Tower Defence 2

Generated by Doxygen 1.8.11

Contents

Chapter 1

Tower Defence 2

Introduction

This is project is part of C++ programming course at Aalto University.

Goal of the project: implement a 2D tower defense game.

"Tower defense (or informally TD) is a subgenre of strategy video game where the goal is to defend a player's territories or possessions by obstructing enemy attackers, usually achieved by placing defensive structures on or along their path of attack." (<a href="https://en.wikipedia.org/wiki/Tower_defense">Wikipedia)

In a tower defense game, the enemies move in waves from some position of the map to another. The goal of the player is to place towers on their path in order to block, impede, attack or destroy the enemies before they are able to reach their goal. The primary object is the survival of the base.

Minimum Requirements

- Basic graphics.
- A functioning tower defense game.
- At least three different types of towers.
- At least three different types of enemies.
- Non-hardcoded maps, i.e. they have to be read from some file in some format.
- The player has to be able to build, repair (if the gameplay supposes that the towers can be damaged) and upgrade towers during the game, either between waves of enemies or without restrictions.
- Controlling the game with mouse.
- Simple user interface that shows information such as resources, number of waves/enemies etc.
- A list of high scores (decide yourself how to calculate points). High scores must be saved between game sessions!
- Fun and overall pleasant gaming experience.

Extra Requirements

- Sounds
- Multiple paths of the enemies
- Dynamic paths that can be altered with the placement of towers
- Special attack types, e.g. ranged
- Special abilities of the towers, e.g. slowing down the enemies
- Special movements of the enemies, e.g. jumping over the towers
- Different attack and defense types for both the towers and the enemies. For example, some attack types may be more effective against some defense types.
- Multiplayer mode (local or networked)
- Your own ideas for increasing the maximum fun!

Project Structure

This is an initial git repository for your programming project. It contains an initial directory structure that we wish you could use as follows:

- `plan/` – Here is your project plan. You may use different sources, but we hope to get the pan **also as pdf file**.
- `doc/` – here are the documentation sources related to your project. You may use different tools for documentation (e.g. latex), but we hope to get the final version of the document **also as pdf file**.
- `src/` – here are the C++ source files. You can have subfolders as needed.
- `index.md` – This file that you should modify to represent your project.

You may also add other new directories as needed, for example for testing tools.

Installation and Developement

`CLion` is powerful C and C++ editor from JetBrains recommended for this project. As a student you can apply for their `student program` to get access to pro version of their products.

`CMake` is used to control the compilation process.

```
1 cmake CMakeLists.txt
```

Documentation is generated using `Doxygen`. Doxygen creates static `HTML` and `latex` files using supplied input markdown files and docstring found inside the source code.

Install Doxygen using

```
1 sudo apt-get install doxygen.
```

Documentation is created into `doc` directory. Create `HTML` and `latex` files using

```
1 doxygen Doxyfile.ini
```

Creating pdf requires installing LaTeX. Create pdf using

```
1 cd doc/latex &&
2 pdflatex refman.tex &&
3 cd ../..
```

The graphics of this game depend on **Simple and Fast Multimedia Library (SFML)**. Install this library using

```
1 sudo apt-get install libsfml-dev
```

Authors

- Jaan Tollander de Balsch
- Roni Hytönen
- Tero Hyytiäinen
- Martti Hallipelto

Chapter 2

Screenshots of the Game

Main Menu

Map 1

Map 2

Chapter 3

Flowchart of the design

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

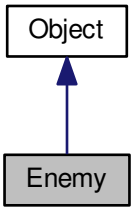
Class Documentation

6.1 Enemy Class Reference

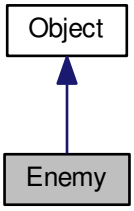
Enemy class.

`#include <enemy.h>`

Inheritance diagram for Enemy:



Collaboration diagram for Enemy:



Public Member Functions

- **Enemy** (double x, double y, double radius, double speed, int health, EnemyType *enemy_type)
- std::string **name** () const
- int **score** () const
- int **money** () const

Additional Inherited Members

6.1.1 Detailed Description

Enemy class.

The documentation for this class was generated from the following files:

- src/objects/enemy.h
- src/objects/enemy.cpp

6.2 EnemyType Class Reference

Description for creating new enemies.

`#include <enemy.h>`

Public Member Functions

- **EnemyType** (const std::string &name, int score, int money, double speed, int health)
- int **score** () const
- int **money** () const
- std::string **name** () const
- Enemy * create_enemy (double x, double y)

  Create new enemy of this type.

6.2.1 Detailed Description

Description for creating new enemies.

The documentation for this class was generated from the following files:

- src/objects/enemy.h
- src/objects/enemy.cpp

6.3 GameEngine Class Reference

`#include <engine.h>`

Public Member Functions

- **GameEngine** (double time, double timestep, int score, GameLevel &game_level, GameMap &game_map)
- double **time** () const
- const double **timestep** () const
- int **score** () const
- int **money** () const
- int **lives** () const
- void change_game_speed (GameSpeed new_speed)

  Change the game speed to fast or normal.
- void increment_time ()

  Increments the game time by one timestep.
- void add_score (int amount)

  Change player's score. Game score can't go below zero.
- bool add_money (int amount)
- void reduce_life ()
- void upgrade_tower (int x, int y, int index)

  Upgrade existing tower into new one.
- void change_targeting (TargetingPolicy new_policy)

  Change targeting policy of all towers.
- void add_enemy (Enemy *enemy)

  Add new enemy to game map.
- void advance_game_level ()
- void enemy_movement ()
- Enemy * find_targets (Tower *tower, Enemies &enemies)
- void towers_attack (graphicsEngine &gE)
- GameState update (graphicsEngine &gE)
- GameMap game_map ()

  Getter for game map.

Friends

- bool update_high_score (const std::string &filename, GameEngine engine)

6.3.1 Detailed Description

Game class contains the game map, enemies, towers and game stats. This class is also responsible for the implementation of the main game loop which modifies the properties of the objects by using the rules of the game logic.

6.3.2 Member Function Documentation

6.3.2.1 bool GameEngine::add_money (int *amount*)

Change player's money.

Returns

false and doesn't change the value if the amount of change would reduce player's money below zero otherwise true and changes the money based on the amount given.

6.3.2.2 void GameEngine::advance_game_level ( )

Handles level specific tasks such as spawning new enemies according to the game level description.

6.3.2.3 void GameEngine::enemy_movement ( )

Move enemies along the path.

- If enemy reaches the end of the path
  - Remove enemy from the game
  - Reduce players score
  - Reduce one life from player

6.3.2.4 Enemy * GameEngine::find_targets (Tower * *tower,* Enemies & *enemies*)

Other combat objects that this object can target.

- withing the attack range
- targeting policy

6.3.2.5 void GameEngine::reduce_life ( )

Reduce one life from the player. Player dies if lives reach zero.

Returns

true if player dies else false.

6.3.2.6 void GameEngine::towers_attack (graphicsEngine & *gE*)

Towers attack enemies. Increase score and money if enemies die and remove dead enemies from the game.

6.3.2.7 GameState GameEngine::update (graphicsEngine & *gE*)

Updates the game loop. In practice this method will be called by the main graphics loop.

6.3.3 Friends And Related Function Documentation

6.3.3.1 bool update_high_score (const std::string & *filename,* GameEngine *engine*) [friend]

Updates high score if needed, takes the name of the file where high scores are kept Returns true if high scores were changed High scores are visible on the menu screen

The documentation for this class was generated from the following files:

- src/engine.h
- src/engine.cpp

6.4 GameLevel Class Reference

`#include <level.h>`

Public Member Functions

- **GameLevel** (int initial_money, int initial_lives, int enemies_spawned, EnemySpawnInterval &enemy_↩ spawn_interval)
- int initial_money () const
- int initial_lives () const
- int enemies_spawned () const

  Count of how many enemies have spawned.
- std::vector< EnemyType * > spawn_enemies (double time)
- bool done ()

  True if all enemies have spawned else false.

6.4.1 Detailed Description

Contains information to start and run different game levels. What types of enemies will spawn, when and where.

6.4.2 Member Function Documentation

6.4.2.1 int GameLevel::initial_lives ( ) const

Initial money

Returns

6.4.2.2 int GameLevel::initial_money ( ) const

Initial money

Returns

6.4.2.3 std::vector< EnemyType * > GameLevel::spawn_enemies (double *time*)

Spawn new enemies according to game time

Parameters

time	

Returns

Collection of enemy type pointers to use for spawning new enemies.

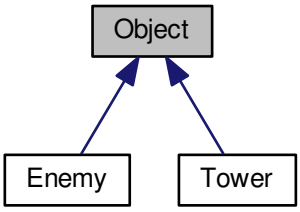
The documentation for this class was generated from the following files:

- src/level.h
- src/level.cpp

6.5 Object Class Reference

`#include <object.h>`

Inheritance diagram for Object:



Public Member Functions

- **Object** (double x, double y, double radius, double speed, int health, int damage, double attack_range, double attack_speed)
- double **x** () const
- double **y** () const
- double **radius** () const
- double **speed** () const
- double **distance_travelled** () const
- int **health** () const
- int **max_health** () const
- int **damage** () const
- double **attack_speed** () const
- double **attack_range** () const
- TargetingPolicy **targeting_policy** () const
- double **time_since_last_attack** () const
- void **position** (double x, double y)

- void speed (double amount)

  Change the speed of the object between zero and maximum speed.
- bool is_dead ()

  Object is regarded dead if it has health below of equal to zero.
- void health (int amount)
- void damage (int new_dmg)

  Change damage.
- void attack_speed (int new_speed)

  Change attack speed.
- void change_policy (TargetingPolicy new_policy)

  Change target policy.
- void distace_travelled (double d)

  Change distace travelled.
- double distance (Object &other)

  Distance from other object.
- bool attack (Object &other, double timestep)

  Attack to another object. Takes account the attack speed of the object.

Protected Attributes

- double m_x

  x coordinate of the object
- double m_y

  u coordinate of the object
- const double m_radius

  Physical radius of the object.
- double m_speed

  Current speed of the object.
- const double m_max_speed

  Maximum speed of the object.
- double m_distance_travelled

  Distance travelled by object.
- int m_health

  Current health of the object.
- const int m_max_health

  Maximum health of the object.
- int m_damage

  Amount of damage each hit deals.
- double m_attack_speed

  Attack speed, how fast does the object deal damage.
- double m_attack_range

  Attack range, how far can the object deal damage.
- TargetingPolicy m_targeting_policy

  Targeting policy, how will the object choose its target.
- double m_time_since_last_attack

  Time since last time tower attacked. Used for attack speed.

6.5.1 Detailed Description

Base class for objects in tower defence. Contains position, speed and the movement of the object. Both tower and enemies inherit from this class.

6.5.2 Member Function Documentation

6.5.2.1 void Object::health (int *amount*)

Negative damages, positive heals, if reached zero, enemy dies, cannot go above max health

The documentation for this class was generated from the following files:

- src/objects/object.h
- src/objects/object.cpp

6.6 Point Class Reference

2-Dimensional point.

`#include <geom2D.h>`

Public Member Functions

- **Point** (double xvalue, double yvalue)

Public Attributes

- double **x**
- double **y**

6.6.1 Detailed Description

2-Dimensional point.

The documentation for this class was generated from the following files:

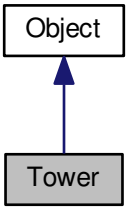
- src/geom2D.h
- src/geom2D.cpp

6.7 Tower Class Reference

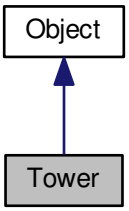
Tower class.

`#include <tower.h>`

Inheritance diagram for Tower:



Collaboration diagram for Tower:



Public Member Functions

- **Tower** (double x, double y, double radius, int damage, double attack_range, double attack_speed, TowerType *tower_type)
- TowerType * **tower_type** ()
- Tower * **upgrade** (int index)

Additional Inherited Members

6.7.1 Detailed Description

Tower class.

The documentation for this class was generated from the following files:

- src/objects/tower.h
- src/objects/tower.cpp

6.8 TowerType Class Reference

Description for creating new towers.

`#include <tower.h>`

Public Member Functions

- **TowerType** (const std::string &name, int cost, int damage, double attack_range, double attack_speed)
- ~TowerType ()

  Destroy the object.
- std::string **name** () const
- const int **cost** () const
- const int **damage** () const
- std::vector< TowerType * > **upgrade_options** () const
- const double **attack_range** () const
- const double **attack_speed** () const
- void add_upgrade_option (TowerType *tower_type)

  Add new upgrade option.
- Tower * create_tower (double x, double y)

  Create new tower of this type.

6.8.1 Detailed Description

Description for creating new towers.

The documentation for this class was generated from the following files:

- src/objects/tower.h
- src/objects/tower.cpp