# CS 251 Intermediate Programming
# Space Invaders Project: Part 2 – Complete Game

Brooke Chenoweth

Spring 2024

## Goals

To carry on forward with the Space Invaders program we have been working on, we are going to make a final push toward the end goal. This project will take quite a bit of programming to get done. I strongly recommend talking to TAs, tutors, and myself, to hash out any problems that you might have, and to do so early on.

The overall goals of this project are:

- Create a complete working Space Invaders game
- Write a larger program that actually does something
- Put the knowledge of many small individual pieces together into a larger system
- Find out how inheritance can really help you, and why it's so neat
- Get some experience writing a GUI program that utilizes both mouse and keyboard input
- Master the process of breaking a large problem down into smaller pieces. You do *not* want to write this entire program in a single main method.

## Program Description

I'm describing this program as a classic Space Invaders game with a ship shooting at aliens, but I encourage you to be creative with the game design. Instead of aliens and a ship, perhaps fend off attacking ants with a can of bug spray? What about ranks of snowmen throwing snowballs at you? Just make sure the basic game mechanic stays the same.

### Ship

The player will control a ship that moves horizontally along the bottom of the screen and shoots a laser at the aliens.

- The ship should be drawn as something more complicated than just a simple rectangle or oval.
- Left and right arrow keys control ship movement. The ship should continue to move when an arrow key is held down. The ship should not move off the edge of the playing area.
- The space bar shoots a laser.

  It is your choice whether the ship cannot fire again until the current laser is destroyed or if the ship can have multiple lasers flying at once, but whichever you choose must be documented in your readme file. *You will not receive full credit for your game if you don't tell us what behavior to expect!*

## Aliens

- Initially there should be about 20-50 aliens. (3-5 rows, 10 or so columns) How many is a good amound will depend on how large you draw them and how hard you make them to hit.
- Each alien should be drawn as something more complicated than a simple rectangle or oval.
- Aliens move back and forth in unison. When the alien group reaches the edge of the playing area, they all should move down and reverse horizontal direction.
- Aliens shoot missiles at random intervals. (How often they shoot will be one of those game configuration settings you'll have to play with until it feels right.)
- If the aliens reach the ship or the bottom of the playing area, the game ends.
- If all the aliens are killed, start a new level with a full complement of aliens and keep playing.

## Laser and Missiles

- If a laser or missile goes out of bounds, it is destroyed.
- If a laser hits an alien or missile, it is destroyed and so is the object it hit.
- If a missile hits the ship, the missile is destroyed and the ship loses a life.

## Game Play and Scoring

- Game begins when user presses the start/pause button. Text of the button changes to pause. If pause button is pressed, pause the game. When game is paused, aliens should not move, ship should not respond to the keyboard, etc.

- The player starts with some number of lives (generally three). Each time a missile hits the ship, lose a life and reset the ship position to its original starting location.
- Score increases when aliens are destroyed. (10 points each, perhaps? You get to choose how much.)
- When the game is over, let the user know.
- Score and lives should be displayed. Feel free to display additional statistics.

## Extras

Adding extra features can make your game more fun. Just make sure you have the basic game functionality first. Here are a few ideas to get you started.

- Custom background image instead of plain color.
- Sound effects and/or background music.
- Fancy game over notification.
- Add some shields (usually 3-5 of them) between the aliens and the ship.
  - Both lasers and missiles are destroyed when they hit a shield. Both cause some damage to the shield.
  - Part of the shield is destroyed after it is hit. After enough hits, the shield will be completely gone.
  - Shields are not restored when starting a new level.
- Aliens move faster after some of them have been killed.
- Have different types of aliens, with different appearances and point values.
- Add animation. Make the aliens wiggle their legs? Add a flame to propel the missile?
- Add some sort of explosion visualization with aliens are destroyed.
- Only allow the bottommost alien in a column to fire missiles. When bottom alien is destroyed, alien above it will be able to start shooting.
- Have a flying saucer go by above the aliens every so often. Shoot it for bonus points.
- Change behaviour as levels increase
  - Aliens move faster
  - More points per alien
  - Background changes color or image
  - Music changes

  - etc.

- Earn extra lives by clearing a level or shooting a bonus ship.
- Display lives as ship graphics instead of just listing a number.
- Gracefully handling resizing the window, so you can rescale the game while maintaining the aspect ratio.
- Save high scores to a file.
- Easter eggs. (Really important to document those if you want us to find them!)

## Suggestions and Hints

Rather than telling you exactly what to do, I will provide a number of hints that you will hopefully benefit from.

- Split things up into smaller pieces! My sample solution that you have seen in class, consists of roughly 800 lines of code. Among these lines of code I have at least 10 classes defined. Some are nested, some are anonymous, and some are higher level classes. Again, what I'm trying to say, this is not a problem that you can just write in a single method.
- Build off of previous code. Obviously, I expect you to use the GameObjects you wrote for part one, possibly adding additional functionality as you need it. You should be able to use a lot of the work you did for the GUI layout practice to at least get you going on the GUI for the game.
- How to approach the problem... One of the harder things to do when implementing a game like this is to figure out where to start, and what to do first. The first thing you need to understand, is that the game isn't going to write itself, and it's not going to be completely done the first time you sit down to write code for it. So the trick is to work on pieces that you can finish and test, individually first, then putting them together into a usable system. For example – when we are writing the space invaders program, start by just drawing the ship on the screen to make sure it will show up, and then once that is working, figure out how to make it move. (Or, maybe you'd rather start with an alien. That works too.)
- Then what... Well, you hopefully know what you want your game to do, how it will work, and what is going to happen as results of something that you do. How many points should be added for a destroyed alien, etc... I encourage you to sit down and think out your own set of rules and policies for the game. I do not want to impose any specific standard in terms of this for your implementation. But, what I'm saying is that if you have a good idea of what you want your program to do, it's easier coming up with the design for that program on your own. I encourage you to come talk to your TA or to me, about your design before you start writing a lot of code. I also

encourage you to talk about design decisions on the discussion board for the program. Sometimes, it helps venting ideas.

If you come up with a design that you think is reasonable, you will likely do well on this assignment as well – that being said, *pleeease* don't hesitate to ask for help, and to pose questions in class. It will benefit everyone.

- Soooo. . . What kind of stuff do we need in order for this game to work? Partially it's up to you, but I can list a few things that I used and that you may feel are useful to you as well.
  - Instance variables - I have quite a few in order to keep track of the state of the game. Examples are scores, lives remaining, and such. These typically need to be initialized at the beginning of a game.
  - Private helper methods - I have lots of them, these are methods that do small tasks, that you may be performing often, but you don't want to write the code for them over and over again. If you find yourself copying and pasting a lot of code, you should probably be thinking – "Hmmm, I should probably make a method for that!", and then figure out what the method is going to look like, and what parameters it needs, etc.
  - Timer. I use a timer to keep track of how fast the game progresses, but. . . not every object responds to every tick of the timer. You'll have to decide what class(es) should pay attention to the Timer event.
  - KeyListener – Probably one of the more important things for this game which is so keyboard-use heavy. Remember that methods called from listeners should usually not be computation heavy as it may slow down the response time to the next event. By varying the time delay on the timer, we can get the objects to move faster, etc.

    Only the component with *focus* will be able to listen for keyboard events. When you press a button, generally the button retains focus. If you want some other component to get the focus (which you likely will, since I really don't suggest putting your main key listener inside the start/pause button), you can use the `requestFocusInWindow` method to do so. So, if you wanted a component named `myGamePanel` to listen for key events, you'd use `myGamePanel.requestFocusInWindow();` somewhere in your code. Bear in mind that when you click on another component (such a button) that component will gain the keyboard focus.

## Turning in your assignment

For this project, I want all the code and resources to be packaged into a self-contained jar file. I also expect you to turn in a readme document describing your project. *These are the only two files you will submit.*

## Jar File

Create a jar file with all the necessary files that you used for your assignment. The jar file must of course include your source files, as well as code from all packages that you used. I.e., the jar should be self-contained and you should be able to run the game completely from the jar.

To make a jar file with a entry point of the `SpaceInvaders` class:

1. Compile all your classes. (`javac *.java` will compile them if all your source files are in the current directory.)

2. Use the `jar` command to create a jar with all your files. (This should include both your `.java` source files and your compiled `.class` files.) Use the `e` option to specify the entry point.

   ```
   jar cvfe JarFileName.jar EntryPointName <List of files and directorys to include>
   ```

   So, if all your source files and class files are in the current directory, you can use:

   ```
   jar cvfe SpaceInvaders.jar SpaceInvaders *
   ```

   If you are using any images, sounds, or other files like that, make sure you include them in your jar. You want the jar to contain all the files your program needs to run in the single jar.

3. Make sure your program runs from the jar file. Use the `-jar` option with java.

   ```
   java -jar SpaceInvaders.jar
   ```

   To properly test this, you should move your jar file to a new location and try running it there to make sure you are not accidentally running from the files you used to make it instead of the jar itself.

## README file

You have enough freedom with this project that we'll need some documentation. Submit a readme file that explains how to use your program and any special features we should be aware of.

At the very least, your readme should include:

- Game play
  - What keys are used to control the game.
  - How is the game scored.
  - Expected behaviour of lasers (single fire or multi-fire), since you are allowed to choose.
- Description of program internals
  - Description of classes. (Where are game logic, data structures, etc.?)
  - Algorithm details, such as:

      * Moving aliens
      * Detecting/handling collisions
      * Detecting end of game

- Any extras. It is especially important to point out the clever things you do so the grader will know to look for them while testing your program.

- Known bugs and feature requests – I know that no matter how long you have work on this assignment, there will be some bug that you can't quite fix or some feature that you won't quite have time to implement. Tell us about them. What would be your next step?

## Submit to Canvas

Submit your jar file and readme document to UNM Canvas. Make sure that your jar file includes your source code!