**ВВЕДЕНИЕ**

В современном мире для большинства молодых пользователей ПК огромную роль играют компьютерные игры. Это самый молодой, и самый стремительно развивающийся рынок за последние 30 лет. Разработка игр на сегодняшний день является не только интересным, но и выгодным делом. Объем игрового рынка уже превосходит киноиндустрию и догоняет спортивный рынок, а доходы некоторых игровых компаний исчисляются в миллиардах долларов.

По мнению аналитиков «SuperData», игры заработали в 2015ом году 64,7 млрд. долларов, при этом мобильные игры заработали 24,7 млрд. долларов. Также ряд аналитиков сообщают, что рынок компьютерных игр вырастет до 45 млрд. долларов к 2018 году.

Кроме того, разработка игр - одно из самых нетривиальных задач, появившихся с массовым коммерческим производством персональных компьютеров. Эта область компьютерной инженерии в которой проявляется необходимость в самых разных специалистах, от маркетологов и data science - инженеров, до 2D и 3D художников, и звукорежиссеров.

Из этого мы можем сделать вывод - что производство игр - очень творческая деятельность, и данная индустрия крайне чувствительна к инструментам разработки благодаря которым индустрия может развиваться, и экспериментировать с продуктами, прототипировать новые интересные механики, и разрабатывать огромные проекты, над которыми могут трудиться до сотен человек. Основным инструментом разработки современных игр является «Игровой движок». Это программная система, состоящая из множества подсистем:

* Графическая подсистема.
* Подсистема симуляции физики.
* Подсистема аудио вывода.
* Подсистема контроля ввода.
* Подсистема взаимодействия с сетью.

Кроме того, игровой движок абстрагирует пользователя от низкоуровневых понятий, позволяя оперировать понятиями, наиболее близкими как человеку, так и компьютеру.

**Предметом исследования** данного дипломного проекта является процесс разработки игрового 2D движка Glance Engine

**Объектом исследования** является игровой движок Glance Engine.

Перед данным исследованием поставлена следующая **основная цель**:

* Написание интерпретатора скриптов, и метаинформации об игре в код на C++.
* Написание непосредственно игрового движка на языке C++

Достижение поставленной цели осуществлялось через постановку и **решение следующих задач**:

* Написание технического задания к интерпретатору скриптов
* Написание документации к интерпретатору скриптов
* Написание интерпретатора скриптов на C#
* Предоставление API для редактора на C#
* Написание игрового движка на C++

Основные программные инструменты, используемые в данной работе:

* Microsoft Visual Studio 2015 Community
* Sublime Text 3
* GitHub

**1 МАКЕТИРОВАНИЕ СИСТЕМЫ**

Макетирование - это еще одна форма проектно-исследовательского моделирования.

Макеты различают по степени проработки, используемому материалу, масштабу. С помощью макетов решают различные задачи - от выявления вариантов изделия до специальных его испытаний поэтому макеты могут быть разные, от поисковых до демонстрационных.

Классификация макетов в зависимости от их функции в проектировании несколько условна, поскольку любой макет может играть разные роли: при определенных условиях поисковый макет может выступать в качестве демонстрационного, а в процессе изготовления демонстрационного макета можно вести поиск с целью, устранения конструктивных, композиционных или иных ошибок. От функции макетов в проектировании зависят и используемые материалы, и особенности изготовления. Как правило, чем доступнее макетные материалы, чем проще технология его изготовления, тем быстрее и более широкий круг проектных задач решается с его помощью.

Основная цель макета – снять неопределенности в требованиях заказчика.

Макетирование – это процесс создания модели требуемого программного продукта.

Модель может принимать одну из трех форм:

- бумажный макет или на основе ПК (человеко-машинный диалог);

- работающий макет;

- существующая программа:

**1.1 Постановка задачи**

К основным функциям будущего программного обеспечения можно отнести следующие:

- Предоставление интерфейса для игрового редактора

- Выявление ошибок при работе пользователя

- Генерация кода конечного продукта

- Интеграция пользовательских скриптов в поведение объектов конечного продукта

- Предоставление интерфейса для взаимодействия с игровым движком из пользовательских скриптов

- Написание документации к игровому движку

**1.2 Выбор метода решения**

Для проектирования программных продуктов разработаны объектно-ориентированные технологии, которые включают в себя специализированные языки программирования и инструментальные средства разработки

Традиционные подходы к разработке программных продуктов всегда подчеркивали различия между данными и процессами их обработки, так как технологии, ориентированные на информационное моделирование, сначала специализирует данные, а затем описывает процессы, использующие эти данные. Технологии структурного подхода ориентированы, в первую очередь, на процессы обработки данных и организации информационных потоков между связанными процессами.

Объектно-ориентированная технология разработки программных продуктов объединяет данные и процессы в логические сущности – объекты, которые имеют способность наследовать характеристики одного или более объектов, обеспечивая тем самым повторное использование программного кода. Это приводит к значительному уменьшению затрат на создание программного продукта, превышает эффективность жизненного цикла программного изделия.

Для решения данной задачи было рассмотрено несколько технологий, и языков программирования

Pascal ABC.Net – это язык программирования Pascal нового поколения, сочетающий простоту классического языка Паскаль, ряд современных расширений и огромные возможности платформы .NET.

С++ – компилируемый, статически типизированный язык программирования общего назначения. Поддерживает все основные парадигмы программирования: Объектно-ориентированное программирование, Функциональное программирование, Процедурное программирование. Широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр).

C# - объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. Относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

Для решения поставленных задач было выбрано две технологии:

C# для реализации генератора исходного кода конечного продукта, ключевым фактором для выбора именно данной технологии было три фактора:

* На данном языке программирования можно относительно быстро описывать программу.
* Этот язык имеет приятный синтаксис, обширную стандартную библиотеку, большое community, и его использование доставляет удовольствие программисту.
* Этот язык программирования входит в семейство .Net языков, что позволяет прозрачно интегрировать продукт с другими продуктами, написанными на языках семейства .Net, а это важно, так как графическая оболочка движка написана на этом - же языке, что позволяет писать меньше кода для того чтобы «подружить» две программы, и больше усилий сосредоточить на собственно программном продукте.

C++ для реализации собственно игрового движка, и на то есть два фактора:

* Высокая производительность данного языка, что крайне важно для подобных приложений.
* Последние стандарты C++ позволяют писать не только производительный, но и очень понятный и читаемый код, это условие также важно, так как одной из концепций Glance Engine является «фокус на конечном продукте» что означает что мы не должны обязывать клиента писать лишний код для того чтобы «компилятор понял, что мы имеем ввиду» вместо написания полезного кода.

**1.3 Описание групп пользователей**

Игровой движок состоит из двух больших частей - публичной и приватной.

Публичная часть доступна всем пользователям данного движка, и является API для взаимодействия с внутренними системами.

Публичная часть включает в себя:

- Публичное API;

- Документацию к публичному API;

Приватная часть включает в себя:

- Приватное API для получения метаданных об игре для подключения редактора;

- Документация к приватному API.

Пользователями публичного API являются разработчики конечного продукта (игры).

Пользователями приватного API являются разработчики редакторов.

**2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ**

**2.1 Конструирование сценария диалога**

Для начала нужно знать, что же представляет из себя проектирование структуры игрового движка.

Проектирование – неотъемлемая часть комплекса работ по созданию электронного игрового движка. В этой части осуществляется работа по формированию информационной структуры, работы по дизайну и программированию. Начальный этап проектирования характерен разработкой концепции проекта создания движка, следующие же стадии работ – это проектирование интерфейса и функциональной, а также информационной структуры.

Игровой движок Glance Engine предоставляет интерфейс для программирования на языке C++, все его функции и классы располагаются в пространстве имен (namespace) «gc», внутренняя реализация движка спрятана от пользователя в пространстве имен «::gc::priv».

Glance Engine предоставляет интерфейс для:

* Работы с окнами
* Работы с картинками(спрайтами)
* Работы с устройствами ввода (мышь, клавиатура)
* Работы со звуками
* Работы с симуляцией физики
* Работы с файловой системой
* Работа со стандартными контейнерами из STL, и добавлением им функциональности

Также Glance Engine предоставляет интерфейс для работы с «Событиями». Проще всего представить себе событие как подписку на новости интересного вам источника в любимой социальной сети:

Как только вам начинает нравится какой-либо источник контента – вы «подписываетесь» на него, и каждый раз, когда этот источник производит контент – вам приходит уведомление и вы становитесь оповещены о том, что данный источник произвел контент, как только данный контент вам перестает быть интересен, вы от него «отписываетесь» и перестаете получать уведомления.

В мире программирования игр под событием можно подразумевать всё что угодно. От нажатия на кнопку мышки, или получения данных по сети – до получения ранения одного врага, или взрыва космической станции.

**2.2 Логическое проектирование**

Логическая структура игрового движка отображает то, как именно связанны между собой компоненты приложения. Логическая структура движка тщательно документирована, и представлена в соответствии с рисунком 2.1

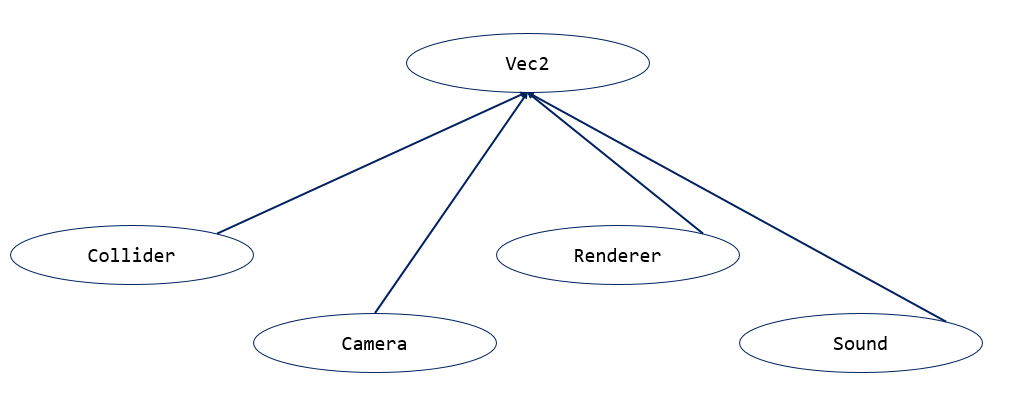


Рисунок 2.1 – Логическая структура физических компонентов Glance Engine

Логическая структура графической части Glance Engine похожа на Дерево графов (Рисунок 2.2), где каждый лист – объект, который можно нарисовать, однако реально на экране рисуются только нижние листья, так называемые «Объекты». Каждый объект имеет ровно один графический компонент (Спрайт или Анимация) который в итоге и будет отрисован в соответствии с его внутренним состоянием.

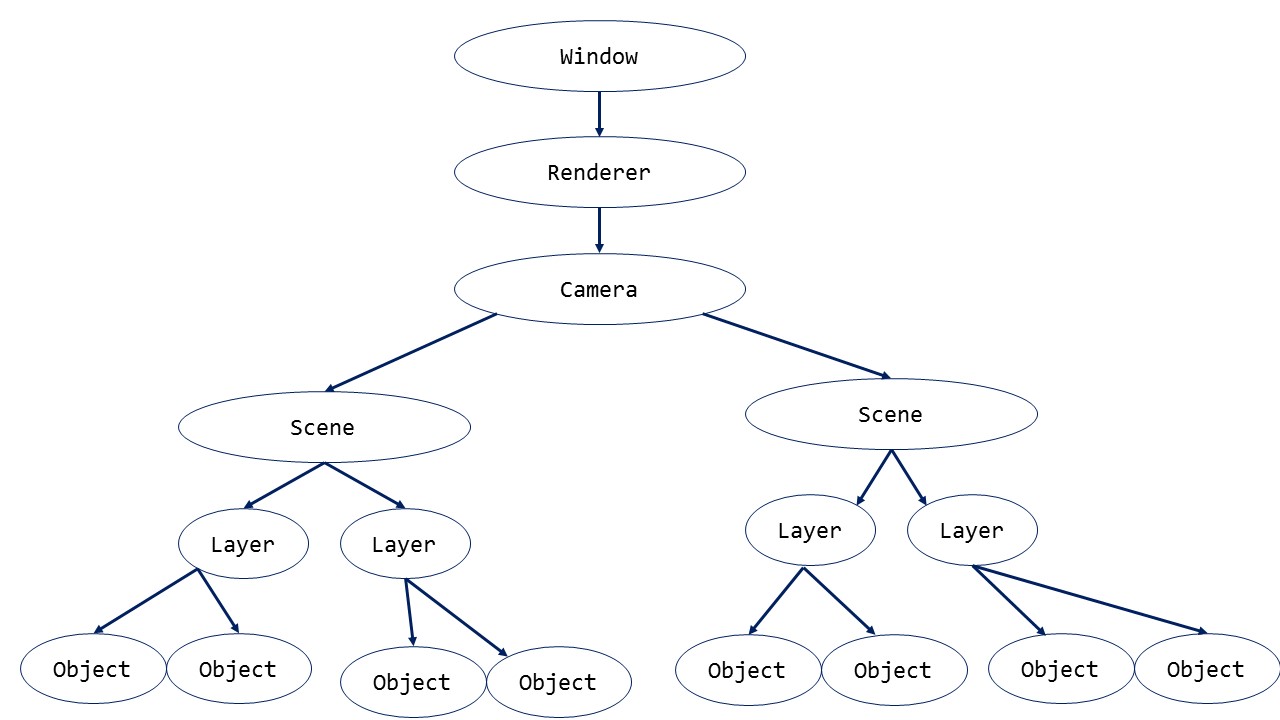


Рисунок 2.2 – Логическая структура графического модуля Glance Engine

**3 РАЗРАБОТКА ПРОЕКТА**

**3.1 Описание программного продукта**

**3.1.1 Glance Engine**

Данный игровой движок является обёрткой над уже существующей библиотекой SFML для языка C++. То есть пользователь «снаружи» не видит то, из чего на самом деле состоит движок, ему предоставляется лишь интерфейс Glance Engine. Glance Engine предоставляет интерфейс для работы с:

Анимацией,

Камерой,

Коллайдерами,

Выводом отладочной информации,

Делегатами,

Событиями,

Стандартными контейнерами в функциональном стиле,

Клавиатурой,

Мышью,

Линиями,

Built-in типами в ООП стиле,

Ошибками исполнения,

Выводом информации в стандартный поток,

«Свойствами»,

Случайными числами,

Процессом рендера сцены,

Звуками,

Спрайтами,

Векторами.

**3.1.1.1 Animation**

Анимация в Glance Engine представляет собой набор сущностей, называемых SpriteFrame. Это Спрайт, с интрузивным счетчиком кадров, которые должен длиться этот спрайт.

Реализована она в виде std::list с шаблонным параметром gc::SpriteFrame, а также итератором по этому листу, и прочими дополнительными полями.

Содержание файла GC/Animation.h

#pragma once

#include "Utils.h"

#include "SpriteFrame.h"

#include "Numerics.h"

#include "Event.h"

#include "Optional.h"

namespace gc{

enum class AnimationType

{

Single,

Cyclic,

PingPong

};

enum class AnimationDirection

{

Forward,

Backward

};

template<AnimationType T>

class Animation;

namespace priv {

template<AnimationType T>

struct \_Animation\_Update\_Helper {

void update(Animation<T> &, float dt);

};

template<>

struct \_Animation\_Update\_Helper<AnimationType::Single>{

void update(Animation<AnimationType::Single> & a, float dt);

};

template<>

struct \_Animation\_Update\_Helper<AnimationType::Cyclic>{

void update(Animation<AnimationType::Cyclic> & a, float dt);

};

template<>

struct \_Animation\_Update\_Helper<AnimationType::PingPong>{

gc::AnimationDirection \_currentAnimationDirection;

\_Animation\_Update\_Helper():\_currentAnimationDirection(gc::AnimationDirection::Forward){}

void update(Animation<AnimationType::PingPong> & a, float dt);

};

}

template<AnimationType T>

class Animation

{

public:

//only constructor

Animation();

//get arguments, and emplace SpriteFrame in self

template<class ... Args>

Optional<Animation &> emplaceFrame(Args && ... args) noexcept;

//return true if animation is player right now

const Bool isPlay() const noexcept;

//return const reference to current SpriteFrame

const SpriteFrame & getCurrentSpriteFrame() const noexcept;

//return const reference to current Sprite

const Sprite & getCurrentSprite() const noexcept;

//return current speed of animation

const float getCurrentSpeed() const noexcept;

//return current index of animation

const U32 getCurrentFrameIndex() const noexcept;

//setters

//sets the current animation speed passing '1' will make animation run normally, 2 make it run twice faster

Animation & setCurrentSpeed(float speed) noexcept;

//starts the animation

Animation & start() noexcept;

//pause the animation

Animation & pause() noexcept;

//stop the animation

Animation & stop() noexcept;

//update the animation, dt is elapsed milliseconds

Animation & update(float dt) noexcept;

//events

//emited when whole animation is ended

const Event<Animation &> & onEnd;

//emited when new frame is became current

const Event<Animation &> & onNextFrame;

//emited when start() is called

const Event<Animation &> & onStart;

//emited when stop() is called

const Event<Animation &> & onStop;

//emited when pause() is called

const Event<Animation &> & onPause;

private:

bool \_isPlays;//is animation played now

std::list<SpriteFrame> \_SFlist;//Sprite frame list

std::list<SpriteFrame>::iterator \_currentAnimationFrame;//current SpriteFrame

float \_currentAnimationTime;//time, measured from last frame started

float \_currentAnimationSpeed;//speed of animation

u32 \_currentAnimationFrameIndex;//current number of animation frame

priv::\_Animation\_Update\_Helper<T> \_updateHelper;

Animation(const Animation &) = delete;

Animation(Animation &&) = delete;

void operator = (const Animation &) = delete;

void operator = (Animation &&) = delete;

Event<Animation &> \_onEnd;

Event<Animation &> \_onNextFrame;

Event<Animation &> \_onStart;

Event<Animation &> \_onStop;

Event<Animation &> \_onPause;

friend struct priv::\_Animation\_Update\_Helper<T>;

};

}

//--------------------------------implementation---------------------------------

namespace gc{

namespace priv{

inline void \_Animation\_Update\_Helper<AnimationType::Single>::update(Animation<AnimationType::Single> & a, float dt){

if(!a.\_isPlays) return;

a.\_currentAnimationTime += dt;

auto dur = a.\_currentAnimationFrame->getDuration() / a.\_currentAnimationSpeed;

if(a.\_currentAnimationTime > dur){

a.\_currentAnimationTime -= dur;

++a.\_currentAnimationFrame;

++a.\_currentAnimationFrameIndex;

a.\_onNextFrame.emit(a);

}

if(a.\_currentAnimationFrame == a.\_SFlist.end()){

a.\_isPlays = false;

a.\_currentAnimationFrame = a.\_SFlist.begin();

a.\_currentAnimationFrameIndex = 0;

a.\_onEnd.emit(a);

}

return;

}

inline void \_Animation\_Update\_Helper<AnimationType::Cyclic>::update(Animation<AnimationType::Cyclic> & a, float dt){

if (!a.\_isPlays) return;

a.\_currentAnimationTime += dt;

auto dur = a.\_currentAnimationFrame->getDuration() / a.\_currentAnimationSpeed;

if (a.\_currentAnimationTime >= dur){

a.\_currentAnimationTime -= dur;

++a.\_currentAnimationFrame;

++a.\_currentAnimationFrameIndex;

a.\_onNextFrame.emit(a);

}

if (a.\_currentAnimationFrame == a.\_SFlist.end()){

a.\_currentAnimationFrame = a.\_SFlist.begin();

a.\_currentAnimationFrameIndex = 0;

a.\_onEnd.emit(a);

}

return;

}

inline void \_Animation\_Update\_Helper<AnimationType::PingPong>::update(Animation<AnimationType::PingPong> & a, float dt){

if(!a.\_isPlays) return;

a.\_currentAnimationTime += dt;

if(a.\_currentAnimationTime > a.\_currentAnimationFrame->getDuration() / a.\_currentAnimationSpeed){

a.\_currentAnimationTime -= a.\_currentAnimationFrame->getDuration() / a.\_currentAnimationSpeed;

//change current Frame

\_currentAnimationDirection == AnimationDirection::Forward ? ++a.\_currentAnimationFrame : --a.\_currentAnimationFrame;

//change current Index

\_currentAnimationDirection == AnimationDirection::Forward ? ++a.\_currentAnimationFrameIndex : --a.\_currentAnimationFrameIndex;

a.\_onNextFrame.emit(a);

}

if(a.\_currentAnimationFrame == a.\_SFlist.begin()){

\_currentAnimationDirection = AnimationDirection::Forward;

a.\_onEnd.emit(a);

}

if(a.\_currentAnimationFrame == a.\_SFlist.end()){

\_currentAnimationDirection = AnimationDirection::Backward;

--a.\_currentAnimationFrame;

--a.\_currentAnimationFrame;

}

return;

}

}

template<AnimationType T>

Animation<T>::Animation() :

\_SFlist(), \_currentAnimationFrame(\_SFlist.begin()), \_isPlays(false),

\_currentAnimationTime(0.0f), \_currentAnimationSpeed(1.0f), \_currentAnimationFrameIndex(0),

//events init

\_onEnd(), onEnd(\_onEnd),

\_onNextFrame(), onNextFrame(\_onNextFrame),

\_onStart(), onStart(\_onStart),

\_onStop(), onStop(\_onStop),

\_onPause(), onPause(\_onPause)

{}

template<AnimationType T>

template<class ... Args>

Optional<Animation<T> &> Animation<T>::emplaceFrame(Args && ... args) noexcept{

IF\_FAIL(

\_SFlist.emplace\_back(std::forward<Args>(args)...);

return \*this;

){return fail\_exception;}

}

template<AnimationType T>

const Bool Animation<T>::isPlay() const noexcept{

return \_isPlays;

}

template<AnimationType T>

const SpriteFrame & Animation<T>::getCurrentSpriteFrame() const noexcept{

return \*\_currentAnimationFrame;

}

template<AnimationType T>

const Sprite & Animation<T>::getCurrentSprite() const noexcept{

return \_currentAnimationFrame->getSprite();

}

template<AnimationType T>

const float Animation<T>::getCurrentSpeed() const noexcept{

return \_currentAnimationSpeed;

}

template<AnimationType T>

const U32 Animation<T>::getCurrentFrameIndex() const noexcept{

return \_currentAnimationFrameIndex;

}

template<AnimationType T>

Animation<T> & Animation<T>::setCurrentSpeed(float speed) noexcept{

\_currentAnimationSpeed = speed;

return \*this;

}

template<AnimationType T>

Animation<T> & Animation<T>::start() noexcept{

\_isPlays = true;

\_currentAnimationFrame = \_SFlist.begin();

return \*this;

}

template<AnimationType T>

Animation<T> & Animation<T>::pause() noexcept{

\_isPlays = false;

return \*this;

}

template<AnimationType T>

Animation<T> & Animation<T>::stop() noexcept{

\_isPlays = false;

\_currentAnimationFrame = \_SFlist.begin();

\_currentAnimationFrameIndex = 0;

\_currentAnimationTime = 0.0f;

\_onStop.emit(\*this);

return \*this;

}

template<AnimationType T>

Animation<T> & Animation<T>::update(float dt) noexcept{

\_updateHelper.update(\*this, dt);

return \*this;

}

}

**3.1.1.2 Camera**

Камера в Glance Engine – сущность, при помощи которой можно управлять видимой для игрока областью сцены. Реализована в виде адаптера к соответствующему классу в нижележащей библиотеке SFML

Содержание файла GC/Camera.h

#pragma once

/\*

gc/Camera provides Class for Camera object

Nearest analog is camera from our real world,

actually it is viewport, which limited visible part of scene

\*/

#include "Vec2.h"

#include <SFML/Graphics.hpp>

namespace gc

{

class Camera

{

::sf::View \_view;

friend class Renderer;

public:

inline Camera() noexcept;

inline Camera(const Vec2 & pos, const Vec2 & size) noexcept;

inline Camera & moveTo(float x, float y) noexcept; //set

inline Camera & moveTo(const Vec2 & pos) noexcept; //set

inline Camera & moveOn(float dx, float dy) noexcept;//change

inline Camera & moveOn(const Vec2 & dp) noexcept;

inline Camera & setSize(float w, float h) noexcept;//set

inline Camera & setSize(const Vec2 & size) noexcept;

inline Camera & changeSize(float dw, float dh) noexcept;//change

inline Camera & changeSize(const Vec2 & ds) noexcept;

inline Camera & setRotation(float r) noexcept;//set

inline Camera & rotate(float dr) noexcept;//change

inline Camera & zoom(float z) noexcept;//change (deprecated)

//getters

inline Vec2 const & getPosition() const noexcept;

inline Vec2 const & getSize() const noexcept;

inline const float getRotation() const noexcept;

};

/\*-------------------------IMPLEMENTATION----------------\*/

#pragma region GC\_CAMERA

inline Camera::Camera() noexcept :

\_view()

{}

inline Camera::Camera(const Vec2 & pos, const Vec2 & size) noexcept :

\_view(pos, size)

{}

inline Camera & Camera::moveTo(float x, float y) noexcept {

\_view.setCenter(x, y);

return \*this;

}

inline Camera & Camera::moveTo(const Vec2 & pos) noexcept {

\_view.setCenter(pos);

return \*this;

}

inline Camera & Camera::moveOn(float dx, float dy) noexcept {

\_view.move(dx, dy);

return \*this;

}

inline Camera & Camera::moveOn(const Vec2 & dp) noexcept {

\_view.move(dp);

return \*this;

}

inline Camera & Camera::setSize(float w, float h) noexcept {

\_view.setSize(w, h);

return \*this;

}

inline Camera & Camera::setSize(const Vec2 & size) noexcept {

\_view.setSize(size);

return \*this;

}

inline Camera & Camera::changeSize(float dw, float dh) noexcept {

auto size = \_view.getSize();

\_view.setSize(size.x + dw, size.y + dh);

return \*this;

}

inline Camera & Camera::changeSize(const Vec2 & ds) noexcept {

auto size = \_view.getSize();

\_view.setSize(size.x + ds.x, size.y + ds.y);

return \*this;

}

inline Camera & Camera::setRotation(float r) noexcept {

\_view.setRotation(r);

return \*this;

}

inline Camera & Camera::rotate(float dr) noexcept {

\_view.rotate(dr);

return \*this;

}

inline Camera & Camera::zoom(float z) noexcept {

\_view.zoom(z);

}

inline Vec2 const & Camera::getPosition() const noexcept {

return \_view.getCenter();

}

inline Vec2 const & Camera::getSize() const noexcept {

return \_view.getSize();

}

inline const float Camera::getRotation() const noexcept {

return \_view.getRotation();

}

#pragma endregion

}

**3.1.1.3 Collider**

Коллайдер в Glance Engine – сущность, основа для симуляции физики в Glance Engine. Это такие «твердые объекты», которые умеют обнаруживать столкновения друг с другом, и реагировать на это, оповещая через события, и не пересекаясь.

Содержание файла GC/Collider.h

#pragma once

/\*

gc/Collider provides Types for Collider primitives as Rectangle and Circle

it also has next Config defines:

GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

GC\_COLLIDER\_CIRCLE\_NEGATIVE\_RADIUS\_ENABLE

GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

if it is not defined, rectangle colliders size will always be positive,

and on trying to make it negative, it will throw gc::InvalidArgumentException

as example:

#undef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

#include "gc/Collider"

int main(){

gc::Collider<gc::ColliderType::Rectangle>(gc::Vec2(5,5), gc::Vec2(-10, 50));//exception will be thrown here

}

or

#define GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

#include "gc/Collider"

int main(){

gc::Collider<gc::ColliderType::Rectangle>(gc::Vec2(5,5), gc::Vec2(-10, 50));//ok

}

also, if this macro is defined, performance can be decreased

GC\_COLLIDER\_CIRCLE\_NEGATIVE\_RADIUS\_ENABLE

if it is not defined, circle colliders radius will be always positive,

and on trying to makeit negative, it will throw gc::InvalidArgumentException

as example

#undef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

#include "gc/Collider"

int main(){

gc::Collider<gc::ColliderType::Circle>(gc::Vec2(5,5), -4);//exception will be thrown here

}

or

#define GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

#include "gc/Collider"

int main(){

gc::Collider<gc::ColliderType::Circle>(gc::Vec2(5,5), -4);//ok

}

Both macro are undefined by default

\*/

#include "Vec2.h"

#include "Utils.h"

namespace gc

{

enum class ColliderType

{

Rectangle,

Circle

};

template<ColliderType T>

class Collider

{

//static\_assert(false, "new gc::ColliderType was added, please specialize gc::Collider for it");

};

template<>

class Collider<ColliderType::Circle>;

template<>

class Collider<ColliderType::Rectangle>

{

const Vec2 & \_pos;//left up corner;

Vec2 \_size;//where x is width, and y is height

public:

inline Collider(Collider const & c);

inline Collider(Collider && c);

inline Collider(Vec2 const & pos, Vec2 const & size);

inline Collider(Vec2 const & pos, float width = 0, float height = 0);

inline Vec2 const & getPosition() const;

inline Vec2 const & getSize() const { return \_size; }

inline Collider & changeSize(Vec2 const & v);

inline Collider & changeSize(const float & dw, const float & dh);

inline Collider & setSize(Vec2 const & s);

inline Collider & setSize(const float & width, const float & height);

inline const float getTopLevel() const;

inline const float getLowLevel() const;

inline const float getLeftLevel() const;

inline const float getRightLevel() const;

inline const Vec2 getTopLeftCorner() const;

inline const Vec2 getLowLeftCorner() const;

inline const Vec2 getTopRightCorner() const;

inline const Vec2 getLowRightCorner() const;

inline const bool isCollide(const Collider & c) const;

inline const bool isCollide(const Collider<ColliderType::Circle> & a) const;

};

template<>

class Collider<ColliderType::Circle>

{

const Vec2 & \_pos;

float \_radius;

public:

inline Collider(Collider const & c);

inline Collider(Collider && c);

inline Collider(Vec2 const & pos, const float & radius = 0.0f);

inline Vec2 const & getPosition() const;

inline const float & getRadius() const;

inline Collider & changeRadius(const float & dr);

inline Collider & setRadius(const float & r);

inline const float getTopLevel();

inline const float getLowLevel();

inline const float getLeftLevel();

inline const float getRightLevel();

inline const bool isCollide(Collider const & a) const;

inline const bool isCollide(Collider<ColliderType::Rectangle> const & a) const;

};

/\*--------------------------------------------------IMPLEMENTATION--------------------------------------------------------\*/

Collider<ColliderType::Rectangle>::Collider(const Collider<ColliderType::Rectangle> & c) :

\_pos(c.\_pos), \_size(c.\_size)

{}

Collider<ColliderType::Rectangle>::Collider(Collider<ColliderType::Rectangle> && c):

Collider(c)

{}

Collider<ColliderType::Rectangle>::Collider(Vec2 const & pos, Vec2 const & size) :

\_pos(pos), \_size(size)

{

#ifndef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

if (size.x < 0 || size.y < 0)

throw 228;

#endif

}

Collider<ColliderType::Rectangle>::Collider(Vec2 const & pos, float width, float height) :

\_pos(pos), \_size(width, height)

{

#ifndef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

if (width < 0 || height < 0)

throw 228;

#endif

}

Vec2 const & Collider<ColliderType::Rectangle>::getPosition() const {

return \_pos;

}

Collider<ColliderType::Rectangle> & Collider<ColliderType::Rectangle>::changeSize(Vec2 const & v) {

\_size += v; return \*this;

}

Collider<ColliderType::Rectangle> & Collider<ColliderType::Rectangle>::changeSize(const float & dw, const float & dh) {

\_size.x += dw; \_size.y += dh; return \*this;

}

Collider<ColliderType::Rectangle> & Collider<ColliderType::Rectangle>::setSize(Vec2 const & s) {

\_size = s; return \*this;

}

Collider<ColliderType::Rectangle> & Collider<ColliderType::Rectangle>::setSize(const float & width, const float & height) {

\_size.x = width; \_size.y = height; return \*this;

}

const float Collider<ColliderType::Rectangle>::getTopLevel() const {

#ifdef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

return (\_size.y > 0 ? \_pos.y : \_pos.y + \_size.y);

#else

return \_pos.y;

#endif

}

const float Collider<ColliderType::Rectangle>::getLowLevel() const {

#ifdef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

return (\_size.y > 0 ? \_pos.y + \_size.y : \_pos.y);//just inverted

#else

return \_pos.y + \_size.y;

#endif

}

const float Collider<ColliderType::Rectangle>::getLeftLevel() const {

#ifdef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

return (\_size.x > 0 ? \_pos.x : \_pos.x + \_size.x);

#else

return \_pos.x;

#endif

}

const float Collider<ColliderType::Rectangle>::getRightLevel() const {

#ifdef GC\_COLLIDER\_RECT\_NEGATIVE\_SIZE\_ENABLE

return (\_size.x > 0 ? \_pos.x + \_size.x : \_pos.x);

#else

return \_pos.x + \_size.x;

#endif

}

const Vec2 Collider<ColliderType::Rectangle>::getTopLeftCorner() const{

return Vec2(this->getLeftLevel(), this->getTopLevel());

}

const Vec2 Collider<ColliderType::Rectangle>::getLowLeftCorner() const {

return Vec2(this->getLeftLevel(), this->getLowLevel());

}

const Vec2 Collider<ColliderType::Rectangle>::getTopRightCorner() const {

return Vec2(this->getRightLevel(), this->getTopLevel());

}

const Vec2 Collider<ColliderType::Rectangle>::getLowRightCorner() const {

return Vec2(this->getRightLevel(), this->getLowLevel());

}

const bool Collider<ColliderType::Rectangle>::isCollide(const Collider<ColliderType::Rectangle> & c) const

{

auto selfTop = this->getTopLevel();

auto selfBot = this->getLowLevel();

auto selfLeft = this->getLeftLevel();

auto selfRight = this->getRightLevel();

auto targetTop = c.getTopLevel();

auto targetBot = c.getLowLevel();

auto targetLeft = c.getLeftLevel();

auto targetRight = c.getRightLevel();

if ((selfLeft >= targetLeft && selfLeft <= targetRight) || (selfRight <= targetRight && selfRight >= targetLeft))

if ((selfTop >= targetTop && selfTop <= targetBot) || (selfBot <= targetBot && selfBot >= targetBot))

return true;

return false;

}

const bool Collider<ColliderType::Rectangle>::isCollide(Collider<ColliderType::Circle> const & a) const {

return a.isCollide(\*this);

}

//-----------------------------------------------------------------------------------

Collider<ColliderType::Circle>::Collider(Collider<ColliderType::Circle> const & c) :

\_pos(c.\_pos), \_radius(c.\_radius)

{}

Collider<ColliderType::Circle>::Collider(Collider<ColliderType::Circle> && c):

Collider(c)

{}

Collider<ColliderType::Circle>::Collider(Vec2 const & pos, const float & radius) :

\_pos(pos), \_radius(radius)

{

#ifndef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

if (radius < 0)

throw 228;

#endif

}

Vec2 const & Collider<ColliderType::Circle>::getPosition() const {

return \_pos;

}

const float & Collider<ColliderType::Circle>::getRadius() const {

return \_radius;

}

Collider<ColliderType::Circle> & Collider<ColliderType::Circle>::changeRadius(const float & dr) {

#ifndef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

if (dr < 0 && \_radius < dr)

throw 228;

#endif

\_radius += dr; return \*this;

}

Collider<ColliderType::Circle> & Collider<ColliderType::Circle>::setRadius(const float & r) {

#ifndef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

if (r < 0)

throw 228;

#endif

\_radius = r;

return \*this;

}

const float Collider<ColliderType::Circle>::getRightLevel() {

#ifdef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

return (\_radius < 0 ? \_pos.x - \_radius : \_pos.x + \_radius);

#else

return \_pos.x + \_radius;

#endif

}

const float Collider<ColliderType::Circle>::getLeftLevel() {

#ifdef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

return (\_radius < 0 ? \_pos.x + \_radius : \_pos.x - \_radius);

#else

return \_pos.x - \_radius;

#endif

}

const float Collider<ColliderType::Circle>::getTopLevel() {

#ifdef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

return (\_radius < 0 ? \_pos.y + \_radius : \_pos.y - \_radius);

#else

return \_pos.y - \_radius;

#endif

}

const float Collider<ColliderType::Circle>::getLowLevel() {

#ifdef GC\_COLLIDER\_CIRCLE\_NEGATIVE\_SIZE\_ENABLE

return (\_radius < 0 ? \_pos.y - \_radius : \_pos.y + \_radius);

#else

return \_pos.y + \_radius;

#endif

}

const bool Collider<ColliderType::Circle>::isCollide(Collider<ColliderType::Circle> const & a) const {

float dx = a.\_pos.x - \_pos.x;

float dy = a.\_pos.y - \_pos.y;

float distance = abs(sqrt(dx \* dx + dy \* dy));

if (distance <= (\_radius + a.\_radius))

return true;

return false;

}

const bool Collider<ColliderType::Circle>::isCollide(Collider<ColliderType::Rectangle> const & a) const {

if (\_pos.x < a.getLeftLevel()) {//we left than argument

if (\_pos.y < a.getTopLevel()) //nearest is Top, Left corner

return (\_pos - a.getTopLeftCorner()).getLength() < \_radius;

else if (\_pos.y > a.getLowLevel()) {//nearest is Low, Left corner

return (\_pos - a.getLowLeftCorner()).getLength() < \_radius;}

else//nearest is Left side

return (a.getLeftLevel() - \_pos.x) < \_radius;

}

else if (\_pos.x > a.getRightLevel()) {//we right than argument

if (\_pos.y < a.getTopLevel())//nearest is Top, Right corner

return (\_pos - a.getTopRightCorner()).getLength() < \_radius;

else if (\_pos.y > a.getLowLevel())//nearest is Low, Right corner

return (\_pos - a.getLowRightCorner()).getLength() < \_radius;

else//nearest is Right side

return (\_pos.x - a.getRightLevel()) < \_radius;

}

else {//we inside on x

if (\_pos.y < a.getTopLevel())//nearest is Top side

return (a.getTopLevel() - \_pos.y) < \_radius;

else if (\_pos.y > a.getLowLevel())//nearest is Bottom side

return (\_pos.y - a.getLowLevel()) < \_radius;

else //our center inside rectangle

return true;

}

}

}

**3.1.1.4 Debug**

Debug в Glance Engine существует для вывода отладочной информации на экран в процессе работы игры. Отличие от gc::print в том, что информация выводится на экран в левый верхний угол, а не в консоль. Это удобнее по двум причинам:

Нет необходимости переводить взгляд со сцены на окно консоли.

Нет необходимости собственно запускать окно консоли.

Кроме того пользователь может специализировать вывод отдельного типа для вывода в консоль по правилам описанным в документации

**3.1.1.5 Delegate**

**3.1.1.6 Event**

**3.1.1.7 Iterate**

**3.1.1.8 Keyboard**

**3.1.1.9 Mouse**

**3.1.1.10 Line**

**3.1.1.11 Numerics**

**3.1.1.12 Print**

**3.1.1.13 Property**

**3.1.1.14 Random**

**3.1.1.15 Renderer**

**3.1.1.16 Sound**

**3.1.1.17 Sprite**

**3.1.1.18 Vec2**

**3.2 Контрольный пример**

Данный программный продукт предназначен для облегчения доступа к учебному материалу.

Рассмотрим пример пользования электронного пособия.

Для начала запускаем программу. Появляется окно, изображенное на рисунке 2.8

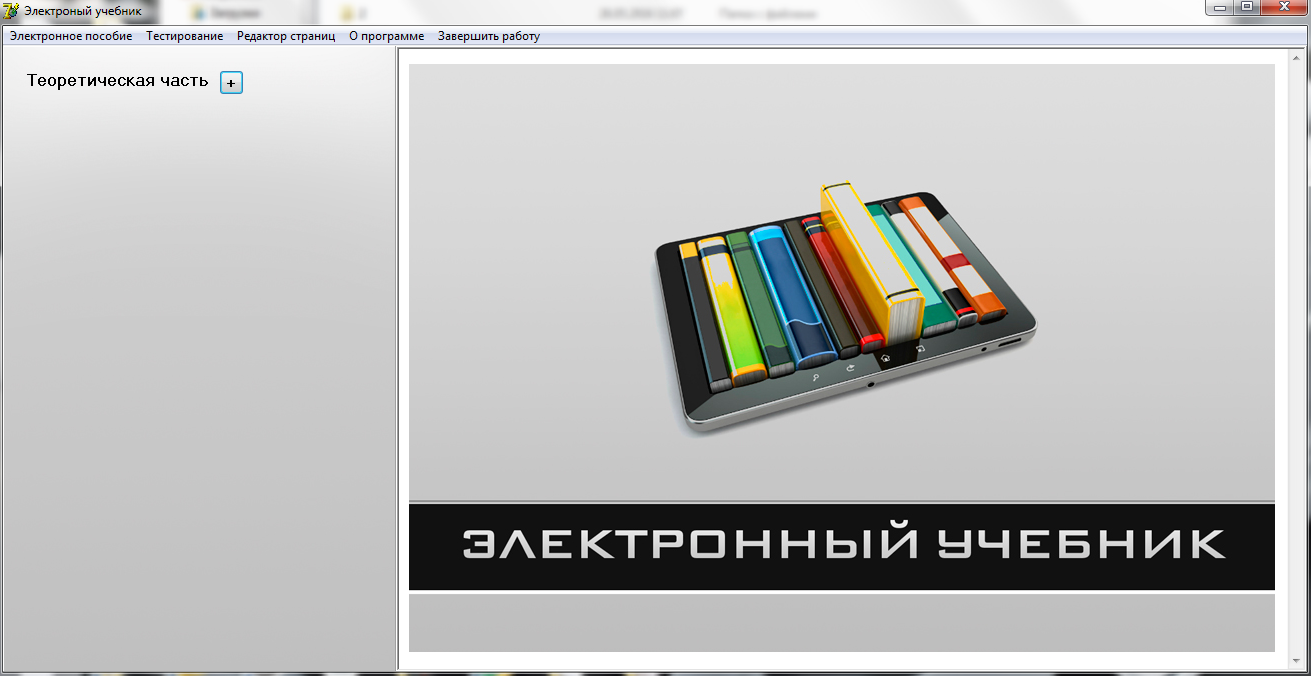


Рисунок 2.8 – Окно электронного пособия

Нажимаем на кнопку "+", перед нами открывается поле с доступным учебным материалом, изображенном на рисунке 2.9

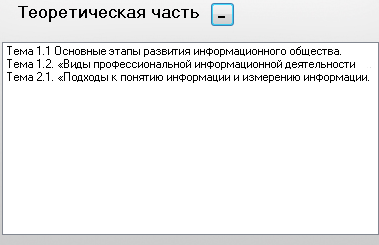


Рисунок 2.9 – Поле с доступным учебным материалом

Далее выбираем любой интересующий нас учебный материал, он откроется в поле справа, как на рисунке 2.10

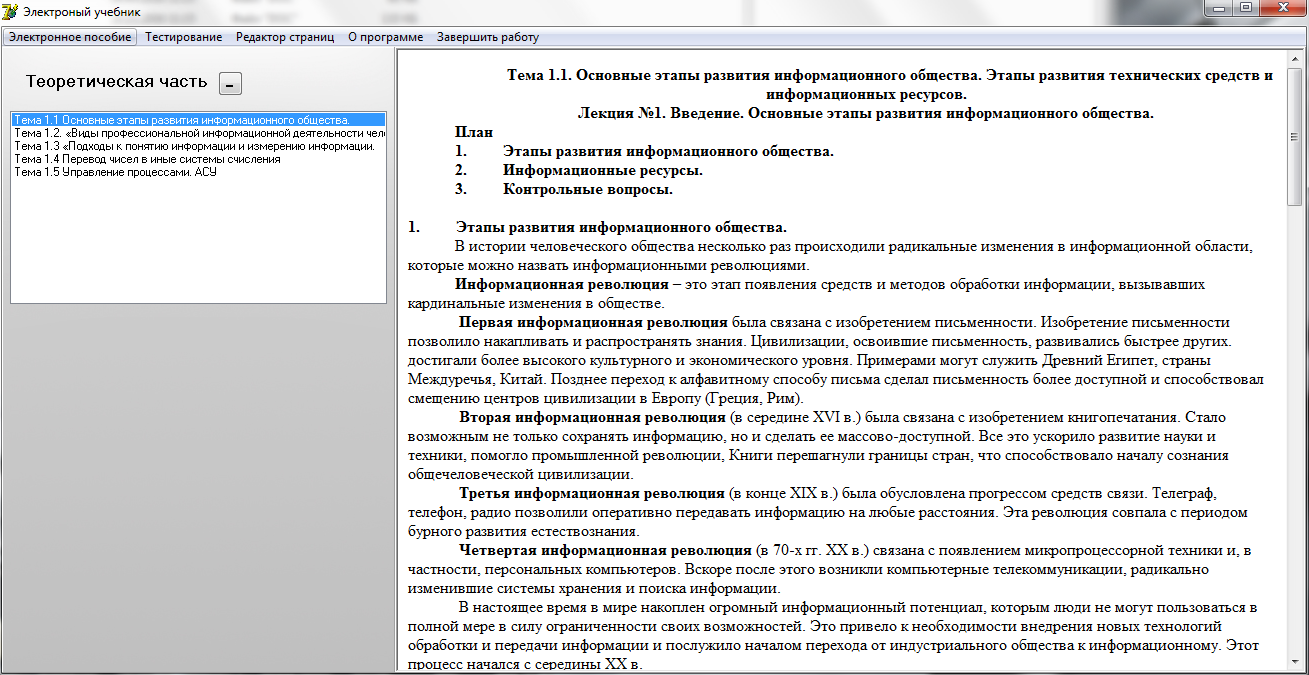


Рисунок 2.10 - Просмотр теоретического материала

**3.3 Тестирование программного продукта**

Тестирование – это процесс многократного выполнения программы с целью выявления и устранения ошибок.

Цель тестирования состоит в выявлении как можно большего числа ошибок.

Тестирования программного продукта сосредоточим вокруг проверки следующего критерия – проверка работы редактора страниц;

Проведем тестирование добавления, редактирования страниц.

Отредактируем существующий учебный материал и попробуем добавить новый.

Переходим в модуль "Редактор страниц", откроется окно как на рисунке 2.11

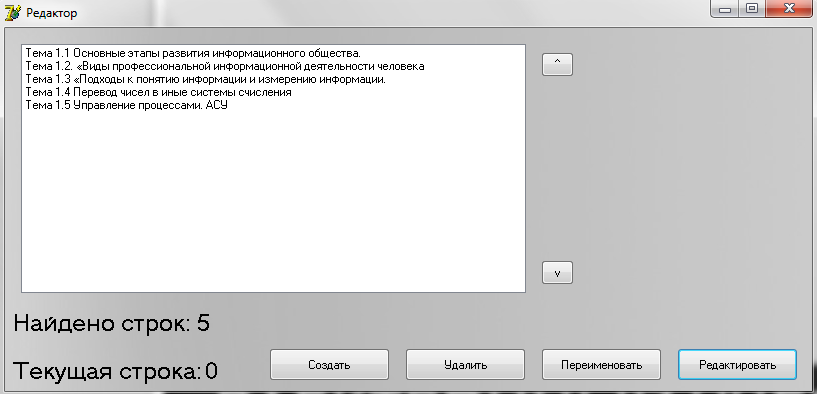


Рисунок 2.11 – Окно редактора страниц

Мы видим все существующие на данный момент страницы учебного материала в виде строк с их названиями. Каждую строку можно переименовать, удалить, передвинуть в списке вверх или вниз, а также отредактировать содержимое каждой страницы.

Попробуем добавить новую страницу, нажимаем "Создать". Появляется окошко, а затем запускается MSWord как на рисунке 2.12

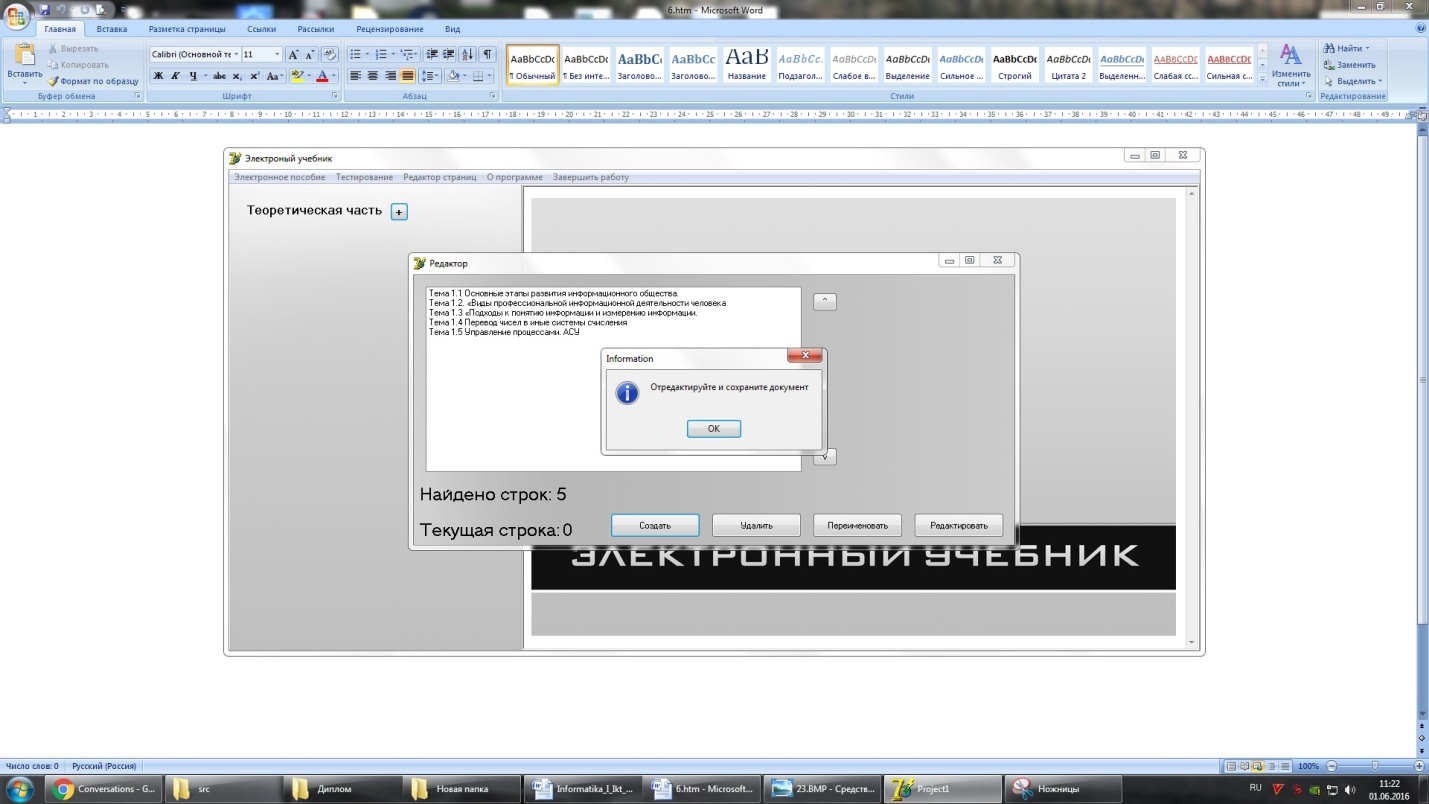


Рисунок 2.12 – Создание новой страницы

После заполнения документа закрываем MSWord, появляется диалог, изображенный на рисунке 2.13

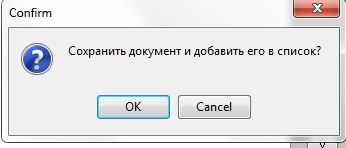


Рисунок 2.13 – Диалог о сохранении документа

Нажимаем "OK", наш новый документ появляется в конце списка, как на рисунке 2.14

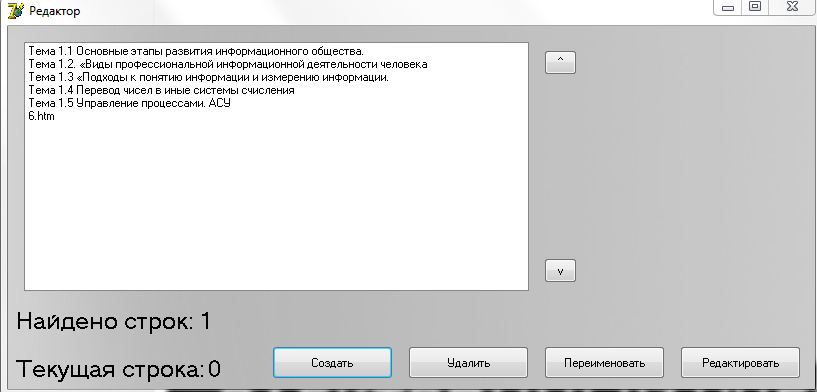


Рисунок 2.14 – Редактор с новой строкой

Теперь нашу строку можно переименовать, а затем проверить её работоспособность.

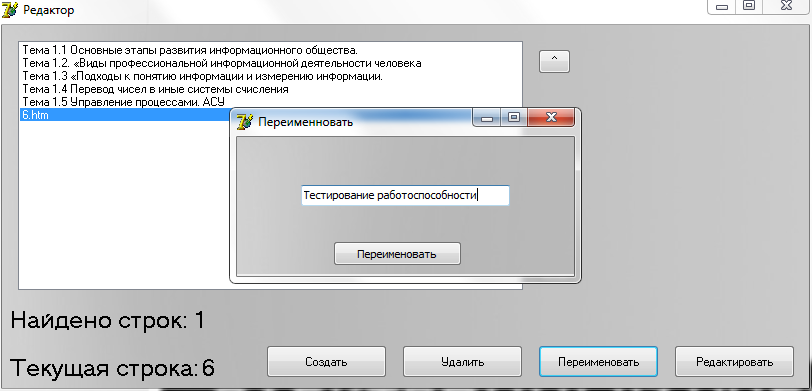


Рисунок 2.15 – Переименование новой строки

Переходим во вкладку "Электронный учебник" и видим результат:

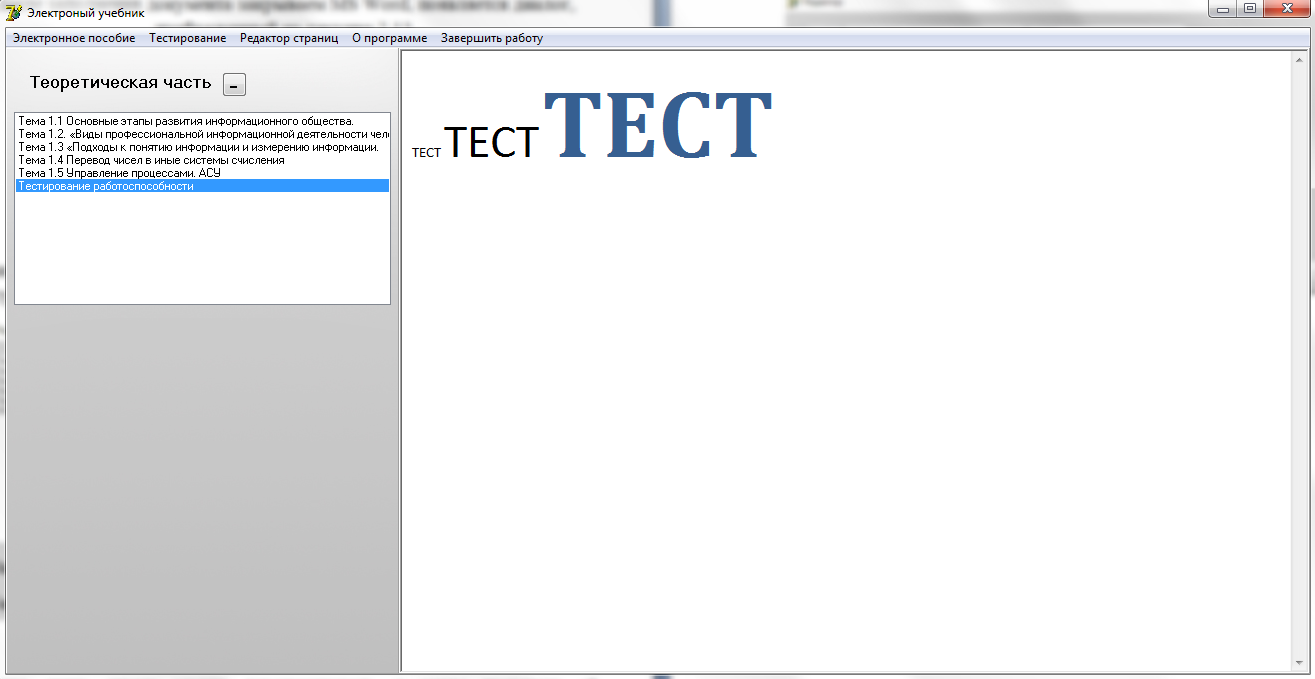


Рисунок 2.16 – Результат редактирования списка

**3.4 Описание технических и программных средств**

Минимальные системные требования:

- OSWindowsXP/Vista/Sevenиновее;

- процессор IntelPentium;

- оперативная память 256 Мб;

- Свободное пространство на жестком диске 50 Мб;

- Монитор, поддерживающий разрешение экрана 800 на 600 точек;

- Мышь, клавиатура;

Рекомендуемые системные требования:

- OSWindowsXP/Vista/Sevenиновее;

- процессор IntelPentium 4;

- оперативная память 512 Мб;

- Свободное пространство на жестком диске 50 Мб;

- Монитор, поддерживающий разрешение экрана 1024 на 768 точек;

- Мышь, клавиатура;

**4 ОХРАНА ТРУДА**

Управление охраной труда представляет собой деятельность государства по обеспечению конституционного права граждан на здоровые и безопасные условия труда.

В общем виде механизм управления охраной труда включает в себя методы управления, функции (виды деятельности) и органы управления.

Методы управления - это способы воздействия на поведение и деятельность управляемых объектов с целью снижения производственного травматизма, общей и профессиональной заболеваемости путем создания безопасной и безвредной производственной среды.

Основными методами управления являются административные, экономические и социально-психологические.

При создании сложных автоматизированных систем управления все чаще практикуют системное проектирование, на ранних стадиях которого поднимаются вопросы эргономического обеспечения, таящего в себе большие резервы повышения эффективности и надежности всей системы. Это связано с всесторонним учетом человеческого фактора в процессе проектирования. Основной задачей эргономического обеспечения является оптимизация взаимодействия между человеком и машиной не только в период эксплуатации человеко-машинных систем, но и при изготовлении и даже утилизации технических компонентов. Это достигается в результате проведения и выполнения комплекса, взаимоувязанных по значению, логике и последовательности эргономических процедур и мероприятий, осуществляемых в ходе разработки системы человек-машина и при ее эксплуатации.

Эксплуатация комплекса предполагается на ПЭВМ. Источником питающего напряжения является сеть переменного тока с напряжением 220В, на которую распространяется ГОСТ 25861-83 [2].

В соответствии с требованиями для предупреждения поражений электрическим током необходимо:

* чётко и в полном объёме выполнять правила производства работ и правила технической эксплуатации;
* исключить возможность доступа оператора к частям оборудования, работающим под опасным напряжением, неизолированным частям, предназначенным для работы при малом напряжении и не подключенным к защитному заземлению;
* применять изоляцию, служащую для защиты от поражения электрическим током, выполненную с применением прочного сплошного или многослойного изоляционного материала, толщина которого обусловлена типом обеспечиваемой защиты;
* подводить электропитание к ПЭВМ от розетки здания при помощи специальной вилки с заземляющим контактом;
* защитить от перегрузок по току, рассчитывая на мощность, потребляемую от сети; а также защитить от короткого замыкания оборудование, встроенное в сеть здания;
* надёжно подключить к заземляющим зажимам металлические части, доступные для оператора, которые в результате повреждения изоляции могут оказаться под опасным напряжением;
* проверить, что защитный заземляющий проводник не имеет выключателей и предохранителей, а также надёжно изолирован.

Противопожарная защита помещения обеспечивается применением автоматической установки пожарной сигнализации (ПС-Л1), наличием средств пожаротушения, применением основных строительных конструкций здания с регламентированными пределами огнестойкости, организацией своевременной эвакуации людей, применением средств коллективной и индивидуальной защиты людей.

В рабочих помещениях основными источниками акустических шумов являются шумы ПЭВМ. ЭВМ являются также источниками шумов электромагнитногопроисхождения(колебанияэлементовэлектромеханических устройств под влиянием переменных магнитных полей). Кроме того, в данных помещениях, возникает структурный шум, то есть шум, излучаемый поверхностями колеблющихся конструкций стен, перекрытий, перегородок здания в звуковом диапазоне частот.

Систематический шум может вызвать утомление слуха и ослабление звукового восприятия, а также значительное утомление всего организма. Однако не все шумы вредны. Так, привычные не резко выраженные шумы, сопровождающие трудовой процесс, могут благоприятно влиять на ход работы; нерезкие шумы, характеризующиеся периодичностью звуков, например, музыка, в силу своей ритмичности не только не отвлекают от работы, но и вызывают положительные эмоции, способствуют повышению эффективности труда.

Для устранения или ослабления неблагоприятных шумовых воздействий целесообразно изолировать рабочие помещения, размещая их в частях здания, наиболее удаленных от городского шума - расположенных в глубине здания, обращенных окнами во двор и т.п. Шум ослабевает также благодаря зеленым насаждениям, поглощающим звуки.

Допустимый уровень шума при умственном труде, требующем сосредоточенности, - 50дБ. Для уменьшения шума и вибрации в помещении оборудование, аппараты и приборы устанавливаются на специальные фундаменты и амортизирующие прокладки. Если стены и потолки помещения являются источниками шумообразования, они должны быть облицованы звукопоглощающим материалом.

В целях борьбы с пылью и загрязнением в рабочем помещении каждый день должна проводиться влажная уборка.

Высокая температура отрицательно сказывается и на ряде психологических функций человека. Уменьшается объем запоминаемой информации, резко снижается способность к ассоциациям, ухудшается протекание ассоциативных и счетных операций, понижается внимание.

Относительная влажность в пределах 40 – 60% мало сказывается на состоянии человека. При влажности 99 – 100% практически выключается регулирующий механизм потоотделения и быстро наступает перегревание.

Для поддержания необходимых температуры и влажности рабочее помещение оснащено системами отопления и кондиционирования, обеспечивающими постоянный и равномерный нагрев, циркуляцию, а также очистку воздуха от пыли и вредных веществ.

Для поддержания в помещениях нормального, отвечающего гигиеническим требованиям состава воздуха, удаления из него вредных газов, паров и пыли используют вентиляцию.

Вентиляция – это регулируемый воздухообмен в помещении. Вентиляцией называют также устройства, которые её создают. По способу перемещения воздуха в помещении различают естественную и искусственную вентиляцию. Возможно их сочетание – смешанная вентиляция. Естественная вентиляция подразделяется на аэрацию и проветривание.

При недостаточной освещенности и напряжении зрения состояние зрительных функций находится на низком функциональном уровне, в процессе выполнения работы развивается утомление зрения, понижается общая работоспособность и производительность труда, возрастает количество ошибок.

Освещенность на рабочем месте должна соответствовать зрительным условиям труда согласно гигиеническим нормам. Так, в соответствии с ГОСТ 12.1.006-84, освещенность при работе с дисплеем должна быть 200 лк, а в сочетании с работой с документами – 400 лк.

Применяется мягкий рассеянный свет из нескольких источников, светлая окраска потолка, стен и оборудования.

Направление света определяется необходимостью объемного восприятия объекта и стремлением не допустить ослепления прямым или отраженным светом. Удобным направление искусственного света считается слева сверху и немного сзади.

Основным вредным воздействием на природу для данного проекта являются различные излучения. В помещении, где предполагается эксплуатация системы, основным источником электромагнитного, ионизирующего и лазерного излучения, электростатического и магнитного поля является ПЭВМ, а точнее, ее монитор – устройство для визуального представления информации, хранимой в памяти ЭВМ. Использующиеся в качестве мониторов жидкокристаллические дисплеи не дают вредных излучений, поэтому рассмотрим только излучения мониторов на основе электронно-лучевых трубок. Такие мониторы являются источником нескольких видов электромагнитного излучения определенных диапазонов электромагнитного спектра. Реальная интенсивность каждого диапазона, частота и другие параметры зависят от технической реализации конкретного монитора, наличия экранирования и других факторов.

Работа с дисплеем при неправильном выборе яркости и освещенности экрана, контрастности знаков, цветов знака и фона, при наличии бликов на экране, дрожании и мелькании изображения приводит к зрительному утомлению, головным болям, к значительной физиологической и психической нагрузке, к ухудшению зрения и т.п.

Если при работе на ПК необходимо одновременно пользоваться документами, то следует иметь в виду, что зрительная работа с печатным текстом и с изображением на экране имеет принципиального отличия: изображение светится, мелькает, дрожит, состоит из дискретных элементов, менее контрастно. Снизить или устранить утомление можно только правильным выбором режима воспроизведения изображения на экране, источника освещения (местного или общего), расположения материалов (в целях уменьшения длины или частоты перевода взгляда).

Человек должен так организовать свое рабочее место, чтобы условия труда были комфортными и соответствовали требованиям СНиП:

* удобство рабочего места (ноги должны твердо опираться на пол; голова должна быть наклонена немного вниз; должна быть специальная подставка для ног);
* достаточное пространство для выполнения необходимых движений и перемещений (руки при работе с клавиатурой должны находиться перед человеком; пальцы должны обладать наибольшей свободой передвижения; клавиши должны быть достаточно чувствительны к легкому нажатию);
* необходимый обзор (центр экрана монитора должен быть расположен чуть ниже уровня глаз; монитор должен отстоять от глаз человека на расстоянии 45-60 сантиметров; должна регулироваться яркость и контрастность изображения);
* рациональное расположение аппаратуры и ее органов управления и контроля (монитор должен быть расположен на расстоянии 60 сантиметров и более от монитора соседа; человек должен использовать держатель бумаги);
* достаточное освещение (внешнее освещение должно быть достаточным и равномерным; должна быть настольная лампа с регулируемым плафоном для дополнительного подсвета рабочей документации);
* нормальные условия в отношении шума и вибрации;
* нормальный температурный режим;
* нормальная влажность воздуха;
* необходимая вентиляция.

Запылённость воздуха не должна превышать 0.75 мг/м. На одного инженера - программиста должен приходиться объём помещения 15mjпри площади 4.5 м (без учёта проходов и оборудования). В течение трудового дня необходимо обеспечить воздухообмен помещения объёмом 25-50 м3, отвод влаги 350-500 г и тепла 50 кДж на каждый килограмм массы тела работающего.

Нормативное значение коэффициента естественного освещения (КЕО) для третьего пояса при боковом освещении равно 1.2 %, освещённость при работе с экраном дисплея – 200 лк, при работе с экраном дисплея и документом – 300 лк.

Время работы за дисплеем не должно превышать 4-х часов в сутки.

Рабочее место оператора ЭВМ состоит из монитора, системного блока, клавиатуры, мыши, принтера. Клавиатура должна быть расположена непосредственно перед оператором. Расстояние от глаз оператора до монитора должно составлять 0.5-0.7 м. На столе, на котором расположена ПЭВМ, должно оставаться место для наглядного, графического материала, для возможности работать с литературой, делать какие-либо пометки.

ЗАКЛЮЧЕНИЕ

Целью дипломного проекта является разработка информационной системы для оптимизации обучения.

Исходя из поставленной цели работы, были решены следующие задачи:;

* разработка структуры электронного пособия;
* выявить особенности программных продуктов;
* разработка логотипа и навигационной панели учебника;
* разработка дизайна программы;
* оптимизация программного продукта;

Таким образом, проект содержит функциональную, информационную и событийную модели требований к системе. Виды и последовательность работ аналогичны соответствующим работам по построению моделей деятельности.

В разработанном электронном пособии удачно сочетается интерфейс с его функциональностью и простотой использования.

Разработана максимально удобная и доступная работа для потенциального пользователя, доступные и понятные диалоговые окна. Разбитые по категориям модули, имеющие подробное описание с иллюстрациями.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

f

1. Справочник "Основы Delphi". http://www.delphibasics.ru/
2. Справочник функций и процедурDelphi. http://www.delphisources.ru/
3. М.Голованов, Создание компонентов в средеDelphi.
4. Content management system [http://www.brutto.ru/informacija/uznat- bolshe/content-management-system](http://www.brutto.ru/informacija/uznat-bolshe/content-management-system)
5. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi7. - СПб.: БХВ-Петербург, 2014. - 784 с.
6. Калверт Ч. Delphi4. Энциклопедия пользователя: Пер. с англ./ Чарлз Калверт. - К.: Издательство «ДиаСофт», 2010. - 800 с.
7. Каратыгин С., Тихонов А., Долголаптев В. Базы данных: простейшие средства обработки информации, электронные таблицы, системы управления базами данных: В 2 т. - М.: АВР, 2015.
8. Матросов А.В., Сергеев А.О., Чаунин М.П. HTML4.0. - СПб.: БХВ- Петербург, 2015. - 672 с.
9. Мещеряков Е.В., Хомоненко А.Д. Публикация баз данных в Интернете. - СПб.: БХВ-Петербург, 2011. - 560 с.