Урок 2

Базовые возможности фреймворка LibGDX

Работа с графикой. Векторная математика. Обработка логики игры.

Работа с векторами

Если в игре применяется позиционирование, направление, скорость или ускорение объектов, придётся иметь дело с векторами. В играх векторы используются для хранения положений, направлений и скоростей объектов, как показано на рисунке ниже.

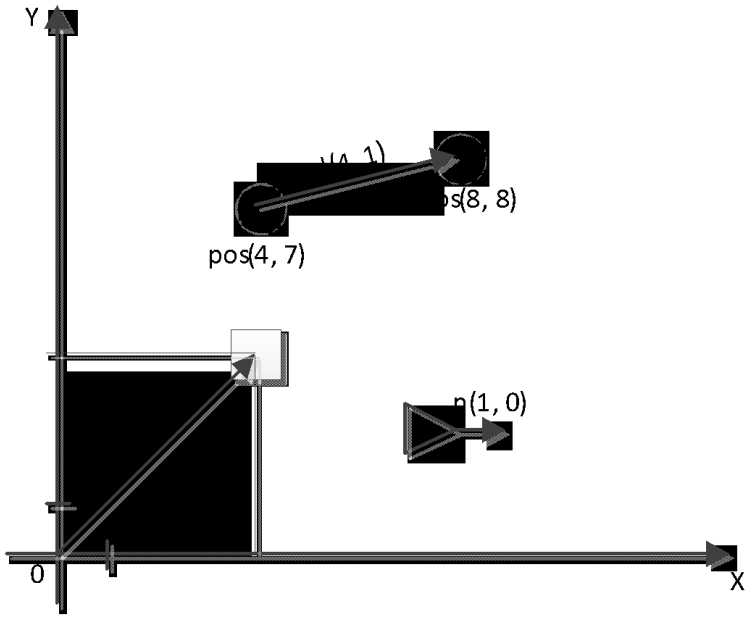


Рис. 1. Различные смыслы векторов (pos – положение, vel – скорость, n – направление).

По сути двумерный вектор — всего пара чисел, и он обретает тот или иной смысл в зависимости от контекста. В таблице ниже представлена часть методов по работе с векторами в LibGDX:

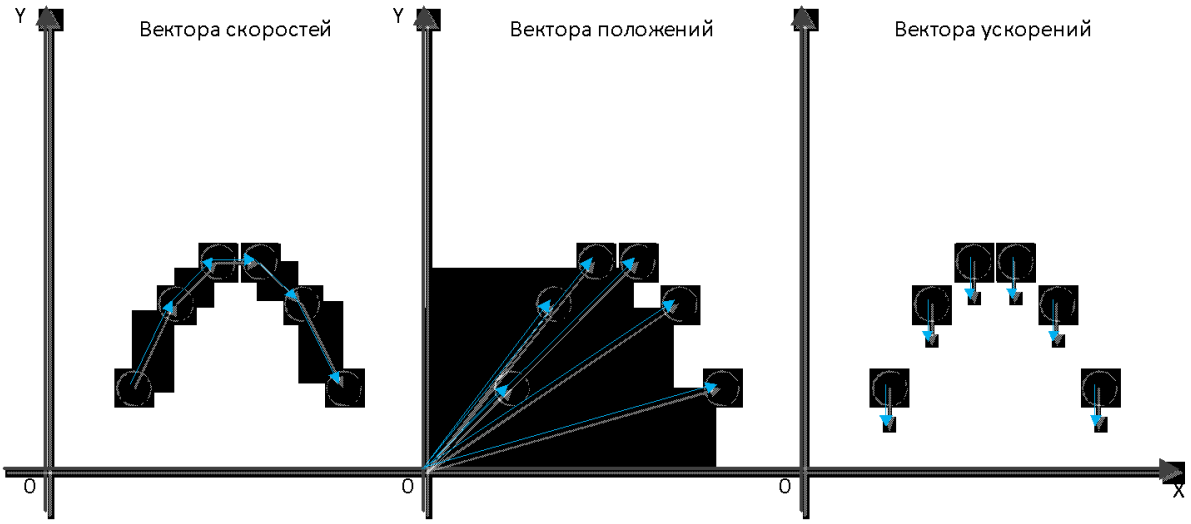
Метод	**Задача**
add()	Сложение двух векторов.
sub()	Вычитание векторов.
scl()	Умножение вектора на скаляр.
len()	Получение длины вектора.
nor()	Нормирование вектора.
cpy()	Копирование вектора.
dot()	Скалярное произведение.

Сложение векторов

Чтобы сложить векторы, нам надо выполнить сложение каждой их составляющей друг с другом:

v1(x1, y1) + v2(x2, y2) = (x1 + x2, y1 + y2).

В качестве примера, где может использоваться сложение, можно рассмотреть движение объекта, у которого есть положение, скорость и ускорение. На каждом кадре к скорости прибавляется ускорение, а к положению — вектор скорости, за счет этого объект начинает двигаться с течением времени. Рассмотрим пример с брошенным объектом.



(а) (б) (в)

Рис. 2. (а) изменение вектора скорости, (б) изменение вектора положения, (в) изменение вектора ускорения

Для сложения векторов используем метод add().

```
Vector2 v1 = new Vector2(5, 8);
Vector2 v2 = new Vector2(2, 4);
Vector2 v3 = v1.cpy().add(v2);
System.out.println("vec: " + v3.x + "; " + v3.y);
v3.add(1, 1);
System.out.println("vec: " + v3.x + "; " + v3.y);
Результат:
vec: 7, 12
vec: 8, 13
```

Вычитание векторов

Вычитание рассчитывается по тому же принципу, что и сложение — вычитаем соответствующие компоненты векторов. Вычитание векторов удобно для получения вектора, который показывает из одного местоположения на другое:

v1(x1, y1) – v2(x2, y2) = (x1 – x2, y1 – y2).

Для вычитания векторов используем метод sub().

```
Vector2 v1 = new Vector2(4, 5);
```

```
Vector2 v2 = new Vector2(1, 1);
v1.sub(v2);
System.out.println("vec: " + v1.x + "; " + v1.y);
Результат:
vec: 3, 4
```

Умножение вектора на скаляр

В играх часто бывает нужно умножить вектор на число (скаляр), для этого надо умножить каждый компонент вектора на скаляр:

n * v(x, y) = (n * x, n * y).

Для умножения вектора на скаляр используем метод scl().

```
Vector2 v = new Vector2(2, 2);
v.scl(3);
System.out.println("vec: " + v.x + "; " + v.y);
Результат:
vec: 6, 6
```

Длина вектора

Длина (модуль) вектора V с компонентами (x, y) обозначается вертикальными линиями как |V| и рассчитывается по формуле:

$|V| = sqrt(x^2 + y^2)$.

Для получения длины вектора используем метод len().

```
Vector2 v = new Vector2(3, 4);
System.out.println("len: " + v.len());
Результат:
len: 5
```

Расстояние

Если какой-либо эффект в игре зависит от расстояния между объектами, например, урон от расстояния до центра взрыва, это расстояние можно вычислить, используя вычитание векторов и поиск длины:

distance = |v1 — v2|.

```
Vector2 v1 = new Vector2(1, 1);
Vector2 v2 = new Vector2(2, 1);
System.out.println("dist: " + v2.cpy().sub(v1).len());
Результат:
dist: 2
```

Нормализация

При работе с направлениями желательно, чтобы вектор направления имел длину, равную единице. Вектор с длиной, равной единице, называется нормализованным. Для нормализации необходимо поделить каждый компонент вектора на его длину.

Для нормализации вектора используем метод nor().

```
Vector2 v = new Vector2(4, 0);
v.nor();
System.out.println("x y: " + v.x + " " + v.y);
Результат:
dist: 1 0
```

Скалярное произведение векторов

Расчет скалярного произведения двух векторов вычисляется по формуле:

v1(x1, y1) • v2(x2, y2) = x1 * x2 + y1 * y2.

Давайте посмотрим, какое значение принимает скалярное произведение при различном направлении векторов:

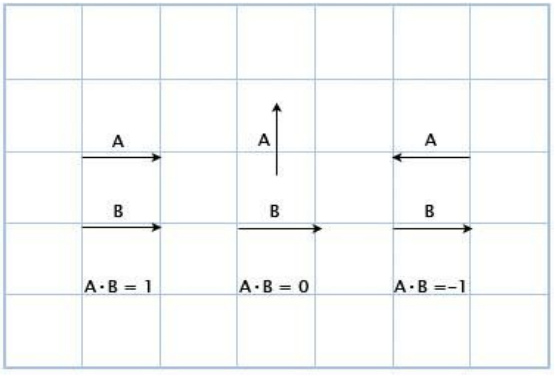


Рис. 3. Скалярное произведение, в зависимости от направления векторов.

Здесь мы видим, что если векторы указывают в одном направлении, их скалярное произведение больше нуля. Когда они перпендикулярны друг другу, то скалярное произведение равно нулю. И когда они указывают в противоположных направлениях, их скалярное произведение меньше нуля. С помощью скалярного произведения векторов можно рассчитать, сколько их указывает в одном направлении. В примере ниже рассчитаем угол (в радианах) между двумя векторами.

```
Vector2 v1 = new Vector2(1, 1);
Vector2 v2 = new Vector2(-1, 1);
v1.nor();
v2.nor();
System.out.println(Math.acos(v1.dot(v2)));
Результат:
1.5707963267948966
```

Управление

Клавиатура

Клавиатура сообщает о пользовательском вводе, генерируя события нажатия и отпускания клавиши. Каждое событие несет в себе код клавиши, который идентифицирует нажатую/отпущенную клавишу. Сами по себе коды клавиш не несут информации о введенном пользователем символе. Часто эта информация зависит от состояния нескольких клавиш. Например, символ «A» генерируется одновременным нажатием «A» и «Shift». Получение зависящих символов (которые были нажаты) из состояния клавиатуры нетривиально. К счастью, операционная система обычно имеет средство для обработки событий, которые сообщают и о коде клавиши, и о символе. LibGDX использует этот механизм, чтобы предоставить вам информацию о символе.

Мышь и сенсорный экран

Мышь и сенсорный экран позволяют пользователю указывать на что-либо на экране. Оба механизма ввода сообщают о местонахождении взаимодействия, передавая 2D-координаты относительно левого верхнего угла экрана. Ось X указывает направление вправо, а ось Y — вниз.

Мышь дает дополнительную информацию: какая кнопка была нажата. Большинство мышей имеют левую и правую кнопку, а также среднюю кнопку. Кроме того, часто имеется колесо прокрутки, которое может быть использовано для масштабирования или прокрутки во многих приложениях.

Сенсорный экран не имеет такого понятия, как кнопки, и сложен из-за возможности аппаратного оборудования отслеживать несколько прикосновений руки одновременно.

LibGDX абстрагирует обработку сообщений мыши и сенсорного экрана. Мышь рассматриваем как специализированную форму сенсорного ввода. Отлеживается только нажатие одного пальца, вдобавок к координатам сообщается, какие кнопки были нажаты. Для сенсорных экранов поддерживается отслеживание нажатия нескольких пальцев и для всех сообщается как о нажатии левой кнопки мыши.

Работа с файловой системой

LibGDX-приложения работают на платформах: Desktop (Windows, Linux, Mac OS X), Android и в браузерах. Каждая платформа работает с файлами по-разному. В LibGDX Files модуль предоставляет следующие возможности: чтение, запись, копирование, перемещение, удаление файлов, проверка существования и получение списка файлов/директорий.

Рассмотрим основные понятия файловых систем для всех поддерживаемых платформ.

Desktop

На Desktop файловая система является одним большим куском памяти. Файлы могут быть доступны по относительному пути к рабочей директории (директории, в которой запущено приложение) или абсолютному пути. При этом игнорируются права доступа, файлы и директории всегда доступны на чтение и запись для всех приложений.

Android

На Android ситуация немного сложнее. Файлы можно хранить внутри APK-приложения, либо как ресурсы или assets. Эти файлы доступны только для чтения. LibGDX использует только asset-механизм, так как он предоставляет прямой доступ к байтовым потокам и больше напоминает традиционную файловую систему. Ресурсы предоставляются для обычных Android-приложений, но вызывают проблемы при использовании в играх. Android совершает манипуляции во время их загрузки, например, автоматически изменяет размер.

Asset хранится в assets-директории Android-проекта и затем автоматически упаковывается в APK-файл. Ни одно другое приложение не может получить доступ к этим файлам.

Файлы также могут храниться во внутренней памяти, где они доступны для чтения и записи. Каждое установленное приложение имеет специальную выделенную внутреннюю директорию для хранения. Эта директория тоже доступна только для этого приложения.

Наконец, файлы можно хранить на внешнем накопителе данных, например SD-карте. Внешние накопители не всегда могут быть доступны, например, пользователь может вытащить SD-карту. Файлы в этом месте хранения следует считать не всегда доступными. Чтобы иметь возможность записи на внешние накопители, вам нужно будет добавить разрешения в файл AndroidManifest.xml, смотрите их ниже.

Типы хранения файлов

Файл в LibGDX представлен экземпляром FileHandle-класса, который определяет, где находится файл. Следующая таблица иллюстрирует наличие и местоположение каждого типа файлов.

Тип	Описание, путь файла и особенности
Classpath	Classpath-файлы хранятся непосредственно в директории исходников. Они упакованы вместе с jar-файлами и всегда доступны только для чтения. Они имеют свое назначение, но по возможности нужно избегать их использования.

Внутренний	Внутренние файлы относятся к корневой или рабочей директории приложения на Desktop, к assets-директории на Android. Эти файлы доступны только для чтения. Если файл не может быть найден во внутреннем хранилище, файловый модуль возвращается к поиску файла в classpath.
Локальный	Локальные файлы хранятся относительно корневой или рабочей директории на Desktop и относятся к внутреннему (частному) хранилищу приложений на Android. Локальный и внутренний тип в основном одно и то же для Desktop.
Внешний	Пути к внешним файлам являются относительными корня SD-карты на Android и домашней директории текущего пользователя на Desktop-системах.
Абсолютный	Абсолютные файлы должны иметь полностью указанный путь. Примечание: Ради портативности эта опция должна использоваться только в случае крайней необходимости.

Абсолютные и classpath-файлы в основном используются для таких инструментов, как редакторы на Desktop, которые имеют более сложные требования файлового ввода/вывода. Для игр их можно спокойно игнорировать. Порядок, в котором вы должны использовать типы файлов, выглядит следующим образом:

- Внутренние файлы — все assets (изображения, аудиофайлы и так далее), упакованные вместе с приложением, являются внутренними файлами.
- Локальные файлы — если необходимо записывать небольшие файлы, например, сохранить состояние игры, используйте локальные файлы. Они имеют частный доступ для вашего приложения.
- Внешние файлы — если необходимо записывать большие файлы, например, снимки экрана или скачивать файлы из интернета, следует использовать внешние накопители. Помните, что внешний накопитель не всегда доступен.

Базовые возможности работы с файловой системой

Проверка наличия хранилища и путей. Различные типы хранилищ могут быть недоступны в зависимости от платформы, на которой запущено приложение. Вы можете запросить такую информацию через модуль Files. Можно также запросить корневой путь для внешнего и локального хранилища.

```
boolean isExtAvailable = Gdx.files.isExternalStorageAvailable();
boolean isLocAvailable = Gdx.files.isLocalStorageAvailable();
String extRoot = Gdx.files.getExternalStoragePath();
String locRoot = Gdx.files.getLocalStoragePath();
```

Получение дескриптора файла. Получить FileHandle из любого типа хранилища можно с помощью модуля Files.

```
FileHandle handle1 = Gdx.files.internal("data/file.txt");
// внутреннее хранилище
FileHandle handle2 = Gdx.files.classpath("file.txt");
```

```
// локальное хранилище
FileHandle handle3 = Gdx.files.external("file.txt");
// внешнее хранилище
FileHandle handle4 = Gdx.files.absolute("/some_dir/subdir/myfile.txt"); // по
абсолютному пути
```

Экземпляры FileHandle передаются методам соответствующих классов, отвечающих за чтение и запись данных.

Список файлов и проверка свойств. Иногда необходимо проверить конкретный файл на его наличие или просмотреть содержимое директории. Данные методы для работы с файлами очень похожи на методы класса java.io.File.

```
boolean exists = Gdx.files.external("doitexist.txt").exists();        //
Проверка существования файла
boolean isDirectory = Gdx.files.external("test").isDirectory();       //
Проверка является указанный файл директорией
FileHandle[] files = Gdx.files.local("mylocaldir/").list();           //
Получение списка файлов из директории
```

Чтение и запись данных. После получения FileHandle мы можем либо передать его в класс, который знает, как загружать содержимое из файла, либо читать его сами. Последнее делается через любой из методов ввода в FileHandle-классе. Второй параметр метода writeString() указывает, должно ли содержимое быть добавлено к файлу или будет перезаписано.

```
FileHandle file = Gdx.files.internal("myfile.txt");
String text = file.readString();
FileHandle file2 = Gdx.files.local("myfile2.txt");
file.writeString("12345", false);
```

Удаление, копирование, переименование и перемещение файлов/директорий. Эти операции возможны только для типов файлов, доступных для записи. Источником операции копирования может быть FileHandle, доступный только для чтения. Обратите внимание, что источником и приемником могут быть файлы или директории.

```
FileHandle from = Gdx.files.internal("myresource.txt");
from.copyTo(Gdx.files.external("myexternalcopy.txt");
Gdx.files.external("myexternalcopy.txt").rename("mycopy.txt");
Gdx.files.external("mycopy.txt").moveTo(Gdx.files.local("mylocalcopy.txt"));
Gdx.files.local("mylocalcopy.txt").delete();
```

Обработка ошибок. При возникновении ошибок применяются RuntimeException вместо проверяемых исключений по причине того, что в 90% времени мы получаем доступ к файлам, и знаем, что они существуют и доступны для чтения, например, внутренние файлы, упакованные вместе с приложением.

Домашнее задание

С домашним заданием вы можете ознакомиться на странице урока.

Дополнительные материалы

1. http://www.gamefromscratch.com/page/LibGDX-Tutorial-series.aspx
2. https://github.com/libgdx/libgdx/wiki
3. https://github.com/libgdx/libgdx/wiki/External-tutorials

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. https://habrahabr.ru/post/131931/
2. https://github.com/libgdx/libgdx/wiki/File-handling
3. www.libgdx.ru/2013/09/file-handling.html