C++ Code Design and Implementation

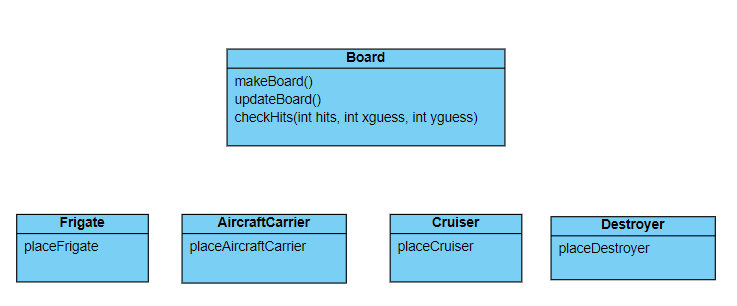
# Design:

The main design of the board and the console remains the same as it did from the python application:

A screenshot of a computer screen

Description automatically generated

As there would be a wide use of classes the following UML depicts these to ensure that every class and method is coded.



# Implementation:

Steps for programming:

1. Create a board class.
2. Create board methods to display the board, update the board, and check for any hits or misses.
3. Allow the user to select a difficulty level.
4. Create boat classes.
5. Create instances of each boat class
6. Allows the user to input guesses.
7. Validate user inputs.
8. Calculate a win or a loss.

OOP Pseudocode:

A screenshot of a computer program

Description automatically generated

# First Iteration of Code:

|  |
| --- |
| #include <algorithm>  #include <cstdlib>  #include <ctime>  #include <iostream>  #include <vector>  using namespace std;  vector<vector<int>> UsedTiles;  class board {  public:  vector<vector<string>> T;  board() : T(10, vector<string>(10, "|\_\_")) {}  void makeBoard() {  cout << " 0 1 2 3 4 5 6 7 8 9" << endl;  for (int i = 0; i < T.size(); ++i) {  for (int j = 0; j < T[i].size(); ++j) {  cout << T[i][j] << ' ';  }  cout << endl;  }  }  void checkHits(int hits, int xguess, int yguess) {  vector<vector<int>> guessed = {{xguess, yguess}};  if (any\_of(UsedTiles.begin(), UsedTiles.end(),  [&guessed](const vector<int> &x) {  return any\_of(guessed.begin(), guessed.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  cout << "Hit!" << endl;  hits = hits + 1;  if (yguess == 9) {  T[xguess][yguess] = "|XX|\n";  } else {  T[xguess][yguess] = "|XX";  }  } else {  cout << "Miss!" << endl;  if (yguess == 9) {  T[xguess][yguess] = "|00|\n";  } else {  T[xguess][yguess] = "|00";  }  }  }  void updateBoard() {  cout << " 0 1 2 3 4 5 6 7 8 9" << endl;  for (int i = 0; i < T.size(); ++i) {  for (int j = 0; j < T[i].size(); ++j) {  cout << T[i][j] << ' ';  }  cout << endl;  }  }  };  class aircraftCarrier {  public:  void placeAircraftCarrier() {  // Generate a random coordinate for the Aircraft Carrier  int UpOrDown = rand() % 2;  int aircraftpositionx, aircraftpositiony;  if (UpOrDown == 0) {  aircraftpositionx = rand() % 6;  aircraftpositiony = rand() % 10;  } else {  aircraftpositionx = rand() % 10;  aircraftpositiony = rand() % 5;  }  // Map the Aircraft Carrier to the given coordinates  vector<vector<int>> AircraftTiles = {  {aircraftpositionx, aircraftpositiony},  {aircraftpositionx + 1, aircraftpositiony},  {aircraftpositionx + 2, aircraftpositiony},  {aircraftpositionx + 3, aircraftpositiony},  {aircraftpositionx + 4, aircraftpositiony}};  // Add all the tiles used to a used tile list  UsedTiles.insert(UsedTiles.end(), AircraftTiles.begin(),  AircraftTiles.end());  }  };  class Frigate {  public:  void placeFrigate() {  // Frigate Generation  vector<vector<int>> FrigateTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position  while (FrigateTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&FrigateTiles](const vector<int> &x) {  return any\_of(  FrigateTiles.begin(), FrigateTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int FrigatePositionX = rand() % 7;  int FrigatePositionY = rand() % 10;  FrigateTiles = {{FrigatePositionX, FrigatePositionY},  {FrigatePositionX + 1, FrigatePositionY},  {FrigatePositionX + 2, FrigatePositionY},  {FrigatePositionX + 3, FrigatePositionY}};  } else {  int FrigatePositionX = rand() % 10;  int FrigatePositionY = rand() % 6;  FrigateTiles = {{FrigatePositionX, FrigatePositionY},  {FrigatePositionX, FrigatePositionY + 1},  {FrigatePositionX, FrigatePositionY + 2},  {FrigatePositionX, FrigatePositionY + 3}};  }  }  UsedTiles.insert(UsedTiles.end(), FrigateTiles.begin(), FrigateTiles.end());  }  };  class Cruiser {  public:  void placeCruiser() {  // Cruiser Generation  vector<vector<int>> CruiserTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position  while (CruiserTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&CruiserTiles](const vector<int> &x) {  return any\_of(  CruiserTiles.begin(), CruiserTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int CruiserPositionX = rand() % 7;  int CruiserPositionY = rand() % 10;  CruiserTiles = {{CruiserPositionX, CruiserPositionY},  {CruiserPositionX + 1, CruiserPositionY},  {CruiserPositionX + 2, CruiserPositionY}};  } else {  int CruiserPositionX = rand() % 10;  int CruiserPositionY = rand() % 6;  CruiserTiles = {{CruiserPositionX, CruiserPositionY},  {CruiserPositionX, CruiserPositionY + 1},  {CruiserPositionX, CruiserPositionY + 2}};  }  }  UsedTiles.insert(UsedTiles.end(), CruiserTiles.begin(), CruiserTiles.end());  }  };  class Destroyer {  public:  void placeDestroyer() {  // Destroyer Generation  vector<vector<int>> DestroyerTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position  while (DestroyerTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&DestroyerTiles](const vector<int> &x) {  return any\_of(  DestroyerTiles.begin(), DestroyerTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int DestroyerPositionX = rand() % 7;  int DestroyerPositionY = rand() % 10;  DestroyerTiles = {{DestroyerPositionX, DestroyerPositionY},  {DestroyerPositionX + 1, DestroyerPositionY}};  } else {  int DestroyerPositionX = rand() % 10;  int DestroyerPositionY = rand() % 6;  DestroyerTiles = {{DestroyerPositionX, DestroyerPositionY},  {DestroyerPositionX, DestroyerPositionY + 1}};  }  }  UsedTiles.insert(UsedTiles.end(), DestroyerTiles.begin(),  DestroyerTiles.end());  }  };  int main() {  board gameBoard;  gameBoard.makeBoard();  aircraftCarrier carrier;  carrier.placeAircraftCarrier();  Frigate gate;  gate.placeFrigate();  Cruiser cruiser;  cruiser.placeCruiser();  Destroyer destroyer1;  destroyer1.placeDestroyer();  Destroyer destroyer2;  destroyer2.placeDestroyer();  int hits = 0;  while (hits <= 16) {  int xguess, yguess;  cout << "Enter row coordinate: ";  cin >> xguess;  cout << "Enter column coordinate: ";  cin >> yguess;  gameBoard.checkHits(hits, xguess, yguess);  gameBoard.updateBoard();  }  cout << "You hit every ship!" << endl;  return 0;  } |

Issues with iteration 1:

1. Lack of comments.
2. No difficulty selection.
3. No tutorial or introduction
4. No input validation

## Iteration 2:

|  |
| --- |
| #include <algorithm>  #include <cstdlib>  #include <ctime>  #include <iostream>  #include <limits>  #include <vector>  using namespace std;  vector<vector<int>> UsedTiles;  int lives;  // board class, allows a board to be created and manipulated  class board {  public:  vector<vector<string>> T;  board() : T(10, vector<string>(10, "|\_\_")) {}  // displays the first iteration of board to the user  void makeBoard() {  cout << " 0 1 2 3 4 5 6 7 8 9" << endl;  for (int i = 0; i < T.size(); ++i) {  for (int j = 0; j < T[i].size(); ++j) {  cout << T[i][j] << ' ';  }  cout << endl;  }  }  // checks if the users guess was a hit or miss, and manipulates the board  // accordingly  void checkHits(int hits, int xguess, int yguess) {  vector<vector<int>> guessed = {{xguess, yguess}};  if (any\_of(UsedTiles.begin(), UsedTiles.end(),  [&guessed](const vector<int> &x) {  return any\_of(guessed.begin(), guessed.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  cout << "Hit!" << endl;  hits = hits + 1;  if (yguess == 9) {  T[xguess][yguess] = "|XX|\n";  } else {  T[xguess][yguess] = "|XX";  }  } else {  cout << "Miss!" << endl;  if (yguess == 9) {  T[xguess][yguess] = "|00|\n";  } else {  T[xguess][yguess] = "|00";  }  }  }  // displays the new updated board to the user when called upon  void updateBoard() {  cout << " 0 1 2 3 4 5 6 7 8 9" << endl;  for (int i = 0; i < T.size(); ++i) {  for (int j = 0; j < T[i].size(); ++j) {  cout << T[i][j] << ' ';  }  cout << endl;  }  }  };  // aircraft carrier class, allows the aircraft carrier to be created and  // manipulated  class aircraftCarrier {  public:  void placeAircraftCarrier() {  // Generate a random coordinate for the Aircraft Carrier  int UpOrDown = rand() % 2;  int aircraftpositionx, aircraftpositiony;  if (UpOrDown == 0) {  aircraftpositionx = rand() % 6;  aircraftpositiony = rand() % 10;  } else {  aircraftpositionx = rand() % 10;  aircraftpositiony = rand() % 5;  }  // Map the Aircraft Carrier to the given coordinates  vector<vector<int>> AircraftTiles = {  {aircraftpositionx, aircraftpositiony},  {aircraftpositionx + 1, aircraftpositiony},  {aircraftpositionx + 2, aircraftpositiony},  {aircraftpositionx + 3, aircraftpositiony},  {aircraftpositionx + 4, aircraftpositiony}};  // Add all the tiles used to a used tile list  UsedTiles.insert(UsedTiles.end(), AircraftTiles.begin(),  AircraftTiles.end());  }  };  class Frigate {  public:  void placeFrigate() {  // Frigate Generation  vector<vector<int>> FrigateTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position, and adds the frigate to  // the board  while (FrigateTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&FrigateTiles](const vector<int> &x) {  return any\_of(  FrigateTiles.begin(), FrigateTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int FrigatePositionX = rand() % 7;  int FrigatePositionY = rand() % 10;  FrigateTiles = {{FrigatePositionX, FrigatePositionY},  {FrigatePositionX + 1, FrigatePositionY},  {FrigatePositionX + 2, FrigatePositionY},  {FrigatePositionX + 3, FrigatePositionY}};  } else {  int FrigatePositionX = rand() % 10;  int FrigatePositionY = rand() % 6;  FrigateTiles = {{FrigatePositionX, FrigatePositionY},  {FrigatePositionX, FrigatePositionY + 1},  {FrigatePositionX, FrigatePositionY + 2},  {FrigatePositionX, FrigatePositionY + 3}};  }  }  UsedTiles.insert(UsedTiles.end(), FrigateTiles.begin(), FrigateTiles.end());  }  };  // cruiser class, allows the cruiser to be created and manipulated  class Cruiser {  public:  void placeCruiser() {  // Cruiser Generation  vector<vector<int>> CruiserTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position  while (CruiserTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&CruiserTiles](const vector<int> &x) {  return any\_of(  CruiserTiles.begin(), CruiserTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int CruiserPositionX = rand() % 7;  int CruiserPositionY = rand() % 10;  CruiserTiles = {{CruiserPositionX, CruiserPositionY},  {CruiserPositionX + 1, CruiserPositionY},  {CruiserPositionX + 2, CruiserPositionY}};  } else {  int CruiserPositionX = rand() % 10;  int CruiserPositionY = rand() % 6;  CruiserTiles = {{CruiserPositionX, CruiserPositionY},  {CruiserPositionX, CruiserPositionY + 1},  {CruiserPositionX, CruiserPositionY + 2}};  }  }  UsedTiles.insert(UsedTiles.end(), CruiserTiles.begin(), CruiserTiles.end());  }  };  // destroyer class, allows the desroyer to be created and manipulated  class Destroyer {  public:  void placeDestroyer() {  // Destroyer Generation  vector<vector<int>> DestroyerTiles;  int UpOrDown = rand() % 2;  // Check there's no boat in the current position  while (DestroyerTiles.empty() ||  any\_of(UsedTiles.begin(), UsedTiles.end(),  [&DestroyerTiles](const vector<int> &x) {  return any\_of(  DestroyerTiles.begin(), DestroyerTiles.end(),  [&x](const vector<int> &y) { return x == y; });  })) {  if (UpOrDown == 0) {  int DestroyerPositionX = rand() % 7;  int DestroyerPositionY = rand() % 10;  DestroyerTiles = {{DestroyerPositionX, DestroyerPositionY},  {DestroyerPositionX + 1, DestroyerPositionY}};  } else {  int DestroyerPositionX = rand() % 10;  int DestroyerPositionY = rand() % 6;  DestroyerTiles = {{DestroyerPositionX, DestroyerPositionY},  {DestroyerPositionX, DestroyerPositionY + 1}};  }  }  UsedTiles.insert(UsedTiles.end(), DestroyerTiles.begin(),  DestroyerTiles.end());  }  };  int main() {  // creates an instance of board and displays it to the user  board gameBoard;  gameBoard.makeBoard();  // creates and places an instance of aircraft  aircraftCarrier carrier;  carrier.placeAircraftCarrier();  // creates and places an instance of frigate  Frigate gate;  gate.placeFrigate();  // creates and places an instance of cruiser  Cruiser cruiser;  cruiser.placeCruiser();  // creates and places an instance of destroyer  Destroyer destroyer1;  destroyer1.placeDestroyer();  Destroyer destroyer2;  destroyer2.placeDestroyer();  int hits = 0;  int xguess, yguess;  // displays an introduction for the user  cout << "Welcome to Battleships!, the game where you try to sink the enemy's "  "ships. You must find and destroy 5 different boats on a 10x10 grid: "  "an Aircraft Carrier which is 5 spaces long, a Frigate which is 4 "  "spaces long, a Cruiser which is 3 spaces long and 2 Destroyers "  "which are 2 spaces long. Developed by 21307114.\nPlease select a "  "difficulty:\n1. Casual\n2. Advanced\n3. Impossible\n"  << endl;  int difficulty;  // allows the user to select a difficulty and validates the user input  while (true) {  cout << "Difficulty: ";  cin >> difficulty;  if (cin.fail() || difficulty < 1 || difficulty > 3) {  // Invalid input, clear the buffer and ignore the rest of the line  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid input. Please enter a number between 1 and 3." << endl;  } else {  // Valid input, break out of the loop  break;  }  }  if (difficulty == 1) {  lives = 10000;  }  if (difficulty == 2) {  lives = 50;  }  if (difficulty == 3) {  lives = 25;  }  // validates the users guesses of co-ords  while (hits <= 16) {  while (true) {  cout << "Enter row coordinate (0-9): ";  cin >> xguess;  if (cin.fail() || xguess < 0 || xguess > 9) {  // Invalid input, clear the buffer and ignore the rest of the line  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid input. Please enter a number between 0 and 9." << endl;  } else {  // Valid input, break out of the loop  break;  }  }  while (true) {  cout << "Enter column coordinate (0-9): ";  cin >> yguess;  if (cin.fail() || yguess < 0 || yguess > 9) {  // Invalid input, clear the buffer and ignore the rest of the line  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid input. Please enter a number between 0 and 9." << endl;  } else {  // Valid input, break out of the loop  break;  }  }  // checks if the user has hit a boat  gameBoard.checkHits(hits, xguess, yguess);  // displays the new board  gameBoard.updateBoard();  }  cout << "You hit every ship!" << endl;  return 0;  } |

Addressing iteration 1 issues:

1. Additional comments were added to explain sections of the code
2. Added a difficulty selection with validation
3. Added an introduction for the user
4. Added validation for all user inputs:

A screenshot of a computer

Description automatically generated