

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ**

**Государственное бюджетное профессиональное образовательное**

**учреждение Московской области**

**«Люберецкий техникум имени Героя Советского Союза,**

**лётчика-космонавта Ю.А. Гагарина»**

**КУРСОВАЯ РАБОТА**

**по МДК.01.01 «Разработка программных модулей»**

**по МДК.01.03 «Разработка мобильных приложений»**

специальность

09.02.07 «Информационные системы и программирование»

**Тема:** «Основы программирования на языке GML»

Группа 185

Выполнил/а/ студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Якушев Александр Сергеевич

(подпись) (ФИО полностью)

Руководитель Жирнова Юлия Витальевна

(подпись) (ФИО полностью)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Люберцы-2022 год

Оглавление

[Введение 3](#_Toc95394299)

[Актуальность 4](#_Toc95394300)

[Теория 4](#_Toc95394301)

[Особенности IDE 5](#_Toc95394302)

[Постановка задачи 7](#_Toc95394303)

[Изучение программных средств 7](#_Toc95394304)

[Основная структура языка 7](#_Toc95394305)

[Игровой ввод 8](#_Toc95394306)

# Введение

GameMaker Language - это письменный язык сценариев, который использует функции, переменные и другие конструкции кодирования для создания логики игр.

Создатель языка Марк Овермарс большею частью вдохновлялся языком C++, но также присутствует некоторая схожесть синтаксиса с Pascal, Assembler. Однако, хоть язык и похож на другие языки программирования, это вовсе не означает, что для работы на движке Game Maker не нужно изучать GML, но знание C++ заметно упростит изучение.

Целью курсовой работы является изучения языка GML и создание небольшого приложения на игровом движке GMS 2, в котором получится использовать все полученные знания.

Для достижения данной цели нам потребуется:

1. Анализ и изучение основных функций языка.
2. Использование основных функций в приложении.

# Актуальность

Интерпретируемый язык программирования Game Maker Language (GML) был специально разработан для игрового движка Game Maker 1999 года выпуска, впоследствии также используемый и в следующих частях: GameMaker Studio (GMS) 2012 года выпуска и GameMaker Studio 2 (GMS 2) 2017 года выпуска. Данный язык может использоваться разработчиками исключительно в разработке игр, используя при этом только игровые движки Game Maker’a.

# Теория

Изначально поддержка языка была внедрена в Game Maker для дополнения системы кнопочных событий, однако позже все кнопочные события были включены в GML, позволяя программисту избежать использования кнопочных функций. GML очень сильно связан со средой Game Maker. Game Maker организован так, чтобы не было необходимости программирования вручную таких вещей, как управление событиями, дизайн уровней и настройка объектов.

В Game Maker совокупность кнопочных событий образует библиотеку. В интерфейсе программы библиотеки отображаются как закладки, в которых находятся различные иконки событий. Каждое такое событие — это GML-скрипт или функция, которую пользователь может использовать в игре. В поставку Game Maker входят несколько стандартных библиотек, которые содержат основные события, используемые в большинстве игр. Также существует возможность создавать свои собственные библиотеки, используя [Library Maker](https://ru.wikipedia.org/w/index.php?title=Library_Maker&action=edit&redlink=1" \o "Library Maker (страница отсутствует)). В GMS2 встроен механизм конвертации действий, заданных «кнопками», в код GML и обратно, что позволяет новичкам быстрее перейти на GML и улучшает понимание того, как работают стандартные действия.

# [Особенности IDE](https://docs.google.com/document/d/1tZ-twA5VU-YKM5sSyGWBFLZ0T9qE171S/edit#heading=h.2et92p0)

Game Maker содержит обширную библиотеку встроенных функций для обеспечения основной функциональности. Программист может создавать свои собственные скрипты, которые вызываются точно таким же способом, как и функции.

Обычно, в GML не нужно предварительно объявлять переменную, как это делается в некоторых других языках. Переменная создается автоматически, сразу после присваивания ей какого-либо значения.

Присваивание переменных — это всегда оператор, и поэтому не может быть использован в выражениях. Например, следующая строка всегда генерировала бы ошибку потому, что она бы вычисляла вложенное выражение как true или false, а затем сравнивала бы булевый результат со строкой «Yes» (неправильное сравнение).

Стоит помнить, что знак равно «=» является оператором присвоения и булевым оператором сравнения в выражениях, тогда как в С++ в выражениях пишут двойной знак «==». Тем не менее, двойной знак равно «==» будет правильно интерпретирован в случае использования его в выражениях. Использование такого знака в качестве оператора присваивания вызовет ошибку исполнения.

В Game Maker есть множество встроенных переменных и [констант](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0). Каждый экземпляр [объекта](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) содержит множество локальных переменных, например «х» и «у». Также существует несколько встроенных глобальных переменных, например «score». Эти переменные существуют независимо от экземпляров объектов. Эти переменные не содержат приставку «global.», в отличие от глобальных переменных, указанных программистом. Одномерные и двумерные [массивы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) также поддерживаются.

В GML поддерживаются следующие типы данных:

* **string** (строка) - последовательность символов, заключенных в одинарные или двойные кавычки (начиная с GMS2 следует использовать двойные кавычки).
* **real** (число) - целое или с плавающей запятой. Хотя все значения, созданные в GML, хранятся как числа с плавающей запятой двойной точности, для работы с расширениями можно использовать и другие типы
* **array** (массив) - переменная, использующая индексы для доступа к элементам. Могут содержать любые данные - числа, строки, другие массивы, дескрипторы других структур данных и т.п. Их также можно передать как параметр в функцию, и они могут быть возвращены функцией как результат.
* **boolean** - может принимать значения **true** или **false**. Имейте в виду, что в настоящее время GML не поддерживает «настоящие» булевы значения и на самом деле принимает как **false** любые числа меньше 0.5, а всё, что равно или больше - как **true**.
* **pointer** (указатель) - указатель на область памяти. Используется в некоторых специфических функциях вроде **buffer\_get\_address()** и др.
* **enum** (перечисление) - заданная пользователем коллекция констант, хранящихся в переменной.
* **undefined** (не задано) - специальное значение, возвращаемое в случаях, когда запрашиваемые данные не найдены.

# Постановка задачи

Разберем уникальные особенности и функции языка GML, которые отсутствуют в большинстве других языках программирования. Изучим теоретическую часть этих функций и рассмотрим как их можно использовать практически.

# Изучение программных средств

Игровые движки Game Maker в котором используется GML можно скачать с официального сайта разработчика. Старые версии Game Maker и GameMaker Studio не доступны для скачивания на сайте разработчика, они считаются устаревшими. На сайте для скачивания доступна последняя версия GameMaker Studio 2.

На сайте необходимо будет зарегистрироваться и скачать бесплатную демоверсию. Демоверсия доступна на один месяц, далее можно продлить на один год или приобрести безлимитную версию, цены указаны на официальном сайте. Программа также доступна для покупки в Steam, но там все равно требуется регистрация на официальном сайте производителя.



Рисунок 1. GameMaker Studio 2

# Основная структура языка

Код пишется блоками, и типичный блок кода состоит из набора инструкций, называемыхоператорами, которые затем интерпретируются GameMaker Studio 2 и используются для того, чтобы что-то произошло в вашей игре. Фактическая структура программы может сильно варьироваться в зависимости от используемых функций, но в общих чертах она выглядит следующим образом:

<statement>;  
<statement>;

Утверждения могут разделяться символом «;» для предотвращения ошибок при объявлении переменной и для поддержания чистоты и порядка в коде. Основное отличие GML от других языков в том, что символ «;» вовсе не обязателен для правильной работы кода, но вовсе не помешает использовать знак для профилактики. Стоит уточнить, что некоторые функции будут работать неправильно, если их не разделить данным символом.

Также можно "сгруппировать" утверждения в блок с помощью фигурных скобок {}, чтобы они выполнялись вместе, как в следующем условном примере:

if (<conditional>)  
{  
    <statement>;  
    <statement>;  
    ...  
}

Таким образом можно сказать, что по своей структуре GML не отличается от большинства других языков программирования.

# Ввод

Для того чтобы игра стала игрой, необходимо чтобы игрок мог с ней взаимодействовать. Это достигается различными способами: с помощью клавиатуры и мыши на компьютерах, с помощью сенсорного экрана и наклона устройства на мобильных, или с помощью геймпада на всех платформах.

Рассмотрим основные функции ввода с помощью клавиатуры.

* **keyboard\_check(key)** – c помощью этой функции можно проверить, удерживается ли клавиша нажатой или нет.
* **keyboard\_check\_pressed(key)** –с помощью этой функции можно проверить, была ли нажата клавиша или нет.
* **keyboard\_check\_released(key)** – спомощью этой функции можно проверить, была ли клавиша отпущена или нет.

В скобках следует указать соответствующую константу обозначающую нужную клавишу. Таблица данных констант представлена ниже:

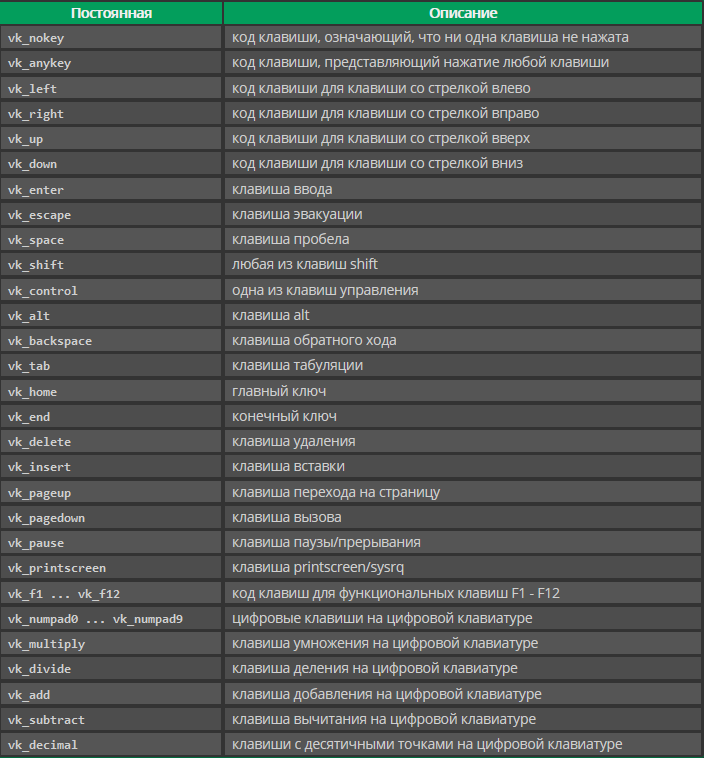


Рисунок 2. Таблица констант для ввода на клавиатуре

Для того чтобы произвести ввод символа любой другой клавиши, не указанной в таблице, необходимо использовать функцию:

ord()

Код ниже проверит была ли нажата клавиша «R» и если да, то выведет на экран окно с сообщением «hello world»:

if keyboard\_check\_pressed(ord("L"))

{

show\_message("hello world")

}

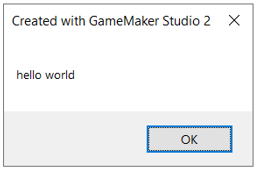


Рисунок 3. Окно с сообщением

# Переменные

Как и любой другой язык программирования, GML использует переменные в качестве основной единицы для большинства операций программирования. Переменные используются для хранения информации в памяти устройства для последующего (или мгновенного) использования, и им присваивается имя, чтобы была возможность ссылаться на них во время выполнения и в функциях сценария. Переменная в GML может хранить множество различных **типы данных**, например, вещественное и целое число, строку или булевое число, а также другие вещи:

var \_num = 126.4545;

var \_str = "Hello World";

new\_num = \_num \* 100;

my\_string = \_str + " I said";

Во многих языках программирования необходимо создать переменную "присвоением", прежде чем ее использовать. В основном это означает, что вы сообщаете компьютеру имя, которое хотите использовать для переменной, и присваиваете ей начальное значение. Затем переменной отводится место в памяти для хранения значения или выполнения операций над ним. Присвоение переменной имеет вид:

<variable> = <expression>;

Выражение может быть простым значением, но может быть и более сложным, поэтому вместо присвоения значения переменной можно также добавить значение к текущему значению переменной, сделать это можно несколькими способами:

a = 100;

b = 150;

a += b;

a = a + b;

Аналогично можно вычитать с помощью **-=**, умножать с помощью **\*=**, делить с помощью **/=**. Также можно прибавить или отнять *единицу* от значения, используя **++**, **--**.

В программировании с помощью GameMaker Studio 2 существует четыре основных категории переменных, каждая из них имеет свою **область применения.**

**1) Переменные экземпляра –** переменная **экземпляра** создается внутри экземпляра объекта и считается уникальной для этого экземпляра - т.е. многие экземпляры одного и того же объекта могут иметь одну и ту же переменную, но каждая переменная может иметь разное значение, поскольку она уникальна для каждого экземпляра. каждого экземпляра.

Чтобы использовать данную переменную необходимо просто объявить ее в нужном экземпляре:

name = “Ivan”

В GML есть встроенные переменные экземпляра, которые относятся ко все созданным экземплярам. После создания экземпляра значения этих переменных уникальны для этого и только этого экземпляра. Некоторые из этих переменных будут использоваться часто, например x и y, в то время как другие нужны меньше, но в целом они также очень полезны, по возможности следует использовать их, а не создавать отдельные переменные.

Рассмотрим встроенные переменные, которые связаны с положением и перемещением экземпляра:

* **direction** – может быть использовано для задания направления движения экземпляра, если параметр **speed** отличен от нуля. Направления рассчитывается как 0° - вправо, 90° - вверх, 180° - влево и 270° - вниз.
* **friction** - может быть использовано для замедления экземпляра со временем, если параметр **speed** отличен от нуля. Оно работает просто, вычитая определенную величину из скорости на каждом шаге, пока скорость объекта не станет равной 0.
* **hspeed** – пределяет скорость горизонтального перемещения (вдоль оси x) экземпляра в пикселях за шаг.
* **vspeed** – определяет скорость вертикального перемещения (по оси y) экземпляра в пикселях за шаг.
* **speed** – задает скорость экземпляру, от показателя direction зависит направление экземпляра. По умолчанию оно равно нулю. Установка данного параметра изменит значения переменных vspeed и hspeed. параметров speed и/или direction, также изменит значения переменных hspeed и vspeed,
* **xstart** – переменная хранит начальную x-позицию экземпляра при его первом создании в комнате. Это не переменная только для чтения, и ее можно как установить, так и прочитать.
* **ystart** – переменная хранит начальное положение y экземпляра при его первом создании в комнате. Это не переменная только для чтения, и ее можно как установить, так и прочитать.
* **x** – горизонтальное положение в текущей комнате, измеряемое в пикселях.
* **y** – вертикальное положение в текущей комнате, измеряемое в пикселях.

**2) Местные (локальные) переменные** – это переменная, которую мы создаем только для определенного **события** или **функции**, а затем отбрасываем по завершении события или функции. Если она создается в пользовательской функции, то локальная переменная доступна только для функции, а затем отбрасывается по ее завершении. Переменные занимают место в памяти, и может оказаться, что их использование потребуется только для одной операции или функции, в таком случае нужно, чтобы они находились в памяти только в течение того короткого времени, когда они используются. Это позволяет поддерживать чистоту и порядок в кодовой базе, а также оптимизировать пространство памяти для тех вещей, которые действительно в ней нуждаются.

Чтобы объявить локальную переменную, необходимо использовать оператор **var** следующим образом:

var i = 0;

или

var b = 1, c = 11.3;

или

var \_min, \_num;

\_min = 4;

\_num = 100;

**3) Константы -** это тип переменной, которая устанавливается один раз в начале игры и затем никогда не изменяется. Это делает их идеальными для хранения значений, которые используются на протяжении всей игры для идентификации специальных данных.

Есть два типа пользовательских констант: **макросы** и **перечисления.**

**3.1) Макросы**

Хоть макросы и не совсем переменные, они похожи на них по способу использования, т.е. это именованные значения, которые можно использовать в коде для замены с жестким кодом значений. По сути, макрос - это именованная переменная, которая содержит постоянное единственное значение любого [типа данных](https://manual-ru.yoyogames.com/GameMaker_Language/GML_Overview/Data_Types.htm). Вы можете определить свои собственные макросы с помощью [редактора сценариев](https://manual-ru.yoyogames.com/GameMaker_Language/GML_Overview/Script_Functions.htm) и затем использовать их в коде как обычные переменные, с той лишь разницей, что они не могут быть изменены в игре или в коде котором были вызваны.

Синтаксическая структура макроса выглядит следующим образом:

#macro <variable> <value>

Использование макросов может значительно упростить написание кода, например, если вы часто используете одну функцию, то вы можете присвоить эту функцию к макросу и запись данной функции будет значительно короче:

#macro PRESSED\_R keyboard\_check\_pressed(ord("L"))

Данный код присвоит функцию к константе, с более короткой записью, в этом случае можно обновить код с выводом на экран окна с сообщением:

if PRESSED\_L

{

show\_message("hello world")

}

Если вы решите присвоить значение константы к другому значению, то это вызовет критическую ошибку.

**3.2) Enums (перечисление)**

Перечисление - это "перечислитель", оно позволяет создать свой собственный ограниченный тип данных со списком постоянных значений, они имеют следующую структуру:

enum <variable>

{

<constant> [= <value>],

<constant> [= <value>],

…

}

Например можно создать п перечисление для цветов радуги и присвоить ему различные константы и значения по умолчанию:

enum <rainbow>

{

red,  
orange,  
yellow,  
green,  
blue,  
indigo,  
violet

}

Элементы перечисления могут быть только целыми числами или выражением с предыдущими перечислениями, которые оцениваются в целое число. По умолчанию нумеруются от 0 сверху, поэтому в примере, приведенном выше, по умолчанию будет red = 0, orange = 1, yellow = 2 и так далее.

**4) Глобальные переменные**

Основное описание **глобальной** переменной - это переменная, которая, будучи объявленной, не принадлежит ни одному конкретному экземпляру, но к ней могут обращаться все. Как и [локальные переменные](https://manual-ru.yoyogames.com/GameMaker_Language/GML_Overview/Variables/Local_Variables.htm), глобальные переменные должны быть объявлены с использованием идентификатора, но в отличие от локальной переменной, глобальная переменная остается в памяти до конца игры. Таким образом, можно создать глобальную переменную для отслеживания (например) количества жизней, которые есть у игрока, а затем просто обновлять эту переменную в разные моменты игры, формировать любой экземпляр или функцию в любое время. По сути, глобальная переменная не принадлежит какому-либо конкретному экземпляру и может быть доступна, изменена и использована всеми экземплярами, а любые изменения, внесенные в переменную, являются "глобальными", так как все экземпляры, использующие переменную, будут затронуты этим изменением.

Для того чтобы объявить глобальную переменную сначала нужно написать «global», затем «.» и уже после написать название вашей переменной:

global.hp = 5;

После создания глобальной переменной hp любой экземпляр или функция может использовать и изменять эту переменную любым способом, а все остальные экземпляры будут "видеть" это.

С помощью глобальных переменных можно изменять значения и видеть, как эти изменения отражаются во всех экземплярах объектов, которые ссылаются на эту переменную. Как и в случае с локальными переменными, необходимо следить за тем, чтобы глобальные переменные не назывались так же, как и все переменные экземпляра, поскольку это может вызвать проблемы и привести к появлению ошибок в играх из-за дублирования переменных.

# Использование встроенных переменных

Используем изученные встроенные переменные на практике.

Для начала напишем макросы на нажатие клавиш, для того чтобы можно было экономить время в дальнейшем:

#macro KEY\_LEFT keyboard\_check(vk\_left) or keyboard\_check(ord("A"))

#macro KEY\_RIGHT keyboard\_check(vk\_right) or keyboard\_check(ord("D"))

#macro KEY\_UP keyboard\_check(vk\_up) or keyboard\_check(ord("W"))

#macro KEY\_DOWN keyboard\_check(vk\_down) or keyboard\_check(ord("S"))

#macro KEY\_SPACE keyboard\_check\_pressed(vk\_space)

#macro KEY\_ENTER keyboard\_check\_pressed(vk\_enter)

Макросы, которые указаны выше сократят запись кода и оптимизирует его, делая более гибким и понятным.

С помощью логической операции «or» присваиваем макросу сразу две клавиши.

Макросы можно записать где угодно, но желательно создать для них отдельный скрипт, чтобы в дальнейшем можно было бы быстро воспользоваться ими и вписать новые.

Теперь создадим новый скрипт и впишем туда код, используя встроенные переменные «vspeed» и «hspeed».

if KEY\_RIGHT {

hspeed = 5

}

if KEY\_LEFT {

hspeed = -5

}

if KEY\_UP {

vspeed = -5

}

if KEY\_DOWN {

vspeed = 5

}

Экземпляром, который использует данный код, можно будет управлять используя клавиши, которые были указаны и записаны в макрос. Нажимая на клавишу, экземпляр получит вертикальную и/или горизонтальную скорость, с помощью чего он начнет движение в сторону, которая соответствует нажатой клавише. Скорость перемещения можно изменить.

В данном коде не описано обнуление скорости, а соответственно передвижение экземпляра будет бесконечным, потому что он получил скорость «5» и получает ее постоянно. Следует записать код, в котором экземпляр будет уменьшать свою скорость до нуля, для того чтобы остановиться. Снизу указаны два варианта кода, с помощью которых можно обнулить значение скорости:

hspeed = 0

vspeed = 0

и второй вариант:

if abs(speed) > 0 {

friction = 0.3

}

else { friction = 0 }

Данные коды необходимо написать перед тем, как будет описано движение экземпляра.

Первый вариант сразу обнуляет скорость экземпляра, из-за чего он резко прекратит движение. При втором варианте используется встроенная переменная «friction», благодаря этому экземпляр будет «скользить» перед тем как окончательно остановиться.

Теперь напишем код, который позволит ему вернуться на позицию, с которой он начал свое движение:

if KEY\_SPACE { x = xstart }

if KEY\_ENTER { y = ystart }

Нажав на «space» экземпляр вернется свою изначальную позицию по x, нажав на «enter» экземпляр вернется на свою изначальную позицию по y. Соответственно нажав на «space» и «enter» экземпляр вернется на свою изначальную позицию, с которой он начал свое движение.

С помощью встроенных переменных х и у можно переместить экземпляр в любой позицию, например:

if keyboard\_check\_pressed(ord("1"))

{

x = 1

y = 1

}

Этот код позволит экземпляру получить позицию 1 пиксель по х и 1 пиксель по у, если была нажата соответствующая кнопка. В данном случае это цифра 1.

# Выводы

В процессе выполнения курсовой работы была изучена основная структура языка, а также ввод данных и переменные, которые были использованы практически. Были решены поставленные залачи:

1. Проанализированы и изучены основные функции языка.
2. Функции были использованы в приложении.

# Список литературы

1. https://manualru.yoyogames.com/GameMaker\_Language/GML\_Reference/GML\_Reference.htm
2. https://ru.wikipedia.org/wiki/Game\_Maker\_Language