Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, физики, информатики и технологий
Кафедра информатики, информационных технологий
и методики обучения информатики

# Методика обучения школьников созданию 3D игр с использованием визуального языка программирования Kodu

*Выпускная квалификационная работа*
*бакалавра по направлению подготовки*
*44.03.01 – Педагогическое образование*
*Профиль «Информатика»*

Квалификационная работа
допущена к защите

«____»_____ 2020 г.

Зав. кафедрой _____

Исполнитель: студент группы ИНФ-1501z
Института математики, физики,
информатики и технологий
Козлова Юлия Вячеславовна

_____

Руководитель: к.п.н., доцент
кафедры ИИТ и МОИ
Рожина Ирина Венокентьевна

_____

Екатеринбург – 2020

# Оглавление

## Введение

Актуальность исследования. В сочетании с интенсивным использованием новых, постоянно меняющихся технологий, быстрое развитие науки и техники затрагивает различные области деятельности человека, что, конечно, предъявляет ряд требований к людям в XXI веке. Профессиональные знания могут быть получены в соответствующих областях техники и технологий, и благодаря естественнонаучному образованию может быть создана определенная культура научной мысли.

Информатика, относительно молодая научная отрасль. Повышенное внимание к информатике связано с быстрым ростом объёма человеческих знаний, которые часто называют «информационным взрывом».

Информатика задействована почти во всех науках, помогая визуализировать процесс, недоступный человеческому глазу, моделируя опасные ситуации (опасные для жизни человека) или просто создавая модели их применения в жизни, автоматизируя работу машин. Поэтому информатика чрезвычайно полезна в современном мире и играет ведущую роль в образовательных учреждениях.

На данном этапе обучения цель состоит в том, чтобы создать целостное развитие творческой, активной личности, умения самостоятельно приобретать и применять знания. В школах, не специализирующихся на программировании, ученикам трудно изучать языки программирования и при выборе высшего учебного заведения, они пренебрегают профессиями, связанными с ИКТ.

Современная молодежь серьёзно интересуется игровой индустрией, дизайном, моделированием, анимацией. Этим интересом можно воспользоваться, чтобы подтолкнуть их к изучению информатики и, в будущем, сосредоточить внимание на профессиях, связанных с информационно-коммуникационными технологиями и программированием.

Практический опыт показывает, что существуют динамические компьютерные игры (например, игры с большим количеством движения,

действий, изменения объектов и их свойств с течением времени), которые учащиеся могут разрабатывать в процессе обучения. Разработка динамических игр, несложная с точки зрения программирования, может значительно повысить мотивацию к обучению, помочь преодолеть познавательные трудности и ускорить интеллектуальное развитие школьников.

Объект исследования: процесс обучения информатике.

Предмет исследования: обучение визуальному программированию на основе создания учащимися динамических компьютерных игр на базе визуального языка программирования Kodu.

Цель исследования состоит в разработке элементов методики обучения визуальному программированию учащихся на основе создания компьютерной игры.

Для достижения поставленной цели предполагается решение следующих задач:

1. Изучить и проанализировать теоретические и методические основы обучения школьников программированию;
2. Рассмотреть программные средства для обучения визуальному программированию;
3. Систематизировать основные типы компьютерных игр и сред их разработки для обучения программированию;
4. Проанализировать среду для разработки технологии создания игр для обучения учащихся;
5. Разработать цикл занятий обучения программированию на основе создания динамической компьютерной игры.

Для решения поставленных задач применялись следующие методы исследования:

- теоретические: системный анализ отечественной и зарубежной психолого-педагогической, научно-методической литературы по педагогике

и информатике; анализ существующих подходов к обучению информатике и программированию.

- эмпирические: обобщение опыта преподавания информатики в старшей школе; анализ содержания программ и учебно-методических комплексов по вопросам обучения программированию.

Выпускная квалификационная работа состоит из введения, двух глав, заключения и списка использованных источников.

## Глава 1. Методические особенности изучения основ алгоритмизации и программирования

Базовый курс по информатике (7-9 классы) обеспечивает минимальный уровень обязательного общего образования, который даёт школьникам оптимальные знания. Он направлен на то, чтобы предоставить школьникам методы и инструменты информационных технологий для решения задач, для формирования и развития навыков сознательного и рационального использования компьютеров для обучения, а затем и для профессиональной деятельности. Изучение базового курса – это понимание единства процессов получения, модификации, передачи и хранения информации в природе, обществе и технике.

Представляется, что содержание базового курса может объединить все три основных, на данный момент, направления преподавания информатики в школе сегодня и отражает наиболее важные аспекты его образовательной значимости:

- аспект мировоззрения, связанный с развитием представлений из метода системного анализа для анализа окружающего мира, о роли информации в управлении, общих закономерностях об информационных процессах в различных типах систем;
- «пользовательский» аспект, связанный с развитием компьютерных навыков, подготовкой школьников к практическим занятиям, связанным с широким использованием информационных технологий;
- алгоритмический аспект, который сейчас наиболее тесно связан с развитием мышления школьников.

Основное содержание курса включает в себя следующие группы вопросов:

- вопросы понимания природы информационных процессов, базы знаний о различных системах управления процессами; вопросы о представлении передачи информации, канале связи, объеме информации;
- способы подачи информации;

- методы и инструменты для официального описания действий исполнителя (условно «алгоритмическая линия»);
- вопросы по выбору исполнителя для решения проблемы, анализ его характеристик; его возможности и эффективность для решения данной задачи;
- вопросы о методе рендеринга, моделирования реальных объектов и явлений для компьютерного изучения, проведения компьютерного эксперимента;
- компьютерные шаги по устранению неполадок, использование различных типов программного обеспечения для решения задач, идея современных компьютерных информационных технологий, основанная на его использовании.

Алгоритмическая линия содержит обязательный минимум учебного материала для школьного курса, который должен быть усвоен полностью.

Изучение материалов учебного курса обеспечивает школьникам возможность:

- понимать (на основе анализа примеров) значение термина «алгоритм», знать свойства алгоритмов, понимать возможности автоматизации человеческой деятельности при применении алгоритмов;
- владеть основными алгоритмическими конструкциями (цикл, ветвление, процедура), применять алгоритмические конструкции для создания алгоритмов для решения образовательных задач;
- получить представление о «библиотеке алгоритмов», узнать, как использовать библиотеку для создания более сложных алгоритмов;
- получить представление об одном из языков программирования (или учебном алгоритмическом языке), использовать этот язык для написания алгоритмов для решения простых задач.

Образовательный стандарт формулирует основные требования к уровню подготовки учащихся.

Учащиеся должны:

- понимать суть воспроизведения алгоритма, знать его основные особенности, иллюстрировать его конкретными примерами алгоритмов;
- понимать возможности автоматизации человеческой деятельности при применении алгоритмов;
- знать базовые алгоритмические конструкции и уметь использовать их при создании алгоритмов;
- определять способность применения исполнителя использовать систему для решения конкретной проблемы в соответствии с его системой команд, разработать и выполнить алгоритм на компьютере для учебного исполнителя;
- чтобы решить простую задачу, написать алгоритм на естественном языке (или языке программирования).

### 1.1. Теория и методика формирования основ алгоритмического мышления в процессе обучения информатике

Согласно федеральному стандарту начального общего образования, важными задачами математического образования являются [4]:

- формирование основ логического и алгоритмического мышления у учащихся; умение регистрировать и выполнять алгоритмы;
- умение действовать по алгоритму и строить простые алгоритмы;
- исследовать, узнавать и представлять геометрические фигуры;
- работать со схемами, представлять, анализировать и интерпретировать данные.

В то же время, когда преподается раннее программирование, возникает другая проблема: учащиеся начальных классов не могут запоминать сложные команды, длинные письменные коды, которые, как правило, написаны на иностранном языке, ведь они только начинают учиться. Чтобы разрешить это противоречие, нам нужен язык программирования, близкий к мышлению детей, содержащий команды для работы с интересными и понятными объектами, но в то же время обеспечивающий прочную основу для изучения других языков программирования.

В 80-х годах прошлого века один из основателей теории искусственного интеллекта, создатели языка логотипов Сеймур Пейперт и Алан Кей [12], указывали, что это означает, что существенное изменение в мышлении должно быть доступно для ребенка как можно скорее.

С осени 2014 года основные принципы программирования начали изучаться в школах Соединенного Королевства. Учащиеся начальной школы в британских школах используют такие программы, как Scratch, Kodu, MIT Logo, чтобы научиться создавать простые блочные программы, и в одиннадцать лет учащиеся должны понимать основные алгоритмические структуры и использовать их для создания учебных программ. Финский проект Koodi 2016, учебный план нескольких стран, таких как Южная Корея, Эстония, Франция и Австралия, также включает обучение детей основам программирования из начальной школы.

Следует отметить, что тенденция раннего обучения школьному программированию поддерживается многими ведущими компаниями в области информационных технологий, которые предоставляют доступные инструменты программирования. Большое количество пользователей ресурсов, таких как Scratch и AppInventor, Codecademy, Code.org и другие в MIT, демонстрируют растущий интерес современных людей к знаниям и пониманию искусства программирования.

В таких средах специфические «функциональные органы мозга» формируются во время тренировочного программирования. И очень важно, чтобы эти «органы» формировались в общении и деятельности субъекта ребенка. Среда программирования – представляет переходные объекты, который служат метафорами, с помощью которых учащиеся превращают опыт телесных манипуляций с вещами (поворот направо, шаг вперед и другие) в понятийные обобщения и абстракции, что важно в младшем школьном возрасте, когда умственная деятельность не отделена от моторной деятельности субъекта.

Рассмотрим некоторые элементы методики обучения для учащихся младшего школьного возраста в среде Code.org [13]. При обучении информатике в начальной школе следует иметь в виду, что ученик не может проводить более 15 минут в классе перед компьютером. В результате урок должен быть разделен на два взаимосвязанных этапа: на первом этапе ученики изучают новый материал о теоретической части информатики (например, кодирование информации), а на втором этапе они работают на компьютерах.

Следующие характеристики среды Code.org можно выделить для формирования алгоритмического мышления у учащихся начальной школы.

Первая важная особенность – способ, которым изучают игру. Работа учеников в среде – это онлайн-игра, в которой дети изучают основы программирования. Во время игры дети управляют зомби, пчелой, художником или фермером, которые перемещаются по игровому полю и выполняют задания. Для этого игрок должен разработать цепочку команд, а затем выполнить их в процессе исполнения. Прежде чем дать подсказки на каждом уровне, все задачи имеют визуальную и звуковую окраску. Эта характеристика среды призвана решить главную проблему в обучении программированию: мотивацию освоить новое и сложное поле.

Другой характеристикой среды является двойное представление алгоритма в виде языка визуальных блоков и в JavaScript. Программирование исключает написание текста и выполняется простым перетаскиванием элементов из палитры. Чтобы программа работала, просто напишите несколько строк кода, которые должны быть выполнены после нажатия кнопки «Выполнить». Игровая визуализация и интерактивное выполнение позволяют ученику видеть исполнение этапов алгоритма, помогают ему анализировать и корректировать алгоритм.

Третья характеристика среды – последовательность и согласованность в области возможностей языка программирования. Этот сайт содержит несколько курсов, предназначенных для разных возрастных групп: от

изучения простого оборудования для четырехлетних детей до работы с циклами, разнообразных процедур и функций, предназначенных для учащихся от 16 до 18 лет и старше.

Опыт иностранных коллег показывает, что визуальная составляющая очень важна для учеников младшего возраста. Среда программирования должна быть красочна и функциональна, это пробуждает интерес и побуждает школьников к решению различных задач.

Рассмотрим изучение программирования и формирование алгоритмического мышления в рамках использования компьютерных игр для образовательных целей.

За последние три десятилетия многие зарубежные исследователи изучали, как естественный энтузиазм учащихся по поводу цифровых игр (включая компьютерные) может быть использован для достижения образовательных целей. Компьютерные игры являются предметом серьезных исследований в области психологии, педагогики, информатики и социологии. Все эти науки пытаются связать компьютерные игры с обучением. Основным фактором, влияющим на эффективность использования компьютерных игр для решения образовательных задач, является мотивация.

Необходимо остановиться на темах мотивации учащегося и определить виды мотивации, которые могут участвовать в обучении программированию путем создания компьютерных игр. Мотивация – это комплекс факторов, которые направляют и стимулируют поведение человека. Термин «мотивация» может означать две вещи:

- система мотивов определенного человека;
- система действий по активизации мотивов определенного человека.

Следует упомянуть психологические теории, которые формируют теоретическую основу мотивации в компьютерных играх. Эмпирические источники внутренней мотивации изучаются в теории когнитивной оценки, теории потока и теории когнитивного любопытства в аспекте ассимиляции и

аккомодации Пиаже. Наряду с мотивационными исследованиями, существуют также исследования возможных барьеров для эффективного использования игр в обучении, например, педагогические теории рефлексии и передачи знаний [1].

В системе образовательных мотивов внешние и внутренние мотивы взаимосвязаны. Внешняя мотивация хорошо описана в педагогической литературе.

В аспекте создания компьютерных игр особенно интересна концепция внутренней мотивации («intrinsic» на английском языке) Э. Деси [2]. В литературе это также известно, как «мотивация, возникающая в результате процесса» или предоперационная в теории развития Пиаже [3]. Это «желание участвовать в деятельности ради собственного блага, ради наград, содержащихся в процессе деятельности». Поведенческие теории основаны на предположении об отсутствии внутренней мотивации к обучению, но это не согласуется с тем фактом, что маленькие дети, как в дошкольных учреждениях, так и дома, постоянно изучают и манипулируют объектами, с которыми они сталкиваются. Они бросают вызов себе, чтобы быть более компетентными и знающими, просто для удовольствия. Сами дети активно участвуют в учебном процессе. Без сомнения, они мотивированы учиться изнутри. Вероятно, существует внутренняя потребность испытывать чувство личной автономии или самоопределения.

Внутренняя потребность:

- разнообразность навыков;
- чувство важности работы;
- высокая внутренняя мотивация;
- качество выполненной работы;
- высокая удовлетворенность работой.

Внешняя потребность:

- определение задачи;
- важность домашней работы;

- автономия;
- опыт ответственности за результаты работы;
- комментарии к результатам работы;
- знание результата своей работы.

В общих чертах, модель работает следующим образом: пять ключевых характеристик работы вызывают три психологических состояния, которые, в свою очередь, приводят к ряду благоприятных результатов для человека и для работы. Модель постулирует, что человек испытывает положительные эмоции тогда, когда узнает (знание результатов), что он лично (опыт ответственности) хорошо выполнил задание (важность самостоятельной работы). Позвольте нам объяснить этот тезис в аспекте обучения программированию через создание компьютерных игр [31]. Разнообразие навыков – это степень, в которой процесс требует нескольких действий, что подразумевает использование различных человеческих навыков и способностей. Дж. Хакман и Дж. Олдхем считают, что разнообразие навыков, необходимых для решения игровых задач, является основной причиной их привлекательности для людей. Если процесс требует нескольких навыков одного человека, он будет казаться более значимым, чем на самом деле. В этой части использование компьютерных игр в обучении посредством внутренней мотивации тесно связано с развитием общеобразовательных навыков [28]. Идентификация задачи: степень, в которой работа требует завершения целостной и конкретной задачи или стадии работы от начала до конца с видимым результатом. Основываясь на этой особенности, компьютерные игры очень подходят в качестве учебной задачи, если их создание четко разделено на этапы, каждый из которых имеет видимый конечный результат, осознаваемый в начале этапа выполнения. Важность задания: в нашем случае степень важности результата (финальной игры) для обучающегося. Этот показатель обычно очень высокий. Создание собственной игры с нуля является важным и значительным достижением для школьника.

Автономия – это степень, в которой работа обеспечивает реальную свободу и способность действовать по своему усмотрению при планировании и определении того, как ее достичь. Это самая проблемная особенность с точки зрения методологии. В существующих системах обучения программированию с помощью разработки игр автономность обычно строго ограничена методологией или средой разработки. Чтобы активировать автономность, нужно бороться за максимальную свободу выбора сюжета создаваемой игры и минимальные ограничения со стороны программного обеспечения.

Цель курса: научить школьников структурному и объектно-ориентированному программированию в соответствии с Федеральным государственным образовательным стандартом, основанным на задачах кросс-секционного проектирования (разработка динамических игр).

Определение целей учителя основывается на изучении целей учащихся и их сочетании с общей целью обучения:

- создание собственной игры на языке программирования невозможно без глубокой разработки; в то же время требования стандарта в отношении компетенций в области алгоритмов и программирования будут превышены;
- необходимость демонстрации результатов требует организации полного цикла обучения при разработке игры, что гарантирует результат (законченную игру) на том или ином уровне, в зависимости от уровня подготовки учащихся;

Удовольствие от процесса создания игры является основой для того, чтобы большее количество времени уделялось самостоятельной работе и, следовательно, это поможет устранить непродуктивные способы использования компьютерных ИКТ. Планируемый внешний результат: оригинальные динамические игры, разработанные учениками самостоятельно или в занятиях. Достижение этого результата необходимо и достаточно для достижения целей учащегося.

Результаты обучения подразделяются на три группы в соответствии с требованиями Федерального государственного образовательного стандарта.

Результаты компьютерного обучения включают в себя:

- навыки в алгоритмическом мышлении и понимание необходимости формального описания алгоритмов;
- знание универсального языка программирования высокого уровня (опционально), понимание основных типов данных и структур данных; умение использовать базовые структуры управления;
- владение навыками и опытом разработки программ в выбранной среде программирования, включая тестирование и отладку программ; элементарные навыки для формализации прикладных задач и документирования программ;
- умение работать с библиотеками программного обеспечения.

Однако возможность использования базовых структур управления, указанных в стандарте, не сопровождается перечнем, но, на основе анализа учебной литературы, примерной программы по предмету и опыта обучения информатике его можно сформулировать. Основные структуры управления:

- условное ветвление;
- процедуры/функции, работа с циклами;
- рекурсивные функции;
- инструменты матричной обработки.

Перейдём ближе к нашей теме. Образовательные средства обучения традиционно включают элементы игры в качестве дополнительного стимула в процессе приобретения знаний. Затем, в 90-х годах, термин «edutainment» появился в английской педагогике, который состоял из слов education («образование», английский) и entertainment («развлечение», английский), и так называемая концепция «брокколи в шоколаде», предназначенная для упрощения восприятия сложных проблем, «воздействия» на их игровые элементы [10].

Такой подход реализует две основные задачи:

1. Позволяет поддерживать интерес и вовлечённость учащихся;

2. Игра даёт возможности для создания непосредственного опыта. Тренировка и освоение навыков происходит прямо в процессе обучения.

Современное слово «школа» происходит от древнегреческого «досуг, отдых, время, свободное от обязанностей и задач». Можно сказать, что эдьютейнмент, совмещающий образовательный и развлекательный компоненты, возвращает к изначальному пониманию процесса обучения, которое было заложено еще древними греками.

Через игровые форматы взрослые и дети вовлекаются в процесс. Они получают положительные эмоции и яркие впечатления, благодаря чему сохраняется интерес к продолжению участия. В итоге они приобретают новый опыт с возможностью грамотно отрефлексировать его, что приводит к осознанному освоению навыка.

Однако важно понимать, что эдьютейнмент, несмотря на всю свою привлекательность, не является альтернативой обычному образованию. На данный момент – это направление не может заменить традиционные формы обучения, а скорее становится хорошим дополнением к ним.

### 1.2. Геймификация в образовании

Геймификация в образовании – это использование игровых элементов в процессе обучения. Не путайте это с игрой: цель геймификации – это прежде всего достижение результатов [9]. Для учеников начальной школы это похоже на конфету, где серединка горькая, а оболочка сладкая. Преподавателю не нужно долго и монотонно объяснять материал, гораздо удобнее передать его в игровой форме. Тогда «конфета» в виде таблицы алгебры логики быстро усваивается, а сам процесс обучения становится увлекательным.

Игровые элементы используются на протяжении всего периода обучения, не только в школе, но и в университете. Оценки, рейтинг, успеваемость – все это скрытая геймификация, которая настолько закрепилась в системе образования, что воспринимается как нечто

естественное. Даже итоговые тесты и переход в следующий класс аналогичны битве с последним боссом и своеобразному повышению уровня.

Геймификация в образовании сама по себе не новое явление, а скорее новая концепция и определение. В упрощённом виде она существовала и в советской школе: Константин Ушинский рекомендовал включать в монотонную учёбу игровые упражнения, например, викторины. Этот тип обучения лучше работает потому что в процессе игры задействована эмоциональная составляющая, и, именно благодаря этому, материал лучше усваивается.

Олег Федоров, ведущий научный сотрудник института экономического образования НИУ ВШЭ, считает, что образование всегда было тесно связано с играми: настольными, деловыми, ролевыми [42]. Но теперь цифровые возможности делают игры более привлекательными и продуктивными с точки зрения обучения. Их можно использовать для моделирования, создавать симуляции и для прогнозирования. К занятиям можно подключить детей дистанционно, из любой точки мира.

Основная причина того, что сейчас всё больше внимания уделяется геймификации связана с новыми техническими возможностями. Она становится не дополнением к уроку, а его неотъемлемой частью.

Традиционная школа создает искусственную среду, которая противоречит интересам детей и ограничивает их восприятие стенами классных комнат. Получается парадокс: детей приводят в место, где они могут получать знания и находить ответы на многие «почему» и «зачем», но они убивают желание хотя бы задать вопросы. Геймификация этот парадокс устраняет и делает обучение интерактивным. Она вовлекает всех людей в процесс, потому что использует следующие элементы:

1. Динамика, создание легенды. Это может быть история с неожиданными сюжетными поворотами, где исход событий зависит от решений ученика. У них должно быть чувство сопричастности, вклада в общую цель.

2. Мотивация. Постепенное изменение и усложнение целей по мере того, как ученики приобретают новые навыки и компетенции. Это помогает сохранить внимание и вовлечённость учеников.

3. Взаимодействие пользователей. Постоянные отзывы от учителя или одноклассников. Это позволяет получить оценку своих действий и скорректировать их при возникновении ошибок.

Основная проблема геймификации – отсутствие времени на уроке. Теоретически, это очень простая, но эффективная методика, требующая от учителя лишь небольшой домашней подготовки. Однако на практике оказалось, что все гораздо сложнее: урок длится всего 45 минут, из которых от 10 до 15 посвящаются организационным моментам, готовят детей к пониманию нового материала и контролируют выполнение домашней работы. Как, ещё и с помощью игрового метода, можно объяснить новую тему в оставшиеся полчаса? А ещё нужно учесть, что не все в классе успевают за учителем или другими учениками. В этой ситуации учителям больше не до геймификации, успеть бы выдать программу. Потому, элементы геймификации должны вводиться элективно, что мы и предлагаем, с учётом построенной методики для обучения визуальному программированию с помощью среды Kodu Game Lab.

Обучение с помощью языка Kodu является ярким примером геймификации в обучении. Поскольку программа имеет «детский» интерфейс, многие могут не воспринимать её всерьёз, но визуальный язык Kodu не так, прост, как кажется с первого взгляда. Рассмотрим подробнее функции и возможности визуальной среды.

### 1.3. Описание и особенности технологии визуального программирования в среде Kodu Game Lab

Kodu Game Lab – это визуальный конструктор для разработки игровых приложений для персонального компьютера, а также для игровой консоли XBOX 360.

Kodu Game Lab не направлен на изучение какого-либо промышленного языка программирования, а обучает детей основам алгоритмизации и взаимодействию с компьютерами. Проект предоставляет инструментарий для проектирования компьютерных игр с помощью визуального интерфейса. Ученик может создать свое игровое поле, добавить туда персонажей и задать для них логику поведения с помощью визуального языка. Этот проект является инициативой компании Microsoft. Kodu – универсальное приспособление для того, чтобы творить. Оно, подкупая своим приветливым дизайном и понятным интерфейсом, мотивирует к планированию и постройке самых разных вселенных.

Присутствуют такие возможности, как выбор объектов для размещения и окружающей среды, в которой персонажи и объекты будут «жить», программирование объектов (то есть их поведения при определённых условиях), установка отношений среди любых объектов в создаваемом мире. Еще Kodu допускает возможность управлять настройками создаваемого мира, графической составляющей и даже окружающими цветами.

Графический интерфейс позволяет пользователю с легкостью манипулировать исполнителями на дисплее, редактировать поведение игрока, управлять визуализацией, звуками и сценарием. Манипулируя исполнителями в их виртуальных мирах, учащиеся получат первоначальный опыт работы с базовыми алгоритмическими структурами.

Использование интегрированной среды разработки, выполненной в качестве визуального редактора Kodu, способствует быстрому развитию следующих моментов:

- формирование структурного мышления;
- формирование знаний о главных понятиях в изучении: алгоритм, модель, их свойства;
- прогресс в развитии алгоритмического мышления;
- прогресс в развитии навыков составления алгоритмов под определенного исполнителя;

- прогресс в развитии в области коммуникации общества.

Следуя из выше сказанных пунктов, проектирование вселенных и время создания поведения для «жителей вселенной», т.е. программирование, отлично помогают сформировать и укрепить ряд учебных действий:

- постановка и формулировка цели;
- мотивация учебной деятельности;
- самостоятельная разработка плана по достижению поставленной цели, его оптимизация для самого результативного решения необходимых задач;
- сопоставление текущих действий с необходимым результатом;
- наблюдение и анализ проделанного на пути к желаемому конечному результату – цели;
- нахождение инструментов и действий для выполнения задач, ограничиваясь данными возможностями, опираясь на установленные требования;
- оптимизация своих действий, опираясь на текущую ситуацию процесса выполнения задач;
- оценка корректности процесса достижения цели;
- организация коммуникационных связей как с учителем, так и с одноклассниками/одногруппниками;
- правильное формулирование, аргументация и отстаивание своей позиции.

Тот социальный опыт и способности, которые учащийся получает в условиях работы со средой Kodu, имеет значимое практическое применение.

Если говорить о главных навыках по предмету, которые постепенно формируются и закрепляются у ученика, то можно выделить формирование в себе следующих умений:

- применение терминологии, а именно: «программа» и «алгоритм»;
- знание отличительных черт при использовании терминологии как в области информатики, так и в повседневной жизни;

- построение простых линейных программ-алгоритмов для манипулирования исполнителями;
- построение программ-алгоритмов, созданных с применением условных операторов, то есть конструкций ветвления, а также циклов;
- построение и исполнение программ-алгоритмов, необходимых для выполнения поставленной несложной алгоритмической задачи;
- свободное пользование готовым программным обеспечением и интернет-ресурсами в указанной сфере, свободное изучение документации к программному обеспечению и интернет-ресурсам.

Преподаватель обретает по-своему исключительную среду разработки, в которой ученик с легкостью сможет изучить и понять все базовые понятия области информатики. Ученик в приятной и доступной для него форме игры, постепенно познает и усвоит смысл ООП (объектно-ориентированного программирования), благодаря возможности работы с такими понятиями, как класс, родитель, объект. Все эти понятия являются стартовыми на пути изучения более сложных в области программирования: наследование, инкапсуляция, полиморфизм.

Подводя итог всего выше сказанного, вывод такой, что описанные пункты по реализации педагогических возможностей графической среды разработки Kodu, позволяют использовать Kodu в качестве инструмента образовательной программы в учебном процессе, дают основные условия для успешного формирования у учеников стремления к развитию в области информационных технологий, а может и к овладению профессии в области информатики и программирования. Благодаря применению среды Kodu в обучении совершенствуется игровая форма представления знаний. Возможность самостоятельно строить алгоритмы разной сложности для своих игр активизирует познавательную и учебную деятельность младших школьников на уроках информатики и ИКТ.

Среда предоставляет возможность показать учащимся, что программирование – это вовсе не скучное занятие, оно может быть

интересным и увлекательным [39]. И любой учащийся сможет проявить творческие навыки в процессе создания своей игры. Все, что требуется для разработки игры – создать игровой мир, в котором будут обитать добавленные пользователем персонажи, и взаимодействовать с внешним миром по установленным пользователем правилам, конечно, не забывая про законы физики. Причем составляются эти правила весьма незамысловатым способом: в среде используется конструкция «когда X делать Y». А методами и полями в этой конструкции служат примитивные «карточки» с действиями или состояниями объекта.

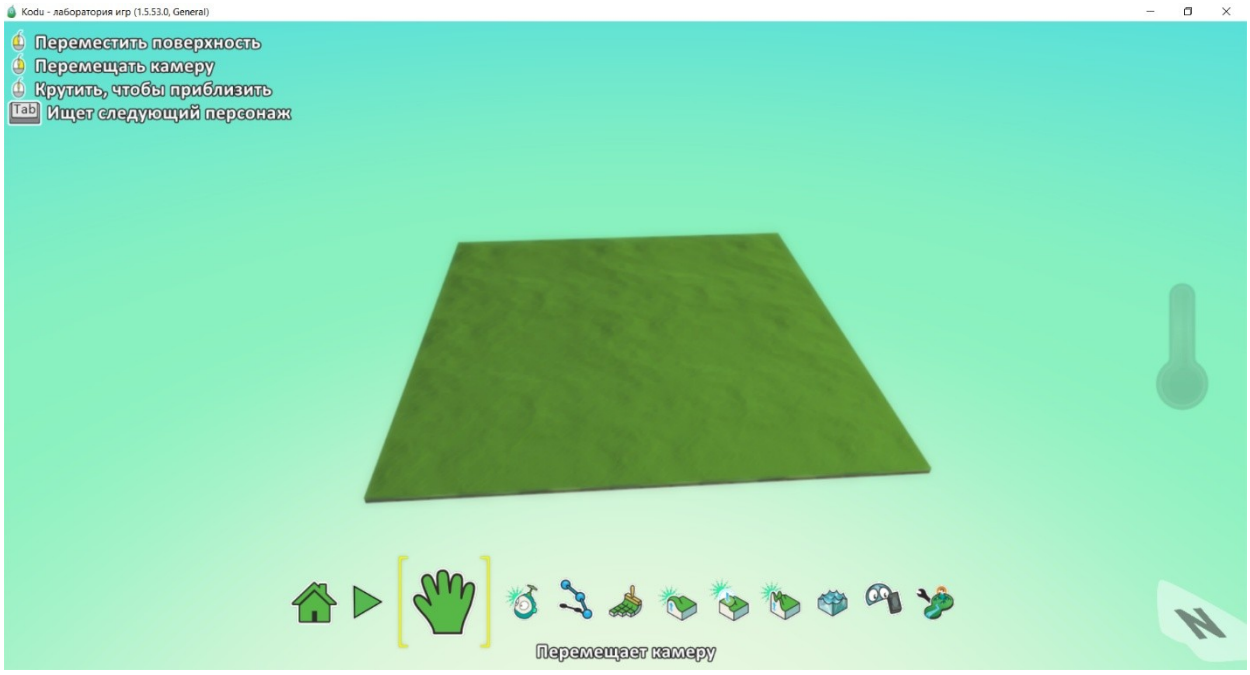



Рисунок 1 – Среда Kodu Game Lab

При первом запуске Kodu Game Lab можно выбрать один из уже существующих демонстрационных миров, в которые входят уровни с начальным обучением программирования в данной среде, а также создать игру, начав с пустого мира (Рисунок 1). Рассмотрим кнопки внизу экрана. С их помощью и происходит создание новых объектов и изменение освещения и ландшафта поля. Первый значок в виде домика, позволяет нам выйти в главное меню (Рисунок 2).

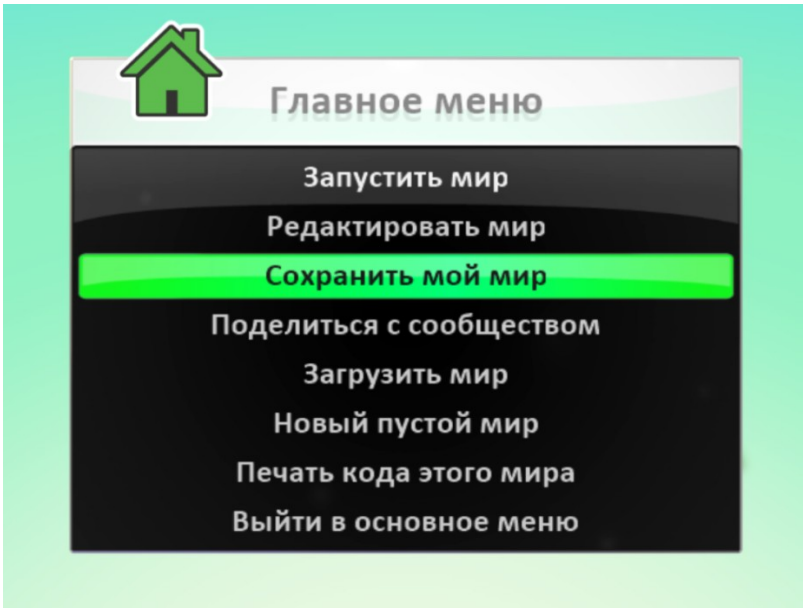



Рисунок 2 – Кнопка «Главное меню»

Вторая кнопка в виде треугольника запускает игру. Третья отвечает за перемещение камеры. Значок с персонажем kodu отвечает за создание и редакцию всех персонажей и объектов, которые представлены в визуальной среде программирования Kodu Game Lab (Рисунок 3).

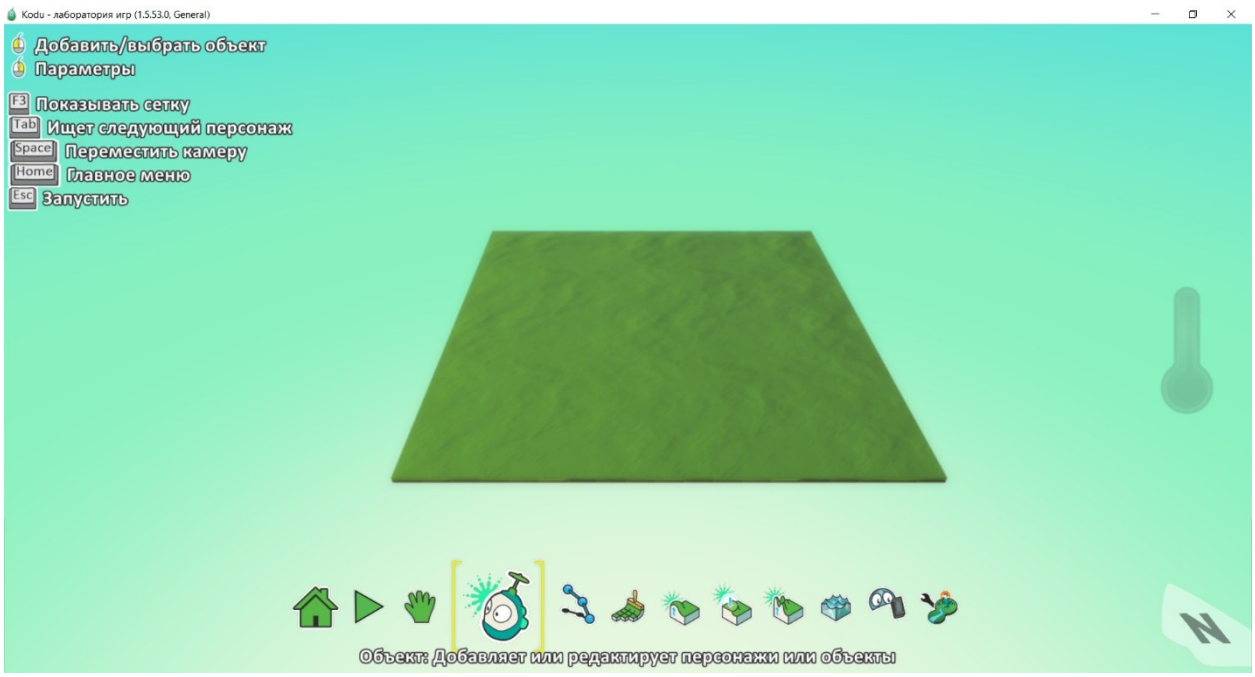



Рисунок 3 – Кнопка «Объект»

При нажатии на левую клавишу мыши выпадает меню с возможными вариантами создания того или иного персонажа/объекта (Рисунок 4). Острый «лепесток» обозначает категорию объектов. При нажатии на него

раскрывается ещё одно подменю, в котором мы можем увидеть все возможные варианты. Возьмём, например, лепесток с обозначением замка, ракеты и монеты. Кликнув на него левой клавишей мыши увидим «лепестки» с визуальным обозначением каждого из объектов. При наведении на любой из них мышкой появляется всплывающее окно с кратким описанием объекта (Рисунок 5).


Рисунок 4 – Выпадающее меню кнопки «Объект»


Рисунок 5 – Подменю кнопки «Объект»

В левом углу экрана есть все необходимые подсказки для взаимодействия со средой. Подсказки меняются в зависимости от выполняемых действий в рамках данной функции. В данном случае выбрана функция создания объекта/персонажа и в углу показаны возможные действия в пределах этой функции.

Переходим к следующей функции меню, это кнопка «Путь» (Рисунок 6).

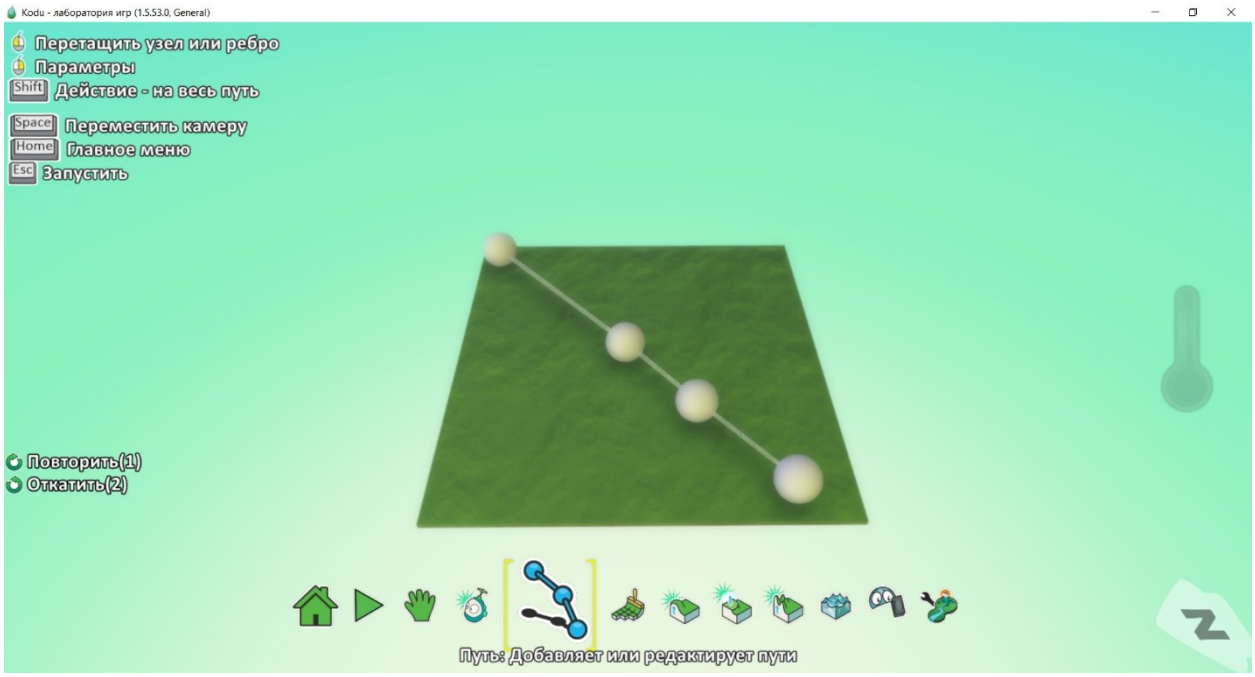



Рисунок 6 – Кнопка «Путь»

Она отвечает за создание и редактирование путей. С её помощью мы можем наметить, например, траекторию движения персонажа. По нажатию правой клавиши мыши на узел или ребро появляется меню с функциями, которые позволяют рёбрам менять направление движения, которое может производить персонаж, а узлам менять высоту, тип, а также добавлять новые узлы из того, по которому было произведено нажатие (Рисунок 7).

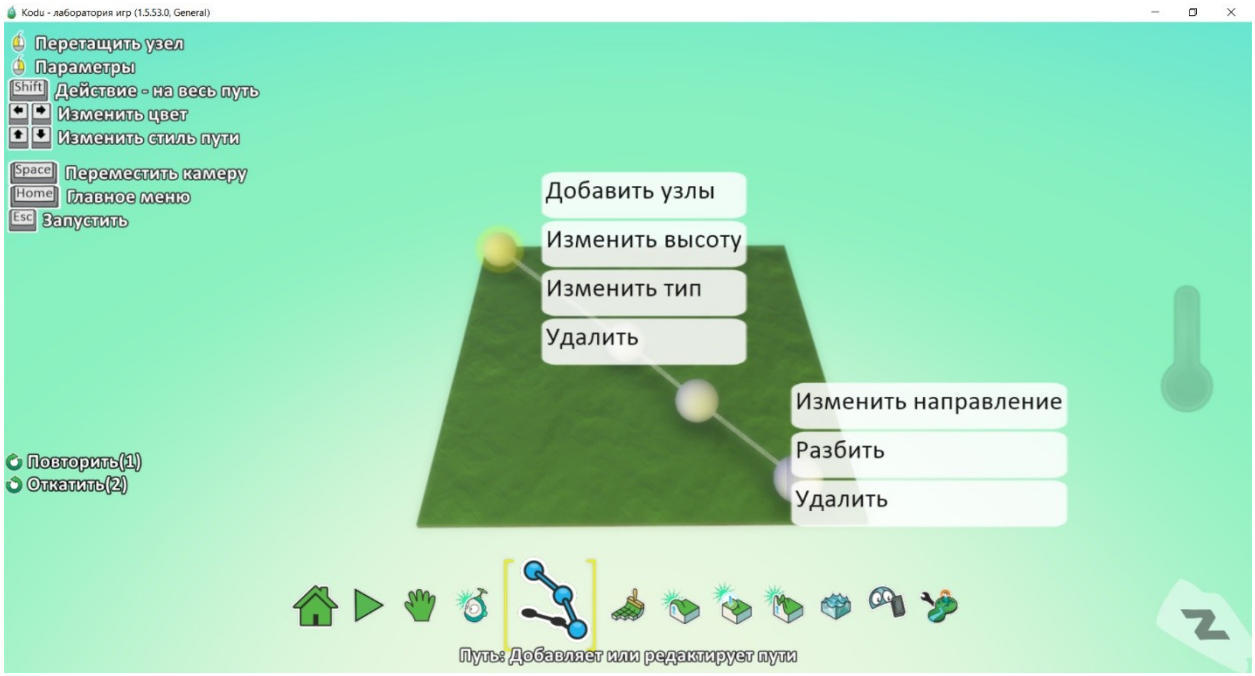



Рисунок 7 – Меню для узла и ребра пути

Следующие четыре кнопки «Кисть для земли», «Вверх/Вниз», «Сглаживание» и «Неровности» изменяют параметры ландшафта: меняют структуру земли, создают холмы и долины, делают землю более гладкой или ровной и создают пики или холмистую местность, соответственно (Рисунок 8).

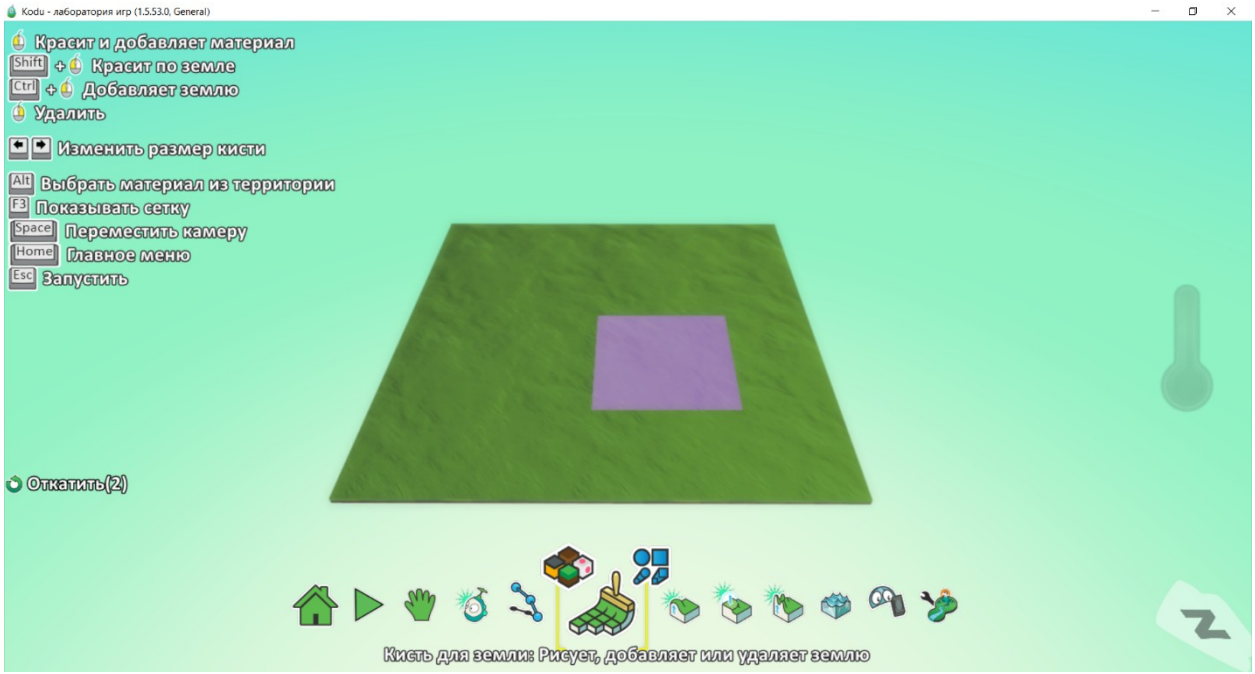



Рисунок 8 – Кнопка «Кисть для земли»

Кнопка «Вода» создаёт воду в низинах. Если поместить её просто на поле, она растечётся. Значит, если мы хотим разместить воду, нам нужен рельеф (Рисунок 9). Предпоследняя функция меню, кнопка «Удалить». Она работает как кисть. С её помощью можно удалять объекты, которые находятся на определённой площади, круглой или квадратной, в зависимости от формы кисти. Последняя кнопка меню параметров мира. Здесь отражены все функции мира: управления цветом неба и освещения, ветер, с чего будет начинаться игра при запуске, вывод счётчика на экран и выбор его цвета (при создании игры на количество очков) и многое другое.

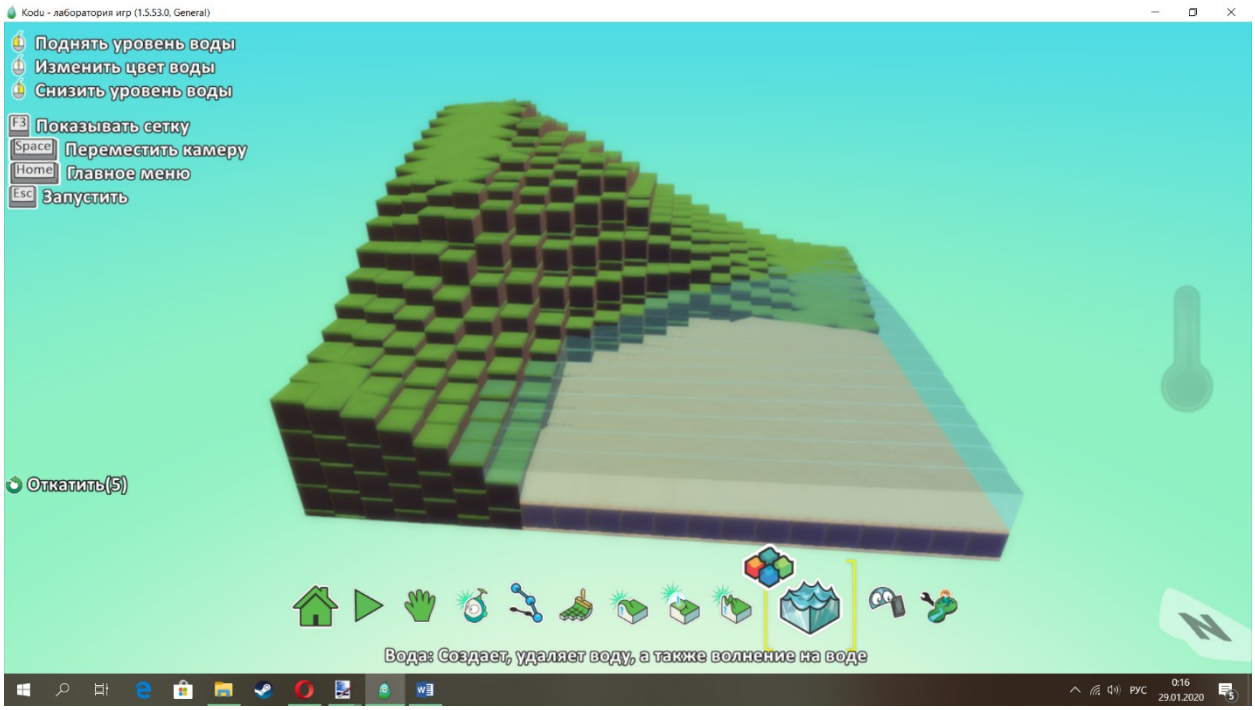



Рисунок 9 – Кнопка «Вода» и пример размещения

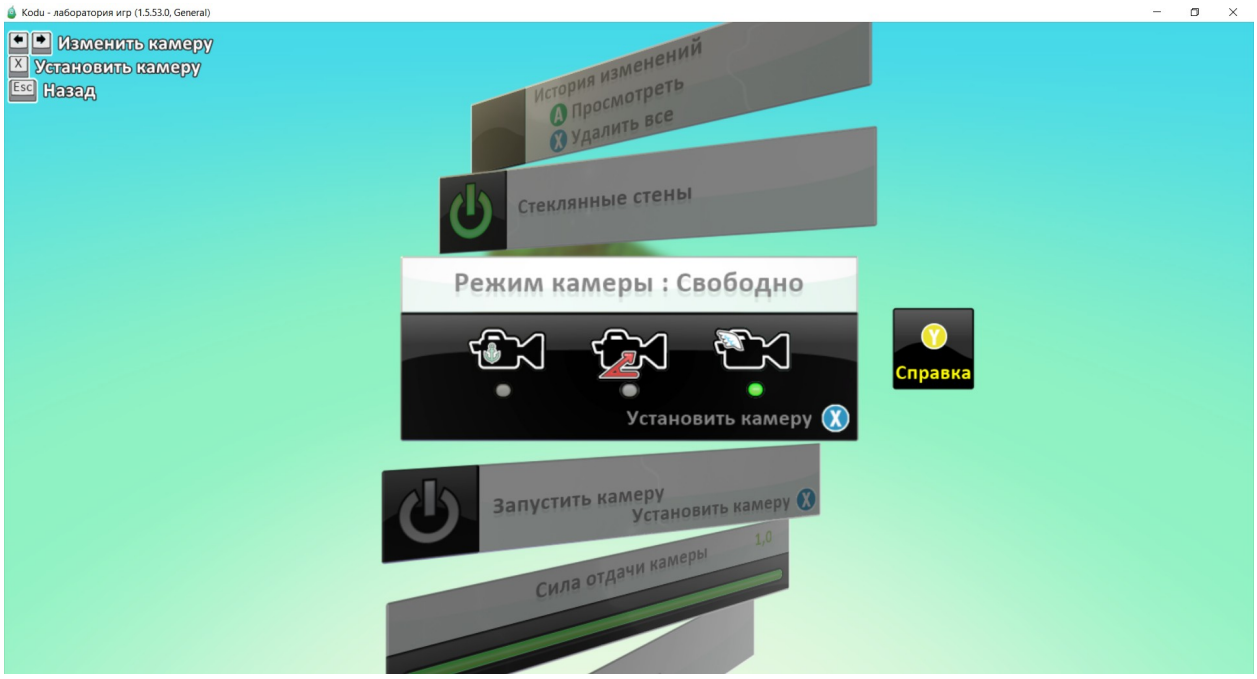



Рисунок 10 – Параметры мира

Внутри мира могут жить абсолютно разные объекты: «коду», яблоки, монеты, деревья, камни, черепахи, корабли, пушки и т.д. Изначально объекты уже имеют некое поведение. Так, например, пушка, добавленная в игровой мир, может «улыбаться», опрокидываться на спину и обратно. Также действия объекта могут быть добавлены вами – например, при нажатии на клавишу «пробел» можно заставить пушку выстрелить, а при нажатии на клавиши W, A, S, D – передвигаться и поворачиваться вокруг своей оси. Снаряд при столкновении с каким-либо объектом взрывается, нанося ущерб «здоровью» объекта, в которого он попал. Когда показатель «здоровья» у объекта достигнет нуля, последний уничтожается. Все это и многое другое уже изначально заложено в ряд функциональных возможностей. Поэтому, чтобы получить работающий вариант своей игры, надо не так уж и много – разместить объекты и наделить их хоть каким-то минимальным поведением. Перед тем, как писать правила нашим объектам и размещать их, необходимо создать и настроить ландшафт. Для этого используются соответствующие параметры, включающие большое разнообразие текстур: от гор до равнин, от водоемов до дыр в земле, в которые объекты могут упасть. Все это настраивается различными видами кисточек: квадратная, круглая, их

аналоги, позволяющие рисовать прямые линии, а также волшебная кисть, с помощью которой можно изменить состояние всей области одинакового цвета, части которой контактируют друг с другом. Огромный набор текстур предоставляет возможность каждому пользователю придумать уникальный ландшафт для будущей игры. Используя разнообразные цветные текстуры для создания воды, начинающий программист может создать виртуальный мир, в котором действие может происходить и на суше, и на воде, и под водой. В этом плане Kodu Game Lab является очень хорошей площадкой для воплощения своих творческих способностей и воображения.

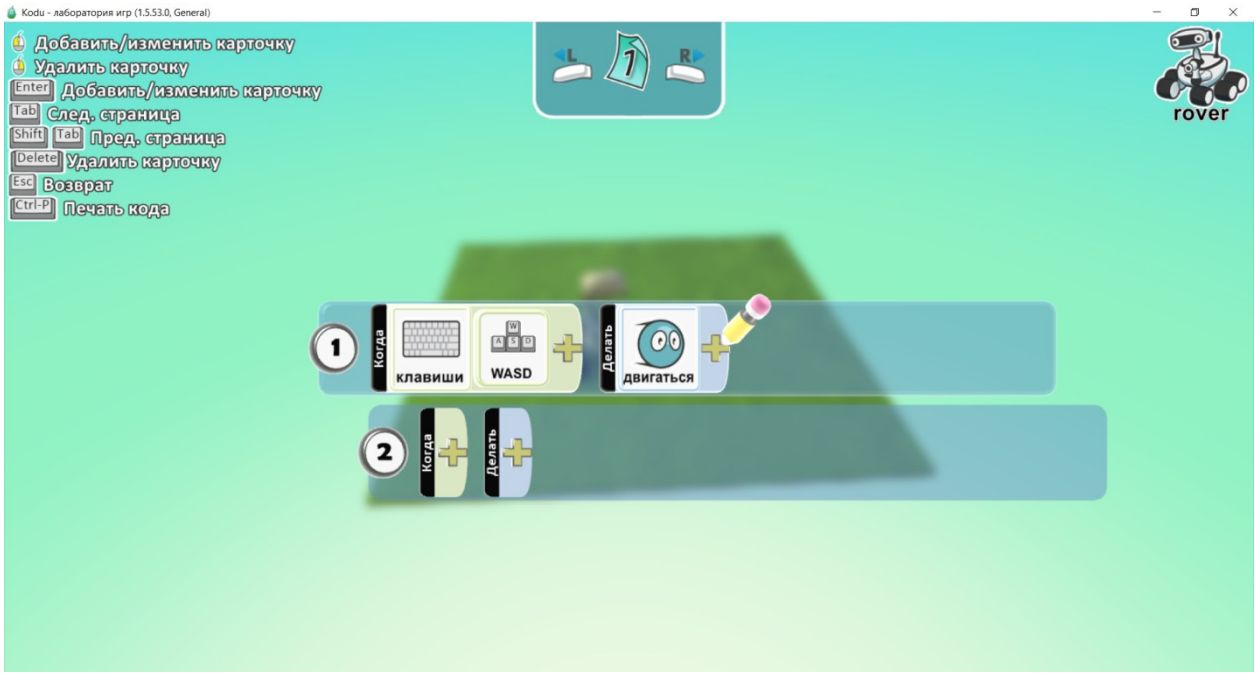



Рисунок 11 – Реализация многоуровневых условий

После создания ландшафта будущей игры, начинается этап размещения и программирования поведения объектов. Окно с программой объекта, в которой описывается его поведение, имеет целых 12 страниц – 12 состояний объекта. Kodu Game Lab позволяет создавать многоуровневые условия и действия (Рисунок 11). Для этого необходимо подвинуть строчку с поведением правее, чем та, что над ней, организовав многоуровневый порядок действий. Самое простое, с чего можно начать – научить объект ходить (Рисунок 12). И уже на этом этапе понятно, что игру можно сделать даже на несколько человек, которые будут играть на одной клавиатуре, так

как действия можно задать почти на все клавиши. Уже только эта возможность открывает огромные границы для творчества. Далее можно научить созданный объект еще какому-либо действию, например, эмоциям при каких-то определенных ситуациях. Скажем, когда он видит дерево, будет испытывать эмоцию радость (Рисунок 13).

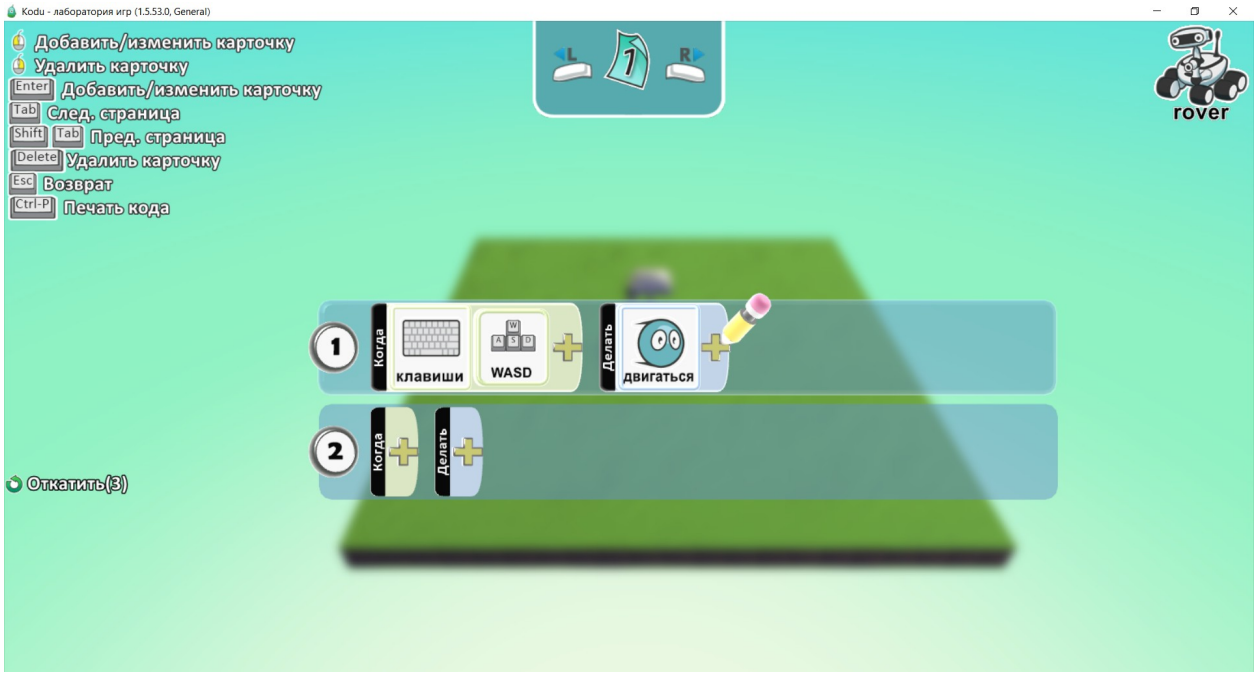



Рисунок 12 – Программа для передвижения объекта

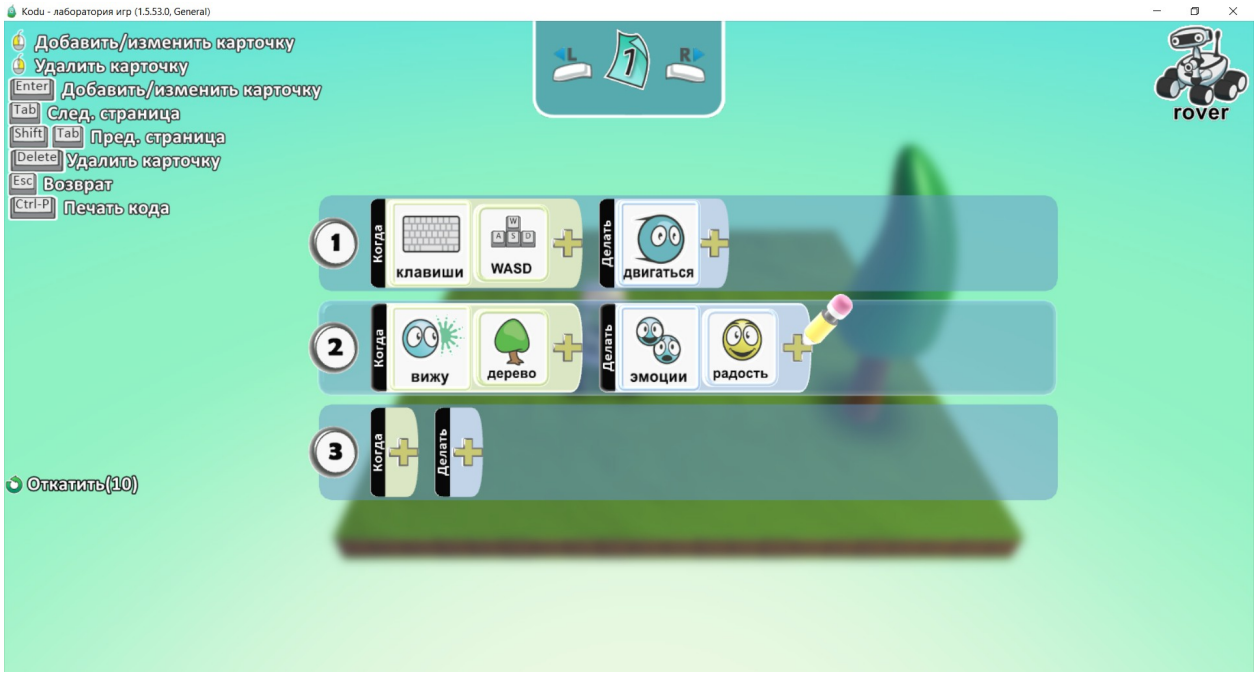



Рисунок 13 – Программа «эмоция»

Отдельно стоит отметить такой параметр у объектов, как «родитель». При его активации появляется возможность создать до ста экземпляров данного объекта с точно такой же программой поведения, с точно такими же параметрами самого объекта (скорость, урон, количество здоровья и т.д.). Это способствует уже какому-то введению в объектно-ориентированное программирование. Учащийся будет понимать, что не обязательно создавать десятки клонов, достаточно просто создать экземпляры одного объекта. Также присутствует возможность копировать программы поведения, но, к сожалению, только по одной строчке (одному поведению), либо же по одному многоуровневому условию с подчинением. При создании мира, в правой области экрана находится термометр. Данный индикатор показывает объем используемых ресурсов создаваемой игры. Как только объектов станет очень много, или карта будет чересчур большая, термометр оповестит, что выбран лимит ресурсов – ничего нового уже нельзя будет добавить, пока что-нибудь не удалить. Еще одна очень полезная возможность среды – организация переходов с уровня на уровень (переходить из одного мира в другой). При достижении игроком, например, пяти игровых очков, загрузится следующий уровень игры. Визуальная объектно-ориентированная среда Kodu Game Lab позволяет создавать не только обычные шутеры (стрелялки) от перового лица, но и многоуровневые сюжетные квесты, в которых может быть всё, что угодно. В заключении стоит отметить, что в нашем веке быстро прогрессирующих технологий никуда без навыков мышления, а среда Kodu Game Lab позволяет начать формирование алгоритмического мышления уже с раннего возраста [24].

# Глава 2. Разработка обучающего курса для школьников по созданию 3D игр с использованием визуального языка программирования KODU

## 2.1. Проектирование комплекса учебных занятий для обучения младших школьников основам алгоритмизации

Программу обучения следует начать со знакомства с возможностями среды Kodu. Для этого мы предлагаем ученикам пофантазировать и создать свой собственный мир с уникальным ландшафтом и персонажем. Это вводное занятие позволит погрузить и заинтересовать ученика вариативностью и возможностями визуальной среды Kodu Game Lab.

Среда конструирования Kodu Game Lab знакомит с логикой программирования и способами решения проблем, обходясь без сложного синтаксиса, и побуждает пользователей глубоко анализировать проблему, структурируя свое решение. Это многофункциональный инструмент для творчества, который вдохновляет пользователей создавать истории.

Комплекс занятий созданию 3D игр в данной среде имеет огромный потенциал.

Цель: пропедевтика базовых понятий программирования и получение первоначального практического опыта.

Задачи:

- развитие алгоритмического стиля мышления;
- формирование мотивации к получению образования в ИТ-сфере посредством организации продуктивной творческой деятельности и создания ситуации успеха.

Курс построен на основе практико-ориентированного подхода по принципу дидактической спирали:

- знакомство обучающихся с понятием или видом деятельности через выполнение практических заданий;
- более подробное изучение понятий или объектов с включением некоторых новых функций, свойств и т.п.;

- применение изученных понятий на практике в заданиях творческого характера.

Ожидаемые образовательные результаты:

- предметные:

- освоение основных понятий, таких как «алгоритм», «программа» и т.д.;

- соотнесение ключевых подходов визуального и объектно-ориентированного программирования;

- практические навыки создания алгоритмов управления исполнителями;

- умение создавать и выполнять программы для решения несложных алгоритмических задач;

- личностные:

- готовность и способность обучающихся к личностному самоопределению, мотивация к целенаправленной познавательной деятельности;

- метапредметные:

- умение самостоятельно планировать пути достижения целей, соотносить свои действия с планируемыми результатами, осуществлять контроль и коррекцию своей деятельности в процессе достижения результата;

- умение организовывать продуктивное сотрудничество и совместную деятельность с учителем и сверстниками.

Каждое занятие имеет определенную структуру:

1. Дидактическая беседа с учащимися на заданную тему, постановка образовательной задачи.

2. Докомпьютерное решение поставленной задачи (манипулирование с реальными объектами и дидактическими карточками).

3. Моделирование задачи с помощью среды Kodu Game Lab, которое может состоять из двух частей:

- простое программирование: содержит руководство и задания для учащихся, работая над которыми, учащиеся знакомятся с объектами и простыми понятиями программирования.
- создание и редактирование ландшафтов: при наличии времени можно использовать и вторую половину этого плана «Изменение мира» и на ее основе обсудить создание ландшафта.

Оба аспекта создания игры обычно довольно интересны учащимся, поэтому рекомендуется при наличии времени познакомить их с обеими частями плана.

4. Обсуждение полученных результатов.

В ходе изучения в занятие могут включаться дополнительные этапы [11]:

- просмотр рекомендованных видеороликов;
- опросы для контроля;
- игры и тренинги на сплочение коллектива и преодоления эмоциональных состояний;
- парная и групповая работа по программированию и моделированию миров;
- рефлексия и т.д.

Рекомендуется использовать следующие методы работы на занятиях:

- *Индивидуальная работа* используется при отработке основных навыков, а также при проведении самоконтроля (блиц-опросы);
- *Работа в парах* со сменой пар на каждом занятии для лучшего развития коммуникативных умений;
- *Работа в малых группах* подходит для разработки творческих проектов;
- *Работа с дополнительными заданиями* или подсказками (для детей с особенными образовательными потребностями и одаренных детей);
- *«Сильные помогают слабым»* применяется в группах, где навыки и темп работы обучающихся различается;

- *Игра в игру друга*;
- *Взаимооценка и др*.

| № | Описание занятия | Содержание |
|---|---|---|
| Занятие 1 | Знакомство с визуальной средой программирования Kodu;<br>Подсчёт баллов | Создание первой программы, создание игры для двух игроков;<br>Начисление баллов за действия объекта;<br>Создание ландшафтов;<br>Задание для самостоятельной работы |
| Занятие 2 | Новые возможности для перемещения объектов и персонажей. Построение пути | Повторение пройденного;<br>Создание произвольного пути движения игрового объекта;<br>Создание клонов объектов;<br>Опция Родитель;<br>Создание порожденных объектов;<br>Задание для самостоятельной работы |
| Занятие 3 | Знакомство с визуальной средой программирования Kodu: подсчёт баллов, индикатор здоровья, объект таймер | Назначение времени действия игрового объекта;<br>Начисление баллов за действия объекта;<br>Использование индикатора уровня жизни;<br>Задание для самостоятельной работы |
| Занятие 4 | Использование страниц в Kodu Game Lab. Создание уникальных историй и персонажей | Работа с несколькими страницами;<br>Написание игры с несколькими уровнями |
| Занятие 5 | Разработка и создание | Самостоятельное создание игры по |

| | | |
|---|---|---|
| | игры своими руками от начала и до конца. | предложенному плану. |

**2.2. Содержание комплекса учебных занятий.**

**Занятие 1. Знакомство с визуальной средой программирования Kodu. Делаем первые шаги.**

Программа в Kodu – это набор правил, которые определяют действия объекта. Для написания правил в Kodu используются два оператора:

**When <условие> Do <действие>**

**When** (англ. - «когда», - «если», - «в то время как») – оператор, определяющий условие;

**Do** (англ. - «делать») – оператор, определяющий непосредственное действие, которое должен выполнить объект при соответствующем условии.

*Программа создается для каждого объекта индивидуально!*

**Упражнение 1. Создание первой программы.**

**Сюжет игры:** Летучая рыба оплывает кувшинки.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Выберите пункт Новый пустой мир (New World). Появится зеленое поле – основа для размещения игровых объектов в мире.

Внизу окна размещены иконки, отображающие основные команды программы.

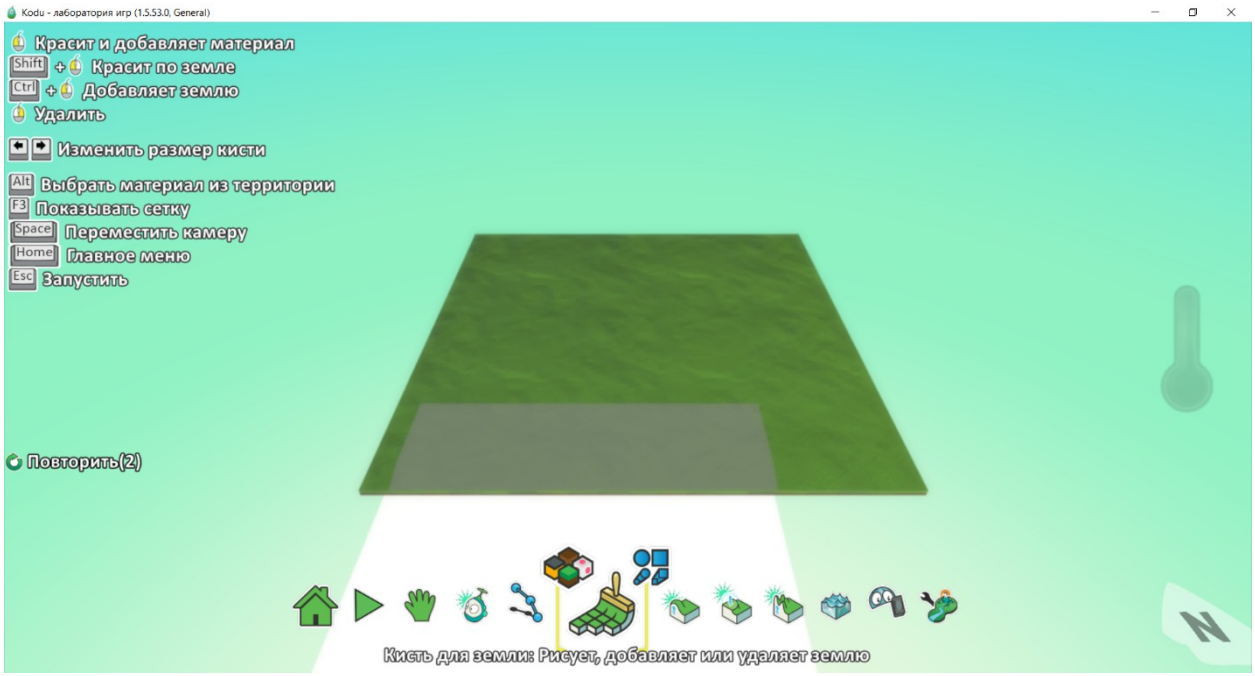



*Рис 1.1. Кисть для земли*

3. Измените ландшафт на водный. Для этого:

a. Наведите мышь на значок кисти;

b. Выберите левое изображение (рис. 1.1.) и найдите 17 цвет земли, имитирующий песок;

c. Измените размер кисти стрелкой вправо и нажмите один раз, чтоб применить выбранный цвет;

d. Затем выберите значок **Вода**. Переведите значок мыши на поле и нажмите левую клавишу мыши, чтобы заполнить пространство водой.

4. Добавьте на поле 5 **кувшинок**;

Для добавления объектов выберите иконку **Объект** и однократно щелкните по полю левой кнопкой мыши. Появится список доступных объектов. Зайдите в раздел **Underwater** и выберите объект **lily pad**. Повторите действия ещё три раза. Либо скопируйте объект **lily pad** наведя на него мышь и нажав сочетание клавиш *ctrl+c*. Для того, чтобы вставить скопированный объект нужно нажать сочетание клавиш *ctrl+v*.

5. Измените размер **кувшинок**. Для этого наведите курсор на объект и нажмите на правую клавишу мыши. После чего выберите пункт меню **Изменить размер**. Задайте величину 1,5.

6. Добавьте объект **Летучая рыба** (рис. 1.2) и задайте для него программу действий. Щелкнув правой кнопкой мыши на объекте **Летучая рыба**, вызовите контекстное меню и перейдите в режим создания Программы.


*Рис. 1.2. Режим выбора объектов*

7. В открывшемся окне кода составьте инструкцию для движения вокруг **кувшинок**. Команды задаются выбором из списка необходимых инструкций. Пример программы приведен на рис. 1.3. При щелчке по карточке (иконка со знаком «+») открывается перечень доступных действий, из которых следует выбрать необходимое (подтвердив действия нажатием левой клавиши мыши).


*Рис. 1.3. Программа, позволяющая Летучей рыбе объезжать кувшинки*

8. Запустите программу на выполнение клавишей **Esc** или нажатием на кнопку в виде треугольника. Понаблюдайте за движением **Летучей рыбы**. Если траектория движения не соответствуют поставленной задаче (объект не двигается или не объезжает **кувшинки**), то проверьте корректность кода (рис. 1.3).

9. Сохраните программу на жестком диске. Для этого перейдите в главное меню (кнопка в виде домика) и выберите команду «Сохранить мой мир».

**Задание для самостоятельной работы:**

1. Попробуйте изменить цвет одной из **кувшинок**, например, на красный.

2. Напишите программу, в которой предусмотрите ситуацию столкновения **Летучей рыбы** с **кувшинкой**.

3. Сохраните игру с пометкой «Самостоятельно».

**Упражнение 2. Создание игры для двух игроков.**

**Сюжет игры:** два **Kodu** едят **яблоки**.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Создайте новый мир (New World);

2. Разместите в нем объекты **Kodu** и **Яблоки**. Для добавления объектов выберите иконку **Объект** и однократно щелкните по полю левой кнопкой мыши. Появится список доступных объектов. Найдите нужные;

3. Напишите программу для управления движением **Kodu** с помощью клавиатуры (пример на рис. 1.4);

4. Добавьте правило поедания **Яблок** («когда **Kodu** коснется **яблока**, он должен его съесть»). Пример кода приведен на рис. 1.4;

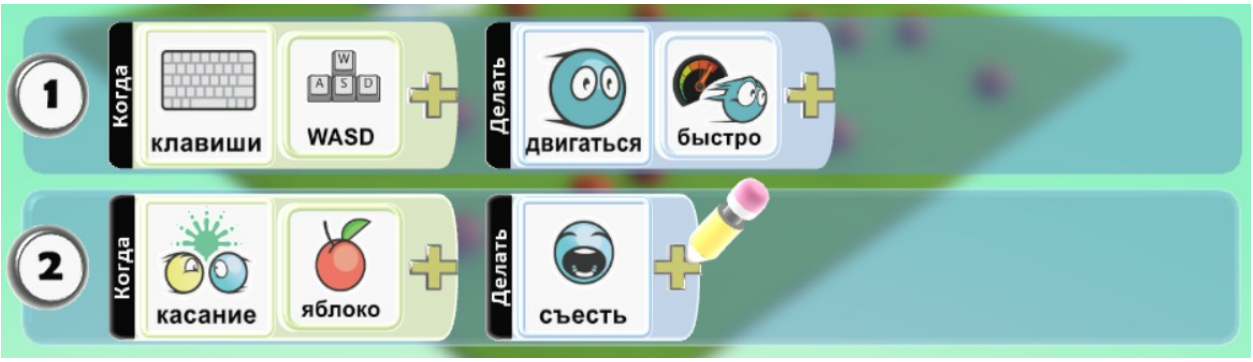



*Рис 1.4. Программная строка для движения Коду и поедания яблок*

5. Создайте второго **Kodu**, выполнив операцию копирования;

6. Измените правило управления вторым **Kodu** в его программе. Для этого назначьте новые управляющие клавиши, например, стрелки;

7. Так как игра будет заключаться в соревновании между двумя персонажами **Kodu**, измените цвета персонажей, допустим, на красный и синий. Сделать это можно с помощью стрелок на клавиатуре. Наведите на персонажа и нажмите на стрелку вправо или влево.

Вы создали игру для двоих участников. Посоревнуйтесь с другом в поедании **Яблок**. Для того, чтобы игра стала интереснее, выполните задание для самостоятельной работы.

**Примечание:** обратите внимание, что при копировании объекта копируется и его программа.

**Задание для самостоятельной работы:**

1. Создайте счётчик для подсчёта собранных яблок для каждого **Kodu**;

2. Напишите условие увеличения счётчика для яблок. Когда оно касается **Kodu** определённого цвета (синего или красного), нужно, чтобы увеличивался счётчик соответствующего персонажа.

3. Сохраните игру, дав ей название и, по желанию, напишите описание игры и добавьте теги. Например, вы сможете отсортировать игры-приключения или стрелялки. Для созданной вами игры, где **Kodu** поедают **яблоки**, подойдут теги **Приключения** или **Гонки** (рис. 1.8). Теги в Kodu Game Lab используются для того, чтобы объединить игры сходной тематики и лучше ориентироваться при поиске

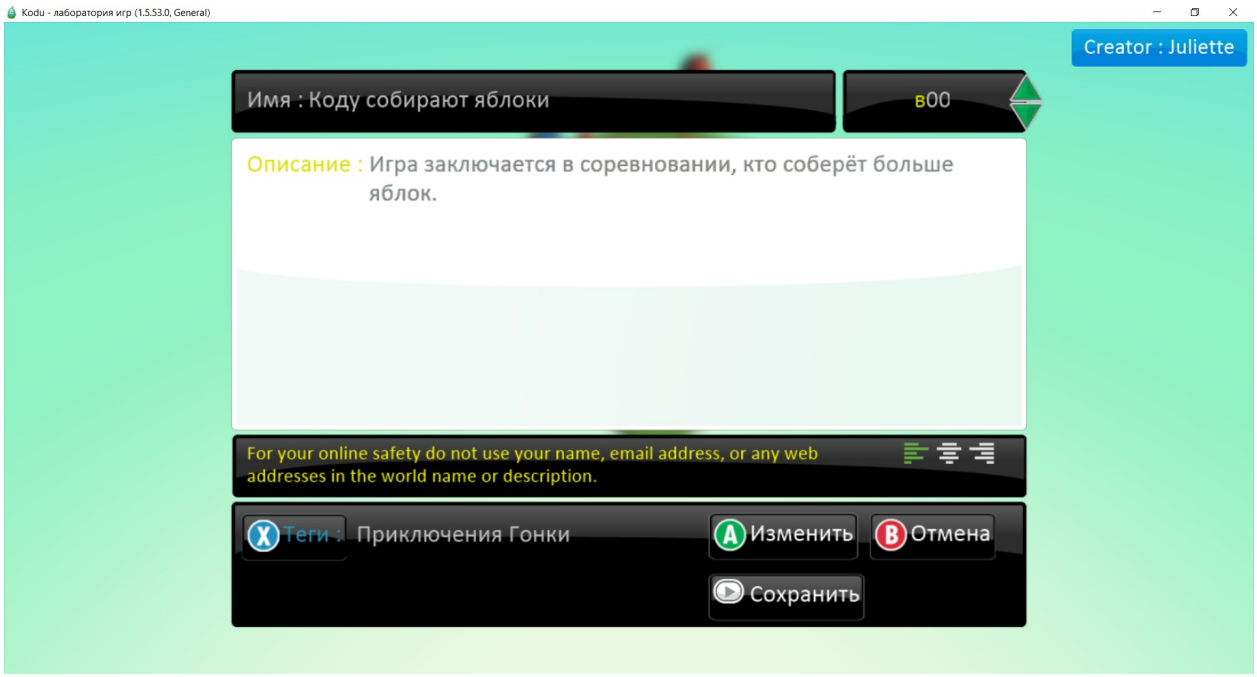



*Рис. 1.8. Экран сохранить с описанием и тегами*

**Упражнение 3. Создание ландшафтов.**

В этом упражнении вы научитесь проектировать вид местности (ее ландшафт), в которой происходят действия игры.

**Ландшафт** – это общий вид местности. Также вы должны создать рельеф. **Рельеф** – (французское relief, от латинского relevo - поднимаю), совокупность форм земной поверхности (суши, дна океанов и морей), различающихся по очертаниям, размерам, происхождению и истории развития.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Создайте новый пустой мир (New World);

2. Создайте ландшафт игрового мира в виде зеленой травы, небольших гор и холмов;

3. Для создания зеленого поля выберите инструмент **Кисть для земли** и произвольно нарисуйте на игровом поле землю;

4. Цвет земли выберите под номерами 15 или17, в зависимости от желаемого типа местности;

5. Поднимите отдельные участки земли, сделав из них холмы и горы;

6. Для создания неровностей используйте инструмент **Вверх/Вниз**. Добавьте воду в созданный ландшафт при помощи инструмента **Вода**, цвет воды под номером 1,2 или 4;

7. Измените цвет неба при помощи инструмента **Параметры мира** (рис 1.9);

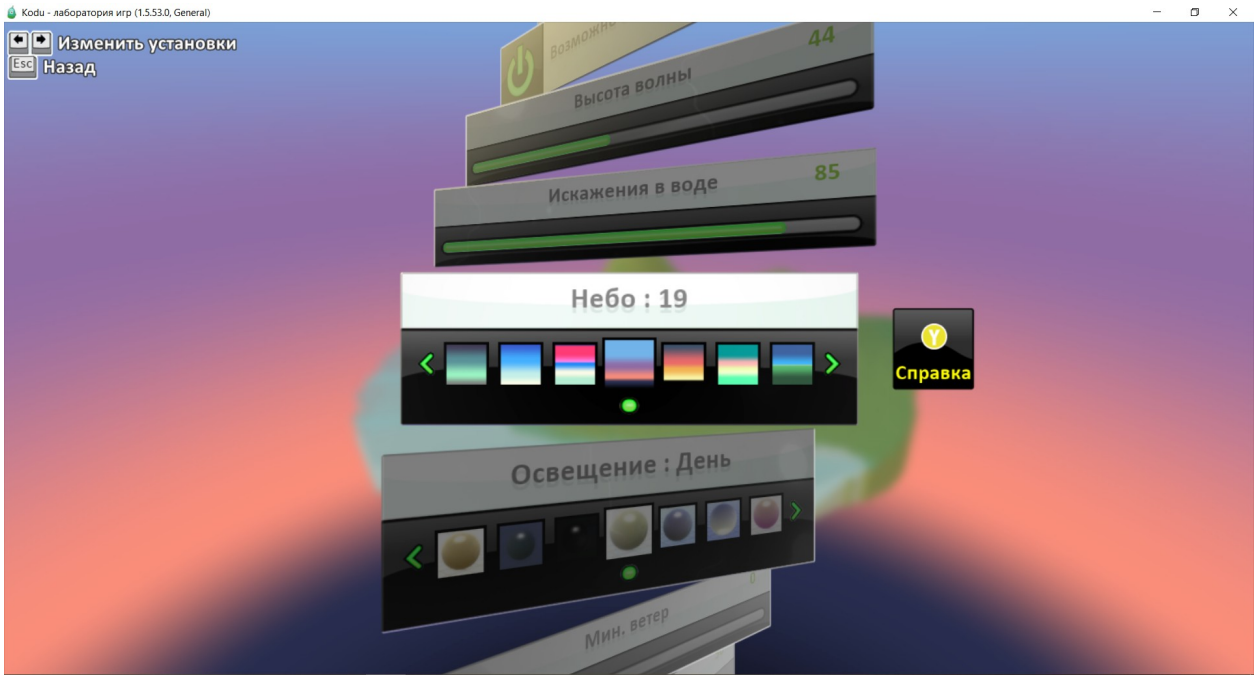



*Рис. 1.9. Инструмент «Параметры мира»*

8. Включите волны для воды.

Пример ландшафта приведен на рис. 1.10.

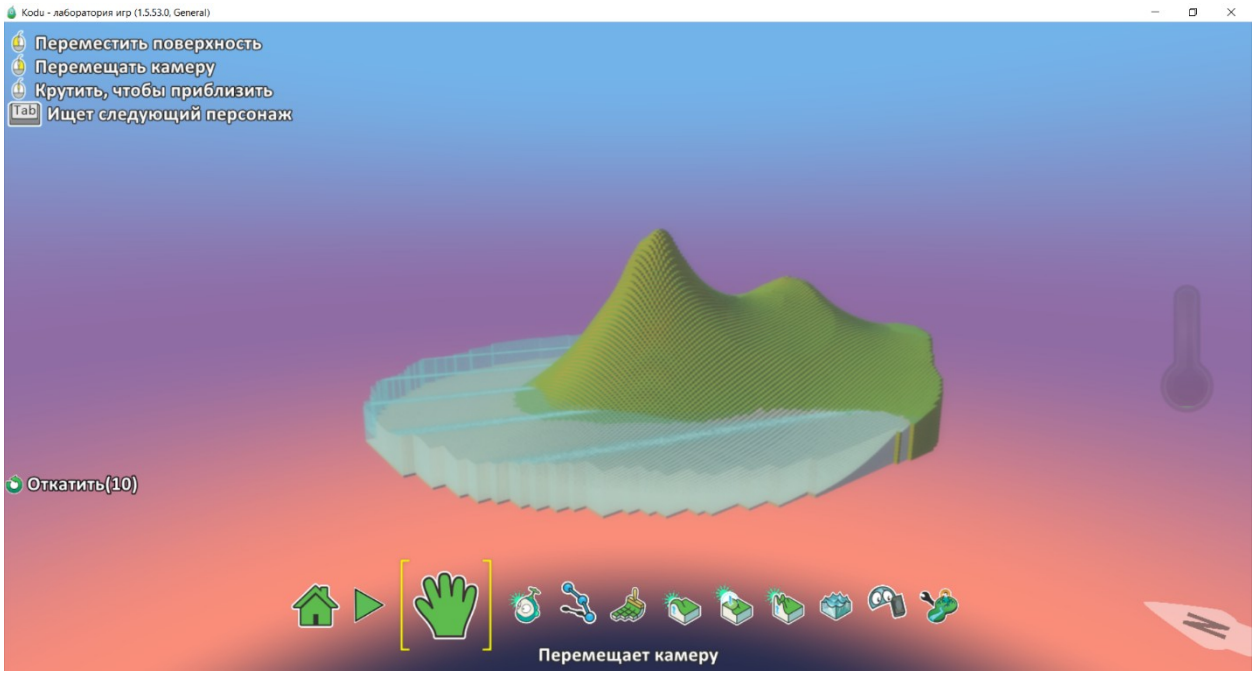



*Рис. 1.10. Пример ландшафта с водоемом*

Сохраните мир под названием **Ландшафт** и запустите игру на выполнение. Обратите внимание на движение воды и набегающие волны.

**Задание для самостоятельной работы:**

1. Создайте в **Kodu** модель местности, используя кисть и её различные типы, а также увеличение и уменьшение размера кисти (размер меняется с помощью стрелок влево/вправо);

2. Добавьте на созданном рельефе холмы и впадины, используя соответствующий инструмент;

3. Постройте стены на моделируемой местности при помощи волшебной кисти;

4. Добавьте несколько дорожек на создаваемой территории;

5. Измените модель местности, добавив водоемы, парки и здания.

6. Сохраните игру с пометкой «Самостоятельно».

**Практическая работа с миром «Стрельба по рыбам (Shooting fish)»**

1. Откройте игровой мир «Стрельба по рыбам (Shooting fish)»;

2. Добавьте объект **Kodu** на игровое поле.

Напишите программу для **Kodu**, с помощью которой можно будет стрелять по рыбам ракетами в разных направлениях. Проведите эксперимент:

сравните действие ракет и пулек. Сделайте вывод о том, какое оружие эффективнее. Изучите различные варианты стрельбы. Настройте стрельбу пульками случайных цветов, вылетающих после каждого нажатия на пробел. Пройдите игру, используя стрельбу ракетами. Сохраните созданный мир.

**Занятие 2. Новые возможности для перемещения объектов и персонажей. Построение пути. Создание клонов и порождаемых объектов.**

На этом занятии вы узнаете, как создаются пути движения персонажей, научитесь создавать клонов и порождения объектов, а также узнаете, как научить объект «говорить». После знакомства с опцией **Родитель,** вы познакомитесь с объектно-ориентированным программированием, которое используется для создания сложных программных систем.

**Объектно-ориентированное программирование (ООП)** – это программирование, в котором основными понятиями являются **объекты** и **классы**. При таком программировании создается класс (например, **—** «форма для печенья»), в который входят различные объекты (например, «печенье»). При этом каждый объект имеет общие характеристики, присущие **всему** классу (мука, вода, сахар и т.д.), но может приобретать и индивидуальные черты (например, «изюм», «шоколад»).

**Повторение пройденного.**

В упражнениях *занятия 1* мы узнали, как создавать коды для объектов в игровом мире. Чтобы закрепить пройденный материал, давайте вспомним самостоятельную работу в *упражнении 2.* В задании к самостоятельной работе *упражнения 2* вам нужно было написать код для **Kodu**, который создаёт счётчик. На рис. 2.1 показан код с ошибкой. Найдите и исправьте эту ошибку.

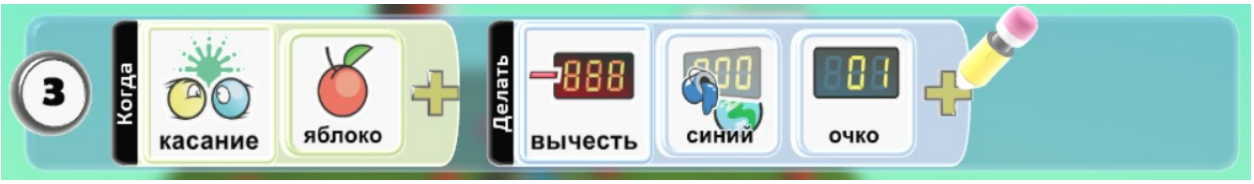



*Рис. 2.1. Ошибочный код*

**Упражнение 1. Создание произвольного пути движения игрового объекта.**

**Сюжет игры: Rover** движется по кругу.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Выберите команду Новый пустой мир (New World).

3. Добавьте объект **Rover**. Для добавления объектов выберите иконку **Объект** и однократно щелкните по полю левой кнопкой мыши и выберите объект **Rover**;

4. Нарисуйте круговую траекторию движения **Rover'а**. Для создания пути следования объекта выберите инструмент **Путь** и при помощи точек отобразите путь следования **Rover'а**, пример на рис. 2.2. Траекторию можно оборвать, нажав на **Esc**;

5. Запрограммируйте движение **Rover'а** по созданному пути, пример кода на рис.2.3;

6. Запустите программу клавишей **Esc** или кнопкой в форме треугольника и проверьте её работу. Байкер должен ездить по кругу;

7. Остановите программу клавишей **Esc**;

8. Сохраните программу под именем **Rover**. Для сохранения программы надо выйти в главное меню (клавиша в форме домика) и выбрать пункт **Сохранить мой мир**.

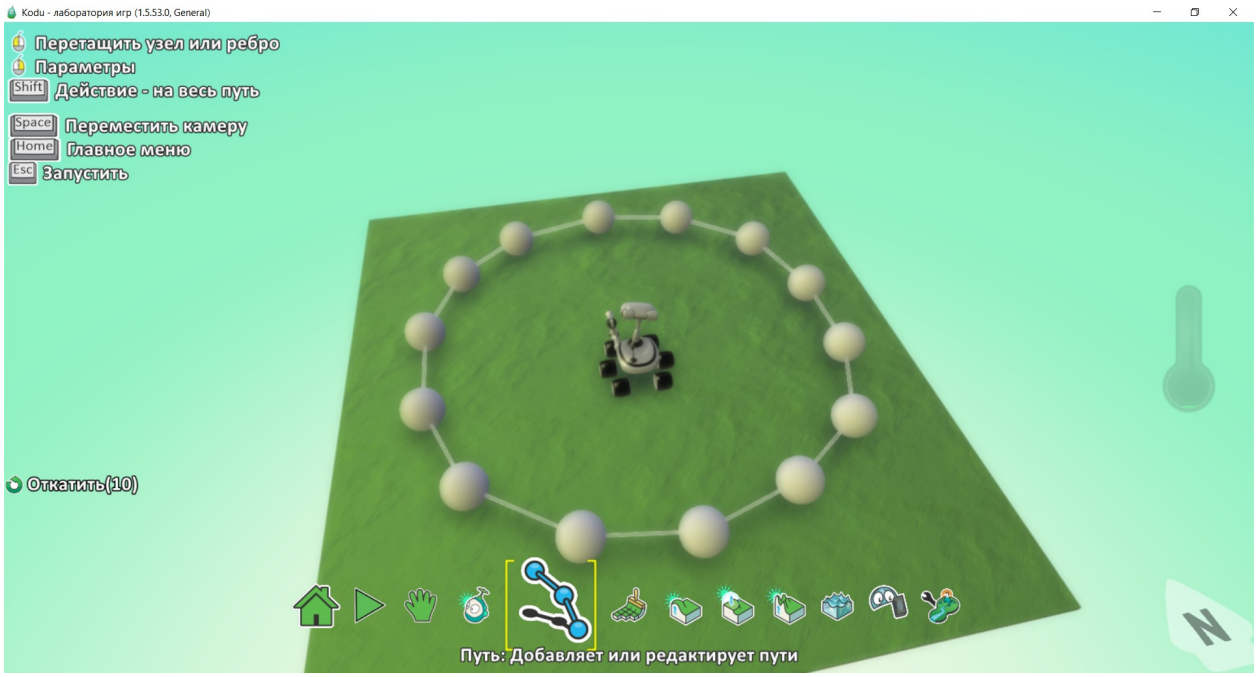



*Рис. 2.2. Создание кругового пути следования объекта Rover*


*Рис. 2.3. Программа для движения объекта по нарисованному пути*

**Задание для самостоятельной работы:**

1. Добавьте ещё одного **Rover'а**, измените его цвет, например, на красный;

2. Нарисуйте путь его движения и обязательно измените цвет пути, например, на красный;

3. Напишите программу, в которой **Rover** движется по красному пути;

4. Напишите код, где в случае столкновения **Rover'ов** друг с другом они говорят: «Прошу прощения!». Примеры пути и кода на рис. 2.4 и 2.5.

5. Сохраните игру с пометкой «Самостоятельно».

Код для столкновения следует написать для обоих **Rover'ов**!

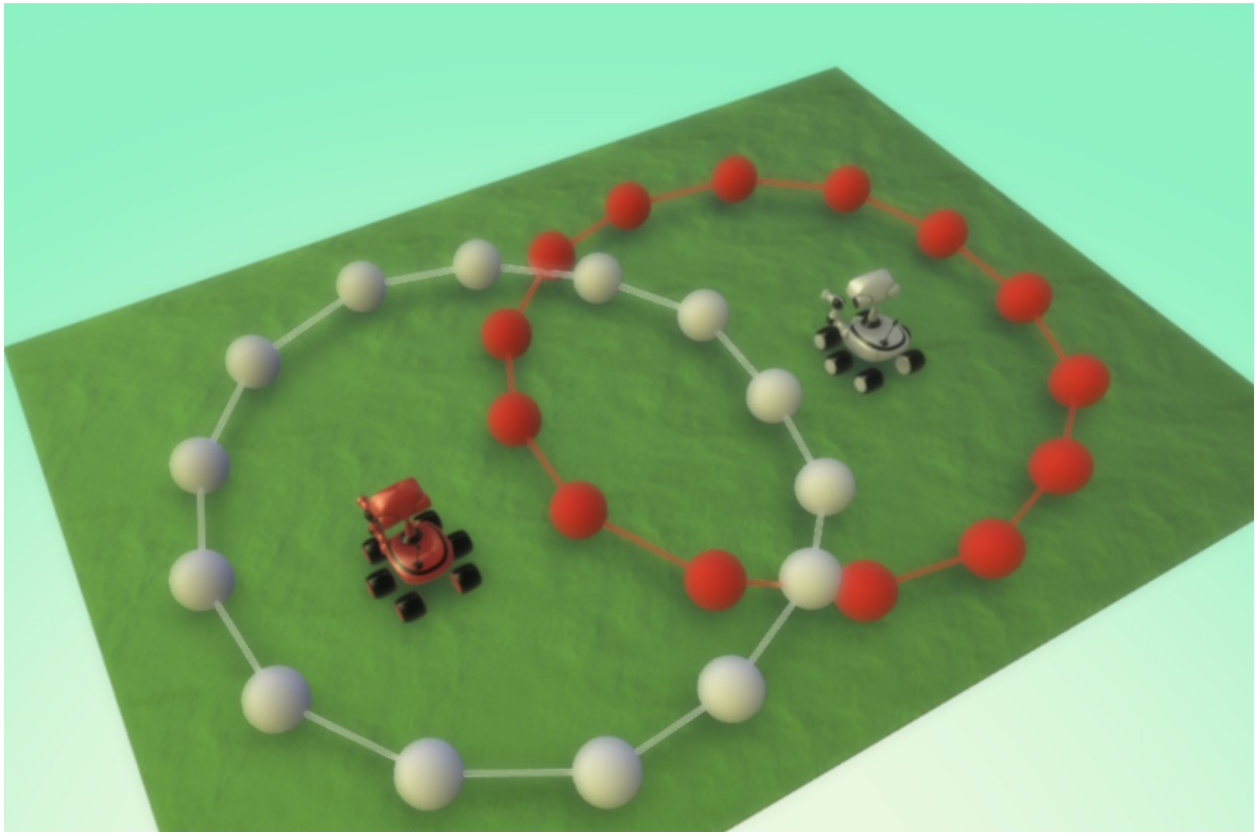


*Рис. 2.4. Создание пути движения для двух объектов Rover*


*Рис. 2.5. Код программы, в которой при столкновении Rover'ов произносятся извинения*

**Упражнение 2. Создание клонов объектов.**

**Сюжет игры:** три **Аэростата** стреляют по **Самолету**.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Создайте новый мир (New World).

2. Разместите в нем объекты **Самолет** и **Аэростат**.

3. Напишите для **Аэростата** код, выполняющий следующие действия: когда **Аэростат** замечает вдали **Самолет**, звучит громкая музыка. Приближаясь к **Самолёту**, **Аэростат** начинает светиться;

4. Составьте программу, позволяющую управлять **Аэростатом** с помощью клавиатуры. Пример программы приведен на рис.2.6, строки 1 – 3;


*Рис. 2.6. Код, управляющий работой Аэростата*

5. Сохраните мир под именем **Самолёт**;

6. Запустите программу на выполнение клавишей **Esc** или кнопкой в виде треугольника. Подведите **Аэростат** к **Самолету**, послушайте музыку, которая была выбрана для игры.

7. Создайте эскадрилью **Аэростатов** из трех объектов. Для этого щелкните на уже имеющемся **Аэростате** правой кнопкой мыши и выберите пункт «Копировать», затем поместите указатель мыши туда, где надо создать новый **Аэростат**, щелкните правой кнопкой мыши и выберите пункт **Вставить** (**Аэростат**). Повторите вставку так, чтобы на поле оказалось три **Аэростата**;

8. Запустите игру и убедитесь, что все **Аэростаты** управляются и двигаются одновременно;

9. Добавьте код, позволяющий каждому **Аэростату** выстреливать пульками, рис. 2.6, строка 4. Сколько шагов нужно изменить, чтобы выполнить *задание 7*? Сохраните игру, запустите её и убедитесь в работоспособности.

**Упражнение 3. Создание порожденных объектов.**

**Сюжет игры:** новая эскадрилья **Аэростатов** стреляет по Самолету.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Откройте созданный в упражнении 2 мир **Самолет**;

2. Добавьте на игровое поле ещё одну группу **Аэростатов**, созданных при помощи технологии порождения. Для этого добавьте новый **Аэростат**, поменяйте его цвет, например, на зелёный;

3. Переместите созданный Аэростат за пределы игрового поля. Данный объект не будет виден в процессе игры, а нужен только для того, чтобы создавать новые объекты **Аэростат**;

4. Задайте созданному **Аэростату** параметр Родитель. Для этого щелкните правой кнопкой мыши по **Аэростату** и выберите пункт «Изменить установки». Активируйте параметр **Родитель**. Запустите программу на выполнение клавишей **Esc** или кнопкой в виде треугольника и убедитесь, что новый **Аэростат** не виден. Остановите программу и перейдите в режим редактирования (клавиша **Esc**);

5. Создайте три **Аэростата** в новой эскадрилье. Для этого скопируйте созданный родительский **Аэростат** (правой клавишей мыши или горячими клавишами *ctrl+c*), а затем трижды вставьте объект **Аэростат** в любое место игрового поля. Если поместить указатель мыши на один из **Аэростатов**, между исходным и новым объектом появится пунктирная линия, указывающая, что эти **Аэростаты** являются частью цепочки созданных объектов (рис.2.7).

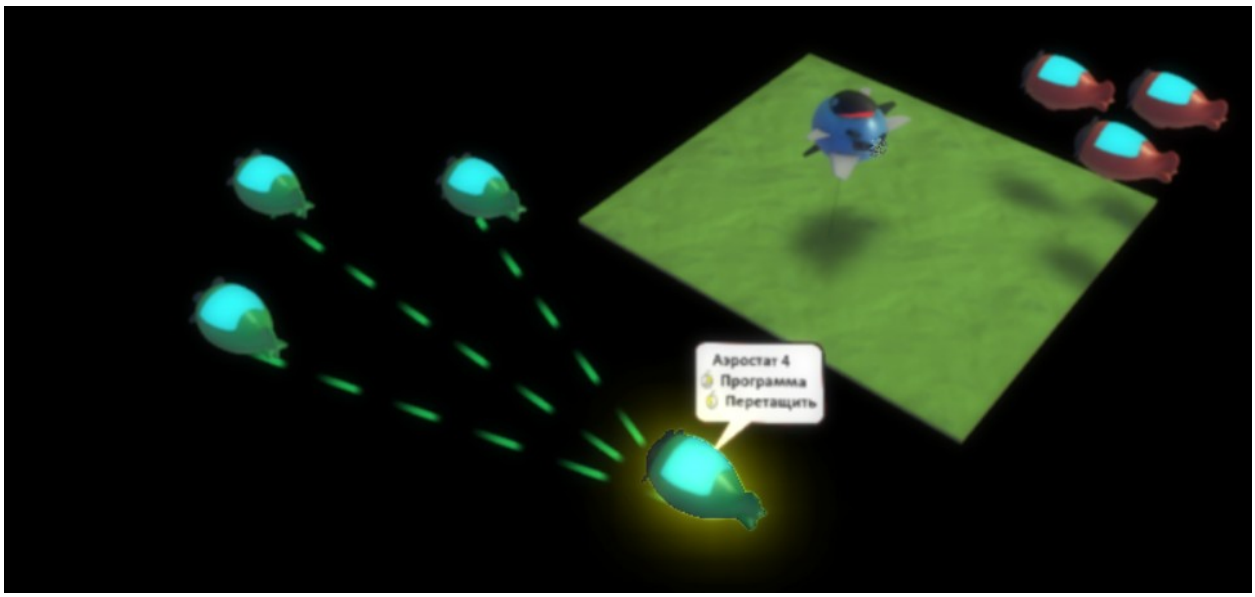



*Рис. 2.7. Порожденные объекты Аэростат связаны с родительским объектом*

**Задание для самостоятельной работы:**

1. Напишите код для родительского **Аэростата**, который заставляет порождённые объекты двигаться к **Самолету**. Обратите внимание, что теперь вам нужно написать код только один раз (для родительского объекта), все порождаемые объекты автоматически получат свойства объекта-родителя;

2. Сохраните программу, запустите её на выполнение, убедитесь в её работоспособности;

3. Напишите код, позволяющий при столкновении **Аэростатов** с **Самолётом** издавать звук.

4. Сохраните игру с пометкой «Самостоятельно».

**Упражнение 4. Опция Родитель.**

**Сюжет игры:** Каждые 3 секунды объект **Clam** создает новый объект **Летающая рыба**. **Kodu** стреляет ракетами по появляющимся рыбам.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Загрузите мир **«Стрельба по рыбам (Shooting fish)»**;

3. Очистите мир от объектов;

4. Создайте объект **Летающая Рыба** и измените её установки, включив опцию **Родитель**;

5. Добавьте объект **Clam** и задайте ему действия таким образом, чтобы каждые 3 секунды он создавал объект **Летающая Рыба**. Используйте таймер и опцию **Родитель**. Пример кода приведен на рис. 2.8.


*Рис. 2.8. Пример кода для создания объекта Рыба*

**Задание для самостоятельной работы:**

1. Создайте программу для **Kodu**, благодаря которой он сможет стрелять по появляющимся рыбам ракетами;

2. Напишите программу для рыб. Они должны двигаться по путям;

3. Измените установки **Kodu** так, чтобы перезарядка ракетами происходила за 1,5 секунды;

4. Сохраните игру с пометкой «Самостоятельно».

**Проверь себя!**

Экспериментируя с местностью в среде Kodu, ответьте на вопросы:

1. Что позволяет делать опция Родитель?

2. Для чего предназначен инструмент, изображенный на рис. 2.9?

*Рис. 2.9. Для чего нужен этот инструмент?*

3. Какие параметры (опции) объектов можно изменять?

4. Как получить информацию по опциям?

5. По каким признакам можно понять, что для объекта включена опция Родитель?

6. Как сделать действие дочерним и что это означает?

**Занятие 3. Знакомство с визуальной средой программирования Kodu: подсчёт баллов, индикатор здоровья, объект таймер.**

На этом занятии вы познакомитесь с тем, как заставить игровой объект выполнять команды в определенные моменты времени, программировать характеристики и поведение персонажей, начислять им баллы за определенные действия, отслеживать успех действий в виде индикатора здоровья.

Использование таймера, индикатора здоровья и функции подсчета очков позволит придать игре профессиональный вид.

**Упражнение 1. Назначение времени действия игрового объекта.**

**Сюжет игры: Рыба** выпускает айсберг каждые 3 секунды и камни каждые 5 секунд.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Создайте новый пустой мир (New World);

3. Добавьте объект **Рыба**;

4. Рыба хорошо себя чувствует в воде, поэтому измените цвет земли на тот, что напоминает песок;

5. Выберите кнопку **Вода** и наполните пространство водой**;**

6. Напишите код для объекта **Рыба**, согласно которому она движется произвольно и каждые 3 секунды Рыба должна создавать айсберг, а каждые 5 секунды – камни (**iceberg** и **rock** соответственно). Для написания кода щелкните правой кнопкой мыши по объекту **Рыба** и выберите команду **Программа**. В открывшемся окне создайте сначала алгоритм движения, затем создание самих объектов (пример приведен на рис. 3.1):

**Если** Таймер – 3 сек, **Делать** действия – запуск – айсберг.

**Если** Таймер - 5 сек, **Делать** действия – создать – камень.


*Рис. 3.1. Пример кода с использованием опций таймера и создания объектов*

7. Запустите программу на выполнение клавишей **Esc** или кнопкой в виде треугольника. Обратите внимание на отличие действий **Запуск** и **Создать**. Действие **Запуск** характеризуется более динамичным действием для создания объекта. Сохраните созданный игровой мир под именем **Рыба**. Для сохранения нажмите клавишу **Главное меню** (кнопка в виде домика) и выберите команду **Сохранить мой мир**. По желанию добавьте описание и теги.

**Задание для самостоятельной работы:**

1. Измените код для Рыбы таким образом, чтобы создаваемые камни были каждый раз разного цвета.

2. Сохраните игру с пометкой «Самостоятельно».

**Упражнение 2. Начисление баллов за действия объекта.**

**Сюжет игры:** Корабль при столкновении с айсбергом получает 1 очко и айсберг уничтожается.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

**1.** Запустите программу Kodu;

**2.** Откройте созданный в *упражнении 1* игровой мир **Рыба**. Для загрузки мира выберите команду **Загрузить мир** в главном меню;

**3.** Добавьте на поле объект **Корабль**;

**4. Корабль** должен плыть к айсбергу, как только он его увидит;

**5.** Напишите код для объекта **Корабль**, чтобы при столкновении с айсбергом происходил подсчет баллов. Пример кода приведен на рис. 3.2.

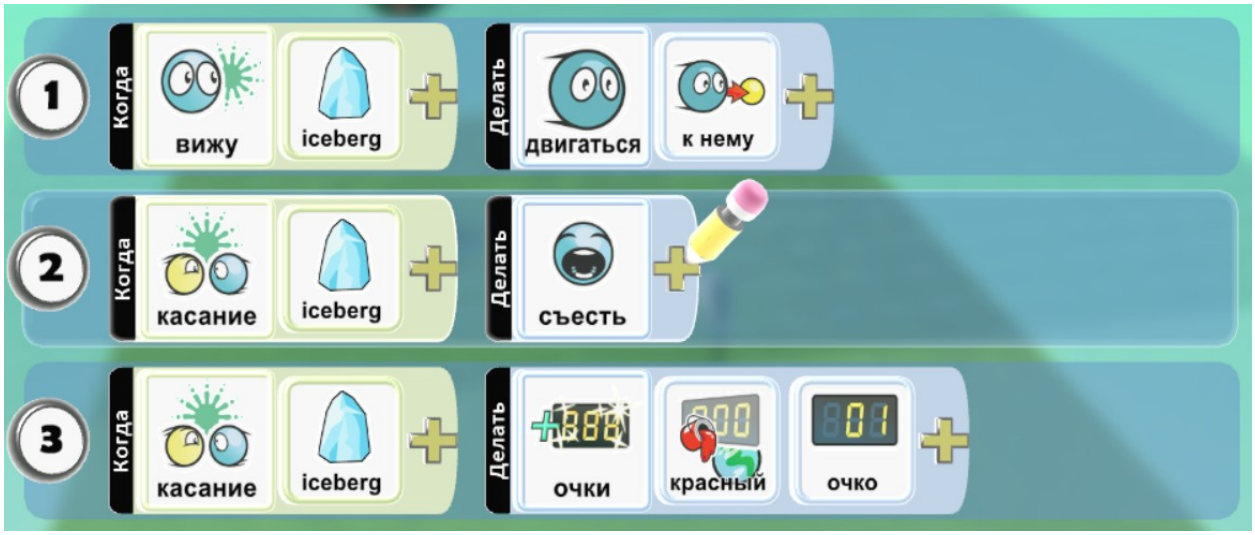



*Рис. 3.2. Пример кода с использованием опции подсчета баллов и исчезновения айсберга при соприкосновении с объектом Корабль*

**6.** Запустите программу на выполнение клавишей **Esc** или кнопкой в виде треугольника. Примерный результат игры приведен на рис. 3.3.

**7.** Сохраните игру.

**Задание для самостоятельной работы:**

1. Измените условие начисления очков. Пусть теперь за каждый айсберг, в который врезается Корабль, начисляется по 2 очка.

2. Создайте Осьминога, который будет плавать по спирали (нужно построить путь) и, когда он будет врезаться в камни, они должны исчезать, а у Осьминога должен появиться счётчик камней. Один камень – одно очко.

3. Напишите условие окончания игры. Когда счёт одного из персонажей (Корабль или Осьминог) будет равен 20 очкам, игра должна закончиться победой.

4. Запустите игру и дождитесь окончания, убедитесь, что всё работает;

5. Удалите путь;

6. Измените управление осьминога на клавиши (w, a, s, d или стрелки), чтобы вы могли за него играть.

7. Попробуйте собрать камни быстрее, чем **Корабль** соберёт айсберги;

8. Сохраните игру с пометкой «Самостоятельно».

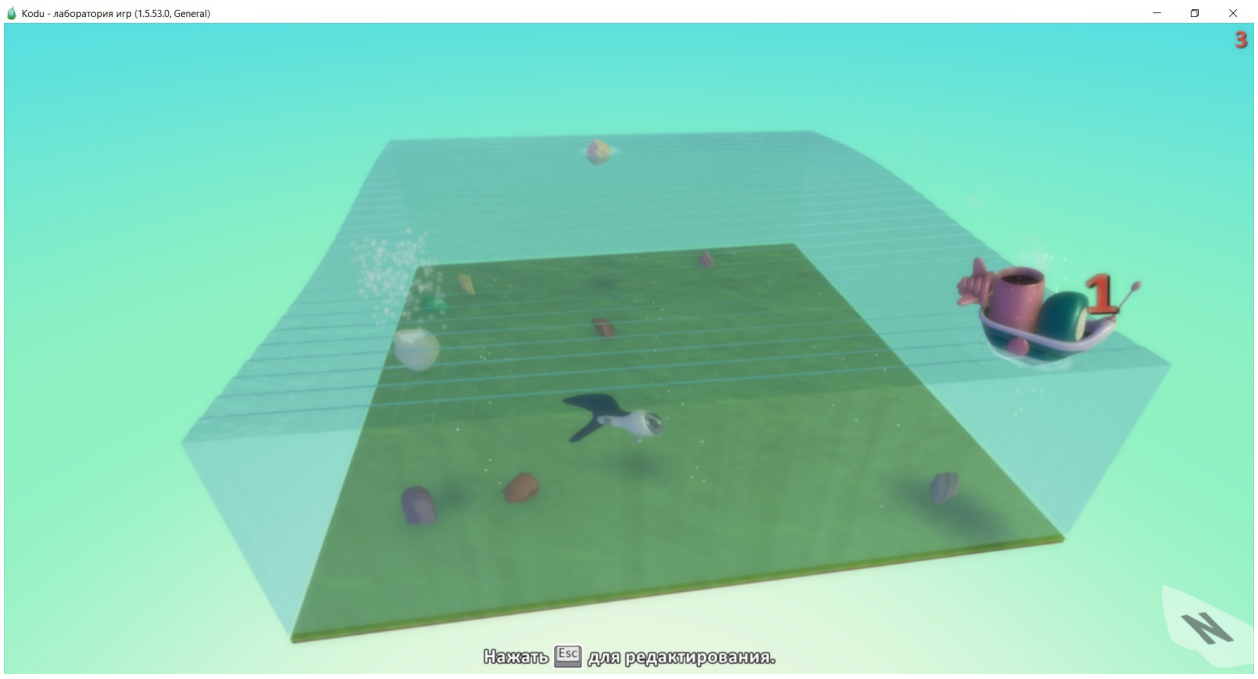



*Рис. 3.3. Пример игры по выпуску рыбой айсбергов и камней и начисления кораблю баллов за столкновение*

**Упражнение 3. Использование индикатора уровня жизни.**

**Сюжет игры:** Игра-квест. Мир, состоящий из островов. **Роботу** нужно собирать монеты и попытать выжить. При низком уровне здоровья он светится красным. Игра завершается, когда собрано нужное количество монет.

На этом этапе мы постараемся использовать все инструменты, которыми вы учились пользоваться на протяжении всего курса.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Создайте несколько островов с помощью инструмента **Кисть для земли;**

3. Измените их высоту, каждый следующий остров должен быть выше предыдущего. Чтобы это сделать, вам понадобится инструмент **Вверх/Вниз.** Нажмите на него и выберите волшебную кисть. Левой клавишей мыши рельеф становится выше, правой – ниже;

4. Добавьте на острова, по желанию, декорации в виде башен, деревьев и других объектов, всё зависит от вашей фантазии;

5. Соедините острова дорогой с помощью кнопки **Путь** (чтобы изменить тип пути, нужно нажать на узел правой клавишей мыши и выбрать пункт **Изменить тип** или нажать на клавишу стрелок вверх/вниз); Дорогу стоит строить из трёх и более узлов. Сделайте так, чтобы она была похожа на мост. Для этого надо изменить высоту центрального узла/узлов. Нажмите на правую клавишу мыши и выберите пункт **Изменить высоту**, после чего потяните за ползунок до желаемой высоты;

6. Создайте объект **Байкер**;

7. Используйте меню **Изменить установки**, для того, чтобы на экран выводилась полоска жизней **Байкера**. Также измените линейную скорость на 2. И, если требуется, размер;

8. Создайте объект **Монета** и сразу измените размер на 3, аналогично изменениям параметров **Байкера**;

9. Напишите программу для монеты. Она должна светиться жёлтым, чтобы её было лучше видно. При касании с **Байкером** должна играть музыка (выберите подходящую). Пример написания на рис. 3.4;


*Рис. 3.4. Пример кода для монеты*

Раз у нас есть полоска здоровья, значит на неё должен кто-то влиять. Давайте создадим врагов и траекторию их полёта.

10. Создайте объект **Спутник**;

11. Измените установки: сделайте **Спутник** неуязвимым, **Размер** – 2, **Время перезарядки пульками** – 0,50, **Пулек за раз** – 5;

12. Постройте маршрут следования с помощью кнопки **Путь**. Измените его цвет, например, на серый, клавишами вправо/влево. Пример пути представлен на рис 3.5 (тип изменён для наглядности);


*Рис. 3.5. Пример размещения пути для объекта спутник*

13. Напишите программу для **Спутника**. Он должен двигаться по путям определённого цвета. Пример написания представлен на рис. 3.6;


*Рис. 3.6. Пример написания программы для объекта спутник*

14. Создайте аналогичные пути на остальных островах;

15. Скопируйте объект **Спутник.** Вставьте хотя бы по одному **Спутнику** на остров;

16. Скопируйте объект **Монета.** Вставьте необходимое количество, минимум 5;

Так как на здоровье **Байкера** будут влиять **Спутники**, давайте создадим объекты, с помощью которых **Байкер** сможет повышать шкалу здоровья.

17. Создайте объект **Сердце.** Измените размер по желанию;

18. Напишите программу для объекта **Сердце.** Когда **Байкер** касается его, **Сердце** должно издать звук и излечить, например, на 20 пунктов здоровья. Пример написания программы представлен на рис. 3.7;


*Рис. 3.6. Пример написания программы для объекта сердце*

19. Разместите на карте хотя бы два **Сердца**;

20. Наконец, напишем программу для главного героя этой игры, для **Байкера**. Сначала программа для движения. Вы делали это уже много раз, так что сложностей возникнуть не должно. Далее, описание сценария для соприкосновения с монетой. Она должна исчезнуть и счётчик монет должен увеличиться, например, на 10 очков. При соприкосновении с сердцем, оно должно исчезнуть. Когда у Байкера остаётся меньше половины здоровья, допустим, 20 пунктов из 50, он должен начать светиться красным (рис. 3.8);

21. Напишите условие завершения игры. Нам интересны и положительный и отрицательные исходы. Условие победы – собрать 5 монет (набрать 50 очков, при условии, что одна собранная монета приносит 10

очков). Условие проигрыша – **Байкер** убит **Спутниками**. Пример написания программы представлен на рис. 3.9;


*Рис. 3.8. Программа для Байкера. Создание счётчика, условие исчезновения монет и сердец и создание свечения при малом запасе здоровья*


*Рис. 3.9. Программа для Байкера. Создание условия победы и проигрыша*

22. Запустите игру клавишей **Esc** или кнопкой в форме треугольника и проверьте её работу. Если всё выполнено по руководству, программа должна работать корректно. Если что-то не получилось, не расстраивайтесь, проверьте написанный код по скриншотам и попробуйте снова;

23. Сохраните игру.

**Занятие 4. Использование страниц в Kodu Game Lab. Создание уникальных историй и персонажей.**

На этом занятии вы узнаете, как использовать несколько страниц в игре. Это полезно для назначения персонажам разных свойств и правил поведения. Использование многостраничных сценариев в Kodu сделает игру более привлекательной и выведет ее на новый уровень. Мы также рассмотрим правила, по которым создаются игры.

**Упражнение 1. Работа с несколькими страницами.**

**Сюжет игры**: Байкер должен выжить, собирая монеты.

Четко следуйте рекомендованному алгоритму, чтобы успешно завершить упражнение и создать игру:

1. Запустите программу Kodu;

2. Создайте новый мир (New World);

3. С помощью инструмента **Кисть для земли**, изменения размера кисти и разных текстур, создадим заготовку для будущей игры, поле должно быть достаточно большим;

4. Создайте объект **Завод,** он будет создавать наши монеты;

5. Напишите программу для завода. Каждые 5 секунд завод должен выбрасывать по одной монете. Через 20 секунд должен произойти переход на 2 страницу кода (рис. 4.1);


*Рис. 4.1. Программа для Завода. Код создания монеты и перехода на 2 стр.*

6. Напишите код, который будет выполняться после перехода на стр. 2. Для того, чтобы перейти на неё, надо нажать на клавишу **Tab.** Пусть теперь монеты создаются каждые 10 секунд и летят слабо и низко (рис 4.2);


*Рис. 4.1. Программа для Завода стр. 2*

7. Создайте объект **Байкер;**

8. Запрограммируйте его на движение, сбор и подсчёт монет и условие проигрыша и перехода на следующую страницу, как на рис. 4.2;

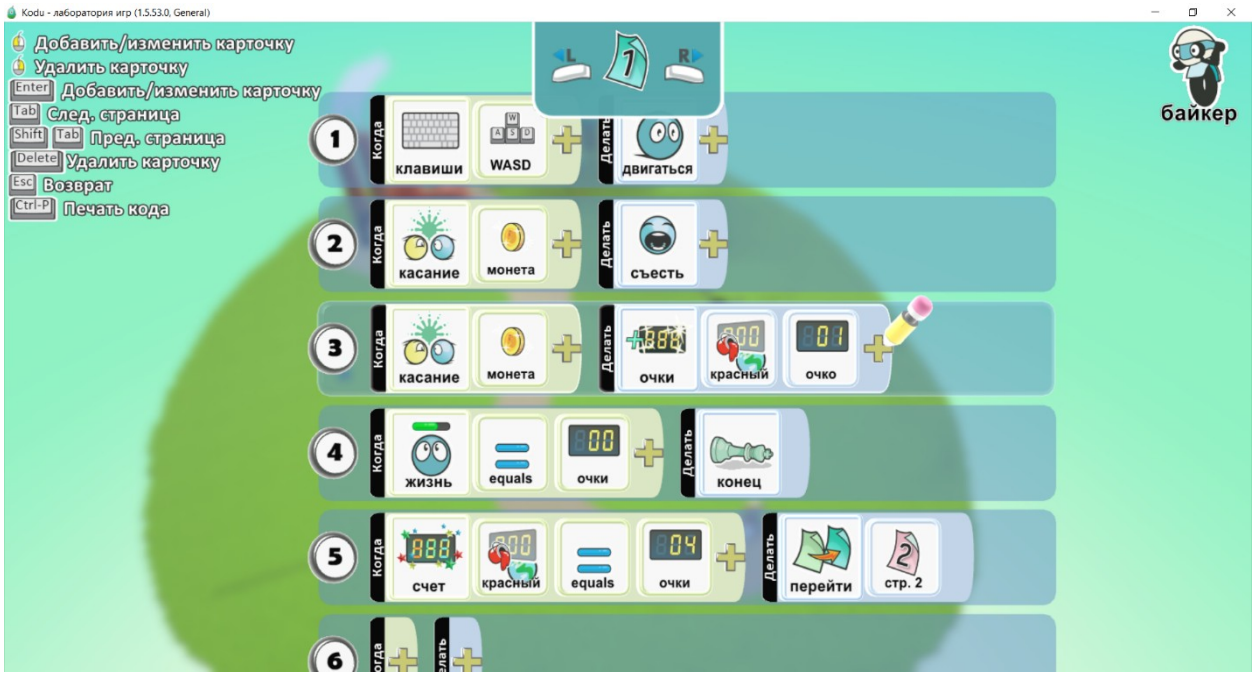



*Рис. 4.2. Программа для Байкера. Создание кода движения, сбора и подсчёта монет, и условия окончания игры и перехода на следующую страницу*

9. Измените установки **Байкера**: линейная скорость и скорость на поворотах – максимальная, ускорение – 5, ускорение на поворотах – 3;

10. Напишите код 2 страницы. Пусть теперь монета приносит по 2 очка. Опишем условие победы. Например, когда счёт достигнет отметки в 10 очков. Условие проигрыша мы уже описали на стр. 1, поэтому можно просто скопировать код оттуда. Для этого нужно нажать правой клавишей мыши на нужной строчке и выбрать пункт «копировать строку». Пример написания программы представлен на рис. 4.3;

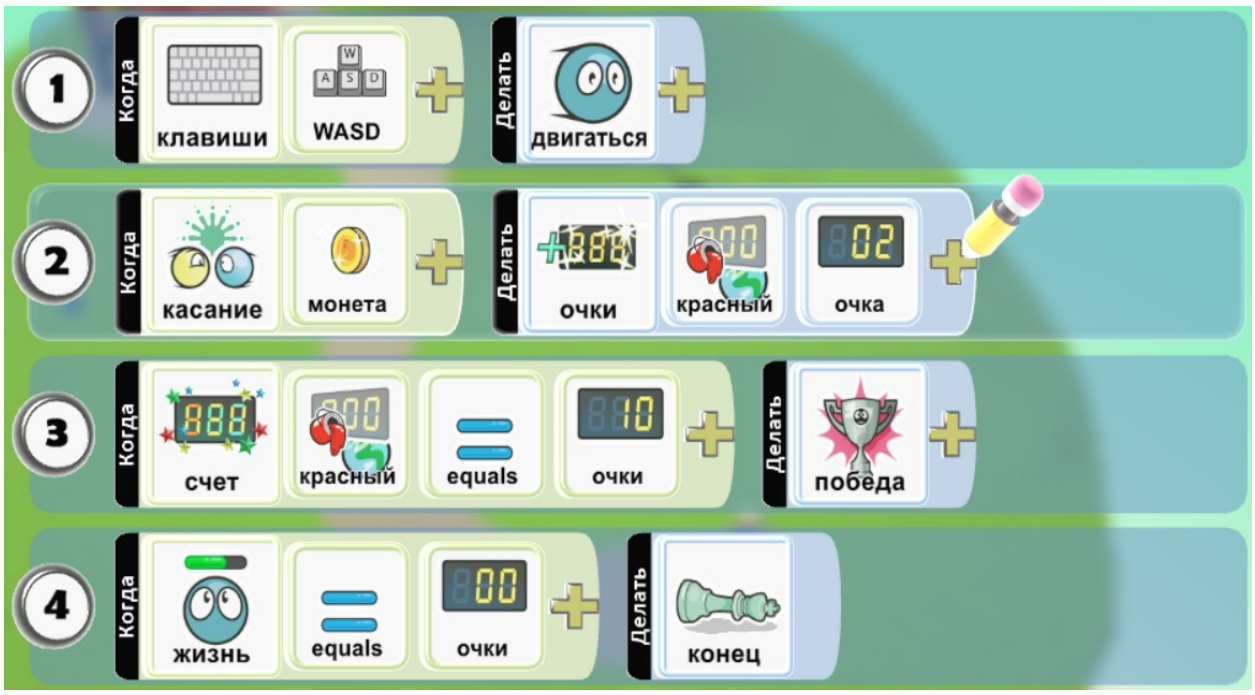



*Рис. 4.3. Программа для Байкера стр. 2*

11. Теперь, чтобы было сложнее и интереснее, создадим врага. Им будет **Спутник.** Измените его размер на максимально возможный и сделайте неподвижным. Он должен быть неподвижен для того, чтобы байкер при столкновении не мог его сбить;

12. Напишите код для **Спутника**: когда он видит байкера, должен стрелять по нему ракетами (рис 4.4).

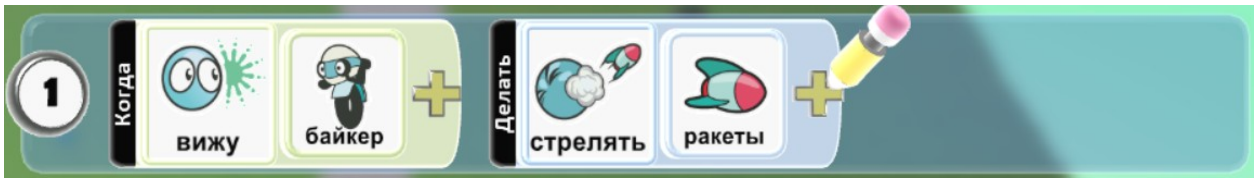



*Рис. 4.4. Программа для Спутника*

Игра готова! Теперь запустите игру клавишей **Esc** или кнопкой в форме треугольника и проверьте её работу. Пример получившегося мира представлен на рис. 4.5.

**Задание для самостоятельной работы:**

Измените код так, чтобы переход на следующую страницу происходил при накоплении определённого количества очков, а по времени.


*Рис. 4.5. Мир, где байкер собирает монеты*

**Упражнение 2. Создание игры с несколькими уровнями.**

Вы познакомились ещё не со всеми основными компонентами среды программирования **Kodu**. Нам предстоит узнать о последней функции – переход на следующий уровень. Для этого нам нужно создать столько миров, сколько мы хотим уровней и сделать функцию перехода из одного в другой.

Давайте возьмём шаблон ландшафта, который мы создавали в упражнении 3, задания 1. Это будет второй мир. Для того, чтобы перейти из одного мира в другой мы можем:

1. Создать таймер, по истечению которого произойдёт переход;

2. Создать условие перехода – счётчик, для персонажа, который, например, собирает какие-то объекты;

3. Поместить на поле объект, касаясь которого произойдёт переход.

Напишем вместе вариант с таймером. Для этого:

**1.** Запустите программу Kodu;

**2.** Откройте мир, под названием **Ландшафт;**

**3.** Добавьте в него дерево, как отличительную черту;

**4.** Сохраните мир под названием **Уровень 2**;

**5.** Снова откройте мир под названием **Ландшафт;**

**6.** Поместите на поле ещё один любой объект: кувшинку, рыбу, башню, это не имеет особого значения, выбирайте на свой вкус;

**7.** Теперь напишите для этого объекта программу, в котором реализуйте первый вариант программы со счётчиком, как на рис. 4.6. Карточку **Next level** ищите в разделе «игра», когда вы её подставите в поле **Делать**, программа предложит вам выбрать следующий уровень, выберите мир под названием **Уровень 2**;

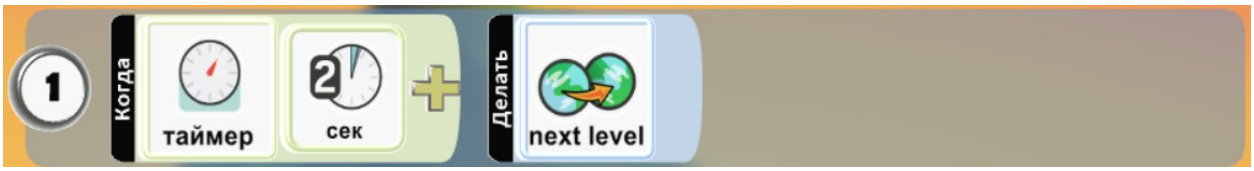



*Рис. 4.6. Условие перехода на следующий уровень с таймером*

**8.** Обязательно сохраните мир под названием **Уровень 1;**

**9.** После этого, проверьте работоспособность программы, запустив её клавишей **Esc** или кнопкой в виде треугольника.

**Задание для самостоятельной работы:**

Реализуйте два оставшихся возможных варианта перехода на следующий уровень, а именно:

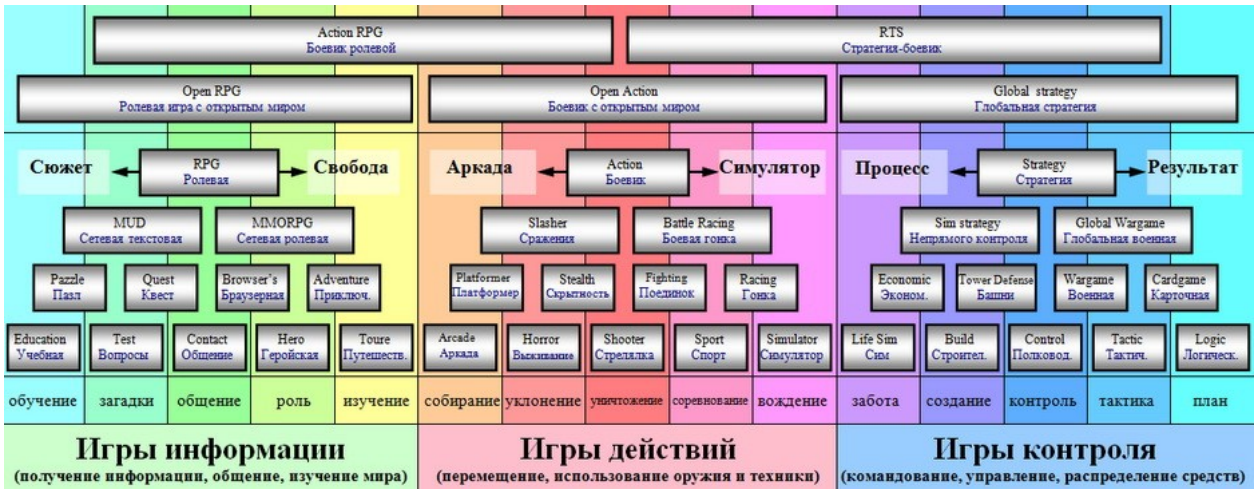
1. Создать условие перехода – счётчик, для персонажа, который, например, собирает какие-то объекты;

2. Поместить на поле объект, касаясь которого произойдёт переход. Для этого вам нужен персонаж, который будет этого объекта касаться.

**Занятие 5. Разработка и создание оригинальной игры своими руками от начала и до конца.**

Для нас проектом будет созданная игра. Какая игра? Это зависит от вас! Но постарайтесь сделать её интересной не только для вас, но и для ваших друзей, родителей, родственников, которые будут в неё играть. Итак, изучив все возможности **Kodu**, приступи к разработке и созданию своей игры!

**Шаг 1.** Выберите жанр игры, чтобы начать. Какая это будет игра? Жанров, как таковых, слишком много, полноценная таблица изображена на рис 5.1.




Уточним их содержание и какие из них мы можем выбрать для создания игры конкретно в **Kodu**. Выберите свой вариант:

**1. Аркада** (Arcade) – основной задачей которой является сбор всех специальных объектов на уровне (иногда есть более конкретные задачи: найти выход, достичь определенной точки). Довольно часто такие игры бесконечны, и цель игры - набрать как можно больше очков. Чаще всего используется вид сверху, реже – вид сбоку. Безмятежный процесс сбора вещей нарушается присутствием на уровнях врагов или ловушек;

**2. Обучающая игра** (Education) – главная задача в которой – получить новую информацию или, вернее, обучить. Самые простейшие из них – изучение цифр, алфавита, названий вещей, отсутствие свободных действий, всё должно идти последовательно.

**3. Стрелялка** (Shooter) – основным видом деятельности является уничтожение всего, что движется. Боеприпасы почти всегда безграничны, поэтому вы можете стрелять непрерывно. Камера может быть двух типов: вид сверху (тогда герой находится либо в середине экрана, и враги приближаются со всех сторон, либо он движется вперед с постоянной скоростью, тем самым в границы экрана попадают всё новые и новые враги) или вид из глаз (при этом герой либо всегда стоит на месте, либо

автоматически передвигается на новую позицию, после уничтожения всех врагов на текущем месте);

**4. Поединок** (Fighting) – основное действие – уничтожение соперников. Два или более противника встречаются на ринге (или в любом другом ограниченном пространстве), сражаются между собой с помощью ударов, специальных движений и оружия. Каждый успешный удар снижает здоровье врага, и в результате тот, кто потерял всё здоровье, считается проигравшим, выживший - победителем.

Этот шаг позволит развить **основную идею** игры [14]. В общем, вы должны описать, где происходит игра, какие предметы игрок контролирует и для какой цели.

**Шаг 2. Проектируем сюжет игры. Для этого:**

**1.** Придумайте и создайте набросок будущего мира на бумаге. На что будет похож виртуальный игровой мир (ландшафта и его объектов: вода, дороги, деревья, горы и т.д.), в котором будут развиваться события;

**2.** Придумайте героев (персонажей, объекты), которыми вы и будете управлять (или которые сами будут производить какие-то действия) продумайте развитие сюжета. Заметим, что сюжет игры, во многом будет определяться возможностями тех объектов, которыми будет управлять играющий.

Во всех играх используют такие элементы как:

- Основной персонаж/персонажи (главные герои).
- Главный враг или препятствие, цель которого – противостоять достижению цели игры (победе). Например, препятствием может стать условие или ограничение, например, по времени.
- Второстепенные персонажи.

Сюжет игры предполагает описание последовательности действий, которые происходят в процессе игры с главным героем/персонажем. Не забудьте сначала сформулировать и написать интересную идею для сюжета игры, а также выбрать подходящее название.

Для того, чтобы сюжет игры был реализуемым, необходимо тщательно изучить возможности управляемых объектов и персонажей. Попробуйте ответить на следующие вопросы:

**1.** Какая миссия у главного героя (или героев, персонажей, объектов)?

**2.** Нужны ли в вашей игре дополнительные герои и объекты? Для чего?

**3.** Есть ли персонажи и объекты, которые мешают вам достичь своей цели?

**4.** Что каждый из героев умеет/будет делать (роль каждого из них в сюжете)?

**5.** Каковы особенности и характеристики выбираемых объектов и как они будут использоваться в процессе игры?

**6.** При каких условиях и при каких обстоятельствах будут использоваться те или иные характеристики (способности) персонажей (объектов)?

**7.** Как будет осуществляться управление персонажами (объектами)?

Итак, на этом этапе вы должны:

- придумать ландшафт и выбрать управляемые объекты;
- изучить/освежить в памяти функции контролируемых персонажей (объектов);
- выбрать варианты поведения персонажей согласно сюжету.

**Шаг 3.** Детализируем цель игры. Очевидно, что интерес к игре вызывает, как процесс, так и достижение определённого результата. Любой, кто играет в вашу игру, должен понимать, что ему нужно делать и каков будет результат. Цели игры (и, следовательно, победа игрока) во многом определяются ее жанром. В зависимости от жанра игры, который вы выбрали, герой должен достичь определённой цели, а именно:

**1.** набрать как можно больше очков;

**2.** выполнить то или иное действие за ограниченный период времени (например, лабиринт надо пройти за 2 минуты);

**3.** прийти к финишу первым;

**4.** собрать/создать/открыть/разрушить/освободить/ те или иные объекты, персонажей;

**5.** соревноваться с другим персонажем, управляемым вторым игроком.

Результатом этого шага будет цель предлагаемой игры и миссии персонажей (героев, объектов) в процессе достижения этой цели.

**Шаг 4.** Реализуйте все планы, которые вы построили, на свой игровой мир. Создайте ландшафт и атмосферу согласно вашему сюжету.

**Шаг 5.** Добавьте персонажей. Запрограммируйте действия персонажей и объектов в соответствии с разработанными правилами и сюжетом игры.

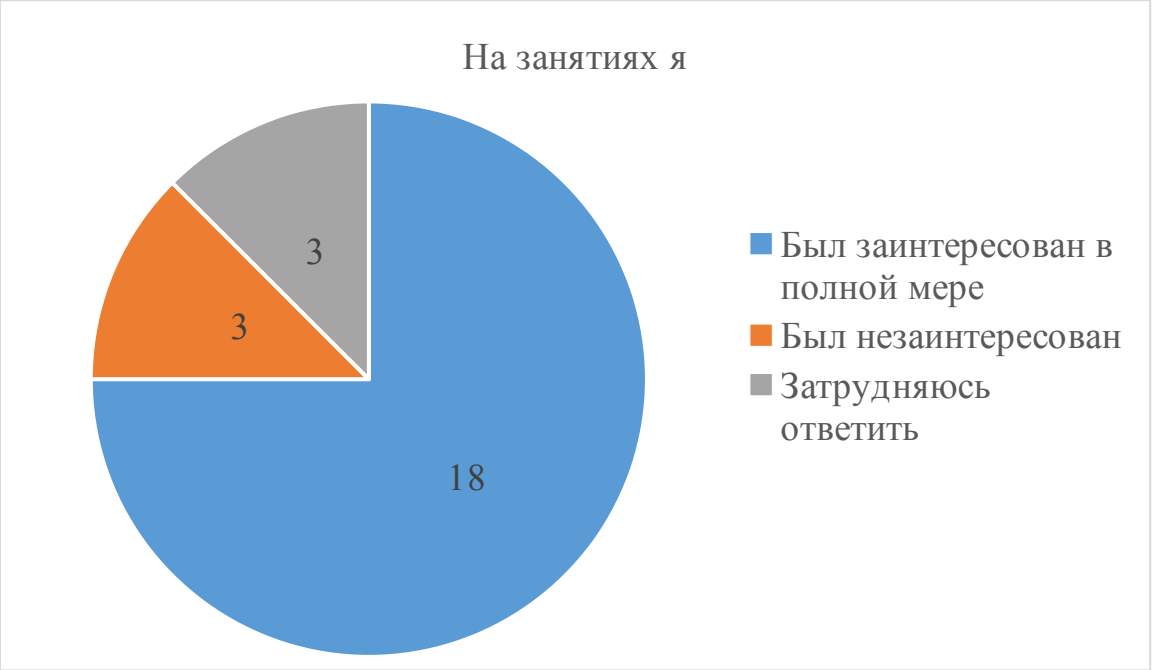
**Шаг 6.** Проведите тестирование игры.

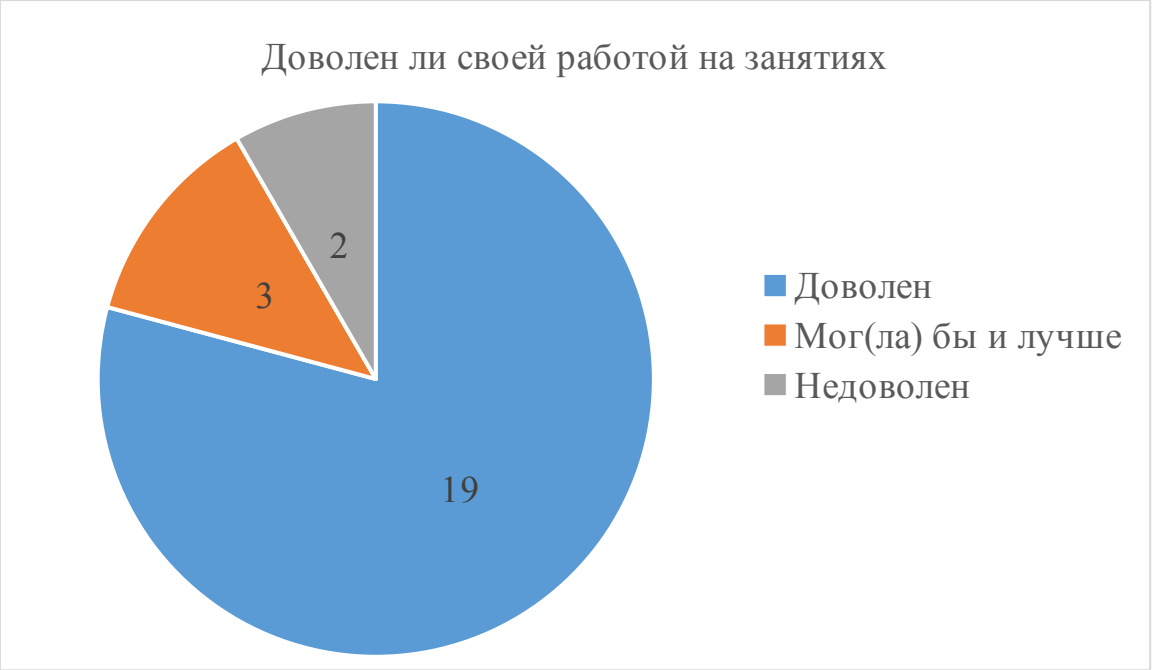
**Шаг 7.** Презентация игры. Расскажите о сюжете и правилах вашей игры, что вас вдохновило на ее создание. Представьте вашу команду разработчиков (которая была ответственна за разработку, расскажите, кто и за что отвечал), отметьте вклад каждого участника. Запустите игру и предложите сыграть в нее всем желающим. Поинтересуйтесь мнением. Обратите внимание на комментарии и предложения, на возникшие трудности, с которыми у игроков возникли проблемы. Это поможет вам улучшить игру и привлечь новых игроков. Это самый большой успех программиста!

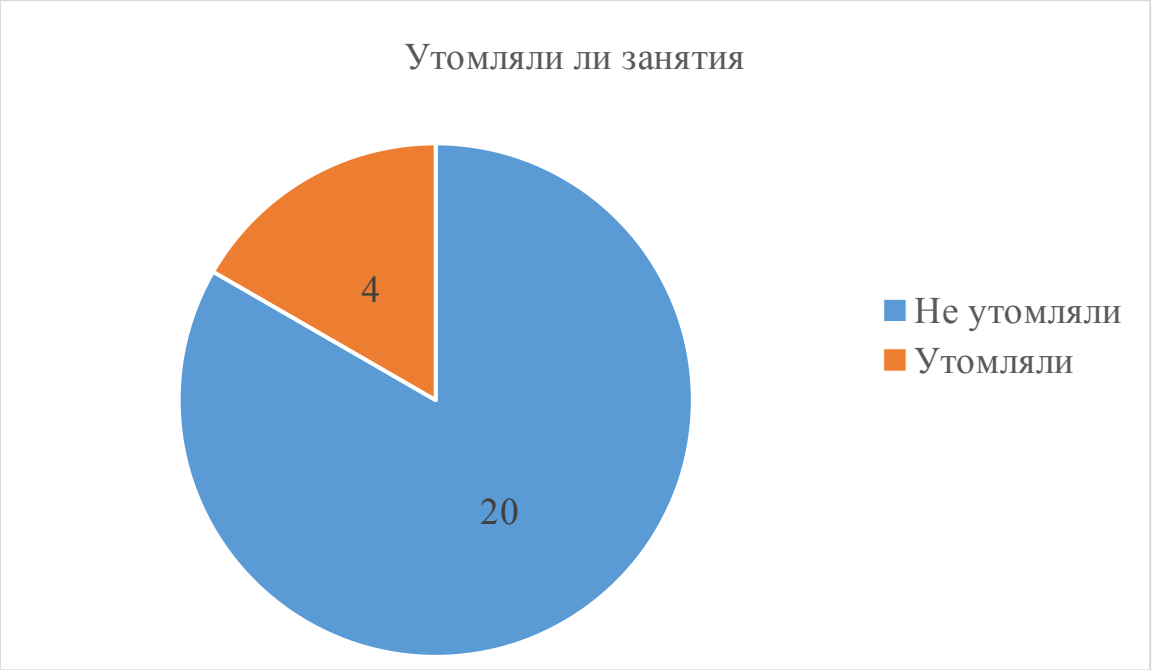
### 2.3. Апробация

Была проведена апробация разработанного обучающего курса по разработке 3D-игр с использованием визуального языка программирования Kodu среди учащихся пятых классов БМАОУ СОШ № 9 г. Берёзовского.

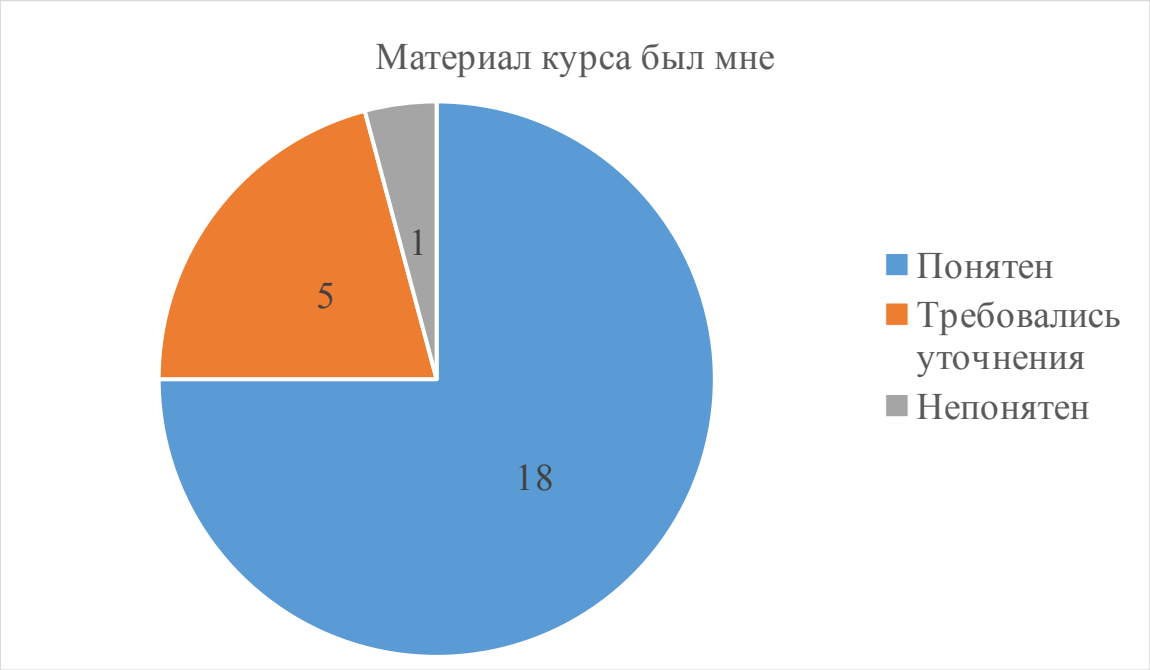
Всего в апробации участвовало две группы 11 и 12 человек. Результат апробации помог оценить эффективность курса, а также помог ответить на вопросы: можно ли было сделать лучше, насколько точно была достигнута цель? Поставленная цель была достигнута. В конце апробации была проведена фаза рефлексии с использованием метода анкетирования. Получены следующие результаты:


На занятиях я
3
3
18
Был заинтересован в полной мере
Был незаинтересован
Затрудняюсь ответить


Доволен ли своей работой на занятиях
2
3
19
Доволен
Мог(ла) бы и лучше
Недоволен


Утомляли ли занятия
4
20
Не утомляли
Утомляли


Моё настроение на занятиях
2
5
17
Улучшалось
Изменялось незначительно
Ухудшалось


Материал курса был мне
1
5
18
Понятен
Требовались уточнения
Непонятен


Появилось ли желание узнавать новые виды информации на уроках информатики, в том числе и "взрослого" программирования
4
5
15
Да
Нет
Затрудняюсь ответить

## Заключение

В соответствии с целью и задачами, сформулированными в выпускной квалификационной работе, было проделано следующее:

1. определены методические особенности изучения основ алгоритмизации и программирования и их реализация с помощью визуальной среды программирования Kodu Game Lab, а именно:
    a. построение алгоритма происходит в зависимости от желаемых, а самое главное, от системных команд, т.е. возможностей исполнителя (например, объект яблоко не может двигаться самостоятельно, он статичен);
    b. алгоритм может быть многоуровневым;
    c. алгоритм может быть многостраничным, среда это позволяет;
    d. алгоритм может быть цикличным, в зависимости от условий;
    e. написание алгоритма должно быть дискретным;
    f. программирование того или иного объекта должно быть поэтапным и строится от простого к сложному (например, сначала задаём условие движения, потом выполнение действий при условиях, таких как, собирание других объектов или условие перехода на другую страницу/уровень);
    g. программирование в Kodu Game Lab знакомит учеников с режимами: редактирования программы, компиляции текста программы, исполнения, работы с файлами, помощи;
2. изучены возможности программы Kodu Game Lab, такие как:
    a. реализация собственного ландшафта;
    b. разработка алгоритма действия в зависимости от объекта;
    c. создание игры по собственному сценарию;
3. разработан цикл занятий по созданию 3D игр с использованием визуального языка программирования Kodu. Всего в курсе 13 упражнений, каждое из которых рассчитано на 1-2 академических часа. Было проведено 10

уроков в 5б классе. Задания, которые выполнены не были или были выполнены частично, отдавались на дом;

4. проведена апробация разработанного цикла занятий, которая подтвердила эффективность обучения с помощью визуального программирования в лице среды Kodu Game Lab.

Таким образом, можно утверждать, что цель выпускной квалификационной работы достигнута, задачи выполнены в полном объеме.

**Библиографический список**

1. Сизикова Т. Э., Волошина Т. В., Повещенко А. Ф. ОБЗОР ИССЛЕДОВАНИЙ РЕФЛЕКСИИ В ПСИХОЛОГИИ. ПЕДАГОГИЧЕСКАЯ РЕФЛЕКСИЯ // Научное обозрение. Педагогические науки. – 2016. – № 2. – С. 89-102; URL: https://science-pedagogy.ru/ru/article/view?id=1488 (дата обращения: 19.11.2019).

2. Дергачева О. Е. Основные положения теории самодетерминации Э. Деси и Р. Райана // Материалы VIII Международной конференции студентов и аспирантов по фундаментальным наукам «Ломоносов–2001». М., 2001.

3. Обухова Л. Ф. Концепция Жана Пиаже: за и против. - М.: Изд-во МГУ, 1981. - 191 с.

4. Федеральный государственный образовательный стандарт среднего общего образования от 17 мая 2012 г. № 413.

5. Алейникова О. М. Методика преподавания непрерывного курса алгоритмизации в общеобразовательной школе // Известия РГПУ им. А. И. Герцена. 2007. №45 311-314 с.

6. Kodu Classroom Kit for Educators [Электронный ресурс] // Режим доступа: http://fuse.microsoft.com /page/ kodu.asp

7. Яникова Н. В., Михеева О. П., Брыксина О. Ф., Останин Я. Е. 5 простых шагов к созданию 3D игр вместе с KODU. 2013 – 51с.

8. Информатика. 5–6 классы: методическое пособие / Л. Л. Босова, А. Ю. Босова. – 2-е изд., перераб. – М.: БИНОМ. Лаборатория знаний, 2017. – 384 с.

9. Геймификация: как превратить урок в игру и не перестараться [Электронный ресурс] / Анастасия Яковлева. – Электрон. текстовые дан. –

2019. – Режим доступа: https://mel.fm/shkola/6783041-gamification, свободный.

10. Актуальная ситуация развития сектора «эдьютейнмент» для детей в России / С. Г. Косарецкий, М. А. Кудрявцева, К. А. Фиофанова; Национальный исследовательский университет «Высшая школа экономики», Институт образования. – М.: НИУ ВШЭ, 2018. – 36 с. – 300 экз. – (Факты образования № 3(18)).

11. Малев В. В. Общая методика преподавания информатики: Учебное пособие. - Воронеж: ВГПУ, 2005. - 271 с.

12. Авраменко А. П. НАУЧНЫЕ ВЕДОМОСТИ [Электронный ресурс] / А. П. Авраменко. – Электрон. журн. – 2013. – Режим доступа: http://dspace.bsu.edu.ru/bitstream/123456789/11823/1/Avramenko_Metodika.pdf.

13. Code.org [Электронный ресурс] /. – Электрон. текстовые дан. – Режим доступа: https://code.org, свободный.

14. Киризлеев А. Ю. Компьютерные игры как искусство [Электронный ресурс] / А. Ю. Киризлеев. – Электрон. текстовые дан. – 2009. – Режим доступа: http://gamesisart.ru/janr.html, свободный.

15. . Алейникова О. М. Методика преподавания непрерывного курса алгоритмизации в общеобразовательной школе // Известия РГПУ им. А.И. Герцена. 2007. №45

16. Апатова Н. В. Информационные технологии в школьном образовании. – М.: Просвещение-АСТ, 1994. – 362 с.

17. Боссова Л. Л., Боссова А. Ю. Информатика // 2014. – 80 с.

18. Булгакова Н. Н. Активизация учебно-познавательной деятельности младших школьников на уроках информатики // Сб. «Учебные технологии». – СПб.: НОВА, 2004. – 482 с.

19. Вербицкий А. А. Активное обучение в школе: контекстный подход. - М.: Просвещение, 1991. – 218 с.

20. Гин А. А. Приемы педагогической техники: Свобода выбора. Открытость. Деятельность. Обратная связь. Идеальность: Пособие для учителя. – М.: Вита-Пресс, 1999.

21. Головина Л. М. Активизация познавательной деятельности учащихся. – М.: Проспект, 2003. – 242 с.

22. Горячев А. В., Суворова Н. И., Информатика «Логика и алгоритмы» // 2015. – 95 с.

23. Куличкова А. Г., Информатика // 2015. – 125 с.

24. Лапчик М. П. Вычисления. Алгоритмизация. Программирование: Пособие для учителя – М.: Просвещение, 1988. -167с.

25. Лапчик М. П., Семакин И. Г., Хеннер Е. К., Рагулина М. И. Теория и методика обучения информатике. – М.: Академия, 2008. – 592 с

26. Платонова Т. А. Роль мотивации в познавательной активности // Сб. «Активность личности в обучении». - М.: Педагогика, 1986. – 308 с.

27. Подковыров А. М. Педагогический опыт информатизации школьного образования // Материалы научно-практической конференции. – М.: Изд. МГУ, 2003. – 487 с. 68

28. Попов М. В. Технология применения компьютера в учебном процессе // Сб. «Учебные технологии». – СПб.: НОВА, 2004. – 482 с.

29. Редькина А. В. Обучение синтезу алгоритмов // Вестник СибГАУ. 2008. №1 С.30-34. 17. Розенберг Н. М. Информационная культура в содержании общего образования // Советская педагогика. - 1989. - № 3.

30. Селевко Г. К. Педагогические технологии на основе информационно-коммуникационных средств. – М.: Народное образование, 2005. – 208 с.

31. Селевко Г. К. Педагогические технологии на основе эффективности управления и организации учебного процесса. Компьютерные (новые информационные) технологии обучения // Информатика и информационные технологии в образовании. - №12. – 2004.

32. Софронова Н. В. Теория и методика обучения информатике - М.: Высшая школа, 2004. - 145с.

33. Ткаченко В. А. О выборе конструкторов игр для использования в программах дополнительного образования детей // Вестник НВГУ. 2011. №3 С.69-74.

34. Цветкова М. С., Информатика 3-4 класс // 2013. – 68 с.

35. Шамова Т. И. Активизация учения школьников. – М.: Педагогика, 1982. – 278 с.

36. Шеншев Л. В. Компьютерное обучение: прогресс или регресс? // Педагогика. – 1992. - № 11

37. Шиянова Ю. В. Методика преподавания программирования в школе // Актуальные проблемы гуманитарных и естественных наук. 2014. №12-2 С.144-146. 69

38. Якименко О. В., Стась А. Н. Применение обучающих программ тренажёров в обучении программированию // Вестник ТГПУ. 2009. №1 С.54-56

39. Брыксина О. Ф. Внеурочная деятельность в условиях ФГОС. Визуальное программирование в Kodu: первый шаг к ИТ-образованию – М., 2013.

40. Овчинникова И. Г., Карманова Е. В. Объектно-ориентированный анализ и моделирование: учебное пособие. – Магнитогорск: МаГУ, 2012. - 150 с.

41. Климова Т. Е. Подготовка учителя к использованию новых информационных технологий в профессиональной деятельности: учебно-методическое пособие/Т. Е. Климова, Е. П. Романов, Е. В. Федченко. - Магнитогорск, МаГУ, 2006. -175 с.

42. Новые информационные технологии в автоматизированных системах [Электронный ресурс] / Е. А. Ерохина, Д. В. Хруслова. –Электрон. журн. – Москва: Московский институт электроники и математики НИУ ВШЭ, 2018. –Режим доступа к журн.: –Электрон. версия печ. публикации