GML

Game Maker Language (GML) — это интерпретируемый язык программирования, разработанный для использования вместе с программой для разработки компьютерных игр называемой Game Maker. Изначально поддержка языка была внедрена в Game Maker Марком Овермарсом 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. Существует заблуждение, что GML поддерживает вставки фрагментов кода на других языках, таких как Pascal, Assembler или C++. Заблуждение возникло из-за частичной схожести синтаксиса GML с Pascal и C++. (Например, оператор «&&» может быть заменен на «and»).

1. Библиотеки.

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. 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.

2. Синтаксис и семантика GML.

GML структурно похож на язык C своими блоками кода, вызовами функций, присваиванием переменных, синтаксисом операторов и так далее. GML различает операторы и выражения. Например, g $<$ 1; не является правильным оператором и вызовет ошибку. Также, 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): if ((answer = get_string("Yes or No?)) == "Yes") Стоит помнить, что знак равно «=» является оператором присвоения и булевым оператором сравнения в выражениях, тогда как в C++ 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. GML также поддерживает операторы инкремента: g++; // поддерживается как постфиксная, так и префиксная запись и g += 1; то же самое, что и g = g + 1; Также существуют операторы: -=, *=, /=, |=, &= и ≙. Начиная с GMS2 введена поддержка тернарного оператора ?:. Операторы в GML могут быть разделены точкой с запятой, однако это не является обязательным условием (хотя и может привести к ошибке в некоторых специфичных случаях). Функции 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. Функции рисования в Game Maker используют Direct3D API. При необходимости Game Maker также позволяет вызывать нативный код платформы посредством расширений (DLL на Windows, Java на Android, JS на HTML5 и т.п).

3. Переменные.

Обычно, GML не нужно предварительно объявлять переменную, как это делается в некоторых других языках. Переменная создается автоматически, сразу после присваивания ей какого-либо значения: foo = "bar"; В Game Maker есть множество встроенных переменных и констант. Каждый экземпляр объекта содержит множество локальных переменных, например «x» и «y». 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 также поддерживаются.

4. Структуры данных.

В GML есть функции для создания и редактирования структур данных шести типов: стек, очередь, список, карта (ассоциативный массив), приоритетная очередь и сетка. К сетке, списку и карте также есть возможность доступа посредством аксессоров, предоставляющих синтаксис, подобный массивам: var value = list[| 0]; // вместо ds_list_find_value(list, 0) map[? "name"] = "Username"; // вместо ds_map_add(map, "name "Username") var value = map[? "name"]; // вместо ds_map_find_value(map, "name");

5. Типы.

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.

6. Область действия переменных.

Хотя GML и можно рассматривать как объектно-ориентированный язык, природа объектов и экземпляров объектов в Game Maker создает некоторые важные отличия в способе разграничения переменных. Существует два типа локальности: локальность в объекте и локальность в скрипте (или другом куске кода, содержащемся в отдельном контейнере). То, что переменная является локальной для экземпляра объекта, означает, что переменная привязана к конкретному экземпляру объекта и извне этого экземпляра может быть использована только с приставкой, определяющей этот экземпляр; то, что переменная является локальной для скрипта, означает, что эта переменная может быть использована только в этом скрипте (и уничтожается после окончания скрипта). Далее термин «локальный» будет означать локальность в объекте. По умолчанию, переменная локальна для объекта, но не локальна для скрипта, в котором она используется. Для того чтобы сделать переменную доступной всем экземплярам объектов, она может быть определена через глобальное пространство имен: global.foo = "bar"; Также существует возможность объявлять глобальные переменные используя ключевое слово globalvar: globalvar foo, bar; Но такого способа следует избегать, так как это может легко привести к сложно выявляемым ошибкам, из-за пересечения областей видимости переменных (то же самое рекомендуют и непосредственно разработчики GMS; более того, возможно, что в будущем это ключевое слово будет полностью убрано из языка - в данный момент оно оставлено исключительно из соображений обратной совместимости). Для того, чтобы сделать переменную локальной для скрипта, её нужно определять так: var foo, bar; Областью видимости локальной переменной является скрипт, внутри которого она объявлена. Это подразумевает, что при переключении контекста (с использованием with) она по-прежнему будет доступна. Например: var foo = "bar"; with other show_message(foo); // переменная foo доступна Доступ к локальным переменным объекта можно получить, используя идентификатор экземпляра объекта как приставку instance.varname но, тем не менее, таким образом невозможно получить локальные переменные одного скрипта из другого, пока они не передаются как параметры функции. Текущее пространство имен объекта может быть изменено с помощью конструкции «with». Например, следующий скрипт, если его поместить в событие столк-

новения, уничтожит другой экземпляр объекта, вовлеченный в это событие (заметим, что в событии столкновения Game Maker автоматически устанавливает переменную other на второй экземпляр объекта, с которым произошло столкновение): with other instance_destroy();

7. Распределение памяти.

GML автоматически распределяет память под переменные на ходу, и использует динамические типы, поэтому присваивание переменным значения различных типов также возможно. Например, сначала можно создать целочисленную переменную, а потом изменить её на строковую: intNumber = 1; intNumber = "Эта переменная теперь содержит строку"; В GML нет специального функционала, позволяющего освобождать память, занятую под переменную, однако при необходимости можно присваивать переменной новое значение, меньшего размера. Например, если у вас есть переменная, в которой хранится большой текст, то, присвоив переменной значение пустой строки, можно добиться высвобождения памяти. То же самое касается и массивов: data = [1, 2, 3, 4, 5]; // создали массив (такой синтаксис создания массивов доступен начиная с GMS2) data = 0; // уничтожили массив (теперь это просто переменная) При уничтожении объекта также уничтожаются все переменные, локальные для него, а любые глобальные переменные существуют независимо от них. Поэтому предпочтение нужно отдавать локальным переменным, а глобальные переменные использовать только в случае реальной необходимости этого. Для хранения больших объёмов информации более эффективно, в Game Maker есть поддержка нескольких структур данных - таких, как стек, очередь, список, карта, приоритетная очередь и сетка. Эти структуры создаются, модифицируются и уничтожаются посредством встроенных функций. Также есть функции практически во всех структурах для сортировки данных в них. В некоторых случаях более удобным и более эффективным будет использование буферов, позволяющих хранить произвольные данные и являющимися, по сути, просто выделенными кусками памяти.

8. Объекты и ресурсы.

В основе работы Game Maker с ресурсами лежат уникальные идентификаторы, которые служат для определения конкретного ресурса или экземпляра объекта. Эти идентификаторы могут быть использованы скриптами или функциями для указания необходимого ресурса. Так как создание ресурсов непосредственно в Game Maker подразумевает указание имени, то это имя служит константой, содержащей идентификатор ресурса. Идентификатор конкретного экземпляра хранится в локальной переменной «id». При динамическом создании ресурсов, всегда возвращается идентификатор созданного ресурса, который может быть использован в дальнейшем.