Петрозаводский государственный университет
Физико-технический факультет

Пикалев А. А.

Технологии программирования

Сборник заданий к командным проектам

Петрозаводск 2016

Оглавление

Предисловие

В этой работе вам предстоит спроектировать с помощью Visual Paradigm и написать на C++ небольшую программу. Поскольку наша цель — научиться проектировать объектно-ориентированные приложения с использованием UML, при проектировании мы не всегда будем принимать оптимальные для небольшой задачи решения. Помните, наша задача научиться стрелять из пушки, даже если в качестве мишени выбраны воробьи.

Поскольку время на разработку ограничено, при анализе требований выделим минимальный набор функций и реализуем их в первую очередь. Однако при проектировании будем считать что предполагается длительное развитие проекта, поэтому отметим функции, реализация которых предполагается в перспективе и будем учитывать их при проектировании архитектуры.

Над написанием программы вы будете работать в командах с использованием системы контроля версий. Каждый будет писать свою часть проекта, поэтому важно на этапе проектирования определить интерфейсы классов, чтобы разные члены команды могли использовать классы друг друга.

Пособие подготовлено при поддержке Программы стратегического развития ПетрГУ.

Вариант 1

Векторный графический редактор

1.1 Определение требований к системе

1.1.1 Постановка задачи разработки

Программа должна иметь графический интерфейс. Во время редактирования пользователь должен видеть текущее состояние рисунка ("What you see is what you get"). На первом этапе разработки достаточно реализовать только некоторые графические примитивы, например, прямоугольники, эллипсы и отрезки прямых. В дальнейшем предполагается реализация и других примитивов: ломанных, многоугольников, текста и так далее.

У пользователя должна быть возможность перемещать нарисованные объекты и менять их размеры при помощи мыши. При этом удобно иметь возможность выделять и перемещать сразу несколько объектов.

Фигуры имеющие площадь (прямоугольник, эллипс, ...) имеют определённые цвета контура и заливки, а линии — только цвет контура. Сейчас реализуем только однотонную заливку, но предполагается возможность градиентной и текстурной заливок.

Одна фигура может накладываться на другую, поэтому нужно реализовать возможность переупорядочить фигуры.

Пользователь должен иметь возможность сохранить свою работу и загрузить ранее сохранённый файл.

При дальнейшей разработке планируется реализовать другие возможности векторных графических редакторов: Загрузку и сохранение рисунков в других форматах, группировку фигур, поворот, преобразование одного типа фигур в другой, работа с буфером обмена и так далее.

1.1.2 Глоссарий предметной области

Глоссарий предназначен для описания терминологии области, в которой будет работать ПО. Выделяются термины, им даётся описание, рассчитанное на широкий круг читателей (пользоваться этим описанием будут все лица, заинтересованные в разработке системы). Глоссарий составляется на русском языке. Термины сопровождаются переводом на английский на тот случай, если термин будет использован в модели системы как название класса, пакета и т. п.

Составьте глоссарий для вашего задания. Сравните его с глоссариями остальных членов команды. Одинаково ли вы понимаете терминологию?

1.1.3 Дополнительная спецификация

Требования к реализации

Система должна быть написана на языке C++ и поддерживать работу в операционной системе, установленной в компьютерном классе. Дополнительный бонус за переносимость программы. Программа должна корректно работать при различных разрешениях монитора.

Надёжность

Программа не должна вызывать системных сбоев (crash) или зависать. Программа должна освобождать всю выделенную память. В программе должны отсутствовать конструкций, приводящих к неопределённому поведению.

Эти требования должны выполняться, в том числе, при ошибочных или злонамеренных действиях пользователя, а также при неверных входных данных.

Сохранение и последующая загрузка рисунка не должны приводить к потере данных.

Производительность

Программа должна обеспечивать комфортную работу с рисунком, содержащим до 100 объектов на компьютерах в компьютерном классе, однако максимальное число объектов на рисунке не должно ограничиваться реализацией.

1.1.4 Создание диаграммы вариантов использования

Функциональные требования к системе моделируются и документируются с помощью вариантов использования (use case).

Модель вариантов использования состоит из диаграмм вариантов использования, текстовых описаний вариантов использования и диаграмм деятельности, моделирующих потоки событий вариантов использования.

Для графического редактора единственным действующим лицом будет пользователь.

1. Создайте новый проект в Visual Paradigm, в нём модель **Use Case Model**, а в ней — диаграмму вариантов использования **Main**.

2. Создайте диаграмму использования. Поскольку наш редактор предполагается простым, её можно сделать чуть более подробной, чем необходимо. Результат должен получиться похожим на рисунок 1.1.

1.1.5 Описание вариантов использования

Составим описание варианта использования «Open file», а описание остальных вариантов использования составьте самостоятельно.

Краткое описание

Данный вариант использования описывает открытие пользователем файла.

Основной поток событий

1. Пользователь нажимает на кнопку или выбирает пункт меню «открыть файл».

2. Система закрывает текущий документ.

3. Система выдаёт диалоговое окно открытия файла.

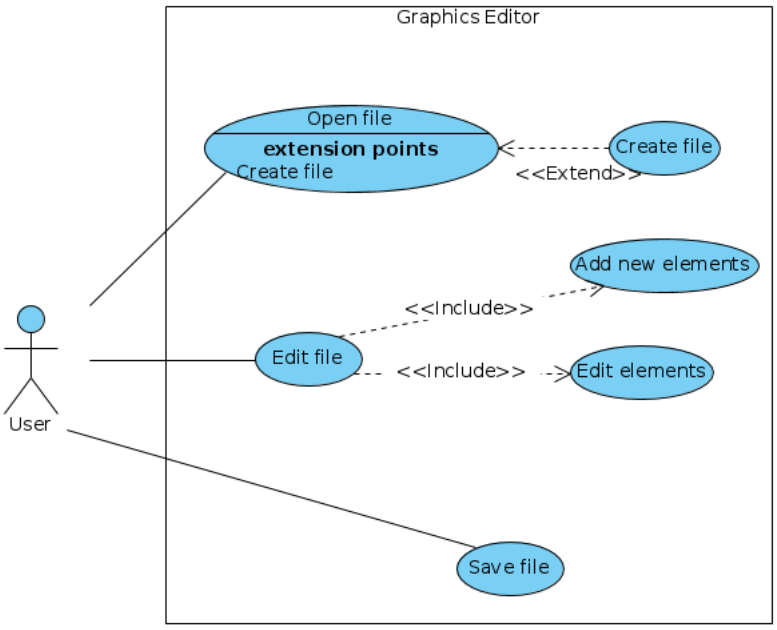

Рис. 1.1: Диаграмма вариантов использования

4. Пользователь выбирает файл

5. Система отображает рисунок из файла на экране и переходит в режим редактирования.

Альтернативные потоки

2А. Пользователь отменяет закрытие документа

1. Система возвращается в режим редактирования исходного документа.

4А. Пользователь отменяет открытие файла

1. Система возвращается в режим редактирования исходного документа (завершение варианта использования).

5А. Невозможно получить доступ к выбранному файлу или файл имеет неверный формат

1. Система выдаёт сообщение пользователю и предлагает ему отменить открытие файла или выбрать другой файл.

2. Пользователь сообщает о своём выборе.

3. Система переходит на пункт 3 основного потока или завершает вариант использования.

Постусловие

При успешном открытии файла в области редактирования отображается рисунок из файла, а у пользователя появляется возможность его редактировать. В случае отмены или невозможности открытия файла остаётся открытым предыдущий рисунок или область редактирования очищается.

Проиллюстрируйте варианты использования диаграммами деятельности.

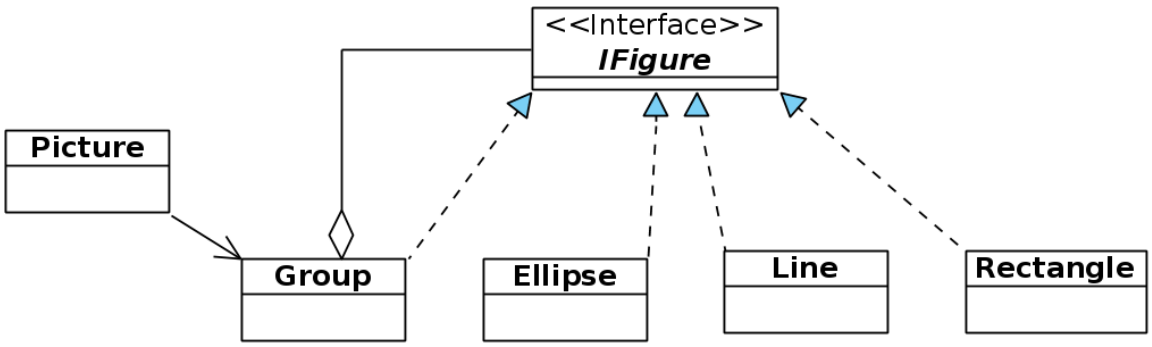

Рис. 1.2: Диаграмма классов **Key Abstractions**

1.2 Анализ системы

Прежде чем приступать к анализу системы примем следующие соглашения:

1. Все имена должны быть на английском языке.
2. Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.
3. Имена классов должны начинаться с заглавной буквы.
4. Имена атрибутов и операций должны начинаться со строчной буквы.
5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.
6. Диаграммы, предназначенные для анализа системы расположим в модели «**Analysis model**»

Создайте модель «**Analysis model**», а в ней диаграмму классов «**Key Abstractions**».

Основной сущностью, с которой будет работать наша программа, является рисунок (Picture). Рисунок содержит некоторое число фигур (Figure). Фигуры могут быть разного вида (прямоугольник (Rectangle), эллипс (Ellipse), линия (Line)). Фигуры можно группировать, введём для группы фигур класс Group. С разными видами фигур и группами фигур удобно работать единообразно, поэтому введём интерфейс IFigure, который они будут реализовывать. Изобразим вышеизложенное на диаграмме классов как показано на рисунке 1.2.

Продумайте, какие атрибуты должны быть у этих классов и внесите их на UML-диаграмму.

1.2.1 Сохранение файла

Проведём анализ варианта использования **Save File**. Для этого в **Analysis Model** создайте пакет **Use Case Realisations**, а в нём диаграмму последовательности **Save File** и диаграмму классов **VOPC Save File**.

Сохранение может быть реализовано по-разному. При выборе варианта нужно учесть необходимость лёгкого добавления новых типов фигур и новых форматов файлов. Первое, что приходит на ум — сделать в каждом классе фигур метод saveXxx(...), где Xxx — формат файла, который бы сохранял фигуру в виде, соответствующем данному формату. Недостаток этого подхода — расширение зоны ответственности классов фигур. При

данном подходе получится, что фигура должна знать об особенностях форматов файлов, поддерживаемых программой.

Мы пойдём другим путём. Поскольку сохранение предполагается в разных форматах, реализуем сохранение в каждый формат в виде отдельного класса (SaverPNG, SaverSVG, SaverMySuperPuperFormat, ...), реализующего интерфейс ISaver. Чтобы избежать огромной функции сохранения, работающей со всеми типами фигур, поступим следующим образом:

- в интерфейсе ISaver введём перегружаемую для каждого типа фигур операцию `save(ConcreteFigure figure)`,
  которая выполняет действия по сохранению данного конкретного типа фигур. Под `ConcreteFigure` понимается какой-нибудь из классов, реализующих интерфейс `IFigure`.
- в интерфейсе `IFigure` введём операцию `save(ISaver saver)`, каждая реализация которой будет вызывать `saver.save(this)`.

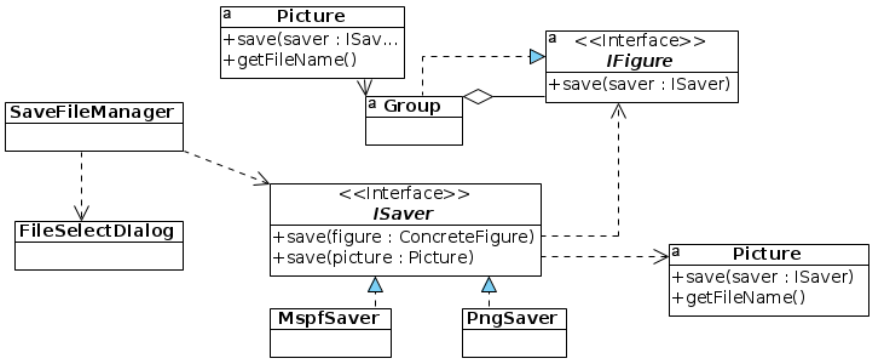

Рис. 1.3: Диаграмма классов **VOPC Save File**

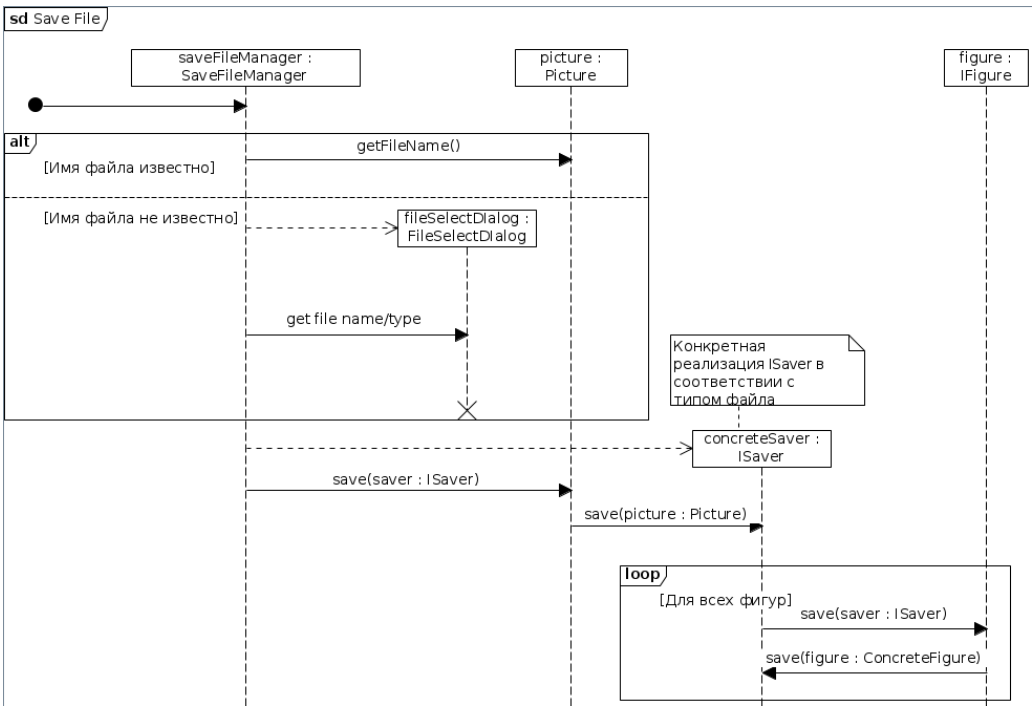

Рис. 1.4: Диаграмма последовательности **Save File**

Диаграмма классов, иллюстрирующая сохранение файла, показана на рисунке 1.3, а диаграмма последовательности — на рисунке 1.4. На диаграмме последовательности показано, что если имя файла рисунка уже известно (например, мы его уже сохраняли или открыли существующий файл), то используется существующее имя, а если имени файла нет (новый файл или пользователь выбрал "Сохранить как ..."), то запрашивает его у пользователя. Далее, в соответствии с типом файла, создаётся saver конкретного типа, который передаётся методу `save` рисунка. Предполагается, что перегруженный для рисунка метод `ISaver.save` будет аналогично вызывать методы `save` для фигур рисунка.

На диаграмме последовательности для наглядности опущен вариант, когда пользователь отменяет сохранение, или когда возникает какая-либо ошибка.

1.2.2 Редактирование файла

Редактирование рисунка может происходить в разных вариантах, имеющих много общего. Если пользователь редактирует фигуры, уже присутствующие на рисунке, то он должен сначала их выделить. Реализуем выделение в виде отдельного класса, содержащего группу выделенных объектов, а также маркеры выделения.

Диаграмма классов, участвующих в выделении, а также диаграмма последовательности, соответствующая обработке клика мыши с точки зрения выделения фигур, показаны на рисунках 1.5 – 1.6.

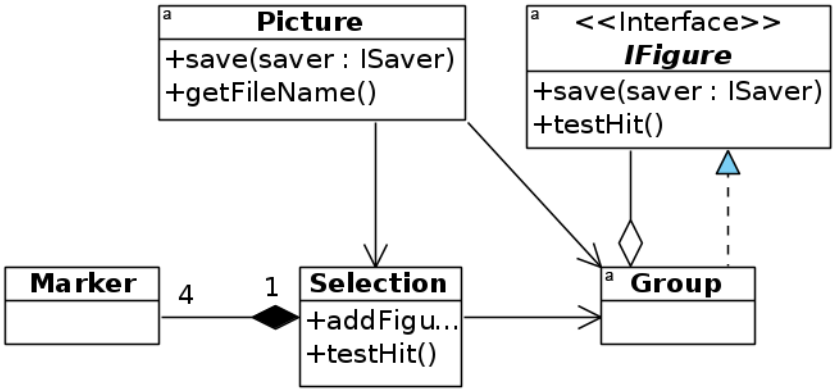

Рис. 1.5: Диаграмма классов **VOPC Select**

На диаграмме последовательности показано, что если выделения не было, а пользователь кликнул по фигуре, то создаётся выделение, в которое добавляется данная фигура. Возможна ситуация, когда пользователь хочет добавить фигуру в уже имеющееся выделение (продумайте, каким образом пользователь будет это делать). Также пользователь может сбросить выделение. Обратите внимание, что проверка попадания клика в фигуру проводится в порядке, обратном порядку отрисовки фигур. Это сделано для того, чтобы при наложении фигур выделялась верхняя.

Добавьте на диаграмму последовательности снятие выделения с отдельной фигуры.

Проанализируйте другие аспекты редактирования.

Теперь, когда каждый член команды проанализировал проект, обсудите работу друг друга. Сравните результаты и подготовьте единую модель анализа, включающую в себя лучшие наработки. Покажите её преподавателю.

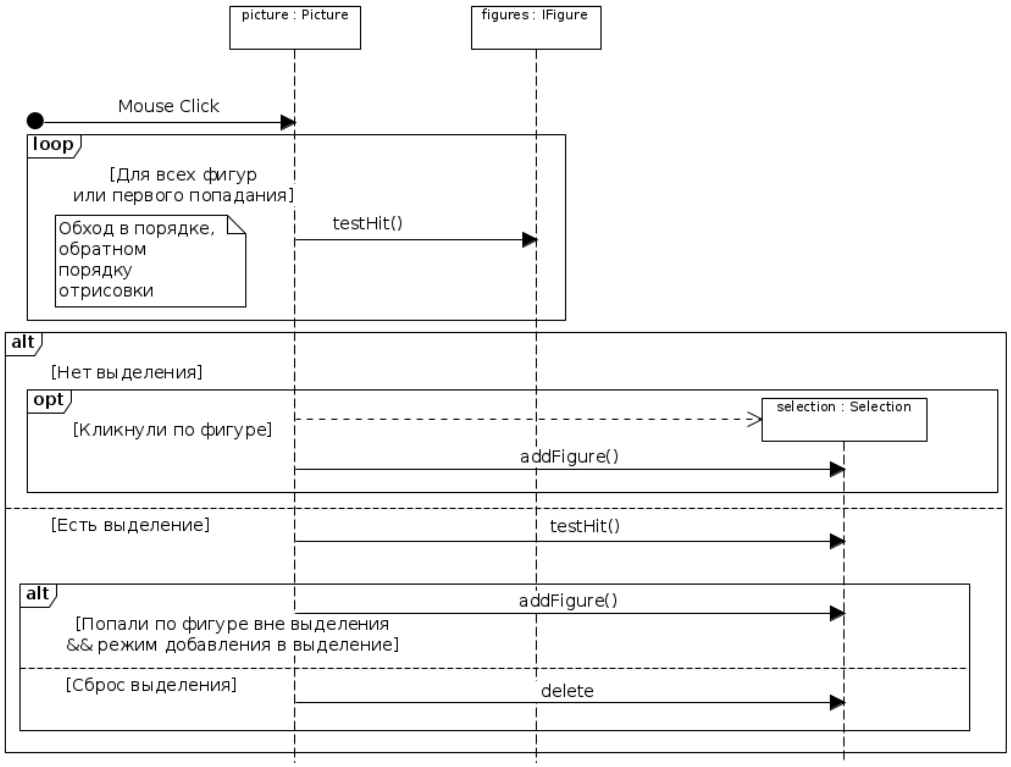

Рис. 1.6: Диаграмма последовательности **Select**

1.3 Проектирование системы

Проектирование системы проводите совместно.

Придумайте / выберите основной формат файлов, с которым будет работать ваша программа. Подготовьте его описание.

Выберите библиотеки, которые будете использовать для написания программы. Исходя из выбранных библиотек определите, какие классы вы будете реализовывать самостоятельно, а какие использовать готовыми.

В проекте создайте **«Design Model»**, а в ней пакеты **«Picture Elements»** (классы, относящиеся непосредственно к рисунку), **«UI»** (классы пользовательского интерфейса), **«File IO»** (открытие и сохранение файлов), а также по пакету для каждой используемой библиотеки.

Рекурсивно продублируйте модель анализа и распределите классы из неё в подходящие пакеты проектной модели, а также перенесите пакет **«Use Case Realization»** целиком. Удалите то, что осталось от дубля и не вошло в проектную модель. Проведите преобразование классов анализа в проектные классы. Уточните поля и методы. Обозначьте используемые типы. Ответьте на такие вопросы как:

- Какие контейнеры использовать для хранения фигур?
- Какой класс будет использован для области рисования? Какие его методы будут задействованы?
- Какие классы принимают участие в процессе открытия/сохранения файла?
- Какие аргументы передаются в каждый из методов, используемый на диаграмме?
- В каких случаях используются сами экземпляры классов, а в каких — указатели и ссылки на них?

Уточните диаграммы последовательности в соответствии с принятыми вами проектными решениями.

Покажите результаты преподавателю и получите от него доступ к репозиторию и дальнейшие указания.

Вариант 2

LaserAge

2.1 Определение требований к системе

2.1.1 Постановка задачи разработки

Игровой экран делится на две части: в нижней перемещается космический корабль игрока, а в верхней — космические корабли противника. Уничтожив всех противников на данном уровне, игрок переходит на следующий уровень. Существует несколько типов кораблей противников, которые отличаются прочностью, поведением, частотой выстрелов, типом снарядов, прицельностью огня. Разные снаряды отличаются скоростью, областью поражения и наносимым уроном. На некоторых уровнях есть специальный контейнер, сбив и поймав который, игрок получает дополнительную жизнь. В зависимости от количества жизней меняется тип оружия игрока.

При дальнейшей разработке планируется добавление новых противников и уровней, статистики, сохранения игры.

2.1.2 Глоссарий предметной области

Глоссарий предназначен для описания терминологии области, в которой будет работать ПО. Выделяются термины, им даётся описание, рассчитанное на широкий круг читателей (пользоваться этим описанием будут все лица, заинтересованные в разработке системы). Термины сопровождаются переводом на английский на тот случай, если термин будет использован в модели системы как название класса, пакета и т. п.

Составьте глоссарий для вашего задания. Сравните его с глоссариями остальных членов команды. Одинаково ли вы понимаете терминологию?

2.1.3 Дополнительная спецификация

Требования к реализации

Система должна быть написана на языке C++ и поддерживать работу в операционной системе, установленной в компьютерном классе. Дополнительный бонус за переносимость программы. Программа должна корректно работать при различных разрешениях монитора. Скорость игры не должны зависеть от производительности компьютера.

Надёжность

Программа не должна вызывать системных сбоев (crash) или зависать. Программа должна освобождать всю выделенную память. В программе должны отсутствовать конструкций,

приводящих к неопределённому поведению.

Эти требования должны выполняться, в том числе, при ошибочных или злонамеренных действиях пользователя.

Производительность

Игра должна иметь производительность не менее 40 кадров в секунду на компьютерах в компьютерном классе. Игра не должна потреблять все доступные ресурсы.

2.1.4 Создание диаграммы состояний

Обычно функциональные требования к системе моделируются и документируются с помощью вариантов использования (use case). В нашем случае имеется единственное действующее лицо — игрок и единственный вариант использования, поэтому нет смысла создавать диаграмму вариантов использования. Вместо этого поясним функционирование системы с помощью диаграммы состояний.

1. Создайте новый проект в Visual Paradigm, в нём модель **Use Case Model**, а в ней — диаграмму состояний **Main**.

2. Создайте диаграмму состояний. Результат должен получиться похожим на рисунок 2.1.

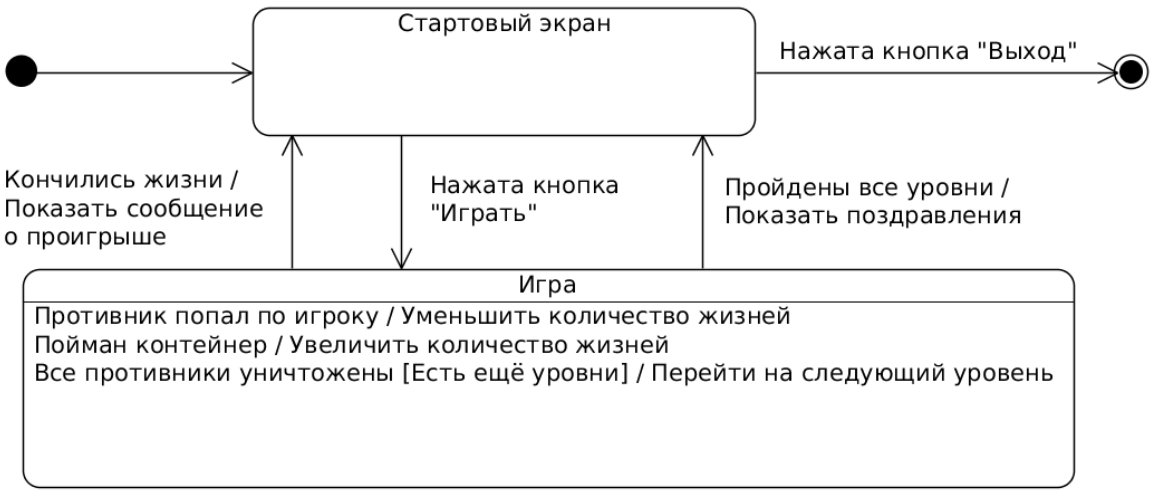

Рис. 2.1: Диаграмма состояний

2.2 Анализ системы

Прежде чем приступать к анализу системы примем следующие соглашения:

1. Все имена должны быть на английском языке.

2. Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.

3. Имена классов должны начинаться с заглавной буквы.

4. Имена атрибутов и операций должны начинаться со строчной буквы.

5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.

6. Диаграммы, предназначенные для анализа системы расположим в модели «**Analysis model**»

Создайте модель «**Analysis model**», а в ней диаграмму классов «**Key Abstractions**».

Для всех объектов, которые могут перемещаться по игровому полю введём абстрактный класс `IGameObject`, имеющий операцию `update(...)`, которая периодически вызывается для каждого объекта и с помощью которой объект реализует своё поведение. Переопределение данной операции позволит реализовать различную тактику врагов, траекторию снарядов и т. д.

Каждый игровой объект изображается в виде некоторой текстуры.

Для всех типов врагов введём интерфейс `IEnemy`, для всех типов пуль — `IBullet`.

Продумайте, какие типы врагов и пуль стоить реализовать в игре.

Изобразим вышеизложенное на диаграмме классов как показано на рисунке 2.2.

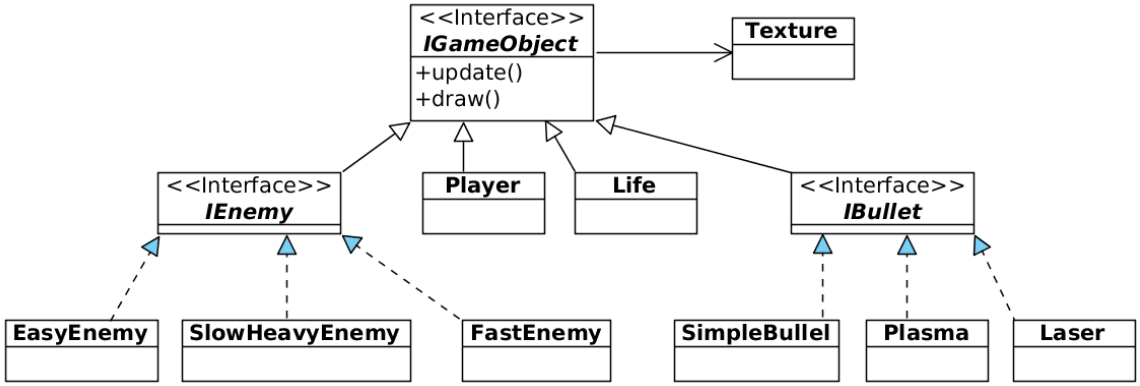

Рис. 2.2: Диаграмма классов **Key Abstractions**

Продумайте, какие атрибуты должны быть у этих классов и внесите их на UML-диаграмму.

2.2.1 Расчёт следующего кадра

Для каждого кадра игры нужно вычислить новые положения игровых объектов и их столкновения. Если за время между кадрами игровые объекты смещаются мало, то можно проводить расчёты с временным шагом равным длительности кадра. Если скорости объектов велики, то это может привести к тому, что не все столкновения будут учтены. Тогда необходимо выполнять расчёт несколько раз для каждого кадра.

На каждом шаге нужно вычислить:

- новые положения игрока, каждой пули, каждого противника и дополнительной жизни;
- столкновения игрока с пулями противника и противников с пулями игрока (считаем, что игрок не может быть поражён своим выстрелом, а противники — выстрелами своей стороны);
- столкновение жизни с пулями игрока (если жизнь ещё не подбита) или столкновение жизни с игроком, если жизнь уже подбита;

- удаление пуль и подбитой жизни, если они улетели за пределы игрового поля.

Для управления этим процессом введём класс `GameManager`. Изобразим расчёт кадра на диаграмме последовательности. Часть диаграммы показана на рисунке 2.3, недостающее изобразите самостоятельно.

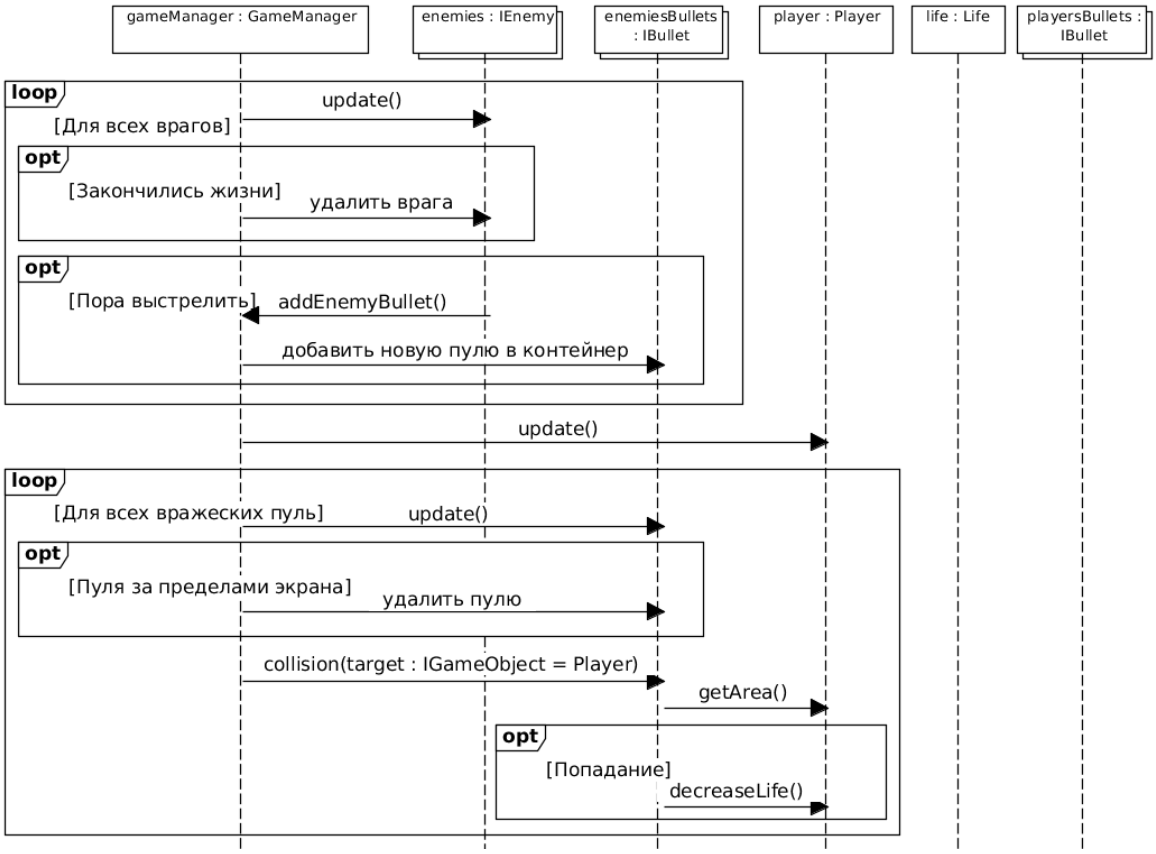

Рис. 2.3: Диаграмма последовательности расчёта нового кадра

2.2.2 Стартовый экран

Помимо самой игры у нас есть стартовый экран, а при дальнейшем развитии потребуется экран настроек. Для взаимодействия с пользователем нам потребуются кнопки. Введём класс `Button`, а для каждой конкретной кнопки создадим по классу-наследнику, в котором переопределим метод `mouseButtonReleased`, реализующий поведение кнопки при отпускании клавиши мыши (см. рисунок 2.4).

Есть два варианта отображения кнопок на экране

1. рисовать каждую кнопку программно;

2. отображать заранее подготовленную картинку.

Первый вариант идеален для приложений, предназначенных для работы, в которых большое число кнопок, которые должны отображаться стандартным образом в соответствии с настройками операционной системы.

Для игры больше подойдёт второй вариант, так как кнопок мало, а у дизайнеров должны быть развязаны руки, поэтому именно его мы реализуем. У кнопки может быть до 4 состояний: обычное, подсвеченное (на кнопку наведён курсор), нажатое и недоступное. Поэтому покажем на диаграмме ассоциацию с четырьмя текстурами.

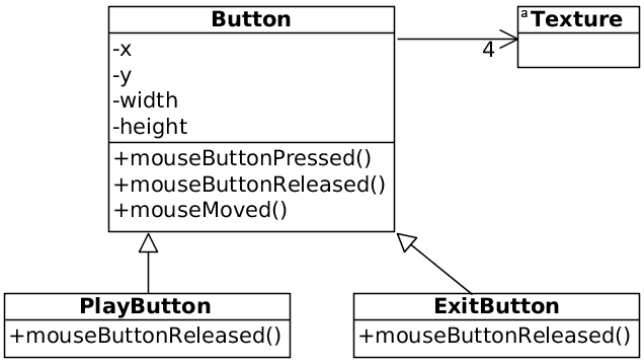

Рис. 2.4: Кнопки

Теперь, когда каждый член команды проанализировал проект, обсудите работу друг друга. Сравните результаты и подготовьте единую модель анализа, включающую в себя лучшие наработки. Покажите её преподавателю.

2.3 Проектирование системы

Проектирование системы проводите совместно.

Выберите библиотеки, которые будете использовать для написания программы (рекомендуется использовать SFML). Исходя из выбранных библиотек определите, какие классы вы будете реализовывать самостоятельно, а какие использовать готовыми.

В проекте создайте **Design Model**, а в ней пакеты **Game**, **Start screen** и **Services**, а также по пакету для каждой используемой библиотеки. В пакет **Services** будем складывать служебные элементы.

Рекурсивно продублируйте модель анализа и распределите классы из неё в подходящие пакеты проектной модели, а также перенесите пакет **Use Case Realization** целиком. Удалите то, что осталось от дубля и не вошло в проектную модель.

Как видно из анализа выше, многие классы в игре имеют ассоциации с классом текстур. На этапе проектирования необходимо решить, как хранить эти текстуры и как их загружать. При этом нужно учесть следующие особенности:

- текстуры занимают большой объём памяти;
- загрузка текстуры с диска — длительная операция;
- многие экземпляры классов имеют одинаковые текстуры;
- организация хранения и загрузки текстур выходит за рамки ответственности классов-сущностей.

Поэтому для работы с текстурами введём класс `TextureManager`, имеющий метод `Texture* TextureManager::get(const char *name)`

Обратите внимание, что этот класс должен кэшировать текстуры, чтобы не загружать их повторно при новом обращении. Кроме того, метод `get` должен быть доступен из многих мест программы. Поэтому нужно или сделать глобальный экземпляр класса, или класс со статическими методами, или синглтон.

Обращение за текстурой можно проиллюстрировать диаграммой последовательности (см. рис. 2.5).

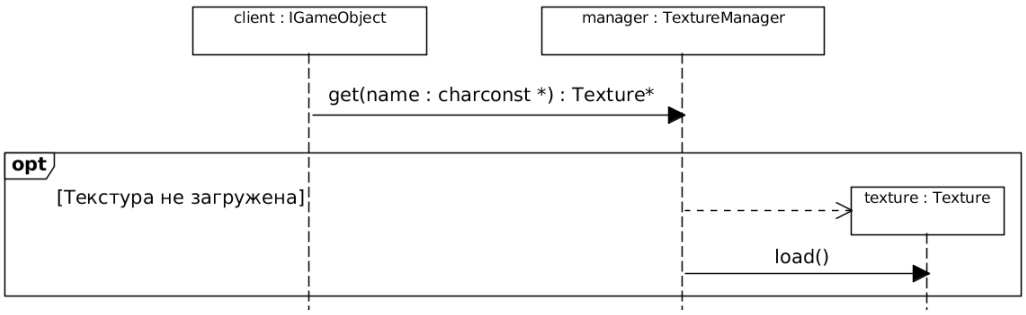

Рис. 2.5: Обращение за текстурой

Ещё один важный вопрос — как осуществлять переход с уровня на уровень. В требованиях указано, что необходимо обеспечить лёгкость добавления новых уровней. Уровни отличаются составом и начальными положениями противников, наличием или отсутствием контейнера с жизнью, фоновой текстурой. Уровни можно или жёстко задать в программе, или загружать из конфигурационных файлов. Загрузка сохранённой игры во многом аналогична переходу на новый уровень.

Для конфигурирования уровня введём класс `LevelBuilder`, имеющий метод `nextLevel(GameManager &game)`, преобразующий игру в соответствии с новым уровнем.

Изобразите переход на новый уровень в виде диаграммы последовательности.

Проведите преобразование классов анализа в проектные классы. Уточните поля и методы. Обозначьте используемые типы. Ответьте на такие вопросы как:

- Какие контейнеры использовать для игровых объектов?
- Какие аргументы передаются в каждый из методов, используемый на диаграмме?
- В каких случаях используются сами экземпляры классов, а в каких — указатели и ссылки на них?
- Каким образом менеджеру игры передаётся пуля, которую нужно добавить в контейнер?
- Как обрабатывается нажатие клавиши мыши для стрельбы или нажатия кнопки?

Уточните диаграммы последовательности в соответствии с принятыми вами проектными решениями.

Покажите результаты преподавателю и получите от него доступ к репозиторию и дальнейшие указания.

Вариант 3

Калькулятор

3.1 Определение требований к системе

3.1.1 Постановка задачи разработки

Калькулятор должен иметь 2 режима работы: консольный и графический. В консольном режиме калькулятор считывает выражения со стандартного ввода и вычисляет их, пока не получит команду `exit` или не достигнет конца файла. Это позволит работать с калькулятором не только в интерактивном режиме, но и вычислять выражения из текстового файла, выполнив перенаправление ввода.

В графическом режиме пользователь формирует вычисляемое выражение с помощью кнопок, а нажатие кнопки «=» приводит к вычислению текущего выражения. **Запрещено использовать библиотеки с готовыми элементами графического интерфейса (Qt, GTK, .net, ...).**

Выбор режима можно осуществить несколькими способами:

- режим определяется параметрами командной строки;
- основная программа имеет консольный режим работы, а графический интерфейс запускает её для вычислений;
- консольный и графический интерфейсы реализуются независимыми программами, а общий код реализован в виде разделяемой библиотеки.

Для простоты остановимся на первом варианте.

На первом этапе разработки программа должна вычислять выражения, содержащие сложение, вычитание, умножение, деление, квадратный корень. При дальнейшей разработке потребуется добавить возведение в произвольную степень, тригонометрические функции, логарифмы и т. д.. Также потребуется работа с памятью, а в консольном режиме — с переменными.

3.1.2 Глоссарий предметной области

Глоссарий предназначен для описания терминологии области, в которой будет работать ПО. Выделяются термины, им даётся описание, рассчитанное на широкий круг читателей (пользоваться этим описанием будут все лица, заинтересованные в разработке системы). Термины сопровождаются переводом на английский на тот случай, если термин будет использован в модели системы как название класса, пакета и т. п.

Составьте глоссарий для вашего задания. Сравните его с глоссариями остальных членов команды. Одинаково ли вы понимаете терминологию?

3.1.3 Дополнительная спецификация

Требования к реализации

Система должна быть написана на языке C++ и поддерживать работу в операционной системе, установленной в компьютерном классе. Дополнительный бонус за переносимость программы. Программа должна корректно работать при различных разрешениях монитора. Программа не должна использовать библиотеки с готовыми элементами графического интерфейса.

Надёжность

Программа не должна вызывать системных сбоев (crash) или зависать. Программа должна освобождать всю выделенную память. В программе должны отсутствовать конструкций, приводящих к неопределённому поведению.

Эти требования должны выполняться, в том числе, при ошибочных или злонамеренных действиях пользователя.

Производительность

При работе в консольном режиме длина входного выражения не ограничена.

3.1.4 Создание диаграммы вариантов использования

Модель вариантов использования состоит из диаграмм вариантов использования, текстовых описаний вариантов использования и диаграмм деятельности, моделирующих потоки событий вариантов использования.

1. Создайте новый проект в Visual Paradigm, в нём модель **Use Case Model**, а в ней - диаграмму состояний **Main**.

2. Создайте диаграмму вариантов использования. Результат должен получиться похожим на рисунок 3.1.

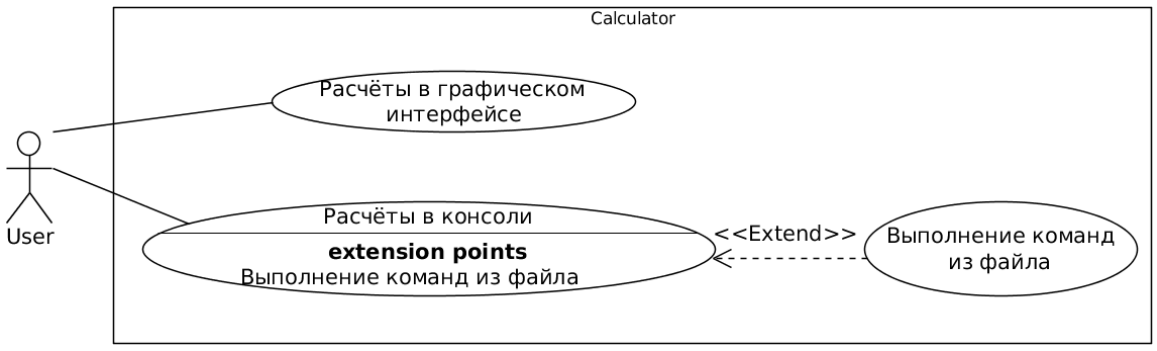

Рис. 3.1: Диаграмма вариантов использования

3.1.5 Описание вариантов использования

Составим описание варианта использования «Расчёты в консоли», а описание остальных вариантов использования составьте самостоятельно.

Краткое описание

Данный вариант использования описывает консольный режим работы калькулятора.

Основной поток событий

1. Пользователь запускает программу.
2. Система выдаёт на экран приглашение на ввод выражения.
3. Пользователь вводит выражение.
4. Система вычисляет выражение и выводит результат на экран.
5. Переход на шаг 3.

Альтернативные потоки

3А. Пользователь вводит не выражение, а команду

1. Система выполняет команду.
2. Если выполнение команды не привело к завершению программы, вернуться на шаг 3 основного потока.

4А. Невозможно вычислить выражение

1. Система выдаёт на экран сообщение об ошибке и возвращается на шаг 3 основного потока.

Постусловие

Значения, выводимые программой должны быть корректным результатом вычисления выражения, если это невозможно, должно быть выведено сообщение об ошибке.

Составьте список команд, которые должна выполнять программа. Составьте примеры выражений, которые должна вычислять программа после завершения первого этапа разработки. В каких ситуациях невозможно вычислить выражение? Проиллюстрируйте варианты использования диаграммами деятельности.

3.2 Анализ системы

Прежде чем приступать к анализу системы примем следующие соглашения:

1. Все имена должны быть на английском языке.
2. Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.
3. Имена классов должны начинаться с заглавной буквы.
4. Имена атрибутов и операций должны начинаться со строчной буквы.
5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.
6. Диаграммы, предназначенные для анализа системы расположим в модели «**Analysis model**»

Создайте модель «**Analysis model**».

Задача распадается на две слабо связанные части:

1. вычислительное ядро;
2. графический интерфейс.

При этом пользовательский интерфейс зависит от ядра, но не наоборот.

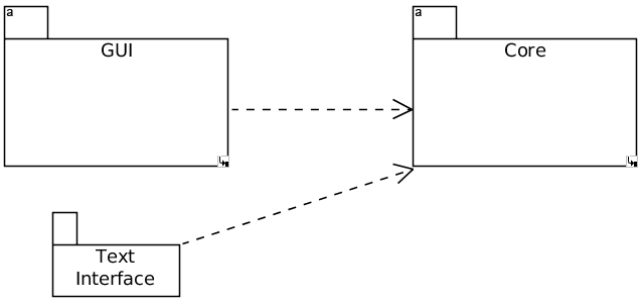

Рис. 3.2: Общая структура программы

3.2.1 Вычислительное ядро

Вычислительное ядро получает от пользовательского интерфейса выражение в виде потока байт. Его задача состоит в том, чтобы выбелить в этом потоке отдельные элементы выражения и провести вычисления с учётом приоритетов операций. Обе части задачи нетривиальны, поэтому поручим их разным классам.

Класс `Tokenizer` выделяет из входного потока отдельные элементы — токены. В нашем случае, токенами будут отдельные числа, имена функций и команд, знаки операций. Затем токены интерпретируются классом `Calculator`, который производит вычисления.

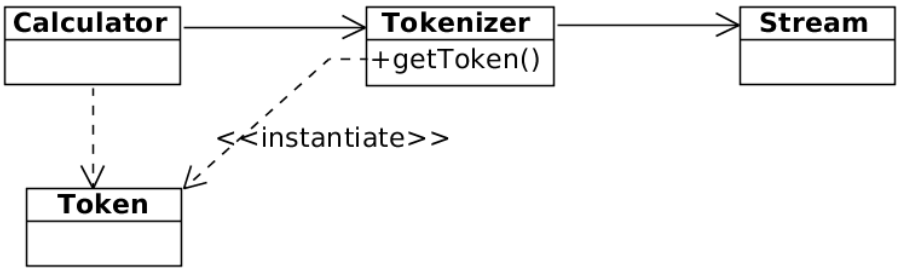

Рис. 3.3: Вычислительное ядро

3.2.2 Графический интерфейс пользователя

Так как по условию задачи запрещено пользоваться библиотеками элементов графического интерфейса, необходимо сделать их самостоятельно. Для единообразной работы с элементами интерфейса введём абстрактный класс `Widget`, в котором объявлены операции `draw()` (отображает виджет) и методы, реагирующие на события, например, на нажатие клавиши. Реакцией на нажатие клавиши может быть проверка, произведён ли клик внутри данного виджета, и если да, то выполнение некоторого действия. В основном цикле при возникновении какого либо события проходим список виджетов и вызываем для каждого из них соответствующую функцию.

Есть два варианта отображения кнопок на экране

1. отображать заранее подготовленную картинку;

2. рисовать каждую кнопку программно.

Первый вариант подходит для игр, где кнопок мало, а у дизайнеров должны быть развязаны руки. Мы воспользуемся вторым вариантом, так как он хорошо масштабируется, а при необходимости позволяет менять оформление под текущую тему операционной системы.

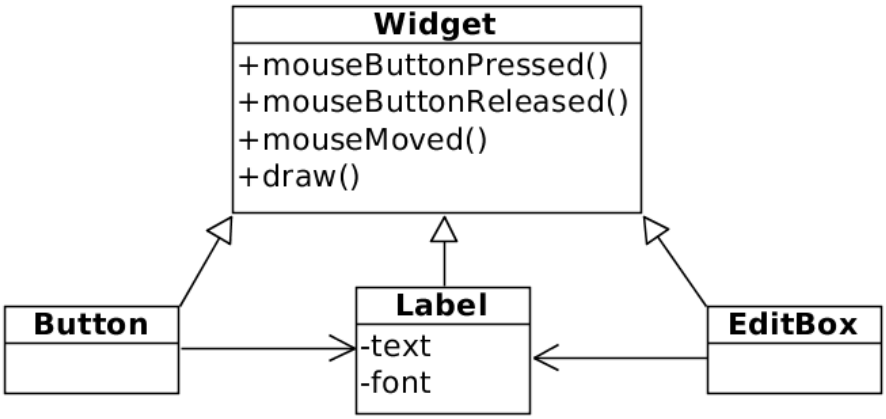

Рис. 3.4: Элементы графического интерфейса

Теперь, когда каждый член команды проанализировал проект, обсудите работу друг друга. Сравните результаты и подготовьте единую модель анализа, включающую в себя лучшие наработки. Покажите её преподавателю.

3.3 Проектирование системы

Проектирование системы проводите совместно.

Выберите библиотеки, которые будете использовать для написания программы. Исходя из выбранных библиотек определите, какие классы вы будете реализовывать самостоятельно, а какие использовать готовыми.

В проекте создайте **Design Model**, а в ней пакеты **Core** и **GUI**, а также по пакету для каждой используемой библиотеки.

Рекурсивно продублируйте модель анализа и распределите классы из неё в подходящие пакеты проектной модели. Удалите то, что осталось от дубля и не вошло в проектную модель.

Проведите преобразование классов анализа в проектные классы. Уточните поля и методы. Обозначьте используемые типы.

В программе используются токены разных типов. Определитесь, какие типы токенов будут реализованы и как они будут различаться. При необходимости, используйте для каждого типа токена свой подкласс.

Составьте диаграммы последовательности для работы графического интерфейса.

Ответьте на такие вопросы как:

- Каким образом организовано хранение элементов графического интерфейса?
- Как происходит обработка событий?
- Каким образом передаётся токен от токенайзера вычислителю?

Уточните диаграммы последовательности в соответствии с принятыми вами проектными решениями.

Покажите результаты преподавателю и получите от него доступ к репозиторию и дальнейшие указания.

Вариант 4

САПР

4.1 Определение требований к системе

4.1.1 Постановка задачи разработки

Программа должна иметь графический интерфейс. Во время редактирования пользователь должен видеть текущее состояние схемы («What you see is what you get»). Дополнительная возможность — масштабирование представления схемы.

На первом этапе разработки достаточно реализовать только некоторые условные графические изображения, например, резисторы, конденсаторы, npn-транзисторы. В дальнейшем предполагается реализация и других радиоэлементов, в том числе, хранящихся в пользовательских библиотеках, а также редактора, позволяющего создавать эти библиотеки.

У пользователя должна быть возможность перемещать элементы при помощи мыши, поворачивать их на прямой угол, соединять проводами. При этом удобно иметь возможность выделять и перемещать сразу несколько объектов. При дальнейшей разработке потребуется реализовать зеркальное отражение элементов.

Пользователь должен иметь возможность сохранить свою работу и загрузить ранее сохранённый файл. Дополнительная возможность — сохранение схемы в виде растрового рисунка.

При дальнейшей разработке планируется реализовать другие возможности радиоэлектронных САПРов, например, анализ схем на ошибки.

4.1.2 Глоссарий предметной области

Глоссарий предназначен для описания терминологии области, в которой будет работать ПО. Выделяются термины, им даётся описание, рассчитанное на широкий круг читателей (пользоваться этим описанием будут все лица, заинтересованные в разработке системы). Глоссарий составляется на русском языке. Термины сопровождаются переводом на английский на тот случай, если термин будет использован в модели системы как название класса, пакета и т. п.

Составьте глоссарий для вашего задания. Сравните его с глоссариями остальных членов команды. Одинаково ли вы понимаете терминологию?

4.1.3 Дополнительная спецификация

Требования к реализации

Система должна быть написана на языке C++ и поддерживать работу в операционной системе, установленной в компьютерном классе. Дополнительный бонус за переносимость программы. Программа должна корректно работать при различных разрешениях монитора.

Надёжность

Программа не должна вызывать системных сбоев (crash) или зависать. Программа должна освобождать всю выделенную память. В программе должны отсутствовать конструкций, приводящих к неопределённому поведению.

Эти требования должны выполняться, в том числе, при ошибочных или злонамеренных действиях пользователя, а также при неверных входных данных.

Сохранение и последующая загрузка схемы не должны приводить к потере данных.

Производительность

Программа должна обеспечивать комфортную работу со схемой, содержащей до 100 объектов на компьютерах в компьютерном классе, однако максимальное число объектов на схеме не должно ограничиваться реализацией.

4.1.4 Создание диаграммы вариантов использования

Функциональные требования к системе моделируются и документируются с помощью вариантов использования (use case).

Модель вариантов использования состоит из диаграмм вариантов использования, текстовых описаний вариантов использования и диаграмм деятельности, моделирующих потоки событий вариантов использования.

Для САПРа единственным действующим лицом будет пользователь.

1. Создайте новый проект в Visual Paradigm, в нём модель **Use Case Model**, а в ней — диаграмму вариантов использования **Main**.

2. Создайте диаграмму использования. Поскольку наш САПР предполагается простым, её можно сделать чуть более подробной, чем необходимо. Результат должен получиться похожим на рисунок 4.1.

4.1.5 Описание вариантов использования

Составим описание варианта использования «Open file», а описание остальных вариантов использования составьте самостоятельно.

Краткое описание

Данный вариант использования описывает открытие пользователем файла.

Основной поток событий

1. Пользователь нажимает на кнопку или выбирает пункт меню «открыть файл».

2. Система закрывает текущий документ.

3. Система выдаёт диалоговое окно открытия файла.

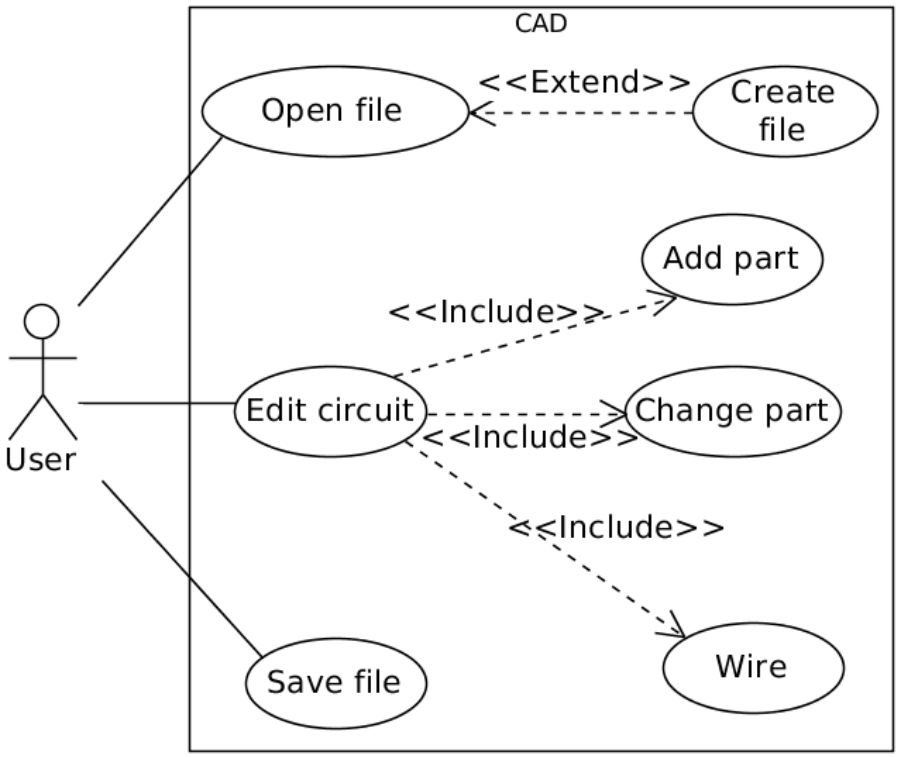

Рис. 4.1: Диаграмма вариантов использования

4. Пользователь выбирает файл

5. Система отображает схему из файла на экране и переходит в режим редактирования.

Альтернативные потоки

2А. Пользователь отменяет закрытие документа

1. Система возвращается в режим редактирования исходного документа.

4А. Пользователь отменяет открытие файла

1. Система возвращается в режим редактирования исходного документа (завершение варианта использования).

5А. Невозможно получить доступ к выбранному файлу или файл имеет неверный формат

1. Система выдаёт сообщение пользователю и предлагает ему отменить открытие файла или выбрать другой файл.

2. Пользователь сообщает о своём выборе.

3. Система переходит на пункт 3 основного потока или завершает вариант использования.

Постусловие

При успешном открытии файла в области редактирования отображается схема из файла, а у пользователя появляется возможность её редактировать. В случае отмены или невозможности открытия файла остаётся открытым предыдущая схема или область редактирования очищается.

Проиллюстрируйте варианты использования диаграммами деятельности.

4.2 Анализ системы

Прежде чем приступать к анализу системы примем следующие соглашения:

1. Все имена должны быть на английском языке.
2. Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.
3. Имена классов должны начинаться с заглавной буквы.
4. Имена атрибутов и операций должны начинаться со строчной буквы.
5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.
6. Диаграммы, предназначенные для анализа системы расположим в модели «**Analysis model**»

Создайте модель «**Analysis model**», а в ней диаграмму классов «**Key Abstractions**».

Основной сущностью, с которой будет работать наша программа, является схема (Circuit). Схема состоит из радиоэлементов и проводов, которые будем представлять в программе с помощью классов Element и Wire соответственно. Возникает вопрос, как реализовать разные типы элементов. Если использовать наследование, то добавление каждого нового элемента потребует создания нового класса и пересборки программы, что потребует наличия сотен классов, прежде чем наш САПР будет применим для решения хоть сколько-нибудь широкого круга задач, и сделает невозможным создание новых элементов пользователем. Поэтому для различных элементов будем использовать единый класс Element.

Каждый элемент имеет условное графическое изображение, поэтому каждый экземпляр должен хранить указатель/ссылку на изображение, соответствующее его типу. Каждый элемент имеет какие-то характеристики. Проблема в том, что разные элементы имеют разный набор характеристик. Например, резистор имеет сопротивление, однако для другого типа резисторов помимо сопротивления может быть указана максимальная мощность. Для конденсатора указывают ёмкость, но для конкретного типа конденсаторов может потребоваться указать номинальное напряжение, ток утечки и т.д. Поэтому жёстко задавать количество характеристик в классе Element неправильно. Введём класс ElementAttribute, содержащий имя и значение характеристики, в классе Element будем хранить список характеристик.

Аналогичная ситуация с контактами. У разных элементов разное число контактов, поэтому введём класс Pin и будем хранить в классе Element список контактов.

Для представления проводов создадим класс Wire, представляющий собой пару контактов(Pin) и изображаемый прямой линией. Для изгиба проводов и соединения между проводами должна быть возможность создавать новые Pin-ы внутри проводов. Возможно потребуется ввести подкласс класса Pin для представления контактов только между проводами.

Продумайте, какие атрибуты должны быть у этих классов и внесите их на UML-диаграмму.

4.2.1 Добавление элемента на схему

Все радиоэлементы в программе являются экземплярами класса Element, однако очень сильно отличаются между собой. Информация конкретных типах элементов (условное

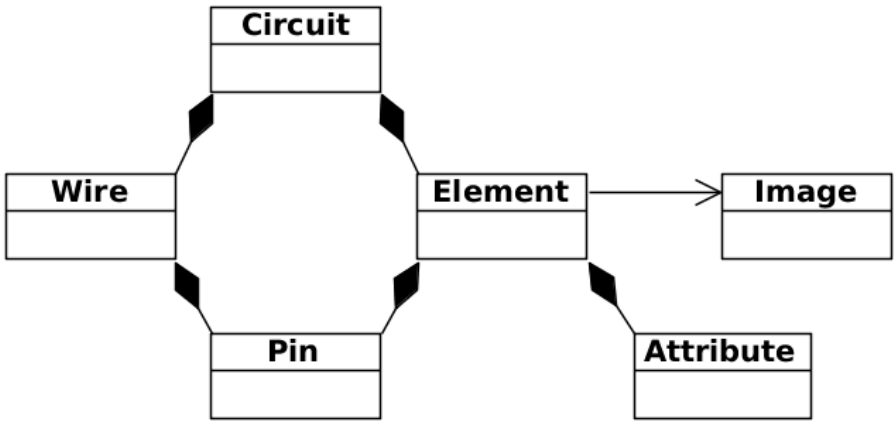

Рис. 4.2: Диаграмма классов **Key Abstractions**

графическое изображение, контакты, характеристики, ...) хранится в библиотеке элементов. Создание нужных элементов поручим отдельным классам, реализующим интерфейс AbstractLibrary. Для создания встроенных элементов создадим класс InternalLibrary, а когда потребуется реализовать работу с пользовательскими элементами, будет создан класс FileLibrary. Интерфейс AbstractLibrary должен требовать операции:
`createElement(typeName)` — создание элемента типа typeName;
`getElementsList()` — получить список элементов, доступных в данной библиотеке. Создание нового элемента показано на диаграммах 4.3 – 4.4.

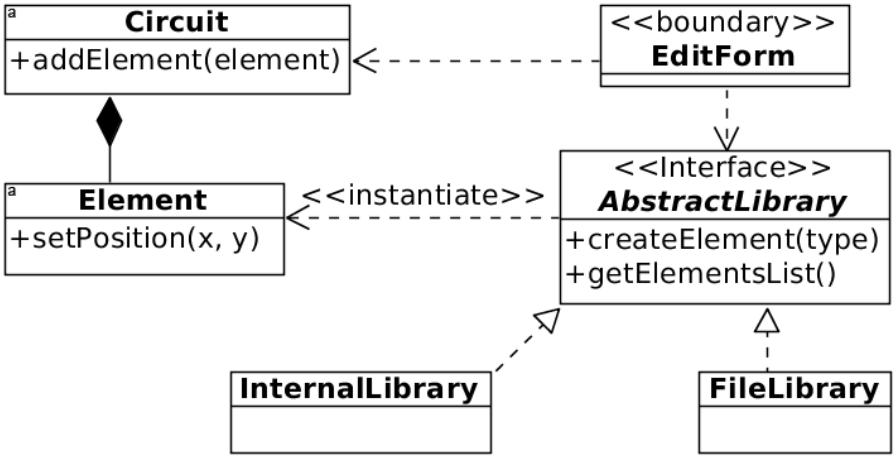

Рис. 4.3: Диаграмма классов **VOPC Add Element**

4.2.2 Изменение существующего элемента

Если пользователь редактирует (перемещает, поворачивает, меняет свойства, ...) элементы / провода, уже присутствующие на рисунке, то он должен сначала их выделить.

Есть несколько вариантов реализации выделения:

- Можно добавить в классы `Wire` и `Element` флаг, определяющий, выделен ли провод или элемент. В зависимости от наличия данного флага рисовать объекты разными цветами.
- Ввести специальный класс, содержащий списки выделенных элементов и проводов. При рисовании окружать выделенные объекты маркерами.

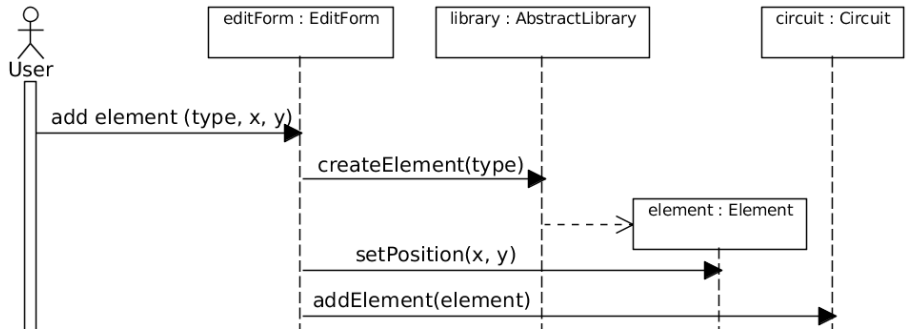

Рис. 4.4: Диаграмма последовательности **Add Element**

В зависимости от описания варианта Change part выберите вариант реализации выделения

На первом этапе реализуем выделение одного объекта, а выделение группы объектов оставим на дальнейшие этапы разработки.

Для того, чтобы узнать какой элемент нужно выделить, программа должна обрабатывать событие клика мыши. Если клик произошёл в момент, когда программа готова к выделению элемента, и пришёлся на область схемы, то форма редактирования передаёт его классу `Circuit`, который опрашивает все элементы и провода, попали ли в них координаты клика. Если да, то опрос завершаем, а элемент или провод делаем выделенным. Опрос необходимо проводить в порядке, обратном порядку отрисовки.

Изобразите выделение с помощью диаграммы последовательности.

Проанализируйте другие аспекты редактирования и сохранение файла.

Теперь, когда каждый член команды проанализировал проект, обсудите работу друг друга. Сравните результаты и подготовьте единую модель анализа, включающую в себя лучшие наработки. Покажите её преподавателю.

4.3 Проектирование системы

Проектирование системы проводите совместно.

Придумайте / выберите основной формат файлов, с которым будет работать ваша программа. Подготовьте его описание.

Выберите библиотеки, которые будете использовать для написания программы. Исходя из выбранных библиотек определите, какие классы вы будете реализовывать самостоятельно, а какие использовать готовыми.

В проекте создайте **«Design Model»**, а в ней пакеты **« Circuit»** (классы, относящиеся непосредственно к схеме), **«UI»** (классы пользовательского интерфейса), а также по пакету для каждой используемой библиотеки.

Рекурсивно продублируйте модель анализа и распределите классы из неё в подходящие пакеты проектной модели, а также перенесите пакет **«Use Case Realization»** целиком. Удалите то, что осталось от дубля и не вошло в проектную модель. Проведите преобразование классов анализа в проектные классы. Уточните поля и методы. Обозначьте используемые типы. Ответьте на такие вопросы как:

- Какие контейнеры использовать для хранения элементов, проводов, атрибутов, контактов?
- Какой класс будет использован для области рисования? Какие его методы будут задействованы?

- Какие классы принимают участие в процессе открытия/сохранения файла?
- Какие аргументы передаются в каждый из методов, используемый на диаграмме?
- В каких случаях используются сами экземпляры классов, а в каких — указатели и ссылки на них?

Уточните диаграммы последовательности в соответствии с принятыми вами проектными решениями.

Покажите результаты преподавателю и получите от него доступ к репозиторию и дальнейшие указания.

Вариант 5

Танки

5.1 Определение требований к системе

5.1.1 Постановка задачи разработки

В игре разворачиваются танковые бои в условиях города. Ваш танк и танки противника находятся в лабиринте. Цель игры — уничтожить бронетехнику противника до того, как они уничтожат вашу крепость и герб города. При попадании в вас, вы теряете часть жизни. Стены разрушаются при попадании снаряда со второго попадания. В лабиринте случайным образом могут появляться ремкомплекты — дополнительные жизни. При завершении игра должна выводить время игры и количество подбитой вражеской техники.

В следующих версиях игры планируется:

- игра с человеком (по сети или за одной клавиатурой)
- разные виды техники и снарядов;
- несколько игровых уровней;
- стрельба и движение под углом.

5.1.2 Глоссарий предметной области

Глоссарий предназначен для описания терминологии области, в которой будет работать ПО. Выделяются термины, им даётся описание, рассчитанное на широкий круг читателей (пользоваться этим описанием будут все лица, заинтересованные в разработке системы). Термины сопровождаются переводом на английский на тот случай, если термин будет использован в модели системы как название класса, пакета и т. п.

Составьте глоссарий для вашего задания. Сравните его с глоссариями остальных членов команды. Одинаково ли вы понимаете терминологию?

5.1.3 Дополнительная спецификация

Требования к реализации

Система должна быть написана на языке C++ и поддерживать работу в операционной системе, установленной в компьютерном классе. Дополнительный бонус за переносимость программы. Программа должна корректно работать при различных разрешениях монитора. Скорость игры не должны зависеть от производительности компьютера.

Надёжность

Программа не должна вызывать системных сбоев (crash) или зависать. Программа должна освобождать всю выделенную память. В программе должны отсутствовать конструкций, приводящих к неопределённому поведению.

Эти требования должны выполняться, в том числе, при ошибочных или злонамеренных действиях пользователя.

Производительность

Игра должна иметь производительность не менее 40 кадров в секунду на компьютерах в компьютерном классе. Игра не должна потреблять все доступные ресурсы.

5.1.4 Создание диаграммы вариантов использования

Функциональные требования к системе моделируются и документируются с помощью вариантов использования (use case). Изобразим на диаграмме вариантов использования различные варианты игры.

1. Создайте новый проект в Visual Paradigm, в нём модель **Use Case Model**, а в ней - диаграмму вариантов использования **Main**.

2. Создайте диаграмму вариантов использования. Результат должен получиться похожим на рисунок 5.1.

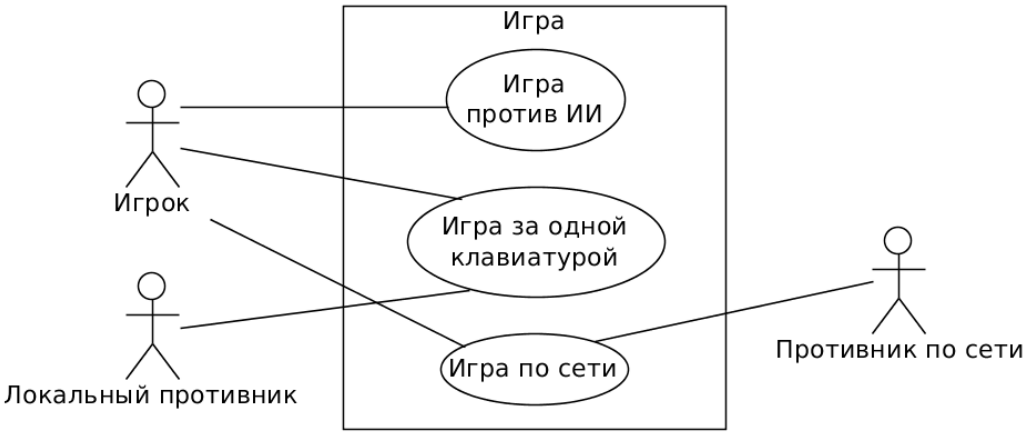

Рис. 5.1: Диаграмма вариантов использования

Опишите, как будет происходить игра. Проиллюстрируйте вариант использования «Игра против искусственного интеллекта» с помощью диаграммы деятельности.

5.2 Анализ системы

Прежде чем приступать к анализу системы примем следующие соглашения:

1. Все имена должны быть на английском языке.

2. Имена классов должны быть существительными, соответствующими, по возможности, понятиям предметной области.

3. Имена классов должны начинаться с заглавной буквы.

4. Имена атрибутов и операций должны начинаться со строчной буквы.

5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.

6. Диаграммы, предназначенные для анализа системы расположим в модели «**Analysis model**»

Создайте модель «**Analysis model**», а в ней диаграмму классов «**Key Abstractions**».

Для представления лабиринта введём класс `Maze`, содержащий двумерный массив клеток и игровые объекты. Для простоты поручим ему также функции контроллера игры. Так как клетка может быть не только пустой, но и содержащей участок стены, а также меняться из-за столкновений, введём для представления клеток класс `Cell`. Для танков введём абстрактный класс `Tank` и его подклассы для игроков разных типов. Также введём класс `Bullet` для снарядов.

Для всех объектов, изменяющихся со временем, введём операцию `update()`, которую будем вызывать на каждом шаге игрового цикла для каждого объекта. Данная операция позволяет реализовать движение объектов или какую-либо иную анимацию.

Изобразим вышеизложенное на диаграмме классов как показано на рисунке 5.2.

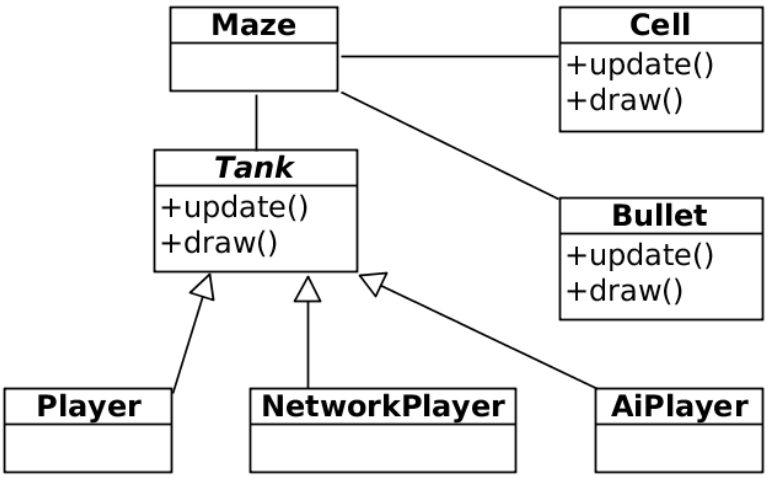

Рис. 5.2: Диаграмма классов **Key Abstractions**

Продумайте, какие атрибуты должны быть у этих классов и внесите их на UML-диаграмму.

5.2.1 Расчёт следующего кадра

Для каждого кадра игры нужно вычислить новые положения игровых объектов и их столкновения. Если за время между кадрами игровые объекты смещаются мало, то можно проводить расчёты с временным шагом равным длительности кадра. Если скорости объектов велики, то это может привести к тому, что не все столкновения будут учтены. Тогда необходимо выполнять расчёт несколько раз для каждого кадра.

На каждом шаге нужно вычислить:

- новые положения игрока, каждого снаряда и каждого противника;

- столкновения игрока со снарядами противника и противников со снарядами игрока (считаем, что игрок не может быть поражён своим выстрелом, а противники — выстрелами своей стороны);
- столкновения со стенами, разрушение стен снарядами, уничтожение флага;
- появление ремкомплектов и их захват танком.

Изобразите расчёт кадра на диаграмме последовательности.

5.2.2 Кнопки

Для взаимодействия с пользователем вне игрового процесса (начало / конец игры, настройка параметров) нам потребуются кнопки. Введём класс `Button`, а для каждой конкретной кнопки создадим по классу-наследнику, в котором переопределим метод `mouseButtonReleased()`.

Есть два варианта отображения кнопок на экране

1. рисовать каждую кнопку программно;
2. отображать заранее подготовленную картинку.

Первый вариант идеален для приложений, предназначенных для работы, в которых большое число кнопок, которые должны отображаться стандартным образом в соответствии с настройками операционной системы.

Для игры больше подойдёт второй вариант, так как кнопок мало, а у дизайнеров должны быть развязаны руки, поэтому именно его мы реализуем. У кнопки может быть до 4 состояний: обычное, подсвеченное (на кнопку наведён курсор), нажатое и недоступное. На диаграмме классов это следует показать как ассоциацию с четырьмя текстурами.

Обсудите с командой, какие кнопки вам понадобятся и изобразите диаграмму классов.

5.2.3 Построение игрового уровня

Ещё один важный вопрос — как осуществлять построение лабиринта. В требованиях сказано, что в дальнейшем планируется реализовать игру с несколькими уровнями. Уровни могут отличаться количеством врагов, структурой лабиринта, графическим оформлением.

Для конфигурирования уровня введём класс `LevelBuilder`, имеющий метод `nextLevel(Maze &game)`, преобразующий игру в соответствии с новым уровнем.

Теперь, когда каждый член команды проанализировал проект, обсудите работу друг друга. Сравните результаты и подготовьте единую модель анализа, включающую в себя лучшие наработки. Покажите её преподавателю.

5.3 Проектирование системы

Проектирование системы проводите совместно.

Выберите библиотеки, которые будете использовать для написания программы (рекомендуется использовать SFML). Исходя из выбранных библиотек определите, какие классы вы будете реализовывать самостоятельно, а какие использовать готовыми.

В проекте создайте **Design Model**, а в ней пакеты **Game** и **Services**, а также по пакету для каждой используемой библиотеки. В пакет **Services** будем складывать служебные элементы.

Рекурсивно продублируйте модель анализа и распределите классы из неё в подходящие пакеты проектной модели, а также перенесите пакет **Use Case Realization** целиком. Удалите то, что осталось от дубля и не вошло в проектную модель.

Как видно из анализа выше, многие классы в игре имеют ассоциации с классом текстур. На этапе проектирования необходимо решить, как хранить эти текстуры и как их загружать. При этом нужно учесть следующие особенности:

- текстуры занимают большой объём памяти;
- загрузка текстуры с диска — длительная операция;
- многие экземпляры классов имеют одинаковые текстуры;
- организация хранения и загрузки текстур выходит за рамки ответственности классов-сущностей.

Поэтому для работы с текстурами введём класс `TextureManager`, имеющий метод `Texture* TextureManager::get(const char *name)`

Обратите внимание, что этот класс должен кэшировать текстуры, чтобы не загружать их повторно при новом обращении. Кроме того, метод `get` должен быть доступен из многих мест программы. Поэтому нужно или сделать глобальный экземпляр класса, или класс со статическими методами, или синглтон.

Спроектируйте загрузку уровня. `LevelBuilder` может иметь методы, где будет жёстко заданы характеристики каждого уровня, а может загружать уровни из внешних ресурсов. Изобразите построение нового уровня в виде диаграммы последовательности.

Проведите преобразование классов анализа в проектные классы. Уточните поля и методы. Обозначьте используемые типы. Ответьте на такие вопросы как:

- Какие контейнеры использовать для игровых объектов?
- Какие аргументы передаются в каждый из методов, используемый на диаграмме?
- В каких случаях используются сами экземпляры классов, а в каких — указатели и ссылки на них?
- Каким образом контроллеру игры передаётся пуля, которую нужно добавить в контейнер?
- Как обрабатывается нажатие клавиш клавиатуры при игре или клавиши мыши при нажатии кнопки?

Уточните диаграммы последовательности в соответствии с принятыми вами проектными решениями.

Покажите результаты преподавателю и получите от него доступ к репозиторию и дальнейшие указания.