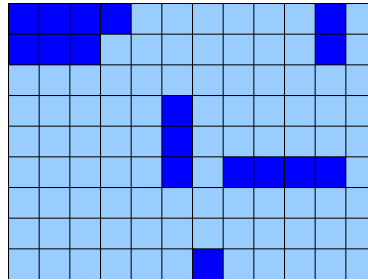


# BattleShip

Programming Lab Semester Project

**Introduction:** the goal of this project is to implement a battleship game as depicted in the figure bellow. The game should consider two players, a given set of game pieces, i.e., ships, a map with  $n*n$  cells, and it must follow a turn-based game play, i.e., players take turns.



## Objectives

- a) *Implement the game locally (single machine), where both players share the same shell interface. Max grade: 12/20*
  - 1) The initialization of the game must support **both a randomize setup and manual selection**. In the first, the game pieces must be placed randomly across the map (orientation must also be random). In the manual setup, each player must deploy each game piece through a simple command line interface.
  - 2) Each game piece must be backed by a **user defined data structure**, and the map should be a data structure that references these game pieces.
  - 3) Algorithmic efficiency on insertion, lookup and deletion will be accounted for your final grade.
- b) *Implement the game state using a quad-tree data structure. Max grade: 18/20*
  - a) Algorithmic efficiency on insertion, lookup and deletion will be accounted for your final grade.
- c) *Implement the game locally (single machine) where players do not share the same shell interface. The two shell interfaces must be connected through an inter-process communication mechanism:*
  - 1) Use system v semaphores and **text files** to synchronize both clients *Max grade: 19/20*
  - 2) Implement another version that uses inter-process communication (IPC) such as **pipes**. The buffers used for performing IPC must be controlled by memory(s) poll(s), i.e., a set of pre-allocated memory buffers normally with a fixed size. The buffers should use a zero-copy approach, where they should be passed from layer to layer without copying (only by reference). *Max grade: 20/20*
- d) **Bonus:**
  - 1) *Implement the game across the network (two machines), using TCP/IP sockets. Max grade: 2/20*
    - 1) One machine can act as the server (no needed for true peer-to-peer design).
    - 2) Use threads to separate network, logical and interface code executions.
  - 2) *Implement a graphical interface using SDL, QT or similar. Max grade: 2/20*