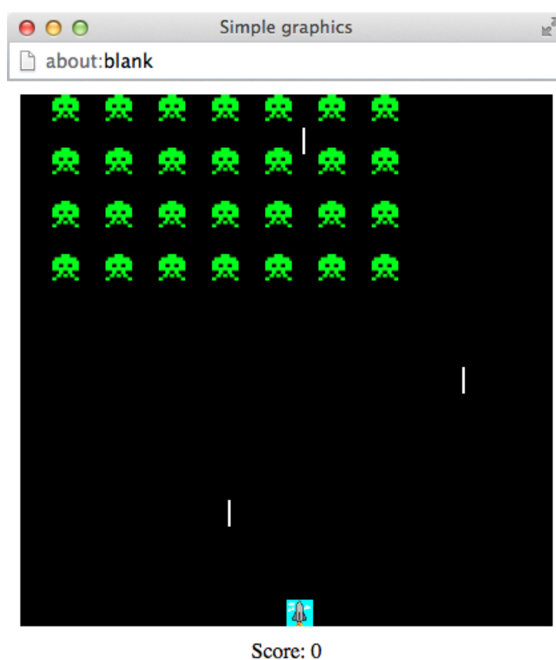
# CS 51 Test Program #2
# Space Invaders

Due: Wednesday, May 2 at 4 p.m., Design due: April 24, by 11:00 p.m.

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. You may not consult with anyone but the instructor on this program. If you have problems with the assignment, please see me.

Space Invaders, the popular arcade game, was designed and programmed for Taito, Japan in 1978. The game was licensed from Taito by Midway for production in the US. By 1980, it had been licensed by Atari and was the first arcade game adapted for Atari's home system. It remains as popular as it was when it was first introduced. If you'd like to play the classic space invaders, go to http://www.mysteinbach.ca/game-zone/85/space-invaders/




In case you haven't guessed, for your final test program we would like you to write a program that implements Space Invaders. To be more precise, we would like you to write a program that implements "almost Space Invaders". We have simplified the game somewhat from the original in order to make the assignment more reasonable.

The starting configuration for our Space Invaders game is shown above. The game begins with several rows of aliens at the top of the screen. At the bottom is a lone space ship that will try to defend itself against the attack of the 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. They move right until the rightmost surviving ship is close to the right edge of the screen, and then move left until the

leftmost surviving ship is close to the left edge of the screen. (You can decide what "close" means – make it look good!)

The space ship also moves from left to right and back, 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 missiles shot by the aliens, the ship can shoot back. This is also controlled by the user, by clicking the space bar.

Each time a defense missile 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 "You Won" or "Game Over".

**Implementation Details** You should begin 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. We have provided gifs for the aliens and the ship. They are "invader1.png" and "rocket.png", and can be found in http://www.cs.pomona.edu/classes/cs051G/Images/. Feel free to use them, if you'd like.

The scorekeeper should display the score at the bottom of the screen. It needs to be able to increase the score by 10 when an alien is hit. It should decrease the score by 1 point every time the user fires a defensive missile.

The good-guy ship is an object that will appear at the bottom of the screen. It must respond to the player's key presses. 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 disappear.

The missile shot at an alien will be animated. It should move up the screen and stop either when it reaches the top or when it hits an alien.

To represent the aliens you should use a two-dimensional array. The aliens, after all, do appear on the screen in a grid-like pattern. The aliens should move as a group. They move together from left to right; and then they move together from right to left. When a member of the group disappears, no one takes its place. It simply stays blank.

Each entry in the two-dimensional array will be an alien, a complex object with behaviors and capabilities all its own. It can move, it can shoot, it can disappear when shot.

The missiles shot by the aliens move down the screen, stopping either when they reach the bottom or when they hit the ship. They too should be removed from the canvas once they stop.

Your program will comprise several classes, corresponding to the objects just described.

**spaceInvaders** This is the main program. It will set up the game as well as accept user input in the form of key presses. 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 the score when they've been shot. The space ship will also need to know about it as the score should decrease whenever the space ship shoots a missile.

**spaceShip** The `spaceShip` class will create the image of the space ship at the bottom of the screen and respond to the key presses. It will also need to explode when hit by a missile from the aliens. Other methods might also be necessary.

**invaders** The `invaders` class will hold a two-dimensional array of aliens. The aliens move as a group from left to right across the screen and then back again. From time to time they shoot at the space ship below.

**alien** The group of Invaders is made up of individual Aliens, each with behavior all its own. An alien can move, it can launch a missile, and it can disappear when shot.

**missile** Aliens shoot missiles. An object from the missile class drops down the screen, stopping either when it reaches the bottom or when it hits the space ship. Note that to achieve this behavior, the missile needs to know about the space ship. Since missiles 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 so the missile can determine whether or not it hits the ship.

**defensiveMissile** Last, but not least, are the missiles shot by the defender/space ship. They move up the screen, stopping either when they reach the top or when they hit an alien.

I have provided the types of all objects used in the program in the file SpaceInvadersType.grace. It will need to be imported into the other files in the program in order to get access to these types. While I have provided some methods in the types, others are left for you to figure out.

### Design Issues

**Group Alien movement** Note that the aliens all move in perfect unison, and that the block of moving Aliens turns left when the first Alien hits the right edge, or turns right when the first Alien hits the left edge. Both of these behaviors would be difficult to implement if each Alien was an independently animated object.

Rather, we suggest that a single animated loop (in the Invaders class) moves and turns all of the Aliens, and fires occasional Missiles downwards. We suggest that your Alien class should implement methods to move, fire missiles and remove itself from the screen ... but that these methods should be called from the Invaders animated loop.

**Alien Missile firing** Periodically a random Alien should fire a missile. The two most obvious ways to implement this are:

1. choose a random time before each alien launches its next missile and have the move loop check whether or not the currently moving Alien should fire.
2. in the move loop, after some number of aliens have moved, choose a random (still living) alien and have him launch a missile.

**Missile Targets** Each missile (either alien or defensive) is an independent animated object, that follows an upwards/downwards path until it reaches the arena top/bottom or hits its target. This means that each missile needs (a) to know where the top/bottom of the arena is and (b) a reference to its target(s) so it can determine whether or not it has hit it (e.g. whether the missile image overlaps the target image).

There is only a single SpaceShip, and each alien Missile must be passed a reference to it. There are numerous (and a continuously changing number of) Aliens. Rather than having each DefenseMissile continuously enumerate the list of Aliens, it would probably be simpler to have the DefenseMissile regularly ask the Invaders object if it overlaps any Alien, and let the Invader object check all of the Aliens to see if the DefenseMissile has hit it.

The need for Missiles to have references to their targets creates a circular reference problem: The SpaceShip needs a reference to the Invaders object and the Alien Missiles (created by the Aliens under control of the Invaders) needs a reference to the SpaceShip target. If we create the SpaceShip

first, we can pass that object to the Invaders constructor, but how would the `SpaceShip` get a reference to the `Invaders` object? If we reverse the order, creating the `Invaders` (and `Alien`s) first, we could pass a reference to the `Invaders` object to the `SpaceShip` constructor, but how would the `Invaders`→ `Alien`s →`Missile`s get a reference to the `SpaceShip`?

This suggests that we cannot depend on being able to pass target information to the `SpaceShip` and `Alien`s in a parameter to their constructors. Rather we will need to define some other means of passing this information (e.g. a `setTarget` method that can be called after the SpaceShip and/or Aliens have been created).

**Constructors and Methods** For this (final) project, we have not given you code or even method declarations for most of the classes. You have to design these for yourself. The algorithms and the methods will evolve together:

- as you write code, you will realize what methods you need, and will define the methods to perform the functions needed by the callers
- as you define the parameters and functionality of each method, it becomes clearer how the code that calls them has to be structured.

Thus, before you can start serious implementation, you should have

- a preliminary list of the methods (and their parameters) in each class ... so you know what you are writing.
- a preliminary sketch of the more complex algorithms in the program ... so you know how you plan to write it.

Because this project is so much more complicated than any of the earlier labs, we are asking you to prepare and submit a more detailed design before you begin serious implementation:

1. for each class (Invaders, Aliens, Missile, SpaceShip, DefenseMissile, ScoreKeeper, and any others you want to create): list the methods, and parameters, and one-line description of what that method does.
2. provide pseudo-code for the algorithm you will use (in the Invaders class) to move the Aliens.
3. describe how a Missile will determine whether or not it has hit the SpaceShip and how a Defense-Missile will determine whether or not it has hit an Alien.
4. describe how you will decide which Alien should fire a Missile when.

Place all of this information in a single (ASCII text) file and submit it (by the due date) using the standard submission procedure with a name of the form `TP2_studentID.txt` 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 names and types of all definitions and 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 identifier introduced and each 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 Grace 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 assure 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. 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. Don't worry about shooting at this point.
3. Next construct the aliens. Use a `Matrix` to organize them in the `Invaders` class.
4. Now make the aliens move from left to right and then right to left, etc. I suggest you do this with a single animated while loop rather than trying nested while loops.

   Inside that while loop I suggest you call a confidential method (I called mine `moveThem`) that moves all of the aliens by a small amount. If any of the aliens go off the screen to the right then once everyone in the group finishes the move, change the amount to move to a negative number in anticipation of the next move. Similarly, if any of the aliens go off the screen to the left during a move, then after all of the aliens have finished that small move, reset the amount to move to be a positive number.
5. Make the aliens shoot at the ship. (See the earlier advice on this.)
6. Now make the ship shoot at the aliens. The missile shot at the aliens should, as it's moving, be asking them "Have I gotten any of you?".
7. Last, but not least, set up the score keeper.

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. At the end of this handout we have included some basic grading guidelines for this test program. You should note the large number of points assigned to issues of 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 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 10 points extra credit. The number of points possible if you do not do any extra credit is 95. Thus you must do some extra credit to have a chance at getting 100 points or more. Some possible extensions are:

- Replace the simple (rectangle) missiles in the demo with more attractive images.
- Create colorful/expanding explosions when a missile hits its target.
- Implement continuous, smooth motion for the space ship.
- Add sounds for movement and explosions.
- To make the game more interesting, don't let the user fire for a certain period after the previous shot.

- Each time the ships change direction, move them farther down the screen.
- Limit the number of missiles that the user can fire. Show the number of shots remaining on the screen.
- 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.
- Implement multiple lives, so that a few seconds after the spaceship is destroyed, it is re-created and the aliens resume firing at it.
- After one screenful of aliens has been wiped out, start over with a new screenful, but start them closer to the ground. Provide more than one life for the ship. Accumulate extra lives each time you wipe out a screenful.
- Emulate other features of real game, e.g., mother ship for aliens, bunkers to hide behind, etc.

**Turning it in** Your design should be turned in on paper in class or e-mailed to us by 11 p.m. on Tuesday, April 24. Keep a copy for yourself since we will *not* return it to you until after the program is due.

When your work is complete you should deposit in the dropoff folder. Before turning it in, please be sure that your folder contains all of the .grace files, and make sure it will run with no modifications necessary on our part. If we have to make modifications to get it to run, it will cost you points, even if it is just changing the name of files.

Before turning it in, make sure the folder name has the form `tp2_lastnamefirstname`. Also make sure to double check your work for correctness, organization and style. In particular, the comment at the top of each file MUST include your name.

Table 1: Grading Guidelines

| Value | Feature |
| --- | --- |
| | **Design preparation (20 pts total)** |
| 10 pts. | Method descriptions |
| 5 pts. | Alien movement |
| 2 pts. | Alien missile launching |
| 3 pts. | Missile movement and targeting |
| | |
| | **Syntax Quality (12 pts total)** |
| 6 pts. | Descriptive and helpful comments |
| 2 pts. | Good names |
| 2 pts. | Good use of constants |
| 2 pts. | Appropriate formatting |
| | |
| | **Semantic Quality (25 pts total)** |
| 5 pts. | conditionals and loops |
| 5 pts. | Parameters, variables, and scoping |
| 3 pts. | Appropriate use of public/private |
| 5 pts. | Good correct use of lists/matrices |
| 5 pts. | General design/efficiency issues |
| 5 pts. | Miscellaneous |
| | |
| | **Correctness (35 pts total)** |
| 5 pts. | Ship moves correctly w/ key presses |
| 5 pts. | Ship fires missiles, which move correctly |
| 5 pts. | Aliens move from left to right and back |
| 5 pts. | Aliens fire missiles, which move correctly |
| 5 pts. | Ship missiles destroy aliens |
| 5 pts. | Alien missiles destroy ship |
| 5 pts. | Scoring is correct |
| | |
| | **Extra Credit (5 - 10 pts total)** |