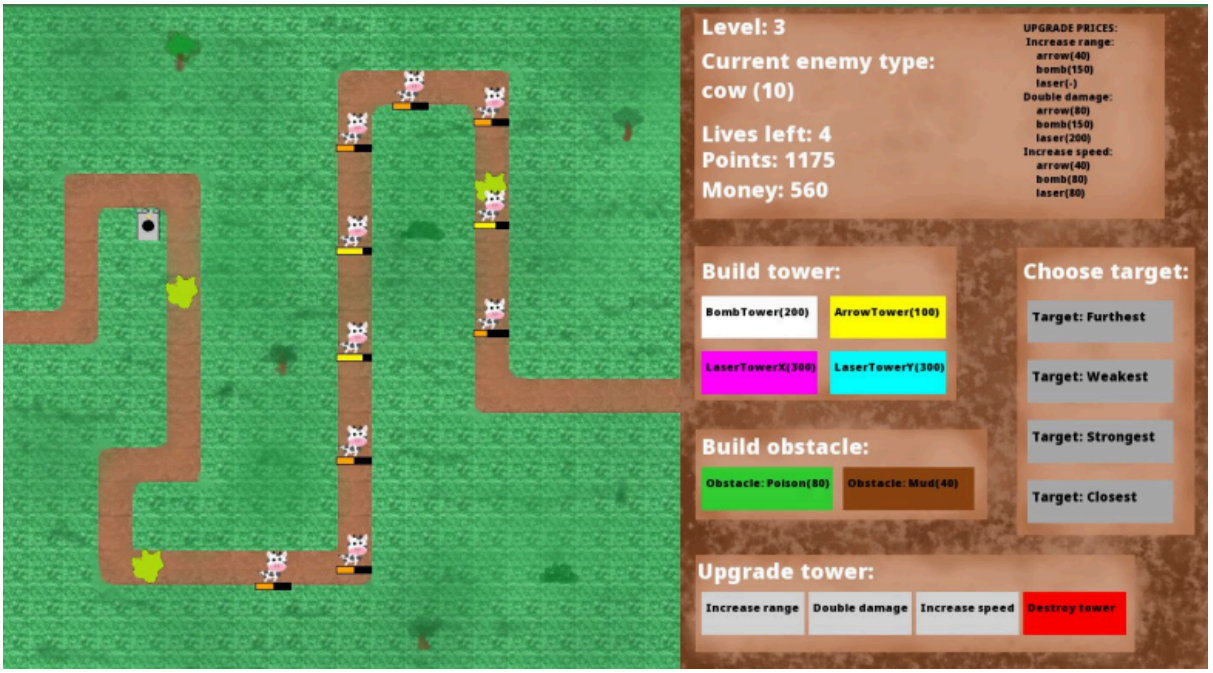


Farm Defence documentation

11.12.2023

Venla Lindgrén, Josetta Rautapää,
Henry Kivijärvi, Aleksi Paunonen

Contents

Overview

Our Farm Defence game is a version of Tower Defence with chickens, pigs, cows, wolves and horses as enemies. The game has five levels and each level has ten enemies. One has to kill the enemies in order to pass the level. The player gets five lives in the beginning, and each enemy that survives through the path, decreases the lives by one. Winning means that all the five levels have been passed successfully. The game is lost when there are no more lives left.

During each level the player can build towers and obstacles that kill or cause damage to the enemies. Towers and obstacles can be bought with money that is gained when the enemies die. One can also get money by selling the already built towers. The towers (their range, damaging effect and speed) can also be upgraded with money. The towers have cooldowns that restrict the frequency in which they shoot.

There is also a point system for evaluating how well the player has played, with a maximum of 5750 points. Points are gained when an enemy is killed.

There are some features in Farm Defence that are different from the classic Tower Defence game. In our Farm Defence version, all towers are unlocked immediately and they can shoot immediately after they have been built. We also have the same menu visible during the whole game, and no specific menu for different towers. The towers can be built to all free land cells, but there are no other restrictions. In addition, we don’t have diagonal movement for the enemies.

Software structure

Classes

Tower: base class for all towers

ArrowTower: inherits Tower, uses Enemy

BombTower: inherits Tower

XLaserTower: inherits Tower

YLaserTower: inherits Tower

Cell: knows whether it has a tower/obstacle/enemy object

Grid: consists of cell objects, handles enemy movement, towers' shooting, obstacle effects, etc.. Uses Enemy, Cell, ArrowTower, BombTower, XLaserTower, YLaserTower, Obstacle and Projectile.

Game: uses Cell, Grid, Graphics and Buttons

Graphics: uses Enemy, Tower, Obstacle, Grid, Button

Enemy: knows if it has an ArrowTower targeting itself

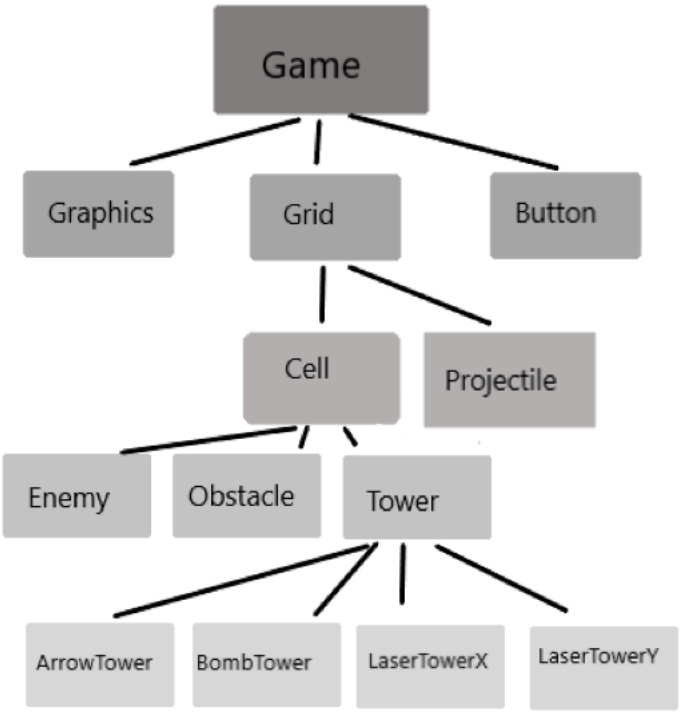
Button

Projectile

Obstacle

Game is is built with Grid, Graphics and Buttons. Grid is built with 40 cell objects and it might also have projectile objects. Cells have enemy, obstacle and tower objects. Towers are ArrowTowers, BombTowers, XLaserTowers or YLaserTowers.

Class hierarchy



External libraries

- SFML (https://github.com/SFML/SFML) for GUI etc.
- googletest (https://github.com/google/googletest) for testing.

Sounds effects

arrow: https://soundbible.com/1891-Flyby.html Conor
bomb: https://soundbible.com/1467-Grenade-Explosion.html Mike Koenig
laser: https://soundbible.com/1770-Ray-Gun.html Mike Koenig, (sped up)
gameOver: https://soundbible.com/1830-Sad-Trombone.html Joe Lamb
gamePassed: https://soundbible.com/1823-Winning-Triumphal-Fanfare.html John Stracke

Instructions for building and using the software

Compiling and building the program

Download cmake (version 3.22) with “sudo-apt install cmake” in linux terminal (ubuntu), if you don’t have it yet.

Using Ubuntu/Linux, install build dependencies required by sfml, with command
“sudo apt-get install -y libfreetype6-dev libx11-dev libxrandr-dev libudev-dev libgl1-mesa-dev libflac-dev libogg-dev libvorbis-dev libvorbisenc2 libvorbisfile3 libopenal-dev libpthread-stubs0-dev”

Create a new folder on your computer, where you want to store the game. Download the necessary folders and files from the repository, and put them to the new folder. The necessary folders and files are
- ‘src’
- ‘resources’
- ‘tests’ and
- ‘CMakeLists.txt’.

Using the terminal, go to the new folder and write commands:
“git init” (to initialize a new git repository)
“git submodule add -b 2.5.x https://github.com/SFML/SFML.git libs/sfml”
“git submodule add https://github.com/google/googletest.git libs/googletest”
“git submodule update --init --recursive --remote”

Using the terminal, go to libs/sfml and write commands: (to build sfml)
“cmake .”
“make”

In the terminal, go back to the new folder you created and write commands:
“cmake -B bin -S .” (to create a bin folder)
“cmake --build bin” (to finally build the program)

Now you should have built the game successfully. To run the game, write command:
“./bin/FarmDefence”

(You can move between the folders in the terminal using “cd.. “ to go back or “cd foldername” to go to the next folder)

Running the tests

Afer compiling and building the game with the previous instructions you can run the tests with the following instructions:

Using the terminal, go to libs/googletest and write commands: (to build googletest)
“cmake .”
“make”

In the terminal, go back to the new folder you created and write "./bin/FarmDefence_tests"

Basic user guide

Game:

You have 5 lives, a life is lost if an enemy gets through the path. You gain money and points by killing enemies and you can only build or upgrade if you have enough money. The prices can be seen in the menu. The prices are in the buttons in parentheses, and the upgrade prices are in the top right corner of the menu. Win the game by passing all five levels without losing all lives. You can play the game by building/removing/upgrading towers and by building obstacles. Try to get as many enemies destroyed as possible, before they reach the end of the path. Position the towers or obstacles in a way that is effective.

Enemies:

Enemies get harder to kill each level. But you also gain more points and more money from them.
-chicken: HP 60, moneyValue 30, pointValue 50, speed 2.5
-pig: HP 100, moneyValue 50, pointValue 75, speed 2.5
-cow: HP 160, moneyValue 75, pointValue 100, speed 3
-wolf: HP 250, moneyValue 100, pointValue 150, speed 3
-horse: HP 350, moneyValue 125, pointValue 200, speed 3

Towers:

Towers can be built on land cells, and obstacles can be built on path cells. You can only build one tower/obstacle per cell. Towers can be removed by destroying them, and then you gain money; 50% of the price you bought them with. Obstacles stay on the path and cannot be removed. Towers shoot one time (and shoot enemies inside their range, depending on the

type), and then have a cooldown. The frequency in which they shoot depends on the tower, and can be upgraded once. Obstacles cause damage (either loss of hit points or slowing down) for every enemy that goes through the obstacle (= visits the same cell).

The statistics for the towers and obstacles are:
-ArrowTower: causes 30 damage to one enemy, speed 3, price 100, range 3 (circle)
-BombTower: causes 40 damage to all enemies inside range, speed 5, price 200, range 2 (5x5 square)
-LaserTowerX: causes 100 dmg to closest enemy in the same row, speed 4, price 200
-LaserTowerY: causes 100 dmg to closest enemy in the same column, speed 4, price 200
(lower speed variable actually indicates shorter cooldown -> shooting more often)
-Obstacle (poison): causes 15 damage to all enemies passing through, price 80
-Obstacle (mud): speed -1 (while in the cell) for all enemies passing through, price 40

Upgrades:

Towers can be upgraded only once per every upgrade. LaserTowers' range cannot be upgraded. Speed upgrade decreases the speed variable of the tower by one, therefore making the tower faster = it will shoot more often. Range upgrade increases the range by one. Damage upgrade doubles the damage that the tower can cause.

Upgrade prices:
for arrowtower: speed: 40, damage: 80, range: 40
bombtower: speed: 80, damage: 150, range: 150
lasertowerX&Y: speed: 80, damage: 200

Buttons:

-If you want to build a tower, click one of the 'build tower' buttons and then click on an empty land cell in the grid.
-If you want to build an obstacle, click one of the 'build obstacle' buttons and then click on a path cell in the grid.
-If you want to change the target of an arrowtower, click one of the 'choose target' buttons, and then click on an arrowtower in the grid.
-If you want to destroy a tower, click the 'destroy tower' button and then click on a tower in the grid that you want to remove.
-If you want to upgrade a tower, click one of the 'upgrade tower' buttons and click on a tower in the grid.

Hint! If you find the game to be too hard, place as many obstacles on the path as you have money for. That is the easiest way to win the game.

Testing

We tested the different classes and functions as soon as they were implemented. Most of them were tested using the game GUI. If something didn't work as intended, we fixed it and tested again. We also built the code and ran the game after new implementations to see if there were any segmentation faults or such. We had to debug some big errors.

For example:
-We started by creating the grid, and tested if it was drawn correctly (the correct cells were path and land)
-We created one enemy and checked if it appeared correctly to the grid (the enemy was drawn to the correct cell)
-We tested if the implementation of enemy movement (in a straight line) was done correctly (the enemy figure moved in a straight line)
-We checked if tower creation worked correctly (the tower was drawn to the correct cell)
-We checked that towershoot worked correctly (enemies disappeared from correct cells)
-We changed the path to be curvy and modified enemy movement, and checked that it was moving as intended (enemy followed the path)
-We added different kinds of towers and tested their towershoot methods (enemies disappeared from correct cells)
-We added images to the GUI and checked that they looked good (correct pictures in correct cells)
-We added projectiles and checked that they moved correctly (arrow moving from tower to enemy)
-We added buttons and tested that clicking them does the correct thing
-We added obstacles and tested them.
etc.
We also created some automated tests:

Automated tests were created using Google's test framework. These tests go through the game's basic functionality, such as testing whether the game is initialized with correct values, does the coolDownTimer work as intended, and does the Tower and Enemy mechanics work correctly.

Test file was created in the "tests" folder. Test file is named "Test1.cpp". Files included in the tests are: "Game.cpp" and "Grid.cpp".
By running "./bin/FarmDefence_tests", user can make the tests run automatically. Test results are printed to the console. All tests (4) passed.