**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**КУРСОВОЙ ПРОЕКТ**

**по дисциплине «Программирование»**

**Тема: «Разработка аркадного космического симулятора»»**

Студент гр. О – 21 – ИВТ2 – ПО – Б

Бахтин Г. А.

№ зач. книжки 21.0285

Преподаватель

Радченко А. О.

**БРЯНСК 2022**

Содержание

# Введение

В наши дни нельзя найти человека, который не слышал про компьютерные игры. Возникнув как незамысловатый плод творческой мысли программистов, они стали настоящим культурным феноменом. С каждым годом они набирают всё большую популярность. По всему миру выросли компании по разработке игр. Некоторые игровые серии стали чрезвычайно знамениты – например, Metal Gear, GTA, DOOM, Fallout, Final Fantast, Starcraft, Need for Speed.

Самым первым игровым жанром стала аркада. Этот жанр имеет нарочно примитивный, но интенсивный игровой процесс. К аркадным играм относят те, которые были изначально на игровых автоматах, этим и объясняется их маленькая продолжительность игрового времени. К ним причисляют такие жанры, как «файтинг» (fighting), часть жанра «гонки» (racing), часть жанра «шутер» (shooter). Больше всего аркадные игры распространены на игровых приставках, ведь устройства управления игровых автоматов имеет больше сходства с контроллерами приставок, чем с компьютерными клавиатурами и мышками. Такие игры подразумевают простое и интуитивно-понятное управление игровыми объектами.

Я расскажу про одну из самых первых игр, когда-либо сделанных человечеством – аркадным космическим симулятором «Spacewar!».

# Аналитическая часть

## Анализ Предметной области

*«Spacewar!»* - одна из первых компьютерных игр в мире.

### История возникновения

Космический симулятор – жанр компьютерных игр, воспроизводящих с различной степенью достоверности управление космическим кораблём.

Жанр космических симуляторов делится на следующие поджанры: управление космическим кораблём, боевые, торговые.

В аркадных космических симуляторах физическая модель обычно не соответствует реальной физике космического аппарата. Характерно поведение корабля как при наличии внешней среды, обеспечивающей существенное сопротивление.

### Spacewar!

*«Spacewar!»* – одна из первых известных цифровых компьютерных игр. Создана Стивом Расселом при участии Мартина Греца и Уэйна Витенена. Игра была задумана в 1961 году в Массачусетском технологическом институте. К февралю 1962 года была готова первая версия *«Spacwar!»*.

В *«Spacewar!»* Участвуют двое игроков, каждый из которых управляет своим космическим кораблём. В начале на борту каждого есть некоторый запас топлива для маневрирования и некоторое число торпед. Цель игры: уничтожить корабль соперника.



Рис. 1. Игра «Spacewar!»

В игре присутствуют следующие механики:

* Два игрока
* Стрельба
* Расход топлива во время работы двигателя
* Физика кораблей частично основывается на ньютоновской механике: они остаются в движении, если игрок не использует двигатели, но инерция вращения отсутствует
* Столкновение со звездой
* Гравитационное воздействие звезды на корабль
* Гравитационный манёвр
* Гиперпространство: перемещение корабля в случайное место. Корабль разобьётся, если он пересечёт какой-либо объект
* Вращение корабля по часовой и против часовой стрелки
* Включение тяги вперёд
* Края экрана завёрнуты. При перемещении корабля за край экрана, он появляется с другой стороны

### Blasteroids

Игровой процесс состоит в управлении игроком или игроками космическими кораблями. Корабль может трансформироваться тремя способами: «*Speeder*» — самый быстрый вариант, «*Fighter*» — вариант с наибольшей огневой силой и «*Warrior*» — вариант с дополнительной бронёй.

Игрок должен уничтожить все астероиды, которые пролетают через игровое поле с определённой скоростью. Астероиды бывают разных размеров. Большие астероиды при попадании разваливаются на несколько маленьких. Полностью уничтожаются при попадании только самые маленькие астероиды.

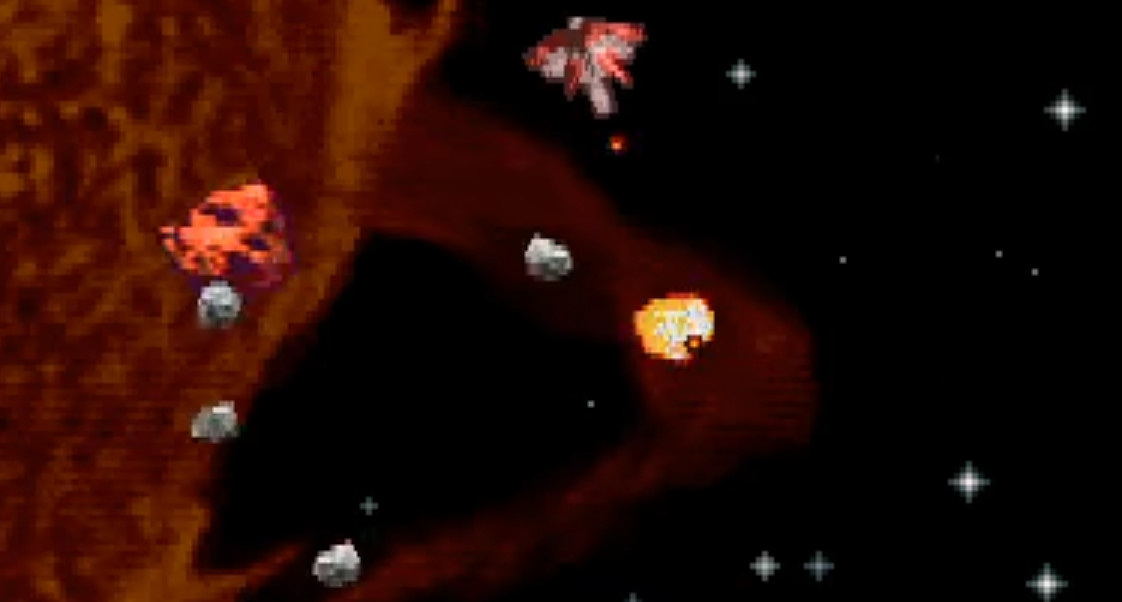


Рис. 2. Игра «Blasteroids»

В игре присутствуют следующие механики:

* Столкновение с астероидами
* Гравитационное воздействие астероидов на корабль
* Вращение текстур астероидов
* Дробление астероидов
* Вращение корабля по часовой и против часовой стрелки
* Трансформация корабля
* Инерция движения
* Несколько типов оружий
* В мультиплеере корабли игроков могут сталкиваться друг с другом
* Вражеские корабли
* Вражеские планеты
* Прохождение на новые уровни реализовано в виде гиперпространств
* Шкалы счётчика, уровня щита и энергии
* Края экрана завёрнуты

### Orbitwar.space

*«Orbitwar.space»* – многопользовательская игра для браузера. Цель игры: набрать как можно больше очков, уничтожая корабли соперников.



Рис. 3. Игра «Orbitwar.space»

В игре присутствуют следующие механики:

* Мультиплеер
* Стрельба разными видами оружия
* Направление корабля задаётся позицией курсора мыши
* Бонусы в виде частиц
* Инерция движения
* Столкновение кораблей
* Столкновение корабля с астероидами
* Шкала очков
* Края экрана завёрнуты

### Сравнительная таблица

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Игры \ Механики | Кол. игроков | Стрельба | Противники | Уровни | Инерция движения | Грав. возд. | Завёрнутый экран | Шкалы | Трансформация корабля |
| Spacewar! | 2 | Один тип | - | - | + | + | + | - | - |
| Blasteroids | 1 - мульти | Несколько типов | Другие игрок, планеты | + | + | - | + | + | + |
| Orbitwar.space | мульти | Несколько типов | Другие игроки | - | + | - | + | + | + |

## Вывод

Во всех играх присутствую механики стрельбы, инерции движения, завёрнутого экрана, астероидов, то есть они являются фундаментальными для аркадных космических симуляторов.

Кроме основных механик, планируется включить в игру: магазин, меню, уровни, шкалу очков.

# Конструкторская часть

## Архитектура приложения

### Инструменты

Приложение было написано на языке программирования C с использованием среды разработки *Microsoft Visual Studio* и следующих библиотек:

* *SDL*
* *SDL\_mixer*
* *SDL\_image*
* *SDL\_ttf*

### Блок-схема проекта

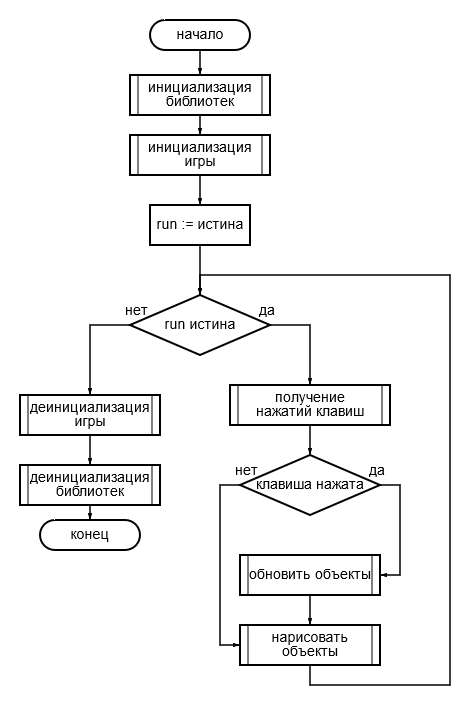


Рис. 4. Блок-схема проекта

## Структуры данных

Основные структуры данных игры.

### Структуры главных объектов

Структуры, которые составляют главную часть программы.

Листинг 1

***Структура Game***

struct Game

{

bool run = false;

bool aboutInited = false;

int state = 0;

SDL\_Event event;

Keys keys;

Menu menu;

Levels levels;

Texture messageTexture;

Texture background;

Texture particles[PARTICLES\_FILENAMES\_NUM];

Asteroids asteroids;

Ship ship1;

Ship ship2;

Enemy enemy;

Music music;

};

Структура Game является самой главной, все основные объекты находятся в ней.

Листинг 2

Структура Keys

struct Keys

{

bool w = false;

bool s = false;

bool a = false;

bool d = false;

bool space = false;

bool left = false;

bool right = false;

bool up = false;

bool down = false;

bool rctrl = false;

bool enter = false;

bool escape = false;

bool leftClick = false;

int mouse\_x;

int mouse\_y;

};

Структура Keys содержит булевы значения состояний нажатия клавиш, требуемые для работы игры, и целые числа для координат курсора.

Листинг 3

***Структура Asteroid***

struct Asteroid

{

SDL\_Point pos; // координаты астероида

SDL\_Rect srcrect; // srcrect и dstrect нужны для смены кадров

SDL\_Rect dstrect; // картинки

Vec vel; // вектор скорости

int type; // тип астероида

int frame; // кадр анимации в картинке

int ticks; // последнее время обновления

int health; // значение здоровья

Asteroid\* next; // указатель на предыдущий астероид, если он есть

Asteroid\* prev; // указатель на следующий астероид, если он есть

};

Структура Asteroid является элементом списка Asteroids.

Описание полей структуры:

* *pos* – координаты астероида
* *srcrect* и *dstrect* нужны для смены кадров
* *vel* – вектор скорости
* *type* – тип астероида
* *frame* – кадр анимации в картинке
* *ticks* – последнее время обновления
* *­health* – значение здоровья
* *next* – указатель на предыдущий астероид, если он есть
* *prev* – указатель на следующий астероид, если он есть

Листинг 4

***Структура Asteroids***

struct Asteroids

{

Animation explosion;

SDL\_Point explosionPos;

Texture texture[ASTEROIDS\_TYPE\_NUM];

int frames[ASTEROIDS\_TYPE\_NUM];

Asteroid\* head = 0;

};

Описание полей структуры:

* *explosion* – структура *Animation*
* *explosionPos* – позиция анимации взрыва
* *texture* – массив с загруженными картинками астероидов
* *frames* – массив, содержащий количество кадров картинок астероидов
* *head* – указатель на первый элемент списка астероидов

Листинг 5

***Структура Bullet***

struct Bullet

{

Vec vel;

SDL\_Point pos;

int type;

int ticks;

int affiliation;

Bullet\* next;

Bullet\* prev;

};

Описание полей структуры:

* *type* – тип пули

Листинг 6

***Структура Bullets***

struct Bullets

{

Bullet\* head = 0;

Texture texs[BULLETS\_TYPE\_NUM];

int ticks = 0;

};

Имеющиеся поля структуры идентичны полям структуры *Asteroids*.

Листинг 7

***Структура Ship***

struct Ship

{

ShipActions acts;

Animation engine;

Texture tex;

Bullets bullets;

Health health;

Score score;

float angularVel;

int rad;

int ticks = 0;

int instance;

int bulletType;

bool active;

Vec vel = { 0, 0 };

Vec acc = { 0, 0 };

};

Описание полей структуры:

* *acts* – структура, которая содержит булева значения действий корабля (движение прямо, повороты, выстрел и так далее)
* *engine* – структура *Animation*
* *tex* – структура *Texture*, содержащая текстуру корабля
* *bullets* – массив пуль
* *health* – структура *Health*, отвечающая за здоровье
* *score –* структура Score, отвечающая за набор очков
* *angularVel –* скорость вращения корабля
* *rad –* радиус столкновения корабля с другими объектами
* *instance –* определяет к какому игроку принадлежи корабль
* *bulletType –* тип пули
* *active –* активность корабля
* *vel –* вектор скорости корабля
* *acc –* вектор ускорения корабля

Листинг 8

***Структура Enemy***

struct Enemy

{

Texture tex;

Bullets bullets;

Health health;

Vec vel = { 0, 0 };

Vec acc = { 0, 0 };

int damage = ENEMY\_DAMAGE;

int ticks;

int damageTicks;

bool active;

};

Описание полей структуры:

* *damage* – какой урон может нанести враг игроку
* *damageTicks* –время с прошлого нанесения урона игроку

Листинг 9

***Структура Animation***

struct Animation

{

Texture\* textures;

int frame;

int ticks;

};

Описание полей структуры:

* *damage* – какой урон может нанести враг игроку
* *damageTicks* –время с прошлого нанесения урона игроку

### Вспомогательные структуры

Структуры, которые часто содержатся в других структурах.

Листинг 10

***Структура Animation***

typedef SDL\_FPoint Vec;

Является синонимом *SDL\_FPoint* и содержит вещественные числа – *x*, *y*.

Листинг 11

***Структура Texture***

struct Texture

{

SDL\_Texture\* tex = NULL;

SDL\_Rect dstrect;

double angle = 0;

};

Описание полей структуры:

* *tex* – указатель на текстуру в памяти
* *dstrect* – структура прямоугольника, который содержит верхнюю левую точку прямоугольника и его стороны
* *angle* – значение отклонения угла от начального положения (в градусах)

Листинг 12

Структура Animation

struct Animation

{

Texture\* textures;

int frame;

int ticks;

};

Описание полей структуры:

* *textures* – массив структур *Texture*

## Основные функции

Одной из самых интересных функций является загрузка уровней из файла (см. листинг 13). Сначала происходит чтение буфера файла, и если это действие прошло не удачно – выдаётся ошибка и программа завершает свою работу. После в цикле читаются строки, в зависимости от их содержания меняются состояния, которые указывают на действия, которые необходимо совершить с полученными данными. Затем происходит закрытие файла и возвращение из функции заполненных данных.

Листинг 13

Функция LevelLoadFile

Levels LevelLoadFile(const char\* filename)

{

FILE\* f;

if (fopen\_s(&f, filename, "r"))

{

char message[120];

sprintf\_s(message, 120, "Error while trying to read file with filename: %s in LevelLoadFile", filename);

logError(message);

}

Levels levels;

Level levelInfo = LEVEL\_STANDART\_INFO;

LevelState state = LEVEL\_STATE;

LevelState tmpState = state;

while (!feof(f))

{

char string[120];

fscanf\_s(f, "%s", string, 120);

state = readState(state, string);

if (state != tmpState)

{

tmpState = state;

continue;

}

if (state == LEVEL\_NUM)

{

levels.levels = (Level\*)realloc(levels.levels, sizeof(Level) \* (levels.num + 1));

if (levels.num)

levels.levels[levels.num-1] = levelInfo;

levelInfo = LEVEL\_STANDART\_INFO;

levels.num++;

}

processState(state, string, levelInfo);

}

levels.levels[levels.num-1] = levelInfo;

fclose(f);

return levels;

}

Следующей интересной функцией является *ExplosionUpdate* (см. листинг 14). Если кадр картинки взрыва равен 0, произойдёт выход из функции, и если не прошло достаточно времени, случится тоже самое. Если же ни одного предварительного выхода не произошло, то значение тиков обновится до текущего, а значение кадра будет обновлено.

Листинг 14

Функция ExplosionUpdate

void ExplosionUpdate(Asteroids& self)

{

if (!self.explosion.frame) return;

int ticks = SDL\_GetTicks();

if (ticks - self.explosion.ticks < EXPLOSION\_DELAY) return;

self.explosion.ticks = ticks;

self.explosion.frame = self.explosion.frame < EXPLOSION\_FRAMES - 1 ? ++self.explosion.frame : 0;

}

Функция *TextureDrawAsInfiniteImage* не менее интересна (см. листинг 15). Она отвечает за симуляцию бесконечного заднего фона для игры. Переменные целого типа *wdt* и *hgt* играют очень важную роль. Если текстура изображения зашла за окно программы с левой стороны, то *wdt* будет равна

Листинг 15

Функция ExplosionUpdate

void TextureDrawAsInfiniteImage(Texture& self)

{

SDL\_Rect rect;

int wdt = 0;

int hgt = 0;

if (self.dstrect.x < 0)

wdt = self.dstrect.w;

else if (self.dstrect.x > 0)

wdt = -self.dstrect.w;

if (self.dstrect.y < 0)

hgt = self.dstrect.h;

else if (self.dstrect.y > 0)

hgt = -self.dstrect.h;

drawTextureWithOffset(self);

if (wdt)

drawTextureWithOffset(self, { wdt, 0 });

if (hgt)

drawTextureWithOffset(self, { 0, hgt });

if (wdt && hgt)

drawTextureWithOffset(self, { wdt, hgt });

}

Функция kajsbdlasbd не менее интересна (см. ).

# Заключение

Мы познакомились с замечательным, очень мощным Python модулем itertools, который позволяет упростить реализации некоторых алгоритмов. Знание этого модуля выводит код программиста на средний уровень, так как зачастую код становится более понятным и читаемым, увеличивая при этом скорость.

В реферате мы рассмотрели самые полезные и чаще встречающиеся функции этого модуля и написали практический скрипт.

# Список Литературы

1. Официальный сайт Python. [Электронный ресурс]. – Режим доступа: <https://www.python.org>. (Дата обращения 15.05.2022).
2. Веб–сервис «kpolyakov.spb.ru». [Электронный ресурс]. – Режим доступа: https://kpolyakov.spb.ru/school/ege/generate.htm. (Дата обращения 15.05.2022).