

Program Design

Problem Definition

This project solves the classic Asteroids game problem, designed for a casual, classic-game enthusiast. The objective of this game is for the player to control a spaceship through space, destroying asteroids along the way. This is challenged by the presence of further asteroids being created upon the destruction of another, increasingly difficult levels, and scorekeeping. Interactions with the game are achieved through arrow key (for movement), the spacebar (for firing shots), and mouse clicked (for interacting with special option menus). In response to these user-provided interactions, the game will respond by moving the ship, firing shots, exploding asteroids, or resetting some portion of the game.

The method to process valid input consists of recognizing keystrokes and executing certain functions upon their being pressed and released. Because this game utilizes realtime interaction with various parts moving without any user intervention, it refreshes frame by frame to reflect such updates. Therefore, user input affects these frame refreshes differently. Additionally, Asteroids is based on “spacelife” physics (comets can fly over each other without collision), mixed in with several “earth” physics (shots colliding with comets destroy them; acceleration is limited; speed is based on angle and velocity). Once a user performs a valid interaction with the game, an event is fired that will affect the gameplay. This change becomes effective immediately within the code and is displayed to the screen upon frame refresh (which is intended to be rapid to simulate latency-free movement).

This rendition of the Asteroids game takes a different approach to the classic. It does not limit lives, but rather encourages continued gameplay in a non-linear timeline, allowing the user unlimited deaths (although this negatively impacts their score) and the ability to replay previous levels or restart the game in its entirety.

Criteria for Success

This rendition of the Asteroids game should run in realtime, with little to no latency between frames. It deals with invalid data by terminating the game and presenting a detailed error message. Extreme data input is scaled down to fit into a reasonable input for processing. This program is both completely independent and completely based off player interaction at the same time. On the first end of the spectrum, all space objects (comets and the ship) will interact with other, their surroundings, and space independently of user input. However, the movement of the ship, firing capabilities, and all button-related capabilities are completely user dependent and interactive. The proper response to user input is to complete the appropriate action if valid

(ie: accelerate on arrow up being pressed) and a detailed error message or data scaling if the input is invalid or extreme.

Overview of Program Architecture

Data structures being used in this project are the different classes for objects of space objects and vectors for the collection of these similar objects. A (or multiple) configuration text files store information necessary for starting a level is also used, along with various numerical variables for scorekeeping purposes.

The foundational structure for this program is a while loop that runs the entire game infinitely, or until the last level is completely. This allows the game to continue on a single level until the user passes it or quits the game. Other major algorithms include physics formulas for the calculation of velocity, speed, and movement.

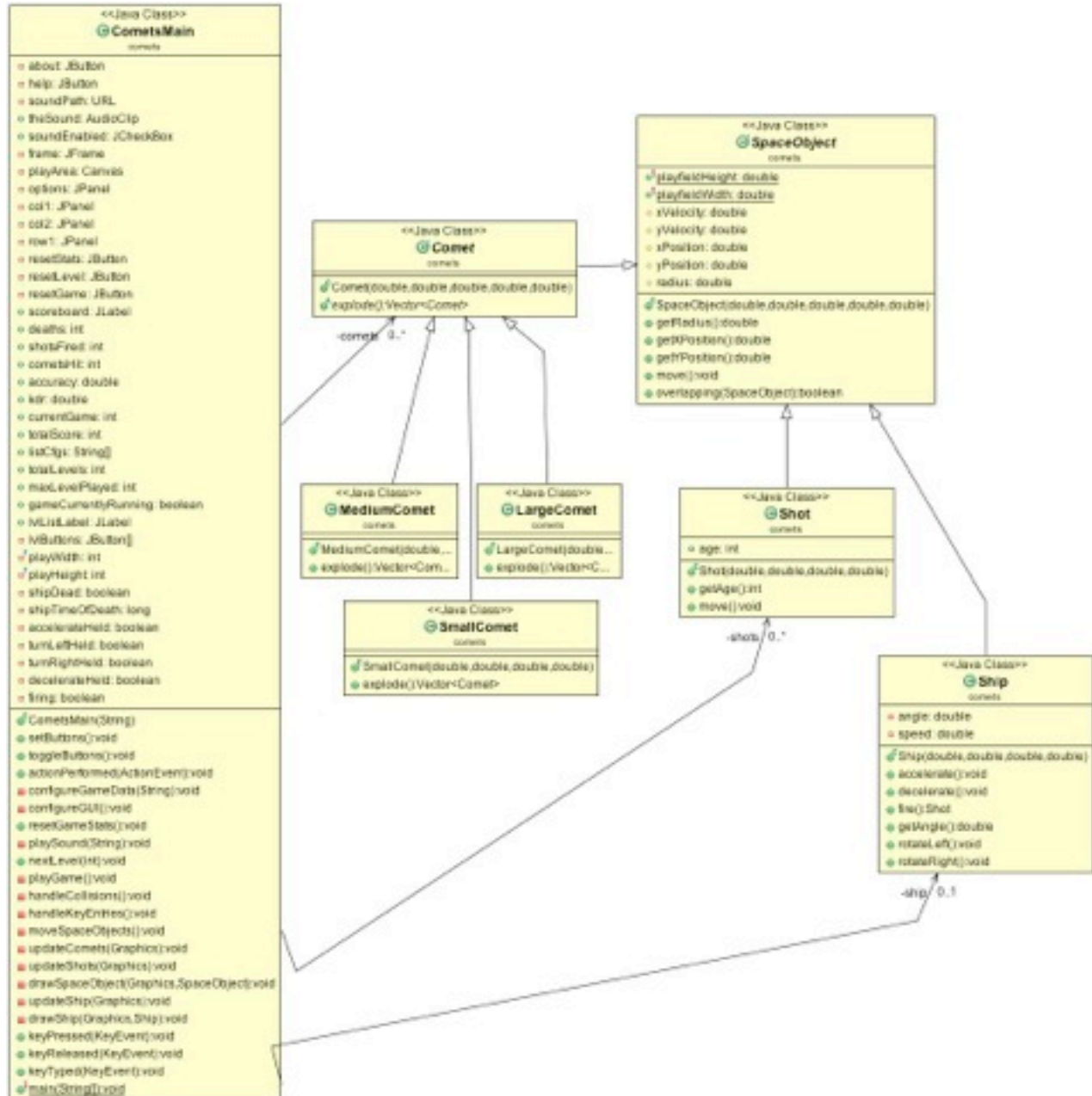
Hierarchical dependencies in this program stem from the comets package, which is the main supporting framework for every space object. The collections data structure stores all related space objects on the screen at any given time - one for comets and one for shots. These are the only two structures stored collectively, and the only two which actually relate to each other in same way (through collisions; although the ship can collide with a comet and thereby destroy the ship, but we'll ignore this fact since it is a simple conditional). The physics algorithms relate not only similar space objects, but also the space objects being stored collectively.

Program Implementation

Problem Evaluation

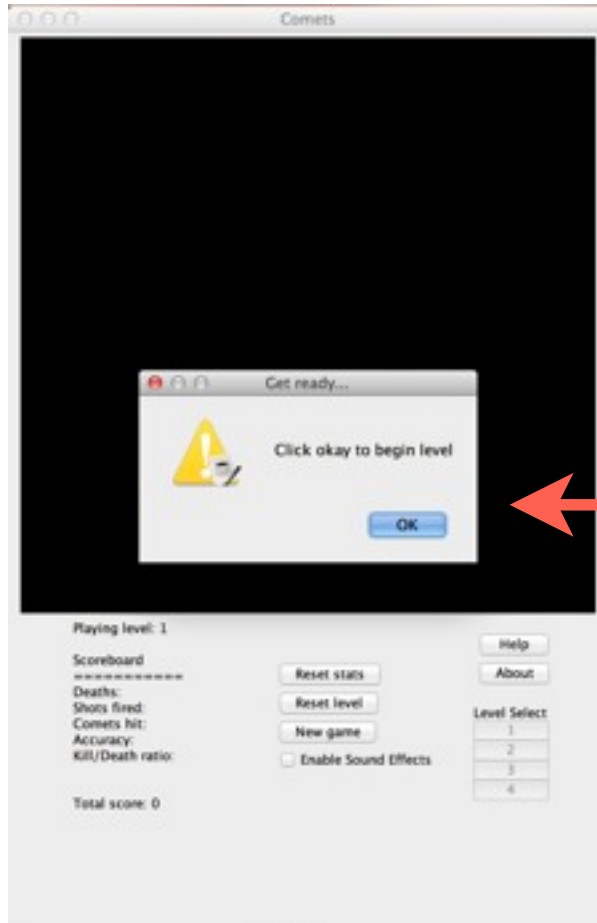
This project provides various user-friendly features including being latency-free, navigation between levels, a scores and stats board, and unlimited deaths. It also handles errors nicely, by either simply stopping the game and displaying an error message, or continuing to another pre-determined portion of the game, so the user does not know something went wrong. At this time, possible improvements to be released in future versions include adding a background image (currently, the chosen implementation of such feature results in extreme amounts of latency), further sound effects to complement that of the comet exploding, and the ability to chose between unlimited play (with no lives and no timer), a timed game, and a lives game. Such improvements would further enhance this game and make it a more complete version of the classic Asteroids game. The design used was very object oriented, making this an effective solution to such an intricate game. I learned various concepts of object oriented design, including the proper use of abstract classes, and several GUI topics, such as combining multiple layouts, nesting JFrame/JPanel/Canvas elements, and the use of sound and background images.

Program UML

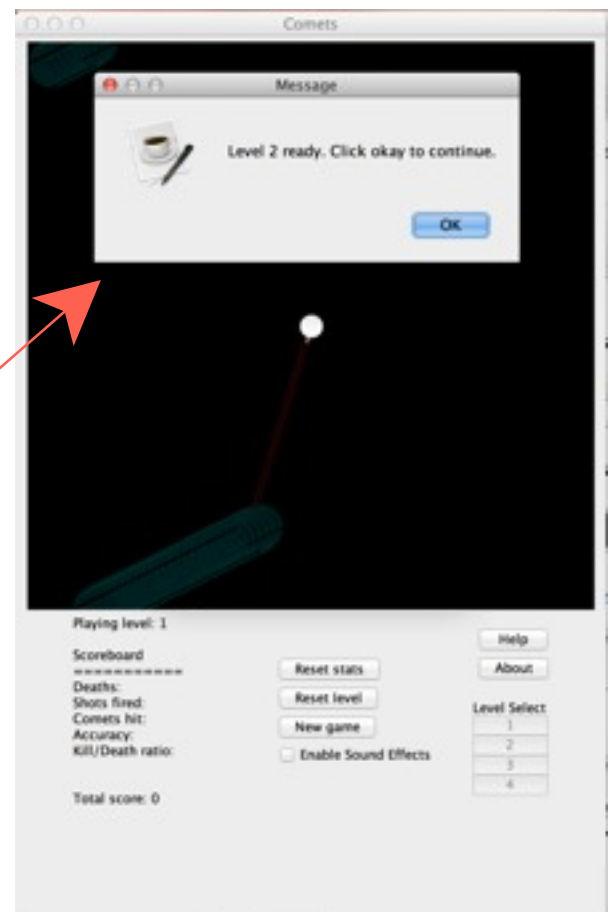


Program Documentation

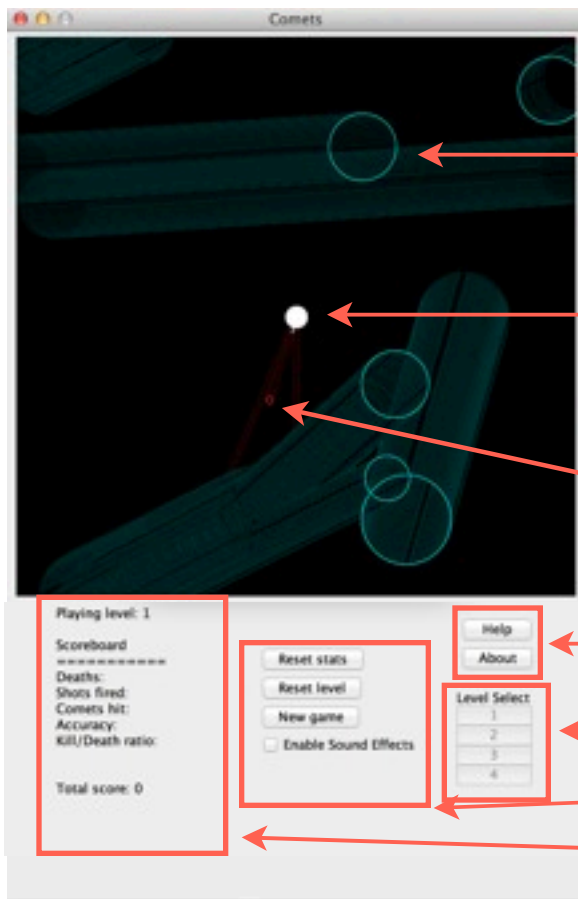
User's manual



A popup will appear before starting the first level, so you'll have a chance to prepare yourself between launching the game and the first level beginning.



Upon completing a level, you'll see this popup. Again, it gives you some information and a chance to prepare for the next level to begin.



This is a comet

This is your ship

This is a shot

How to play and about the developer

Select a previously played level

These are gameplay options

This is the score and stats board



Single press: fire a shot
Extended press: fire multiple shots

↑: accelerate

↓: decelerate

← / →: rotate left / right

Program Testing

Action activity log:

```
Console [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jul 26, 2013 10:10:55 AM)
<terminated> CometsMain [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jul 26, 2013 10:10:55 AM)
Accelerated from 0.0to 0.1
Accelerated from 0.1to 0.2
Accelerated from 0.2to 0.30000000000000004
Accelerated from 0.30000000000000004to 0.4
Accelerated from 0.4to 0.5
Accelerated from 0.5to 0.6
Accelerated from 0.6to 0.7
Accelerated from 0.7to 0.7999999999999999
Decelerated from 0.7999999999999999to 0.7
Decelerated from 0.7to 0.6
Decelerated from 0.6to 0.5
Decelerated from 0.5to 0.4
Decelerated from 0.4to 0.30000000000000004
Decelerated from 0.30000000000000004to 0.20000000000000004
Decelerated from 0.20000000000000004to 0.10000000000000003
Decelerated from 0.10000000000000003to 2.775575615628914E-17
Decelerated from 2.775575615628914E-17to -0.09999999999999998
Decelerated from -0.09999999999999998to -0.19999999999999998
Decelerated from -0.19999999999999998to -0.3
Decelerated from -0.3to -0.4
Decelerated from -0.4to -0.5
Decelerated from -0.5to -0.6
SHOT FIRED

SHOT FIRED

Accelerated from 0.0to 0.1
Accelerated from 0.1to 0.2
Accelerated from 0.2to 0.30000000000000004
Accelerated from 0.30000000000000004to 0.4
Accelerated from 0.4to 0.5
SHOT FIRED

SHOT FIRED

NEW COMET FOUND:
xVel: 1.5502323993169607      yVel: 1.0568415678684142      speed: 1.8762021723328721


NEW COMET FOUND:
xVel: 1.0200158412044389      yVel: 1.2410001609185515      speed: 1.6063977451764149

NEW COMET FOUND:
xVel: 1.1059473307346803      yVel: 1.515361952126212      speed: 1.8760174157802822
Accelerated from 0.5to 0.6
Accelerated from 0.6to 0.7
Accelerated from 0.7to 0.7999999999999999
Accelerated from 0.7999999999999999to 0.8999999999999999
Accelerated from 0.8999999999999999to 0.9999999999999999
```

Comets configuration file not found:

```
Console [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jul 26, 2013 10:07:28 AM)
<terminated> CometsMain [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jul 26, 2013 10:07:28 AM)
Unable to locate comets.cfg
```

Generating new comets upon exploding old comets (shows the new comet(s) x and y velocities and speed):

```
Console 
<terminated> CometsMain [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jul 26, 2013 9:44:42 AM)

NEW COMET FOUND:
xVel: 1.3405869974682978      yVel: 1.3762025748416482      speed: 1.9212253966627257

NEW COMET FOUND:
xVel: 1.3518734321888521      yVel: 1.1127028869766384      speed: 1.750905334775188

NEW COMET FOUND:
xVel: -1.2434032500007477      yVel: -1.3587634621267222      speed: 1.8418169800832596

NEW COMET FOUND:
xVel: 1.06442979205368      yVel: -1.475046961738617      speed: 1.8190036617736
```

Program Prompts

Why is SpaceObject abstract? How is this useful?

The entire game consists of components found in space and, therefore, they all have several attributes in common. Rather than having to redefine these common attributes per object created, it is more sensible to create a class that contains all these attributes. This class, SpaceObject, won't have all the attributes necessary to create an entire comet, ship, shot, etc; therefore, an object of type SpaceObject won't be able to be created, making this an abstract class. However, it contains all elements common to every space object, making this program DRY and flexible.

How could this program be modified to allow for comets of varying sizes? Is it possible to use a single comet class instead of four?

This program could be modified to allow for comets of varying sizes by generating a random size between a certain range when a comet explodes, and assigning different ranges for each comet category. For example, a large comet could be in the range 30-40, a medium comet 20-30, and a small comet 10-20. Using this method, when a large comet explodes, a random size within the range of a medium comet can be generated and used in the creation of the new comet. Likewise for a medium comet exploding into small comets. This method also makes it possible to combine the four comet classes into a single class by simply making Comet.java not an abstract class and implementing explode() within it. The only caveat with this model is that the explode method would require an extra step for determining what range the current comet falls into (large, medium, or small depending on the radius) in order to determine what size the new comets should be, if any.

Draw a UML diagram outlining the relationships between all of the classes used in the game.

The UML diagram for this project can be found on page three of this document.