Компьютерная игра «Снайпер» на языке программирования C++

Евгений Никитенко[*]
7 класс, лицей «Вторая школа», Москва

Научный руководитель: И. Р. Дединский, лицей «Вторая школа», Москва

Целью работы является создание графической компьютерной игры «Снайпер» на языке программирования C++ с помощью компилятора GCC в среде OS Windows. Главная задача игры – сбить все мишени за минимальное количество выстрелов. Для этого дается 10 обойм по 6 пуль.

Пользователю также можно создавать свои уровни с расширением .lvl. При разработке преследовалась цель максимально упростить их создание. Вся информация уровня записывается в отдельный файл. Таким образом, можно создавать уровни, не изменяя ни одной строчки программы.

Для отображения интерфейса используется графика в формате BMP, что придает игре красочность.

Программа передает уровни, созданные пользователем, на FTP-сервер, чтобы автор мог добавлять их в игру. Для того чтобы минимизировать задержки при передаче, создается второй поток, который запускает программу обмена данных с сервером. Таким образом, это не влияет на скорость работы программы.

В игре предусмотрено сохранение процесса игры. Сохранять можно несколько игр. Существует 6 «ячеек» для сохранения. В директории с игрой есть папка save, а в ней 6 файлов: slot1.save, slot2.save,…, slot6.save. В каждом хранится номер уровня. Если номер уровня нулевой, то это новая игра.

В исходном коде документируются все функции с помощью системы автоматического документирования Doxygen.

1. Об игре

1.1. Цель работы

Целью работы является создание графической компьютерной игры «Снайпер» на языке программирования *C++* с помощью компилятора *GCC* в среде *OS Windows*.

Главная задача игры – сбить все мишени за минимальное количество выстрелов. Для этого дается 10 обойм по 6 пуль.

Пользователю также можно создавать свои уровни с расширением *.lvl*. При разработке преследовалась цель максимально упростить их создание. Для отображения интерфейса используется графика в формате *BMP*, что придает игре красочность.

Программа передает уровни, созданные пользователем, на *FTP*-сервер, чтобы автор мог добавлять их в игру. Для того чтобы минимизировать задержки при передаче, создается второй поток, который запускает программу обмена данных с сервером. Таким образом, это не влияет на скорость работы игры.

В исходном коде документируются все функции с помощью системы автоматического документирования *Doxygen*.

[*] E-mail для связи: zenya1996@yandex.ru.

2. Структура и алгоритмы программы

2.1. Главный цикл

Алгоритм главного цикла игры приведен на Листинге 1.

```
01. void mainLoop (Level_t* level)
02. {
03.     while (true)
04.     {
05.          checkButtons      (level);
06.          checkCheats       (&level->cheats);
07.          runCheats         (level);
08.          newLevel          (level);
09.          levelImage        (level);
10.          drawTargets       (level);
11.          drawBullets       (level);
12.          sight             (mouseX(), mouseY(), 699, TX_BLACK);
13.          drawInformation   (level);
14.          checkFire         (level);
15.          checkRechange     (level);
16.          checkMouseButtons (level);
17.          drawVersion       ();
18.
19.          txSleep           (10);
20.     }
21.}
```

Листинг 1. Главный цикл программы.

Рассмотрим алгоритм главного цикла:

Сначала проверяем нажатие кнопок, таких как F1 (вызов помощи) и escape (возврат в главное меню).

Далее проверяем, введен ли «читерский» код (инженерный пароль). Если да, то выполняем нужную операцию.

Потом проверяем, пройден ли уровень (см. 2.2).

Зарисовываем все необходимое.

Проверяем функции кнопок мыши (нажата ли левая кнопка, нажата ли правая кнопка, отпущены ли кнопки).

В завершение выводим версию игры на экран.

После этого нужно некоторое время подождать (задержка).

2.1.1. Проверка, сбита ли мишень

Алгоритм проверки, сбита ли мишень, приведен на рис. 1.

Если произведен выстрел, мы проверяем, совпадает ли цвет точки выстрела с цветом мишени.

В противном случае опять делаем проверку, произведен выстрел или нет.

Если цвета совпадают, то проверяем циклом с помощью координат, какая сбита мишень.

Опять же в противном случае переходим к проверке, произведен ли выстрел.

Дальше элементу логического массива, который соответствует номеру мишени, присваиваем *true*.

В функции *newLevel* проверяется, все ли элементы массива равны *true*. Если это так, то уровень пройден.

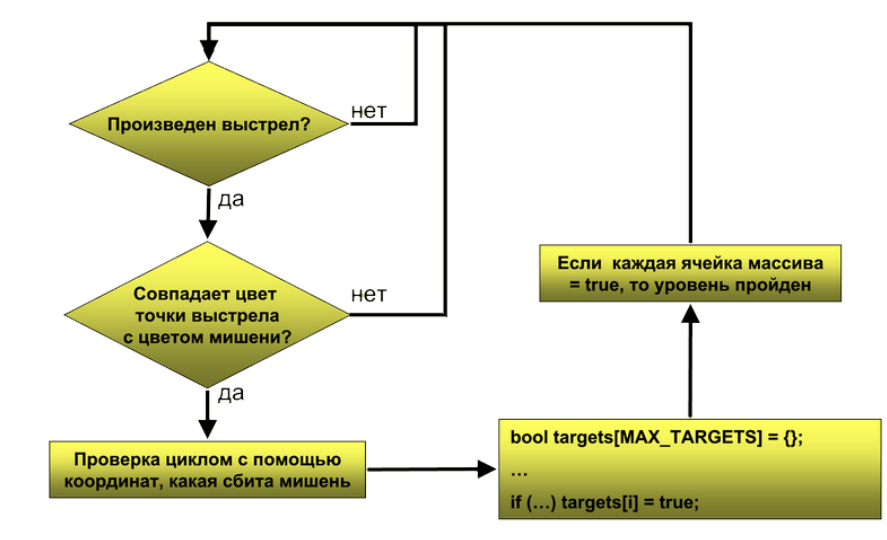

Рис. 1. Проверка, сбита ли мишень.

2.2. Загрузка нового уровня

Алгоритм загрузки нового уровня приведен на листинге 2.

```
01. void newLevel (Level_t* level)
02. {
03.      if (!levelComplite (level) && !level->cheats.useCheat[0]) return;
04.      level->numOfPoints += PLUS_POINTS_NEW_LEVEL;
05.
06.      if (level->image.levelImage != 0)  sleep (level, 1000);
07.
08.      saveLevel             (level);
09.      ifYourLevelClose      (level);
10.      shop                  (level);
11.
12.      levelUp               (level);
13.      deleteLevelImage      (level);
14.      deleteTargetsImages   (level);
15.      zeroData              (level);
16.
17.      loadLevel             (level);
18.
19.      loadBoomImages        (level);
20.      prepareImageAddresses (level);
21.      loadLevelImage        (level);
22.      loadTargetsImages     (level);
23.
24.      drawLoad              ("Готово!", 100, RGB (255, 255, 80));
25.
26.      ifYourLevelFtp        (level);
27.      ifErrorsDrawIt        (level);
28.      drawHelp              (level->helpText, strlen (level->helpText));
29.
30.      level->cheats.useCheat[0] = false;
31. }
```

Листинг 2. Алгоритм создания нового уровня.

Если уровень не пройден и не использован «читерский» код (инженерный пароль), то идет возврат в главный цикл.

Сохраняем процесс игры (см. 2.3. Сохранение игры).

Если пройден уровень, созданный пользователем, то идет возврат в главное меню.

Магазин. В будущем за очки пользователю можно будет улучшать оружие и покупать новое. Это как раз будет происходить в магазине (сейчас магазин есть, но делать там ничего нельзя).

Происходит увеличение номера уровня.

Происходит удаление изображения уровня.

Происходит удаление изображений мишеней уровня (типов мишеней может быть много, и на каждый тип приходится свое изображение).

Происходит обнуление данных.

Происходит считывание информации нового уровня из файла (см. 2.2.1).

Происходит загрузка изображений.

На экран выводится информация об удачной загрузке уровня.

Если запущен уровень, созданный пользователем, то передаем его на ftp-сервер (см. 2.2).

Если есть ошибки, то выводим их на экран.

Выводим на экран помощь уровня.

Присваиваем ячейке массива, которая показывает, произведен ли «читерский» код, позволяющий перейти на следующий уровень, false, чтобы на следующем обороте цикла уровень не загружался заново.

Во время загрузки нового уровня то, что происходит, выводится на экран, поэтому отображаются даже простые действия (рис. 2).

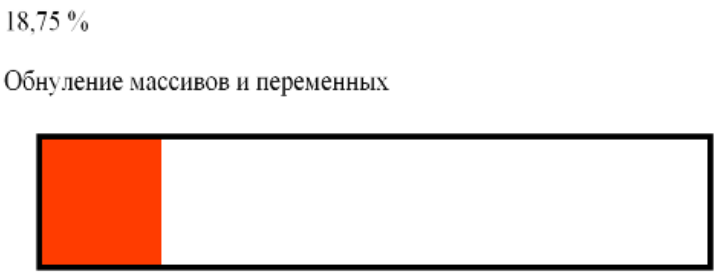

Рис 2. Прогресс-бар загрузки уровня.

2.2.1. Считывание информации нового уровня из файла

Алгоритм считывания информации уровня из файла приведен на рис. 3.

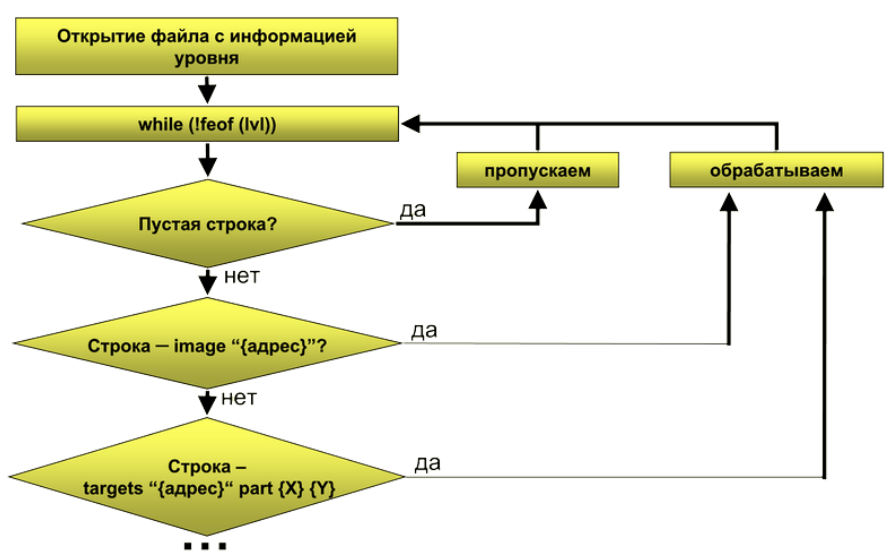

Рис 3. Алгоритм загрузки нового уровня.

1. Открываем файл с информацией уровня.
2. Запускаем цикл, который будет работать, пока файл не кончится.
3. Если строчка состоит полностью из пробельных символов, то пропускаем ее.
4. Если строчка выглядит так*: image «{адрес}»,* то записываем в переменную адрес изображения уровня.
5. Если строчка такая: *targets «{адрес}» part {x} {y},* то записываем нужную информацию.
6. (Далее повторяем аналогичные действия для других типов строк.)

n. Закрываем файл с информацией уровня.

2.2.2. Передача уровня на *ftp*-сервер через Интернет

Передача данных на *ftp*-сервер реализована следующим образом (рис. 4):

Рис. 4. Передача уровня на *ftp*-сервер.

a. Проверяем, отдельный это уровень или уровень, созданный автором.
b. Если это уровень, созданный автором, то выходим из функции.
c. В противном случае создаем второй поток, в котором передаем данные на сервер. Второй поток создается для того, чтобы пользователь не ждал, пока закончится передача данных на сервер.
d. Запись информации уровня в командный файл для *ftp*-клиента.
В файле записываются:

1. «Адрес *ftp*-сервера».
2. «*login*».
3. «пароль». В программе пароль шифруется, чтобы избежать его считывания в явном виде из исполняемого файла игры.
4. «*binary*». Это слово говорит о том, что передача данных должна идти бит в бит.
5. *«cd "\levels"».* «*cd*» (*change directory*) изменяет директорию.
6. «*put*» файл с информацией уровня». Команда *put* передает файл на сервер
7. «*put*» изображение уровня».
8. цикл, который будет работать, пока не кончатся изображения мишеней уровня: «*put*» изображение мишени уровня».
9. «*close*» (отключаем передачу данных).
10. «*quit*» (выходим из *ftp*-клиента).

e. Программа запускает *ftp*-клиент и указывает ему этот файл в командной строке.
f. Программа удаляет командный файл для *ftp*-клиента.

2.3. Сохранение игры

В игре существует возможность сохраняться. В директории с игрой есть папка *save*, а в ней 6 файлов: *slot1.save, slot2.save,…, slot6.save.* В каждом хранится номер уровня. Если номер уровня нулевой, то это новая игра.

В игре существует 2 запускаемых файла: *run.exe* (это меню, его и надо запускать) и *game.exe* (это сама игра). В меню пользователь выбирает ячейку для сохранения. *run.exe* записывает в файл номер уровня и номер ячейки. Файл *game.exe* считывает номер ячейки и номер уровня и сохраняет их.

Далее в начале функции *newLevel* сохраняется процесс игры. Из переменной берется номер ячейки, открывается файл, и туда записывается номер уровня.

Меню, где надо выбрать ячейку для сохранения, см. на рис. 5.

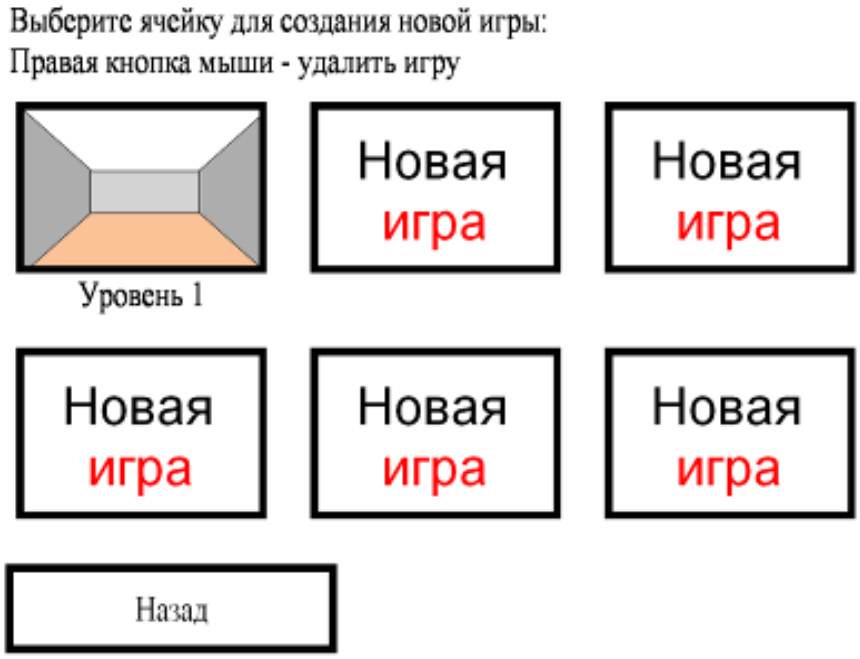

Рис. 5. Меню создания новой игры.

3. Структуры

3.1. Структура горячих точек в игре (***Boom_t***)

int numBoom	*Количество горячих точек в уровне*
HDC boomImage *[*MAX_BOOM*]*	*Изображения горячих точек в игре*
char boomImageAddress *[*MAX_BOOM*][100]*	*Адреса изображений горячих точек*
char boomSoundAddress *[*MAX_BOOM*][100]*	*Адреса звуков горячих точек в игре*
COLORREF boomColor *[*MAX_BOOM*]*	*Цвета горячих точек в игре*
POINT boomImageSize *[*MAX_BOOM*]*	*Размер изображений горячих точек в игре*

3.2. Структура, где хранятся изображения и адреса (***Image_t***)

char levelImageAddress *[*MAX_STR*]*	*Адрес изображения уровня*

char targetsImageAddress *[*MAX_TYPES_OF_TARGETS*][* MAX_STR*]*	*Адрес изображений мишеней уровня*
HDC levelImage	*Изображение уровня*
HDC targetsImage *[*MAX_TYPES_OF_TARGETS*]*	*Изображения мишеней уровня*
HDC information	*Изображение обоймы*
HDC bullet	*Изображение пули (в игре будет нарисована обойма с пулями)*

3.3. Структура с информацией мишеней (*Targets_t*)

POINT targetsPosition *[*MAX_TARGETS*]*	*Массив с координатами мишеней*
bool levelTargets *[*MAX_TARGETS*]*	*Логический массив, показывающий состояние мишеней (если* true*, то мишень сбита, а если* false*, то нет)*
int targetsType *[*MAX_TARGETS*]*	*Массив с типами мишеней*
int numOfTargets	*Количество мишеней в уровне*
POINT partTargets *[*MAX_TYPES_OF_TARGETS*]*	*Размер частей изображений мишеней уровня (*X*/2,* Y*)*
COLORREF colorTarget *[*MAX_TYPES_OF_TARGETS*]*	*Цвет точки попадания в мишень*

3.4. Структура с информацией о кодах игры (*Cheats_t*)

bool useCheat *[*MAX_OF_CHEATS*]*	*Если ячейка логического массива* = true*, то использован читерский код*
char cheatCode *[*MAX_OF_CHEATS*][1000]*	*Массив, который хранит все коды игры*

3.5. Главная структура уровня (*Level_t*)

Boom_t boom	*Структура* Boom_t
Image_t image	*Структура* Image_t
Targets_t targets	*Структура* Targets_t
Cheats_t cheats	*Структура* Cheats_t
COLORREF colorNotBullets *[*MAX_NOT_BULLETS*]*	*Цвет точек, где пуля не должна появляться после выстрела (например, небо)*
int numNotBullets	*Количество областей, в которых, если стрельнуть, следа от пули не будет видно*
Int levelNum	*Переменная, которая хранит в себе номер уровня*
POINT shootPosition*[*MAX_SHOOTS*]*	*Массив для хранения координат пуль*
bool mouseButtonNull *[2]*	*Если* mouseButtonNull *[0]* = true*, то левая кнопка отжата, если* mouseButtonNull *[1]* = true*, то правая кнопка отжата*
int shoot	*Номер точек выстрела в данный момент (если выстрел попадает в небо, то номер*

	точки не должен прибавляться)
int numOfBullet	*Номер выстрела, который будет выводиться на экран*
Char helpText *[*MAX_HELP_TEXT*]*	*Текст помощи*
bool yourLvl	*Если* yourLvl = true*, то игра открывается с помощью уровня пользователя, а если* false*, то с помощью уровня автора*
char addressFile*[1000]*	*Адрес отдельного* lvl *файла с информацией уровня*
bool rechange	*Если* rechange = true*, то произошла перезарядка, а если* = false*, то нет*
int numOfPoints	*Количество очков у игрока*
int slot	*Номер ячейки, откуда загружали игру (для сохранения процесса игры)*

4. Итоги

4.1. Результаты работы

Разработана графическая компьютерная игра. Сделано удобное создание уровней. Создана передача уровня на сервер с целью добавления новых уровней в игру. Также сделано сохранение процесса игры.

Пример работы игры см. на рис. 6.

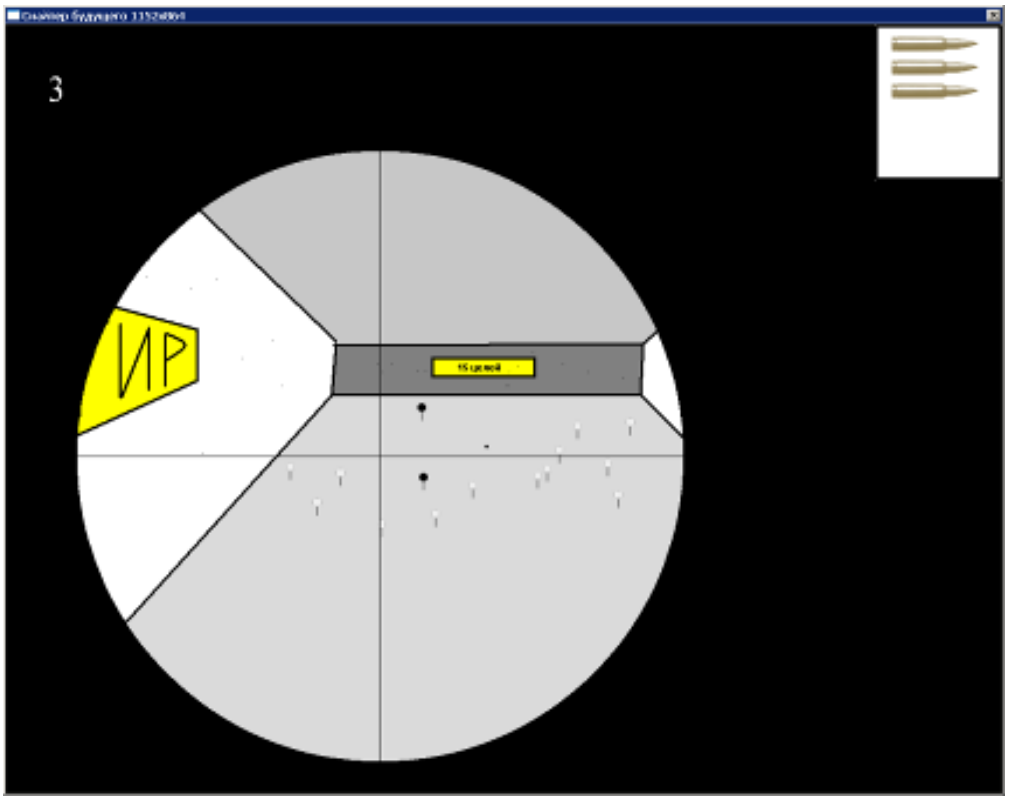

Рис 6. Пример работы игры.

4.2. Планируется сделать

1. Автоматическое обновление игры.
2. Магазин, где можно будет покупать и улучшать оружие.
3. Игра на время.
4. Аркадный режим.
5. Редактор уровней.