Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Воронежский государственный технический университет»

(ФГБОУ ВО «ВГТУ», ВГТУ)

ФАКУЛЬТЕТ экономики, менеджмента и информационных технологий

КАФЕДРА систем управления и информационных технологий в

строительстве

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Обучающийся \_\_\_\_\_\_\_\_Тюленев Д. В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(Ф.И.О. обучающегося)*

Группа бИСТ-214

Вид практики учебная

Тип практики ознакомительная

Наименование предприятия ВГТУ, кафедра СУИТС

Обучающийся Д. В. Тюленев

*(подпись, И.О.Фамилия)*

Руководитель по

практической подготовке \_\_\_\_\_\_\_\_\_\_\_И.В. Гончаров

*(подпись, И.О.Фамилия)*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Воронеж – 2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc106049446)

[1 Требования к программному обеспечению 4](#_Toc106049447)

[2 Разработка программного средства 7](#_Toc106049448)

[3 Контрольный пример работы программного средства 32](#_Toc106049449)

[ЗАКЛЮЧЕНИЕ 37](#_Toc106049450)

[Список использованных источников 38](#_Toc106049451)

[ПРИЛОЖЕНИЕ А код программного средства 39](#_Toc106049452)

# ВВЕДЕНИЕ

Целью ознакомительной практики является систематизация полученных в процессе обучения теоретических знаний, знакомства с объектами будущей профессиональной деятельности, развития мотивации к выполнению будущей профессиональной деятельности.

В результате прохождения практики обучающийся должен:

знать принципы сбора, отбора и обобщения информации, современные информационные технологии и программные средства, принципы, методы и средства решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий;

уметь соотносить разнородные явления и систематизировать их в рамках избранных видов профессиональной деятельности, выбирать современные информационные технологии и программные средства при решении задач, решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий;

владеть работой с информационными источниками, опытом научного поиска, навыками применения современных информационных технологий и программных средств при решении задач создания научных текстов, навыками подготовки обзоров, аннотаций, составления рефератов, научных докладов, публикаций и библиографии по научно-исследовательской работе.

Для прохождения учебной практики было получено индивидуальное задание: разработать программное средство «Vampire», которое позволяет пользователю расслабиться, или скоротать время.

Программное обеспечение реализовано как приложение с графическим интерфейсом, язык программирования – «C#», инструментальная среда - «Visual Studio 2022».

## 1 Требования к программному обеспечению

Разработанное приложение должно выполнять ряд задач, связанных с предоставлением пользователю возможности: комфортно поиграть в игру, настроить сеанс игры под свои требования.

Первостепенной задачей является создание гибкого инструмента, позволяющий создавать своеобразную сцену, на которую монтируются объекты, отвечающие за определённые аспекты игрового процесса. Данный инструмент обязан предоставлять следующие возможности для разработчика: создание и добавление объектов на сцену и предоставление к ним доступа при необходимости, настройка конфигураций объектов (размер, позиционирование, сторонние уникальные параметры), группировка объектов под одним «родительским узлом», с частичным наследование свойств и параметров (создание дочерних объектов). Также необходимо собранный компонент позволял обрабатывать столкновения определённых объектов, действия пользователя с клавиатуры и компьютерной мыши, создавать спрайты и по кадровую анимацию.

Непосредственно, сама игра должна быть выполнена в жанрах «Rougelike», «Battle Arena», для которых характерными особенностями являются генерируемые случайным образом уровни, [пошаговость](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D0%B0%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D1%82%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0" \o "Пошаговая тактика) и необратимость смерти персонажа － в случае его гибели игрок не может загрузить игру и должен начать её заново. Игровой процесс должен начинаться с появления на карта главного героя посредине поля, окружённого непроходимым лесом. Покинуть пределы сцены игрок не может, поэтому для свободного перемещения доступна небольшая прямоугольная область. В пределах краёв леса появляются «монстры», которые с определённой скоростью движутся к персонажу. При столкновении вражеского персонажа и главного героя у последнего убавляются доступные очки «жизни», исчерпав которые игра закончится и пользователь проиграет. Чтобы этого избежать, игроку необходимо противостоять, использую доступную возможность атаковать при помощи меча (для этого используется компьютерная мышь, чтобы прицелиться). Радиус применения оружия не большой, поэтому для эффективной игры в добавок нужно ещё уклоняться, перемещаясь по доступной области игровой арены. Игрок, также, должен видеть сколько ударов остаётся для того, чтобы одолеть противника, поэтому над каждым «монстром» отображаются оставшееся количество здоровья. После устранения противника на краю сцены создаётся ещё один вражеский объект с увеличенной скоростью передвижения. Чтобы хоть как-то упростить геймплей, на карте с установленной периодичностью во времени, появляется ящики для восстановления здоровья главного героя; их расположение определяется случайным образом. Предполагаемый игровой процесс показан в виде концепта в рисунке 1.

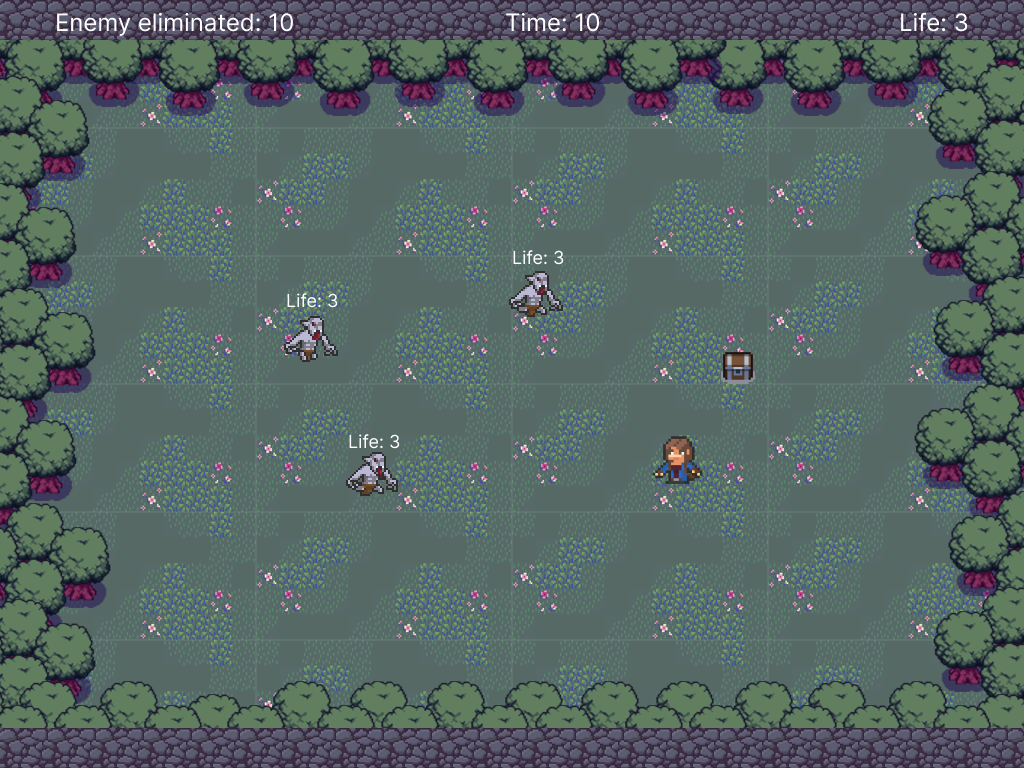


Рисунок 1 － Демонстрация игрового процесса

Перед началом разработки программного решения необходимо определить, с помощью каких средств будет реализована одна из важнейших составляющих программы, а именно диалог с пользователем. Для решения этой задачи было принято решение использовать платформу «Windows Form», для пользовательского интерфейса. Платформа разработки Windows Forms поддерживает широкий набор функций для разработки приложений, включая элементы управления, графику, привязку данных и ввод пользователя. Характерной чертой платформы является использование визуального конструктора с функцией перетаскивания в Visual Studio для упрощения создания приложений Windows Forms.

Для корректной работы игры «Vampire», необходимо реализовать следующие требования:

* Разработка принципа и проектирование модуля обработки столкновений внутриигровых объектов;
* Создание модуля, отвечающий за отображение игровых текстур и по кадровых анимации на основе подготовленных изображений;
* Проектирование алгоритма появления и перемещения вражеских персонажей;
* Разработка модуля для отслеживания пользовательского ввода с клавиатуры и компьютерной мыши;
* Создать возможность пользователю настраивать грядущую игровую сессии, предоставить доступ к конфигурации объектов;
* Реализация модуля, ответственного за регистрацию ударов от врагов и от пользователя;
* Создание принципа генерации ящика поддержки на локации в пределах определённой установленной области.

## 2 Разработка программного средства

При старте собранной программы запускается статический метод «Program.Main», внутри которого вызывается метод «Application.Run», запускающий стандартный цикл обработки сообщений приложения в текущем потоке и делает указанную форму видимой. В качестве начальной формы выступает класс «MainMenu», который отвечает за первоначальную настройку игровой сессии, используя для этого элементы управление: «NumericUpDown» － для установки целочисленных и дробных значений конфигураций, «CheckBox» － для определения режима игры (возможность установить режим отладки, для отображения областей коллизии). Также на данную форму добавлены элементы «Label» для уточнения принадлежности некоторых элементов, «PictureBox» для составления баннера главного меню игры, устанавливается изображение на задний план при помощи свойства «Form.BackgroundImage». Каждый вышеперечисленный объект генерируется автоматически в внутрь класса (добавляются объявления, вызываются конструкторы, устанавливаются свойства). По центру окна расположена элемент «Button», на нажатие которого, с помощью открытого события класса «Form.Button.Click», устанавливается экземпляр делегата «EventHandler», отвечающий за действие по нажатию.

Внутри привязанного метода «play\_button\_Click» создаётся экземпляр класса «MainScene», (наследуется от «Form») для которого при помощи инициализатора устанавливаются свойства, считанные с элементов управления главного меню: «PlayerLife» － значение здоровья главного героя при запуске игры, «PlayerSpeed» － скорость игрока, «EnemyCount» － максимальное количество противников, «MaxEnemySpeed» － максимальная скорость, которую может развить противник, «DebugMode» － включить или отключить отладочный режим, «ChestSpawn» － временной интервал между создание «сундуков поддержки» на сцене. После у созданного объекта формы для отображения вызывается метод «ShowDialog», отображает форму как диалоговое окно.

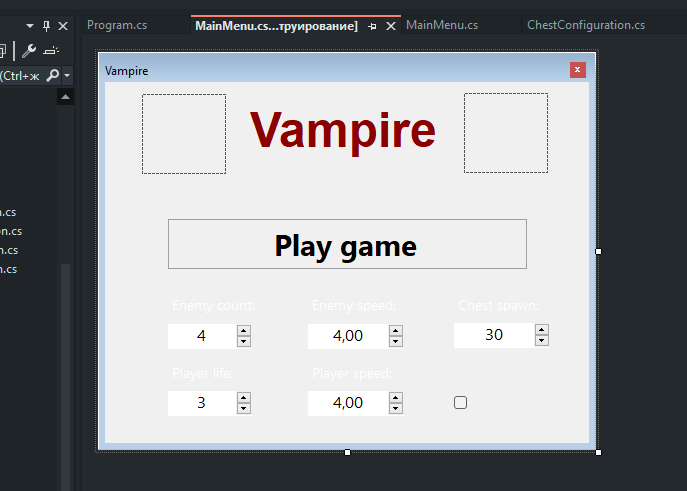


Рисунок 2 － Демонстрация главного меню через конструктор форм

Форма отвечающая за непосредственно за игровую механику, содержит лишь контейнер «Panel» на который монтируется обработка игрового взаимодействия игрока и объектов, выступает в качестве внутриигровой сцены. Внутри конструктора вызывается метод «InitialComponent» для создания элементов управления из сгенерированного конструктором формы участка кода, «InitializeMainPanel» для создания и настройки главной панели, где будет отрисовываться графическая составляющая игры. С помощью статического фабричного метода «FromFile» класса «Image» создаётся изображение заднего фона сцены игры, после чего ссылка на экземпляр присваивается в свойство «Panel.BackgroundImage». Для выполнения действий по инициализации дочерних объектов сцены и их конфигураций используется событие «Form.Load», привязанный метод вызывается до первоначального отображения формы, что помогает настроить экземпляр объекта внутриигровой сцены и запустить циклическую обработку.

Для создания объекта сцены, отвечающей за игровой процесс проекта, используется порождающий паттерн проектирования － «Строитель», который предоставляет пошаговый способ создания составного объекта. Отправной точкой первоначальной загрузки игрового процесса является метод «MainScene.SceneOnLoadAction». Для начала в теле функции создаётся экземпляр класса «EngineSceneBuilder» － вызывается конструктор, внутри которого указывается имя сцены и привязывается ссылка на объект элемента панели, предназначенной для отрисовки графики. Все доступные методы класса «EngineSceneBuilder» представлены в таблице 1.

Таблица 1 － Список объявляемых методов класса «EngineSceneBuilder»

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Название метода | Описание метода |
| Public | RegisterSceneConfiguration | Обобщенный метод, позволяющий добавлять конфигурацию для созданных объектов сцены. В качестве параметров принимает название объекта привязки данных, название свойства (задаётся с помощью аттрибута «EngineObject ImportConfiguration»), для которого ставиться значение и само значение. |
| Public | RegisterSceneChild | Обобщенный метод, позволяет создать и добавить объект на сцену на основе указанного класса. В качестве параметров указывается уникальное имя объекта, список дочерних объектов. Универсальный параметр типа используется для определения типа класса на основе которого будет создан объект. |
| Private | SetSceneObjectConfig | Метод позволяет установить свойства ранее созданного объекта на основе указанных данных при помощи метода «RegisterSceneConfiguration». В качестве параметра принимает ссылку на запись, состоящую из имени объекта и его дочерних объектов. |
| Public | BuildScene | Создаёт и возвращает экземпляр класса «EngineScene», конструктор которого принимает на вход ссылку на панель (к которой крепится отрисовка игровой сцены), список из составленных узлов объектов и список их конфигураций. Для каждого объекта, установленного через метод «RegisterSceneChild», подключаются значения конфигурации, доступные из того же списка дочерние объекты сцены, а также в каждый объект подключается ссылка на созданный экземпляр сцены. |

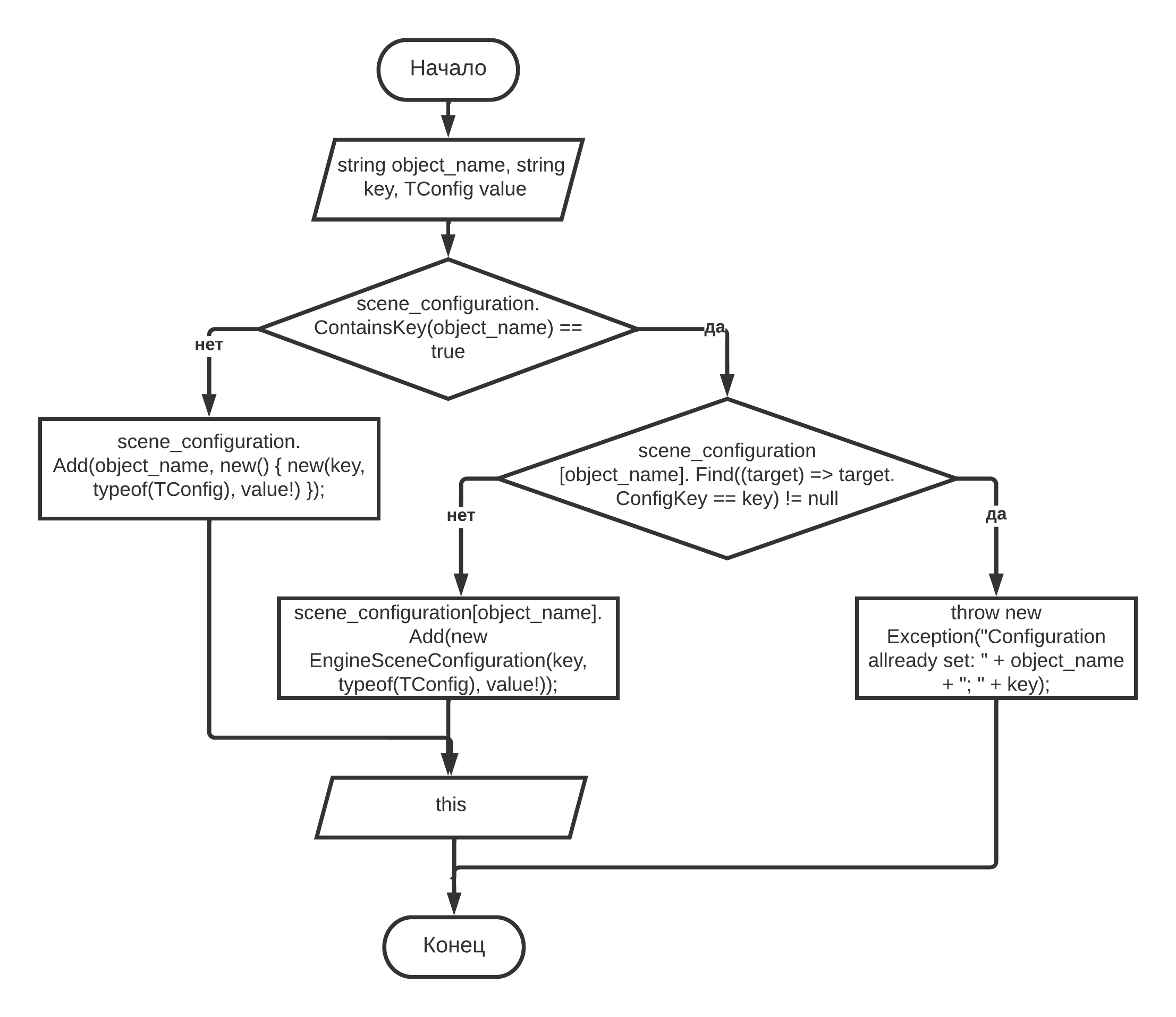


Рисунок 3 － Блок-схема метода «RegisterSceneConfiguration»

В начале тела метода «RegisterSceneConfiguration» проверяется зарегистрирована ли уже какая-либо конфигурация на указанное имя объекта: если ключ для данного имени уже создан, то далее проверяется имеется ли для этого объекта уже зарегистрированная настройка для указанного свойства, и если она уже установлена то генерируется исключение, иначе создаётся запись на этот объект, состоящая из имени свойства, типа загруженного значения и самого значения; в случае, если ни одна конфигурация не была установлена на объект, то создаётся новая запись. В конце возвращается ссылка на текущий экземпляр «строителя» сцены.

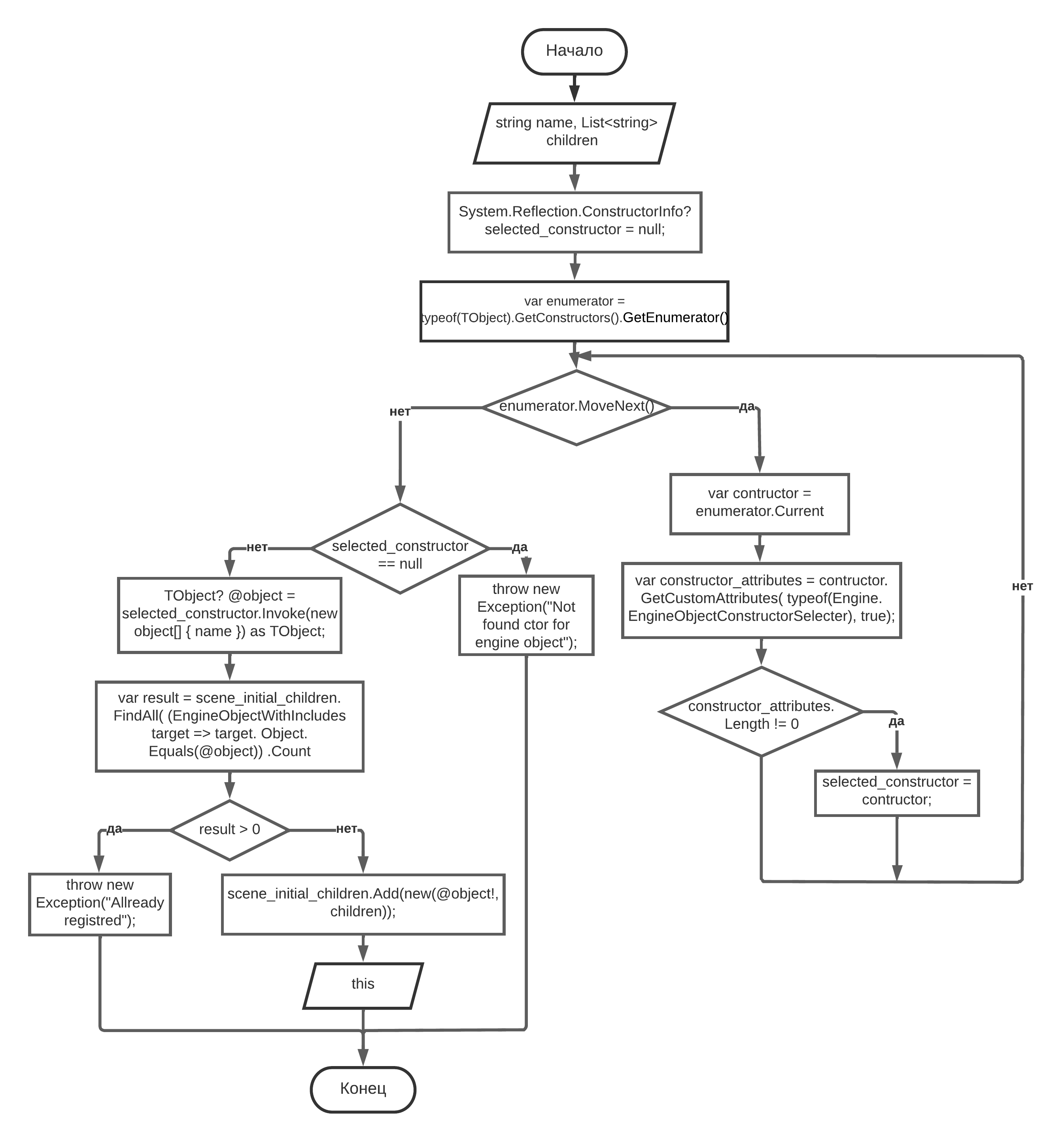


Рисунок 4 － Блок-схема метода «RegisterSceneChild»

Для определения объект какого типа должен создать метод, используется обобщение с его универсальным параметром (ко всему этому накладывается ограничение на использование только дочерних классов от «EngineObject»), тип которого определяет создаваемый объект. При вызове метода, он начинает поиск подходящего конструктора (необходим атрибут «EngineObjectConstructorSelecter») путём рефлексии над передаваемым типом для получения метаданных. Если конструктор не найден － генерируется исключение, иначе происходит вызов конструктора, передаётся зарегистрированное имя и создаётся экземпляр. Далее проверяется при помощи виртуального метода «Equals», имеется ли уже в коллекции объектов сцены созданный экземпляр: если хранится, то генерируется исключение, иначе добавляется в контейнер. В конце возвращается ссылка на текущий экземпляр «строителя» сцены.

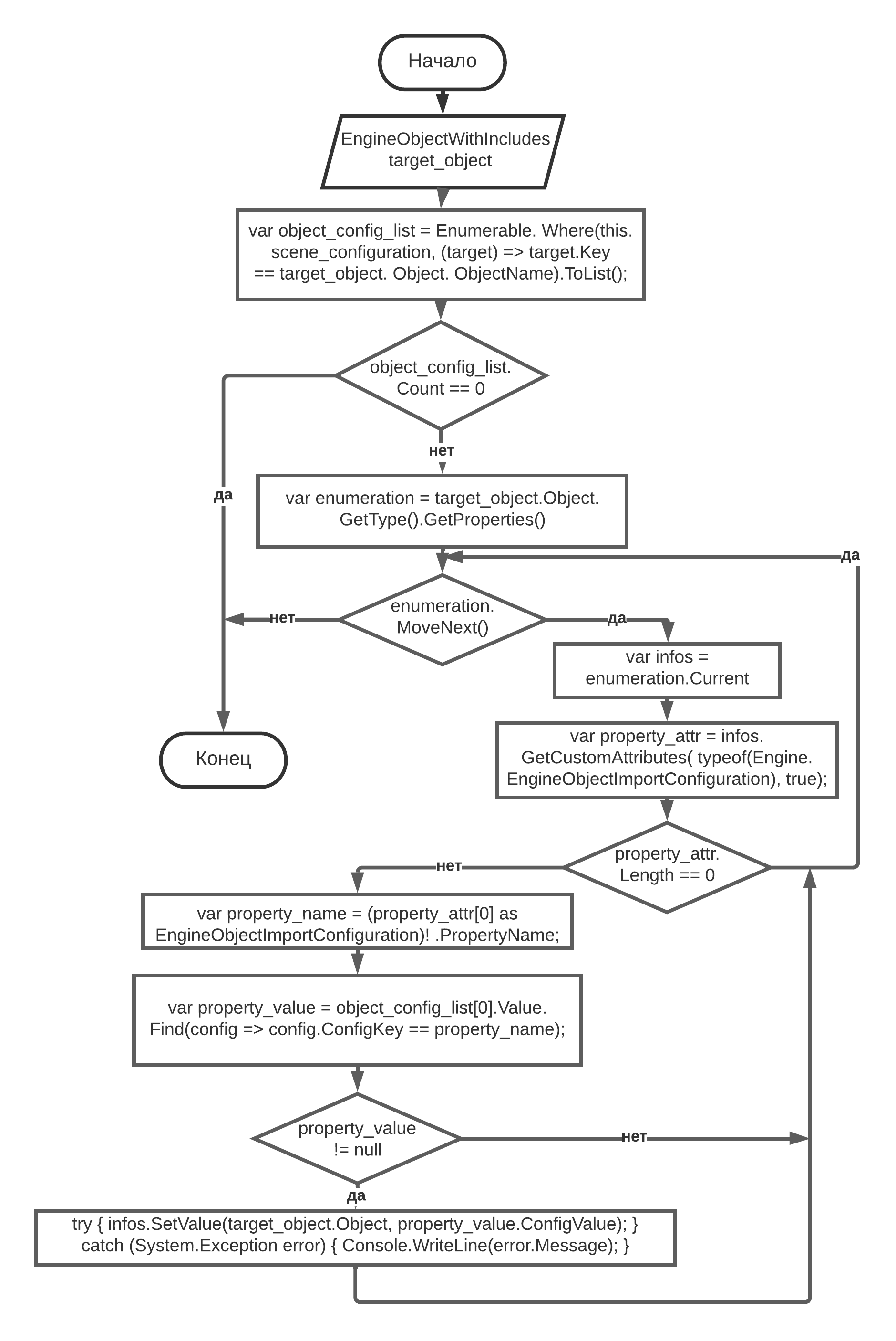


Рисунок 5 － Блок-схема метода «SetSceneObjectConfig»

Метод создаёт объект сцены класса «EngineScene» на основе собранных коллекций объектов, дочерних узлов, конфигураций. Внутри тела функции происходит подключение значений свойств объектов (используется метод «SetSceneObjectConfig»), связывание запрашиваемых дочерних объектов через метод «ConnectLinkToChildren» класса «EngineObject». После чего загружается ссылка на текущую созданную сцены, за которой закреплён объект. Данные операции проводятся для каждого добавленного узла из коллекции объектов. Возвращает созданную сцену.

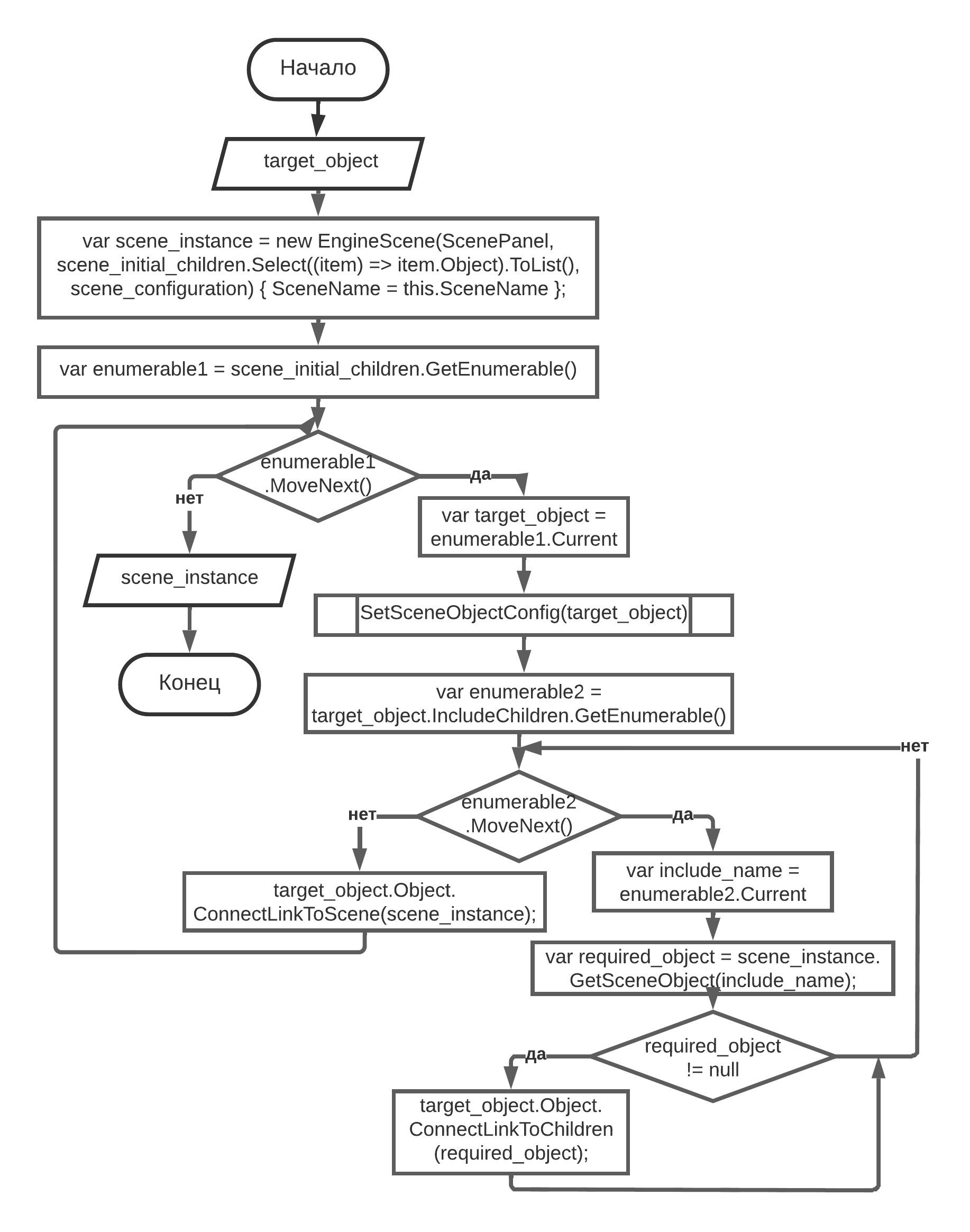


Рисунок 6 － Блок-схема метода «BuildScene»

Класс «EngineScene» отвечает за запуск обработки игровой сцены, выступает в качестве провайдера для зарегистрированных ресурсов: записей конфигураций и ссылки на активные объекты сцены. В классе объявлены некоторые автоматические свойства: «SceneName» представляет имя созданной сцены, «IsExit» － определяет состояние сцены: была она закрыта или нет, «SceneSize» － предоставляет доступ к размерам сцены (а точнее подключённого для вывода изображения элемента «Panel»). Класс реализует интерфейс «IEngineScene», определяющий некоторую функциональность для получения доступа к разным аспектам сцена.

Таблица 2 － Список объявляемых методов класса «EngineScene»

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Название метода | Описание метода |
| Private | InitializeSceneHandling | Метод производит подключение методов объектов сцены к различным события для обработки пользовательского ввода с устройств, предоставления функции отрисовки и обновления состояний в определённом временном интервале. |
| Public | GetSceneObject | Позволяет получить доступ к запрашиваемому объекту со сцены по имени, в качестве фактического параметра принимает строку с именем и возвращает объект (если он был найден, в противном случае вернёт «null»). |
| Public | GetSceneObjects | Предоставляет доступ к списку необходимых объектов, определяемых передаваемым типом универсального параметра. Если объекты не были найдены, то возвращается пустой список. |
| Public | GetSceneConfiguration | Возвращает запись о найденной конфигурации: строка ключ, тип установленного значения, значение конфигурации. Если запрашиваемый ресурс не найден, то возвращается «null» |
| Public | RunSceneHandler | Запускает цикличный поток для обработки: отрисовки игрой сцены на указанную панель, пользовательского ввода с различных устройств. |
| Public | ExitSceneHandler | Останавливает цикл обработки игровой сцены. |

При вызове конструктора инициализируются основные поля и свойства, а после управление передаётся методу «InitalizeSceneHandling», который занимается уже подключением определённых событий в зависимости от установленного типа объекта. Например, от объектов типа «EngineInputController» объявленные интерфейсы методов обработки ввода со стороны игрока подключаются к событиям ввода, которые объявлены в классе объекта главной панели: «KeyDown», «KeyUp», «MouseMove», «MouseDown» и «MouseUp». Для всех установленных объектов подключаются соответствующие обработчики для предоставления возможности объекту отображать графическую информацию на сцене (событие «Panel.Paint»). Также для того, чтобы объект был способен обновлять своё логическое состояние с течением времени, необходимый обработчик цепляется к объявленному внутри класса событию «EngineScene.UpdateHandling» (возможные подключаемые методы определяются типом делегата «UpdateOperationHandling», который устанавливает необходимую сигнатуру: в качестве параметра метода должен быть объект интерфейса сцены).



Рисунок 7 － Блок-схема метода «InitalizeSceneHandling»

Для запуска обработки объектов сцены используется метод «RunSceneHandler», который на основе экземпляра класса «Windows.Form.Timer» создаёт замкнутый цикл в отдельном потоке. Внутрь метода передаётся временной интервал между обновлениями отрисовки и логической части в миллисекундах. Также для вызова необходимо указать экземпляр делегата «Action», который будет определять действие при остановке работы игровой сцены при помощи метода «ExitSceneHandler».

Также для всех зарегистрированных объектов вызывается метод, отвечающий за первоначальную инициализацию. После чего запускается циклическая обработка. Чтобы указать операции, которые будут производиться постоянно с определённым интервалом во времени, используется событие «Tick». К нему привязывается анонимная функция обработчик, отвечающая за обновления объектов. Внутри её тела производится проверка на завершение обработки: в случае если триггер был включён, происходит остановка обработки и выполняется установленное конечное действие. Далее производится вызов всех функций привязанных к локальному событию «UpdateHandling», выполняющих обновление логики и состояния объекта. После посылается сообщение на фокусировку управления, ввода на элементе «Panel», вызывается метод «Invalidate», который вызывает отправку сообщения о необходимости повторной отрисовки элементу управления. В конце цикла необходимо очистить простаивающую область памяти (мусор) в куче при помощи метода статического метода «GC.Collect».

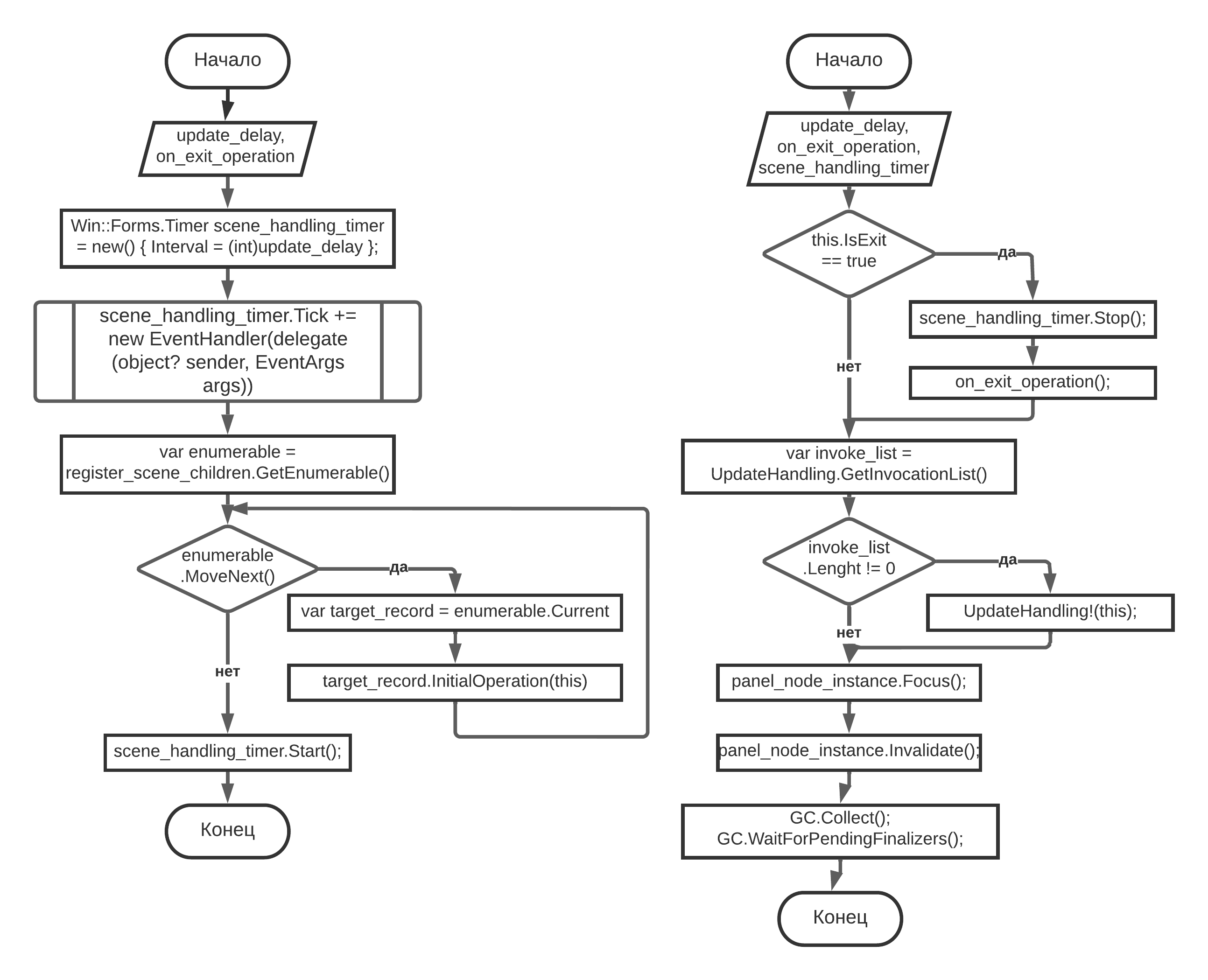


Рисунок 8 － Блок-схема метода «RunSceneHandler»

Внутри метода «SceneOnLoadAction», как ранее описывалось, происходит первоначальная настройка класса «EngineScene», отвечающего за игровой процесс, распределение настроек и объектов, настройка управления и связи с игроком. После создания экземпляра класса «EngineSceneBuilder», ответственного за добавление и настройку внутриигровых объектов, происходит подключение настроек и объектов, создание дочерних связей. Чтобы упросить и инкапсулировать создание и настройку объектов ответственных за различные аспекты игрового процесса, используются методы расширения класса «EngineSceneBuilder»: «AddChestSupport» － добавление ящика поддержки на сцену, столкновение с которым увеличивает значение здоровья игрока, «AddPlayer» － добавление главного персонажа с возможностью полного управления (перемещение, атаки и т. д.), модель которого полностью анимирована для разных команд, «AddSceneBorder» － создание границы карты (с графическим представлением), которая обрабатывает столкновения игрока и препятствует возможности покинуть локацию, «AddUserInterface» － подключение пользовательского графического интерфейса, предоставляющий информацию о времени текущего сеанса, доступного здоровья игрока и числа побеждённых противников.

Для добавления противников на карту используется цикл, в котором, на отрезке от 0 до максимального количества с шагом в единицу, вызывается метод «AddEnemy», растущее значение используется в создании уникального имени объекта. Также имена добавляются в коллекцию, чтобы в дальнейшем использовать их для соединения всех подключённых противников в родительский узел «EnemyManager» для управления поведением противников.

В конце загрузки создаётся сцена и посылается запрос на запуск обработчика внутриигровых процессов. Для вызова метода передаётся интервал обработки и анонимная функция для инициализации делегата на действие по окончанию работы сцены: вывод сообщение об завершении сессии, при помощи статического метода «Show» класса «MessageBox», после чего форма закрывается и управление передаётся на форму с главным меню.

Главным классом, определяющим свойства и базовую функциональность игровых объектов, подключаемых на сцену, является абстрактный класс «EngineObject». Внутри класса объявлены следующие свойства: «LinkedScene» － ссылка, предоставляющая доступ к родительской сцене, «Position» － текущее положение объекта относительно начала координат (левый верхний угол сцены), «Geometry» － установленные размеры тела объекта, «ParentObject» － ссылка на родительский объект (если текущий объект является для кого-то дочерним), «ObjectName» － уникальное имя объекта, «ObjectGuid» － уникальный зарегистрированный идентификатор объекта. Для того чтобы сборщик сцены был способен определить для каких необходимых свойств устанавливались значения конфигураций, используется атрибут «EngineObjectImportConfiguration», который устанавливается перед свойством, в конструкторе указывается имя запрашиваемой конфигурации. Сам базовый класс всех объектов также хранит коллекцию, подключённых дочерних объектов для работы с ними.

Интерфейс класса определяет некоторые методы для базовой функциональности объекта сцены. Все методы и их описание указаны в таблице 3.

Таблица 3 － Список объявляемых методов класса «EngineObject»

|  |  |  |
| --- | --- | --- |
| Модификатор доступа | Название метода | Описание метода |
| Public | ConnectLinkToChildren | Метод производит подключение дочерних объектов, где в случае успеха для каждого такого объекта в качестве родительского объекта через свойство «ParentObject» устанавливается настраиваемый экземпляр. Также для подключаемого объекта наследуется местоположение родителя и его размер. |
| Public | GetChildrenObject | Позволяет получить доступ к дочернему объекту по имени. Если запрашиваемый объект не был найден возвращается null. |
| Public | GetChildrenObjects | Позволяет получить доступ к множеству дочерних объектов на основе указанного типа. Если ничего не получилось найти, то возвращается пустая коллекция. |
| Public | ConnectLinkToScene | При помощи метода производится подключение главной сцены, внутри которой монтируются объекты. |
| Public | SetPosition | Позволяет переместить объект на указанную точку или в указанное направление. На ряду с перемещением объекта, перемещаются и все его подключённые дочерние сущности. |
| Public | PaintingOperation | Абстрактный метод, переопределение которого позволит определить совокупность операций, необходимых для отображения необходимой графической составляющей объекта при запросе на прорисовку со стороны сцены. В качестве параметра передаётся объект класса «Graphic», позволяющий использовать определённые методы отрисовки, результат которых отображается на сцене. |
| Public | InitialOperation | Виртуальный метод доступный для переопределения, используется, чтобы выполнить некоторые действия по начальной инициализации объекта. |
| Public | UpdateOperation | Виртуальный метод используемый для определения действий по обновлению состояния объекта. Вызов метода происходит постоянно пока запущена сцена, что позволяет установить логику для совершения действий над членами класса, которые должны регулярно проверяться или опрашиваться на наличие подходящего значения. |
| Public | Equals | Метод предоставляет инструмент для проверки идентичности двух объектов, используя для этого установленное имя. |

Создания пользовательского типа объекта сцены, необходимо унаследовать функционал класса «EngineObject». Для начала необходимо разобрать модуль, отвечающий создание изображения, отрисовку спрайтов и анимацию.

Классом отвечающим за создание спрайтов является «EngineSprite». Он содержит одно конфигурационное свойство «SpriteImagePath», для которого устанавливается путь до отображаемого файла. Внутри конструктора, на который необходимо устанавливать атрибут «EngineObjectCtorSelecter» (чтобы сборщик смог понять на основе чего создавать объект сцены), создаётся альтернативная отображаемая картина на случай, когда путь до отображаемой картинки отсутствует.

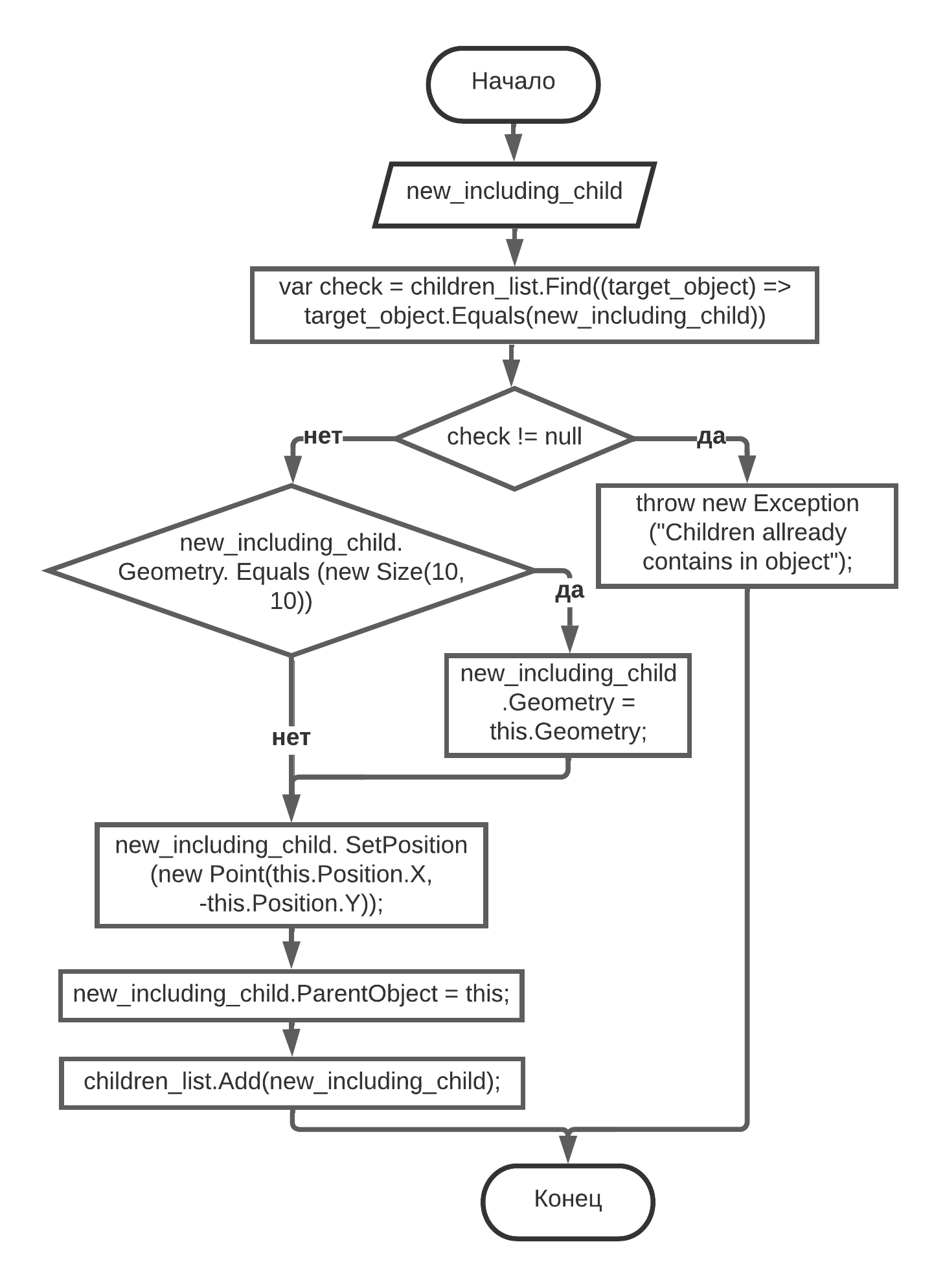


Рисунок 9 － Блок-схема метода «ConnectLinkToChildren»

Также класс переопределяет действие на виртуальный метод «InitialOperation», который теперь используется, чтобы на основе установленного пути попытаться открыть файл с изображением и загрузить его в виде класса «Image». В методе «PaintingOperation» происходит отрисовка установленной картинки по месту объекта (определяется свойством Position).

Для создания анимации используется класс «EngineAnimator», который предоставляет возможность загружать список с анимациями, запускать на отображение определённую и при необходимости останавливать. У класса в наличии имеется несколько свойств: «AnimationIsPlaying» позволяет определить запущена в данный момент какая-либо анимация, «AnimationName» возвращает имя запущенной анимации, «AnimationContainer» используется за создании конфигурации в виде списка с названиями подключаемых анимаций и путями до директорий с кадрами, «AnimationSpeed» используется для настройки скорости воспроизведения.

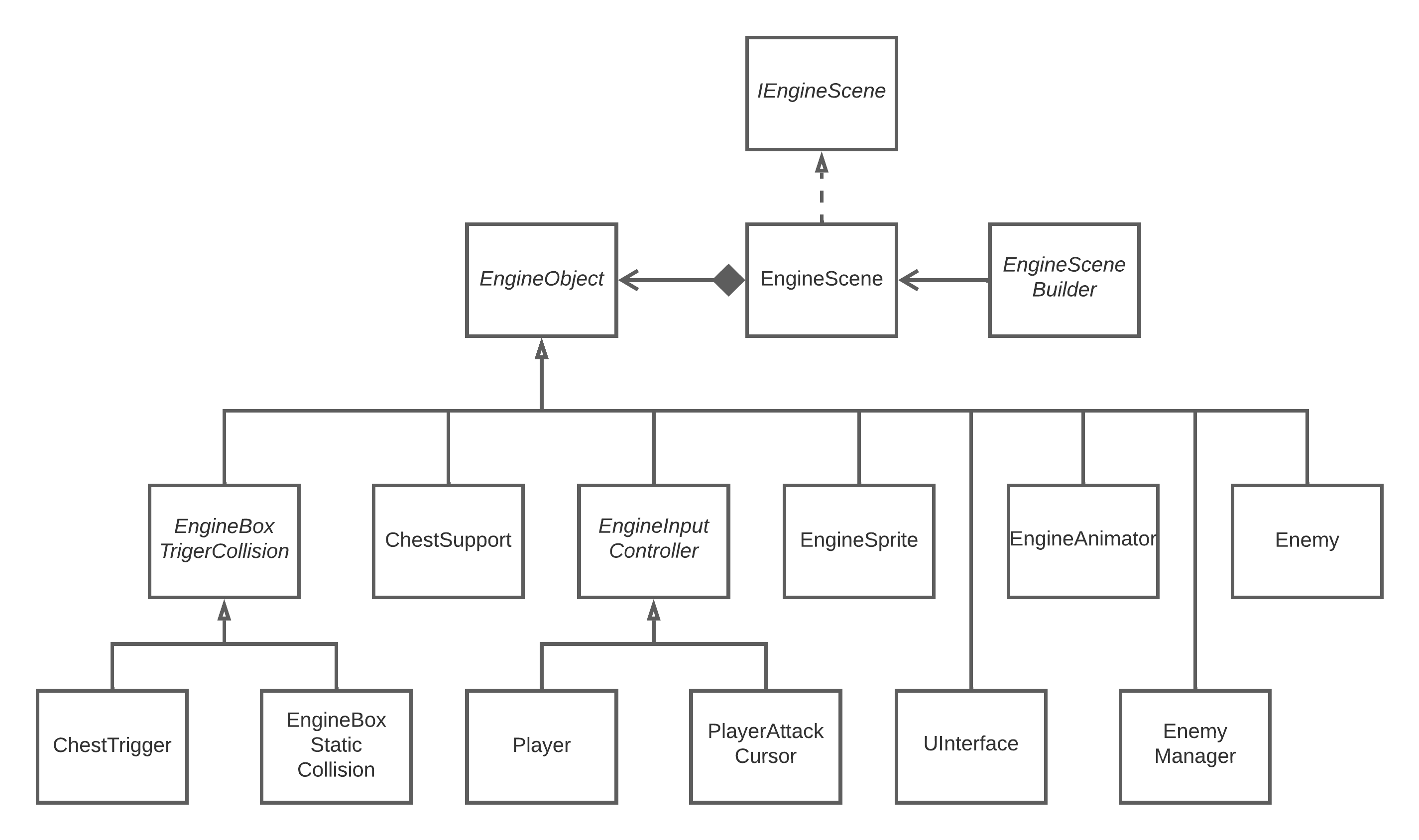


Рисунок 10 － Диаграмма отношений классов внутри сборки

В методе «InitialOperation» производится проход по каждой добавленной записи о подключаемой анимации, на основе пути до директории создаётся список из доступных файлов, относящихся к одной анимации. После собирается новая запись в контейнере, состоящая из ключа в виде названия зарегистрированной анимации и списка файлов относящихся к этой анимации. Далее открывается отдельный поток для обработки, который совершает переключение кадров запущенной анимации. Имеется два режима запуска анимации: с повтором － анимация бесконечно проигрывается, пока не будет заменена или остановлена, и без повтора － после того, как будут отрисованны кадры анимации выполнение завершается.

Для запуска анимации используется метод «PlayAnimation», который на основе передаваемого имени устанавливает значения для записи о текущей проигрываемой анимации: имя, номер текущего проигрываемого кадра и значение, определяющее повторяется ли анимация. Для остановки используется метод «StopCurrentAnimation», который производит очистку вышеназванной записи. Метод «PaintingOperation» производит отрисовку изображения на основе запущенной анимации.

Для определение события столкновения двух объектов прямоугольной формы, обладающих коллизией, используется абстрактный класс «EngineBoxTrigerCollision», который обладает некоторыми свойствами: «BorderDraw» определяет будет ли проводится отрисовка границы коллизии, «TypesForCollide» определяет множество типов, с дочерними коллизиями которых будет регистрироваться столкновение. Внутри объявлен абстрактный метод «OnTriggerDetectCollision», который определяет действие в производных классах на событие столкновения с другими объектами; в качестве параметров принимает ссылку на объект, с которым произошло столкновение, и значение, определяющее с какой стороны ударились объекты. В методе «UpdateOperation» проверяется было ли совершено столкновение с одним из зарегистрированным объектом данного класса. Для начала при помощи подключённой ссылке на экземпляр сцены получаем множество объектов данного класса, которые потенциально пригодны для столкновения. После производится фильтрация по ранее установленным в конфигурации типам, определяющим с какими объектам регистрировать столкновения, а каких стоит игнорировать. Алгоритм расчёта столкновения заключается в вычислении значения координат для каждой оси между противоположными краями объектов, и если при вычислении оказывается, что один их краёв заступает за поверхность, то регистрируется столкновение.

Рисунок 10 － Диаграмма отношений классов внутри сборки



Рисунок 11－ Принцип определения столкновений двух тел

Производный класс «EngineBoxStaticCollision» используется для создания столкновений, при которых объекты не позволяют друг другу пройти сквозь себя. Для этого в переопределенном методе «OnTriggerDetectCollision» сохраняется значение о предыдущем положении объекта, на основе которого рассчитывается вектор перемещения. Учитывая с какой стороны произошло столкновение, создаётся обратный вектор, с помощью которого происходит отталкивание объекта в пространстве.

Для создания главного персонажа используется класс «Player», наследуемый от абстрактного класса «EngineInputController», который определяет интерфейс для создания обработки событий управления со стороны пользователя программы. Сам объект главного героя является многослойным, так как включает в себя некоторые дочерние объекты, такие как: объект класса «EngineAnimator», отвечающий за анимацию героя, объект класса «EngineBoxStaticCollision», который определяет возможность игрока сталкиваться с другими объектами на карте (например, со стеной, окружающей уровень, или противниками) и др.

Класс определяет свойства: «LifeCount» показывает текущее количество здоровья, «IsAlive» － состояние героя (живой он или уже нет), «IsReadyToAttack» － состояние готовности атаковать, «MovementSpeed» － скорость передвижения персонажа. В методе «InitialOperation», запрашиваются и устанавливаются ссылки на дочерние объекты. Метод «HealingAction» позволяет увеличить значение здоровья главного героя, в то время как метод «DamageRegistration» используется для регистрации урона по персонажу от внешних фактором: для начала проверяется является ли объект ещё «живым», после чего значение здоровья уменьшается и проводится проверка является ли текущее значение отрицательным или равным нулю: если утверждение истинно, то состояние инвертируется и запускается анимация «гибели» главного героя; иначе запускается анимация получения урона.

За обработку события нажатия кнопки на клавиатуре используется метод «KeyInputOperation», где с помощью формального параметра можно узнать, что за клавиша была нажата. Эта возможность используется, чтобы составить единичный вектор направления движения по двум осям. Для определения момента высвобождения клавиши, используется метод «KeyReleaseOperation», который выполняет обнуление единичных векторов.

Обработка нажатия кнопки мыши происходит при помощи метода «MouseReleaseOperation», в теле которого в начале совершается проверка запущена какая-либо анимация атаки, и если утверждение ложно, то запускается одна из трёх доступных анимация (выбор происходит случайным образом) и меняется состояние доступности атаки.

Метод обновления состояния «UpdateOperation» начинается с определения текущей доступной скорости персонажа: если направление единичного вектора движения и направление взгляда персонажа различаются, то скорость уменьшается в 1.5 раза. После происходит проверка на активность объекта, чтобы совершить перемещение в пространстве в соответствии с произведением единичного вектора направления движения на скорость.

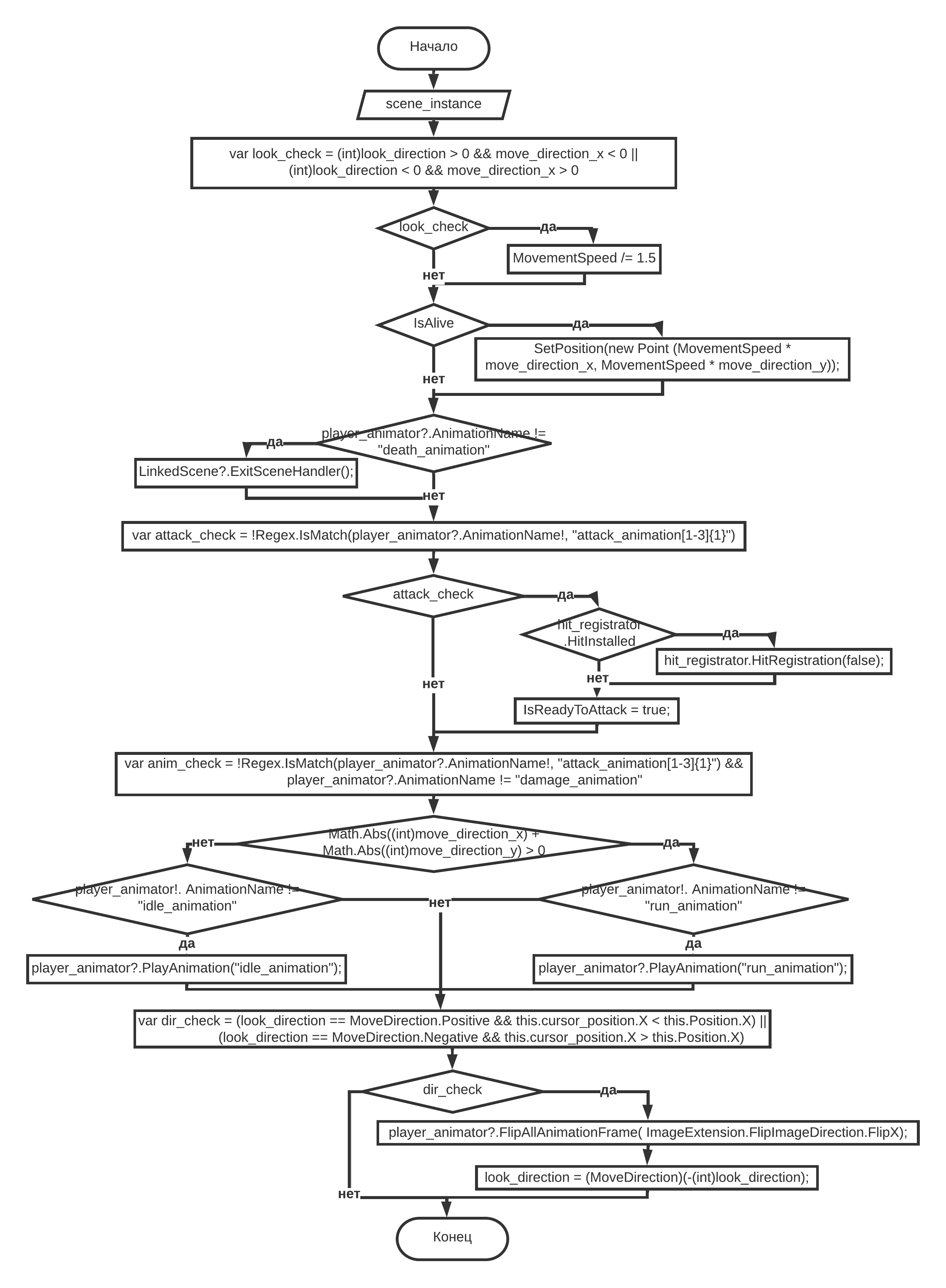


Рисунок 12－ Блок-схема метода «UpdateOperation» класса «Player»

Далее совершается проверка завершения выполнения анимации «гибели» и значения состояния жизни героя: в случае истинного утверждения выполнение обработки сцены прекращается и совершается выход. После проверяется условие, что была ли завершена анимация «атаки»: если да, то обновляются состояние готовности к атаки и регистратора урока по врагам. Далее необходимо совершить проверку возможности запуска зацикленной анимации «бездействия» или «передвижения», в зависимости от значений единичных векторов направления движения.

В конце происходит обновление направления обзора игрока и инвертируется направление каждого кадра всех анимации, в соответствием с положение курсора мыши относительно модели персонажа.

Класс «PlayerAttackCursor» занимается составлением логики и графической составляющей курсора (триггера) для регистрации попадания атаки, которая становиться доступна при вхождении коллизии врага в поле обработки триггера.

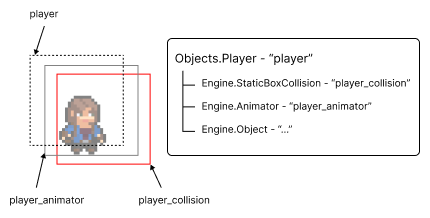


Рисунок 13－Представление слоистой структуры сложных объектов

Объект противника описан в классе «Enemy», который является производным от абстрактного класса «EngineObject». Сущность противники составляется из разных узлов: объект коллизии, объект анимации, объект регистрации урона и объект фиксации атаки по герою. Внутри класса объявлены свойства: «IsAlive» － состояние активности объекта в сеансе, «IsReadyToRefresh» － готовность объекта для обновления или перезапуска на сцене, «TargetUpdateSpeed» － определяет скорость поиска цели для преследования и атаки, «MaxMovementSpeed» － устанавливает максимальную скорость, которую может развить враг. Используя метод «EnemyHitPlayer» для атаки героя, запускается анимация «удара» в зависимости от состоянии активности объекта. Метод «DamageRegistration» регистрирует получаемый урон, со стороны игрока, для этого уменьшается значение здоровья противника и запускается разовая анимация «получения урона» или «гибели». Для обновления состояния и значений объекта используется метод «Refresh», который позволяет сбросить урон, полученный до момента вызова, и вернуть начальное состояние объекта.

При вызове метода «InitialOperation» выполняется создание локальные ссылок на дочерние объекта и глобальные объекты сцены (экземпляр игрока, объект анимации, регистратор попаданий ударов), запускается анимация «передвижения», а в конце открывается дополнительный поток, который со скоростью установленной в свойстве «TargetUpdateSpeed» запрашивает текущее положение цели (игрока). Внутри тела метода «PaintingOperation» происходит отображение текущего значение здоровья.

Метод «UpdateOperation» отвечает за создание вектора перемещения в сторону указанной цели, проверка и обновление состояний перезапуска и активности, запуск нужных анимаций.

Чтобы управлять объектами, являющимися противниками игрока, используется объект класса «EnemyManager», все экземпляры объектов данного типа выступают в качестве дочерних узлов. Менеджер занимается инициализацией объектов «врагов» и отвечает за их повторное добавление после отключения (устранения).

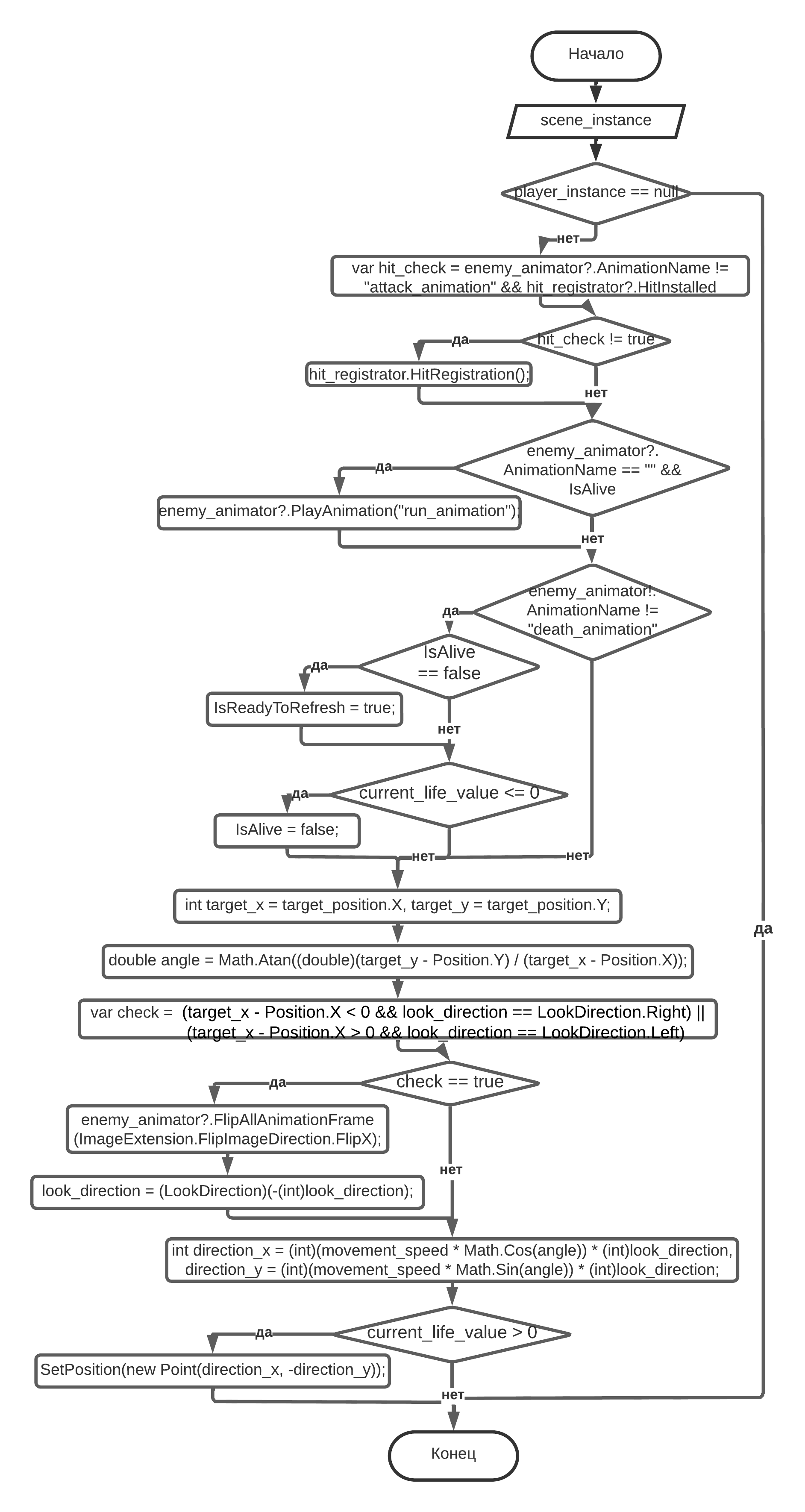


Рисунок 14－ Блок-схема метода «UpdateOperation» класса «Enemy»

## 3 Контрольный пример работы программного средства

При запуске исполняемого файла открывается окно главного меню, откуда пользователь получает возможность настроить грядущую игровую сессию под себя, подобрав необходимую конфигурацию.



Рисунок 15 － Представление окна главного меню

Для настройки игроку доступны 6 параметров: «EnemyCount» － максимальное количество врагов в сессии, «EnemySpeed» － максимальная развиваемая вражеская скорость, «ChestSpawn» － интервал между появлением ящиков помощи (для пополнения здоровья), «PlayerLife» － начальное количество здоровья, «PlayerSpeed» － максимальная скорость игрока, «Debug» － флаг для включения или отключения режима отладки.

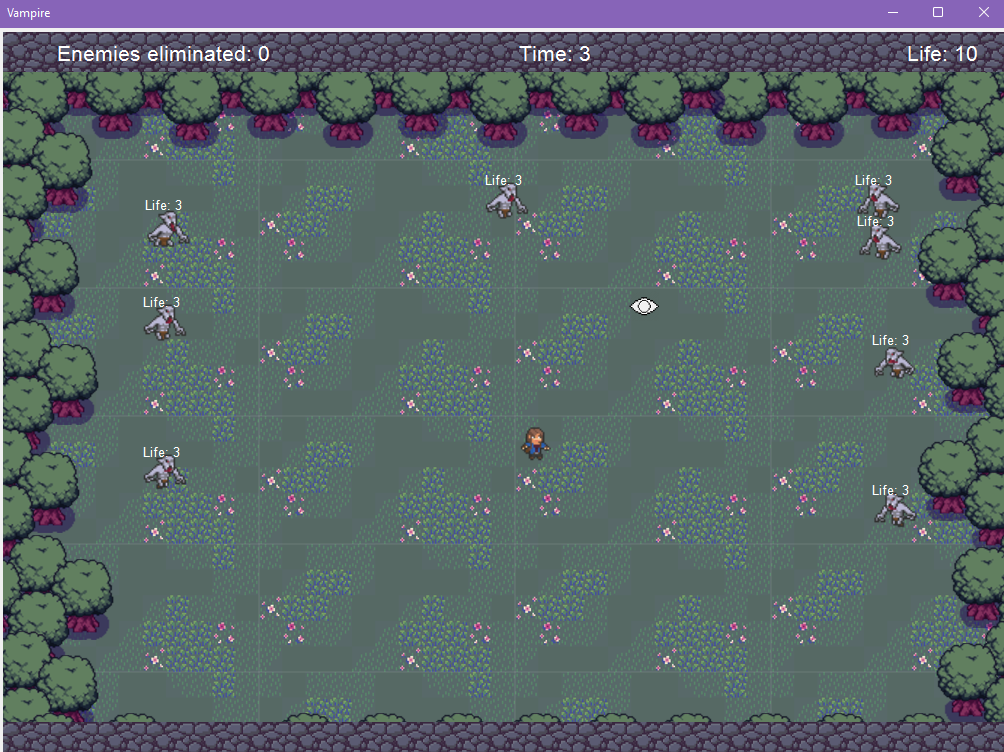


Рисунок 16 － Начальное положение персонажей при запуске

При нажатии на кнопку «Play game» запускается окно с игрой. Главный герой появляется посредине поля окружённого непроходимым лесом, из которого появляются противники, которые стремятся нанести урон игроку. Сверху пользователь может посмотреть информацию о количестве устранённых врагов, прошедшем с начала игры времени (в секундах) и оставшемся количестве здоровья. На месте курсора демонстрируется иконка, обозначающая текущее совершаемое действие игрока: «Глаз» － игрок стоит на месте или передвигается, «Меч» － игрок готовиться атаковать в выбранном направлении.

Для перемещения используются «стрелки» на клавиатуре, где каждая кнопка отвечает за определённое направление. Если направление взгляда персонажа и направление движения различаются, то игрок будет перемещаться с пониженной скоростью. Чтобы изменить направление взгляда главного героя необходимо переместить указатель курсора в нужную сторону относительно модели персонажа игрока.

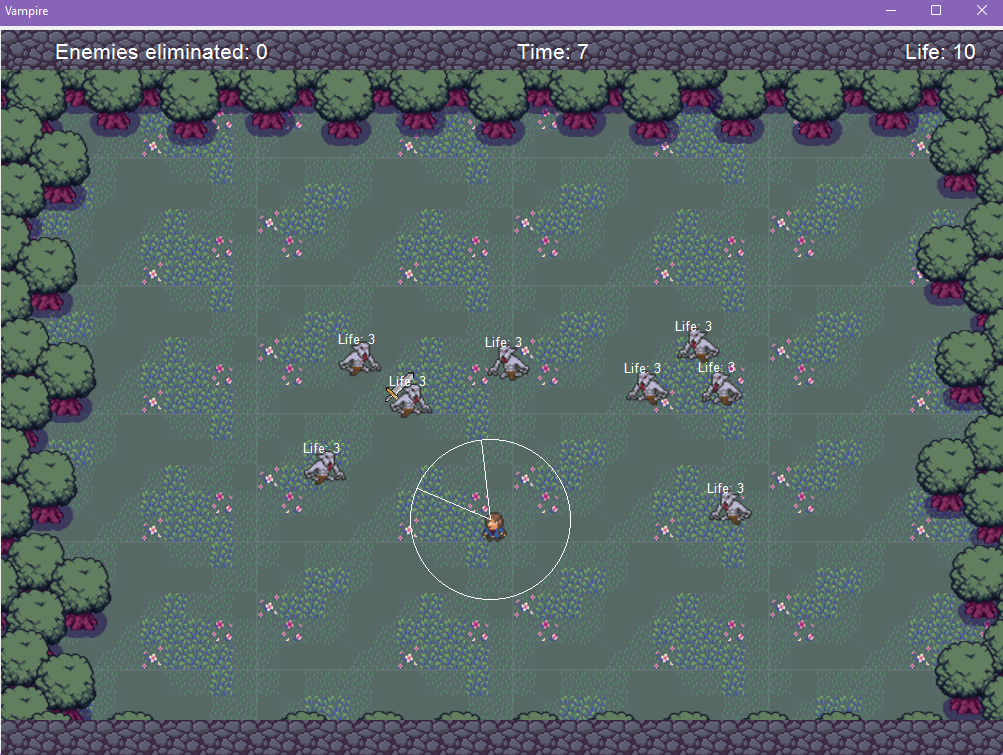


Рисунок 17 － Демонстрация режима атаки игрока

Для совершения атаки используется компьютерная мышь: удержание левой кнопки мыши переводит главного героя в режим атаки, с помощью перемещения указателя мыши выбирается направление атаки. При высвобождении кнопки мыши производиться удар и ближайшему противнику, попавшему в зону атаки, наносится урон (значение здоровья декрементируется). Когда количество жизней врага достигнет отметки 0, то после проигрывания анимации, объект переместиться на случайную стартовую позицию, скорость передвижения персонажа увеличиться.

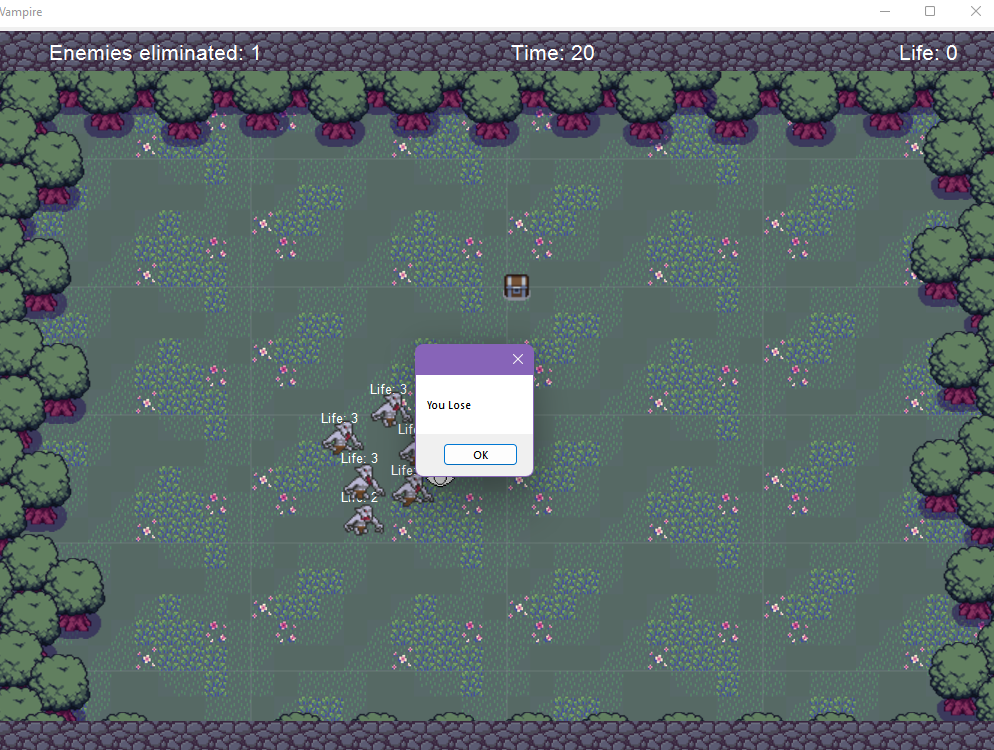


Рисунок 18 － Сообщение о поражении

Если не успеть уклониться от атаки противника, то у игрока декрементируется значение здоровья. Поражение засчитывается, когда у игрока значение «жизней» достигает нуля. В этом случае на экране появиться сообщение о поражении.

На арене с установленным временным интервалом появляются сундуки, подойдя и открыв которые можно восстановить некоторое количество здоровья. Позиция ящиков определяется случайным образом.



Рисунок 19 － Демонстрация сундука поддержки

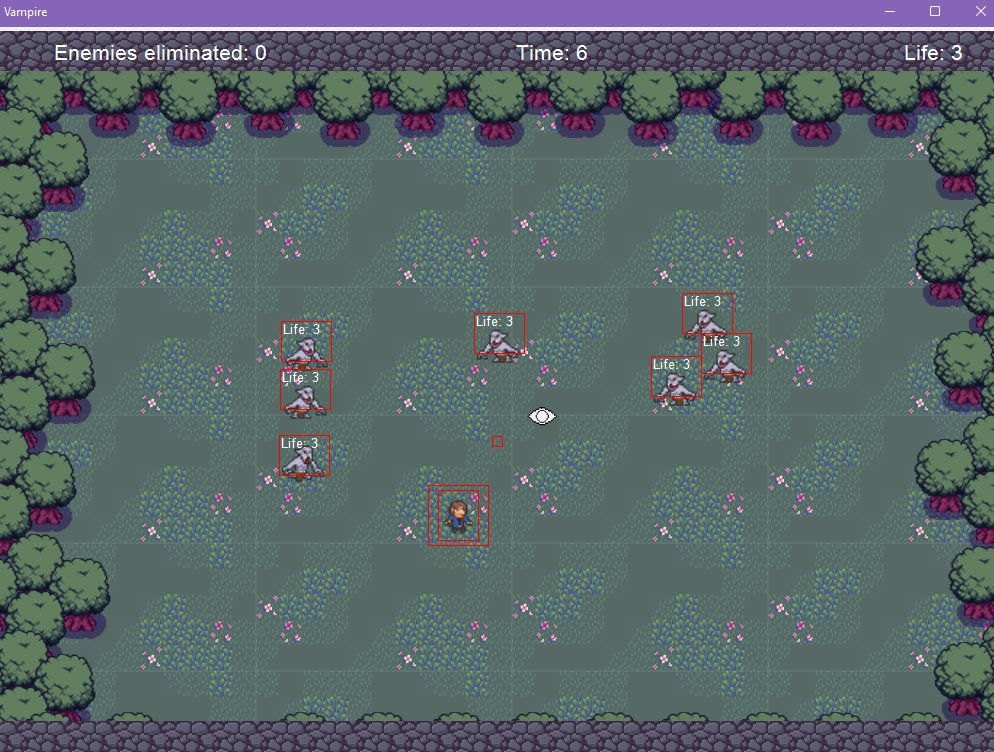


Рисунок 20 － Режим отладки

# ЗАКЛЮЧЕНИЕ

В ходе прохождения практики было разработано программное обеспечение (игра) «Vampire» с использованием среды разработки Visual Studio 2022.

Программа состоит из двух форм: главного меню, которое позволяет настроить параметры текущей игровой сессии, и сцены активного уровня, который предоставляет пользователю возможность управлять игровым процессом.

Программное приложение позволяет пользователю проверить свою реакцию и способность анализировать окружение для получения наибольшего количества очков за устранение врагов, спокойно провести своё сводное время в игре и отдохнуть.

Все требования к программе выполнены успешно. Разработан удобный инструмент для настройки и создания новых объектов, разработки новых пользовательских типов. Добавлена поддержка отображения спрайтов, текстур игровых объектов и создания по кадровых анимаций. Разработан принцип и спроектирован модуль обработки столкновений внутриигровых объектов. Создан алгоритм появления и перемещения вражеских персонажей по арене.

Внутри игрового процесса производится отслеживания пользовательского ввода с клавиатуры и компьютерной мыши. Также у пользователя есть возможно создать пользователю настраивать грядущую игровую сессии, предоставить доступ к конфигурации объектов;

Реализован компонент, отвечающий за регистрацию ударов от врагов и от игрока, а также генерации ящиков поддержки на локации в пределах определённой установленной области.

# Список использованных источников

1. Microsoft Corporation. Принципы проектирования и разработки программного обеспечения. Учеб. курс MCSD. М.: Изд.-торг. дом «Русская редакция», 2000 (дата обращения: 22.06.2022).
2. Windows Forms. Программирование на C# [Электронный ресурс]. — Режим доступа: http://csharpcoding.org/category/windows-forms/ (дата обращения: 23.06.2022).
3. Мазалов В. В. Математическая теория игр и приложения: учебное пособие для вузов (дата обращения: 24.06.2022).
4. Биллиг В.A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] (дата обращения: 25.06.2022).

# ПРИЛОЖЕНИЕ А

# код программного средства

**Program.cs**

namespace PracticeWork

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

Application.Run(new MainMenu());

}

}

}

**MainScene.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace PracticeWork

{

public partial class MainMenu : Form

{

public MainMenu()

{

InitializeComponent();

this.BackgroundImage = Image.FromFile(@"..\..\..\Assets\Menu\background.png");

this.player\_picturebox.Image = Image.FromFile(@"..\..\..\Assets\Player\Idle\test\_01.png");

Image enemy\_logo = Image.FromFile(@"..\..\..\Assets\Enemy\Idle\enemy\_01.png");

enemy\_logo.RotateFlip(RotateFlipType.RotateNoneFlipX);

this.enemy\_picturebox.Image = enemy\_logo;

}

private void play\_button\_Click(object sender, EventArgs e)

{

Form form = new MainScene()

{

PlayerLife = (int)this.playerlife\_numeric.Value,

PlayerSpeed = (double)this.playerspeed\_numeric.Value,

EnemyCount = (int)this.enemycount\_numeric.Value,

MaxEnemySpeed = (double)this.enemyspeed\_numeric.Value,

DebugMode = this.debug\_checkbox.Checked,

ChestSpawn = (int)this.chestspawn\_numeric.Value

};

form.ShowDialog();

}

}

}

**MainScene.cs**

using PracticeWork.Objects;

using PracticeWork.Engine;

using PracticeWork.Configuration;

namespace PracticeWork

{

public partial class MainScene : Form

{

public System.Int32 EnemyCount { get; set; } = 5;

public System.Int32 PlayerLife { get; set; } = 3;

public System.Int32 ChestSpawn { get; set; } = 30;

public System.Double MaxEnemySpeed { get; set; } = 4.0;

public System.Double PlayerSpeed { get; set; } = 4.0;

public System.Boolean DebugMode { get; set; } = false;

public MainScene()

{

InitializeComponent();

InitializeMainPanel(new Point(4, 4), 1024, 768);

this.SetStyle(ControlStyles.OptimizedDoubleBuffer | ControlStyles.AllPaintingInWmPaint

| ControlStyles.UserPaint, true);

this.UpdateStyles();

this.main\_panel!.BackgroundImage = Image.FromFile(@"..\..\..\Assets\Scene\background2.png");

this.Load += SceneOnLoadAction;

}

private void SceneOnLoadAction(object? sender, EventArgs e)

{

Engine.EngineSceneBuilder scene\_builder = new("Scene", this.main\_panel!);

scene\_builder.AddChestSupport(ChestSpawn, DebugMode);

scene\_builder.AddPlayer(DebugMode, PlayerLife, PlayerSpeed);

List<string> eneme\_manager\_children = new();

scene\_builder.RegisterSceneConfiguration<int>("enemy\_manager", "EnemyCount", EnemyCount);

for (int i = 0; i < EnemyCount; i++)

{

scene\_builder.AddEnemy(DebugMode, i, MaxEnemySpeed);

eneme\_manager\_children.Add("enemy" + i);

}

scene\_builder.RegisterSceneChild<Objects.EnemyManager>("enemy\_manager", eneme\_manager\_children);

scene\_builder.AddSceneBorder(@"..\..\..\Assets\Scene\border2.png");

scene\_builder.AddUserInterface(16);

scene\_builder.BuildScene().RunSceneHandler(15, () => { MessageBox.Show("You Lose"); this.Close(); });

}

}

}

**EngineCollision.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Engine

{

public abstract class EngineBoxTrigerCollision : Engine.EngineObject

{

public enum CollideSide : System.UInt16 { None = 0, Left, Top, Right = 4, Bottom = 8 };

[Engine.EngineObjectImportConfiguration("BorderDraw")]

public System.Boolean BorderDraw { get; protected set; } = false;

[Engine.EngineObjectImportConfiguration("TypesForCollide")]

public List<System.Type>? CollisionTargets { get; protected set; } = null;

public EngineBoxTrigerCollision(string object\_name) : base(object\_name) { }

protected abstract void OnTriggerDetectCollision(Engine.EngineObject target\_object, CollideSide side);

public override void UpdateOperation(IEngineScene scene\_instance)

{

this.LinkedScene?.GetSceneObjects<EngineBoxTrigerCollision>().ForEach((target\_object) =>

{

if (this.CollisionTargets != null)

{

if (this.CollisionTargets?.Find((System.Type searching) =>

{

return searching == target\_object.ParentObject!.GetType();

}) == null) return;

}

else return;

if (this.CollisionChecker((Engine.EngineBoxTrigerCollision)target\_object, out var side))

this.OnTriggerDetectCollision(target\_object, side);

});

}

public override void PaintingOperation(Graphics graphic)

{

if (this.BorderDraw) graphic.DrawRectangle(Pens.Red, new Rectangle(this.Position, this.Geometry));

}

private System.Boolean CollisionChecker(EngineBoxTrigerCollision target\_collision, out CollideSide side)

{

if (target\_collision.ObjectName == this.ObjectName) { side = default; return false; }

bool check\_x = (this.Position.X + this.Geometry.Width >= target\_collision.Position.X) &&

(this.Position.X <= target\_collision.Position.X + target\_collision.Geometry.Width);

bool check\_y = (this.Position.Y + this.Geometry.Height >= target\_collision.Position.Y) &&

(this.Position.Y <= target\_collision.Position.Y + target\_collision.Geometry.Height);

side = CollideSide.None;

int target\_delta\_y = target\_collision.Position.Y + target\_collision.Geometry.Height,

target\_delta\_x = target\_collision.Position.X + target\_collision.Geometry.Width;

int body\_delta\_y = Position.Y + Geometry.Height, body\_delta\_x = Position.X + Geometry.Width;

bool left\_topbottom = ((body\_delta\_y - target\_collision.Position.Y < target\_delta\_x - Position.X)

|| (target\_delta\_y - Position.Y < target\_delta\_x - Position.X));

bool right\_topbottom = ((target\_delta\_y - Position.Y < body\_delta\_x - target\_collision.Position.X)

|| (body\_delta\_y - target\_collision.Position.Y < body\_delta\_x - target\_collision.Position.X));

int corner\_delta\_x = Math.Min(target\_collision.Position.X + target\_collision.Geometry.Width - this.Position.X,

this.Position.Y + this.Geometry.Height - target\_collision.Position.Y);

int corner\_delta\_y = Math.Min(target\_collision.Position.Y + target\_collision.Geometry.Height - this.Position.Y,

this.Position.Y + this.Geometry.Height - target\_collision.Position.Y);

if (check\_x == true && !(left\_topbottom && right\_topbottom))

{

if (target\_collision.Position.X + target\_collision.Geometry.Width - this.Position.X <

this.Position.X + this.Geometry.Width - target\_collision.Position.X) side = CollideSide.Left;

else side |= CollideSide.Right;

}

if(check\_y == true && (left\_topbottom && right\_topbottom))

{

if (target\_collision.Position.Y + target\_collision.Geometry.Height - this.Position.Y <

this.Position.Y + this.Geometry.Height - target\_collision.Position.Y) side |= CollideSide.Top;

else side |= CollideSide.Bottom;

}

return (check\_x && check\_y);

}

}

public sealed class EngineBoxTriggerCollisionNone : Engine.EngineObject

{

[Engine.EngineObjectConstructorSelecter]

public EngineBoxTriggerCollisionNone(string object\_name) : base(object\_name) { }

public override void PaintingOperation(Graphics graphic) { return; }

}

public sealed class EngineBoxStaticCollision : Engine.EngineBoxTrigerCollision

{

private System.Drawing.Point previous\_position = new Point(0, 0);

[Engine.EngineObjectConstructorSelecter]

public EngineBoxStaticCollision(string object\_name) : base(object\_name) { }

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

int moving\_direction\_x = this.Position.X - this.previous\_position.X,

moving\_direction\_y = this.previous\_position.Y - this.Position.Y;

if ((side & CollideSide.Left) != CollideSide.None)

{

if (moving\_direction\_x > 0) moving\_direction\_x = 0; else moving\_direction\_x = -moving\_direction\_x;

}

if ((side & CollideSide.Right) != CollideSide.None)

{

if (moving\_direction\_x < 0) moving\_direction\_x = 0; else moving\_direction\_x = -moving\_direction\_x;

}

if ((side & CollideSide.Top) != CollideSide.None)

{

if (moving\_direction\_y < 0) moving\_direction\_y = 0; else moving\_direction\_y = -moving\_direction\_y;

}

if ((side & CollideSide.Bottom) != CollideSide.None)

{

if (moving\_direction\_y > 0) moving\_direction\_y = 0; else moving\_direction\_y = -moving\_direction\_y;

}

this.ParentObject?.SetPosition(new(moving\_direction\_x, moving\_direction\_y));

}

public override void UpdateOperation(IEngineScene scene\_instance)

{

base.UpdateOperation(scene\_instance);

this.previous\_position = new(this.Position.X, this.Position.Y);

}

}

}

**EngineGraphic.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Win = System.Windows;

namespace PracticeWork.Engine

{

public sealed class EngineSprite : Engine.EngineObject

{

[Engine.EngineObjectImportConfiguration("SpriteImagePath")]

public System.String? SpriteImagePath { get; protected set; }

public System.Drawing.Image? SpriteImage { get; private set; }

[Engine.EngineObjectConstructorSelecter]

public EngineSprite(string object\_name) : base(object\_name)

{

System.Drawing.Bitmap bitmap = new Bitmap(this.Geometry.Width, this.Geometry.Height);

using (System.Drawing.Graphics image\_graphic = Graphics.FromImage(bitmap))

{

image\_graphic.DrawRectangle(Pens.Black, new Rectangle(this.Position, this.Geometry));

}

this.SpriteImage = (Image)bitmap.Clone();

}

public void SpriteFlip(ImageExtension.FlipImageDirection direction) => this.SpriteImage?.FlipImage(direction);

public void SpriteRotate(System.Double rotation\_angle) => this.SpriteImage?.RotateImage((float)rotation\_angle);

public override void InitialOperation(IEngineScene scene\_instance)

{

if (this.SpriteImagePath is not null)

{

using (System.IO.FileStream file\_stream = new FileStream(this.SpriteImagePath, FileMode.Open))

this.SpriteImage = Image.FromStream(file\_stream);

}

}

public override void PaintingOperation(Graphics graphic)

{

graphic.DrawImage(this.SpriteImage!, new Rectangle(this.Position, this.Geometry));

}

}

public sealed class EngineAnimator : Engine.EngineObject

{

public record AnimationRecord(List<System.Drawing.Image> AnimationFrames);

public sealed record AnimationLinkedFrame(string AnimationName, int CurrentFrame, bool Repeat);

private Dictionary<string, EngineAnimator.AnimationRecord> animation\_list = new();

private System.Double animation\_playing\_speed = 1.0;

private EngineAnimator.AnimationLinkedFrame? current\_frame = null;

public System.Boolean AnimationIsPlaying { get; private set; } = false;

public System.String? AnimationName { get => this.current\_frame?.AnimationName; }

[Engine.EngineObjectImportConfiguration("AnimationsContainer")]

public Dictionary<string, string> AnimationsContainer { get; protected set; }

[Engine.EngineObjectImportConfiguration("AnimationSpeed")]

public System.Double AnimationsSpeed

{

protected set { if (value <= 1.0 && value > 0) this.animation\_playing\_speed = value; }

get => this.animation\_playing\_speed;

}

[Engine.EngineObjectConstructorSelecter]

public EngineAnimator(string object\_name) : base(object\_name) => this.AnimationsContainer = new();

public override void InitialOperation(IEngineScene scene\_instance)

{

foreach (KeyValuePair<string, string> target\_item in this.AnimationsContainer)

{

List<string> filepath\_list = Directory.EnumerateFiles(target\_item.Value, "\*.png").ToList();

if (filepath\_list.Count == 0) continue;

this.animation\_list.Add(target\_item.Key, new AnimationRecord(new(0)));

filepath\_list.ForEach(delegate (string target\_file)

{

this.animation\_list[target\_item.Key].AnimationFrames.Add(Image.FromFile(target\_file));

});

this.current\_frame = new(target\_item.Key, 0, false);

}

Win::Forms.Timer animation\_handling\_timer = new() { Interval = (int)(60 / this.AnimationsSpeed) };

animation\_handling\_timer.Tick += new EventHandler(this.AnimationHandling);

animation\_handling\_timer.Start();

}

private void AnimationHandling(object? sender, System.EventArgs event\_arg)

{

if (this.AnimationIsPlaying == true && this.current\_frame != null)

{

var max\_frame = this.animation\_list[current\_frame?.AnimationName!].AnimationFrames.Count;

if (this.current\_frame?.CurrentFrame < max\_frame - 1)

{

this.current\_frame = new(current\_frame.AnimationName, current\_frame.CurrentFrame + 1,

current\_frame.Repeat);

}

else

{

this.current\_frame = new(current\_frame!.AnimationName, 0, current\_frame.Repeat);

if (this.current\_frame!.Repeat != true) this.StopCurrentAnimation();

}

}

}

public void PlayAnimation(System.String animation\_name, bool is\_repeat = true)

{

if (this.animation\_list.ContainsKey(animation\_name) == true)

{

this.current\_frame = new EngineAnimator.AnimationLinkedFrame(animation\_name, 0, is\_repeat);

this.AnimationIsPlaying = true;

}

}

public void StopCurrentAnimation() => (this.AnimationIsPlaying, this.current\_frame)

= (false, new ("", 0, false));

public void FlipAllAnimationFrame(ImageExtension.FlipImageDirection direction)

{

foreach (KeyValuePair<string, AnimationRecord> target\_item in this.animation\_list)

{

target\_item.Value.AnimationFrames.ForEach((Image image) => image.FlipImage(direction));

}

}

//public override void UpdateOperation(IEngineScene scene\_instance) { return; }

public override void PaintingOperation(Graphics graphic)

{

if (this.current\_frame != null)

{

(string animation\_name, int frame\_index, \_) = this.current\_frame;

if (this.animation\_list.ContainsKey(animation\_name) != true) return;

Image output\_image = this.animation\_list[animation\_name].AnimationFrames[frame\_index];

try

{

graphic.DrawImage(output\_image, new Rectangle(this.Position, this.Geometry));

}

catch (System.Exception)

{

Console.WriteLine(ParentObject!.Position.X + "; " + ParentObject!.Position.Y);

this.ParentObject.SetPosition(this.LinkedScene!.SceneSize.Width / 2, 0);

}

}

}

}

public static class ImageExtension

{

public enum FlipImageDirection : UInt16 { FlipX = 0, FlipY };

public static Image RotateImage(this Image sourse\_image, float rotation\_angle)

{

System.Drawing.Bitmap buffer\_image = new(sourse\_image.Width, sourse\_image.Height);

using (System.Drawing.Graphics image\_graphic = Graphics.FromImage(buffer\_image))

{

image\_graphic.TranslateTransform((float)buffer\_image.Width / 2, (float)buffer\_image.Height / 2);

image\_graphic.RotateTransform(rotation\_angle);

image\_graphic.TranslateTransform(-(float)buffer\_image.Width / 2, -(float)buffer\_image.Height / 2);

image\_graphic.DrawImage(sourse\_image, new Point(0, 0));

}

return buffer\_image;

}

public static Image FlipImage(this Image sourse\_image, FlipImageDirection flip\_direction)

{

switch (flip\_direction)

{

case FlipImageDirection.FlipX: sourse\_image.RotateFlip(RotateFlipType.RotateNoneFlipX); break;

case FlipImageDirection.FlipY: sourse\_image.RotateFlip(RotateFlipType.RotateNoneFlipY); break;

}

return sourse\_image;

}

}

}

**EngineObject.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Win = System.Windows;

namespace PracticeWork.Engine

{

[System.AttributeUsage(AttributeTargets.Constructor, AllowMultiple = false)]

public sealed class EngineObjectConstructorSelecter : System.Attribute { }

[System.AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]

public sealed class EngineObjectImportConfiguration : System.Attribute

{

public string PropertyName { get; private init; }

public EngineObjectImportConfiguration(string property\_name) => (this.PropertyName) = (property\_name);

}

public abstract class EngineObject : System.Object

{

protected Engine.IEngineScene? LinkedScene { get; private set; } = null;

private List<Engine.EngineObject> children\_list = new List<Engine.EngineObject>(0);

[Engine.EngineObjectImportConfiguration("ObjectPosition")]

public System.Drawing.Point Position { get; protected set; } = new Point(0, 0);

[Engine.EngineObjectImportConfiguration("ObjectGeometry")]

public System.Drawing.Size Geometry { get; set; } = new Size(10, 10);

public Engine.EngineObject? ParentObject { get; set; } = null;

public System.String ObjectName { get; init; }

protected System.Guid ObjectGuid { get; init; } = Guid.NewGuid();

protected EngineObject(string object\_name) : base() => this.ObjectName = object\_name;

public void ConnectLinkToChildren(Engine.EngineObject new\_including\_child)

{

if (children\_list.Find((target\_object) => target\_object.Equals(new\_including\_child)) != null)

throw new Exception("Children allready contains in object");

if(new\_including\_child.Geometry.Equals(new Size(10, 10)))

new\_including\_child.Geometry = this.Geometry;

//if (new\_including\_child.ObjectName == "player\_damage")

// Console.WriteLine(new\_including\_child.Position.X + ";\t" + new\_including\_child.Position.Y);

new\_including\_child.SetPosition(new Point(this.Position.X, -this.Position.Y));

//if (new\_including\_child.ObjectName == "player\_damage")

// Console.WriteLine(new\_including\_child.Position.X + ";\t" + new\_including\_child.Position.Y);

new\_including\_child.ParentObject = this;

this.children\_list.Add(new\_including\_child);

}

public Engine.EngineObject? GetChildrenObject(string object\_name)

=> this.children\_list.Find((EngineObject target\_object) => target\_object.ObjectName == object\_name);

public List<EngineObject> GetChildrenObjects<TObject>() where TObject : EngineObject

=> this.children\_list.FindAll((EngineObject target\_object) => (target\_object as TObject) != null);

public void ConnectLinkToScene(Engine.IEngineScene scene\_instance\_link)

{

if (this.LinkedScene is not null) throw new Exception("Scene link already set");

this.LinkedScene = scene\_instance\_link;

}

private object locker\_object = new();

public void SetPosition(System.Drawing.Point direction)

{

lock(locker\_object)

{

this.Position = new Point(Position.X + direction.X, Position.Y - direction.Y);

children\_list?.ForEach((Engine.EngineObject obj) => obj.SetPosition(direction));

}

}

public void SetPosition(int pos\_x, int pos\_y)

{

lock (locker\_object)

{

this.Position = new Point(pos\_x, pos\_y);

children\_list?.ForEach((Engine.EngineObject obj) => obj.SetPosition(pos\_x, pos\_y));

}

}

public abstract void PaintingOperation(System.Drawing.Graphics graphic);

public virtual void InitialOperation(Engine.IEngineScene scene\_instance) { return; }

public virtual void UpdateOperation(Engine.IEngineScene scene\_instance) { return; }

public override bool Equals(object? target\_object)

{

if(target\_object is EngineObject typed) return this.ObjectName == typed.ObjectName;

throw new Exception("Object equals: different type for argument");

}

public override int GetHashCode() => ObjectGuid.GetHashCode();

}

public abstract class EngineInputController : EngineObject

{

protected EngineInputController(string object\_name) : base(object\_name) { }

public virtual void MouseMoveOperation(Win::Forms.MouseEventArgs mouse\_arg) { return; }

public virtual void MouseInputOperation(Win::Forms.MouseEventArgs mouse\_arg) { return; }

public virtual void MouseReleaseOperation(Win::Forms.MouseEventArgs mouse\_arg) { return; }

public virtual void KeyInputOperation(Win::Forms.KeyEventArgs key\_arg) { return; }

public virtual void KeyReleaseOperation(Win::Forms.KeyEventArgs key\_arg) { return; }

}

}

**EngineScene.cs**

using System;

using System.Collections.Generic;

using Win = System.Windows;

namespace PracticeWork.Engine

{

public record EngineSceneConfiguration(string ConfigKey, Type ConfigType, object ConfigValue);

public record EngineObjectWithIncludes(Engine.EngineObject Object, List<string> IncludeChildren);

public interface IEngineScene

{

List<EngineObject> GetSceneObjects<TObject>() where TObject : EngineObject;

EngineObject? GetSceneObject(string required\_object\_name);

EngineSceneConfiguration? GetSceneConfiguration(string object\_name, string config\_name);

void ExitSceneHandler();

void RunSceneHandler(double update\_delay, System.Action on\_exit\_operation);

System.Drawing.Size SceneSize { get; }

}

public sealed class EngineSceneBuilder : System.Object

{

private readonly List<EngineObjectWithIncludes> scene\_initial\_children = new(0);

private readonly Dictionary<string, List<EngineSceneConfiguration>> scene\_configuration = new(0);

public Win::Forms.Panel ScenePanel { get; init; }

public string SceneName { get; init; }

public EngineSceneBuilder(string scene\_name, Win::Forms.Panel scene\_panel)

=> (this.SceneName, this.ScenePanel) = (scene\_name, scene\_panel);

public EngineSceneBuilder RegisterSceneConfiguration<TConfig>(string object\_name, string key, TConfig value)

{

if (this.scene\_configuration.ContainsKey(object\_name) == true)

{

if (this.scene\_configuration[object\_name].Find((target) => target.ConfigKey == key) != null)

throw new Exception("Configuration allready set: " + object\_name + "; " + key);

this.scene\_configuration[object\_name].Add(new EngineSceneConfiguration(key, typeof(TConfig), value!));

}

else this.scene\_configuration.Add(object\_name, new() { new(key, typeof(TConfig), value!) });

return this;

}

public EngineSceneBuilder RegisterSceneChild<TObject>(string name, List<string> children) where TObject: EngineObject

{

System.Reflection.ConstructorInfo? selected\_constructor = null;

foreach (var contructor in typeof(TObject).GetConstructors())

{

var constructor\_attributes = contructor.GetCustomAttributes(typeof(Engine.EngineObjectConstructorSelecter), true);

if (constructor\_attributes.Length != 0) selected\_constructor = contructor;

}

if (selected\_constructor == null) throw new Exception("Not found ctor for engine object");

TObject? @object = selected\_constructor.Invoke(new object[] { name }) as TObject;

if (this.scene\_initial\_children.FindAll((EngineObjectWithIncludes target)

=> target.Object.Equals(@object)).Count > 0) throw new Exception("Allready registred");

this.scene\_initial\_children.Add(new(@object!, children));

return this;

}

private void SetSceneObjectConfig(EngineObjectWithIncludes target\_object)

{

var object\_config\_list = Enumerable.Where(this.scene\_configuration, (target) => target.Key

== target\_object.Object.ObjectName).ToList();

if (object\_config\_list.Count == 0) return;

foreach (System.Reflection.PropertyInfo infos in target\_object.Object.GetType().GetProperties())

{

var property\_attr = infos.GetCustomAttributes(typeof(Engine.EngineObjectImportConfiguration), true);

if (property\_attr.Length == 0) continue;

var property\_name = (property\_attr[0] as EngineObjectImportConfiguration)!.PropertyName;

var property\_value = object\_config\_list[0].Value.Find(config => config.ConfigKey == property\_name);

if(property\_value != null)

{

try { infos.SetValue(target\_object.Object, property\_value.ConfigValue); }

catch (System.Exception error) { Console.WriteLine(error.Message); }

}

}

}

public IEngineScene BuildScene()

{

Engine.IEngineScene scene\_instance = new EngineScene(ScenePanel, scene\_initial\_children.Select(

(item) => item.Object).ToList(), scene\_configuration) { SceneName = this.SceneName };

this.scene\_initial\_children.ForEach(delegate(EngineObjectWithIncludes target\_object)

{

this.SetSceneObjectConfig(target\_object);

target\_object.IncludeChildren.ForEach(delegate(string include\_name)

{

Engine.EngineObject? required\_object = scene\_instance.GetSceneObject(include\_name);

if (required\_object != null) target\_object.Object.ConnectLinkToChildren(required\_object);

});

target\_object.Object.ConnectLinkToScene(scene\_instance);

});

return scene\_instance;

}

}

public delegate void UpdateOperationHandling(Engine.IEngineScene scene\_link);

public sealed class EngineScene : IEngineScene

{

private readonly List<Engine.EngineObject> registred\_scene\_children;

private readonly Dictionary<string, List<EngineSceneConfiguration>> registred\_scene\_configuration;

private readonly Win::Forms.Panel panel\_node\_instance;

private event UpdateOperationHandling? UpdateHandling = default;

public System.String SceneName { get; init; } = "Scene";

public System.Boolean IsExit { get; private set; } = false;

public Size SceneSize { get => this.panel\_node\_instance.Size; }

public EngineScene(Win::Forms.Panel panel, List<EngineObject> scene\_children,

Dictionary<string, List<EngineSceneConfiguration>> scene\_configuration)

{

(this.registred\_scene\_children, this.registred\_scene\_configuration, this.panel\_node\_instance)

= (scene\_children, scene\_configuration, panel);

this.InitializeSceneHandling();

}

private object painter\_locker = new();

private void InitializeSceneHandling()

{

this.registred\_scene\_children.Where((EngineObject target\_object) => target\_object is EngineInputController)

.ToList().ForEach(delegate (EngineObject selected\_object)

{

EngineInputController? controller = (selected\_object as EngineInputController);

(this.panel\_node\_instance as Control).KeyDown += (sender, arg) => controller?.KeyInputOperation(arg);

(this.panel\_node\_instance as Control).KeyUp += (sender, arg) => controller?.KeyReleaseOperation(arg);

(this.panel\_node\_instance as Control).MouseMove += (sender, arg) => controller?.MouseMoveOperation(arg);

(this.panel\_node\_instance as Control).MouseDown += (sender, arg) => controller?.MouseInputOperation(arg);

(this.panel\_node\_instance as Control).MouseUp += (sender, arg) => controller?.MouseReleaseOperation(arg);

});

//this.registred\_scene\_children.ForEach((EngineObject target\_object)

// => (this.panel\_node\_instance.Paint) += (sender, arg) => { target\_object?.UpdateOperation(arg.Graphics); });

this.panel\_node\_instance.Paint += delegate (object? sender, PaintEventArgs arg)

{

this.registred\_scene\_children.ForEach((EngineObject target\_object) =>

{

lock(this.painter\_locker) { target\_object?.PaintingOperation(arg.Graphics); }

});

};

this.registred\_scene\_children.ForEach((target\_object) => this.UpdateHandling += target\_object.UpdateOperation);

}

public EngineObject? GetSceneObject(string required\_object\_name) => this.registred\_scene\_children.Find(

(Engine.EngineObject target) => target.ObjectName == required\_object\_name);

public List<EngineObject> GetSceneObjects<TObject>() where TObject : Engine.EngineObject

=> this.registred\_scene\_children.FindAll((target) => (target as TObject) != null);

public EngineSceneConfiguration? GetSceneConfiguration(string object\_name, string config\_name)

{

if (this.registred\_scene\_configuration.TryGetValue(object\_name, out var config\_list))

{

return config\_list.Find((EngineSceneConfiguration target\_config) => target\_config.ConfigKey == config\_name);

}

else throw new Exception("Cannot find required object");

}

private object timer\_locker = new();

public void RunSceneHandler(double update\_delay, Action on\_exit\_operation)

{

Win::Forms.Timer scene\_handling\_timer = new() { Interval = (int)update\_delay };

scene\_handling\_timer.Tick += new EventHandler(delegate (object? sender, EventArgs args)

{

if(this.IsExit) { scene\_handling\_timer.Stop(); on\_exit\_operation(); return; }

lock (this.timer\_locker)

{

if(this.UpdateHandling?.GetInvocationList().Length != 0) this.UpdateHandling!(this);

this.panel\_node\_instance.Focus();

this.panel\_node\_instance.Invalidate();

}

System.GC.Collect();

System.GC.WaitForPendingFinalizers();

});

this.registred\_scene\_children.ForEach((target\_record) => target\_record.InitialOperation(this));

scene\_handling\_timer.Start();

}

public void ExitSceneHandler() => this.IsExit = true;

}

}

**ChestSupport.cs**

using PracticeWork.Engine;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Objects

{

internal sealed class ChestSupport : Engine.EngineObject

{

private Objects.ChestTrigger? chest\_trigger = default;

private Engine.EngineAnimator? chest\_animator = default;

private Objects.Player? player\_instance = default;

private System.Int32 timer\_value = default;

private System.Boolean chest\_is\_open = default;

[Engine.EngineObjectImportConfiguration("ChestSpawnTime")]

public System.Int32 ChestSpawnTime { get; private set; } = 5;

[Engine.EngineObjectImportConfiguration("HealingValue")]

public System.Int32 HealingValue { get; private set; } = 3;

[Engine.EngineObjectImportConfiguration("SpawnMinPosition")]

public System.Drawing.Point SpawnMinPosition { get; private set; } = new();

[Engine.EngineObjectImportConfiguration("SpawnMaxPosition")]

public System.Drawing.Point SpawnMaxPosition { get; private set; } = new();

[Engine.EngineObjectConstructorSelecter]

public ChestSupport(string object\_name) : base(object\_name) { }

public override void InitialOperation(IEngineScene scene\_instance)

{

System.Windows.Forms.Timer chest\_timer = new() { Interval = 1000 };

chest\_timer.Tick += delegate (object? sender, EventArgs e)

{

if (++timer\_value >= ChestSpawnTime) { ChestSpawn(); timer\_value = 0; }

};

chest\_timer.Start();

this.chest\_trigger = (ChestTrigger?)(this.GetChildrenObjects<ChestTrigger>()[0]);

this.chest\_animator = (EngineAnimator?)(this.GetChildrenObjects<EngineAnimator>()[0]);

this.player\_instance = (Player?)(this.LinkedScene?.GetSceneObject("player"));

}

public void ChestSpawn()

{

if (!this.chest\_is\_open)

{

var random\_position = new Random();

this.SetPosition(random\_position.Next(this.SpawnMinPosition.X, this.SpawnMaxPosition.X),

random\_position.Next(this.SpawnMinPosition.Y, this.SpawnMaxPosition.Y));

this.chest\_trigger!.IsReady = true;

}

}

public void ChestOpen()

{

this.player\_instance?.HealingAction(this.HealingValue);

this.chest\_is\_open = true;

this.chest\_animator?.PlayAnimation("open\_animation", false);

}

public override void UpdateOperation(IEngineScene scene\_instance)

{

if (this.chest\_animator!.AnimationName != "open\_animation" && this.chest\_is\_open == true)

{

this.chest\_animator?.PlayAnimation("idle\_animation");

this.SetPosition(0, 0);

this.chest\_is\_open = false;

}

}

public override void PaintingOperation(Graphics graphic) { return; }

}

internal sealed class ChestTrigger : Engine.EngineBoxTrigerCollision

{

private Objects.ChestSupport? chest\_instance = default;

public System.Boolean IsReady { get; set; } = default;

[Engine.EngineObjectConstructorSelecter]

public ChestTrigger(string object\_name) : base(object\_name) { }

public override void InitialOperation(IEngineScene scene\_instance)

{

base.InitialOperation(scene\_instance);

this.chest\_instance = this.ParentObject as Objects.ChestSupport;

}

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

if (this.IsReady)

{

this.chest\_instance?.ChestOpen();

this.IsReady = false;

}

}

}

}

**Enemy.cs**

using PracticeWork.Engine;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Win = System.Windows;

namespace PracticeWork.Objects

{

public sealed class Enemy : Engine.EngineObject

{

public const int LifeValue = 3;

public const double SpeedUpValue = 0.5;

private int current\_life\_value = Enemy.LifeValue;

public enum LookDirection : System.Int16 { Right = 1, Left = -1 };

public System.Boolean IsAlive { get; set; } = true;

public System.Boolean IsReadyToRefresh { get; set; } = false;

private Engine.EngineAnimator? enemy\_animator = null;

private Engine.EngineInputController? player\_instance = null;

private Objects.EnemyHitRegistrator? hit\_registrator = default;

private System.Drawing.Point target\_position = new();

private System.Double target\_update\_speed = 1.0;

private Enemy.LookDirection look\_direction = default;

private System.Double movement\_speed = 2.0;

[Engine.EngineObjectImportConfiguration("TargetUpdateSpeed")]

public System.Double TargetUpdateSpeed

{

private set { if (value <= 1.0 && value > 0) this.target\_update\_speed = value; }

get { return this.target\_update\_speed; }

}

[Engine.EngineObjectImportConfiguration("MovementSpeed")]

public System.Double MaxMovementSpeed { get; private set; } = default;

[Engine.EngineObjectConstructorSelecter]

public Enemy(string object\_name) : base(object\_name) => this.look\_direction = LookDirection.Right;

public void Refresh()

{

this.IsReadyToRefresh = !(this.IsAlive = true);

this.current\_life\_value = Enemy.LifeValue;

if (this.movement\_speed + Enemy.SpeedUpValue <= this.MaxMovementSpeed) movement\_speed += SpeedUpValue;

}

public void EnemyHitPlayer()

{

if(this.IsAlive) this.enemy\_animator?.PlayAnimation("attack\_animation", false);

}

public void DamageRegistration()

{

if(--this.current\_life\_value <= 0)

{

this.enemy\_animator?.PlayAnimation("death\_animation", false);

}

else this.enemy\_animator?.PlayAnimation("damage\_animation", false);

}

public override void InitialOperation(IEngineScene scene\_instance)

{

try {

this.player\_instance = (EngineInputController?)(this.LinkedScene?.GetSceneObject("player")); ;

this.enemy\_animator = (EngineAnimator?)(this.GetChildrenObjects<EngineAnimator>()[0]);

this.hit\_registrator = (EnemyHitRegistrator)(this.GetChildrenObjects<Objects.DamageHolder>()[1]

.GetChildrenObjects<EnemyHitRegistrator>()[0]);

this.hit\_registrator.HitRegistration();

}

catch(System.Exception error) { Console.WriteLine(error.Message); }

this.enemy\_animator?.PlayAnimation("run\_animation");

Win::Forms.Timer enemy\_handling\_timer = new() { Interval = (int)(60 / target\_update\_speed )};

enemy\_handling\_timer.Tick += new EventHandler(delegate (object? sender, EventArgs args)

{

if(this.player\_instance != null) this.target\_position = this.player\_instance.Position;

});

enemy\_handling\_timer.Start();

}

public override void UpdateOperation(IEngineScene scene\_instance)

{

if (this.player\_instance == null) return;

if (this.enemy\_animator?.AnimationName != "attack\_animation" && this.hit\_registrator?.HitInstalled != true)

{

this.hit\_registrator.HitRegistration();

}

//if (this.enemy\_animator?.AnimationName != "damage\_animation" && this.enemy\_animator?.AnimationName != null &&

// this.enemy\_animator?.AnimationName != "run\_animation" && this.enemy\_animator?.AnimationName != "death\_animation"

// && this.enemy\_animator.AnimationName != "attack\_animation" && IsAlive)

// this.enemy\_animator?.PlayAnimation("run\_animation");

if (this.enemy\_animator?.AnimationName == "" && IsAlive) this.enemy\_animator?.PlayAnimation("run\_animation");

//if (this.current\_life\_value <= 0 && this.IsAlive)

//{

// this.enemy\_animator?.PlayAnimation("death\_animation", false);

// this.IsAlive = false;

//}

if (this.current\_life\_value <= 0 && this.enemy\_animator!.AnimationName != "death\_animation") this.IsAlive = false;

if(this.IsAlive == false && this.enemy\_animator!.AnimationName != "death\_animation") this.IsReadyToRefresh = true;

int target\_x = this.target\_position.X, target\_y = this.target\_position.Y;

double angle = Math.Atan((double)(target\_y - Position.Y) / (target\_x - Position.X));

if ((target\_x - Position.X < 0 && this.look\_direction == LookDirection.Right) ||

(target\_x - Position.X > 0 && this.look\_direction == LookDirection.Left))

{

this.enemy\_animator?.FlipAllAnimationFrame(ImageExtension.FlipImageDirection.FlipX);

this.look\_direction = (LookDirection)(-(int)look\_direction);

}

int direction\_x = (int)(movement\_speed \* Math.Cos(angle)) \* (int)this.look\_direction,

direction\_y = (int)(movement\_speed \* Math.Sin(angle)) \* (int)this.look\_direction;

if(this.current\_life\_value > 0) this.SetPosition(new Point(direction\_x, -direction\_y));

}

public override void PaintingOperation(Graphics graphic)

{

graphic.DrawString("Life: " + this.current\_life\_value.ToString(), new Font("Arial", 10),

Brushes.White, this.Position.X, this.Position.Y);

}

}

public sealed class EnemyManager : Engine.EngineObject

{

public const System.Int32 SceneMaxEnemy = 10;

public class EnemyConfiguration

{

public Objects.Enemy EnemyInstance { get; set; }

public System.Drawing.Point EnemyStartPosition { get; set; }

}

private List<EnemyManager.EnemyConfiguration> enemies = new();

private int current\_level = 1;

[Engine.EngineObjectImportConfiguration("EnemyCount")]

public int EnemyCount { get; private set; } = 0;

public int EnemyEliminated { get; private set; } = 0;

[Engine.EngineObjectConstructorSelecter]

public EnemyManager(string object\_name) : base(object\_name) { }

public override void InitialOperation(IEngineScene scene\_instance)

{

if (EnemyCount > EnemyManager.SceneMaxEnemy) throw new Exception("So many enemies :)");

for (int i = 0; i < this.EnemyCount; i++)

{

var selected\_enemy = (Objects.Enemy?)this.GetChildrenObject("enemy" + i);

if (selected\_enemy != null)

{

this.enemies?.Add(new() { EnemyInstance = selected\_enemy, EnemyStartPosition = new() });

}

}

bool trigger\_break = false;

for(int row = 0; row < EnemyManager.SceneMaxEnemy / 2; row++)

{

for(int side = 0; side <= 1; side++)

{

if (row \* 2 + side >= this.enemies?.Count) { trigger\_break = true; break; }

(this.enemies?[row \* 2 + side])!.EnemyStartPosition = new(1024 \* side, 150 \* row);

(this.enemies?[row \* 2 + side])!.EnemyInstance.SetPosition(1024 \* side, 150 \* row);

}

if (trigger\_break) break;

}

}

public override void UpdateOperation(IEngineScene scene\_instance)

{

foreach (var item in this.enemies)

{

Objects.Enemy enemy = item.EnemyInstance;

if (!enemy.IsAlive)

{

enemy.SetPosition(item.EnemyStartPosition.X, item.EnemyStartPosition.Y);

enemy.Refresh();

this.EnemyEliminated++;

}

}

}

public override void PaintingOperation(Graphics graphic) { return; }

}

public sealed class LevelWallContainer : Engine.EngineObject

{

[Engine.EngineObjectConstructorSelecter]

public LevelWallContainer(string object\_name) : base(object\_name) { }

public override void PaintingOperation(Graphics graphic) { return; }

}

}

**HitRegistrator.cs**

using PracticeWork.Engine;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Objects

{

class DamageHolder : Engine.EngineObject

{

[Engine.EngineObjectConstructorSelecter]

public DamageHolder(string object\_name) : base(object\_name) => this.Geometry = new(50, 50);

public override void PaintingOperation(Graphics graphic) { return; }

}

internal sealed class PlayerHitRegistrator : Engine.EngineBoxTrigerCollision

{

public System.Boolean HitInstalled { get; private set; } = default;

[Engine.EngineObjectConstructorSelecter]

public PlayerHitRegistrator(string object\_name) : base(object\_name) { }

public void HitRegistration(bool value = true) => this.HitInstalled = value;

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

if(target\_object is EnemyDamageRegistrator registrator && this.HitInstalled)

{

registrator.DamageRegistration();

this.HitInstalled = false;

}

}

}

internal sealed class EnemyDamageRegistrator : Engine.EngineBoxTrigerCollision

{

[Engine.EngineObjectConstructorSelecter]

public EnemyDamageRegistrator(string object\_name) : base(object\_name) { }

public void DamageRegistration()

{

(this.ParentObject?.ParentObject as Objects.Enemy)?.DamageRegistration();

}

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

return;

}

}

internal sealed class EnemyHitRegistrator : Engine.EngineBoxTrigerCollision

{

//private System.Boolean hit\_installed = default;

public System.Boolean HitInstalled { get; private set; } = default;

private Objects.Enemy? enemy\_instance\_link = null;

[Engine.EngineObjectConstructorSelecter]

public EnemyHitRegistrator(string object\_name) : base(object\_name) { }

public override void InitialOperation(IEngineScene scene\_instance)

{

this.enemy\_instance\_link = (this.ParentObject?.ParentObject as Enemy);

}

public void HitRegistration(bool value = true) => this.HitInstalled = value;

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

if (target\_object is PlayerDamageRegistrator registrator && this.HitInstalled && this.enemy\_instance\_link!.IsAlive)

{

registrator.DamageRegistration();

this.enemy\_instance\_link?.EnemyHitPlayer();

this.HitInstalled = false;

}

}

}

internal sealed class PlayerDamageRegistrator : Engine.EngineBoxTrigerCollision

{

[Engine.EngineObjectConstructorSelecter]

public PlayerDamageRegistrator(string object\_name) : base(object\_name) { }

public void DamageRegistration() => (this.ParentObject?.ParentObject as Objects.Player)?.DamageRegistration();

public override void InitialOperation(IEngineScene scene\_instance)

{

Console.WriteLine(this.Position.X + ";\t" + this.Position.Y);

}

protected override void OnTriggerDetectCollision(EngineObject target\_object, CollideSide side)

{

}

}

}

**Player.cs**

using PracticeWork.Engine;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace PracticeWork.Objects

{

internal sealed class Player : PracticeWork.Engine.EngineInputController

{

public enum MoveDirection : System.Int16 { Positive = 1, Negative = -1, Idle = 0 };

private System.Drawing.Point cursor\_position = new Point(0, 0);

private Player.MoveDirection move\_direction\_x = default, move\_direction\_y = default,

look\_direction = Player.MoveDirection.Positive;

private Engine.EngineAnimator? player\_animator = null;

private Engine.EngineBoxStaticCollision? player\_collision = null;

private Objects.PlayerHitRegistrator? hit\_registrator = default;

[Engine.EngineObjectImportConfiguration("LifeCount")]

public System.Int32 LifeCount { get; private set; } = 3;

public System.Boolean IsAlive { get; private set; } = true;

public System.Boolean IsReadyToAttack { get; set; } = true;

[Engine.EngineObjectImportConfiguration("MovementSpeed")]

public System.Double MovementSpeed { get; protected set; } = default;

[Engine.EngineObjectConstructorSelecter]

public Player(string object\_name) : base(object\_name)

=> this.move\_direction\_x = this.move\_direction\_y = Player.MoveDirection.Idle;

public override void InitialOperation(IEngineScene scene\_instance)

{

this.player\_collision = (EngineBoxStaticCollision)(this.LinkedScene?.GetSceneObject("player\_colission"))!;

this.player\_animator = (EngineAnimator)(this.LinkedScene?.GetSceneObject("player\_animator"))!;

this.hit\_registrator = (PlayerHitRegistrator)(this.LinkedScene?.GetSceneObject("player\_hit")!);

}

public void HealingAction(int heal\_value) => this.LifeCount += heal\_value;

public void DamageRegistration()

{

if (this.IsAlive)

{

if (--this.LifeCount <= 0)

{

this.player\_animator?.PlayAnimation("death\_animation", false);

this.IsAlive = false;

}

else this.player\_animator?.PlayAnimation("damage\_animation", false);

}

}

public override void UpdateOperation(IEngineScene scene\_instance)

{

double current\_speed = this.MovementSpeed;

if ((int)look\_direction > 0 && this.move\_direction\_x < 0 || (int)look\_direction < 0 && this.move\_direction\_x > 0)

current\_speed /= 1.5;

if(this.IsAlive) this.SetPosition(new Point((int)(current\_speed \* (int)this.move\_direction\_x),

(int)(current\_speed \* (int)this.move\_direction\_y)));

if (!this.IsAlive && this.player\_animator?.AnimationName != "death\_animation")

{

this.LinkedScene?.ExitSceneHandler();

}

if (!Regex.IsMatch(this.player\_animator?.AnimationName!, "attack\_animation[1-3]{1}") )

{

if(hit\_registrator.HitInstalled) this.hit\_registrator.HitRegistration(false);

this.IsReadyToAttack = true;

}

if (!Regex.IsMatch(this.player\_animator?.AnimationName!, "attack\_animation[1-3]{1}") &&

this.player\_animator?.AnimationName != "damage\_animation" && this.IsAlive)

{

if (Math.Abs((int)this.move\_direction\_x) + Math.Abs((int)this.move\_direction\_y) > 0)

{

if (this.player\_animator!.AnimationName != "run\_animation")

this.player\_animator?.PlayAnimation("run\_animation");

}

else

{

if (this.player\_animator!.AnimationName != "idle\_animation")

this.player\_animator?.PlayAnimation("idle\_animation");

}

}

if ((look\_direction == MoveDirection.Positive && this.cursor\_position.X < this.Position.X) ||

(look\_direction == MoveDirection.Negative && this.cursor\_position.X > this.Position.X))

{

this.player\_animator?.FlipAllAnimationFrame(ImageExtension.FlipImageDirection.FlipX);

this.look\_direction = (MoveDirection)(-(int)look\_direction);

}

}

public override void PaintingOperation(Graphics graphic) { }

public override void KeyInputOperation(KeyEventArgs key\_arg)

{

if (key\_arg.KeyCode == Keys.Up) this.move\_direction\_y = MoveDirection.Positive;

if (key\_arg.KeyCode == Keys.Down) this.move\_direction\_y = MoveDirection.Negative;

if (key\_arg.KeyCode == Keys.Right) this.move\_direction\_x = MoveDirection.Positive;

if (key\_arg.KeyCode == Keys.Left) this.move\_direction\_x = MoveDirection.Negative;

}

public override void KeyReleaseOperation(KeyEventArgs key\_arg)

{

if (key\_arg.KeyCode == Keys.Up || key\_arg.KeyCode == Keys.Down) move\_direction\_y = MoveDirection.Idle;

if (key\_arg.KeyCode == Keys.Left || key\_arg.KeyCode == Keys.Right) move\_direction\_x = MoveDirection.Idle;

}

public override void MouseMoveOperation(MouseEventArgs mouse\_arg)

{

this.cursor\_position = new Point(mouse\_arg.X, mouse\_arg.Y);

}

public override void MouseReleaseOperation(MouseEventArgs mouse\_arg)

{

if (!Regex.IsMatch(this.player\_animator?.AnimationName!, "attack\_animation[1-3]{1}") &&

this.player\_animator?.AnimationName != null)

{

this.player\_animator!.PlayAnimation("attack\_animation" + new Random().Next(1, 4), false);

this.IsReadyToAttack = false;

}

}

}

internal sealed class PlayerAttackCursor : PracticeWork.Engine.EngineInputController

{

private System.Drawing.Image look\_cursor\_image, attack\_cursor\_image;

private Objects.PlayerHitRegistrator? hit\_registrator = default;

private Objects.Player? player\_instance = default;

private System.Boolean is\_attacking = default;

private System.Drawing.Point cursor\_position = new Point(0, 0);

[Engine.EngineObjectImportConfiguration("MaxAttackRadius")]

public System.Double MaxAttackRadius { get; private set; } = 100.0;

[Engine.EngineObjectConstructorSelecter]

public PlayerAttackCursor(string object\_name) : base(object\_name)

{

this.look\_cursor\_image = Image.FromFile(@"..\..\..\Assets\Interface\look\_cursor.png");

this.attack\_cursor\_image = Image.FromFile(@"..\..\..\Assets\Interface\sword\_cursor.png");

}

public override void InitialOperation(IEngineScene scene\_instance)

{

this.hit\_registrator = (PlayerHitRegistrator?)this.GetChildrenObjects<PlayerHitRegistrator>()[0];

this.player\_instance = (Player?)this.LinkedScene?.GetSceneObject("player");

}

public override void MouseMoveOperation(MouseEventArgs mouse\_arg)

{

this.cursor\_position = new Point(mouse\_arg.X, mouse\_arg.Y);

}

public override void MouseInputOperation(MouseEventArgs mouse\_arg) => this.is\_attacking = true;

public override void MouseReleaseOperation(MouseEventArgs mouse\_arg)

{

if (this.player\_instance!.IsReadyToAttack) this.hit\_registrator?.HitRegistration();

this.is\_attacking = false;

}

private System.Drawing.Point center = new();

private System.Double angle = default, direction = default;

public override void UpdateOperation(IEngineScene scene\_instance)

{

double delta\_h = ((this.Position.Y + this.Geometry.Height / 2.0) - this.cursor\_position.Y),

delta\_w = (this.cursor\_position.X - (this.Position.X + this.Geometry.Width / 2.0));

double R = (Math.Sqrt(Math.Pow(delta\_w, 2) + Math.Pow(delta\_h, 2)));

R = (R < this.MaxAttackRadius) ? R : this.MaxAttackRadius;

this.angle = (delta\_w == 0) ? 1 : Math.Atan(delta\_h / delta\_w);

this.center = new((int)(this.Position.X + this.Geometry.Width / 2.0), (int)(this.Position.Y + this.Geometry.Height / 2.0));

this.direction = (this.cursor\_position.X - center.X > 0) ? 1 : -1;

this.hit\_registrator?.SetPosition(center.X + (int)(MaxAttackRadius \* Math.Cos(angle) \* direction),

center.Y + (int)(MaxAttackRadius \* Math.Sin(angle) \* -direction));

}

public override void PaintingOperation(Graphics graphic)

{

graphic.DrawImage(this.is\_attacking ? this.attack\_cursor\_image : this.look\_cursor\_image,

new Rectangle(new(this.cursor\_position.X + 15, this.cursor\_position.Y + 15), new(30, 30)));

if (this.is\_attacking == true)

{

graphic.DrawEllipse(Pens.White, new Rectangle(new(center.X - (int)(MaxAttackRadius),

center.Y - (int)(MaxAttackRadius)), new((int)(MaxAttackRadius \* 2), (int)(MaxAttackRadius \* 2))));

graphic.DrawLine(Pens.White, center, new(center.X + (int)(MaxAttackRadius \* Math.Cos(angle + 30 \* Math.PI / 180)

\* direction), center.Y + (int)(MaxAttackRadius \* Math.Sin(angle + 30 \* Math.PI / 180) \* -direction)));

graphic.DrawLine(Pens.White, center, new(center.X + (int)(MaxAttackRadius \* Math.Cos(angle - 30 \* Math.PI / 180)

\* direction), center.Y + (int)(MaxAttackRadius \* Math.Sin(angle - 30 \* Math.PI / 180) \* -direction)));

}

}

}

}

**UInterface.cs**

using PracticeWork.Engine;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Objects

{

internal sealed class UInterface : Engine.EngineObject

{

[Engine.EngineObjectImportConfiguration("LifeCountOffset")]

public int LifeCountOffset { get; private set; } = 900;

[Engine.EngineObjectImportConfiguration("EnemyEliminatedOffset")]

public int EnemyEliminatedOffset { get; private set; } = 50;

[Engine.EngineObjectImportConfiguration("UIFontSize")]

public int UIFontSize { get; private set; } = 16;

private Objects.EnemyManager? enemy\_manager = null;

private Objects.Player? player\_instance = null;

private System.Int32 session\_time = default;

[Engine.EngineObjectConstructorSelecter]

public UInterface(string object\_name) : base(object\_name) { }

public override void InitialOperation(IEngineScene scene\_instance)

{

this.enemy\_manager = (EnemyManager?)scene\_instance.GetSceneObject("enemy\_manager");

this.player\_instance = (Player?)scene\_instance.GetSceneObject("player");

System.Windows.Forms.Timer timer = new() { Interval = 1000 };

timer.Tick += (object? sender, EventArgs e) => this.session\_time++;

timer.Start();

}

public override void PaintingOperation(Graphics graphic)

{

graphic.DrawString("Enemies eliminated: " + this.enemy\_manager?.EnemyEliminated.ToString(),

new Font("Arial", this.UIFontSize), Brushes.White, this.EnemyEliminatedOffset, 10f);

graphic.DrawString("Time: " + this.session\_time.ToString(), new Font("Arial", this.UIFontSize),

Brushes.White, this.LinkedScene!.SceneSize.Width / 2, 10f);

graphic.DrawString("Life: " + (this.player\_instance?.LifeCount).ToString(),

new Font("Arial", this.UIFontSize), Brushes.White, this.LifeCountOffset, 10f);

}

}

}

**ChestConfiguration.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Configuration

{

internal static class ChestConfiguration

{

public static void AddChestSupport(this Engine.EngineSceneBuilder scene\_builder, int spawn\_time, bool debug)

{

scene\_builder

.RegisterSceneConfiguration<Point>("chest", "SpawnMinPosition", new Point(192, 192))

.RegisterSceneConfiguration<Point>("chest", "SpawnMaxPosition", new Point(832, 576))

.RegisterSceneConfiguration<Size>("chest", "ObjectGeometry", new Size(28, 28))

.RegisterSceneConfiguration<int>("chest", "ChestSpawnTime", spawn\_time)

.RegisterSceneConfiguration<List<Type>>("chest\_trigger", "TypesForCollide", new() { typeof(Objects.Player) })

.RegisterSceneConfiguration<Dictionary<string, string>>("chest\_animator", "AnimationsContainer", new()

{

{ "open\_animation", @"..\..\..\Assets\Chest\Open" },

{ "idle\_animation", @"..\..\..\Assets\Chest\Idle" },

})

.RegisterSceneConfiguration<double>("chest\_animator", "AnimationSpeed", 0.5)

.RegisterSceneConfiguration<bool>("chest\_trigger", "BorderDraw", debug);

scene\_builder

.RegisterSceneChild<Engine.EngineAnimator>("chest\_animator", new())

.RegisterSceneChild<Objects.ChestTrigger>("chest\_trigger", new())

.RegisterSceneChild<Objects.ChestSupport>("chest", new() { "chest\_animator", "chest\_trigger" });

}

}

}

**EnemyConfiguration.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using PracticeWork.Objects;

namespace PracticeWork.Configuration

{

internal static class EnemyConfiguration

{

public static void AddEnemy(this Engine.EngineSceneBuilder scene\_builder, bool debug, int number, double speed)

{

scene\_builder

.RegisterSceneConfiguration<double>("enemy" + number, "MovementSpeed", speed)

.RegisterSceneConfiguration<Size>("enemy" + number, "ObjectGeometry", new Size(50, 50))

.RegisterSceneConfiguration<double>("enemy" + number, "TargetUpdateSpeed", 0.1)

.RegisterSceneConfiguration<Size>("enemy\_collision" + number, "ObjectGeometry", new Size(50, 40))

.RegisterSceneConfiguration<Point>("enemy\_collision" + number, "ObjectPosition", new Point(0, 10))

.RegisterSceneConfiguration<bool>("enemy\_collision" + number, "BorderDraw", debug)

.RegisterSceneConfiguration<List<Type>>("enemy\_collision" + number, "TypesForCollide", new() { typeof(Player), typeof(Enemy) })

.RegisterSceneConfiguration<List<Type>>("enemy\_hit" + number, "TypesForCollide", new() { typeof(Objects.DamageHolder) })

.RegisterSceneConfiguration<Dictionary<string, string>>("enemy\_animator" + number, "AnimationsContainer", new()

{

{ "run\_animation", @"..\..\..\Assets\Enemy\Movement" },

{ "attack\_animation", @"..\..\..\Assets\Enemy\Attack" },

{ "idle\_animation", @"..\..\..\Assets\Enemy\Idle" },

{ "damage\_animation", @"..\..\..\Assets\Enemy\Damage" },

{ "death\_animation", @"..\..\..\Assets\Enemy\Death" },

})

.RegisterSceneConfiguration<double>("enemy\_animator" + number, "AnimationSpeed", 0.75);

scene\_builder

.RegisterSceneChild<Engine.EngineAnimator>("enemy\_animator" + number, new())

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("enemy\_collision" + number, new())

.RegisterSceneChild<Objects.EnemyDamageRegistrator>("enemy\_damage" + number, new())

.RegisterSceneChild<Objects.DamageHolder>("damage\_holder" + number, new() { "enemy\_damage" + number })

.RegisterSceneChild<Objects.EnemyHitRegistrator>("enemy\_hit" + number, new())

.RegisterSceneChild<Objects.DamageHolder>("hit\_holder" + number, new() { "enemy\_hit" + number })

.RegisterSceneChild<Objects.Enemy>("enemy" + number, new() { "enemy\_animator" + number, "enemy\_collision" + number,

"damage\_holder" + number, "hit\_holder" + number});

}

}

}

**PlayerConfiguration.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Configuration

{

internal static class PlayerConfiguration

{

public static void AddPlayer(this Engine.EngineSceneBuilder scene\_builder, bool debug, int life, double speed)

{

scene\_builder

.RegisterSceneConfiguration<bool>("player\_damage", "BorderDraw", debug)

.RegisterSceneConfiguration<bool>("player\_collision", "BorderDraw", debug)

.RegisterSceneConfiguration<bool>("player\_hit", "BorderDraw", debug)

.RegisterSceneConfiguration<double>("player", "MovementSpeed", speed)

.RegisterSceneConfiguration<Int32>("player", "LifeCount", life)

.RegisterSceneConfiguration<Size>("player", "ObjectGeometry", new Size(50, 50))

.RegisterSceneConfiguration<Point>("player", "ObjectPosition", new Point(512, 380))

.RegisterSceneConfiguration<Size>("player\_collision", "ObjectGeometry", new Size(40, 50))

.RegisterSceneConfiguration<Point>("player\_collision", "ObjectPosition", new Point(0, 5))

.RegisterSceneConfiguration<double>("player\_attack", "MaxAttackRadius", 80)

.RegisterSceneConfiguration<List<Type>>("player\_collision", "TypesForCollide", new()

{

typeof(Objects.Enemy), typeof(Objects.LevelWallContainer)

})

.RegisterSceneConfiguration<List<Type>>("player\_hit", "TypesForCollide", new() { typeof(Objects.DamageHolder) })

.RegisterSceneConfiguration<Point>("player\_damage", "ObjectPosition", new Point(-10, 0))

.RegisterSceneConfiguration<Size>("player\_damage", "ObjectGeometry", new Size(60, 60))

.RegisterSceneConfiguration<Dictionary<string, string>>("player\_animator", "AnimationsContainer", new()

{

{ "run\_animation", @"..\..\..\Assets\Player\Movement" },

{ "attack\_animation1", @"..\..\..\Assets\Player\Attack1" },

{ "attack\_animation2", @"..\..\..\Assets\Player\Attack2" },

{ "attack\_animation3", @"..\..\..\Assets\Player\Attack3" },

{ "idle\_animation", @"..\..\..\Assets\Player\Idle" },

{ "damage\_animation", @"..\..\..\Assets\Player\Damage" },

{ "death\_animation", @"..\..\..\Assets\Player\Death" },

})

.RegisterSceneConfiguration<double>("player\_animator", "AnimationSpeed", 0.9);

scene\_builder

.RegisterSceneChild<Engine.EngineAnimator>("player\_animator", new())

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("player\_collision", new())

.RegisterSceneChild<Objects.PlayerHitRegistrator>("player\_hit", new())

.RegisterSceneChild<Objects.PlayerAttackCursor>("player\_attack", new() { "player\_hit" })

.RegisterSceneChild<Objects.PlayerDamageRegistrator>("player\_damage", new())

.RegisterSceneChild<Objects.DamageHolder>("damage\_holder\_player", new() { "player\_damage" })

.RegisterSceneChild<Objects.Player>("player", new() { "player\_animator", "player\_collision",

"player\_attack", "damage\_holder\_player" });

}

}

}

**SceneConfiguration.cs**

using PracticeWork.Objects;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Configuration

{

internal static class SceneConfiguration

{

public static void AddSceneBorder(this Engine.EngineSceneBuilder scene\_builder, string sprite\_path)

{

scene\_builder

.RegisterSceneConfiguration("border\_sprite", "SpriteImagePath", sprite\_path)

.RegisterSceneConfiguration("border\_sprite", "ObjectGeometry", new Size(1024, 768))

.RegisterSceneConfiguration("border\_wall1", "ObjectGeometry", new Size(96, 1024))

.RegisterSceneConfiguration("border\_wall2", "ObjectGeometry", new Size(1024, 96))

.RegisterSceneConfiguration("border\_wall3", "ObjectGeometry", new Size(96, 1024))

.RegisterSceneConfiguration("border\_wall3", "ObjectPosition", new Point(928, 0))

.RegisterSceneConfiguration("border\_wall4", "ObjectGeometry", new Size(1024, 96))

.RegisterSceneConfiguration("border\_wall4", "ObjectPosition", new Point(0, 672));

scene\_builder

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("border\_wall1", new())

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("border\_wall2", new())

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("border\_wall3", new())

.RegisterSceneChild<Engine.EngineBoxStaticCollision>("border\_wall4", new())

.RegisterSceneChild<Engine.EngineSprite>("border\_sprite", new())

.RegisterSceneChild<LevelWallContainer>("border", new() {

"border\_sprite", "border\_wall1", "border\_wall2", "border\_wall3", "border\_wall4"

});

}

}

}

**UIConfiguration.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PracticeWork.Configuration

{

internal static class UIConfiguration

{

public static void AddUserInterface(this Engine.EngineSceneBuilder scene\_builder, int font\_size)

{

scene\_builder

.RegisterSceneConfiguration<string>("ui\_top", "SpriteImagePath", @"..\..\..\Assets\Scene\ui-container.png")

.RegisterSceneConfiguration<string>("ui\_bottom", "SpriteImagePath", @"..\..\..\Assets\Scene\ui-container.png")

.RegisterSceneConfiguration<Point>("ui\_bottom", "ObjectPosition", new Point(0, 690))

.RegisterSceneConfiguration<Size>("ui\_top", "ObjectGeometry", new Size(1024, 40))

.RegisterSceneConfiguration<Size>("ui\_bottom", "ObjectGeometry", new Size(1024, 40))

.RegisterSceneConfiguration<int>("interface", "UIFontSize", font\_size);

scene\_builder

.RegisterSceneChild<Engine.EngineSprite>("ui\_top", new())

.RegisterSceneChild<Engine.EngineSprite>("ui\_bottom", new())

.RegisterSceneChild<Objects.UInterface>("interface", new() { "ui\_top", "ui\_bottom" });

}

}

}