**Colegiul Naţional „Roman-Vodă” Roman**

***PROIECT PENTRU ATESTAREA***

***COMPETENŢELOR PROFESIONALE***

Profesor îndrumător:

***Silviu-George LARIU***

Candidat:

***Dumea ALEXANDRU***

**2019**

***Tema:***

***JOC INTERACTIV***

***(imitație Flappy Bird)***

***Cuprins***

***Introducere***

***Partea I. Noţiuni teoretice***

1. Limbajul C++

1.1. Introducere..............................................................................................................

1.2. Programarea orientată pe obiecte............................................................................

1.3. Clase și obiecte........................................................................................................

2. Librăria SFML

2.1. Ce este SFML?.........................................................................................................

2.2. Module SFML..........................................................................................................

***Partea II. Descrierea realizării părţii practice***

Realizarea părții practice.............................................................................................

***Introducere***

Motivația alegerii acestei teme - recrearea jocului Flappy Bird - constă, în mare parte, în accesibilitatea acestei sarcini datorită complexității relativ mică. Așadar, consider că această temă este o bună introducere în tărâmul jocurilor 2D, ea impunând un model mintal de gândire pragmatic, specific dezvoltării nu doar a jocurilor, ci și a software-ului, in general.

***Partea I. Noţiuni teoretice***

1. **Limbajul C++**

**1.1 Introducere**

**C++** este un [limbaj de programare general](https://ro.wikipedia.org/w/index.php?title=Limbaj_de_programare_general&action=edit&redlink=1), [compilat](https://ro.wikipedia.org/w/index.php?title=Compilare&action=edit&redlink=1). Este un [limbaj multi-paradigmă](https://ro.wikipedia.org/wiki/Limbaj_de_programare_multi-paradigm%C4%83), cu verificarea statică a tipului variabilelor ce suportă [programare procedurală](https://ro.wikipedia.org/wiki/Programare_procedural%C4%83), [abstractizare a datelor](https://ro.wikipedia.org/w/index.php?title=Abstractizare_(informatic%C4%83)&action=edit&redlink=1), [programare orientată pe obiecte](https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte). În [anii 1990](https://ro.wikipedia.org/wiki/Anii_1990), C++ a devenit unul dintre cele mai populare limbaje de programare comerciale, rămânând astfel până azi[[1]](https://ro.wikipedia.org/wiki/C%2B%2B#cite_note-tiobe-1).

[Bjarne Stroustrup](https://ro.wikipedia.org/wiki/Bjarne_Stroustrup) de la [Bell Labs](https://ro.wikipedia.org/wiki/Bell_Labs) a dezvoltat C++ (inițial denumit *C cu clase*) în anii [1980](https://ro.wikipedia.org/wiki/1980), ca o serie de îmbunătățiri ale limbajului [C](https://ro.wikipedia.org/wiki/Limbajul_C). Acestea au început cu adăugarea noțiunii de [clase](https://ro.wikipedia.org/w/index.php?title=Clas%C4%83_(informatic%C4%83)&action=edit&redlink=1), apoi de [funcții virtuale](https://ro.wikipedia.org/wiki/Func%C8%9Bie_virtual%C4%83), [suprascrierea operatorilor](https://ro.wikipedia.org/w/index.php?title=Suprascrierea_operatorilor&action=edit&redlink=1), [moștenire multiplă](https://ro.wikipedia.org/w/index.php?title=Mo%C8%99tenire_multipl%C4%83&action=edit&redlink=1), [șabloane](https://ro.wikipedia.org/wiki/%C8%98ablon_(programare)) ([engleză](https://ro.wikipedia.org/wiki/Limba_englez%C4%83) *template*) și [excepții](https://ro.wikipedia.org/w/index.php?title=Excep%C8%9Bie_(programare)&action=edit&redlink=1). Limbajul de programare C++ a fost standardizat în [1998](https://ro.wikipedia.org/wiki/1998) ca și [ISO](https://ro.wikipedia.org/wiki/ISO) 14882:1998, versiunea curentă fiind din 2003, ISO 14882:2003. Următoarea versiune standard, cunoscută informal ca [C++0x](https://ro.wikipedia.org/w/index.php?title=C%2B%2B0x&action=edit&redlink=1), este în lucru.

**1.2 Programarea orientată pe obiecte**

Programarea orientată pe obiecte (POO, uneori și Programarea orientată obiect, uneori denumită ca și în limba engleză, Object Oriented Programming (OOP)) este o [paradigmă de programare](https://ro.wikipedia.org/wiki/Paradigm%C4%83_de_programare), axată pe ideea încapsulării, adică grupării datelor și codului care operează asupra lor, într-o singură structură. Un alt concept important asociat programării orientate obiect este polimorfismul, care permite abstractizări ce permit o descriere conceptuală mai simplă a soluției.

**1.3 Clase și obiecte**

Programarea orientată pe obiect (Programare Orientată Obiectual) este unul din cei mai importanți pași făcuți în evoluția [limbajelor de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai simplu, pentru a putea fi înțeleasă de cât mai mulți programatori. Astfel unitățile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât modul de lucru al calculatorului. Până la apariția programării orientate pe obiect, programele erau implementate în [limbaje de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) procedurale (C, Pascal) sau în limbaje care nici măcar nu ofereau o modalitate de grupare a instrucțiunilor în unități logice (funcții, proceduri) cum este cazul limbajului de asamblare (asembler). Altfel spus o problemă preluată din natură trebuia fragmentată în repetate rânduri astfel încât să se identifice elementele distincte, implementabile într-un limbaj de programare. O mare problemă a [programării procedurale](https://ro.wikipedia.org/wiki/Programare) era separarea datelor de unitățile care prelucrau datele (subrutinele), ceea ce făcea foarte dificilă extinderea și întreținerea unui program. Astfel s-a pus problema ca aceste două [entități](https://ro.wikipedia.org/wiki/Entitate) (date și subrutine) să fie grupate într-un anumit mod, astfel încât subrutinele să "știe" în permanență ce date prelucrează și, mai mult decât atât, ele să formeze un modul, adică o unitate care separă implementarea de interfață, ceea ce implică posibilitatea refolosirii codului. A apărut astfel conceptul de [clasă](https://ro.wikipedia.org/wiki/Clas%C4%83). [Clasa](https://ro.wikipedia.org/wiki/Clasa) realizează, în speță, ceea ce am văzut mai înainte: grupează datele și unitățile de prelucrare a acestora într-un modul, unindu-le astfel într-o entitate mult mai naturală. Deși tehnica se numește "Programare Orientată Obiectual", conceptul de bază al ei este [Clasa](https://ro.wikipedia.org/wiki/Clasa). Clasa, pe lângă faptul că abstractizează foarte mult analiza/sinteza problemei, are proprietatea de generalitate, ea desemnând o mulțime de obiecte care împart o serie de proprietăți.

De exemplu: Clasa "floare" desemnează toate plantele care au flori, precum clasa "Fruct" desemnează toate obiectele pe care noi le identificam ca fiind fructe. Bineînțeles, în implementarea efectivă a programului nu se lucrează cu entități abstracte, precum clasele ci se lucrează cu obiecte, care sunt "instanțieri" ale claselor. Altfel spus, plecând de la exemplul de mai sus, dacă se construiește un program care să lucreze cu fructe, el nu va prelucra entitatea "fruct" ci va lucra cu entități concrete ale clasei "fruct", adică "măr", "pară", "portocală", etc.

Apare însă următoarea problemă: "cum se trece de la o structură generală la una particulară, mai precis ce înseamnă efectiv procesul de instanțiere?". Instanțierea (trecerea de la clasă la obiect) înseamnă atribuirea unor proprietăți specifice clasei, astfel încât aceasta să indice un obiect anume, care se diferențiază de toate celelalte obiecte din clasă printr-o serie de atribute. Dacă vom considera ca "fruct\_exotic" care desemnează clasa tuturor fructelor exotice ce conține proprietatea "culoare" atunci atribuind acesteia valoarea "galben" noi vom crea o nouă mulțime (clasa fructelor exotice care au culoarea galbenă) care este o subclasă a clasei "fruct\_exotic", deci realizăm astfel o particularizare. Mai mult decât atât, dacă vom adăuga noi și noi atribute vom individualiza clasa astfel încât să ajungem la un caz concret, care este Obiectul.

**2. Librăria SFML**

**2.1. Ce este SFML?**

**SFML** (Simple and Fast Multimedia Library) este o bibliotecă multimedia portabilă și ușor de folosit. Este scrisă în [C++](https://ro.wikipedia.org/wiki/C%2B%2B), dar poate fi și portată in [C](https://ro.wikipedia.org/wiki/C), [D](https://ro.wikipedia.org/wiki/D), [Python](https://ro.wikipedia.org/wiki/Python), [Ruby](https://ro.wikipedia.org/wiki/Ruby), [OCaml](https://ro.wikipedia.org/w/index.php?title=OCaml&action=edit&redlink=1), [.NET](https://ro.wikipedia.org/w/index.php?title=.NET&action=edit&redlink=1). Este o alternativă [orientată obiect](https://ro.wikipedia.org/w/index.php?title=Orientat%C4%83_obiect&action=edit&redlink=1) pentru [SDL](https://ro.wikipedia.org/w/index.php?title=SDL&action=edit&redlink=1). SFML este folosit în mod principal pentru jocuri.

SFML pune la dispoziție grafică 2D accelerată prin hardware, folosind [OpenGL](https://ro.wikipedia.org/wiki/OpenGL), și câteva module pentru a ușura programarea jocurilor și a aplicațiilor multimedia. SFML poate fi folosit și pentru furnizarea unei ferestre pentru OpenGL. Website-ul SFML oferă întregul SDK pentru download și tutoriale pentru a ajuta dezvoltatorii.

**2.2. Module SFML**

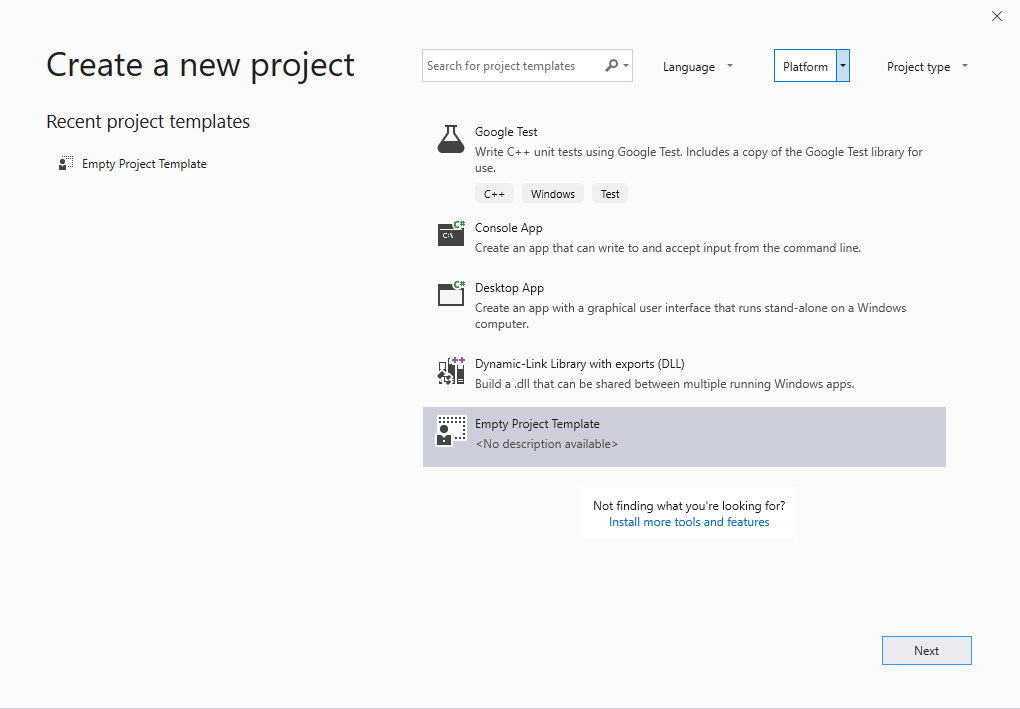
SFML conține o multitudine de module:

* Sistem - facilități de măsurare a timpului și crearea firelor de execuție portabile (threading)
* Fereastră (Window) - suport pentru management-ul comenzilor de input și al ferestrelor
* Grafică - accelerație hardware pentru grafică 2D (texturi, poligoane, text etc.)
* Audio
* Rețea (Network) - socket-uri de rețea TCP și UDP, facilități de încapsulare a datelor, clase HTTP și FTP

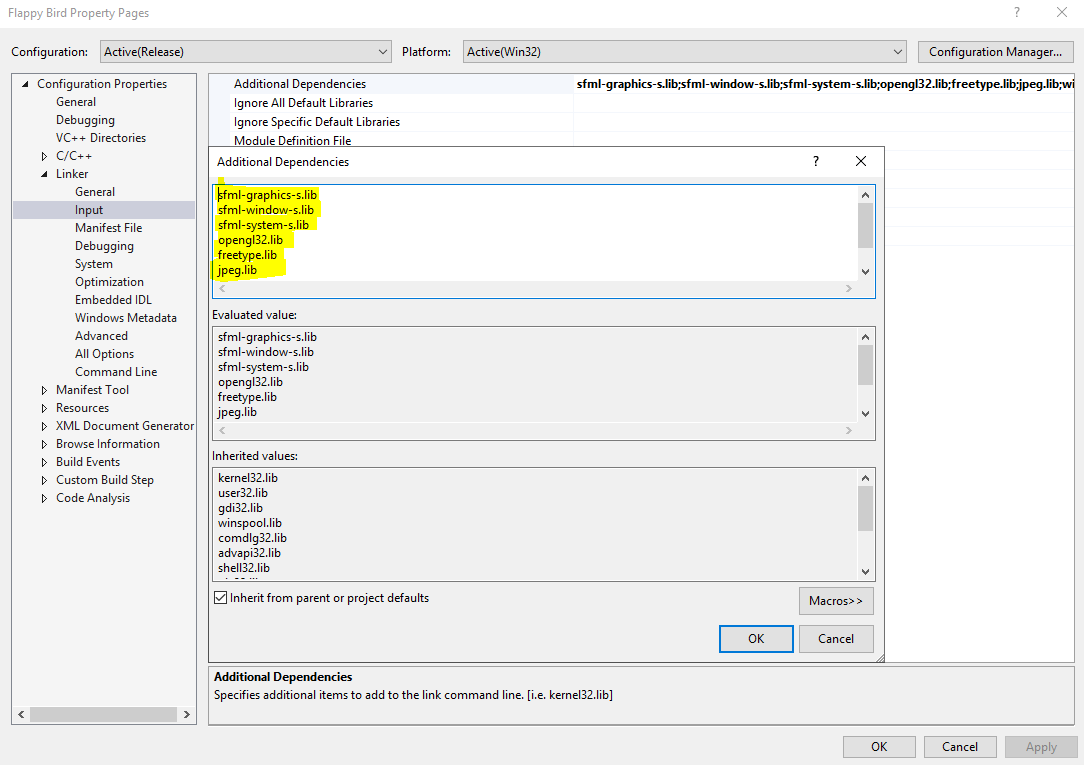
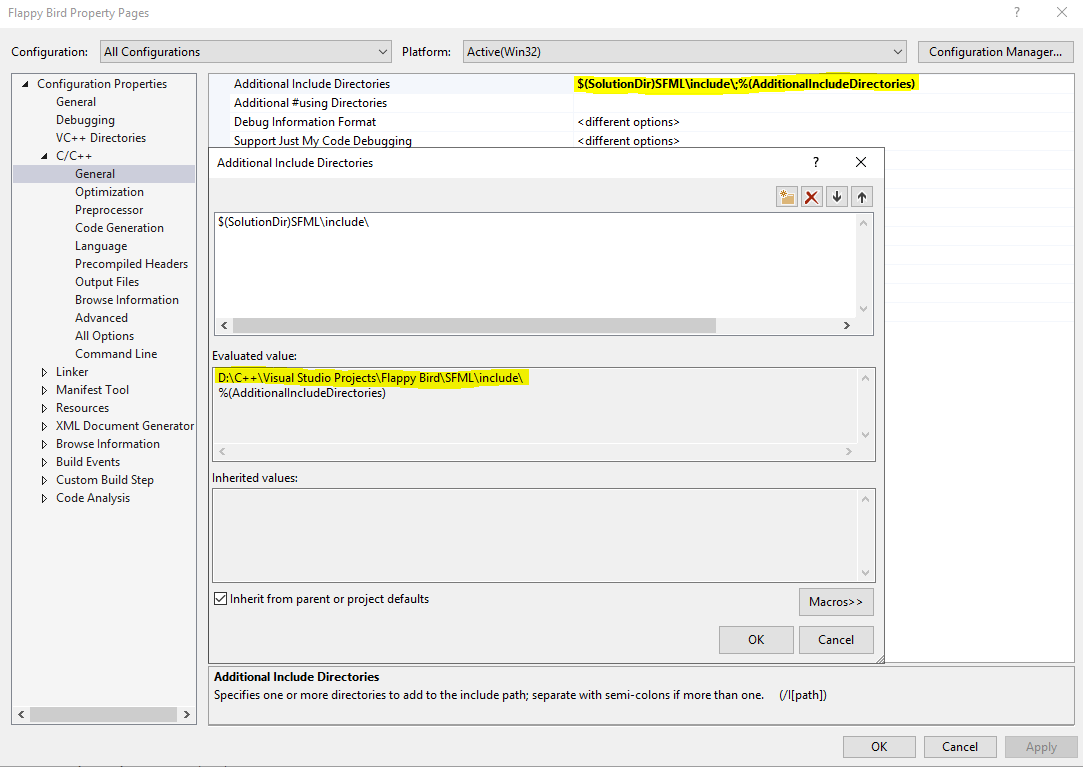
***Partea II. Descrierea realizării părţii practice***

Proiectul a fost realizat și compilat Visual Studio Preview 2019 folosind varianta 32bit a librăriilor adiacente (SFML, versiunea 2.4.2), iar codul sursă a fost adăugat într-un repository la adresa: <https://github.com/alexanderdx/Flappy-Bird>

Primul pas pentru realizarea proiectului a fost crearea unui nou proiect gol (“Empty Project Template”) în Visual Studio:



Următorul pas este configurarea proiectului - adăugarea header-elor “.h” și conectarea fișierelor “.lib” ale librăriei SFML la proiectul nostru prin indicarea, în setările proiectului, a locației acestora (Proces denumit “linking”).

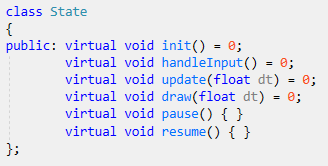


Din acest moment putem utiliza funcționalitățile librăriei SFML, ce ne va permite sa creăm texturi/sprite-uri (imagini) 2D, sa le animam și sa definim eventualele interacțiuni necesare intre acestea.

Codul proiectului a fost conceput, în mare parte, conform paradigmei de Programare Orientată pe Obiect (OOP, în engleză) axată pe ideea încapsulării, adică pe ideea grupării datelor și a codului ce operează asupra acestora într-o singura structură omogenă.

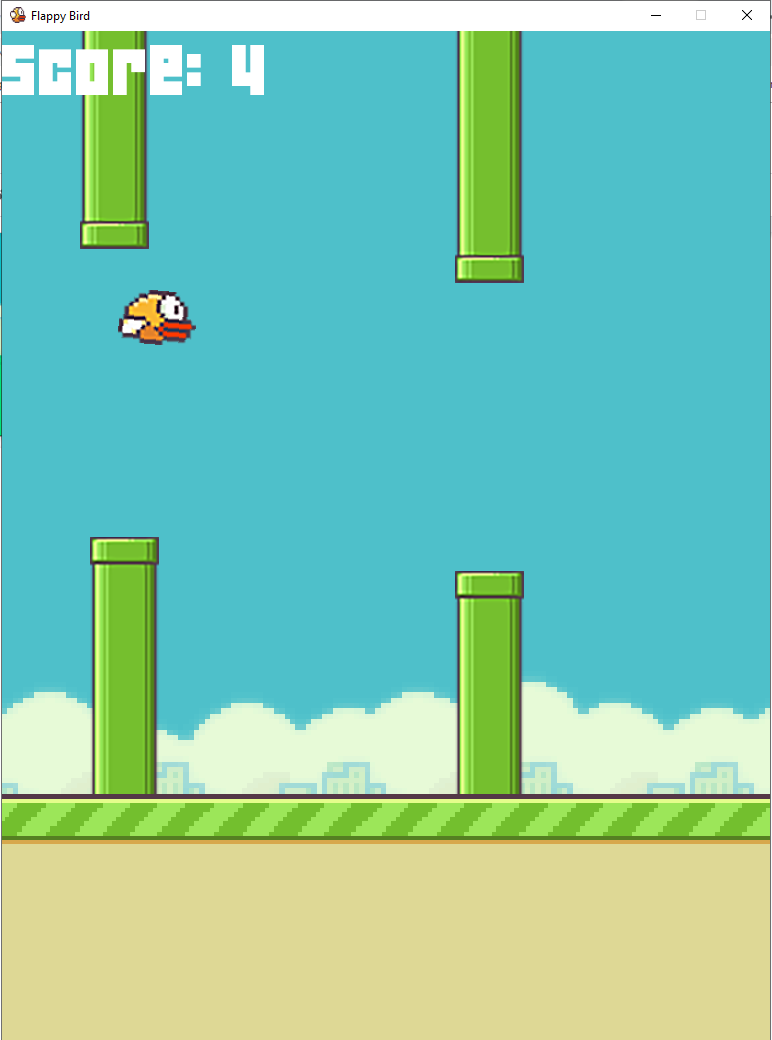
Adică, pe scurt, codul este structurat în multiple clase, fiecare clasă având un rol unic și abordând o anumita problemă.

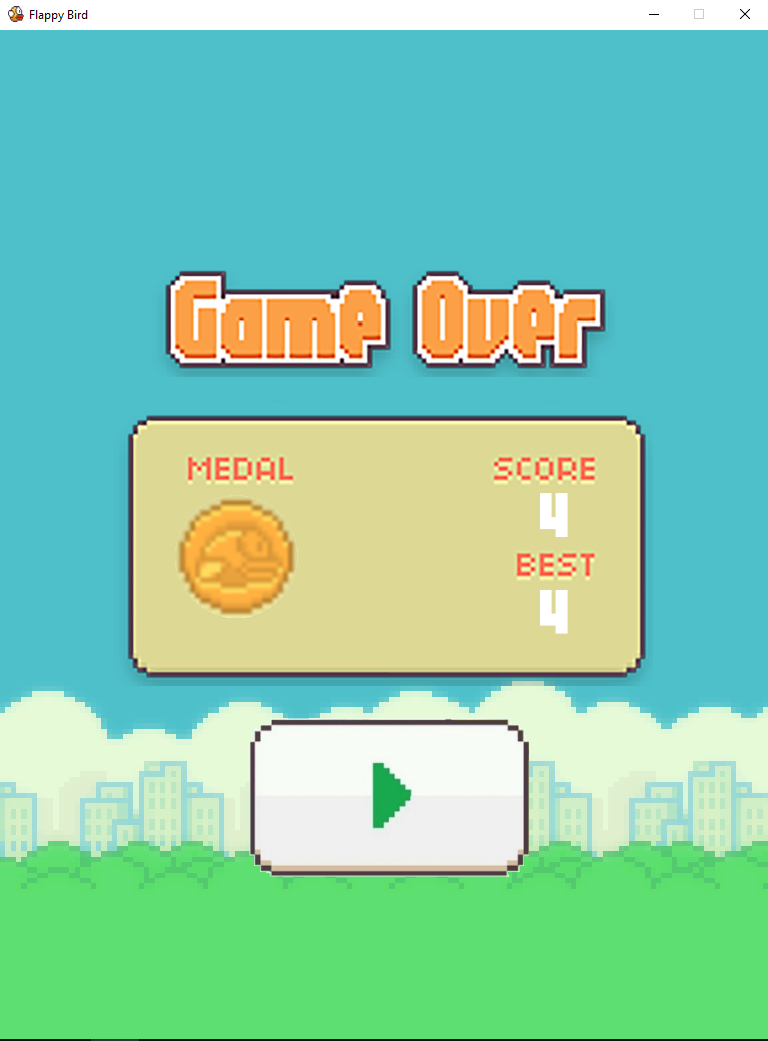
Prima clasă construită este următoarea:



Ea reprezintă un model, un șablon (marcat prin cuvântul cheie “virtual”) din care vor deriva multiple alte clase ce vor reprezenta diferite stări ale jocului:

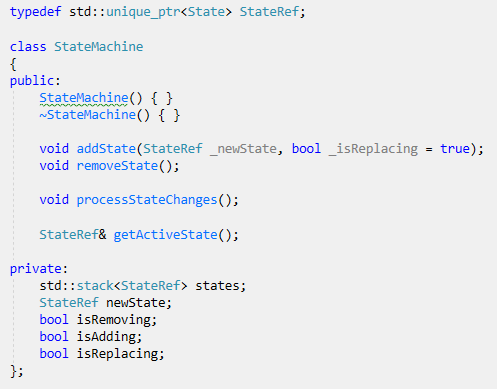
* Meniul principal (Main Menu)
* **Jocul în sine (Game playing)**

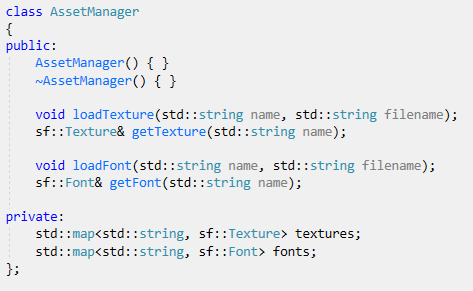


* **Sfarșit joc (Game Over)**

Clasa “StateMachine” se va ocupa, esențial, de schimbarea stării jocului când este necesar, acestea fiind stocate într-o stivă.

(De exemplu: schimbarea din starea “Game playing” -> “Game over” o dată ce ne-am ciocnit de o conductă)



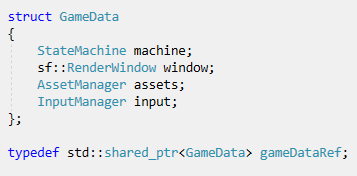
De asemenea, pe lângă manipularea stărilor jocului, avem nevoie de o modalitate de stocare a tuturor resurselor jocului (fonturi, texturi, imagini, sunete etc.). Acest lucru este realizat prin clasa “AssetManager”, definită astfel:

și, în sfărșit, o clasă ce se va ocupa cu verificarea comenzilor (input-urilor) date de utilizator (De exemplu, verificarea dacă un anumit buton, cum ar fi “Play Game”, este apăsat):



De menționat este faptul ca entitățile precedate de prefixul “sf::” fac parte din librăria SFML atașată proiectului.

Aceste clase au fost grupate împreună într-o structură care va fi marcată ca unica de-a lungul rulării programului și va conține toate datele jocului într-un moment dat: starea actuală, resursele încărcate în memorie, fereastra jocului.



Prin urmare, fiecare clasă va avea ca parametru un obiect de tip “gameDataRef” pentru a avea acces la informațiile stocate de acesta.

Acum vom explora structura fiecărei clase corespunzătoare celor 3 stări menționate mai sus.

* Meniul principal (Main Menu)

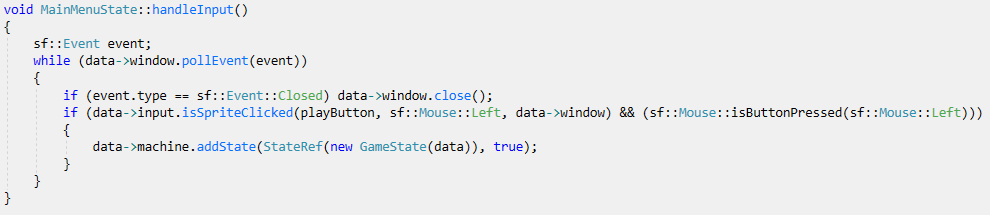
Să disecăm acum fiecare funcție în parte:

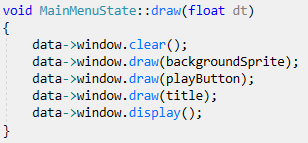


Acesta este constructorul clasei, nu face decăt să atribuie datele jocului către membrul local al clasei.



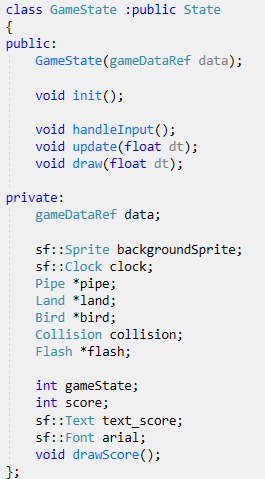
Funcția inițializează toate texturile și imaginile necesare meniului principal și setează poziția titlului și a butonului “Play”, relativ cu rezoluția ferestrei, însă nu le și afișează - de acest lucru se ocupă funcția “draw( )”.

Această funcție “ascultă” pentru comenzile utilizatorului. Dacă acesta apasă butonul “X”, fereastra se va închide. Dacă acesta apasă pe butonul “Play” atunci funcția adaugă o nouă stare de tip “Game State” în stiva stărilor, ceea ce va determina începerea jocului.

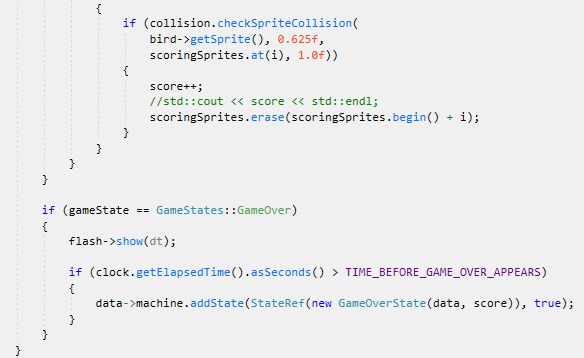


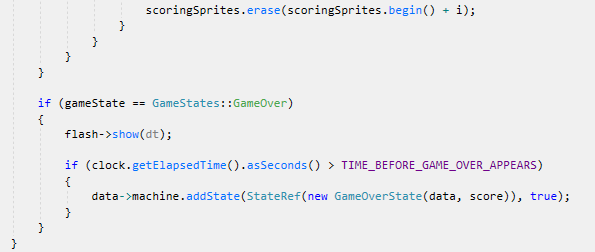
Afișează, în această ordine, imaginile și texturile aferente:

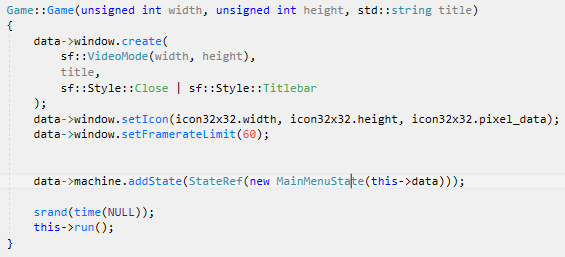
Imaginea de fundal -> butonul “Play” -> titlul “Flappy Bird”

* Game playing (Game State)

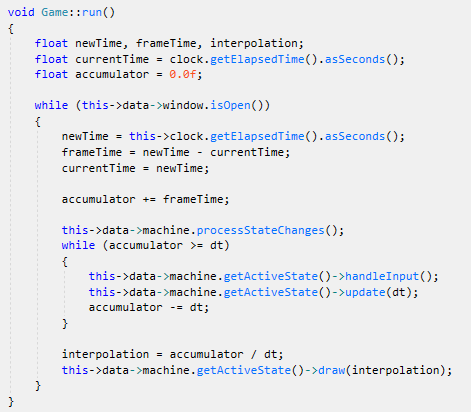
Identic cu corespondentul ei din clasa “MainState”, funcția “init( )” încarcă în memorie texturile necesare pentru pasăre, conducte, imaginea de fundal, pămănt și fontul scorului.





Funcția se ocupă cu verificarea coliziunilor dintre pasăre și pămănt/conducte, mișcarea acestora și actualizarea, după caz, a stării jocului.

Clasa “Game” va iniția jocul în sine - fereastra, iconiță -, setează limita cadrelor pe secundă la 60 și conține bucla principală a jocului:



Căt timp fereastra jocului e deschisă, este calculat timpul scurs dintre cadre și, în funcție de rata actualizării aleasă (“dt”) sunt procesate toate schimbările de stare ale jocului, iar pentru fiecare stare sunt executate operațiile aferente (desenarea elementelor, verificarea input-ului utilizatorului etc.).

***Bibliografie***

* <https://www.udemy.com/flappy-bird-sfml/>
* <https://ro.wikipedia.org/wiki/C%2B%2B>
* <https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte>
* <https://www.sfml-dev.org/tutorials/2.5/start-vc.php>
* <https://ro.wikipedia.org/wiki/SFML>