Programming Project: Game
CPSC 298-6 Programming in C++
jbonang@chapman.edu

Introduction

In this programming project, you'll implement a computer game called "Battle Boat" as a C++ class. A framework for the game has been implemented for you but is incomplete. You'll need to finish the implementation.

"Battle Boat" is similar to the classic game "Battleship" but much simpler.


Battleship game played by crewmembers of USS George H.W. Bush

The opponent has only one boat, albeit a stealthy, autonomous, robotic boat, that is hidden in an ocean represented by a two-dimensional grid. The game is a command line game and all output is rendered in ASCII (American Standard Code for Information Interchange) characters. For example, the grid is shown below, with the tilde character ~ representing waves. Each grid square is flanked by a vertical bar character to help the user see the squares.

You control another boat with a magazine of torpedoes and must hunt the Battle Boat.

You fire a torpedo at the battle boat by selecting a grid square as your aim point. For example, in the sequence below the user enters grid location 2, 2, indicating row 2 and column 2 in the grid as the aim point.

```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
2
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|*|~|~|
|~|~|~|~|~|
Location row: 2; column: 2 is a miss.
Magazine: 9 torpedoes
```

An asterisk, indicating a torpedo explosion is rendered at the aim point. The game reports the torpedo missed its target, the wily Battle Boat.

If the torpedo comes close to the battle boat, where close means one of the eight grid squares surrounding the Battle Boat, the game reports the torpedo was "close." For example, if the Battle Boat is hidden in location 2, 2 and the user fires the torpedo at grid location 2, 3, the game reports row 2 and column 3 is close to the Battle Boat.

```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
3
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|*|~|
|~|~|~|~|~|
Location row: 2; column: 3 is close!
Magazine: 9 torpedoes
```

The eight squares surrounding 2, 2 are shown numbered in the figure below.

```
|~|~|~|~|~|
|~|1|2|3|~|
|~|4|V|5|~|
|~|6|7|8|~|
```

Your boat has a limited magazine capacity; the number of torpedoes remaining is shown after each turn.

If the torpedo aim point grid coordinates correspond exactly with the coordinates of the Battle Boat, the Battle Boat is hit and sunk and you win the game.

```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
2
|~|~|~|~|~|
|~|~|~|~|~|
|~|*|X|*|~|
|~|~|~|~|~|
Hit!  Kaboom!
Victory!
```

The game reports the hit and the explosion and announces victory.

If, on the other hand, all torpedoes are expended without hitting the Battle Boat, the game ends (with a victory for the Battle Boat and a loss for the player).

```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
1
Please enter grid column number:
1
|~|~|~|~|~|
|~|*|*|~|*|
|~|~|*|*|*|
|*|*|*|*|~|
Location row: 1; column: 1 is close!
Magazine: 0 torpedoes
|V|~|~|~|~|
|~|*|*|~|*|
|~|~|*|*|*|
|*|*|*|*|~|
Torpedoes expended
Game over
```

The hidden location of the Battle Boat, represented as a 'V' character is displayed to the player.

The locations of all the torpedo explosions, represented by asterisk characters, remain displayed until the end of the game. This allows the user to remember the grid squares they've targeted previously as they hunt the Battle Boat.

The game consists of a sequence of turns where each turn involves the player firing a torpedo and the game reporting the results. The sequence of turns continues until either the Battle Boat is sunk or until all torpedoes have been expended.

When a game ends, the player is prompted as to whether they wish to play another game.

```
Play another game? (y/n)
```

If they answer yes (y), the game is reset and a new sequence of turns begins.

The section below describes the (partial) implementation of the game program. It provides very helpful information on certain details that may make the assignment easier for you. However, reading it is not essential to completing the assignment and you can skip ahead to the assignment section if you'd prefer.

Implementation

Though Battle Boat is a simple game, it's implementation resembles that of large commercial computer games. For example, Battle Boat uses a main game loop in which the user undertakes some action (firing a torpedo in this case) and the game assesses the results of that action - is it a hit, a close miss or a complete miss; has the player run out of torpedoes?

The Game Class

Battle Boat is implemented as a single C++ class called Game. The class is declared in a header file, Game.h and implemented in an implementation file, Game.cpp. There is also a main function in source file main.cpp.

The game grid is maintained in a two-dimensional character array, which is a member variable of the Game class. The grid is rendered by a displayGrid method which adds vertical bars around the grid elements to improve clarity.

Class Constants

The class Game includes a number of constants that give the default size of the grid (4 by 5), and the default number of torpedoes (equal to the number of grid squares divided by 2).

```
class Game
{
    static const int k_nGridRowsDefault = 4;                  ///< Default number of Rows in the 2-D Game Grid
    static const int k_nGridColumnsDefault = 5;               ///< Default number of Columns in the 2-D Game Grid
    static const int k_nMaxTorpedoes = (k_nGridRowsDefault * k_nGridColumnsDefault) / 2; ///< Starting number of torpedoes
```

Member Variables

The number of rows and columns doesn't deviate from the default in this implementation of the game and is maintained by member variables m_nGridRows and m_nGridColums, respectively.

```
int m_nGridRows;                                    ///< Number of rows in Game Grid; initialized to default
int m_nGridColumns;                                 ///< Number of columns in Game Grid; initialized to default

int m_iRowBoat;                                     ///< Row number target boat positioned in
int m_iColumnBoat;                                  ///< Column number target boat positioned in

int m_nTorpedoes;                                   ///< Number of torpedoes remaning in magazine
```

Similarly, the location within the two-dimensional grid of the stealthy Battle Boat is maintained in member variables m_iRowBoat and m_iColumnBoat. Another variable, m_nTorpedoes, stores the number of torpedoes remaining.

Public Member Functions

The Game class exposes only two public member functions, the constructor, Game and play.

```
public:
    /// Default constructor for Game object.
    /// @details calls initializeGrid to initialize 2-dimensional game grid. Initializes member variables in
    ///          initializer list.
    Game();

    /// Game Loop; plays a complete game.
    /// @return void
    /// @details positions target boat randomly, allows users to launch topedoes, reports results.
    ///          Continues looping until target hit (victory) or all torpedoes expended (game over, loss).
    void play();
```

Private Member Functions

The play member function contains the main game loop, in which all activity occurs. play calls on several private member functions to carry out all of its actions.

```
private:
    /// Initialize two dimensinal playing grid to contain all "wave" characters (~).
    /// @return void
    void initializeGrid();

    /// Reset the Game object for a new game.
    /// @return void
    /// @details places the Battle Boat at a random location in the grid, resets the
    ///          number of torbepoes to the maximum number of torpedoes and
    ///          re-initializes the grid to hold all wave characters ('~').
    void reset();

    /// Displays the game grid to standard output using ASCII characters; displays or hides the
    /// position of the target boat based on the value of the Boolean parameter bDisplayBoat.
    /// @param[in] bDisplayBoat Boolean value that if true indicates the position of the target boat
    ///            is to be displayed in its grid position using a 'V' character. Otherwise, the target
    ///            boat's position is rendered as a "wave" character, ~.
    /// @return void
    void displayGrid(bool bDisplayBoat);

    /// Queries the user for the grid location (row,column) at which to fire a torpedo;
    /// updates the game grid with torpedo location and updates the number of torpedoes
    /// remaining.
    /// @return tuple object containing two integer values the first of which is the
    ///         row selected by the user and the second is the column selected by the user.
    /// @details Assigns a torpedo explosion character, '*', to the grid location that the
    ///          torpedo targets within the a_cGrid character grid member variable. Also
    ///          decrements the number of torpedoes by 1.
    std::tuple<int, int>  fireTorpedo();

    /// Places the target boat at a randomly chosen grid location.
    /// @return void
    /// @details updates the m_iRowBoat and int m_iColumnBoat member variables.
    void placeBoatRandomly();
```

```
/// Places the target boat at a grid location chosen by the user; used for
/// testing as it allows the target boat to be positioned at a predetermined
/// grid location.
/// @return void
/// @details updates the m_iRowBoat and int m_iColumnBoat member variables.
void placeBoat();

/// Returns true if the grid location specified by parameters iRow and iColumn is
/// identical to the location of the target boat.
/// @param[in] iRow integer row number of a grid location targeted
/// @param[in] iColumn integer column number of a grid location targeted
/// @return true if the grid location specified by arguments iRow and iColumn is
/// identical to the location of the target boat and false otherwise.
bool isOnTarget(int iRow, int iColumn);

/// Returns true if the grid location specified by parameters iRow and iColumn
/// is adjacent to the grid location where the target boat is located.
/// @param[in] iRow integer row number of a grid location targeted
/// @param[in] iColumn integer column number of a grid location targeted
/// @return true if the grid location specified by parameters iRow and iColumn
///         is adjacent to the actual position of the target boat. An adjacent
///         grid location is any of the eight grid squares surrounding the
///         location of the target boat:
///         |1|2|3|
///         |4|V|5|
///         |6|7|8|
///         Returns false if the grid location specified by parameters iRow and
///         iColumn is not adjacent to the grid location where the target boat
///         is located.
bool isCloseToTarget(int iRow, int iColumn);

/// Renders a sunk character ('X') in the grid location where the target boat
/// is located.
/// @return void
void showSunk();

/// Queries the user for a grid location (row,column); utility function used in
/// other functions.
/// @return tuple object containing two integer values the first of which is the
///         row selected by the user and the second is the column selected by the user.
std::tuple<int, int> promptForGridCoord();
```

The private functions include one that is used only for test purposes, placeBoat. In actual games, placeBoatRandomly is used. However, for testing, you may want to place the BattleBoat in a specific location; placeBoat lets you do that.

Conditionally-compiled Code

placeBoat is conditionally compiled when the macro TEST is defined:

```
void Game::reset()
{
    this->initializeGrid();
#if defined TEST
    // Place the target boat at a specified location for testing/diagnostic purposes.
    // This code is executed only if the TEST macro is defined (#define TEST or -DTEST).
    // If the macro is defined, the program is in "test mode."
    this->placeBoat();
#else
    // The target boat is placed at a random location on the grid; this is used for
    // normal gameplay.
    this->placeBoatRandomly();
#endif
    this->m_nTorpedoes = Game::k_nMaxTorpedoes;
}
```

To define the macro TEST, insert the following line near the top of your C++ implementation file, Game.cpp:

```
#define TEST  // macro to enable test mode functionality (not used during ordinary gameplay)
```

There is also a DIAG macro which may be defined to activate diagnostic output.

For example, member function placeBoatRandomly will display the random grid location of the Battle Boat if DIAG is defined.

```
#if defined(DIAG)
    std::cout << "DIAG: Boat location (Row/Column): " << this->m_iRowBoat << "; " << this->m_iColumnBoat << std::endl;
#endif
```

To define the macro DIAG, insert the following line near the top of your C++ implementation file, Game.cpp:

```
#define DIAG  // macro to enable diagnostic output (not used during ordinary gameplay)
```

Both lines are presented, but commented out in Game.cpp, so you actually only need to uncomment these lines to activate them.

```
//#define TEST  // macro to enable test mode functionality (not used during ordinary gameplay)
//#define DIAG  // macro to enable diagnostic output (not used during ordinary gameplay)
```

std::tuple Template Class

The Game class's implementation makes use of a C++ Standard Template Library (STL) template class called std::tuple, which is similar to the std::vector class you've already encountered. The sd::tuple class holds a collection of items, which may be of different types.

In the Game class, the tuple template class is used to store a set of two integers, corresponding to the row and column numbers of a grid location.

Like std::vector, tuple is a template class and the data types of the two elements it stores must be specified in angle brackets as shown. In the case shown below, the two values are initialized in the tuple's constructor to iRow and iColumn, two variables declared and assigned to earlier.

```
// Return the coordinates of the torpedo explosion location as a tuple object
std::tuple<int, int> tupleTorpedoCoord(iRow, iColumn);
return tupleTorpedoCoord;
```

The contents of the tuple can be extracted using the std::tie function, which ties the content of the tuple to other variables.

For instance, member function fireTorpedo returns a tuple containing two integers, the row and column number of the location where a torpedo has been fired. The std::tie function receives this tuple (via the assignment statement) and "ties" the two integer values to local variables iRow and iColumn, which have been previously declared (int iRow; int iColumn;).

```
std::tie(iRow, iColumn) = this->fireTorpedo();
```

The tuple template class makes it easy to pass an ordered pair of values representing grid coordinates around.

The main Function

The game program's main function declares a single instance of the Game class and later calls it's public member function play to execute the game loop.

```
int main()
{
    bool bPlay = true;
    Game game;
```

The main function calls the play public member function of the Game object, `game` in a while loop which repeats so long as the user wishes to play another game.

The Assignment

The Battle Boat game is implemented in three source files, main.cpp Game.h and Game.cpp. These files are presented in incomplete versions that you must finish. Instructions for finishing the game are give in TODO comments spread throughout the code.

The TODO comments appear similar to the following:

```
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\ //
// TODO: WRITE WHILE LOOP TO ASK IF USER WISHES TO PLAY ANOTHER GAME
//
// Within the while loop, hve the Game object, game, call the play() member function.
// Then prompt the user if they want to play another game. If so, iterate through the
// while loop again, otherwise, set bPlay to false (so that the while loop will cease iterating).
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\ //
```

You may leave these comments in after you've implemented the functionality (it makes it easier on the instructor to find your changes).

Diagnostic Mode

Once you've implemented all the changes and have a complete, working game, place the game in diagnostic mode by defining the DIAG macro. You need only uncomment the line shown in Game.cpp to define the macro.

```
//#define DIAG  // macro to enable diagnostic output (not used during ordinary gameplay)
```

Once uncommented, as shown, diagnostic output code will be enabled.

```
#define DIAG  // macro to enable diagnostic output (not used during ordinary gameplay)
```

In particular, the following line in placeBoatRandomly will be compiled in

```
#if defined(DIAG)
    std::cout << "DIAG: Boat location (Row/Column): " << this->m_iRowBoat << "; " << this->m_iColumnBoat << std::endl;
#endif
```

This causes the location of the Battle Boat, normally kept secret, to be displayed at the beginning of the program.

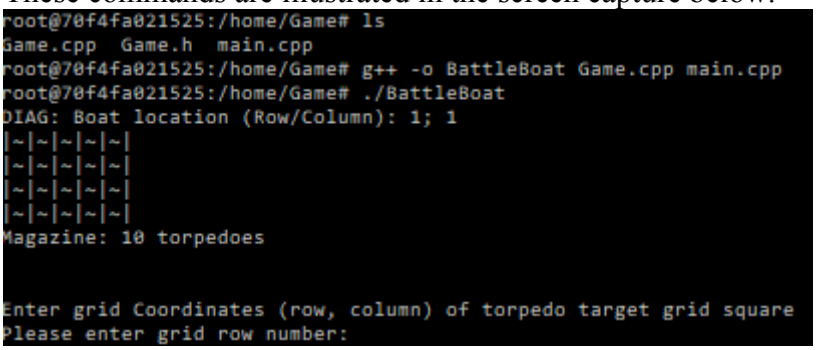
```
DIAG: Boat location (Row/Column): 3; 0
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
Magazine: 10 torpedoes


Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
```

Building
To compile and link the Battle Boat game under Linux, execute the following g++ command:

```
g++ -o BattleBoat Game.cpp main.cpp
```

To execute type `./BattleBoat`

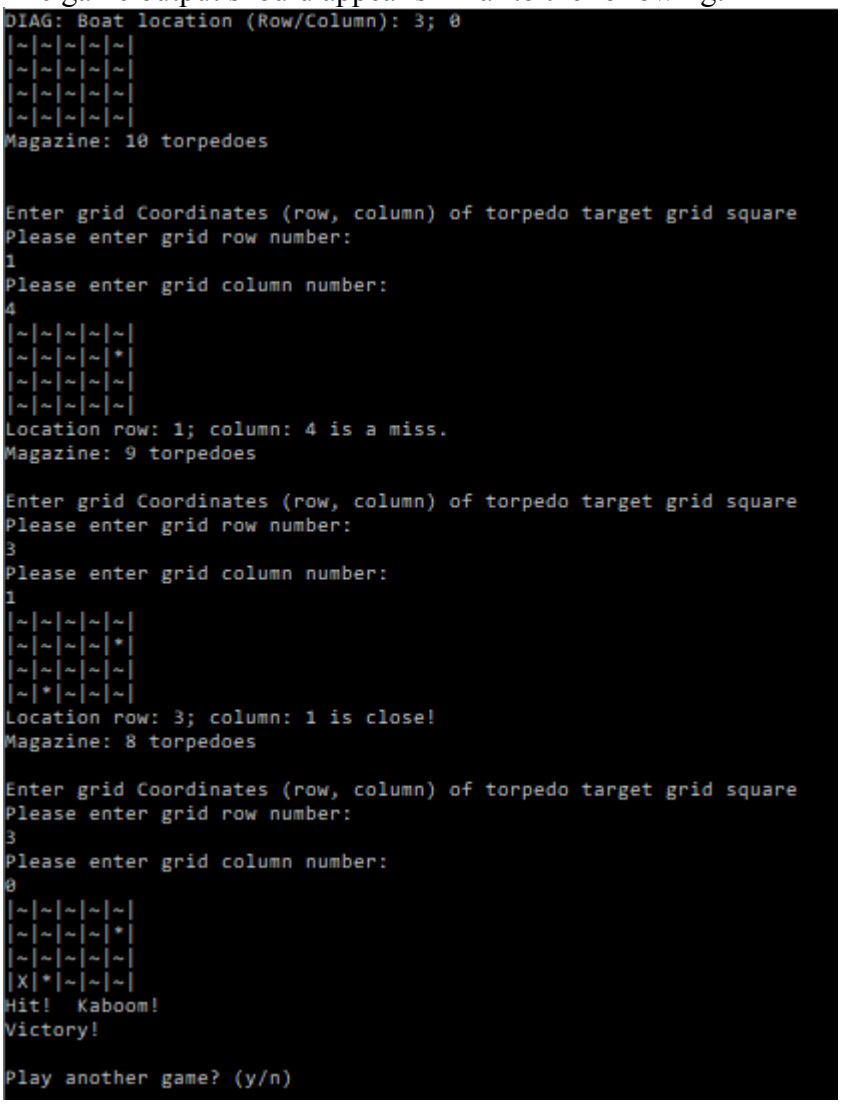
These commands are illustrated in the screen capture below.



Execute Two Games

Using this location given in the diagnostic ouput, execute a game where you have at least one miss, one "close" and then sink the Battle Boat.

The game output should appear similar to the following:



```
DIAG: Boat location (Row/Column): 3; 0
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
Magazine: 10 torpedoes


Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
1
Please enter grid column number:
4
|~|~|~|~|~|
|~|~|~|~|*|
|~|~|~|~|~|
|~|~|~|~|~|
Location row: 1; column: 4 is a miss.
Magazine: 9 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
3
Please enter grid column number:
1
|~|~|~|~|~|
|~|~|~|~|*|
|~|~|~|~|~|
|~|*|~|~|~|
Location row: 3; column: 1 is close!
Magazine: 8 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
3
Please enter grid column number:
0
|~|~|~|~|~|
|~|~|~|~|*|
|~|~|~|~|~|
|X|*|~|~|~|
Hit!  Kaboom!
Victory!

Play another game? (y/n)
```

Capture this output and submit it with your source code.

When prompted to "Play another game?", enter y and execute a game where the player does not sink the Battle Boat. The player must expend all torpedoes and not hit the Battle Boat, either with a "miss" or a "close."

```
Play another game? (y/n)
y
DIAG: Boat location (Row/Column): 0; 4
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
Magazine: 10 torpedoes


Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
3
Please enter grid column number:
4
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|~|
|~|~|~|~|*|
Location row: 3; column: 4 is a miss.
Magazine: 9 torpedoes
```

Beginning of Game

```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
0
Please enter grid column number:
3
|~|~|*|*|~|
|~|*|~|*|~|
|~|*|~|*|~|
|*|*|*|~|*|
Location row: 0; column: 3 is close!
Magazine: 0 torpedoes
|~|~|*|*|V|
|~|*|~|*|~|
|~|*|~|*|~|
|*|*|*|~|*|
Torpedoes expended
Game over

Play another game? (y/n)
```

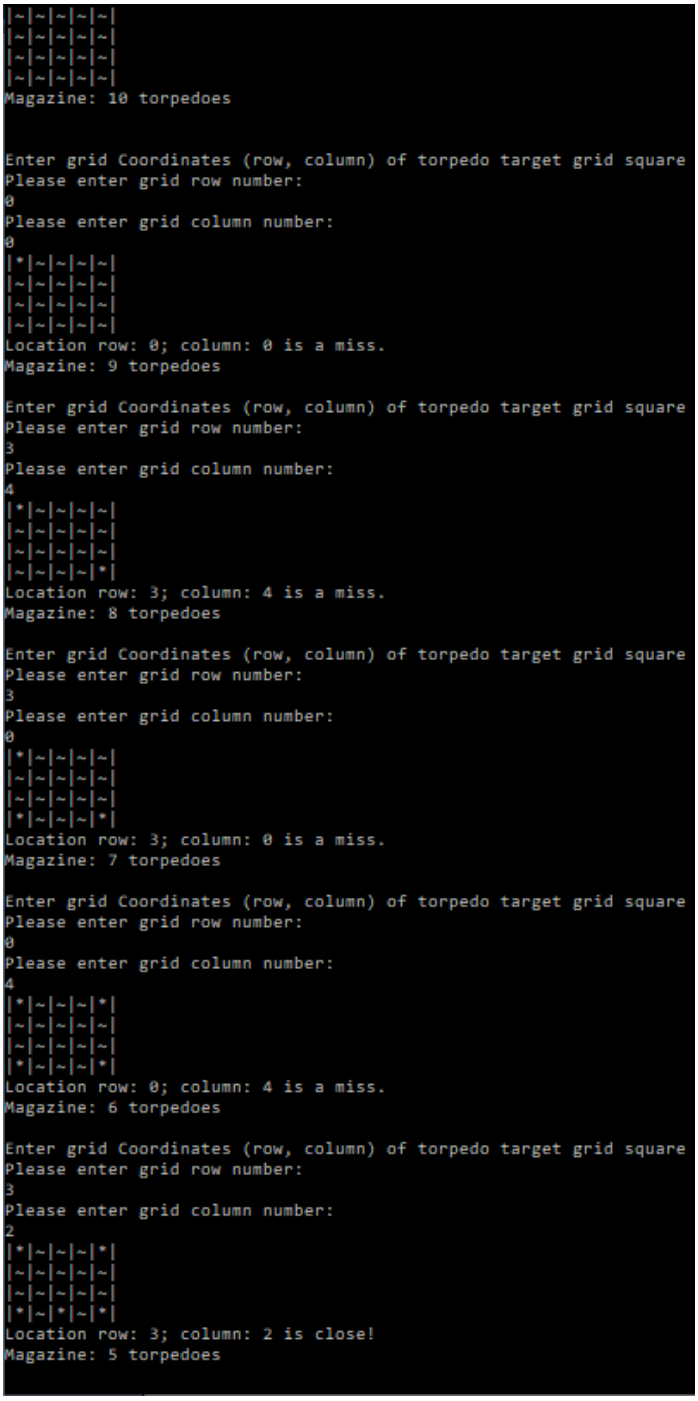
End of Game

Submit screen captures of the beginning and end of the game that results in missing the Battle Boat.

Appendix A shows a complete game where the Battle Boat is not sunk.

Appendix A: Complete Game Output Resulting in Missing the Battle Boat

The game output for a game resulting a loss is shown in the following screen captures.



```
Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
1
Please enter grid column number:
4
|*|~|~|~|*|
|~|~|~|~|*|
|~|~|~|~|~|
|*|~|*|~|*|
Location row: 1; column: 4 is a miss.
Magazine: 4 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
1
Please enter grid column number:
0
|*|~|~|~|*|
|*|~|~|~|*|
|~|~|~|~|~|
|*|~|*|~|*|
Location row: 1; column: 0 is a miss.
Magazine: 3 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
1
|*|~|~|~|*|
|*|~|~|~|*|
|~|*|~|~|~|
|*|~|*|~|*|
Location row: 2; column: 1 is close!
Magazine: 2 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
4
|*|~|~|~|*|
|*|~|~|~|*|
|~|*|~|~|*|
|*|~|*|~|*|
Location row: 2; column: 4 is a miss.
Magazine: 1 torpedoes

Enter grid Coordinates (row, column) of torpedo target grid square
Please enter grid row number:
2
Please enter grid column number:
3
|*|~|~|~|*|
|*|~|~|~|*|
|~|*|~|*|*|
|*|~|*|~|*|
Location row: 2; column: 3 is close!
Magazine: 0 torpedoes
|*|~|~|~|*|
|*|~|~|~|*|
|~|*|V|*|*|
|*|~|*|~|*|
Torpedoes expended
Game over
```