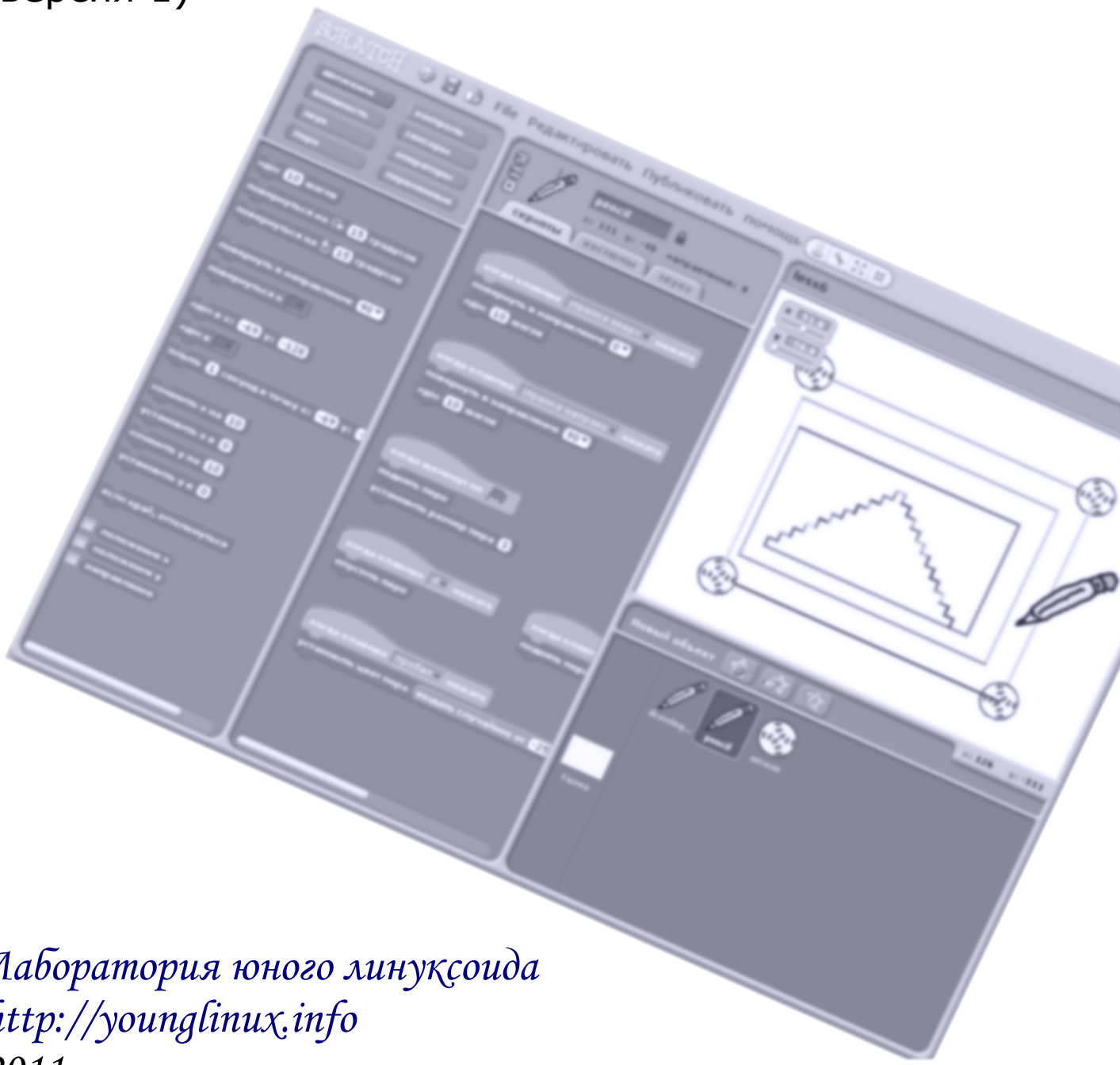
С. Шапошникова (plustilino)

Введение в Scratch

Цикл уроков по программированию для детей
(версия 1)



Лаборатория юного линуксоида
http://younglinux.info
2011

Пояснительная записка

Курс “Введение в Scratch” представляет собой цикл из десяти уроков по основам работы в среде программирования Scratch. Также данный цикл подспутно знакомит учащихся с некоторыми принципами парадигм программирования (структурного, объектно-ориентированного, событийного).

Курс содержит методические разработки (конспекты) уроков, адаптирован для учащихся ориентировочно 5-7 классов и предназначен для самостоятельного изучения или освоения тем под руководством педагога.

В цикле уроков “Введение в Scratch” рассматриваются: организация интерфейса среды программирования Scratch; понятие о программе (сценарии, скрипте) объекта (спрайта); система координат и направление движения; циклы и условные операторы; последовательное и параллельное выполнение команд; изменение свойств объекта; события, интерактивность и диалоговый режим выполнения программы; использование переменных и генератора случайных чисел; составление программ, рисующих на холсте; создание и изменение объектов и библиотеки объектов; создание эффекта смены сцены.

Поскольку данный курс предназначен для первого знакомства с особенностями работы в среде Scratch, ни один его урок не включает разработку законченной программы или анимации.

Язык команд и интерфейса — русский.

Версия приложения Scratch — 1.4.

Цикл уроков “Введение в Scratch” первоначально публиковался на сайте http://younglinux.info в период с января по март 2011 года.

Содержание

Урок 1. Знакомство со Scratch 4

Урок 2. Управление несколькими объектами 10

Урок 3. Последовательное и одновременное выполнение 14

Урок 4. Интерактивность, условия и переменные 18

Урок 5. Случайные числа 21

Урок 6. Рисование в Scratch 24

Урок 7. Диалог с программой 28

Урок 8. Создание объектов и костюмов 30

Урок 9. Использование библиотеки объектов 34

Урок 10. Смена фона 38

Другие источники информации 41

Урок 1. Знакомство со Scratch

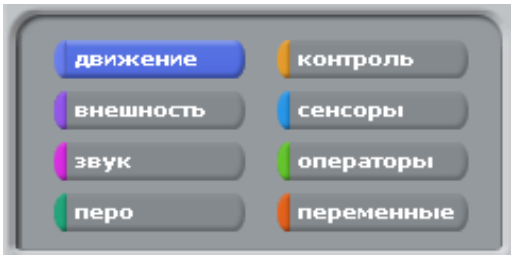
Приложение Scratch представляет собой среду для создания программ и анимации. Среда Scratch позволяет делать очень многое. Поэтому ее интерфейс (внешний вид после запуска) достаточно сложный. Однако когда мы разберемся с принципами работы в Scratch, поймем как составлять программы и создавать несложную анимацию, то нам откроются большие просторы для самостоятельного творчества и исследования.

Интерфейс Scratch и основы работы в нем

Будем знакомиться с интерфейсом Scratch постепенно, разбирать работу лишь того, что нам потребуется на текущий момент.

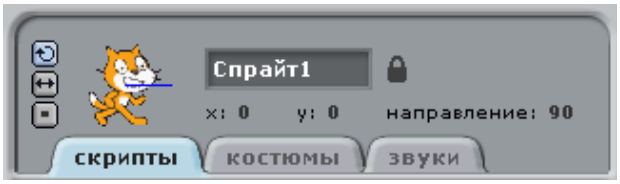
После того, как программа запустилась, перед нами появляется окно, в котором можно выделить три части (три столбца). В свою очередь каждый столбец состоит как бы из двух ячеек: верхней и нижней. Все ячейки разные и каждая из них предназначена для своей конкретной цели.

Верхняя ячейка в первом столбце содержит восемь кнопок, которые называются **движение, контроль, внешность, сенсоры, звук, операторы, перо, переменные**. При включении одной кнопки все остальные выключаются. Включенная кнопка вся окрашивается в соответствующий ей цвет. При этом в нижней ячейке первого столбца появляются команды, связанные с включенной кнопкой. У всех кнопок разные привязанные к ним команды. Попробуйте нажимать кнопки и посмотрите, как меняются команды в нижней ячейке.



На этом уроке нас будут интересовать только команды, связанные с кнопками **движение** и **контроль**.

Рассмотрим верхнюю ячейку второго столбца.



Здесь показаны свойства объекта, которым мы будем управлять (писать программы для него). Сейчас - это кот. Его имя написано в поле - *Спрайт1*.

Странное имя для кота, не так ли? Спрайтами в Scratch называются все объекты по умолчанию, отличаются лишь их номера. Обычно предполагается, что мы сами их будем переименовывать и называть более адекватными именами. Например, для кота лучше в поле имени вписать *кот* или *Вася* или что-нибудь еще. Сделайте это.

Ниже поля имени отображены три свойства объекта (в данном случае, кота) — это его положение (координаты *x* и *y*) и направление. Если вы уже изучили работу в среде KTurtle, то вспомните, что направление, когда черепашка «смотрит» вверх равно 0 градусам, когда точно направо — равно 90 градусам, а полная окружность равна 360-ти градусам. Здесь то же самое. Кот смотрит направо, поэтому его направление равно 90.

Внизу мы видим три кнопки-вкладки — **скрипты**, **костюмы** и **звуки**. От того какая из них нажата, зависит ячейка внизу. Если нажата кнопка **скрипты**, то нижняя ячейка второго столбца покажет программы (скрипты) для объекта, который отображен в верхней ячейки. Сейчас нажата кнопка **скрипты** и мы видим пустую ячейку внизу. Это значит, что для кота пока нет никакой программы. Мы составим ее чуть позже. Кнопки **костюмы** и **звуки** позволяют настраивать и менять соответственно внешний вид объекта и издаваемые им звуки. Оставьте включенной кнопку **скрипты**.

Перейдем к третьему столбцу и опишем в нем только верхнюю ячейку, где видим белое поле и кота на нем. Белое поле — это холст. Многие действия, которые задает программист на вкладке **скрипты**, объект выполняет именно на холсте. Так, если мы запрограммируем ходьбу кота, то он будет перемещаться по холсту.

Если зажать левую кнопку мыши над котом, а затем не отпуская ее перемещать мышь, то кот будет перемещаться. Таким образом, мы можем менять его положение на холсте. Переставьте кота в другое место и посмотрите в ячейку свойств объекта (второй столбец). Заметьте, что координаты *x* и *y* изменили свои значения.

Над холстом находятся две вот такие кнопки:

Когда для объектов (кота и других) будут составлены какие-нибудь скрипты (программы) на вкладке **скрипты**, то чтобы объекты начали их выполнять, надо нажать зеленый флажок. Чтобы остановить выполнение, надо нажать красный круг. Не забывайте останавливать свои скрипты!

Над флагом и кругом мы видим еще три кнопки.

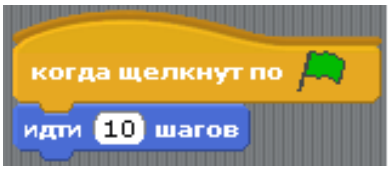
Они предназначены для изменения всего окна приложения Scratch. Сейчас у нас включена вторая кнопка, поэтому окно Scratch примерно поровну разделено на три столбца. Однако если мы составляем какую-нибудь очень сложную программу и у нас получается очень большой скрипт, то лучше нажать первую кнопку. При этом второй столбец займет существенную часть окна. Третья кнопка предназначена исключительно для просмотра созданной программы-анимации. Если ее нажать, то холст занимает центр экрана, и доступными остаются только три кнопки: запуск программы (зеленый флажок), остановка программы (красный круг) и выход из режима презентации (стрелка), осуществляющий возврат к предыдущему окну. Попробуйте переключаться между режимами работы.

Первая программа

Мы уже рассмотрели интерфейс достаточно, чтобы написать первую программу для кота.

Сейчас включена кнопка **движение** и видны команды, отвечающие за перемещение объекта. Если перетащить мышью любую их этих команд в ячейку **скрипты**, то она станет командой для кота. Например, *идти 10 шагов* заставит животное переместиться на 10 точек экрана. После того как команда перемещена в скрипты, можно посмотреть, как она работает, совершив двойной клик по ней. Кот шагнет на холсте. Задайте команду *идти 10 шагов* для объекта и проверьте, как она работает.

У нас есть программа для кота, состоящая из одной команды. По идее при нажатии на кнопку запуска (зеленый флаг) она должна срабатывать. Но на самом деле этого не произойдет. Чтобы программа запускалась при нажатии флажка, надо в ее начало поместить специальную команду, которая связана с кнопкой **контроль**. На ней написано *когда щелкнут по* и изображен зеленый флажок. Если взять эту команду, перетащить в поле скриптов и соединить с командой *идти 10 шагов*, то программа станет запускаться при нажатии на кнопку запуска. Сделайте это.



Обратите внимание на то, как соединяются между собой команды: как блоки в конструкторе Лего. В каких-то случаях это может служить своего рода подсказкой. Какие-то команды можно соединять между собой, а какие-то не соединяются, и, значит, мы составляем не очень правильную программу.

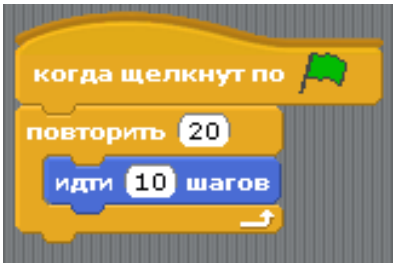
Чтобы разорвать блоки, надо потянуть за нижнюю команду, чтобы переместить весь блок - за самую верхнюю. Попробуйте и затем верните все на место.

Сделаем некоторые выводы. Наш код даже не шагает, а просто прыгает на 10

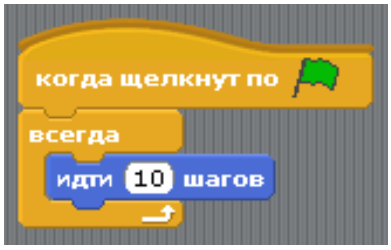
точек вперед и останавливается. Конец программы. Хорошо бы было, чтобы он все-таки шагал и проделывал путь несколько больше. Самое первое, что может прийти в голову, — это увеличить количество шагов, например, до 100. Для этого надо в команде *иди 10 шагов* щелкнуть по числу 10 и вписать в поле число 100. Переместите кота ближе к левой границе холста, поменяйте число шагов и запустите программу.

Теперь кот шагает больше. Но разве это похоже на шаги? Он просто скачет на большее расстояние. Поэтому вернем число 10 и подумаем над другим решением.

В командах контроля есть такая команда как *повторить 10*. Это цикл, который прокручивает то, что в него вложено такое количество раз, какое число написано в его поле. По умолчанию — это 10. Если команду *иди 10 шагов* поместить внутрь этого цикла, то он будет ее «крутить». Поместите кота ближе к левой границе холста и составьте вот такую программу для него. Посмотрите, как теперь наш объект будет двигаться. Согласитесь, что у нас получилась настоящая анимация объекта.



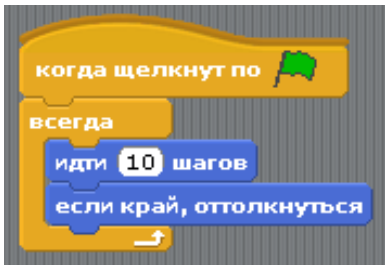
Продолжим улучшать и видоизменять нашу первую программу. Допустим, мы не знаем, сколько раз надо повторять шаги. Мы хотим, чтобы объект двигался всегда, пока мы не нажмем кнопку остановки (красный круг). Для этого надо поступить так: вытащить команду *иди 10 шагов* из цикла *повтори ...*, убрать цикл *повтори ...* (перетащить его в левый столбец), взять команду *всегда* и собрать такую программу:



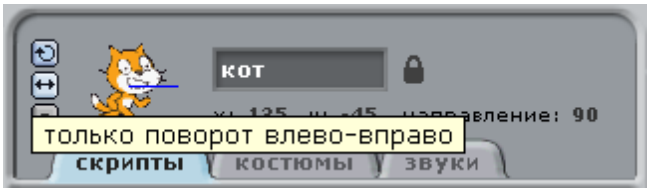
Теперь программа должна работать вечно, пока не будет остановлена. Так и происходит. Если нажать кнопку запуска, то зеленый флажок будет гореть до тех пор, пока не будет нажат красный. Это значит, что программа не может сама остановиться. Но посмотрите, что происходит с котом. Он останавливается, достигнув границы холста. Программа работает, а кот стоит. Тут что-то не так. Останавливаем программу! Возвращаем кота на место.

С кнопкой **движение** связана такая команда как *если край, оттолкнуться*.

Если ее поместить в цикл *всегда*, то объект уже не остановится достигнув края, а оттолкнется от него и пойдет в другую сторону. Составьте вот такой скрипт и посмотрите, как он работает.



Скрипт работает замечательно, но кот не совсем нормален при движении справа налево. Он идет вниз головой. Для исправления этого недочета остановим программу и снова обратим свой взгляд на ячейку свойств объекта.

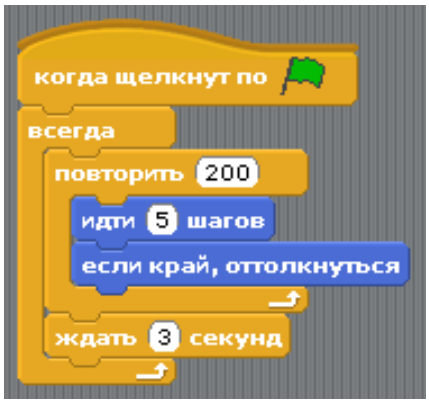


В левой части этой ячейки есть три маленькие кнопки: со скругленной стрелкой, двуголовой стрелкой и квадратной точкой. По умолчанию нажата первая кнопка и поэтому объект при столкновении поворачивается так, как мы наблюдали. Если нажать вторую кнопку, то он будет поворачиваться так, как нам надо в данный момент (слева направо). Третья кнопка вообще запрещает какие-либо повороты. Нажмите вторую кнопку и посмотрите, как ходит кот. Не забудьте после этого остановить программу.

Посмотрите на изображение кота в ячейке свойств. У него есть какой-то синий отрезок. Зажав на нем мышь, его можно поворачивать. При этом меняется значение направления. Это означает, что объект при запуске программы будет двигаться в установленном таким образом направлении. Поэкспериментируйте с выбором направления движения кота.

Самостоятельная работа

Составьте вот такую программу, испытайте (протестируйте) ее и попытайтесь объяснить, как она работает.



Вопросы к самостоятельной работе:

1. Сколько циклов в программе? Назовите их.
2. Какой цикл является вложенным, а какой внешним?
3. Какие команды содержит цикл *повтори …*?
4. Какие команды содержит цикл *всегда*?
5. Сколько всего шагов делает кот, прежде чем останавливается на 3 секунды?

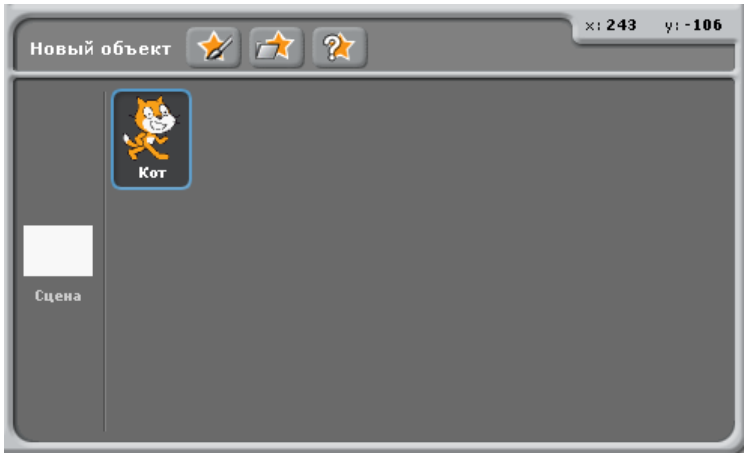
Урок 2. Управление несколькими объектами

Координаты

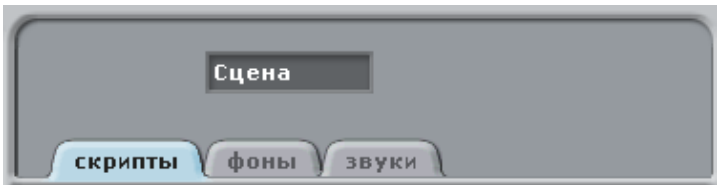
Когда окно Scratch только открылось, мы видим, что кот стоит в центре холста. Посмотрим теперь на его координаты *x* и *y* в ячейке свойств объекта. Там написано, что **x: 0** и **y: 0**. Почему так? Если вспомнить приложение KTurtle, с помощью которого мы изучали язык Logo, то точка с координатами (0,0) должна быть в верхнем левом углу холста. Теперь придется забыть об этом. В Scratch начало системы координат - это центр холста. От центра вправо значение *x* увеличивается, влево — уменьшается (становится минусовым, отрицательным). Аналогично с осью *y*. По направлению вверх идут положительные значения, вниз — отрицательные. Чем дальше от центра, тем больше абсолютное значение числа. Переместите несколько раз кота по холсту и отметьте, как меняются значения *x* и *y* в ячейке свойств объекта.

Сцена

Чтобы было легче ориентироваться в координатах, проделаем кое-какие манипуляции с холстом - наложим на него картинку с системой координат. Но сначала рассмотрим ячейку, которую мы пропустили на прошлом занятии.



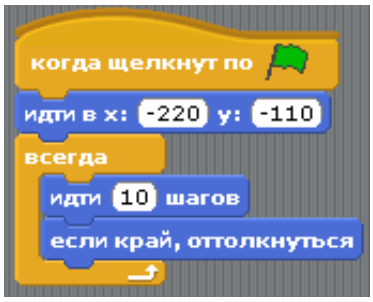
Эта нижняя правая ячейка окна Scratch предназначена для работы с объектами текущего проекта. Здесь можно добавлять новые объекты и переключаться между теми, которые у нас уже имеются. У нас пока существуют только кот и сцена. Переключаться между ними можно, щелкая по ним левой кнопкой мыши. При этом меняется ячейка свойств объекта. Например, если щелкнуть по картинке *Сцена*, то ячейка свойств будет выглядеть так:



Теперь самое интересное. Мы хотим поменять фон холста. На вкладке **фоны** есть кнопка **Импорт**. Если ее нажать, появляется возможность добавить картинку, которая будет служить фоном холста. Нам нужна картинка *xy-grid*, на которой изображены оси координат. После этого следует снова переключиться на кота, щелкнув по его изображению в нижней ячейке третьего столбца окна Scratch и переключиться на вкладку **скрипты**. Добавьте фон для сцены, затем снова вернитесь на вкладку **скрипты** кота.

Повторение программы

Вспомним программу, которую мы составили на прошлом занятии. Снова составим ее, добавив лишь точные координаты начального положения кота с помощью команды *идти в x: ... y: ...*. Эта команда установит объект в указанную точку. Пусть это будет точка с координатами (-220, -110), т.е. кот окажется внизу. Соберите такой скрипт:



Не забудьте указать правильное направление поворота для кота с помощью соответствующей кнопки в ячейке свойств объекта.

Новые объекты

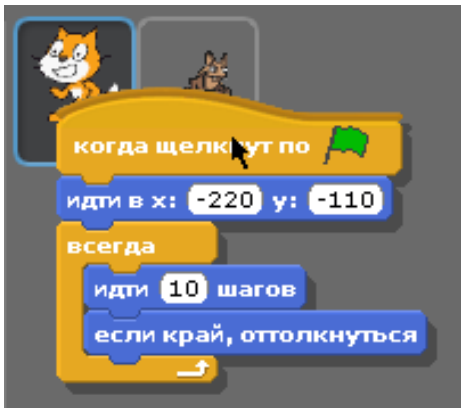
Теперь предположим, что на сцене у нас будет бегать помимо кота еще какая-нибудь живность. Как в Scratch добавить новые объекты? Следует нажать на вторую кнопку в ряде кнопок под холстом:

После чего перед нами открывается диалоговое окно, где из папок *Animals* (животные), *Fantasy* (фантазия), *Letters* (буквы), *People* (люди), *Things* (вещи) и *Transportation* (транспорт) можно выбрать любой объект. Давайте пока ограничимся только папкой *Animals* и добавим на холст какое-нибудь животное, птицу или насекомое. Сделайте это.

Объект добавляется в центре холста. Теперь дайте ему соответствующее имя в ячейке свойств объекта.

Хорошо бы, чтобы второй объект тоже как-нибудь двигался. Поскольку мы уже запрограммировали кота, то можно не составлять новый скрипт, а скопировать

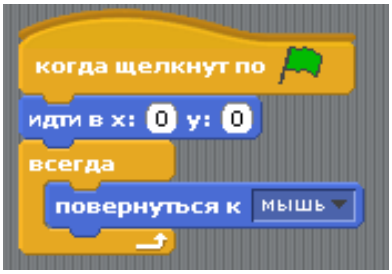
программу кота и перенести копию на новый объект. Делается это так. Переключаемся на кота → щелкаем правой кнопкой по скрипту и в контекстном меню выбираем команду **дублировать** → перемещаем курсор мыши с прилипшей к нему копией на иконку второго объекта в нижней правой ячейке (будьте внимательны: при этом вокруг иконки должна появиться серая рамка!) и щелкаем мышью.



Теперь если переключиться на второй объект, то мы здесь увидим новый скрипт. Его следует немного подправить, чтобы кот и мышь (например) двигались по-разному:

Сделайте это.

Добавим третий объект, но программу для него не будем копировать. Пусть третий объект стоит на месте в центре холста и всегда поворачивается ко второму объекту. Поскольку второй объект постоянно бегает по холсту, то и третий объект постоянно будет вертеться. Добавьте третий объект и соберите для него такой скрипт:



Команда *повернуться к ...* заставляет объект, к которому она применяется

поворачиваться в направлении того объекта, который выбран в раскрывающемся списке этой команды. Цикл *всегда* здесь также необходим. Иначе объект повернется только один раз в самом начале. Поскольку второй объект у нас постоянно двигается, то и следить за ним надо всегда.

Нажмите кнопку запуска и посмотрите получившуюся анимацию. Все три объекта начинают свое движение одновременно, но двигаются по-разному, т.к. каждый из них управляется собственным скриптом.

Слои

Обратите внимание, какой из объектов при движении находится сверху, как бы перекрывает другие. Скорее всего, это будет третий объект. Второй объект при перемещении как бы проходит под ним и в тоже время, если вы заметили, находится выше первого. Получается, что на холсте есть как бы три слоя, и каждый объект двигается только по своему. Что же делать, если надо, чтобы третий объект находился ниже второго? Для этого достаточно взять второй объект (зажав левую кнопку мыши, когда курсор находится над ним) и положить его сверху третьего. Проделайте это и посмотрите анимацию. Отметьте, какой из объектов находится ниже. Остановите выполнение, теперь положите третий объект на второй. Снова запустите программу и отметьте разницу.

Следует знать, что перемещать объекты на другие слои на холсте можно не только вручную с помощью мыши, но и программно, когда в сценарий объекта встраиваются команды *перейти в верхний слой* и *перейти назад на ... слоев*.

Урок 3. Последовательное и одновременное выполнение

Одновременное выполнение скриптов (программ)

Вспомните, как двигались три объекта на прошлом занятии. Они шли вместе, одновременно. Несмотря на то, что движение у каждого было свое (кот ходил слева направо, второй герой под углом в 45 градусов, а третий просто крутился на месте), они все равно начинали движение в одно и то же время — при нажатии на кнопку запуска. Это был пример, когда мы можем говорить об одновременном (параллельном) выполнении разных блоков команд (скриптов). В серьезном программировании используются другие, более умные слова (многопоточность и т.п.).

При этом в Scratch можно сделать так, что два (или больше) скрипта одновременно будет выполнять вообще один объект. Допустим, кот будет шагать и при этом менять свои размеры, форму и другие свойства. При этом ходьба и изменение свойств друг с другом никак не будут связаны. Сделать это можно, например, разместив два разных скрипта в ячейке **скрипты** для кота:

При щелчке на кнопке запуска начнут работать оба скрипта сразу. Первый будет заставлять кота ходить слева направо. Это нам уже знакомо. Второй скрипт будет менять внешний вид кота. Фиолетовые команды связаны с кнопкой **внешность**. Во втором скрипте выше используется команда *изменить ... эффект на ...*. После слова «изменить» в раскрывающемся списке можно выбрать понравившийся нам эффект, а в поле с числом прописать, на сколько единиц его изменять. Команда *убрать графические эффекты* возвращает объект к его исходному внешнему виду. Итак, второй скрипт в цикле *всегда* выполняет следующие действия: изменяет цвет объекта → оставляет объект в таком состоянии на 1 секунду → возвращает объект к прежнему цвету → искривляет объект с помощью завихрения → оставляет в таком состоянии на 1 секунду → возвращает к исходной форме. Составьте два подобных скрипта для одного объекта. Можете попробовать другие графические эффекты (например, рыбий глаз или мозаику).

Последовательное выполнение скриптов (программ)

При последовательном выполнении скриптов сначала все действия должен совершать один объект, затем второй и т.д. Как же это можно сделать в Scratch?

Например, можно просто использовать команду ждать в начале скрипта второго объекта. Но на самом деле это будет никакое не последовательное выполнение скриптов, т.к. оба они начнут работать одновременно, просто у второго объекта сначала будет долго работать команда *ждать*. Хотя на сцене можно таким образом получить желаемый эффект: сначала действия будет совершать один объект, а через некоторое время - другой.

Рассмотрим, более грамотный способ организации последовательного выполнения скриптов. Когда один объект завершает выполнение ряда своих команд, он должен подать какой-нибудь сигнал-сообщение второму объекту, а тот, в свою очередь, должен его принять. В Scratch для этого есть две специальные команды, связанные с кнопкой **контроль**: *передать ...* и *когда я получу ...*. Команда *передать ...* дается объекту, который уже закончил все или некоторые свои действия, а *когда я получу ...* применяется к объекту, который начинает работать вторым. Вместо трех точек в этих командах вписывается сообщение, которые мы сами назначаем.

Организуем для двух объектов последовательное выполнение скриптов. Допустим, по нашему сценарию планируется такая анимация: кот сначала ходит туда-сюда, после чего исчезает, и по экрану начинает метаться сыр (например).

Если у вас сейчас в ячейке скриптов кота две программы, то удалите ту, которая меняет внешний вид объекта, чтобы она нас никак не вводила в заблуждение.

Первый скрипт кота надо изменить на такой:

В команде *передать ...* надо вставить сообщение. Для этого открывается специальное окно, куда можно его вписать. Команда *спрятаться* используется для того, чтобы скрыть кота, после того, как он пошлет сообщение. Команда *показаться* в

начале скрипта требуется, чтобы снова отобразить объект, если он был спрятан до этого. Понятно, что при первом выполнении скрипта она не играет никакой роли. Цикл *всегда* был заменен на цикл *повторить …*, т.к. нам больше не надо, чтобы объект двигался всегда. Соберите такой скрипт.

Теперь посмотрим, что будет делать второй объект. Он должен получить сообщение, после чего совершить какие-нибудь действия:

Таким образом, второй скрипт не начнет выполняться, пока первый не передаст ему сообщение *«первый вызывает второго»*. Обратите внимание, что команда *спрятаться* первого объекта находится ниже, чем *передать …*. Это значит, что у нас не совсем последовательное выполнение скриптов. Ведь первый еще не завершается, когда начинается второй. В принципе, чаще именно так и бывает. Добавьте на сцену второй объект и запрограммируйте его как показано выше.

Изменение размеров объектов

Дополнительно на этом уроке уделим внимание еще одной функциональной возможности Scratch, которая позволяет изменять размер объектов. Вы уже могли заметить, что холст слишком маленький, а добавляемые на него объекты зачастую слишком крупные. В Scratch можно менять размер объектов вручную или программно.

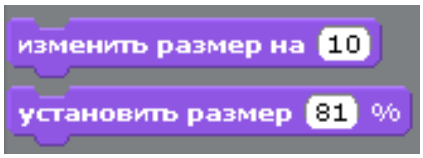
В первом случае используются две последние кнопки в ряде из четырех кнопок вверху:

Здесь первая кнопка позволяет дублировать (копировать) объект. При этом копируются и все его скрипты. Вторая кнопка предназначена для удаления объекта. Может случиться, что добавленный объект оказался лишним. Его можно будет удалить с помощью этой кнопки. Третья кнопка отвечает за увеличение объекта, а четвертая — за его уменьшение.

После того, как любая из этих кнопок выбрана, курсор мыши меняет свой вид.

Далее следует щелкнуть по объекту, которые мы планировали изменять. После окончания изменений следует щелкнуть курсором мыши в пустое место холста, чтобы сбросить выбранный инструмент.

Другой способ изменить размер объектов — использование специальных команд в скрипте (программе). Это значит, что размер объекта можно изменять с помощью программы. Вот эти две команды:



Команда *изменить размер на ...* увеличивает или уменьшает объект на указанное количество точек. Положительные числа в поле будут увеличивать объект, а отрицательные (с минусом) — уменьшать его.

Команда *установить размер ... %* вычисляет размер объекта, относительно его оригинального (самого первоначального) размера. 100% - это и есть оригинальный размер. Если прописать в поле 50, то объект уменьшится в 2 раза, а если 25, то в 4. Скажем, чтобы увеличить героя в 2 раза, требуется задать 200%.

Самостоятельная работа

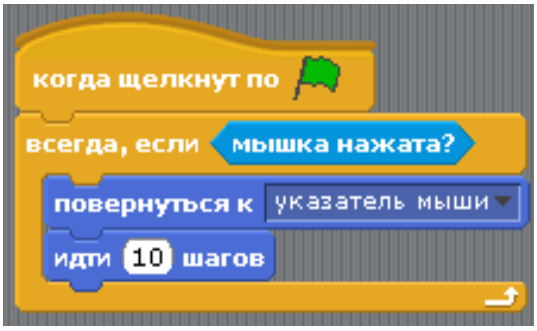
Добавьте третий объект, который появляется в центре холста, после того как второй закончит выполнять свои команды. Новый объект должен сначала постепенно уменьшаться, а затем постепенно увеличиваться до прежних размеров.

Урок 4. Интерактивность, условия и переменные

Интерактивность

Под **интерактивностью** будем понимать возможность взаимодействия между объектами, принадлежащими разным средам. Например, если в написанной нами программе задано, что кот из Scratch догоняет мышь из Scratch, а она реагирует на него и убегает, то это не будем считать интерактивностью. И кот и мышь - объекты одной среды. А вот если кот из Scratch реагирует на действия реального человека (например, нажатие пользователем определенной клавиши), то это уже интерактивность, т.к. объекты принадлежат разным "средам обитания".

Пусть требуется сделать так, чтобы человек мог управлять поведением кота на холсте Scratch. Если написать такой сценарий



, то после его запуска, объект будет преследовать указатель, если зажать левую кнопку мыши.

Команда *мышка нажата?* находится в разделе **сенсоры**, где сгруппированы команды, назначение которых — проверять наличие заданных изменений в окружающей среде объекта. Например, команда *мышка по x* отслеживает, где находится курсор мыши по оси **x**.

Управляющая конструкция *всегда, если ...* похожа на цикл *всегда*, за исключением того, что эта конструкция сочетает в себе **цикл с условием**. Такой цикл выполняется лишь тогда, когда условие истинно. В данном случае, когда мышка нажата.

Составьте такой сценарий. Желательно в ячейке свойств объекта включить кнопку "только поворот влево-вправо". После чего добавьте на сцену еще один объект, продублируйте сценарий первого объекта и перенесите его на второй. В итоге по холсту следом за курсором мыши должны "ходить" два объекта.

Переменные и условный оператор

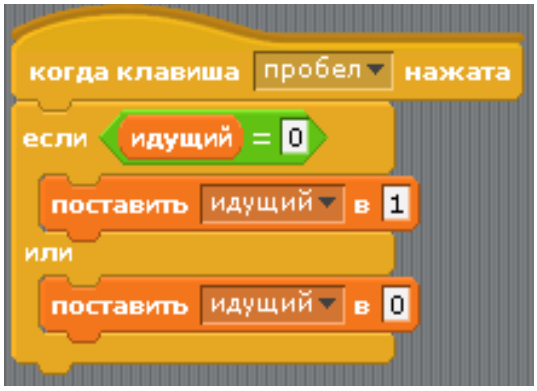
Усложним программу следующим образом. Пусть объекты перемещаются не вместе, а врозь: сначала один бегает за курсором, потом — другой. Переключение

"активности" между объектами пусть происходит с помощью клавиши **пробел**. Как организовать такую программу?

Ответить на этот вопрос будет очень сложно, если не знать об очень важной составляющей любого программирования — о **переменных**. Переменные представляют собой ячейки, в которых хранятся какие-либо данные (число, строка и др.). Причем содержимое ячеек может меняться в процессе выполнения программы. В какой-то момент можно проверить, что там находиться и в зависимости от этого выполнять определенные команды или нет.

Создадим ячейку-переменную под именем **идущий**. Для этого нужно нажать на кнопку **переменные** и, затем, в ячейке команд Scratch нажать **Создать переменную**; в открывшемся диалоговом окне ввести ее имя и нажать **ОК**. После чего у нас появится сама переменная и несколько команд для ее изменения.

Нам нужна эта переменная, чтобы в зависимости от ее значения двигался первый объект или второй. Пусть если **идущий** = 0, то двигается первый объект, а если переменная равна единице, то — второй. Изменение значения переменной будет происходить при нажатии клавиши **пробел**. Поскольку данная часть программы относится к обоим объектам, то уместно расположить скрипт в объекте **Сцена** (хотя не обязательно), переключившись на него. Вот этот сценарий:



Вместо пробела можно выбрать другую клавишу в раскрывающемся списке первой команды сценария. Далее идет сложная инструкция *если — или*. Она работает так: если условие в части *если* истинно, то выполняются вложенные команды после *если*, иначе — будут выполняться вложенные команды после *или*. Обе "ветки" никогда вместе не выполняются, они взаимно исключают друг друга.

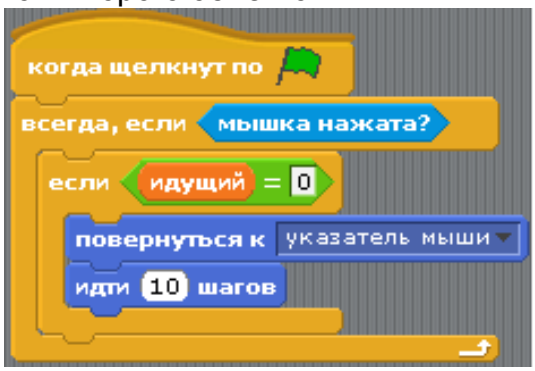
Теперь рассмотрим условие при инструкции *если*. Зеленые команды относятся к **операторам**. Операторы — это такие команды, которые что-то делают с данными. Например, выполняют математические операции или сравнивают числа между собой. В данном случае выбран оператор сравнения на равенство. Он проверяет равно ли выражение слева от знака "=" выражению справа. Если — да, то будет работать код после *если*, если — нет, то код после *или*. В нашем случае сравниваются значение переменной **идущий** и 0.

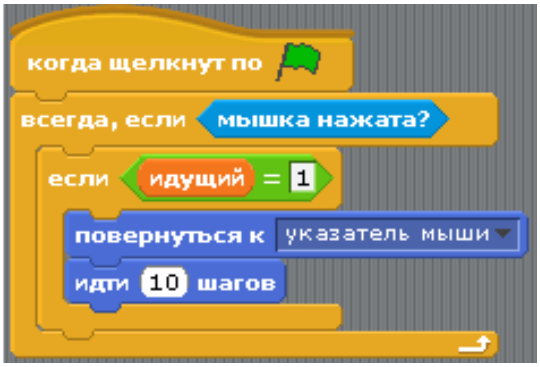
Мы ничего не "клали" в переменную, однако по-умолчанию ей и так

присваивается значение 0, поэтому нам это было делать не обязательно. К тому же, если вдруг изначально в переменной будет не ноль, то ничего страшного: сработает ветка *или*.

Что же все-таки делает этот скрипт? Если значение переменной равно 0, то оно меняется на 1. Если же нет, то меняется на 0. Получается, что нажатие клавиши пробел попеременно меняет значение переменной **идущий**. Однако при выполнении программы мы не увидим никаких изменений. Действительно, ведь объекты еще "не знают", к какому значению переменной привязаны их действия. Составьте вышеприведенный скрипт и переключитесь на первый объект.

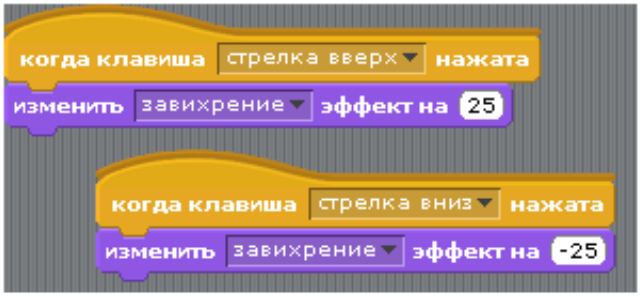
Требуется организовать проверку значения переменной. Для этой цели прекрасно подойдет простая инструкция *если*. Вот такие сценарии получаются для первого и второго объектов.





Разница между ними только в том, с чем сравнивается значение переменной: с нулем или единицей. А как мы знаем, ее значение меняется при нажатии пробела. В результате, когда программа запущена, мы можем менять "активный" объект. Доделайте программу и посмотрите, как она работает.

Можно и дальше поиграть с интерактивностью. Пусть объекты меняют свой внешний вид при нажатии определенных клавиш. Например, так:



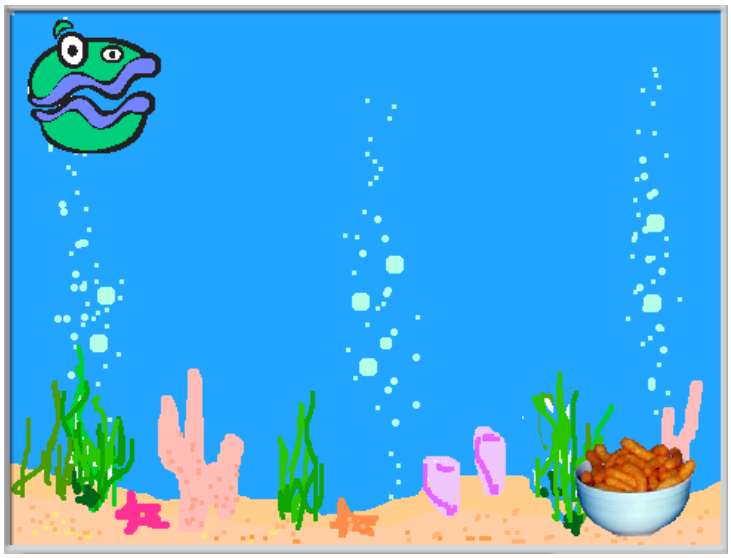
Можете попробовать свои варианты.

Самостоятельная работа

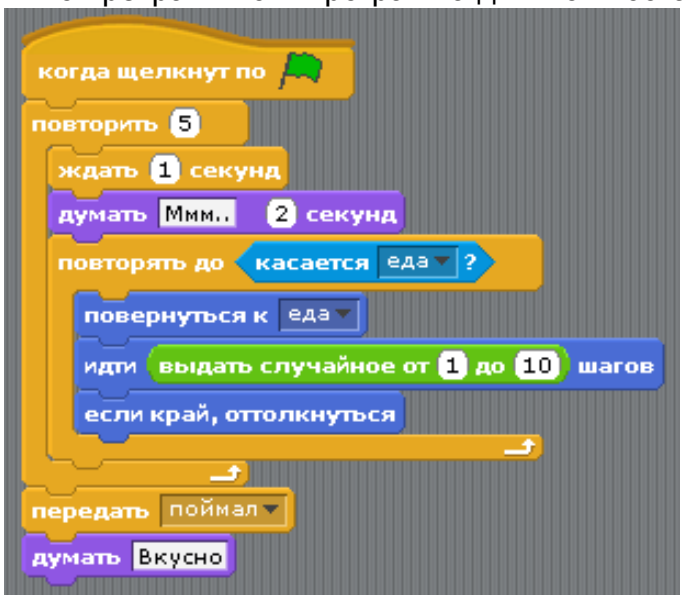
Составьте программу в среде Scratch, при выполнении которой пользователь может управлять объектом с помощью стрелок на клавиатуре, а при нажатии каких-нибудь других клавиш, объект "думает" разные "мысли".

Урок 5. Случайные числа

Предположим, что требуется в Scratch реализовать следующий сценарий. Под водой моллюск пытается поймать пищу, которая от него постоянно отпрыгивает в новое случайное место. Через какое-то время моллюску все-таки удается поймать и съесть еду.



Итак, моллюск всегда должен двигаться за едой. Причем через определенный ряд действий это должно прекратиться. Программа для моллюска может быть такой:



Проследим логику работы этого скрипта. Сначала моллюск стоит и думает. После чего он поворачивается к еде и идет к ней до тех пор, пока не коснется ее. Все эти действия повторяются 5 раз. Обратите внимание, в команду *идти ... шагов* вставлена команда *выдать случайное число от ... до ...*. Вторая команда выдает случайное для нас число в указанном диапазоне, т.е. любое число от первого числа в команде до второго включительно. В данном случае эта команда влияет на то, с какой скоростью двигается моллюск. Объясните, почему?

После того, как моллюск совершит несколько вышеописанных циклов, он передаст в окружающую среду сообщение "поймал". В результате у него должна появиться соответствующая ситуации мысль.

Если сейчас запустить программу, то моллюск подойдет к еде, пять раз подумает "Ммм.." и один раз "Вкусно". Составьте скрипт для моллюска.

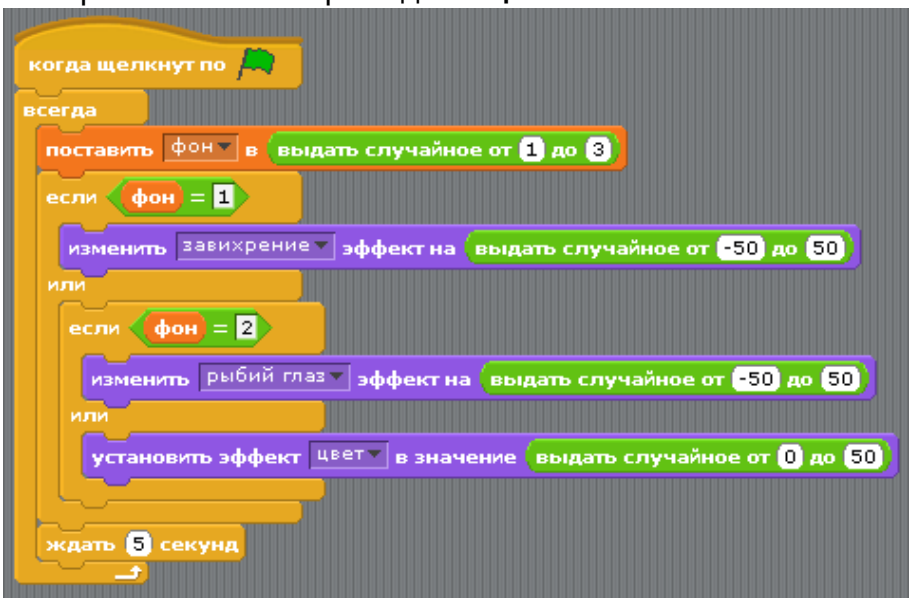
Теперь подумаем, что должна делать еда.

Она всегда должна оказываться в неожиданном для моллюска месте, если тот касается ее. "Случайное место" определяется двумя командами *идти в x: ... y: ...* и *выдать случайное число от ... до ...*. Т.е. после того, как координаты **x** и **y** определяются случайным образом, объект перемещается в заданную точку.

Второй скрипт еды очень прост. Если в "пространстве" появляется сообщение "поймал", то еда должна его получить и спрятаться.

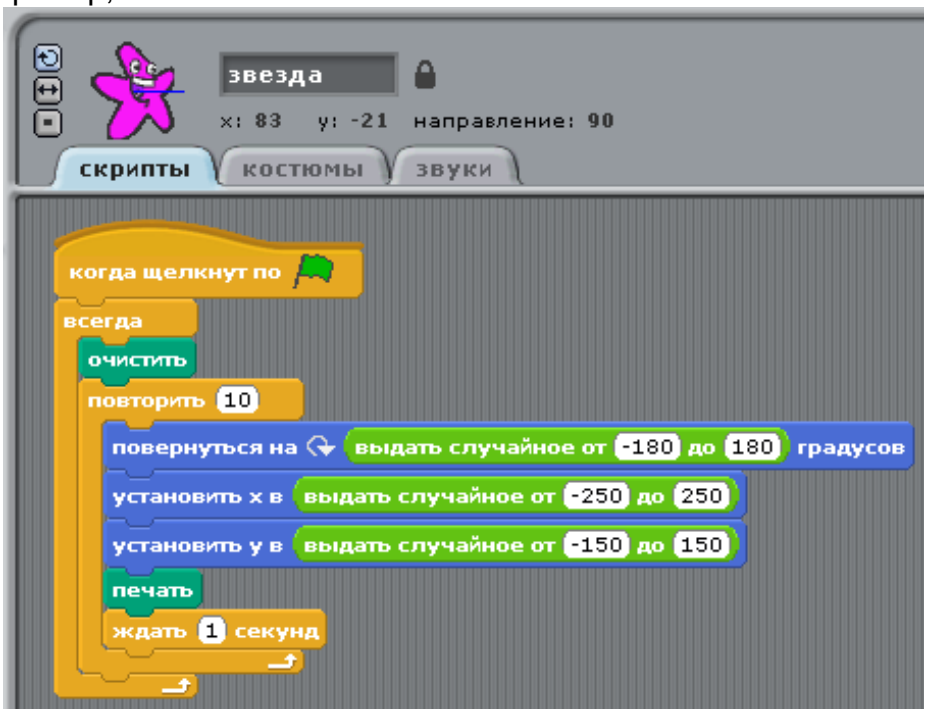
Теперь программа будет работать так, как надо: еда появляться в случайном месте, а моллюск бегать за ней до определенной поры. Составьте сценарии для второго объекта и оцените, как работает вся программа.

Анимацию можно сделать интересней, если придать сцене больший динамизм. Пусть картинка сцены слегка видоизменяется, причем тоже случайным образом. Рассмотрим вот такой скрипт для **Сцены**:



В нем присутствует переменная **фон**, значение которой определяется случайно, и это значение может быть только числами 1, 2 или 3. Если **фон** = 1, то на картинке появятся завихрения случайной силы, если **фон** = 2, то появится эффект "рыбий глаз". Во всех остальных случаях будет меняться цвет фона. Значение переменной будет обновляться через каждые 5 секунд. Составьте подобный сценарий для своей программы.

Иногда бывает необходимо, чтобы объект как бы множился на сцене, т.е оставлял свои копии. В Scratch это можно сделать, используя команду *печать*, относящуюся к кнопке **перо**, где собраны команды, позволяющие рисовать на холсте. Если объекту дать команду *печать*, а затем переместить в другое место, то на прежнем останется изображение объекта. Сценарий можно составлять по-разному, например, вот так:



Попробуйте сами.

Самостоятельная работа

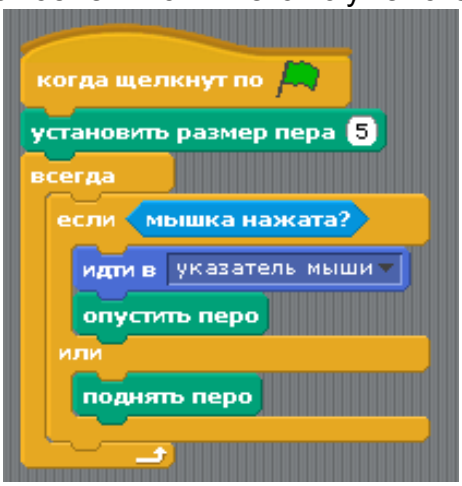
1. Придумайте сценарий, в котором необходимо было бы использовать случайные числа.
2. Попробуйте реализовать его в среде Scratch.

Урок 6. Рисование в Scratch

Рисование мышью

В среде программирования Scratch можно почти также рисовать как в среде KTurtle. Но Scratch более многофункциональная среда, в ней можно писать программы, которые не просто выводят изображения на холсте. В Scratch можно запрограммировать, скажем так, интерактивное рисование, когда пользователь сам формирует изображение на холсте уже в процессе выполнения программы (т.е. после нажатия кнопки запуска).

В KTurtle можно было организовать взаимодействие программы с пользователем лишь посредством диалоговых окон. В Scratch можно использовать мышь и клавиатуру. Для наглядной иллюстрации этих возможностей импортируем из библиотеки Scratch объект *Drawing pencil (рисующий карандаш)* он находится в каталоге *Things (Вещи)*. Этот объект появляется с уже готовым скриптом:



Удалите со сцены кота и добавьте рисующий карандаш. Запустите программу и выясните, что делает этот карандаш? Остановите выполнение программы.

Теперь подробно разберем скрипт карандаша. После того как программа запущена, толщина пера увеличивается до 5 точек из-за команды *установить размер пера ...*, которая находится в группе команд, отображающихся при нажатии кнопки **перо**. На самом деле рисует вовсе не карандаш, а именно невидимое перо, которое мы привязываем к нашему объекту. Другими словами, можно использовать для рисования абсолютно любой объект (кота, человека, мяч и т.п.).

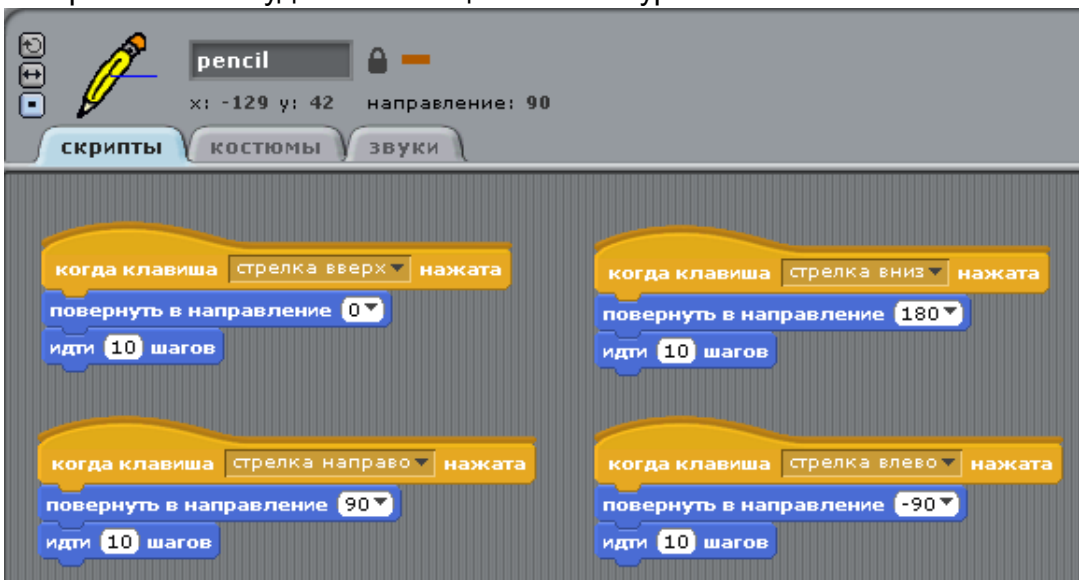
Далее в скрипте используется цикл *всегда*, следовательно, то, что находится внутри него, выполняется постоянно, пока программа работает. Всегда выполняется следующее: если левая кнопка мыши нажата, то карандаш перемещается в место, где находится ее указатель и перо опускается; если кнопка не нажата, то перо поднимается. Когда перо опущено на холсте остаются следы от его перемещения, т.е. двигая мышью, мы двигаем и карандашом и привязанном к нему пером в

опущенном состоянии. В результате получается, что при выполнении программы карандаш рисует, когда кнопка мыши нажата, и не рисует, когда отжата. Рисование происходит в том месте, где находится курсор мыши.

Также добавьте скрипт для объекта **Сцена**, который очищает холст при запуске программы: *когда щелкнут по (зеленый флаг)* → *очистить*.

Рисование с помощью клавиатуры

Теперь составим «рисующий» скрипт самостоятельно. Для этого добавим на сцену еще один карандаш, удалим имеющуюся у него программу и назовем его **`pencil`**. Управлять им будем с помощью клавиатуры.



Такие вот скрипты позволят нам двигать карандаш по холсту с помощью стрелок на клавиатуре. Вспомните KTurtle: когда черепашка смотрела вверх, то ее направление было равно 0, когда вправо, то 90 и т.д. Здесь то же самое. Составьте скрипты и не забудьте переключиться на режим «не поворачивать» в ячейке свойств объекта, чтобы карандаш не вертелся на холсте.

Если теперь запустить программу и управлять карандашом с помощью стрелок, то мы увидим, как он перемещается по холсту. Однако следов не оставляет, т.к. перо не было опущено. Чтобы исправить этот недочет, надо добавить еще один маленький скрипт для карандаша: *когда щелкнут по (зеленый флаг)* → *опустить перо*. Попробуйте, но имейте в виду, что этот скрипт будет сейчас исправлен. Теперь карандаш должен не только двигаться, но и рисовать.

Усложним программу для карандаша, добавив еще несколько небольших сценариев:

Они очень простые и очевидные. Теперь, когда запускается программа, перо должно всегда подниматься. Оно будет опускаться при нажатии клавиши **d**. Имейте ввиду, что буквенные клавиши могут срабатывать лишь при включении английской раскладки клавиатуры. При нажатии клавиши **u** перо снова поднимется. За счет всего этого мы можем перемещать карандаш не рисуя на холсте, затем по желанию опускать перо и рисовать. Наш карандаш может рисовать случайным цветом за счет использования комбинации команд *установить цвет пера ...* и *выдать случайное число от ... до ...*. Переключение цвета происходит при нажатии пробела. Добавьте эти скрипты вашему карандашу.

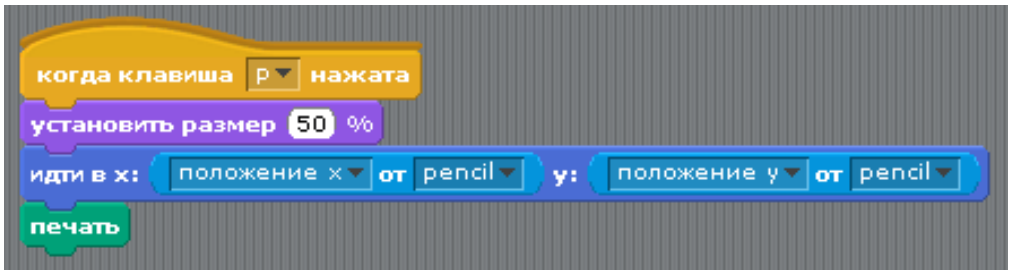
Управляемая печать

В конце давайте сделаем следующее. Добавим на сцену третий объект, например, мяч, и назовем его **штамп**. Пусть он оставляет копию своего изображения при нажатии клавиши **p** там, где находится запрограммированный нами карандаш в текущий момент. Как это сделать?

1. Во-первых надо сначала получить текущие значения координат **x** и **y** карандаша.
2. Во вторых, надо переместить мяч в точку с координатами **x** и **y**.
3. И в третьих, надо выполнить команду *печать*.

Самый сложный здесь первый пункт. Чтобы узнать значение какого-либо свойства объекта, используется команда *<свойство> от <объект>*, находящаяся в **сенсорах**. Скорее всего, она у вас выглядит так: *положение x от drawing_pencil*.

Итоговый скрипт будет примерно таким:



Команда *установить размер ... %* не обязательна. Когда выполняется конструкция *идти в x: положение x от pencil y: положение y от pencil*, то сначала выясняются координаты карандаша, а затем они подставляются в команду *идти в x: ... y: ...*.

Запрограммируйте ваш **штамп**. Запустите программу и оцените ее возможности.

Самостоятельная работа

Вспомните, как программировались изображения на холсте в среде KTurtle. Напишите программу в Scratch, в результате выполнения которой на холсте появлялись бы две геометрические фигуры (например, окружность и пятиугольник).

Урок 7. Диалог с программой

Языки программирования по сути выполняют те или иные операции над данными (числами, строками, списками и др.). Эти данные в программу может "заложить" программист или же они будут поступать от пользователя в процессе выполнения программы. Например, если программа выполняет умножение числа на само себя, то это число может быть получено от пользователя. Чтобы пользователь ввел число, должна выполниться какая-то встроенная команда, предоставляющая интерфейс для ввода данных.

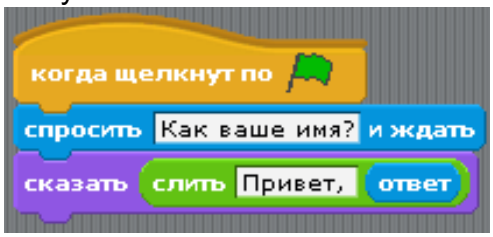
В Scratch команду, позволяющую пользователю ввести данные, можно найти в разделе **сенсоры**. Она называется *спросить ... и ждать*. Вместо фразы "Как ваше имя?" можно вписать другую фразу, а также поставить туда переменную или что-то другое. Когда команда *спросить ... и ждать* выполняется, то внизу холста появляется поле для ввода. Пользователь должен туда ввести данные и нажать **Enter** или галку в конце поля. Проверьте самостоятельно, как работает эта команда.

После того, как команда *спросить ... и ждать* выполнена, текстовое поле исчезает с холста. Куда же попадают данные, которые мы в него вводили? Они оказываются в переменной *ответ*. Это встроенная переменная (ее не нужно создавать) и находится она в **сенсорах**.

Попробуем составить сценарий, при выполнении которого кот спрашивает у нас имя, а затем приветствует нас по имени (например, "Привет, Саша").

1. Сначала требуется спросить у пользователя имя.
2. Затем следует склеить "Привет, " и то, что ввел пользователь. Это можно сделать с помощью команды *слить ... ...*, находящейся в **операторах**.
3. Команда *сказать ...* поможет вывести сообщение на экран.

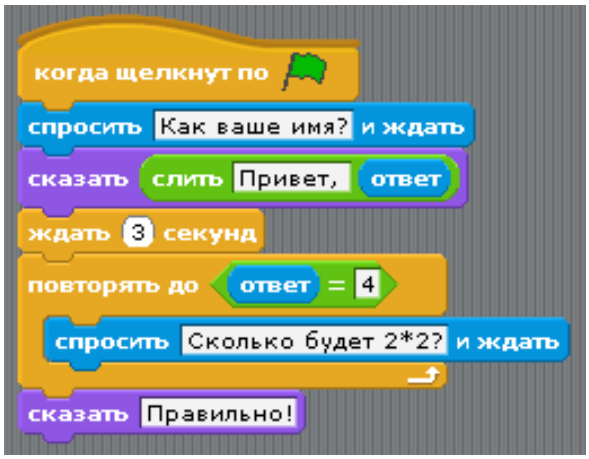
В результате скрипт получится таким:



Соберите его и проверьте, как это работает. Обратите внимание, что после слова "Привет" и запятой добавлен также пробел, чтобы разделить слова.

Усложним нашу программу. Пусть после того, как кот поздоровается, он спросит, сколько будет 2*2? Пусть он спрашивает это до тех пор, пока пользователь не даст правильный ответ. Для того, чтобы команда *спросить ... и ждать* выполнялась постоянно до определенного момента, ее необходимо поместить в конструкцию *повторять до ...*. Условием, когда эта конструкция должна прекратить выполняться, будет правдивость результат выполнения операции *ответ = 4*.

Соберите такой сценарий:



Если теперь его запустить на выполнение, то произойдет следующее:

1. Кот спросит имя.
2. Поздоровается. Чтобы приветствие сразу не исчезло, ждет 3 секунды.
3. Будет спрашивать "Сколько будет 2 * 2?" до тех пор, пока пользователь не введет правильный ответ.
4. Когда пользователь введет цифру 4, кот скажет "Правильно!".

Самостоятельная работа

1. Составьте в среде Scratch программу, которая спрашивала бы у пользователя, на сколько процентов увеличить или уменьшить кота. После чего изменяла бы размер объекта на холсте.
2. Напишите программу в Scratch, запрашивающую у пользователя количество сторон фигуры и угол между сторонами, а затем рисующую полученную фигуру на холсте.

   Подсказка. Во втором задании необходимо использовать две переменные.

Урок 8. Создание объектов и костюмов

Обычно Scratch поставляется с большой коллекцией готовых спрайтов и фонов. Новички в Scratch охотно ими пользуются. Но что делать, если вы придумали какой-то сценарий или программу, для которой требуются объекты, которых нет? Их можно нарисовать самим прямо в среде Scratch. Здесь есть встроенный редактор, позволяющий создавать спрайты. Чтобы его запустить, нужно нажать на кнопку **Рисовать новый объект** в ячейке, где отображаются объекты программы (нижняя правая ячейка Scratch).

Описание графического редактора

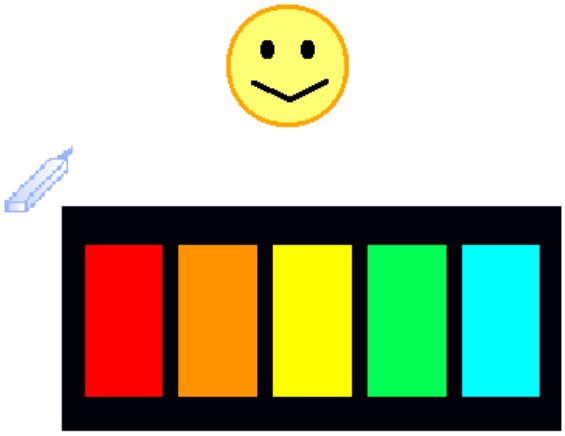
Если вы знакомы с любым графическим редакторам, то легко разберетесь, как работать в редакторе Scratch. Здесь можно делать следующее (начнем сверху):

- изменять размер объекта, поворачивать его по часовой и против часовой стрелки, переворачивать объект по горизонтали и вертикали;
- импортировать готовый объект, чтобы изменить его;
- полностью очищать рабочую область;
- отменять действия, а также применять их снова;
- использовать для рисования кисть и геометрические примитивы (эллипс, прямоугольник, линия); при использовании кисти и линии можно выбрать их толщину, эллипсы и прямоугольники могут представлять собой контуры или заполненные области;
- удалять элементы ластиком;
- выбирать цвет и заливать им области; заливка может быть сплошной или градиентной;
- работать с текстом;
- выделять области изображения для перемещения или дублирования;
- можно задавать цвет фона; это важно, например, при градиентной заливке;
- ну и наконец, можно менять масштаб изображения.

Создание объектов

Создадим в Scratch такую программу. На холсте находится смайлик, пульт и указатель. В зависимости от того, где указатель располагается на пульте, смайлик меняет выражение своего "лица". Положение указателя пусть определяется с помощью цвета, а не координат.

Для реализации такой программы нам понадобится создать примерно такие объекты:



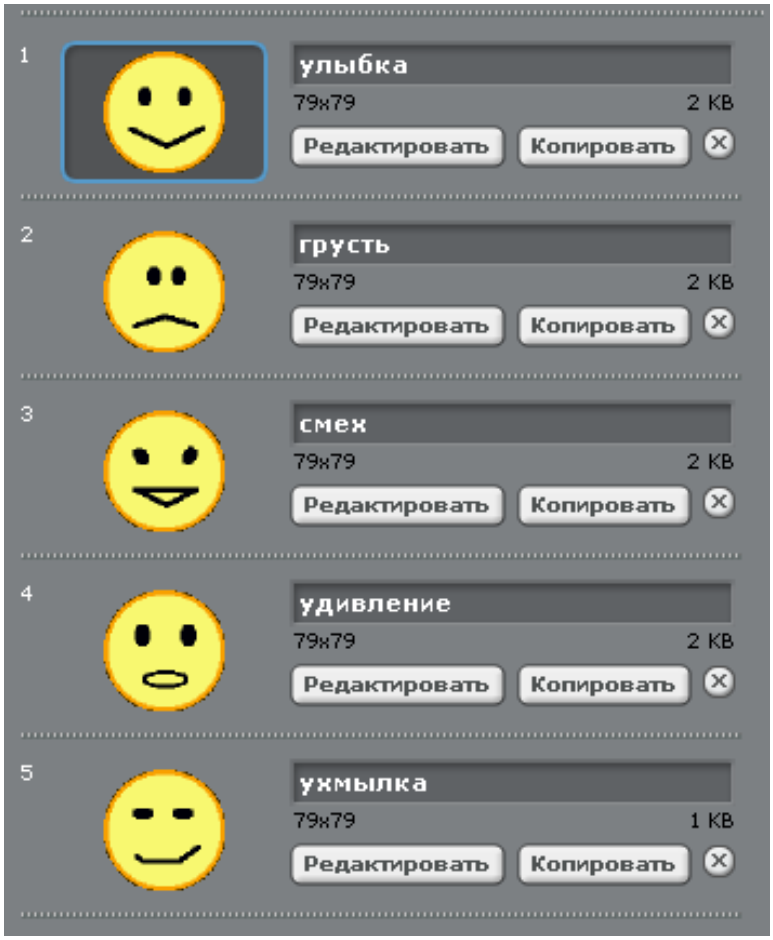
Нарисуйте что-то подобное в Scratch самостоятельно. Назовите объекты "Смайлик", "Пульт" и "Указатель".

Создание костюмов

По идее в нашей программе смайлик должен изменять свою "улыбку" и "выражение глаз"; т.е. он должен оставаться, с одной стороны, самим собой, а с другой, все-таки меняться.

Для изменения объектов в Scratch используется понятие *костюмов*. Каждый объект имеет хотя бы один костюм. В этом можно убедиться, если перейти на вкладку **костюмы** в среднем столбце окна Scratch. Чтобы добавить новый костюм надо нажать на кнопку **Рисовать**, **Импорт** или **Камера**. Однако чаще всего, придется изменять уже готовый костюм. В этом случае копируют уже существующий костюм, а затем редактируют его.

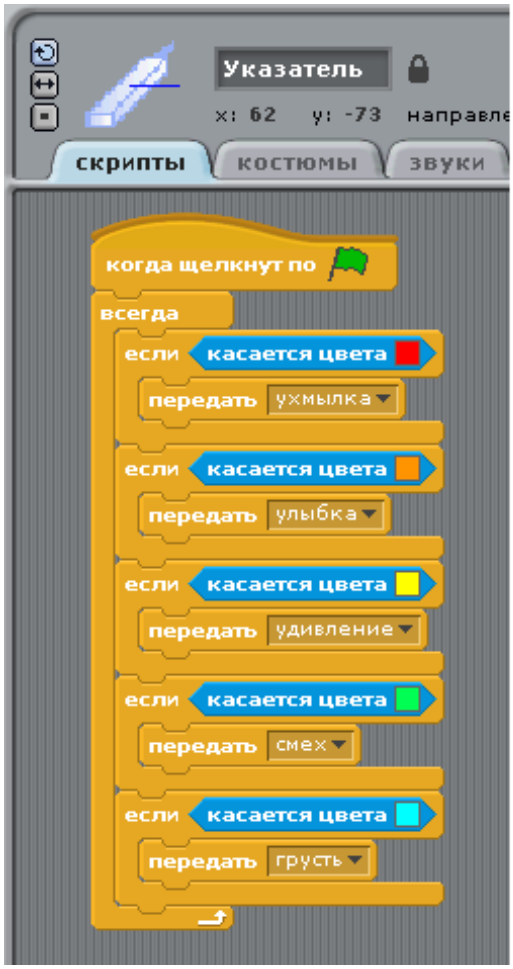
Создайте для смайлика пять разных костюмов. Должно получиться примерно так:

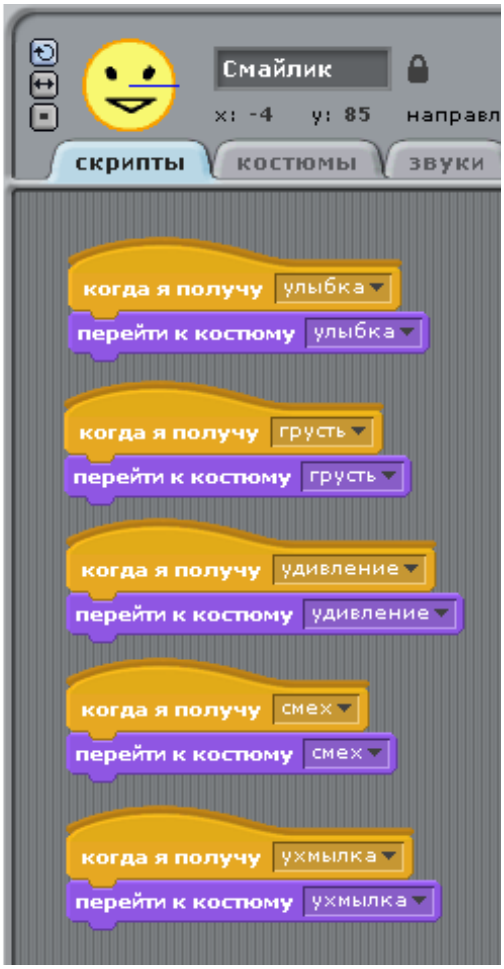


Создание программы

Теперь у нас все готово для написания самой программы. Когда указатель будет находиться на определенном цвете пульта, то смайлик будет "одеваться" в соответствующий этому цвету костюм.

Как же связать положение указателя с костюмом совершенно другого объекта (смайла)? Можно использовать сообщения. Когда указатель касается заданного цвета, то он передает соответствующее этому цвету сообщение. Смайл в свою очередь "ловит" сообщение и применяет тот костюм, которое ему соответствует. Вот какие должны быть скрипты для указателя и смайлика:





Составьте скрипты. Чтобы определить цвет для команды-сенсора *касается цвета ...* надо щелкнуть по цвету в ней, появится пипетка, после этого надо щелкнуть по нужному цвету на пульте.

Обязательно сохраните созданную программу (проект). Она нам еще понадобится.

Урок 9. Использование библиотеки объектов

Когда мы добавляем кем-то ранее подготовленный объект из библиотеки Scratch (кнопка **Выбрать новый объект из файла**), то, можно сказать, *импортируем* его в наш проект. Однако мы с таким же успехом можем делать обратное: из рабочего проекта Scratch добавлять объект в общую библиотеку. Это можно назвать *экспортом* объектов. Потом наш объект можно использовать в других проектах наряду с другими стандартными (существующими до этого) объектами.

Если к объекту был "привязан" программный код, то они экспортируются и импортируются совместно. Мы это уже видели в уроке 6 "Рисование в Scratch", когда взятый их коллекции объектов карандаш имел при себе небольшой скрипт. Теперь попробуем сделать обратное — добавим в библиотеку созданный нами на прошлом занятии смайлик. Сделать это можно так:

1. Открыть проект, содержащий интересующий нас объект.
2. Выделить объект.
3. В меню выбрать команду **Файл → Экспорт объекта...**.
4. В открывшемся диалоговом окне выбирать место для сохранения, ввести название файла и нажать ОК. Для собственных объектов желательно создать отдельную папку.

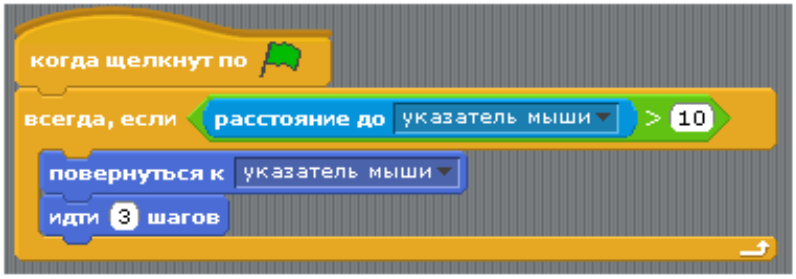
После того как объект экспортирован, его можно добавлять в новые проекты Scratch с помощью кнопки **Выбрать новый объект из файла**. Экспортируйте смайлик, затем создайте новый проект и добавьте в него этот объект.

Скрипт смайлика в новом проекте появится, однако работать не будет, т.к. в старом проекте он зависел от работы сценария другого объекта. В этом нет ничего страшного. Программу всегда можно дописать, переписать, либо удалить. Хотя обычно экспортируют достаточно независимые от других объектов спрайты, которые после импорта сразу же можно использовать.

Обычно начинающие программисты после того, как познакомятся с основами программирования на том или ином языке, изучают программы других, более опытных, коллег. Таким образом можно быстро приобрести опыт, узнать много нового, познакомиться с принятыми в сообществе принципами разработки и наилучшими способами решения тех или иных задач.

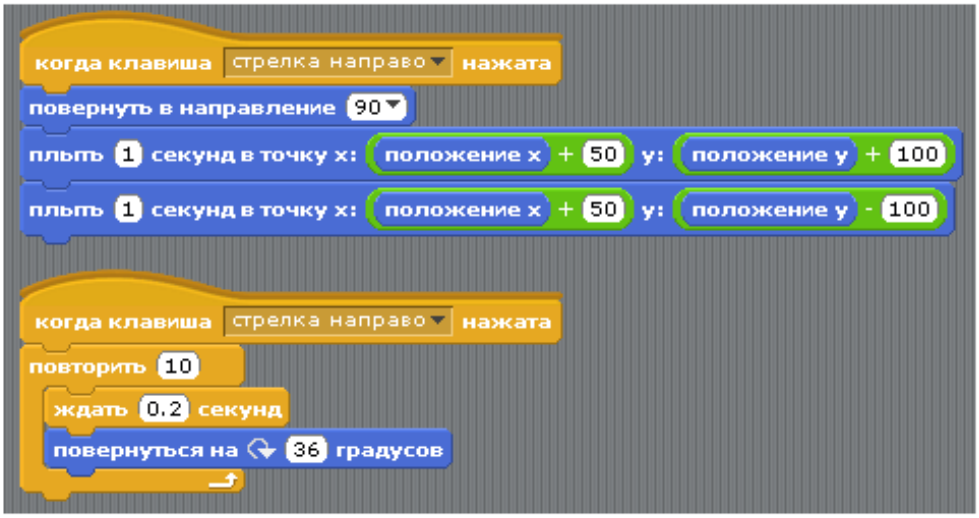
Рассмотрим некоторые объекты со скриптами, которые уже есть в стандартной библиотеке.

Hungry fish (голодная рыба)



У этого объекта очень простая программа. После запуска рыбка начинает преследовать указатель мыши. В коде программы можно отметить лишь одну особенность: объект останавливается, когда расстояние до курсора становится меньше, чем 10 точек. Это очень незначительное расстояние, поэтому рыба останавливается, когда касается курсора почти своим центром. Если увеличить это расстояние до можно добиться эффекта, когда объект останавливается, не доходя до курсора. Иногда именно это и требуется.

Jump-flip monster (кувыркающийся монстр)



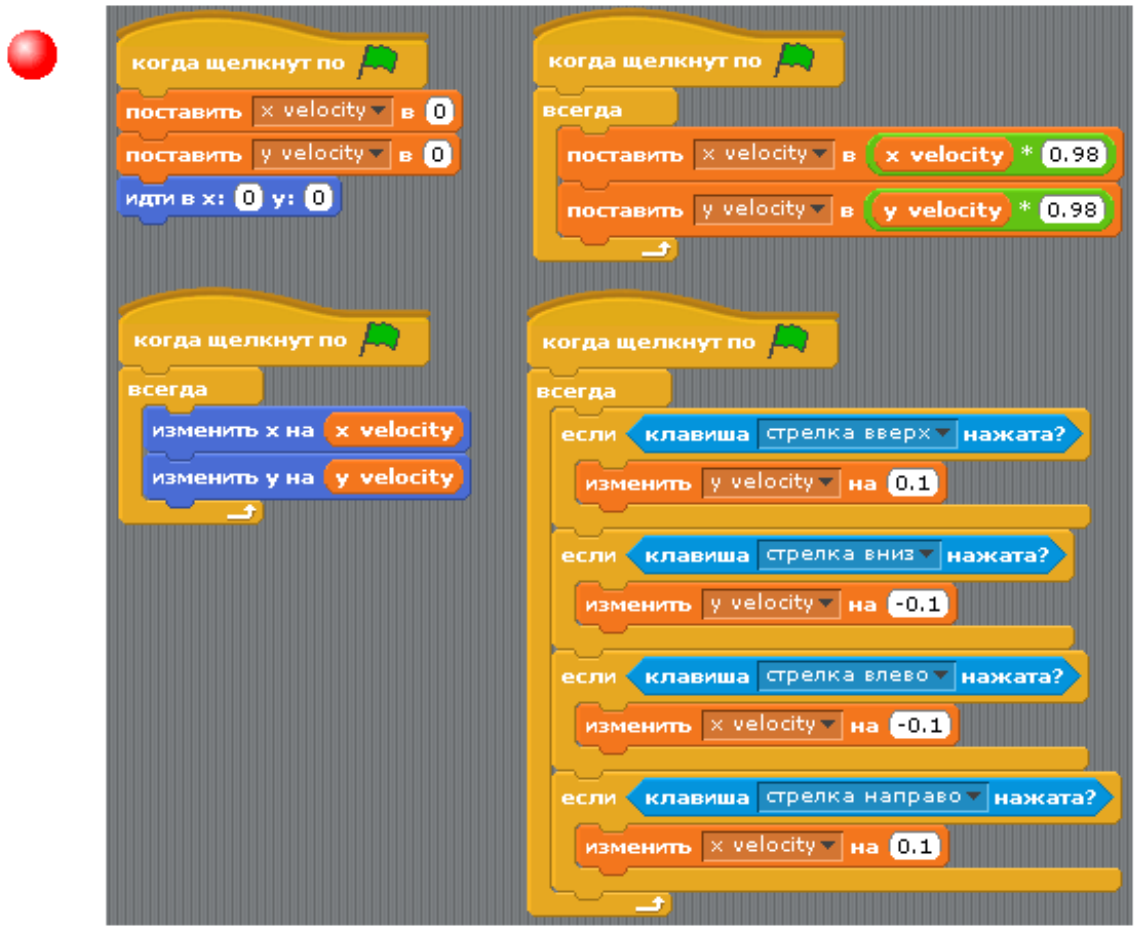
У монстра более интересная программа, чем у рыбы. Он совершает один кувырок при однократном нажатии стрелки вправо на клавиатуре. Разберемся, почему?

Первый скрипт (тот, который выше), заставляет объект сначала переместиться вперед на 50 точек и вверх на 100, а затем снова вперед на 50 и уже вниз на 100. Получается, что монстр как бы прыгает вперед. Команда *повернуть в направлении 90* просто устанавливает правильную ориентацию объекта перед движением.

Второй скрипт работает параллельно с первым. Он придает объекту вращательное движение. За одно нажатие стрелки вправо объект совершает 10 поворотов по 36 градусов, что в итоге составляет 360 градусов, а это, как известно,

полный оборот вокруг своей оси. Кроме того, полный оборот совершается ровно за 2 секунды (0.2 * 10). Именно столько выполняется первый скрипт. Поэтому получается синхронность: за один прыжок объект совершает один полный оборот.

Friction marble (трущийся камень)



Чтобы понять, как работает эта программа, надо немного разбираться в физике. Когда мы толкаем какой-то предмет горизонтально (допустим, мяч), то обычно он останавливается через некоторое время. Это происходит из-за трения о грунт. Если мы постоянно будем толкать мяч, то скорость его движения будет постепенно увеличиваться.

Программа для объекта friction_marble как раз и моделирует эти физические законы. Чем больше мы жмем на стрелку, тем большую скорость развивает шар. При прекращении воздействия его скорость постепенно уменьшается, после чего шар останавливается.

С помощью чего это достигается в программе?

В программе существуют две переменные — **`x velocity`** и **`y velocity`**, значения которых сразу после запуска программы приравнивается к нулю. Значения этих переменных постоянно (*всегда*) влияют на то, насколько единиц будет изменяться положение объекта.

Когда нажимаются клавиши со стрелками, то значения переменных меняются. Чем больше нажимается определенная стрелка, тем больше значение переменной отклоняется от нуля. В результате то количество единиц, на которые меняются **x** и **y** также увеличивается. Это создает эффект увеличения скорости перемещения объекта.

С другой стороны, когда воздействие отсутствует (клавиши не нажимаются), значение переменных постепенно приближается к нулю. Этот достигается за счет умножения на десятичное число от 0 до 1 (в нашем случае переменные умножаются на 0.98).

Самостоятельная работа

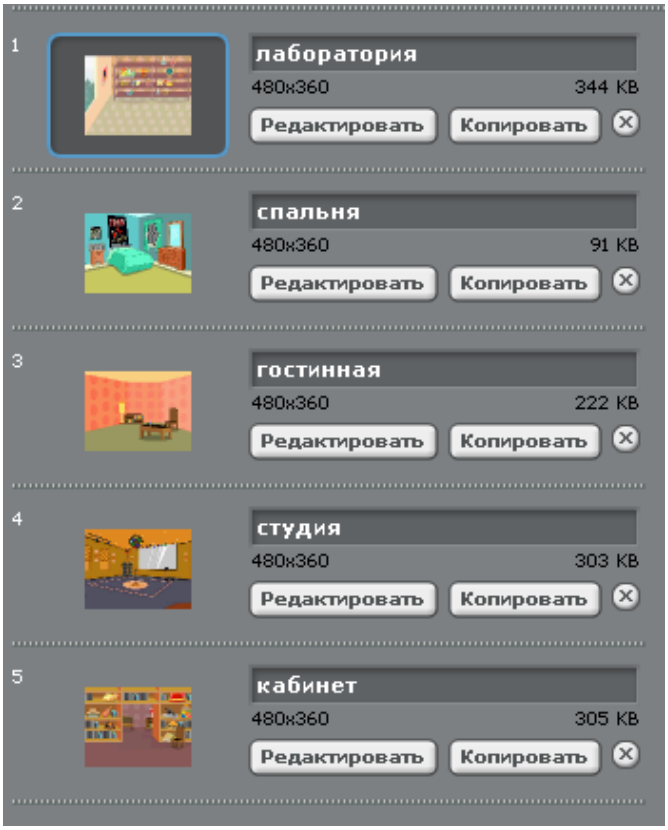
Создайте два-три оригинальных объекта с программами и экспортируйте их в общую библиотеку.

Урок 10. Смена фона

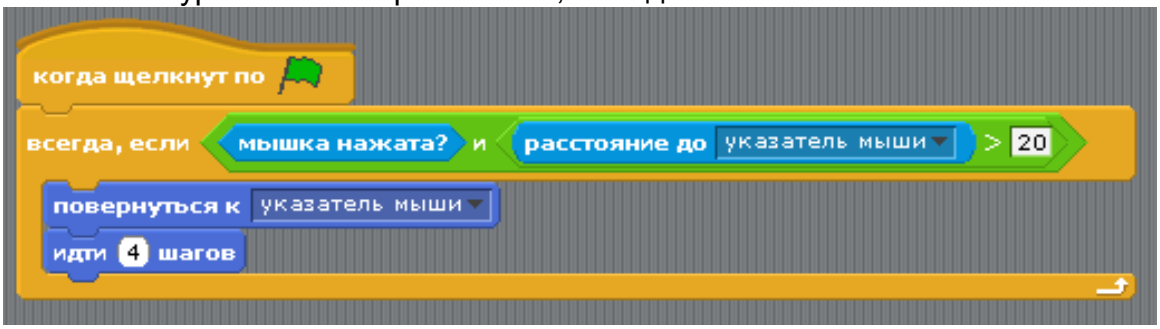
Представьте, что мы создаем игру, в которой герой, управляемый пользователем, может переходить из комнаты в комнату. При этом из определенной комнаты герой может попасть только в одну или несколько других, а не во все. Например, пусть схема расположения комнат будет такой:

Т.е. из комнаты 1 можно попасть в комнату 2. Из второй комнаты возможен переход в первую и третью и т.п. Событием, после которого будет производиться переход, будет местонахождение героя у соответствующего края сцены. Так для первой комнаты это будет правый край.

В среде программирования Scratch к фону сцены обращаются по его имени, (если программный код составляется для объекта **Сцена**) или по номеру фона (для других объектов). В нашей программе важно правильно расставить фоны по-порядку и назвать их.



Теперь выберем какой-нибудь объект в качестве героя и заставим его двигаться. Управляемое перемещение объекта может осуществляться с помощью мыши и клавиатуры. Если выбрать мышь, то код может быть таким:



Здесь, когда пользователь нажимает мышь, то объект движется по направлению к курсору.

Вспомним, что размер холста в Scratch равен 480x360 пикселей, а начальная точка системы координат находится в центре. Поэтому у правого края значение **x** = 240, у левого **x** = -240. Верхний край: **y** = 180; нижний край: **y** = -180. Пусть в дальнейшем фон меняется, когда соответствующая координата объекта равна по модулю 200 или 150. Это связано с удобством управления.

Алгоритм перехода объекта из комнаты 1 (лаборатория) в комнату 2 (спальня) может быть таким, как описано ниже.

Для героя:

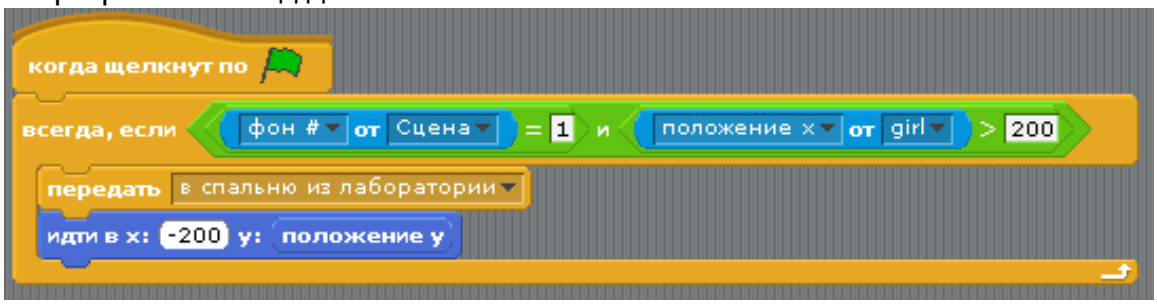
- Когда координата **x** объекта становится равной 200, он посылает соответствующее сообщение (например, "в спальню из лаборатории").

- Значение координаты **x** изменяется на противоположное, а **y** остается прежним. Это создаст эффект вхождения в другую комнату.

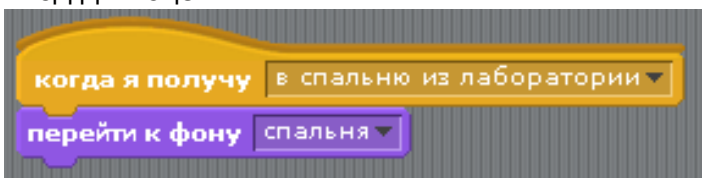
Для сцены:

- В зависимости от того, какое сообщение получено для сцены устанавливается соответствующий сообщению фон.

Программный код для объекта:



Программный код для сцены:



Сценарии перехода в другие комнаты аналогичны приведенным выше, за исключением значений. Всего должно получиться восемь комбинаций перехода из комнаты в комнату. Запрограммируйте эти переходы самостоятельно. Для помощи воспользуйтесь таблицей ниже.

фон от сцена = ...	*положение от ... ...*	*передать ...*	*идти в ... ...*
1	> 200	В спальню из лаборатории	x: -200 y: положение y
2	< -200	В лабораторию из спальни	x: 200 y: положение y
2	> 200	В гостиную из спальни	x: -200 y: положение y
3	< -200	В спальню из гостиной	x: 200 y: положение y
3	> 150	В студию из гостиной	x: положение x y: -150
4	< -150	В гостиную из студии	x: положение x y: 150
3	< -150	В кабинет из гостиной	x: положение x y: 150
5	> 150	В гостиную из кабинета	x: положение x y: -150

Другие источники информации

1. Евгений Патаракин. Учимся готовить в Скретч. Версия 2.0
2. В.Г. Рындак, В.О. Дженжер, Л.В. Денисова. Проектная деятельность школьника в среде программирования Scratch. Учебно-методическое пособие. Оренбург — 2009.

Введение в Scratch

Цикл уроков по программированию для детей

(версия 1)

Шапошникова Светлана Вячеславовна

Лаборатория юного линуксоида - http://younglinux.info

2011

Подготовлено в OpenOffice.org Writer