Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Рязанский государственный радиотехнический университет»

Кафедра вычислительной и прикладной математики

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту по дисциплине

«Конструирование программного обеспечения»

на тему

«Разработка 2D-платформера Alien Explorer»

Выполнил:

студент гр. 443 Тярт Н.А.

Проверил:

доц. Столчнев В.К.

Рязань 2017

Оглавление

[Введение 3](#_Toc499757939)

[1. Анализ задачи 4](#_Toc499757940)

[1.1 Разработка иерархии классов 4](#_Toc499757941)

[1.1.1 Выделение сущностей 4](#_Toc499757942)

[1.1.2 Зависимости между классами. Диаграмма классов 5](#_Toc499757943)

[1.2 Алгоритмы 9](#_Toc499757944)

[1.3 Разработка интерфейса программы 12](#_Toc499757945)

[2. Написание программы 13](#_Toc499757946)

[2.1 Разработка программы 13](#_Toc499757947)

[2.1.1 Описание классов, перечислений и интерфейсов проекта 14](#_Toc499757948)

[2.1.2 Описание разработанных процедур и функций 15](#_Toc499757949)

[2.2 Описание шаблонов проектирования, которые использовались при написании программы 22](#_Toc499757950)

[2.2.1 MVC (Model-View-Controller, Модель-Представление-Контроллер) 22](#_Toc499757951)

[2.2.2 Factory Method (Фабричный метод) 22](#_Toc499757952)

[2.2.3 Singleton (Синглтон, Одиночка) 22](#_Toc499757953)

[2.3 Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы 24](#_Toc499757954)

[2.3.1 Инкапсуляция поля 24](#_Toc499757955)

[2.3.2 Выделение метода 24](#_Toc499757956)

[2.3.3 Выделение класса 26](#_Toc499757957)

[2.3.4 Подъем метода 26](#_Toc499757958)

[2.3.5 Переименование метода 26](#_Toc499757959)

[2.4 Разработка тестов 28](#_Toc499757960)

[3. Результат работы программы 29](#_Toc499757961)

[Заключение 34](#_Toc499757962)

[Приложения 35](#_Toc499757963)

# Введение

Данный курсовой проект направлен в первую очередь на изучение и закрепление знаний по различным шаблонам проектирования, используемым в программировании. Для этого поставлена задача разработать игровую программу на языке *C#* для ОС *Microsoft Windows*. При этом структура приложения должна соответствовать шаблону *Model-View-Controller*, реализованному самостоятельно, а использование специализированных средств для создания игр не допускается.

Для реализации была выбрана идея 2D-платформера под кодовым названием «Alien Explorer». Игрок управляет персонажем, имеющим определенный запас очков жизни. Необходимо за как можно меньшее время пройти уровень, уворачиваясь от врагов и преодолевая препятствия. Лучшее время прохождения уровня заносится в таблицу рекордов.

Разработка велась для ОС *Windows 10* в среде *Visual Studio Community 2017* на языке *C# 6.0*. Для вывода изображения использованы встроенные графические средства языка и *Windows Forms*. При разработке использовалась система контроля версий *git*, проект и документация размещены в репозитории университета *GitLab*.

# Анализ задачи

## Разработка иерархии классов

Согласно требованиям, основным шаблоном проектирования, применяемым в данном проекте, является шаблон *Model-View-Controller (MVC)*. Части *View* и *Controller* будут максимально просты, а вот часть *Model*, являющаяся моделью предметной области, будет разделяться на уровни.

Верхним уровнем является объект всей модели. Он включает в себя несколько слоев: уровень (в физическом смысле), цель, враги, игрок, интерфейс. Эти слои накладываются друг на друга при отображении. Внутри каждого слоя может быть своя иерархия классов.

У главного объекта модели существует отдельный управляющий объект логики. Такие же логики существуют у каждого движущегося объекта модели, а именно – у врагов и персонажа игрока.

Также существует обособленный объект, связанный с файлом сохранения на диске. Он отвечает за загрузку данных из файла и актуализацию этих данных на диске.

Однотипные объекты (например, логики врагов) можно создавать фабричным способом в том или ином варианте. Файл сохранения логично сделать синглтонным.

### Выделение сущностей

Исходя из описанного в предыдущем параграфе, основных сущностей в программе будет три: модель (*Model*), представление (*View*) и контроллер (*Controller*).

Представление решено реализовывать на основе *Windows Forms* ввиду простоты применения. При этом пользовательская форма будет реализовывать определенный интерфейс, определяющий функциональность представления в шаблоне *MVC*.

Как следствие, необходим контроллер, ориентированный на использование *Windows Forms*. Он должен наследоваться от базового класса контроллеров, определяющего функциональность по работе с моделью, и специфицировать тонкости работы с формами. Например, он будет перехватывать события нажатия клавиш, принимаемые формой, и передавать их в модель в подготовленном виде.

Основой модели будет базовый объект, хранящий в себе всё текущее состояние модели. При этом стоит выделить два типа таких объектов – меню и игровой уровень. Впрочем, следует унифицировать работу с обоими типами моделей с точки зрения представления, что упростит реализацию однооконного приложения без видимых переходов при смене моделей. Поэтому классы меню и игрового уровня будут наследниками общего (абстрактного) класса модели.

Как уже говорилось ранее, объект модели будет включать в себя несколько основных слоев данных. В первую очередь, это уровень, враги, персонаж и интерфейс. Первые три слоя будут просто пустыми для модели меню, содержащей только интерфейс, что позволит унифицировать подход к отображению модели в представлении. Также не стоит забывать про логики врагов и персонажа, к которым у логики модели должен быть прямой доступ. Это необходимо для запуска и остановки уровня, а также для постановки уровня на паузу. Поэтому инициализацию логик объектов можно вынести в логику модели.

Для врагов было решено выделить четыре вида: пассивный неподвижный (пики), пассивный ходячий (слизень), пассивный летающий (летучая мышь) и активный летающий (призрак). Пассивные враги не реагируют на игрока и просто создают помехи на его пути. Активные враги (призраки) реагируют на попадание игрока в зону видимости и атакуют его.

При реализации различных логик для определения их состояний удобным будет использование механизма автоматов состояний, реализуемых отдельными классами. Также в разных местах программы понадобятся различные классы-утилиты, например, для загрузки уровней и меню, загрузке ресурсов с диска, хранения данных и ресурсов в памяти.

### Зависимости между классами. Диаграмма классов

В большинстве случаев связи классов будут реализованы в виде включения объектов одного класса в объекты другого класса в виде полей. Однако можно выделить иерархии наследования и реализации для контроллеров, представлений, объектов модели и их логик, автоматов состояний. Представленные ниже диаграммы сгенерированы в среде *Visual Studio*.

На рисунке 1.1.2.1 показана диаграмма классов контроллеров. В ней от базового абстрактного класса *GameController*, определяющего общий набор функций по работе с моделью и представлением, наследуется класс *WinFormController*, специализированный для работы с *Windows Forms*.

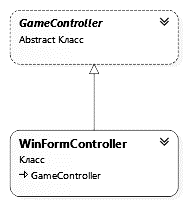


Рисунок 1.1.2.1. Диаграмма классов контроллеров.

На рисунке 1.1.2.2 показана иерархия классов части представления. Интерфейс *IViewable* определяет функциональность объекта представления, предоставляемую контроллеру. Его реализует класс формы *FormMain*. Для создания кадров он использует в свою очередь экземпляр класса *BufferedWinFormDrawer*, а этот класс оперирует графическими спрайтами, загружаемыми *ResourceLoader*-ом и хранимые в *SpritesContainer*-е.

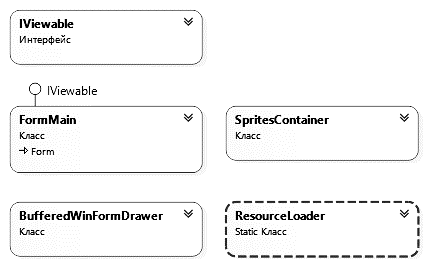


Рисунок 1.1.2.2. Диаграмма классов представления.

На рисунке 1.1.2.3 представлены основные классы модели. Это базовый класс *GameModel* и два инициализатора объекта этого класса – классы *LevelLoader* и *MenuLoader*. Они, соответственно, осуществляют загрузку игрового уровня и меню.



Рисунок 1.1.2.3. Диаграмма классов модели.

Иерархия объектов модели представлена на рисунке 1.1.2.4. От базового класса *GameObject* наследуются классы объектов уровня (*LevelObject*), врагов (*EnemyObject*), персонажа (*PlayerObject*) и интерфейса (*UIObject*).

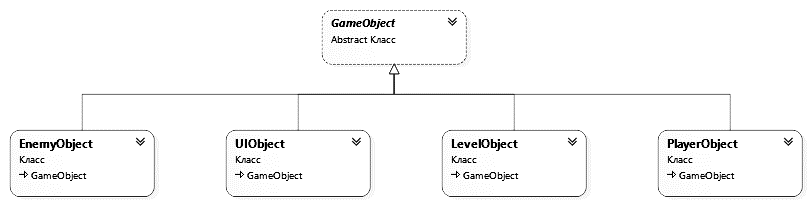


Рисунок 1.1.2.4. Диаграмма классов объектов модели.

Диаграмма классов логик объектов модели представлена на рисунке 1.1.2.5. Интерфейс логики *ILogic* реализуется абстрактным обобщённым классом *BaseObjectLogic*, от которого наследуются классы логик подвижных врагов и персонажа. Логики врагов инициализируются фабричным классом *EnemyLogicFactory*.

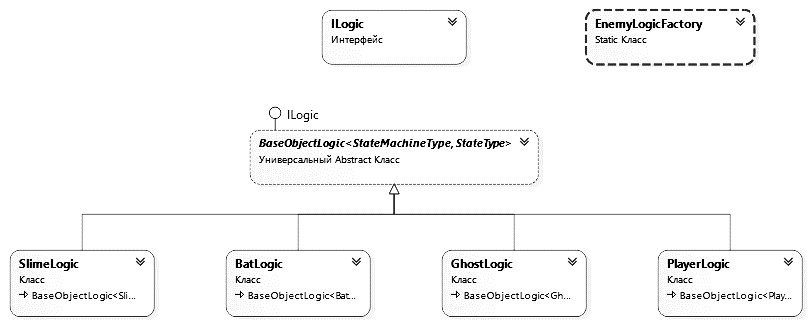


Рисунок 1.1.2.5. Диаграмма классов логик объектов модели.

С классами логик неразрывно связаны классы их автоматов состояний. Их иерархия представлена на рисунке 1.1.2.6 и полностью аналогична иерархии самих логик.

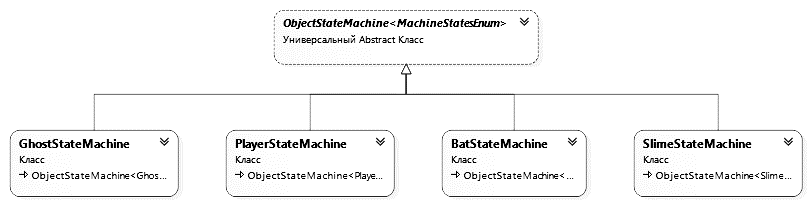


Рисунок 1.1.2.6. Диаграмма классов автоматов состояний объектов модели.

Схема логика – автомат состояний применяется также и в случае с основным объектом модели соответствующие иерархии классов с разделением на классы меню и игрового уровня представлены на рисунке 1.1.2.7.

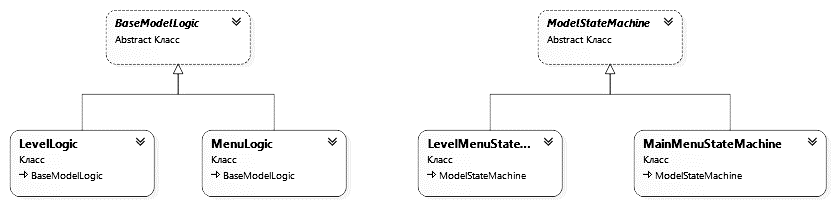


Рисунок 1.1.2.7. Диаграммы классов логик и автоматов состояний модели.

Помимо описанных выше классов, в программе также задействованы некоторые вспомогательные классы. Например, часто поведение при генерации чего-либо определяется содержимым соответствующего перечисления, а оно в свою очередь дополняется привязкой полю перечисления объекта класса *CustomAttribute*. Файлом сохранения оперирует объект класса *SaveFile*. Оставшиеся классы программы показаны на рисунках 1.1.2.8 – 1.1.2.10.

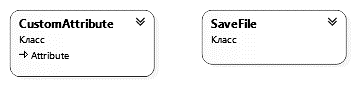


Рисунок 1.1.2.8. Диаграмма дополнительных классов модели.

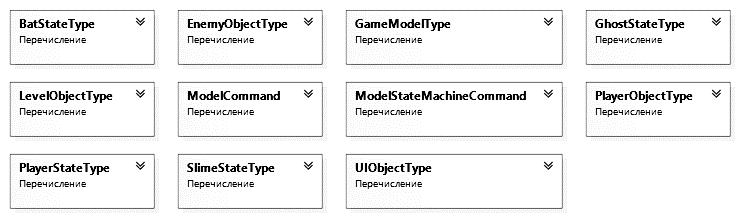


Рисунок 1.1.2.9. Диаграмма классов всех перечислений модели.



Рисунок 1.1.2.10. Диаграмма классов всех делегатов.

## Алгоритмы

В данном разделе рассмотрены ключевые алгоритмы работы приложения.

В первом приближении последовательность действий при запуске приложения можно продемонстрировать с помощью алгоритма, представленного на рисунке 1.2.1. При запуске программы инициализируется объект определенного класса контроллера. Этот объект в свою очередь инициализирует нужные ему представление и модель. После этого в различных потоках начинают выполняться две параллельные задачи – работа модели и отображение модели. При оправке приложению сигнала прекращения работы оба процесса останавливаются и приложение закрывается.

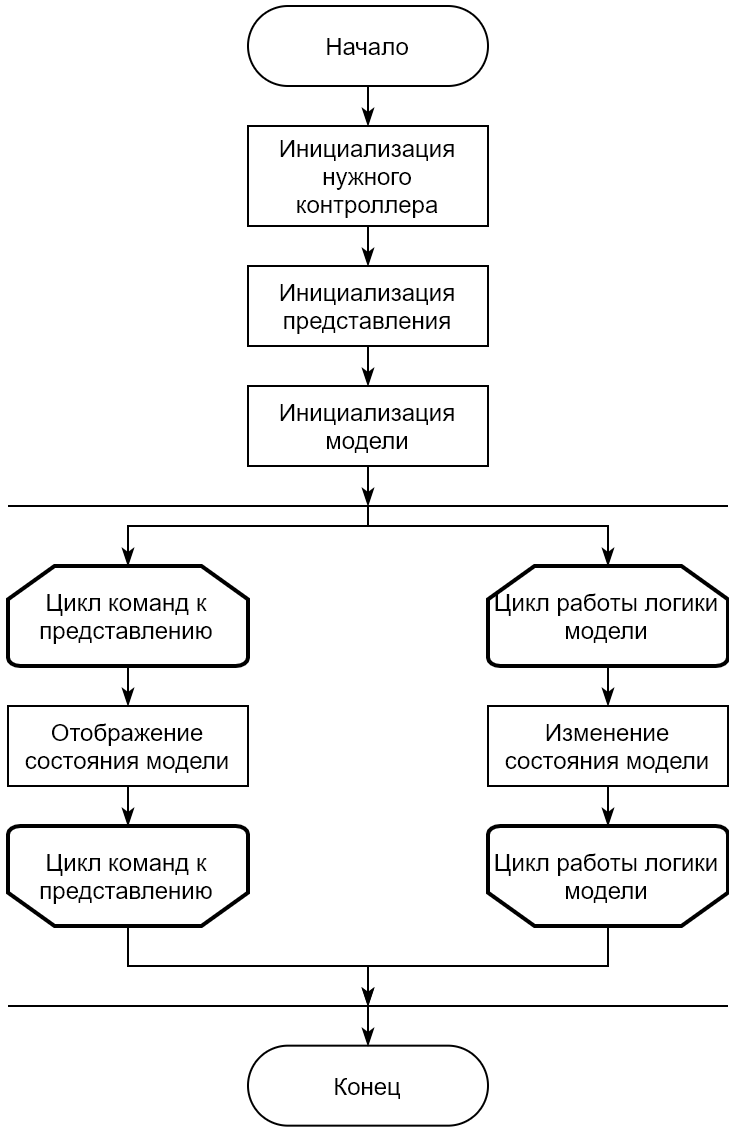


Рисунок 1.2.1. Блок-схема общего алгоритма работы приложения.

На рисунке 1.2.2 показан алгоритм механизма отображения модели (для ее графического представления). Из него видно, что сначала состояние модели последовательно (слой за слоем) выводится в графический буфер, а затем содержимое этого буфера за одно действие отображается на форму. Такой подход позволяет повысить производительность при графических операциях и избежать проблем при выводе изображения на экран.

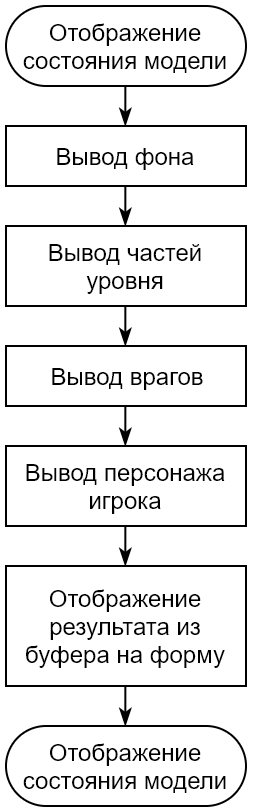


Рисунок 1.2.2. Блок-схема алгоритма отображения модели.

На рисунке 1.2.3 показан основной алгоритм работы логики модели уровня. Он соответствует отдельному методу класса логики модели уровня, вызываемому в отдельном потоке. Суть алгоритма такова: сначала происходит подготовка цикла вычислений. Затем запускается сам цикл, внутри которого постоянно производятся вычисления и обновляется состояние той части модели, за которую ответственна логика. В самом начале цикла стоит операция ожидания сигнального состояния от события синхронизации потоков. В качестве такого события в программе использован экземпляр стандартного класса *ManualResetEventSlim*. Цикл завершается естественным путем, когда в объекте логики изменяется значение флага завершения потока на истинное (в том числе извне).

Описанная схема работы логики уровня действительна также и для логик объектов (врагов и персонажа игрока).

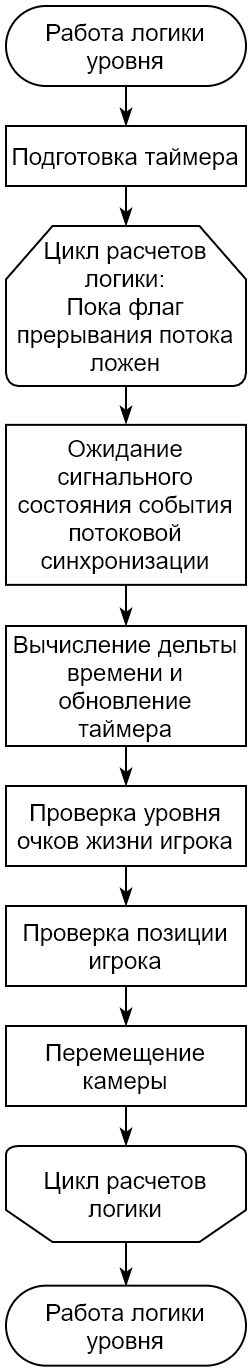


Рисунок 1.2.3. Блок-схема алгоритма работы логики модели уровня.

## Разработка интерфейса программы

Основным интерфейсом разрабатываемого приложения было решено стандартное текстовое меню в виде вертикального списка пунктов. Это достаточно типичное решение, позволяющее взаимодействовать с меню посредством минимального количества клавиш.

Главное меню игры состоит из следующих пунктов:

* начать новую игру;
* загрузить игру;
* выбрать уровень для загрузки;
* открыть таблицу рекордов;
* выйти из игры.

При выборе начала/загрузки игры или выхода пользователю должно выдаваться сообщение с запросом на подтверждение действия.

Интерфейс пользователя во время прохождения уровня должен быть минималистичен, отображая только нужную информацию: очки жизни и таймер прохождения уровня.

При постановке уровня на паузу или при его завершении пользователю отображается текстовое меню, аналогичное по своей организации главному меню программы. Ниже описаны пункты различных меню.

Меню паузы уровня:

* продолжить игру;
* начать уровень заново;
* вернуться в главное меню.

Меню выигрыша уровня:

* загрузить следующий уровень (при наличии);
* начать уровень заново;
* вернуться в главное меню.

Меню проигрыша уровня:

* начать уровень заново;
* вернуться в главное меню.

Из способов ввода решено оставить только клавиатуру. Компьютерная мышь не используется в игровой механике, а ее задействование в меню значительно усложнит механизм обмена командами между слоями *MVC*.

Приложение предоставляет следующую привязки клавиш к командам:

* *←, A* – влево;
* *→, D* – вправо;
* *↑, W, Space* – вверх;
* *↓, S, Control* – вниз;
* *Enter* – подтверждение;
* *Escape, Backspace* – отмена.

Данные ассоциации клавиш общеприняты и используются во многих других проектах.

# Написание программы

## Разработка программы

Процесс разработки программы можно разделить на следующие этапы:

1. Проработка идеи проекта.
2. Реализация шаблона *MVC*.
3. Проработка общей структуры модели.
4. Реализация контроллера и представления, ориентируясь на созданную структуру модели (включая графические спрайты и механизмы работы с ними).
5. Реализация главного меню игры (включая логику меню и работу с файлом сохранения).
6. Реализация логики персонажа (включая физическую модель).
7. Реализация логики врагов (включая физическую модель).
8. Организация взаимодействия персонажа и врагов.
9. Реализация игрового интерфейса (включая меню паузы уровня).
10. Создание контента игры (игровых уровней).
11. Окончательная доработка и отладка приложения.

### Описание классов, перечислений и интерфейсов проекта

|  |  |  |
| --- | --- | --- |
| Пространство имен | Класс/интерфейс/ перечисление | Описание |
| AlienExplorer.Controller | GameController | Абстрактный базовый класс контроллеров. |
| WinFormController | Контроллер для вида на Windows Forms. |
| AlienExplorer.Model | BaseModelLogic | Абстрактный базовый класс логики модели. |
| BaseObjectLogic`2 | Абстрактный базовый класс для логик объектов. |
| BatLogic | Логика летучей мыши. |
| BatStateMachine | Автомат состояний летучей мыши. |
| BatStateType | Тип состояния летучей мыши. |
| CustomAttribute | Пользовательский атрибут для полей в перечислениях. |
| dCloseApplication | Закрытие приложения. Вызывается при закрытии вида или из логики модели через делегат. |
| dLoadAnotherModel | Команда контроллеру загрузить другую модель. Вызывается логикой старой модели через делегат. |
| dSetCameraSize | Изменение размеров камеры модели. Вызывается видом через делегат. |
| EnemyLogicFactory | Фабрика логик врагов. |
| EnemyObject | Враг. |
| EnemyObjectType | Тип врага. |
| GameModel | Базовый контейнер данных модели. |
| GameModelType | Тип модели. |
| GameObject | Абстрактный базовый класс объектов модели. |
| GhostLogic | Логика призрака. |
| GhostStateMachine | Автомат состояний призрака. |
| GhostStateType | Тип состояния призрака. |
| ILogic | Общий интерфейс для логик объектов, определяет действия для управления их внутренними потоками вычислений. |
| LevelLoader | Загрузчик уровней. |
| LevelLogic | Логика игрового уровня. |
| LevelMenuStateMachine | Автомат состояний игрового уровня. |
| LevelObject | Элемент уровня. |
| LevelObjectType | Тип элемента уровня. |
| MainMenuStateMachine | Автомат состояний главного меню. |
| MenuLoader | Загрузчик главного меню игры. |
| MenuLogic | Логика меню. |
| ModelCommand | Тип команды от контроллера к модели. |
| ModelStateMachine | Абстрактный базовый класс автомата состояний модели. |
| ModelStateMachine Command | Тип команды от автомата состояний модели к логике модели. |
| ObjectStateMachine`1 | Абстрактный базовый класс автомата состояний объекта уровня. |
| PlayerLogic | Логика игрока. |
| PlayerObject | Объект игрока. |
| PlayerObjectType | Тип игрока. |
| PlayerStateMachine | Автомат состояний игрока. |
| PlayerStateType | Тип состояния игрока. |
| SaveFile | Класс по работе с файлом сохранения. |
| SlimeLogic | Логика слизня. |
| SlimeStateMachine | Автомат состояний слизня. |
| SlimeStateType | Тип состояния слизня. |
| UIObject | Элемент интерфейса. |
| UIObjectType | Тип объекта интерфейса. |
| AlienExplorer.View | BufferedWinFormDrawer | Буферизированный создатель кадров. |
| FormMain | Вид - форма Windows. |
| IViewable | Общий интерфейс для всех видов. |
| ResourceLoader | Загрузчик ресурсов. |
| SpritesContainer | Хранилище всех спрайтов игры. |

### Описание разработанных процедур и функций

|  |  |  |
| --- | --- | --- |
| Класс | Метод | Параметр |
| GameController | public GameController()  Произвоит загрузку главного меню игры. | – |
| protected void LoadLevel(int parLevelID)  Загрузка игрового уровня. | int parLevelID  ID уровня. |
| protected void LoadMenu()  Загрузка главного меню игры. | – |
| protected void SendModelToView()  Бесконечный цикл показа кадров модели в виде. | – |
| WinFormController | public WinFormController()  Производит загрузку главного меню игры и вида, запуск цикла отрисовки кадров. | – |
| public void KeyDown(object sender, KeyEventArgs e)  Обработчик события вида - нажатия кнопки на клавиатуре. | object sender  Отправитель. |
| KeyEventArgs e  Аргументы события. |
| public void KeyUp(object sender, KeyEventArgs e)  Обработчик события вида - отпускания кнопки на клавиатуре. | object sender  Отправитель. |
| KeyEventArgs e  Аргументы события. |
| BaseModelLogic | public BaseModelLogic(GameModel parModel)  Инициализирует логику модели. | GameModel parModel  Модель. |
| public abstract void ReceiveCommand(ModelCommand parCommand, bool parBeginCommand)  Получение команды от контроллера. | ModelCommand parCommand  Команда. |
| bool parBeginCommand  Флаг начала команды (true, если начата). |
| protected abstract void HandleCommand(ModelCommand parCommand)  Обработка команды контроллера внутри логики модели. | ModelCommand parCommand  Команда. |
| BaseObjectLogic | public BaseObjectLogic(GameModel parLevel)  Инициализирует логику объекта. | GameModel parLevel  Уровень. |
| public void Start(ManualResetEventSlim parManualResetEventSlim)  Запуск потокового цикла вычислений. | ManualResetEventSlim parManualResetEventSlim  Событие для дальнейшей постановки потока на паузу. |
| public void Stop()  Остановка потокового цикла вычислений. | – |
| public virtual void Resume()  Дополнительные действия при возобновлении работы потокового цикла вычислений после паузы. | – |
| protected abstract void IterativeAction()  Потоковый цикл вычислений. | – |
| protected float[ ] FindFreeSpace()  Нахождение свободного пространства вокруг целевого объекта. | <returns>Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу) |
| protected bool IsIntersected(float parMin1, float parMax1, float parMin2, float parMax2)  Проверка пересечения двух отрезков. | float parMin1  Левая граница отрезка 1. |
| float parMax1  Правая граница отрезка 1. |
| float parMin2  Левая граница отрезка 2. |
| float parMax2  Правая граница отрезка 2. |
| <returns>True, если отрезки пересекаются. |
| protected float[ ] FindMove(float[ ] parSpeed, float[ ] parFreeSpace, float parDeltaSeconds)  Нахождение вектора перемещения. | float[ ] parSpeed  Вектор скорости (X, Y). |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| <returns>Вектор перемещения (X, Y). |
| protected void MoveObject(float[ ] parMove)  Перемещение объекта. | float[ ] parMove  Вектор перемещения (X, Y). |
| protected void FlipObject(float[ ] parMove)  Отражение объекта по вертикали (при необходимости). | float[ ] parMove  Вектор перемещения (X, Y). |
| BatLogic | public BatLogic(GameModel parLevel, EnemyObject parEnemy)  Инициализирует логику объекта. | GameModel parLevel  Уровень. |
| EnemyObject parEnemy  Объект. |
| protected override void IterativeAction()  Потоковый цикл вычислений. | – |
| BatStateMachine | public BatStateMachine()  Инициализирует автомат состояний объекта. | – |
| public override void ChangeState(GameObject parEnemy, float[ ] parFreeSpace, float[ ] parMove, float parDeltaSeconds)  Изменение состояния автомата и состояния объекта на основе входных данных. | GameObject parEnemy  Целевой объект. |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float[ ] parMove  Вектор перемещения (X, Y). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| CustomAttribute | public CustomAttribute(object parValue)  Инициализирует атрибут значением. | object parValue  Значение для хранения. |
| public static object GetValue(Type parEnumType, string parEnumValue)  Получение значения пользовательского атрибута по типу перечисления и строковому имени нужного поля. | Type parEnumType  Тип перечисления. |
| string parEnumValue  Имя поля. |
| <returns>Значение пользовательского атрибута. |
| EnemyLogicFactory | public static ILogic CreateLogic(GameModel parLevel, EnemyObject parObject)  Создание логики для врага. | GameModel parLevel  Уровень. |
| EnemyObject parObject  Целевой объект. |
| <returns>Объект логики для врага. |
| GameModel | public void SetCameraSize(float parSizeX, float parSizeY)  Изменение размеров камеры модели. Вызывается видом через делегат. | float parSizeX  Размер камеры по горизонтали (в клетках). |
| float parSizeY  Размер камеры по вертикали (в клетках). |
| GhostLogic | public GhostLogic(GameModel parLevel, EnemyObject parEnemy)  Инициализирует логику объекта. | GameModel parLevel  Уровень. |
| EnemyObject parEnemy  Объект. |
| protected override void IterativeAction()  Потоковый цикл вычислений. | – |
| GhostStateMachine | public GhostStateMachine()  Инициализирует автомат состояний объекта. | – |
| public override void ChangeState(GameObject parEnemy, float[ ] parFreeSpace, float[ ] parMove, float parDeltaSeconds)  Изменение состояния автомата и состояния объекта на основе входных данных. | GameObject parEnemy  Целевой объект. |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float[ ] parMove  Вектор перемещения (X, Y). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| ILogic | void Start(ManualResetEventSlim parManualResetEventSlim)  Запуск потокового цикла вычислений. | ManualResetEventSlim parManualResetEventSlim  Событие для дальнейшей постановки потока на паузу. |
| void Stop()  Остановка потокового цикла вычислений. | – |
| void Resume()  Дополнительные действия при возобновлении работы потокового цикла вычислений после паузы. | – |
| LevelLoader | public static GameModel Load(int parLevelID)  Загрузка указанного уровня. | int parLevelID  ID уровня |
| <returns>Уровень. |
| public static List<int> CheckAvailableLevels()  Анализ папки с уровнями и выдача списка ID имеющихся уровней. | <returns>Список ID уровней. |
| LevelLogic | public LevelLogic(GameModel parModel)  Инициализирует логику модели. | GameModel parModel  Модель. |
| public void Start()  Запуск работы логики уровня и всех логик объектов. | – |
| public void Stop()  Остановка работы логики уровня и всех логик объектов. | – |
| public void Pause()  Пауза работы логики уровня и всех логик объектов. | – |
| public void Resume()  Возобновление работы логики уровня и всех логик объектов. | – |
| public override void ReceiveCommand(ModelCommand parCommand, bool parBeginCommand)  Получение команды от контроллера. | ModelCommand parCommand  Команда. |
| bool parBeginCommand  Флаг начала команды (true, если начата). |
| protected override void HandleCommand(ModelCommand parCommand)  Обработка команды контроллера внутри логики модели. | ModelCommand parCommand  Команда. |
| LevelMenuState Machine | public LevelMenuStateMachine(GameModel parModel, int parSelectedMenuItem = 0)  Инициализирует автомат состояний начальными значениями. | GameModel parModel  Модель. |
| int parSelectedMenuItem = 0  (Необязательно) выбранный пункт меню. |
| public override void ChangeState(ModelCommand parCommand)  Изменение состояния согласно команде извне. | ModelCommand parCommand  Команда. |
| public void EnterToMenu(UIObjectType parType)  Выбор процедуры инициализации нужного меню. | UIObjectType parType  Тип меню. |
| protected override void AcceptAction()  Обработка действия подтверждения (обычно нажатие клавишы Enter). | – |
| protected override void CancelAction()  Обработка действия отмены (обычно нажатие клавишы Escape). | – |
| MainMenuState Machine | public MainMenuStateMachine(GameModel parModel, int parSelectedMenuItem = 0)  Инициализирует автомат состояний начальными значениями. | GameModel parModel  Модель. |
| int parSelectedMenuItem = 0  (Необязательно) выбранный пункт меню. |
| public override void ChangeState(ModelCommand parCommand)  Изменение состояния согласно команде извне. | ModelCommand parCommand  Команда. |
| protected override void AcceptAction()  Обработка действия подтверждения (обычно нажатие клавишы Enter). | – |
| protected override void CancelAction()  Обработка действия отмены (обычно нажатие клавишы Escape). | – |
| MenuLoader | public static GameModel Load()  Загрузка меню. | <returns>Меню. |
| MenuLogic | public MenuLogic(GameModel parModel)  Инициализирует логику модели. | GameModel parModel  Модель. |
| public override void ReceiveCommand(ModelCommand parCommand, bool parBeginCommand)  Получение команды от контроллера. | ModelCommand parCommand  Команда. |
| bool parBeginCommand  Флаг начала команды (true, если начата). |
| protected override void HandleCommand(ModelCommand parCommand)  Обработка команды контроллера внутри логики модели. | ModelCommand parCommand  Команда. |
| ModelStateMachine | public ModelStateMachine(GameModel parModel, int parSelectedMenuItem = 1)  Инициализирует автомат состояний начальными значениями. | GameModel parModel  Модель. |
| int parSelectedMenuItem = 0  (Необязательно) выбранный пункт меню. |
| public abstract void ChangeState(ModelCommand parCommand)  Изменение состояния согласно команде извне. | ModelCommand parCommand  Команда. |
| protected void SelectMenuItem(int parItemNumber)  Выбор указанного элемента меню. | int parItemNumber  Номер элемента для выбора. |
| protected void SelectPrevMenuItem()  Выбор предыдущего элемента меню. | – |
| protected void SelectNextMenuItem()  Выбор следующего элемента меню. | – |
| protected abstract void AcceptAction()  Обработка действия подтверждения (обычно нажатие клавишы Enter). | – |
| protected abstract void CancelAction()  Обработка действия отмены (обычно нажатие клавишы Escape). | – |
| ObjectState Machine | public ObjectStateMachine()  Инициализирует автомат состояний объекта. | – |
| public abstract void ChangeState(GameObject parGameObject, float[ ] parFreeSpace, float[ ] parMove, float parDeltaSeconds)  Изменение состояния автомата и состояния объекта на основе входных данных. | GameObject parGameObject  Целевой объект. |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float[ ] parMove  Вектор перемещения (X, Y). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| public void SetState(MachineStatesEnum parState)  Установка нужного состояния автомата. | MachineStatesEnum parState  Целевое состояние. |
| PlayerLogic | public PlayerLogic(GameModel parLevel)  Инициализирует логику объекта. | GameModel parLevel  Уровень. |
| public void ReceiveCommand(ModelCommand parCommand, bool parBeginCommand)  Получение команды. | ModelCommand parCommand  Команда. |
| bool parBeginCommand  Флаг начала команды (true, если начата). |
| public override void Resume()  Дополнительные действия при возобновлении работы потокового цикла вычислений после паузы. | – |
| protected override void IterativeAction()  Потоковый цикл вычислений. | – |
| PlayerState Machine | public PlayerStateMachine()  Инициализирует автомат состояний объекта. | – |
| public override void ChangeState(GameObject parPlayer, float[ ] parFreeSpace, float[ ] parMove, float parDeltaSeconds)  Изменение состояния автомата и состояния объекта на основе входных данных. | GameObject parPlayer  Целевой объект. |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float[ ] parMove  Вектор перемещения (X, Y). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| SaveFile | public static SaveFile GetInstance()  Получение синглтонного экземпляра объекта для файла сохранения. | <returns>Объект для файла сохранения. |
| public void CheckAndSetRecord(int parLevelNumber, TimeSpan parRecord)  Проверка и установка (при необходимости) рекорда. | int parLevelNumber  ID уровня. |
| TimeSpan parRecord  Рекорд. |
| SlimeLogic | public SlimeLogic(GameModel parLevel, EnemyObject parEnemy)  Инициализирует логику объекта. | GameModel parLevel  Уровень. |
| EnemyObject parEnemy  Объект. |
| protected override void IterativeAction()  Потоковый цикл вычислений. | – |
| SlimeStateMachine | public SlimeStateMachine()  Инициализирует автомат состояний объекта. | – |
| public override void ChangeState(GameObject parEnemy, float[ ] parFreeSpace, float[ ] parMove, float parDeltaSeconds)  Изменение состояния автомата и состояния объекта на основе входных данных. | GameObject parEnemy  Целевой объект. |
| float[ ] parFreeSpace  Массив свободных расстояний вокруг объекта (слева, сверху, справа, снизу). |
| float[ ] parMove  Вектор перемещения (X, Y). |
| float parDeltaSeconds  Время, прошедшее с предыдущего шага (в секундах). |
| BufferedWinForm Drawer | public BufferedWinFormDrawer(Graphics parFormGraphics, int parWidth, int parHeight)  Инициализирует создатель кадров параметрами целевой формы. | Graphics parFormGraphics  Объект графики для отрисовки готового кадра. |
| int parWidth  Ширина формы. |
| int parHeight  Высота формы. |
| public void SendCameraSizeToModel()  Установка размеров камеры в модели. | – |
| public void DrawFrameToForm(GameModel parModel, Graphics parFormGraphics)  Создание кадра и его отрисовка на форме. | GameModel parModel  Модель. |
| Graphics parFormGraphics  Объект графики для отрисовки готового кадра. |
| public void FormSizeChanged(Graphics parFormGraphics, int parWidth, int parHeight)  Реакция на событие изменения размеров формы. | Graphics parFormGraphics  Объект графики для отрисовки готового кадра. |
| int parWidth  Ширина формы. |
| int parHeight  Высота формы. |
| FormMain | public FormMain()  Инициализирует форму начальными значениями. | – |
| public void SendCameraSizeDelegateSending(dSetCameraSize parSetCameraSize)  Отправка ссылки на метод установки размеров камеры в модели. | dSetCameraSize parSetCameraSize  Ссылка на метод. |
| public void SendCameraSizeToModel()  Установка размеров камеры в модели. | – |
| public void ViewModel(GameModel parModel)  Отображение модели. | GameModel parModel  Модель. |
| IViewable | void SendCameraSizeDelegateSending(dSetCameraSize parSetCameraSize)  Отправка ссылки на метод установки размеров камеры в модели. | dSetCameraSize parSetCameraSize  Ссылка на метод. |
| void SendCameraSizeToModel()  Установка размеров камеры в модели. | – |
| void ViewModel(GameModel parModel)  Отображение модели. | GameModel parModel  Модель. |
| ResourceLoader | public static SpritesContainer LoadSprites()  Загрузка спрайтов. | <returns>Хранилище спрайтов. |
| public static PrivateFontCollection LoadFontCollection()  Загрузка коллекции шрифтов. | <returns>Коллекция шрифтов. |
| SpritesContainer | public SpritesContainer(  Dictionary<int, Image> parBackgrounds,  Dictionary<int, List<Image>> parLevelObjectSprites,  Dictionary<int, List<Image>> parEnemySprites,  Dictionary<int, List<Image>> parPlayerSprites,  Dictionary<int, Image> parUISprites)  Инициализирует хранилище спрайтов. | Dictionary<int, Image> parBackgrounds  Словарь фонов. |
| Dictionary<int, List<Image>> parLevelObjectSprites  Словарь спрайтов элементов уровня. |
| Dictionary<int, List<Image>> parEnemySprites  Словарь спрайтов врагов. |
| Dictionary<int, List<Image>> parPlayerSprites  Словарь спрайтов игрока. |
| Dictionary<int, Image> parUISprites  Словарь спрайтов элементов интерфейса. |
| public Image GetLevelSprite(GameObject parObject)  Получение спрайта для объекта. | GameObject parObject  Объект. |
| <returns>Спрайт. |
| public Image GetUISprite(UIObject parObject) | UIObject parObject  Объект. |
| <returns>Спрайт. |
| public Image GetBackground(GameModelType parType)  Получение спрайта для фона. | GameModelType parType  Тип модели. |
| <returns>Спрайт. |

## Описание шаблонов проектирования, которые использовались при написании программы

### MVC (Model-View-Controller, Модель-Представление-Контроллер)

*MVC* – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

* Модель (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.
* Представление (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* Контроллер (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

В данном приложении это основной шаблон проектирования. Часть контроллера реализована в виде абстрактного класса *GameController* и его наследника для работы с *Windows Forms* – *WinFormController*. За представление отвечает класс *FormMain* (наследник стандартного класса Form), реализующий интерфейс *IViewable*. Базовым классом модели является *GameModel*, хранящий в себе все текущие объекты модели.

### 2.2.2 Factory Method (Фабричный метод)

Фабричный метод – порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Существуют различные вариации данного шаблона. В частности, в данной программе в классе *EnemyLogicFactory* реализован универсальный фабричный метод, который на основании типа входного объекта создает для него логику нужного конкретного класса (реализующего общий интерфейс *ILogic*).

### 2.2.3 Singleton (Синглтон, Одиночка)

Одиночка – порождающий шаблон проектирования, гарантирующий, что в приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

У класса есть только один экземпляр, и он предоставляет к нему глобальную точку доступа. Существенно то, что можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность.

Глобальный «одинокий» объект – именно объект, а не набор процедур, не привязанных ни к какому объекту – бывает нужен:

* если используется существующая объектно-ориентированная библиотека;
* если есть шансы, что один объект когда-нибудь превратится в несколько;
* если интерфейс объекта (например, игрового мира) слишком сложен и не стоит засорять основное пространство имён большим количеством функций;
* если, в зависимости от каких-нибудь условий и настроек, создаётся один из нескольких объектов.

В данном приложении по шаблону Одиночка реализован класс по работе с файлом сохранения – *SaveFile*. При работе программы необходимо быть уверенным, что создан только один объект по работе с файлом сохранения и не возникнет никаких коллизий при многократном обращении к этому файлу. Более того, объект класса *SaveFile* старается минимизировать частоту обращения к файлу сохранения на диске, загружая все данные в свои поля и обновляя файл лишь при необходимости.

## Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы

### Инкапсуляция поля

Если поле является открытым, другие объекты имеют к нему прямой доступ и могут изменять его незаметно для объекта, которому принадлежит это поле. Инкапсуляция поля с помощью свойств позволяет запретить прямой доступ к полям.

В C# для удобной реализации инкапсулированных полей есть автосвойства, скрывающие создание приватного поля и его инкапсуляцию свойством.

Примером может служить изменение класса *PlayerLogic*. В момент, когда одно из его полей потребовалось сделать доступным извне, была произведена его инкапсуляция через замену автосвойством.

До:

public class PlayerLogic

{

private GameLevel \_level;

После:

public class PlayerLogic

{

public GameLevel Level { get; set; }

### Выделение метода

Выделение метода заключается в выделении из длинного и/или требующего комментариев кода отдельных фрагментов и преобразовании их в отдельные методы, с подстановкой подходящих вызовов в местах использования. В этом случае действует правило: если фрагмент кода требует комментария о том, что он делает, то он должен быть выделен в отдельный метод. Также правило: один метод не должен занимать более чем один экран (25-50 строк, в зависимости от условий редактирования), в противном случае некоторые его фрагменты имеют самостоятельную ценность и подлежат выделению. Из анализа связей выделяемого фрагмента с окружающим контекстом делается вывод о перечне параметров нового метода и его локальных переменных.

Пример применения данного вида рефакторинга при написании класса :

До:

/// <summary>

/// Проверка позиции игрока. При необходимости изменяет состояние автомата

/// и сохраняет рекорд.

/// </summary>

/// <returns>True, если игрок у цели и нужно остановить поток.</returns>

private bool CheckPlayerPosition()

{

bool stopThread = false;

bool playerInGoalPoint = false;

foreach (LevelObject elDoor in \_model.Doors)

{

if (elDoor.State == 1)

{

playerInGoalPoint |= IsIntersected(  
 \_model.Player.X + \_model.Player.SizeX \* PLAYER\_TO\_DOOR\_WIN\_OFFSET,

\_model.Player.X + \_model.Player.SizeX \* (1 - PLAYER\_TO\_DOOR\_WIN\_OFFSET),

elDoor.X,

elDoor.X + elDoor.SizeX)

&& IsIntersected(

\_model.Player.Y + \_model.Player.SizeY \* PLAYER\_TO\_DOOR\_WIN\_OFFSET,

\_model.Player.Y + \_model.Player.SizeY \* (1 - PLAYER\_TO\_DOOR\_WIN\_OFFSET),

elDoor.Y,

elDoor.Y + elDoor.SizeY);

}

}

if (playerInGoalPoint)

{

...

stopThread = true;

}

return stopThread;

}

После:

/// <summary>

/// Проверка позиции игрока. При необходимости изменяет состояние автомата

/// и сохраняет рекорд.

/// </summary>

/// <returns>True, если игрок у цели и нужно остановить поток.</returns>

private bool CheckPlayerPosition()

{

bool stopThread = false;

if (IsPlayerInGoalPoint())

{

...

stopThread = true;

}

return stopThread;

}

/// <summary>

/// Проверка достижения игроком цели.

/// </summary>

/// <returns>True, если игрок у цели.</returns>

private bool IsPlayerInGoalPoint()

{

bool result = false;

foreach (LevelObject elDoor in \_model.Doors)

{

if (elDoor.State == 1)

{

result |= IsIntersected(  
 \_model.Player.X + \_model.Player.SizeX \* PLAYER\_TO\_DOOR\_WIN\_OFFSET,

\_model.Player.X + \_model.Player.SizeX \* (1 - PLAYER\_TO\_DOOR\_WIN\_OFFSET),

elDoor.X,

elDoor.X + elDoor.SizeX)

&& IsIntersected(

\_model.Player.Y + \_model.Player.SizeY \* PLAYER\_TO\_DOOR\_WIN\_OFFSET,

\_model.Player.Y + \_model.Player.SizeY \* (1 - PLAYER\_TO\_DOOR\_WIN\_OFFSET),

elDoor.Y,

elDoor.Y + elDoor.SizeY);

}

}

return result;

}

### Выделение класса

Этот прием рефакторинга также рассматривают как один из структурных шаблонов проектирования. Его суть заключается в разделении обязанностей между различными классами. Он позволяет соблюсти «Принцип единственной ответственности», тем самым делая классы более надёжными и устойчивыми к изменениям. Выделение класса необходимо производить в том случае, если в текущем классе наблюдается перегруженность различными функциями, которые возможно разделить.

Примером выделения класса может служить создание класса *BufferedWinFormDrawer*, выделенного из методов работы с графикой класса *FormMain*. Код этих классов находится в Примечаниях.

### Подъем метода

Данный вид рефакторинга состоит в переносе метода из класса-потомка в родительский класс. Это необходимо в том случае, когда один и тот же метод используется в нескольких классах-потомках.

Примером может послужить рефакторинг иерархии логик объектов. Изначально была спроектирована и отлажена логика персонажа игрока, затем началась работа над логиками врагов. При этом возникла необходимость в однотипных методах, которые по возможности были вынесены из класса логики игрока *PlayerLogic* в базовый класс *BaseObjectLogic*. Один из таких методов – запуск цикла вычислений:

/// <summary>

/// Запуск потокового цикла вычислений.

/// </summary>

/// <param name="parManualResetEventSlim">Событие для дальнейшей постановки

/// потока на паузу.</param>

public void Start(ManualResetEventSlim parManualResetEventSlim)

{

\_manualResetEventSlim = parManualResetEventSlim;

\_stopThread = false;

Thread logicThread = new Thread(IterativeAction)

{

IsBackground = true

};

logicThread.Start();

}

### Переименование метода

Переименование различных идентификаторов в коде программы призвано повысить читаемость кода без изменения его структуры. Необходимость в переименовании может быть вызвана тем, что изначально правильному имени идентификатора не придавалось большое значение и оно было подобранно плохо. Также нередки ситуации, когда даже хорошее имя требует замены ввиду изменения функциональности.

Наглядным примером второго случая является момент, когда при разработке программы было решено унифицировать механизм отображения меню и игрового уровня, рассматривая их как одинаковые объекты модели. При этом во многих местах в различных классах возникла необходимость сменить упоминания уровня на более абстрактное понятие модели. Например, ниже представлена часть изменений в классе *GameController*.

До:

protected void LoadLevel(int parLevelID)

После:

protected void LoadModel(int parModelID)

## Разработка тестов

Помимо ручного тестирования в процессе разработки программы, программа частично протестирована модульными тестами. Стоит заметить, что данный процесс был осложнен сильно ограниченной в плане модульного тестирования функциональностью бесплатной редакции Visual Studio. В этом случае полностью протестировать возможно только те классы, которые не используют в своих методах функциональность других классов.

Было решено протестировать класс автомата состояний для логики игрока *PlayerStateMachine*, как важную часть логики работы приложения. Этот класс содержит два публичных метода по работе с состояниями – *SetState()* (установка указанного состояния) и *ChangeState()* (смена состояния с учетом внешних условий). Эти методы были протестированы для всех классов эквивалентности исходных данных, то есть проверены для всех возможных типовых ситуаций. В результате тестирования ошибок не было выявлено.

# Результат работы программы

Готовая программа поставляется в виде исполняемого файла «AlienExplorer.exe» и находящейся с ним в одном каталоге папки «resources» с файлами ресурсов.

Для старта программы необходимо запустить исполняемый файл. При том на полный экран будет открыто окно приложения. Оно будет содержать главное меню игры, представленное на рисунке 3.1.



Рисунок 3.1. Главное меню.

В программе назначены следующие ассоциации клавиш и команд:

* *←, A* – влево;
* *→, D* – вправо;
* *↑, W, Space* – вверх;
* *↓, S, Control* – вниз;
* *Enter* – подтверждение;
* *Escape, Backspace* – отмена.

Для начала новой игры выберите пункт меню «New game» и нажмите Enter. Если существует ранее сохраненный прогресс, игра уведомит о его потере при начале новой игры и запросит подтверждение действия (рисунок 3.2). Подтвердите или отмените действие выбором соответствующего пункта меню и нажатием Enter (или нажатием Escape, означающем отмену).



Рисунок 3.2. Подтверждение начала новой игры.

В игре присутствует автосохранение прогресса. Для загрузки игры с текущего уровня выберите пункт меню «Load game» (рисунок 3.1). Подтвердите действие в показанном диалоге, после чего будет загружен последний на момент предыдущей игры уровень.

Вы также можете загрузить на выбор любой уровень из тех, что были открыты за все предыдущие игры. Для этого выберите пункт меню «Choose level», затем выберите нужный уровень (рисунок 3.3).

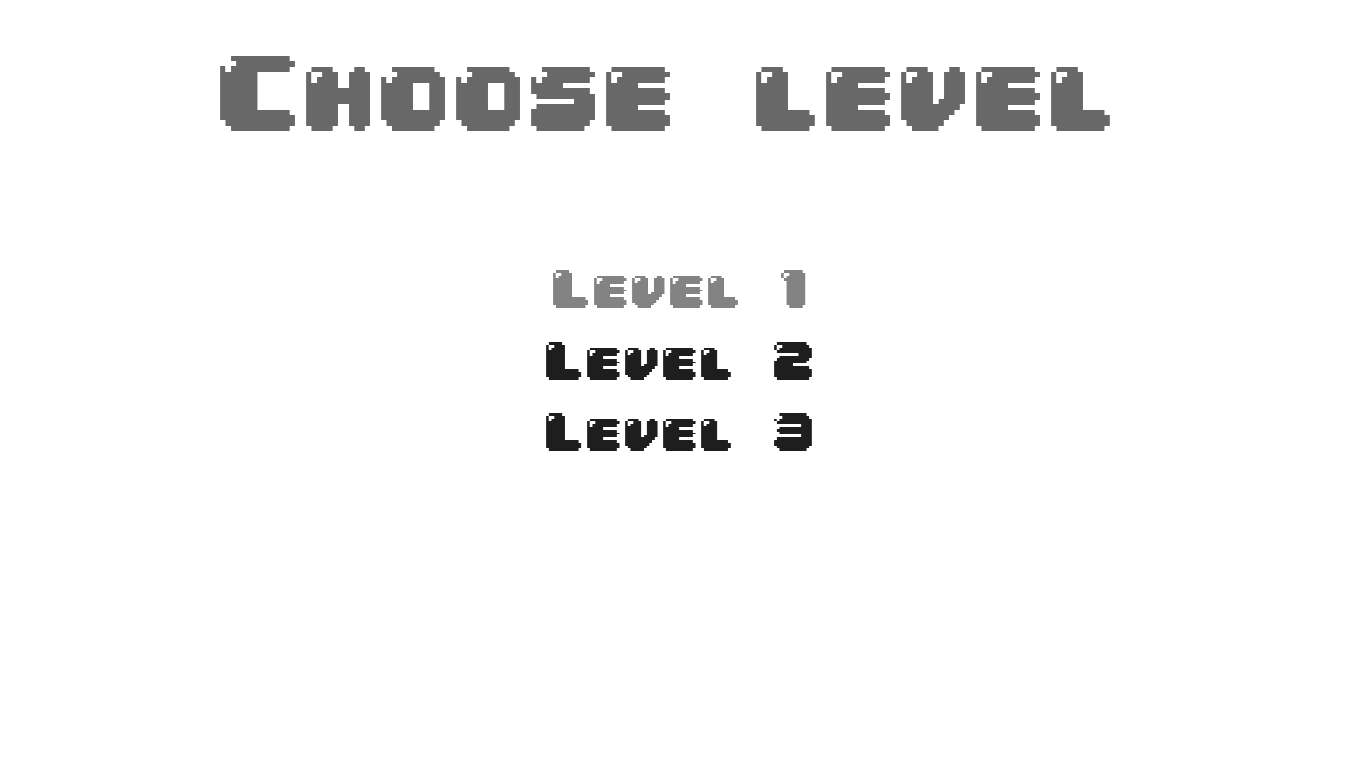


Рисунок 3.3. Выбор уровня.

Главной целью игры является прохождение уровня за как можно меньшее время. При этом рекорд времени фиксируется только в том случае, если персонажу не было нанесено урона. Текущие рекорды можно просмотреть, перейдя из главного меню (рисунок 3.1) в меню «Records», представленному на рисунке 3.4.



Рисунок 3.4. Таблица рекордов.

Для выхода из игры выберите пункт меню «Exit» и подтвердите действие.

При загрузке уровня вы увидите интерфейс, представленный на рисунке 3.5. В левом нижнем углу показаны очки жизни игрока. В верхней части окна выведен таймер прохождения уровня.



Рисунок 3.5. Интерфейс уровня.

Для паузы игры нажмите на кнопку Escape. При этом вы увидите меню паузы, показанное на рисунке 3.6, а процесс игры приостановится (вместе с таймером). Похожие меню будут выведены и в случаях выигрыша и проигрыша. Чтобы выиграть уровень, достигните открытой двери. В случае, если количество очков жизни опустится до нуля, уровень будет проигран.

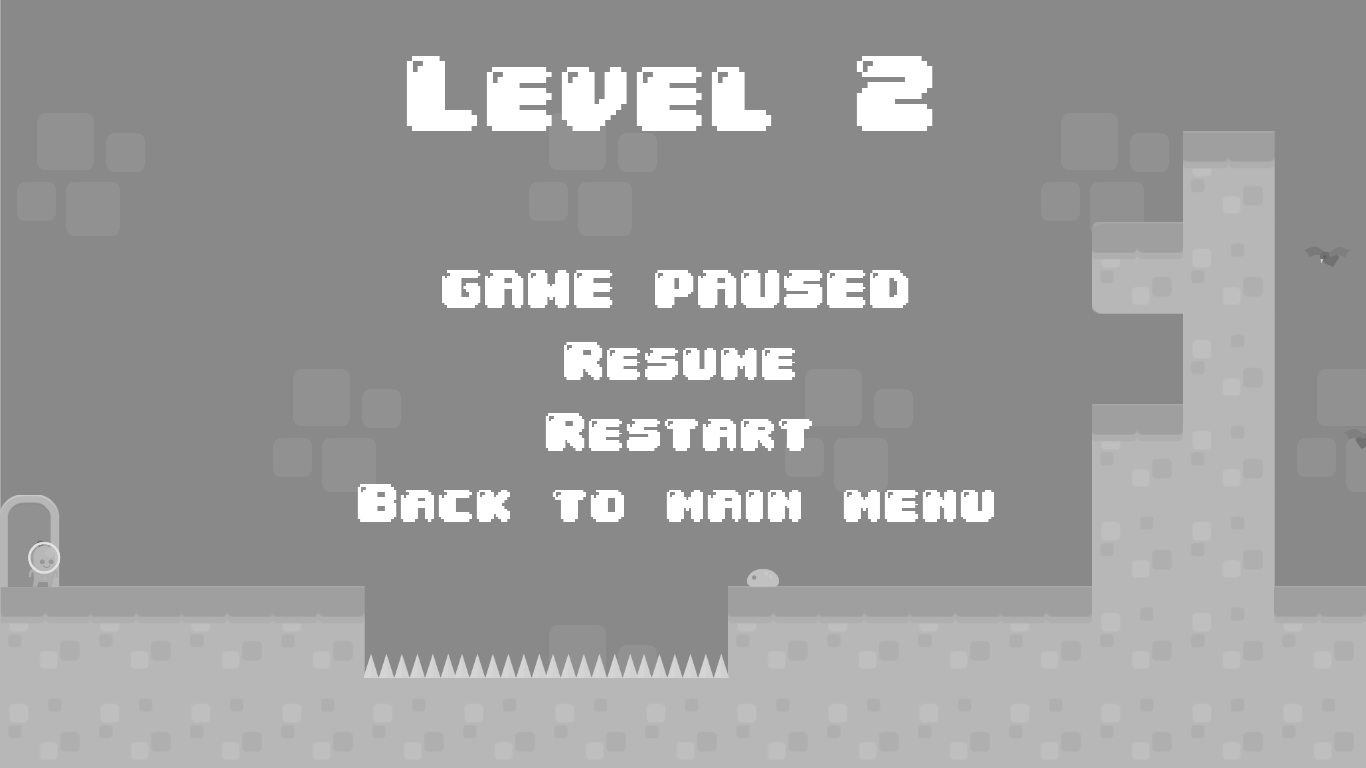


Рисунок 3.6. Меню паузы игрового уровня.

На данный момент в игре доступно три уровня. Вы можете самостоятельно добавлять уровни, создавая их конфигурационные файлы в формате *XML* и помещая их в папку «resources/levels». При этом название файла должно соответствовать шаблону «levelX», где X – номер уровня (уникальное целое число).

Обзорные виды разработанных уровней представлены на рисунках 3.7, 3.8 и 3.9.

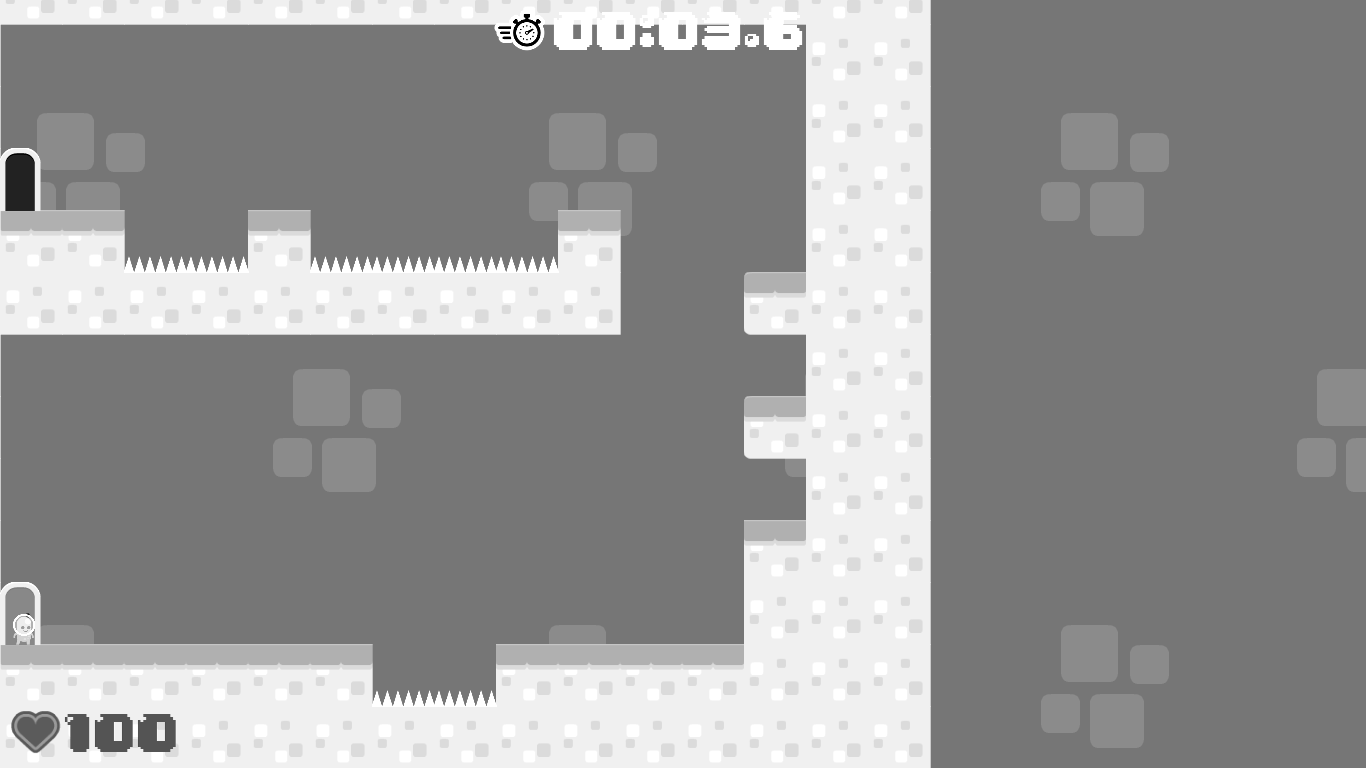


Рисунок 3.7. Уровень 1.

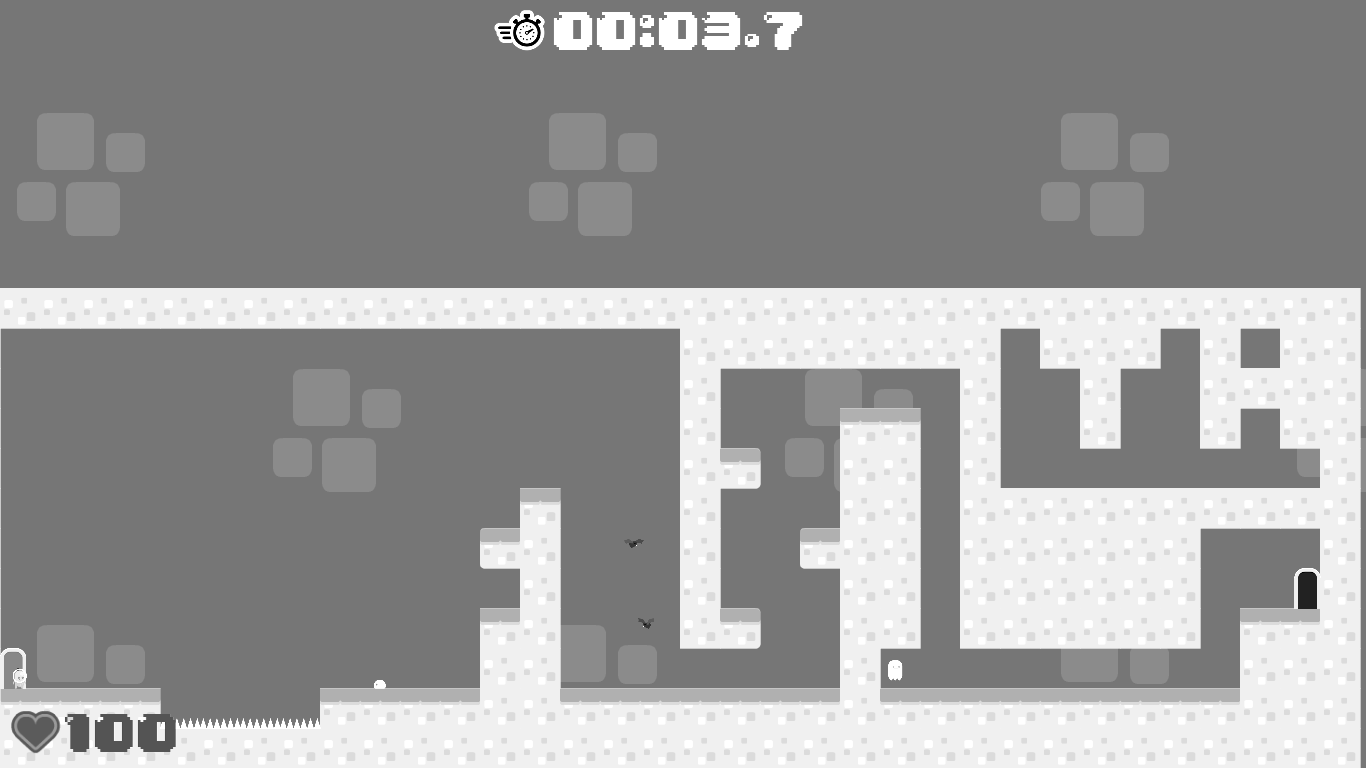


Рисунок 3.8. Уровень 2.

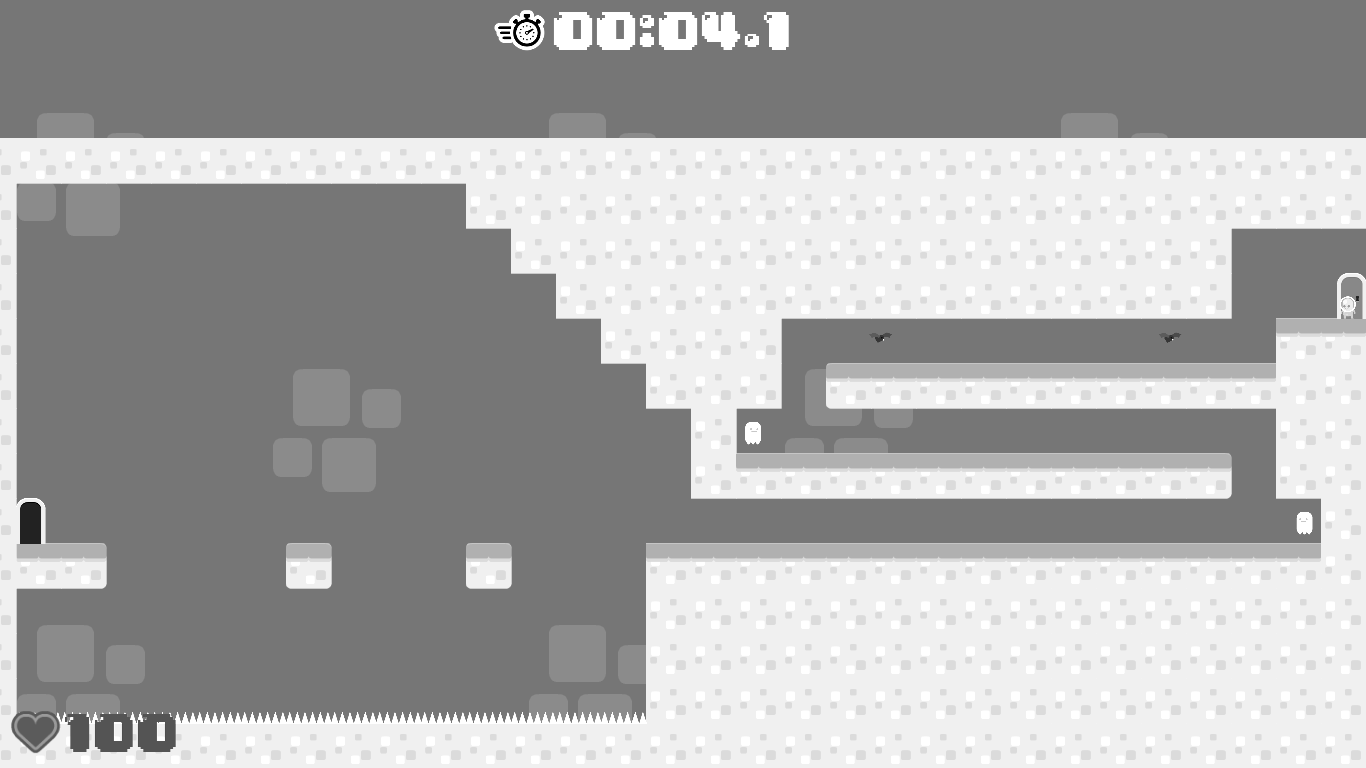


Рисунок 3.9. Уровень 3.

Заключение

В ходе данного проекта была разработана игровая программа, реализующая все запланированные функции и соответствующая предъявляемым требованиям. При проектировании были использованы различные шаблоны, упрощающие разработку и развитие программы. Код программы соответствует установленному соглашению по написанию кода. Были применены различные виды рефакторинга кода, повысившие его читаемость и оптимальность. Также приложение частично покрыто модульными тестами, что снижает вероятность возникновения ошибок.

Для дальнейшего развития проекта возможно добавление новых уровней, врагов и механик. Персонажу игрока можно установить ограничение по запасу энергии, которая тратится на действия (например, прыжки). Также существенным плюсом будет перенос механизма отображения изображения с *Windows Forms* на более подходящий способ (например, использовать для этого *DirectX*). Кроме того, если сделать удобный визуальный редактор уровней с генерацией выходных *XML* файлов, то к наполнению игры содержимым можно будет привлечь заинтересованное сообщество.

Приложения

В данном разделе помещен код наиболее важных классов программы.

*GameController.cs*

using System;

using System.Threading;

using AlienExplorer.Model;

using AlienExplorer.View;

namespace AlienExplorer.Controller

{

/// <summary>

/// Абстрактный базовый класс контроллеров.

/// </summary>

public abstract class GameController

{

/// <summary>

/// Вид

/// </summary>

protected IViewable View { get; set; }

/// <summary>

/// Модель

/// </summary>

protected GameModel Model { get; set; }

/// <summary>

/// Произвоит загрузку главного меню игры.

/// </summary>

public GameController()

{

LoadMenu();

}

/// <summary>

/// Загрузка игрового уровня.

/// </summary>

/// <param name="parLevelID">ID уровня.</param>

protected void LoadLevel(int parLevelID)

{

try

{

Model = LevelLoader.Load(parLevelID);

if (View != null)

{

View.SendCameraSizeDelegateSending(Model.SetCameraSize);

View.SendCameraSizeToModel();

}

((LevelLogic)Model.ModelLogic).Start();

Model.ModelLogic.LoadAnotherModel += LoadAnotherModel;

}

catch

{

LoadMenu();

}

}

/// <summary>

/// Загрузка главного меню игры.

/// </summary>

protected void LoadMenu()

{

Model = MenuLoader.Load();

if (View != null)

{

View.SendCameraSizeDelegateSending(Model.SetCameraSize);

View.SendCameraSizeToModel();

}

Model.ModelLogic.LoadAnotherModel += LoadAnotherModel;

((MenuLogic)Model.ModelLogic).CloseApplication += CloseApplication;

}

/// <summary>

/// Команда контроллеру загрузить другую модель. Вызывается логикой старой модели через делегат.

/// </summary>

/// <param name="parModelType">Тип модели для загрузки.</param>

/// <param name="parLevelID">(Необязательно) ID уровня.</param>

private void LoadAnotherModel(GameModelType parModelType, int parLevelID = 1)

{

if (parModelType == GameModelType.Menu)

{

LoadMenu();

}

else

{

LoadLevel(parLevelID);

}

Thread delayedGC = new Thread(GCcollectWithDelay);

delayedGC.Start(500);

}

/// <summary>

/// Отсроченная сборка мусора.

/// </summary>

/// <param name="parData">Задержка перед сборкой в миллисекундах.</param>

private void GCcollectWithDelay(object parData)

{

Thread.Sleep((int)parData);

GC.Collect();

}

/// <summary>

/// Бесконечный цикл показа кадров модели в виде.

/// </summary>

protected void SendModelToView()

{

while (true)

{

if ((View != null) && (Model != null))

{

View.ViewModel(Model);

}

}

}

/// <summary>

/// Закрытие приложения. Вызывается при закрытии вида или из логики модели через делегат.

/// </summary>

private void CloseApplication()

{

Environment.Exit(0);

}

}

}

*WinFormController.cs*

using System.Threading;

using System.Windows.Forms;

using AlienExplorer.View;

using AlienExplorer.Model;

namespace AlienExplorer.Controller

{

/// <summary>

/// Контроллер для вида на Windows Forms.

/// </summary>

public class WinFormController : GameController

{

/// <summary>

/// Производит загрузку главного меню игры и вида, запуск цикла отрисовки кадров.

/// </summary>

public WinFormController() : base()

{

View = new FormMain();

View.SendCameraSizeDelegateSending(Model.SetCameraSize);

((Form)View).Show();

((Form)View).KeyDown += KeyDown;

((Form)View).KeyUp += KeyUp;

Thread framesSender = new Thread(SendModelToView);

framesSender.Start();

}

/// <summary>

/// Обработчик события вида - нажатия кнопки на клавиатуре.

/// </summary>

/// <param name="sender">Отправитель.</param>

/// <param name="e">Аргументы события.</param>

public void KeyDown(object sender, KeyEventArgs e)

{

bool beginCommand = true;

SendCommandToPlayer(e, beginCommand);

}

/// <summary>

/// Обработчик события вида - отпускания кнопки на клавиатуре.

/// </summary>

/// <param name="sender">Отправитель.</param>

/// <param name="e">Аргументы события.</param>

public void KeyUp(object sender, KeyEventArgs e)

{

bool beginCommand = false;

SendCommandToPlayer(e, beginCommand);

}

/// <summary>

/// Универсальная обработка событий клавиш.

/// </summary>

/// <param name="parEventArgs">Аргументы события.</param>

/// <param name="parBeginCommand">Флаг нажатия клавишы (если нажата, то true).</param>

private void SendCommandToPlayer(KeyEventArgs parEventArgs, bool parBeginCommand)

{

if ((parEventArgs.KeyCode == Keys.Left) || (parEventArgs.KeyCode == Keys.A))

{

Model.ModelLogic.ReceiveCommand(ModelCommand.Left, parBeginCommand);

}

else if ((parEventArgs.KeyCode == Keys.Right) || (parEventArgs.KeyCode == Keys.D))

{

Model.ModelLogic.ReceiveCommand(ModelCommand.Right, parBeginCommand);

}

else if ((parEventArgs.KeyCode == Keys.Up) || (parEventArgs.KeyCode == Keys.W) || (parEventArgs.KeyCode == Keys.Space))

{

Model.ModelLogic.ReceiveCommand(ModelCommand.Up, parBeginCommand);

}

else if ((parEventArgs.KeyCode == Keys.Down) || (parEventArgs.KeyCode == Keys.S) || (parEventArgs.KeyCode == Keys.ControlKey))

{

Model.ModelLogic.ReceiveCommand(ModelCommand.Down, parBeginCommand);

}

else if (parEventArgs.KeyCode == Keys.Enter)

{

Model.ModelLogic.ReceiveCommand(ModelCommand.OK, parBeginCommand);

}

else if ((parEventArgs.KeyCode == Keys.Escape) || (parEventArgs.KeyCode == Keys.Back))

{

Model.ModelLogic.ReceiveCommand(ModelCommand.Escape, parBeginCommand);

}

}

}

}

*FormMain.cs*

using System;

using System.Drawing;

using System.Windows.Forms;

using System.Threading;

using AlienExplorer.Model;

namespace AlienExplorer.View

{

/// <summary>

/// Вид - форма Windows.

/// </summary>

public partial class FormMain : Form, IViewable

{

/// <summary>

/// Объект графики формы.

/// </summary>

private Graphics \_formGraphics;

/// <summary>

/// Объект буферизированного создателя кадров.

/// </summary>

private BufferedWinFormDrawer \_bufferedDrawer;

/// <summary>

/// Инициализирует форму начальными значениями.

/// </summary>

public FormMain()

{

InitializeComponent();

Cursor.Hide();

\_formGraphics = this.CreateGraphics();

\_bufferedDrawer = new BufferedWinFormDrawer(\_formGraphics, this.Width, this.Height);

}

/// <summary>

/// Отправка ссылки на метод установки размеров камеры в модели.

/// </summary>

/// <param name="parSetCameraSize">Ссылка на метод.</param>

public void SendCameraSizeDelegateSending(dSetCameraSize parSetCameraSize)

{

\_bufferedDrawer.SetCameraSize += parSetCameraSize;

}

/// <summary>

/// Установка размеров камеры в модели.

/// </summary>

public void SendCameraSizeToModel()

{

\_bufferedDrawer?.FormSizeChanged(\_formGraphics, this.Width, this.Height);

}

/// <summary>

/// Отображение модели.

/// </summary>

/// <param name="parModel">Модель.</param>

public void ViewModel(GameModel parModel)

{

Invoke((MethodInvoker)(() => \_bufferedDrawer.DrawFrameToForm(parModel, \_formGraphics)));

}

/// <summary>

/// Обработчик события изменения размеров формы.

/// </summary>

/// <param name="sender">Отправитель события.</param>

/// <param name="e">Аргументы события.</param>

private void FormMain\_SizeChanged(object sender, EventArgs e)

{

\_formGraphics = this.CreateGraphics();

\_bufferedDrawer?.FormSizeChanged(\_formGraphics, this.Width, this.Height);

Thread delayedGC = new Thread(GCcollectWithDelay);

delayedGC.Start(500);

}

/// <summary>

/// Сборка мусора с задержкой.

/// </summary>

/// <param name="parData">Задержка (в миллисекундах).</param>

private void GCcollectWithDelay(object parData)

{

Thread.Sleep((int)parData);

GC.Collect();

}

/// <summary>

/// Обработчик события закрытия формы.

/// </summary>

/// <param name="sender">Отправитель события.</param>

/// <param name="e">Аргументы события.</param>

private void FormMain\_FormClosed(object sender, FormClosedEventArgs e)

{

Environment.Exit(0);

}

}

}

*BufferedWinFormDrawer.cs*

//#define FPSMETER

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Text;

using AlienExplorer.Model;

using System.Threading;

namespace AlienExplorer.View

{

/// <summary>

/// Буферизированный создатель кадров.

/// </summary>

public class BufferedWinFormDrawer

{

/// <summary>

/// Максимальная емкость экрана по ширине (в клетках).

/// </summary>

private static readonly int CELLS\_CAPACITY\_MAX = 15;

/// <summary>

/// Множитель размера шрифта относительно ширины экрана.

/// </summary>

private static readonly float FONT\_MULTIPLIER = 1f / 40;

/// <summary>

/// Емкость меню (в строках).

/// </summary>

private static readonly int MENU\_CAPACITY = 7;

/// <summary>

/// Смещение меню от верха экрана (доля высоты экрана).

/// </summary>

private static readonly float MENU\_OFFSET\_Y = 0.35f;

/// <summary>

/// Контекст буферизированной графики.

/// </summary>

private BufferedGraphicsContext \_bufGraphicsContext;

/// <summary>

/// Объект буферизированной графики.

/// </summary>

private BufferedGraphics \_bufGraphics;

/// <summary>

/// Хранилище спрайтов.

/// </summary>

private SpritesContainer \_spritesContainer;

/// <summary>

/// Текстурные кисти для заливки фона.

/// </summary>

private TextureBrush[ ] \_backgroundBrushes;

/// <summary>

/// Ширина экрана.

/// </summary>

private int \_width;

/// <summary>

/// Высота экрана.

/// </summary>

private int \_height;

/// <summary>

/// Размер клетки в пикселях.

/// </summary>

private int \_cellSize;

/// <summary>

/// Текущая емкость экрана по ширине (в клетках).

/// </summary>

private int \_cellsCapacity = CELLS\_CAPACITY\_MAX;

/// <summary>

/// Коррекция размера клетки (в большую сторону) для устранения промежутков между клетками (в пикселях).

/// </summary>

private float \_drawingCorrection;

/// <summary>

/// Коллекция шрифтов.

/// </summary>

private PrivateFontCollection \_fontCollection;

/// <summary>

/// Стандартный шрифт.

/// </summary>

private Font \_standartFont;

/// <summary>

/// Шрифт заголовка.

/// </summary>

private Font \_headerFont;

#if FPSMETER

/// <summary>

/// Массив счетчиков FPS.

/// </summary>

private int[ ] \_counter;

/// <summary>

/// Таймер для отсчета FPS.

/// </summary>

private DateTime \_time;

#endif

/// <summary>

/// Ссылка на метод модели по установке размеров камеры.

/// </summary>

public dSetCameraSize SetCameraSize { get; set; }

/// <summary>

/// Инициализирует создатель кадров параметрами целевой формы.

/// </summary>

/// <param name="parFormGraphics">Объект графики для отрисовки готового кадра.</param>

/// <param name="parWidth">Ширина формы.</param>

/// <param name="parHeight">Высота формы.</param>

public BufferedWinFormDrawer(Graphics parFormGraphics, int parWidth, int parHeight)

{

\_width = parWidth;

\_height = parHeight;

\_bufGraphicsContext = BufferedGraphicsManager.Current;

\_bufGraphicsContext.MaximumBuffer = new Size(\_width + 1, \_height + 1);

\_bufGraphics = \_bufGraphicsContext.Allocate(parFormGraphics, new Rectangle(0, 0, \_width, \_height));

\_spritesContainer = ResourceLoader.LoadSprites();

\_backgroundBrushes = new TextureBrush[2];

\_backgroundBrushes[0] = new TextureBrush(\_spritesContainer.GetBackground((GameModelType)0));

\_backgroundBrushes[1] = new TextureBrush(\_spritesContainer.GetBackground((GameModelType)1));

\_fontCollection = ResourceLoader.LoadFontCollection();

\_standartFont = new Font(\_fontCollection.Families[0], \_width \* FONT\_MULTIPLIER, FontStyle.Regular, GraphicsUnit.Point, 0);

\_headerFont = new Font(\_fontCollection.Families[0], \_width \* FONT\_MULTIPLIER \* 2, FontStyle.Regular, GraphicsUnit.Point, 0);

\_cellsCapacity = CELLS\_CAPACITY\_MAX;

#if FPSMETER

\_counter = new int[4];

\_time = DateTime.UtcNow;

#endif

SendCameraSizeToModel();

}

/// <summary>

/// Вычисление емкости экрана в клетках и размера клетки.

/// </summary>

/// <param name="parModel">Модель.</param>

private void FindCellsCapacityAndSize(GameModel parModel)

{

int newCellsCapacity;

if (((parModel.SizeX - 1) > CELLS\_CAPACITY\_MAX) || (parModel.SizeX < 2))

{

newCellsCapacity = CELLS\_CAPACITY\_MAX;

}

else

{

newCellsCapacity = (parModel.SizeX - 1);

}

if (newCellsCapacity != \_cellsCapacity)

{

\_cellsCapacity = newCellsCapacity;

FindCellSize();

}

}

/// <summary>

/// Вычисление размера клетки.

/// </summary>

private void FindCellSize()

{

\_cellSize = \_width / \_cellsCapacity;

\_drawingCorrection = \_cellSize / 100f;

SendCameraSizeToModel();

}

/// <summary>

/// Установка размеров камеры в модели.

/// </summary>

public void SendCameraSizeToModel()

{

SetCameraSize?.Invoke(\_width \* 1.0f / \_cellSize, \_height \* 1.0f / \_cellSize);

}

/// <summary>

/// Создание кадра и его отрисовка на форме.

/// </summary>

/// <param name="parModel">Модель.</param>

/// <param name="parFormGraphics">Объект графики для отрисовки готового кадра.</param>

public void DrawFrameToForm(GameModel parModel, Graphics parFormGraphics)

{

FindCellsCapacityAndSize(parModel);

DrawBackground(parModel.Type);

DrawLevel(parModel);

DrawUI(parModel);

ViewCanvas(parFormGraphics);

}

/// <summary>

/// Отрисовка фона.

/// </summary>

/// <param name="parModelType">Тип модели.</param>

private void DrawBackground(GameModelType parModelType)

{

\_bufGraphics.Graphics.FillRectangle(\_backgroundBrushes[(int)parModelType], 0, 0, \_width, \_height);

}

/// <summary>

/// Отрисовка модели.

/// </summary>

/// <param name="parModel">Модель.</param>

private void DrawLevel(GameModel parModel)

{

float cameraX = parModel.CameraX;

float cameraY = parModel.CameraY;

foreach (GameObject elLevelElement in parModel.ModelObjects)

{

DrawGameObject(elLevelElement, parModel, cameraX, cameraY);

}

foreach (GameObject elDoor in parModel.Doors)

{

DrawGameObject(elDoor, parModel, cameraX, cameraY);

}

foreach (GameObject elEnemy in parModel.Enemies)

{

DrawGameObject(elEnemy, parModel, cameraX, cameraY);

}

if (parModel.Player != null)

{

DrawGameObject(parModel.Player, parModel, cameraX, cameraY);

}

}

/// <summary>

/// Отрисовка объекта модели.

/// </summary>

/// <param name="parObject">Объект.</param>

/// <param name="parModel">Модель.</param>

/// <param name="parCameraX">Координата X камеры.</param>

/// <param name="parCameraY">Координата Y камеры.</param>

private void DrawGameObject(GameObject parObject, GameModel parModel, float parCameraX, float parCameraY)

{

if (IsVisible(parObject, parModel))

{

Image sprite = \_spritesContainer.GetLevelSprite(parObject);

\_bufGraphics.Graphics.DrawImage(

sprite,

parObject.X \* \_cellSize - parCameraX \* \_cellSize - \_drawingCorrection,

\_height - (parObject.Y \* \_cellSize + parObject.SizeY \* \_cellSize - parCameraY \* \_cellSize) - \_drawingCorrection,

parObject.SizeX \* \_cellSize + \_drawingCorrection \* 2,

parObject.SizeY \* \_cellSize + \_drawingCorrection \* 2);

}

}

/// <summary>

/// Определение, виден ли объект на экране.

/// </summary>

/// <param name="parObject">Объект.</param>

/// <param name="parModel">Модель.</param>

/// <returns>True, если виден.</returns>

private bool IsVisible(GameObject parObject, GameModel parModel)

{

double leftBound = parModel.CameraX;

double rightBound = parModel.CameraX + \_width / \_cellSize;

double downBound = parModel.CameraY;

double upBound = parModel.CameraY + \_height / \_cellSize;

return ((parObject.X < rightBound)

|| ((parObject.X + parObject.SizeX) > leftBound))

&& ((parObject.Y < upBound)

|| ((parObject.Y + parObject.SizeY) > downBound));

}

/// <summary>

/// Отрисовка интерфейса.

/// </summary>

/// <param name="parModel">Модель.</param>

private void DrawUI(GameModel parModel)

{

if (parModel.ModelLogic.ShadowLevel)

{

\_bufGraphics.Graphics.FillRectangle(new SolidBrush(Color.FromArgb(180, Color.FromArgb(31, 68, 82))), 0, 0, \_width, \_height);

}

string header = parModel.ModelLogic.MenuHeader;

if (!header.Equals(""))

{

int offsetX = (int)((\_width - \_bufGraphics.Graphics.MeasureString(header, \_headerFont).Width) / 2);

\_bufGraphics.Graphics.DrawString(header, \_headerFont, Brushes.LightBlue, offsetX, \_height / 15f);

}

List<UIObject> uiList = null;

int offset = 0;

bool drawUpArrow = false;

bool drawDownArrow = false;

uiList = GetVisibleMenuRange(parModel, ref offset, ref drawUpArrow, ref drawDownArrow);

if (drawUpArrow)

{

DrawUIObject(new UIObject() { Type = UIObjectType.Text, Text = "⯅" }, 0);

}

for (int i = 0; i < uiList.Count; i++)

{

Image sprite = null;

float uiSize = \_bufGraphics.Graphics.MeasureString("0", \_standartFont).Height;

switch (uiList[i].Type)

{

case UIObjectType.Health:

sprite = \_spritesContainer.GetUISprite(uiList[i]);

\_bufGraphics.Graphics.DrawImage(sprite, uiSize / 5f, \_height - uiSize \* 1.2f, uiSize, uiSize);

\_bufGraphics.Graphics.DrawString(

parModel.Player.Health.ToString(), \_standartFont, Brushes.Crimson, uiSize \* 1.1f, \_height - uiSize \* 1.1f);

break;

case UIObjectType.Timer:

sprite = \_spritesContainer.GetUISprite(uiList[i]);

TimeSpan time = ((LevelLogic)parModel.ModelLogic).LevelTimer;

string timeString = $"{time.Minutes:D2}:{time.Seconds:D2}.{(time.Milliseconds / 100):D1}";

int offsetX = (int)((\_width - \_bufGraphics.Graphics.MeasureString("00:00:0", \_standartFont).Width) / 2);

\_bufGraphics.Graphics.DrawImage(sprite, offsetX - uiSize, uiSize / 10f, uiSize, uiSize);

\_bufGraphics.Graphics.DrawString(

timeString, \_standartFont, Brushes.White, offsetX, uiSize / 5f);

break;

default:

DrawUIObject(uiList[i], i + offset);

break;

}

}

if (drawDownArrow)

{

DrawUIObject(new UIObject() { Type = UIObjectType.Text, Text = "⯆" }, MENU\_CAPACITY - 1);

}

}

/// <summary>

/// Получение видимого диапазона элементов меню.

/// </summary>

/// <param name="parModel">Модель.</param>

/// <param name="refOffset">Смещение по строкам на экране.</param>

/// <param name="refDrawUpArrow">Флаг отрисовки кнопки "вверх".</param>

/// <param name="refDrawDownArrow">Флаг отрисовки кнопки "вниз".</param>

/// <returns>Список видимых элементов меню.</returns>

private List<UIObject> GetVisibleMenuRange(

GameModel parModel,

ref int refOffset,

ref bool refDrawUpArrow,

ref bool refDrawDownArrow)

{

List<UIObject> result = null;

refOffset = 0;

if (parModel.UIItems.Count <= MENU\_CAPACITY)

{

result = parModel.UIItems;

}

else

{

int selected = parModel.ModelLogic.SelectedMenuItem;

int indexFrom = 0;

int count = MENU\_CAPACITY;

int itemsBelow = parModel.UIItems.Count - selected - 1;

if (itemsBelow > (MENU\_CAPACITY - 1) / 2)

{

refDrawDownArrow = true;

count--;

}

if (selected > (MENU\_CAPACITY / 2))

{

refDrawUpArrow = true;

count--;

refOffset = 1;

if (refDrawDownArrow)

{

indexFrom = selected + 1 - MENU\_CAPACITY / 2;

}

else

{

indexFrom = selected + 2 - MENU\_CAPACITY + itemsBelow;

}

}

result = parModel.UIItems.GetRange(indexFrom, count);

}

return result;

}

/// <summary>

/// Отрисовка элемента меню.

/// </summary>

/// <param name="parObject">Элемент меню.</param>

/// <param name="parRowNumber">Номер строки для вывода.</param>

private void DrawUIObject(UIObject parObject, int parRowNumber)

{

string text = "";

Brush brush = Brushes.White;

if (parObject.State == 1)

{

brush = Brushes.LawnGreen;

}

if (parObject.Type == UIObjectType.Text)

{

text = parObject.Text;

if ((parRowNumber == 0) && !text.StartsWith("Level "))

{

brush = Brushes.Yellow;

}

}

else

{

text = parObject.Type.ToString().Replace('\_', ' ');

}

int offsetX = (int)((\_width - \_bufGraphics.Graphics.MeasureString(text, \_standartFont).Width) / 2);

int offsetY = (int)(\_height \* MENU\_OFFSET\_Y + parRowNumber \* (\_height \* (1 - MENU\_OFFSET\_Y)) / MENU\_CAPACITY);

\_bufGraphics.Graphics.DrawString(text, \_standartFont, brush, offsetX, offsetY);

}

/// <summary>

/// Вывод кадра из буфера на форму.

/// </summary>

private void ViewCanvas(Graphics parFormGraphics)

{

#if FPSMETER

\_bufGraphics.Graphics.DrawString($"FPS: {\_counter[0] + \_counter[1] + \_counter[2]}", \_standartFont, Brushes.White, 0, 0);

FormPaint();

#endif

\_bufGraphics.Render(parFormGraphics);

}

/// <summary>

/// Реакция на событие изменения размеров формы.

/// </summary>

/// <param name="parFormGraphics">Объект графики для отрисовки готового кадра.</param>

/// <param name="parWidth">Ширина формы.</param>

/// <param name="parHeight">Высота формы.</param>

public void FormSizeChanged(Graphics parFormGraphics, int parWidth, int parHeight)

{

\_width = parWidth;

\_height = parHeight;

\_bufGraphicsContext.MaximumBuffer = new Size(\_width + 1, \_height + 1);

\_bufGraphics = \_bufGraphicsContext.Allocate(parFormGraphics, new Rectangle(0, 0, \_width, \_height));

FindCellSize();

if (\_fontCollection != null)

{

\_standartFont = new Font(\_fontCollection.Families[0], \_width \* FONT\_MULTIPLIER, FontStyle.Regular, GraphicsUnit.Point, 0);

\_headerFont = new Font(\_fontCollection.Families[0], \_width \* FONT\_MULTIPLIER \* 2, FontStyle.Regular, GraphicsUnit.Point, 0);

}

SendCameraSizeToModel();

Thread delayedGC = new Thread(GCcollectWithDelay);

delayedGC.Start(500);

}

/// <summary>

/// Сборка мусора с задержкой.

/// </summary>

/// <param name="parData">Задержка (в миллисекундах).</param>

private void GCcollectWithDelay(object parData)

{

Thread.Sleep((int)parData);

GC.Collect();

}

#if FPSMETER

/// <summary>

/// Реакция на запуск отрисовки готового кадра на форме.

/// </summary>

private void FormPaint()

{

\_counter[3]++;

if ((DateTime.UtcNow - \_time).TotalMilliseconds > 333)

{

\_counter[0] = \_counter[1];

\_counter[1] = \_counter[2];

\_counter[2] = \_counter[3];

\_counter[3] = 0;

\_time = DateTime.UtcNow;

}

}

#endif

}

}