МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ИГР С ИСПОЛЬЗОВАНИЕМ ДВИЖКА PIXIJS

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Варфоломеевой Татьяны Денисовны

Научный руководитель
зав. каф. техн. прогр., к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2019

ВВЕДЕНИЕ

Разработка компьютерных игр является одной из популярных отраслей информационных технологий. Основной задачей большинства компьютерных игр является развлечение, однако игры могут использоваться в образовательных целях, для развития воображения и логического мышления. Поэтому актуальным является вопрос об использовании различных средств разработки компьютерных игр. Эти средства должны быть наглядными, хорошо задокументированными и удобными в использовании.

Цель бакалаврской работы — рассмотрение различных типов движков для двумерной графики и реализация на одном из движков компьютерной игры, основанной на физике двумерных объектов.

Поставленная цель определила следующие задачи:

- рассмотрение принципов работы двумерных движков на примере движка PixiJS;
- реализация алгоритмов обнаружения и обработки коллизий;
- разработка приложений с помощью движка PixiJS.

Методологические основы разработки двумерных компьютерных игр представлены в работах Rex van der Spuy, D. Gregorius, Ch. Hecker, G. Weir, D. Crockford.

Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и четырех приложений. Общий объем работы — 89 страниц, из них 70 страниц — основное содержание, включая 6 рисунков, цифровой носитель в качестве приложения, список использованных источников информации — 24 наименования.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Использование двумерных движков в разработке игр» посвящен рассмотрению принципов разработки двумерных игр, использующих движки. Первый раздел состоит из двух подразделов.

Первый подраздел «Обзор двумерных игровых движков» посвящен краткому обзору и сравнению нескольких наиболее известных двумерных игровых движков.

Игровой движок — программное обеспечение, используемое в качестве основы для создания игры. Движок может представлять из себя как небольшой набор функций, так и сложную систему из нескольких программных модулей. То, какие функции входят в движок, зависит от целей, для которых был создан этот движок. Основное преимущество игрового движка заключается в том, что несколько игр могут использовать один и тот же движок. Это позволяет ускорить разработку игры, поскольку часть функций уже реализована внутри движка, и их не нужно создавать заново.

Были рассмотрены такие игровые движки, как Unity, RPG Maker, Game Editor, Corona и PixiJS. Для разработки приложения был выбран движок PixiJS, который включает в себя такие возможности, как загрузка изображений с помощью атласа текстур, группирование спрайтов, создание интерактивных объектов и другие.

Во втором подразделе «Принципы работы движка PixiJS» рассмотрены наиболее часто используемые классы и методы движка PixiJS [1,2]. Большинство двумерных графических движков содержат аналогичные классы, которые работают похожим образом. Также были рассмотрены специальные возможности, характерные только для движка PixiJS.

С помощью класса Application создается сцена. Под сценой понимается прямоугольная область, на которую выводятся все остальные изображения. Объект класса `Application` автоматически создает элемент `<canvas>` и определяет, как располагать на этом элементе изображения.

В PixiJS есть специальный класс `Container`, который позволяет создавать контейнеры — объекты, которые позволяют работать с группой изображений, как с одним объектом. Однако, чтобы сгруппировать изображения, класс `Container` использовать не обязательно. Большинство классов PixiJS, которые работают с изображениями, обладают теми же методами, что и класс

`Container`, и могут использоваться в качестве контейнера. Сцена, созданная с помощью класса `Application` работает как базовый контейнер, в котором содержаться все изображения, используемые приложением.

Изображения, добавляемые на сцену, хранятся в виде объектов, называемых спрайтами. От обычного изображения спрайт отличается только тем, что спрайт можно контролировать с помощью программного кода. В PixiJS спрайты создаются с помощью специального класса `Sprite`.

Существует три способа создать спрайт:

— с помощью файла с картинкой;
— с помощью тайлсета (tileset);
— с помощью атласа текстур (texture atlas).

Тайлсет — большое изображение, составленное из отдельных картинок, которые используются в игре [3]. Другими словами, тайлсет — все необходимые спрайты, объединенные в одно изображение. Атлас текстур — файл JSON, содержащий информацию о размере и положении спрайтов, хранящихся в тайлсете.

Так как PixiJS отображает картинки с помощью графического процессора (GPU) и библиотеки WebGL, изображение должно быть в том формате, которое графический процессор может обрабатывать. Изображение, с которым может работать WebGL, называется текстурой (texture). Прежде, чем создать спрайт, необходимо преобразовать обычное изображение в текстуру. Для создания текстур в движке PixiJS используется класс `Texture`.

Чтобы загрузить файл с изображением и преобразовать его в текстуру, PixiJS использует объект `loader` и два его метода — `add` и `load`. Метод `add` загружает изображение из файла, путь к которому передается в качестве параметра. Метод `load` принимает в качестве параметра функцию, которая будет запущена после загрузки изображений.

Если проект небольшой и использует не так много спрайтов, достаточно загружать изображения из отдельных файл. Однако в большинстве игр спрайты загружаются из тайлсетов. Чтобы извлечь изображение из тайлсета, нужно сначала создать прямоугольную область, размер и координаты которой такие же, как и у изображения в тайлсете. Прямоугольная область создается с помощью класса `Rectangle`.

Существует более удобный способ извлекать текстуры из тайлсета — с

помощью атласа текстур, файла JSON, который содержит словари JavaScript. Ключами этих словарей являются названия изображений, а значениями — словари, внутри которых хранится информация о размере и положении изображений. Благодаря этому JSON файлу для создания текстуры необходимо знать только название соответствующего изображения. А значит, координаты и размер картинки больше не имеют значения. Если тайлсет изменится, например, в него будут добавлены новые изображения, достаточно будет обновить только файл JSON. Если названия изображений не изменились, менять код программы не придется. Атлас текстур, т. е. файл JSON, может загружаться так же, как и файл с изображением, с помощью объекта `loader` и метода `add`.

Управление спрайтами осуществляется с помощью методов и свойств класса `Sprite`. Чтобы переместить спрайт, достаточно изменить значения свойств `x` и `y`, в которых хранятся координаты спрайта. Чтобы изменить размер спрайта, можно изменить значения его свойств `width` и `height`, ширины и высоты соответственно. Помимо ширины и высоты у спрайта есть свойство `scale`, которое позволяет изменять его размеры пропорционально. Чтобы повернуть спрайт, нужно изменить значение свойства `rotation`, в котором хранится угол поворота спрайта.

Чтобы анимировать спрайт, необходимо создать игровой цикл. Внутри игрового цикла меняются координаты спрайтов, и за счет того, что выполнение игрового цикла постоянно повторяется, спрайты будут двигаться в заданном направлении. В движке PixiJS есть специальная функция для создания игрового цикла — `ticker`.

Помимо игрового цикла обычно создаются функции, называемые игровыми состояниями. Такие функции вызываются при выполнении некоторых условий. Например, если персонаж игрока погиб, игра переходит в состояние «GAME OVER», при котором на экран выводится соответствующее сообщение, а персонажем больше нельзя управлять.

Помимо изображений, загружаемых из файлов, движок PixiJS позволяет создавать простые геометрические фигуры, такие как прямоугольники, окружности, линии и многоугольники. Все фигуры создаются с помощью класса `Graphics`, которому принадлежат методы `beginFill` и `endFill`, отвечающие за заливку фигур, и метод `lineStyle`, который определяет стиль линии вокруг фигуры, а также методы для рисования фигур разной формы: прямоугольни-

ки создаются с помощью метода `drawRect`, окружности — с помощью метода `drawCircle`, многоугольники — с помощью метода `drawPolygon`.

Помимо класса `Graphics`, который позволяет рисовать фигуры, в движке PixiJS есть классы `Text` и `TextStyle`, которые позволяют изображать текст. Чтобы просто вывести текст, достаточно создать объект класса `Text`. С текстом можно работать так же, как и с обычным спрайтом. Стиль текста можно задать с помощью класса `TextStyle`.

Таким образом, в первом разделе были кратко рассмотрены наиболее известные движки, которые могут использоваться для разработки двумерных игр. Для разработки приложения был выбран движок PixiJS, классы и методы которого также были описаны в данном разделе.

Второй раздел «Создание приложения с помощью движка PixiJS» посвящен реализации двумерной игры, задача которой — построение машины Голдберга, некоторого механизма, выполняющего длинную последовательность действий «по принципу домино». В данном приложении будет три типа игровых объектов — блоки, стеклянные и металлические шарики. Блоки являются статичными прямоугольными объектами, которые используются в качестве платформ для шариков. Таким образом, задача игры сводится к построению как можно более сложного пути из блоков, по которым будут катиться шарики.

Второй раздел содержит два подраздела. Первый подраздел «Физика в двумерных играх» посвящен реализации алгоритмов обнаружения и обработки коллизий.

Коллизия — пересечение фигур [4].

Обнаружение коллизий — поиск пересекающихся фигур [4].

В большинстве игр коллизии бывают двух видов: коллизия между окружностями и коллизия между прямоугольниками. Даже если объект в игре по форме немного отличается от окружности или прямоугольника, он все равно рассматривается как круглый или прямоугольный. Для этого вместо самого объекта при обнаружении коллизий используются ограничивающие окружности и ограничивающие параллелограммы. Ограничивающая окружность и ограничивающий параллелограмм — фигуры, которые используются при обнаружении коллизий и примерно представляют игровой объект, однако в самой игре не отображаются.

После обнаружении коллизии необходимо разделить пересекающиеся

объекты. Чтобы это сделать, необходимо знать нормальный вектор и глубину пересечения. Под нормальным вектором в данном случае понимается вектор, вдоль которого необходимо переместить объекты, чтобы разделить их. Этот вектор называется нормальным, поскольку часто он перпендикулярен ребру одной из фигур. Глубина пересечения — расстояние, на которое необходимо переместить объекты, чтобы они больше не пересекались. Вычисление нормального вектора и глубины пересечения является частью алгоритмов обнаружения коллизий.

В данном разделе рассматриваются различные случаи пересечения фигур.

Самый простой случай — столкновение двух окружностей. Две окружности пересекаются, если расстояние между их центрами меньше, чем сумма их радиусов. В качестве нормального вектора в данном случае берется вектор, соединяющий центры окружностей, а глубина пересечения равна разнице между суммой радиусов и расстоянием между их центрами. Для обнаружения коллизии между двумя окружностями была реализована функция `circleVScircle`. Функция возвращает ассоциативный массив, в котором содержится информация о коллизии: значение `true` или `false` — произошла ли коллизия, нормальный вектор (`n`), глубину пересечения (`penetration`).

Ограничивающий параллелограмм бывает двух видов: ориентированный и неориентированный. Стороны неориентированного ограничивающего параллелограмма всегда параллельны осям координат. Ориентированный ограничивающий параллелограмм может быть повернут на некоторый угол.

Неориентированные прямоугольники могут пересекаться по оси OX и по оси OY. Прямоугольники пересекаются по оси OX, если разница между координатами X центров этих прямоугольников меньше, чем сумма половин их ребер, параллельных оси OX. Аналогично для оси OY. Прямоугольники могут пересекаться по обеим осям или только по одной оси. Например, если один прямоугольник выше другого, они пересекаются по оси OX, но не пересекаются по оси OY. Коллизия возникает только тогда, когда прямоугольники пересекаются по обеим осям. В качестве нормального вектора для такой коллизии берется нормальный вектор одного из пересекающихся ребер. Под нормальным вектором ребра в данном случае понимается единичный вектор, перпендикулярный этому ребру. В качестве глубины пересечения берется ми-

нимальная из двух величин: либо пересечение по оси OX, либо пересечение по оси OY. Для обнаружения коллизий между неориентированными прямоугольниками была реализована функция `rectangleVSrectangleParallel`.

Теперь рассмотрим пересечение окружности и прямоугольника, параллельного осям координат. Идея алгоритма обнаружения такой коллизии состоит в поиске точки прямоугольника, которая ближе всех остальных к центру окружности. Когда такая точка найдена, проверка пересечения фигур становится похожа на случай с двумя окружностями. Нормальный вектор коллизии идет от этой точки к центру окружности, а глубина пересечения — разница между расстоянием от центра окружности до ближайшей точки прямоугольника и радиусом окружности. Однако, если окружность оказалась внутри прямоугольника, нормальный вектор должен быть повернут в противоположную сторону. Для обнаружения коллизий между неориентированными прямоугольниками была реализована функция `circleVSrectangleParallel`.

Теперь рассмотрим алгоритмы, работающие с ориентированными ограничивающими параллелограммами. Другими словами, пусть теперь прямоугольник может быть повернут на произвольный угол. Самый сложный случай — пересечение двух прямоугольников. Алгоритм обнаружения коллизий для данного случая основывается на так называемых опорных точках.

Опорная точка — вершина многоугольника, находящаяся дальше всех в заданном направлении [5]. Если в заданном направлении существует две точки, находящиеся на равном расстоянии, опорной точкой может считаться любая из них.

Чтобы найти опорную точку, необходимо перебрать все вершины прямоугольника, и для каждой вершины получить проекцию на вектор, задающий направление. Вершина с наибольшей проекцией берется в качестве опорной точки. Проекцию можно получить с помощью скалярного произведения двух векторов — самой вершины (пара координат является вектором из двух компонент) и направления, в котором ищется опорная точка. Для поиска опорных точек была реализована функция `getSupportPoint`, которая принимает два параметра — прямоугольник и вектор, в направлении которого ищется опорная точка, и возвращает вершину прямоугольника, т. е. найденную опорную точку, заданную в виде вектора. Для поиска максимального расстояния до опорной точки была реализована функция `faceDirections`, которая

принимает в качестве параметров два прямоугольника и возвращает число — наибольшее расстояние. В итоге, наличие коллизии проверяется функцией `rectangleVSrectangle`, которая дважды вызывает функцию `faceDirections`, меняя параметры местами. Функция `rectangleVSrectangle` возвращает `true` или `false`, соответственно, есть коллизия или нет.

Последний случай — пересечение окружности с ориентированным прямоугольником. Чтобы упростить вычисления, изображение поворачивается относительно центра прямоугольника таким образом, чтобы стороны прямоугольника были параллельны осям координат. После того, как изображение было повернуто, выполняется алгоритм обнаружения коллизий для неориентированного прямоугольника и окружности. Этот алгоритм уже был рассмотрен ранее. Для нормального вектора коллизии, полученного в результате работы алгоритма, необходимо выполнить обратный поворот. Чтобы выполнить поворот, достаточно вычислить новые координаты каждой точки, которая будет задействована в алгоритме обнаружения коллизии. Для вычисления новых координат используется матрица поворота. Коэффициентами этой матрицы являются косинусы и синусы угла поворота.

После того, как была обнаружена коллизия, пересекающиеся объекты разделяются. Объекты разделяются с помощью импульса [4, 6]. Импульс можно понимать, как большую физическую силу, которая мгновенно меняет скорости объектов, изменяя направление их движения.

Если функцией `circleVScircle`, `rectangleVSrectangle` или `circleVSrectangle` была обнаружена коллизия, для пересекающихся объектов вызывается функция `resolveCollision`, которая рассчитывает величину импульса и разделяет объекты, меняя их скорость.

Во втором подразделе «Запуск приложения и игровой цикл» описаны скрипты, отвечающие за работу приложения, а также процесс запуска приложения [7].

Для запуска приложения используется веб-сервер `node.js`, который запускается с помощью командной строки. После запуска веб-сервера можно открыть браузер и перейти по адресам, указанным в командной строке. В результате откроется веб-страница, на которой должен отображаться интерфейс приложения. Содержимое этой страницы задается в файле `index.html`.

В файле `index.html` содержится основной скрипт, отвечающий за запуск

и дальнейшую работу приложения. В теге <head> подключаются файлы с расширением js, в которых хранятся необходимые функции, в том числе файл pixi.min.js, содержащий классы и функции движка PixiJS.

Тег <body> содержит единственный тег <script>, в теле которого загружается атлас текстур, а также определяются функции setup, build и play. Функция setup отвечает за создание интерфейса приложения и запуск игрового цикла с помощью функции ticker движка PixiJS. Функции build и play отвечают за соответствующие игровые состояния.

Игра может принимать два состояния — build и play. Если игра находится в состоянии build, пользователь может создавать, перемещать и удалять игровые объекты, а шарики не двигаются. Если игра находится в состоянии play, шарики перемещаются под действием силы тяжести, но пользователь не может управлять игровыми объектами. Переключение между состояниями игры и управление игровыми объектами осуществляется с помощью кнопок интерфейса.

В теле функции setup создаются обработчики событий, которые отвечают за поворот блоков по нажатию на клавиши E и Q. Для создания таких обработчиков используется функция keyboard, которая принимает в качестве параметра строку, содержащую код клавиши, и возвращает обработчик событий. После этого вызывается функция menu_setup, с помощью которой создается интерфейс приложения.

Окно приложения разделено на две области: сверху находится область, содержащая кнопки, снизу — область, содержащая шарики и блоки. Вторая область занимает большую часть экрана и изначально пуста, поскольку располагаемые на ней шарики и блоки создаются пользователем. Обе области задаются в виде фигур с помощью класса Graphics.

В верхней области содержатся кнопки build и play, которые позволяют переключать состояния игры, а также кнопки для создания, перемещения и удаления объектов. Изображения для кнопок создаются с помощью класса Graphics, а надписи на кнопках создаются с помощью классов Text и TextStyle. С помощью средств движка PixiJS для кнопок определяются обработчики событий, которые при нажатии на кнопку вызывают соответствующую функцию.

При создании или перемещении объекта выполняется проверка того, что

в результате выполненного действия не возникло коллизии. Поиск коллизий выполняется с помощью функций `circleVScircle`, `rectangleVSrectangle` и `circleVSrectangle`. Если коллизия обнаружена, выполненные действия отменяются.

В теле функции `build` выполняется поворот блоков. Каждый блок поворачивается в соответствии с его угловой скоростью, если угловая скорость равна нулю, блок поворачиваться не будет. Угловая скорость блока меняется, когда пользователь нажимает кнопки E и Q, отвечающие за поворот блоков.

В теле функции `play` осуществляется перемещение игровых объектов, а также выполняется поиск коллизий между парами круглых объектов и между прямоугольными и круглыми объектами. Чтобы перемещать объекты, для каждого объекта вычисляется вектор скорости, координаты которого прибавляются к координатам объекта. Если в результате перемещения некоторый объект оказался за пределами экрана, этот объект удаляется. Для поиска коллизий между объектами вызываются функции `circleVScircle` и `circleVSrectangle`.

Таким образом, во втором разделе была описана реализация двумерной компьютерной игры, использующей движок PixiJS. В том числе была описана реализация алгоритмов обнаружения и обработки коллизий, а также были описана разработка интерфейса приложения и игровых состояний.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были рассмотрены средства, которые предоставляет движок PixiJS для разработки приложений, использующих двумерную графику. Были рассмотрены наиболее часто используемые классы и их методы, различные способы загрузки изображений и работа со спрайтами. Также были реализованы алгоритмы обнаружения коллизий для неориентированных и ориентированных объектов. В ходе выполнения практики было разработано приложение, в котором продемонстрированы специальные возможности движка PixiJS, такие как загрузка изображений с помощью атласа текстур, группирование спрайтов, создание интерактивных объектов и другие. Также были созданы классы объектов и реализовано наследование этих классов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Rex van der Spuy,*. Learn Pixi.js: Create Great Interactive Graphics for Games and the Web / Rex van der Spuy. — Нью-Йорк: Apress, 2015.

2 PixiJS - The HTML5 Creation Engine [Электронный ресурс]. — URL: http://pixijs.download/release/docs/index.html (Дата обращения 24.05.2019). Загл. с экр. Яз. англ.

3 *Rex van der Spuy,*. Foundation Game Design with HTML5 and javaScript / Rex van der Spuy. — Нью-Йорк: Apress, 2012.

4 *Hecker, Ch.* Physics, Part 3: Collision Response [Электронный ресурс] / Ch. Hecker. — URL: http://chrishecker.com/images/e/e7/Gdmphys3.pdf (Дата обращения 13.05.2019). Загл. с экр. Яз. англ.

5 *Gregorius, D.* Physics for Game Programmers: The Separating Axis Test between Convex Polyhedra [Электронный ресурс] / D. Gregorius. — URL: https://gdcvault.com/play/1017646/Physics-for-Game-Programmers-The (Дата обращения 15.05.2019). Загл. с экр. Яз. англ.

6 *Weir, G.* The coefficient of restitution for the idealized impact of a spherical, nano-scale particle on a rigid plane / G. Weir, P. McGavin // *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*. — 2008. — Vol. 464. — Pp. 1295–1307.

7 *Crockford, D.* JavaScript: The Good Parts / D. Crockford. — Себастопол: O'Reilly, 2008.