###### Оглавление

[Введение 3](#_Toc383688042)

[Техническое задание 4](#_Toc383688043)

[Анализ технического задания 6](#_Toc383688044)

[Разработка проекта 10](#_Toc383688045)

[Руководство пользователя 14](#_Toc383688046)

[Заключение 15](#_Toc383688047)

[Список литературы 16](#_Toc383688048)

[Приложение 17](#_Toc383688049)

# Введение

Игра «Нападение и Оборона» выполнена в жанре **Tower Defense**.

**Tower Defense** (англ. *Башенная защита*), сокращенно **TD** — название жанра компьютерных стратегических игр. Задача игрока в играх подобного жанра — расправиться с наступающими врагами, называемыми в некоторых играх «крипы» (от англ. *creep*, ползучая тварь), до того, как они пересекут карту, с помощью строительства башен, атакующих их, когда те проходят вблизи. Противники и башни обычно различаются по характеристикам и цене. Когда враги побеждены, игрок зарабатывает деньги или очки, которые используются для покупки или модернизации башен.

Подбор башен и их расположение — неотъемлемая стратегия игры. Обычно *ползучие твари* пробегают через подобие лабиринта, что дает игроку возможность стратегического размещения башен, но также существуют известные версии игры, названные линейными TD, где используются прямые пути вместо лабиринтов. В некоторых версиях игрок может сам выстраивать лабиринт из башен и блоков.

В свою очередь Tower Defense делится на несколько видов. Например, в большинстве версий, когда башня модернизируется, её радиус действия, уровень и мощность увеличиваются одновременно. Однако в версии игры под названием *Onslaught Defence*, каждый параметр может быть улучшен отдельно. В некоторых играх враг может обороняться. В отдельных версиях маршрут продвижения врага не ограничен какими-либо стенками. Волны наступления могут начинаться по команде игрока (тем самым давая возможность спокойно подготовиться), либо с определённым промежутком времени. Могут присутствовать элементы экономической стратегии (например, можно построить банк, который будет в геометрической прогрессии увеличивать денежные средства). Противники в разных играх могут различаться по своим свойствам — например, некоторые из них могут быть летающими и для их поражения требуются специальные «противовоздушные» башни.

В игре GemCraft разделили башни (и ловушки) от их активных способностей по атаке (которые обеспечиваются драгоценными камнями). Камни могут выниматься из башен и объединяться, усиливаясь при этом. Дополнительно, появились характеристики игрока, развивающиеся от карты к карте. Уже появились другие игры, использующие данную находку.

Также существует тип игр *Tower Invasion*, где человек управляет «*крипами»*, которые должны миновать вражеские башни.

# Техническое задание

1. **Техническое задание**
   1. **Цель разработки**

* Создание дружелюбного и понятного пользовательского интерфейса
* Разнообразить режимы и сложности самой игры
* Создание новых типов компьютерных противников
* Разнообразие текстур и башен в игре
  1. **Назначение и область применения**

Игровая сфера.

* 1. **Требования:**
     1. **Функциональные**
* Снабдить игру игровым меню
* Создание дополнительных уровней игры
* Добавление новых противников и защитных башен
* Добавление уровней сложности
  + 1. **Количественные требования**

Количество игроков в этой игре:

1 игрок

* + 1. **Требования надёжности**

Защитить программу от утечки памяти.

Организовать защиту от «дурака»:

* Нельзя удалять заданные препятствия
* Нельзя ставить башню и блок на одной и той же клетке
* Нельзя ставить блоки на точку входа и целевую точку
* Нельзя удалять исходные игровые поля
  + 1. **Требования совместимости**

Программа должна быть совместима с Win XP, Vista, 7, 8 32 и 64 bit, Linux.

Работать в оконном режиме. Реагировать на манипулятор мыши.

* + 1. **Требования к интерфейсу**

Должен быть дружественный и интуитивно понятный интерфейс:

* Интерфейс программы должен включать в себя стандартные элементы Windows (кнопка закрытия, элементы меню).
* Окно игры не должно выходить за пределы экрана с минимальным разрешением 1024х768.
* Окно игры должно свободно перетаскиваться по экрану с помощью мышки.
* Цвета не должны быть предельно контрастными.
* Фон поля должен быть нейтрального цвета, светлее знаков, выставленных на нём.
* Клетка поля должна вмещать в себя стандартную курсор-стрелку полностью.

# Анализ технического задания

1. **Выбор среды разработки**

При выборе среды разработки и языка реализации, отталкиваемся от языка которого мы знаем, и среды разработки, в которой мы работали.

* Игра будет реализована на Qt Creator.
* Игра будет реализована на языке С++

1. **Выбор игрового меню**

«Меню́ (англ. menu, фр. menu) — элемент интерфейса пользователя, позволяющий выбрать одну из нескольких перечисленных опций программы. В современных операционных системах меню является важнейшим элементом графического интерфейса пользователя»,- Википедия.

В современных играх игровое меню играет не последнюю роль. Почти каждая игра может похвастаться своим уникальным меню, тем не менее, устройство меню везде практически одинаковое. В основу идут 3 основных пункта: начало игры, настройки и выход из программы. Так же возможны пункты сохранения и кнопка дополнительной информации об игре и разработчиках. Так, как сохранение в игре нет, как и надобности делать информацию об игре, было решено оставить только первые 3 пункта

Далее мы делаем дополнительные меню для первых двух пунктов основного меню (начала игры и настроек). В подменю Play игроку предлагается сделать выбор карты и соответствующий уровень сложности. В подменю Options находится регулятор громкости. Кнопка Exit завершает работу виджета.

1. **Анализ игровых уровней**

Игры данного жанра содержат множество уровней для игры. Было решено сделать три основных уровня:

* Пустыня (Desert of Death)
* Лес (Hidden Forest)
* Арктика (Silent Arctic)

Каждый уровень будет иметь свою «изюминку» и отличаться различной сложностью прохождения. Такое разнообразие сделает игру более интересной.

1. **Анализ игрового окружения**

Для создания атмосферы в игре и оживления игрового процесса в игру необходимо добавить следующие элементы:

* Обновленные башни с различными характеристиками
* Обновленные крипы с различными характеристиками

1. **Анализ алгоритма поиска кратчайшего пути**

Одним из самых простых и надежных, но в ту же очередь одним из самых долгих алгоритмов является алгоритм **Автоматного подхода**

* компьютер вычисляет все возможные варианты прохода к целевой точке, а затем выбирает кратчайший путь
* вычисление в процессе игры и в ходе изменения игрового поля такие просчеты могут занять огромное количество времени

Постоянные вычисления перенагружают игру и могу послужить её зависанию и потере времени игрового процесса на вычисления, из-за чего данный алгоритм ведётся не эффективным в данной ситуации.

**Волновой алгоритм**

Волновой алгоритм — это алгоритм, который позволяет найти минимальный путь в графе. Алгоритм поиска в ширину лежит в основе этого метода. В основном волновой алгоритм применяется для нахождения самого кратчайшего пути в графе, в общем случае находит лишь его длину.

Волновой алгоритм можно назвать одним из самых уникальных алгоритмов трассировки. Волновой алгоритм позволяет сформировать путь (трассу) между двумя ключевыми точками (элементами) в любом лабиринте, здесь необходимо отметить тот факт, что задача является разрешимой.

Основная идея волнового алгоритма описывается следующими этапами:

1. Из начального положения (элемента) волна распространяется в 4-х направлениях (рис. 1). Элемент, в который пришла волна, создает новый фронт волны. На рисунках цифрами обозначены номера фронтов волны.

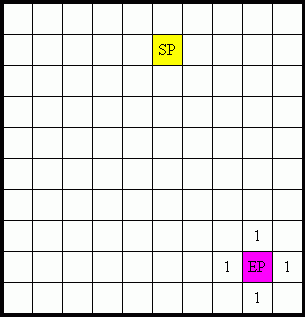
[](http://orionxl.ru/wp-content/uploads/2009/12/wave-algorithm-pic1.gif)

Рис. 1 Первый шаг

Каждый из элементов первого фронта волны будет является источником вторичной волны (рис 2.). Элементы второго фронта волны будут генерировать волну третьего фронта и т.д. Процесс формирования волн продолжается, пока не буде достигнут конечный элемент.

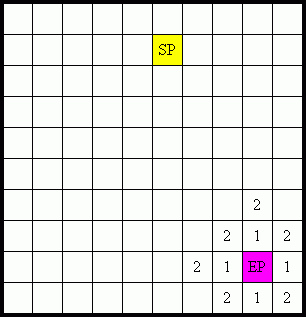
[](http://orionxl.ru/wp-content/uploads/2009/12/wave-algorithm-pic2.gif)

Рис. 2 Последующее распространение волны

2. На втором этапе волнового алгоритма строится сама трасса. Ее построение необходимо осуществлять в соответствии со следующими правилами:

a) Движение при построении трассы необходимо осуществлять в соответствии с выбранными приоритетами.

b) При движении от конечного элемента к начальному номер фронта волны (путевые координаты) должны уменьшатся.

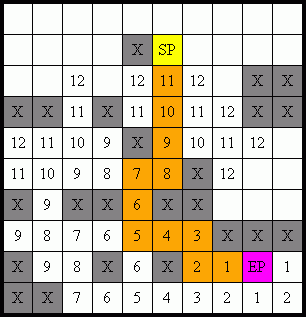
[](http://orionxl.ru/wp-content/uploads/2009/12/wave-algorithm-pic3.gif)

Рис. 3 Результат действия

Приоритеты направления движения при использовании волнового алгоритма нахождения пути выбираются на стадии разработки. Если изменять эти приоритеты, то можно получить разные трассы, НО длина трассы в любом случае остается одной и той же.

Преимущества волнового алгоритма в том, что с его помощью можно найти трассу в любом лабиринте и с любым количеством запретных элементов (стен). Единственным недостатком волнового алгоритма является, то, что при построении трассы требуется большой объем памяти.

# Разработка проекта

1. **Создание игрового меню**

Структура меню(см. рис.1):

1. Play
   1. Choose map1
   2. Choose map2
   3. Choose map3
2. Options
   1. Sound
3. Exit

Реализация игрового меню будет с помощью виджета Qt Creator и встроенных в него функций отрисовки.

Для связывания меню и подменю используется стековая структура.

Дизайн кнопок реализован с помощью графического редактора Paint.NET и

CorelDraw, а так же с использованием внутри игровых текстур.



Рисунок 1.

1. **Создание игровых уровней**

Для создания игровых уровней была использована программа для создания «тайловых» карт Tiled Map Editor. Игровые уровни были созданы со следующими условиями размер одной ячейки (тайла) 32 х 32 пикселя, так же карта содержит следующие слои:

* Background(рис. 2)
* Ground(рис. 3)
* Points

Первый слой задает основу карты, второй наносит объекты (растительность, деревья и т.д.), которые играют роль естественного заграждения, места, куда нельзя ничего поставить и где крипы не могут ходить. Последний слой задает начальную и конечную точку для крипов и алгоритма поиска пути.

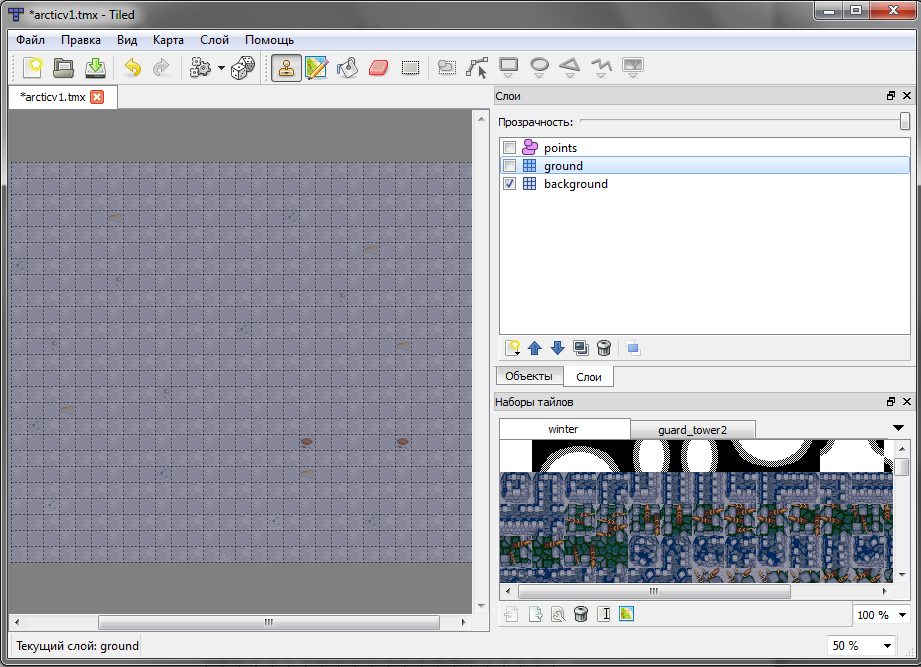


Рисунок 2

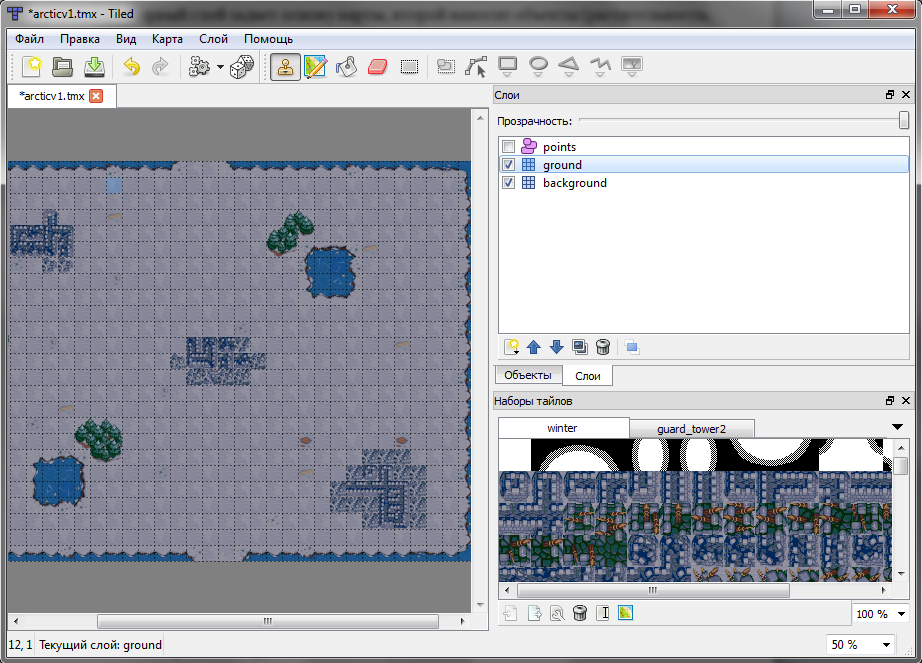


Рисунок 3

1. **Создание игрового окружения**

Создание игрового окружения началось с отрисовки карты и других объектов. В начале в классе gamewidget происходит парсинг карты созданно в TiledMap. Парсер, основанный на встроенной функции QXmlStreamReader, считывает параметры и передает их на отрисовку в другие функции. После того, как карта отрисована, она передаеться в класс Field, где происходит прорисовка крипов и башен.Для начала был создан класс Creep, который задает параметры одного крипа. Затем другой класс Creeps приминает этот класс создает массив классов Creep. Далее с помощью класса Field они появляются на карте.

Реалитзация отрисовки крипов:

void GameWidget::drawCreeps()

{

int fieldX = field.getSizeX();

int fieldY = field.getSizeY();

for(int y = 0; y < fieldY; y++)

{

for(int x = 0; x < fieldX; x++)

{

int pxlsX = spaceWidget + x\*sizeCell;//+1;

int pxlsY = spaceWidget + y\*sizeCell;//+1;

int localSizeCell = sizeCell;//-1;

int localSpaceCell = sizeCell/4;

if(int num = field.containCreep(x, y))

{

QColor color;

switch (num) {

case 1:

color = QColor(255, 0, 0);

break;

case 2:

color = QColor(0, 255, 0);

break;

case 3:

color = QColor(0, 0, 255);

break;

default:

break;

}

p.fillRect(pxlsX+1 + localSpaceCell, pxlsY+1 + localSpaceCell, localSizeCell-1 - 2\*(localSpaceCell), localSizeCell-1 - 2\*(localSpaceCell), color);

p.drawPixmap(pxlsX, pxlsY, localSizeCell, localSizeCell, field.getCreepPixmap(x, y));

}

}

}

}

Реализация отрисовки башен:

void GameWidget::drawTowers()

{

int fieldX = field.getSizeX();

int fieldY = field.getSizeY();

for(int y = 0; y < fieldY; y++)

{

for(int x = 0; x < fieldX; x++)

{

int pxlsX = spaceWidget + x\*sizeCell;//+1;

int pxlsY = spaceWidget + y\*sizeCell;//+1;

int localSizeCell = sizeCell;//-1;

if(field.containTower(x, y))

{

if(!mapLoad)

p.fillRect(pxlsX+1, pxlsY+1, localSizeCell-1, localSizeCell-1, QColor(127, 255, 0));

else

p.drawPixmap(pxlsX, pxlsY, localSizeCell, localSizeCell, field.getTowerPixmap(x, y));

}

}

}

}

# Руководство пользователя

Минимальные системные требования:

* 25 MB свободного места на жестком диске.
* 30 MB ОЗУ.
* Процессор с тактовой частотой 500MГц и выше
* Устройство ввода: мышь или клавиатура
* Видео-карта с видео-памятью 64 и выше

# Заключение

1. **В программе реализовано:**
   * Игра по схеме игрок-компьютер
   * Меню:
     + Новая игра
     + Опции
     + Выход из игры
2. **Достижения проекта:**

* Реализован алгоритм поиска пути, у которого время работы не превышает ~~0.1 секунду.
* Реализация атаки башен
* Реализация думающего алгоритма, его непредсказуемости
* Реализация редактора карт (на уровне разработчика)
* Приятное сочетание цветов и интерфейса

1. **Недостатки проекта:**

* Отсутствует графическое оформление атаки башен
* Отсутствие «горячих» клавиш

1. **Степень соответствия с техническим заданием:**

Полное соответствие тех. заданию

# Список литературы

1. « Вогел, Джефф. Шесть советов по написанию более понятного программного кода / Д. Вогел; — <<http://www.ibm.com/developerworks/ru/library/l-clear-code/index.html> > (26.07.2007 ).
2. C/C++ Visual C++ 2008 express для начинающих // Борис Пахомов (2009)
3. Searching for Solutions in Games and Artificial Intelligence // Allis, L. Victor (23.09.1994) <<http://fragrieu.free.fr/SearchingForSolutions.pdf> >
4. База данных MSDN <http://www.msdn.microsoft.com>
5. Основы программирования на С++ с использованием библиотеки Qt и среды разработки Qt Creator

# Приложение

Блок схема волнового алгоритма (поиска кратчайшего пути в лабиринте):

