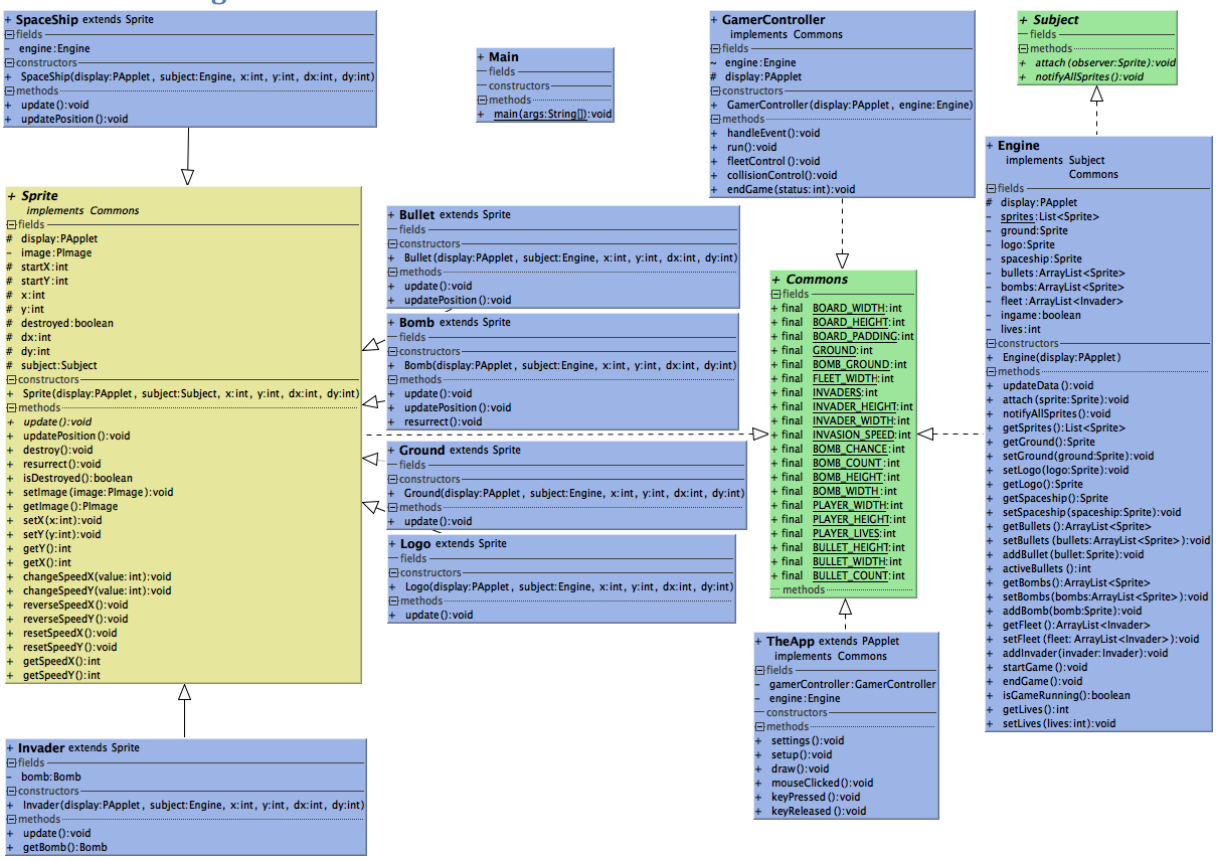
Space Invaders

Documentation

1. Introduction

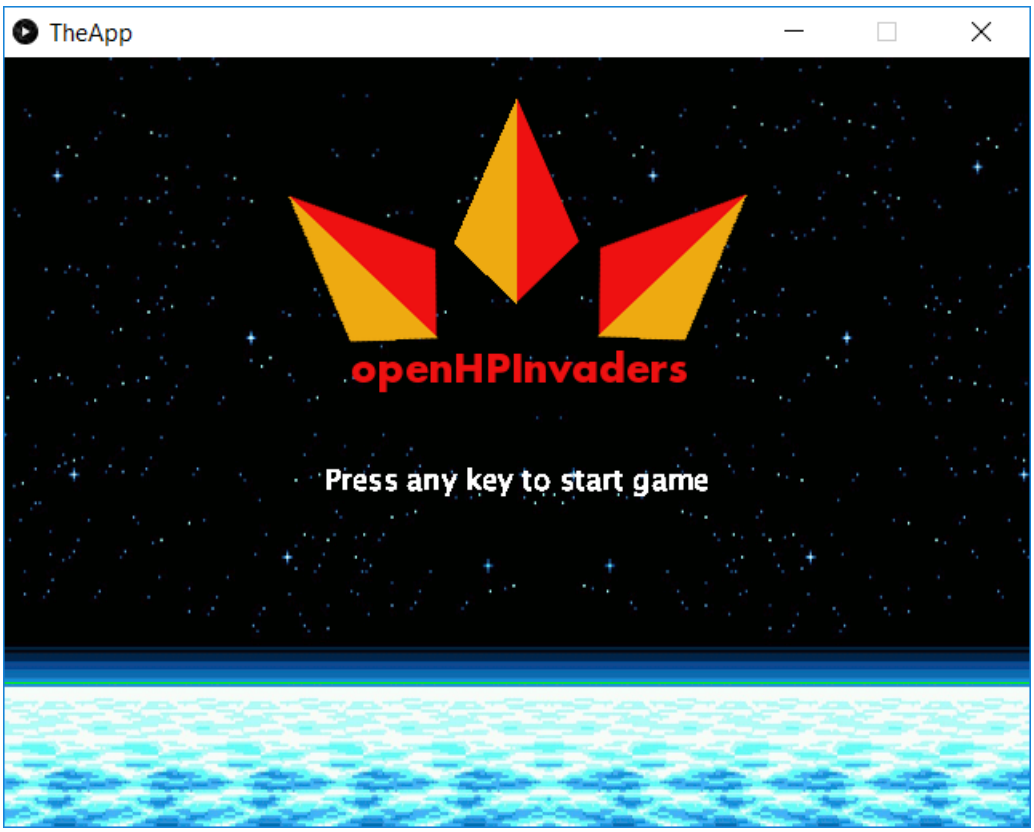
We created a game called “openHPInvaders”, a clone of Space Invaders which is an arcade game created in Japan and first released in 1978. It is a two-dimensional shooter game in which the player controls laser cannon by moving it horizontally across the bottom of the screen and firing at descending aliens.

2. UML Diagram



3. Game start

The game is started by pressing any key, on the intro screen:



4. Gameplay

The game controls are as follows:

Key “a”	**move left**
Key “d”	**move right**
Key “s”	**fire**

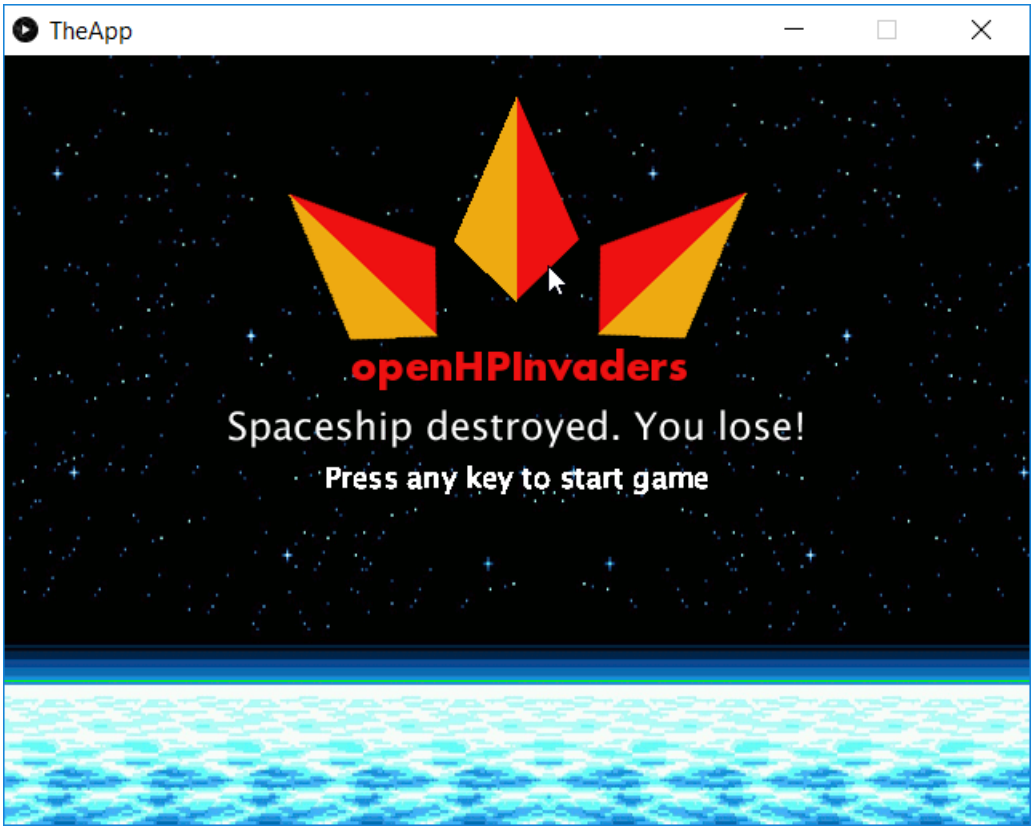
At the beginning, there are 6x4 = **24 aliens created**. The aliens are descending and firing the bombs randomly. An alien can’t fire a bomb, if there is another alien beneath.

The spaceship can fire laser bullets; there can be a **maximum of 3 active / visible laser bullet** on the screen at a time. An alien hit by a bullet is destroyed and removed from the screen.

The same applies to the spaceship, if hit by a bomb, it is destroyed.

The **spaceship / player has 3 lives**, displayed as spaceship picture in the right bottom of the screen.

End of the game:

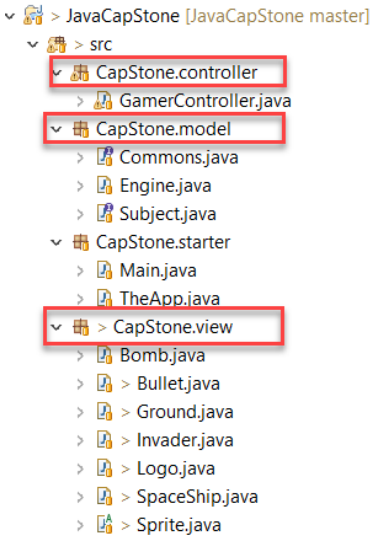


5. Code and game logic

The application was created based on the **model-view controller principle**. Four Java packages were created, one for model, then for view and the controller, each containing several Java classes.

The fourth package was created for the Main and TheApp class.

As you can see in the UML diagram, all the visible game objects were created (inherited from) based on the Sprite class.



Classes **Engine and GamerController** represent the game logic and the game control mechanisms.

The class **Commons** contains all the important global variables for the application.

Lab Report

The project started by knowing each other in the team. The team consists of **8 people**, unfortunately **only the few were actively participating in the project**.

We managed to have one telco, in which 4 members of the team participated. Besides the telco, the communication was quite active.

Telco arrangement:

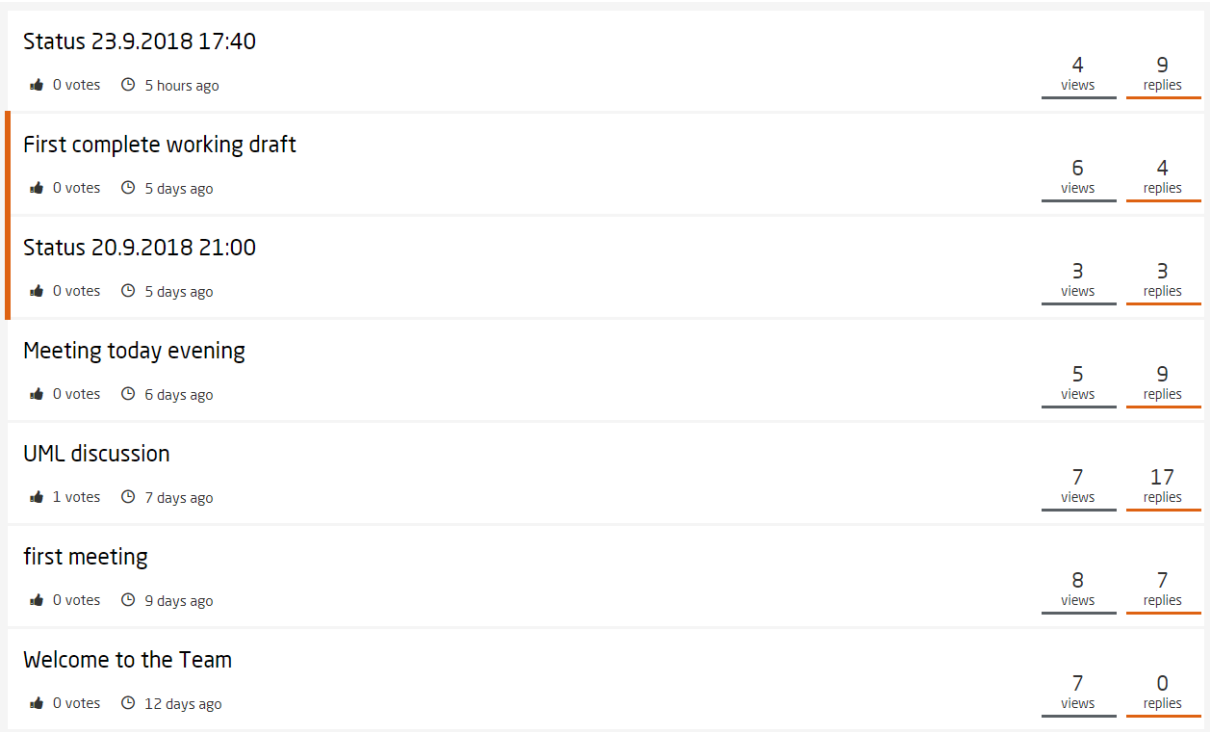
Hi all,

we planned to align today evening, 20:00 german time. I think we can use GoogleHangout. Even not worked with Google before, but it seems to be very easy to use. It seems, that someone has to start it and then a link can be shared with all other participants, so that they can attend. I will logon today evening at 19:00 and can share this link in this discussion forum.

Would you agree with that approach? If somebody has another possibility to setup a group discussion -> please feel free to make another proposal. Who will be able to attend?

It would be a great goal, if we could align on a 90% version of the UML diagram, which could be than shared on the GitHub

Discussion topics:



UML discussion:

I don't know if someone already created a draft for an UML diagram, at least nothing was posted, yet. I would like to agree on a terminology first which also gives us the chance to get the list of objects straight.

Here is my current idea (with **classes** and *Interfaces*):

(1) The player controls a **Cannon** which is both *Movable* (Left/Right) and *Shootable*. (2) The spaceship emits a **Rocket**, which is *Movable* (Up). This function is defined by *Shootable*. (3) Each foe is a **Ship** which is *Shootable*. (4) A ship drops a **Bomb** which is *Movable* (Down). (5) A collection of ships built the alien **Fleet** which itself is *Movable* (Left/Right and Down upon hitting borders).

I guess that would be the bare minimum but we can add additional stuff like counters, levels, more weapons and so on.

What do you think? Any better naming conventions? Additional objects/interfaces?

First working draft discussion:

Dear all,

I have been busy the last days and created a first complete working draft of the game. It includes:

- A start screen with a simple custom game logo (openHPInvaders), it shows a message to press any key to start the game.
- Once started, the spaceship (Player) and the fleet of 24 invaders (4 rows with 6 invaders each) are initialised.
- The fleet moves left and right, on each change of direction, the fleet moves down a bit.
- Each invader can (randomly) drop a bomb, there is a setting for a maximum of 6 bombs at any time.
- The spaceship can be controlled using the A and D keys to move left and right while the S key fires a bullet.
- CollisionControl checks if a bullet hits an invader (invader disappears) or if the bomb hits the spaceship (game over).
- Additionally, if any living invader reaches the spaceship the game is over as well.
- If all invaders are destroyed, the player wins. A counter in the bottom right corner shows the number of remaining invaders.

The game has one major issue right now: hitting multiple keys at the same time (e.g. shooting while moving), the game often gets stuck and the spaceship cannot be moved anymore. I am not sure right now how to resolve that.

To go the extra mile, we could add

- Multiple lives: if the spaceship is hit by a bomb, the game is not over, yet, but the player loses a life.
- Timer: Either show a timer at the end screen ("X seconds required to defend the planet.") or have a countdown. But I guess the countdown is already the down movement of the fleet as the game is over if the invaders reach the spaceship.
- Award points based on time passed by, accuracy of shots, ...
- Make a highscore table, possibly with the ability to enter a name.
- Add background image.
- Add explosion images for destroying invaders/spaceship.
- Add bombs of different variety, either different size or speed, and color code them.
- Add another weapon type to spaceship. Ideas would be (1) rocket that still shoots straight up but destroys all invaders in its path instead of only one or (2) heat-seeking rocket that always aims for the nearest invader.

The working environment was set up first - the GitHub account. Then the JavaCapstone project from the course was downloaded as a basis for the game.

We started with the simple UML diagram, where the basic game Java classes were drafted.

Then the progress of the implementation itself was very smooth, due to one member of the team, who implemented most of the game logic and the code.

The whole project from the beginning to the end was finished pretty much in 12 days, but working **actively maybe 2-4 days**.