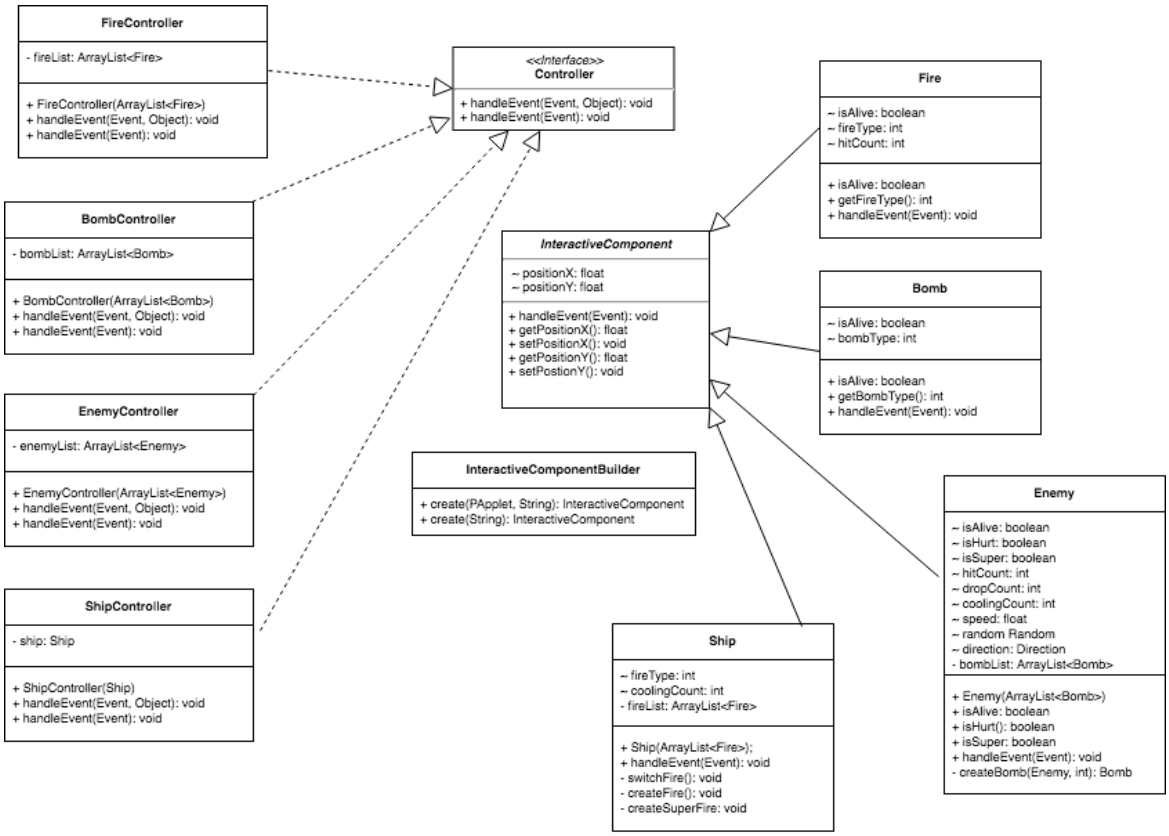
Capstone Project Documentation

UML diagram

Note: enum classes are not included in the diagram. Also, the app class in starter package is not included. What's more, since the AbstractView class is not used, respective methods are not mentioned here as well. The reason why it's not used will be explained later in the document.

This diagram is to show the implementation of interface, and extension of abstract class, as well as details of each controller and model classes.

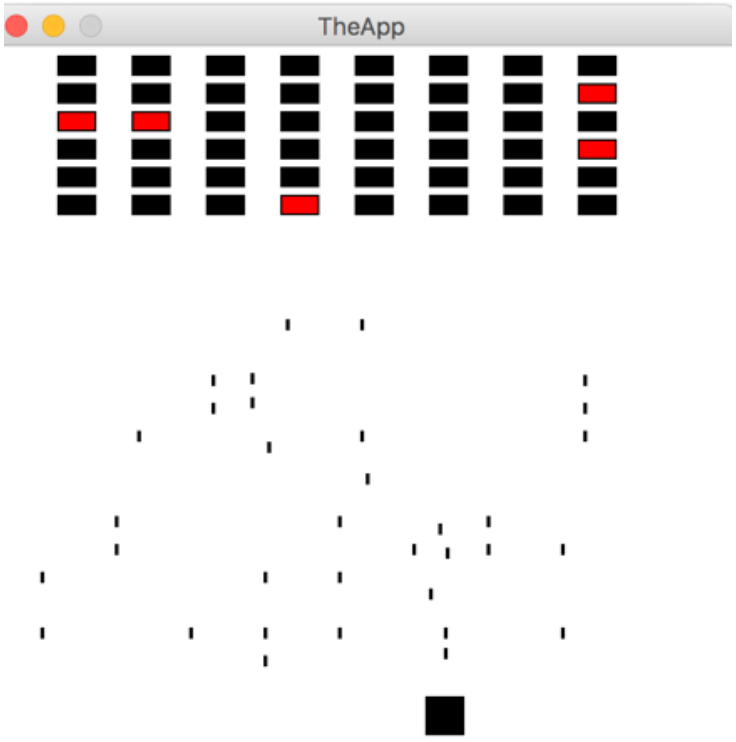


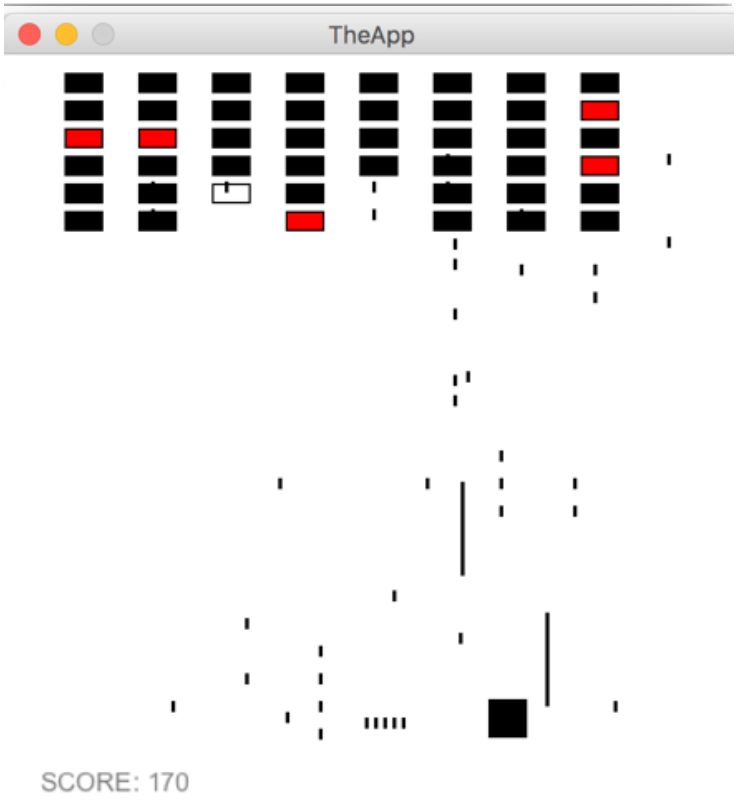
ScreenShots

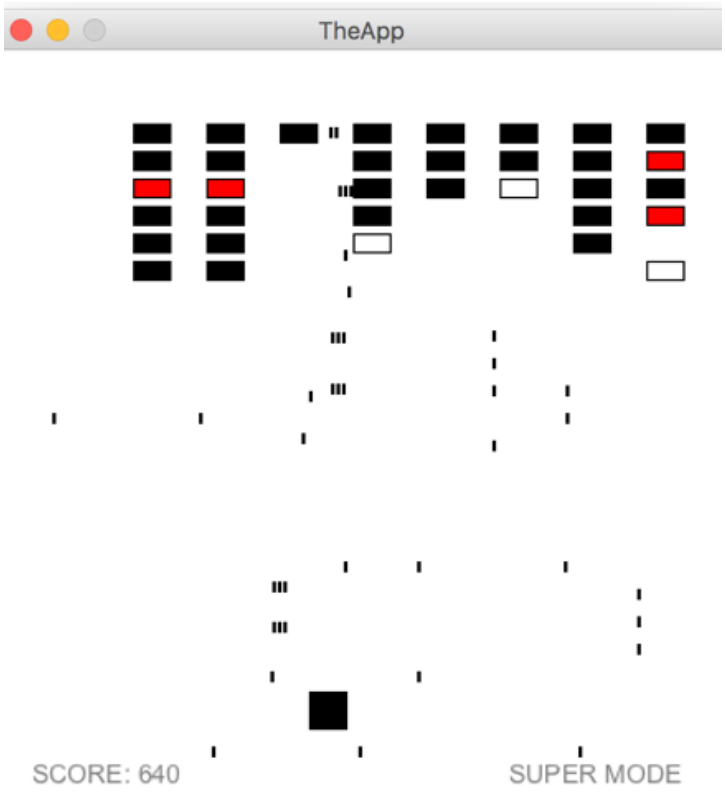
The screenshots are taken from the running game. The first one is starting phase, where the matrix of enemies are dropping bombs and the ship at the bottom hasn't yet fire back. The second one is showing the engagement of fire, some enemies were shot down, the score number revealed this. The third one is showing the super mode, where the weapon is stronger and faster for the ship to shoot more enemies.

From the screenshots it's still visible that the matrix of enemies is moving back and forth, also it reveals there're different color of enemies, and different bombs that they drop.

The size of the screen is 400 x 400.


TheApp
SCORE: 0


TheApp
SCORE: 170



Feature introduction

As basic features for space invaders, there's a matrix of enemies, in the shape of rectangle, they move from left to right, hit the edge and move back and forth, every time they hit the edge they will get a step closer to the space ship at the bottom. Once there's any enemy reached close enough to the ship, game is over.

The ship, in a shape of square, can be moved only horizontally and is able to shoot fire to the enemies. Enemies being hit and died will disappear. Player wins if all the enemies are killed and disppeared.

Enemy can drop bomb, if the bomb hit the ship, game is over.

There's different color of enemies. Normally the enemies are in black, once they got hit by fire, they turn to white, indicating they're wounded. One more hit of fire would kill them. This means normal enemy need two hits to kill. There're also super enemies, only a few, in red color. The super enemies can bear additional 5 hits of fire, this makes them much harder to kill.

No matter normal enemy or super enemy, if any one gets hit, the whole matrix of enemy would increase its moving speed to approach the ship quicker. However, there's an upper limit of the speed.

What's more, the bomb dropped by enemies are different. Enemies drop bombs randomly as they are approaching, these bombs are moving slow. Once got hit, enemies would also drop bombs as reaction, these "reacting" bombs are moving faster. Especially, for super enemies, their "reacting" bombs are 5 in a row, much more dangerous than normal enemies.

However, "reacting" bombs have their cool down time, meaning after dropping one, the enemy has to wait till the cool down time passed. This means, if the ship can kill enemies with strong fire, there'll be less bombing in return.

The ship has three kinds of weapons. The first weapon fires small size of bullet, but quite flexible, so that the ship can move and shoot. The second weapon is like laser pulse or missle, it's long, and one bullet can hit more than one enemy before it explodes. It's more powerful, but it has cool down time, so the frequency of firing is lower. The third

weapon is called super mode, once enabled there's a tag on screen indicating it. Super mode shoots 3 bullets of first weapon at once and in a doubled speed, makes things a lot easier.

User Guide

Mousemove moves the ship.
Mouseclick on left opens fire.
Mouseclick on right switches between 1st and 2nd weapons.
Press key "S" toggles the super mode.

Lab Report

I didn't have much time in the first week, so after some consideration I figured work alone would be more efficient, given the fact that time is too limited. I spent several days learning the videos and work through the excercises, played around with hands on activies.

When I finally start to clone the project git, it's 23rd. I've got the weekend as buffer.

I spend quite some time to study the processing web site, went through the turtorials and examples and found them very useful. According to the documents there, redraw() is not functioning well if it's put within draw(). And it's only used for events like mouse or keyboard, not for animation. That's my first doubt about the AbstractView class provided in the structure code.

I feel processing app (Papplet) is not really suitable for making several custom views. The draw() method in the main application is acting like a view in loop. It's automatically updating itself, based on the frame rate.

What's more, since I chose space invader, this game has matrix of enemies and there're lots of fired bullets and bombs, apparently arraylist will be used. There will be many animations, and many events would be triggered by such animations instead of mouse or keyboard. To draw then, the View need to go through the arraylist, it needs to have access to the arraylist. If redraw() is not to be put in draw(), the View.update() still need to go into draw().

So, animation in draw() triggers event, calls controller for handleEvent, then go to model component to handleEvent, then calls View.update(), and what should I put in this View.update()? I cannot put the animation part in, otherwise there's a dead loop, that update() itself triggers event to handler to call itself.

If I put only those stuff that doesn't involve animation and event triggering, then the question is, why bother? The draw() method is updating automatically, once I call model component and updated the data, the draw() method can already drew them. I don't need a redraw() in between, which only consumes system resource.

In the end I was convinced that just using the draw() method as the only "View" makes more sense, and focused on the other parts.

To come up with the classes is no brainer, I quickly worked out Enemy, Ship, Fire, Bomb as main model classes, respectively the controllers. The builder class is used to create the model components in a builder pattern.

Then I start to think about the use cases and try to achieve the animations using draw() method. I initialized required objects in setup() and start to make the game work in the "View". At this stage I was not heavily dealing with controller or model logic, but focused on showing the stuff and see how to detect the various events that I need to take actions, e.g. movement, create a bullet dynamically, create a bomb randomly, disappear enemy when it's hit and died.

This took me a long day to finish. In the end all basic features are running, and I feel more confident with the usage of processing. At this stage it's already a running game, but the View part is too heavy.

Then I spent another long day to split as much as possible the logic from the presentation. Moving all the model component property changes, or calculations into the model objects' handleEvent() method.

I realized model component is talking about single enemy, fire, bomb, while the View is actually dealing with all of them together. So, for some event, the handling applies to all the objects in the same arraylist, but for others, only the current object is affected. In order to deal with this, I overloaded the handleEvent() method in controllers.

Since it's a small game, I don't want to create complex domain design for ideal decoupling but kept some association. E.g. the Ship class need to have a reference to the fireList, so that when it fires a new bullet, it can be added into the fire/bullet arraylist. Same for Enemy and Bomb.

By then it's already Tuesday. I added some advanced features like different enemies, bombs, weapons and this also makes the game more fun. I spent all the rest of the time working on documentation, which I feel might not be 10 word pages, but for sure more than 10 powerpoint pages.

I didn't have time to take screenshots or picture during my work, one reason is, I was concentrated on resolving issues, didn't think of taking screenshots; another is, I was not sure if I can make it at that time, no room to think about it.

Now looking back, I do recall some dump along the way but none of them was hard to resolve. The most time-consuming part is my thought fight against the AbstractView class. I couldn't convince myself it's necessary. Maybe I can get more insights during the peer assessment.