содержание

[ВВЕДЕНИЕ 4](#_Toc438248399)

[1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ 5](#_Toc438248400)

[2 Проектирование программы 6](#_Toc438248401)

[3 РЕАЛИЗАЦИЯ ПРОГРАММЫ 8](#_Toc438248402)

[4 ТЕСТИРОВАНИЕ ПРОГРАММЫ 20](#_Toc438248403)

[ЗАКЛЮЧЕНИЕ 24](#_Toc438248404)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc438248405)

[ПРИЛОЖЕНИЕ А. МОДУЛЬНАЯ СХЕМА ПРОГРАММЫ 26](#_Toc438248406)

[ПРИЛОЖЕНИЕ Б. ФУНКЦИОНАЛЬНАЯ СХЕМА ПРОГРАММЫ 27](#_Toc438248407)

[ПРИЛОЖЕНИЕ В. ПРОТОТИП ГРАФИЧЕСКОГО ИНТЕРФЕЙСА 28](#_Toc438248408)

**ВВЕДЕНИЕ**

Java – объектно-ориентированный язык программирования общего назначения. Особенностью языка является компиляция программ в специальный промежуточный код (байт-код), который затем интерпретируется виртуальной машиной Java (JVM). Преимуществом данного подхода является отличная переносимость программ, так как программы могут быть запущены без изменений на любой системе, под которую написана виртуальная машина. Также достоинством является безопасность выполнения кода, так как JVM контролирует выполнение программы и прерывает ее выполнение при превышении допустимых полномочий.

Недостатками данного подхода является снижение производительности из-за интерпретации байт-кода виртуальной машиной во время выполнения программ. Однако данная проблема частично решена благодаря применению JIT-компиляции и платформенно-ориентированного кода в стандартной библиотеке.

Также особенностью языка Java является использование сборщика мусора. Он представляет собой подпрограмму JVM, который выполняет удаление из памяти неиспользуемых объектов. Такой подход облегчает разработку программ, так как разработчику не приходится самому следить за использованием памяти и вручную удалять неиспользуемые объекты.

Android – это операционная система предназначенная для смартфонов, планшетов, различных носимых устройств, нетбуков и телевизоров. Основным разработчиком является компания Google. Android позволяет создавать Java-приложения, взаимодействующие с устройством через разработанные компанией Google библиотеки. Данная операционная система основана на ядре Linux и реализации JVM от Google. Она является наиболее распространенной на мобильных платформах, и разработка приложений для нее очень актуальна.

Целью выполнения данной курсовой работы является систематизация и закрепление теоретических знаний, полученных за время обучения, а также приобретение практических навыков в таких сферах разработки как проектирование программы, поиск и реализация необходимых алгоритмов, тестирование программы и разработка документации.

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ

Темой данной курсовой работы является разработка программы для анализа статистики игроков в игре World of Tanks на языке программирования Java для операционной системы Android.

Данное приложение должно обладать следующим функционалом:

* Поиск игрока по имени
* Отображение общей статистики по аккаунту;
* Отображение статистики игрока по технике игрока с выводом средних значений;
* Возможность сортировки техники по различным показателям;
* Отображение специальных рейтингов РЭ, WN6, WN8;
* Отображение общей сессионной статистики;
* Отображение сессионной статистики отдельно по каждому танку;
* Построение графиков динамики изменения средних показателей по каждому танку;
* Построение графиков динамики изменения средних показателей по аккаунту;
* Графики статистики по каждому танку;
* Возможность вывода статистики по танкам с определенным количеством боев;
* Хранение истории статистики и поиска.

Данные получаются с серверов компании Wargaming, отдельно для каждого региона с помощью Wargaming API путем отправки HTTP-запросов. Результаты запроса возвращаются в формате JSON.

Для разработки приложения нам понадобится:

* Среда разработки IntelliJ IDEA 15.0.3;
* Enterprise Architect для построения диаграммы вариантов использования.
* Смартфон с установленной системой Android 5.0 и выше либо аналогичный эмулятор для тестирования работы приложения.

**2 программНОЕ ПРОЕКТИРОВАНИЕ**

Для начала реализации проекта нам необходимо создать и настроить проект Android приложения в среде разработки IntelliJ IDEA. При разработке будут использованы библиотеки android-support-v4 и android-support-v13, которые содержат классы, облегчающие реализацию интерфейса с использованием вкладок. Также нам необходимо скачать библиотеку MPAndroidChartLibrary. Она содержит богатый функционал для построения графиков и диаграмм различных типов. Для обработки ответов от сервисов Wargaming будет использоваться библиотека com.parse, которая позволяет работать с JSON объектами. Интерфейс программы для операционной системы Android будет построен на основе XML разметки.

При первом запуске должно будет выводиться окно, с предложением выбрать игрока. Для этого необходимо будет щелкнуть на кнопку в правом верхнем углу экрана. Затем после выбора игрока отобразится окно «Статистика игрока», являющееся главным окном программы. Данное окно будет содержать две вкладки:

* Общая статистика;
* Техника игрока.

Переключение между различными меню будет осуществляется с помощью выдвигающегося бокового меню. Там будет содержаться шесть пунктов:

* Статистика игрока
* Динамика статистики
* Сессии
* История статистики
* Инфографика
* Графики по технике

Если был осуществлен первый запуск, то при выборе пунктов меню «Динамика статистики» и «Сессии» будет выведено предложение сыграть несколько боев для построения графиков. Иначе на вкладке «Динамика статистики» будут выведены графики, показывающие динамику изменения статистики игрока. Различные показатели будут вынесены в отдельные вкладки. А во вкладке «Сессии» будут отображаться подробные показатели игрока за сессии, сыгранные в определенный период времени.

В пункте меню «История статистики» будет отображаться упрощенный список сессий игрока, который позволит быстро проследить количественное изменение ключевых показателей статистики.

На вкладке «Инфографика» будут представлены диаграммы, показывающие распределение различных показателей статистики по различным критериям.

Пункт меню «Графики по технике» будет содержать графики, которые будут визуально представлять статистику игрока на каждом отдельном танке.

Для обновления статистики будет необходимо открыть пункт меню «Статистика игрока».

* Вся информация будет храниться в базе данных SQLite, которая изначально поддерживается системой Android. Для работы с базой данных будет создан отдельный класс. База данных будет содержать таблицы для хранения данных об аккаунтах, игровых сессиях, общей статистике игрока на отдельных танках, последней сессии игрока на танках. Также там будут храниться информация о танках и средние показатели игроков на всех танках.

Для каждого пункта меню и окна выбора игрока будут созданы отдельные классы и соответствующие им XML документы, содержащие разметку графического интерфейса.

Также необходимо создать класс, который будет взаимодействовать с сервисами Wargaming. Он будет отвечать за отправку запросов на получение данных и принимать ответы в формате JSON, которые затем будут обработаны с помощью библиотеки com.parse и добавлены в базу данных.

Также будут созданы вспомогательные классы, представляющие различные составляющие элементы интерфейса приложения.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

**3.1 Детальная реализация функциональных частей ПО**

Ключевым классом является класс MainActivity. Он отвечает за первоначальную инициализацию приложения, создание меню и управляет переключением между различными пунктами меню.

Основные методы данного класса:

* onCreate
* onSectionAttached
* onActivityResult
* onOptionsItemSelected

Также внутри класса MainActivity содержится класс PlaceHolderFragment, который отвечает за передачу сообщения о том, что в меню был какой-либо пункт.

Метод onCreate вызывается при запуске приложения и выполняет начальную инициализацию. Он представлен в листинге 1:

1. protected void onCreate(Bundle savedInstanceState) {
2. super.onCreate(savedInstanceState);
3. setContentView(R.layout.activity\_main);
4. Utils.init(this);
5. mNavigationDrawerFragment = (NavigationDrawerFragment) getFragmentManager().findFragmentById(R.id.navigation\_drawer);
6. mTitle = getTitle();
7. dbHelper = DatabaseHelper.createDatabaseHelper(this);
8. SQLiteDatabase sdb = dbHelper.getWritableDatabase();
9. mainPlayer = getString(R.string.ttl\_search\_name);
10. Cursor cursor = sdb.rawQuery("SELECT \* FROM " + DatabaseHelper.DATABASE\_TABLE\_MAIN\_PLAYER + " WHERE active = 1", null);
11. if (cursor.getCount() > 0) {
12. cursor.moveToFirst();
13. mainPlayer = cursor.getString(1);
14. }
15. cursor.close();
16. this.mNavigationDrawerFragment.setUp(R.id.navigation\_drawer, (DrawerLayout) findViewById(R.id.drawer\_layout));
17. }

При реализации код программы был разбит на три модуля.

В модуле **arkanoid.cpp** содержатся следующие функции:

* **int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)**
* **void Init(int& argc, char \*argv[]);**
* **void TimerFunc(int value);**
* **void KeyboardFunc(unsigned char key, int x, int y);**
* **void KeyboardUpFunc(unsigned char key, int x, int y);**
* **void SpecialPressFunc(int key, int x, int y);**
* **void SpecialReleaseFunc(int key, int x, int y);**
* **void MouseMove(int x, int y);**
* **void MouseClick(int button, int state, int x, int y);**
* **void EndGame();**
* **int Cmp(const void\*, const void\*);**
* **void LoadFile();**
* **void SaveFile();**

В функции **int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)** производится начальная инициализации игры и запуск основного игрового цикла

В функции **void Init(int& argc, char \*argv)** производитсязагрузка необходимых библиотек, инициализация и создание главного игрового окна, инициализация функций, которые будут отвечать за вывод изображения на экран, обработку пользовательского ввода, а также производится загрузка необходимых изображений текстур из файлов для дальнейшей работы с ними.

Функция **void TimerFunc(int value)** отвечает за обработку основных игровых состояний и их изменение. Она играет роль таймера, при срабатывании которого проверяется текущее состояние игры, в зависимости от этого состояния происходит необходимая модификация переменных. После этого игровое поле перерисовывается с учетом изменений. В состояниях, отвечающих за начало игры, все переменные инициализируются начальными значениями. При окончании игры воспроизводится звук окончания игры, вызывается функция EndGame, отвечающая за обработку окончания игры, а затем игра переходит в режим отображения таблицы рекордов на экране.

В состоянии начала движения шарика он помещается в центр платформы, и его скорость устанавливается в значение 0. В состоянии процесса игры, вызываются функции, отвечающие за обработку движения шарика и приза. Здесь также происходит обработка активированного выпадающего приза и переход на следующий уровень, если все кирпичи уничтожены. Листинг данной части функции приведен ниже:

Листинг 3.1 – Реализация процесса игры в функции void TimerFunc(int value)

1. void TimerFunc(int value)
2. {
3. //остальной код функции
4. …
5. case GAME\_STATE\_BALL\_START:
6. prize = 0;
7. ball\_x = paddle\_x+(pwidth-bsize)/2.0f,
8. ball\_y = paddle\_y+pheight;
9. ball\_dx = 0.0f, ball\_dy = 0.0f;
10. break;
11. case GAME\_STATE\_RUN:
12. MoveBall();
13. ProcessBall();
14. if (prize) ProcessPrize();
15. textcolor[1] += 0.1f;
16. if ((pbonuscount % 401) == 0)
17. {
18. pwidth -= begpwidth;
19. pbonuscount--;
20. }
21. else if (pbonuscount > 0) pbonuscount--;
22. if (textcolor[1] >= 1.0f) textcolor[1] = 0.5f;
23. textcolor[2] -= 0.1f;
24. if (textcolor[2] <= 0.5f) textcolor[2] = 1.0f;
25. if (blocks\_hit >= NUM\_BLOCK\_ROWS\*NUM\_BLOCK\_COLUMNS)
26. {
27. game\_state = GAME\_STATE\_START\_LEVEL;
28. level++;
29. }
30. break;
31. }

Функции **void KeyboardFunc(unsigned char key, int x, int y)**, **void KeyboardUpFunc(unsigned char key, int x, int y), void SpecialPressFunc(int key, int x, int y) , void SpecialReleaseKey(int key, int x, int y)** выполняют обработку нажатий клавиш на клавиатуре.

Две первых функции вызываются при нажатии и отпускании клавиш, которые генерируют ASCII коды, соответственно. Две последние функции вызываются при нажатии и отпускании клавиш, которые ASCII не генерируют, соответственно. Как параметры функции принимают код нажатой клавиши и координаты мыши в момент нажатия.

Функция **void MouseMove(int x, int y)** вызывается при обнаружении движения мыши по экрану. Параметрами данной функции являются координаты мыши.

Функция **void MouseClick(int button, int state, int x, int y)** срабатывает при щелчке кнопкой мыши. Как параметры функция принимает код клавиши мыши, состояние клавиши(нажата или отпущена), а также координаты мыши.

Функция **EndGame()** вызывается при завершении игры и производит загрузку лучших результатов из файла, их сортировку с текущим результатом по убыванию, и сохранение лучших результатов в файл. При этом в памяти остается отсортированный по убыванию массив из 10 элементов с лучшими результатами, которые в дальнейшем выводятся на экран. Листинг данной функции имеет вид:

Листинг 3.2 – Реализация функции void EndGame()

1. void EndGame()
2. {
3. if (scr)
4. {
5. delete [] scr;
6. scr = NULL;
7. hiscrFileSize = 0;
8. }
9. LoadFile();
10. if (hiscrFileSize == 0) { scr = new int[1]; hiscrFileSize = 1; }
11. scr[hiscrFileSize-1] = score;
12. qsort(scr, hiscrFileSize, sizeof(int), Cmp);
13. if (hiscrFileSize > 10) hiscrFileSize = 10;
14. SaveFile();
15. }

Функция **void LoadFile()** производит загрузку лучших результатов из файла. Листинг данной функции имеет вид.

Листинг 3.3 – Реализация функции void LoadFile()

1. void LoadFile()
2. {
3. ifstream in;
4. in.open("scores.bin", ios::in | ios::binary);
5. if (!in)
6. {
7. ofstream out;
8. out.open("scores.bin", ios::out | ios::binary);
9. if (!out) return;
10. out.close();
11. hiscrFileSize = 0;
12. return;
13. }
14. in.seekg(0, ios::end);
15. hiscrFileSize = in.tellg()/sizeof(int);
16. in.seekg(0, ios::beg);
17. hiscrFileSize++;
18. scr = new int[hiscrFileSize];
19. in.read((char\*)(scr), hiscrFileSize\*sizeof(int));
20. in.close();
21. }

Функция **void SaveFile()** сохраняет лучшие результаты в файл. Листинг данной функции имеет вид:

Листинг 3.4 – Реализация функции void SaveFile()

1. void SaveFile()
2. {
3. ofstream out("scores.bin", ios::out | ios::binary);
4. if (!out) return;
5. out.write((char\*)scr, hiscrFileSize\*sizeof(int));
6. out.close();
7. }

Функция int Cmp(const void\* val1, const void\* val2) производит сравнение элементов val1 и val2 и используется библиотечной функцией qsort.

Модуль **objects.cpp** реализует основную игровую логику. В нем реализован дополнительный тип данных **Block,** в котором хранятся свойства кирпичей. Его листинг имеет следующий вид:

Листинг 3.5 – Структура Block

1. struct Block
2. {
3. GLfloat left;
4. GLfloat top;
5. GLfloat right;
6. GLfloat bottom;
7. BOOL active;
8. GLuint score;
9. GLuint hp;
10. GLuint tex;
11. GLuint prizetex;
12. };

В модуле **objects.cpp** реализуютсяследующие функции:

* **void MoveBall();**
* **void ProcessBall();**
* **GLuint HitDetection(GLfloat cx1, GLfloat cy1, GLfloat width1, GLfloat height1, GLfloat cx2, GLfloat cy2, GLfloat width2, GLfloat height2);**
* **void InitBlocks();**
* **void ProcessPrize();**

Функция **void MoveBall()** отвечает за движение шарика по игровому полю и его отталкивание от стен. Она взаимодействует с остальной частью программы посредством глобальных переменных. Для движения шарика просто изменяются его координаты на величину скорости. Если шарик сталкивается со стеной, то направление его скорости изменяется в соответствии со стеной, с которой он столкнулся. Листинг данной функции имеет вид:

Листинг 3.6 – Реализация функции MoveBall()

1. void MoveBall()
2. {
3. if (ball\_x > windowWidth-bsize || ball\_x < -windowWidth)
4. {
5. ball\_dx = -ball\_dx;
6. }
7. if (ball\_y > windowHeight-bsize || ball\_y < paddle\_y-5)
8. {
9. if (ball\_y < paddle\_y-5)
10. {
11. sndPlaySound("sounds/lose.wav", SND\_ASYNC);
12. lives-=1;
13. game\_state = GAME\_STATE\_BALL\_START;
14. if (lives == 0)
15. {
16. game\_state = GAME\_STATE\_SHUTDOWN;
17. }
18. return;
19. }
20. ball\_dy = -ball\_dy;
21. ball\_y += ball\_dy;
22. }
23. ball\_x += ball\_dx;
24. ball\_y += ball\_dy;
25. if (ball\_x > (windowWidth-bsize+ball\_dx))
26. ball\_x = windowWidth - bsize - 1;
27. else if (ball\_x < -(windowWidth+ball\_dx))
28. ball\_x = -windowWidth - 1;
29. if (ball\_y > windowHeight)
30. ball\_y = windowHeight - bsize-1;
31. }

Функция **void ProcessBall()** реализует отталкивание шарика от кирпичей и платформы. Вначале проверяется, столкнулся ли шарик с платформой посредством функции **HitDetection**. Если результат положительный, то происходит изменение направления движения шарика, в результате которого шарик отталкивается. Также воспроизводится звук столкновения. Если результат отрицательный, то мы проверяем, находится ли шарик ниже платформы. Если да, то это означает, что игрок не сумел поймать шарик платформой, следовательно, нужно отнять одну жизнь и перезапустить шарик с платформы. Листинг данной функции имеет вид:

Листинг 3.7 – Реализация проверки столкновения с платформой

1. void ProcessBall(void)
2. {
3. srand(time(NULL));
4. GLfloat ball\_cx = ball\_x+(bsize/2.0f),
5. ball\_cy = ball\_y+(bsize/2.0f);
6. GLfloat paddle\_cx = paddle\_x+pwidth/2.0f,
7. paddle\_cy = paddle\_y+pheight/2.0f;
8. if (HitDetection(paddle\_cx, paddle\_cy, pwidth, pheight,
9. ball\_cx, ball\_cy, bsize, bsize) != NOHIT)
10. {
11. if (ball\_cy > paddle\_cy)
12. {
13. ball\_dy=-ball\_dy;
14. ball\_y+=ball\_dy;
15. }
16. if (leftKeyPressed || rightKeyPressed)
17. {
18. float addAngle = (-1+rand()%3)\*15.0f;
19. float ball\_v\_sqr = ball\_dx\*ball\_dx+ball\_dy\*ball\_dy;
20. int sign = (ball\_dx > 0) ? 0 : 1;
21. ball\_dx = sqrt(ball\_v\_sqr)\*cos((45+addAngle)\*3.14159f/180.0f);
22. ball\_dx = (sign == 1) ? -ball\_dx : ball\_dx;
23. sign = (ball\_dy > 0) ? 0 : 1;
24. ball\_dy = sqrt(ball\_v\_sqr-ball\_dx\*ball\_dx);
25. ball\_dy = (sign == 1) ? -ball\_dy : ball\_dy;
26. if (leftKeyPressed && ball\_dx > 0) ball\_dx = -ball\_dx;
27. else if (rightKeyPressed && ball\_dx < 0) ball\_dx = -ball\_dx;
28. }
29. sndPlaySound("sounds/hit.wav", SND\_ASYNC);
30. return;
31. }
32. //остальной код функции
33. …
34. }

Далее в функции происходит проверка на столкновение с блоком. Это происходит в два этапа. Вначале идет так называемая «широкая фаза», в ходе которой сужается диапазон блоков, с которыми будет необходимо проверить столкновение. В результате останется максимум четыре блока, которые и будут проверяться. Ниже приведен листинг данной фазы проверки:

Листинг 3.8 – Реализация «широкой фазы» проверки столкновений

1. void ProcessBall(void)
2. {
3. //остальной код функции
4. …
5. int hitTiles[4][2];
6. int hitTilesCount = 4;
7. hitTiles[0][0] = (int)((ball\_x+windowWidth)/(blockWidth+xdist));
8. hitTiles[0][1] = (int)(-(ball\_y+bsize/2.0f-windowHeight)/(blockHeight+ydist));
9. hitTiles[1][0] = (int)((ball\_x+bsize/2.0f+windowWidth)/(blockWidth+xdist));
10. hitTiles[1][1] = (int)(-(ball\_y+bsize/2.0f-windowHeight)/(blockHeight+ydist));
11. hitTiles[2][0] = (int)((ball\_x+bsize/2.0f+windowWidth)/(blockWidth+xdist));
12. hitTiles[2][1] = (int)(-(ball\_y-windowHeight)/(blockHeight+ydist));
13. hitTiles[3][0] = (int)((ball\_x+windowWidth)/(blockWidth+xdist));
14. hitTiles[3][1] = (int)(-(ball\_y-windowHeight)/(blockHeight+ydist));
15. for (int i=0; i<hitTilesCount; i++)
16. {
17. for (int j=i; j<hitTilesCount; j++)
18. {
19. if (hitTiles[i][0] == hitTiles[j][0] && hitTiles[i][1] == hitTiles[j][1])
20. hitTilesCount--;
21. }
22. //остальной код функции
23. … }

После широкой фазы идет «узкая фаза», в ходе которой мы проверяем кирпичи, полученные в «широкой фазе» на столкновение с шариком. Для этого используется функция **HitDetection**. В случае столкновения шарика с кирпичом проверяется, уменьшается счетчик ударов для кирпича, и если он равен нулю, то шарик отскакивает от него и кирпич уничтожается. Если счетчик нулю не равен, то произойдет простое отталкивание шарика. Шарик отталкивается в соответствии с гранью кирпича, о которую он ударился. При разрушении из кирпича может выпасть случайный приз. Листинг данной фазы приведен ниже:

Листинг 3.9 – Реализация «узкой фазы» проверки столкновений

1. void ProcessBall(void)
2. {
3. //остальной код функции
4. …
5. for (int i=0; i<hitTilesCount; i++)
6. {
7. int row = hitTiles[i][1];
8. int col = hitTiles[i][0];
9. GLfloat block\_cx = blocks[row][col].left+blockWidth/2.0f;
10. GLfloat block\_cy = blocks[row][col].bottom+blockHeight/2.0f;
11. GLuint hitLine = HitDetection(block\_cx, block\_cy, blockWidth, blockHeight, ball\_cx, ball\_cy, bsize, bsize);
12. if (blocks[row][col].active && hitLine != NOHIT) {
13. blocks[row][col].hp--;
14. if (blocks[row][col].hp == 0) {
15. blocks[row][col].active = FALSE;
16. blocks\_hit++;
17. sndPlaySound("sounds/point.wav", SND\_ASYNC);
18. }
19. else sndPlaySound("sounds/hit.wav", SND\_ASYNC);
20. if (hitLine == VERTICALHIT) {
21. ball\_dx = -ball\_dx;
22. }
23. else if (hitLine == HORIZONTALHIT) {
24. ball\_dy=-ball\_dy;
25. }
26. if (blocks[row][col].hp == 0) {
27. score += blocks[row][col].score;
28. if (!prize) {
29. prize = rand() % 5;
30. if (prize > 3) prize = 0;
31. prize\_x = blocks[row][col].left + blockWidth/2.0f-prizesize/2.0f;
32. prize\_y = blocks[row][col].bottom + blockHeight/2.0f-prizesize/2.0f;
33. }
34. }
35. return;
36. }
37. }
38. }

Функция **GLuint HitDetection(GLfloat cx1, GLfloat cy1, GLfloat width1, GLfloat height1, GLfloat cx2, GLfloat cy2, GLfloat width2, GLfloat height2)** производит проверку двух объектов на столкновение. В качестве параметров она принимает координаты центров объектов и их размеры. Функция может возвращать три значения:

* + NOHIT, если столкновения не обнаружено
  + HORIZONTALHIT, если столкновение произошло о верхнюю или нижнюю грань
  + VERTICALHIT, если столкновение произошло о правую или левую грань

Для определения столкновения, строятся проекции объектов на оси координат. Если проекции перекрываются по обеим осям, то столкновение произошло, причем характерстолкновения определяется в зависимости от величины перекрытия по каждой из осей: там где проекции перекрываются больше, перпендикулярно той оси и будет произведено отталкивание. Листинг данной функции имеет вид:

Листинг 3.10 – Реализация функции GLuint HitDetection()

1. GLuint HitDetection(GLfloat cx1, GLfloat cy1, GLfloat width1, GLfloat height1,
2. GLfloat cx2, GLfloat cy2, GLfloat width2, GLfloat height2)
3. {
4. GLfloat dx12 = abs(cx1-cx2),
5. dy12 = abs(cy1-cy2);
6. GLfloat xoverlay = (width1+width2)/2.0f-dx12,
7. yoverlay = (height1+height2)/2.0f-dy12;
8. int hitLine = 0;
9. if (xoverlay > 0 && yoverlay > 0)
10. {
11. if (xoverlay > yoverlay)
12. {
13. hitLine = HORIZONTALHIT;
14. }
15. else hitLine = VERTICALHIT;
16. }
17. else hitLine = NOHIT;
18. return hitLine;
19. }

В функции **void InitBlocks()** происходит инициализация массива, в котором хранятся кирпичи, начальными значениями. Листинг данной функции имеет вид:

Листинг 3.11 – Реализация функции void InitBlocks()

1. void InitBlocks()
2. {
3. srand(time(NULL));
4. GLfloat x1 = -windowWidth;
5. GLfloat y1 = windowHeight;
6. for (int row=0; row < NUM\_BLOCK\_ROWS; row++)
7. {
8. for (int col=0; col < NUM\_BLOCK\_COLUMNS; col++)
9. {
10. blocks[row][col].left = x1;
11. blocks[row][col].top = y1;
12. blocks[row][col].right = x1 + blockWidth;
13. blocks[row][col].bottom = y1 - blockHeight;
14. blocks[row][col].active = TRUE;
15. blocks[row][col].hp = rand()%2+1;
16. if (blocks[row][col].hp == 1)
17. {
18. blocks[row][col].tex = bricktex;
19. blocks[row][col].score = 25;
20. }
21. else
22. {
23. blocks[row][col].tex = metaltex;
24. blocks[row][col].score = 50;
25. }
26. x1 += blockWidth + xdist;
27. }
28. x1 = -windowWidth;
29. y1 -= blockHeight + ydist;
30. }
31. }

Функция **void ProcessPrize()** отвечает за движение призов и определение их столкновения с платформой. В случае столкновения с платформой, приз активируется, а также воспроизводится звук, сигнализирующий о том, что приз пойман. В случае пропускания приза ниже платформе, приз уничтожается. Листинг данной функции приведен ниже:

Листинг 3.12 – Реализация функции void ProcessPrize()

1. void ProcessPrize()
2. {
3. GLfloat prize\_cx = prize\_x+prizesize/2.0f;
4. GLfloat prize\_cy = prize\_y+prizesize/2.0f;
5. GLfloat paddle\_cx = paddle\_x+pwidth/2.0f;
6. GLfloat paddle\_cy = paddle\_y+pheight/2.0f;
7. if (prize\_y+prizesize < paddle\_y)
8. {
9. prize = 0;
10. return;
11. }
12. if (HitDetection(prize\_cx, prize\_cy, prizesize, prizesize, paddle\_cx, paddle\_cy, pwidth, pheight) != NOHIT)
13. {
14. switch (prize)
15. {
16. case 1: score += 100; break;
17. case 2: lives++; break;
18. case 3: pwidth += begpwidth; pbonuscount += 401; break;
19. }
20. sndPlaySound("sounds/point.wav", SND\_ASYNC);
21. prize = 0;
22. }
23. prize\_y -= 3.0;
24. }

В модуле **graphics.cpp** реализован вывод графики на экран. Он содержит следующие функции:

* **void Draw();**
* **void DrawBlocks();**
* **void DrawPrize();**
* **void Draw\_Text(char\* str, int x, int y, GLfloat\* color);**
* **void HighscrDraw();**
* **GLuint LoadTexture(char \*filename);**
* **void ChangeSize(GLsizei w, GLsizei h);**

Функция **void Draw()** отвечает за вывод всех изображений на экран, таких как изображение заднего фона, шарика, платформы, кирпичейб призов. Также она выводит весь необходимый для текущего состояния игры текст. При необходимости, на экране будет нарисована таблица рекордов. Для примера приведен код отрисовки шарика. Отрисовка остальных игровых объектов производится аналогично.

Листинг 3.13 – Вывод шарика на экран

1. void Draw()
2. {
3. //отрисовка остальных элементов
4. …
5. glBindTexture(GL\_TEXTURE\_2D, balltex);
6. glBegin(GL\_QUADS);
7. glColor3f(1.0f, 1.0f, 1.0f);
8. glVertex2f(ball\_x, ball\_y+bsize);
9. glTexCoord2f(1, 0);
10. glVertex2f(ball\_x+bsize, ball\_y+bsize);
11. glTexCoord2f(1, 1);
12. glVertex2f(ball\_x+bsize, ball\_y);
13. glTexCoord2f(0, 1);
14. glVertex2f(ball\_x, ball\_y);
15. glTexCoord2f(0, 0);
16. glEnd();
17. //отрисовка остальных элементов
18. …
19. }

В функцию **void DrawBlocks()** для удобствавынесен код отрисовки кирпичей. Листинг данной функции имеет вид:

Листинг 3.14 – Вывод кирпичей на экран

1. void DrawBlocks()
2. {
3. static int count = 0;
4. GLfloat x1 = -windowWidth,
5. y1 = windowHeight;
6. for (int i=0; i<NUM\_BLOCK\_ROWS; i++)
7. {
8. for (int j=0; j<NUM\_BLOCK\_COLUMNS; j++)
9. {
10. if (blocks[i][j].hp == 1)
11. glColor3f(0.0f, 1.0f, (level%9)\*0.1f);
12. else
13. glColor3f(1.0f, 0.0f, (level%9)\*0.1f);
14. if (blocks[i][j].active)
15. {
16. glBindTexture(GL\_TEXTURE\_2D, blocks[i][j].tex);
17. glBegin(GL\_QUADS);
18. glColor3f(1.0f, 1.0f, 1.0f);
19. glVertex2f(x1, y1);
20. glTexCoord2f(1, 0);
21. glVertex2f(x1+blockWidth, y1);
22. glTexCoord2f(1, 1);
23. glVertex2f(x1+blockWidth, y1-blockHeight);
24. glTexCoord2f(0, 1);
25. glVertex2f(x1, y1-blockHeight);
26. glTexCoord2f(0, 0);
27. glEnd();
28. }
29. x1 += blockWidth + xdist;
30. }
31. x1 = -windowWidth;
32. y1 -= blockHeight + ydist; }
33. }

В функции **void DrawPrize()**  реализована отрисовка призов. В зависимости от типа приза на него накладывается соответствующая текстура.

Функция **void Draw\_Text(char \*str, int x, int y, GLfloat\* color)** выводит на экран текст. В качестве параметров она принимает строку, которую необходимо вывести, координаты строки и ее цвет. Листинг данной функции имеет вид:

Листинг 3.15 – Реализация функции void Draw\_Text(char\*, int, int, GLfloat)

1. void Draw\_Text(char \*str, int x, int y, GLfloat\* color)
2. {
3. int len = strlen(str);
4. glPushMatrix();
5. glTranslatef(x, y, 0);
6. glScalef(0.05, 0.05, 0.05);
7. glColor3fv(color);
8. for (int i=0; i<len; i++)
9. {
10. glutStrokeCharacter(GLUT\_STROKE\_ROMAN, str[i]);
11. }
12. glPopMatrix();
13. }

Функция **void HighscrDraw()** выполняет отрисовку таблицы рекордов на 10 записей. Для этого по центру экрана рисуется прямоугольник, поверх которого выводится 10 строк текста, хранящихся в массиве. Если записей меньше 10, то выводится пустая запись с номером.

Функция **GLuint LoadTexture(char \*filename)** производит загрузку изображения из файла в память и создание текстуры OpenGL. В качестве параметра функция принимает имя файла с картинкой. Функция возвращает идентификатор созданной текстуры Для загрузки изображения используется функция **SOIL\_load\_image** из библиотеки SOIL. Для выгрузки изображения из памяти используется функция **SOIL\_free\_image\_data**.

3.4 Тестирование Программы

При запуске программы пользователю показывается основное окно программы.

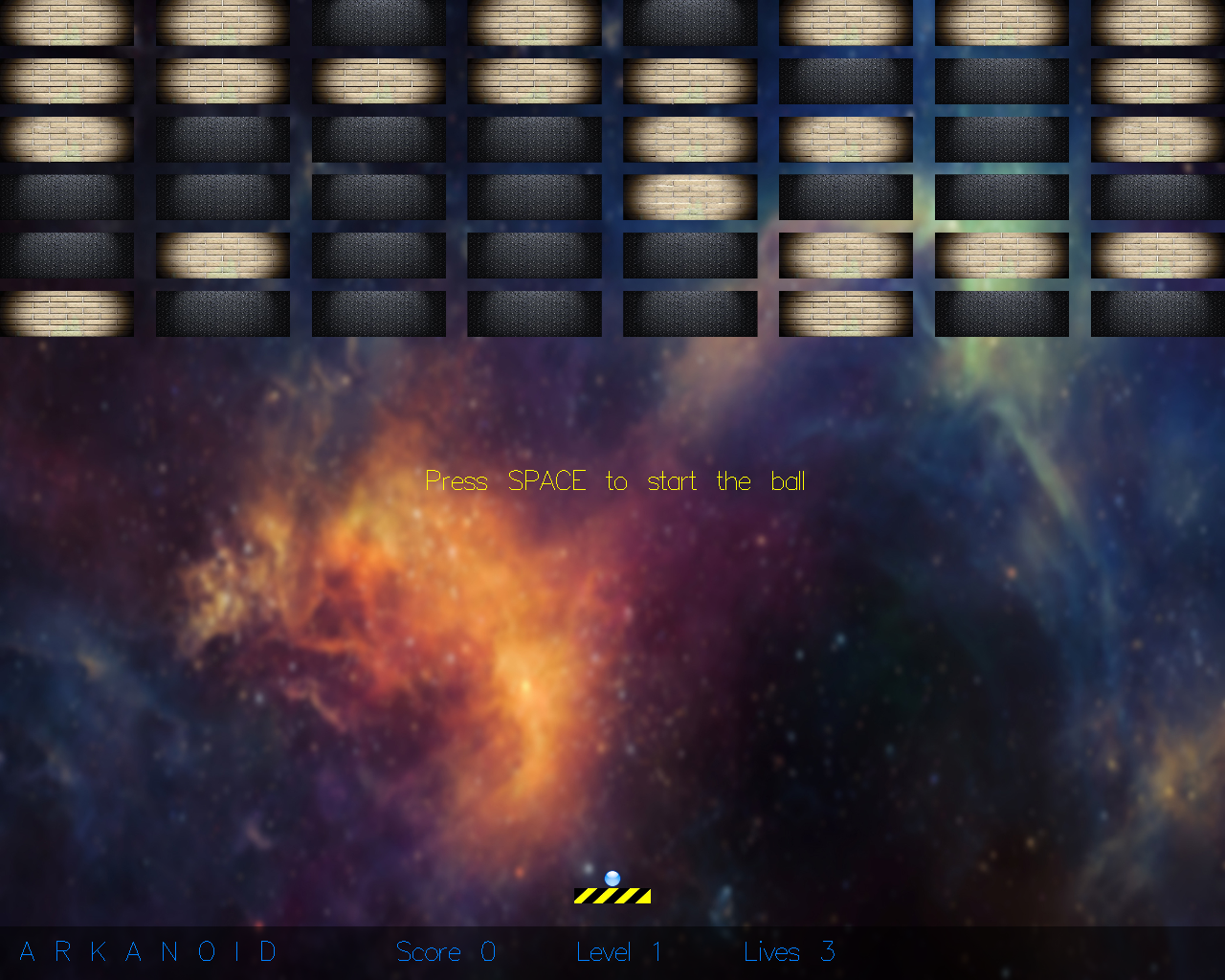


Рисунок 4.1 – Стартовое окно программы

При нажатии на клавишу пробел произойдет запуск шарика. При столкновении шарика с блоком, блок уничтожается, а шарик отскакивает в нужном направлении. Начисляется 25 очков.



Рисунок 4.2 – Запуск шарика и удар по блоку

При потере шарика количество жизней уменьшается на 1, а шарик восстанавливается на платформе.

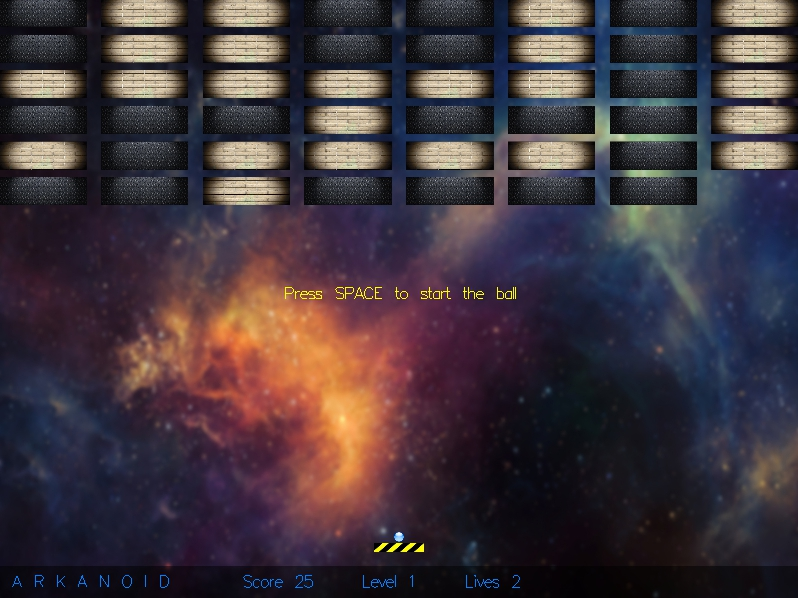


Рисунок 4.3 – Потеря шарика

При уничтожении блока есть шанс на выпадение приза.

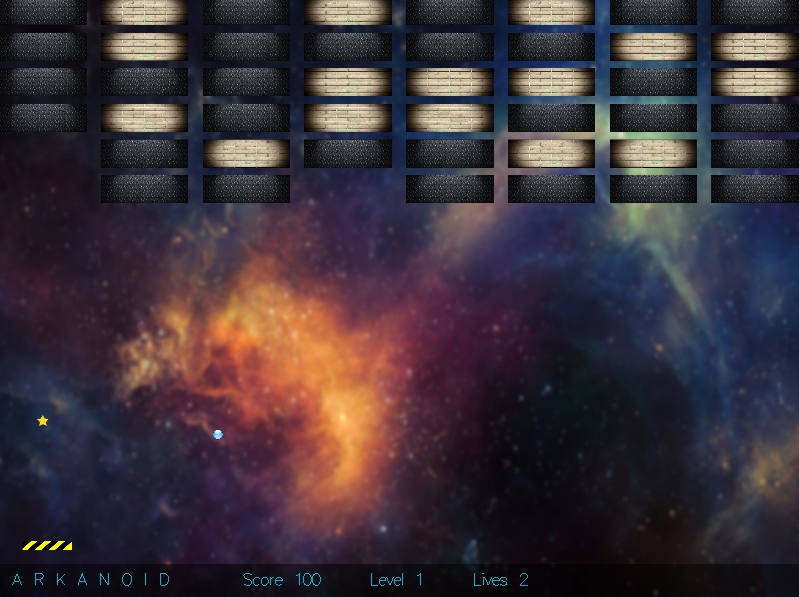


Рисунок 4.4 – Выпадение приза

Если поймать этот приз платформой, то он активируется и увеличит платформу.



Рисунок 4.5 – Активация приза

При нажатии на клавишу TAB вызывается таблица рекордов.



Рисунок 4.6 – Таблица рекордов при нажатии TAB

При исчерпании количества жизней игра заканчивается и на экран выводится таблица рекордов. Она отличается от таблицы, вызываемой клавишей TAB, наличием крестика в правом верхнем углу, при нажатии на который таблица закрывается и запускается новая игра.



Рисунок 4.7 – Окончание игры

При наведении на крестик он меняет цвет.



Рисунок 4.8 – Изменение цвета крестика

ЗАКЛЮЧЕНИЕ

В курсовом проекте была поставлена задача разработки игрового работоспособного ПО с использованием языка C++ без объектно-ориентированных возможностей. Программа должна представлять собой игру «Арканоид». Программа должна обладать интуитивно понятным интерфейсом.

Удалось реализовать программу, которая способна:

1. Обеспечивать движение и столкновение игровых объектов
2. Использовать звуки и изображения из файлов
3. Начислять очки, используя систему подсчета очков
4. Использовать систему выпадающих призов
5. Информировать об окончании игры
6. Отображать таблицу рекордов на 10 записей

Интерфейс программы интуитивно понятен любому пользователю, который будет использовать программу.

Был произведен разбор всех функций приложения и их тестирование с разбором основных возможностей.

Программа оснащена звуковыми эффектами и умеет загружать картинки игровых объектов из файлов.

В итоге реализовано приложение с интуитивно понятным интерфейсом, обладающее всеми необходимыми возможностями. Были реализованы все функции, задуманные вначале.

При выполнении курсового проекта были получены практические навыки проектирования графического интерфейса, разработки и реализации алгоритма работы программы, ее тестирования и разработки документации. Были изучены возможности языка программирования C++, программного интерфейса OpenGL и библиотеки GLUT(OpenGL Utility Toolkit).

Для использования OpenGL и GLUT произведена настройка Visual Studio и подключены все необходимые библиотеки и заголовочные файлы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ламот А. Программирование игр для Windows. Советы профессионала, 2-е изд. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2003. — 880 с. : ил. — Парал. тит. Англ
2. Лафоре Р. Объектно-ориентированное программирование в C++-СПБ.: Издательство «Питер», 2004. – 928 с.
3. Райт Р. Липчак Б. OpenGL. Суперкнига, 3-е издание.: Пер. с англ – М.: Издательский дом «Вильямс», 2006. – 1040 с.: ил. – Парал. тит. англ.
4. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3 //[Электронный ресурс]. 1996 - https://www.opengl.org/documentation/specs/glut

/spec3/spec3.html – Дата доступа: 30.10.2015

1. Урок: базовые алгоритмы определения столкновений //[Электронный ресурс]. 2004 - http://noregret.org/tutor/n/collision/ - Дата доступа: 05.12.2015

# ПРИЛОЖЕНИЕ А. МОДУЛЬНАЯ СХЕМА ПРОГРАММЫ

ARKANOID

OBJECTS

GRAPHICS

Рисунок А.1 – Модульная схема программы

**ПРИЛОЖЕНИЕ Б. ФУНКЦИОНАЛЬНАЯ СХЕМА ПРОГРАММЫ**

WinMain

glutMainLoop

Init

ChangeSize

MouseMove

LoadTexture

MouseClick

KeyboardUpFunc

SpecialReleaseFunc

KeyboardFunc

SpecialPressFunc

ProcessBall

HitDetection

MoveBall

TimerFunc

ProcessPrize

EndGame

InitBlocks

Draw

DrawBlocks

LoadFile

DrawPrize

SaveFile

HighscrDraw

Draw\_Text

Рисунок Б.1 – Функциональная схема программы

**ПРИЛОЖЕНИЕ В. ПРОТОТИП ГРАФИЧЕСКОГО ИНТЕРФЕЙСА**

Рисунок В.1 – Прототип графического интерфейса