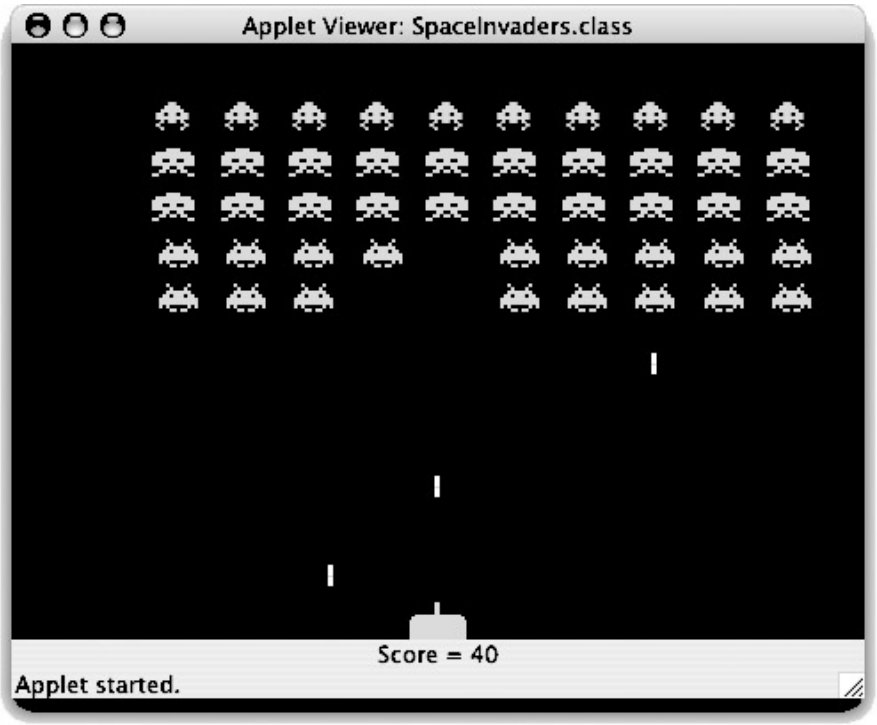
Programming Project 2

Design Due: 30 April, in class
Program Due: 9 May, 4pm
(late days cannot be used on either part)

Handout 13
CSCI 134: Spring, 2008
23 April

Space Invaders



Space Invaders has a long and illustrious history, first appearing in video arcades in 1978. By 1980, it had been licensed by Atari and became the first arcade game adapted to Atari's new home video game system.

Your final project will be to write Space Invaders. We have simplified the game somewhat from the original, and you can find a working version of what we have in mind on the course web page. If you want to experience the original version, visit `http://www.spaceinvaders.de/`.

The game begins with several rows of aliens at the top of the screen. At the bottom is a lone space ship that must defend the earth from the attacking aliens. The aliens move across the screen from left to right and then back again, occasionally shooting at the good-guy space ship. They move at a constant rate and in sync with each other.

The space ship also moves from left to right, but its movement is controlled by the player. If the player clicks on the right-arrow button on the keyboard, the ship moves to the right; if the player clicks on the left-arrow button, the ship moves the left. In addition to dodging the projectiles shot by the aliens, the ship can shoot back. This is also controlled by the user by clicking the space bar or up arrow.

Each time a defense projectile hits an alien, the player gets 10 points. The game is over either when the player gets all the aliens or when the player is hit. In either case, a message is displayed to the user, indicating "Game Over" and showing the final score.

Programming Project Rules

A programming project 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, our on-line

examples, and the web pages associated with the course web page, but use of any other source (human or otherwise) for code is forbidden. You are encouraged to reuse the code from your labs or our class examples.

You may not discuss these problems with anyone aside from the course instructors. You may only ask the TA's for help with hardware problems or difficulties in retrieving your program from a disk or network. The use of any other outside help or sources is a violation of the Honor Code.

Implementation Details

You should begin writing the game by setting it up. This involves creating a black sky, a score-keeping mechanism, a good-guy ship, and all those nasty aliens. The overall look and feel of the game is entirely up to you, as long as it implements the basic behavior outlined below. We have provided image files for the aliens in the starter folder that you may use if you like. (We provide several files in case you want to add variety or a little animation, but you only need to use one of the provided images for the basic game.)

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

The good-guy ship is an object that will appear at the bottom of the screen. It must respond to the player's key clicks. That is, it should be able to move to the right and to the left. It must also be able to shoot at aliens. If it is hit, it should stop firing. If you want, you can make it disappear in some interesting way.

The projectiles shot at the aliens will be active objects. They should move up the screen and stop either when they reach the top or hit an alien.

The aliens should move as a group. They move together from left to right; and then they move together from right to left. When an alien is hit by a projectile, it should disappear from the screen. You need to be a bit careful about how you keep track of the aliens. We suggest you use a two dimensional array (or an array of objects representing columns). Also define a class to represent a single alien. **Don't try to delete aliens that have been hit from the array and shift other elements in the array over to fill the hole.** If you do this, bad things may happen if a projectile tries to rearrange your array to delete an alien that has been hit at the same time that the alien is being moved across the screen. Instead, when an alien is hit, just set a variable within the object that represents the alien to indicate that it is dead and remove it from the screen.

The projectiles shot by the aliens must move down the screen, stopping either when they reach the bottom or when they hit the ship.

To summarize, your program should include the following classes:

SpaceInvaders: The controller will set up the game. It will also accept user input in the form of key clicks. In response to the different key clicks, it should invoke methods of the ship, making it move or shoot. We have provided the skeleton for listening to the user's key clicks. It is your responsibility to set up the game and to fill in the lines where the ship's methods need to be invoked.

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

SpaceShip: The `SpaceShip` object moves in response to each key press of the left and right arrow keys. When the space key or up arrow key is pressed, it should lauch a "defense projectile." Our space uses a `FilledRoundedRect`. However, any reasonable SpaceShip representation is fine.

Invaders: The `Invaders` class will extend `ActiveObject`. This object will hold an array of aliens.

Alien: An `Invaders` object keeps track of a bunch of `Aliens`, each with behavior all its own. An alien can move, it can launch a projectile, and it can disappear when shot.

Projectile: `Aliens` shoot projectiles. A `Projectile` is an `ActiveObject` that moves down the screen, stopping either when it reaches the bottom or when it hits the space ship. Note that to

achieve this behavior, the projectile needs to know about the space ship. Since projectiles are created by aliens who are members of the group of `Invaders`, the ship must be passed as a parameter through all of these classes.

DefenseProjectile: Last, but not least, are the projectiles shot by the good guy. They move up the screen, stopping either when they reach the top or when they hit an alien.

Consider this: The aliens need to know about the ship so that they can aim for it with their projectiles; but the ship needs to know about the aliens so that it can shoot at them. If this seems circular to you, you're right. You might handle this by creating the ship and passing it as a parameter to the `Invaders` when you create them. Then, after you've created the `Invaders`, you might invoke a method of the ship class (perhaps called `setTarget`), that will pass the `Invaders` as a parameter to the ship.

You may also want to define other classes if you believe they will simplify your design.

The Design

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

You should include in your design a sketch of each class including the types and names of all instance variables you plan to use, and the headers of all methods you expect to write. You should write a brief description of the purpose/function of each instance variable and method.

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.

Implementation Order

We strongly encourage you to proceed as suggested below to ensure that you can turn in a running program. While a partial program will not receive full credit, a program that does not run at all generally receives a lower grade. Moreover it is easier to debug a program if you know that some parts do run correctly.

1. Experiment with the demonstration program.
2. Write a program that draws a `SpaceShip` object.
3. Add code to make the `SpaceShip` respond to the user's right- and left-arrow key clicks. Don't worry about shooting at this point.
4. Next construct the aliens. If you don't feel comfortable constructing a two-dimensional array of aliens, start by constructing one row of aliens. If you construct one row of aliens, you should be using a one-dimensional array.
5. Now make the aliens move from left to right and then right to left, etc.
6. Make the aliens shoot at the ship. There are a number of possible ways to do this. Here are a couple of ideas.
   (a) At each time step randomly select an alien to do the shooting.
   (b) When you construct each alien, give it a randomly selected "shooting interval". As the time interval passes, the alien should shoot and then restart its timer.
7. Now make the ship shoot at the aliens. The projectile shot at the aliens should, as it's moving, be asking them "Have I gotten any of you?". Our demo only allows you to shoot one projectile at the aliens at a time, but you do not need to implement this feature.

8. Finally, set up the score keeper.

There is a great deal of functionality to aim for in this programming project. **Do not worry if you cannot implement all of the functionality**. Get as much of it working as you can. As we have throughout the semester, we will consider issues of style and design, in addition to correctness, when we grade your final program. 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 and sloppy, poorly commented program.

Extensions

Since we have deliberately left out many of the features of the original Space Invaders game, there are clearly many additional features you could add to your program. We will award 1-2 points for each extension, for a maximum of 6 points extra credit. Some possible extensions are:

- Implement continuous, smooth motion for the space ship.
- Modify the scoring: for each alien in the bottom row, award 10 points; for each in the second row, award 20 points; for each in the third row, award 30 points, etc.
- As aliens near the right and left edges are hit, adjust the motion of the remaining aliens so that they move all the way to the right and left edges of the screen.
- Use multiple images for aliens to animate their motion.

Getting Started

You will write most of this program from scratch. However, the handouts page does contain a BlueJ starter project with a window controller to handle key strokes and the image files for aliens.

Submitting Your Work

Design. Your design should be turned in on paper in class on the due date. Keep a copy for yourself since we will not be able to return them to you promptly. We will look at them and provide feedback. The design will constitute a part of your grade on this project.

Program. Once you have saved your work in BlueJ, please perform the following steps to submit your assignment. Be sure to rename your folder to include your name, as usual.

- First, return to the Finder. You can do this by clicking on the smiling Macintosh icon in your dock.
- Click “Go” and select “Connect to Server.”
- For the server address, type in “Cortland” and click “Connect.”
- Select the button next to “Guest” and click “Connect.”
- A selection box should appear. Select “Courses” and click “Ok.”
- You should now see a Finder window with a “cs134” folder. Open this folder.
- You should now see the drop-off folders for the three labs sections. Drag your project folder into the appropriate lab section folder. When you do this, the Mac will warn you that you will not be able to look at this folder. That is fine. Just click “OK.”
- Log off of the computer before you leave.

Grading Guidelines

Points will be assigned roughly as follows:

Design (16 points)

- Description of classes
- type and description of instance variables
- signatures and descriptions of methods
- outline of complex methods

Programming Style (42 points)

- Descriptive comments
- Good names
- Good use of constants
- Appropriate formatting (indenting, white space, etc.)
- Parameters used appropriately
- Proper use of boolean conditions
- Proper use of ifs/whiles
- Proper use of variables (instance/local)
- Proper use of public/private
- Proper use of arrays

Correct Functionality (42 points)

- Ship moves correctly w/ key presses
- Ship fires projectiles
- Aliens move from left to right and back
- Aliens fire projectiles
- Ship projectiles destroy aliens
- Alien projectiles destroy ship
- Scoring is correct

Extra Credit (up to 6 points)