Beschreibung der Methoden in den verschiedenen Programmmodulen:

ai.c:

* Int \*get\_ai\_turn
  + Ruft je nach übergebener Schwierigkeit eine der drei Methoden zum Berechnen des Spielzugs auf
  + Gibt Array mit x, y Koordinaten aus
* Int \*ai\_diff1
  + Schießt auf zufällige Felder innerhalb von size
  + Gibt Array mit x, y Koordinaten aus
* Int \*ai\_diff2
  + Schießt auf ein sinnvolles Feld innerhalb von size, analysiert dazu die Matrix mithilfe von is\_useful
  + Gibt Array mit x, y Koordinaten aus
  + Weitere Infos: Dokumentation Kapitel 3: Die KI
* Int \*ai\_diff3
  + Schießt auf ein sinnvolles Feld innerhalb von size, analysiert dazu die Matrix mithilfe von is\_useful
  + Gibt Array mit x, y Koordinaten aus
  + Weitere Infos: Dokumentation Kapitel 3: Die KI
* Int is\_useful
  + bestimmt abhängig von übergebener Schwierigkeit, ob ein Schuss sinnvoll wäre
  + gibt einen Wahrheitswert zurück

console.c:

* Void draw\_intro
  + Gibt eine Begrüßung für den Spieler aus
* Void draw\_sceen
  + Erstellt eine Matrix der Größe “size”, die vom Benutzer angepasst werden kann
  + Ruft “check\_tile\_orientation” auf ( --> entscheidet so, ob Bug, Deck, oder Heck gesetzt wird )
* Void player\_draw\_screen
  + Zeichnet ein zweites, leeres Spielfeld, um die Schiffe des Gegners zu verstecken
  + Lediglich versenkte Schiffe, Treffer und Fehlschüsse werden angezeigt
* Int check\_tile\_orientation
  + Prüft die Umgebung des ausgewählten Feldes auf Wasser, oder einem schon beschossenen Feld
  + Übergibt das Ergebnis an “draw\_screen”
* Void set\_tiles
  + Initialisiert die graphische Darstellung der Schiffe
* Void print\_stats
  + Gibt am Ende des Spiels die Statistik aus
* Void clear\_screen
  + “Säubert” die Konsole durch das Ausgeben vieler \n Escape-Sequenzen
* Void player\_set\_ships
  + Lässt den Spieler zunächst entscheiden, ob die eigenen schiffe selbst setzten will:
  + Ja --> der Spieler wählt die Schiffart, dann die gewollte Koordinate und die Richtung, in der das Schiff gesetzt wird
  + Nein --> Eingabe wird an “rand\_set\_ships” übergeben
* Void get\_settings
  + Spieler kann die Spieleinstellungen festlegen
  + Lässt zwischen Ein- und Zweispielermodus entscheiden
  + Ein Spieler: Schwierigkeitsgrad der K.I. kann aus drei Modi gewählt werden.
  + Zwei Spieler: Ausführen von “player\_set\_ship” für Spieler zwei
  + Spieler kann zwischen Standard- und Eigeneinstellungen wählen
* Int get\_battleground\_size
  + Es wird eine individuelle Feldgröße zwischen 10 und 20 festgelegt
  + Der Wert wird an “draw\_screen” übergeben
* Int get\_ships
  + Spieler wählt Anzahl der Schiffe (1 – 20)
  + Spieler wählt welche Art von Schiffen wie oft auftreten
* Int \*player\_move
  + Säubert die Konsole und gibt das Spielfeld aus
  + Spieler entscheidet, durch Angeben der Koordinaten, auf welches Feld er schießen möchte
* Void show\_player\_battleground
  + Zeigt dem Spieler das Eigene Feld mit den gesetzten Schiffen und durch den Gegner beschossene Felder
* Int response
  + Prüft ob ein Schiff getroffen wurde und gibt das Ergebnis aus
* Void flush
  + Puffer leeren

logic.c:

* Int init\_battleground
  + Initialisiert eine Matrix der Größe size
  + Ruft reset\_battleground auf
  + Gibt im Erfolgsfall 1, im Fehlerfall OUT\_OF\_MEMORY zurück
* Int init\_stats
  + Initialisiert die Statistikspeicher je nachdem, welche Schiffe es gibt
  + Ruft hierzu auch set\_ships\_stats auf
  + Gibt im Erfolgsfall 1, im Fehlerfall OUT\_OF\_MEMORY zurück
* Void set\_ships\_stats
  + Wertet ships[] aus und extrahiert die Schiffstypen
* Void update\_stats
  + Erhöht die Stats je nach getroffener Schiffsklasse
* Void reset\_battleground
  + Setzt die Matrix auf den Anfangszustand (Wasser überall) zurück
* Int rand\_set\_ships
  + Setzt die Schiffe zufällig auf dem Spielfeld
  + Ruft hierzu set\_ship auf
* Int set\_ship
  + Überprüft die Position, an die das Schiff gesetzt werden soll und schreibt es in die Matrix, falls möglich
  + Gibt im Erfolgsfall 1, sonst 0 zurück
* Int ship\_mass\_validation
  + Prüft, ob die Anzahl an Schiffen auch in die vorliegende Matrix passt
  + Gibt im Erfolgsfall 1, sonst 0 zurück
* Int shoot
  + Schießt auf das Feld an den übergebenen Koordinaten
  + Gibt zurück welche Schiffsklasse getroffen wurde
* Int check\_downed
  + Überprüft, ob ein Schiff versenkt wurde oder nicht und setzt dieses Schiff im Bedarfsfall auf versenkt
  + Ruft hierzu check\_downed\_helper (Hilfsmethode, hier nicht aufgeführt) und downed\_maker auf
  + Gibt 1 oder 0 zurück
* Int downed\_maker
  + Setzt ein Schiff rekursiv auf (D)owned
* Int is\_end\_game
  + Prüft ob das Spiel beendet ist (sich also noch Ziffern in der Matrix befinden)
  + Gibt 1 oder 0 zurück
* Int compare
  + Vergleichsfunktion für qsort in main() Vgl. http://www.cplusplus.com/reference/cstdlib/qsort/