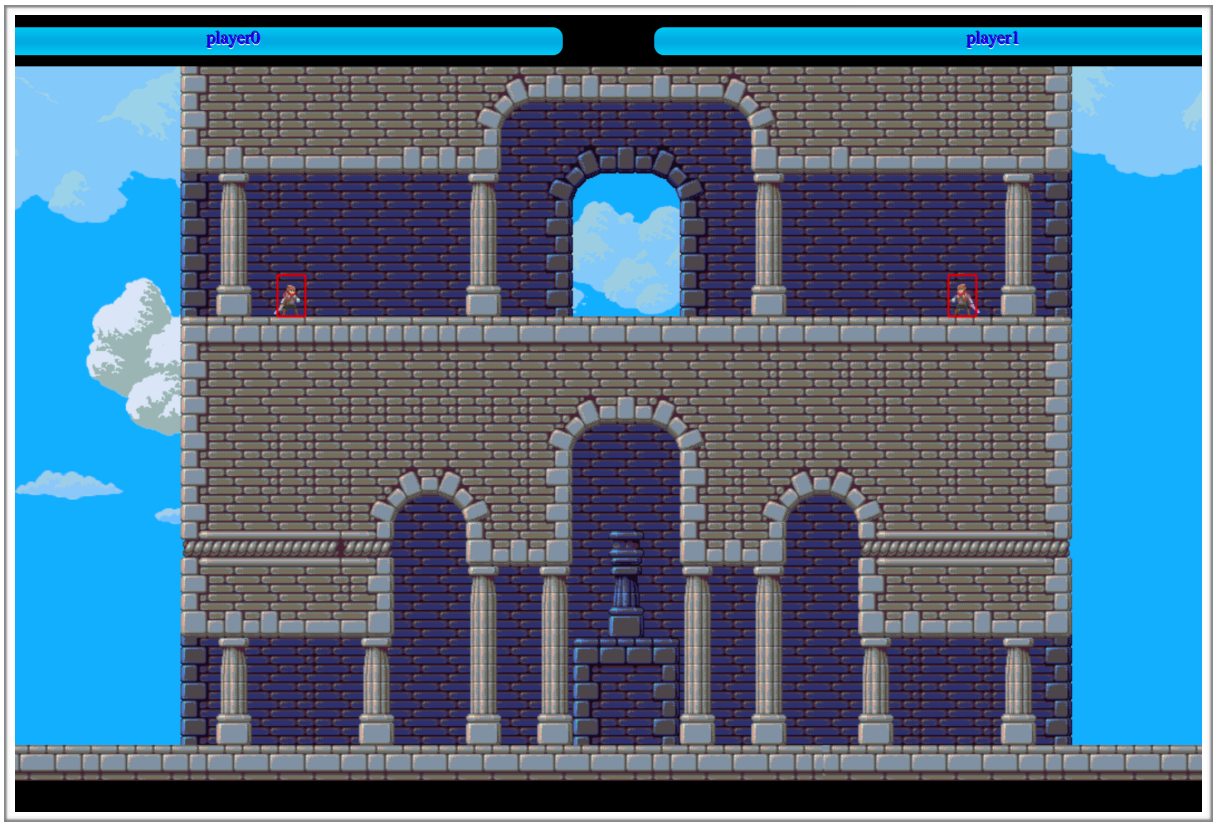
Отчет по УТП



Логачев Кирилл Александрович

ИУ9-21Б

2D-игра

Отчет по УТП

Описание игры

Мы решили создать 2D файтинг в пиксельной стилистике. На небольшой карте (заброшенный храм) расположены два персонажа (рыцари) с холодным оружием. Из боевых механик взяты базовые элементы: простая атака, блок. В шапке сайта находятся два прогресс бара жизней, когда один из прогресс баров опустеет игра закончится проигрышем персонажа с пустым прогресс баром.

«Backend»

Используемые библиотеки :

1) http - библиотека для работы с http протоколом
2) fs - библиотека для работы с файловой системой

Наш backend довольно прост. Web клиент формирует Get запрос, Web сервер получает запрос, анализирует url запроса и ищет подстроку с расширением, а после при помощи switch … case определять MIME-тип и отправлять статический файл на сервер (знаю это не самый правильный метод. Можно было бы при помощи библиотеки проверять содержимое файла и на его основе определить тип. Но в нашем случае наш способ более эффективный). Если произошла ошибка отправляем страницу 404.html.

Создание и обработка карты

Карта преобразуется в формат JSON (формированием JSON занимался Даниил Сазонов). В качестве источника информации для JSON используется рисунки, которые разбиваются на прямоугольники заданного размера, так называемые «Наборы тайлов» (блоков). Именно из этих наборов формируется карта.

Основные свойства **height** и **width** описывают количество блоков по горизонтали и по вертикали, из которых состоит карта, соответственно **tileheight** и **tilewidth** показывают размеры одного блока. Массив **layers** содержит объекты для каждого слоя, массив **tilesets** — объекты, описывающие каждый рисунок, из которого строится кар та игры.

Каждый объект **tilesets** содержит описание для набора блоков, из которых строится карта. Ключевые свойства:

- **tileheight** и **tilewidth** хранят высоту и ширину блока, на которые разбивается изображение;

- **image** — путь до изображения;

- **firstgid** — номер первого блока, используемого из дан
ного изображения;

- **name** — отображаемое в редакторе имя набора блоков.

Следует обратить внимание, что **firstgid** формируется с учетом всех изображений, использованных в карте. Чем больше блоков поместилось на предыдущем рисунке, тем больше номер **firstgid** в следующем наборе блоков. Считается, что нумерация блоков в рамках одного изображения слева направо и сверху вниз, но нигде явно эта информация не сохраняется. Соответственно, номер следующего **firstgid** на единицу больше, чем номер максимального блока в предыдущем рисунке.

Массив **layers** хранит объекты двух типов (поле **type**): **tilelayer** и **objectgroup**. Объекты типа **tilelayer** описывают слой блоков карты, а объекты типа **objectgroup** описыва ют объекты, размещенные на карте.

Поля **height** и **width** объекта типа **tilelayer** содержат количество блоков, помещающихся, соответственно, по высоте и по ширине в данном слое, **name** — его имя, и самое главное поле — массив **data**, который содержит номе ра всех блоков, отображаемых на карте. Несмотря на то что массив одномерный, он хранит информацию обо всей карте: известно количество блоков по горизонтали, соответственно, первые **width** блоков относятся к первой строке блоков на карте, вторые — ко второй и т. д. По номеру блока можно восстановить его координаты и в каком изображении (**tilesets**) он находится.

Объект для управлением картой (или менеджером карты) будет называться **mapManager**.

MapManager:

Для корректной работы с картой необходимо знать ее ширину и высоту в блоках — это поля **xCount** и **yCount**, размер блока — объект **tSize**, содержащий размеры по ширине и высоте **(x, y)**. Полезным будет размер карты **mapSize**, который содержит размеры по ширине и высоте (**x**, **y**) и легко вычисляется по размеру блока и их количеству по ширине и высоте, но позволяет сэкономить время во время вычислений, так как размер карты будет использоваться на каждом цикле ее обновления.

Массив **tilesets** хранит описания для каждого блока карты: их номера, размеры, координаты и т. д.

Загрузка карты в формате JSON из внешнего файла выполняется с использованием так называемой AJAX технологии. Эта технология позволяет отправлять запросы на сервер и получать их асинхронно, т. е. не останавливая процесс выполнения JavaScript в браузере.

Все исходные данные карты представляют собой объект JSON, который полезно хранить внутри менеджера карты, для этого в нем создадим поле **mapData**.

Для загрузки карты (JSON карты) реализован метод функция **loadMap,** принимающая в качестве параметра путь **(path)** к файлу, который необходимо загрузить. Важно обратить внимание, что вызов функции обработки **parseMap** осуществляется с указанием объекта **mapManager**. Для обращения к полям и функциям текущего объекта принято использовать указатель **this**, зарезервированное ключевое слово (пример использования: **this.mapData**). В данном случае функция принадлежит объекту **mapManager**, но использовать указатель **this** нельзя, так как функция **onreadystatechange** будет вызвана не в контексте **mapManager** и **this** будет указывать не на **mapManager**, а на глобальный объект **window.**

Функция **parseMap** разбирает JSON на удобные для обращения и работы поля. Плюс данная функция подгружает все нужные изображения, которые разбиваются потом на тайлы при помощи которых вся карта прорисовывается.

Функция **draw** предназначена для отображения кар ты на холсте, контекст **(ctx)** ей передается в качестве параметра.

Для корректной работы функции **draw** должна быть определена функция **getTile**, обеспечивающая получение блока по ее индексу из **tilesets**.

Для корректной работы функции getTile необходима функция getTileset, которая возвращает найденный tileset.

Создание менеджера событий

Для взаимодействия с пользователем реализован менеджер событий (**eventsManager**). Массив **bind** менеджера событий предназначен для хранения соответствия между кодом действия и клавишей, при нажатии на которую должно выполняться это действие. Использование массива **bind** позволяет при необходимости заменить клавишу действия или использовать несколько клавиш для одного и того же действия.

Массив **action** в качестве ключа использует строковое поле (код действия), а в качестве значения **true** (действие необходимо выполнить) или **false** (действие необходимо прекратить).

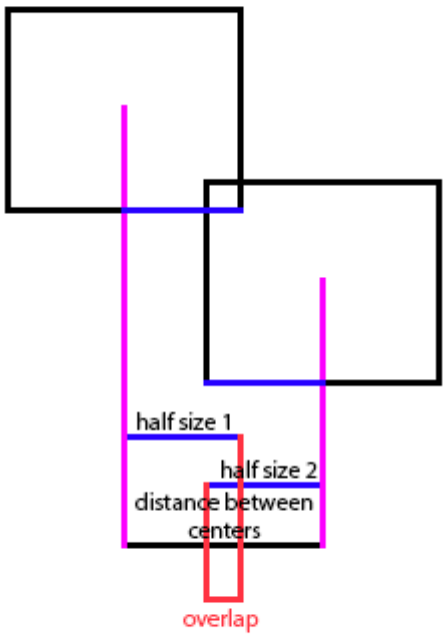
Функция **setup** в качестве параметра принимает **canvas**, чтобы настроить слушатели на действия мыши именно в пределах **canvas**, а не всей страницы.

Создание менеджера физики

Начнем с определения типа форм которые будут использоваться в нашей физике. Мы используем для представления физического объекта в игре Axis Aligned Bounding Box (AABB). По сути AABB это не повернутый прямоугольник.

Все что нам необходимо это данные двух векторов; первый будет центром AABB, а второй половинным размером.

Функцию Overlaps необходима для проверки столкновений. Во-первых, сделана простая проверку столкновений двух AABB друг с другом. Нам просто нужно видеть является ли расстояние между центрами каждой оси меньшим чем сумма половинных размеров.



Менеджер физики объектов содержит две функции: основную (**update**) для обновления состояния объекта и вспомогательную (**entityAtXY**) для определения столкновения с объектом карты (но не другим игроком) по заданным координатам.

При реализации функции **update** используются следующие особенности объекта: уникальное имя объекта (**name**), его координаты (**pos_x**, **pos_y**), размеры (**size_x**, **size_y**), направление движения по координатам (**move_x**, **move_y**), скорость движения (**speed**). На основании этих данных предсказывается следующее положения игрока. При помощи предсказывается следующее положения игрока мы можем определить является ли это объектом карты через который нельзя пройти или это часть игровой зоны.

Создание менеджера игры

Менеджер игры обеспечивает инициализацию, загрузку всех необходимых ресурсов, хранение и управ ление всеми объектами игры, регулярное обновление и отображение пользователю игрового мира.

Поле **factory** представляет собой фабрику объектов, предполагается, что в данной фабрике хранятся эталонные объекты, которые в дальнейшем используются для создания объектов, размещаемых на карте.

Поле **entities** хранит все объекты игры, которые может увидеть пользователь. Предполагается, что в массиве **entities** хранятся все «не убитые» объекты, для которых необходимо регулярно вызывать метод **update**.

Поле player предназначено для хранения ссылок на объекты, управляемые игроками. Это уникальные объекты, которые в том числе хранятся в массиве entities. Отдельные указатели на объекты игроков необходимы для организации управления, изменения параметров объектов в зависимости от команд пользователей, и в дальнейшем корректной обработки расстояния до событий, создающих звуки в игре (звук в игре так и не был реализован, потому что человек отвечающий за разработку звука ушел в академ).

Функция отображения игрового поля пользователю (**draw**) принимает в качестве параметра контекст холста. В данной функции вызывается команда отображения (**draw**) для каждо го объекта (**entities[e]**) карты.

Функция обновления (**update**) вызывается на каждом такте игры и обеспечивает обновление информации об игроке, остальных объектах игры и вызов функции **draw**.

Для корректной работы всех менеджеров игры они должны быть корректно инициализированы в правильной последовательности, для этого должна быть

вызвана функция загрузки (**loadAll**). Использование отдельной функции загрузки позволяет вносить изменения только в одном методе при изменении исходных данных для программы.

Исполнение начинается после вызова функции **play** менеджера игры. С использованием встроенной функции **setInterval** настраивается вызов **updateWorld** каждые 100 мс.

Полезная информация

Кроме выше сказанного мной была сделана html структура, половина css (остальное дорабатывал Сиротин Глеб).

Изначально все был сделано через классы, но в браузере одного из членов команды они не работали, и пришлось все переписывать на объекты и использовать прототипы.

Все js файлы связанны при помощи модулей:

Модуль – это просто файл. Один скрипт – это один модуль.

Модули могут загружать друг друга и использовать директивы `export` и `import`, чтобы обмениваться функциональностью, вызывать функции одного модуля из другого:

- `export` отмечает переменные и функции, которые должны быть доступны вне текущего модуля.
- `import` позволяет импортировать функциональность из других модулей.

Иструкция по запуску:

1)В консоли вводим: node serv

2)Заходим в браузер

3)И вводим: http://localhost:3000/index.html

4)Отключить сервер: ctrl+C

Управление:

Игрок номер один:

W - прыжок

A - влево

D - вправо

LeftShift - атака

Игрок номер два:

↑ - прыжок

← - влево

→ - вправо

RightShift - атака

Команда:

Кирилл Логачев(капитан), Зайцев Арсений(академ), Даниил Сазонов, Сиротин Глеб.