Сделать ветку с готовым вариантом

Импорты в блоки и скрыть

Добавить в начало примерное описание игры, которую мы будем делать

Типы сущностей

Пару слов про марио

Вместо указаний программ рекомендации

Абстрагировать движение, пускай его тоже делают сами

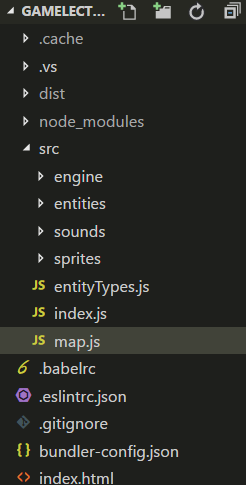
В начале реализовать движение сущности игрока отдельно от всего остального

Чтобы запустить движок:

* откройте Visual Studio Code
* файл -> открыть папку -> выберите marioJS -> выбор папки
* нажмите ctrl + ~, чтобы открыть консоль
* введите команду **npm start** и нажмите enter
* откройте Google Chrome
* введите в адресную строку **localhost:1234** и нажмите enter

Вы должны увидеть пустой уровень с кирпичным полом внизу. Давайте добавим туда персонажа игрока =)

В Visual Studio Code в списке файлов на левой панели найдите файл **src/map.js** и нажмите на него дважды



В этом файле вы увидите текстовое представление нашего уровня:

*const* asciiMapRows = [

' ',

' ',

' ',

' ',

' ',

' ',

' ',

' ',

' ',

' ',

' ',

'##########################################################'

];

Чтобы добавить в уровень персонажа игрока, надо заменить один из символов пробелов на символ **@**.

После каждого изменения в коде игры надо сохранить измененный файл (**ctrl+s)** и перезагрузить страницу с игрой в браузере **(F5)**

Теперь мы видим, что в игре появился Марио. Мы можем использовать <- и ->, чтобы двигать его, и пробел, чтобы прыгать. Но, как мы могли заметить, при движении его анимация по-прежнему остается статичной. Давайте исправим это:

* Откройте файл **src/entities/mario.js**
* Найдите внутри функции **function updateMario(mario)** следующие строчки:

// Рисуем спрайт марио на экране

drawSprite(sprMarioIdle, mario, 0, dir);

Функция **drawSprite(sprite, entity, speed, direction)** рисует на экране спрайт

- **sprite –** спрайт, который будет нарисован (в данном случае sprMarioIdle)

- **entity –** сущность (игровой объект), координаты которого будут использоваться при рисовании спрайта

- **speed –** скорость анимации, частота смены кадров спрайта (в данном случае 0)

- **direction –** направление (1 = слева направо, -1 = справа налево)

* Внесите следующие изменения:

// Рисуем спрайт марио на экране

if (absSpeedX > 1) {

drawSprite(sprMarioRunning, mario, 0.2, dir);

} else {

drawSprite(sprMarioIdle, mario, 0, dir);

}

Теперь если скорость движения Марио будет больше небольшого значения (1), то мы будем рисовать на экране анимацию бега, которая будет анимироваться со скоростью 0.2. Сохраните изменения и проверьте результат.

Теперь осталось только добавить анимацию прыжка:

// Рисуем спрайт марио на экране

if (mario.isOnGround === false) {

drawSprite(sprMarioJumping, mario, 0, dir);

} else if (absSpeedX > 1) {

drawSprite(sprMarioRunning, mario, 0.2, dir);

} else {

drawSprite(sprMarioIdle, mario, 0, dir);

}

mario.isOnGround равно true, если марио в данный момент стоит на твердом блоке.

Откройте файл **map.js**  и замените пустые символы в карте на **#,** чтобы добавить больше блоков. Попробуйте попрыгать по ним =)

Также добавьте на карту несколько врагов (символ **G**). Как мы видим, они просто стоят на месте и ничего не делают. Давайте добавим им поведение:

Откройте файл **src/entities/goomba.js**  и найдите в нем функцию **updateGoomba(goomba)**

*function* updateGoomba(*goomba*) {

// весь код ниже выполняется каждый кадр

drawSprite(sprGoomba, goomba, 3 \* time.deltaTime);

}

Как мы видим, весь код гумбы только рисует его спрайт, и больше ничего не делает. Добавьте следующие строчки:

goomba.speedY += settings.gravity \* time.deltaTime;

*const* { horizWall } = moveAndCheckForObstacles(goomba, [ENTITY\_TYPE\_WALL]);

Теперь гумбы, появившиеся в воздухе, должны почувствовать силу гравитации.

Функция **moveAndCheckForObstacles(entity, obstacleEntityTypes)** передвигает сущности в зависимости от их свойств speedX и speedY (от которых зависит скорость их передвижения по оси x и y соответственно)

- **entity –** сущность, которая будет передвинута (в данном случае goomba)

- **obstacleEntityTypes** – набор типов сущностей, которые будут восприниматься как препятствия при движении, заключенный в **[ ].** ENTITY\_TYPE\_WALL – означает, что наш враг не будет проходить сквозь стены. Можете убрать его из квадратных скобок и посмотреть, что будет)

В оригинальной игре гумбы идут направо, пока не столкнутся со стеной, после чего меняют направление, и так до бесконечности. К счастью, у нас уже есть все что нужно, чтобы воплотить такое поведение: функция **moveAndCheckForObstacles** возвращает нам объекты, с которыми наша сущность столкнулась:

*const* { horizWall } = moveAndCheckForObstacles(goomba, [ENTITY\_TYPE\_WALL]);

В данном случае **horizWall** будет равна **null (**пустое значение**),** если наша сущность не столкнулась ни с одной горизонтальной стеной в этот кадр, или сущность стены в обратном случае. Теперь нам осталось только проверить, находимся ли мы слева или справа от стены, если мы столкнулись с ней, и изменить скорость соответственно:

if (horizWall !== null) {

if (goomba.x < horizWall.x) {

goomba.speedX = -2;

} else {

goomba.speedX = 2;

}

}

Поставьте на карте несколько стен и проверьте, как гумбы будут отталкиваться от них.

Дальше давайте добавим в игру монетки. Для этого надо вставить в карту несколько символов **0**.

Как мы видим, монетки появились, но было бы неплохо, чтобы Марио умел их собирать. Откройте файл **src/entities/coin.js** и найдите функцию updateCoin

// Код ниже выполняется каждый кадр

drawSprite(sprCoin, coin, 2 \* time.deltaTime);

Нам нужно проверить, сталкивается ли данная монетка с Марио в данном кадре, и если да, она будет уничтожена (собрана). Для этого нам пригодится функция

**checkCollision(entity, collidingEntityTypes)**

- **entity –** сущность, для которой будет проверяться столкновение

- **collidingEntityType -** набор типов сущностей, столкновение с которыми будет проверяться, заключенный в **[ ].**

Она очень похоже на функцию **moveAndCheckForObstacles**, которую мы использовали ранее. Ее отличие в том, что она не передвигает сущность, а лишь проверяет, не сталкивается ли она с другими сущностями, и возвращает **null,** если пересечения не было, или сущность, с которой произошло столкновение.

Добавим следующий код в функцию **updateCoin:**

*const* mario = checkCollision(coin, [ENTITY\_TYPE\_MARIO]);

if (mario !== null) {

removeEntity(coin);

}

Функция **removeEntity(entity)** убирает сущность **entity** из игры.

Теперь мы можем видеть, что монетки собираются, когда к ним подходит Марио, но было бы неплохо добавить еще пару эффектов, например, звук.

Функция **playSound(sound, loop?, volume?)** проигрывает звук **sound.** Изменим код, добавив ее:

*const* mario = checkCollision(coin, [ENTITY\_TYPE\_MARIO]);

if (mario !== null) {

removeEntity(coin);

playSound(sndCoin);

}

А еще было бы неплохо, если бы сбор монеток увеличивал счетчик в углу экрана. Добавим в конец еще одну строчку:

*const* mario = checkCollision(coin, [ENTITY\_TYPE\_MARIO]);

if (mario !== null) {

removeEntity(coin);

playSound(sndCoin);

scoreEntity.count++;

}

И раз уж мы теперь умеем воспроизводить звуки, давайте запустим фоновую музыку. Для этого в файле **src/map.js** добавим

playSound(sndMainTheme, true);

Переданный вторым аргументом **true** указывает функции, что звук должен быть зациклен.