Учебный движок для создания 2D-игр "GameStudy"

Учебный движок предназначен для быстрой и легкой разработки простейших 2D-игр с использованием спрайтов, анимаций, надписей, графических примитивов и звуковых эффектов. Сама игра реализуется как программа на диалекте JavaScript путем создания файлов скриптов, каждый из которых содержит коды трех функций - инициализации игры, рендера игры и обработки состоянии игры.

Содержание

1. Структура движка

В корневом каталоге движка размещается файл GameStudy.exe, запуск которого загружает созданную игру. Все файлы .dll и подкаталог platforms - это библиотеки, необходимые для работы движка. Подкаталог scripts - содержит программу main.js на диалекте JavaScript (см. Приложение 1) с кодом, с которой начинается выполнение игры, и если нужно, другие файлы скриптов и включаемые блоки кода. Остальные подкаталоги содержат ресурсы игры, которые загружаются в коде.

- fonts - шрифты TTF
- sounds - звуки
- sprites - спрайты

2. Создание игры

Чтобы создать игру, нужно скопировать ресурсы разрабатываемой игры в соответствующие каталоги движка и наполнить функции в шаблонном файле main.js загрузкой ресурсов, рендером (выводом графики) и обработкой состояния игры. Предопределенные функции main.js:

```
function Init(params) {
}
```

Функция вызывается один раз в момент старта игры, в ней нужно разместить все загрузки ресурсов и инициализацию переменных. Функция должна возвращать значение true. Параметр функции - для файла main это массив аргументов командной строки, для остальных скриптов - то, что передано при вызове goToScript.

```
function Render() {
}
```

Функция вызывается каждый раз, когда движок перерисовывает игровое окно, в ней нужно выводить все спрайты и надписи. Функция должна возвращать значение true.

```
function Frame(dt) {
}
```

Функция вызывается каждый раз, когда движок пересчитывает состояние игры, в ней нужно обновить состояние игровых объектов (позиции, счетчики и прочее). Параметр dt - время в секундах с момента последнего вызова функции Frame, число двойной точности. Функция должна возвращать значение true, возврат false означает остановку игры.

3. Простейший пример игры

В каталоге sprites размещаем файл player.png В файле main.js размещаем код

```
// Переменную для спрайта нужно объявить в начале, чтобы её видели все функции
var spr_player ;
// То же для переменных координат спрайта
var player_x ;
var player_y ;

// Функция инициализации
function Init(params) {
    // Загрузка спрайта
    spr_player = game.loadSprite('player.png') ;
    // Установка координат для вывода
    player_x=400 ;
    player_y=300 ;
    return true ;
}

// Функция рендера
function Render() {
    // Выводим спрайт в заданные координаты
    spr_player.renderTo(player_x,player_y) ;
    return true ;
}

// Функция фрейма
function Frame(dt) {
    // Обрабатываем нажатия клавиш
    if (game.isKeyDown(KEY_LEFT)) player_x-=10 ;
    if (game.isKeyDown(KEY_RIGHT)) player_x+=10 ;
    if (game.isKeyDown(KEY_UP)) player_y-=10 ;
    if (game.isKeyDown(KEY_DOWN)) player_y+=10 ;
    return true ;
}
```

Данная игра выводит спрайт player.png и двигает его под действием клавиш-стрелок на клавиатуре.

4. Более сложная структура проекта

При создании сложной игры целесообразно использовать расширенную структуру проекта. 1. Создать несколько файлов скриптов, помимо main.js - например, оставить в main.js код для меню, а в файле game.js разместить основной код игры. Каждый файл скрипта должен содержать три функции (инициализации, рендера и обработки состояния), передача управления между файлами осуществляется командой goToScript. 2. Вынести общие функции, переменные и константы во включаемые файлы, которые можно подключить в файлы скриптов метакомандой $include<файл>. Движок обрабатывает команду путем замены команды на содержимое файла. 3. Вынести конфигурационные и текстовые данные в файлы JSON, размещаемые в корневом каталоге игры и загружаемые по мере необходимости командой loadObject. Это позволит отделить данные настройки от кода и разделить общие данные между разными файлами скриптов.

Пример файлов для проекта, содержащего, помимо скрипта main.js, еще скрипт game.js, включаемый файл consts.inc и файл конфигурации texts.json: В каталоге scripts - файлы main.js, game.js, consts.inc В корневом каталоге игры - файл texts.json Структура файлов main.js и game.js:

```
$include<consts.inc>

function Init(params) {
...
}
function Render() {
...
}
function Frame(params) {
...
}
```

Структура файла consts.inc

```
const PLAYERSPEED = 150 ;
var username ;
function getFieldRect() {
    return {x:100,y:200} ;
}
```

Структура файла texts.json

```
{
    "textmenu":"Меню",
    "textstart":"Играть"
    "textexit":"Выход"
}
```

Передача управления в файл game.js с параметром - номер уровня 3 и свободным режимом игры.

```
game.goToScript("game",{level:3,freeplay:true}) ;
```

5. Продвинутое использование движка для создания игры на C++ Qt

Помимо использования готового движка, можно так же использовать систему классов движка для организации игры на C++ Qt, разделив игровую логику между JavaScript и C++. На C++ можно реализовать собственные классы со сложной логикой, добавив их в скрипт, а обработку игровых объектов реализовать на JavaScript. Для реализации подобной схемы, можно использовать шаблон проекта GameStudyNoGUI, включенный в исходный текст движка, и расширить его необходимой логикой. Чтобы добавить собственный объект в скрипт, его класс надо унаследовать от QObject и добавить к игре методом:

```
game->addObjectToEngine(имя_объекта,ссылка на объект)
```

после чего он будет доступен в скрипте под заданным именем со всеми его публичными слотами

Данный подход имеет смысл, если нужно разработать сложную игру на C++ с SFML, но хочется использовать готовые объекты движка вроде спрайтов и анимаций, а также поддержку скриптования игровой логики на JavaScript.

6. Объекты движка и их функции

Глобальные объекты

В движке определены несколько глобальных объектов, методы которых вызываются из кода игры.

Объект game

Содержит функции загрузки ресурсов, создания графических примитивов, обработки нажатия клавиш и мыши.

Функции объекта game

```
loadSprite(filename)
```

Создает и возвращает объект спрайта. Параметр - имя файла из каталога sprites.

```
loadAnimationFromFiles(filenames, fps)
```

Создает и возвращает объект анимации из набора файлов. Параметры - массив имен файлов из каталога sprites, которые образуют отдельные кадры анимации, и FPS анимации. Пример вызова - loadAnimationFromFiles(['1.png','2.png','3.png',7)

```
loadAnimation(filename, w, h, framecount, fps)
```

Создает и возвращает объект анимации из одного файла, содержащего несколько кадров. Параметры - имя файла из каталога sprites, ширина и высота отдельного фрейма, число фреймов и FPS анимации. Фреймы считываются из файла слева направо, сверху вниз.

```
loadAnimation(filename, framecount, fps)
```

Создает и возвращает объект анимации из одного файла, содержащего несколько кадров. Параметры - имя файла из каталога sprites, число фреймов и FPS анимации. Фреймы считываются из файла слева направо, сверху вниз. В отличии от функции loadAnimation, в которой явно указывается ширина и высота спрайта - эта функция автоматически вычисляет высоту и ширину спрайта, исходя из размеров файла и числа фреймов.

```
loadText(fontname,text,size)
```

Создает и возвращает объект надписи. Параметры - имя шрифта из каталога fonts (вместе с расширением, например, "Arial.ttf"), текст надписи, размер

шрифта в надписи.

```
loadSound(filename)
```

Создает и возвращает объект звука. Параметр - имя файла из каталога sounds.

```
createLine(r,g,b)
```

Создает и возвращает объект линии. Параметры - компоненты цвета линии в RGB.

```
createRect(r,g,b)
```

Создает и возвращает объект прямоугольника. Параметры - компоненты цвета прямоугольника в RGB.

```
isKeyDown(keycode)
```

Проверяет, была ли нажата клавиша клавиатуры. Параметр - код клавиши (именованные константы клавиш приведены в Приложении 2)

```
isOneOfKeysDown(keycodes)
```

Проверяет, была ли нажата любая одна из кнопок клавиатуры. Параметр - массив кодов клавиш (именованные константы клавиш приведены в Приложении 2)

```
isLeftButtonClicked()
```

Проверяет, была ли нажата левая кнопка мыши.

```
isRightButtonClicked()
```

Проверяет, была ли нажата правая кнопка мыши.

```
getMousePos()
```

Возвращает координаты мыши в игровом окне как объект с полями x и y.

```
setGameTitle(title)
```

Устанавливает заголовок окна с игрой. Параметр - строка заголовка.

```
getTotalTime()
```

Возвращает общее время с момента начала игры в секундах.

```
resetTotalTime()
```

Сбрасывает счет общего времени с момента начала игры в ноль.

```
getFPS()
```

Получает текущий FPS, усредненный за последнюю секунду.

```
setBackgroundColor(r,g,b)
```

Устанавливает цвет фона окна. Параметры - компоненты цвета текста в RGB

```
goToScript(scriptname,params)
```

Передает движку команду на загрузку и обработку другого скрипта, с функциями Init, Frame и Render. Параметры - название файла скрипта без расширения и аргумент (любое значение, массив или объект), который будет передан в функцию Init. Допускается передача параметра null, если аргумент не планируется использовать. После вызова команды, текущий скрипт и все созданные им объекты выгружаются из памяти.

Объект system

Содержит функции вывода отладочного текста и отключения встроенного курсора мыши (это нужно, если игра реализует собственный курсор).

Функции объекта system

```
writeMessage(msg)
```

Выводит в текстовое поле окна отладочное сообщение. Параметр - текст сообщения.

```
showCursor(show)
```

Позволяет включить или отключить показ системного курсора в окне игры. Параметр - значение true или false.

```
writePair(key,value)
```

Выводит отладочную информацию в окно программы в виде пары "ключ-значение". Если повторно вызывать с тем же ключом, то значение в окне заменяется. Это позволяет организовать наблюдение за переменными.

```
setInterval(code,ms)
```

Устанавливает таймер, который вызывает код, переданный строкой, каждое заданное число миллисекунд. Для разового выполнения кода, используйте функцию setTimeout.

```
setTimeout(code,ms)
```

Устанавливает таймер, который один раз вызывает код, переданный строкой, спустя заданное число миллисекунд. Для регулярного выполнения кода, используйте функцию setInterval.

```
saveObject(filename, value)
```

Позволяет записать в файл JSON заданный объект или массив. Параметры - имя файла и объект. Пример - мы создаём объект var state1 = { x: 1, msg:«OK», q:true }, и записывая его в файл, получим содержимое

{
"msg": "OK",
"q": true,
"x": 1
}

это позволяет сохранять данные из игры на диск.

```
loadObject(filename)
```

Считывает из файла JSON объект или массив и возвращает его. Параметр - имя файла. Работает как обратная функция для saveObject, восстанавливая объект или массив. Это позволяет как считывать ранее сохраненные данные, так и загружать в код данные из JSON-файлов, чтобы не перегружать код игры текстовыми значениями.

```
loadObjectFromAppData(filename)
```

Работает аналогично loadObject, но вместо каталога игры, работает с каталогом локальных данных в каталоге пользователя (обычно это AppData\Local). Это позволяет записывать файлы на диск, даже если сама игра установлена в каталог без прав записи, например, Program Files.

```
saveObjectToAppData(filename, value)
```

Работает аналогично saveObject, но вместо каталога игры, работает с каталогом локальных данных в каталоге пользователя (обычно это AppData\Local).

Загружаемые объекты ресурсов

Функции загрузки объектов создают объекты ресурсов, которые далее можно использовать в коде игры.

Объект спрайта

Содержит графический спрайт, создается методом loadSprite, может выводиться в заданное место на экране.

Функции объекта спрайта

```
renderTo(x,y)
```

Выводит спрайт со всеми установленными параметрами в окно игры. Параметры - координаты вывода спрайта относительно его начальной точки (см. функцию setHotSpot)

```
render()
```

Выводит спрайт со всеми установленными параметрами в окно игры, используя текущее положение x и y.

```
setXY(x,y)
```

Устанавливает для спрайта положение на экране. Параметры - координаты вывода спрайта относительно его начальной точки (см. функцию setHotSpot)

```
setScale(scale)
```

Устанавливает масштаб спрайта. Параметр - масштаб объекта относительно исходного в процентах.

```
setScaleX(scale)
```

Устанавливает масштаб спрайта по оси X. Параметр - масштаб объекта относительно исходного в процентах.

```
setScaleY(scale)
```

Устанавливает масштаб спрайта по оси Y. Параметр - масштаб объекта относительно исходного в процентах.

```
setAngle(angle)
```

Устанавливает поворот спрайта. Параметр - угол поворота спрайта вокруг его начальной точки (см. функцию setHotSpot) в радианах.

```
setSmooth(bool)
```

Устанавливает использование интерполяции пикселей при масштабировании. По умолчанию, интерполяция включена. Отключение имеет смысл для пиксельных игр в стиле старой графики.

```
setAlpha(alpha)
```

Устанавливает прозрачность спрайта. Параметр - значение прозрачности от 0 до 255, где 0 - это полностью прозрачный объект, 255 - полностью непрозрачный.

```
setHotSpot(x,y)
```

Устанавливает начальную точку спрайта, относительно которой работают операции вывода спрайта на экран и поворота на угол. Параметры - координаты точки в пикселях от левого верхнего угла спрайта. По умолчанию, функция loadSprite устанавливает после загрузки спрайта его точку в геометрический центр, таким образом, поворот идет сразу вокруг центра.

```
mirrorHorz(mirror)
```

Устанавливает отражение спрайта по горизонтали. Параметр - true или false, если передано true, то объект разворачивается при выводе зеркально слева направо.

```
mirrorVert(mirror)
```

Устанавливает отражение спрайта по вертикали. Параметр - true или false, если передано true, то объект разворачивается при выводе зеркально сверху вниз.

```
setBordered(bordered)
```

Включает или отключает отображение белой рамки толщиной в один пиксель вокруг спрайта. Рамка при отображении учитывает все изменения спрайта, масштабы и повороты.

```
setContactModelBox()
```

Устанавливает прямоугольную модель проверки коллизии для спрайта.

```
setContactModelCircle()
```

Устанавливает эллиптическую модель проверки коллизии для спрайта.

```
isContactWith(sprite)
```

Возвращает наличие коллизии спрайта и спрайта, переданного как параметр. Модель проверки коллизии устанавливается процедурами setContactModelBox и setContactModelCircle, по умолчанию равна прямоугольной. При использовании прямоугольной модели - коллизия возникает, если прямоугольники спрайтов пересекаются. При использовании эллиптической модели - коллизия возникает, если расстояние между центрами спрайтов меньше, чем сумма условных радиусов эллипсов, в которые вписывается спрайт. Для спрайтов визуально круглой формы (шары) целесообразно

использовать эллиптическую модель, для спрайтов квадратных (коробки) - лучше работает прямоугольная модель. Обе модели учитывают все изменения спрайтов, масштабы и повороты.

```
isPointIn(x,y)
```

Возвращает вхождение точки в спрайт, с учётом всех изменений спрайтов, масштабы и повороты. Параметры - координаты проверяемой точки.

```
setTag(tagname,tagvalue)
```

Устанавливает для спрайта тэг с именем tagname в значение tagvalue. tagname должен быть строкой, а tagvalue может быть любым, включая примитивы, массивы и объекты с полями.

```
getTag(tagname)
```

Возвращает тэг с именем tagname, заданный ранее.

```
getWidth()
```

Возвращает ширину спрайта.

```
getHeight()
```

Возвращает высоту спрайта.

```
convertPixels(funcname)
```

Преобразует пиксели спрайта через переданную по названию функцию. Функция должна быть доступна в файле кода JavaScript, принимать аргумент-объект с компонентами цвета (r,g,b,a) и возвращать объект с полями-компонентами цвета (r,g,b,a). Пример функции, которая делает спрайт серым.

```
function funcGray(c) {
if (c.a==0) return c ;
var gr = (c.r+c.g+c.b)/3 ;
return { r: gr, g: gr, b: gr, a: 255 } ;
}
```

и её вызов для спрайта

```
mysprite.convertPixels("funcGray")
```

Объект анимации

Содержит анимацию, создается методами loadAnimationFromFiles и loadAnimation. Он поддерживает все методы объекта-спрайта, а также несколько специфичных методов для анимации

Функции объекта анимации

```
play()
```

Запускает бесконечный цикл анимации объекта.

```
playOneTime()
```

Запускает однократный цикл анимации объекта, по завершении, анимация останавливается.

```
stop()
```

Останавливает цикл анимации объекта.

```
isPlayed()
```

Возвращает true, если анимация в данный момент выполняется, и false в противном случае.

Объект надписи

Содержит надпись, создается методом loadText, может выводиться в заданное место на экране.

Функции объекта надписи

```
printTo(x,y)
```

Выводит надпись со всеми установленными параметрами в окно игры. Параметры - координаты вывода текста относительно его левого верхнего угла.

```
setText(text)
```

Устанавливает текст надписи. Параметр - текст. Возможно использовать в тексте Escape-последовательность \n для установки переноса текста в надписи.

```
setSize(size)
```

Устанавливает размер текста надписи. Параметр - размер.

```
setColor(r,g,b)
```

Устанавливает цвет текста надписи. Параметры - компоненты цвета текста в RGB

```
setAlignCenter()
```

Устанавливает выравнивание текста по центру относительно точки вывода. По умолчанию, вывод идет от верхнего левого угла. После включения центровки, центруется по горизонтали.

Объект звука

Содержит звуковой эффект, создается методом loadSound, может воспроизводиться.

Функции объекта звука

```
play()
```

Воспроизводит звуковой эффект от начала до конца, один раз. Не приводит к остановке выполнения программы на время воспроизведения, звук запускается в отдельном потоке! Также звуки могут накладываться друг на друга, если вызывать play() чаще, чем длина звука.

```
stop()
```

Останавливает воспроизведение звука.

```
isPlayed()
```

Возвращает логическое значение, соответствующее статусу звука - воспроизводится или нет в данный момент.

```
setLoop(isloop)
```

Устанавливает тип воспроизведения - если true, то звук будет крутиться в кольце. По умолчанию, звук воспроизводится только один раз.

Объект линии

Содержит линию толщиной в один пиксель, создается методом createLine, может выводиться в заданное место на экране.

Функции объекта линии

```
drawTo(x1,y1,x2,y2)
```

Выводит линию со всеми установленными параметрами в окно игры. Параметры - координаты начала и конца линии.

```
setColor(r,g,b)
```

Устанавливает цвет линии. Параметры - компоненты цвета линии в RGB

Объект прямоугольника

Содержит прямоугольник заданного цвета, создается методом createRect, может выводиться в заданное место на экране.

Функции объекта прямоугольника

```
drawTo(x,y,w,h)
```

Выводит прямоугольник со всеми установленными параметрами в окно игры. Параметры - координаты верхнего левого угла, ширина, высота.

```
setFillColor(r,g,b)
```

Устанавливает цвет заливки прямоугольника. Параметры - компоненты цвета линии в RGB.

```
setBorderColor(r,g,b)
```

Устанавливает цвет рамки прямоугольника. Параметры - компоненты цвета линии в RGB.

```
setLineWidth(w)
```

Устанавливает толщину рамки прямоугольника. Параметр - толщина рамки в пикселях.

Приложение 1. Особенности и ограничения

- Размер игрового окна: 800x600 пикселей.
- Кодировка файлов скриптов: UTF-8
- Кодировка сохраняемых и загружаемых файлов JSON: UTF-8
- Сохранение данных функцией saveObject - позволяет записать либо массив простых типов, либо объект с полями, содержащими простые типы. Чтение же функцией loadObject - позволяет прочитать в память JSON-объект любой сложности и структуры.
- Для использования шрифта, его обязательно нужно скопировать из %WINDIR%/Fonts в каталог fonts.
- Звуковые файлы должны иметь форматы WAV или OGG. MP3 не поддерживается, рекомендуется выполнить преобразование в несжатый WAV.
- Файлы спрайтов могут иметь форматы JPG, PNG, GIF, BMP, TGA (формат GIF поддерживается как статичный - если загружаем анимированный спрайт, то будет выводиться только первый кадр)
- Язык программирования файла main.js совместим с языком JavaScript, и выполняет стандарт ECMAScript (подробнее см. на https://ru.wikipedia.org/wiki/ECMAScript)

Приложение 2. Список констант клавиш

Для удобства разработки игры, движок определяет набор констант кодов большинства клавиш. Их можно использовать в функции isKeyDown по имени константы.

```
KEY_ESCAPE
KEY_TAB
KEY_ENTER
KEY_LEFT
KEY_UP
KEY_RIGHT
KEY_DOWN
KEY_SHIFT
KEY_CONTROL
KEY_ALT
KEY_F1
KEY_F2
KEY_F3
KEY_F4
KEY_F5
KEY_F6
KEY_F7
KEY_F8
KEY_F9
KEY_F10
KEY_F11
KEY_F12
KEY_SPACE
KEY_PLUS
KEY_MINUS
KEY_0
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6
KEY_7
KEY_8
KEY_9
KEY_A
KEY_B
KEY_C
KEY_D
KEY_E
KEY_F
KEY_G
KEY_H
KEY_I
KEY_J
KEY_K
KEY_L
KEY_M
KEY_N
KEY_O
KEY_P
KEY_Q
KEY_R
KEY_S
KEY_T
KEY_U
KEY_V
KEY_W
KEY_X
KEY_Y
KEY_Z
```