Введение

Темой проекта является разработка многопользовательской игры, созданной на основе популярной настольной игры «Каркассон».

Целью проекта заключается в создании настольной игры в версии на ПК с добавлением игрового дизайна, анимацией, дополнительных игровых правил и функций.

Ниже приводится краткое описание разделов пояснительной записки.

Первый раздел носит название “Постановка задачи ”. В нем вы сможете ознакомиться с исследованием предметной области поставленной задачи, определение ее организационно-экономической сущности а так же ознакомиться с инструментами разработки.

В разделе “Проектирование задачи” рассмотрен выбор стратегии разработки и модели ЖЦ, описываются инструменты разработки и UML-диаграммы, а также представлены прототипы страниц сайта.

“Реализация” – это третий раздел пояснительной записки, в котором будут подробно описаны все процессы и правила которым следовал разработчик при разработке программного обеспечения, все функции и элементы управления а также спецификация проекта.

Четвертый раздел – “Тестирование” в котором приведено описание тестирования данного ресурса, т.е. протестирован каждый пункт меню, каждая операция, которая выполняется приложением, а также пользовательский интерфейс.

Раздел “Руководство пользователя” содержит информацию, необходимую для использования системы. В этом разделе описываются все функциональные возможности системы а так же приводятся пошаговые инструкции по её использованию.

В заключении подведены итоги по разработке интернет-ресурса.

В разделе «Список использованных источников» будет приведён список используемых при разработке источников.

В «Приложении А» будет приведена диаграмма вариантов использования.

В «Приложении Б» будет приведена диаграмма последовательности.

В «Приложении В» будет приведена диаграмма структуры проекта.

В «Приложении Г» будет приведена диаграмма деятельности.

В «Приложении Д» будут приведены диаграмма объектов

В «Приложении Е» будет приведена модель данных.

В «Приложении Ж» будет приведена диаграмма компонентов.

В «Приложении З» будет приведена модель бизнес-процессов.

В «Приложении И» будет приведена диаграмма классов.

В «Приложении Й» будет приведены прототипы UX-интерфейсов.
В «Приложении К» будет привидены прототипы UI-интерфейсов.
В «Приложении Л» будет приведён листинг программного продукта.
Схема работы системы будет представлена в графической чаской части.

1 Анализ задачи

1.1 Постановка задачи

1.1.1 Организационно-экономическая сущность задачи

Темой проекта является разработка многопользовательской игры, созданной на основе популярной настольной игры «Каркассон».

Целью проекта заключается в создании настольной игры в версии на ПК с добавлением игрового дизайна, анимацией, дополнительных игровых правил и функций.

Периодичность использования игры «Каркассон» зависит от потребностей и интересов пользователей. Поиграть в данную игру могут как и проффесионалы в сфере настольных игр, так и новички, желающие попробовать себя в сфере настольных игр на ПК.

На данный момент существует одна аналогичная игра, такая как:

- Игра Carcassone Tiles & Tictics

(https://thebyrut.org/12035-carcassonne-tiles-and-tactics.html);

Однако, она отличаются по функциональности, уровню интерактивности и доступности. Проект стремится предоставить более удобную и понятную игру.

1.1.2 Функциональные требования

Используя данный программный продукт (ПП), пользователь сможет:

- регистрироваться и авторизовываться
- настраивать громкость музыки и звуков игры
- создавать сетевую игру
- создавать одиночную игру
- присоединяться к сетевой игре
- проводить действия с пазлами
- проводить действия с фигурками
- передвигаться по карте
- просматривать правила игры
- просматривать авторов игры

1.1.3 Описание входной, выходной и условно-постоянной информации.

Вся информация, с которой работает разрабатываемый программный продукт можно разделить на:

– входную информацию;

– выходную информацию;

– условно-постоянную информацию.

Входной информацией выступают фишки и плитки, карты и карточки, различные ресурсы, фигурки и персонажи а так же очки.

Выходной информацией выступают очки игроков и их счета, состояние ресурсов, взаимодействия с другими игроками.

Условно-постоянной информацией являются правила игры, компоненты игры, описание игрового мира а также характеристики игровых компонентов.

1.1.4 Нефункциональные (эксплуатационные) требования.

В данном разделе представлены нефункциональные требования к игровому программному продукту основанном на настольной игре «Каркассон». Нефункциональные требования включают требования к интерфейсу, применению, производительности и реализации.

Требования к применению: игра Каркассон должна предоставлять пользователям возможность участвовать в увлекательной стратегической игре, развивать тактическое мышление и осуществлять построение фэнтезийного города.

Требования к производительности: время восстановления системы после сбоя зависит от скорости реакции создателей на вероятную проблему.

Требования к реализации: Для реализации данного приложения будет использоваться игровая платформа Unity основанная на языке C#, которая позволяет создать игру данного типа.

Требования к интерфейсу (определяют внешние сущности (т.е. пользователей и любые внешние устройства), с которыми может взаимодействовать система, и регламент этого взаимодействия): при разработке приложения должны быть использованы преимущественного коричневые/серые оттенки. Основные разделы приложения должны быть доступны в главном меню. Грамотный пользовательский интерфейс.

1.2 Диаграмма вариантов использования

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы. Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю

качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать.

Актер – это внешняя по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для решения определенных задач.

Рисунок 1 - Актёр

Вариант использования применяется для спецификации общих особенностей поведения системы или другой сущности без рассмотрения ее внутренней структуры.

Рисунок 2 – Вариант использования

С диаграммой вариантов использования для игрового программного продукта «Каркассон» можно ознакомиться в Приложении А.

2 Проектирование

2.1 Выбор модели, метода и подхода разработки программы.

Для разработки игрового программного продукта, основанного на основе настольной игры «Каркассон» надо выбрать стратегию разработки и модель жизненного цикла ПО.

Таблица 2 – Выбор модели жизненного цикла на основе характеристик требований

№ критерия	Критерии категории требования	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипирования	Эволюционная
1	2	3	4	5	6	7	8
1	Являются ли требования к проекту легко определимыми и реализуемыми?	Да	Да	Да	Нет	Нет	Нет
2	Могут ли требования быть сформулированы в начале ЖЦ?	Да	Да	Да	Да	Нет	Нет
3	Часто ли будут изменяться требования на протяжении ЖЦ?	Нет	Нет	Нет	Нет	Да	Да
4	Нужно ли демонстрировать требования с целью их определения?	Нет	Нет	Да	Нет	Да	Да
5	Требуется ли проверка концепции программного средства или системы?	Нет	Нет	Да	Нет	Да	Да
6	Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ?	Нет	Нет	Нет	Да	Да	Да
7	Нужно ли реализовать основные требования на ранних этапах разработки?	Нет	Нет	Да	Да	Да	Да

Вычисления: 5 за каскадную, 5 за V-образную, 6 за RAD, 4 за инкрементную, 2 за быстрого прототипирования и 2 за эволюционную.

Итог: на основе результатов заполнения таблицы 2, подходящей является RAD модель.

Таблица 3 – Выбор модели жизненного цикла на основе характеристик команды разработчиков

№ критерия	Критерии категории требования	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипировани	Эволюционная
1	2	3	4	5	6	7	8
1	Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Нет	Нет	Да	Да
2	Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков?	Да	Да	Нет	Нет	Нет	Да
3	Изменяются ли роли участников проекта на протяжении ЖЦ?	Нет	Нет	Нет	Да	Да	Да
4	Является ли структура процесса разработки более значимой для разработчиков, чем гибкость?	Да	Да	Нет	Да	Нет	Нет
5	Важна ли легкость распределения человеческих ресурсов проекта?	Да	Да	Да	Да	Нет	Нет
6	Приемлет ли команда разработчиков оценки, проверки, стадии разработки?	Да	Да	Нет	Да	Да	Да

Вычисления: 5 за каскадную, 5 за V-образную, 4 за RAD, 5 за инкрементную, 2 за быстрого прототипирования и 1 за эволюционную.

Итог: на основе результатов заполнения таблицы 3, подходящей является каскадная, V-образная или инкрементная модель.

Таблица 4 – Выбор модели жизненного цикла на основе характеристик коллектива пользователей

№ критерия	Критерии категорииколлектива пользователей	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипирования	Эволюционная
1	2	3	4	5	6	7	8
1	Будет ли присутствие пользователей ограничено в ЖЦ разработки?	Да	Да	Нет	Да	Нет	Да
2	Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки?	Нет	Нет	Нет	Да	Да	Да

№ критерия	Критерии категорииколлектива пользователей	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипирования	Эволюционная
1	2	3	4	5	6	7	8
3	Будут ли пользователи вовлечены во все фазы ЖЦ разработки?	Нет	Нет	Да	Нет	Да	Нет
4	Будет ли заказчик отслеживать ход выполнения проекта?	Нет	Нет	Нет	Нет	Да	Да

Вычисления: 3 за каскадную, 3 за V-образную, 1 за RAD, 2 за инкрементную, 1 за быстрого прототипирования и 3 за эволюционную.

Итог: на основе результатов заполнения таблицы 4, подходящей является каскадная, V-образная и эволюционная модель.

Таблица 5 – Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

№ критерия	Критерии категории типов проекта и рисков	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипирования	Эволюционная
1	2	3	4	5	6	7	8
1	Разрабатывается ли в проекте продукт нового для организации направления?	Нет	Нет	Нет	Да	Да	Да
2	Будет ли проект являться расширением существующей системы?	Да	Да	Да	Да	Нет	Нет
3	Будет ли проект крупно- или среднемасштабным?	Нет	Да	Да	Да	Нет	Нет
4	Ожидается ли длительная эксплуатация продукта?	Да	Да	Нет	Да	Нет	Да
5	Необходим ли высокий уровень надежности продукта проекта?	Нет	Да	Нет	Да	Нет	Да
6	Предполагается ли эволюция продукта проекта в течение ЖЦ?	Нет	Нет	Нет	Да	Да	Да
7	Велика ли вероятность изменения системы (продукта) на этапе сопровождения?	Нет	Нет	Нет	Да	Да	Да
8	Является ли график сжатым?	Нет	Нет	Да	Да	Да	Да

№ критерия	Критерии категории типов проекта и рисков	Каскадная	V-образная	RAD	Инкрементная	Быстрого прототипирования	Эволюционная
1	2	3	4	5	6	7	8
9	Предполагается ли повторное использование компонентов?	Нет	Нет	Да	Да	Да	Да
10	Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)?	Нет	Нет	Нет	Нет	Да	Да

Вычисления: 4 за каскадную, 5 за V-образную, 5 за RAD, 5 за инкрементную, 5 за быстрого прототипирования и 7 за эволюционную.

Итог: на основе результатов заполнения таблицы 5, подходящей является эволюционная модель.

Общий итог: в итоге заполнения таблиц 2-5 наиболее подходящей является V-образная модель.

2.2 Инструменты разработки

Для разработки данного проекта была выбрана конструктор платформа Unity которая является ведущей платформой с использованием языка C#, она позволит значительно ускорить процесс разработки. В случае с Dekstop приложением использование платформы Unity может быть особенно полезным. Данная платформа позволит создать интуитивно понятный и простой интерфейс, который будет легко использовать пользователям.

Иные инструменты, используемые при разработке и написании сопутствующей документации:

- Приложение DRAW.IO – будет использоваться для создания графической части и разработки UML-диаграмм;
- Adobe Photoshop CS6 – будет использоваться для создания картинок приложения.
- Microsoft Office Word – для написания документации к программному продукту;
- Figma – будет использоваться для построение UX и UI интерфейсов.

Разработка проекта будет происходить на компьютере со следующими параметрами:

- процессор Intel® Core ™ i3-3110M CPU @ 2.40GHz;
- объем оперативной памяти 4.00 GB;
- объем места на SDD-диске 256 GB;
- ОС Windows 10 Home.

2.3 Разработка UML-диаграмм

2.3.1 Диаграмма последовательности

Диаграмма последовательности UML – это диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления. Основные элементы диаграммы последовательности это: обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

При разработке программного продукта была поставлена задача разработать диаграмму последовательности, которая показывает порядок взаимодействия пользователя с системой при прохождении теста. С данной диаграммой можно ознакомиться в Приложении Б.

2.3.2 Диаграмма компонентов

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы.

Компонентами могут быть программные компоненты, такие как база данных или пользовательский интерфейс; или аппаратные компоненты, такие как схема, микросхема или устройство; или бизнес-подразделение, такое как поставщик, платежная ведомость или доставка.

При разработке программного продукта была поставлена задача разработать диаграмму компонентов. С данной диаграммой можно ознакомиться в Приложении В.

2.3.3 Диаграмма деятельности

Диаграмма деятельности UML позволяет более детально визуализировать конкретный случай использования. Это поведенческая диаграмма, которая иллюстрирует поток деятельности через систему.

Диаграммы деятельности UML также могут быть использованы для отображения потока событий в бизнес-процессе. Они могут быть использованы для изучения бизнес-процессов с целью определения их потока и требований.

При разработке программного продукта была поставлена задача разработать диаграмму деятельности. С данной диаграммой можно ознакомиться в Приложении Г.

2.4 Разработка пользовательского интерфейса

Важным элементом проектирования программного продукта является описание пользовательского интерфейса разрабатываемого программного продукта.

Для разработки визуального дизайна использовались сдержанные, мягкие цвета для удобства использования программного продукта.

В ходе разработки был спроектирован дизайн главного меню игрового программного продукта «Каркассон». Разработанная структура ПП расположена в Приложении Д.

Для организации эффективной работы пользователя нужно создать целостный программный продукт данной предметной области, в котором все компоненты будут сгруппированы по функциональному назначению. При этом необходимо обеспечить удобный графический интерфейс пользователя. ПП должен позволить пользователю решать задачи, затрачивая значительно меньше усилий, чем при работе с разрозненными объектами. Все исходные данные будут разделены на несколько групп.

Прототип – это наглядная модель пользовательского интерфейса. В сущности, это «черновик» созданный на основе представления разработчика о потребностях пользователя. Итоговое отображение программы может отличаться от прототипа. С прототипом главной страницы вы можете ознакомиться в Приложении Й и Приложении К.

UX-прототипы: *ссылка*

UI-прототипы: *ссылка*

3 Реализация

3.1 Руководство программиста

В этом данном руководстве рассказывается о ключевых аспектах разработки и реализации этой увлекательной стратегической игры. В руководстве подробно указано как пользоваться платформой Unity (создание проекта, регистрация и т.п). Основные руководство по использованию будут предоставлены в руководстве пользователя.

3.1.1 Создание программного продукта

После нажатия кнопки «Got it» откроется установщик который предложит загрузить последнюю актуальную версию Unity. Нужно нажать на кнопку «Install». По умолчанию папкой для установки будет C://Program Files/Unity/Hub/Editor (Рисунок 3).

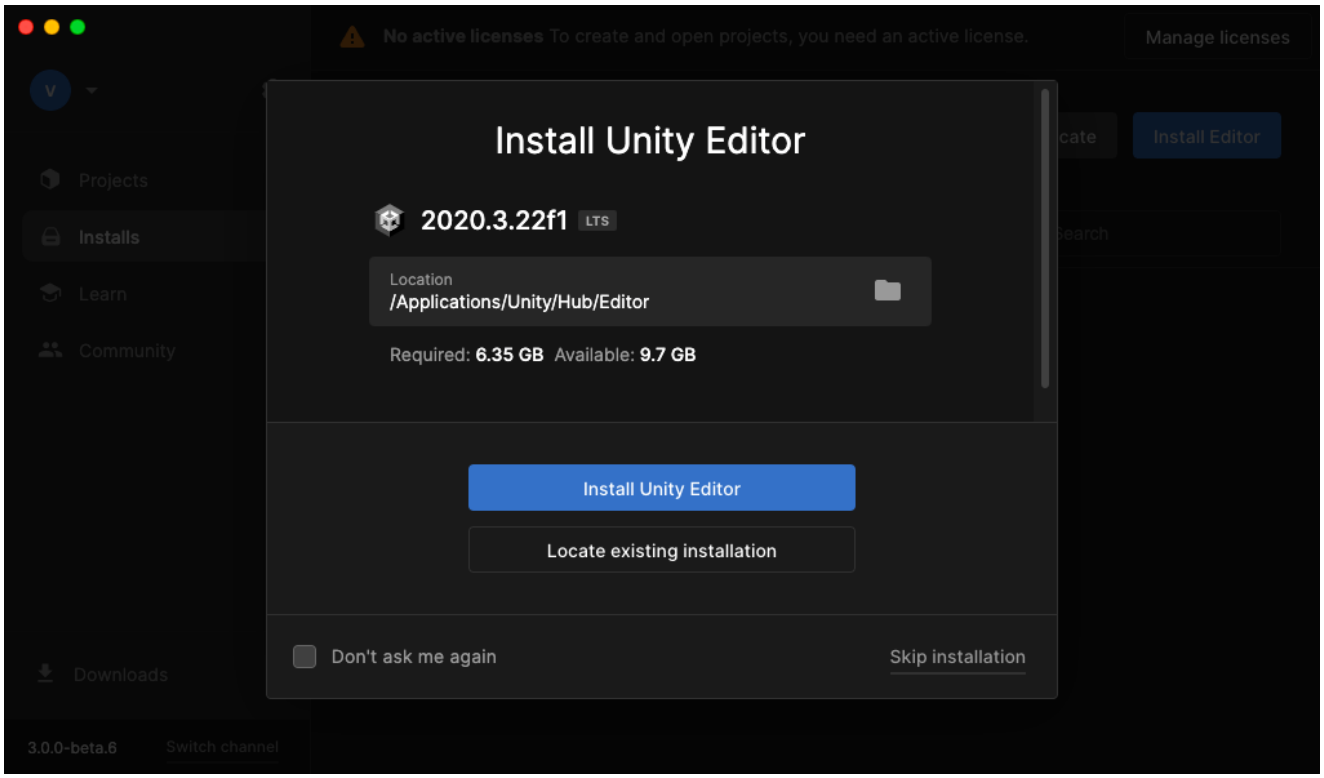


Рисунок 3 – Установщик Unity

Для разработки также нужно получить лицензию, нужно нажать «Agree and get person edition license». Unity можно использовать бесплатно, если игры,

которые вы разрабатываете, не приносят больше 100 000$ в год, это описано в условиях (Рисунок 4).

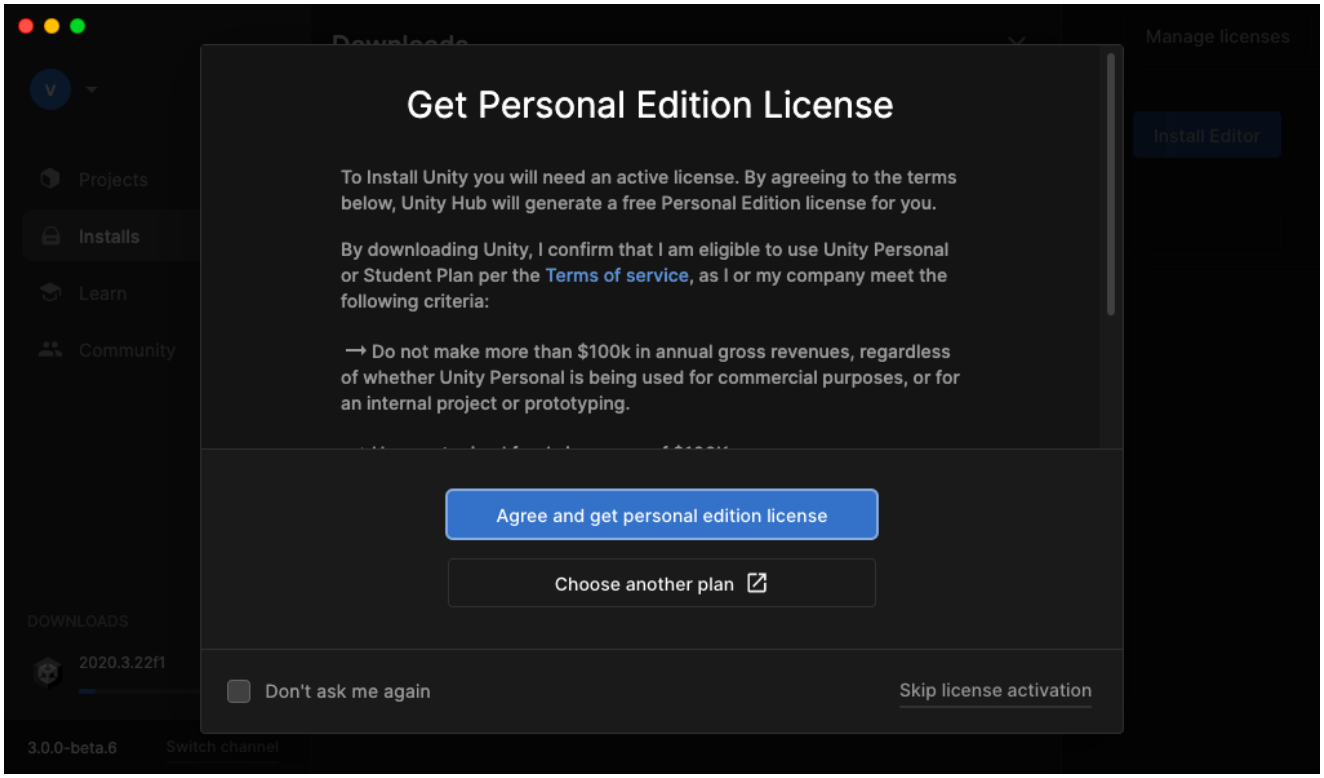


Рисунок 4 – Лицензионные условия

После выполнения всех вышеперечисленных действий начнётся загрузка. Естественно на установку Unity Hub потребуется время (Рисунок 5).

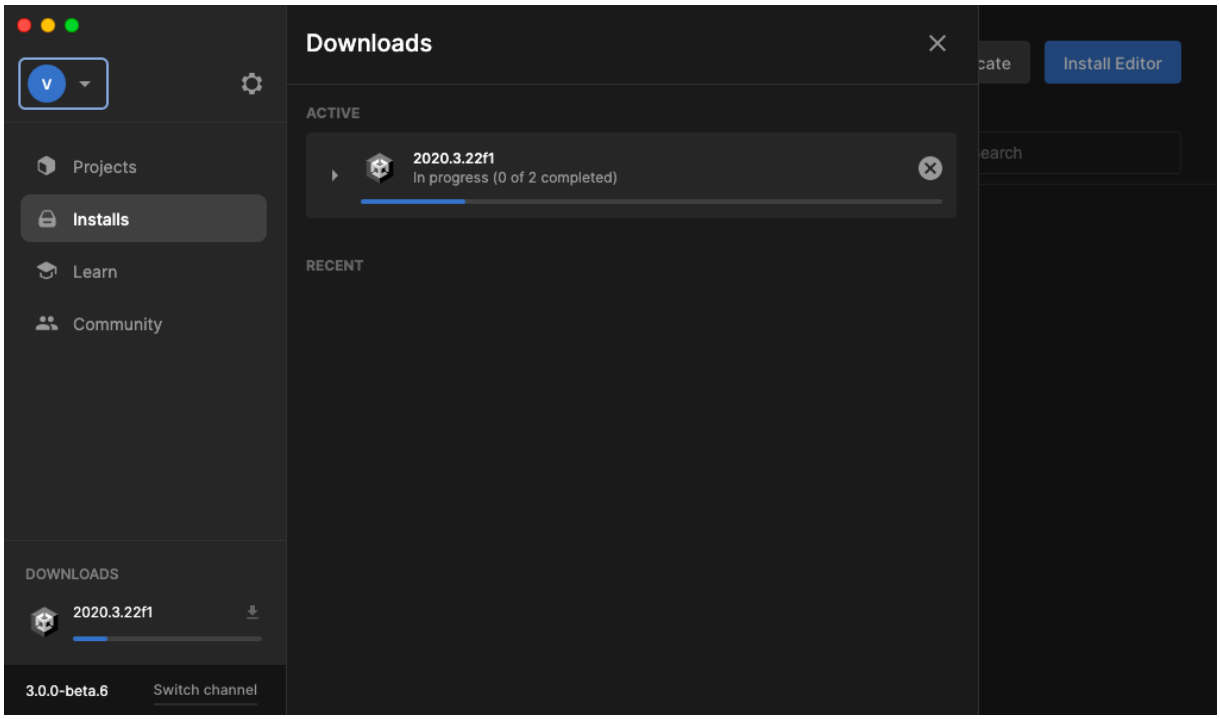


Рисунок 5 – Процесс установки Unity

Рисунок 6. Параметры установки редактора или проектов можно поменять в настройках Unity (Preferences).

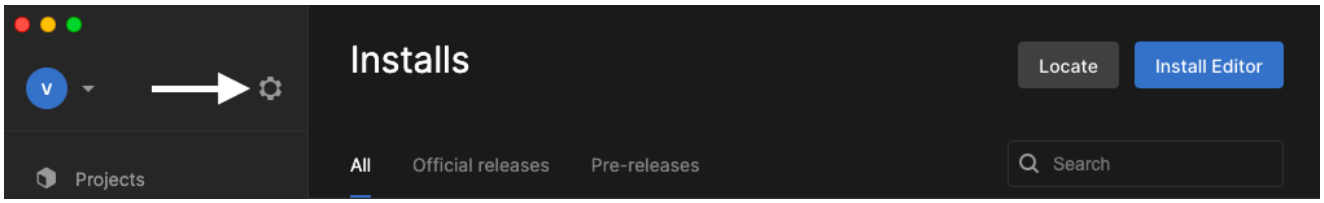


Рисунок 6 – Параметры установки

Выберите Location, если нужно изменить папку сохранения. Далее закройте настройки (смотреть Рисунок 7).

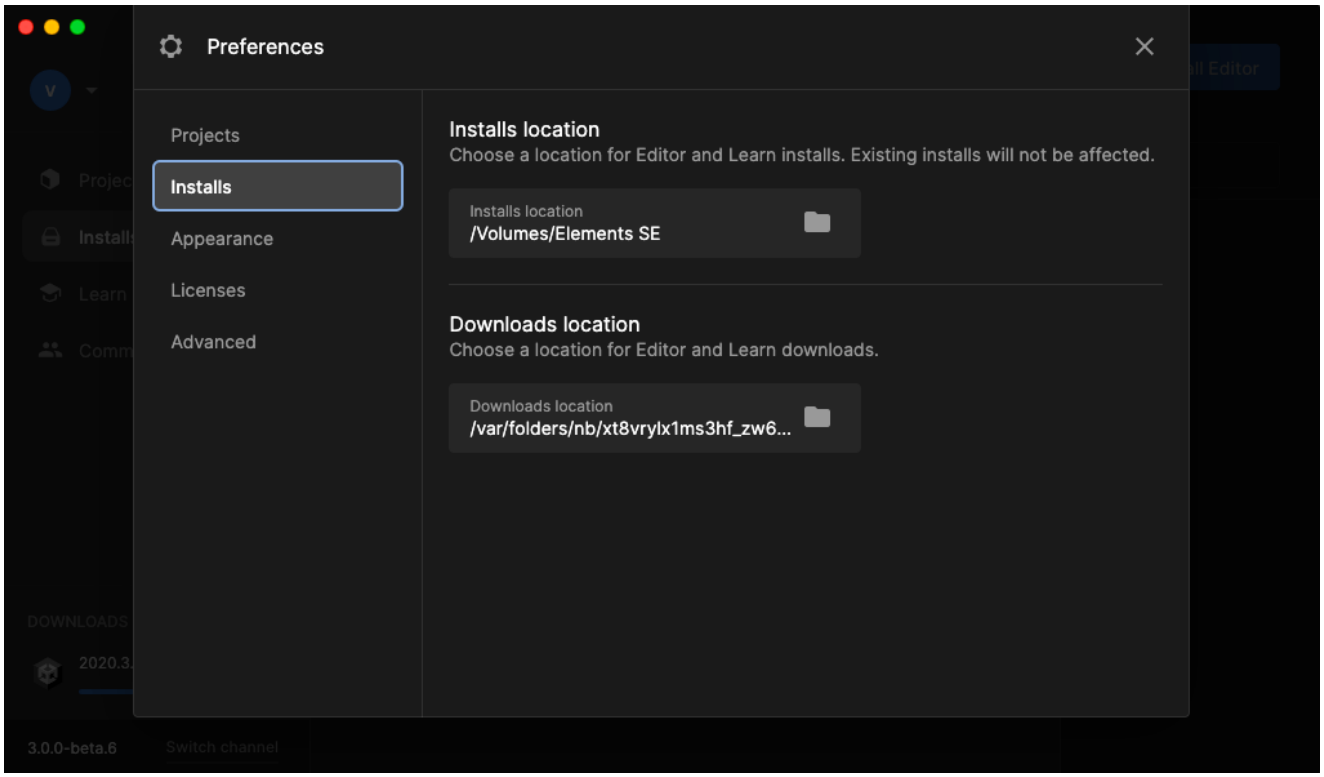


Рисунок 7 – Изменение папки сохранения

Если нужно установить другие версии редактора, нажмите Install Editor, выберите подходящую для себя версию и нажмите кнопку «Install». (Рисунок 8).

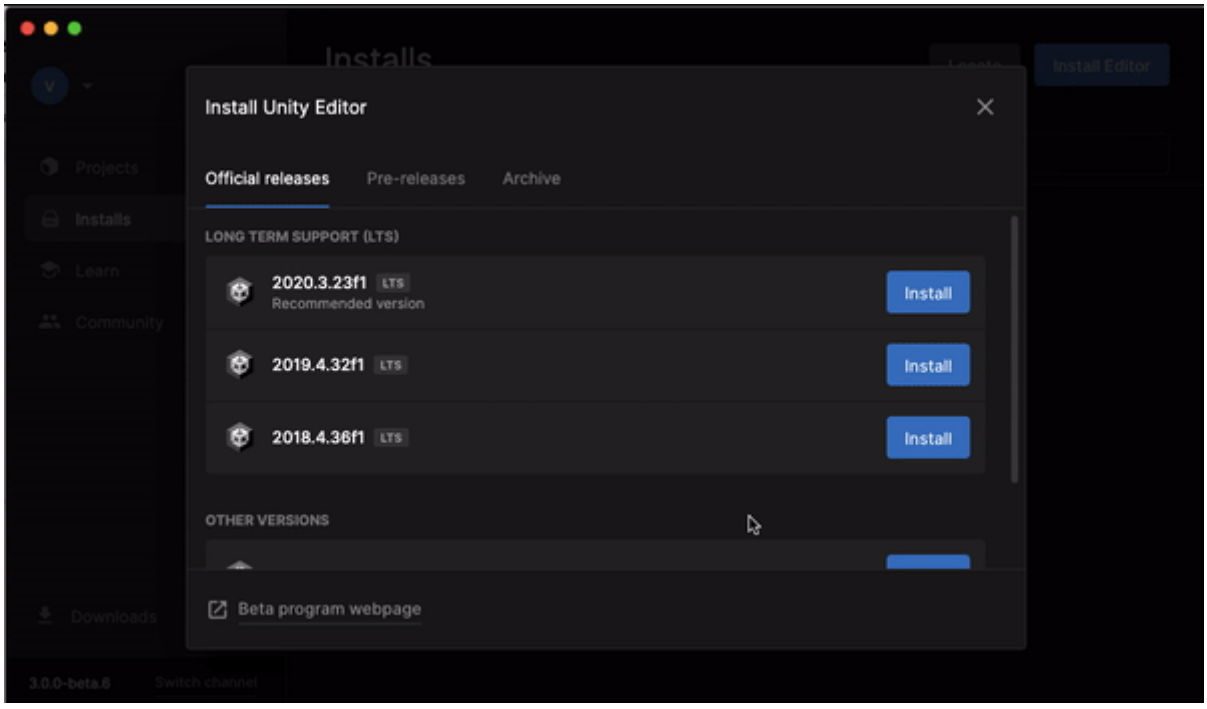


Рисунок 8 – Выбор другой версии Unity

3.1.2 Работа с платформой Unity

После завершения загрузки можно попробовать зайти в проект: Projects – New Project (смотреть Рисунок 9).

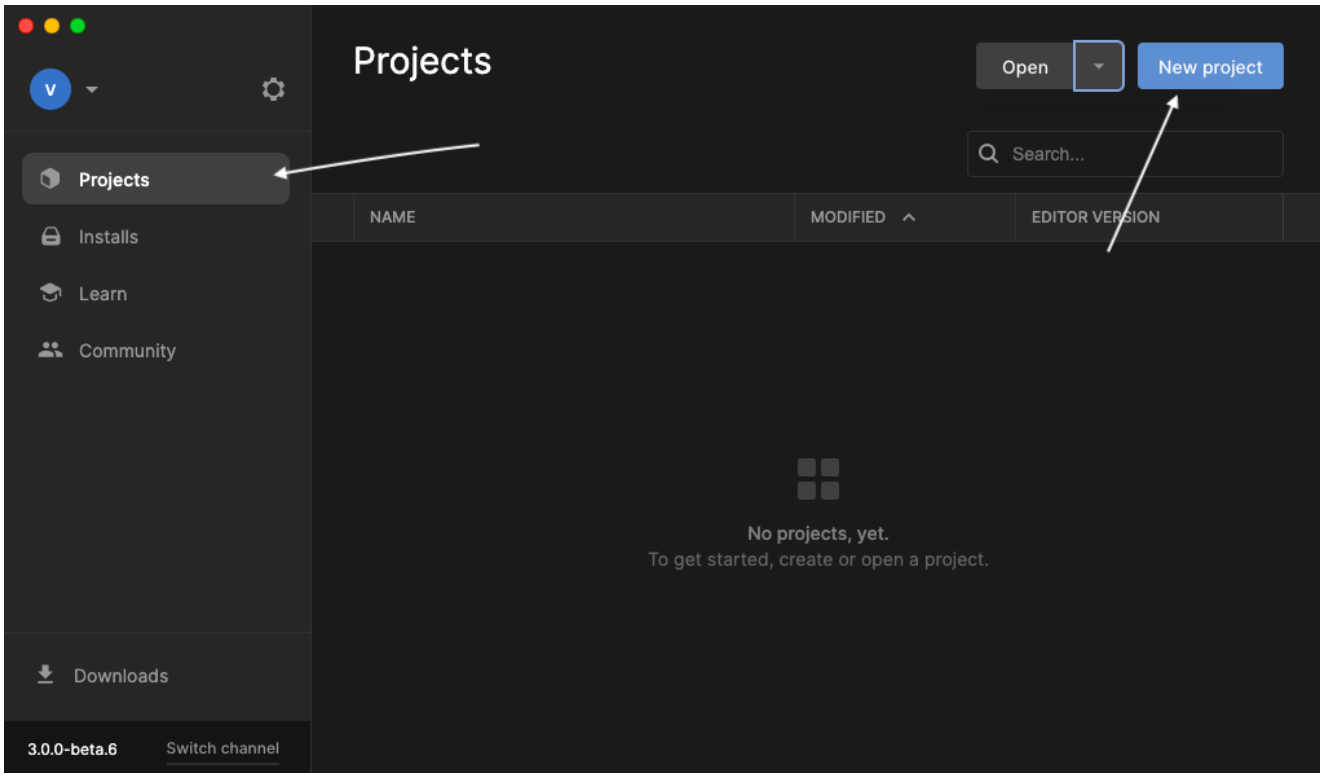


Рисунок 9 – Создание нового проекта

Выберите любой из предложенных вариантов. Затем нажмите на кнопку «Create project» (смотреть Рисунок 10).

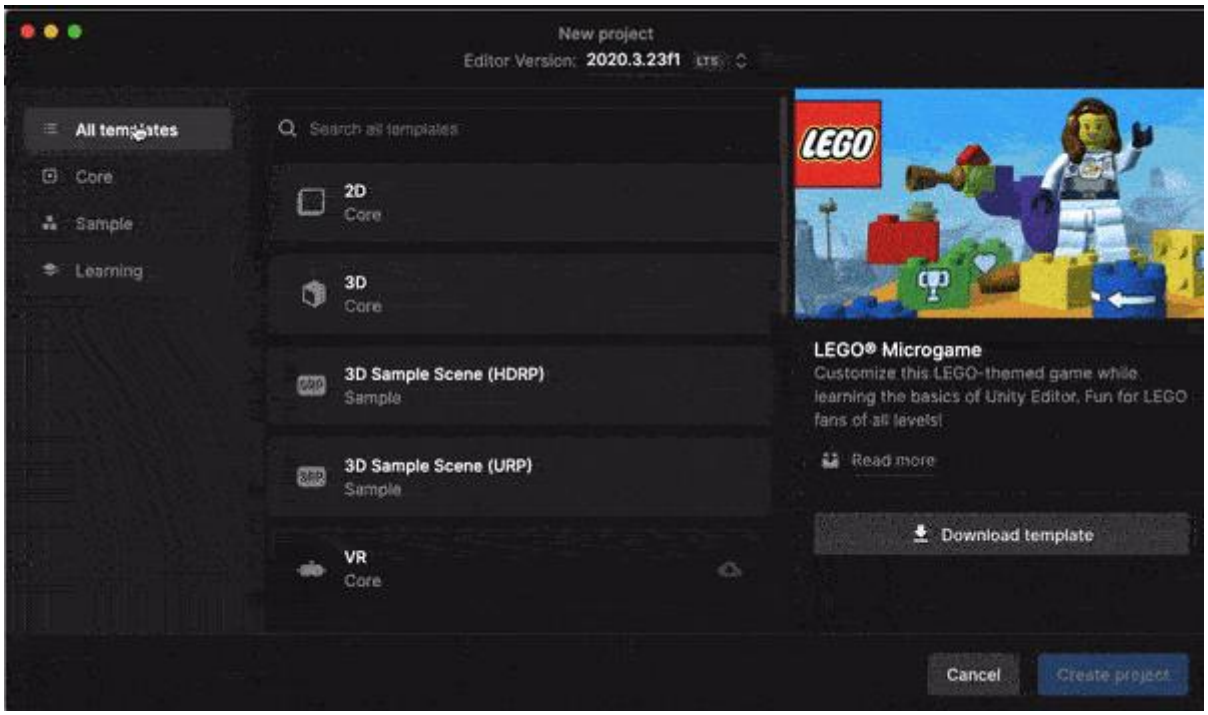


Рисунок 10 – Варианты создания проекта

Соответственно вот проект и был загружен, теперь можно начинать работу с платформой (смотреть Рисунок 11).

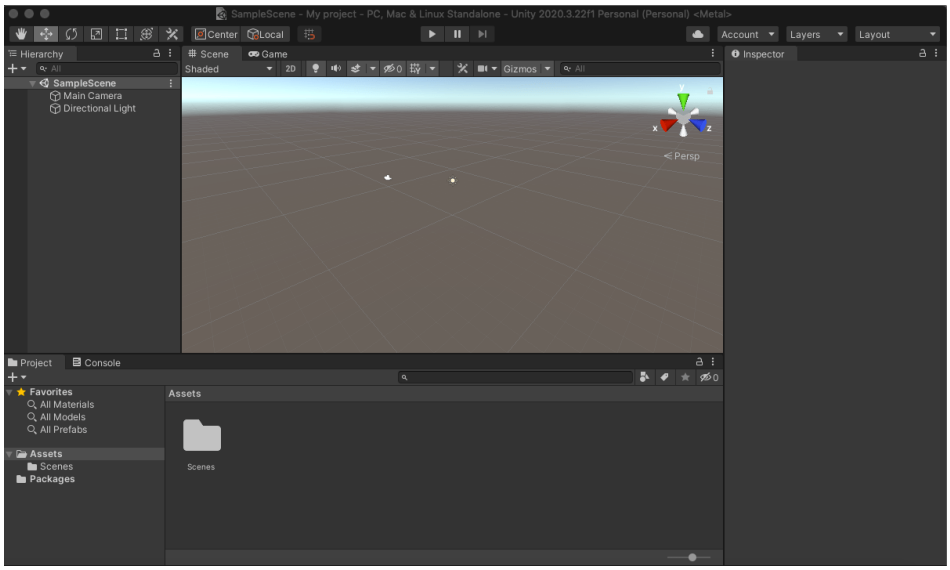


Рисунок 11 – Созданный проект

В стандартном интерфейсе проекта шесть элементов рабочей области: Верхняя панель инструментов, Scene, Games, Hierarchy, Projec, Inspector (Рисунок 12).

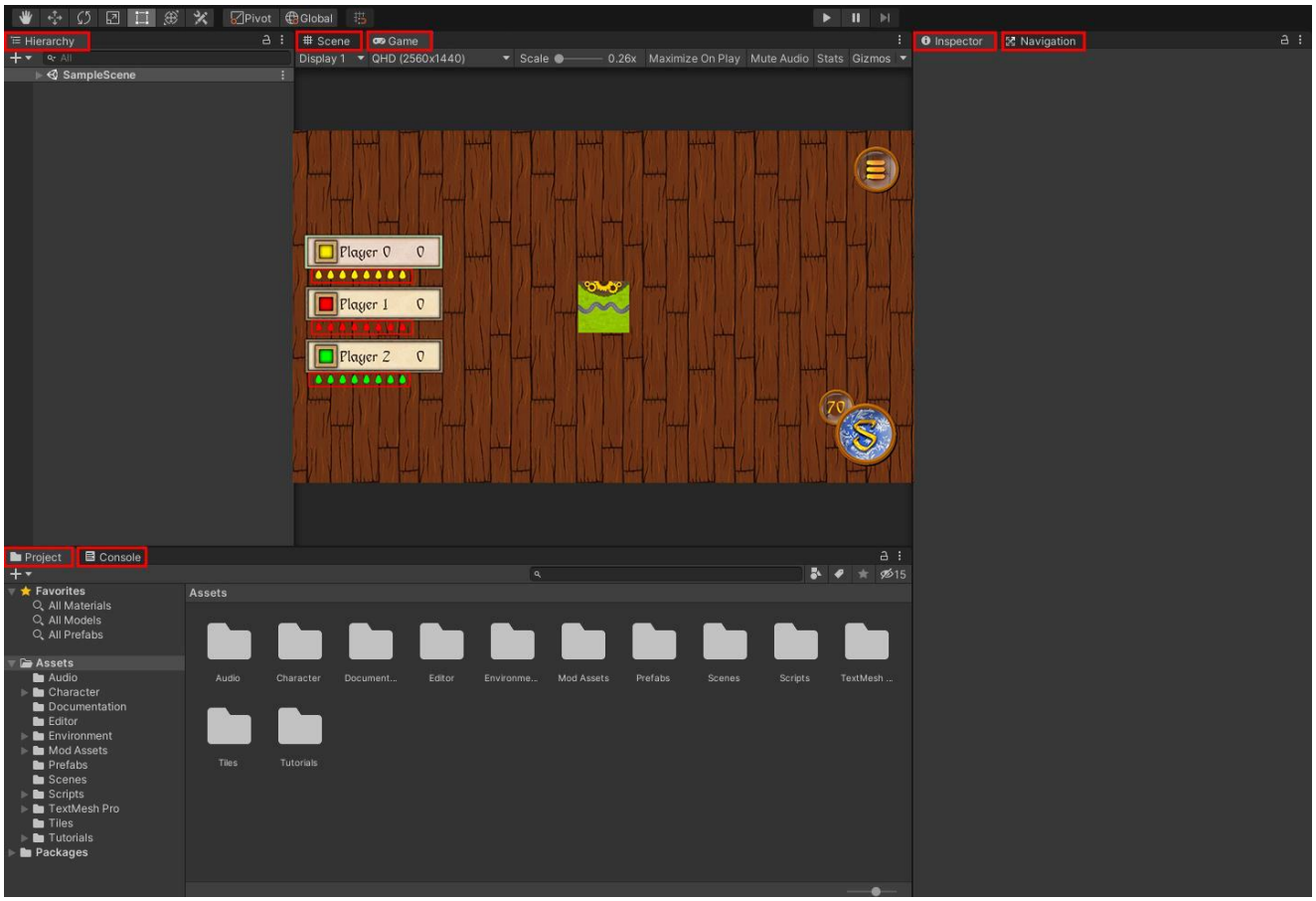


Рисунок 12 – Стандартный рабочий интерфейс

На рисунке 13 представлена рабочая область Scene. Именно здесь задаётся код.

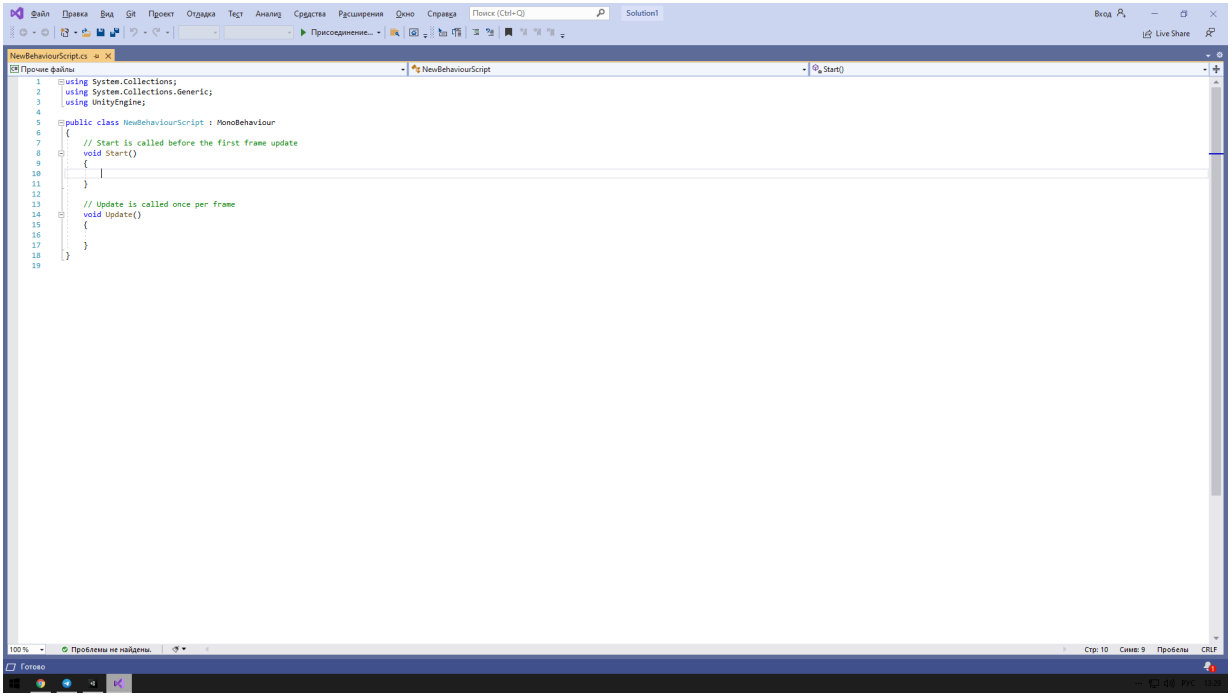


Рисунок 13 – Рабочая область Scene

Для того чтобы работать с триггерами изначально нужно освоить что такое Collaider. Разберём как же создать триггер. Чтобы создать триггер в Unity, нужно накинуть тот же самый компонент коллайдера, но поставить галочку «Is Trigger» (смотреть Рисунок 14).

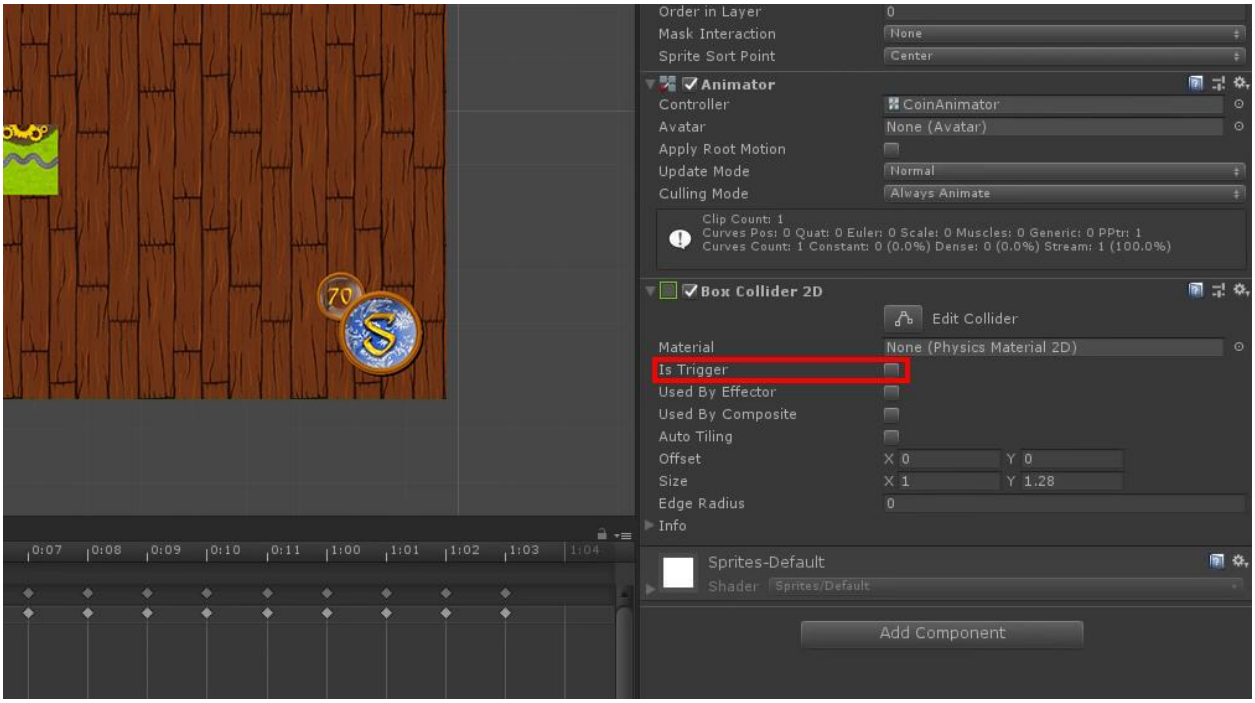


Рисунок 14 – Создание триггера

4 Тестирование

При разработке данного проекта возникало довольно небольшое количество ошибок так как платформа Unity помогает решить создателю ошибки предлагая свои функции (например при неправильном распределении компонентов Unity предлагает заменить их на более совместимые), основные тесты заключались в проверке всех кнопок и функицй игры. После завершения этапа написания программы было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме. Отчёт о результатах тестирования представлен в таблице 6.

Таблица 6 – Отчёт о результатах тестирования

№	Название теста	Действия	Исходная информация	Ожидаемая информация
1	2	3	4	5
Т1	Проверка кнопки взятия пазла	Нажатие на кнопку «Взять пазл»	-	Успешно появленный пазл.
Т2	Проверка кнопки переворота пазла.	Нажатие на кнопку «Перевернуть пазл».	-	Успешно перевернётый на 90 градусов пазл.
Т3	Выбор места постановки пазла.	Перетаскание пазла на любое свободное место.	-	Успешное перетаскивание пазла.
Т4	Проверка кнопки постановки пазла.	Нажатие на кнопку «Поставить пазл».	-	Успешно поставленный пазл.
Т5	Проверка функции «Отдаление карты»	Прокрутка колесом компьютерной мыши «Вверх» и «Вниз»	-	Успешно отдалённая или приближённая карта.

Т6	Проверка функции «Перемещение по карте».	Зажатие ЛКМ и перетягивание компьютерной мыши «Влево» или «Вправо»	-	Успешно перемещение по карте.
Т7	Выбор места постановки персонажа.	Выбрать из предложенных мест место для постановки персонажа.	Данные о карточке которую поставил пользователь.	Успешно выбранное место для постановки персонажа.
Т8	Отмена выбора места постановки персонажа.	Нажать на кнопку «Отмена» после выбора места для постановки персонажа.	Данные о месте куда пользователь хочет поставить персонажа.	Успешно отменнёное действие.
Т9	Постановка персонажа на выбранное место.	Нажать на кнопку «Подтвердить» после выбора места для постановки персонажа.	Данные о месте куда пользователь хочет поставить персонажа.	Успешное постановление персонажа.
Т10	Проверка начисление очков.	Установить персонажа на место карточки и дождаться добавления очков.	Данные о захваченной территории пользователем.	Успешное начисление очков.
Т11	Проверка уменьшения количества пазлов при постановке нового.	Пользователь устанавливает пазл, после его установки количество пазлов должно быть уменьшено на 1.	Данные о количествах пазлов.	Успешное уменьшение количества пазлов на 1.

При тестировании программного продукта, в первую очередь нужно обратить внимание на правильную работу окон игрового программного продукта. Показываются ли все изображения, виден ли текст, нет ли лишних пробелов и больших отступов.

В результате проведения тестирования выяснилось, что все требования, предъявляемые к ПП, были разработаны и протестированы. Тесты показали, что все функции работают правильно.

5 Руководство пользователя

5.1 Общие сведения о программном продукте

Разрабатываемый программный продукт будет носить название «Игровой программный продукт основанный на основе настольной игры «Каркассон».

Данный программный продукт предназначен для людей всех возрастов которые желают поиграть в одну из самых популярных настольных игр на персональном компьютере. Для определённого круга людей такие как: люди, которые увлекаются стратегическими играми и любят развивать своё тактическое мышление, данная игра привлекает любителей фэнтэзи-жанра, где игроки могут погрузиться в захватывающий мир настольной игры.

Данный программный продукт может быть использован на компьютере или ноутбуке, так как программный продукт является Dekstop-продуктом.

5.2 Выполнение программы

5.2.1 Запуск программы

Для запуска программного продукта «Каркассон» необходимо скачать данную игру после чего запустить .exe файл в установленной папке (Рисунок 15).

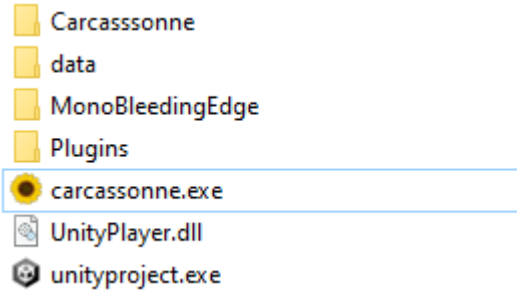


Рисунок 15 – Файл запуска игры

После этого игра запуститься сразу же (Рисунок 16).

Рисунок 16 – Игра «Каркассон».

5.2.2 Инструкция по работе с программой

После того как игра запустилась можно начинать играть. По умолчанию стоит 3 игрока. Слева находится имя игроков, их цвет, количество очков и количество фигурок которые у них осталось. На рисунках 17 – 22 расписан весь интерфейс игры (выделено красными квадратами).

Рисунок 17 – Очки игроков

Рисунок 18 – Ники игроков

Рисунок 19 – Уникальные значки и цвета игроков

Рисунок 20 – Количество оставшихся фишек у игроков

Рисунок 21 – Размещение пазла и количество оставшихся пазлов

Рисунок 22 – Кнопка постановки пазла и кнопка выхода из игры

Перейдём к тому, как же игроки будут играть в данную игру. Нажимаем на кнопку постановки пазла (Рисунок 22). Игрок который в данный момент ходит выделен ярко-голубым цветом (в нашем случае это Player 0, это видно на рисунке 22). После нажатии на кнопку нам выдаётся рандомный пазл и подсвечиваются места на которые можно поставить пазл (смотреть рисунок 23).

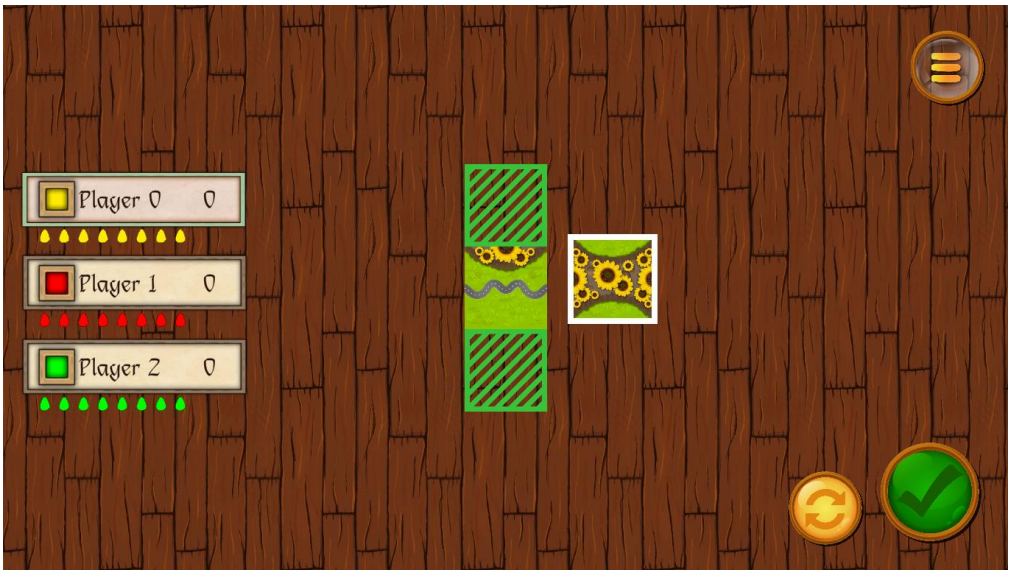


Рисунок 23 – Варианты постановки пазла

В данный момент можно либо перевернуть пазл, либо сразу же его поставить. Для того чтобы перевернуть пазл на 90 градусов, нажимаем на жёлтую кнопку переворота пазла (кнопка находится в нижнем правом углу на рисунке 23). На рисунке 24-25 показан процесс переворота пазла.

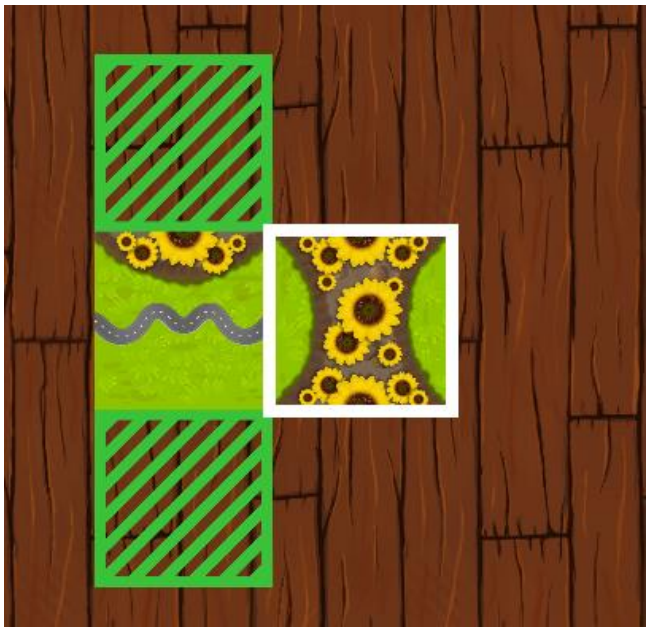


Рисунок 24 – Изначальный пазл

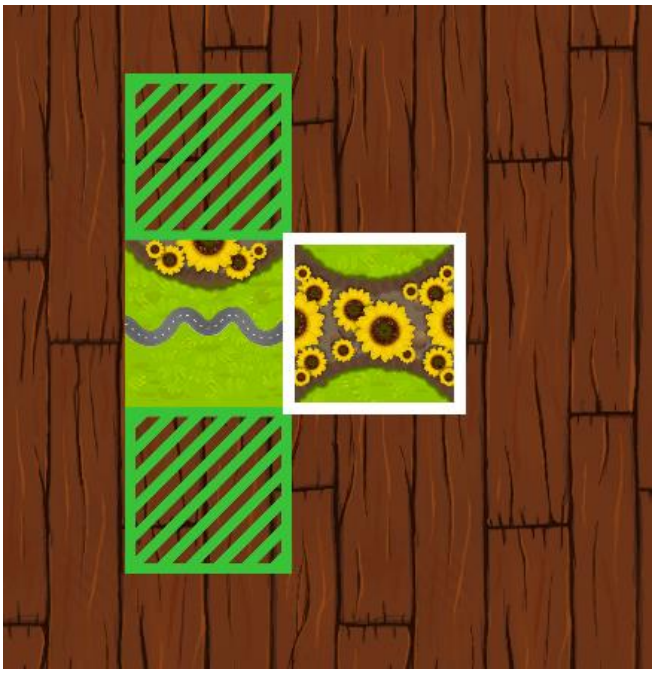


Рисунок 25 – Перевернутый на 90 градусов пазл

После того как нужный пользователю ракурс пазла подобран, он может поставить его нажав на зелёную кнопку с галочкой (смотреть рисунок 26).

Рисунок 26 – Кнопка постановки пазла

После того как игрок поставил пазл, ему даётся выбор, ставить ли свою фигуру на данный пазл, или же не ставить. Варианты постановки фигуры представлены на рисунке 27.

Рисунок 27 – Варианты постановки фигуры

Если игрок хочет поставить фигурку то он нажимает на один из «Плюсов». Если же он этого делать не хочет, он нажимает на кнопку «Отмена» при данном действии (смотреть Рисунок 28).

Рисунок 28 – Отмена постановки фигуры

В случае если игрок поставит фигурку, то при сборе какой либо комбинации на поле ему начислятся очки, так как его фигурка находится в зоне действии комбинации. Комбинации могут быть разные (собранная дорога, собранная поле с цветочками и т.п). Если же игрок не поставит фигурку некуда, очки за комбинацию на поле ему начисляться не будут.

Постанавленная фигурка на поле одним из игроков выглядит следующим образом (смотреть рисунок 29).

Рисунок 29 – Поставленная фигурка на поле

Игра будет продолжаться до тех пор, пока все пазлы не закончаться. В конце игры подсчитываются очки и выявляется победитель.

Некоторые комбинации представлены на рисунках 30 – 32.

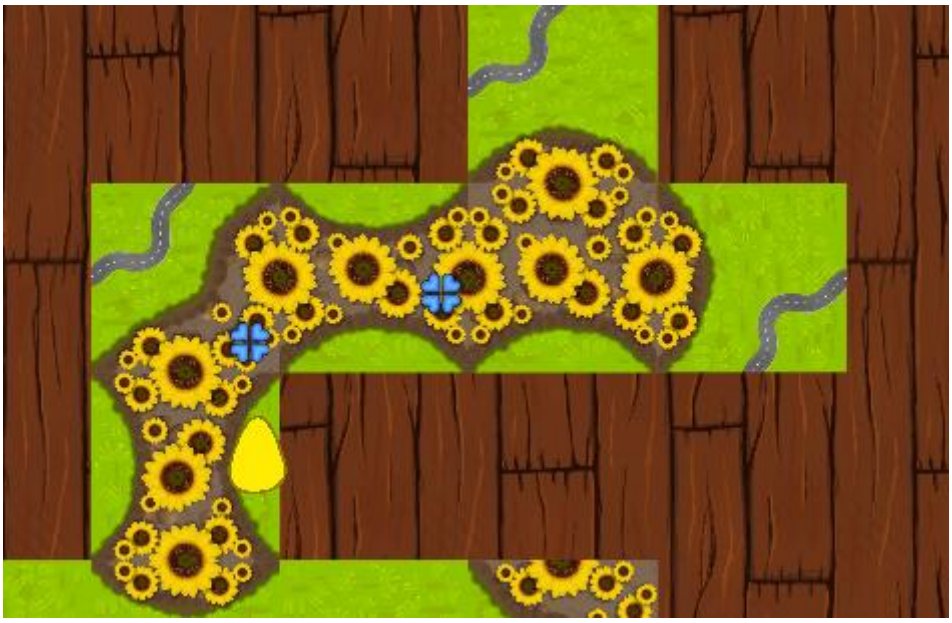


Рисунок 30 – Комбинации заполненных подсолнухов

Рисунок 31 – Мелкая комбинация заполненных подсолнухов

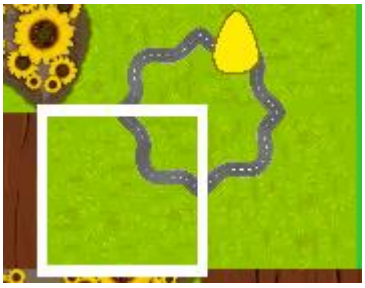


Рисунок 32 – Комбинация мелкой заполнненой дороги

Заключение

Целью данного учебного проекта являлась разработка игрового программного продукта «Каркассон».

В ходе реализации поставленной задачи были укреплены знания по использованию платформы Unity.

Следует также учесть, что в поставленной задачи был реализован простой и грамотный интерфейс который позволяет использовать программный продукт пользователю не обладающему дополнительными знаниями в области программирования.

После тщательного тестирования программного продукта были выявлены некоторые недоработки, которые были полностью исправлены на стадии проектирования или полностью исключены на стадии тестирования программы. В целом, при разработке программного продукта были выполнены не все условия, а именно: не реализована сетевая игра, главное меню, настройки, авторы и правила для обучения, отсутствует кнопка выхода, нету завершения игры а так же имеются некоторые баги при постановке некоторых пазлов, пазлы выглядят не так качественно как планировалось изначально, замест замков были выбраны подсолнухи, а так же нельзя настроить количество игроков, никнеймы и фотографии при запуске игры. Таким образом можно сказать что программный продукт «Каркассон» был реализован не совсем успешно, но реализован полноценный алгоритм игры который можно реально использовать и играть.

Список использованных источников

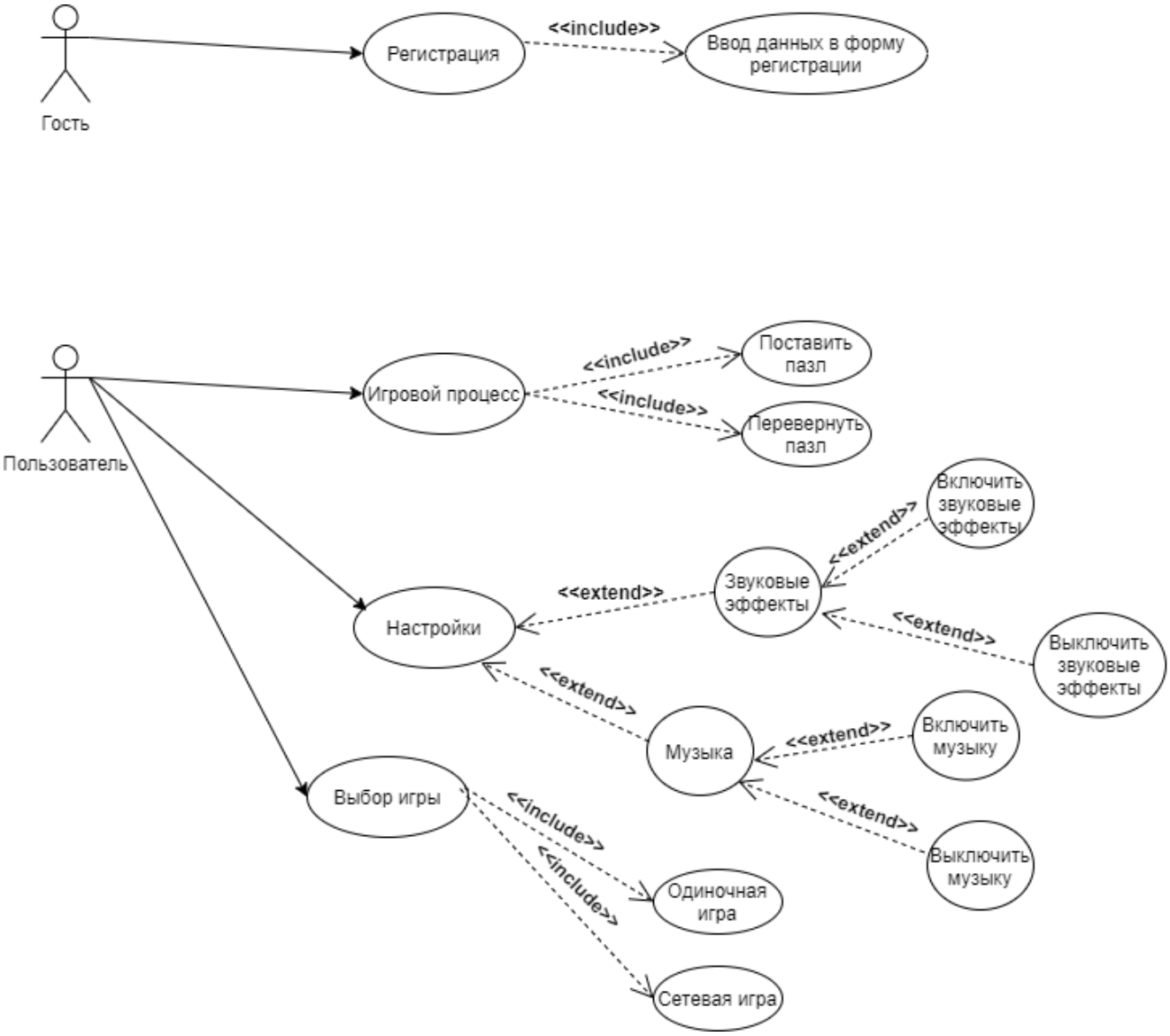
1 Как создать игру на Unity - https://blog.skillfactory.ru/kak-sozdat-igru-na-unity/

2 Установка Unity (пошаговая инструкция) - https://blog.skillfactory.ru/ustanovka-unity-instruktsiya/

3 Алгоритм Каркассон (модуль Scene) - https://github.ink/mariofv/Carcassone

Приложение А

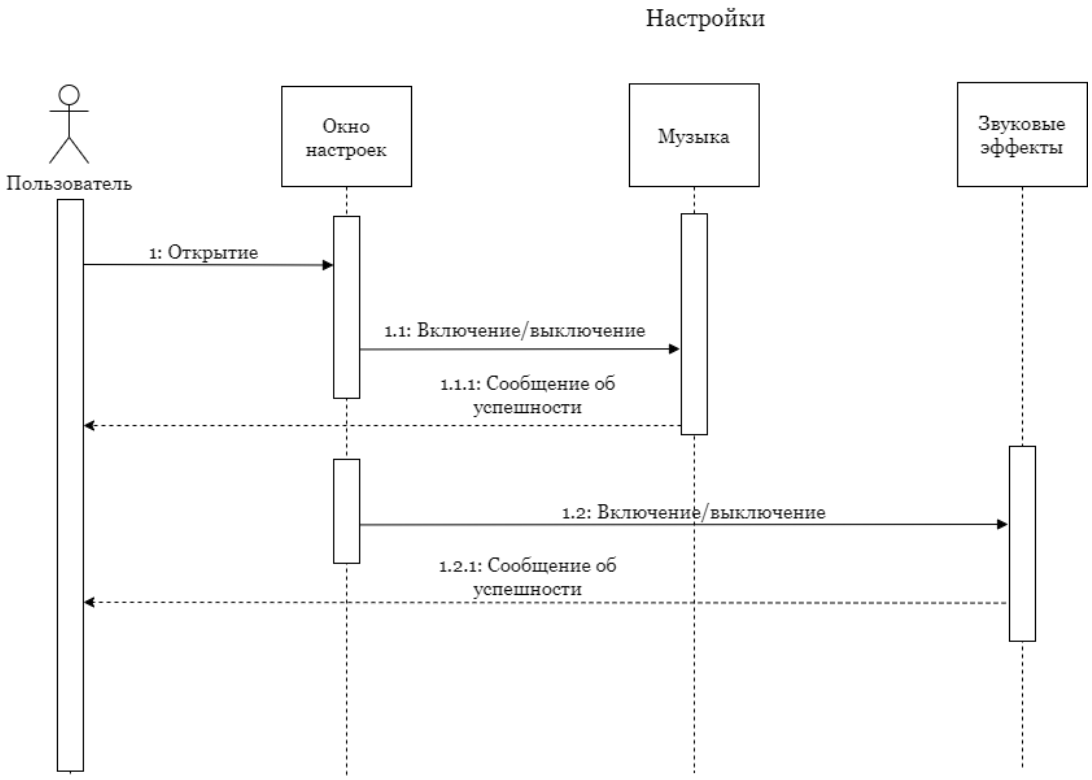
Диаграмма вариантов использования



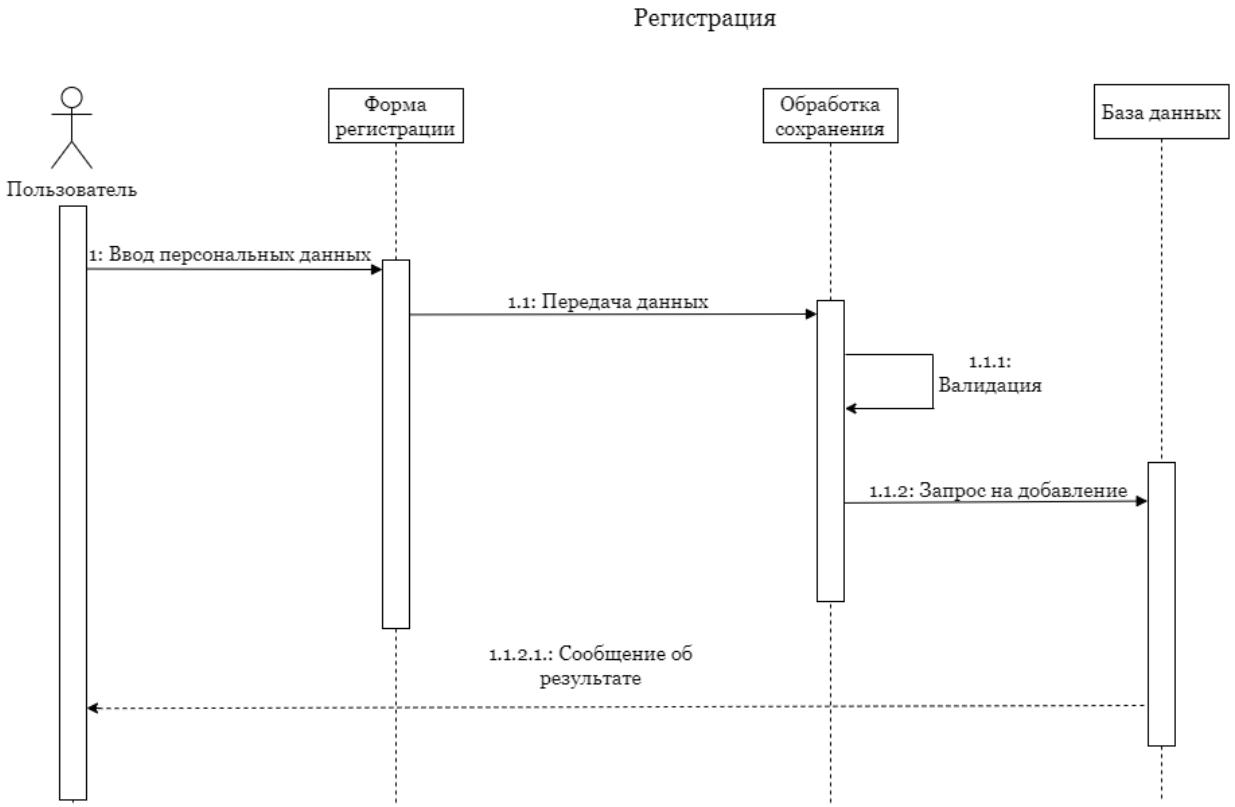
Приложение А.1 – Диаграмма вариантов использования

Приложение Б

Диаграмма последовательности



Приложение Б.1 – Диаграмма последовательности (Настройки)



Приложение Б.2 – Диаграмма последовательности (Регистрация)

Приложение В

Диаграмма структуры проекта

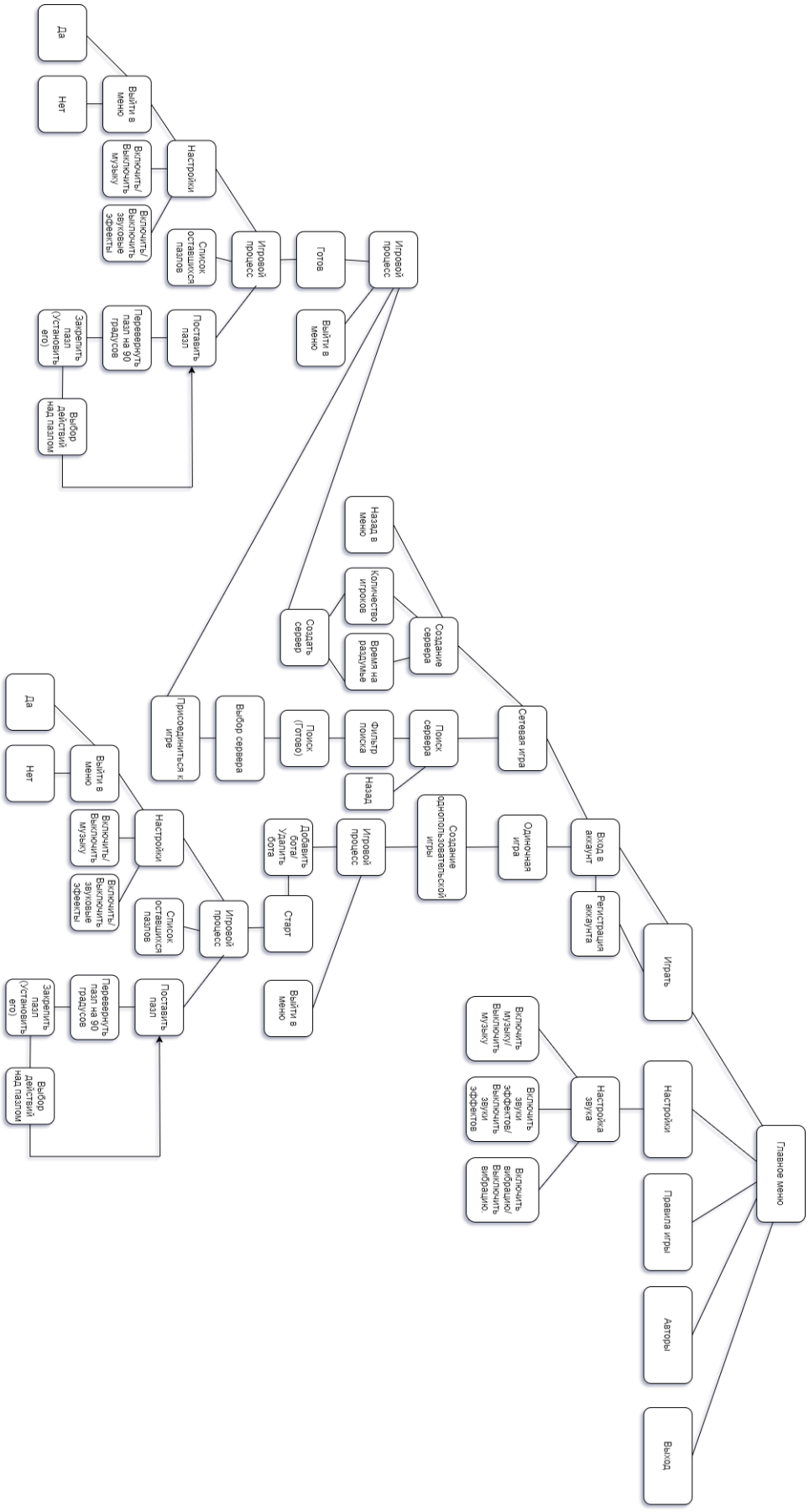


Диаграмма В.1 – Диаграмма структуры проекта

Приложение Г

Диаграмма деятельности

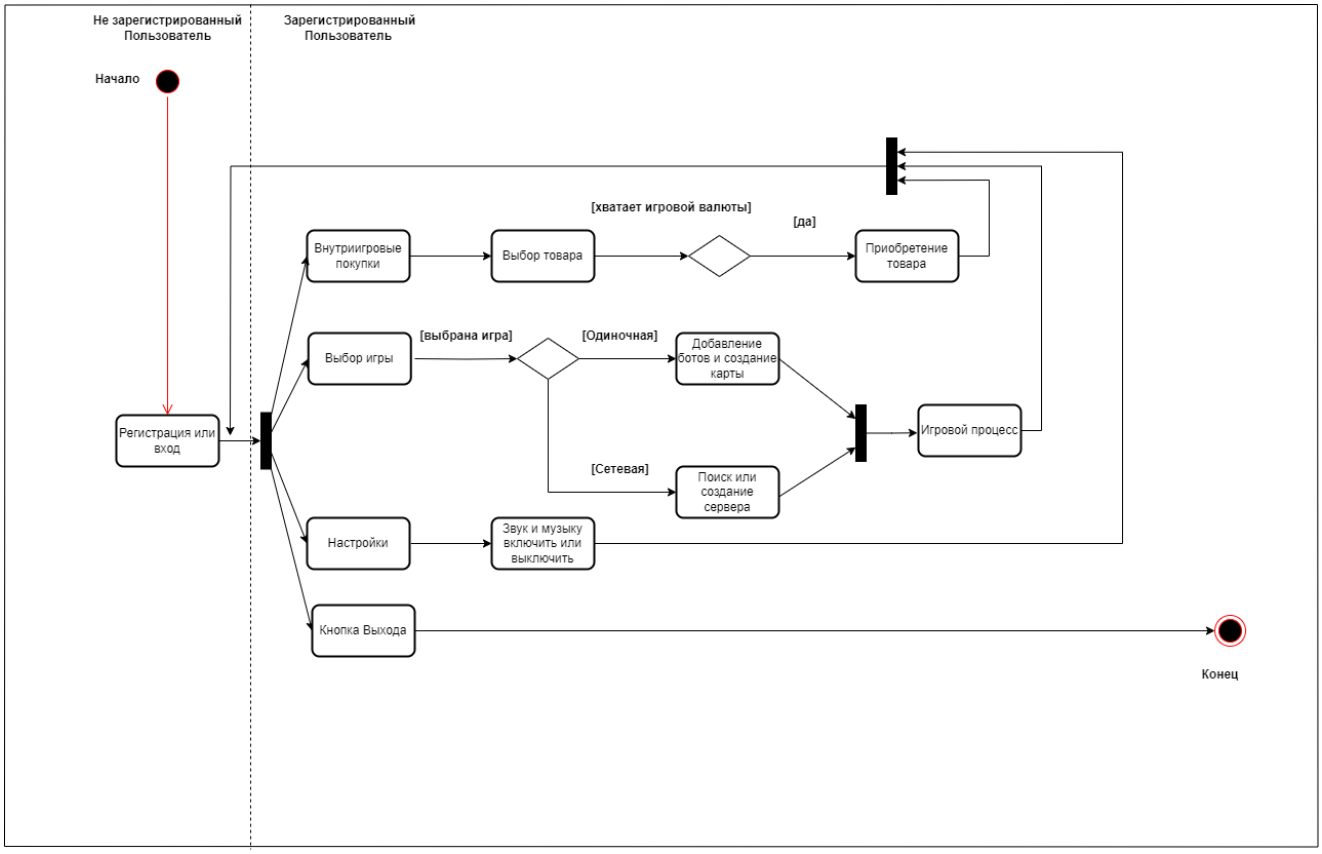
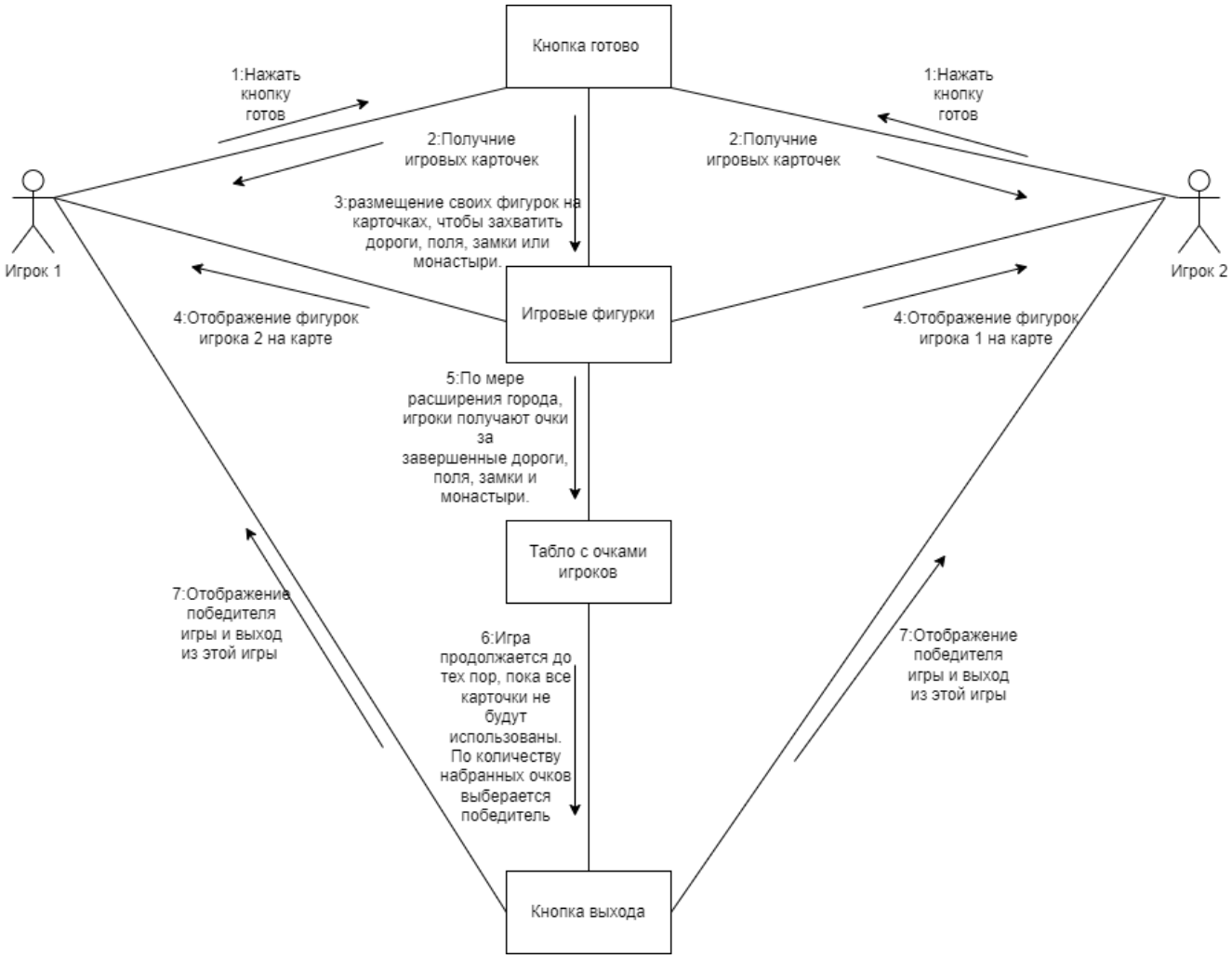


Диаграмма Г.1 – Диаграмма деятельности

Приложение Д

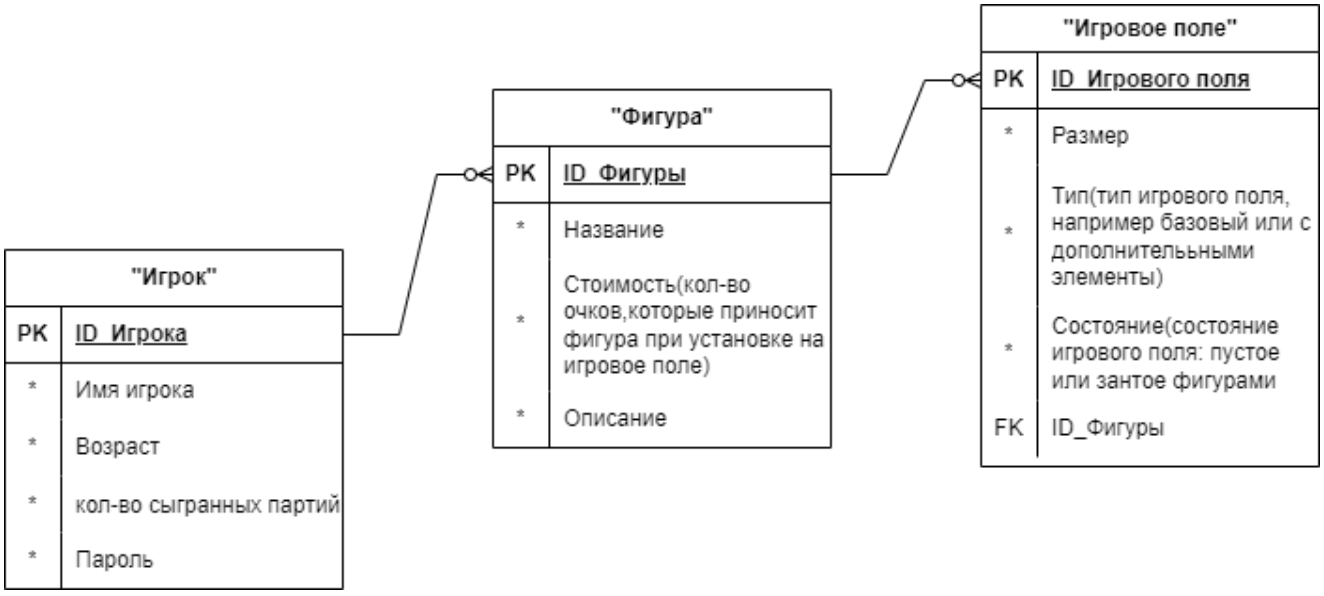
Диаграмма объектов



Приложение Д.1 – Диаграмма объектов

Приложение Е

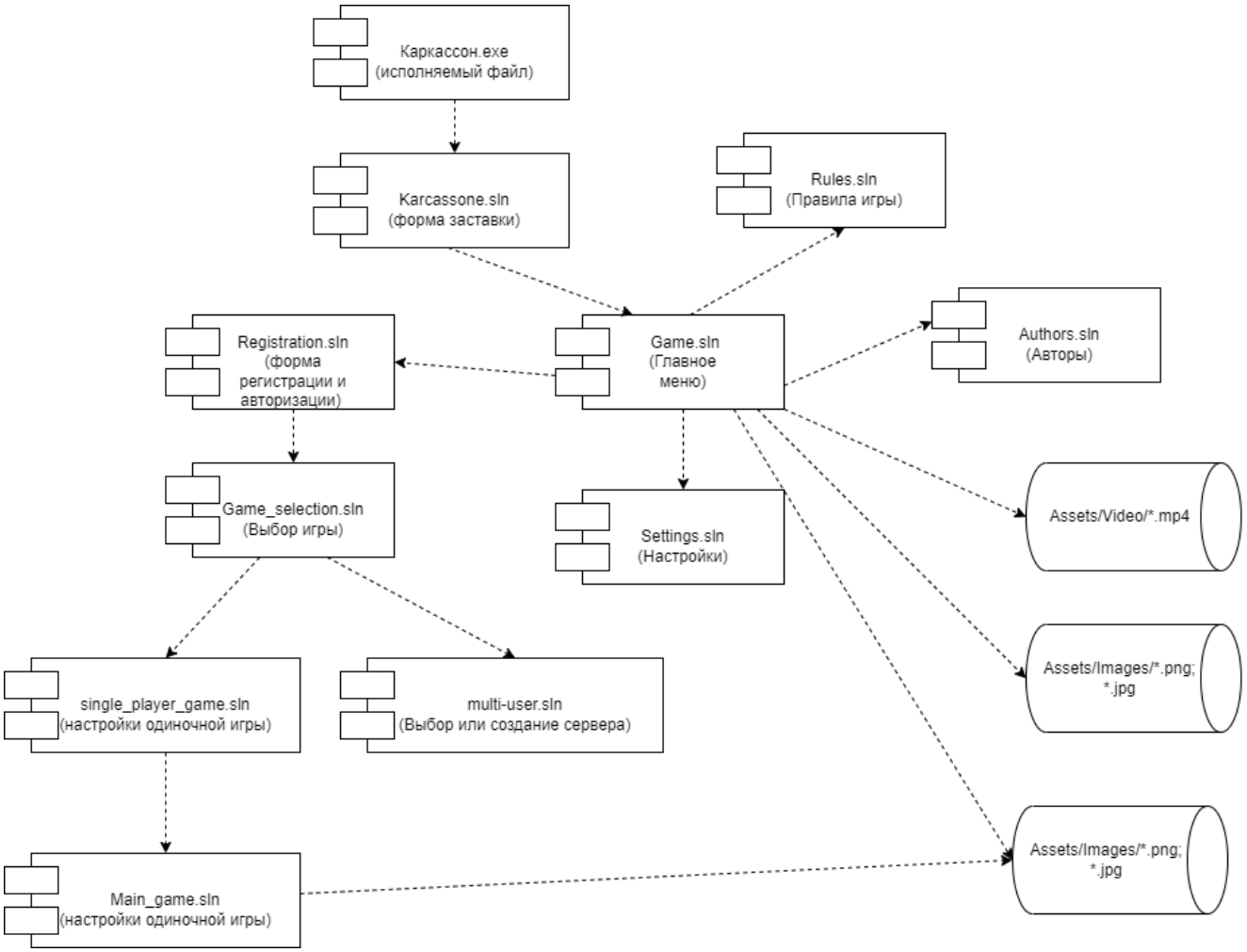
Модель данных



Приложение Е.1 – Модель данных

Приложение Ж

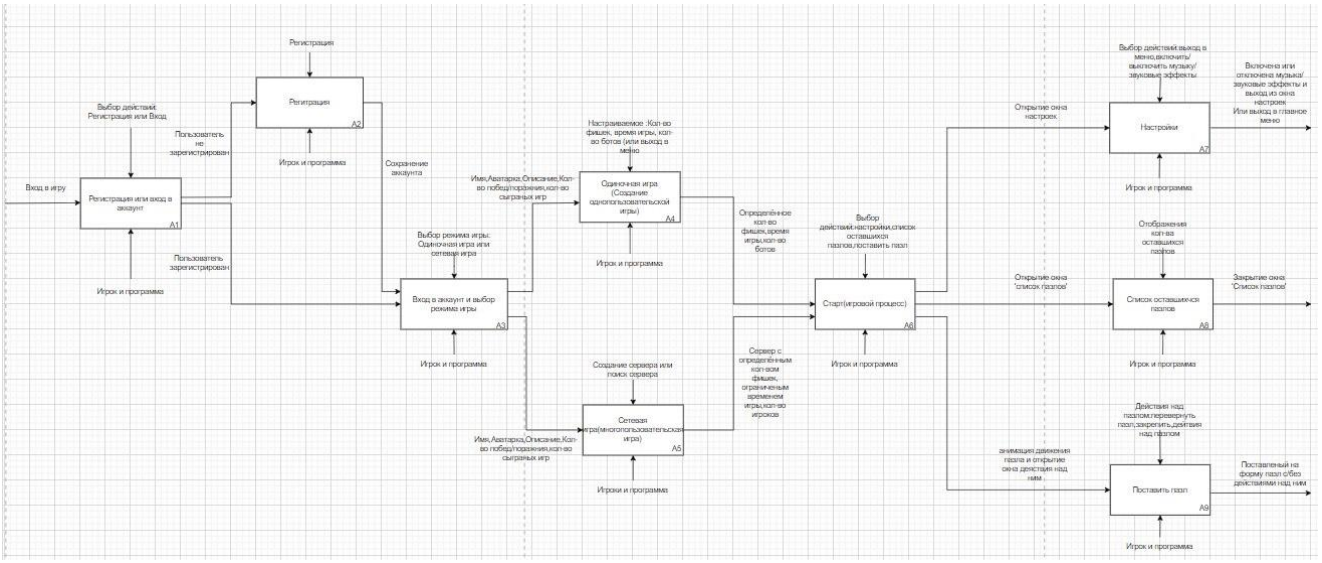
Диаграмма компонентов



Приложение Ж.1 – Диаграмма компонентов

Приложение З

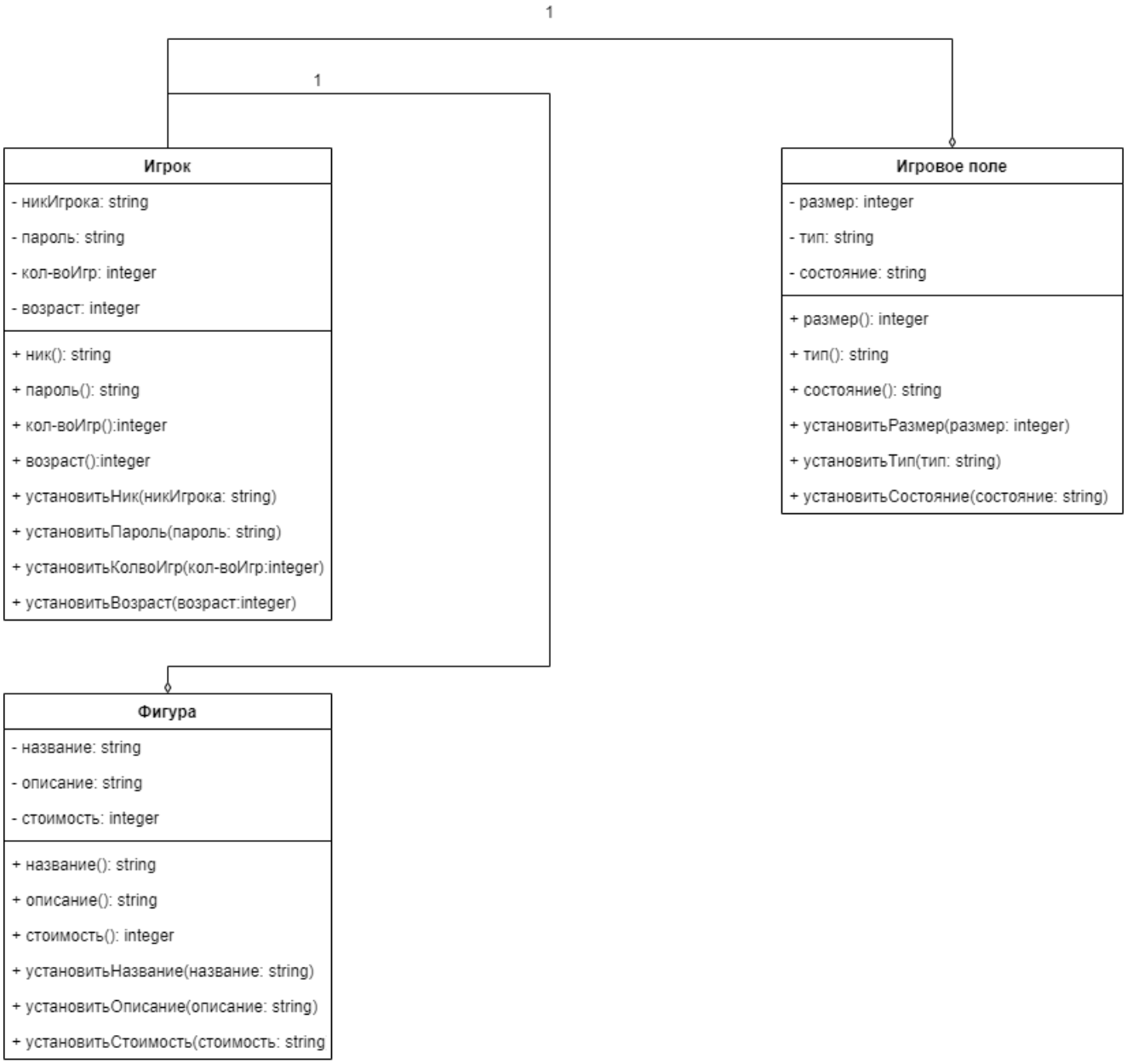
Модель бизнес-процессов



Приложение 3.1 – Модель бизнес-процессов

Приложение И

Диаграмма классов



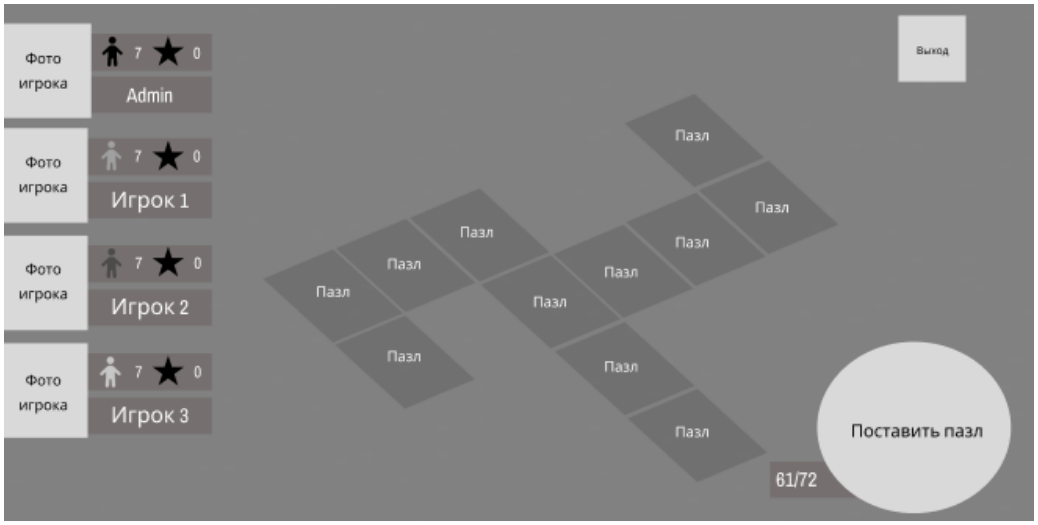
Приложение И.1 – Диаграмма классов

Приложений Й

Прототипы UX-интерфейсов

Приложение Й.1 – Прототип главного меню

Приложение Й.2 – Прототип формы регистрации

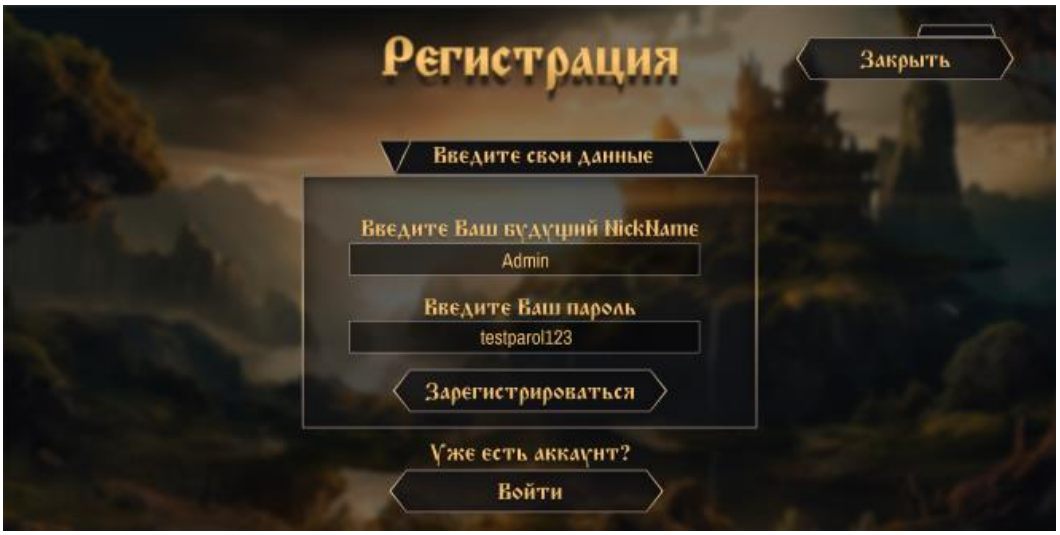


Приложение Й.2 – Прототип игрового процесса

Приложение К

Прототипы UI-интерфейсов

Приложение К.1 – Прототип главного меню



Приложение К.2 – Прототип формы регистрации

Приложение К.3 – Прототип игрового процесса

Приложение Л

Листинг программы

```
using System;
using System.Collections.Generic;

namespace CarcassonneGame
{
    class Program
    {
        static void Main(string[] args)
        {
            CarcassonneGame carcassonneGame = new CarcassonneGame();
            carcassonneGame.Start();
        }
    }

    class CarcassonneGame
    {
        private GameBoard board;
        private List<Puzzle> placedPuzzles;
        private List<Player> players;
        private Player currentPlayer;

        public CarcassonneGame()
        {
            board = new GameBoard();
            placedPuzzles = new List<Puzzle>();
            players = new List<Player>
            {
                new Player("Player 1"),
                new Player("Player 2")
            };
            currentPlayer = players[0];
        }

        public void Start()
        {
            while (true)
            {
                Console.WriteLine($"{currentPlayer.Name}'s turn");
                Console.WriteLine("1. Place Puzzle");
                Console.WriteLine("2. Rotate Puzzle");
                Console.WriteLine("3. Move Board");
                Console.WriteLine("4. Zoom In");
                Console.WriteLine("5. Zoom Out");
                Console.WriteLine("6. Place Figure");
                Console.WriteLine("7. Undo Placement");
                Console.WriteLine("8. Scoreboard");
                Console.WriteLine("9. End Turn");
                Console.WriteLine("0. Exit");

                int choice = Convert.ToInt32(Console.ReadLine());

                switch (choice)
                {
                    case 1:
                        PlacePuzzle();
                        break;
                    case 2:
                        RotatePuzzle();
                        break;
                    case 3:
                        MoveBoard();
                        break;
                    case 4:
                        ZoomIn();
```

```
                    break;
                case 5:
                    ZoomOut();
                    break;
                case 6:
                    PlaceFigure();
                    break;
                case 7:
                    UndoPlacement();
                    break;
                case 8:
                    DisplayScoreboard();
                    break;
                case 9:
                    EndTurn();
                    break;
                case 0:
                    Environment.Exit(0);
                    break;
                default:
                    Console.WriteLine("Invalid choice. Please try again.");
                    break;
            }
        }
    }

    private void PlacePuzzle()
    {
        Console.WriteLine("Enter puzzle coordinates (x y):");
        string[] coordinates = Console.ReadLine().Split(' ');
        int x = Convert.ToInt32(coordinates[0]);
        int y = Convert.ToInt32(coordinates[1]);

        Console.WriteLine("Enter puzzle type:");
        string puzzleType = Console.ReadLine();

        Puzzle puzzle = new Puzzle(puzzleType);
        if (board.PlacePuzzle(puzzle, x, y))
        {
            placedPuzzles.Add(puzzle);
            Console.WriteLine("Puzzle placed successfully!");
        }
        else
        {
            Console.WriteLine("Invalid puzzle placement.");
        }
    }

    private void RotatePuzzle()
    {
        Console.WriteLine("Enter puzzle index to rotate:");
        int index = Convert.ToInt32(Console.ReadLine());

        if (index >= 0 && index < placedPuzzles.Count)
        {
            placedPuzzles[index].Rotate();
            Console.WriteLine("Puzzle rotated successfully!");
        }
        else
        {
            Console.WriteLine("Invalid puzzle index.");
        }
    }
```

```
        private void MoveBoard()
        {
            Console.WriteLine("Enter movement direction (UP, DOWN, LEFT, RIGHT):");
            string directionStr = Console.ReadLine();
            Direction direction = (Direction)Enum.Parse(typeof(Direction),
directionStr.ToUpper());

            board.Move(direction);
            Console.WriteLine("Board moved successfully!");
        }

        private void ZoomIn()
        {
            board.ZoomIn();
            Console.WriteLine("Board zoomed in.");
        }

        private void ZoomOut()
        {
            board.ZoomOut();
            Console.WriteLine("Board zoomed out.");
        }

        private void PlaceFigure()
        {
            Console.WriteLine("Enter figure coordinates (x y):");
            string[] coordinates = Console.ReadLine().Split(' ');
            int x = Convert.ToInt32(coordinates[0]);
            int y = Convert.ToInt32(coordinates[1]);

            if (board.PlaceFigure(x, y, currentPlayer))
            {
                Console.WriteLine("Figure placed successfully!");
            }
            else
            {
                Console.WriteLine("Invalid figure placement.");
            }
        }

        private void UndoPlacement()
        {
            Console.WriteLine("Enter puzzle index to undo placement:");
            int index = Convert.ToInt32(Console.ReadLine());

            if (index >= 0 && index < placedPuzzles.Count)
            {
                Puzzle removedPuzzle = placedPuzzles[index];
                placedPuzzles.RemoveAt(index);
                board.UndoPlacement(removedPuzzle);
                Console.WriteLine("Placement undone successfully!");
            }
            else
            {
                Console.WriteLine("Invalid puzzle index.");
            }
        }

        private void DisplayScoreboard()
        {
            Console.WriteLine("Scoreboard:");
            foreach (Player player in players)
            {
                Console.WriteLine($"{player.Name}: {player.Score} points");
```

```
            }
        }

        private void EndTurn()
        {
            currentPlayer = (currentPlayer == players[0]) ? players[1] : players[0];
            Console.WriteLine($"{currentPlayer.Name}'s turn");
        }
    }

    enum Direction
    {
        UP,
        DOWN,
        LEFT,
        RIGHT
    }

    class GameBoard
    {
        private List<List<Puzzle>> board;
        private int zoomLevel;
        private int boardSize;

        public GameBoard()
        {
            board = new List<List<Puzzle>>();
            zoomLevel = 1;
            boardSize = 10;
            InitializeBoard();
        }

        private void InitializeBoard()
        {
            for (int i = 0; i < boardSize; i++)
            {
                List<Puzzle> row = new List<Puzzle>();
                for (int j = 0; j < boardSize; j++)
                {
                    row.Add(null);
                }
                board.Add(row);
            }
        }

        public bool PlacePuzzle(Puzzle puzzle, int x, int y)
        {
            if (IsValidPlacement(x, y) && board[x][y] == null)
            {
                board[x][y] = puzzle;
                return true;
            }

            return false;
        }

        public void UndoPlacement(Puzzle puzzle)
        {
            for (int i = 0; i < boardSize; i++)
            {
                for (int j = 0; j < boardSize; j++)
                {
                    if (board[i][j] == puzzle)
                    {
```

```
                    board[i][j] = null;
                }
            }
        }
    }

    private bool IsValidPlacement(int x, int y)
    {
        return x >= 0 && x < boardSize && y >= 0 && y < boardSize;
    }

    public void Move(Direction direction)
    {

        switch (direction)
        {
            case Direction.UP:

                break;
            case Direction.DOWN:

                break;
            case Direction.LEFT:

                break;
            case Direction.RIGHT:

                break;
            default:
                break;
        }
    }

    public void ZoomIn()
    {

        zoomLevel++;
        UpdateBoardSize();
    }

    public void ZoomOut()
    {

        if (zoomLevel > 1)
        {
            zoomLevel--;
            UpdateBoardSize();
        }
    }

    private void UpdateBoardSize()
    {

        boardSize = 10 * zoomLevel;
        InitializeBoard();
    }

    public bool PlaceFigure(int x, int y, Player player)
    {

        return false;
    }
}
```

```
    class Puzzle
    {
        private string type;
        private int rotation;

        public Puzzle(string type)
        {
            this.type = type;
            rotation = 0;
        }

        public void Rotate()
        {
            rotation = (rotation + 90) % 360;
        }
    }

    class Player
    {
        public string Name { get; }
        public int Score { get; set; }

        public Player(string name)
        {
            Name = name;
            Score = 0;
        }
    }
}
```