Московский авиационный институт

(научно исследовательский институт)

Факультет: «информационные технологии и прикладная математика».

Кафедра: «Компьютерная математика».

**Реферат по теме:**

|  |  |
| --- | --- |
| **Студент:**  **Тарабукин Марк Иванович** |  |
| **Группа: М8О-113Б-21** |  |
| **Преподаватель: Довженко А.А.** |  |
| **Оценка** |  |
| **Дата** |  |

Алгоритмический тренажёр “Математор”

Оглавление

[Предисловие 4](#_Toc89467652)

[Введение 4](#_Toc89467653)

[Идея 4](#_Toc89467654)

[Реализация 5](#_Toc89467655)

[Выбор средств разработки 5](#_Toc89467656)

[Создание сцены 5](#_Toc89467657)

[Создания тела Математора 6](#_Toc89467658)

[Создание кубов для ввода 6](#_Toc89467659)

[Создание экрана Математора 7](#_Toc89467660)

[Мигающее подчёркивание 7](#_Toc89467661)

[Считывание кубами 8](#_Toc89467662)

[Перемещение кубов 8](#_Toc89467663)

[Считывание ввода 8](#_Toc89467664)

[Создания состояний 9](#_Toc89467665)

[Создание первого уровня 9](#_Toc89467666)

[Создание кнопок 10](#_Toc89467667)

[Создание интерактивного окружения 11](#_Toc89467668)

[Вывод ввода при помощи кубов 11](#_Toc89467669)

[Удаление введённых кубов 12](#_Toc89467670)

[Ввод кубов в Математор 13](#_Toc89467671)

[Уничтожение серых кубов 13](#_Toc89467672)

[Вывод ответа снизу 14](#_Toc89467673)

[Добавления кнопки стирания выведенных кубов 15](#_Toc89467674)

[Поведение кубов во время тестов 16](#_Toc89467675)

[Сохранность кубов 16](#_Toc89467676)

[История действий 17](#_Toc89467677)

[Создание макета 17](#_Toc89467678)

[Запись и просмотр истории 17](#_Toc89467679)

[Создание свода правил игры 19](#_Toc89467680)

[Уровни сложности 19](#_Toc89467681)

[Уровни 20](#_Toc89467682)

[Добавление новых уровней 20](#_Toc89467683)

[Ограниченный рандом 21](#_Toc89467684)

[Завершение разработки 22](#_Toc89467685)

[Тестирование 22](#_Toc89467686)

[Вывод 24](#_Toc89467687)

# Предисловие

## Введение

Любой человек встречается со множеством задач, от самых простых и банальных, например, сделать бутерброд, до самых сложных, таких как создание ЭВМ. Каждую задачу человек подсознательно разбивает на этапы, которые затем выполняет.

Последовательность, реализация этих этапов может быть очень разной, но суть остаётся одна — выполнить поставленную задачу. Не важно, что простой обыватель нарежет в первую очередь — колбасу или хлеб — если в итоге получит приятное дополнение к чаю.

Последовательность действий, выполняемая для достижения определённой цели или решения задачи и называется алгоритмом.

Примерами алгоритмов являются рецепты, инструкции, планы действий.

Алгоритмы нашли широкое применение в программировании. Самые популярные из них — алгоритмы сортировки, используемые повсеместно.

## Идея

При помощи алгоритмов можно определить склонность к математике человека, оценить его мышление и интеллект. Делается это довольно просто: необходимо дать человеку устройство ввода, обработать введённые человеком данные при помощи алгоритма, вывести результат человеку.

Задав некоторое количество вводных данных и получив результат, тестируемый сможет понять, какой алгоритм обрабатывает данные, и будет способен сам обработать ввод, получив ответ, совпадающий с ответом алгоритма на такие вводные данные.

Идея заключается в создании тренажёра, который будет проводить тестирование.

У тренажёра будет несколько состояний, основными из которых будут выполнение алгоритма и проверка того, смог ли пользователь распознать идею алгоритма.

Основная задача — дать человеку возможность вводить свои специфичные данные и получать закономерный ответ. После этого пользователь должен сам суметь дать ответ на любые входные данные.

# Реализация

## Выбор средств разработки

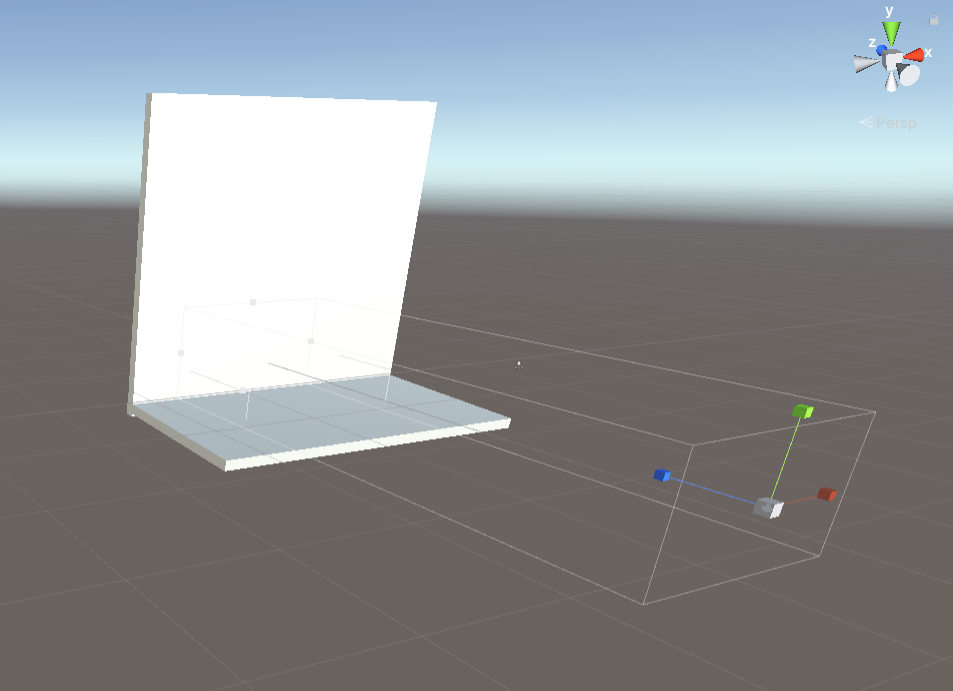
Для реализации данной идеи воспользуемся игровым движком Unity.

Мой выбор пал именно на него, так как у него достаточно низкий порог входа и научиться работать в нём на базовом уровне можно за пару дней. Также он популярен, значит найти обучающие видео и гайды не составит труда, а кроссплатформенность позволит собрать проект под разные платформы.

Использовать в качестве среды программирования я буду Microsoft Visual Studio, так как она оптимизирована для работы с Unity и обладает широким функционалом.

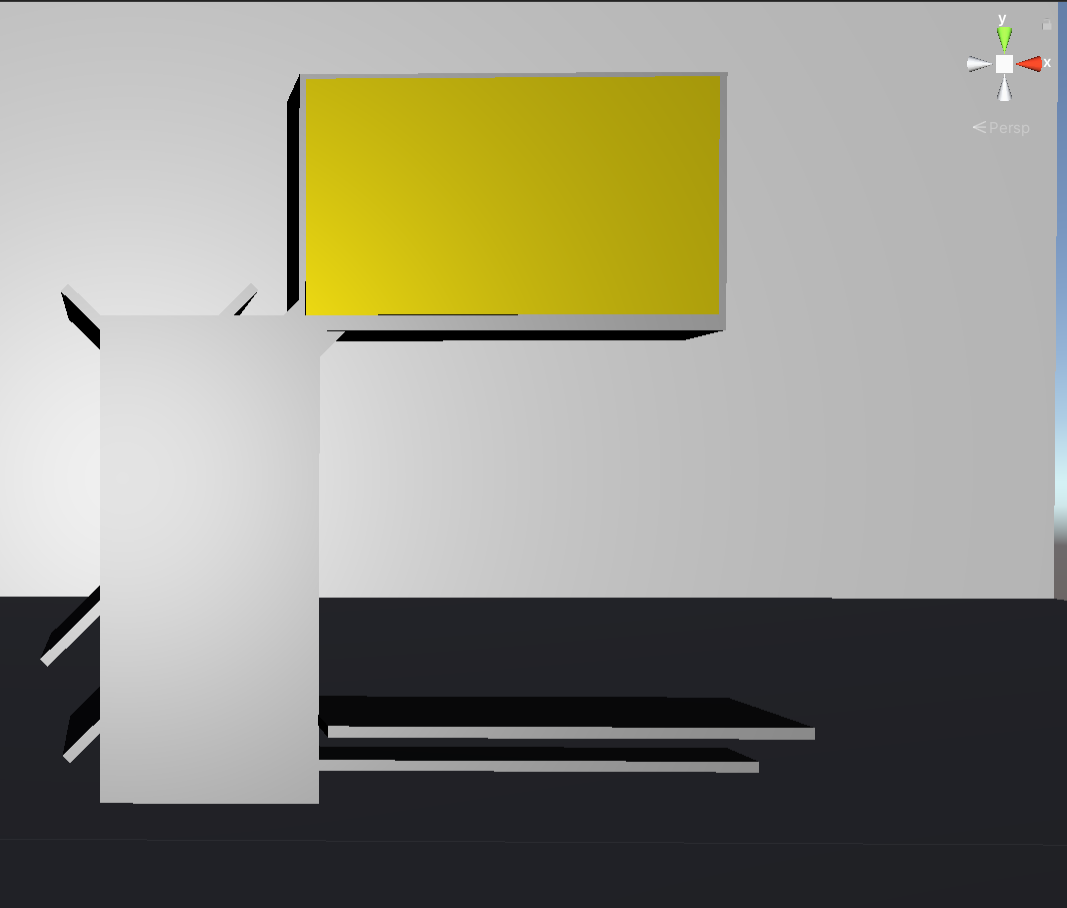
Выбора языка как такового нет, так как скрипты в Unity пишутся на C#.

## Создание сцены

Для начала создадим сцену и расположим на ней такие объекты как Canvas (Пространство, где происходит отрисовка UI), камера, свет и два параллелепипеда — пол и стена.

### Создания тела Математора

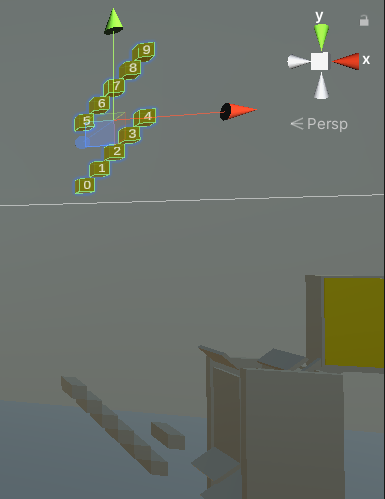
Придумаем тело Математора. Пускай он будет иметь воронку, куда будут падать кубы с числами, при помощи которых будет производиться ввод. У Математора будет экран, где будет происходить общение с пользователем. Снизу будет труба, куда будет выводится ответ кубиками.



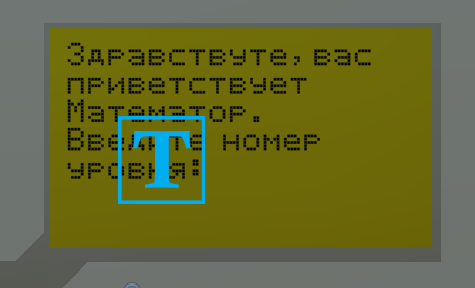
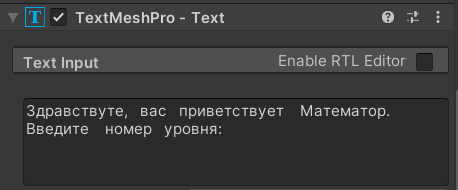
Труба слева будет возвращать кубы для выбора уровня, которые тоже нужно добавить.

### Создание кубов для ввода

Добавим кубы жёлтого цвета с цифрами, при помощи которых будут вводится номера уровней. Также пускай они падают сверху при запуске игры и приземляются на платформы.



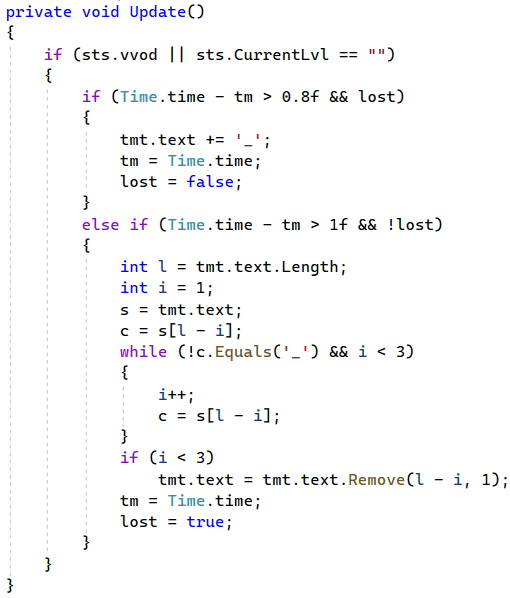
### Создание экрана Математора

Создадим объект, содержащий текст, пускай пока что он выводит приветствие на экран:

На этом экране и будет происходить взаимодействие с пользователем.

### Мигающее подчёркивание

Немаловажным элементом экрана является мигающее нижнее подчёркивание. Создадим скрипт SpotSpace, который будет осуществлять мигание нижнего подчёркивания “\_” при ожидании ввода.

Если Математор ожидает ввод числа или номера уровня, то через 0.8 секунды появляется нижнее подчёркивание, если его ещё нет (логическая переменная lost означает отсутствие такового). Также обновляется переменная времени, чтобы отсчёт времени пошёл условно заново.

Если же символ “\_” больше секунды находится на экране Математора, то при помощи цикла происходит проверка наличия подчёркивания среди последних трёх символов. Если оно находится, то сразу же удаляется с экрана. Переменная времени обновляется, как и состояние нижнего подчёркивания lost.

Таким образом подчёркивание появляется вовремя, при этом оно не дублируется в строке благодаря циклу проверки.

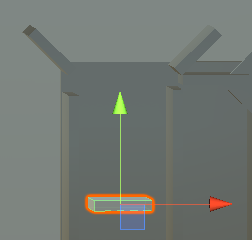
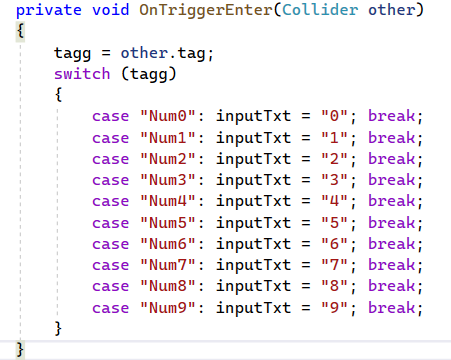
## Считывание кубами

### Перемещение кубов

Чтобы вводить при помощи кубов цифры, нужно создать скрипт, осуществляющий Drug and Drop. Теперь можно перемещать кубы.

### Считывание ввода

Для того, чтобы ввести уровень, пользователь должен бросить куб в воронку Математора. Соответственно внутри нужно поставить триггер, считывающий цифру, написанную на кубе. Для этого каждому кубу добавим свой тег, а триггер будет посылать на экран соответствующую цифру. Напишем метод OnTriggerEnter.

Тело метода OnTriggerEnter выполняется, когда какой-либо объект пересекается своим коллайдером о коллайдер объекта, которому принадлежит скрипт.

Код ниже находится в скрипте триггера. Кода куб пересекается с ним, программа считывает тег куба и по нему узнаёт, какая цифра была введена.

Попробуем бросить куб, на экран выводится число:

Теперь создадим “оперативную память”, куда будет записываться ввод пользователя. Это понадобится, когда появятся уровни. Номер уровня, состояния и прочие важные значения будут храниться в пустом объекте, в нём же будет осуществляться логика игры. Назовём этот объект Processor, а скрипт, прикреплённый к нему — Settings.

При нажатии на цифру пускай в оперативную память записываются введённые цифры. Оттуда их будет брать Processor и обрабатывать.

## Создания состояний

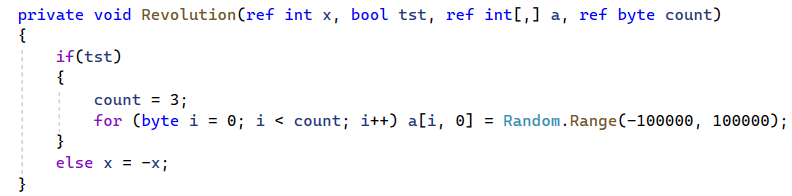
Всего в программе будет два состояния: меню и не меню, то есть уровень. В меню будет выбираться уровень при помощи кубов, а внутри состояния уровень будет ещё два состояния: тестирование и не тестирование. Для этого создадим переменные типа bool.

Когда игрок находится в меню, он не может вводить ничего на экран при помощи клавиатуры, только используя кубы для выбора уровня. Когда пользователь ввёл номер уровня и нажал Enter, состояние меняется и человек попадает на уровень. Пусть, когда он снова введёт при помощи кубов уровень, состояние игры изменится и программа вернётся в состояние “меню”. Чтобы переключиться на состояние “тестирование”, нужно будет нажать кнопку “r” (от английского “ready” — готов). Позже мы добавим все кнопки на экран, чтобы можно было играть на сенсорных устройствах или -при помощи только манипулятора-мыши.

## Создание первого уровня

Создадим отдельный скрипт, в который будем записывать алгоритмы. В нём мы будем прогонять через алгоритмы ввод и выдавать ответ, также здесь будут придумываться тесты для пользователя и проверяться правильность ответов игрока.

Напишем первый уровень с примитивным алгоритмом, который просто меняет число на противоположное по знаку.



В аргументы метода подаются введённое число x, состояние уровня (тестирование/не тестирование), массив “a”, куда будут заносится ответы и тесты для пользователя, которые будут, конечно, выводится на экран.

Немного углубив логику, исправив баги, расширив пользовательский интерфейс и добавив тем самым ещё кучу кода, уровень начинает работать.

Во время не тестирования пользователь вводит числа, которые записываются в переменную x, алгоритм использует данное число как аргумент и записывает в переменную x результат, который выводится на экран.

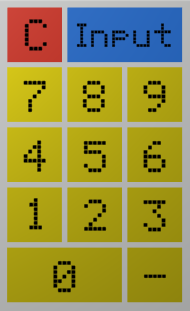
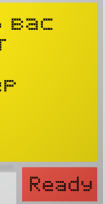
Во время тестирования массив заполняется ограниченнорандомными значениями, которые подаются пользователю как тесты. Во второй слот этого же двумерного массива “a” записываются ответы, полученные путём выполнения алгоритма со сгенерированными элементами. Когда пользователь вводит свой ответ, он сравнивается с ответом, полученным заранее и выводится либо положительный результат, либо просьба попытаться пройти уровень снова.

## Создание кнопок

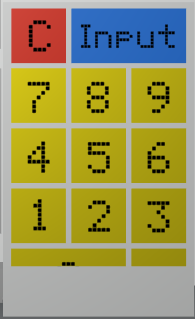
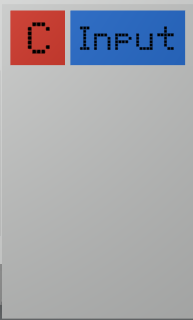
Чтобы не было много пустого места, чтобы не обделять пользователей, использующих сенсорные устройства, чтобы можно было не пользоваться клавиатурой добавим на уровень NumPad, кнопки ввода, стирания ввода, кнопку запуска тестирования.

Для этой непростой задачи при помощи простых объектов создадим NumPad и все кнопки, для каждой из которых создадим скрипт, выполняющий функцию соответствующей кнопки на клавиатуре.

На экран добавляются такие объекты:

Чтобы пользователь, когда находится в меню, вводил номера уровней при помощи жёлтых кубов, нужно добавить панель, закрывающую NumPad.

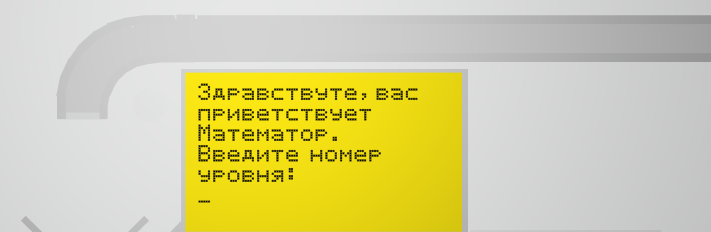
 Для этого создадим анимацию подёма и опускания панели, которые будут воспроизводиться, когда пользователь меняет состояние игры с “Меню” на ”Уровень” и обратно.

## Создание интерактивного окружения

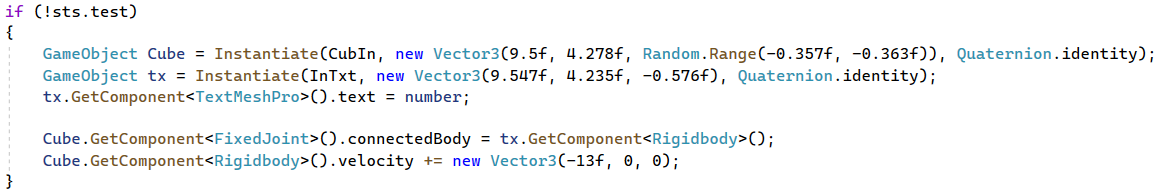
Для того, чтобы было интереснее наблюдать за процессами в Математоре, создадим инсценирование ввода кубов с цифрами, вводимые пользователем, вывод также будет при помощи кубов подаваться через нижнюю трубу, выходящую из Математора.

### Вывод ввода при помощи кубов

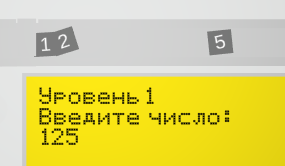
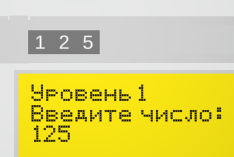
Попадать в Математор вводимые кубы будут через трубу сверху, она будет в форме крана, наподобие таких, которые стоят в раковинах и душевых.



Во избежание багов на выходе трубы установлен триггер, блокирующий ввод, когда кубы будут попадать внутрь. Кроме того, если присмотреться, то можно увидеть перегородку, которая будет останавливать летящие справа кубы. Это будет визуализация ввода, то число, которое находится в оперативной памяти. При нажатии на какой-либо символ на NumPad’е он тут же приедет в трубу.



Этот код создаёт куб с символом и даёт ему ускорение, которое позволяет кубу добраться до места.

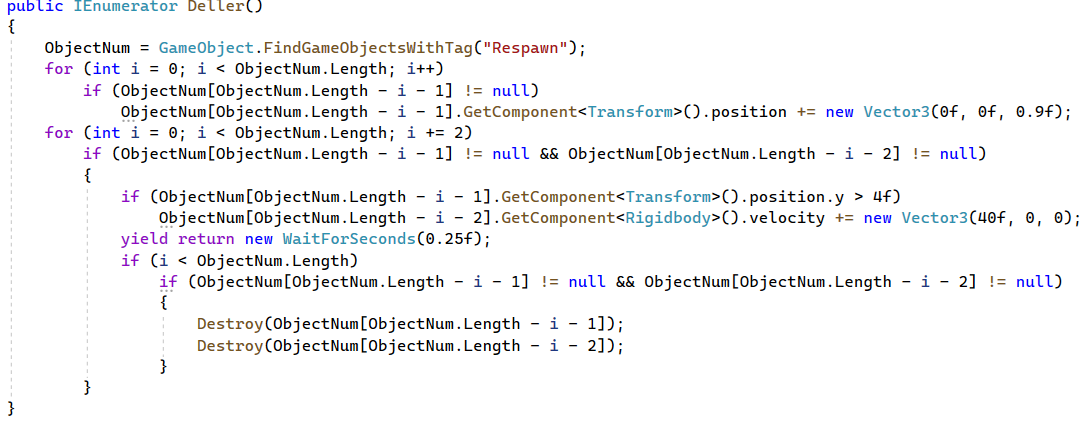


### Удаление введённых кубов

При нажатии на кнопку Backspace или же C на Нумпаде нужно убрать введённое число как из памяти и экрана, так и из трубы. Убрать из трубы можно при помощи корутины.

Корутины — это способ запускать функции, работающие параллельно в течении некоторого времени.

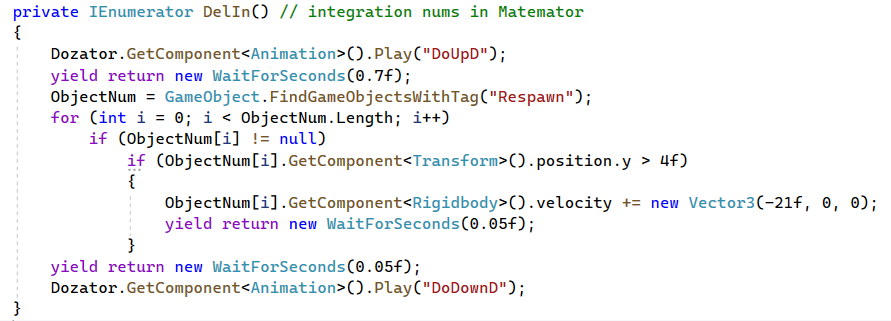
Создадим корутину Deller, удаляющую ввод:



Объектами с тегом “Respawn” являются серые кубы. Через промежутки времени, заданные в строках “yield return new” кубы будут удаляться, а до этого они будут отправлены вправо, при помощи придачи им ускорения.

### Ввод кубов в Математор

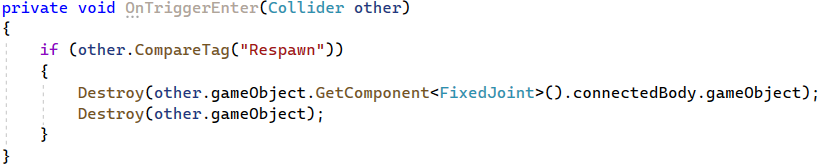
При нажатии кнопки Enter или Input на Нумпаде происходит сбрасывание кубов в воронку. Для реализации сего действия сначала нужно поднять перегородку, затем последовательно закинуть кубы в Математор.

Сделать это снова можно при помощи корутины DelIn, она по своей сути удаляет ввод, поскольку при попадании в Математор кубы будут удаляться при помощи скрипта триггера.

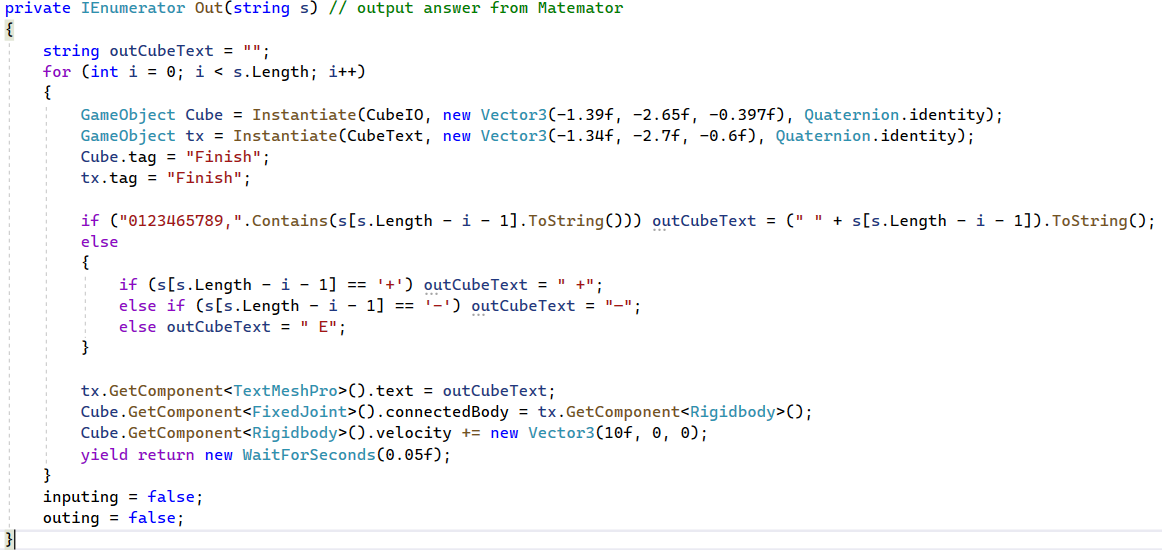
Алгоритм корутины прост:

1. Сначала поднимается перегородка при помощи анимации;
2. Через 0.7 секунд находятся кубы по тегу записываются в массив;
3. В цикле проверяется наличие куба и его положение;
4. Если куб по оси Y находится достаточно высоко, то есть в трубе, то ему придаётся ускорение, которое сталкивает его в Математор;
5. Через пять сотых секунды то же самое происходит со следующим кубом и так далее, пока не закончатся кубы;
6. В конце перегородка возвращается в исходное положение;

### Уничтожение серых кубов

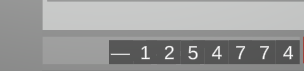
Создадим триггер, аналогичный тому, который мы создавали для считывания номера уровня, но вместо определения номера будем уничтожать сначала прикреплённый к кубу текст, а потом и сам куб:

### Вывод ответа снизу

 После загрузки кубов в Математор надо бы вывести ответ такими же кубами. Для этого нужно выводить ответ начиная с последнего разряда, а заканчивая первым или знаком “-“. Корутина получилась немного сложнее предыдущих:

Алгоритм корутины:

1. На вход корутины Out поступает ответ, записанный в string;
2. Создадим переменную типа string, в которую будет записываться цифра, которая будет на кубе;
3. В цикле, количество итераций которого равно длине ответа, создаём куб, текст на нём;
4. Установим теги “Finish” на текст и куб, чтобы в последствии можно было идентифицировать данные объекты;
5. На кубах пишем символы ответа, начиная с меньшего разряда;
6. Если ответ большой, то он содержит мантиссу, поэтому появляются два новых символа: “E” и “+”;
7. Пишем на объекте типа TextMeshPro соответствующий символ, прикрепляем его к кубу и придаём ускорение;
8. Через незначительный промежуток времени создаём следующий куб и так далее, пока ответ полностью не выведется;

Вот, что получается в итоге:

ИЛИ

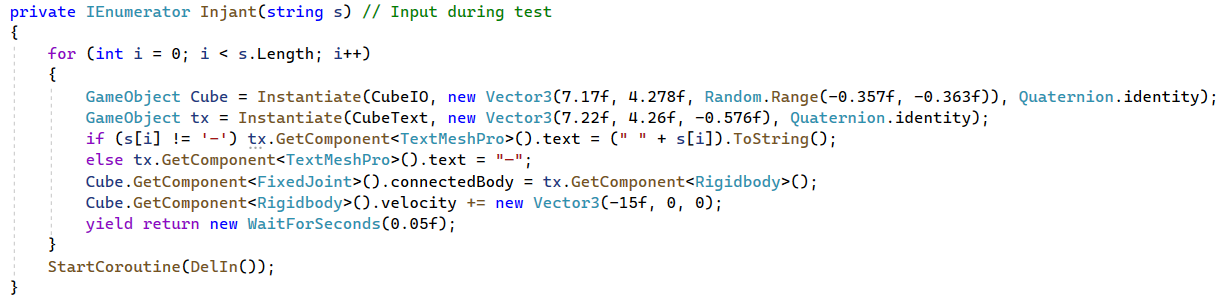
### Добавления кнопки стирания выведенных кубов

Чтобы можно было избавиться от выведенных кубов, нужно создать соответствующую кнопку. Мало ли они будут мозолить глаза.

 Заталкивать кубы обратно в Математор — уже не так интересно, поэтому осуществить удаление можно просто закрыванием “строки” с ними и удалением их. Для этого создадим анимацию поднимания и опускания “заслонки”, после чего напишем скрипт, который прикрепим к кнопке.

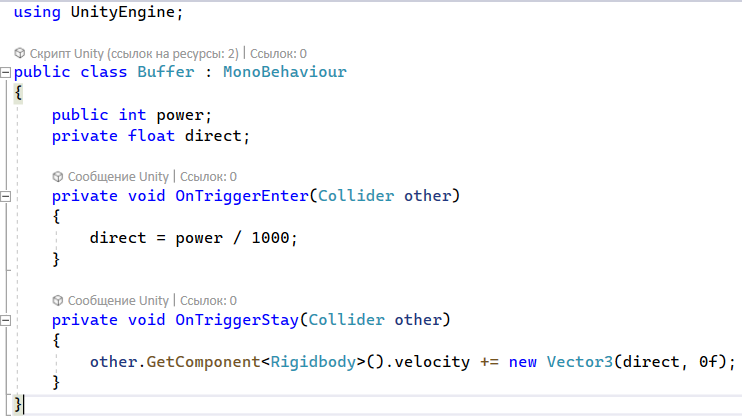


### Поведение кубов во время тестов

 Когда проводится тестирование, кубы создаются из чисел, которые сгенерированы уровнем, они вводятся похожим образом, как когда ввод осуществляет пользователь. Поэтому не будем заострять внимание на этом аспекте.

Кубы попадают в Математор, но ответ не выводится, сначала его должен ввести тестируемый. Если игрок правильно отвечает на вопрос, то генерируются кубы на вывод, которые как-бы подтверждают правильность ответа пользователя. Но если человек ошибается, то Математор не покажет ему правильный ответ, пусть пользователь сам задаст такой же вопрос Математору.

### Сохранность кубов

 Кубы можно кидать ускорением и отрыванием курсора. Это весело, но кубы могут улететь за края экрана, придётся перезапускать Математор. Для решения проблемы напишем скрипт, который будет придавать кубам ускорение в обратную сторону. Скрипт получится несложный:

На этом с кубами покончено, теперь пора реализовать некоторые функции, которые добавят удобство пользователю.

## История действий

Во время разгадывания алгоритма не очень удобно держать в уме все ответы на вопросы, данные Математором. Некоторым людям будет не удобно записывать историю своих действий на бумажку.

Чтобы облегчить жизнь пользователю создадим историю действий.

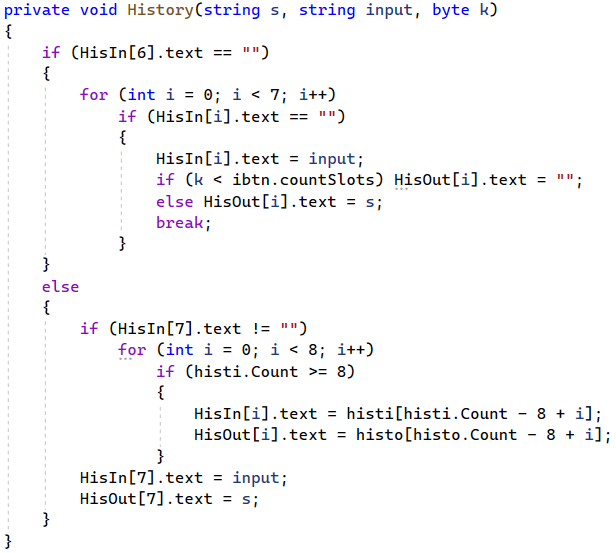
### Создание макета

Для начала установим на сцене панель, где будет содержаться история. Она должна быть достаточно широкая, чтобы в неё влезали любые выводимые значения. Представлять она из себя будет экран.

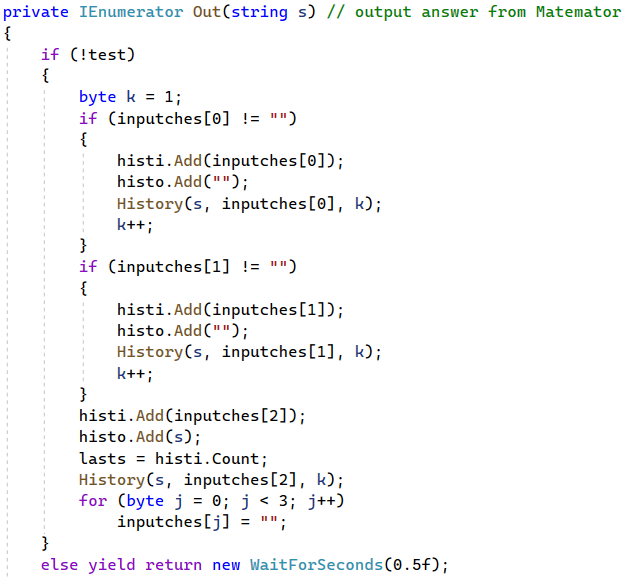
### Запись и просмотр истории

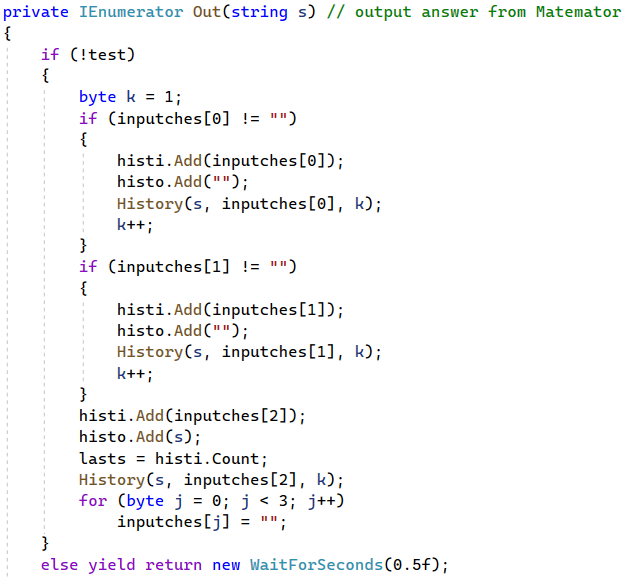
Для записи истории создадим два динамических массива: один для истории ввода, другой — вывода.



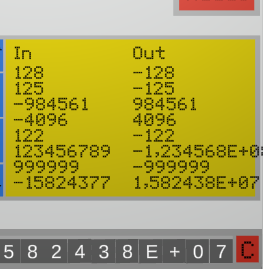
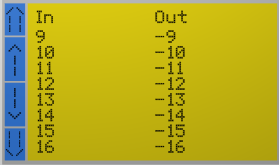
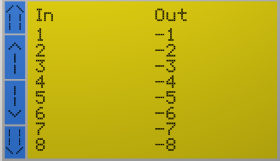
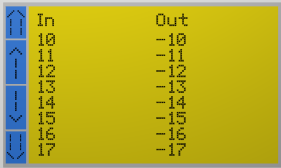
Напишем метод, заполняющий панель с историей, которая содержит 8 строк и состоит из 16 объектов TextMeshPro — 8 для ввода (In), 8 для вывода (Out).

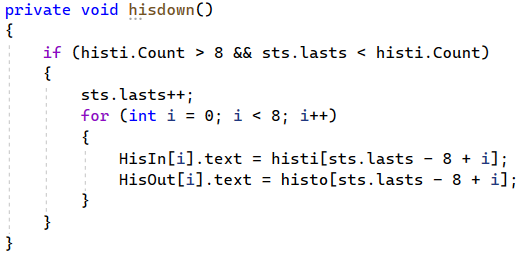
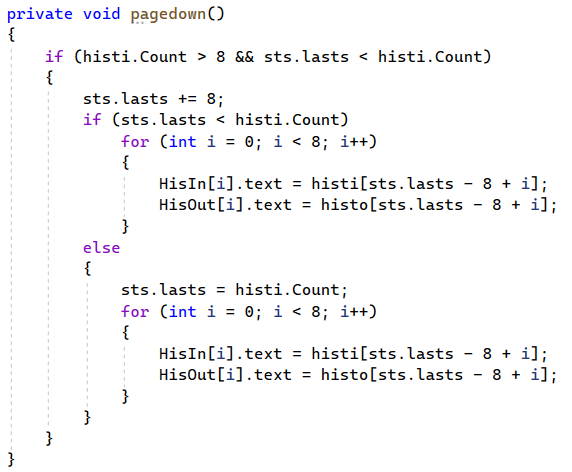
Корутину вывода ответа из Математора дополним добавлением ввода и вывода в массивы:



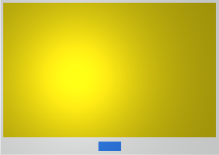
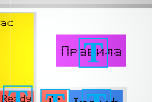


Массив inputches содержит выводы и вводы, которые после добавляются в историю, хранит он до 3-х элементов, это задел на будущее, когда появятся уровни, принимающие два или три числа в аргументы.

Теперь можно хранить вменяемые значения на дополнительном экране. Но старые данные будет нельзя просмотреть. Добавим кнопки Up, Down, PageUp и PageDown для навигации по истории, теперь можно не записывать и не запоминать тесты, даже во время тестирования.

Так выглядят методы для перемещения по истории.

## Создание свода правил игры

 Создадим экран, на котором будут написаны правила игры. По кнопке они будут исчезать, а для того, чтобы вызвать этот экран добавим специальную кнопку над NumPad’ом:

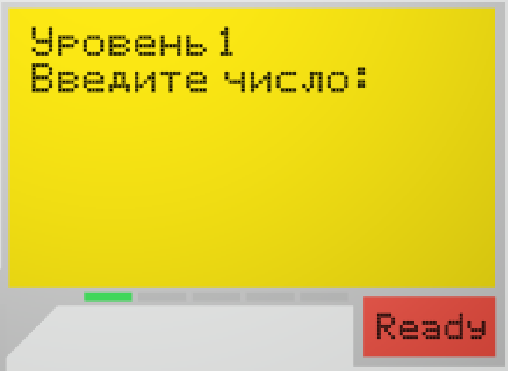
Экран будет опускаться сверху, после чего, по нажатии на кнопку, правила появятся, а после второго нажатия правила исчезнут, экран уедет вверх и уничтожится.

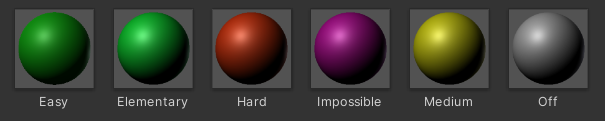
## Уровни сложности

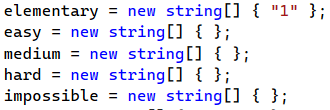
Чтобы пользователь мог сразу понять, потратит ли на прохождение уровня 2 минуты или 2 часа, добавим шкалу, показывающую уровень сложности уровня.

Диапазон значений будет делиться на пять частей:

1. Элементарный (elementary);
2. Лёгкий (easy);
3. Средний (medium);
4. Сложный (hard)
5. Невозможный (impossible);

Под экраном создадим область, состоящую из 5 частей. В зависимости от показателя сложности для уровня будет подсвечиваться соответствующий квадрат. Для каждого квадрата подберём соответствующий цвет. Например, для лёгкого уровня будет подсветка цвета 00B401.

Создадим массивы, в которых будут содержаться уровни с каждым уровнем сложности:



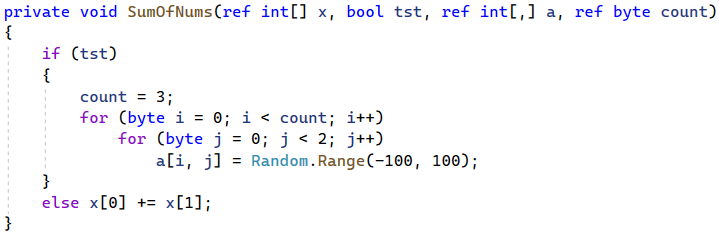
Остальные номера добавим, когда создадим новые уровни.

## Уровни

Основная часть программы — это, конечно же, сами уровни. Существует достаточно большое количество алгоритмов, которые можно использовать в Математоре.

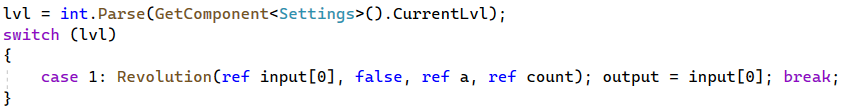
### Добавление новых уровней

Математор поддерживает в данный момент ввод до трёх аргументов, значит, количество возможных уровней огромно. Для двух аргументов самое первое, что приходит на ум — сложить эти два числа. Реализуем такой алгоритм:



На вход поступают те же аргументы, как и в уровне, где вводится только одно число, но теперь присутствует массив x, содержащий два числа-аргумента.

Таким образом создаются любые уровни в Математоре.

Выбор уровня в коде происходит при помощи оператора switch, аргументом которого является номер уровня.

### Ограниченный рандом

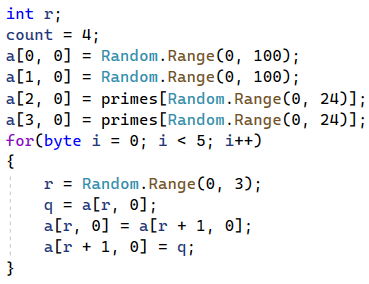
Чтобы точно убедиться, что человек полностью разгадал алгоритм, нужно дать ему такие тесты, которые охватывают все варианты выходных данных.

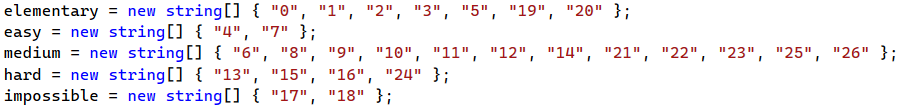
Для реализации данной идеи нужно либо ограничивать границы рандома, либо составлять алгоритм, создающий тесты.

Чтобы, к примеру, в тестах присутствовали простые числа, достаточно создать массив простых чисел primes, после чего брать рандомное число из этого массива.

После создания ограниченнорандомных чисел для теста нужно перемешать их местами, чтобы не было такого, что, например, третий и четвёртый тест всегда содержал простые числа, а первые два нет.

Данный код создаёт два рандомных числа от нуля до ста и два простых числа из массива (от двух до девяносто семи). После создания при помощи незамысловатого цикла они перемешиваются случайным образом:



С использованием данного приёма создаём пару десятков уровней с различными алгоритмами. Все уровни помещаем в конструкцию switch для их работы, а также определяем их уровень сложности от одного до пяти и записываем соответствующие значения в списки:

Таким образом это выглядит в скрипте.

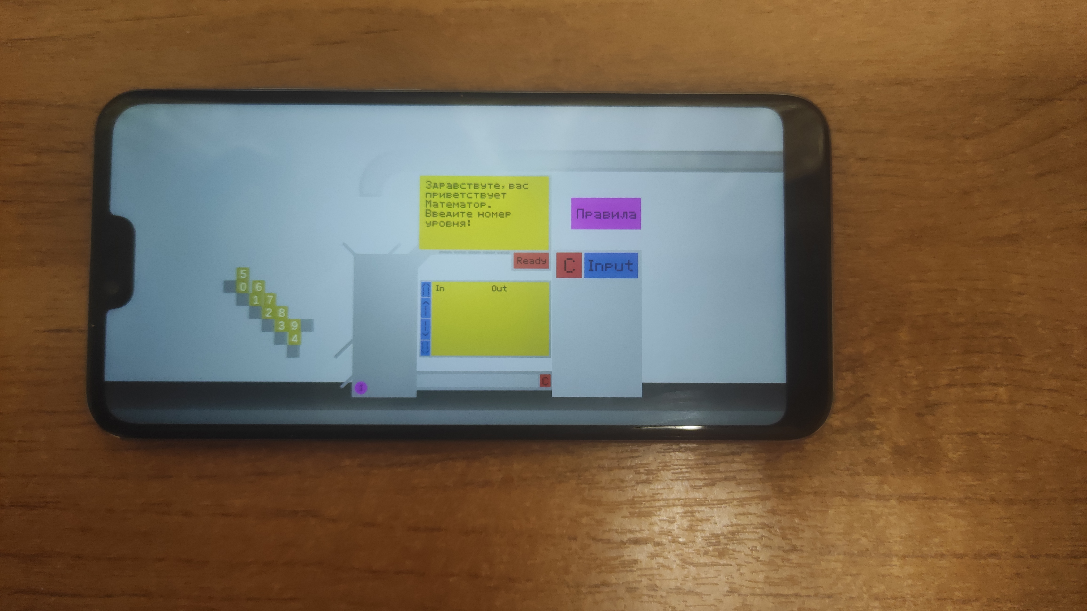
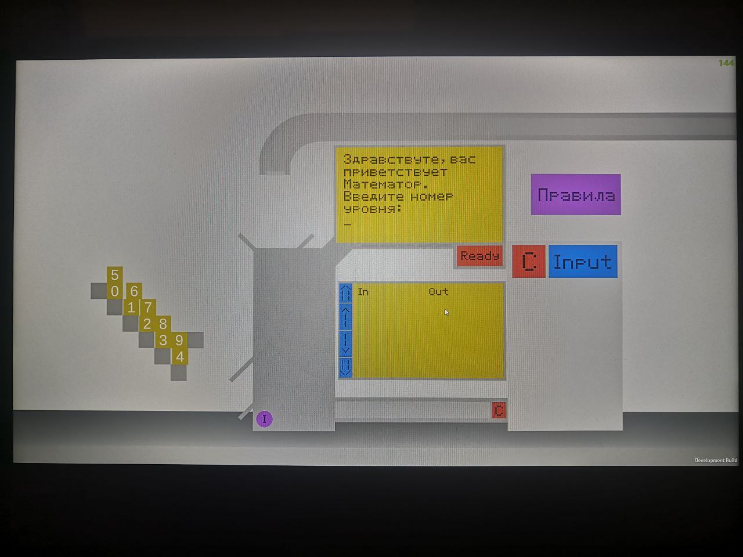
## Завершение разработки

На этом моменте можно завершить разработку Математора.

В общей сложности на изучение основ Unity, реализацию идеи и исправление неисправностей ушло 17 дней. В среднем работа велась по 6 часов в сутки. Путём несложных вычислений можно подсчитать, что работа была выполнена примерно за 102 часа.

# Тестирование

Для проведения тестирования совершим сборку приложения на две популярные операционные системы: MS Windows и Android. Кроссплатформенность Unity играет нам на руку.

Как мы видим, всё работает исправно:

Распространим приложение среди некоторых пользователей, чтобы составить статистику о том, какие уровни окажутся наиболее лёгкими, а какие вызовут трудности у пользователей. Пять человек приняли участие в эксперименте.

Запишем в таблицу результаты тестирования:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер уровня | Количество участников | Количество прошедших | Процент прохождения | Сложность |
| 0 | 5 | 5 | 100% | 1 |
| 1 | 5 | 5 | 100% | 1 |
| 2 | 5 | 5 | 100% | 1 |
| 3 | 5 | 5 | 100% | 1 |
| 4 | 5 | 5 | 100% | 2 |
| 5 | 5 | 5 | 100% | 1 |
| 6 | 5 | 5 | 100% | 3 |
| 7 | 5 | 5 | 100% | 2 |
| 8 | 5 | 4 | 80% | 3 |
| 9 | 5 | 4 | 80% | 3 |
| 10 | 5 | 5 | 100% | 3 |
| 11 | 5 | 4 | 80% | 3 |
| 12 | 5 | 5 | 100% | 3 |
| 13 | 5 | 5 | 100% | 4 |
| 14 | 5 | 4 | 80% | 3 |
| 15 | 5 | 3 | 60% | 4 |
| 16 | 5 | 2 | 40% | 4 |
| 17 | 5 | 1 | 20% | 5 |
| 18 | 3 | 1 | 33% | 5 |
| 19 | 3 | 3 | 100% | 1 |
| 20 | 3 | 3 | 100% | 1 |
| 21 | 3 | 3 | 100% | 3 |
| 22 | 3 | 2 | 67% | 3 |
| 23 | 2 | 2 | 100% | 3 |
| 24 | 2 | 1 | 50% | 4 |
| 25 | 2 | 1 | 50% | 3 |
| 26 | 2 | 2 | 100% | 3 |

Результаты примерно совпадают с оценкой сложности уровня, но бывают и погрешности, например, в 13 уровне.

Уровни с номерами 17 и 18 содержат очень сложные алгоритмы для разгадывания. Один человек их смог пройти, выучив все ответы, но не разгадав алгоритм.

После попыток пройти “невозможные” уровни два человека были утомлены и не смогли продолжить эксперимент.

При помощи программы удалось выявить людей с хорошим мышлением и логикой, также стрессоустойчивых, которые не сдавались, смогли пройти все уровни, кроме 17 и 18. Эти два уровня помогли найти самого настойчивого и хитрого игрока.

# Вывод

Разработанный алгоритмический тренажёр Математор справился со своей задачей — смог оценить мышление участников эксперимента и их склонность к математике, при помощи алгоритмов. Помимо этого, у участников были выявлены такие качества как упорство и настойчивость, а также хитрость.

# Источники, использованные при написании реферата:

https://sites.google.com/site/urokalgoritm/cto-takoe-algoritm

<https://habr.com/ru/post/216185/>

https://docs.unity3d.com/ScriptReference/