroidInvasion.exe
```

  UNIX:

```
PS C:\Users\MatthewAlgo\Projects\AlienInvadersRetroBuild> ./build/standalone/AsteroidInvasion
```

  Prezentarea elementelor de interfață grafică

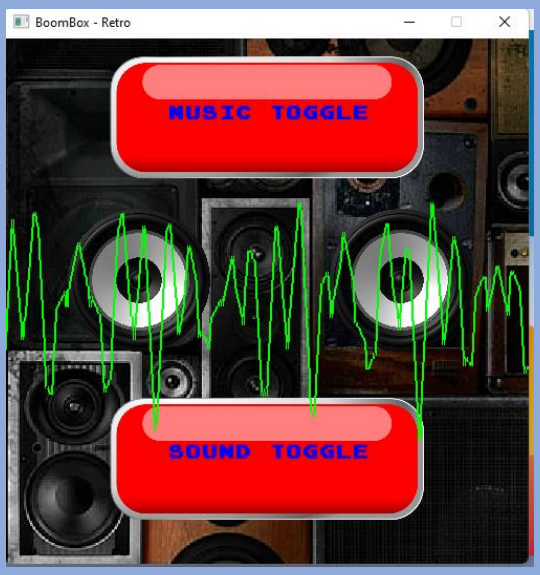
  Interfața grafică a jocului începe cu meniul principal, o fereastră cu rezoluție fixă (1000x500px) – menirea acestuia este să fie dinamic, activ pe tot parcursul executării aplicației. Acesta este punctul de unde pot fi lansate ferestre adiționale:

  - Cea a jocului propriu-zis, o fereastră de rezoluție 1000x700px, cu dimensiune fixă

- O fereastră de control a muzicii – un vizualizator al undei melodiei cu 2 butoane de activare/dezactivare a sunetului/muzicii.

  - **Fereastra principală**

  - **Fereastra de control a sunetului – vizualizatorul de unde**



  - **Fereastra de desfășurare a jocului – La începutul documentației**

Prezentarea elementelor de limbaj, tehnologii folosite, tehnici de programare

Limbajul de programare folosit în proiect, atât pentru randare grafică, cât și pentru algoritmi interni este C++, versiunea 17, ce introduce concepte noi în limbaj față de versiunile precedente, intre care std::optional, std::filesystem, pe care le-am folosit în proiect.

Conform wikipedia.org, C++ este un limbaj de programare general, compilat. Este un limbaj multi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programare procedurală, abstractizare a datelor, programare orientată pe obiecte. În anii 1990, C++ a devenit unul dintre cele mai populare limbaje de programare comerciale, rămânând astfel până azi.

Bjarne Stroustrup de la Bell Labs a dezvoltat C++ (inițial denumit C cu clase) în anii 1980, ca o serie de îmbunătățiri ale limbajului C. Acestea au început cu adăugarea noțiunii de clase, apoi de funcții virtuale, suprascrierea operatorilor, moștenire multiplă (engleză multiple inheritance), șabloane (engleză template) și excepții. Limbajul de programare C++ a fost standardizat în 1998 ca și ISO 14882:1998, versiunea curentă fiind din 2003, ISO 14882:2003. Următoarea versiune standard, cunoscută informal ca C++0x, este în lucru.

Stroustrup a început să lucreze la C cu clase în 1979. Ideea creării unui nou limbaj a venit din experiența de programare pentru pregătirea tezei sale de doctorat. Stroustrup a descoperit că Simula avea facilități foarte utile pentru proiecte mari, însă era prea lent, în timp ce BCPL era rapid, însă nu era de nivel înalt și era nepotrivit pentru proiecte mari. Când a început să lucreze pentru Bell Labs, avea sarcina de a analiza nucleul UNIX referitor la calcul distribuit. Amintindu-și de experiența sa din perioada lucrării de doctorat, Stroustrup a început să îmbunătățească C cu facilități asemănătoare Simula. C a fost ales deoarece era rapid și portabil. La început facilitățile adaugate C-ului au fost clase, clase derivate, verificare a tipului, inline și argumente cu valori implicite.

În timp ce Stroustrup a proiectat C cu clase (mai apoi C++), a scris de asemenea și Cfront, un compilator care genera cod sursă C din cod C cu clase. Prima lansare comercială a fost în 1985.

În 1982, numele limbajului a fost schimbat de la C cu clase la C++. Au fost adaugate noi facilități, inclusiv funcții virtuale, supraîncărcarea operatorilor și a funcțiilor, referințe, constante, alocare dinamică, un control al tipului mai puternic și noua variantă de comentariu pe un singur rând (liniile care încep cu caracterele '//').

În 1985 a fost lansată prima ediție a cărții "The C++ Programming Language" (Limbajul de programare C++), oferind informații importante despre limbaj, care încă nu era un standard oficial. În 1989 a fost lansată versiunea 2.0 a C++. Au apărut acum moștenirea multiplă, clase abstracte, funcții statice, funcții constante și membri protected. În 1990 o altă carte a fost lansată, oferind suport pentru standarde viitoare. Ultimele adăugări includeau template-uri, excepții, spații de nume (namespace-uri) și tipul boolean.

O dată cu evoluția limbajului C++, a evoluat și o bibliotecă standard. Prima adăugire a fost biblioteca de intrări/ieșiri (I/O stream), care oferea facilități pentru a înlocui funcțiile tradiționale C cum ar fi printf și scanf. Mai târziu, printre cele mai semnificative adăugari la biblioteca standard a fost STL (Standard Template Library) (Biblioteca de formate standard).

După ani de lucru, un comitet ANSI-ISO a standardizat C++ în 1998 (ISO/IEC 14882:1998).

- *Librării adiționale*

Utilizând librăria SFML, posibilitățile de dezvoltare grafică sunt extinse, librăria standard fiind mai limitată în acest domeniu. Bineînțeles, pot fi executate direct apeluri de GLFW sau OpenGL, dar acestea sunt implementate și apelate deja de către librăria noastră. Libraria este cross-platform, deci executabilul construit (ELF pentru Linux si DOS pentru Windows) va putea rula fara schimbări în cod.

- *Version control engines: GitHub (Git)*

Git-ul este o unealtă esențială în viața oricărui programator (și nu numai). Ca și definiție, el este un sistem de versionare (version-control) care urmărește modificările fișierelor și care poate ajuta colaborarea între mai multe persoane care lucrează la un anumit proiect. În cazul meu, pentru a face saltul de la un sistem la altul sau de la un calculator la altul, am nevoie doar de o conexiune la internet si toată munca mea este la îndemână.

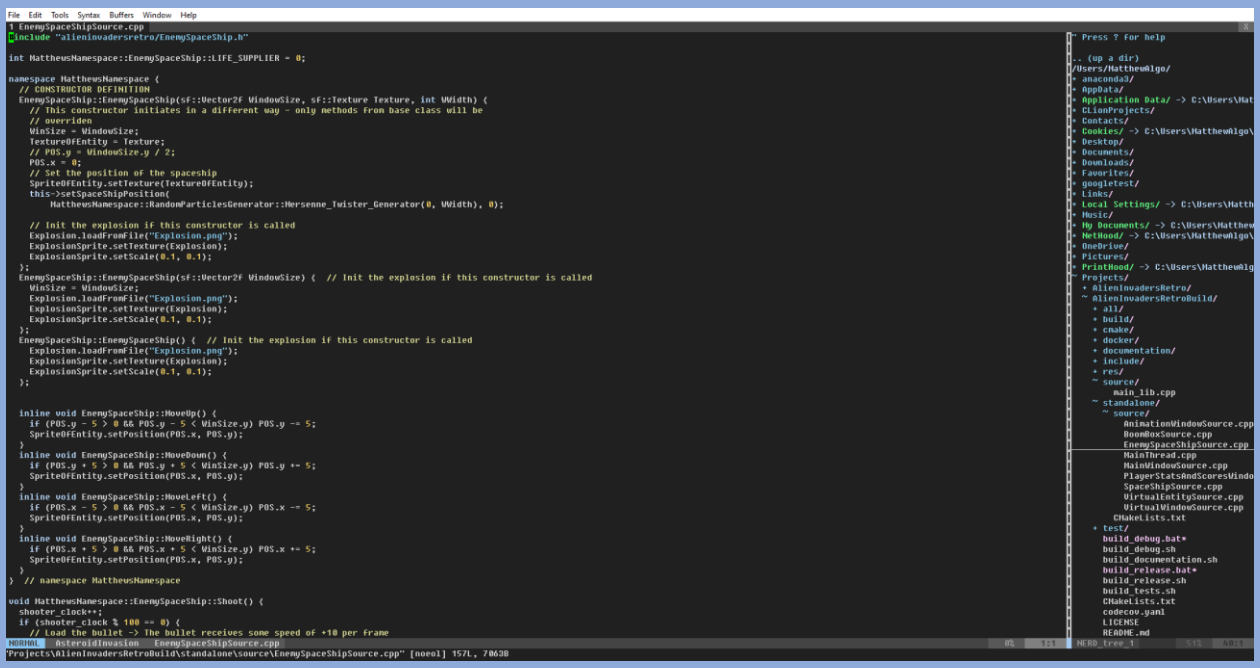
Exemplu de clonare a unui branch din cadrul unui repository

```
PS C:\Users\MatthewAlgo\Projects> git clone https://github.com/MatthewAlgo/AlienInvadersRetroBuild.git -b AsteroidInvas
on
Cloning into 'AlienInvadersRetroBuild'...
remote: Enumerating objects: 662, done.
remote: Counting objects: 100% (662/662), done.
remote: Compressing objects: 100% (407/407), done.
remote: Total 662 (delta 223), reused 646 (delta 207), pack-reused 0
Receiving objects: 100% (662/662), 28.51 MiB | 10.32 MiB/s, done.
Resolving deltas: 100% (223/223), done.
Updating files: 100% (128/128), done.
```

Repository-ul meu de Github are doua branch-uri, main și AsteroidInvasion. Branch-ul AsteroidInvasion conține versiunea finală a proiectului, master fiind cea de test. În cadrul versiunii master urmăresc adăugarea unor noi librării, ImGUI și GLFW, dar și boost::network pentru a adăuga funcții aplicației mele (un sistem de vizualizare dinamic al scorurilor, sau posibilitatea de realizare a unui sistem server-client pentru transmiterea pachetelor în rețea, deci opțiunea de joc în LAN).

- Editoare text: VSCode (Pe Windows) si VIM / Neovim pe Linux

Pentru mentinerea organizarii la nivel de proiect si de fisiere, am folosit editoare text populare in lumea programarii: VSCode, dezvoltat de Microsoft si VIM, care a aparut ca un editor lightweight pentru sistemele cu Linux si poate fi operat direct din terminal, fiind scris in C.



Se poate observa personalizarea editorului. Cu ajutorul plugin-ului NERDTree, pot obtine o vizualizare laterala a ierarhiei folderelor, facand procesul de accesare al lor mult mai la indemana. Am folosit, de asemenea, scriptul de personalizare scris in bash, facut public pe GitHub:

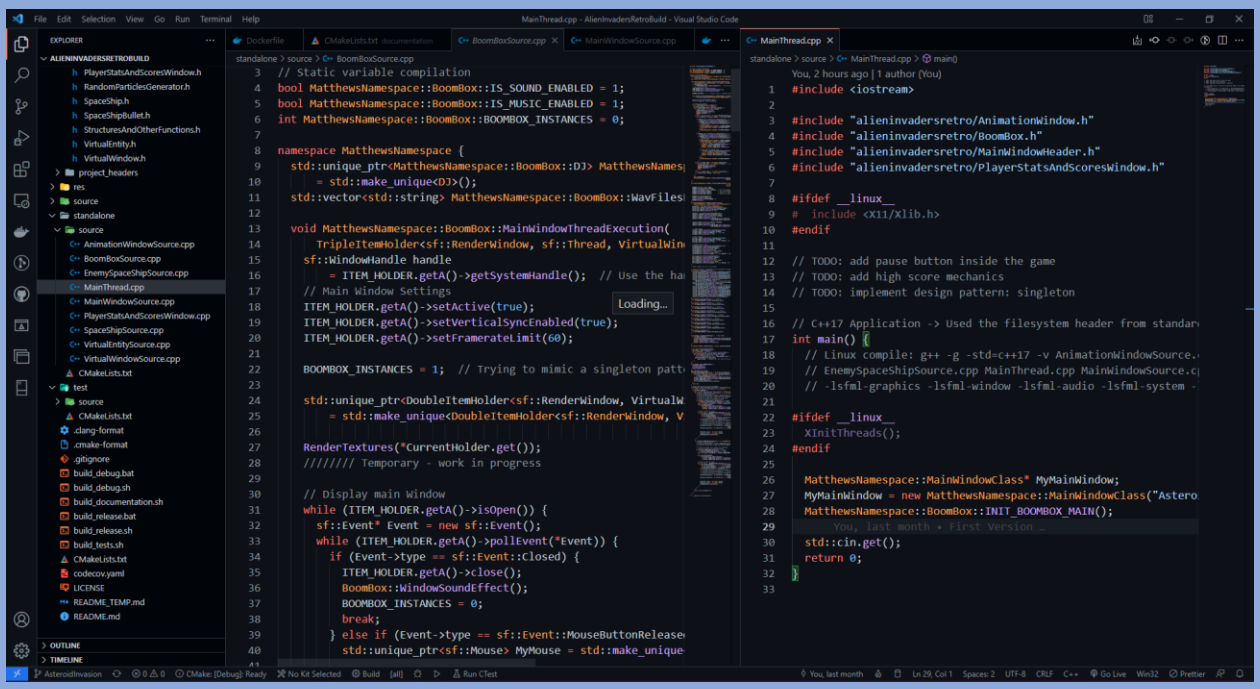
- GitHub - amix/vimrc: The ultimate Vim configuration (vimrc)

VIM este un editor foarte personalizabil, existand numeroase repository-uri in care diferite persoane pot incarca proiecte proprii, pe care utilizatorii le pot instala utilizand comanda :PlugInstall. Link catre NERDTree:

- GitHub - preservim/nerdtree: A tree explorer plugin for vim.

Visual Studio este un editor complet facut de Microsoft ce suporta mai multe limbi, oferind plugin-uri pentru fiecare limbă. De exemplu, modulul de C++ aduce Intellisense, modulul de corectare al erorilor din cod, precum si syntax highlighting, ambele esentiale pentru un programator.

VSCode in acțiune

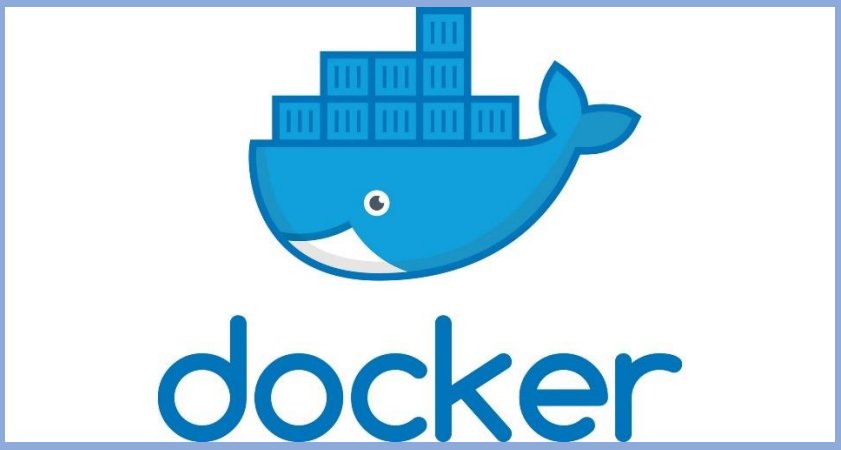


- CMake: The cross-platform build system

CMake este un sistem de generare a tututor fisierelor necesare pentru a face un proiect sa ruleze, in special proiecte realizate in C/C++. El actioneaza in esenta ca un cross-compiler, avand capacitatea sa compileze executabilul pe mai multe platforme, fara modificari in cod. CMake poate fi considerat un limbaj in sine, avand capacitatea de a apela metode interne interpretate de program.

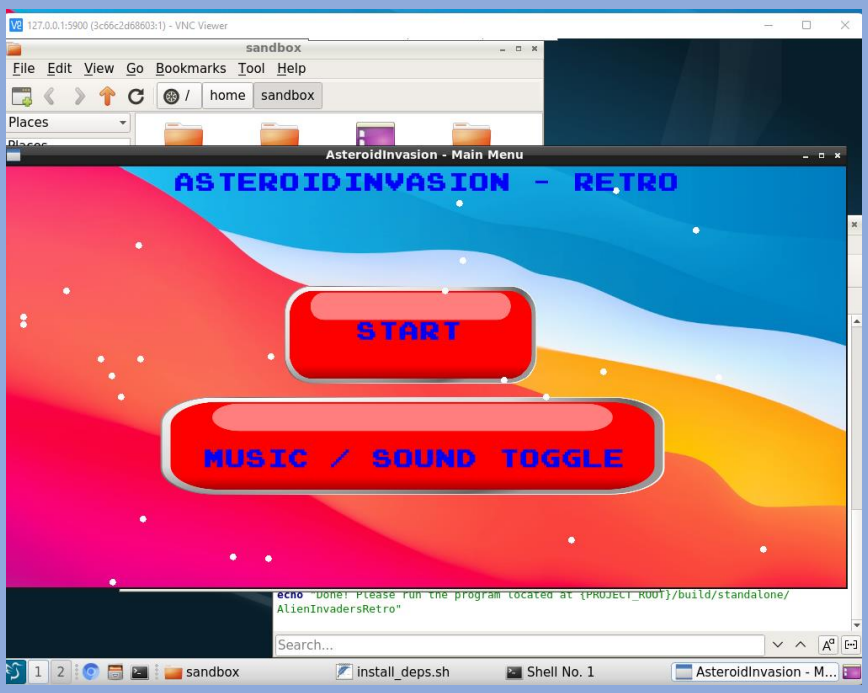
```
cmake_minimum_required(VERSION 3.14 FATAL_ERROR)      You, last month • First Version …
set(CMAKE_CXX_STANDARD 17)

project(AsteroidInvasionTests LANGUAGES CXX)

# ---- Options ----

option(ENABLE_TEST_COVERAGE "Enable test coverage" OFF)
option(TEST_INSTALLED_VERSION "Test the version found by find_package" OFF)

# --- Import tools ----

include(../cmake/tools.cmake)

# ---- Dependencies ----

include(../cmake/CPM.cmake)

CPMAddPackage("gh:onqtam/doctest#2.4.5")
CPMAddPackage("gh:TheLartians/Format.cmake@1.7.0")

if(TEST_INSTALLED_VERSION)
  find_package(AsteroidInvasionTests REQUIRED)
else()
  CPMAddPackage(NAME AsteroidInvasionTests SOURCE_DIR ${CMAKE_CURRENT_LIST_DIR}/..)
endif()

# ---- Create binary ----

file(GLOB sources CONFIGURE_DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/source/*.cpp)
add_executable(${PROJECT_NAME} ${sources})
target_link_libraries(${PROJECT_NAME} doctest::doctest)
set_target_properties(${PROJECT_NAME} PROPERTIES CXX_STANDARD 17)
```

- Docker Containers: O versiune mai eficienta a VirtualBox



Fiecare programator trebuie sa aiba la indemana o unealta rapida pentru software deployment. In cazul nostru, accesul la un sistem Linux cu server video X11 integrat se face direct prin intermediul daemon-ului Docker. Programul descarca de pe server un container pre-setat si pregatit pentru a rula aplicatii cu interfata grafica, iar tot ce trebuie sa facem este sa clonam repository-ul pe noul sistem si sa construim executabilul.

```
FROM dorowu/ubuntu-desktop-lxde-vnc:bionic-lxqt

COPY install_deps.sh /home/sandbox/install_deps.sh
```



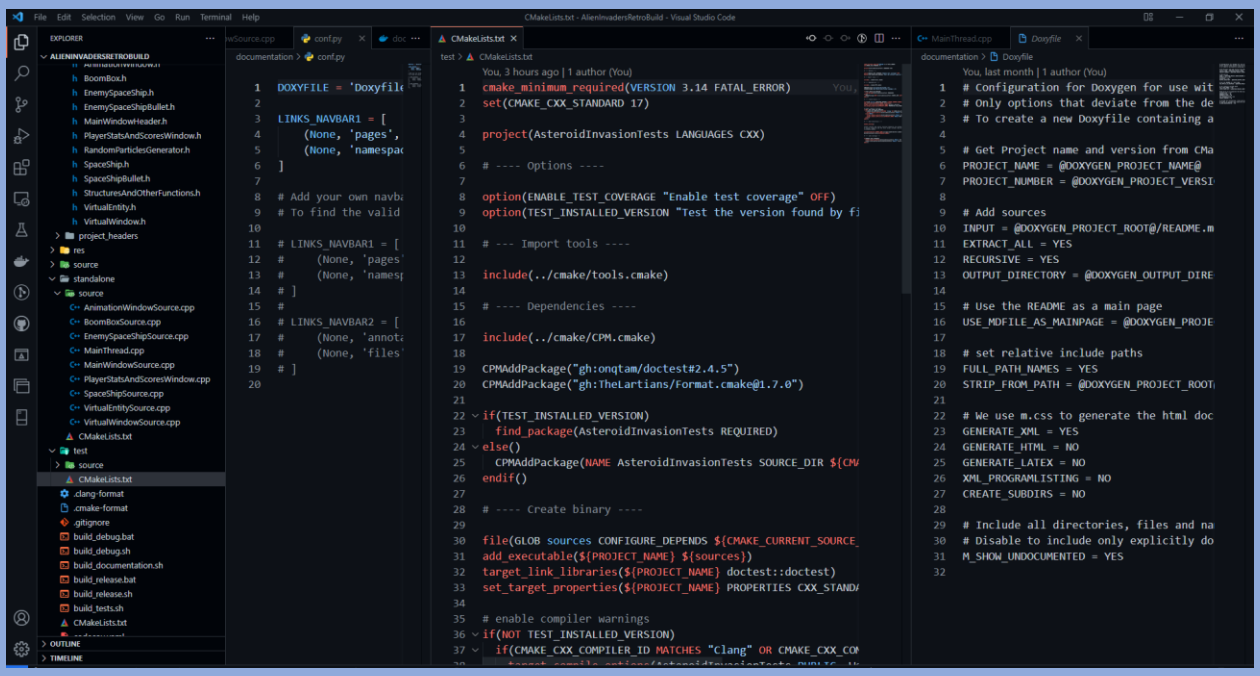
- Modern CPP Starter: Structura proiectului

[GitHub - TheLartians/ModernCppStarter: 🚀 Kick-start your C++! A template for modern C++ projects using CMake, CI, code coverage, clang-format, reproducible dependency management and much more.]

Deoarece GitHub ajuta mult in cadrul procesului de dezvoltare, comunitatea open source pune la dispozitie un template de folosire a unei structuri de proiect Cross-platform, care se foloseste de CMake si include module de unit testing, cu ajutorul doctest, dar si un generator de

documentatii in format HTML, Doxygen, ce interpreteaza anumite anotatii direct in cod si le include in documentatie.

Doxygen si doctest in cadrul proiectului



- Scripturi BASH si Batch – pentru automatizarea comenzilor de terminal: Aplicabilitate in Powershell, CMD si Linux

```
# Joystick
yes Y | sudo apt-get install libudev-dev

# Compiler and build Tool
yes Y | sudo apt-get install vim git wget


# Install CMake
wget https://github.com/Kitware/CMake/releases/download/v3.23.0-rc1/cmake-3.23.0-rc1-linux-x86_64.sh
bash cmake-3.23.0-rc1-linux-x86_64.sh

# Install a Compiler
# wget http://ftp.de.debian.org/debian/pool/main/g/gcc-defaults/g++_8.3.0-1_amd64.deb
# yes Y | sudo dpkg -i g++_8.3.0-1_amd64.deb
yes Y | sudo apt --fix-broken install
yes Y | sudo add-apt-repository ppa:ubuntu-toolchain-r/test
yes Y | sudo apt-get update
yes Y | sudo apt install gcc-11 g++-11
# yes Y | sudo apt install clang-11 --install suggests

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 700 --slave /usr/bin/g++ g++ /usr
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-11 800 --slave /usr/bin/g++ g++ /us

# Clone the game from Github
git clone https://github.com/MatthewAlgo/AlienInvadersRetroBuild.git -b AsteroidInvasion
# Build and run
cd AlienInvadersRetroBuild && ../cmake-3.23.0-rc1-linux-x86_64/bin/cmake -S standalone -B build/stan
  ../cmake-3.23.0-rc1-linux-x86_64/bin/cmake --build build/standalone --config Release
# Copy the resources
cp -r ./res/* ./build/standalone/
echo "Done! Please run the program located at {PROJECT_ROOT}/build/standalone/AlienInvadersRetro"
```

- Server VNC care ruleaza pe localhost, client VNC (TigerVNC sau VNCViewer)

Containerul Docker amintit in sectiunile anterioare mapeaza portul 5800 al containerului cu portul 5900 al calculatorului, putandu-se accesta, astfel, modulul grafic al containerului direct pe localhost.

Elemente de limbaj, tehnici de programare

- Un programator bun este intotdeauna organizat cu codul sau, fie ca este vorba de o aplicatie simpla sau de una complexa, trebuie sa se asigure ca tot ce scrie se intelege si poate fi continuat oricand spre imbunatatire, el nedevenind ambiguu pe masura ce sunt adaugate functii noi.
- Structura proiectului: 12 fisiere header, 9 translation units

E de retinut faptul ca am tinut cont de paradigme de programare precum, regula inexistentei circularitatii in includerea reciproca a fisierelor header: Utilizarea directivelor de preprocesor “#pragma once” si “#ifdef”, ele fiind vizibile in cadrul fiecarui header.

Directive de preprocesor in cadrul unui fisier header

```
#pragma once        You, last month • First
#ifndef ENEMY_SPACE_SHIP_H
#  define ENEMY_SPACE_SHIP_H

#  include <SFML/Graphics.hpp>
#  include <SFML/Window.hpp>
#  include <deque>

#  include "EnemySpaceShipBullet.h"
#  include "BoomBox.h"
#  include "VirtualEntity.h"

#  pragma region ENEMY SPACESHIP REGION
```

Notiuni avansate de programare au fost implementate in proiect, precum programarea orientata pe obiect, inheritanta, metode virtuale, functii si variabile statice, membri privati, publici si protejati.

- **Clase partial virtuale, niveluri de abstractizare**

```
namespace MatthewsNamespace {
  MatthewAlgo-Debian11Dell, 3 hours ago | 2 authors (You and others)
  class VirtualEntity {
  private:
      short EntityLife = 10; // Each entity has a life
  protected:
    You, last week | 1 author (You)
    typedef struct Position {
      double x, y, speed = 0;
    };
    Position POS;  // Each entity has a position

    sf::Texture TextureOfEntity;
    sf::Sprite SpriteOfEntity;
    sf::Vector2f WinSize;

    // It produces explosions on its bullet collisions -> their data is located here
    sf::Texture Explosion;
    sf::Sprite ExplosionSprite;

  public:
    // Constructors
    VirtualEntity(sf::Vector2f WindowSize, sf::Texture Texture);
    VirtualEntity(sf::Vector2f WindowSize);
    VirtualEntity();

    virtual ~VirtualEntity() = default;        // Default destructor


    std::deque<SpaceShipBullet*> BulletDeque;  // Each entity has its own stack of bullets

    // FUNCTIONS - EACH ENTITY CAN MOVE
    virtual void MoveUp();
    virtual void MoveDown();
    virtual void MoveLeft();
```

Clasa mentionata anterior nu are sens sa fie instantiata, chiar daca poate sustine instante de una singura. Ea este construita la un nivel abstractizat, referindu-se in special la obiecte care sunt inheritate din ea, reprezentand o clasa de baza pentru o serie de sub-clase ale unor obiecte care se pot misca, respectiv au texturi (clase precum inamici – Asteroizii, dar si jucatorul principal, ambele fiind inheritate din aceasta clasa):

```
class SpaceShip : public virtual VirtualEntity {
public:
  short Life = 50;       You, last week • Made some changes to the internal structure (C++)
  // Each spaceship has a queue of bullets
  std::deque<SpaceShipBullet*> BulletDeque;

  /*
  inline void MoveUp() override;
  inline void MoveDown() override;
  inline void MoveLeft() override;
  inline void MoveRight() override;
  */ // We are using the pre-defined versions of those functions (that are inherited)

  // Constructors
  SpaceShip(sf::Vector2f WindowSize, sf::Texture Texture) {
    WinSize = WindowSize;
    TextureOfEntity=Texture;
    VirtualEntity(WindowSize, Texture);

  }
  SpaceShip(sf::Vector2f WindowSize) {
    // Init the explosion if this constructor is called
    WinSize=WindowSize;
    VirtualEntity(Winsize);
  }
```

Exemplu de inheritanta din cadrul proiectului: sub-clasa SpaceShip, care gazduieste naveta spatiala protagonist

```
namespace MatthewsNamespace {
  You, last week | 1 author (You)
  class EnemySpaceShip : public virtual VirtualEntity {
  private:
    short Life = 10; // The enemies have reduced life
  protected:
    int shooter_clock = 0;
  public:
    // Each spaceship has a queue of bullets
    std::deque<EnemySpaceShipBullet*> BulletDeque;
    static int LIFE_SUPPLIER;

    inline void MoveUp() override;
    inline void MoveDown() override;
    inline void MoveLeft() override;
    inline void MoveRight() override;

    EnemySpaceShip(sf::Vector2f WindowSize, sf::Texture Texture, int WWidth);
    EnemySpaceShip(sf::Vector2f WindowSize);
    EnemySpaceShip();
    ~EnemySpaceShip() = default;
```

Sub-clasa EnemySpaceShip, care face override metodelor implementate deja in metoda virtuala: Miscarea este diferita, nu mai este controlata, ci intra in controlul unui random engine, Mersennes Twister.

```
  unsigned short generator_clock = 0;
    inline MatthewsNamespace::RandomParticlesGenerator::RandomParticlesGenerator()
    +1 overload
  RandomParticlesGenerator() = default;
  ~RandomParticlesGenerator() = default;      You, last week • Made some changes to the internal stru

  static int Mersenne_Twister_Generator(int minvalue, int maxvalue) {
    std::random_device device;
    std::mt19937 generator(device());
    std::uniform_int_distribution<int> distribution(minvalue, maxvalue);
    return distribution(generator);
  }
  void Generate() {
    generator_clock++;
    if (generator_clock % 2 == 0) {
      WhiteParticle Xparticle;
      sf::Texture Texture;
      Texture.loadFromFile("WhiteDot.png");
      VectorOfParticles.push_back(new WhiteParticle(Xparticle));
      VectorOfParticles.back()->speed = Mersenne_Twister_Generator(5, 20);
      VectorOfParticles.back()->x = Mersenne_Twister_Generator(0, 1000);
      VectorOfParticles.back()->Texture = new sf::Texture(Texture);
      VectorOfParticles.back()->Sprite.setTexture(*VectorOfParticles.back()->Texture);
      VectorOfParticles.back()->Sprite.setScale(0.01, 0.01);
      VectorOfParticles.back()->Sprite.setPosition(sf::Vector2f(VectorOfParticles.back()->x, 0));
      generator_clock = 0;
    }
  }
```

Implementarea in cod a Mersennes Twister

Motorul Mersennes Twister are mai multe aplicabilitati si este folosit in mai multe situatii in cadrul proiectului. De exemplu, pozitiile initiale ale bilutelor albe care apar in ecranul principal si cel al jocului sunt generate cu ajutorul acestui motor.

```
void InLoopForParticles(sf::RenderWindow* X) {
  for (int i{}; i < VectorOfParticles.size(); ++i) {
    X->draw(VectorOfParticles.at(i)->Sprite);
    VectorOfParticles.at(i)->y += VectorOfParticles.at(i)->speed;
    VectorOfParticles.at(i)->Sprite.setPosition(
        sf::Vector2f(VectorOfParticles.at(i)->x, VectorOfParticles.at(i)->y));
  }
}
void ClearMemory(sf::RenderWindow* X) {
  for (int i{}; i < VectorOfParticles.size(); ++i) {
    if (VectorOfParticles.at(i)->y > X->getSize().y) {
      delete VectorOfParticles.at(i)->Texture;
      WhiteParticle* it = VectorOfParticles.at(i);
      delete it;
      it = nullptr;
      VectorOfParticles.erase(VectorOfParticles.begin() + i);
    }
  }
}
```

Motorul de generare are implmentat un mecanism de eliberare a memoriei. De indata ce o biluta iese din raza ecranului, ea este stearsa definitive din memorie, fara a lasa dangling pointers. Avem, astfel, un sistem dinamic de eliberare a memoriei din pointeri.

Mecanismul de miscare al unui asteroid

```
void MatthewsNamespace::EnemySpaceShip::Draw_IterateExistingItem(sf::RenderWindow* Window) {
  POS.y += this->POS.speed;
  MoveRandomLeftOrRightorUpOrDown();
  this->setSpaceShipPosition(POS.x, POS.y);
  this->getSpaceShipSprite()->setScale(0.1, 0.1);
  Window->draw(*this->getSpaceShipSprite());
}
void MatthewsNamespace::EnemySpaceShip::MoveRandomLeftOrRightorUpOrDown() {
  int randomx = MatthewsNamespace::RandomParticlesGenerator::Mersenne_Twister_Generator(-10, 10);
  int randomy = MatthewsNamespace::RandomParticlesGenerator::Mersenne_Twister_Generator(-10, 10);

  if (POS.x + randomx > 0 && POS.x + randomx < WinSize.x) POS.x += randomx;
  if (POS.y + randomy > -20 && POS.y + randomy < WinSize.x) POS.y += randomy;
}
void MatthewsNamespace::EnemySpaceShip::Die() {              // Free up the buffer of bullets
  for (unsigned int i{}; i < this->BulletDeque.size(); i++) {  // Manage and free up the memory
    EnemySpaceShipBullet* it = this->BulletDeque.at(i);
    delete it;
    it = nullptr;
    this->BulletDeque.erase(this->BulletDeque.begin() + i);
  }
}       You, last month • First Version …
```

Algoritmul de generare al undei: O serie de obiecte sf::Lines interlegate, cu puncte de extrem determinate in functie de analiza arrayului de short integers din SFML/System.h. Algoritmul analizeaza timpul de la inceputul melodiei. Cu regula de 3 simpla, realizeaza range-ul de sample-uri corespunzator secundei curente. Amplitudinea este redusa pentru ca unda sa incapa in inaltimea ferestrei.

```
if (IS_MUSIC_ENABLED) {
  try {  // The song might have been deleted in another thread
    if (BoomBox::getMainTheme()->getStatus() == sf::SoundSource::Status::Playing) {
      // Get the current position
      sf::Time CurrentPosition = BoomBox::getMainTheme()->getPlayingOffset();
      // We need to do the math to get the current samples to be displayed
      // Freeze the thread until samples are available
      // while(CurrentPosition.asSeconds()==0){}
      if (BoomBox::getMainTheme()->getBuffer() != nullptr
          || BoomBox::getMainTheme()->getBuffer() != NULL) {
        const sf::Int16* LocalBuffer = BoomBox::getMainTheme()->getBuffer()->getSamples();
        int CurrentPosInSamples
            = CurrentPosition.asMilliseconds()
              * BoomBox::getMainTheme()->getBuffer()->getSampleCount()
              / BoomBox::getMainTheme()->getBuffer()->getDuration().asMilliseconds();

        // Build a vector of Lines
        for (int i{}; i < BoomBoxWindow->getSize().y; i += precision) {
          if (CurrentPosInSamples + i - 3 >= 0 && CurrentPosInSamples + i - 2 >= 0
              && CurrentPosInSamples + i - 1 >= 0 && CurrentPosInSamples + i >= 0
              && CurrentPosInSamples + i
                     < BoomBox::getMainTheme()->getBuffer()->getSampleCount()) {
            sf::VertexArray Vertex(sf::LinesStrip, 2);
            // Vertex[0].position = sf::Vector2f(LocalBuffer[CurrentPosInSamples + i - 3] / 100,
            // LocalBuffer[CurrentPosInSamples + i - 2] / 100 + BoomBoxWindow->getSize().y/2);
            // Vertex[1].position = sf::Vector2f(LocalBuffer[CurrentPosInSamples + i - 1]/100,
            // LocalBuffer[CurrentPosInSamples + i]/100+ BoomBoxWindow->getSize().y / 2);
            Vertex[0].position = sf::Vector2f(i, LocalBuffer[CurrentPosInSamples + i - 2] / 200
                                                 + BoomBoxWindow->getSize().y / 2);
            Vertex[1].position = sf::Vector2f(i + 1, LocalBuffer[CurrentPosInSamples + i] / 200
                                                     + BoomBoxWindow->getSize().y / 2);

            Vertex[0].color = sf::Color::Green;
            Vertex[1].color = sf::Color::Green;
            BoomBoxWindow->draw(Vertex);
          }
        }
      }
```

Alte tehnici de programare: **Multithreading si paralelism in cadrul proceselor ferestrelor, notiuni de metaprogramming, template-uri**

clase construite din template-uri, structuri de date personalizate, holderi de obiecte si pointeri.

```
int main() {
  // Linux compile: g++ -g -std=c++17 -v AnimationWindowSource.cpp BoomBoxSource.cpp
  // EnemySpaceShipSource.cpp MainThread.cpp MainWindowSource.cpp SpaceShipSource.cpp
  // -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system -lX11 -o AlienInvadersRetro.elf
      You, last month • First Version …
#ifdef __linux__
  XInitThreads();
#endif

  MatthewsNamespace::MainWindowClass* MyMainWindow;
  MyMainWindow = new MatthewsNamespace::MainWindowClass("AsteroidInvasion - Main Menu", 1000, 500);
  MatthewsNamespace::BoomBox::INIT_BOOMBOX_MAIN();

  std::cin.get();
  return 0;
}
```

Inghetarea threadului principal, apelul threadului ferestrei (constructorul clasei), precum si apelul unei functii care depind de sistemul pe care se ruleaza programul: Serverul X11 este inexistent pe Windows.

```
namespace MatthewsNamespace {
  You, last week | 1 author (You)
  class MainWindowClass : public virtual MatthewsNamespace::VirtualWindowClass {
  private:
    sf::Font GlobalWindowFont;
    sf::Text GreetingText, TextBTN1, TextBTN2;
    ImageToBeDrawn MenuBox1, MenuBox2, MenuBox3;
    std::unique_ptr<ImageToBeDrawn> WindowTitleTextbox;
      You, last week • Made some changes to the internal structure (C++)…
    // Variables related to the textures and design elements
  public:
    MainWindowClass(const std::string TITLE, int W, int H) : VirtualWindowClass(TITLE, W, H){};
    ~MainWindowClass() = default;  // Auto deallocate smart pointers

    void MainWindowThreadExecution(
        TripleItemHolder<sf::RenderWindow, sf::Thread, VirtualWindowClass>& ITEM_HOLDER) override;
    void DrawInsideMainWindow(sf::RenderWindow* WINDOW, sf::Thread* WINTHREAD,
                              VirtualWindowClass* C) override;
    void RenderTextures(
        DoubleItemHolder<sf::RenderWindow, VirtualWindowClass> ITEM_HOLDER) override;
  };
};  // namespace MatthewsNamespace

#  pragma endregion CLASS_REGION
```

Fereastra principala inheritand clasa virtuala abstractizata, suprascriind functiile ei in ierarhie si apeland functia care ia ca parametru clasa personalizata (holder of templates).

```
VirtualWindowClass(const std::string TITLE, int W, int H)
    : WindowTitle(TITLE),
      MainWindowVideo(new sf::VideoMode(W, H)),
      WWidth(static_cast<int>(W)),
      WHeight(static_cast<int>(H)),
      ParticleGenerator(new MatthewsNamespace::RandomParticlesGenerator()) {
  // MainWindowThread = new
  // sf::Thread(std::bind(&MainWindowClass::MainWindowThreadExecution,this, *TripleHolder));

  MainWindowThread = std::make_unique<sf::Thread>([&]() -> void {
    // Create window and set active
    VirtualWindowClass::WindowPointer
        = new sf::RenderWindow(*MainWindowVideo, WindowTitle,
                               sf::Style::Titlebar | sf::Style::Close);  // Create the window
    WindowPointer->setActive(false);
    You, last week • Made some changes to the internal structure (C++)…
    std::unique_ptr<TripleItemHolder<sf::RenderWindow, sf::Thread, VirtualWindowClass>>
        TripleHolder
        = std::make_unique<TripleItemHolder<sf::RenderWindow, sf::Thread, VirtualWindowClass>>(
            WindowPointer, MainWindowThread.get(), this);

    this->MainWindowThreadExecution(*TripleHolder);
  });
  // Create and launch the window thread
  MainWindowThread->launch();
};
```

Crearea threadului ferestrelor, constructorul este apelat din ierarhie, fara sa fie nevoie repetarea lui in toate clasele. Este folosita o functie lambda cu mentiunea ca returneaza “Void”, o functie anonima care este data mai departe unui pointer unic ce retine un sf::Thread, care, executat, randeaza grafic instructiunile din cadrul apelului. De asemenea, se creaza pentru fiecare instanta o structura personalizata ce ia ca parametru o serie de pointeri catre membrii clasei. Functia primeste, astfel, un singur parametru format din 3 entitati in loc de 3 parametri separati.

```cpp
template <typename A = void, typename B = void, typename C = void> struct TripleItemHolder {
private:
  A* A_VAR;
  B* B_VAR;
  C* C_VAR;

public:
  TripleItemHolder(A* a, B* b, C* c) {
    A_VAR = a;
    B_VAR = b;
    C_VAR = c;
  }
  TripleItemHolder(TripleItemHolder* INST) {
    A_VAR = INST->getA();
    B_VAR = INST->getB();
    C_VAR = INST->getC();
  }
  ~TripleItemHolder() {
    // Should just hold the items, not delete them
  }
  A* getA() const { return A_VAR; }
  B* getB() const { return B_VAR; }       You, last week • Made some changes to the internal struct
  C* getC() const { return C_VAR; }
  TripleItemHolder& getInstance() { return *this; };
  TripleItemHolder* getCopy() { return new TripleItemHolder(this); }
};
```

Structura personalizata: Un template care ia ca parametri typename-uri (denumiri de clase sau tipuri de date) si retine pointeri catre ei. Datele sunt, astfel, organizate fara a fi copiate.

Design patterns: Un singleton “reinventat”

```cpp
protected:
  sf::Font GlobalWindowFont;
  sf::Text ScoreText, LivesText, GameOverText, PresskeyText,
      LevelUpText;  // Text displayed in-app

  MatthewsNamespace::SpaceShip SpaceShipMainPlayer;  // Spaceships
  std::unique_ptr<ImageToBeDrawn> WindowTitleTextbox;
  long long Player1Score = 0;  // Player Score

  std::vector<EnemySpaceShip*> VectorOfEnemies;
  int enemy_spawn_clock = 0;
  short Cnt1000 = 0;
  short LevelUpConstant = 0;

public:
  static int ANIMATION_INSTANCES;
```

Se cunoaste faptul ca membri statici din cadrul unei clase nu depind de instantele acelei clase. Putem sa ne asiguram, asftel, ca exista o singura instanta a clasei ferestrei jocului (dar si a vizualizatorului de unde) prin crearea unui membru static, un contor, care creste la fiecare apel al constructorului clasei. Daca valoarea este deja 1, constructorul isi termina procesul fara sa instantieze clasa.

Ce sunt design patterns?

Design patterns reprezintă cele mai bune practici utilizate de dezvoltatorii de software cu experiență OOP. Modelele de design sunt soluții la problemele generale cu care s-au confruntat dezvoltatorii de software în timpul dezvoltării software. Aceste soluții au fost obținute prin încercare și eroare de numeroși dezvoltatori de software pe o perioadă destul de substanțială de timp.

Platformă comună pentru dezvoltatori

Modelele de design oferă o terminologie standard și sunt specifice unui anumit scenariu. De exemplu, un model de design singleton înseamnă utilizarea unui singur obiect, astfel încât toți dezvoltatorii familiarizați cu modelul de design unic vor folosi un singur obiect și își pot spune reciproc că programul urmează un model singleton.

Cele mai bune practici

Modelele de design au evoluat pe o perioadă lungă de timp și oferă cele mai bune soluții la anumite probleme cu care se confruntă în timpul dezvoltării software. Învățarea acestor modele ajută dezvoltatorii neexperimentați să învețe designul software într-un mod ușor și mai rapid.

Tipuri de modele de design

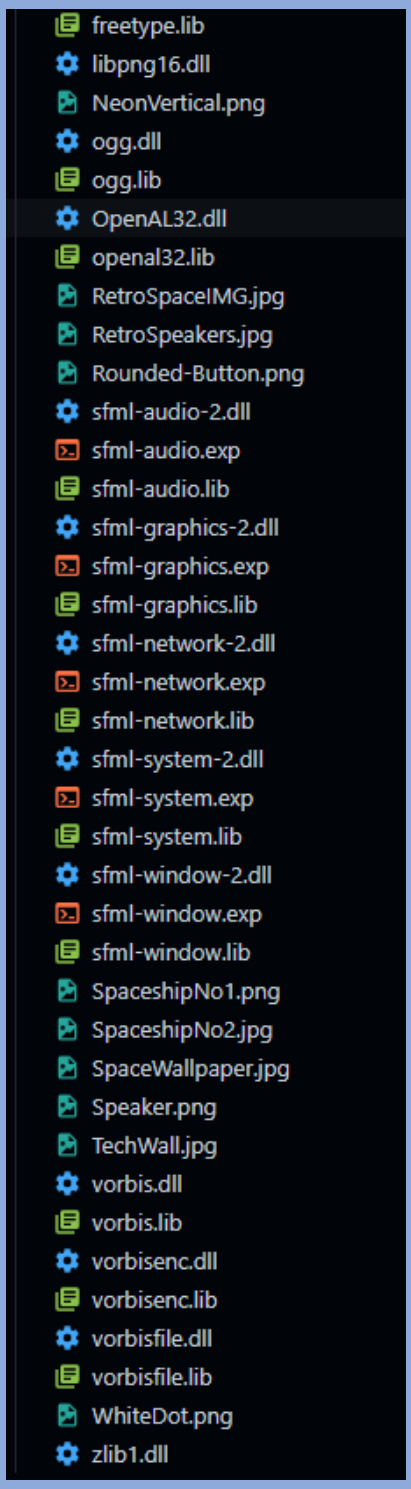
Conform cărții de referință pentru modele de design Design Patterns - Elements of Reusable Object-Oriented Software, există 23 de modele de design care pot fi clasificate în trei categorii: **modele creaționale – Factory, abstract, singleton, prototype, builder**, structurale și comportamentale.

Singleton design pattern is a software design principle that is used to restrict the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. For example, if you are using a logger, that writes logs to a file, you can use a singleton class to create such a logger. You can create a singleton class using the following code –

Example

```
#include <iostream>

using namespace std;

class Singleton {
   static Singleton *instance;
   int data;

   // Private constructor so that no objects can be created.
   Singleton() {
      data = 0;
   }

   public:
   static Singleton *getInstance() {
      if (!instance)
      instance = new Singleton;
      return instance;
   }

   int getData() {
      return this -> data;
   }

   void setData(int data) {
      this -> data = data;
   }
};

//Initialize pointer to zero so that it can be initialized in first call to getInstance
Singleton *Singleton::instance = 0;

int main(){
   Singleton *s = s->getInstance();
   cout << s->getData() << endl;
   s->setData(100);
   cout << s->getData() << endl;
   return 0;
}
```

Resurse necesare rularii: Organizate si copiate direct din folderul ${PROJ_ROOT}/res in folderul cu executabilul prin intermediul scripturilor (Fisiere DLL pentru Windows – librarii linkate dinamic, plus imagini ce sunt incarcate pe placa video – sprite-uri, texturi)



SFÂRȘIT