****

Flappy Bird IGRA

U Visual Studio razvojnom okruženju sa SFML bibliotekom

Seminar iz kolegija Računalna grafika (250)

IZRADILE:

Lucia Bolanča

Rea Sunara

**Sadržaj**

[1 Uvod 2](#_Toc532562259)

[1.1 Opis seminara 2](#_Toc532562260)

[1.2 Visual Studio 2](#_Toc532562261)

[1.3 SFML biblioteka 2](#_Toc532562262)

[2 Realizacija seminara (praktični dio) 3](#_Toc532562263)

[2.1 Postavljanje polaznog prozora aplikacije 3](#_Toc532562264)

[2.2 Automat igre 4](#_Toc532562265)

[2.2.1 State Machine 4](#_Toc532562266)

[2.2.2 Asset Manager 6](#_Toc532562267)

[2.2.3 Input Manager 7](#_Toc532562268)

[2.2.4 Game Loop 8](#_Toc532562269)

[2.3 Kreiranje stanja 10](#_Toc532562270)

[2.3.1 Splash stanje/prozor 10](#_Toc532562271)

[2.3.2 Main Menu stanje/prozor 12](#_Toc532562272)

[2.3.3 Game stanje/prozor 14](#_Toc532562273)

[2.3.4 Game Over stanje/prozor 16](#_Toc532562274)

[2.4 Implementacija logike igre 17](#_Toc532562275)

[2.4.1 Kreiranje klase Pipe 17](#_Toc532562276)

[2.4.2 Prikazivanje i pomicanje cijevi klase Pipe 19](#_Toc532562277)

[2.4.3 Automatsko kreiranje i brisanje cijevi klase Pipe 21](#_Toc532562278)

[2.4.4 Kreiranje klase Land za pomično dno 22](#_Toc532562279)

[2.4.5 Postavljanje nasumičnih visina cijevi 24](#_Toc532562280)

[2.4.6 Kreiranje klase Bird 25](#_Toc532562281)

[2.4.7 Animiranje ptice 27](#_Toc532562282)

[2.4.8 Realizacija kretanja ptice 28](#_Toc532562283)

[2.4.9 Realizacija rotacije ptice 30](#_Toc532562284)

[2.4.10 Sudar ptice s podom 31](#_Toc532562285)

[2.4.11 Sudar ptice s cijevima 33](#_Toc532562286)

[2.4.12 Pojava bljeska pri završetku igre 34](#_Toc532562287)

[2.4.13 Sustav za računanje rezultata 36](#_Toc532562288)

[2.4.14 Prikaz rezultata 38](#_Toc532562289)

[2.5 Game Over 40](#_Toc532562290)

[2.5.1 Setup 40](#_Toc532562291)

[2.5.2 Prikaz ostvarenog i najboljeg rezultata 43](#_Toc532562292)

[2.5.3 Prikaz osvojene medalje 44](#_Toc532562293)

[2.5.4 Dodavanje zvuka 45](#_Toc532562294)

[Popis literature 48](#_Toc532562295)

# Uvod

U ovom poglavlju izložene su opće informacije o seminaru, igri koju ćemo raditi te okruženju i dodatnoj biblioteci koju ćemo koristiti kako bi realizirali seminar.

## Opis seminara

U okviru ovog seminara realizirana je igra Flappy Bird za Windows OS uz koju smo se odlučili koristiti Visual Studio okruženju kako bi bila podržana na što više platformi.

## Visual Studio

Visual Studio je Microsoftovo integrirano razvojno okruženje (IDE). Koristi se za razvoj računalnih programa za Windows, web-stranica i aplikacija. Koristi Microsoftove platforme za razvoj poput aplikacijskog programskog sučelja (API) za Windows, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight.

Visual Studio sadrži uređivač izvornog koda (engl. *code editor*) koji podržava IntelliSense (komponenta koja predlaže ostatak koda) kao i refaktoriranje koda. Integrirani program za otklanjanje grešaka (engl. *debugger*) radi na nivou izvornog i mašinskog koda. Program također sadrži alate poput dizajnera oblika koji se koristi za izradu aplikacija s grafičkim korisničkim sučeljem, web-dizajnera, dizajnera klasa i dizajnera shema baza podataka. Prihvaća proširenja koja poboljšavaju funkcionalost na skoro svakom nivou dodajući podršku sistema za upravljanje izvornim kodom i dodajući nove nizove alata poput tekstualnih uređivača i vizualnih dizajnera.

Visual Studio podržava različite programske jezike i dozvoljava uređivaču koda i debuggeru da podržava gotovo bilo koji programski jezik, pod uvjetom da usluga za taj jezik postoji. Ugrađeni jezici su C, C++ i C++/CLI (preko Visual C++), VB.NET (preko Visual Basic .NET)-a, C# (preko Visual C#) i F#. Podrška za ostale programske jezike poput Pythona, Rubyja i ostalih dostupna je instalacijom jezičnih servisa koji se mogu zasebno instalirati, a također podržava XML, HTML, JavaScript i CSS. [1]

## SFML biblioteka

SFML (engl. *Simple and Fast Multimedia Library*) je multimedijalna, višeplatformna i višejezična bibilioteka. SFML pruža jednostavno sučelje za različite komponente računala kako bi olakšao razvoj igara i multimedijalnih aplikacija. Sastoji se od pet modula: sistem, prozor, grafika, zvuk i mreža. Omogućava da kompajliramo i pokrenemo aplikaciju na operacijskim sustavima Windows, Linux, macOS. Može se koristiti unutar projekata koji su napisani u C, .NET, Java, Ruby, Python, Go i drugim programskim jezicima. [2]

# Realizacija seminara (praktični dio)

U ovom poglavlju izložena je cjelokupna realizacija seminara. Koristit će se razvojno okruženje Visual Studio 2015 i SFML verzije 2.5.1 za Visual C++ 14 64-bit kako bi realizirali Windows Flappy Bird igru.

## Postavljanje polaznog prozora aplikacije

Kreirat ćemo Win32 konzolnu aplikaciju u VS 2015, te preuzeti SFML s interneta i nazvati njegov direktorij SFML. Napravit ćemo direktorij External Libraries unutar direktorija projekta i u njega ubaciti direktorij SFML te ćemo SFML uključiti u naš projekt tako što ćemo u svojstima projekta u C/C++/General kao dodatni include direktorij dodati putanju $(SolutionDir)/../External Libraries/SFML/include i u Linker/General kao dodatni direktorij biblioteke dodati putanju $(SolutionDir)/../External Libraries/SFML/lib. U Linker/Input treba kao dodatni dependency dodati biblioteke sfml-audio-d.lib, sfml-graphics-d.lib, sfml-window-d.lib, sfml-network-d.lib i sfml-system-d.lib gdje 'd' označava debug verziju. Kreiramo main.cpp te napišemo kod sa slike kako bi dobili aplikaciju koja u polaznom prozoru ne sadrži ništa.

#include <SFML/Graphics.hpp>

#include <iostream>

#define SCREEN\_WIDTH 1024

#define SCREEN\_HEIGHT 1024

using namespace sf;

int main()

{

RenderWindow window(VideoMode(SCREEN\_WIDTH, SCREEN\_HEIGHT), "Awesome SFML");

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

switch (event.type)

{

case Event::Closed:

window.close();

break;

}

}

window.clear();

window.display();

}

return EXIT\_SUCCESS;

}

## Automat igre

Automat igre sastoji se od kreiranja State Machine, Asset Manager, Input Manager i Game Loop, te predstavlja kostur našeg projekta. Razlog tome je što želimo odvojiti kod na što više manjih dijelova u različite datoteke i rasteretiti Main.cpp datoteku.

### State Machine

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati StateMachine.h, StateMachine.cpp i State.h datoteke. U State.h ćemo dodati klasu State u kojoj ćemo definirati metode koje će označavati trenutni status stanja.

#pragma once

namespace ComputerGraphics

{

class State {

public:

virtual void Init() = 0;

virtual void HandleInput() = 0;

virtual void Update( float dt ) = 0;

virtual void Draw( float dt ) = 0;

virtual void Pause() {}

virtual void Resume() {}

};

}

U StateMachine.h dodati ćemo klasu StateMachine koja će imate metode za upravljanje stanjima i promjenama njihovih svojstava.

#pragma once

#include <memory>

#include <stack>

#include "State.h"

using namespace std;

namespace ComputerGraphics

{

typedef unique\_ptr<State> StateRef;

class StateMachine

{

public:

StateMachine() {}

~StateMachine() {}

void AddState(StateRef newState, bool isReplacing = true);

void RemoveState();

void ProcessStateChanges();

StateRef &GetActiveState();

private:

stack<StateRef> \_states;

StateRef \_newState;

bool \_isRemoving;

bool \_isAdding;

bool \_isReplacing;

};

}

Zatim ćemo u StateMachine.cpp implementirati metode iz StateMachine.h.

#include "StateMachine.h"

using namespace std;

namespace ComputerGraphics

{

void StateMachine::AddState(StateRef newState, bool isReplacing)

{

this->\_isAdding = true;

this->\_isReplacing = isReplacing;

this->\_newState = move(newState);

}

void StateMachine::RemoveState()

{

this->\_isRemoving = true;

}

void StateMachine::ProcessStateChanges()

{

if (this->\_isRemoving && !this->\_states.empty())

{

this->\_states.pop();

if (!this->\_states.empty())

{

this->\_states.top()->Resume();

}

this->\_isRemoving = false;

}

if (this->\_isAdding)

{

if (!this->\_states.empty())

{

if (this->\_isReplacing)

{

this->\_states.pop();

}

else

{

this->\_states.top()->Pause();

}

}

this->\_states.push(move(this->\_newState));

this->\_states.top()->Init();

this->\_isAdding = false;

}

}

StateRef &StateMachine::GetActiveState()

{

return this->\_states.top();

}

}

Konačno, u Main.cpp ćemo obrisati sve iz prethodnog koraka te će ostati samo sljedeći kod.

#include <iostream>

int main()

{

return EXIT\_SUCCESS;

}

### Asset Manager

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati AssetManager.h i AssetManager.cpp datoteke. U AsserManager.h ćemo dodati klasu AssetManager u kojoj ćemo definirati metode za učitavanje teksture i fonta kako bi to mogli upotrijebiti za bilo koji objekt.

#pragma once

#include <map

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class AssetManager

{

public:

AssetManager() {}

~AssetManager() {}

void LoadTexture(string name, string fileName);

Texture &GetTexture(string name);

void LoadFont(string name, string fileName);

Font &GetFont(string name);

private:

map<string, Texture> \_textures;

map<string, Font> \_fonts;

};

}

U AssetManager.cpp ćemo implementirati metode koje smo napravili u AssetManager klasi.

#include "AssetManager.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

void AssetManager::LoadTexture(string name, string fileName)

{

Texture tex;

if (tex.loadFromFile(fileName))

{

this->\_textures[name] = tex;

}

}

Texture &AssetManager::GetTexture(string name)

{

return this->\_textures.at(name);

}

void AssetManager::LoadFont(string name, string fileName)

{

Font;

if (font.loadFromFile(fileName))

{

this->\_fonts[name] = font;

}

}

Font &AssetManager::GetFont(string name)

{

return this->\_fonts.at(name);

}

}

### Input Manager

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati InputManager.h i InputManager.cpp datoteke. U InputManager.h ćemo dodati klasu InputManager u kojoj ćemo definirati metode za hvatanje pozicije kursora i je li kliknut miš kada je kursor unutar prozora.

#pragma once

#include<SFML/Graphics.hpp>

using namespace sf;

namespace ComputerGraphics

{

class InputManager

{

public:

InputManager() {}

~InputManager() {}

bool IsSpriteClicked(Sprite object, Mouse::Button, RenderWindow &window);

Vector2i GetMousePosition(RenderWindow &window);

};

}

U InputManager.cpp ćemo implementirati metode koje smo napravili u InputManager klasi. Kao prozor ćemo definirati pravokutnik određenih dimenzija te ćemo gledati je li kliknut miš kada je kursor bio unutar tog pravokutnika.

#include "InputManager.h"

using namespace sf;

namespace ComputerGraphics

{

bool InputManager::IsSpriteClicked(Sprite object, Mouse::Button, RenderWindow &window)

{

if (Mouse::isButtonPressed(button))

{

IntRect tempRect(object.getPosition().x, object.getPosition().y, object.getGlobalBounds().width, object.getGlobalBounds().height);

if (tempRect.contains(Mouse::getPosition(window)))

{

return true;

}

}

return false;

}

Vector2i InputManager::GetMousePosition(RenderWindow &window)

{

return Mouse::getPosition(window);

}

}

### Game Loop

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati Game.h i Game.cpp datoteke. U Game.h ćemo dodati strukturu GameData koja se sastoji od upravljača stanjima, prozora, svojstava objekata i ulaza, te ćemo dodati klasu Game koja se sastoji od sata, reference na strukturu GameData i varijable dt koja će označavati željeni fps(engl. *frame per second*) našeg projekta.

#pragma once

#include <memory>

#include <string>

#include <SFML/Graphics.hpp>

#include "StateMachine.h"

#include "AssetManager.h"

#include "InputManager.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

struct GameData

{

StateMachine machine;

RenderWindow window;

AssetManager assets;

InputManager input;

};

typedef shared\_ptr<GameData> GameDataRef;

class Game

{

public:

Game(int width, int height, string title);

private:

const float dt = 1.0f / 60.0f;

Clock \_clock;

GameDataRef \_data = make\_shared<GameData>();

void Run();

};

}

U Game.cpp ćemo implementirati metode koje smo napravili u Game klasi. Stvorit ćemo prozor u kojem će se naša igra pokretati te ćemo pratiti vrijeme okvira igre radi redovitog ažuriranja sadržaja i akumuliranja ukupnog vremena rada igre. Za slučaj da je vrijeme okvira prebrzo smanjit ćemo ga na željeno vrijeme. Za vrijeme trajanja igre također ćemo hvatati aktivna stanja, njihove ulaze i ažurirati ih.

#include "Game.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Game::Game(int width, int height, string title)

{

\_data->window.create(VideoMode(width, height), title, Style::Close | Style::Titlebar);

this->Run();

}

void Game::Run()

{

float newTime, frameTime, interpolation;

float currentTime = this->\_clock.getElapsedTime().asSeconds();

float accumulator = 0.0f;

while (this->\_data->window.isOpen())

{

this->\_data->machine.ProcessStateChanges();

newTime = this->\_clock.getElapsedTime().asSeconds();

frameTime = newTime - currentTime;

if (frameTime > 0.25f)

{

frameTime = 0.25f;

}

currentTime = newTime;

accumulator += frameTime;

while (accumulator >= dt)

{

this->\_data->machine.GetActiveState()->HandleInput();

this->\_data->machine.GetActiveState()->Update(dt);

accumulator -= dt;

}

interpolation = accumulator / dt;

this->\_data->machine.GetActiveState()->Draw(interpolation);

}

}

}

U Main.cpp datoteci ćemo sada pokrenuti projekt koji i dalje ne bi trebao prikazivati ništa, ali smo napravili poslovnu logiku projekta.

#include <iostream>

#include "Game.h"

int main()

{

ComputerGraphics::Game(800, 600, "SFML Starter");

return EXIT\_SUCCESS;

}

## Kreiranje stanja

U ovom poglavlju ćemo kreirati sva moguća stanja aplikacije, koja su Splash Screen, Main Menu, Game Screen i Game Over Screen stanje.

### Splash stanje/prozor

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati SplashState.h, SplashState.cpp i DEFINITIONS.h datoteke. U Game.h ćemo uključiti biblioteku SplashState.h. Promijenit ćemo main.cpp sljedećim kodom:

#include <iostream>

#include "Game.h"

#include "DEFINITIONS.h"

int main()

{

ComputerGraphics::Game(SCREEN\_WIDTH, SCREEN\_HEIGHT, "Flappy Bird");

return EXIT\_SUCCESS;

}

Unutar DEFINITIONS.h datoteke ćemo definirati širinu i visinu prozora, vrijeme prikazivanja Splash prikaza i putanju do direktorija u kojem se nalazi slika koja će se prikazati.

#pragma once

#define SCREEN\_WIDTH 768

#define SCREEN\_HEIGHT 1024

#define SPLASH\_STATE\_SHOW\_TIME 3.0

#define SPLASH\_SCENE\_BACKGROUND\_FILEPATH "Resources/res/SplashBackground.png"

U SplashState.h ćemo dodati klasu SplashState koja će nasljeđivati klasu State.

#pragma once

#include <SFML/Graphics.hpp>

#include "State.h"

#include "Game.h"

using namespace sf;

namespace ComputerGraphics

{

class SplashState : public State

{

public:

SplashState(GameDataRef data);

void Init();

void HandleInput();

void Update(float dt);

void Draw(float dt);

private:

GameDataRef \_data;

Clock \_clock;

Texture \_backgroundTexture;

Sprite \_background;

};

}

U SplashState.cpp ćemo implementirati metode klase SplashState. Unutar metode Init() ćemo učitati teksturu pozadine, a unutar metode Update() ćemo ispisati poruku korisniku da je isteklo vrijeme prikazivanja Splash prikaza. Uz to ćemo još implementirati metode HandleInput() i Draw() koje sudjeluju u prikazu Splash screena.

#pragma once

#include <sstream>

#include "SplashState.h"

#include "DEFINITIONS.h"

#include <iostream>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

SplashState::SplashState(GameDataRef data) : \_data(data)

{

}

void SplashState::Init()

{

this->\_data->assets.LoadTexture("Splash State Background", SPLASH\_SCENE\_BACKGROUND\_FILEPATH);

\_background.setTexture(this->\_data->assets.GetTexture("Splash State Background"));

}

void SplashState::HandleInput()

{

Event;

while ( this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

}

}

void SplashState::Update(float dt)

{

if (this->\_clock.getElapsedTime().asSeconds() > SPLASH\_STATE\_SHOW\_TIME)

{

cout << "Go To Main Menu" <<endl;

}

}

void SplashState::Draw(float dt)

{

this->\_data->window.clear();

this->\_data->window.draw(this->\_background);

this->\_data->window.display();

}

}

### Main Menu stanje/prozor

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati MainMenuState.h i MainManu.cpp datoteke. U DEFINITIONS.h ćemo dodati putanje do pozadine, naslova i dugmeta za početak igre.

#define MAIN\_MENU\_BACKGROUND\_FILEPATH "Resources/res/sky.png"

#define GAME\_TITLE\_FILEPATH "Resources/res/title.png"

#define PLAY\_BUTTON\_FILEPATH "Resources/res/PlayButton.png"

U SplashState.cpp u metodi Update() promijenit ćemo da se nakon 3 sekunde prelazi u MainMenu prikaz.

void SplashState::Update(float dt)

{

if (this->\_clock.getElapsedTime().asSeconds() > SPLASH\_STATE\_SHOW\_TIME)

{

this->\_data->machine.AddState(StateRef(new MainMenuState(\_data)), true);

}

}

U MainMenuState.h ćemo dodati klasu MainMenuState koja će nasljeđivati klasu State.

#pragma once

#include <SFML/Graphics.hpp>

#include "State.h"

#include "Game.h"

using namespace sf;

namespace ComputerGraphics

{

class MainMenuState : public State

{

public:

MainMenuState(GameDataRef data);

void Init();

void HandleInput();

void Update(float dt);

void Draw(float dt);

private:

GameDataRef \_data;

Texture \_backgroundTexture;

Sprite \_background;

Sprite \_title;

Sprite \_playButton;

};

}

U MainMenuState.cpp ćemo implementirati metode klase MainMenuState. Unutar metode Init() ćemo učitati teksturu pozadine, naslova i dugmeta. Uz to ćemo još implementirati metode HandleInput() i Draw() koje sudjeluju u prikazu MainMenu screena.

#include <sstream>

#include "MainMenuState.h"

#include "DEFINITIONS.h"

#include <iostream>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

MainMenuState::MainMenuState(GameDataRef data) : \_data(data)

{

}

void MainMenuState::Init()

{

this->\_data->assets.LoadTexture("Main Menu Background", MAIN\_MENU\_BACKGROUND\_FILEPATH);

this->\_data->assets.LoadTexture("Game Title", GAME\_TITLE\_FILEPATH);

this->\_data->assets.LoadTexture("Play Button", PLAY\_BUTTON\_FILEPATH);

\_background.setTexture(this->\_data->assets.GetTexture("Main Menu Background"));

\_title.setTexture(this->\_data->assets.GetTexture("Game Title"));

\_playButton.setTexture(this->\_data->assets.GetTexture("Play Button"));

\_title.setPosition((SCREEN\_WIDTH / 2) - (\_title.getGlobalBounds().width / 2), \_title.getGlobalBounds().height / 2);

\_playButton.setPosition((SCREEN\_WIDTH / 2) - (\_playButton.getGlobalBounds().width / 2), (SCREEN\_HEIGHT / 2) - (\_playButton.getGlobalBounds().height / 2));

}

void MainMenuState::HandleInput()

{

Event;

while (this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

if (this->\_data->input.IsSpriteClicked(this->\_playButton, Mouse::Left, this->\_data->window))

{

cout << "Go To Game Screen" << endl;

}

}

}

void MainMenuState::Update(float dt)

{

}

void MainMenuState::Draw(float dt)

{

this->\_data->window.clear(Color::Red);

this->\_data->window.draw(this->\_background);

this->\_data->window.draw(this->\_title);

this->\_data->window.draw(this->\_playButton);

this->\_data->window.display();

}

}

### Game stanje/prozor

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati GameState.h i Game.cpp datoteke. U DEFINITIONS.h ćemo dodati putanju do pozadine prikaza igre.

#define GAME\_BACKGROUND\_FILEPATH "Resources/res/sky.png"

U MainMenuState.cpp u metodi Update() promijenit ćemo da se prelazi u Game prikaz.

{

this->\_data->machine.AddState(StateRef(new GameState(\_data)), true);

}

U GameState.h ćemo dodati klasu GameState koja će nasljeđivati klasu State.

#pragma once

#include <SFML/Graphics.hpp>

#include "State.h"

#include "Game.h"

using namespace sf;

namespace ComputerGraphics

{

class GameState : public State

{

public:

GameState(GameDataRef data);

void Init();

void HandleInput();

void Update(float dt);

void Draw(float dt);

private:

GameDataRef \_data;

Texture \_backgroundTexture;

Sprite \_background;

};

}

U GameState.cpp ćemo implementirati metode klase GameState. Unutar metode Init() ćemo učitati teksturu pozadine. Uz to ćemo još implementirati metode HandleInput() i Draw() koje sudjeluju u prikazu Game screena.

#include <sstream>

#include "GameState.h"

#include "DEFINITIONS.h"

#include <iostream>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

GameState::GameState(GameDataRef data) : \_data(data)

{

}

void GameState::Init()

{

cout << "Game State" << endl;

this->\_data->assets.LoadTexture("Game Background", GAME\_BACKGROUND\_FILEPATH);

\_background.setTexture(this->\_data->assets.GetTexture("Game Background"));

}

void GameState::HandleInput()

{

Event;

while (this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

}

}

void GameState::Update(float dt)

{

}

void GameState::Draw(float dt)

{

this->\_data->window.clear(Color::Red);

this->\_data->window.draw(this->\_background);

this->\_data->window.display();

}

}

### Game Over stanje/prozor

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati GameOverState.h i GameOver.cpp datoteke. U DEFINITIONS.h ćemo dodati putanju do pozadine prikaza igre.

#define GAME\_OVER\_BACKGROUND\_FILEPATH "Resources/res/sky.png"

U GameOverState.h ćemo dodati klasu GameOverState koja će nasljeđivati klasu State.

#pragma once

#include <SFML/Graphics.hpp>

#include "State.h"

#include "Game.h"

using namespace sf;

namespace ComputerGraphics

{

class GameOverState : public State

{

public:

GameOverState(GameDataRef data);

void Init();

void HandleInput();

void Update(float dt);

void Draw(float dt);

private:

GameDataRef \_data;

Texture \_backgroundTexture;

Sprite \_background;

};

}

U GameOverState.cpp ćemo implementirati metode klase GameOverState. Unutar metode Init() ćemo učitati teksturu pozadine. Uz to ćemo još implementirati metode HandleInput() i Draw() koje sudjeluju u prikazu GameOver screena.

#include <sstream>

#include "GameOverState.h"

#include "DEFINITIONS.h"

#include <iostream>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

GameOverState::GameOverState(GameDataRef data) : \_data(data)

{

}

void GameOverState::Init()

{

this->\_data->assets.LoadTexture("Game Over Background", GAME\_OVER\_BACKGROUND\_FILEPATH);

\_background.setTexture(this->\_data->assets.GetTexture("Game Over Background"));

}

void GameOverState::HandleInput()

{

Event;

while (this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

}

}

void GameOverState::Update(float dt)

{

}

void GameOverState::Draw(float dt)

{

this->\_data->window.clear(Color::Red);

this->\_data->window.draw(this->\_background);

this->\_data->window.display();

}

}

## Implementacija logike igre

U ovom poglavlju ćemo dodati programski kod vezan uz aplikacijsku logiku.

### Kreiranje klase Pipe

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati Pipe.h, Pipe.cpp. U DEFINITIONS.h ćemo dodati putanju do slike cijevi. Promijenit ćemo kod u GameState.h i u GameState.cpp.

#define PIPE\_UP\_FILEPATH "Resources/res/PipeUp.png"

#define PIPE\_UP\_FILEPATH "Resources/res/PipeUp.png"

U GameState.h dodajemo biblioteku:

#include "Pipe.h"

i kao privatnu varijablu GameState klase dodajemo:

Pipe \*pipe;

U Pipe.h ćemo dodati klasu Pipe.

#pragma once

#include <SFML/Graphics.hpp>

#include "Game.h"

#include <vector>

#include "DEFINITIONS.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class Pipe

{

public:

Pipe(GameDataRef data);

void DrawPipes();

private:

GameDataRef \_data;

vector<Sprite> pipeSprites;

};

}

U Pipe.cpp ćemo implementirati metode klase Pipe, točnije metodu DrawPipes() da crta cijevi u prozoru.

#include "Pipe.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Pipe::Pipe(GameDataRef data) : \_data(data)

{

}

void Pipe::DrawPipes()

{

for (unsigned short int i = 0; i < pipeSprites.size(); i++)

{

\_data->window.draw(pipeSprites.at(i));

}

}

}

U GameState.cpp ćemo unutar metode Init() dodati učitavanje teksture cijeve, a metodi Draw() ćemo pozvati metodu DrawPipes().

void GameState::Init()

{

this->\_data->assets.LoadTexture("Game Background", GAME\_BACKGROUND\_FILEPATH);

this->\_data->assets.LoadTexture("Pipe Up", PIPE\_UP\_FILEPATH);

this->\_data->assets.LoadTexture("Pipe Down", PIPE\_DOWN\_FILEPATH);

pipe = new Pipe(\_data);

\_background.setTexture(this->\_data->assets.GetTexture("Game Background"));

}

void GameState::Draw(float dt)

{

this->\_data->window.clear(Color::Red);

this->\_data->window.draw(this->\_background);

this->pipe->DrawPipes();

this->\_data->window.display();

}

### Prikazivanje i pomicanje cijevi klase Pipe

Preuredit ćemo trenutnu aplikaciju tako što ćemo unutar Pipe.h i Pipe.cpp dodati i implementirati metode za postavljanje, prikazivanje i pomicanje cijevi unutar prozora aplikacije, a promijenit ćemo kod u GameState.cpp.

U Pipe.h ćemo dodati public metode klase Pipe za prikazivanje i pomicanje cijevi.

#pragma once

#include <SFML/Graphics.hpp>

#include "Game.h"

#include <vector>

#include "DEFINITIONS.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class Pipe

{

public:

Pipe(GameDataRef data);

void SpawnBottomPipe();

void SpawnTopPipe();

void SpawnInvisiblePipe();

void MovePipes(float dt);

void DrawPipes();

private:

GameDataRef \_data;

vector<Sprite> pipeSprites;

};

}

U Pipe.cpp ćemo implementirati nadodane metode klase Pipe. Razlikovat ćemo dvije vrste cijevi, jedne koje će biti pozicionirane gore, a druge koje će biti pozicionirane dolje. Gornje cijevi ćemo kreirati u metodi SpawnTopPipe(), a za njihovu koordinatu apscise uzet ćemo veličinu prozora, a za koordinatu ordinate razliku veličine prozora i veličine cijevi. Donje cijevi ćemo kreirati u metodi SpawnDownPipe(), a za njihovu koordinatu apscise uzet ćemo veličinu prozora, a za koordinatu ordinate nula. Kreirat ćemo i nevidljivu cijev u metodi SpawnInvisiblePipe() koja će nam služiti kao pomoćna cijev. Pomicanje cijevi odredit ćemo u metodi MovePipes(float dt) koja će pomicati cijevi ulijevo ovisno o tome koliko se često prozor aplikacije ažurira.

#include "Pipe.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Pipe::Pipe(GameDataRef data) : \_data(data)

{

}

void Pipe::SpawnTopPipe()

{

Sprite sprite(\_data->assets.GetTexture("Pipe Up"));

sprite.setPosition(\_data->window.getSize().x, \_data->window.getSize().y - sprite.getGlobalBounds().height);

pipeSprites.push\_back(sprite);

}

void Pipe::SpawnBottomPipe()

{

Sprite sprite(\_data->assets.GetTexture("Pipe Down"));

sprite.setPosition(\_data->window.getSize().x, 0);

pipeSprites.push\_back(sprite);

}

void Pipe::SpawnInvisiblePipe()

{

Sprite sprite(\_data->assets.GetTexture("Pipe Up"));

sprite.setPosition(\_data->window.getSize().x, \_data->window.getSize().y - sprite.getGlobalBounds().height);

sprite.setColor(Color(0, 0, 0, 0));

pipeSprites.push\_back(sprite);

}

void Pipe::MovePipes(float dt)

{

for (unsigned short int i = 0; i < pipeSprites.size(); i++)

{

float movement = PIPE\_MOVEMENT\_SPEED \*dt;

pipeSprites.at(i).move(-movement, 0);

}

}

void Pipe::DrawPipes()

{

for (unsigned short int i = 0; i < pipeSprites.size(); i++)

{

\_data->window.draw(pipeSprites.at(i));

}

}

}

U GameState.cpp promijenit ćemo kod tako što ćemo pozivati metode za kreiranje cijevi kada lijevim klikom miša kliknemo unutar prozora aplikacije.

void GameState::HandleInput()

{

Event;

while (this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

if (this->\_data->input.IsSpriteClicked(\_background, Mouse::Left, this->\_data->window))

{

pipe->SpawnInvisiblePipe();

pipe->SpawnBottomPipe();

pipe->SpawnTopPipe();

}

}

}

### Automatsko kreiranje i brisanje cijevi klase Pipe

Preuredit ćemo trenutnu aplikaciju tako što ćemo unutar Pipe.cpp promijeniti metodu Update(float dt), a promijenit ćemo kod u DEFINITIONS.h, GameState.cpp i GameState.h.

U DEFINITIONS.h ćemo dodati brzini cijevi i frekvenciju kreiranja elemenata u prozoru.

#define PIPE\_MOVEMENT\_SPEED 200.0f

#define PIPE\_SPAWN\_FREQUENCY 0.5f

U GameState.h ćemo dodati privatnu varijablu klase GameState:

Clock clock;

U GameState.cpp mijenjamo metodu Update(float dt) kako bi se cijevi kreirale frekvencijom i brzinom kojom želimo.

void GameState::Update(float dt)

{

pipe->MovePipes(dt);

if(clock.getElapsedTime().asSeconds()>PIPE\_SPAWN\_FREQUENCY)

{

pipe->SpawnInvisiblePipe();

pipe->SpawnBottomPipe();

pipe->SpawnTopPipe();

clock.restart();

}

}

U Pipe.cpp mijenjamo metodu MovePipes(float dt) u kojoj ćemo hvatati poziciju cijevi te ako je cijela cijev izvan prozora aplikacije obrisat ćemo je, ako je unutar pomaknut ćemo je ulijevo.

void Pipe::MovePipes(float dt)

{

for (unsigned short int i = 0; i < pipeSprites.size(); i++)

{

if (pipeSprites.at(i).getPosition().x < 0 - pipeSprites.at(i).getGlobalBounds().width)

{

pipeSprites.erase(pipeSprites.begin() + i);

}

else

{

float movement = PIPE\_MOVEMENT\_SPEED \*dt;

pipeSprites.at(i).move(-movement, 0);

}

}

}

### Kreiranje klase Land za pomično dno

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati Land.h i Land.cpp. Promijenit ćemo kod u DEFINITIONS.h, GameState.h i GameState.cpp.

U DEFINITIONS.h ćemo dodati putanju do slike dna.

#define LAND\_FILEPATH "Resources/res/Land.png"

U GameState.h dodajemo biblioteku:

#include "Land.h"

i kao privatnu varijablu GameState klase dodajemo:

Land \*land;

U Land.h ćemo dodati klasu Land.

#pragma once

#include <SFML/Graphics.hpp>

#include "Game.h"

#include<vector>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class Land

{

public:

Land(GameDataRef data);

void MoveLand(float dt);

void DrawLand();

private:

GameDataRef \_data;

vector<Sprite>\_landSprites;

};

}

U Land.cpp ćemo implementirati metode klase Land. U konstruktoru klase smo dohvatili teksturu dna i postavili smo mu poziciju na dno prozora aplikacije. Kreirali smo dva dna za specijalni slučaj kada prikaz jednog dna nestane da i dalje ostaje drugo dno u prozoru aplikacije.

#include "Land.h"

#include "DEFINITIONS.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Land::Land(GameDataRef data) : \_data(data)

{

Sprite sprite(\_data->assets.GetTexture("Land"));

Sprite sprite2(\_data->assets.GetTexture("Land"));

sprite.setPosition(0, \_data->window.getSize().y - sprite.getGlobalBounds().height);

sprite2.setPosition(sprite.getGlobalBounds().width, \_data->window.getSize().y - sprite.getGlobalBounds().height);

\_landSprites.push\_back(sprite);

\_landSprites.push\_back(sprite2);

}

void Land::MoveLand(float dt)

{

for (unsigned short int i = 0; i < \_landSprites.size(); i++)

{

float movement = PIPE\_MOVEMENT\_SPEED \* dt;

\_landSprites.at(i).move(-movement, 0.0f);

if (\_landSprites.at(i).getPosition().x < 0 - \_landSprites.at(i).getGlobalBounds().width)

{

Vector2f position(\_data->window.getSize().x, \_landSprites.at(i).getPosition().y);

\_landSprites.at(i).setPosition(position);

}

}

}

void Land::DrawLand()

{

for (unsigned short int i = 0; i < \_landSprites.size(); i++)

{

\_data->window.draw(\_landSprites.at(i));

}

}

}

U GameState.cpp ćemo dodati sljedeće linije koda unutar metoda Init(), Update(float dt) i Draw(float dt) koje će kreirati, pomicati i prikazati dno brzine ovisne o tome koliko se često prozor aplikacije ažurira.

void GameState::Init()

{

this->\_data->assets.LoadTexture("Land", LAND\_FILEPATH);

land = new Land(\_data);

}

void GameState::Update(float dt)

{

land->MoveLand(dt);

}

void GameState::Draw(float dt)

{

this->land->DrawLand();

}

### Postavljanje nasumičnih visina cijevi

Preuredit ćemo trenutnu aplikaciju tako što ćemo promijeniti kod u Pipe.h,Pipe.cpp, DEFINITIONS.h i GameState.cpp.

U Pipe.h ćemo unutar klase Pipe dodati public metodu i dvije private varijable.

class Pipe

{

public:

void RandomisePipeOffset();

private:

int \_landHeight;

int \_pipeSpawnYOffset;

};

U Pipe.cpp ćemo implementirati nadodanu metodu klase Pipe RandomisePipeOffset() koja će računati nasumične vrijednosti visine dna na temelju kojih će se nasumično mijenjati visina tj.os ordinata gornjih i donjih cijevi u metodama SpawnBottomPipe() i SpawnTopPipe().

void Pipe::RandomisePipeOffset()

{

\_pipeSpawnYOffset = rand() % (\_landHeight + 1);

}

void Pipe::SpawnBottomPipe()

{

Sprite sprite(\_data->assets.GetTexture("Pipe Down"));

sprite.setPosition(\_data->window.getSize().x, -\_pipeSpawnYOffset); pipeSprites.push\_back(sprite);

}

void Pipe::SpawnTopPipe()

{

Sprite sprite(\_data->assets.GetTexture("Pipe Up"));

sprite.setPosition(\_data->window.getSize().x, \_data->window.getSize().y sprite.getGlobalBounds().height - \_pipeSpawnYOffset);

pipeSprites.push\_back(sprite);

}

U GameState.cpp Update() metodi ćemo promijeniti kod unutar if petlje da za svaku cijev poziva metodu koja će računati njenu nasumičnu vrijednost visine.

if(clock.getElapsedTime().asSeconds()>PIPE\_SPAWN\_FREQUENCY)

{

pipe->RandomisePipeOffset();

}

### Kreiranje klase Bird

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati Bird.h i Bird.cpp. Promijenit ćemo kod u DEFINITIONS.h, Game.cpp, GameState.h i GameState.cpp.

U DEFINITIONS.h ćemo dodati putanju do slika ptice. Postoje četiri različite slike ptica zbog različitih pozicija krila ptice.

#define BIRD\_FRAME\_1\_FILEPATH "Resources/res/bird-01.png"

#define BIRD\_FRAME\_2\_FILEPATH "Resources/res/bird-02.png"

#define BIRD\_FRAME\_3\_FILEPATH "Resources/res/bird-03.png"

#define BIRD\_FRAME\_4\_FILEPATH "Resources/res/bird-04.png"

U Bird.h ćemo dodati klasu Bird.

#pragma once

#include <SFML/Graphics.hpp>

#include "DEFINITIONS.h"

#include "Game.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class Bird

{

public:

Bird(GameDataRef data);

void DrawBird();

private:

GameDataRef \_data;

Sprite \_birdSprite;

};

}

U Bird.cpp ćemo implementirati metode klase Bird. U konstruktoru klase ćemo dodati kod koji će dohvatiti teksturu ptice i postaviti je na pticu, a u metodi DrawBird() će prikazati pticu u prozoru aplikacije.

#include "Bird.h"

#include "DEFINITIONS.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Bird::Bird(GameDataRef data) : \_data(data)

{

\_birdSprite.setTexture(\_data->assets.GetTexture("Bird Frame 1"));

}

void Bird::DrawBird()

{

\_data->window.draw(\_birdSprite);

}

}

U Game.cpp ćemo dodati kod:

srand(time(NULL));

U GameState.h dodajemo biblioteku:

#include "Bird.h"

i kao privatnu varijablu GameState klase dodajemo:

Bird \*bird;

U GameState.cpp ćemo dodati sljedeće linije koda unutar metoda Init() i Draw(float dt) koje će kreirati i prikazati pticu.

void GameState::Init()

{

this->\_data->assets.LoadTexture("Bird Frame 1", BIRD\_FRAME\_1\_FILEPATH);

bird = new Bird(\_data);

}

void GameState::Draw(float dt)

{

this->bird->DrawBird();

}

### Animiranje ptice

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u Bird.h, Bird.cpp, GameState.cpp i DEFINITIONS.h.

#define BIRD\_ANIMATION\_DURATION 0.4f

U Bird.h ćemo dodati jednu public metodu i tri private varijable klasi Bird:

class Bird

{

public:

void Animate(float dt);

private:

vector<Texture> \_animationFrames;

unsigned int \_animationIterator;

Clock \_clock;

};

U Bird.cpp ćemo dodati slike ptica kao okvire animacije i iterator za animaciju unutar konstruktora klase Bird kako bi kreirali animaciju te ćemo dodati metodu Animate(float dt) u kojoj ćemo mijenjati slike ptica nakon određenog vremena trajanja animacije pomoću iteratora kako bi nam se činilo da ptica maše krilima.

Bird::Bird(GameDataRef data) : \_data(data)

{

\_animationIterator = 0;

\_animationFrames.push\_back(\_data->assets.GetTexture("Bird Frame 1"));

\_animationFrames.push\_back(\_data->assets.GetTexture("Bird Frame 2"));

\_animationFrames.push\_back(\_data->assets.GetTexture("Bird Frame 3"));

\_animationFrames.push\_back(\_data->assets.GetTexture("Bird Frame 4"));

}

void Bird::Animate(float dt)

{

if (\_clock.getElapsedTime().asSeconds() > BIRD\_ANIMATION\_DURATION / \_animationFrames.size())

{

if (\_animationIterator < \_animationFrames.size() - 1)

{

\_animationIterator++;

}

else

{

\_animationIterator = 0;

}

\_birdSprite.setTexture(\_animationFrames.at(\_animationIterator));

\_clock.restart();

}

}

U GameState.cpp ćemo nadodati kod u metodu Init() koji će učitavati teksture za još tri slike ptica, a u metodi Update() pozvati funkciju Animate(dt) koja će pokrenuti animaciju brzine ovisne o tome koliko se često prozor aplikacije ažurira.

void GameState::Init()

{

this->\_data->assets.LoadTexture("Bird Frame 2", BIRD\_FRAME\_2\_FILEPATH);

this->\_data->assets.LoadTexture("Bird Frame 3", BIRD\_FRAME\_3\_FILEPATH);

this->\_data->assets.LoadTexture("Bird Frame 4", BIRD\_FRAME\_4\_FILEPATH);

}

void GameState::Update(float dt)

{

bird->Animate(dt);

}

### Realizacija kretanja ptice

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u Bird.h, Bird.cpp, GameState.cpp i DEFINITIONS.h.

U DEFINITIONS.h ćemo dodati linije koda koje će definirati brojeve koji označavaju stanja ptice, gravitaciju, brzinu i trajanje letenja:

#define BIRD\_STATE\_STILL 1

#define BIRD\_STATE\_FALLING 2

#define BIRD\_STATE\_FLYING 3

#define GRAVITY 300.0f

#define FLYING\_SPEED 350.0f

#define FLYING\_DURATION 0.25f

U Bird.h ćemo dodati dvije public metode i dvije private varijable klase Bird:

class Bird

{

public:

void Update(float dt);

void Tap();

private:

Clock \_movementClock;

int \_birdState;

};

U Bird.cpp ćemo implementirati metode Update(float dt) i Tap() klase Bird. U metodi Update(float dt) postaviti ćemo if petlje koje će provjeravati stanje ptice i ovisno o tome pomicati pticu, a u metodi Tap() ćemo postaviti stanje ptice na stanje letenja i resetirati sat koji prati kretnje tj.stanja ptice.

Bird::Bird(GameDataRef data) : \_data(data)

{

\_birdSprite.setTexture(\_animationFrames.at(\_animationIterator));

\_birdSprite.setPosition((\_data->window.getSize().x / 4) - (\_birdSprite.getGlobalBounds().width / 2), (\_data->window.getSize().y / 2) - (\_birdSprite.getGlobalBounds().height / 2));

\_birdState = BIRD\_STATE\_STILL;

}

void Bird::Update(float dt)

{

if (BIRD\_STATE\_FALLING == \_birdState)

{

\_birdSprite.move(0, GRAVITY \* dt);

}

else if (BIRD\_STATE\_FLYING == \_birdState)

{

\_birdSprite.move(0, -FLYING\_SPEED \* dt);

}

if (\_movementClock.getElapsedTime().asSeconds() > FLYING\_DURATION)

{

\_movementClock.restart();

\_birdState = BIRD\_STATE\_FALLING;

}

}

void Bird::Tap()

{

\_movementClock.restart();

\_birdState = BIRD\_STATE\_FLYING;

}

}

U GameState.cpp ćemo dodati if petlju u metodi HandleInput() koja će svaki put kada kliknemo u prozor aplikacije računati pozvati metodu Tap() klase Bird, a u metodi Update() ćemo dodati kod kojim ćemo ažurirati poziciju ptice.

void GameState::HandleInput()

{

while (this->\_data->window.pollEvent(event))

{

if (this->\_data->input.IsSpriteClicked(this->\_background, Mouse::Left, this->\_data->window))

{

bird->Tap();

}

}

}

void GameState::Update(float dt)

{

bird->Update(dt);

}

### Realizacija rotacije ptice

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u Bird.h, Bird.cpp i DEFINITIONS.h.

U DEFINITIONS.h ćemo dodati liniju koda koja će definirati brzinu rotacije:

#define ROTATION\_SPEED 100.0f

U Bird.h ćemo dodati private varijablu klase Bird:

class Bird

{

private:

float \_rotation;

};

U Bird.cpp ćemo dodati kod u metodu Update(float dt) i konstruktor klase Bird. U metodi Update(float dt) za svaku if petlju postavit ćemo rotaciju i njenu brzinu koja će ovisiti o stanju ptice tj. je li ptica pada ili se penje. U konstruktoru klase Bird ćemo dodati vektor kako bi postavili početno stanje i poziciju ptice.

Bird::Bird(GameDataRef data) : \_data(data)

{

Vector2f origin = Vector2f(\_birdSprite.getGlobalBounds().width / 2, \_birdSprite.getGlobalBounds().height / 2);

\_birdSprite.setOrigin(origin);

\_rotation = 0;

}

void Bird::Update(float dt)

{

if (BIRD\_STATE\_FALLING == \_birdState)

{

\_birdSprite.move(0, GRAVITY \* dt);

\_rotation += ROTATION\_SPEED \* dt;

if (\_rotation > 25.0f)

{

\_rotation = 25.0f;

}

\_birdSprite.setRotation(\_rotation);

}

else if (BIRD\_STATE\_FLYING == \_birdState)

{

\_birdSprite.move(0, -FLYING\_SPEED \* dt);

\_rotation -= ROTATION\_SPEED \* dt;

if (\_rotation < 25.0f)

{

\_rotation = -25.0f;

}

\_birdSprite.setRotation(\_rotation);

}

}

### Sudar ptice s dnom

Želimo postići da kada naša ptica udari u tlo da aplikacija detektira sudar i zamrzne se. Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u Bird.h, Bird.cpp, DEFINITIONS.h, Land.h, Land.cpp, GameState.cpp i GameState.h te dodati datoteke Collision.h i Collision.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati tri stanja igre:

enum GameStates

{

eReady,

ePlaying,

eGameOver

};

U Collision.h ćemo definirati klasu Collision i njenu public metodu CheckSpriteCollision koju ćemo implementirati u Collision.cpp.

#pragma once

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class Collision

{

public:

Collision();

bool CheckSpriteCollision(Sprite sprite1, Sprite sprite2);

};

}

#include "Collision.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Collision::Collision()

{

}

bool Collision::CheckSpriteCollision(Sprite sprite1, Sprite sprite2)

{

Rect<float> rect1 = sprite1.getGlobalBounds();

Rect<float> rect2 = sprite2.getGlobalBounds();

if (rect1.intersects(rect2))

return true;

else

return false;

}

}

U Bird.h ćemo dodati konstantnu funkciju tipa Sprite.

public:

const Sprite &GetSprite() const;

U Bird.cpp ćemo implementirati tu konstantnu funkciju.

const Sprite &Bird::GetSprite() const

{

return \_birdSprite;

}

U Land.h ćemo definirati konstantnu funkciju tipa vector<Sprite>.

public:

const vector<Sprite> &GetSprites() const;

U Land.cpp ćemo implementirati tu konstantnu funkciju.

const vector<Sprite> &Land::GetSprites() const

{

return \_landSprites;

}

U GameState.h ćemo dodati dvije privatne varijable klasi GameState.

Collision collision;

int \_gameState;

U GameState.cpp smo promijenili funkcije HandleInput() i Update(float dt).

void GameState::HandleInput()

{

Event event;

while (this->\_data->window.pollEvent(event))

{

...

if (this->\_data->input.IsSpriteClicked(this->\_background, Mouse::Left, this->\_data->window))

{

if (GameStates::eGameOver != \_gameState)

{

\_gameState = GameStates::ePlaying;

bird->Tap();

}

}

}

}

void GameState::Update(float dt)

{

if (GameStates::eGameOver != \_gameState)

{

bird->Animate(dt);

land->MoveLand(dt);

}

if (GameStates::ePlaying == \_gameState)

{

pipe->MovePipes(dt);

if (clock.getElapsedTime().asSeconds() > PIPE\_SPAWN\_FREQUENCY)

{

pipe->RandomisePipeOffset();

pipe->SpawnInvisiblePipe();

pipe->SpawnBottomPipe();

pipe->SpawnTopPipe();

clock.restart();

}

bird->Update(dt);

vector<Sprite> landSprites = land->GetSprites();

for (int i = 0; i < landSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), landSprites.at(i)))

{

\_gameState = GameStates::eGameOver;

}

}

}

}

### Sudar ptice s cijevima

Želimo postići da kada naša ptica udari u cijev da aplikacija detektira sudar i zamrzne se. Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u Collision.h, Collision.cpp, Pipe.h i Pipe.cpp.

Prvo ćemo u Pipe.h dodati konstantnu public funkciju GetSprites tipa vector<Sprite> te napisati njen kod u Pipe.cpp na isti način kao i kod Land.h i Land.cpp.

public:

const vector<Sprite> &GetSprites() const;

const vector<Sprite> &Pipe::GetSprites() const {

return pipeSprites;

}

U Collision.h i Collision.cpp ćemo preurediti funkciju CheckSpriteCollision funkciju u kojoj ćemo dodati dva nova argumenta tipa float za skaliranje argumenata tipa Sprite.

bool CheckSpriteCollision(Sprite sprite1, float scale1, Sprite sprite2, float scale2);

bool Collision::CheckSpriteCollision(Sprite sprite1, float scale1, Sprite sprite2, float scale2)

{

sprite1.setScale(scale1, scale1);

sprite2.setScale(scale2, scale2);

Rect<float> rect1 = sprite1.getGlobalBounds();

Rect<float> rect2 = sprite2.getGlobalBounds();

if (rect1.intersects(rect2))

return true;

else

return false;

}

Konačno, u GameState.cpp ćemo implementirati funkciju za detekciju sudara ptice s cijevima.

vector<Sprite> landSprites = land->GetSprites();

for (int i = 0; i < landSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.7f, landSprites.at(i), 1.1f))

{

\_gameState = GameStates::eGameOver;

}

}

vector<Sprite> pipeSprites = pipe->GetSprites();

for (int i = 0; i < pipeSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, pipeSprites.at(i), 1.0f))

{

\_gameState = GameStates::eGameOver;

}

}

### Pojava bljeska pri završetku igre

Želimo postići da kada naša ptica udari u pod ili cijev da aplikacija nakon detekcije sudara pokrene efekt bljeska ekrana. Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati Flash.h i Flash.cpp.

#pragma once

#include <SFML\Graphics.hpp>

#include "Game.h"

#include "DEFINITIONS.h"

namespace ComputerGraphics

{

class Flash

{

public:

Flash(GameDataRef data);

void Show(float dt);

void Draw();

private:

GameDataRef \_data;

RectangleShape \_shape;

bool \_flashOn;

};

}

#include "Flash.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

Flash::Flash(GameDataRef data) : \_data(data)

{

\_shape = RectangleShape(Vector2f(\_data->window.getSize()));

\_shape.setFillColor(Color(255, 255, 255, 0));

\_flashOn = true;

}

void Flash::Show(float dt)

{

if (\_flashOn)

{

int alpha = (int)\_shape.getFillColor().a + (FLASH\_SPEED \* dt);

if (alpha >= 255.0f)

{

alpha = 255.0f;

\_flashOn = false;

}

\_shape.setFillColor(Color(255, 255, 255, alpha));

}

else

{

int alpha = (int)\_shape.getFillColor().a - (FLASH\_SPEED \* dt);

if (alpha <= 0.0f)

{

alpha = 0.0f;

\_flashOn = false;

}

\_shape.setFillColor(Color(255, 255, 255, alpha));

}

}

void Flash::Draw()

{

\_data->window.draw(\_shape);

}

}

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameState.h i GameState.cpp. U DEFINITIONS.h ćemo dodati kod koji će definirati brzinu izvršavanja efekta bljeska:

#define FLASH\_SPEED 1500.0f

U GameState.h ćemo dodati privatni pokazivač na varijablu tipa Flash.

#include "Flash.h"

class GameState : public State

{

...

private:

...

Flash \*flash;

};

}

U GameState.cpp ćemo inicijalizirati flash varijablu te dodati posebni if uvjet unutar Update metode.

flash = new Flash(\_data);

...

if (GameStates::eGameOver == \_gameState)

{

flash->Show(dt);

}

}

void GameState::Draw(float dt)

{

.. .

this->flash->Draw();

}

### Sustav za računanje rezultata

Želimo ostvariti računanje rezultata tijekom igre. Kreirat ćemo nevidljivu cijev koja će služiti za uvećavanje rezultata za 1 kada ptica prođe kroz nju. Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, Pipe.h, Pipe.cpp, GameState.h i GameState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanju do nevidljive cijevi:

#define SCORING\_PIPE\_FILEPATH "Resources/res/InvisibleScoringPipe.png"

U Pipe.h preuredit ćemo kod tako da ćemo dodati public metode SpawnScorigSprites tipa void i &GetScoringSprites tipa vector<Sprite> te jedan private vektor scoringPipes s objektima tipa Sprite:

class Pipe

{

public:

...

void SpawnScoringPipe();

...

vector<Sprite> &GetScoringSprites();

private:

..

vector<Sprite> scoringPipes;

..

};

U Pipe.cpp ćemo dodati kod za metodu SpawnScoringSprites, a onda ćemo dopuniti metodu movePipes(float dt).

void Pipe::SpawnScoringPipe()

{

Sprite sprite(\_data->assets.GetTexture("Scoring Pipe"));

sprite.setPosition(\_data->window.getSize().x, 0);

scoringPipes.push\_back(sprite);

}

vector<Sprite> &Pipe::GetScoringSprites()

{

return scoringPipes;

}

... movePipes(float dt)

for (unsigned short int i = 0; i < scoringPipes.size(); i++)

{

if (scoringPipes.at(i).getPosition().x < 0 - scoringPipes.at(i).getGlobalBounds().width)

{

scoringPipes.erase(scoringPipes.begin() + i);

}

else

{

float movement = PIPE\_MOVEMENT\_SPEED \*dt;

scoringPipes.at(i).move(-movement, 0);

}

}

U GameState.h dodat ćemo privatnu varijablu tipa int za brojanje rezultata pa je koristiti skupa s metodama iz Pipe.h i Pipe.cpp unutar GameState.cpp u Update metodi sve dok traje igra:

#include "Flash.h"

class GameState : public State

{

...

private:

...

int \_score;

};

\_score = 0;

U Update(float dt)

void GameState::Update(float dt)

{

if (GameStates::eGameOver != \_gameState)

{

bird->Animate(dt);

land->MoveLand(dt);

}

if (GameStates::ePlaying == \_gameState)

{

pipe->MovePipes(dt);

if (clock.getElapsedTime().asSeconds() > PIPE\_SPAWN\_FREQUENCY)

{

pipe->RandomisePipeOffset();

pipe->SpawnInvisiblePipe();

pipe->SpawnBottomPipe();

pipe->SpawnTopPipe();

pipe->SpawnScoringPipe();

clock.restart();

}

...

if (GameStates::ePlaying == \_gameState)

{

vector<Sprite> &scoringSprites = pipe->GetScoringSprites();

for (int i = 0; i < scoringSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, scoringSprites.at(i), 1.0f))

{

\_score++;

cout << \_score << endl;

scoringSprites.erase(scoringSprites.begin() + i);

}

}

}

}

Zatim ćemo isto unutar GameState.cpp ubaciti teksturu za Scoring Pipe i postaviti varijablu \_score na nulu u Init metodi:

this->\_data->assets.LoadTexture("Scoring Pipe", SCORING\_PIPE\_FILEPATH);

\_score = 0;

### Prikaz rezultata

Želimo dodati prikaz rezultata tijekom trajanja igre. Kreirat ćemo tekst na ekranu s prikladnim fontom za prikaz rezultata. Dodat ćemo u trenutnu aplikaciju HUD.h i HUD.cpp.

#pragma once

#include <SFML/Graphics.hpp>

#include "DEFINITIONS.h"

#include "Game.h"

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

class HUD

{

public:

HUD(GameDataRef data);

void Draw();

void UpdateScore(int score);

private:

GameDataRef \_data;

Text \_scoreText;

};

}

#include "HUD.h"

#include <string>

using namespace std;

using namespace sf;

namespace ComputerGraphics

{

HUD::HUD(GameDataRef data) : \_data( data)

{

\_scoreText.setFont(\_data->assets.GetFont("Flappy Font"));

\_scoreText.setString("0");

\_scoreText.setCharacterSize(128);

\_scoreText.setFillColor(Color::White);

\_scoreText.setOrigin(\_scoreText.getGlobalBounds().width/2, \_scoreText.getGlobalBounds().height/ 2);

\_scoreText.setPosition(\_data->window.getSize().x/2, \_data->window.getSize().y/5);

}

void HUD::Draw()

{

\_data->window.draw(\_scoreText);

}

void HUD::UpdateScore(int score)

{

\_scoreText.setString(to\_string(score));

}

}

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameState.h i GameState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanju do fonta rezultata:

#define FLAPPY\_FONT\_FILEPATH "Resources/fonts/FlappyFont.ttf"

U GameState.h dodat ćemo privatni pokazivač na klasu HUD:

#include "HUD.h"

private:

HUD \*hud;

U GameState.cpp ćemo potom u Init metodi dohvatiti font, kreirati HUD objekt i ažurirati vrijednost trenutnog rezultata koji će se prikazati u Draw metodi.

this->\_data->assets.LoadFont("Flappy Font", FLAPPY\_FONT\_FILEPATH);

hud = new HUD(\_data);

hud->UpdateScore(\_score);

this->hud->Draw();

U Update metodi također ažurirati trenutni rezultat nakon detekcije sudara.

if (GameStates::ePlaying == \_gameState)

{

vector<Sprite> &scoringSprites = pipe->GetScoringSprites();

for (int i = 0; i < scoringSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, scoringSprites.at(i), 1.0f))

{

\_score++;

hud->UpdateScore(\_score);

scoringSprites.erase(scoringSprites.begin() + i);

}

}

}

## Game Over

U ovom poglavlju ćemo dodati programski kod vezan uz aplikacijsku logiku i prozor nakon završetka igre.

### Setup

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameState.cpp, GameOverState.h, i GameOverState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanju do pozadine i tijela za završetak igre te vrijeme:

#define GAME\_OVER\_TITLE\_FILEPATH "Resources/res/Game-Over-Title.png"

#define GAME\_OVER\_BODY\_FILEPATH "Resources/res/Game-Over-Body.png"

#define TIME\_BEFORE\_GAME\_OVER\_APPEARS 1.5f

U GameState.cpp resetirat ćemo vrijeme kada sw detektira sudar te dodati prijelaz u stanje GameOver:

#include "GameOverState.h"

vector<Sprite> landSprites = land->GetSprites();

for (int i = 0; i < landSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.7f, landSprites.at(i), 1.1f))

{

\_gameState = GameStates::eGameOver;

clock.restart();

}

}

vector<Sprite> pipeSprites = pipe->GetSprites();

for (int i = 0; i < pipeSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, pipeSprites.at(i), 1.0f))

{

\_gameState = GameStates::eGameOver;

clock.restart();

}

}

if (GameStates::eGameOver == \_gameState)

{

flash->Show(dt);

if (clock.getElapsedTime().asSeconds() > TIME\_BEFORE\_GAME\_OVER\_APPEARS)

{

\_data->machine.AddState(StateRef(new GameOverState(\_data)), true);

}

}

U GameOverState.h dodat ćemo tri privatne vrijable:

Sprite \_gameOverTitle;

Sprite \_gameOverContainer;

Sprite \_retryButton;

U GameOverState.cpp dodat ćemo sljedeći kod:

#include "GameState.h"

void GameOverState::Init()

{

this->\_data->assets.LoadTexture("Game Over Background", GAME\_OVER\_BACKGROUND\_FILEPATH);

this->\_data->assets.LoadTexture("Game Over Title", GAME\_OVER\_TITLE\_FILEPATH);

this->\_data->assets.LoadTexture("Game Over Body", GAME\_OVER\_BODY\_FILEPATH);

\_background.setTexture(this->\_data->assets.GetTexture("Game Over Background"));

\_gameOverTitle.setTexture(this->\_data->assets.GetTexture("Game Over Title"));

\_gameOverContainer.setTexture(this->\_data->assets.GetTexture("Game Over Body"));

\_retryButton.setTexture(this->\_data->assets.GetTexture("Play Button"));

\_gameOverContainer.setPosition((\_data->window.getSize().x/2)- (\_gameOverContainer.getGlobalBounds().width/2), (\_data->window.getSize().y / 2) - (\_gameOverContainer.getGlobalBounds().height / 2));

\_gameOverTitle.setPosition((\_data->window.getSize().x / 2) - (\_gameOverTitle.getGlobalBounds().width / 2), \_gameOverContainer.getPosition().y - (\_gameOverTitle.getGlobalBounds().height \* 1.2));

\_retryButton.setPosition((\_data->window.getSize().x / 2) - (\_retryButton.getGlobalBounds().width / 2), \_gameOverContainer.getPosition().y + \_gameOverContainer.getGlobalBounds().height + (\_retryButton.getGlobalBounds().height \* 0.2));

}

void GameOverState::HandleInput()

{

Event event;

while (this->\_data->window.pollEvent(event))

{

if (Event::Closed == event.type)

{

this->\_data->window.close();

}

if (\_data->input.IsSpriteClicked(\_retryButton, Mouse::Left, \_data->window))

{

\_data->machine.AddState(StateRef(new GameState(\_data)), true);

}

}

}

void GameOverState::Draw(float dt)

{

this->\_data->window.clear(Color::Red);

this->\_data->window.draw(this->\_background);

this->\_data->window.draw(this->\_gameOverTitle);

this->\_data->window.draw(this->\_gameOverContainer);

this->\_data->window.draw(this->\_retryButton);

this->\_data->window.display();

}

### Prikaz ostvarenog i najboljeg rezultata

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameState.cpp, GameOverState.h i GameOverState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanju do tekstualne datoteke u kojoj ćemo čuvati najbolji ostvareni rezultat:

#define HIGH\_SCORE\_FILEPATH "Resources/Highscore.txt"

U GameState.cpp ćemo dopuniti AddState metodu() unutar Update(float dt) funkcije u kojoj će GameOverState objekt prilikom kreiranja uz argument \_data primati i argument \_score:

if (GameStates::eGameOver == \_gameState)

{

flash->Show(dt);

if (clock.getElapsedTime().asSeconds() > TIME\_BEFORE\_GAME\_OVER\_APPEARS)

{

\_data->machine.AddState(StateRef(new GameOverState(\_data, \_score)), true);

}

}

S obzirom da se za prikaz koriste varijable za prikaz rezultata, potrebno ih je navesti kao privatne varijable u GameOverState.h:

int \_score;

int \_highScore;

Zatim ćemo u GameOverState.cpp u Draw() metodi nadodati istu stvar kao u GameState.cpp te u Init() metodi definirati font, veličinu, boju i poziciju teksta za prikaz ostvarenog i najboljeg rezultata:

\_scoreText.setFont(\_data->assets.GetFont("Flappy Font"));

\_scoreText.setString(to\_string(\_score));

\_scoreText.setCharacterSize(56);

\_scoreText.setFillColor(Color::White);

\_scoreText.setOrigin(\_scoreText.getGlobalBounds().width/2, \_scoreText.getGlobalBounds().height/2);

\_scoreText.setPosition(\_data->window.getSize().x / 10 \* 7.25, \_data->window.getSize().y / 2.15);

\_highScoreText.setFont(\_data->assets.GetFont("Flappy Font"));

\_highScoreText.setString(to\_string(\_highScore));

\_highScoreText.setCharacterSize(56);

\_highScoreText.setFillColor(Color::White);

\_highScoreText.setOrigin(\_highScoreText.getGlobalBounds().width / 2, \_highScoreText.getGlobalBounds().height / 2);

\_highScoreText.setPosition(\_data->window.getSize().x / 10 \* 7.25, \_data->window.getSize().y / 1.78);

Draw(float dt)

this->\_data->window.draw(this->\_scoreText);

this->\_data->window.draw(this->\_highScoreText);

GameOverState::GameOverState(GameDataRef data, int score) : \_data(data), \_score(score)

Zatim ćemo u istoj datoteci ubaciti funkciju za učitavanje najboljeg rezultata iz datoteke:

void GameOverState::Init()

{

ifstream readFile;

readFile.open(HIGH\_SCORE\_FILEPATH);

if (readFile.is\_open())

{

while (!readFile.eof())

{

readFile >> \_highScore;

}

}

readFile.close();

ofstream writeFile(HIGH\_SCORE\_FILEPATH);

if (writeFile.is\_open())

{

if (\_score > \_highScore)

{

\_highScore = \_score;

}

writeFile << \_highScore;

}

writeFile.close();

### Prikaz osvojene medalje

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameOverState.h i GameOverState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanje do slika medalja te broj bodova potrebnih za njihovo osvajanje:

#define BRONZE\_MEDAL\_FILEPATH "Resources/res/Bronze-Medal.png"

#define SILVER\_MEDAL\_FILEPATH "Resources/res/Silver-Medal.png"

#define GOLD\_MEDAL\_FILEPATH "Resources/res/Gold-Medal.png"

#define PLATINUM\_MEDAL\_FILEPATH "Resources/res/Platinum-Medal.png"

#define BRONZE\_MEDAL\_SCORE 0

#define SILVER\_MEDAL\_SCORE 10

#define GOLD\_MEDAL\_SCORE 25

#define PLATINUM\_MEDAL\_SCORE 100

U GameOverState.h ćemo dodati privatnu varijablu tipa Sprite za prikaz medalje:

Sprite \_medal;

U GameOverState.cpp ćemo dodati sljedeći kod:

void GameOverState::Init()

{

...

this->\_data->assets.LoadTexture("Bronze Medal", BRONZE\_MEDAL\_FILEPATH);

this->\_data->assets.LoadTexture("Silver Medal", SILVER\_MEDAL\_FILEPATH);

this->\_data->assets.LoadTexture("Gold Medal", GOLD\_MEDAL\_FILEPATH);

this->\_data->assets.LoadTexture("Platinum Medal", PLATINUM\_MEDAL\_FILEPATH);

...

if (\_score >= PLATINUM\_MEDAL\_SCORE)

{

\_medal.setTexture(\_data->assets.GetTexture("Platinum Medal"));

}

else if (\_score >= GOLD\_MEDAL\_SCORE)

{

\_medal.setTexture(\_data->assets.GetTexture("Gold Medal"));

}

else if (\_score >= SILVER\_MEDAL\_SCORE)

{

\_medal.setTexture(\_data->assets.GetTexture("Silver Medal"));

}

else

{

\_medal.setTexture(\_data->assets.GetTexture("Bronze Medal"));

}

\_medal.setPosition(175, 465);

}

void GameOverState::Draw(float dt)

{

...

this->\_data->window.draw(this->\_medal);

...

}

### Dodavanje zvuka

Preuredit ćemo trenutnu aplikaciju tako što ćemo dodati kod u DEFINITIONS.h, GameState.h i GameState.cpp.

U DEFINITIONS.h ćemo dodati kod koji će definirati putanje do zvučnih datoteka za let ptice, osvajanje boda i sudaranje s preprekom:

#define HIT\_SOUND\_FILEPATH "Resources/audio/Hit.wav"

#define POINT\_SOUND\_FILEPATH "Resources/audio/Point.wav"

#define WING\_SOUND\_FILEPATH "Resources/audio/Wing.wav"

U GameState.h ćemo u klasi GameState dodati privatne varijable za zvukove:

SoundBuffer \_hitSoundBuffer;

SoundBuffer \_wingSoundBuffer;

SoundBuffer \_pointSoundBuffer;

Sound \_hitSound;

Sound \_wingSound;

Sound \_pointSound;

U GameState.cpp ćemo dodati pozivanje play metoda za zvuk na za to predviđena mjesta:

void GameState::Init()

{

if (!\_hitSoundBuffer.loadFromFile(HIT\_SOUND\_FILEPATH))

{

cout << "Error loading Hit Sound Effect" << std::endl;

}

if (!\_pointSoundBuffer.loadFromFile(POINT\_SOUND\_FILEPATH))

{

cout << "Error loading Point Sound Effect" << std::endl;

}

if (!\_wingSoundBuffer.loadFromFile(WING\_SOUND\_FILEPATH))

{

cout << "Error loading Wing Sound Effect" << std::endl;

}

\_hitSound.setBuffer(\_hitSoundBuffer);

\_pointSound.setBuffer(\_pointSoundBuffer);

\_wingSound.setBuffer(\_wingSoundBuffer);

...

}

void GameState::HandleInput()

{

...

if (GameStates::eGameOver != \_gameState)

{

\_gameState = GameStates::ePlaying;

bird->Tap();

\_wingSound.play();

}

}

}

}

void GameState::Update(float dt)

{

for (int i = 0; i < landSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.7f, landSprites.at(i), 1.1f))

{

\_gameState = GameStates::eGameOver;

clock.restart();

\_hitSound.play();

}

}

for (int i = 0; i < pipeSprites.size(); i++)

{

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, pipeSprites.at(i), 1.0f))

{

\_gameState = GameStates::eGameOver;

clock.restart();

\_hitSound.play();

}

}

if (GameStates::ePlaying == \_gameState)

{

...

if (collision.CheckSpriteCollision(bird->GetSprite(), 0.625f, scoringSprites.at(i), 1.0f))

{

\_score++;

hud->UpdateScore(\_score);

scoringSprites.erase(scoringSprites.begin() + i);

\_pointSound.play();

}

}

}

# Popis literature

1. https://visualstudio.microsoft.com/vs/
2. https://www.sfml-dev.org/