ft_shmup

Summary: You will create a small game using the ncurses library.

Version: 1.1

Contents

Chapter I

Preamble

Here's what Wikipedia has to say about Battlestar Galactica:

Battlestar Galactica is an American science fiction media franchise created by Glen A. Larson. The franchise began with the original television series in 1978, and was followed by a short-run sequel series (Galactica 1980), a line of book adaptations, original novels, comic books, a board game, and video games. A re-imagined version of Battlestar Galactica aired as a two-part, three-hour miniseries developed by Ronald D. Moore and David Eick in 2003. That miniseries led to a weekly television series, which aired until 2009. A prequel series, Caprica, aired in 2010.

All Battlestar Galactica productions share the premise that, in a distant part of the universe, a human civilization has extended to a group of planets known as the Twelve Colonies, to which they have migrated from their ancestral homeworld of Kobol. The Twelve Colonies have been engaged in a lengthy war with a cybernetic race known as the Cylons, whose goal is the extermination of the human species. The Cylons offer peace to the humans, which proves to be a ruse. With the aid of a human named Baltar, the Cylons carry out a massive nuclear attack on the Twelve Colonies and on the Colonial Fleet of starships that protect them. These attacks devastate the Colonial Fleet, lay waste to the Colonies, and virtually destroy all but a small remaining population. Scattered survivors flee into outer space aboard a ragtag array of spaceworthy ships. Of the entire Colonial battle fleet, only the Battlestar Galactica, a gigantic battleship and spacecraft carrier, appears to have survived the Cylon attack. Under the leadership of Commander Adama, the Galactica and the pilots of "Viper fighters" lead a fugitive fleet of survivors in search of the fabled thirteenth colony known as Earth.

There is very little chance of you achieving a good grade on this project if you have not watched Battlestar Galactica in its entirety.

Chapter II

Introduction

The goal of this rush is to implement a simplistic shoot-'em-up-style game in your terminal.

For those of you who don't know what that kind of game is (shame on you, by the way), have a look at Gradius, R-Type, etc.

You will use a 'screen' made up of a grid of 'squares', which you can equate to the characters on your terminal, so that the entities of your game are each represented by a character on the screen.

Chapter III

General rules

- Your assignment must be written in C or C++.

It is strongly recommended to choose C++ to complete this rush. You can use any version of C/C++ you prefer.

- No specific coding norm is required.
- Use `cc or c++` as your compiler.
- You must compile with the following flags: `-Wall -Wextra -Werror`.
- You must include a `Makefile` to compile your source files.
- Your program must not crash unexpectedly (e.g., segmentation fault, bus error, double free, etc.) except in cases of undefined behavior.
- Within the mandatory part, you are allowed to use the following functions:
  - All functions from the standard library.
  - Any clock-based library.
  - The ncurses library.

Chapter IV

Instructions

Here are the basic requirements:

- The program name is `ft_shmup`.
- The game must be single-player.
- The display must use the ncurses library.
- There must be horizontal or vertical scrolling (The screen area moves through the world, much like in R-Type, for example).
- There must be multiple random basic enemies.

`A basic enemy is one that does not use any special abilities. This enemy doesn't even need to move or aim when it shoots.`

- The player can shoot at enemies.
- Your game must have a basic collision mechanic (If an enemy touches you, you die).
- Game entities must occupy only one 'square' on the map.
- The game should have frame-based timing.
- Displaying score, time, number of lives, etc., on the screen.
- Clock-based timing (Use whichever system facility or library you like).
- Enemies can also shoot.
- Scenery (Collidable objects or simple background).

It may seem a little daunting, but the basic requirements can be fulfilled by a game even simpler and far uglier than the already simplistic **Space Invaders**. So it's not actually that hard.

`For students implementing this project in C++, there will be plenty of opportunities to use the various features of C++. For example, representing the various entities of your game (Player ship, enemies, missiles, etc.) as subclasses of a GameEntity class? Maybe even as an interface?`

For those of you who have never made a video game, here's how a very basic game should run:

- Acquire input (Player controls, network, etc.).
- Update game entities.
- Render display.
- Repeat until the game ends!

Of course, the "Update" phase includes a lot of tasks, including handling the spawning of enemies, executing actions required by player input, moving entities according to their current vector, and considering the time since the last update. But those are the basics. When you break it down, a basic video game is a really simple program. It shouldn't be too hard, should it?

When you're pretty sure you've done everything right, you are welcome to improve on the basic requirements and create an even better game.

Chapter V

Bonus Part

Now it's time to improve your beautiful game. Here are some ideas for cool bonuses, but keep in mind that we won't consider them if you haven't completed the mandatory requirements:

- Large and challenging boss enemies.

```
"Large" simply means that the enemy occupies more space on the board.
It will also have more health, similar to a boss in other games.
```

- Enemies with scripted behaviors.

```
Scripted enemies can have additional abilities, such as moving toward
the player or improving their aiming when shooting at the player.
The choice is yours!
```

- Multiplayer, either on the same keyboard or over the network, if you're feeling ambitious.
- Scripted game worlds with pre-determined waves of enemies.

```
The bonus part will only be assessed if the mandatory part is
PERFECT. "Perfect" means the mandatory part has been fully completed
and works without any malfunctions.  If you fail to meet ALL the
mandatory requirements, your bonus part will not be evaluated.
```

Chapter VI

Turn-in and Peer Evaluation

As usual, submit your work on your Git repository. Only the work included in your repository will be reviewed during the evaluation.

Good luck!