Оглавление

Общая информация

Тема занятия

Изучение графической библиотеки SFML.

Цели занятия

1. Изучить возможности библиотеки SFML.
2. Изучить возможности создания динамической сцены с объектами в библиотеке SFML.
3. Применить изученные навыки для создания графического или игрового приложения.

Задачи занятия

1. Ознакомить Слушателей с возможностями библиотеки SFML.
2. Освоить отрисовку объектов с текстурами.
3. Выполнить практические задания.

Ожидаемый результат

По результатам занятия Учащийся должен уметь:

1. Использовать библиотеку SFML в своих программных решениях.
2. Применять изученные базовые алгоритмы по созданию графический и игровых приложений.
3. Понимать значение и место нового изученного материала в общей системе знаний по предмету.

Структура занятия

Тайминг занятия

Таблица 1

№	Этапы	Что делает Преподаватель	Что делает Слушатель	Время	Общее время
1	Приветствие. Проверка посещаемости.	Приветствует Слушателей, отмечает присутствующих.	Приветствуют преподавателя, отзываются на свою фамилию	5 мин.	5 мин.
2	Блиц опрос по тематике пройденного материала. Повестка текущего занятия.	Задает вопросы по теме прошлых занятий, оглашает план работы на текущем.	Поднимая руку, коротко отвечают на вопросы, задают вопросы преподавателю.	5 мин.	10 мин.
3	Теоретическая часть занятия.	Рассказывает про структуру приложения и UML-диаграммы. Объясняет значение основных классов и методов данного приложения.	Слушают, задают вопросы по теме занятия.	5 мин.	
5	Создание графического приложения. Включая: Рубрика «Вопрос – ответ» (2 вопроса)	Объясняет процесс создания графического приложения с использованием библиотеки SFML. Демонстрирует на экране различные варианты реализации. Задает тематические вопросы.	Повторяют действия преподавателя за своими рабочими местами, задают вопросы по теме занятия. Отвечают на тематические вопросы.	45 мин.	45 мин.
8	Практическая часть, решение задач.	Показывает на экране задание. Краткий обзор задачи.	Решают задачу, по сложности соответствующую уровню подготовки. Отвечают на вопросы.	20 мин.	30 мин.
9	Рефлексия	Обсуждает со Слушателями темы, пройденные на данном занятии	Участвуют в обсуждении	10 мин.	

Методика проведения

Приветствие, перекличка

Занятия необходимо начинать с приветствия учащихся, на первых занятиях представляться. Перекличка – это обязательная часть любой активности, как мастер класса, так и курса. Провести перекличку и отметить присутствующих в журнале. После окончания занятий журнал посещаемости необходимо передать администратору или вашему куратору.

Блиц-опрос

Блиц-опрос – еще одна важная часть, которую следует проводить на каждом занятии. Необходимо провести краткий фронтальный опрос учащихся выборочно, спрашивать по основным темам предыдущего занятия, 5-7 вопросов на аудиторию более чем достаточно.

Повестка занятия

Повестка – также обязательная часть любого занятия. Озвучивается, что непосредственно будет изучено на занятии, транслируется на экран ожидаемый результат занятия, подробно объясняется преподавателем, какие темы будут затронуты.

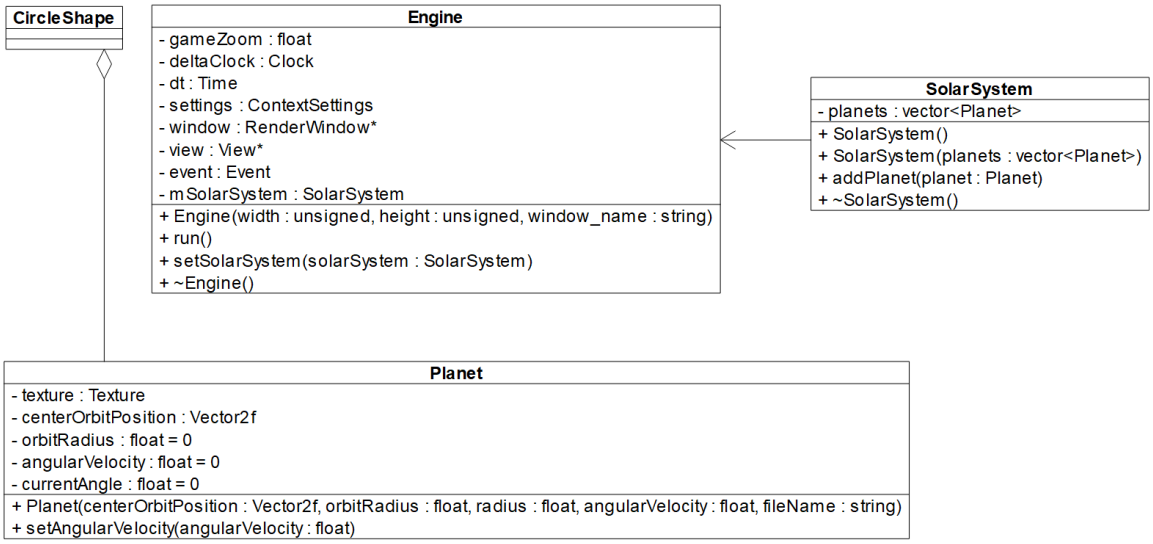
Теоретическая часть занятия.

Слайд 1 (UML-диаграмма классов)

UML *(англ. Unified Modeling Language — унифицированный язык моделирования)* — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Обозначения

UML-диаграмма классов проекта



Слайд 2 (Класс Planet)

Сначала нам понадобится класс Planet для хранения «планеты».

```
#pragma once
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>

class Planet : public sf::CircleShape
{
public:
	Planet(sf::Vector2f, float, float, float, std::string);

	void setAngularVelocity(float);

private:
	friend class Engine;
	sf::Texture texture;
	sf::Vector2f centerOrbitPosition;
	float orbitRadius = 0.0f;
	float angularVelocity = 0.0f;
	float currentAngle = 0.0f;
};
```

Слайд 3 (Класс Planet)

Просто класс планеты нам не совсем подойдет, будет более правильным решением наследовать класс CircleShape.

```
class Planet : public sf::CircleShape
```

Класс Engine будет обрабатывать все объекты «симулятора», поэтому лучше заранее сказать, что класс Engine будет иметь доступ ко всем приватным полям класса Planet.

```
friend class Engine;
sf::Texture texture;
sf::Vector2f centerOrbitPosition;
float orbitRadius = 0.0f;
float angularVelocity = 0.0f;
float currentAngle = 0.0f;
```

Также у планеты будет определенный набор свойств. Это texture – текстура планеты, centerOrbitPosition – центр «круговой» орбиты, orbitRadius – радиус орбиты, angularVelocity – угловая скорость, currentAngle – текущий угол поворота планеты.

Слайд 4 (Класс Planet)

Далее необходимо описать методы класса Planet.

Проинициализируем поля класса и установим все необходимые значения. При загрузке текстуры вы можете использовать другой подход проверки, а в данном примере демонстрируется обработка исключений.

```
Planet::Planet(sf::Vector2f centerOrbitPosition,
	float orbitRadius,
	float radius,
	float angularVelocity,
	std::string fileName) {

	this->orbitRadius = orbitRadius;
	this->centerOrbitPosition = centerOrbitPosition;
	this->angularVelocity = angularVelocity;
	setRadius(radius);
	setPosition(centerOrbitPosition);
	setOrigin(getRadius(), getRadius());

	try
	{
		if (!texture.loadFromFile(fileName))
		{
			throw "Texture wasn\'t loaded.";
		}
	}
	catch (const std::exception& e)
	{
		std::cerr << e.what() << std::endl;
	}

	setTexture(&texture);
}
```

Слайд 5 (Класс Planet)

Реализуем (сеттер) метод установки угловой скорости.

```
void Planet::setAngularVelocity(float angularVelocity) {
	this->angularVelocity = angularVelocity;
}
```

Рубрика «Вопрос – ответ»

Слайд 6 (Вопрос)

Вопрос: Что такое ключевое слово this?
Зачем оно используется?

Слайд 7 (Ответ)

Ответ: Методы каждого объекта имеют доступ к некому волшебному указателю под названием **this**, который ссылается на сам объект. Таким образом, любой метод может узнать адрес, образно говоря, дома, в котором он прописан, то есть адрес своего родного объекта.

Слайд 8 (Вопрос)

Вопрос: Подумайте, какие ещё методы можно добавить в класс Planet?
Какие методы будут полезными, а какие излишними?

Слайд 9 (Класс SolarSystem)

Из названия класса очевидно, что он будет использоваться для хранения объектов Planet, при необходимости вы можете расширить функционал данного класса.

Вектор с планетами мы сможем передать через конструктор или метод **addPlanet**.
Для этого класса дружественным будет так же класс **Engine**.

```
#pragma once
#include "Planet.h"
#include <vector>

class SolarSystem
{
public:
	SolarSystem() = default;
	SolarSystem(std::vector<Planet>);

	void addPlanet(Planet);

	~SolarSystem();

private:
	friend class Engine;
	std::vector<Planet> planets;
};
```

```
while (window.isOpen())
{

    sf::Event event;
    while (window.pollEvent(event))
    {
        switch (event.type)
        {
        case sf::Event::Closed:
            window.close();
            break;
        default:
            break;
        }
    }

    window.clear();
    window.display();
}
```

Слайд 10 (Класс SolarSystem)

Реализуем методы данного класса.

```
#include "SolarSystem.h"

SolarSystem::SolarSystem(std::vector<Planet> planets) {
	this->planets = planets;
}

void SolarSystem::addPlanet(Planet planet) {
	planets.push_back(planet);
}

SolarSystem::~SolarSystem() {
	planets.clear();
}
```

Слайд 11 (Класс Engine)

В этом классе будут происходить все необходимые расчеты, то есть за перемещение планет на плоскости будет отвечать именно этот класс.

```
#pragma once
#include "Planet.h"
#include <vector>

class SolarSystem
{
public:
	SolarSystem() = default;
	SolarSystem(std::vector<Planet>);

	void addPlanet(Planet);

	~SolarSystem();

private:
	friend class Engine;
	std::vector<Planet> planets;
};
```

Слайд 12 (Класс Engine)

Реализуем конструктор данного класса.

```
Engine::Engine(unsigned width, unsigned height, std::string window_name) {
    settings.antialiasingLevel = 8;
    view = new sf::View(sf::FloatRect(0, 0, width * 2, height * 2));
    window = new sf::RenderWindow(
        sf::VideoMode(width, height),
        window_name,
        sf::Style::Fullscreen,
        settings);

    view->setCenter(width / 2, height / 2);
    view->zoom(-100);
    window->setView(*view);
}
```

Слайд 13 (Класс Engine)

Реализуем метод установки объекта солнечной системы.

```
void Engine::setSolarSystem(SolarSystem solarSystem) {
    mSolarSystem = solarSystem;
}
```

И не забываем про освобождение памяти с помощью оператора **delete**.

```
Engine::~Engine() {
    delete window;
    delete view;
}
```

Слайд 14 (Реализация метода run())

Прямоугольные координаты x и y точки A будут выражаться через её полярные координаты:

$$x = \rho \cdot \sin(\varphi)$$
$$y = \rho \cdot \cos(\varphi)$$

Слайд 15 (Реализация метода run())

На этом слайде присутствует анимация с плавным прокручиванием кода, запустите её:

```
void Engine::run() {
    while (window->isOpen())
    {
        sf::Event event;
        while (window->pollEvent(event))
        {
            switch (event.type)
            {
            case sf::Event::MouseWheelMoved:
                if (event.mouseWheel.delta == 1)
                {
                    view->zoom(1.0f / gameZoom);
                    window->setView(*view);
                }
                if (event.mouseWheel.delta == -1)
                {
                    view->zoom(gameZoom);
                    window->setView(*view);
                }
                break;
            case sf::Event::Closed:
                window->close();
                break;
            default:
                break;
            }
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
            window->close();
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {

            view->move(100, 0);
            window->setView(*view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {

            view->move(-100, 0);
            window->setView(*view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {

            view->move(0, 100);
            window->setView(*view);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {

            view->move(0, -100);
```

```
            window->setView(*view);
        }

        window->clear(sf::Color::Black);

        for (int i = 0; i < mSolarSystem.planets.size(); i++)
        {
            mSolarSystem.planets[i].currentAngle +=
mSolarSystem.planets[i].angularVelocity * dt.asSeconds();
            mSolarSystem.planets[i].setPosition(
                mSolarSystem.planets[i].centerOrbitPosition.x +
mSolarSystem.planets[i].orbitRadius * cos(mSolarSystem.planets[i].currentAngle),
                mSolarSystem.planets[i].centerOrbitPosition.y +
mSolarSystem.planets[i].orbitRadius * sin(mSolarSystem.planets[i].currentAngle)
            );
            window->draw(mSolarSystem.planets[i]);

        }
        window->display();
        dt = deltaClock.restart();
    }
}
```

Слайд 16 (Реализация функции main())

Нам необходимо обозначить размеры планет, рекомендуемые значения представлены ниже.

```
#define SUN     1000
#define MERCURY 100
#define VENUS   150
#define EARTH   300
#define MARS    270
#define JUPITER 700
#define SATURN  700
#define URANUS  900
#define NEPTUNE 900
#define PLUTO   100
```

Слайд 17 (Реализация функции main())

Создадим и инициализируем константные переменные размеров окна симулятора.

```
const unsigned screen_size_w = 1920;
const unsigned screen_size_h = 1080;

const unsigned screen_center_w = screen_size_w / 2;
const unsigned screen_center_h = screen_size_h / 2;
```

Также не забываем про классы, которые мы создали ранее.

```
Engine* engine = new Engine(screen_size_w, screen_size_h, "SolarSystem");
SolarSystem solarSystem;
```

В конце функции **main** лучше сразу написать оператор **delete**, который освободит ранее выделенную память.

```
delete engine;
```

Слайд 18 (Реализация функции main())

Проинициализировать объекты Planet лучше всего следующими значениями:

```
Planet sun(sf::Vector2f(screen_center_w, screen_center_h), 0,
    SUN, 0, "assets/planet_textures/sun.png");
Planet mercury (sf::Vector2f(screen_center_w, screen_center_h), 1700,
    MERCURY, 2.5, "assets/planet_textures/mercury.png");
Planet venus(sf::Vector2f(screen_center_w, screen_center_h), 10600,
    VENUS, 1.25, "assets/planet_textures/venus.png");
Planet eath(sf::Vector2f(screen_center_w, screen_center_h), 15500,
    EARTH, 0.7, "assets/planet_textures/earth.png");
Planet mars(sf::Vector2f(screen_center_w, screen_center_h), 22500,
    MARS, 1, "assets/planet_textures/mars.png");
Planet jupiter(sf::Vector2f(screen_center_w, screen_center_h), 25500,
    JUPITER, 0.5, "assets/planet_textures/jupiter.png");
Planet saturn(sf::Vector2f(screen_center_w, screen_center_h), 30500,
    SATURN, 0.7, "assets/planet_textures/saturn.png");
Planet uranus(sf::Vector2f(screen_center_w, screen_center_h), 40500,
    URANUS, 0.3, "assets/planet_textures/uranus.png");
Planet neptune(sf::Vector2f(screen_center_w, screen_center_h), 47000,
    NEPTUNE, 0.2, "assets/planet_textures/neptune.png");
Planet pluto(sf::Vector2f(screen_center_w, screen_center_h), 90000,
    PLUTO, 0.1, "assets/planet_textures/pluto.png");
```

Слайд 19 (Реализация функции main())

В конце мы можем добавить наши планеты в класс **SolarSystem**:

```
solarSystem.addPlanet(sun);
solarSystem.addPlanet(mercury);
solarSystem.addPlanet(venus);
solarSystem.addPlanet(eath);
solarSystem.addPlanet(mars);
solarSystem.addPlanet(jupiter);
solarSystem.addPlanet(saturn);
solarSystem.addPlanet(uranus);
solarSystem.addPlanet(neptune);
solarSystem.addPlanet(pluto);
```

Слайд 20 (Реализация функции main())

Далее необходимо установить солнечную систему в наш объект **Engine**:

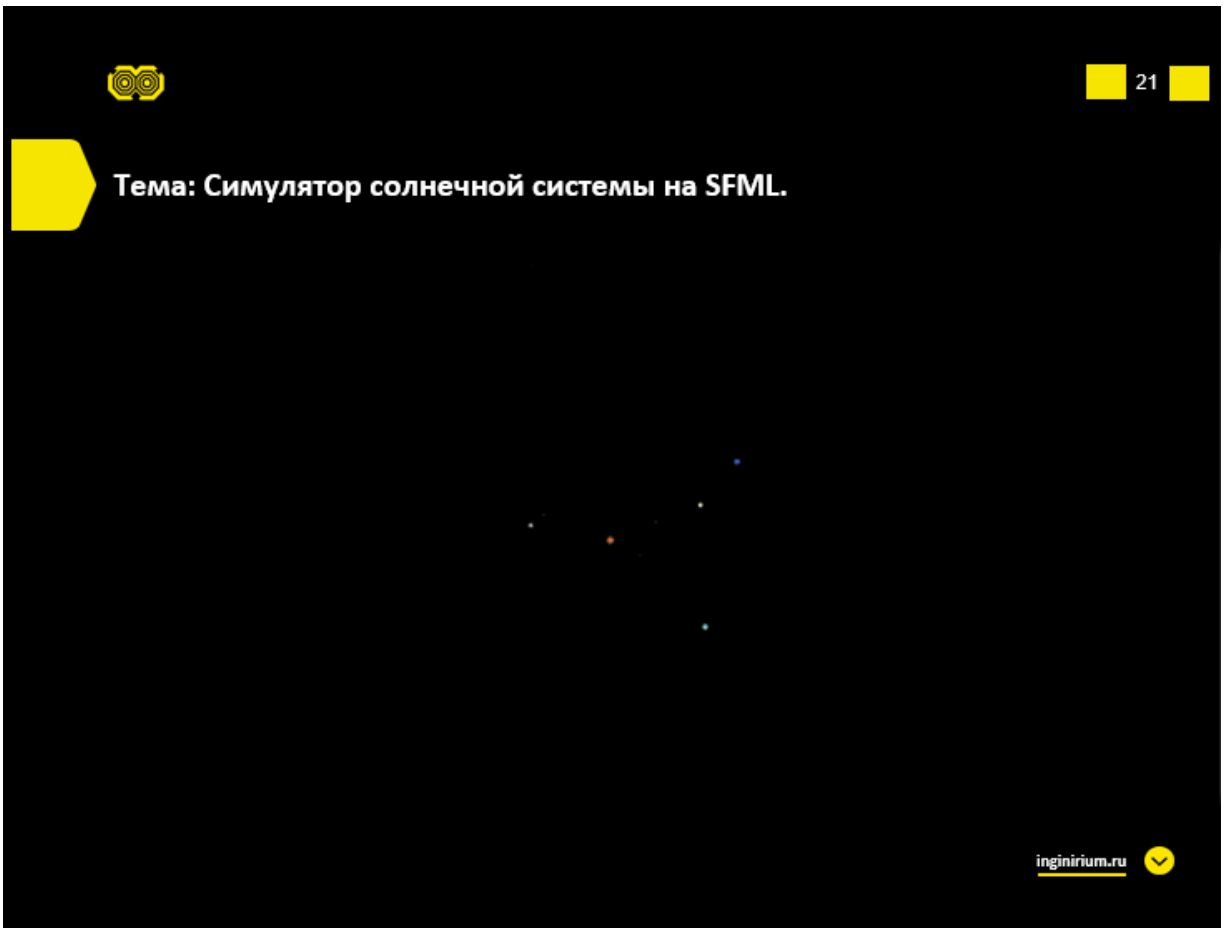
```
engine->setSolarSystem(solarSystem);
```

Запускаем нашу симуляцию:

```
engine->run();
```

Слайд 21 (Реализация функции main())

Продемонстрируйте видео со слайда, там показан результат работы программы:



Практическая часть, решение задач.

Слайд 22 (Задание)

1) Измените характеристики некоторых планет.
2) Измените порядок планет на обратный.

Рефлексия

В конце каждого занятия необходимо подвести итоги. Обсудите с Учащимися следующие вопросы:

1. Почему мы прописали эту строку «`delete engine;`» в конце функции main?
2. Для чего нужен класс Engine?