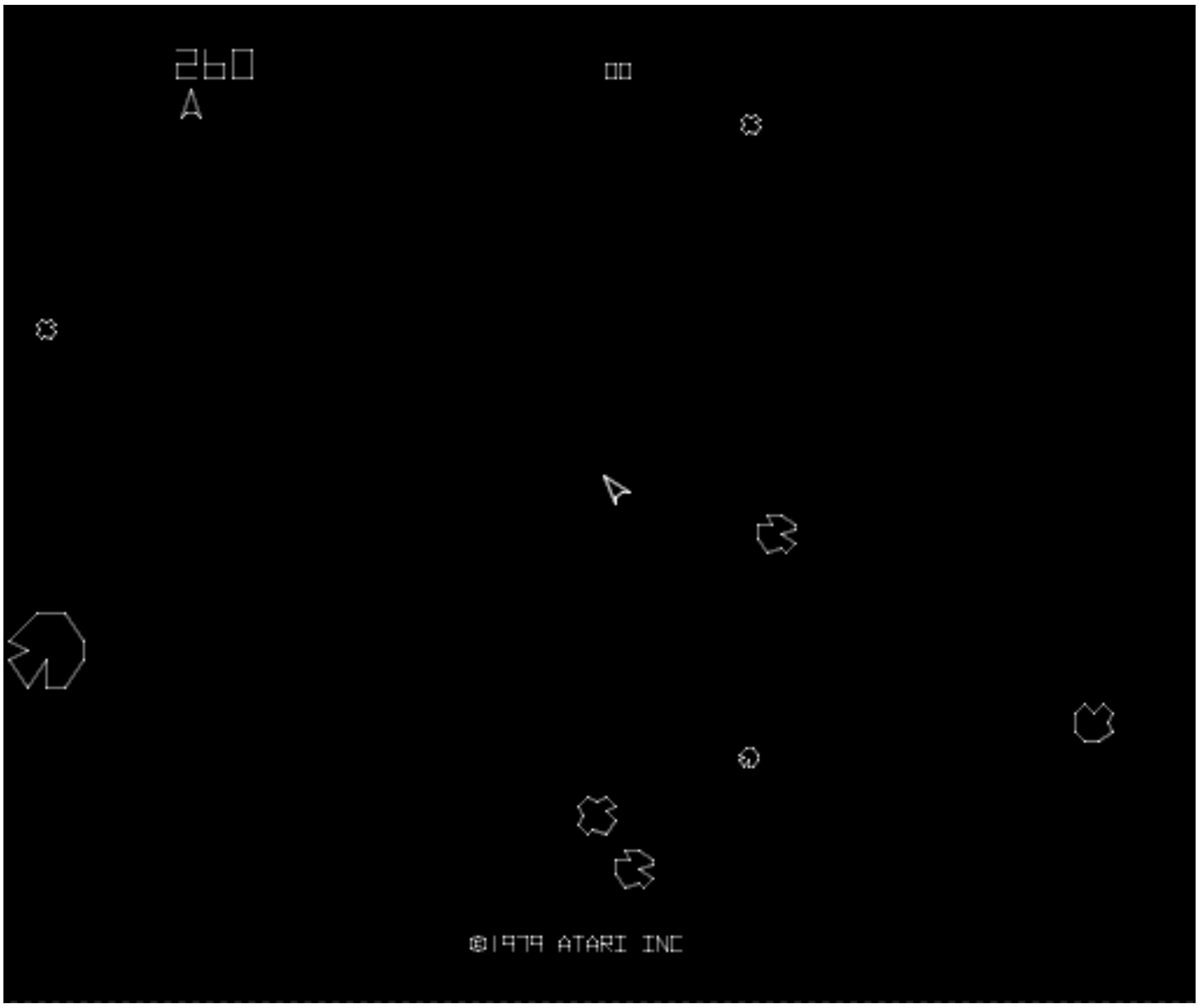
Project 1: C – Arcade Classic – Asteroids

Asteroids was released in 1979 by Atari. The original game used a black and white vector display and featured a lone space ship battling asteroids that threaten to crush it to oblivion. It became one of the early arcade classics (shown below).



This deceptively simple game challenges the player to control a small spacecraft that may rotate and move in the direction it's facing under the influence of a simple physics model (momentum, thrust). Asteroids track across the space and wrap around the edges of the screen. The ship has a laser that it uses to blast the asteroids to pieces. But beware – when hit, the shards of asteroid rock split off in different directions; these further split and get smaller and faster – and harder to hit. This deceptively simple game is surprisingly challenging!

The aim of this project is recreate your version of this classic on UNIX creatively using ASCII character graphics (i.e. on an 80x25 character terminal). The simplest version of the game will feature a ship, an approximation of an asteroid, keyboard controls for thrust and direction and a score. More advanced versions could include the game physics, the simulation of splitting rocks, multiple waves, enemy spaceships, etc.

The exercise focuses on your ability to:

- **Solve the challenge** of creating a simple game (game loop, score etc.);
- Exercise your **problem solving skills**, especially in the simulation of the spaceship, the rocks and their physics;
- **Work out algorithms** to control the ship, simulate the rocks, detect collisions, perform the game logic;
- **Generate a GUI** for the user to interact with the ship (using ASCII art & the UNIX **curses** library)
- Demonstrate good code modularity and style.

Start-up task:

This task is **to be completed in your Week 21 practical** in order to give you a better feel for this project and to give you valuable insight for your design document.

- Visit the asteroids website and play the game a couple of times! http://chrome.atari.com/
- Make notes on the features of the game you can see – pay particular attention to the phases of gameplay (start, middle, end). What happens when you run out of time, rocks, thrust over a screen edge, hyperspace etc.?
- Read the introduction to the ncurses library (sections 1-1.3): http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html
- Look back over term 1 and remind yourself of how to log into 'UNIX' using putty and compile C programs.
- Try the 'hello world' example: http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/helloworld.html - you'll need to `#include <curses.h>` in your C file and compile using '`-lcurses`' with `gcc`.
- Now try to create a C program that uses the curses library to let you control the position of a symbol (e.g. '*') on the screen using keyboard input.
  Tips: lookup `mvprintw`, `getch` and `timeout` in the man pages/online.

Assessment

This work will be assessed through a practical demonstration and code inspection of your work.

Be prepared to show your program in your Week 25 practical session, and to answer questions about it posed by your markers.

To gain marks from this assignment:

- **You MUST submit your code to moodle by the advertised deadline.**
- **You MUST demonstrate your work IN YOUR OWN LAB SESSION.**

FAILURE TO ADHERE TO THE ABOVE MARKS WILL RESULT IN A MARK OF ZERO FOR THIS EXERCISE, OR AT A MINIMUM, THE APPLICATION OF THE UNIVERISTY'S LATE SUBMISSION PENALTY.

Marking Scheme

Your work will be marked based on the following five categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Design Report (week 22)	10%
Functionality (project dependent)	50%
Code Structure and Elegance	20%
Commenting	10%
Ability to Answer Questions on Code	10%

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections describe what is expected to attain each of these grades in each category.

Markers can also recommend the award of a distinction (+) category overall if they feel a piece of work exhibits clearly demonstrable good programming practice.

Marking Guidelines

Marking guidelines provide an idea of the level of achievement expected for a typical grade level. *Projects are always marked on their own individual merit.*

Functionality (Asteroids):

A:	**Working program that meets the criteria for a B, plus:** - Asteroids split when hit into smaller, faster asteroids that

	break off at realistic angles; - More points are awarded differentially for hitting different size asteroids; - Friction (the ship slows after initial thrust); - The player gets more than one ship/turn. - Advanced features, such as one or two of: o Hyperspace (the ship can jump to a random location not on an asteroid); o Progressing 'rounds' with increasing difficulty (more asteroids); o Others, e.g. splash screen, high score table, enemy space ship.
B:	**Working program that meets the criteria for a C, plus:** - Asteroids and the ship can move off the screen and appear (wrap) around the edges; - Asteroids appear at random and move (basic physics); - The user can 'fire' across the screen - Asteroids can be shot and disappear when hit. - score is awarded appropriately and displayed.
C:	**Working program that meets the criteria for a D, plus:** - At least one asteroid present on the screen - The ship can move (thrust) as well as rotate - Basic collision detection (the ship can be destroyed if it hits as asteroid)
D:	**Working program that:** - shows a representation of the space ship. - The player can rotate the ship using the keyboard.
F:	**No working program demonstrated, or program does not meet any requirements listed above.**

Common Criteria

Code Structure and Elegance:
A: Well written, clearly structured code showing student’s own examples of good OO practice.
B: Well written, clearly structured code.
C: Clearly identifiable but **occasional** weakness, such as repetitive code that could be removed through use of a loop, poor use of public/private, unnecessary / unused code, inappropriate naming and scoping of variables.
D: Clearly identifiable **systematic** weakness, such as multiple examples of repetitive code that could be removed through use of a loop, systematically poor use of public/private, large sections of unnecessary / unused code, consistently inappropriately named and scoped variables.
F: All the above.

Commenting:
A: All blocks of code and methods concisely commented.
B: Code is well commented, but **occasionally** vague and/or inaccurate.

C: Code is partially commented **or systematically** vague and/or inaccurate.
D: Code is partially commented **and systematically** vague and/or inaccurate.
F: No Comments provided

Ability to Answer Questions on Code:
A: All questions answered in detail.
B: Student failed to explain one technical question about their code.
C: Student failed to explain two technical questions about their code.
D: Student failed to answer questions, but could provide a basic overview of their code.
F: No evidence that the student understands the code being demonstrated.

Star Categories:
By showing a demonstrable example of additional work and/or good programming practice, your marker may choose to award a star (*) grade to your work. If you feel your work shows additional merit beyond the specification, it is your responsibility to make your marker aware of this during your in lab marking session.