Space Invaders Report

Arno Deceuninck
2nd Bachelor Computer Sciences
University of Antwerp

January 17, 2020

1 Extensions

- Travis-CI integration: The .travis.yml file is updated to be compatible with SFML.
- Multiple levels: Once you've cleared the first level, the next level starts automatically after the "Victory" message.
- Different types of enemies: In the last two levels, there's a boss enemy, which shoots faster and has more lives than normal enemies.
- Multiple control options. (see readme)
- Doxygen documentation (by running "doxygen Doxyfile" in the project root)
- Destroy bullets of the enemy comming towards you by shooting at them.
- Only the current first enemy in the column can shoot, so they don't kill each other (because bullets can't magically go through enemies).

2 Project Structure

The game loop is in an external Game class, to be sure to seperate the contents of the MVC pattern. When you start a new game, all different levels get loaded from a levels file, and the first level is loaded using a LevelLoader. The GameModel is the entity which keeps track of which level is currently playing en what the next level will be. The WorldModel contains a shared pointer to all elements inside the game world. These are stored in a set, because the order is not important, I have to be able to find the element by key, duplicates are not allowed and the key isn't stored seperately from value. Since the observer pattern works with weak pointers, it is easy to completely remove an entity by just removing it from the set in WorldModel. The Controller and Representation have a similar list in GameController and GameRepresentation.

Each representation entity observes the corresponding model entity, so when they broadcast an event that their position is updated, the sprite in the representation gets immediately updated too, meaning it doesn't have to recheck all this variables in the window clear-draw-display phase.
The LevelLoader loads the ships from a level file and creates a controller for them. Every time a new model entity gets created, the creator sends an event. GameModel is observing this events and redirects the creation to GameRepresentation and WorldModel. The first one will create a corresponding representation and the GameWorld will add it to the set of entities in the world.
Each time a bullet moves, an event is broadcasted to the WorldModel, who directs it to all entities in the world. When they've received it, they check whether there is a collision. If there is a collision, the ship takes the damage of the rocket, and if this means the ship doesn't have any health left, it destroys itself. Self destruction sends an event to all observers, so they know the object is destroyed, and can do the corresponding actions (e.g. Remove entity from set in WorldModel, which was the last shared pointer to the object, so it gets destroyed).
The observables (subjects) contain a list with all their observers in it. In this case, I've used a list instead of a set, because there is no relative equality operator (<or >) implemented in weak pointers. An Observable can notify it's observers with an Event as parameter. There are a lot of different events, which all inherit from a base Event class. When an Observable has notified it's observers, their handleEvent function gets called, which has as parameter the event type. This event gets typically dynamic casted do determine whether it's an event the observer is interested in and to get the other details from the event (e.g. the PositionUpdated event contains the new position).
The StopWatch class is only used in the game loop. This keeps track of how many time there was between two cycles and the update functions get the elapsed time as parameter.
As json parser have I used RapidJSON. This mainly because I've already used it in other projects, so I knew how it worked and didn't need any extra time to figure this out. I didn't want to just copy the RapidJSON folder, because that would give a false representation of the number of lines I had written in GitHub, so I decided to merge the RapidJSON repository in my SpaceInvaders repo. The only problem with this is that there are now a lot of contributors accoording to GitHub who contributed to my project, but they only contributed to RapidJSON and I am the only contributor on the SpaceInvaders project. Also be sure, when running inspect code, it's only on my src folder, since RapidJSON gives a lot of warnings with inspect code.
I've decided not to use the default std::exceptions (e.g. domain_error, filesystem::filesystem_error ...) but create another derived class from std::exception. This makes it easier to find the cause of an error: SiExceptions (and it's derived classes) are thrown by myself, while other exceptions are from somewhere else (e.g. rapidjson, stl).