Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
Игра tower defence

Работу выполнил:
Курякин Д. А.
Группа: 23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2016

Содержание

1 Игра tower defense

1.1 Игровые принадлежности

Tower Defense сокращенно TD — название жанра компьютерных стратегических игр. Задача игрока в играх подобного жанра — расправиться с наступающими врагами, называемыми «крипы» до того, как они пересекут карту, с помощью строительства башен, атакующих их, когда те проходят вблизи. Противники и башни различаются по характеристикам и цене. Когда враги побеждены, игрок зарабатывает очки, которые используются для покупки или модернизации башен. Подбор башен и их расположение — неотъемлемая стратегия игры. Ползучие твари пробегают через подобие лабиринта, что дает игроку возможность стратегического размещения башен.

1.2 Порядок использования

Пользователь ставит башенки на поле, затем нажимает на кнопку готовности, начинается появление крипов.

2 Проектирование приложения

2.1 Концепция приложения

В ходе проектирования было разработана концепция продукта. Созданное приложение должно предполагать возможность добавление и удаление башен с поля, появление крипов с последующим уничтожение базы и отстрел крипов башнями.

2.2 Минимально работоспособный продукт

Минимальном работоспособным продуктом было признано приложение, позволяющие производить игру.

2.3 Прецеденты использования

На основе разработанной концепции была составлена UML диаграмма прецедентов использования (рис.1).

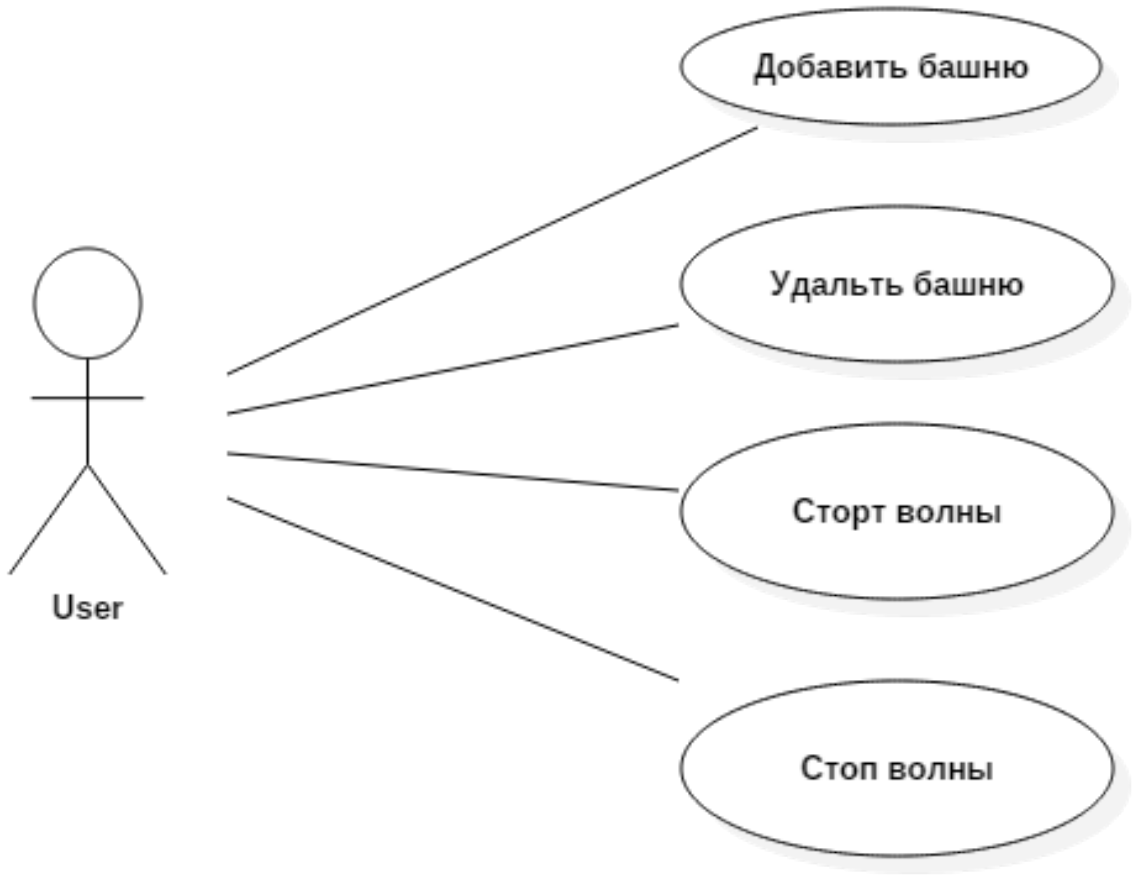

Рис. 1: Диаграмма прецедентов использования

2.4 Основные компоненты приложения

На основе анализа концепции и выделенных прецедентов использования было принято решение выделить два основных компонента, которые будут входить в состав продукта:

1. Библиотека

   Включает в себя игровую модель и реализует игровые механизмы. В ядре должно быть обеспечено регулярное обновление модели в ответ на действие пользователя. Кроме того, должна быть реализована обработка исключительных ситуаций, включающие в себя ошибки игрока, попытки выполнить запрещенные действия и прочее.

2. Графическое приложение

   Графически визуализирует игровую модель, предоставляет пользователю графический интерфейс для взаимодействия с ней и выполнения остальный действий предусмотренных в реализации библиотеки.

2.5 Используемые инструменты

Разработка в основном велась с использованием средств стандартной библиотеки Java в среде IntelliJ IDEA. Для создания графического интерфейса применялась библиотека Swing.

2.5.1 IntelliJ IDEA

IntelliJ IDEA — интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Go, Java, JavaScript, Python, разработанная компанией JetBrains.

2.5.2 Swing

Swing — библиотека для создания графического интерфейса для программ на языке Java. Swing был разработан компанией Sun Microsystems. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д.

Swing относится к библиотеке классов JFC, которая представляет собой набор библиотек для разработки графических оболочек. К этим библиотекам относятся Java 2D, Accessibility-API, Drag & Drop-API и AWT.

2.6 Выводы

Таким образом, была разработана концепция приложения, что позволило определить внешний вид продукта и выделить его основные компоненты.

3 Реализация приложения

3.1 Среда разработки

- Операционная система: Windows 10
- Интегрирование среда разработки: IntelliJ IDEA 2016.2.3
- Система автоматической сборки: Gradle 2.14
- Компилятор: javac, JDK 8.101

3.2 Реализация основных компонентов приложения

3.2.1 Библиотека Core

Для реализации всех запланированных функциональностей было принято решение, создать пятнадцать классов:

1. **Класс Enemy** Данный класс представляет возможность создания списка противников. Конструктор класса **Enemy** получает на вход индекс противника **id**, урон **damage**, жизни **health**и скорость **speed**. И добавляет противника в список.
2. **Класс EnemySlime** Наследуется от класса **Enemy**. Создаёт противника slime.
3. **Класс EnemyBigSlime** Наследуется от класса **Enemy**. Создаёт противника большой slime.
4. **Класс EnemyMove**

   Данный класс хранит в себе координаты каждого противника.

   Конструктор класса **EnemyMove** получает на вход ссылку на противника и координаты появления.
5. **Класс Base** Хранит в себе координаты базы на карте.
6. **Класс EnemyRoute** Конструктор класса **EnemyRoute** получает на вход индекс противника данные с карты и ищет координаты появления противника на карте с помощью класса **SpawnPoint**, добавляет координаты к себе в список. Метод **calculateRoute** с помощью метода **calculateNextPos** прощитывает путь от места появления противников до базы игрока.
7. **Класс EnemyAIMove** Перемещает выбранного противника по карте используя метод **move**. Метод **move** используя класс **EnemyRoute** передвигает противников.
8. **Класс EnemyAI** Конструктор класса **EnemyAI** с помощью класса **EnemyRoute** ищет дорогу и добавляет к себе в поле класса.
9. **Класс Tower** Данный класс представляет возможность создания списка башен. Конструктор класса **Tower** получает на вход индекс башни **id**, стоимость **cost**, радиус поражения **range**, урон **damage**. И добавляет башню в список.
10. **Класс TowerFire** Наследуется от класса **Tower**. Создаёт огненную башню.
11. **Класс TowerLightning** Наследуется от класса **Tower**. Создаёт электрическую башню.

12. **Класс TowerMap** Конструктор класса **TowerMap** создаёт матрицу размером 14 на 14 типа **Tower**. Метод **AddTower** принимает на вход координаты башни **x, y** и индекс башни **id** затем добавляет башню на поле вычитая стоимость постройки башни из класса **User**. Метод **DeleteTower** принимает на вход координаты башни **x, y** затем удаляет башню с поля прибавляя процент от стоимости башни в класса **User**.

13. **Класс User** Хранит в себе монеты и жизни игрока и проверяет. С помощью метода **gameOver** проверяет есть ли у базы игрока жизни если нет то игра окончена.

14. **Класс Map** С помощью класса **LevelFile** считывает данные о уровне с файла и добавляет к себе. С помощью класса **Level** ищет координаты появления противника.

15. **Класс Wave** С помощью метода **spawnEnemies** создает волны противников.

3.2.2 Графический интерфейс

Для создания графического интерфейса использовалась библиотека Java Swing. С помощью Qt Widgets было созданы классыд двух основных окон **Interface** и **FieldGUI**, которые отвечали за основное меню и вывод сотки кроссворда.

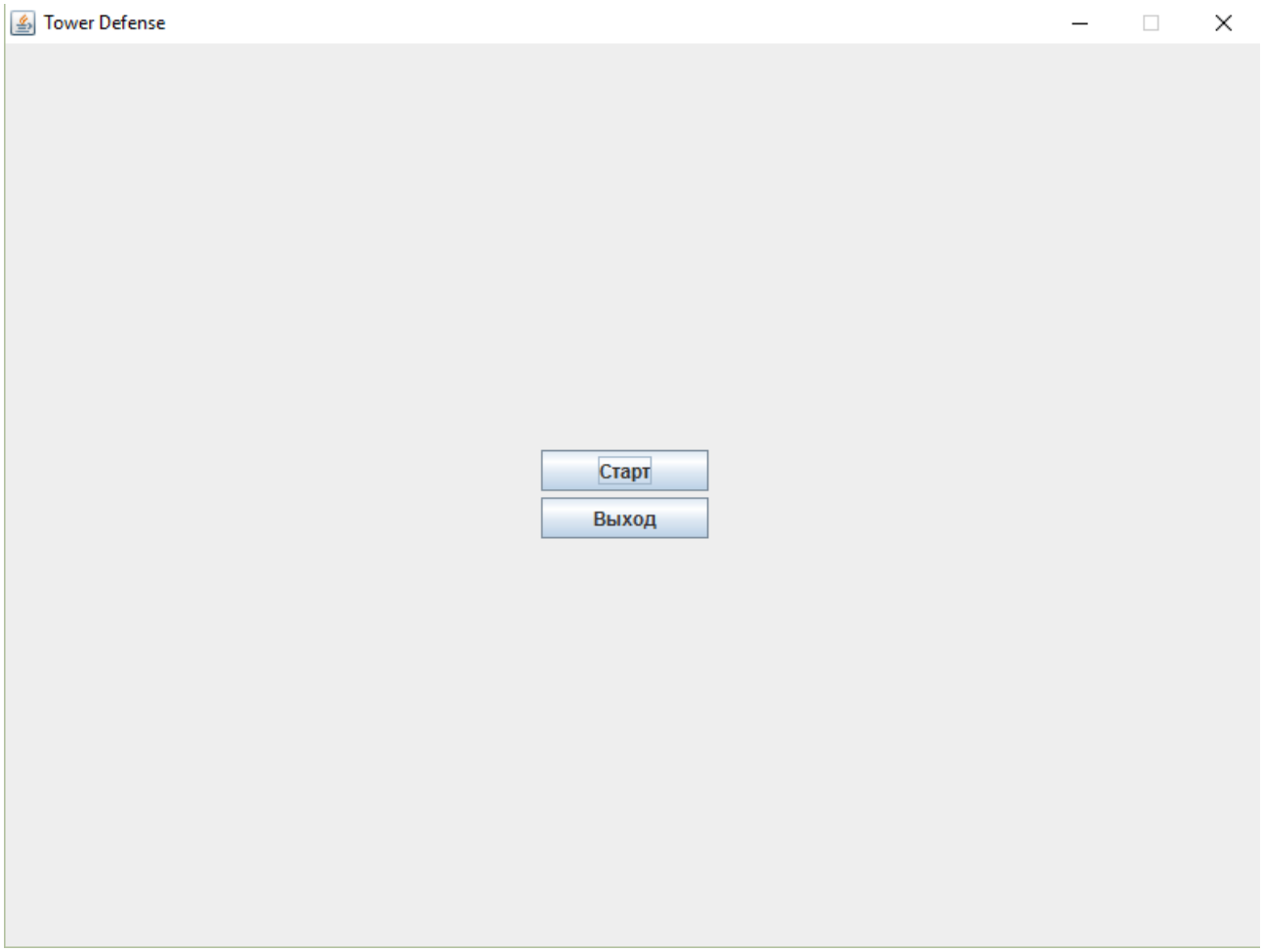

Рис. 2: Меню графического интерфейса

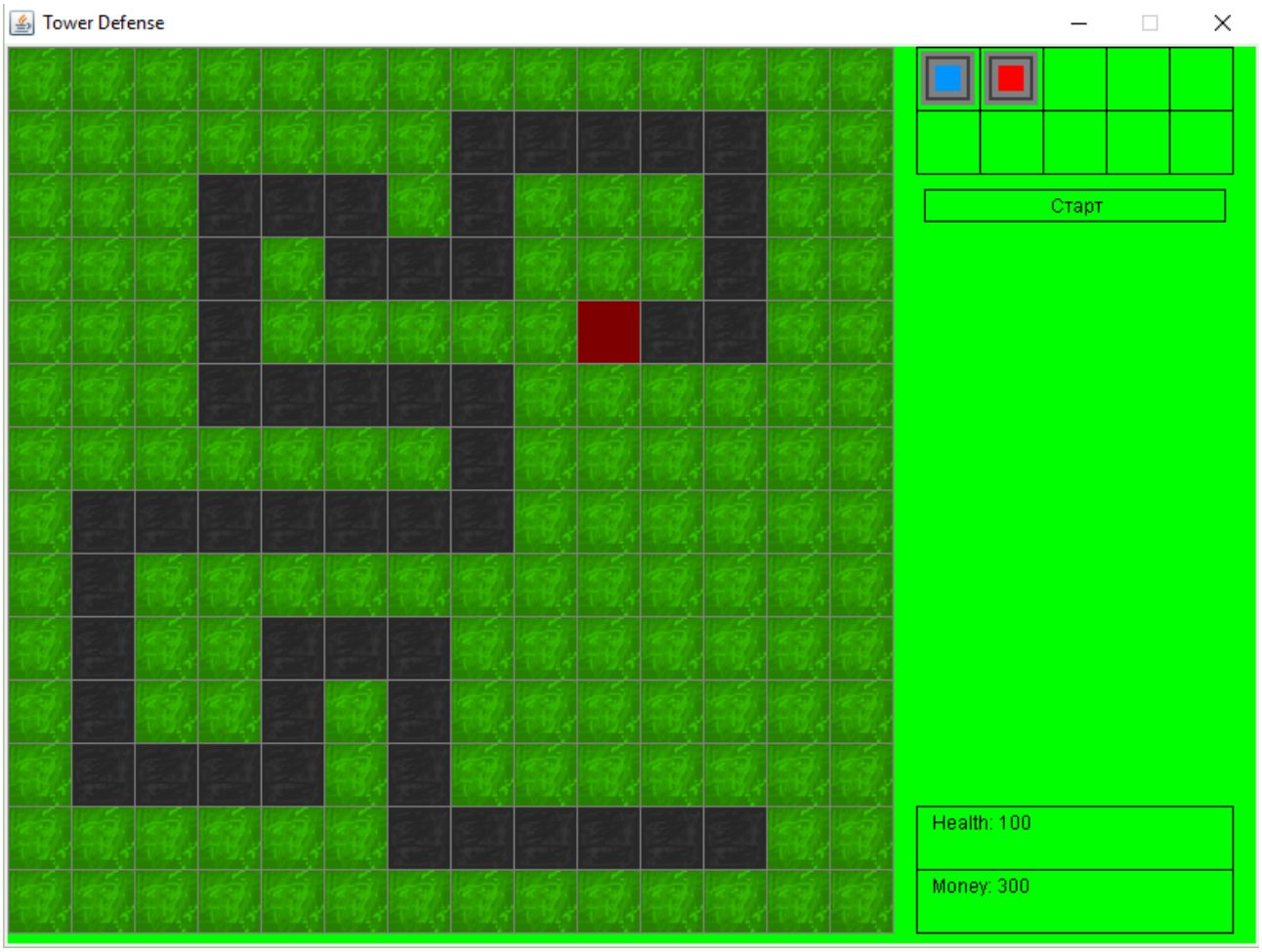

Рис. 3: Игровое окно графического интерфейса

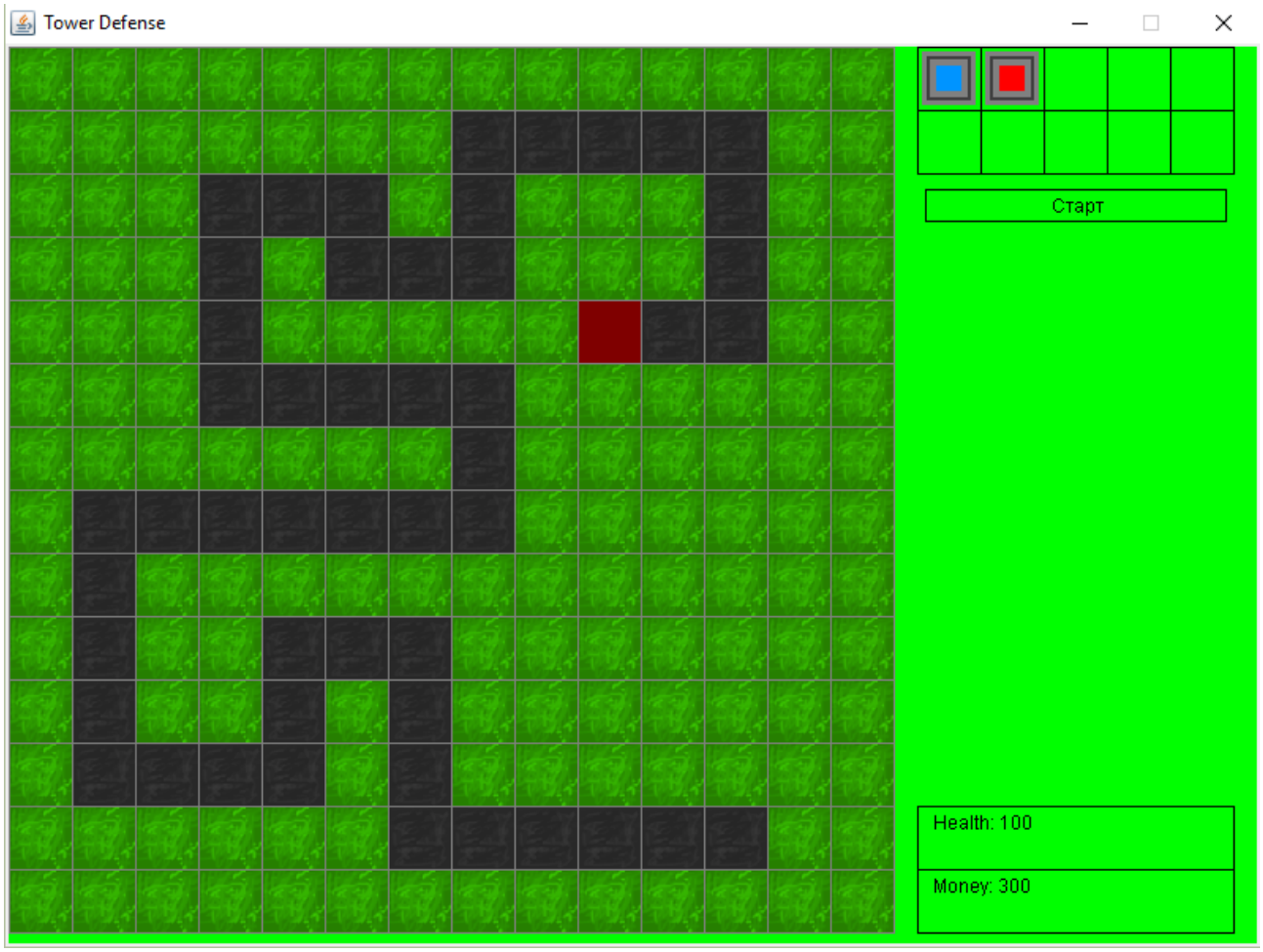

Рис. 4: Игровое окно графического интерфейса

На рисунках 2, 3 и 4 изображены основные окна графического приложения.

3.3 Теститрование

В ходе разработки проекта регулярно проводилось ручное тестирование.
Тестирование позволило обеспечить работоспособность продукта в ходе всего процесса разработки.

3.4 Демонстрации

Во время создания приложения было проведено 1 демонстрации, на которых группой людей, представляющих собой потенциальных пользоваталей разработываемого приложения, были сделаны различные замечания и высказаны множество предложений и пожеланий, основанных на внешнем виде продукта и стандартном цикле работы с ним. Анализ полученной информации позволял обнаруживать недочеты присутсвующие в продукте на том, или ином этапе разработки, а также определять дальнейшие направления улучшения и расширения проекта, что, безусловно, положительно сказалось на конечном результате.

4 Выводы

В ходе работы были получены навыки необходимые для написания программ на языке программирования Java. Во-первых была изучена стандартная библиотека, библиотека Swing и особенности данного языка. Во-вторых был получен опыт, связанный с процессом разработки программного продукта.

5 Приложение 1

5.1 Листинги