Министерство образования и науки Российской Федерации

Национальный исследовательский технологический университет «МИСиС»

Институт информационных технологий и   
автоматизированных систем управления

Кафедра Инженерной Кибернетики

**Курсовая работа   
по дисциплине «Технологии программирования»****на тему «Стратегическая игра Tower Defense»**

Выполнил:  
студент гр. БПМ-18-3  
Шилин В.В.

Проверил:  
доцент кафедры ИК, к.т.н.  
Полевой Д.В.

Москва, 2019

**Оглавление**

[**Описание задачи 3**](#_Toc29783257)

[**Пользовательское описание 5**](#_Toc29783258)

[Описание интерфейса 5](#_Toc29783259)

[**1. Главное меню 5**](#_Toc29783260)

[**2. Меню выбора уровня сложности 6**](#_Toc29783261)

[**3. Диалоговые окна 7**](#_Toc29783262)

[**4. Окно игры 7**](#_Toc29783263)

[**5. Кнопки управления 8**](#_Toc29783264)

[**6. Игровое поле 9**](#_Toc29783265)

[**7. Панель магазина 11**](#_Toc29783266)

[**8. Экран справки 13**](#_Toc29783267)

[**9. Экран “GameOver” 14**](#_Toc29783268)

[Описание игровой механики 14](#_Toc29783269)

[**Описание уровней сложности 14**](#_Toc29783270)

[**Описание постепенного усложнения игрового процесса 15**](#_Toc29783271)

[**Описание типов башен 15**](#_Toc29783272)

[**Описание типов NPC 15**](#_Toc29783273)

[**Описание видов волн 15**](#_Toc29783274)

[**Описание системы улучшений 16**](#_Toc29783275)

[**Техническое описание 17**](#_Toc29783276)

[**Инструкция по сборке 18**](#_Toc29783277)

[**Приложения 20**](#_Toc29783278)

[Приложение 1 20](#_Toc29783279)

# **Описание задачи**

Целью курсовой работы было создание приложения, представляющего собой стратегическую игру “Tower Defense”, суть которой заключается в размещении оборонных башен, атакующих ближайших NPC (противников). Задача NPC состоит в том, чтобы пересечь карту и дойти до финиша. Задача игрока – помешать NPC и уничтожить их до того, как они пересекут карту, а также продержаться как можно дольше, выдержав как можно больше волн.

Программа предназначена для конечного пользователя.

**Требования**

Интерфейс:

* Наличие многоуровневого главного меню (главное меню и меню выбора сложности);
* Наличие диалогового окна подтверждения начала новой игры;
* Возможность сохранить начатую игру перед выходом (выбор осуществляется с помощью диалогового окна);
* Возможность продолжения сохраненной игры;
* Возможность начать игру заново;
* Возможность ставить игру на паузу;
* Возможность отображения экрана справки во время игры;
* Возможность включения и отключения музыкального сопровождения игры;
* Сохранение и загрузка последних настроек музыкального сопровождения.

Игровая механика:

* Возможность выбора 1 из 4 уровней сложности, от которого будет зависеть начальное значение игровой валюты и количество жизней игрока, а также характеристики NPC;
* Наличие 4 типов NPC с разными характеристиками и разной дополнительной сопротивляемостью физическому и магическому урону:
  + без дополнительной сопротивляемости;
  + с дополнительной сопротивляемостью физическому урону;
  + с дополнительной сопротивляемостью магическому урону;
  + с дополнительной сопротивляемостью любому урону;
* Анимированное передвижение NPC, зависимое от скорости перемещения и его направления;
* Наличие 3 типов башен по виду наносимого урона:
  + физический урон;
  + магический урон;
  + общий урон;
* Возможность покупки и установки выбранной башни на игровое поле за имеющуюся игровую валюту;
* Возможность улучшения установленных башен за имеющуюся игровую валюту;
* Игровая валюта дается игроку за уничтожение NPC;
* Наличие 5 разных видов волн;
* Время появление NPC, их тип и количество зависит от номера волны;
* Постепенное усложнение игрового процесса с помощью увеличивающегося множителя;
* Периодическое пополнение здоровья игрока в зависимости от вида пройденной волны.

Язык: C++

Среда разработки: Visual Studio 2019

Сторонние библиотеки: SFML

# **Пользовательское описание**

## Описание интерфейса

При запуске программы открывается окно главного меню. Также открывается консоль, в которой сообщается, если не удалось загрузить сохранение игры и/или конфигурации музыки. В случае ошибок загрузки ресурсов (текстуры/музыка/шрифт) об этом также сообщается в консоли.

### **Главное меню**

Главное меню предоставляет следующие возможности:

* Продолжить сохраненную игру (кнопка “Continue” – неактивна при отсутствии сохранений или уже начатой игры);
* Начать новую игру (кнопка “New game”);
* Выйти из игры (кнопка “Exit”);
* Включение/отключение музыкального сопровождения (кнопка управления в правом верхнем углу).

  
Рисунок 1.1. Главное меню  
(Кнопка “Continue” неактивна)

При наличии сохранения или уже начатой игры кнопка “Continue” активна и есть возможность продолжить игру, тогда при нажатии на нее открывается окно игры (раздел 4). Вернуться в игру также можно с помощью клавиши “Esc”.

  
Рисунок 1.2. Активная кнопка “Continue”

Также при наличии сохранения или уже начатой игры, нажимая “New game”, “Exit” или на “Крестик”, закрывая окно, открывается соответствующее диалоговое окно (раздел 3).

\*Описание кнопки управления музыкальным сопровождением находится в разделе 5.

### **Меню выбора уровня сложности**

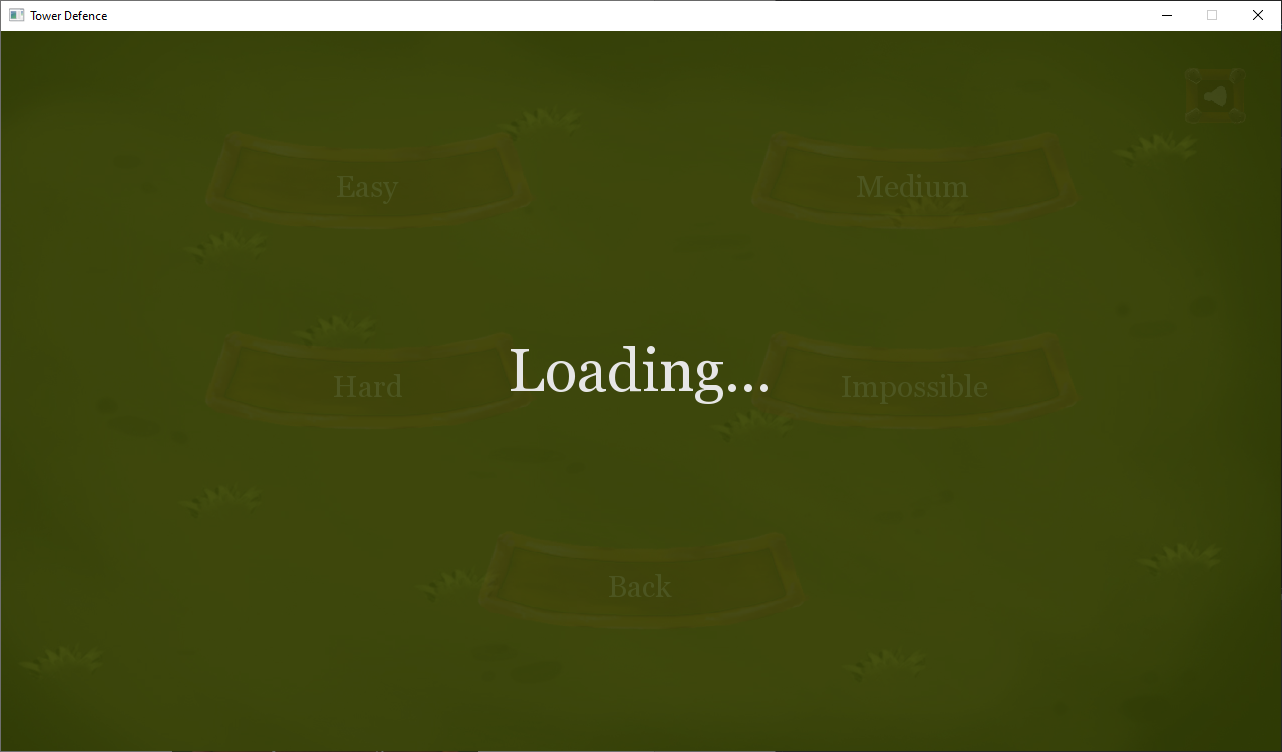
Данное меню открывается при нажатии на кнопку “New game”. Оно предоставляет возможность выбрать сложность для новой игры или вернуться назад в главное меню (раздел 1).

Также, как и в главном меню, имеется кнопка управления музыкальным сопровождением в правом верхнем углу (раздел 5).

Рисунок 2.1. Меню выбора уровня сложности

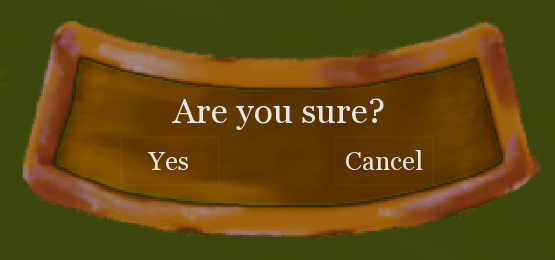
При нажатии на кнопку “Back” игрок возвращается в главное меню (раздел 1).

При выборе одного из четырех уровней сложности и нажатии на соответствующий начинается загрузка (Рисунок 2.2) и перед игроком появляется экран загрузки, после которого открывается экран справка (раздел 8), а после его закрытия – окно игры (раздел 4).

  
Рисунок 2.2 Экран загрузки

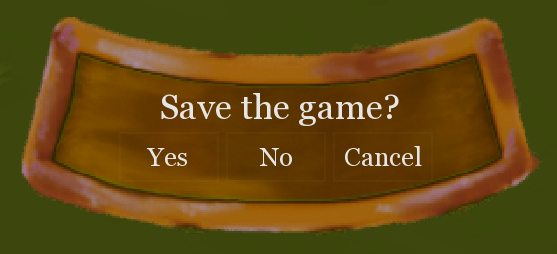
### **Диалоговые окна**

При выборах начать новую игру, выйти из игры (в главном меню – раздел 1) или при желании завершить программу, закрыв окно, появляются диалоговые окна с соответствующими подтверждениями (рисунки 3.1 и 3.2)

  
Рисунок 3.1. Диалоговое окно  
подтверждения начала новой игры

При выборе “Yes” диалоговое окно закрывается и игрок переходит в меню выбора уровня сложности (раздел 2).

В случае нажатия на “Cancel” или в пустое место вне диалогового окна оно также закрывается и игрок возвращается в главное меню (раздел 1).

  
Рисунок 3.2. Диалоговое окно

При выборе “Yes” игра завершается, предварительно сохранив имеющийся прогресс игры и конфигурацию музыкального сопровождения.

В случае с “No” игра завершается, сохранив только настройки музыки.

При нажатии на “Cancel” или в пустое место вне диалогового окна оно закрывается и игрок возвращается к экрану, который был до вызова данного диалогового окна.

### **Окно игры**

Окно игры является основным, так как в нем происходит весь процесс геймплея.

Окно игры делится на 3 визуальные части:

* Левая – игровое поле (раздел 6);
* Правая - верхняя – кнопки управления (раздел 5);
* Правая - нижняя – панель магазина (раздел 7).

Рисунок 4.1. Окно игры при выборе сложности “Easy”  
в самом начале игрового процесса

Для открытия главного меню игроку необходимо нажать клавишу “Esc”.

Также при попытке закрыть окно, нажав на “Крестик”, перед игроком открывается соответствующее диалоговое окно с предложением сохранить игру (раздел 3).

### **Кнопки управления**

* Кнопка управления музыкальным сопровождением включает и выключает музыку в игре, а также меняет оттенок в зависимости от того, включена музыка или нет (зеленый – вкл., красный – выкл.);

   
Рисунки 5.1.1, 5.1.2. Слева – музыка включена,  
справа – отключена

* Кнопка управления игровым процессос запускает и останавливает игровой процесс, а также меняет оттенок в зависимости от того, запущен игровой процесс или нет (зеленый – запущен, красный – остановлен). При остановленном процессе останавливается таймер отсчета до следующей волны, NPC и снаряды замирают на месте, а башни перестают создавать новые снаряды;

** **Рисунки 5.2.1, 5.2.2. Слева – игра запущена,  
справа – стоит на паузе

* Кнопка управления экраном справки открывает и закрывает справку (раздел 8), а также меняет оттенок в зависимости от того, открыта или закрыта справка (зеленый – открыта, красный – закрыта).

   
Рисунки 5.3.1, 5.3.2. Слева – экран справки открыт,  
справа – закрыт

### **Игровое поле**

При запуске новой игры перед игроком появляется “чистое” игровое поле.

****Рисунок 6.1.” Чистое” игровое поле  
в начале игры со сложностью “Easy”

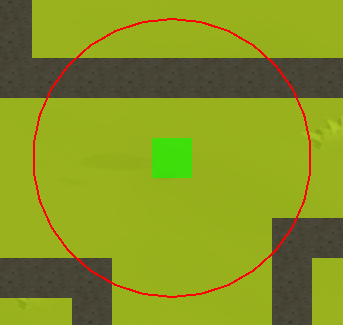
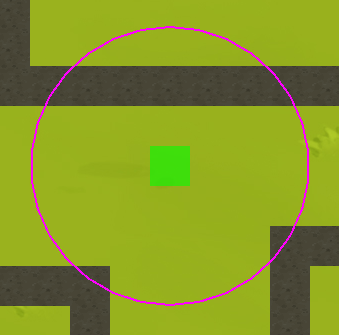
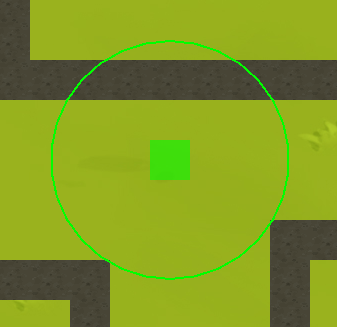
Во время игрового процесса игрок расставляет на игровом поле башни, а также по проложенному пути с анимацией передвигаются NPC, появляющиеся сверху и направляющиеся по дороге вниз. Направление их анимации зависит от направления их движения (в случае с движением по вертикальным путям, то остается последнее направление анимации).

  
Рисунок 6.2. Игровое поле во время игрового процесса

Сверху игрового поля игроку предоставляется базовая информация его игрового процесса (игровая валюта, номер проходящей волны, количество жизней игрока).

  
Рисунок 6.3 Базовая информация об игровом процессе.  
Слева направо: игровая валюта, номер проходящей волны, количество жизней

Взаимодействуя с панелью магазина (раздел 7) игрок может выбирать башни для установки их на игровое поле. При этом отображается радиус поражения выбранной башни, цвет которого зависит от типа башни. (башня с физическим уроном – красный, с магическим – фиолетовый, с общим – зеленый).

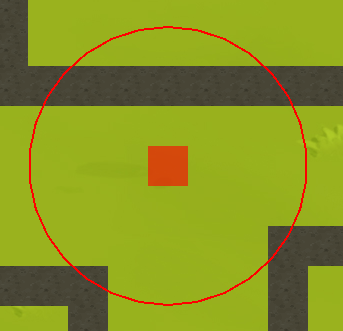
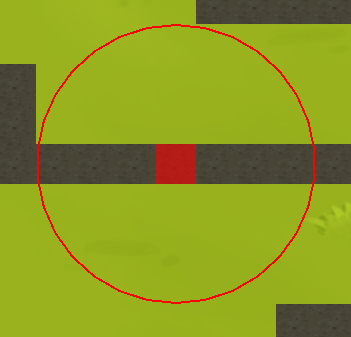
    
Рисунки 6.4.1, 6.4.2, 6.4.3. Отображение на игровом поле  
процесса выбора башни и места ее установки

После удачной установки башни в выбранную ячейку нажатием левой кнопки мыши она приобретается за счет игровой валюты и остается выбранной для последующей возможности ее улучшения. Для отмены выбора конкретной установленной башни необходимо нажать в любое место, либо клавишу “Esc”.

   
Рисунки 6.5.1, 6.5.2. Слева – выбранная установленная башня,  
справа – после отмены выбора установленной башни.

Во время игрового процесса не всегда происходит удачное приобретение башни. Если выделенный квадрат под место установки горит красным (а не зеленым, как на рисунках 6.4), то установка башни в данную клетку не возможна. Этому могут препятствовать следующие причины:

1. Установка производится на место уже установленной башни;
2. Попытка приобрести башню при отсутствии необходимого количества игровой валюты;
3. Установка производится на выделенный путь для NPC.

    
Рисунки 6.6.1, 6.6.2, 6.6.3. Иллюстрации перечисленных причин  
последовательно (a, b, c)

Для отмены установки выбранной башни необходимо кликнуть в пустое место панели магазина (раздел 7) или нажать клавишу “Esc”.

При окончании игры, когда количество жизней игрока становится равным 0, то игровой процесс останавливается и отображается экран “GameOver” (раздел 9).

### **Панель магазина**

На панели магазина отображаются все типы башен, а также кнопка “upgrade”.

Рисунок 7.1. Панель магазина с неактивной кнопкой “upgrade”

Чтобы узнать краткие характеристики одной из башен, игрок может навести на желаемую башню курсором для отображения необходимой информации. (Для отображения более подробной информации необходимо открыть экран справки (раздел 8) с помощью кнопки управления экраном справкой (раздел 5).

  
Рисунок 7.2. Отображение кратких характеристик  
башни при наведении на соответствующую иконку курсором

Чтобы выбрать башню для приобретения и установки необходимо кликнуть на иконку желаемой башни на панели магазина. После этого иконка выделится сначала силуэтом, а затем подсветится.

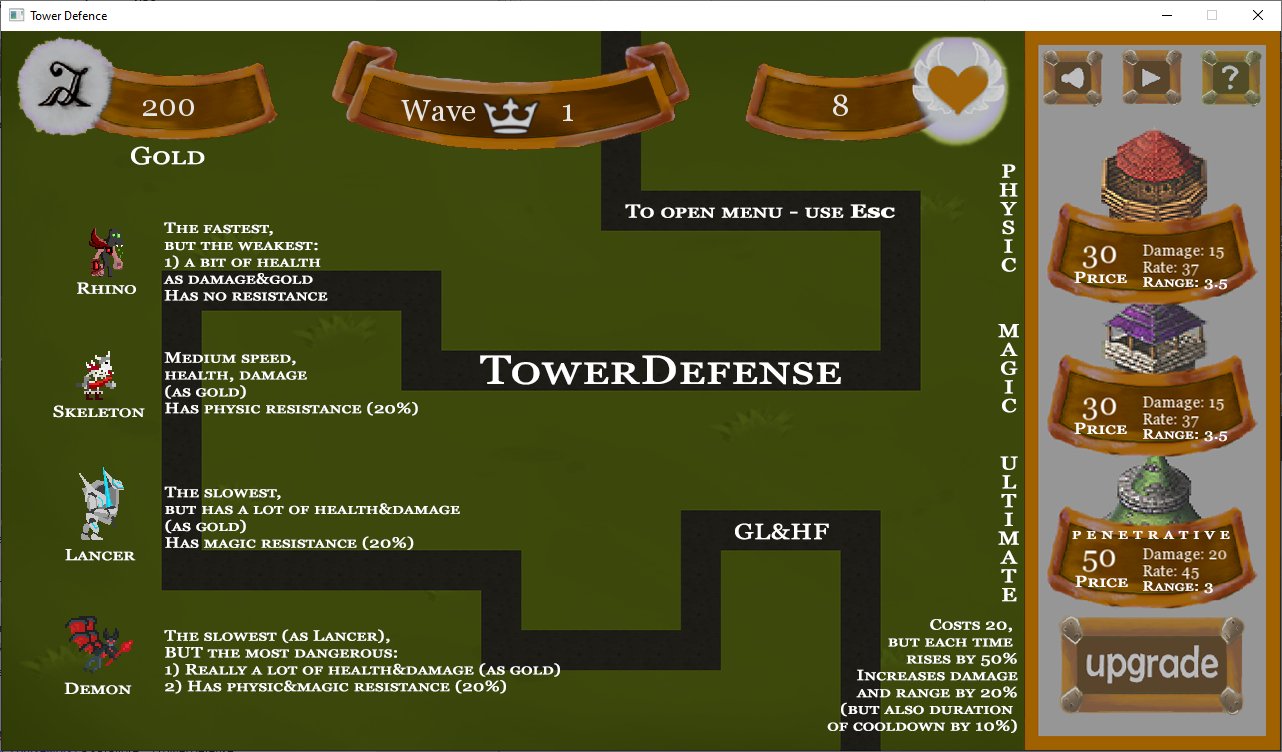
   
Рисунки 7.3.1, 7.3.2. Слева – выделение иконки силуэтом при нажатии,  
справа подсвечивание иконки при выбранной башни для установки

При выбранной установленной башни на игровом поле (вокруг нее отображается ее радиус поражения) кнопка “upgrade” становится активной, если имеется достаточное количество игровой валюты для улучшения выбранной башни. Чтобы улучшить необходимо нажать на активную кнопку “upgrade”.

****Рисунок 7.3. Активная кнопка “upgrade”

### **Экран справки**

На экране справки подробно описаны характеристики башен и NPC, поясняется значение игровой валюты, дается подсказка насчет улучшения башен. Также говорится, как открыть главное меню.

  
5.3.3 Экран справки

Экран справки закрывается нажатием на кнопку управления экраном справкой или в любое другое место окна. Также его можно закрыть нажатием клавиши “Esc”.

### **Экран “GameOver”**

Когда количество жизней игрока заканчивается (становится равное 0), то игровой процесс останавливается и отображается экран “GameOver”.

Рисунок 9.1. Экран “GameOver”

Для выхода в главное меню необходимо кликнуть в любое место или нажать на клавишу “Esc”.

Для выхода из игры можно закрыть окно с помощью “крестика” или выйти в главное меню и нажать “Exit”.

Для перезапуска игры необходимо выйти в меню и нажать на “New game” и выбрать желаемую сложность.

## Описание игровой механики

Было реализовано приложение, удовлетворяющее поставленной задаче. Уровни сложности, типы башен, типы NPC, виды волн, валютная система игры были подобраны опционально и откалиброваны.

### **Описание уровней сложности**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровни сложности: | “Easy” | “Medium” | “Hard” | “Impossible” |
| Начальное количество игровой валюты | 300 | 250 | 250 | 200 |
| Начальное количество жизней игрока | 20 | 20 | 15 | 10 |

Также скорость NPC, количество игровой валюты, получаемое при его уничтожении, и количество жизней NPC зависят прямо пропорционально от множителя сложности и/или порядкового номера набора из 4 волн. Наносимый урон игроку не меняется.

### **Описание постепенного усложнения игрового процесса**

Наборы из 4 видов волн чередуются друг за другом до 21 волны, каждый раз увеличивая множитель сложности в 2 раза. С 21 волны начинаются чередоваться только волны 5 вида, каждый раз увеличивая множитель в 3 раза.

### **Описание типов башен**

В таблице приведены характеристики каждого типа башни.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип башни | Цена | Урон | Радиус поражения, кл | Время перезарядки, с |
| Физического урона | 30 | 15 | 3.5 | 0.37 |
| Магического урона | 30 | 15 | 3.5 | 0.37 |
| Общего урона | 50 | 20 | 3 | 0.45 |

### **Описание типов NPC**

В таблице приведены характеристики каждого типа NPC.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип NPC | Вознаграждение | Здоровье | Урон игроку | Скорость |
| Без сопр. ур. | 1 | 40 | 1 | 3 |
| С сопр. физ. ур. | 2 | 60 | 2 | 2 |
| С сопр. маг. ур. | 3 | 80 | 3 | 1 |
| С сопр. люб. ур. | 4 | 100 | 4 | 1 |

Для баланса наличие сопротивляемости у NPC, снижает получаемый им урон соответствующего вида на 20%.

### **Описание видов волн**

В таблице приведено количество появляющихся NPC каждого типа для каждого вида волны.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид волны | Без сопр. ур. | С сопр. физ. ур. | С сопр. маг. ур. | С сопр. люб. ур. |
| 1 вид | 4 | 3 | 2 | 0 |
| 2 вид | 2 | 5 | 3 | 0 |
| 3 вид | 2 | 2 | 6 | 0 |
| 4 вид | 2 | 4 | 2 | 3 |
| 5 вид | 4 | 3 | 2 | 9 |

Каждая последующая волна начинается через 10 секунд после начала предыдущей независимо от ее вида.

### **Описание системы улучшений**

Стоимость улучшения установленной башни составляет 20 единиц игровой валюты, но стоимость улучшенной башни каждый раз возрастает на 50%.

Каждое улучшение установленной башни увеличивает ее урон и радиус поражения на 20%, а время перезарядки – на 10%.

# **Техническое описание**

Стратегическая игра «Tower Defense» – кроссплатформенное приложение, собираемое с помощью CMake; доступна генерация документации Doxygen. Для реализации использовалась кроссплатформенная мультимедийная библиотека SFML.

В качестве дополнительных ресурсов были использованы изображения, шрифт, аудиофайл.

Реализованные классы с их кратким описанием:

|  |  |
| --- | --- |
| Bullet | Снаряд |
| BulletController | Сопряжение контроллеров башен и NPC и управление снарядами |
| Clickable | Обеспечивание реагирования объектов на Mouse |
| Enemy | Враждебный NPC |
| EnemyController | Управление NPC |
| GameBoard | Взаимодействие с игровым полем |
| GameMenuController | Управление игровым меню |
| GameState | Состояние игры |
| InfoButton | Кнопка справки |
| MenuButton | Кнопка меню |
| PauseButton | Кнопка паузы |
| Timer | Таймер |
| Tower | Оборонительная башня |
| TowerButton | Кнопка башни в меню |
| TowerController | Управление башнями |
| UpgradeButton | Кнопка улучшения |
| VolumeButton | Кнопка музыкального сопровождения |

Более подробное описание – см. Приложение 1.

Создаваемые сохранения хранятся в бинарном виде в текстовых файлах в каталоге data.

# **Инструкция по сборке**

Для сборки данной программы необходим компилятор, совместимый со стандартом C++14 и CMake версии не ниже 3.15.

Последнюю версию CMake можно скачать по ссылке:

<https://cmake.org/download/>

Для сборки проекта необходима библиотека SFML, доступная по ссылке:

<https://www.sfml-dev.org/download.php> для установки.

При необходимости генерации документации необходимо иметь установленную программу Doxygen, доступную по ссылке:

<http://www.doxygen.nl/download.html>

Исходный код, файлы проекта и файлы cmake находятся по ссылке:

<https://mysvn.ru/Jivty/shilin_v_v/shilin_v_v_coursework/>   
(в каталоге …/TowerDefense)

Сборка выполняется при помощи кроссплатформенной утилиты CMake. В инструкции описана сборка с использованием Visual Studio 2019 в операционной системе Windows.

1. Запустить установленный CMake;
2. Указать источник сборки – каталог, содержащий исходные файлы (…/TowerDefense);
3. Указать каталог для генерации файлов проекта.
4. Запустить конфигурацию (выбрать в качестве generator – Visual Studio 16 2019) и выбрать необходимые настройки генерации:
   * В случае возникновении ошибки (“Could not find a package…) в поле SFML\_DIR вручную указать путь до установленной библиотеки SFML (…{prefix}/{SFML}/lib/cmake/SFML). (Значения в фигурных скобках могут отличаться, так как зависят от установки библиотеки SFML)
   * При необходимости генерации документации установить галочку напротив BUILD\_WITH\_DOCS;
5. Запустить генерацию проекта.
6. Запустить собранный проект в Visual Studio 2019.
7. В обозревателе решений (Solution explorer) нажать правой кнопкой мыши по INSTALL. Выбрать: Только проект (Project only) -> Собрать только INSTALL (Build only INSTALL).
8. При необходимости генерации документации в обозревателе решений (Solution explorer) нажать правой кнопкой мыши по doc. Выбрать: Только проект (Project only) -> Собрать только doc (Build only doc). Документация будет находиться внутри каталога со сгенерированным проектом в каталоге …/doc/html.
9. Установить TowerDefense как автозагружаемый проект (Set as startup project).
10. В случае возникновения ошибок:
    * Проверить соответствие папок data, font, images, music внутри каталога со сгенерированный проектом.
    * Для конфигурации Debug: в случае отсутствия поместить библиотеки openal32.dll, sfml-audio-d-2.dll, sfml-graphics-d-2.dll, sfml-system-d-2.dll, sfml-window-d-2.dll из каталога {prefix}/{SFML}/bin в папку Debug внутри каталога со сгенерированным проектом.
    * Для конфигурации Release: в случае отсутствия поместить библиотеки openal32.dll, sfml-audio-2.dll, sfml-graphics-2.dll, sfml-system-2.dll, sfml-window-2.dll из каталога {prefix}/{SFML}/bin в папку Release внутри каталога со сгенерированным проектом.

Если ваша операционная система не Windows, то необходимо вместо пункта 7 выполнить пункт 10, в случае возникновения ошибок воспользуйтесь официальной инструкцией по сборке, опубликованной на сайте библиотеки SFML:

<https://www.sfml-dev.org/tutorials/2.5/compile-with-cmake.php>

В результате сборки образуются следующие файлы и каталоги:

* TowerDefense.exe – основной исполняемый файл проекта;
* Вспомогательные файлы и библиотеки;
* Опционально: каталог doc – автоматически сгенерированная документация.

При желании запускать приложение через TowerDefense.exe (находящийся в каталоге ../Debug или …/Release в зависимости от конфигурации) необходимо скопировать в каталог с данным исполняемым файлом папки data, font, images, music из каталога со сгенерированным проектом (или из каталога, содержащего исходные файлы, …/TowerDefense)

При необходимости генерации документации также можно использовать программу Doxywizard, устанавливаемую вместе с Doxygen.

# **Приложения**

## Приложение 1

**Класс Bullet**

Снаряд.

#include <**Bullet.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **Bullet** ()=default |
|  | Умолчательный конструктор. |
|  | |
|  | **Bullet** (int ind, int damage, sf::Vector2f towerPosition, std::weak\_ptr< **Enemy** > enemyToFireAt, TowerType tType) |
|  | Конструирование снаряда по параметрам. |
|  | |
|  | **~Bullet** ()=default |
|  | Деструктор. |
|  | |
| void | **drawBullet** (std::shared\_ptr< sf::RenderWindow > \_window) |
|  | Отображение снаряда. |
|  | |
| void | **moveObjectTowardsEnemy** (sf::Vector2f enemyPos) |
|  | Перемещение снаряда по пути к атакуемому NPC. |
|  | |
| float | **DistanceFromEnemy** (sf::Vector2f enemy) |
|  | Расстояние до атакуемого NPC. |
|  | |
| bool | **bulletEnemyCollision** (sf::Vector2f enemy) |
|  | Возвращает true, если снаряд достаточно близок к атакуемому NPC. |
|  | |
| sf::Vector2f | **normalize** (sf::Vector2f v) |
|  | Обеспечение плавного перемещения снаряда. |
|  | |
| sf::Vector2f | **getBulletPosition** () |
|  | Позиция снаряда. |
|  | |
| float | **getSpeed** () |
|  | Скорость снаряда. |
|  | |
| std::weak\_ptr <Enemy> | **getEnemyTarget** () |
|  | Атакуемый NPC. |
|  | |
| int | **getDamage** () |
|  | Наносимый снарядом урон. |
|  | |
| sf::Vector2f | **getBulletCenter** () |
|  | Координаты центра снаряда. |
|  | |
| TowerType | **getType** () |
|  | Тип атакуемого NPC. |

Объявления и описания членов классов находятся в файлах:

* **Bullet.h**
* Bullet.cpp

**Класс BulletController**

Сопряжение контроллеров башен и NPC и управление снарядами.

#include <**BulletController.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **BulletController** (std::shared\_ptr< sf::RenderWindow > window, std::shared\_ptr< **GameState** > gameState, std::shared\_ptr< **EnemyController** > enemyController, std::shared\_ptr< std::vector< std::shared\_ptr< **Tower** >>> allTowers, std::shared\_ptr< std::vector< std::shared\_ptr< **Enemy** >>> allEnemy) |
|  | Конструктор по параметрам. |
|  | |
|  | **~BulletController** ()=default |
|  | Деструктор |
|  | |
| void | **addThrowObjectToList** (int index, int damage, sf::Vector2f towerPosition, std::weak\_ptr< **Enemy** > enemyPtr, TowerType tType) |
|  | Создание нового снаряда. |
|  | |
| void | **drawAllBulletsOnGrid** (std::shared\_ptr< sf::RenderWindow > theWindow) |
|  | Отображение всех установленных башен. |
|  | |
| void | **deleteBullet** (**Bullet** \*bullet) |
|  | Удаление снаряда. |
|  | |
| void | **updateBullets** () |
|  | Обновление летящих снарядов. |
|  | |
| void | **updateTowerEnemyRange** () |
|  | Обновление списков NPC в зоне поражения. |
|  | |
| void | **update** () |
|  | Обновление перезарядок и создание новых снарядов. |
|  | |
| void | **render** () |
|  | Отображение снарядов. |

Объявления и описания членов классов находятся в файлах:

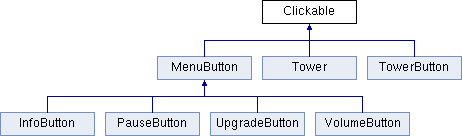
* **BulletController.h**
* BulletController.cpp

**Класс Clickable (abstract)**

Обеспечивание реагирования объектов на Mouse.

#include <**Clickable.h**>

Граф наследования:Clickable:



|  |  |
| --- | --- |
| Открытые члены | |
|  | **Clickable** ()=default |
|  | Умолчательный конструктор. |
|  | |
|  | **Clickable** (sf::Vector2f \_position, sf::Vector2f \_size) |
|  | Конструктор по позиции и размеру. |
|  | |
| virtual | **~Clickable** () |
|  | Виртуальный деструктор. |
|  | |
| const sf::Vector2f & | **getPosition** () const |
|  | \breif Получение позиции. |
|  | |
| void | **setPosition** (const sf::Vector2f &\_position) |
|  | Установка позиции. |
|  | |
| const sf::Vector2f & | **getSize** () const |
|  | Получение размера. |
|  | |
| void | **setSize** (const sf::Vector2f &\_size) |
|  | Установка размера. |
|  | |
| void | **process** (const sf::Event &event, const sf::Vector2i &mousePos) |
|  | Обрабатывание события. |
|  | |
| virtual void | **update** ()=0 |
|  | Обновление. |
|  | |
| virtual void | **render** (sf::RenderWindow &\_window)=0 |
|  | Отображение. |

|  |  |
| --- | --- |
| Статические открытые данные | |
| static const int | **gridSize** { 40 } |
|  | Размер ячейки игрового поля, стандартизирующий масштаб. |

Объявления и описания членов классов находятся в файлах:

* **Clickable.h**
* Clickable.cpp

**Класс Enemy**

Враждебный NPC.

#include <**Enemy.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **Enemy** ()=default |
|  | Умолчательный конструктор. |
|  | |
|  | **Enemy** (sf::Vector2f position, sf::Vector2f size, std::shared\_ptr< sf::Texture > texture, int \_gold, int \_health, int \_damage, float \_pixelSpeed, EnemyType type, int targetPath=0) |
|  | Конструитровение NPC по параметрам. |
|  | |
|  | **~Enemy** ()=default |
|  | Деструктор. |
|  | |
| void | **setFrames** (std::shared\_ptr< sf::Texture > texture, EnemyType type) |
|  | Установка кадров анимации при перемещении. |
|  | |
| void | **nextFrame** () |
|  | Установка следующая кадра. |
|  | |
| void | **render** (std::shared\_ptr< sf::RenderWindow > window) |
|  | Отображение NPC. |
|  | |
| EnemyType | **getEnemyType** () const |
|  | Получение типа NPC. |
|  | |
| int | **getGoldValue** () |
|  | Получаемое количество монет за уничтожение NPC. |
|  | |
| int | **getHealth** () |
|  | Здоровье NPC. |
|  | |
| int | **getDamage** () |
|  | Наносимый урон игркоку по заверешнии пути. |
|  | |
| int | **getPixelSpeed** () |
|  | Скорость передвижения. |
|  | |
| int | **getPathIndex** () |
|  | Индекс участка пути. |
|  | |
| sf::Vector2f | **getPosition** () |
|  | Позиция NPC. |
|  | |
| sf::Vector2f | **getCenterPosition** () |
|  | Позиция центра NPC. |
|  | |
| void | **setPosition** (sf::Vector2f position) |
|  | Установка позиции NPC. |
|  | |
| sf::IntRect | **getTextureRect** () |
|  | Прямоугольный участок текстуры. |
|  | |
| void | **setTextureRect** (sf::IntRect textureRect) |
|  | Установка прямоугольного участка текстуры. |
|  | |
| int | **reduceHealth** (int damage) |
|  | Уменьшение здоровья NPC. |
|  | |
| void | **increasePathIndex** () |
|  | Индексное продвижение по участкам пути. |

|  |  |
| --- | --- |
| Открытые атрибуты | |
| bool | **dir** { true } |
|  | Горизонтальное направление движения. |

Объявления и описания членов классов находятся в файлах:

* **Enemy.h**
* Enemy.cpp

**Класс EnemyController**

Управление NPC.

#include <**EnemyController.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **EnemyController** (std::shared\_ptr< sf::RenderWindow > window, std::shared\_ptr< **GameState** > gameState, const std::vector< sf::Vector2f > pathToFollow) |
|  | Конструирование котроллера. |
|  | |
|  | **~EnemyController** ()=default |
|  | Деструктор. |
|  | |
| std::shared\_ptr< **Enemy** > | **spawnEnemy** (sf::Vector2f position, EnemyType type, int health=-1, int targetPath=0) |
|  | Создание нового NPC. |
|  | |
| void | **removeEnemy** (std::shared\_ptr< **Enemy** > targetEnemy) |
|  | Удаление NPC. |
|  | |
| void | **dealDamage** (std::shared\_ptr< **Enemy** > enemy, int damage) |
|  | Причичнение урона NPC. |
|  | |
| void | **enemyWave** (EnemyType type) |
|  | Создание нового NPC в начале пути. |
|  | |
| void | **waveOne** () |
|  | Вызов волны 1 типа. |
|  | |
| void | **waveTwo** () |
|  | Вызов волны 2 типа. |
|  | |
| void | **waveThree** () |
|  | Вызов волны 3 типа. |
|  | |
| void | **waveFour** () |
|  | Вызов волны 4 типа. |
|  | |
| void | **waveEND** () |
|  | Вызов волны особого типа для быстрого окончания игры. |
|  | |
| void | **update** () |
|  | Обновление контроллера. |
|  | |
| void | **render** () |
|  | Отображение всех NPC. |
|  | |
| bool | **saveData** (const std::string path) |
|  | Сохранение параметров этапа игры. |
|  | |
| bool | **loadData** (const std::string path) |
|  | Загрузка параметров этапа игры. |
|  | |
| void | **setDefault** () |
|  | Установка начальных параметров игры. |
|  | |
| float | **getModifier** () |
|  | Множитель усложнения игры. |
|  | |
| void | **setModifier** (float \_modifier) |
|  | Изменение множителя усложнения игры. |
|  | |
| std::shared\_ptr <std::vector <std::shared\_ptr<**Enemy**> > > | **getEnemyVec** () const |
|  | Vector живых NPC. |

Объявления и описания членов классов находятся в файлах:

* **EnemyController.h**
* EnemyController.cpp

**Класс GameBoard**

Взаимодействие с игровым полем.

#include <**GameController.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **GameBoard** (std::shared\_ptr< **GameState** > gameState, std::shared\_ptr< **TowerController** > towerController, int widthInPixels, std::shared\_ptr< sf::RenderWindow > &window) |
|  | Конструирование по параметрам. |
|  | |
|  | **~GameBoard** ()=default |
|  | Деструктор. |
|  | |
| void | **process** (sf::Event event, sf::Vector2i mousePos) |
|  | Обрабатывание события. |
|  | |
| void | **render** () |
|  | Отображение игрового поля. |
|  | |
| void | **renderLabels** () |
|  | Отображение параметров состояния игры. |
|  | |
| void | **renderRange** (int mouseX, int mouseY, int range, TowerType type) |
|  | Отображение радиуса поражения башни. |
|  | |
| void | **renderShadow** (int mouseX, int mouseY, int range) |
|  | Отображение потенциального места установки башни. |
|  | |
| void | **menu** () |
|  | Вызывает главное меню. |
|  | |
| void | **menuNewGame** () |
|  | Вызывает меню выбора уровня сложности. |
|  | |
| void | **menuConfirmation** () |
|  | Вызывает диалоговое окно подтверждения начала новой игры. |
|  | |
| void | **menuSaving** () |
|  | Вызывает диалоговое окно с предложением сохранить игру. |
|  | |
| void | **deathloop** () |
|  | Состояние окончания игры. |
|  | |
| void | **showMenu** (const bool cont) |
|  | Отображение главного меню. |
|  | |
| void | **showMenuNewGame** () |
|  | Отображение меню выбора уровня сложности. |
|  | |
| void | **showMenuAsking** (const bool SaveConf) |
|  | Отображение диалогового окна. |
|  | |
| void | **showLoading** () |
|  | Отображение экрана загрузки. |
|  | |
| void | **saveMusicSettings** () |
|  | Сохранение состояния музыкального сопровождения. |
|  | |
| bool | **loadMusicSettings** () |
|  | Загрузка состояния музыкального сопровождения. |
|  | |
| bool | **saveData** (const std::string **path**) |
|  | Сохранение параметров состояния игры. |
|  | |
| bool | **loadData** (const std::string **path**) |
|  | Загрузка параметров состояния игры. |
|  | |
| void | **updCounters** () |
|  | Обновление параметров состояния игры. |
|  | |
| void | **setDefault** () |
|  | Установка начальных параметров состояния игры. |

|  |  |
| --- | --- |
| Открытые атрибуты | |
| int | **gridStatus** [32][18] = { 0 } |
|  | Состояние клеток игрового поля. |
|  | |
| sf::RectangleShape | **helpScreen** |
|  | Экран справки. |
|  | |
| bool | **isMenu** { true } |
|  | Состояние главного меню. |
|  | |
| bool | **isMenuNewGame** { false } |
|  | Состояние меню выбора уровня сложности. |
|  | |
| bool | **isRunning** { false } |
|  | Состояние игрового процесса. |
|  | |
| bool | **isEndGame** { false } |
|  | Состояние проигрыша. |
|  | |
| bool | **isRestarted** { false } |
|  | Необходимость перезапуска. |
|  | |
| bool | **isMenuAsking** { false } |
|  | Состояние меню подтверждения. |
|  | |
| bool | **isSaveAndQuit** { false } |
|  | Необходимость сохранения и завершения. |
|  | |
| int | **difficulty** { 0 } |
|  | Сложность игры. |
|  | |
| const std::vector <sf::Vector2f> | **path** |
|  | Путь перемещения NPC. |

Объявления и описания членов классов находятся в файлах:

* **GameController.h**
* GameController.cpp

**Класс GameMenuController**

Управление игровым меню.

#include <**GameMenuController.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **GameMenuController** (std::shared\_ptr< sf::RenderWindow > windowPointer, std::shared\_ptr< **GameState** > gameState) |
|  | Конструирование контроллера по параметрам. |
|  | |
|  | **~GameMenuController** ()=default |
|  | Деструктор. |
|  | |
| void | **process** (sf::Event event, sf::Vector2i mousePos) |
|  | Обрабатывание события. |
|  | |
| void | **update** () |
|  | Обновление игрового меню. |
|  | |
| void | **render** () |
|  | Отображение игрового меню. |
|  | |
| sf::Vector2f | **getMenuPos** () |
|  | Позиция игрового меню. |

Объявления и описания членов классов находятся в файлах:

* **GameMenuController.h**
* GameMenuController.cpp

**Класс GameState**

Состояние игры.

#include <**GameState.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **GameState** (std::shared\_ptr< **Timer** > **timer**) |
|  | Конструктор. |
|  | |
| virtual | **~GameState** () |
|  | Виртуальный деструктор. |
|  | |
| int | **getHealth** () |
|  | Здоровье игрока. |
|  | |
| int | **getGold** () |
|  | Количество монет. |
|  | |
| int | **getCurrentWave** () |
|  | Номер волны NPC. |
|  | |
| void | **setCurrentWave** (int wave) |
|  | Установка номера волны NPC. |
|  | |
| bool | **getHelperState** () |
|  | Состояние экрана справки. |
|  | |
| TowerType | **getPurchaseTower** () |
|  | Тип выбранной башни для покупки. |
|  | |
| void | **setPurchaseTower** (TowerType towerType) |
|  | Установить тип выбранной башний для покупки. |
|  | |
| std::shared\_ptr<**Tower**> | **getBoardTower** () |
|  | Выбранная башня на игровом поле. |
|  | |
| void | **setBoardTower** (std::shared\_ptr< **Tower** > tower) |
|  | Установить выбранную башню на игровом поле. |
|  | |
| void | **setDefault** () |
|  | Установка начальных параметров игры. |
|  | |
| std::map< std::string, int > | **getTowerProps** (TowerType type) |
|  | Характеристика башни указанного типа. |
|  | |
| std::map< std::string, int > | **getEnemyProps** (EnemyType type) |
|  | Характеристика NPC указанного типа. |
|  | |
| std::shared\_ptr< sf::Music > | **getMusic** () |
|  | Музыкальное сопровождение. |
|  | |
| void | **startGame** (const int dif) |
|  | Запуск игры. |
|  | |
| void | **updateHealthBy** (int update) |
|  | Изменение жизней игрока. |
|  | |
| void | **updateGoldBy** (int update) |
|  | Изменение количества монет. |
|  | |
| void | **toggleHelpScreen** () |
|  | Переключить состояние экрана справки. |
|  | |
| bool | **getMusicStatus** () |
|  | Состояние музыкального сопровождения. |
|  | |
| const int | **getDifficulty** () const |
|  | Выбранный уровень сложности. |
|  | |
| bool | **saveData** (const std::string path) |
|  | Сохранение параметры игры. |
|  | |
| bool | **loadData** (const std::string path) |
|  | Загрузка параметров игры. |

|  |  |
| --- | --- |
| Открытые атрибуты | |
| bool | **dirtyBit** = true |
|  | Наличие изменения игрового состояния. |
|  | |
| std::shared\_ptr< **Timer** > | **timer** |
|  | Таймер. |
|  | |
| sf::Vector2i | **mousePos** |
|  | Позиция игрового курсора. |

Объявления и описания членов классов находятся в файлах:

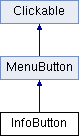
* **GameState.h**
* GameState.cpp

**Класс InfoButton**

Кнопка справки.

#include <**MenuButton.h**>

Граф наследования:InfoButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **InfoButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture, std::shared\_ptr< **GameState** > gameState) |
|  | Конструирование кнопки справки. |
|  | |
|  | **~InfoButton** ()=default |
|  | Деструктор. |

Объявления и описания членов классов находятся в файлах:

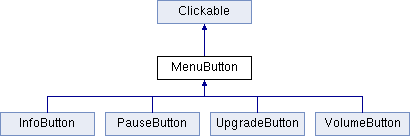
* **MenuButton.h**
* MenuButton.cpp

**Класс MenuButton**

Кнопка меню.

#include <**MenuButton.h**>

Граф наследования:MenuButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **MenuButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture) |
|  | Конструирование кнопки меню по параметрам. |
|  | |
|  | **~MenuButton** ()=default |
|  | Деструктор. |
|  | |
| void | **setTextureRect** (const sf::IntRect &rect) |
|  | Установка прямоугольного участка текстуры. |
|  | |
| void | **setTransparency** (const int transparency) |
|  | Установка степени прозрачности. |
|  | |
| void | **update** () override |
|  | Обновление положения кнопки меню. |
|  | |
| void | **render** (sf::RenderWindow &\_window) override |
|  | Отображение кнопки меню. |
|  | |
| void | **turnColor** (const bool onoff) |
|  | Смена состояния и оттенка цвета. |

Объявления и описания членов классов находятся в файлах:

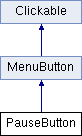
* **MenuButton.h**
* MenuButton.cpp

**Класс PauseButton**

Кнопка паузы.

#include <**MenuButton.h**>

Граф наследования:PauseButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **PauseButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture, std::shared\_ptr< **Timer** > timer) |
|  | Конструирование кнопки паузы. |
|  | |
|  | **~PauseButton** ()=default |
|  | Деструктор. |

Объявления и описания членов классов находятся в файлах:

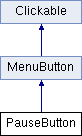
* **MenuButton.h**
* MenuButton.cpp

**Класс PauseButton**

Кнопка паузы.

#include <**MenuButton.h**>

Граф наследования:PauseButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **PauseButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture, std::shared\_ptr< **Timer** > timer) |
|  | Конструирование кнопки паузы. |
|  | |
|  | **~PauseButton** ()=default |
|  | Деструктор. |

Объявления и описания членов классов находятся в файлах:

* **MenuButton.h**
* MenuButton.cpp

**Класс Timer**

Таймер.

#include <**Timer.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **Timer** () |
|  | Умолчательный конструктор. |
|  | |
|  | **~Timer** ()=default |
|  | Деструктор. |
|  | |
| void | **start** () |
|  | Запускает таймер. |
|  | |
| void | **stop** () |
|  | Останавливает таймер. |
|  | |
| double | **elapsedMilliseconds** () |
|  | Количество миллисекунд с момента начала игры. |
|  | |
| double | **elapsedSeconds** () |
|  | Количество секунд с момента начала игры. |
|  | |
| double | **elapsedTicks** () |
|  | Количество тиков с момента начала игры. |
|  | |
| bool | **isRunning** () |
|  | Возвращает true, если таймер запущен. |
|  | |
| bool | **newTick** () |
|  | Возвращает true, если был совершен еще один тик. |

Объявления и описания членов классов находятся в файлах:

* **Timer.h**
* Timer.cpp

**Класс Tower**

Оборонительная башня.

#include <**Tower.h**>

Граф наследования:Tower:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **Tower** ()=default |
|  | Умолчательный коснтруктор. |
|  | |
|  | **Tower** (const sf::Vector2f position, std::shared\_ptr< sf::Texture > texture, TowerType type, int damage, float range, float cooldown, int upgradeCost) |
|  | Конструирование башни по параметрам. |
|  | |
|  | **~Tower** ()=default |
|  | Деструктор. |
|  | |
|  | **Tower** (const **Tower** &tower)=default |
|  | Конструктор копирования. |
|  | |
| **Tower** & | **operator=** (const **Tower** &tower)=default |
|  | Присваивание. |
|  | |
| sf::Sprite | **getTowerSprite** () const |
|  | Установленный спрайт. |
|  | |
| sf::Vector2f | **getPosition** () const |
|  | Позиция башни на игровом поле. |
|  | |
| sf::Vector2f | **getCenterPosition** () const |
|  | Позиция центра башни на игровом поле. |
|  | |
| float | **getDamage** () const |
|  | Наносимый урон. |
|  | |
| float | **getRange** () const |
|  | Радиус поражения. |
|  | |
| TowerType | **getTowerType** () const |
|  | Тип башни. |
|  | |
| int | **getCooldown** () |
|  | Время перезарядки. |
|  | |
| int | **getUpgradeCost** () |
|  | Стоимость улучшения. |
|  | |
| void | **decreaseCooldown** () |
|  | Уменьшение оставшегося времени перезарядки. |
|  | |
| void | **resetCooldown** () |
|  | Восстановить время перезарядки. |
|  | |
| void | **update** () override |
|  | Обновление. |
|  | |
| void | **render** (sf::RenderWindow &\_window) override |
|  | Отображение башни. |
|  | |
| void | **onClick** () override |
|  | Нажатие на башню на игровом поле. |
|  | |
| void | **upgrade** () |
|  | Улучшение башни. |
|  | |
| float | **enemyDistance** (sf::Vector2f enemy) |
|  | Расстояние до NPC. |
|  | |
| bool | **enemyCollides** (sf::Vector2f enemy) |
|  | Возвращает true, если NPC - внутри зоны поражения башни. |
|  | |
| bool | **saveData** (const std::string path) |
|  | Сохранение параметров башни. |

Объявления и описания членов классов находятся в файлах:

* **Tower.h**
* Tower.cpp

**Класс TowerButton**

Кнопка башни в меню.

#include <**TowerButton.h**>

Граф наследования:TowerButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **TowerButton** (sf::Vector2f \_position, int \_radius, std::shared\_ptr< sf::Texture > \_texture, std::string inText, std::string inText2, TowerType \_type, std::shared\_ptr< **GameState** > \_gameState, std::shared\_ptr< sf::Texture > \_menutexture) |
|  | Конструирование кнопки башни в меню. |
|  | |
| void | **setTextureRect** (const sf::IntRect &rect) |
|  | Установка прямоугольного участка текстуры. |
|  | |
|  | **~TowerButton** ()=default |
|  | Деструктор. |
|  | |
| TowerType | **getTowerType** () |
|  | Тип башни. |
|  | |
| void | **setTransparency** (const int transparency) |
|  | Установка прозрачности. |
|  | |
| void | **update** () override |
|  | Обновление положения кнопки башни в меню. |
|  | |
| void | **render** (sf::RenderWindow &\_window) override |
|  | Отображение кнопки башни в меню. |
|  | |
| void | **renderText** (sf::RenderWindow &\_window) |
|  | Отображение текста характеристик башни. |

Объявления и описания членов классов находятся в файлах:

* **TowerButton.h**
* TowerButton.cpp

**Класс TowerController**

Управление башнями.

#include <**TowerController.h**>

|  |  |
| --- | --- |
| Открытые члены | |
|  | **TowerController** (std::shared\_ptr< sf::RenderWindow > window, std::shared\_ptr< **GameState** > gameState) |
|  | Конструирование контроллера по параметрам. |
|  | |
|  | **~TowerController** ()=default |
|  | Деструктор |
|  | |
| std::shared\_ptr< **Tower** > | **spawnTower** (sf::Vector2f position, TowerType type, float damage=-1.0f, float range=-1.0f, float baseCooldown=-1.0f, int upgradeCost=20) |
|  | Создание башни по параметрам. |
|  | |
| void | **process** (const sf::Event &event, const sf::Vector2i &mousePos) |
|  | Обрабатывание событий. |
|  | |
| void | **render** () |
|  | Отображение башен и радиуса выбранной башни. |
|  | |
| bool | **saveData** (const std::string path) |
|  | Сохранение всех башен. |
|  | |
| bool | **loadData** (const std::string path) |
|  | Загрузка всех башен. |
|  | |
| void | **setDefault** () |
|  | Очищение списка установленных башен. |
|  | |
| std::shared\_ptr< std::vector< std::shared\_ptr<**Tower**>>> | **getTowerVec** () |
|  | Список установленных башен. |

Объявления и описания членов классов находятся в файлах:

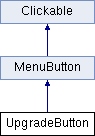
* **TowerController.h**
* TowerController.cpp

**Класс UpgradeButton**

Кнопка улучшения.

#include <**MenuButton.h**>

Граф наследования:UpgradeButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **UpgradeButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture, std::shared\_ptr< **GameState** > gameState) |
|  | Конструирование кнопки улучшения. |
|  | |
|  | **~UpgradeButton** ()=default |
|  | Деструктор. |
|  | |
| bool | **getCanUpgrade** () |
|  | Возможность улучшения. |
|  | |
| void | **setCanUpgrade** (bool status) |
|  | Установка возможности улучшения. |

Объявления и описания членов классов находятся в файлах:

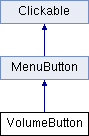
* **MenuButton.h**
* MenuButton.cpp

**Класс VolumeButton**

Кнопка музыкального сопровождения.

#include <**MenuButton.h**>

Граф наследования:VolumeButton:

* 

|  |  |
| --- | --- |
| Открытые члены | |
|  | **VolumeButton** (sf::Vector2f \_position, int \_sizex, int \_sizey, std::shared\_ptr< sf::Texture > \_texture, std::shared\_ptr< sf::Music > music) |
|  | Конструирование кнопки музыкального сопровождения. |
|  | |
|  | **~VolumeButton** ()=default |
|  | Деструктор. |

Объявления и описания членов классов находятся в файлах:

* **MenuButton.h**
* MenuButton.cpp