# Asteroids

SOFT7019 C Programming Assignment 2

In this assignment you will implement a computer player for an "asteroids" game. The game is a side-scroller game where there is a space-ship at the left side and there are obstacles (asteroids) coming in from the right.

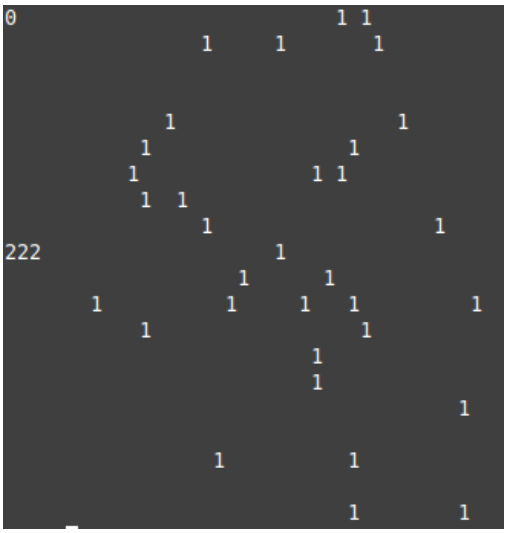


Figure 1: Asteroids game. The asteroids are '1's and the ship is '2's

## Mechanism

- the game board is represented as a 2D matrix of integers
- the matrix contains the following values
  - a 1 represents an asteroid
  - a 2 represents the ship; the ship is made up of a few connected 2s
  - 0s represent empty spaces
- an example is shown in the Figure above
- the asteroids are moving from right to left, towards the ship
- new asteroids are generated probabilistically
- the ship stays on the left edge of the matrix (column 0)
- the ship can move up and down *one row at a time*
- the ship must avoid asteroids; if it crashes into an asteroid it's game over
  - crashing into an asteroid happens when a 1 corresponding to an asteroid moves into one of the matrix positions that contains a 2, corresponding to a ship
- you must implement a function that decides the next action of the ship
  - the actions are move up one row, move down one row, or stay in place
- the game runs step by step; every step of the game consists of
  - updating the asteroid field: moving existing asteroids one square to the left, generating new ones
    - this function checks for collision between the ship and asteroids
  - getting the next action of the ship; this function must be implemented by the student

  - update the ship's position based on the ship's selected action
  - display the screen
- the score of the game is the number of steps that the ship manages to stay alive without colliding with asteroids.

# Requirements

You have to implement a function that decides the next action of the ship.

- the action should depend on the position of the asteroids and that of the ship
- the goal is to avoid asteroids for the longest amount of game steps.

## Function description

- function parameters
  - matrix representing the game
  - state of the algorithm: this is a memory address `(void *)` that allows your solution to maintain state between function calls, for example the current position of the ship
    * using the state correctly will grant you extra points
    * if you use the state, the address MUST be allocated in the function and returned (see below)
    * the first time the function is called the state address will be passed as 0
    * in subsequent calls it is set to the address returned on the previous call
    * note that you can store anything at that address; it can be a basic variable type (`int`) or a complex type (`struct`, array, array of structs, etc) that would allow storing more information, as needed by your algorithm.
- the matrix
  - 2D matrix of integers, `FIELD_HEIGHT` rows and `FIELD_WIDTH` columns
  - the asteroids are marked as 1s
  - the matrix also contains the spaceship, marked as 2s
  - empty spaces are 0s
- the function returns a `struct ship_action` with
  - `int action`: 1 for going down, -1 for going up
  - `void *state`: memory address that can be used for storing state of the algorithm; MUST be set to zero if not used.

## Ship actions

Three types of actions are recommended. They are in increasing order of effectiveness, as well as increasing implementation effort and achievable grade:

- random or fixed action: action does not consider asteroid position
- greedy action: move the ship to the lane where the first asteroid is farthest
- planned action: consider the possibility of dead-ends, tunnels, etc.

# Running your code

You are provided with a template code that generates the matrix of asteroids and updates it at each step, displays the game screen and checks for collisions. The code contains

- `asteroids.c` – this contains the main code with functionality described above
- `asteroids.h` – this contains `struct` definitions as well as constants
- `move_ship.c` – this will contain the function to move the ship that you must implement.

In this assignment you must modify `move_ship.c` with the code for your function.

Notes:

- the code you are given should compile and work properly in `onlinegdb`
- to compile the code in command line, if you use `gcc`:
  - `gcc asteroids.c move_ship.c -lcurses -o asteroids`
- the code makes use of a Unix library called `curses` that provides functions for displaying basic graphics in a terminal
- the code will not work in Windows machines because they do not have the `curses` library
  - in Windows you can try cygwin: `https://www.cygwin.com/index.html`
- if need be I can modify the code so that it compiles on Windows, however this will lose the display functions.

### Submission

Please only submit the `move_ship.c` file.

## Programming competition

This assignment will also run as a programming competition. Sometime in the next weeks I will provide an update with a competition portal where you will be able to upload your function and have it evaluated.

There will be a leaderboard with the scores for the submissions.

Marks for the solutions are up to 30 points. The top five submissions will be rewarded with the last 5 points up to 35 (1st place will get 5 points, 2nd 4 points, etc).

On the competition portal you will have an unlimited number of attempts, until the competition ends.

## Marking rubric

- random or fixed movement: 1
- code compiles: 2
- returning values: 4
  - only action, state not used and set to zero: 1
    - only action, state not used and not set: 0
  - state used and allocated correctly: 3
    - state used but not allocated correctly: 1
- parsing of the matrix: 4
  - basic, without identifying obstacles: 2
  - identifying the position of the ship: 1
  - identifying the obstacles: 1
- greedy path planning: 5
  - the ship travels in a straight line as long as the current line has the most space up to the first obstacle
  - when the line above or below has more space we change lanes
  - it is greedy so it can end up in a tunnel
  - also may have collisions when switching lanes
- changing vertical position of the ship: 4
  - avoiding collisions when moving above/below
  - understanding that the obstacles will move one point to the left on the next frame
- planning a path based on obstacle location: 10
  - the ship avoids obstacles, changing lanes in available spaces
  - path is recomputed at each frame
  - ship follows path
- top 5 positions in the competition: up to 5 points.