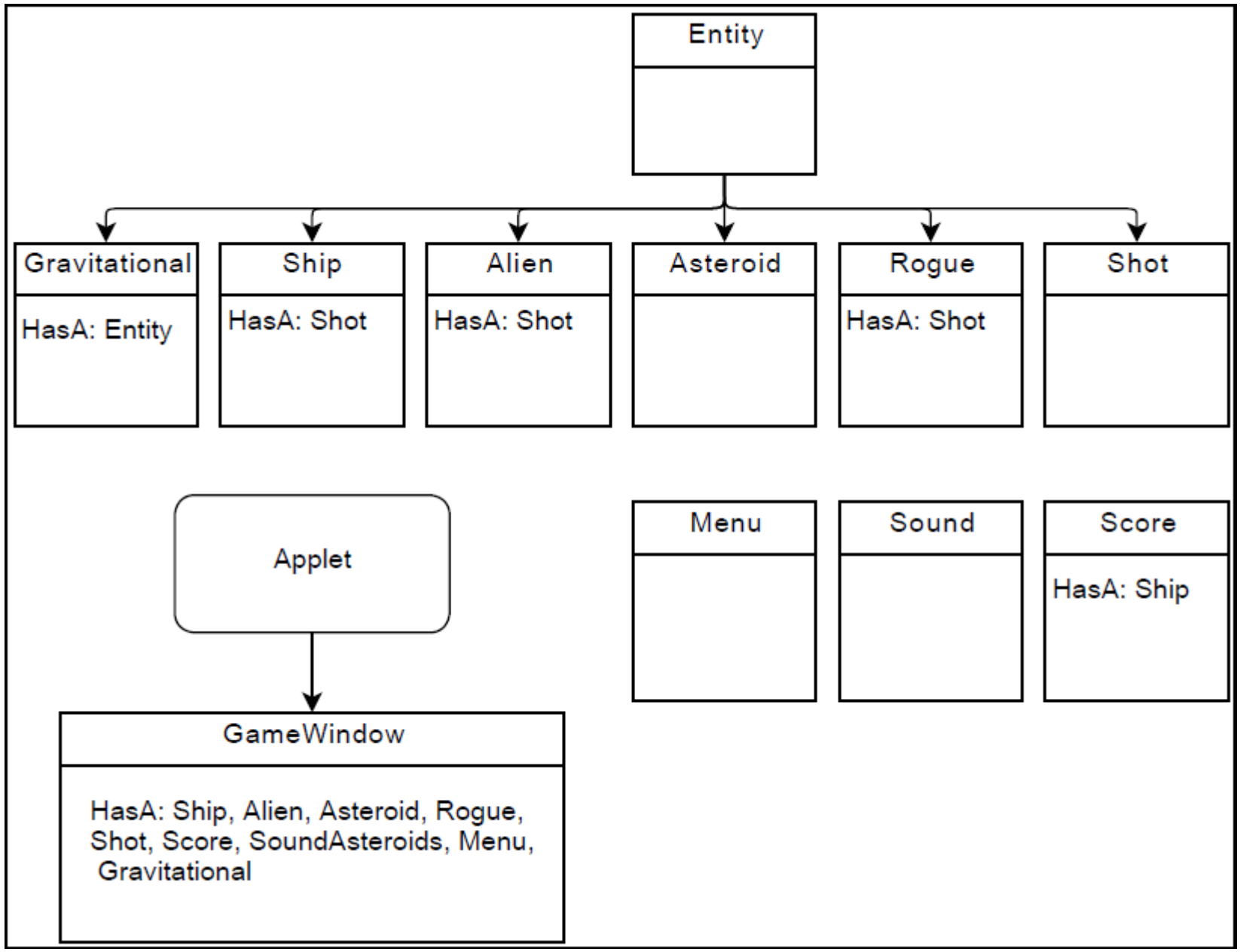
Asteroids

ECE 30862 - Fall 2012 - Midkiff

By: Christopher Ochynski & Ryan Pawling


Entity
Gravitational
HasA: Entity
Ship
HasA: Shot
Alien
HasA: Shot
Asteroid
Rogue
HasA: Shot
Shot
Applet
Menu
Sound
Score
HasA: Ship
GameWindow
HasA: Ship, Alien, Asteroid, Rogue,
Shot, Score, SoundAsteroids, Menu,
Gravitational

What have we learned about OO programming from the game?:

Right from the very beginning, we got to really appreciate the true power of object oriented programming. In order to be able to work efficiently and independently on the project, we made the above class structure. This enabled us to make solutions with guaranteed compatibility. The result? Our team finished the project in record in time. Object oriented programming allowed us to leverage encapsulation to minimize overhead associated with splitting up a programming project.

Furthermore, we learned that fully utilizing encapsulation was extremely beneficial in reducing errors and debugging time. In the beginning, we paid too little attention to making majority of the variables private. We learned quickly that when two programmers are writing code, the lack of private variables encouraged too much chaos. We quickly forced ourselves to use the public keyword as little as possible. Shortly thereafter, our debugging time was reduced drastically.

The use of inheritance also allowed us to create spaceship, alien ship, and asteroid objects much faster. By using a common foundation, a lot of redundant programming was reduced. Again, through this project we got appreciate the time savings of this concept.

What was hard to implement?:

Only two things in particular were hard to implement. One was the use of the Graphics library to generate the game screen. The difficulty came from learning to actually use this new library. The documentation in class and online were thorough. However, it took a little while to digest all the information and to begin applying it to the project. Thankfully, the power of java's object oriented environment meant that nothing extremely complex had to written from scratch. Once our team became familiar with the graphics library, implementation got easier.

On a similar note, the sound object was difficult to implement as well. We made use of static variables and functions to allow the sound object to be played in any scope. Again, the implementation just required upfront overhead of reading through documentation.

What, if anything, could have been changed in the project to make it better.

The project was a great learning opportunity. The only difference our team would've liked to see was more documentation of examples. It would've been nice to have a sample library of sounds to use. That would've helped with reducing the overhead of searching for sounds which doesn't really add to the learning aspect. It would've also helped to see some more screenshots of a working game. We understand this takes away from the creativity of the visuals used in the game. However, this would help to allow more time to focus purely on the programming aspect. Teams should be allowed to make their own graphics if they so choose.