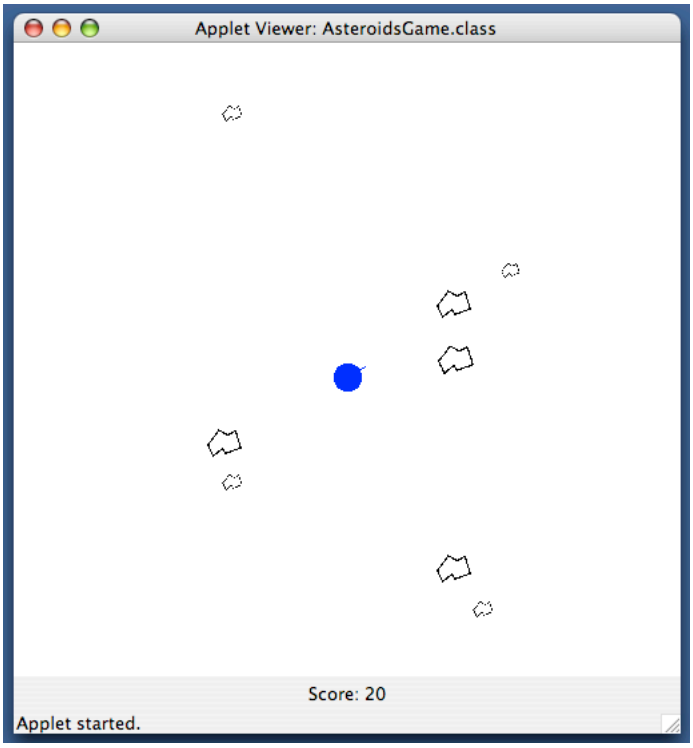
CS 51 Test Program #2
Asteroids

Design due: Thursday April 22, by 1:15 PM.
Due: Tuesday, May 4 at 5 PM.

This document describes what you are to do to complete your final test program. A test program is a laboratory that you complete on your own, without the help of others. It is a form of take-home exam. You may consult your text, your notes, your lab work, the lecture notes, our on-line examples, and other documentation directly available from the course web page, but use of any other source for code is forbidden. If you have problems with the assignment, please see the instructor.

Asteroids is a classic arcade game that was released by Atari in 1979. It was an instant hit, and is still popular today (look at the wikipedia page for more history. If you'd like to play the classic asteroids, go to: http://www.play.vg/games/4-Asteroids.html

For your final test program you will write a program that implements Asteroids, well "almost Asteroids". We have simplified the game considerably from the original in order to make the assignment more reasonable . . . however, there is a long list of extra credit points that you can earn by making your game more closely resemble the original. A demo version of this program is on the on-line version of this handout. You can play with it to see what we have in mind for this test program.



The figure above shows our Asteroids game after the player has already destroyed 2 asteroids (each worth 10 points). The game begins with a spaceship in the middle of the screen, and several asteroids traveling across the screen. The goal is for the spaceship to shoot the asteroids so that none of them hit the ship.

The asteroids are of different sizes and they move across the screen at different speeds. When an asteroid gets to the edge of the screen it wraps around and reappears at the opposite edge. Unlike the original Asteroids, the ship is fixed in the center of the screen, although it can aim its gun in any direction. If the player clicks on the right-arrow button on the keyboard, the gun rotates clockwise by 15 degrees (that is `Math.PI/12` radians); if the player clicks on the left-arrow button the gun rotates counterclockwise. If the player clicks on the space button, a bullet is fired from the end of the gun.

Each time a bullet hits an asteroid, the player gets 10 points. The game is over either when an asteroid hits the ship or when the player gets 200 points. In either case, a message is displayed to the user in the score area, indicating "You Won" or "Game Over" and the player is no longer be able to shoot bullets.

Implementation Details You should begin the game by setting up the black sky, a score-keeping mechanism, a ship, and asteroids.

The scorekeeper should display the score at the bottom of the screen. It needs to be able to increase the score when an asteroid is hit.

The ship is blue and consists of a body and a gun protruding from it (in practice, a circle with a line). The ship appears at the center of the screen. It must respond to the player's key presses: the gun should rotate clockwise and counterclockwise, and it must be able to shoot. If the ship is hit, it should turn grey.

The bullets shot at the asteroids will be active objects. They should move in the direction that they were fired, stopping either when they reach the edge of the screen or when they hit an asteroid.

Each asteroid has its own size, speed, and starting location. You may use the gif file that we provide ("rock.gif"), or something different. An asteroid should move across the screen appropriately, wrapping around whenever it reaches the edge of the screen. When an asteroid is destroyed, another one should appear at one of the *edges* of the screen.

Your program will contain several classes, corresponding to the objects just described:

AsteroidsGame This class sets up the game by creating an array of `Asteroid` objects and creating a `SpaceShip`. It also accepts user input in the form of key presses. In response to the different key presses, it should invoke methods of the ship, making the gun rotate or shoot.

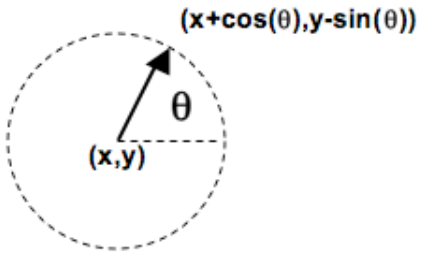
ScoreKeeper The `ScoreKeeper` class displays the score on the screen. Note that the asteroids should probably know about the `ScoreKeeper`, as they will likely need to inform it to increase when they've been shot.

SpaceShip The `SpaceShip` class draws the initial ship and provides methods that allow it to react correctly to key presses.

Asteroid The `Asteroid` class will extend `ActiveObject`. Its image is loaded in from a gif file (you are free to use the one that we provide), but each `Asteroid` should have a randomly chosen speed, direction, and size. When an `Asteroid` is created it should appear after a slight delay at the edges of the screen, then move across the screen (with wrap-around). An `Asteroid` can hit and destroy the ship, and can be hit and destroyed by a `Bullet` from the ship.

Bullet The SpaceShip shoots bullets. A `Bullet` is an `ActiveObject` that moves across the screen in the direction that it was fired, stopping either when it reaches the edge of the screen or when it hits an asteroid. Note that to achieve this behavior, each bullet needs to know about the array of asteroids.

There is one slightly tricky aspect in making the gun rotate appropriately, and having the bullets appear to be fired from the gun. The easiest way to think of this is in terms of an angle θ. θ should be represented in "radians", that is with respect to PI. PI radians represents 180 degrees and the initial angle starts to the right, so $\theta = 0$ is directly to the right, θ=`Math.PI/2` is straight up, θ=`Math.PI` is left, etc.



Given θ we can calulate the location of the line for the gun as well as the dx and dy for the fired bullets. If the gun has length 1 and has one endpoint at (x, y), the other end should be at $(x + \cos(\theta), y - \sin(\theta))$ (see the figure above). To get a line of length l, just multiply each of these by l to scale it.

A shot fired from that gun with speed s should have $dx = s * (\cos(\theta))$ and $dy = s * (-\sin(\theta))$. To calculate sin and cosine in Java, use `Math.sin(double theta)` and `Math.cos(double theta)`.

The Design As indicated in the heading of this document, you will need to turn in a design plan for your Asteroids program well before the program itself. This design should be a clear and concise description of your planned organization of the program.

As always, you should include in your design a sketch of each class including the types and names of all instance variables and constants you plan to use, the headers of all methods you expect to write and appropriate comments explaining what the variables and methods should represent/do.

In addition, you should provide pseudo-code for any method whose implementation is at all complicated. In particular, if a method is complicated enough that it will invoke other methods you write (rather than just invoking methods provided by Java or our library), then include pseudo-code for the method so that we will see how you expect to use your own methods.

Like `Frogger` it is important that you think about how the different objects interact. Ask yourself questions like "how will the ship know that it's been hit by an asteroid?", "how will an asteroid know it's been hit by a bullet?" and "how will I maintain the state of the game and update the score?"

The design will be graded on completeness, but also on how closely it matches the version of your code that you submit, so spend time thinking about how everything fits together and what will be in the classes!

Implementation Order **We strongly encourage that you to proceed as suggested below**. Build your program incrementally, getting each stage working (and running) before moving on! While a partial program will not receive full credit, a program that does not run at all generally receives a lower grade. In addition, it is much easier to debug a program if you know that some parts do run correctly.

1. Write a program that draws a `SpaceShip`.
2. Add code to make the SpaceShip respond to the user's right- and left-arrow key clicks.

3. Construct an `Asteroid` that can move across the screen, wrapping around appropriately, destroying the SpaceShip if it happens to collide with it. As each `Asteroid` moves it should be asking the ship "Have I hit you?".

4. Create an array of `Asteroid`s, each of which moves across the screen at its own speed.

5. Allow the ship to shoot `Bullet` objects. As each `Bullet` moves it should checking if it's hit any of the asteroids and then take the appropriate action.

6. Finally, set up the ScoreKeeper.

There is a great deal of functionality to aim for in this test program. As indicated above, **do not worry if you cannot implement all of the functionality**. Get as much of it working as you can. Note in the grading that a large number of points are assigned to issues of design and style. It is always best to have full functionality, but you are better off having most of the functionality and a beautifully organized program than all of the functionality with a program that is sloppy, poorly commented, etc.

Extensions Since we have deliberately left out many of the features of the original Asteroids game in our implementation, there are many additional features you could add to your program. In general we will award 1-2 points for each extension, for a maximum of 10 points extra credit. Note that the number of points possible if you do not do any extra credit is 95, but the test program is worth 100 points.

At the top of the the `AsteroidsGame` file, put in the comments all of the extensions that you did so that it is easy for myself and the graders to check that functionality. If you do not put them here, you may not get credit!

The following are some suggestions, together with the maximum number of points that will be awarded for each:

- (2 points) Allow the space ship to move across the screen, wrapping around as the asteroids do. As in the real version of the game, pressing the space bar should increase the ship's velocity in the direction that the gun is pointing.

- (1 point) Provide more than one life for the ship.

- (1 point) Provide better graphics for the space ship, perhaps imitating the classic shape in the original game.

- (1 point) To make the game more interesting, don't let the user fire for a certain period after the previous shot.

- (1 point) Limit the number of projectiles that the user can fire. Show the number of shots remaining on the screen.

- (1-2 points) Add sound effects

- (1 point) Modify the scoring so that it depends on the size of the asteroid hit.

- (2 points) After 200 points have been accumulated, increase the speed and/or decrease the size of the new asteroids. Restart the game, keeping track of a running total.

- (2 points) Animate the ship being hit by an asteroid using an `ActiveObject`.

- (2 points) Allow the asteroids to fragment into smaller asteroids.

- (varies) Emulate some other feature of the original game, e.g. add an alien ship that periodically moves across the screen and shoots at the spaceship.

Turning it in Your design should be turned in on paper in class or e-mailed to the professor. Keep a copy for yourself since we won't return it to or give you feedback on it until after the program is due.

When your work is complete you should deposit in the dropbox folder Before turning it in, please be sure that your folder contains all of the .java files, .class files, etc. as we can't give credit for anything we don't receive.

Before turning it in, make sure the folder name includes your name and the phrase "TP2". Also make sure to double check your work for correctness, organization and style.

Table 1: Grading Guidelines

Value	Feature
	Design preparation (20 pts total)
	Syntax Style (15 pts total)
6 pts.	Descriptive and helpful comments
2 pts.	Good names
2 pts.	Good use of constants
2 pts.	Appropriate formatting
3 pts.	Appropriate use of public/private
	Code quality (25 pts total)
5 pts.	Conditionals and loops
5 pts.	General design/efficiency issues
5 pts.	Parameters, variables, and scoping
5 pts.	Good correct use of arrays
5 pts.	Miscellaneous
	Correctness (35 pts total)
5 pts.	Gun rotates correctly w/ key presses
5 pts.	Ship fires projectiles
5 pts.	Asteroids move correctly
5 pts.	Ship projectiles destroy asteroids
5 pts.	Asteroids can destroy ship
5 pts.	Scoring is correct
5 pts.	Asteroids generated correctly
	Extensions (5 - 10 pts total)