Battleship

The game of Battleship is one that probably doesn't require an introduction. Two teams each with 5 ships. The players place their ships on the board, hiding their locations from their enemies. Then each player takes turns firing at their opponent's hidden ships. If you strike a hit you then know where a ship is and you can keep firing at it until it is sunk. The first player to locate and sink off of their enemies ships wins.

In this version you play against a computer opponent. But don't worry, the computer never peeks.

Battleship is written by Joseph Larson based on a board game by Milton Bradley.

BATTLESHIP.C | You will need: a C/C++ complier .

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <math.h>

#define PL 0
#define CP 1
#define FULL(x) ((x + (x < 2)) + 1)
#define ADD(a,b) if (!bd[a][b].chit) {t[i].x = a; t[i].y = b; i++;}

typedef struct {
  unsigned int x : 4;
  unsigned int y : 4;
} COORD;

typedef struct {
  unsigned int pship : 3;
  unsigned int chit : 1;
  unsigned int cship : 3;
  unsigned int phit :3;
} GRID;

GRID bd[10][10];
COORD t[51];
char ship_life[2][5];
char *ship_name[5] =
  {"pt ship", "submarine", "cruser", "battleship", "carrier"};

COORD getloc (void) {
  char input[10];
  COORD loc;

  loc.x = loc.y = input[0] = 0;
  do {
    if (input[0])
      printf ("Invalad location, letter first then number : ");
    scanf ("%s", input);
    if (isalpha (input[0]) && (loc.x = atoi (&input[1]))) {
      loc.y = tolower (input[0]) - 'a';
      if (loc.y > 9 || loc.x > 10 || loc.x < 0) loc.x = 0;
    }
  } while (!loc.x);
  loc.x --;
  return loc;
}

void show_board (void) {
  int x, y;

  printf
    ("%16s\t\t%16s\n_ 1 2 3 4 5 6 7 8 9 10\t\t_ 1 2 3 4 5 6 7 8 9 10"
    ,"R A D A R","F L E E T");
  for (y = 0; y < 10; y++) {
    printf ("\n%c ", y + 'a');
    for (x = 0; x < 10; x++)
      printf ("%c ", (bd[x][y].phit)?(bd[x][y].cship) ? 'X' : 'o' : '.');
    printf ("\t\t%c ", y + 'a');
    for (x = 0; x < 10; x++)
      printf ("%c ", (bd[x][y].chit) ? (bd[x][y].pship) ? 'X'
      : 'o' : ".12345"[bd[x][y].pship]);
  }
  for (y = 4; y >= 0; y--) {
    printf ("\n %10s : ", ship_name[y]);
    if (ship_life[CP][y]) for(x = 0; x < FULL(y); x++) putchar ('#');
    else printf ("SUNK");
    printf ("\t\t %10s : ", ship_name[y]);
    for (x = 0; x < FULL(y); x++) putchar (".#"[ship_life[PL][y] > x]);
  }
}

int valad_ship (COORD s, COORD e, int c) {
  int check, d, v;
  COORD step;

  check = abs ((s.x + 10 * s.y) - (e.x + 10 * e.y));
  if (check % (FULL(c) - 1)) {
    printf ("\nInvalad location. The %s is only %d long\n"
            "and ships can only be placed vertical or horizontal.\n",
            ship_name[c], FULL(c));
    v = 0;
  } else {
```

Listing continued next column...

BATTLESHIP.C | Listing continued from previous column...

```
    step.x = step.y = 0;
    if ((check / (FULL(c) - 1)) - 1) step.y = 1;
    else step.x = 1;
    if (s.x > e.x) s.x = e.x;
    if (s.y > e.y) s.y = e.y;
    for (d = 0; d < FULL(c) && v; d++) {
      check = bd[s.x + d * step.x][s.y + d * step.y].pship;
      if (check && check != 7) {
        printf ("\nInvalad location. Ships can not overlap.\n");
        v = 0;
      }
    }
  }
  if (v) for (d = 0; d < FULL(c); d++)
    bd[s.x + d * step.x][s.y + d * step.y].pship = c + 1;
  return v;
}

void player_setup (void) {
  int ship;
  COORD start, end;

  for (ship = 4; ship >= 0; ship--)
    do {
      show_board ();
      printf ("\nEnter start location for your %s : ", ship_name[ship]);
      start = getloc();
      printf ("Enter end location (length %d) : ", FULL(ship));
      end = getloc();
    } while (!valad_ship (start, end, ship));
  show_board ();
}

void auto_setup (int pl) {
  COORD s, step;
  int c, d;

  for (c = 0; c < 5; c++) {
    do {
      s.x = rand() % 10; s.y = rand() % 10;
      step.x = step.y = 0;
      if (rand() < RAND_MAX / 2) {
        step.x = 1;
        if (s.x + FULL(c) > 10) s.x -= FULL(c);
      } else {
        step.y = 1;
        if (s.y + FULL(c) > 10) s.y -= FULL(c);
      }
      for (d = 0; d < FULL(c) &&
      (pl) ? !bd[s.x + d * step.x][s.y + d * step.y].cship
      : !bd[s.x + d * step.x][s.y + d * step.y].pship ; d++);
    } while (d < FULL(c));
    for (d = 0; d < FULL(c); d++)
      if (pl)
        bd[s.x + d * step.x][s.y + d * step.y].cship = c + 1;
      else bd[s.x + d * step.x][s.y + d * step.y].pship = c + 1;
  }
}

void init (void) {
  int c, d;
  char input;

  srand (time (NULL));
  for (c = 0; c < 10; c++)
    for (d = 0; d < 10; d++)
      bd[c][d].pship=bd[c][d].chit=bd[c][d].cship=bd[c][d].phit=0;
  for (c = 0; c < 5; c++)
    ship_life[PL][c] = ship_life[CP][c] = FULL(c);
  printf
    ("Battleship (R)\n\nDo you want (A)uto or (M)anual setup ? (a/m) ");
  while (!isalpha (input = getchar()));
  if (tolower (input) == 'm')
    player_setup ();
  else auto_setup (PL);
  auto_setup (CP);
}

int check_for_lose (int player) {
  int c;
```

Listing continued on page 2...

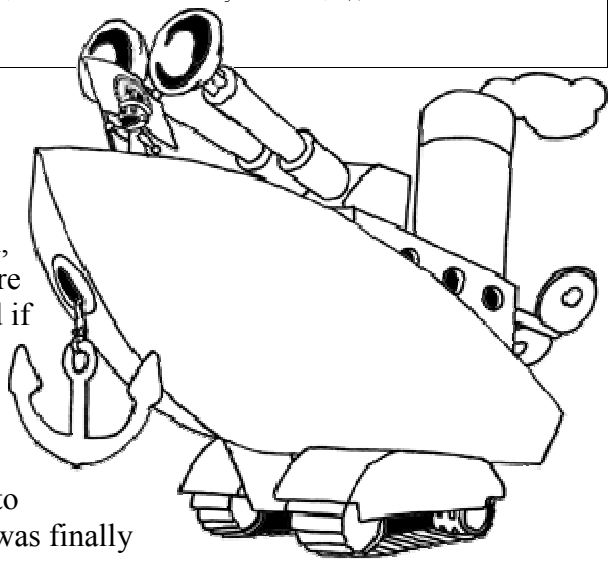
BATTLESHIP.C | Listing continued from page 1...

```
  for (c = 0; c < 5 && !ship_life[player][c]; c++);
  return (c == 5);
}

void player_turn (void) {
  COORD shot;
  int ship;

  show_board ();
  printf ("\n\nYour shot coordinates : ");
  shot = getloc ();
  if (bd[shot.x][shot.y].phit)
    printf ("A wasted shot! You already fired there!\n");
  else {
    bd[shot.x][shot.y].phit = 1;
    ship = bd[shot.x][shot.y].cship;
    if (ship) {
      printf ("HIT!\n");
      if (!(--ship_life[CP][--ship]))
        printf ("You sunk my %s.\n",ship_name[ship]);
    } else printf ("Miss.\n");
  }
}

int hit_no_sink (int x, int y) {
  if (bd[x][y].chit) {
    if (bd[x][y].pship == 7) {
      return 1;
    } else if ((bd[x][y].pship) && (ship_life[PL][bd[x][y].pship - 1]))
      return 1;
  }
  return 0;
}

int fill_t (void) {
  COORD c, d;
  int m[5] = {0, 1, 0, -1, 0};
  int x, i = 0;

  for (c.x = 0; c.x < 10; c.x++)
    for (c.y = 0; c.y < 10; c.y++)
      if (hit_no_sink (c.x,c.y)) {
        for (x = 0; x < 4; x++)
          if (c.x + m[x] >= 0 && c.x + m[x] < 10
            && c.y + m[x + 1] >= 0 && c.y + m[x + 1] < 10) {
            if (hit_no_sink (c.x + m[x], c.y + m[x + 1])) {
              d.x = c.x; d.y = c.y;
              while (d.x >= 0 && d.x < 10 && d.y >= 0 && d.y < 10
                && hit_no_sink (d.x, d.y))
                  {d.x -= m[x]; d.y -= m[x + 1];}
              if (d.x >= 0 && d.x < 10 && d.y >= 0 && d.y < 10)
                ADD (d.x, d.y);
            }
          }
        if (!i)
          for (x = 0; x < 4; x++)
            if (c.x + m[x] >= 0 && c.x + m[x] < 10
              && c.y + m[x + 1] >= 0 && c.y + m[x + 1] < 10)
              ADD (c.x + m[x], c.y + m[x + 1]);
```

Listing continued next column...

BATTLESHIP.C | Listing continued from previous column...

```
      }
  if (!i)
    for (c.x = 0; c.x < 10; c.x++)
      for (c.y = 0; c.y < 10; c.y++)
        if ((c.x + c.y) % 2) ADD (c.x, c.y);
  return i;
}

void compy_turn (void) {
  int z, c;

  c = fill_t ();
  z = rand () % c;
  printf ("\nMy shot : %c%d\n", t[z].y + 'a', t[z].x + 1);
  bd[t[z].x][t[z].y].chit = 1;
  c = bd[t[z].x][t[z].y].pship;
  if (c) {
    printf ("HIT!\n");
    if (!(--ship_life[PL][c - 1]))
      printf ("I sunk your %s.\n", ship_name[c - 1]);
  } else printf ("Miss.\n");
}

void play (void) {
  int winner = 0;

  if (rand () < RAND_MAX / 2) {
    printf ("\nYou go first.\n");
    player_turn ();
  } else printf ("\nI'll go first.\n");
  do {
    compy_turn ();
    if (check_for_lose (PL)) {
      winner = 1;
      printf ("\nI win!\n");
    } else {
      player_turn ();
      if (check_for_lose (CP)) {
        winner = 1;
        printf ("\nYou win!\n");
      }
    }
  } while (!winner);
  show_board ();
}

int play_again (void) {
  char input;

  printf ("\nDo you wish to play again? (y/n) ");
  while (!isalpha (input = getchar()));
  if (tolower (input) != 'n') return 1;
  else return 0;
}

int main (void) {
  init ();
  do {play ();} while (play_again ());
  printf ("\nIt's been fun! So long Admiral!\n");
  exit (0);
}
```



Author's Notes:

The computer picks it's targets in this game from a list of what it has determined are good spaces. If there's no good shots its possible shots are every other square, no point in wasting shots. If it got a hit its possible shots are the 4 to the north, south, east and west of the hit. If it's on the trail, having found 2 or more hits in a row the possible shots are on either end of the trail. And if either end of the trail that it's on are dead ends and it still hasn't sunk the ship clearly it's on the trail of 2 ships that are side by side and it's possible shots are the other directions.

This procedure is known to anyone who's played the game. The hard part was figuring out what order the computer needed to populate the list in. Check out the function fill_t() to see how it was finally done.

In the end the computer doesn't play by any rules that person could, but playing the flawlessly it tends to have the edge. You've really got to be on your toes to beat the computer, or be really lucky.