COMP-361 Project
Battleships v1.2

Battleships is a turn-based strategy game for two players. It is inspired by the classic battleships board game, but has been considerably enhanced and adapted to be played on a computer and over the net. However, the main idea remains the same: each player commands a fleet of ships and tries to sink the enemy fleet. The player who sinks the entire enemy fleet wins the game.

1 The Sea

Figure 1 shows the battlefield, that is a square ocean area (30 x 30 squares). In the middle of the field there is a rectangular (10 x 24 squares) area, 10% of which (that is 24 squares) is filled with coral reefs. The placement of the coral reefs is determined at random at the beginning of every game. For fairness reasons, the players should agree on a reef configuration before starting the game.

On each side of the map is a player base of size 10 x 1. Initially, the ships of a player are docked at the players base (i.e. with the stern (the back) touching the base and the bow (the front) facing the enemy base). Before the game starts, the player can rearrange the position of his ships at the base as long as they remain docked.

During the game, the ships can navigate over the ocean to hunt down enemy ships or attack the opponents base. However, ships can not move onto squares that contain coral reefs, the player base, other ships, or mines.

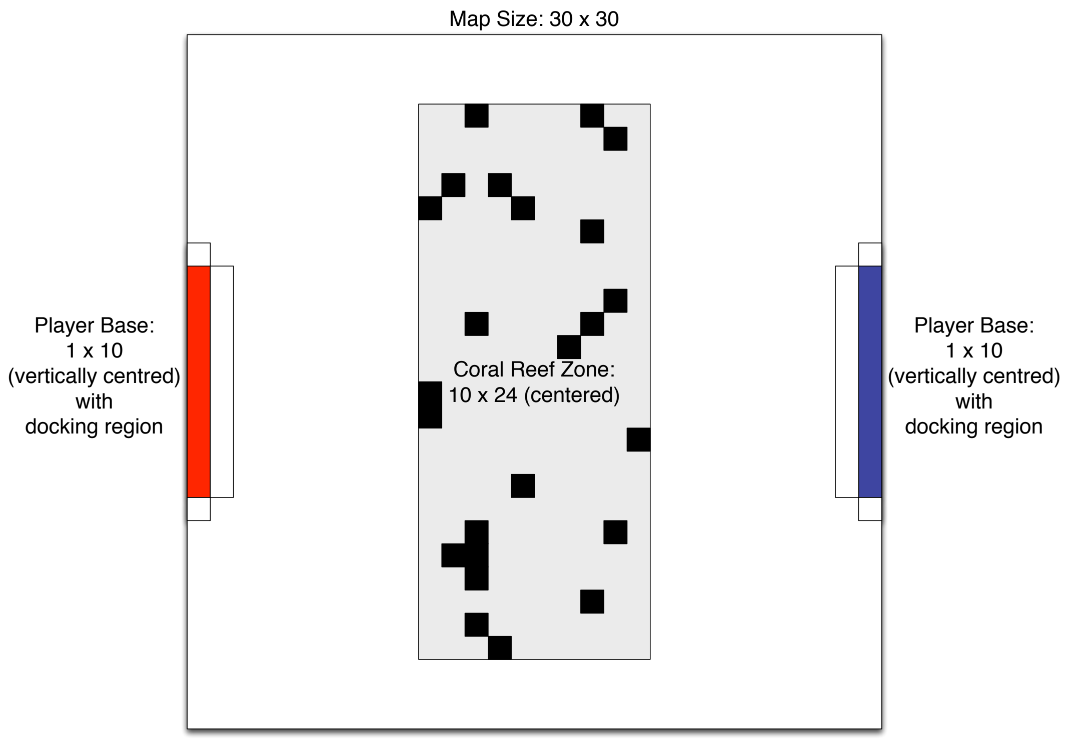


Figure 1: The Battlefield

2 Fleet

Each player commands a fleet of 10 ships. Each ship is equipped with a radar that allows it to “see” enemy ships at a distance. The fleet is composed of:

- Two **cruisers**, strong ships, equipped with long-range cannons.
- Three **destroyers**, medium sized-ships, armed with cannons and torpedoes.
- Two **torpedo boats**, small and mobile ships, equipped with a short range cannon and torpedoes.
- Two **mine layers**, tiny, heavily armoured ships, equipped with a short range cannon. Each boat also carries 5 mines, that it can deposit on any water square. Once dropped, these mines will inflict major damage to any bypassing ship. Mine layers are also equipped with a sonar, which allows them to detect, and potentially even remove enemy mines.
- One **radar boat**, equipped with a short range cannon and a long-range radar.

3 Visibility

Both players play on the same board, but the fun part of the game lies in the fact that they can only “see” that parts of the map that lie within the radar range of their ships. The (fixed) position of the players bases and the coral reefs is known to both players, and therefore always visible.

So, at any given time, a player sees his base, the enemy base, the coral reefs, his own ships, and any parts of enemy ships that are in the radar range of his ships or the base. Enemy ships that are beyond the range of a players radar remain invisible. Deployed mines are not visible by radar, only the mine layers (which are equipped with a sonar) can see them.

4 Ships

Every ship has a certain set of features:

- **Size**:
  Every ship has a given size, that is it occupies a certain number of squares on the map. Visually, it is possible to identify the bow (the front) of the ship.
- **Speed**:
  Every ship moves at a certain speed, that is it can advance up to a certain number of squares at each turn in the direction it is facing. A ship can also move backwards one square, or shift one square sidewise. Finally, ships can rotate 90 degrees to the left or to the right. They rotate around the stern (their last square), except for torpedo boats. Torpedo boats are very mobile: they can turn 180 degrees in one move, and rotate around their centre square.
  A ship can only move onto empty squares. If a ship tries to move onto a square occupied by some other ship, a reef or a base, then the movement is stopped. No damage is done, but the collision (which square of the obstacle was hit) is signalled to both players. When rotating, the area used for the rotation must not contain any obstacles, otherwise the rotation is cancelled, and the collision square is signalled to both players.
  If a ship (other than a mine layer) hits a mine while moving, it explodes and damages the moving ship (see exact rules below). Mine layers on the other hand detect mines with their sonar, and therefore will stop before hitting a mine, just like for any other obstacle.
- **Armour**:
  In order to sink an enemy ship, a player has to destroy all its squares. Squares of ships with normal armour are destroyed by a single hit. Squares of heavily armoured ships are damaged by a first hit, and only the second hit destroys them. (Heavy cannons are an exception, as they can destroy even heavily armoured squares with one hit.) In order to sink a ship, all its squares must be destroyed. When a ship sinks, it is removed from the map, and the players are notified.
  When a ship is damaged, its speed is decreased according to the percentage of destroyed squares. For example, a destroyer (initial speed 8) that has 3 destroyed squares (out of 4) has a remaining speed of 2! All other functionality (shifting sidewise, rotating, radar / sonar, cannons, torpedoes and mines) is not affected. A cruiser (initially speed 10) with 2 destroyed squares, has a remaining speed of 6. A torpedo boat, initially speed 9, with one destroyed square, has a remaining speed of 6.

A damaged ship can be repaired at the players base (if the base has not been destroyed!). In order to perform repairs, at least one square of the ship must "touch" the base (i.e., be located inside the docking region of the base). Each repair turn repairs one destroyed square (regardless if it is heavily armoured or not), starting from the bow of the ship and moving towards the stern.

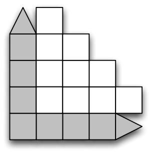
- **Weapons**:

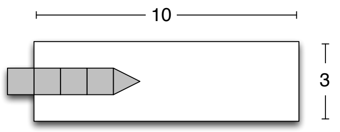
  - **Cannons**:
    Cannons are artillery weapons that can fire in any direction up to a certain range. The shot flies in a parabola; it is therefore possible to overshoot any obstacle, such as coral reefs, or other ships. If a cannon hits an unarmored ship square or a square of a base, it is destroyed. Armoured squares are destroyed by heavy cannons with a single hit, normal cannons however only damage an armoured square on the first hit. The second hit destroys it. If a cannon hits an already destroyed square or a coral reef, it has no effect. If a cannon hits a square with a mine, the mine is destroyed (without exploding). In any case, both players are notified of the coordinates of the impact of a shot.
  - **Torpedoes**:
    Torpedoes are underwater weapons that are fired in a straight line in the direction the ship is facing. Their range is limited to 10 squares. They explode whenever they encounter an obstacle on their way. If a torpedo hits a ship, the square of the impact is hit (that is, destroyed or damaged). If the torpedo hits the ship from the side, an additional square close to the impact is hit (if there is a remaining intact one close to the impact square). Reefs are indestructible. If a torpedo hits a mine, the mine and the torpedo are destroyed. A torpedo hit is signalled to both players. If the torpedo does not hit any obstacle, it disappears.
  - **Mines**:
    Mines are explosive devices placed in water to destroy ships that approach. Unfortunately, mines can not distinguish between enemy and allied vessels, so any ship (except mine layers as explained below) approaching it triggers the explosion. Concretely, any other ship that tries to rotate over a square with a mine, or whose squares touch a square with a mine during movement (i.e., end up located immediately north, south, east or west of a mine), triggers an explosion. As a result, the movement of the ship is stopped (whenever it touches the mine), or remains where it was in case of a rotation, and the square of the ship that has been hit is destroyed (regardless if it is armoured or not). An additional square (towards the bow, or else towards the stern of the ship) is destroyed as well. The mine disappears in the explosion.

- **Radar / Sonar**
  Every ship has a radar that allows it to detect enemy ships that are within range. The radar is sophisticated enough to distinguish the bow of a ship from the other ship squares, and detects if squares are destroyed, damaged or still intact. If only parts of a ship are within radar range, then only these parts are displayed. The radar can also see behind obstacles such as reefs or ships. The only thing a radar can not detect is mines. Only mine layers, who are equipped with a radar as well as a sonar, can see mines. The region made visible by a radar is different for each ship (see below).

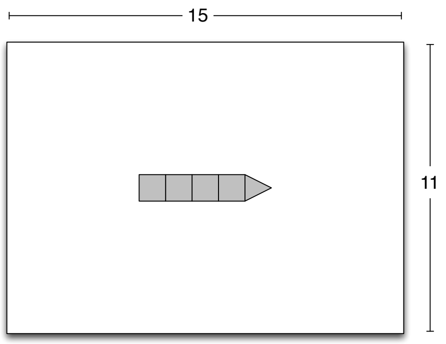
4.1 Ship Details

- Cruiser: Size: 5, Speed: 10, Armour: Heavy, Weapons: Heavy Cannon


Turn Zone

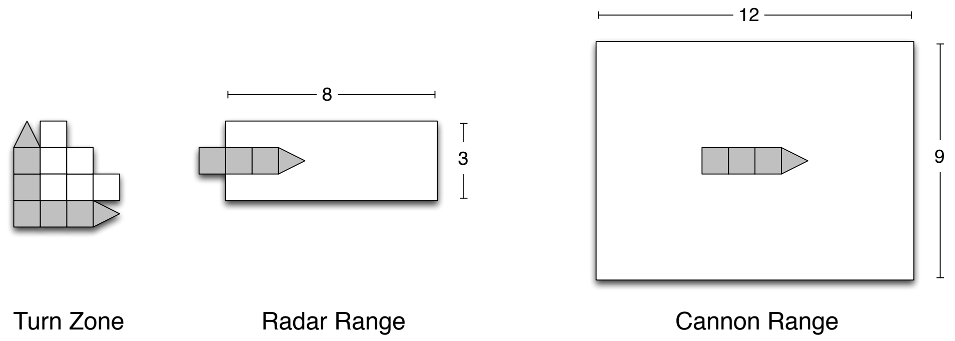


Radar Range

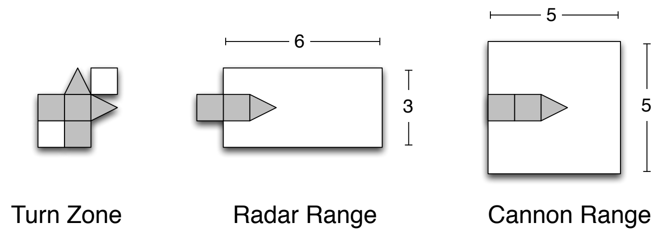


Cannon Range

- Destroyer: Size: 4, Speed: 8, Armour: Normal, Weapons: Cannon, Torpedoes

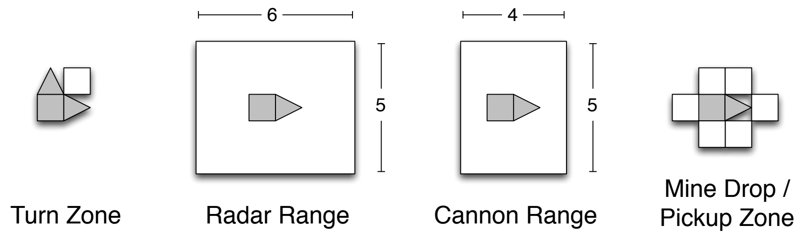


- Torpedo Boat: Size: 3, Speed: 9, Armour: Normal, Weapons: Cannon, Torpedoes



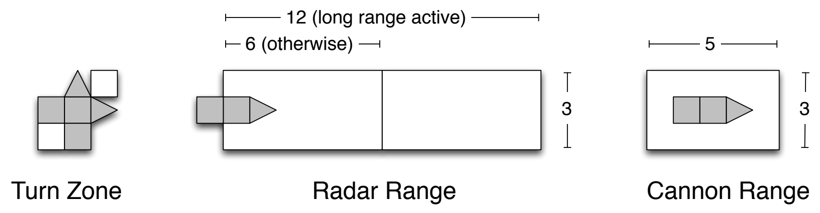
  The torpedo boat is very agile. It can turn 180 degrees in one turn.

- Mine Layer: Size: 2, Speed: 6, Armour: Heavy, Weapons: Cannon, Mine



  Mine layers can drop and pickup mines (your own mines, but also the mines of the enemy). A mine layer can, therefore, pickup an enemy mine and drop it somewhere else, without the enemy knowing it! In the beginning, each mine layer carries 5 mines.
  Mines can be dropped on (or picked up from) any square that lies within the drop / pickup zone. It is, however, not allowed to drop a mine on a square where it would be in immediate contact with some other ship, reef or base.
  Only mine layers can approach a mine without triggering an explosion. If a mine layer tries to move over a square with a mine, its movement is stopped before reaching the square.

- Radar Boat: Size: 3, Speed: 3, Armour: Normal, Weapons: Cannon



  The radar boat is a slow moving boat (speed 3), with only a very short range defensive cannon. It has a long range radar (12 squares) that it can turn on and off. Unfortunately, when the long range radar is active, the radar boat is very vulnerable, since it can not move anymore (it can still turn, however). Just like the torpedo boat, the radar boat can turn 180 degrees in one turn.

5 Base

The players base is a 10 x 1 square building. Initially, all the ships of a player are docked at the players base, facing the enemy base. The base also has a short range radar that covers 1 square around the base.

Damaged ships that are docked at the base (that is, at least one square of the ship occupies the docking region) can be repaired. Each repair turn repairs one destroyed square, starting from the front of the ship and moving towards the back. The squares of a base can be destroyed, just like the squares of a ship. If all the squares of a base are destroyed, it can not repair ships anymore. It is not possible to repair the base itself.

6 Global Visibility

Any missile use, even if it occurs outside the range of the radars of a players ships, is signalled to the player. The information transmitted for each impact is:

- **Location**: the coordinates of the impact.
- **Cause**: what kind of weapon exploded (cannon, torpedo, or mine). The ship that fired the shot, and from what direction remains secret.
- **Effect**: the player is notified of the effect of the explosion. This could be "nothing / shot fell into the water", "mine exploded", "ship hit", "base hit", "ship sunk" or "base destroyed". If a ship is hit, the type of ship remains a secret. Only when a ship sinks, the type of the ship is communicated to the opponent player. If an already destroyed square is hit again, no additional damage is done. However, the shooting player will still get a "ship hit" notification. Torpedo shots that encounter no obstacle are not signalled to the opponent.

The state of all squares of the opponents base are also globally visible.

7 Game Play

At each turn, a player can perform one move out of the following:

- Move a ship
  A ship can move back one square, shift sidewise one square, advance up to as many squares as its speed allows, or turn 90 degrees to the left or to the right (torpedo boats can turn 180 degrees in one move). Of course, visible (or invisible) obstacles can prevent a movement from happening. If a boat advances into an area that is not visible (or tries to turn over an area that is not visible) and there is an obstacle in this area, then the movement is stopped, and the colliding square is signalled to both players. In case of an obstacle blocking a rotation, the turning ship remains in the initial position, and the colliding square is signalled to both players.
  Mines are obstacles that interrupt a movement just as any other obstacle. In addition, they explode and destroy the square of the ship that has collided with the mine. An additional square (towards the front, or else towards the back of the ship) is destroyed as well. The mine disappears in the explosion.
- Shoot with a cannon
  Any ship can use its cannon to shoot a square that lies within range. Coral reefs can not be destroyed. Base squares, and ship squares with normal armour are destroyed with one shot. Heavily armoured squares need two normal cannon shots to be destroyed. Only the heavy cannon of the cruiser can destroy heavily armoured squares with one shot.
- Shoot a torpedo
  Any torpedo equipped ship can shoot a torpedo in the direction it is facing.
- Drop a mine
  Mine layers that still carry mines can drop them on any adjacent square, provided that it would not immediately explode because of an adjacent ship or reef.
- Pickup a mine
  Mine layers can pickup mines on adjacent squares.
- Repair a ship
  If a damaged ship has docked at the players base (that is, at least one square is located in the base's docking zone), then one destroyed square can be repaired at a time, provided that the base has at least one remaining intact square.

8 End of the Game

The game ends once a player has sunk all ships of the opponent. It is not enough to just destroy the opponents base. However, destroying the base is an important step to victory, since it prevents the enemy from repairing his ships.

Additional Rule Clarifications

Additional rule clarifications, if any, are going to be posted on the course webpage during the semester at: http://www.cs.mcgill.ca/~joerg/SEL/COMP-361_FAQ.html.

9 Application to Develop

The rules above describe the game play, but they do not prescribe computer-specific functionality, such as the user interface. In your implementation you have to design your own user interface. Make it as intuitive as possible; provide guidance for the player; provide context-sensitive help; voice messaging; speech recognition, 5.1 channel sound, "the sky is the limit", ...

You are also allowed to provide additional game features such as additional game rules, new units, structures, new winning conditions, etc... However, 80% of your grade is solely based on whether or not you implemented the above game rules as specified. Additionally, certain computer-specific functionality is required as discussed in the following sections.

9.1 User Interface Assistance

The software must help a player to control his ships in an efficient way. This includes, for example, displaying the movement range of a ship when selected, allowing the player to evaluate if it makes sense to move the ship or not. Likewise, highlighting the cannon range of a ship without forcing the player to shoot allows him/her to evaluate if it makes sense to use the weapon or not.

9.2 Game Setup Phase

The actual game should only start once the players have agreed that the randomly generated battlefield is acceptable to both. Also, the players should be allowed to rearrange the initial positions of their fleet before the game begins.

9.3 Distribution Requirements and Game Server

The application must make it possible for (at least) two players to play against each other over a network. Multi-player games (e.g. more than two players), with possible alliances (e.g. 2 against 2), are optional.

To facilitate players to setup a game, you should implement a game server that allows players to connect with other players that are currently online. The server keeps score of how many games each player has played, and how many games she has won in total. After logging in, a player should see the list of players that are currently online together with their game statistics.

Somehow, two (or more) players that want to play a game together should be able to agree on a battlefield, setup their starting positions, and then play their moves turn by turn until one player wins. When a game is completed, the server should update the players statistics.

9.4 Saving

The application must allow any player to save a game. Later, it should be possible to reload a previously saved game, and continue playing from there.

10 Project Milestones

The project is to be done in groups composed of maximally 5 students. Groups of less than 3 students are discouraged. All members of a group will get the same final grade for the project. The final grade of the project is composed of:

- 5% for the user interface sketch (mid October)
- 15% for the requirements document (late November)

- 15% for the design document (early January)
- 15% for the demo (March)
- 25% for the acceptance test (April)

Details for each required hand-in are going to be announced separately, but below you'll find a brief summary of each milestone.

As announced in class, you can use any implementation platform you like, provided that the programming language is object-oriented. However, we will provide technical support for students using Java. Also, we developed a Java-based game programming framework called Minueto (http://minueto.cs.mcgill.ca/), that takes care of efficient rendering of graphics (off-screen drawing, transparency, hardware acceleration, etc....). That way, students who do not know much about Java graphics will not have to waste time learning these concepts, but can focus on the more important functional part of the application. Minueto also does some basic user input handling and provides sound output.

Finally, please also keep in mind that the demonstrations and the acceptance test have to take place either in the Trottier building, or somewhere else on the McGill campus (i.e., you can, for instance, bring your laptop / desktop computers to my office and set up the demo there, or run the game on your PDA's, bring your gameboys / XBoxes / iPhones to school, etc....).

10.1 User Interface Sketch

Good user interfaces should prohibit the user from making "mistakes", such as unauthorized or useless moves. An intuitive user interface should assist players whenever possible, for instance by highlighting possible destination fields for a selected ship. Also, a good user interface should allow a player to quickly find important information, such as what enemy ships have already been destroyed, or where the last weapon impact occurred.

For this hand-in you should prepare a sketch (either hand drawn or a printout) of the main screens of the game. It should show how the player sees the world, how she controls her ships, and how she accesses other functions such as saving the game.

10.2 Requirements Document

This hand-in will comprise requirements models that clearly specify the functionalities that your game implementation needs to provide. This includes a use case model, a concept model, an environment model, and an operation model.

10.3 Design Document

This hand-in will comprise design models that provide a detailed blueprint of the structure (i.e. classes) and behaviour (i.e. methods) of your application that relate to the implementation of the game and its rules. (The design models will not cover classes and methods that are needed to handle graphics and network communication).

10.4 Demo

After spring break you will be asked to give a first demonstration of your product. You will be given a list of minimal functionality that you must show during the demo. 80% of the grade for the demo is based on the correct implementation of those required functionalities, and that you can demonstrate them without visible errors / bugs / crashes. The remaining 20% are attributed as follows: 10% for the quality of the demo (i.e. presentation / explanation of what is happening), and 10% for additional functionality of the game that you can demonstrate, but that was not required for the demo.

During the demo, the group of graders will not touch the computers, nor interfere with the demo in any way. YOU run the show. Demonstrate what your software can do (and do not show what it can not do). Explain what is happening during the demo, i.e. how you set-up the communication between the two players, what functionality you are demoing, etc. Show off your cool features, and do not talk about the bad ones / remaining bugs. Any unforeseen event that hurts the "demo effect" might affect your grade.

10.5 Maintenance Phase

In the week of the demo, I will hand out some slight changes / additions to the game rules. This simulates "real-life" software development, in which the application requirements are often subject to change during the development of an application. In order to prepare for this phase, try to come up with a flexible design / write structured, modular, extensible code.

10.6 Acceptance Test

In the last week of classes your application will have to pass the “acceptance test”. During the test, the group of graders will play your game, looking for bugs / glitches and violations of the game rules. We run the show, and you basically just watch.

80% of the grade for the acceptance tests are based on the correct implementation of the game rules (i.e. if your application fulfills all requirements listed in this document, and if there are no errors / bugs / crashes during the acceptance test, you will get at least an A- (80)). The remaining 20% are attributed for an elegant / intuitive user interface, or other extra features. For instance, you could get extra points by having a 3D user interface, or by allowing more than 2 players to play a game, or by allowing a neutral “observer” to connect to a running game, or by adding another kind of ship, or by allowing players to choose the ships they want to play with during the setup phase, etc.

In order to prepare for the acceptance test, please keep in mind that it takes a long time to “debug” an application of this size. Even under the assumption that you do extensive unit testing during development, I suggest to plan at least 2 weeks for final adjustments.

11 Some Suggestions

Here’s a list of some simple suggestions you should keep in mind, in particular during the implementation phase:

- Keep everyone in the group in ”a good mood”.
- Assign responsibilities to group members.
- Have regular group meetings to consolidate your work.
- Start implementation early, i.e. January at the latest.
- First strive for a simple, correct implementation. Later (if there is enough time), add more sophistication.
- Always keep the deadlines in mind. For the demo you must have a functional, convincing application that provides the required minimal functionality.
- Do not make big changes on the day that precedes the demo or the acceptance test (or else be sure to have a functional backup copy...)
- Testing takes time.
- Plan for the unpredictable!
- Start implementation early :)

12 Change Log

- v 1.1: Added the radar boat.
- v 1.2: Added requirements for game server (see section 9.3)