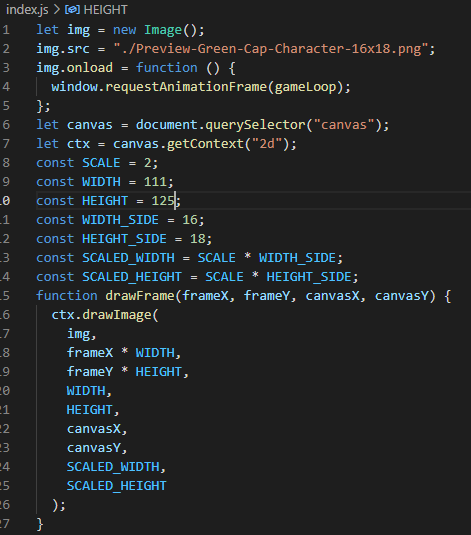
[В части 1](https://webdevblog.ru/peremeshhenie-personazha-na-sprajtah-s-pomoshhju-javascript-chast-1/" \t "_blank) мы рассмотрели, как анимировать персонажа на основе спрайтов по таймеру с помощью **requestAnimationFrame**. Теперь, вместо того, чтобы проходить через временной цикл анимации, мы изменим его на анимацию и перемещение в зависимости от ввода пользователя.

Начало

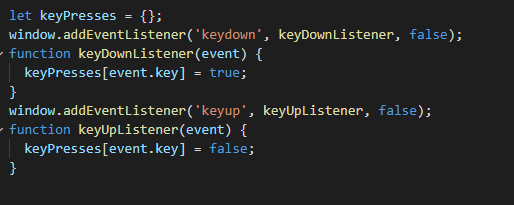
Мы будем использовать код из части 1 в качестве основы. Сделаем его немного проще, для этого изменим часть старого кода, чтобы получить более лучшую отправную точку.



1. Функция init была переименована в **gameLoop**.
2. Функция **step** была удалена.
3. Чтобы цикл продолжался, **window.requestAnimationFrame (gameLoop)**; вызывается в конце **gameLoop**.
4. В соответствии с соглашениями о константах, все константы полностью написаны в верхнем регистре.

### Получение пользовательского ввода

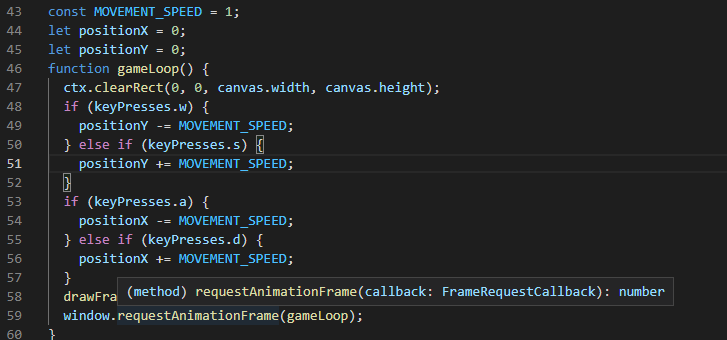
Настроим обработку пользовательского ввода. Нам понадобится пара слушателей событий, чтобы отслеживать, когда клавиши нажимаются и отпускаются. Нам также понадобится что-то для отслеживания этих состояний. Мы можем отслеживать определенные кнопки и реагировать только на них, или мы можем сохранить все нажатия клавиш в объекте и позже проверять, что нам нужно. Лично я предпочитаю использовать последнее.



### Перемещение персонажа

Теперь, когда мы фиксируем ввод пользователя, давайте снова добавим персонажа и обработаем его движение.

Для начала мы будем использовать только первый кадр персонажа, обращенного вниз. Нам также нужно отслеживать позиции x и y персонажа. Мы также должны добавить константу **MOVEMENT\_SPEED**, чтобы мы могли легко изменить ее позже. Она будет означать количество пикселей, перемещаемых за кадр анимации.

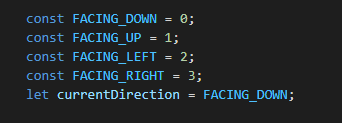


Теперь у нас есть подвижный персонаж!

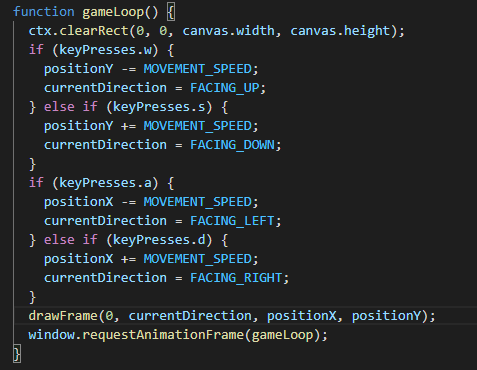
***Примечание.*** Изначально использовались клавиши со стрелками, но из-за прокрутки страницы при нажатии вверх и вниз вместо них использовались клавиши ***WASD***. Однако вы можете использовать любую комбинацию клавиш.

### Изменение направления

В настоящее время персонаж всегда смотрит вниз. Давайте справимся с обращением в разные стороны. Как и в части 1, мы будем использовать переменную **currentDirection** для хранения того, в каком направлении смотрит персонаж. Чтобы сделать это немного более интуитивным, давайте добавим константу для каждого направления.



Теперь, когда это настроено, давайте обновим условия обработки движения и вызов **drawFrame** для обработки заданного направления.



Теперь у нас персонаж меняет направления. Давайте добавим разные кадры для этих направлений. Мы по-прежнему будем придерживаться шаблона 0, 1, 0, 2 кадра для нашей анимации ходьбы. Для этого мы можем вернуть ссылку на **CYCLE\_LOOP [currentLoopIndex]** в нашем вызове drawFrame.

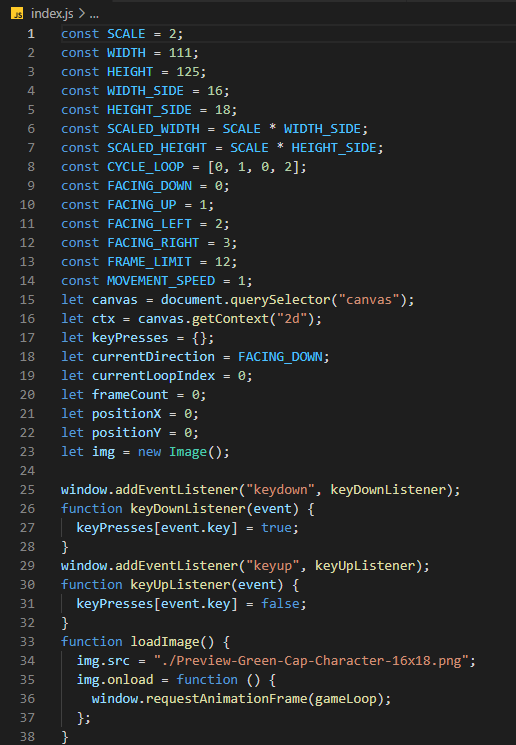


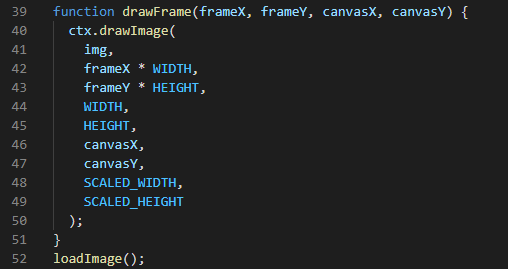
Затем нужно вернуть инкрементор кадра и ограничение. Это немного отличается от части 1. Нам все еще нужно обрабатывать движение, поэтому вместо раннего возврата мы увеличим счетчик кадров, а затем каждые несколько кадров сбрасываем счетчик и обновляем индекс. Так же нам нужно, чтобы кадр увеличивался при любом движении.

Вот оно! Персонаж перемещается по canvas, меняет направление и циклически проходит все кадры анимации.

### Небольшая уборка

Прежде чем мы продолжим, давайте сделаем небольшой рефакторинг:







Это выглядит намного чище. Константы и переменные находятся в одном месте в верхней части (мы могли бы даже переместить их в набор объектов, а не в глобальную область видимости, но в рамках этого руководства мы сделаем проще). Слушатели событий нажатия клавиш являются первыми в наборе функций. Загрузчик изображений, запускающий весь игровой цикл, выполняет свою функцию. А управление движением перенесено в отдельную функцию.

Держаться в границах

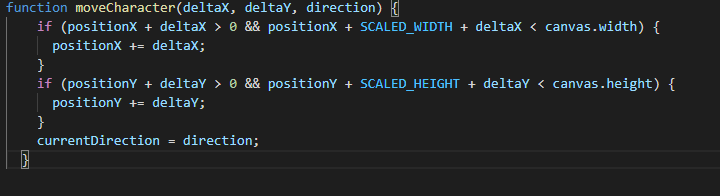
Перенос функционала обработки движения в его собственную функцию на самом деле имеет дополнительную цель. Прямо сейчас персонаж может покинуть границу холста. Но с помощью функции **moveCharacter** мы можем проверить столкновение границ в одном месте вместо четырех.

Наше обнаружение столкновений выглядит примерно так:

1. Левый край персонажа касается или проходит за левый край canvas?
2. Правый край персонажа касается или проходит за правый край canvas?
3. Верхний край персонажа касается или проходит за верхний край canvas?
4. Нижний край персонажа касается или проходит за нижний край canvas?

Если что-то из этого верно, нам нужно остановить движение персонажа в заданном направлении. Поскольку мы обрабатываем два направления одновременно, мы можем разделить проверки и ограничения горизонтального и вертикального движения. Таким образом, если персонаж находится в середине одного края, он может скользить по этому краю, пока не коснется угла.

Давайте обновим нашу функцию движения, чтобы справиться с этими условиями.



Важно помнить, что **positionX** и **positionY** относятся к левому верхнему углу персонажа. Из-за этого **positionX + SCALED\_WIDTH** дает нам правый край персонажа, а **positionX + SCALED\_HEIGHT** дает нам нижний край персонажа.

Имея это в виду, вот как проверки переводятся в соответствие с вопросами выше:

1. positionX + deltaX > 0 проверяет столкновение с левым краем.
2. positionX + SCALED\_WIDTH + deltaX < canvas.width проверяет столкновение с правым краем.
3. positionY + deltaY > 0 проверяет столкновение с верхним краем.
4. positionY + SCALED\_HEIGHT + deltaY < canvas.height проверяет столкновение нижнего края.

### Последние дополнение

Теперь, когда наш персонаж остается в определенных рамках, есть еще один небольшой момент. Если пользователь перестает нажимать клавишу, когда персонаж находится на втором или четвертом кадре цикла анимации, это выглядит немного странно. Персонаж стоит на полпути. Как насчет сброса кадра, когда персонаж не двигается?

В функции **gameLoop** прямо перед вызовом **drawFrame** добавим проверку:



Теперь персонаж всегда будет в естественном положении стоя, когда не движется.