Space Invaders

Introduction

Welcome, to the Space Invaders project, in this docuemnt you find information about how to compile and run the application. Dependencies are SFML and off course a c++ complier+ linker.

The document describes what each class in the system represents, after that I explain my design choises.

The API is documented and can be found in the docs/html directory, open the index.html file and start reading.

Functionality

- Aliens, Guns, Walls
- Basic Space Invaders gameplay
- Levels

Compiling

```
cmake .
make
make install
```

Running

After compiling: ./SpaceInvaders

Generating The Documentation

Change the following line in the CMakeLists.txt file at the root directory from:

```
# Documentation
SET(BUILD_DOCS FALSE)
```

To:

```
# Documentation
SET(BUILD_DOCS TRUE)
```

The System

Some Names

- Controllers : work between views and entities
- Models : set's of entities
- Entities : specific objects, storing information and having a interface for querying information
- Views : objects connected to entities representing them to the user
- Factories : systems to create Entities and views
- SI(Space Invaders Element) : special container containing all the models, views and controllers which is being used through the whole system
- Libraries : classes for building all above

Libraries

Assets A library which can hold textures and fonts so those can be loaded when needed and don't waste memory with endless copies of them.

Controller The base class for a Controller.

Entity The base class for an Entity.

Factory The base class for a Factory.

ScreenEntity The base class for a ScreenEntity, an Entity which will be presented on screen and has special functions for that(location, move, collision detection, ...)

SI The Space Invaders Element

Size A class for representing a size with an width, height and point. Beter said : a box in 2D space.

Utilities File with the util namespace with some useful utilities for the whole system.

View The base class for a View.

Controllers

Each of the following controllers inherits the base controller and define some actions in the system.

Collision Controller Has only one function check() which is called each time after an update of all the elements(Aliens, Walls, Guns, ...) in the game. It will check if certain elements collide with each other and if so it will call the appropriate controller for handeling the collision.

Event Controller The event controller is called each time an event(keyboard touch) happens, it will determine what to do after the event. This means selection the right controller and function.

Game Controller Can be seen as the "main" controller, when the game is started the first controller to be called is the game controller. It has functions to build up the game(initiate all the elements), show the startscreen and the game over or game won screen. And of course it shows the screen where the game is played and gives all the events to the appropriate controllers.

Motion Controller The motion controller is the controller who moves the elements over the screen.

Sometimes these elements are moved automatic, for example the aliens they will be moved each time a certain time passes by in the Game Controller. Sometimes elements are moved by the user, for example the gun is getting moved when the event controller gets an left or right keyboard touch. The Game controller will receive the event and give it to the Event controller who decides to call the Motion controller to move the gun. Besides moving elements has the motion controller also the functions to shoot bullets. These are also automatic as manual by the user called.

Screen Controller The screen controller handles the screen(window) provided. During one cycle of the game each view will be able to draw to the window. But these views won't be showed until the screen controller's redraw method is called from somewhere. It also handles the function to close the window.

Entities

Alien(ScreenEntity) Represents the data from an alien, it looks just like an ScreenEntity from where it is inherited but has also a ticktock function which can be called by the view to show the appropriate image of the alien.

Bullet(ScreenEntity) The bullet Entity represents an bullet on the screen, it knows from where it was fired and has also a special overloaded kill function which will find his view and remove it as soon as the bullet is killed. So we don't fill up our memory with unnecessary things.

Game The only entity which is not an screen entity holds information about the game such as the height and width of the game and the current level playing.

Gun(ScreenEntity) Is almost the same as it's base Screen Entity class but needs to handle sometimes different so some functions are overloaded. It represents of course the player's gun.

Wall(ScreenEntity) Probably the most boring Entity, it is just a ScreenEntity and nothing more.

Factories

This needs no extra information, factories are created for Aliens, Bullets, Guns and Walls. They will create a Entity and place it in the model container. After that a view is created and stored in the view container. The Entity and View get linked to each other for futher information exchange and our new element is created.

Views

Each element(Aliens, Walls, Bullets, Guns) has it's own view. In this view the corresponding entity will be queried for information like the postion and width and height of the element. The appropriate texture for the element will be asked from the assets library and then the element will be drawn on the screen. If the assets library throws an exception because the image of the element can not be found, then the view will generate a geometric shape so the game can still be played.

Besides these elements views there are some other views:

Game Ended View Will be shown to the user when he won or lose the game. No images over here but fonts, when those fonts couldn't be found it will print the information to the console.

Info View Yet another view linked to an Screen Entity, this little text in the left corner of the screen during playing will show the user's score and remaining lives.

Start Screen View The first view the user get's to see when he starts the game, he can select a level and start the game.

Design

Space Invaders Element

Is being used in the whole system, some examples:

- Factories: each Factory stores it's new created views and entities automatically in the right containers, SI provides these containers.
- Gun Entity: needs to know the width of the window, so the gun doesn't go off the screen. The SI element gives it access to the game Entity which stores the game width. Also when the gun moves, the Gun Entity will update the position of the gun and immediately ask the screen controller to redraw the screen.
- Event Controller: will at each event(keyboard touch) redirect to the appropriate function in the Game Controller using the SI element.
- Game Controller: can ask simply information about the game(mostly the current level) and also get the amount of lives the gun(player) has to see if the game is over. Also this information can be provided wit the SI element.
- The Motion Controller: works on large sets stored in SI, like the aliens.

Through the system there can be found dozens of examples why SI is very useful, but some remarks over here:

Hiding

The SI element doesn't provide hiding (yet). So let's say a controller isn't allowed to update the screen, it will be possible through the SI element. Off course this can be controlled by a privileges system in SI. So that each time the SI element is called the caller should ask for privileges and the SI element can decide to give them, but that's for a further version.

Why do views not have access to the SI element?

Views just need to know something about their connected Entity, nothing more.

Why the SI element?

- It reduces the amount of parameters to be given in the constructor
- In the whole system there is only one SI element, this limits the amount of stupid errors being

made
- It makes the code readable, for example to redraw the window the only thing todo id : this->si->controller->screen->redraw();
- You have everything you need everywhere
- Adding new functionality to the system is much easier by using si

MVC

Though there are multiple interpretations of the MVC system mine work a little bit different then the most. Instead of defining the model as a class with entities in and defining functions in that class to work on these entities I just used a list with entities and defined it as model.

Why?

The default interpretation is great for the aliens, you can move them just by calling one function in the controller to the model and say move. The model will then decide how each alien should move but that's in my opinion not the function of the model, there is a controller for that. The model should store data and not work on data.

So the motion controller moves the aliens in my design, it calls on each alien Entity the move function. This move function accepts only 4 values : UP, DOWN, LEFT, RIGHT so the Entity can decide how much he has to move. This can come in handy when you want some Aliens to move faster then others. The computation is done by the controller and the only thing the entity should do is change it's position.

Observer Pattern

Also here I do it a little bit different, the observer pattern should be used to call the function to redraw the window with all the new elements on it. I haven't implemented a special observer class with notifiers.

Why?

One word SI(Space Invaders Element), this element is available through all controllers and Entities. So when some controller or entity decides the screen should be updated, it can call in the SI element the screen controller where the window redraw function is defined. Actually this looks like the observer pattern but it isn't exactly it.

Factory Pattern

In the description of the project it was said we needed to use an abstract factory pattern, I used just a factory pattern.

Why?

I think an abstract factory pattern is too complex for this project, it will require a lot of classes and make the code more complex then it should be. The factory pattern has the same function as the

abstract factory pattern but is much nicer. It has been used to create the different aliens(they have the same speed but a different name and image, raising the speed is just one line of code) and guns. Also each Gun and Alien have a Bullet Factory inside them to provide them at all time with bullets.

Inheritance and polymorphism

The whole system rests on inheritance: each controller, factory, view, entity inherits from a base library(class). Now I have to admit these libraries doesn't look so spectacular. A better example is the ScreenEntity it inherits from the Entity and the Gun, Alien, Wall and bullet entities inherit from it.

The ScreenEntity declares some special functions for entities besides the base entity class. The children of this class sometimes just use these function but it get's more interesting when they get overloaded. A few examples:

- Move: this function is overloaded in Alien Entity, when the Alien is moved it will also update the ticktock which represents if the alien view should show an image of an open or closed alien. Also the gun has a modified move function which will stop moving the gun when it reaches the borders of the screen.
- Kill: this function is overloaded in the bullet Entity, when a bullet is killed it's appropriate view will be searched so it can be removed from the memory.

Another example is the view library, this one defines a virtual draw function. Each view is overloading this function with it's own code. This makes it possible to draw a gun with rectangles and triangles provided by SFML. Th aliens will be drawn by provided images. Though all views work the same way, they are all called within the screen controller's redraw function by using the draw function from teh view.

Exception Handling

Is build into the assets class and views so when an asset(texture, font) is not available a simple representation is shown.

Extending the system

A design is good if it is easy to add functionality without too much effort. Let's have a look at some examples:

How difficult is it to add a new type of Alien

Let's go wild and we want a completely new alien that travels once from left to right on the screen:

- Create a new view which represents the Alien.
- Create a new function in the Alien factory which creates our new alien, give it a special name by using the setName function, and change the speed. Of course we use our new view.
- Change the move Aliens function in the motion controller to move this type of alien only left or

right.
- Add an AlienFactory to the Game Controller's startgame function and let it create our new alien at random moments.

That's it! Now our new alien can be killed and so add points to our gun, it can move, it can shoot bullets. Average time to build this? Maybe one hour.

How difficult is it to add a multiplayer mode?

- Add a new gun in the game build up process, this is just one line of code.
- Add to the movegun function in the motion controller an index to specify the gun, 6 lines of code.
- Add to the event controller the events for moving our second gun, 6 lines of code.
- Check in the startgame function in game controller if one of the controllers is dead instead of one, 4 lines of code.
- Add in the build up process a infoView connected to the second gun to show it's lives and score, 3 lines of code.

Total 20 lines of code, looks quite good. Some things maybe need to be changed like the explanation for the users in the startview what the controls are for the two guns(at this moment it describes only the controls for one gun), but with this 20 lines a working multiplayer option can be implemented.

How difficult is it to add a wall that moves?

- Add a move function to motion controller, 10 lines of code.
- Call this function in the gamecontroller when the aliens move, 1 line of code.

No more to do.