AS-0.3301 Project Documentation: Tower Defense

Joel Pitkänen <joel.pitkanen@tkk.fi> ([redacted]),
Joonas Nissinen <joonas.nissinen@tkk.fi> ([redacted]),
Ilari T. Nieminen <ilari.nieminen@tkk.fi> ([redacted])

19 December 2010

Contents

1 Instructions for compiling and use

The game currently works only on Linux. On Windows, Clanlib misbehaves in interesting ways.

Running:

You need to have ClanLib 2.2.5 (or newer) installed (the version provided for the course is ok). You also need to have Boost installed. The program comes with a Makefile, which is located in the src-directory. To compile, run make in this directory. To run the program, run the resulting "td".

1.1 How to play the game

You will first see a login screen. You can add a player with the chosen name by ticking the "create player" checkbox. You can protect your username by giving it a password. Try not to forget your username/password combination, as otherwise you will need to check it from the user database file.

After you have chosen the "New game" option, you can choose the level you want to play. You can only choose levels which you have reached.

Defend the exit from enemy hordes by building defensive towers which will smite them. You may modify the route your enemies take by building the towers in your enemies' path, but the universe prevents you from building towers so that you would be able to block your enemies' progress. The game screen is shown in Figure 1. The bar in the lower part of the screen shows the available towers, their prices. The player's health is shown as a green bar. Money and score are also shown. When the player has beaten a wave on enemies, the countdown to the next wave will begin.

You will lose if 20 enemies reach the exit. You will win if you survive all the waves. If you win, you can proceed to the next level. Currently, there are three levels.

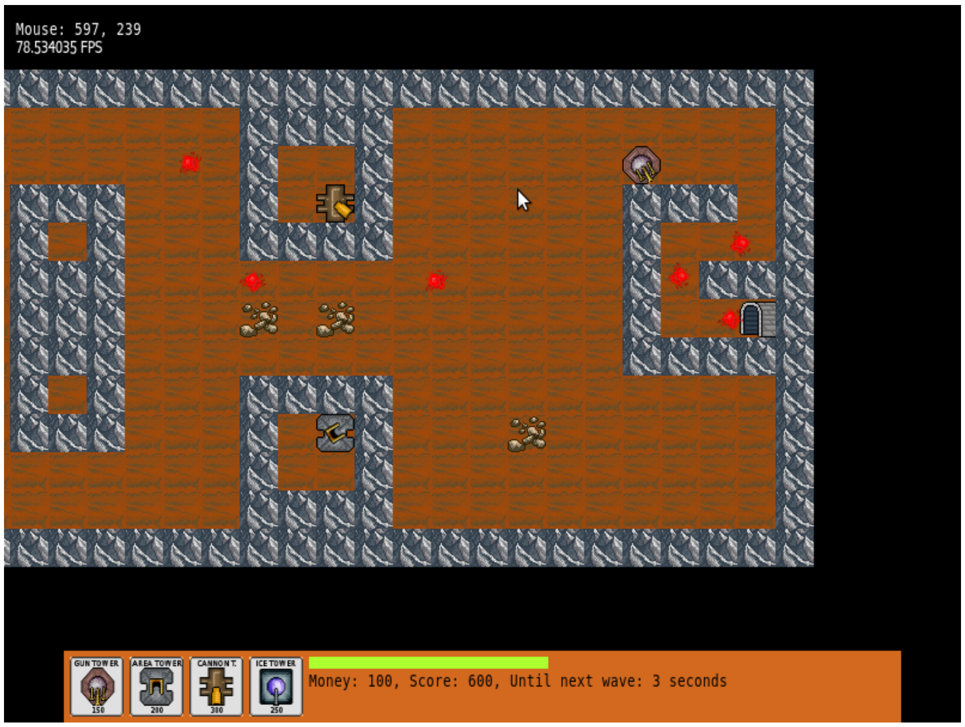
1.2 Controls

Scroll around the map using the mouse and the right mouse button. Build towers by selecting the tower you want to build from the bar in the bottom of the screen, or you can use the shortcuts (number keys). You can upgrade or sell your towers by clicking on them. By pressing Q-key, you can see the range of your towers. Pressing ESC-key brings you back to the menu. For cheaters, there is K-key which will kill every enemy in game; however, this is only enabled if DEBUG is true.

1.3 Mobs and Towers

There are basically two different types of enemies in the game. Normal ones and spawners. Normal monsters have different properties, but simply walk towards the exit. Spawners also walk towards the exit, but while doing so, they spawn more enemies in their path.

There are four towers in the game:



```
Mouse: 597, 239
78.534035 FPS
Money: 100, Score: 600, Until next wave: 3 seconds
```

Figure 1: A screenshot of the game

GunTower A basic tower, which damages a single enemy and has a high rate of fire.

CannonTower A variant of the GunTower, with a larger range and damage, but a lower rate of fire.

AreaTower Tower which causes a small amount of damage to all enemies in its range.

IceTower A variant of the AreaTower which also slows down the enemies in range. It also causes a small amount of damage over a large period of time.

2 Program architechture

The general class structure of the game is shown in 2.

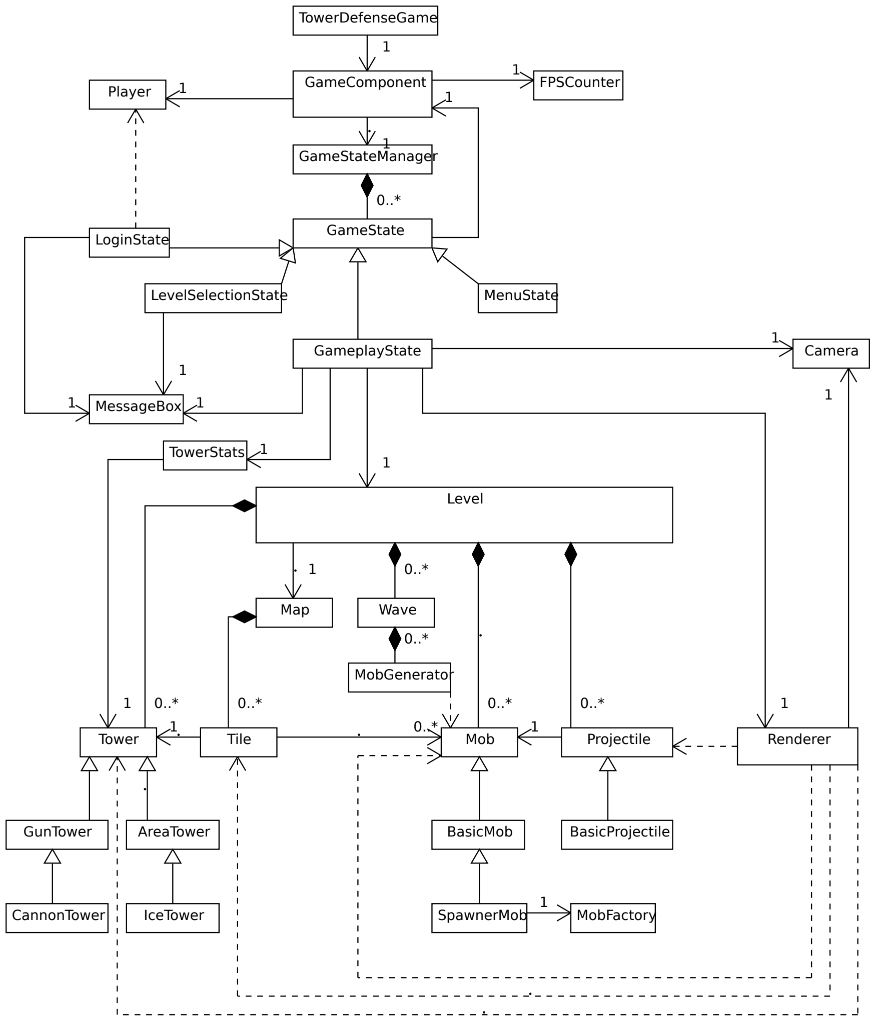


Figure 2: Class diagram for the game

The program consists of different states, which are placed in a stack and managed by the GameStateManager, which calls the update-function of the topmost state.

Primary state is the gameplay state, which is responsible for the user interface for the actual game. Drawing and updating the game state are done separately.

The actual game entities such as Mob, Tower, Projectile are tied to a particular Level, which handles most of the game itself.

The particular class division was natural, even though the dependencies between the classes were not completely shown in the plan. The basic idea has remained constant, however. The state-based window handling makes it easy to separate the different parts of the game. The actual game part is a pretty typical setup, where the game is updated a fixed number of steps every second.

3 Data structures and algorithms

Most entities, such as mobs and towers are stored in lists. This makes it easy to process all elements in a sequential manner and also makes it possible to quickly remove elements from the middle of the list.

For non-instantaneous damage, there is a damage queue, which is implemented as a priority queue.

The routes for enemies are done by calculating the distances from the exit to all the reachable tiles and keeping track of the direction one should go to when on the particular tile by going through the tiles in a breadth-first order. The directions are only updated for the tiles which are currently reachable from the exit, so an enemy that gets “stuck” will simply follow the earlier directions and pass through a tower.

Many problems were solved in a naïve way, leaving opportunities for further optimization in case it becomes necessary.

4 Known bugs

Bugs and "features"

- More of a known feature: The maps are not checked strictly beforehands and invalid information in a map (for example, wave using an entrance not present in the map) will throw an error when the map is loaded for use and will cause the game to crash.

- Projectiles don't behave like one would expect. They are currently just a visual aid to show which enemy the tower is shooting at. Sometimes the projectile will miss its target, which causes flickering in the projectile graphics.
- The database is not initialized if missing.
- Using valgrind to get rid of all problems is a bit more difficult, as even a barebones Clanlib program will have errors valgrind will catch.

What could have been done better

- A more generic way of upgrading the towers. A less manual way of inputting the changes if they affect just the numerical properties of the tower and not the functionality. The tower information could mainly reside in a configurable XML file.
- Adding new tower types to the game requires a few too many steps. Also, there should have been a Tower factory class to hide some of the messy details from the interface itself.
- Pathfinding: Diagonal movement would look better on open maps, also the enemies could actually move in a even more optimal manner (calculating straight-line paths)
- The Level class should be split into several classes. (For example, pathfinding could be extracted)
- Targeting is done in a naïve manner: all the enemies are searched through, even though getting a list of tiles which are in range would not be difficult to do. Going through these in the order of distance to the exit would in practice make targeting much faster, now it is linear to the number of monsters. With these numbers of monsters on the screen, this optimization did not become necessary.
- The wave handling is done in an extremely convoluted way; it should be cleaned up and the logic should be made clear.
- For ease of use, the login screen could have had a list of players in the database.
- The map checking could be more strict, and the maps should be checked on startup to avoid errors later in the game.

- Many changes, especially towards the end were “kludges”; something that could have been done better, but the amount of work for a quick fix was lower than a better fix. (Though this is apparently rather common in the game industry)
- MobGenerator and MobFactory serve partially the same function; this should be reflected in the relation.
- Variable and method naming is not completely consistent. Coding style questions should have been written down in the beginning.
- Sounds could have been implemented.
- The “kill” command doesn’t actually necessary kill all enemies, it just causes a lot of damage.

5 Tasks sharing and schedule

The communication over IRC worked rather well, as expected. Several group coding sessions were also organized.

Joonas put countless hours into the project, produced map handling, player progress accounting and the graphics. He also contributed into various parts of the user interface and the game logic. In addition, Joonas tuned the game balance.

Joel used estimated 70 hours, focusing mainly on the getting the GUI into order, but contributed also into various parts of the other code. He also made the class diagrams for the plan and documentation.

Ilari had a problem with his work scheduling, contributing later in the work, putting around 60 hours of more or less efficient time into game logic programming, bug hunting and writing documentation.

The schedule went wrong due to external time constratints. The original schedule as such left little time for failures and problems in the process. Each group member was ill at some point during this period, which caused an additional delay. Also, the dependencies between the different parts of the program were not completely clear at the time of planning and so to avoid additional time use in the integration phase, implementation of some parts of the program were postponed to a later date. Additional features were not implemented due to lack of time.

The failure to adhere to the original planned schedule is visualized in Figure 3.

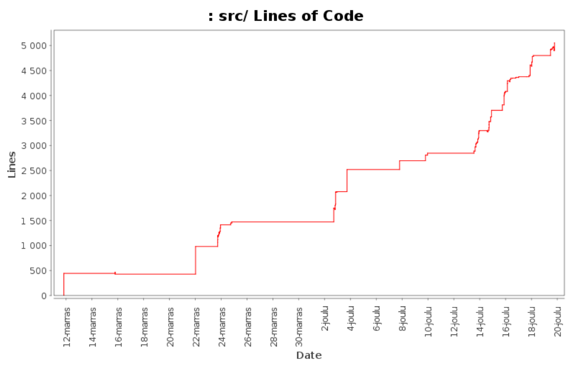


Figure 3: The failure to stay within the schedule: lines of code in the src/ directory over time.

6 Differences to the original plan

The schedule in the original plan ended up being somewhat a work of fiction. This was mostly due to the fact that other projects took up most of the available time.

If the group had had somebody who had designed and/or programmed a game in detail beforehand, a more detailed plan could have worked better. Many details were left ambiguous, decisions to be made during the implementation.

The core features are present:

- Mouse input with keyboard shortcuts
- Multiple enemy types with different properties (hit points, speed)
- Multiple upgradeable tower types: direct, splash and special damage
- Easy-to-read, easy-to-edit format for maps
- Towers can be placed at any time
- Dynamic paths for enemies
- Multiple terrain types
- Player profiles, player progress

However, as we ran out of time, practically no additional features were implemented, even though a few of them would have been easy. Instead, some time was taken to polish the details and to do additional testing. Multiple entry points and scrollable maps were implemented. Automatic testing was not included in the build process; in retrospect several bugs could have been avoided by having a proper testing procedure. Some parts of the code were implemented using tests implemented with the Boost test library, but no large scale testing was done.

7 References

- C++ documentation
- Clanlib documentation