**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**

**По дисциплине «Web-технологии»**

**Тема: Разработка игры на языке JavaScript**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2023

# ЗАДАНИЕ

# на курсовую работу

|  |
| --- |
| Студентка Чернякова В.А. |
| Группа 1304 |
| Тема работы: Разработка игры на языке JavaScript. |
| Исходные данные:  Необходимо выполнить курсовую работу в соответствии с учебным пособием Беляев С.А. «Разработка игр на языке JavaScript».  Все делают в соответствии с общим заданием в соответствии с учебным пособием на «чистом» JavaScript (ES6). В группах по 1 человеку.   * Минимум 2 уровня игры * Реализованы все менеджеры в соответствии с учебным пособием (УП) * Есть таблица рекордов * Есть препятствия * Есть «интеллектуальные» противники и «бонусы» * Используются tiles с редактором Tiled (www.mapeditor.org) в соответствии с УП   Отчёт о выполнении курсовой работы включает: постановку задачи, описание решения (программы), скриншот программы.  Предполагаемый объем пояснительной записки:  Не менее 20 страниц  Дата выдачи задания: 11.09.2023  Дата сдачи работы: 20.11.2023  Дата защиты работы: 20.11.2023   |  |  |  | | --- | --- | --- | | Студентка |  | Чернякова В.А. | | Преподаватель |  | Беляев С.А. | |

# АННОТАЦИЯ

В ходе данной курсовой работе была создана 2D браузерная игра на языке JavaScript. В игре реализованы различные менеджеры, такие как управление картой, управление спрайтами, обработка событий, управление звуком, физическое моделирование, управление "интеллектом" противников и общий менеджер игры. Уровни игры созданы с использованием редактора тайлов Tiled, и в игре реализовано два уровня.

# SUMMARY

In the course of this course work, a 2D browser game in JavaScript was created. Various managers are implemented in the game, such as map management, sprite management, event processing, sound management, physical modeling, control of the "intelligence" of opponents and a general game manager. The levels of the game are created using the Tiles editor Tiles, and the game has two levels implemented.

# СОДЕРЖАНИЕ

[Введение 5](#_Toc151371617)

[1. Основные теоретические положения 6](#_Toc151371618)

[2. Карта игры 8](#_Toc151371619)

[2.1. Уровни игры 8](#_Toc151371620)

[2.2. Ассеты 9](#_Toc151371621)

[3. Разработка программного кода 10](#_Toc151371622)

[3.1. Менеджеры игры 10](#_Toc151371623)

[3.2. Сущности 15](#_Toc151371624)

[3.3. Браузерная часть 23](#_Toc151371625)

[Заключение 24](#_Toc151371626)

[Список использованных источников 25](#_Toc151371627)

[ПРИЛОЖЕНИЕ А 26](#_Toc151371628)

# Введение

Целью данной работы является создание 2D игры на языке программирования JavaScript.

На основе вышеизложенной цели были сформулированы задачи, необходимые для ее достижения:

* 1. Минимум 2 уровня игры
  2. Реализованы все менеджеры в соответствии с учебным пособием (УП)
  3. Есть таблица рекордов
  4. Есть препятствия
  5. Есть «интеллектуальные» противники и «бонусы»
  6. Используются tiles с редактором Tiled (www.mapeditor.org) в соответствии с УП

# Основные теоретические положения

Разработка 2D игры на языке JavaScript включает в себя несколько ключевых теоретических аспектов:

* Графика и отображение:

Организация отображения игровых объектов, таких как спрайты и фоны.

Обработка анимаций и перерисовка кадров для создания плавных движений.

* Управление пользователями и ввод:

Обработка пользовательского ввода с помощью событий мыши и клавиатуры.

Управление действиями персонажей или игровых элементов в ответ на ввод пользователя.

* Менеджеры игры:

Создание менеджеров для управления различными аспектами игры, такими как уровни, спрайты, звуки и физика.

Контроль игрового цикла, включая обновление и отрисовку сцены.

* Работа с уровнями и редактором тайлов:

Использование редактора тайлов, такого как Tiled, для создания уровней и расстановки игровых элементов.

Загрузка и управление уровнями в игре.

* Физика в игре:

Реализация простой физики для объектов в игре, таких как гравитация, столкновения и отскоки.

Интеграция физических законов для создания реалистичного поведения объектов.

* Аудио и звук:

Внедрение звуковых эффектов с использованием Web Audio API или других средств для воспроизведения звука.

Управление звуковыми ресурсами и их воспроизведением в различных сценах игры.

* Оптимизация и производительность:

Работа над оптимизацией кода и ресурсов для обеспечения плавного игрового процесса.

Использование техник кэширования, уменьшение числа запросов и оптимизация алгоритмов.

* Тестирование и отладка:

Разработка системы тестирования для проверки функциональности игры.

Использование отладчика для выявления и устранения ошибок в коде.

# карта игры

## Уровни игры

Уровни игры были спроектированы с помощью утилиты Tiled. Было построено 2 игровых уровня, на которых расположен игрок и несколько противников.

Также, на первом уровне расположена дверь для перехода на следующий уровень, а на втором уровне в конце карты расположена арена с главным босом игры.

Оба уровня имеют размерность 32 на 32 тайлов, каждый тайл размером в 16 пикселя. Карты первого и второго уровня представлены соответственно на рисунках 1 и 2.

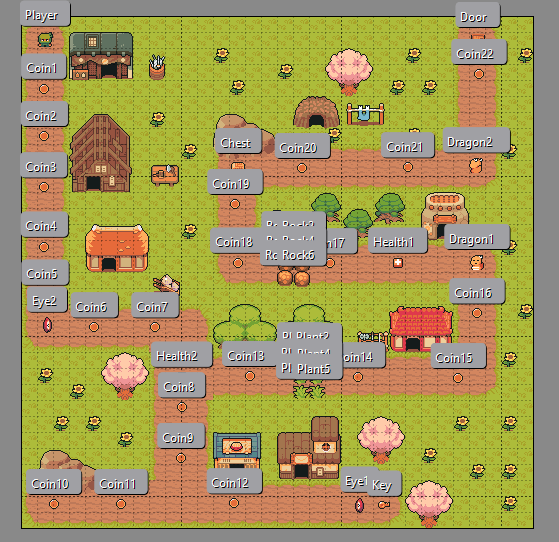


Рисунок 1 – первый уровень игры.



Рисунок 2 – второй уровень игры.

## Ассеты

Для игры были разработаны ассеты, содержащие следующие текстуры и анимации:

* Анимации игрока, врагов и «босса»: покой, бег, стрельба, смерть
* Текстуры окружения: стены, пол, фон, клетка-дверь, бонусы
* Текстуры оружия
* Текстуры для коллизий.

Объекты игрока, противника и босса игры содержат различные спрайты для анимации ходьбы, стрельбы, а также смерти персонажа и нахождения персонажа в состоянии покоя.

# Разработка программного кода

## Менеджеры игры

* Класс EventsManager отвечает за управление событиями в игре, такими как нажатия клавиш и клики мыши.

В конструкторе определены свойства bind и action. bind связывает коды клавиш с определенными действиями в игре.

Метод setup принимает объект canvas и устанавливает обработчики событий для мыши и клавиатуры.

Обработчики событий onMouseDown, onMouseUp, onKeyDown и onKeyUp определены для реакции на соответствующие события.

При нажатии клавиши (keydown) или ее отпускании (keyup) происходит проверка сопоставления кода клавиши с определенным действием из bind.

Если действие определено, то свойство action устанавливается в true при нажатии и false при отпускании клавиши.

* Класс GameManager управляет игровым процессом и взаимодействием между различными компонентами игры.

Инициализация:

Конструктор класса устанавливает различные свойства, такие как менеджеры объектов (factory), массив сущностей (entities), игрока (player) и другие параметры для отслеживания состояния игры.

Методы установки менеджеров:

setManager устанавливает связи между менеджерами игры, такими как менеджер спрайтов, карты, событий, физики и звуков.

Инициализация игрока:

initPlayer устанавливает объект игрока.

Управление сущностями:

kill добавляет объекты для удаления после обновления кадра.

update обновляет состояние игры, обрабатывая ввод игрока и обновляя все объекты в игре.

Обновление информации:

updateInfo обновляет информацию о текущем состоянии игры, такую как уровень, здоровье, количество монет и другие параметры.

Обработка конца игры:

gameOver вызывается при завершении игры, отображает сообщение о проигрыше и останавливает фоновую музыку.

Завершение игры:

finishGame вызывается при завершении уровня, обновляет и сохраняет итоговый счет, а затем перенаправляет на страницу рекордов.

Отрисовка объектов:

draw отрисовывает все объекты на игровом поле.

Загрузка уровня:

loadAll загружает уровень, устанавливает игрока, загружает спрайты, атласы, звуки и другие ресурсы.

Запуск игры:

play запускает игровой цикл, обновляя мир с определенной частотой.

Обновление мира:

updateWorld вызывает метод update для обновления состояния мира.

Установка игрока:

setPlayer отображает имя игрока.

Переход на новый уровень:

newLevel перенаправляет на следующий уровень, сохраняя итоговый счет.

* Класс MapManager отвечает за управление картой, тайлами и объектами в игре.

Инициализация свойств:

Определены свойства, такие как данные карты (mapData), массив тайловых слоев (tLayer), размер тайла (tSize), размер карты (mapSize), массив тайлсетов (tilesets) и другие параметры.

Установка менеджеров:

Метод setManager устанавливает связи между менеджерами игры, спрайтов и физики.

Загрузка карты:

Метод loadMap загружает данные карты из указанного файла.

Парсинг данных карты:

Метод parseMap разбирает данные карты, загружает изображения тайлсетов и устанавливает соответствующие параметры.

Получение тайлсета по индексу тайла:

Метод getTileset возвращает объект тайлсета по индексу тайла.

Получение данных о тайле:

Метод getTile возвращает информацию о тайле по его индексу.

Отрисовка карты:

Метод draw отрисовывает тайлы карты на холсте.

Парсинг объектов на карте:

Метод parseEntities разбирает объекты, находящиеся на карте, создает соответствующие экземпляры объектов и добавляет их в массив сущностей.

Получение индекса тайла по координатам:

Метод getTilesetIdx возвращает индекс тайла по указанным координатам на карте.

Центрирование вида на указанных координатах:

Метод centerAt центрирует вид игрового окна на заданных координатах.

* Класс PhysicManager, отвечающий за физическое взаимодействие объектов в игре.

Инициализация разрешенных тайлов:

Определен объект allowed, содержащий разрешенные тайлы для движения игрока на различных уровнях.

Установка менеджеров:

Метод setManager устанавливает связь с менеджером игры и менеджером карты.

Метод обновления физики объекта:

Метод update обрабатывает обновление позиции объекта в зависимости от его движения и столкновений с тайлами и другими объектами на карте.

Если объект сталкивается с другим объектом и у него определен метод onTouchEntity, вызывается этот метод.

Если объект сталкивается с неразрешенным тайлом и у него определен метод onTouchMap, вызывается этот метод.

Если объект может двигаться на текущий тайл (разрешенный тайл), и если на тайле нет других объектов или есть только монета или звезда, то позиция объекта обновляется.

Проверка сущности по координатам:

Метод entityAtXY проверяет, есть ли другие сущности в указанных координатах (x, y). Возвращает объект сущности, если он есть, иначе возвращает null.

* Класс SoundManager отвечает за управление звуками в игре.

Инициализация:

Конструктор класса инициализирует необходимые свойства, такие как звуковые клипы (clips), аудио-контекст (context), узел усиления звука (gainNode), фоновая музыка (fon) и флаг загрузки (loaded).

Инициализация аудио-контекста:

Метод init создает аудио-контекст и узел усиления звука.

Установка менеджера игры:

Метод setManager устанавливает связь с менеджером игры.

Загрузка звукового файла:

Метод load загружает звуковой файл по указанному пути и выполняет обратный вызов после загрузки.

Загрузка массива звуковых файлов:

Метод loadArray загружает массив звуковых файлов и выполняет обратный вызов после загрузки всех файлов.

Воспроизведение звука:

Метод play воспроизводит звук из загруженных клипов по указанному пути с определенными настройками (громкость, зацикливание).

Остановка фоновой музыки:

Метод stopFon останавливает воспроизведение текущей фоновой музыки.

Воспроизведение звука в мире:

Метод playWorldSound воспроизводит звук в зависимости от расстояния от игрока, при этом учитывается его позиция на карте.

* Класс SpriteManager отвечает за управление спрайтами в игре.

Инициализация:

Конструктор класса инициализирует необходимые свойства, такие как изображение (image), массив спрайтов (sprites) и флаги загрузки (imgLoaded, jsonLoaded).

Установка менеджера карты:

Метод setManager устанавливает связь с менеджером карты.

Загрузка атласа спрайтов:

Метод loadAtlas отправля запрос на загрузку JSON-файла атласа спрайтов и вызывает метод loadImg для загрузки изображения.

Загрузка изображения:

Метод loadImg загружает изображение спрайтов.

Парсинг атласа спрайтов:

Метод parseAtlas разбирает JSON-структуру атласа спрайтов и заполняет массив спрайтов.

Отрисовка спрайта:

Метод drawSprite отрисовывает спрайт по указанным координатам (x, y). Если изображение или JSON атласа еще не загружены, он повторно вызывается через некоторое время.

Получение спрайта по имени:

Метод getSprite возвращает объект спрайта по его имени.

## Сущности

* Класс Entity служит в качестве базового класса для различных игровых сущностей. В данном случае, у него есть базовые свойства для позиции (pos\_x, pos\_y) и размера (size\_x, size\_y).

Конструктор класса устанавливает начальные значения для координат и размеров сущности.

* Класс Boss наследуется от класса Entity и представляет сущность босса в игре.

Конструктор класса инициализирует свойства для управления движением (move\_x, move\_y) и состоянием огня (fireState).

Метод draw использует spriteManager для отрисовки спрайта босса на указанных координатах.

Метод update вызывает функцию checkPlayer и обновляет физическое состояние босса.

Метод checkPlayer вычисляет расстояние между боссом и игроком. Если расстояние меньше 300, босс начинает атаковать.

Создается экземпляр сущности Fire, который представляет собой атаку босса. Устанавливаются параметры направления, скорости и начальной позиции огня.

Экземпляр огня добавляется в массив сущностей игры (gameManager.entities).

Устанавливается таймер, чтобы управлять периодичностью атаки босса.

Метод kill добавлен для удаления босса из игры.

* Класс Chest наследуется от класса Entity. В данном случае, Chest представляет объект сундука в игре.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливается свойство cadr со значением 'close'. Это относится к текущему кадру анимации сундука.

Метод draw использует spriteManager для отрисовки спрайта сундука на указанных координатах.

Используется свойство cadr для выбора соответствующего кадра анимации сундука ('chest\_close' или другие).

* Класс Coin наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливается свойство cadr со значением 1. Это относится к текущему кадру анимации монеты.

Метод draw использует spriteManager для отрисовки спрайта монеты на указанных координатах.

Используется свойство cadr для выбора соответствующего кадра анимации монеты ('Coin\_1', 'Coin\_2', 'Coin\_3', 'Coin\_4').

Метод update обновляет текущий кадр анимации монеты. Если cadr меньше 4, увеличивает его на 1; в противном случае, сбрасывает cadr в 1.

Это реализация анимации мигания монеты.

Метод kill добавлен для удаления монеты из игры. Это вызывается, когда монета собрана игроком.

* Класс Cut наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливаются свойства move\_x и move\_y со значениями 0.

Метод draw использует spriteManager для отрисовки спрайта объекта Cut на указанных координатах.

Используется свойство cadr для выбора соответствующего кадра анимации объекта.

Метод update обновляет текущий кадр анимации объекта Cut. Если cadr меньше 8, увеличивает его на 1; в противном случае, воспроизводит звук "map/music/cut.mp3", сбрасывает cadr в 1 и устанавливает move\_x и move\_y в соответствии с направлением (dir), после чего вызывает метод kill.

Метод onTouchEntity вызывается при соприкосновении с другим объектом. Если объект является экземпляром класса Plant, то оба объекта уничтожаются.

Метод kill добавлен для удаления объекта Cut из игры. Вызывается после завершения анимации объекта или при взаимодействии с другим объектом.

* Класс Door наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливается свойство cadr со значением 'close'.

Метод draw использует spriteManager для отрисовки спрайта объекта Door на указанных координатах.

Используется свойство cadr для выбора соответствующего кадра анимации объекта.

* Класс Dragon наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливаются свойства, такие как move\_x, move\_y, speed, dir (направление движения) и followPlayer (флаг следования за игроком).

Метод draw использует spriteManager для отрисовки спрайта объекта Dragon на указанных координатах. Выбор спрайта осуществляется в зависимости от направления движения (dir) и номера дракона.

Метод update проверяет расстояние между драконом и игроком, и в зависимости от этого регулирует скорость и направление движения дракона. Если дракон находится близко к игроку, он начинает следовать за ним.

Затем вызывается метод physicManager.update для обновления физики дракона.

Метод checkPlayer определяет, находится ли игрок в близкой зоне видимости дракона. Если да, дракон начинает следовать за игроком, изменяя свое направление.

Методы onTouchEntity и onTouchMap обрабатывают столкновения дракона с другими сущностями и элементами карты. Например, если дракон касается игрока, уменьшается количество жизней игрока, если он касается ключа или монеты, его позиция обновляется.

Метод kill вызывается, когда дракон уничтожается, увеличивает финальный счет игры и вызывает метод kill менеджера игры для удаления объекта дракона.

* Класс Eye наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливаются свойства, такие как move\_x, move\_y, speed, dir (направление движения) и followPlayer (флаг следования за игроком).

Метод draw использует spriteManager для отрисовки спрайта объекта Eye на указанных координатах. Выбор спрайта осуществляется в зависимости от направления движения (dir) и номера глаза.

Метод update проверяет расстояние между глазом и игроком, и в зависимости от этого регулирует скорость и направление движения глаза. Если глаз находится близко к игроку, он начинает следовать за ним.

Затем вызывается метод physicManager.update для обновления физики глаза.

Метод checkPlayer определяет, находится ли игрок в близкой зоне видимости глаза. Если да, глаз начинает следовать за игроком, изменяя свое направление.

Методы onTouchEntity и onTouchMap обрабатывают столкновения глаза с другими сущностями и элементами карты. Например, если глаз касается игрока, уменьшается количество жизней игрока, если он касается ключа или монеты, его позиция обновляется.

Метод kill вызывается, когда глаз уничтожается, увеличивает финальный счет игры и вызывает метод kill менеджера игры для удаления объекта глаза.

* Класс Fire наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливаются свойства move\_x и move\_y.

Метод draw использует spriteManager для отрисовки спрайта огня (fire\_${this.cadr}) на указанных координатах.

Метод update обновляет кадр огня и воспроизводит звук огня. Затем вызывается метод physicManager.update для обновления физики огня.

Метод onTouchEntity обрабатывает столкновения огня с другими сущностями. Если огонь касается игрока (Player), уменьшается количество жизней игрока. Если количество жизней становится меньше или равно 0, вызывается метод kill для уничтожения игрока.

Если огонь касается звезды (Star) или здоровья (Health), его позиция обновляется.

Метод onTouchMap вызывает метод kill, когда огонь сталкивается с элементом карты.

Метод kill вызывается, когда огонь уничтожается, и вызывает метод kill менеджера игры для удаления объекта огня.

* Класс Health наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Метод draw использует spriteManager для отрисовки спрайта здоровья (Health) на указанных координатах.

Метод kill вызывается, когда здоровье уничтожается, и вызывает метод kill менеджера игры для удаления объекта здоровья.

* Класс Key наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Метод draw использует spriteManager для отрисовки спрайта ключа (Key) на указанных координатах.

Метод kill вызывается, когда ключ уничтожается, и вызывает метод kill менеджера игры для удаления объекта ключа.

* Класс Pickaxe наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

У Pickaxe есть свойства move\_x и move\_y, которые используются для определения направления движения.

Метод draw использует spriteManager для отрисовки спрайта кирки (pickaxe) на указанных координатах.

Метод update обновляет состояние объекта. В данном случае, он увеличивает кадр анимации и, когда достигает предела, воспроизводит звук кирки и вызывает метод kill для удаления объекта.

Метод onTouchEntity вызывается, когда кирка сталкивается с объектом. В данном случае, если объект — это Rock, то вызывается метод kill для удаления камня и кирки.

Метод kill вызывается, когда кирка уничтожается, и вызывает метод kill менеджера игры для удаления объекта кирки.

* Класс Plant наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Метод draw использует spriteManager для отрисовки спрайта растения (plant) на указанных координатах.

Метод kill вызывается, когда растение уничтожается, и вызывает метод kill менеджера игры для удаления объекта растения.

* Класс Player наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Устанавливаются начальные значения для свойств, таких как move\_x, move\_y, speed, cadr и состояния оружия (starState, cutState, pickaxeState).

Метод draw использует spriteManager для отрисовки спрайта игрока на указанных координатах. Он также учитывает текущее направление движения игрока.

Метод update вызывается для обновления состояния игрока.

Метод onTouchEntity обрабатывает взаимодействие с различными объектами (монеты, ключи, здоровье, сундуки, двери).

Методы star, cut и pickaxe создают соответствующие объекты (звезда, секатор, кирка) и добавляют их в массив сущностей при определенных условиях.

Метод kill вызывается, когда игрок умирает, и вызывает метод gameOver менеджера игры.

* Класс Rock наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

Метод draw использует spriteManager для отрисовки спрайта камня на указанных координатах.

Метод kill вызывается, когда камень должен быть уничтожен, и он вызывает метод kill менеджера игры, чтобы удалить объект камня.

* Класс Slime наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

В данном случае, конструктор инициализирует свойства, связанные с движением (move\_x, move\_y), скоростью (speed), направлением (dir) и флагом следования за игроком (followPlayer).

Метод draw использует spriteManager для отрисовки спрайта слизня на указанных координатах в зависимости от направления.

Метод update вызывает checkPlayer, чтобы решить, следовать ли за игроком, и затем обновляет физические параметры объекта.

Метод checkPlayer вычисляет расстояние между слизнем и игроком.

Если расстояние меньше 150, слизень начинает следовать за игроком, изменяя свои параметры движения и направления.

Методы onTouchEntity и onTouchMap обрабатывают столкновения с игроком и картой соответственно.

Метод kill вызывается, когда слизень должен быть уничтожен, и он уведомляет менеджер игры об удалении этого объекта и добавляет очки игроку.

* Класс Star наследуется от класса Entity.

Конструктор класса вызывает конструктор родительского класса (super()), чтобы унаследовать его свойства.

В данном случае, конструктор инициализирует свойства, связанные с движением (move\_x, move\_y).

Метод draw использует spriteManager для отрисовки спрайта звезды на указанных координатах.

Метод update обновляет кадр звезды и изменяет ее координаты в зависимости от направления.

Метод onTouchEntity обрабатывает столкновения с различными типами объектов в игре.

Если звезда сталкивается с враждебными объектами, такими как Eye, Dragon, Slime или Boss, она убивает их и затем удаляется.

Если звезда сталкивается с объектами, такими как Coin, Health, Key, Chest, Door, она просто меняет свои координаты.

Если звезда сталкивается с препятствиями, такими как Plant или Rock, она умирает.

Метод onTouchMap вызывается, если звезда сталкивается со структурой карты (по индексу idx).

Метод kill вызывается, когда звезда должна быть уничтожена, и он уведомляет менеджер игры об удалении этого объекта.

## Браузерная часть

Для взаимодействия пользователя с игрой была разработана браузерная часть.

Она включает в себя html страницы страта, первого и второго уровня, таблицы рекордов. Для каждой html страницы написан css для стилизации и файлы типа JavaScript для обработки действий пользователя или запуска игры.

В приложении А представлены скриншоты программы.

# Заключение

В ходе выполнения курсовой работы была создана браузерная игра на языке JavaScript. Были разработаны разнообразные менеджеры, ответственные за различные аспекты игры, такие как карта, объекты, звуковое сопровождение и прочее, а также универсальный «движок», позволяющий реализовывать игры.

# Список использованных источников

1. Беляев С. А. Разработка игр на языке JavaScript: Лань, 2020. — 128 с.

# ПРИЛОЖЕНИЕ А

# скриншоты программы

На рисунках 3 – 7 представлены скриншоты программы.

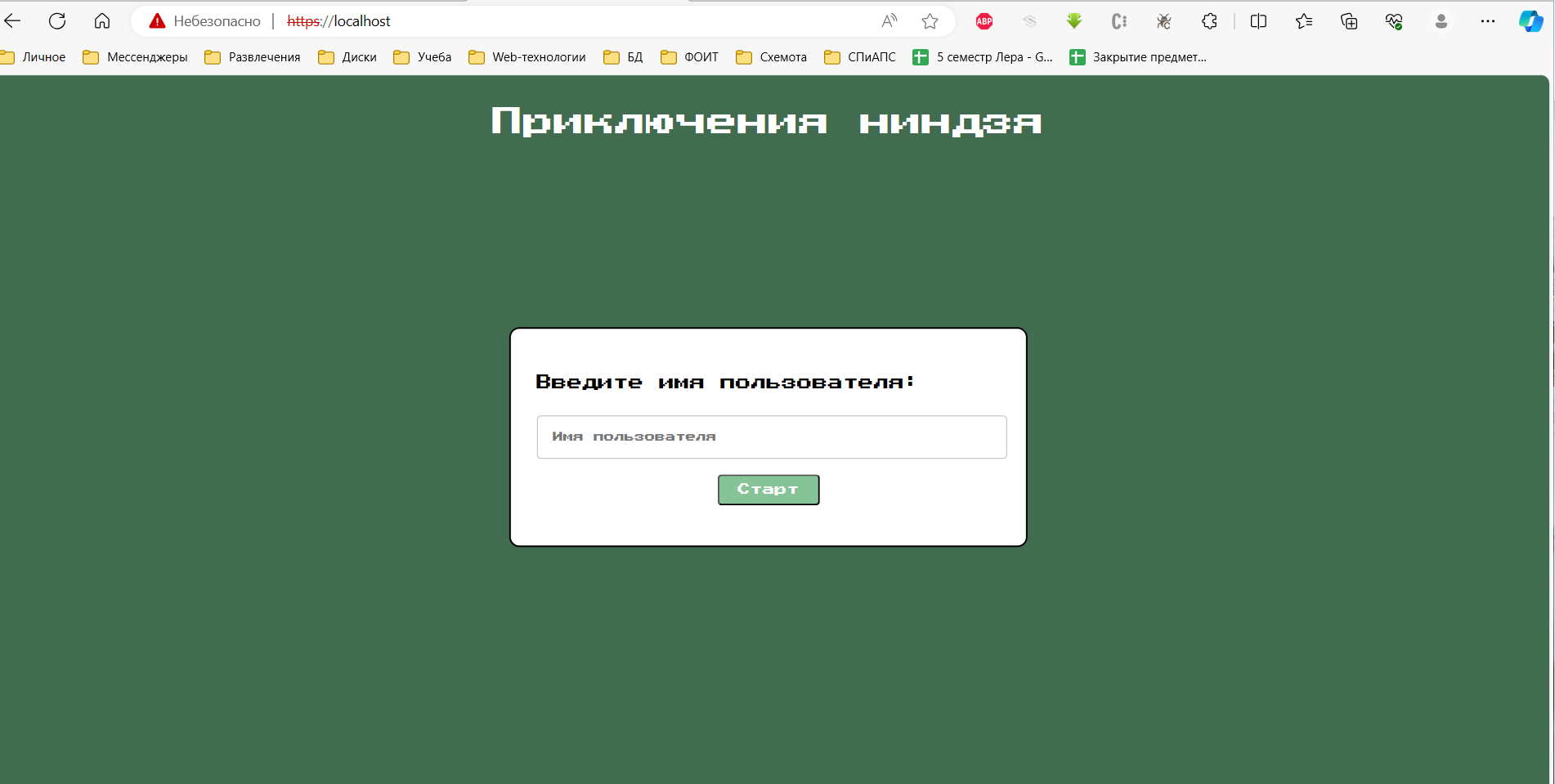


Рисунок 3– стартовая страница.

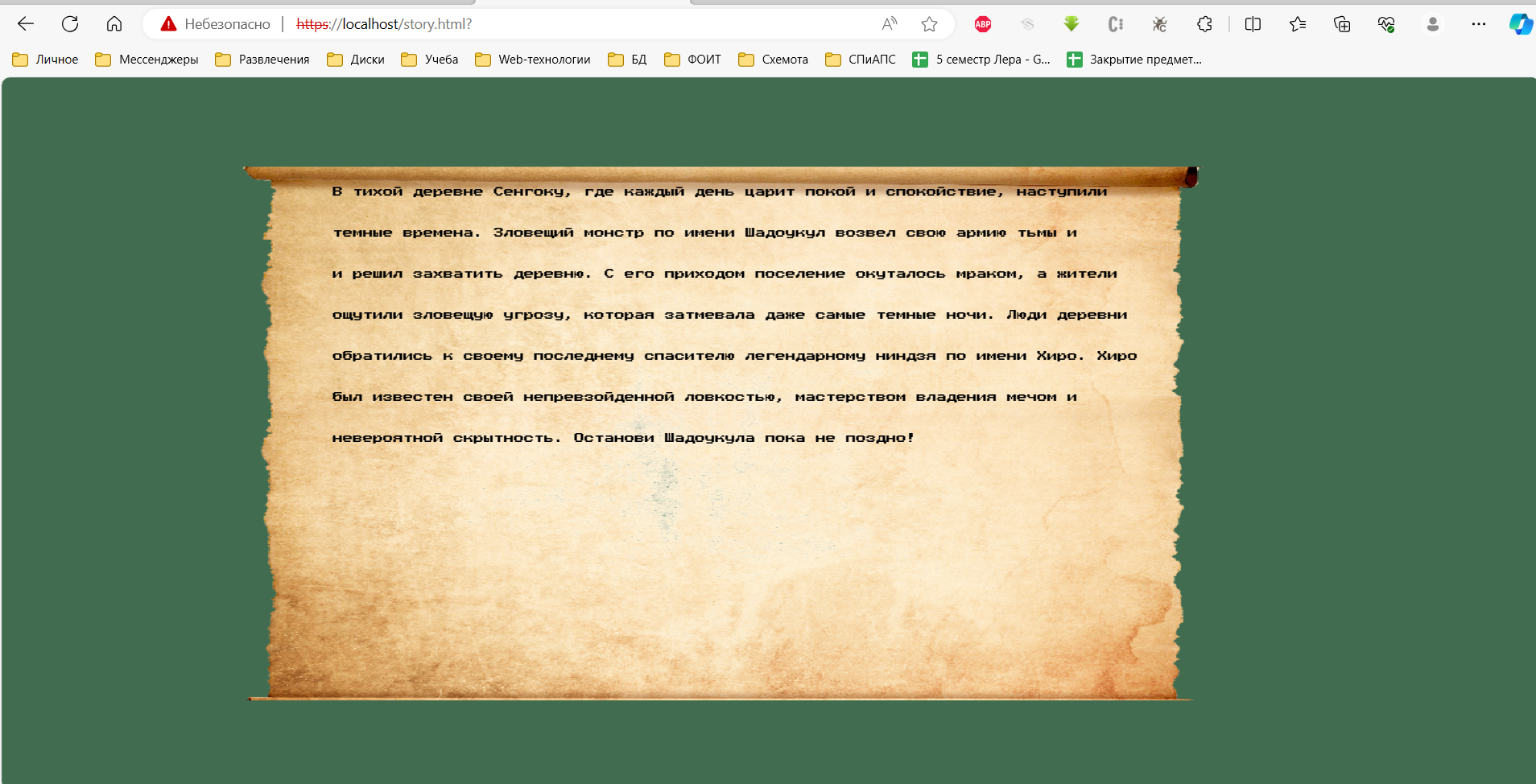


Рисунок 4 – страница с историей.

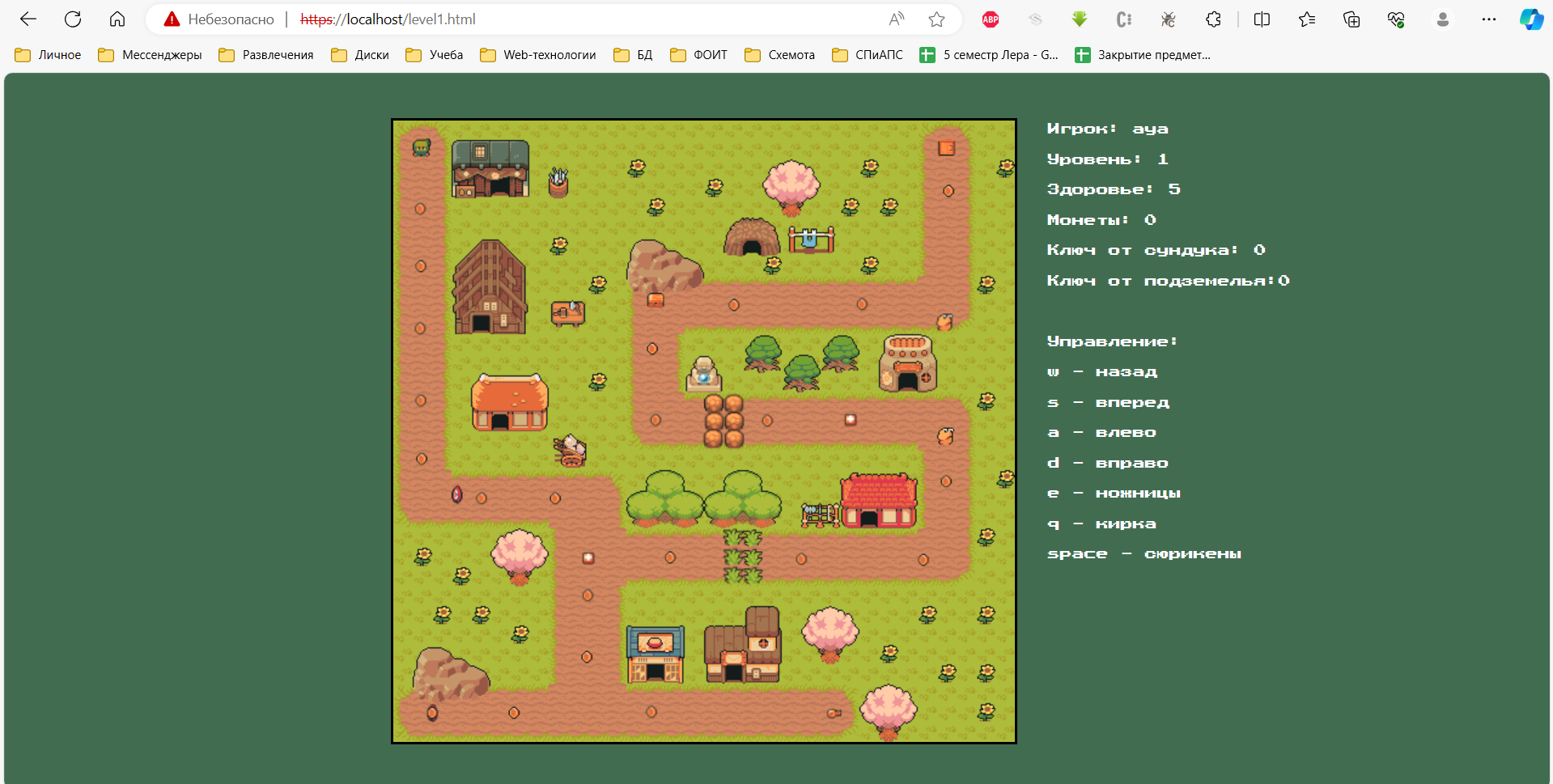


Рисунок 5 – страница первого уровня.

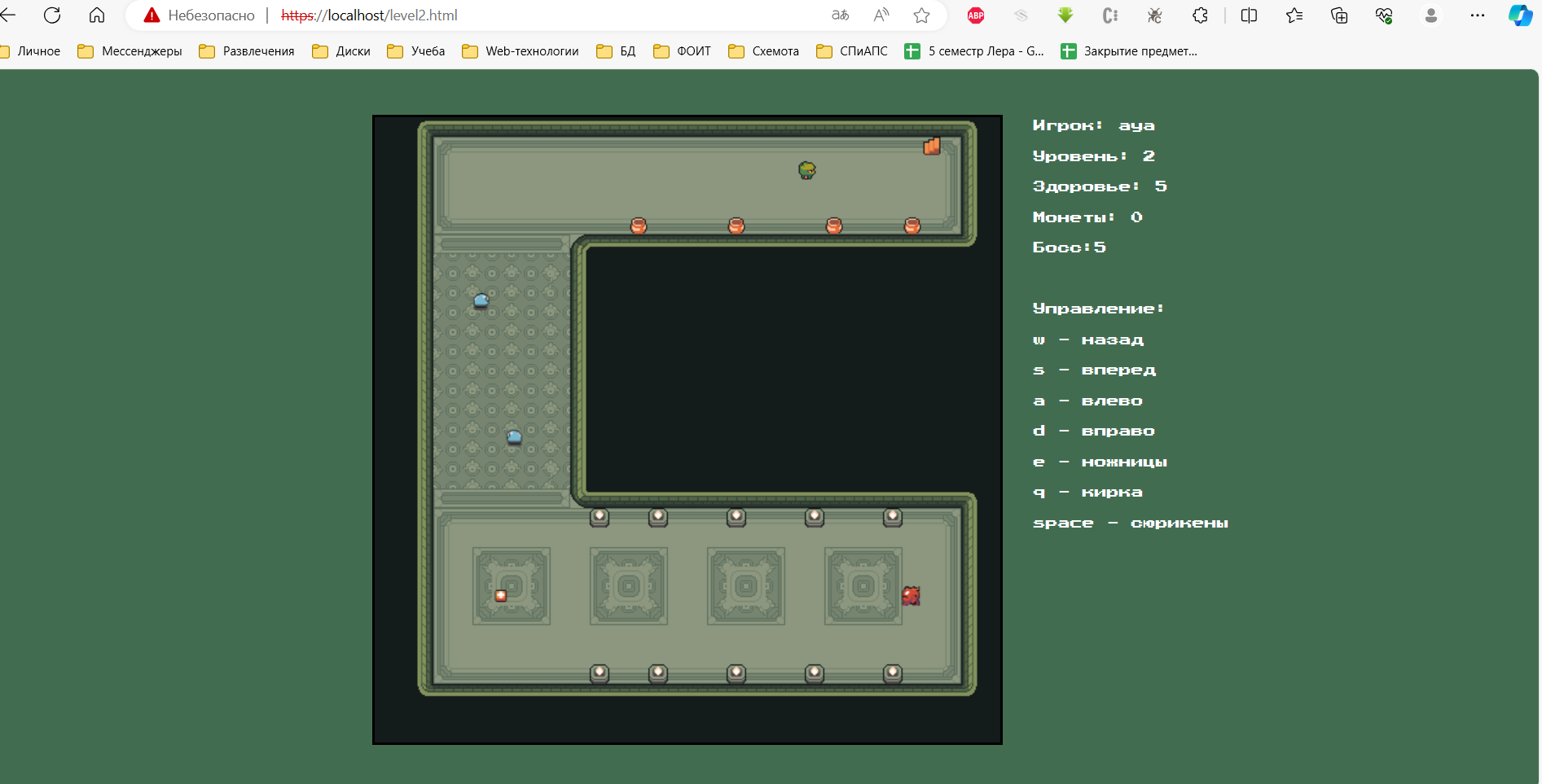


Рисунок 6 – страница второго уровня.

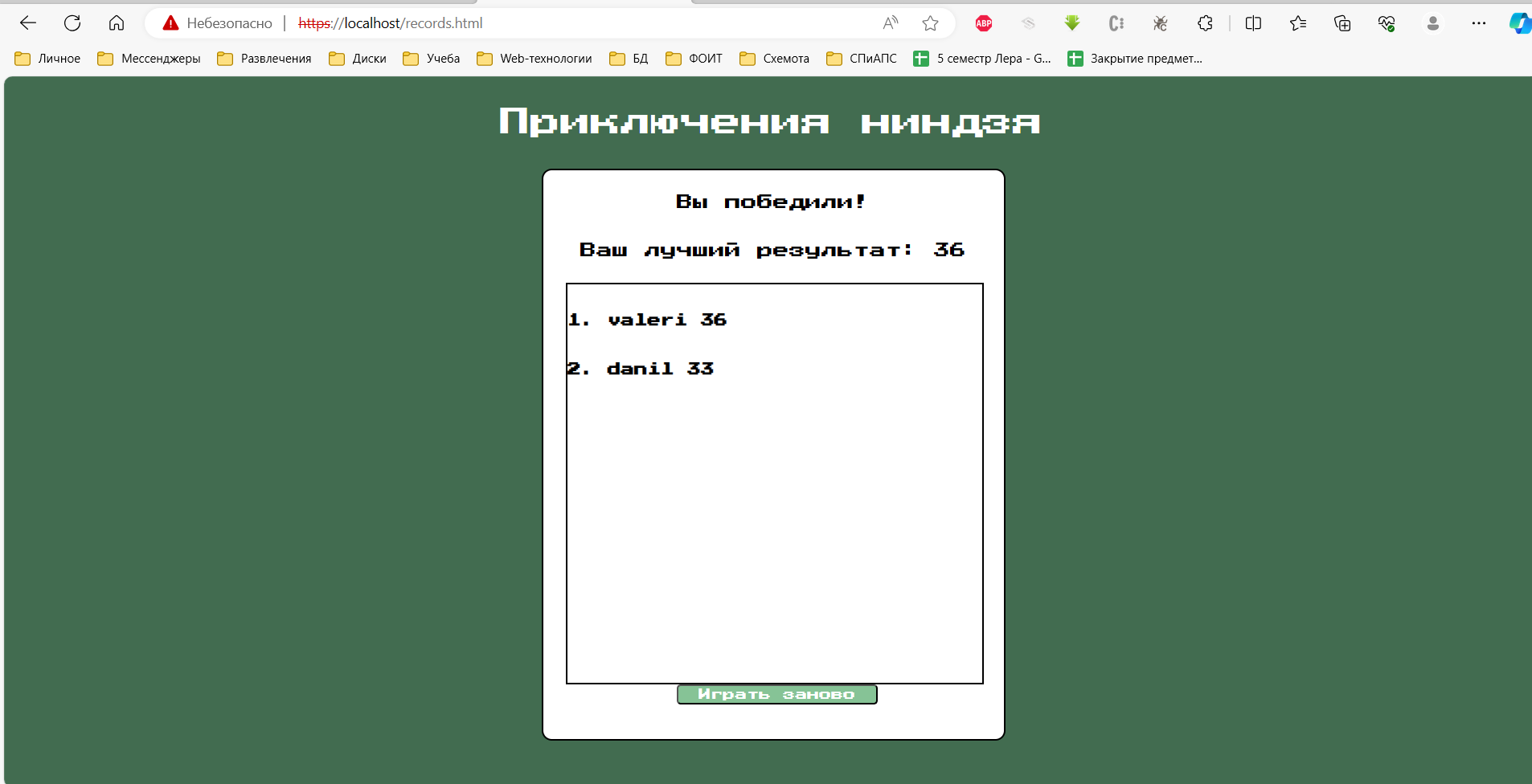


Рисунок 7 – страница с таблицей рекордов.