ゲームを題材としたプログラミング教育支援システムの開発

06H082 東川諒央

目次

1 **はじめに** 1

2 **先行研究に対する考察** 2
- 2.1 ボードゲームの戦略プログラミングを題材とした Java 演習の支援システムの開発 2
- 2.2 アルゴリズム構築能力育成の導入教育 3
- 2.3 プログラミングの学習に必要な要素 3

3 **ゲーム** 4
- 3.1 ゲームを用いた学習の有用性 4
- 3.2 GP 化するゲームの選択基準 5
- 3.3 Battle Ship 6
- 3.4 GP のルール 6

4 **結果と考察** 7
- 4.1 Practice 8
- 4.2 Verification 9
- 4.3 Score 10

5 **まとめ** 11
- 5.1 今後の課題 12

付録 A ソースコード 14
- A.1 Practice 14
- A.2 Verification 28
- A.3 Score 39

概要

現在、プログラミング教育を行う際に、学習者が理解し易くする為の手法が研究されている。それらの中に、ゲームを学習利用している研究がいくつかある。ゲームを学習に利用することで、学習者の意欲維持による自主学習や集中した状態での学習が望める。しかし、現存している研究ではプログラミング初学者や苦手意識を持った学習者を対象としていない。その為複雑なアルゴリズムと文法が必要になり、難易度が高くなる。本研究では、プログラミング初学者や苦手意識を持った学習者が自分でプログラミングが出来るようになる為のシステムを考えた。それには先ず、学習者が学習意欲を維持できる環境が必要と考えた。そして、学習者が自分でアルゴリズムを考え、実装することで、考案したアルゴリズムの評価を自身が行えることが学習意欲の維持と同程度重要であると考え、条件にあうゲームを検討した。

検討した結果、 Battle Ship ゲームを GP(Game Programme) 化した。GP には攻撃用のアルゴリズムは書かれておらず、学習者が攻撃アルゴリズム用関数をプログラミングすることで動作する仕様である。しかし、Battle Ship のルールではアルゴリズムが複雑になってしまうため、ルールを簡潔化し一人用のゲームとすることでプログラミング時の難易度を下げた。また、このゲームをクリアする為に、逐次実行、条件分岐、繰り返しと言ったものが重要になる。その為アルゴリズム概念の理解を手助けするシステムとなり得ると考えられる。

1 はじめに

現在、授業を受けたにも関わらずプログラミングができない学生が数多くいる。プログラミングが出来ない理由として、アルゴリズム[*1]を理解出来ない事が挙げられる。アルゴリズムはプログラミングを行う上で非常に大きなウェイトを占めており、理解が出来なければプログラムを書けないと言っても過言ではない。

先行研究では、プログラミングの学習に於いてアルゴリズムの学習を一番の課題として取り上げている。この事からプログラミングが出来ないという課題の解決には、アルゴリズムについての理解が必要であることが伺える。

大阪産業大学工学部情報システム工学科のプログラミング演習[*2]の方針では、言語の文法に重点を置いて指導している。授業工程はテスト、文法指導、演習課題のルーチンで行われている。授業では全ての学生に対して 1 対 1 で指導することは出来ず、学生の理解が及ぶまで指導する時間も多く取ることは難しい。それに加え、授業を理解できないと意欲を損なってしまうために自主学習もあまり望めない。テストや演習課題は穴埋め形式で、書き方の細かい指定迄ついている為、学生自身がアルゴリズムを考え、実装するという作業を行うことが出来ない。これでは学生がプログラミングを行う上で必要なアルゴリズムの概念[*3]を理解することが難しいと考えた。

この問題を解決するため、本研究ではゲームを用いたアルゴリズムの学習の為のシステムとして、GP[*4]を開発した。

[*1] ある特定の目的を達成する為の処理手順。

[*2] プログラミングの演習授業。

[*3] 本研究で言うアルゴリズムの概念とは次の 2 点を指す。
1) アルゴリズムとは単純な仕事の積み重ねであること。
2) それぞれの手順の解釈には曖昧さがないこと。

[*4] Game Programme : 本研究において開発したシステムで、教育利用の為にコンピュータゲーム化したもの。

2　先行研究に対する考察

プログラミング教育についての先行研究はいくつか存在している。本章では、その中でもプログラミングの初学者、プログラミングに苦手意識を持つ学習者を対象とした研究、もしくはゲームを利用したプログラミング教育の研究について整理する。以下 2 つの先行研究に基いて考察した。

- ボードゲームの戦略プログラミングを題材とした Java 演習の支援システムの開発 [1]
- アルゴリズム構築能力育成の導入教育 [2]

2.1　ボードゲームの戦略プログラミングを題材とした Java 演習の支援システムの開発

この研究ではボードゲームを題材とした、対戦形式の Java プログラミング演習を提案している。使用しているゲームは五五ゲーム[*5]である。プログラミング演習の題材にボードゲーム戦略を用いる利点として、次の点が挙げられる。

- 身近な題材による関心と興味
  - 学生にとって取っつきやすい題材である。
- 試行錯誤的な問題解決の訓練
  - 正解が 1 つに定まらないオープンプロブレムである。
  - 試行錯誤を通して各自の自由な発想を引き出せる。
- 明確な目的と競争による意欲向上
  - 勝つという目的が明確である。
  - 対戦結果による得点比較が客観的な評価基準となる。
  - 学生の競争意識を刺激し、演習意欲を向上させることが期待できる。
- 安易な不正の防止
  - 他人にプログラムを写させると自分の戦績が不利になる為、安易なコピーを防げる。

この研究の報告では、戦略のアイデアを実装や戦略の検証を行ったり、戦譜から他人の戦略を推定し相手に合わせて戦略を修正することを学習している。しかし、対戦形式を取っているためプログラムが盤面の把握をした上で対戦相手への攻撃や防御を行わなくてはならない。その為アルゴリズムは複雑になり、プログラミング初学者や苦手意識をもっている学習者にとっては難易度が高い。

[*5] 五目並べに石取りの要素を加えた二抜き連珠というゲームに、勝利条件や禁じ手などのルールを整備した物。

2.2 アルゴリズム構築能力育成の導入教育

プログラミングの学習に於いて、学習者にアルゴリズムと言語の文法を同時に学習させることは負担が大きい。その為アルゴリズムと文法の学習を分割し、先ず実作業によって学習者にアルゴリズムの概念を理解してもらうことから始めている。この研究では実作業によってアルゴリズムの概念を理解した学生に、“ことだま on Squeak”[*6]を用いてプログラミングを学習させる手法を提案している。

この研究の提案する手法によって学習する利点は次の点である。

- アルゴリズムの概念の理解
  - 問題を手作業で解決させることによって、それを解決するためのアルゴリズムを理解できる。
- プログラミングによるアルゴリズムの構築体験
  - 手作業で行ったことと同様のことをプログラミングするのでアルゴリズムは比較的理解しやすい。
  - 日本語で記述できる教育用プログラミング言語の “ことだま on Squeak” を使用することで、アルゴリズムからプログラミング言語の表現に翻訳する必要がない。

この研究の報告ではアルゴリズム概念の理解がアルゴリズム構築における分析や詳細化の段階の重要性を認識させることが分かった。しかし、難点として “ことだま on Squeak” 自体の言語仕様に学習者の思考を制限されてしまうことがある。“ことだま on Squeak” は実装可能なアルゴリズムが一重の繰り返しループを使ったものに制限されるため、入れ子になった繰り返しの理解を出来ない可能性がある。

2.3 プログラミングの学習に必要な要素

前述した 2 件の先行研究より得られたものは次の点である。

- アルゴリズムの概念の理解はプログラミングにおいて重要。
- 他人のプログラムの出力結果からアルゴリズムを推定、学習できる。
- 構文エラーは学習者にとっては負担である。
- 目的が明確であればアルゴリズムの構築が比較的容易である。
- プログラミングに限らず学習を行うには意欲の維持が必要である。

先行研究が既に有効である学習方法を提示しているため、アルゴリズムの学習に関する研究には、これらを念頭におくことが必要であると考えた。

[*6] 小学生が扱えるように設計された “Squeak eToys” の日本語拡張版で、ビジュアルプログラミング環境である。“ことだま on Squeak” は、“Squeak eToys” のタイルスクリプティングシステムを引き継いでいるため、プログラムの文法を覚える必要がない。

3 ゲーム

現在ゲームという言葉の学術的統一的見解は得られていない [3]。しかし、ゲームを題材として扱う以上定義がなされていなければ使用するゲームを選定することが出来ない。ゲームデザイナーのグレッグ・コスティキャンは雑誌[*7]において、ゲームとは「十分な情報の下に行われた意思決定をもって、プレイヤーが与えられた資源を管理しつつ自ら参加し、立ちはだかる障害物を乗り換えて目標達成を目指すもの」であるとしている。

又、ゲームの定義について語っている web サイト[*8]が在る。このサイトでは次の三点をゲームとしての成立条件として扱っている。

- ルールによって行動のパターンが限定されている。
- 行為、行動、意思決定の指針が目標や評価システムに方向付けられている。
- ゲームの参加者のとった行動 (選択) の差によってゲームの結果及び過程が異なる物である。

ゲームデザイナーであるグレッグの定義と上記三点の条件が酷似している為、この三点の条件を本研究のゲームの定義として扱う。

3.1 ゲームを用いた学習の有用性

学習とは退屈で単調であったり、極端に複雑、もしくはその両方である。その為、学習者にとって苦痛を与える物であることが多い。しかし、ゲームを学習に利用することによって単調になりがちな学習にエンターテインメント要素を取り入れることができ、意欲の向上を狙うことが出来る。

いくつもの先行研究によってゲームを用いることによる学習対象者の意欲の向上と、高い学習効果が認められている。ゲーム学習は既にいくつもの組織において実際に利用されている。

テクノロジー利用学習では、通常の授業と同じ内容を学んだ結果、テストの点数や持続力で改善し、3 割程度の学習時間を短縮したという結果が、数多くある研究のほとんどで示されている。多くの研究は軍の研究機関で行われており、その研究から 3 分の 1 の法則[*9]が見つけられている。このことからゲームの学習利用は有効であることが分かる。

[*7] Interactive Fantasy[4]

[*8] Critique of games[5]

[*9] 教育にかかる時間とコストをおよそ 3 分の 1 削減し、学習効果が 3 分の 1 程度向上している。

3.2 GP 化するゲームの選択基準

本研究はプログラミング学習を始める学生や、学習しているが中々理解が進まない学生を対象にしている。その為 GP 化するゲームは厳選する必要があった。本研究においてゲームに求める条件は次の 5 点である。

- アルゴリズムの評価を容易に行うことが出来る。
- 複雑なアルゴリズムを必要とせず目標達成することが出来る。
- 一人用としてゲームをプレイできる。
- 多岐に渡る解が存在する。
- ルールが簡単であり、且つある程度の知名度がある。

条件に合うゲームを探した結果、マインスイーパと Battle Ship が候補に挙がった。しかしマインスイーパは地雷を踏むとゲームが途中で終了してしまうため、ゲームとしての難易度が上がってしまう。その為、途中で失敗することがない Battle Ship は難易度が比較的低く、プログラミング初学者や苦手意識を持つ学生に対して抵抗がより少ないと考え、本研究で GP 化するゲームに Battle Ship を選択した。

3.3 Battle Ship

Battle Ship とは海戦ゲーム[*10]の変種として日本でタカラから “レーダー作戦” ゲームの名称で販売されていたゲームで、アメリカからの輸入ゲーム (英: Battleship) である。ルールは次の通りである。

- マス目の設定は縦横 10 マス。
- 軍艦に耐久力は設定されない。代わりに大きさが設定され、空母（5 マス)、戦艦（4 マス)、巡洋艦（3 マス)、潜水艦（3 マス)、駆逐艦（2 マス）となっている。軍艦のマス目全部に攻撃を受けると沈没となる。
- 軍艦は移動できない。
- 軍艦の隣接マスに攻撃を受けた場合、「水しぶき」など[*11]の申告をしなくてよい。
- 各軍艦にコストや回数制限付きで特殊な攻撃がある。（索敵・絨毯爆撃・連続攻撃等）

3.4 GP のルール

元々の Battle Ship を学習利用するにはルールが複雑である。その為 GP 化するに辺り特殊な攻撃の要素を無くすことでルールを簡略化した。

GP では一人プレイ用。ルールは次の通りである。

- 海域としてのマス目の設定は縦横 10 マス。
- 敵の軍鑑のみ配置される。配置はランダムに行われる。
- 軍艦には大きさが設定され、戦艦 (5 マス) 、巡洋艦 (4 マス) 、潜水艦 (3 マス) 、駆逐艦 (3 マス) 、輸送艦 (2 マス) となっている。軍艦のマス目全部に攻撃を受けると沈没となる。
- 攻撃は 1 手につき 1 マスにのみ行う。海域内且つ未攻撃のマスであれば他に攻撃の制限は無い。
- 軍鑑は移動しない。
- 全ての軍鑑が沈没した時点でゲームは終了となる。

上記のルールに基づいてゲームを行い、より少ない攻撃回数でのクリアを目指すものとした。

[*10] 紙と筆記用具があればプレイ可能。テーブルゲームの一つで、戦艦ゲーム、軍艦ゲームなどとも呼ばれる。3 人以上でプレイ可能だが、ゲームが非常に複雑になるのでプレイヤーは基本的に二人。

[*11] 敵艦を発見する為の要素。

4　結果と考察

本研究では 3 タイプのプログラムを開発した、。各プログラムには別々の役割りがある。ゲームをアルゴリズムの学習に用いる上で必要な機能を持たせたプログラムが次の 3 タイプである。

Practice　ゲームをプレイすることでルールを把握し、思考中のアルゴリズムに具体性を持たせる。
Verification　記述されたアルゴリズムが理想通りに実装されているかをチェックする。
Score　アルゴリズムの善し悪しを判断する為、評価を行う。

これらの機能が学習者のガイドラインとなることで、 GP を用いてプログラミングを行う際に、設計、実行、評価、再構築のルーチンを行いやすくなる。

Verification と Score は Battle Ship の攻撃を担う関数を shoot という名前で別ファイルに分けている。学習者はこの関数を書き換えることによって攻撃のアルゴリズムを実装する。動作はいずれも CUI によって行われる。

4.1 Practice

この GP は先に説明した二つとは目的が異なり、人間のプレイ用 GP である。Battle Ship を実際にプレイし、アルゴリズムを考えるための物で攻撃したい座標をインタープリタに入力することで指定できる。

Practice の実行画面を図 1 に示す。

図 1　Practice の実行画面

4.2 Verification

この GP は実装した攻撃関数の挙動を調べるための物で、攻撃したマスと攻撃結果、そして現在の手数を一手ずつ出力するプログラムである。軍艦を全て沈没させることによってゲームは終了する。

Verification の実行画面を図 2 に示す。

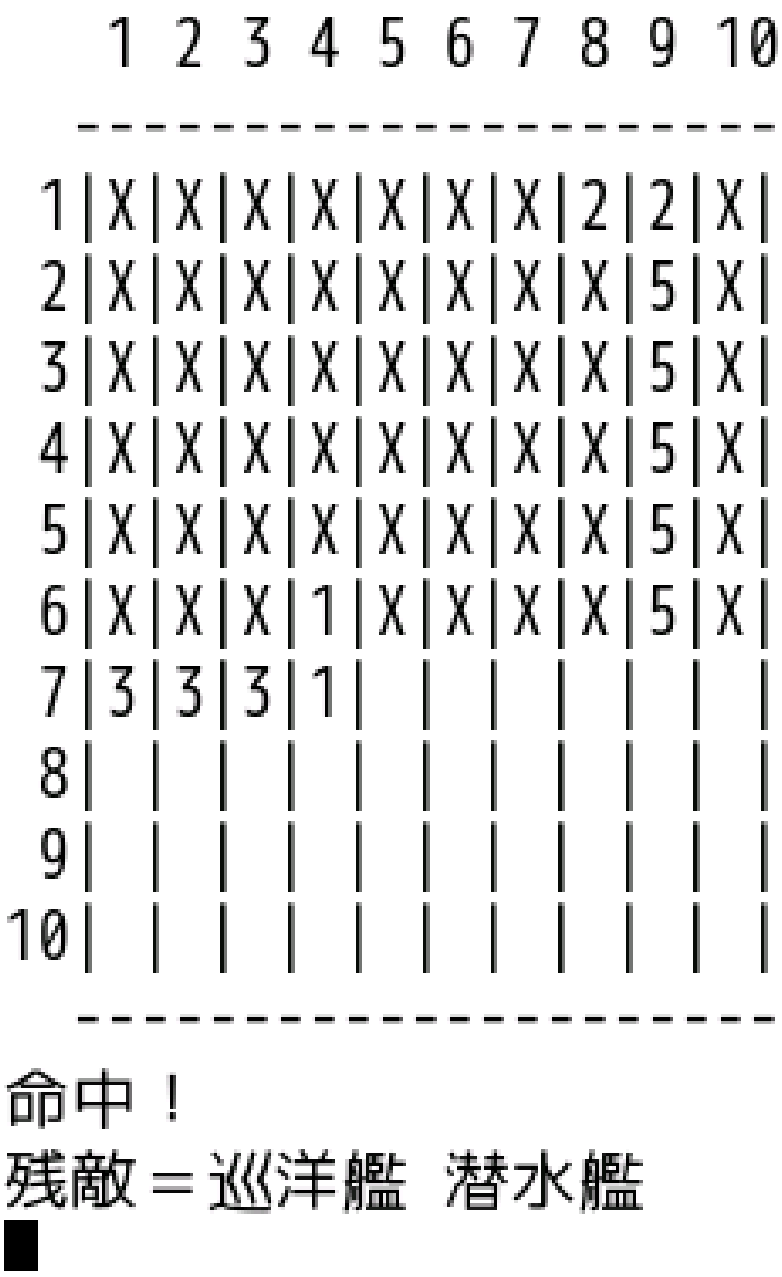

図 2　Verification の実行画面

4.3 Score

この GP は shoot 関数によって実装した攻撃アルゴリズムのゲーム終了までの平均攻撃回数を調べる物である。100 回ゲームを繰り返し、その平均攻撃回数を出力する。

Score の実行画面を図 3 に示す。

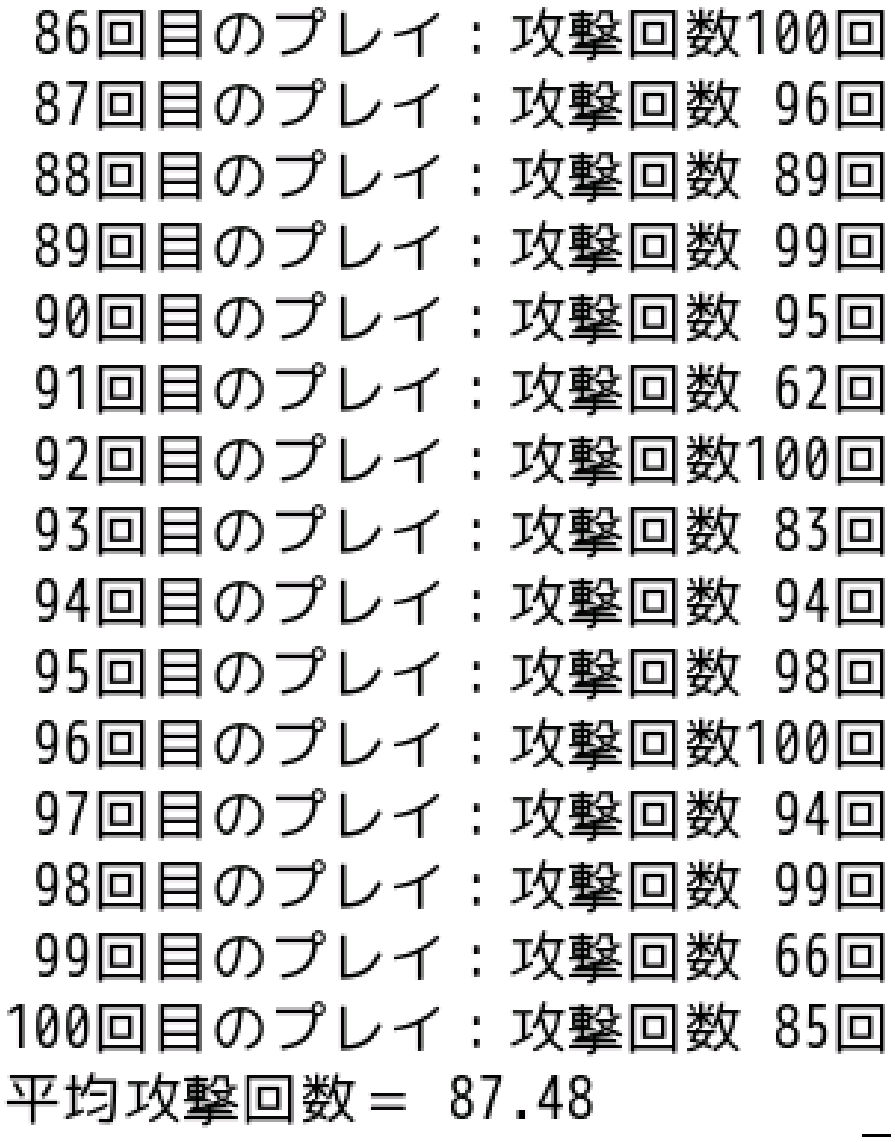

図 3　Score の実行画面

5　まとめ

本研究ではプログラミングが出来ない学生、学習しようという学生がアルゴリズム概念の理解を補助するためのシステムの開発を行った。そのため簡潔なルールと複数の攻略アルゴリズムが存在し、攻略難易度が低いゲームを検討した。そして条件に当てはまるゲームを GP 化し、学習者が攻撃関数のみをプログラミングすればよい限定的なプログラミングシステムを開発した。

次に示した 4 点は、先行研究から得られたプログラミング学習に必要と考える要素を実装できたと言えるものである。

1. 他人のプログラムの出力結果からアルゴリズムを推定、学習できる。
2. 明確な目的が存在している。
3. 学習意欲の維持が可能である。
4. アルゴリズムの善し悪しが数値によって判断できる。

次に実装したと言える根拠を示す。

1 は一手ずつ攻撃の状態を出力する GP により、他人が作成した攻撃関数の動作を確認することができる。これにより自分が思いつかなかったアルゴリズムを出力結果より推定、実装を行うことが出来る。

2 はランダムに配置された軍艦を全て沈没させることがクリア条件である。その為クリア条件が、この場合の明確な目的となる。そのためプログラミングが出来無い人にありがちな、設計段階で何をすれば良いのか分らないという事態が起りにくい。

3 は学習者がゲームをプレイしながらアルゴリズムを考えることが出来るため学習意欲の維持が可能である。自分の書いたプログラムからフィードバックを得られる要素も意欲の維持に繋がる。

4 は Score によって出力された平均攻撃回数を元に学習者が自身でアルゴリズムの評価を下せる。これにより自己による学習が成立ち、他人の介在を常に必要とするわけでは無くなる。

以上の点からアルゴリズムの学習を支援するプログラムの開発という目標は達成できた。

5.1 今後の課題

今後の課題は、実装出来なかった要素を実装することである。前述したアルゴリズムの学習に必要と考える要素で実装出来なかったものを次に示す。

1. システムによってアルゴリズムの概念の理解が可能。
2. 複雑な文法を必要とせず、学習者がアルゴリズムの考察に専念することが可能。

上記 2 点はアルゴリズムのみの学習を行うためには必ず必要になる要素である。前述したが、プログラミングを学習するには、アルゴリズムと言語の文法は同時に行うべきではない。その為、学習者がプログラミングを行わずにアルゴリズムを実装する方法を考える必要がある。

謝辞

本研究を進めていく上で、藤井 信夫教授、大垣 斉講師、御指導及び御協力を戴きました。また、研究室の OB として指導して下さった、水野 貴弘先輩、小山 翔平先輩、fken メイルリストのメンバの方々には御助言を賜りました。ここに深く感謝の意を表します。

参考文献

[1] 尾崎浩和, 富永浩之, 山崎敏範. ボードゲームの戦略プログラミングを題材とした java 演習の支援システムの開発. NII 書誌, 2006.

[2] 杉浦学, 松沢芳昭, 岡田健, 大岩元. アルゴリズム構築能力育成の導入教育: 実作業による概念理解に基づくアルゴリズム構築体験とその効果. NII 書誌, 2008.

[3] Marc Prensky. デジタルゲーム学習: シリアスゲーム導入・実践ガイド. 東京電気大学出版局, 2009.

[4] Greg Costikyan. Interactive fantasy. http://www.costik.com/nowords.html, 1994.

[5] INOUE Akito. ゲームの定義論. http://www.critiqueofgames.net/data/index.php?%A5%B2%A1%BC%A5%E0%A4%CE%C4%EA%B5%C1%CF%C0, 10 2008.

※ URL はすべて 2010 年 1 月 16 日時点のもの

付録 A　ソースコード

A.1　Practice

Prctice とは実際にゲームをプレイすることで学習者がルールを把握し、アルゴリズムを考察する為のプログラムである。

A.1.1　senkanplay.c

```
/*************************************************************************
 * Battle Ship ゲーム                                                    *
 *  - Practice モード                                                    *
 *************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

char line[100];

#define N 0    /* 未攻撃   */
#define BM 1   /* 攻撃外れ */
#define BH 2   /* 攻撃命中 */
#define SK 11  /* 戦艦   : Life 5 */
#define JK 12  /* 巡洋艦 : Life 4 */
#define KK 13  /* 駆逐艦 : Life 3 */
#define SM 14  /* 潜水艦 : Life 2 */
#define YS 15  /* 輸送艦 : Life 2 */

void haichi (int sb[][10]);
void dispsb (int sb[][10]);
int zanteki (int skp, int jkp, int kkp, int smp, int ysp);
void shoot (int sb[][10], int i, int *x, int *y);
int meichup (int sb[][10], int x, int y);
void meichud (int k);

/*************************************************************************
 * グローバル変数 :                                                      *
 *   skp => SK                                                           *
 *   jkp => JK                                                           *
 *   kkp => KK                                                           *
 *   smp => SM                                                           *
 *   ysp => YS                                                           *
 *                                                                       *
```

```
 * 各軍艦の残 Life point を示す変数.                                        *
 ****************************************************************************/

int skp = 5, jkp = 4, kkp = 3, smp = 3, ysp = 2;

/****************************************************************************
 * メイン関数 :                                                             *
 *   flg => 攻撃済み座標への再攻撃防止用フラグ                              *
 *   i   => 現在の攻撃回数カウント用変数                                    *
 *   x   => 縦座標格納用変数                                                *
 *   y   => 横座標格納用変数                                                *
 *   k   => 攻撃結果格納用変数                                              *
 ****************************************************************************/
int
main (void)
{
  int sb[10][10];
  static int  flg[10][10]={0};
  int i, x, y, k;

  haichi (sb);                              /* 各軍艦の配置 */
  dispsb (sb);                              /* Battle Ship ゲームの盤面表示 */
  zanteki (skp, jkp, kkp, smp, ysp);        /* 残敵の表示 */

  do
    {
      shoot (sb, 1, &x, &y);                /* 攻撃座標の指定 */
    }
  while(x < 0 || x > 9 || y < 0 || y > 9);  /* 領海外を指定した場合，再攻撃 */

  flg[x][y] = 1;

  printf ("(x,y) = (%d,%d)\n", x, y);       /* 攻撃を行なった座標の表示 */

  /**************************************************************************
   * 必要条件 :                                                             *
   *    ここまでで，攻撃回数は 1 回かつその攻撃座標はフラグが立っている.    *
   *                                                                        *
   * for 文終了条件 :                                                       *
   *    残敵が無くなる，もしくは攻撃回数が 100 回になる.                    *
   **************************************************************************/

  for (i = 2; i <= 100; i++)
```

```
    {
      k = meichup (sb, x, y);
      dispsb (sb);
      meichud (k);

      if (zanteki (skp, jkp, kkp, smp, ysp)) /* 残敵が 0 の時は真 */
{
  break;
}

      do
{
  shoot (sb, i, &x, &y);
}
      while((x < 0 || x > 9 || y < 0 || y > 9)
    || flg[x][y]);
      /* 指定座標が領海外，もしくは攻撃済みの場合再攻撃．*/

      flg[x][y] = 1;  /* 攻撃成功座標にフラグを立てる．*/
    }

  printf ("攻撃回数＝%d\n", --i); /* Battle Ship 攻略に要した攻撃回数の表示．*/

  printf ("\nPress any key to continue...\n");
  getchar ();
}



/*************************************************************************
 * haichi 関数 : 軍艦をランダムに配置する関数                            *
 *************************************************************************/

void
haichi (int sb[][10])
{
  int i, j, x, y;

  srand ((unsigned int) time (NULL));  /* 時間によりランダムデータを初期化 */

  for (i = 0; i < 10; i++)
    {
      for (j = 0; j < 10; j++)
{
```

```
  sb[i][j] = N;
}
    }

  if (rand () % 2 == 0)
    { /* 戦艦縦方向配置 */
      y = rand () % 10;
      x = rand () % 6;
      sb[x][y] = SK;
      sb[x + 1][y] = SK;
      sb[x + 2][y] = SK;
      sb[x + 3][y] = SK;
      sb[x + 4][y] = SK;
    }
  else
    { /* 戦艦横方向配置 */
      x = rand () % 10;
      y = rand () % 6;
      sb[x][y] = SK;
      sb[x][y + 1] = SK;
      sb[x][y + 2] = SK;
      sb[x][y + 3] = SK;
      sb[x][y + 4] = SK;
    }

  if (rand () % 2 == 0)
    { /* 巡洋艦縦方向配置 */
      y = rand () % 10;
      x = rand () % 7;

      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N
     || sb[x + 3][y] != N)
{
  y = rand () % 10;
  x = rand () % 7;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = JK;
      sb[x + 1][y] = JK;
      sb[x + 2][y] = JK;
      sb[x + 3][y] = JK;
    }
  else
```

```
    { /* 巡洋艦横方向配置 */
      x = rand () % 10;
      y = rand () % 7;

      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N
     || sb[x][y + 3] != N)
{
  x = rand () % 10;
  y = rand () % 7;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = JK;
      sb[x][y + 1] = JK;
      sb[x][y + 2] = JK;
      sb[x][y + 3] = JK;
    }

  if (rand () % 2 == 0)
    { /* 駆逐艦縦方向配置 */
      y = rand () % 10;
      x = rand () % 8;

      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = KK;
      sb[x + 1][y] = KK;
      sb[x + 2][y] = KK;
    }
  else
    { /* 駆逐艦横方向配置 */
      x = rand () % 10;
      y = rand () % 8;

      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/
```

```
      sb[x][y] = KK;
      sb[x][y + 1] = KK;
      sb[x][y + 2] = KK;
    }

  if (rand () % 2 == 0)
    { /* 潜水艦縦方向配置 */
      y = rand () % 10;
      x = rand () % 8;

      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = SM;
      sb[x + 1][y] = SM;
      sb[x + 2][y] = SM;
    }
  else
    { /* 潜水艦横方向 */
      x = rand () % 10;
      y = rand () % 8;

      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = SM;
      sb[x][y + 1] = SM;
      sb[x][y + 2] = SM;
    }

  if (rand () % 2 == 0)
    { /* 輸送船縦方向配置 */
      y = rand () % 10;
      x = rand () % 9;

      while (sb[x][y] != N || sb[x + 1][y] != N)
{
```

```
  y = rand () % 10;
  x = rand () % 9;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = YS;
      sb[x + 1][y] = YS;
    }
  else
    { /* 輸送船横方向配置 */
      x = rand () % 10;
      y = rand () % 9;

      while (sb[x][y] != N || sb[x][y + 1] != N)
{
  x = rand () % 10;
  y = rand () % 9;
} /* 始点から終点までで，既に配置されている軍艦があれば再座標指定．*/

      sb[x][y] = YS;
      sb[x][y + 1] = YS;
    }
}



/*************************************************************************
 * dispsb 関数 : Battle Ship の盤面情報を出力する関数                    *
 *   count => 関数呼び出し初回判断用変数                                 *
 *   csb   => 軍艦の配置座標コピー用配列                                 *
 *************************************************************************/

void
dispsb (int sb[][10])
{
  int i, j;
  static int count=0, csb[10][10];

  if (!count)
    {                           /* 軍艦の配置座標コピー */
      for(i = 0; i < 10; i++)
{
  for(j = 0; j < 10; j++)
    {
      csb[i][j] = sb[i][j];
```

```
    }
}
      count = 1;
    }

  printf ("   1 2 3 4 5 6 7 8 9 10\n");   /* 横座標の数字表示 */
  printf ("  ---------------------\n");   /* グリッド: 上端横線 */

  for (i = 0; i < 10; i++)
    {
      printf ("%2d", i + 1);              /* 縦座標の数字表示 */
      for (j = 0; j < 10; j++)
{
  putchar ('|');                  /* グリッド: 縦線 */
  switch (sb[i][j])
    {                             /* 攻撃情報表示 */
    case BM:                      /* 攻撃が命中した場合 */
      switch (csb[i][j])
{
case SK:                    /* 攻撃箇所が戦艦の場合 */

  /*****************************************************
   * 戦艦が撃沈した場合，配置されていた座標に           *
   * 数字の 5 を代替物として配置する.                   *
   *****************************************************/
  if (!skp)
    {
      putchar('5');
    }

  /*****************************************************
   * 戦艦が撃沈していない場合，数字の 1 を用いて        *
   * 攻撃が命中した事を表現する.                        *
   *****************************************************/
  else
    {
      putchar('1');
    }
  break;

case JK:                    /* 攻撃箇所が巡洋艦の場合 */

  /*****************************************************
```

```
   * 巡洋艦が撃沈した場合，配置されていた座標に          *
   * 数字の 4 を代替物として配置する．                  *
   ****************************************************/
  if (!jkp)
    {
      putchar('4');
    }

  /****************************************************
   * 巡洋艦が撃沈していない場合，数字の 1 を用いて       *
   * 攻撃が命中した事を表現する．                       *
   ****************************************************/
  else
    {
      putchar('1');
    }
  break;

case SM:                    /* 攻撃箇所が潜水艦の場合 */

  /****************************************************
   * 潜水艦が撃沈した場合，配置されていた座標に          *
   * 数字の 3 を代替物として配置する．                  *
   ****************************************************/
  if (!smp)
    {
      putchar('3');
    }

  /****************************************************
   * 潜水艦が撃沈していない場合，数字の 1 を用いて       *
   * 攻撃が命中した事を表現する．                       *
   ****************************************************/
  else
    {
      putchar('1');
    }
  break;

case KK:                          /* 攻撃箇所が駆逐艦の場合 */

  /****************************************************
   * 駆逐艦が撃沈した場合，配置されていた座標に          *
```

```
   * 数字の 3 を代替物として配置する.                    *
   ******************************************************/
  if (!kkp)
    {
      putchar('3');
    }

  /******************************************************
   * 駆逐艦が撃沈していない場合，数字の 1 を用いて       *
   * 攻撃が命中した事を表現する.                          *
   ******************************************************/
  else
    {
      putchar('1');
    }
  break;

case YS:                        /* 攻撃箇所が輸送艦の場合 */

  /******************************************************
   * 輸送艦が撃沈した場合，配置されていた座標に           *
   * 数字の 2 を代替物として配置する.                     *
   ******************************************************/
  if (!ysp)
    {
      putchar('2');
    }

  /******************************************************
   * 輸送艦が撃沈していない場合，数字の 1 を用いて       *
   * 攻撃が命中した事を表現する.                          *
   ******************************************************/
  else
    {
      putchar('1');
    }
}
      break;

    case BH:                           /* 攻撃が外れた場合 */
      putchar ('X');
      break;
```

```
    default:                          /* 未攻撃の場合 */
      putchar (' ');
      break;
    }
}
      printf ("|\n");                 /* グリッド: 右端縦線 */
    }

  printf ("  --------------------\n");       /* グリッド: 横線 */
}


/*************************************************************************
 * zanteki 関数 : 残敵判定処理                                           *
 *************************************************************************/

int
zanteki (int skp, int jkp, int kkp, int smp, int ysp)
{
  if (skp == 0 && jkp == 0 && kkp == 0 && smp == 0 && ysp == 0)
    {
      printf ("全艦撃沈！\n");
      return 1;
    }

  printf ("残敵＝");
  if (skp > 0)
    {
      printf ("戦艦 ");
    }
  if (jkp > 0)
    {
      printf ("巡洋艦 ");
    }
  if (kkp > 0)
    {
      printf ("駆逐艦 ");
    }
  if (smp > 0)
    {
      printf ("潜水艦 ");
    }
  if (ysp > 0)
```

```
    {
      printf ("輸送船");
    }
  putchar ('\n');
  return 0;
}

void
shoot (int sb[][10], int i, int *x, int *y)
{
  int tate, yoko;

  while (1)
    {
      printf ("次弾(%d)発射位置？ 縦,横 ＝ ", i);
      fgets (line, sizeof (line), stdin);
      sscanf (line, "%d,%d", &tate, &yoko);
      //      if (tate >= 1 && tate <= 10 && yoko >= 1 && yoko <= 10)
      //{
  *x = tate - 1;
  *y = yoko - 1;
  return;
  //}
    }
}

int
meichup (int sb[][10], int x, int y)
{
  int a;

  a = sb[x][y];

  if (a == SK)
    {
      sb[x][y] = BM;
      skp--;
      return SK;
    }
  if (a == JK)
    {
      sb[x][y] = BM;
      jkp--;
```

```
      return JK;
    }
  if (a == KK)
    {
      sb[x][y] = BM;
      kkp--;
      return KK;
    }
  if (a == SM)
    {
      sb[x][y] = BM;
      smp--;
      return SM;
    }
  if (a == YS)
    {
      sb[x][y] = BM;
      ysp--;
      return YS;
    }
  if (a == BM)
    {
      return BH;
    }
  sb[x][y] = BH;
  return BH;
}

void
meichud (int k)
{
  if (k == BH)
    {
      printf ("ハズレ！\n");
    }
  else if (k == SK)
    {
      printf ("命中！");
      if (skp <= 0)
{
  printf ("戦艦沈没！");
}
      putchar ('\n');
```

```
    }
  else if (k == JK)
    {
      printf ("命中！");
      if (jkp <= 0)
{
  printf ("巡洋艦沈没！");
}
      putchar ('\n');
    }
  else if (k == KK)
    {
      printf ("命中！");
      if (kkp <= 0)
{
  printf ("駆逐艦沈没！");
}
      putchar ('\n');
    }
  else if (k == SM)
    {
      printf ("命中！");
      if (smp <= 0)
{
  printf ("潜水艦沈没！");
}
      putchar ('\n');
    }
  else if (k == YS)
    {
      printf ("命中！");
      if (ysp <= 0)
{
  printf ("輸送船沈没！");
}
      putchar ('\n');
    }
}
```

A.2 Verification

Verification とは攻撃関数の挙動を調べる為のプログラムである。

A.2.1 senkan001.c

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

char line[100];

#define N 0
#define BM 1
#define BH 2
#define SK 11
#define JK 12
#define KK 13
#define SM 14
#define YS 15

void haichi (int sb[][10]);
void dispsb (int tsb[][10], int sb[][10]);
int zanteki (int skp, int jkp, int kkp, int smp, int ysp);
void shoot (int sb[][10], int i, int k, int *x, int *y);
int meichup (int tsb[][10], int sb[][10], int x, int y);
int meichud (int k);

int skp = 5, jkp = 4, kkp = 3, smp = 3, ysp = 2;

int
main (void)
{

  int tsb[10][10], sb[10][10];
  static int  flg[10][10]={0};
  int i, j, x, y, k, kekka;



  haichi (tsb);
  for (i = 0; i < 10; i++)
    {
      for (j = 0; j < 10; j++)
```

```
{
  sb[i][j] = N;
}
    }

  dispsb (tsb, sb);
  zanteki (skp, jkp, kkp, smp, ysp);

  do
    {
      shoot (sb, 1, BH, &x, &y);
      getchar ();
    }
  while(x < 0 || x > 9 || y < 0 || y > 9);

  for (i = 2; i <= 100; i++)
    {
      k = meichup (tsb, sb, x, y);
      dispsb (tsb, sb);
      kekka = meichud (k);
      if (zanteki (skp, jkp, kkp, smp, ysp))
{
  break;
}

      do
{
  shoot (sb, i, kekka, &x, &y);
  getchar ();
}
      while((x < 0 || x > 9 || y < 0 || y > 9)
    || flg[x][y]);

      flg[x][y] = 1;
    }
  printf ("攻撃回数＝%d\n", --i);
  printf ("Press any key to continue...\n");
  getchar ();
}

void
haichi (int sb[][10])
{
```

```
  int i, j, x, y;

  srand ((unsigned int) time (NULL));

  for (i = 0; i < 10; i++)
    {
      for (j = 0; j < 10; j++)
{
  sb[i][j] = N;
}
    }

  if (rand () % 2 == 0)
    { /* 戦艦縦方向 */
      y = rand () % 10;
      x = rand () % 6;
      sb[x][y] = SK;
      sb[x + 1][y] = SK;
      sb[x + 2][y] = SK;
      sb[x + 3][y] = SK;
      sb[x + 4][y] = SK;
    }
  else
    { /* 戦艦横方向 */
      x = rand () % 10;
      y = rand () % 6;
      sb[x][y] = SK;
      sb[x][y + 1] = SK;
      sb[x][y + 2] = SK;
      sb[x][y + 3] = SK;
      sb[x][y + 4] = SK;
    }

  if (rand () % 2 == 0)
    { /* 巡洋艦縦方向 */
      y = rand () % 10;
      x = rand () % 7;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N
     || sb[x + 3][y] != N)
{
  y = rand () % 10;
  x = rand () % 7;
}
```

```
      sb[x][y] = JK;
      sb[x + 1][y] = JK;
      sb[x + 2][y] = JK;
      sb[x + 3][y] = JK;
    }
  else
    { /* 巡洋艦横方向 */
      x = rand () % 10;
      y = rand () % 7;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N
     || sb[x][y + 3] != N)
{
  x = rand () % 10;
  y = rand () % 7;
}
      sb[x][y] = JK;
      sb[x][y + 1] = JK;
      sb[x][y + 2] = JK;
      sb[x][y + 3] = JK;
    }

  if (rand () % 2 == 0)
    { /* 駆逐艦縦方向 */
      y = rand () % 10;
      x = rand () % 8;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
}
      sb[x][y] = KK;
      sb[x + 1][y] = KK;
      sb[x + 2][y] = KK;
    }
  else
    { /* 駆逐艦横方向 */
      x = rand () % 10;
      y = rand () % 8;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
}
```

```
      sb[x][y] = KK;
      sb[x][y + 1] = KK;
      sb[x][y + 2] = KK;
    }

  if (rand () % 2 == 0)
    { /* 潜水艦縦方向 */
      y = rand () % 10;
      x = rand () % 8;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
}
      sb[x][y] = SM;
      sb[x + 1][y] = SM;
      sb[x + 2][y] = SM;
    }
  else
    { /* 潜水艦横方向 */
      x = rand () % 10;
      y = rand () % 8;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
}
      sb[x][y] = SM;
      sb[x][y + 1] = SM;
      sb[x][y + 2] = SM;
    }

  if (rand () % 2 == 0)
    { /* 輸送船縦方向 */
      y = rand () % 10;
      x = rand () % 9;
      while (sb[x][y] != N || sb[x + 1][y] != N)
{
  y = rand () % 10;
  x = rand () % 9;
}
      sb[x][y] = YS;
      sb[x + 1][y] = YS;
```

```
    }
  else
    { /* 輸送船横方向 */
      x = rand () % 10;
      y = rand () % 9;
      while (sb[x][y] != N || sb[x][y + 1] != N)
{
  x = rand () % 10;
  y = rand () % 9;
}
      sb[x][y] = YS;
      sb[x][y + 1] = YS;
    }
}

void
dispsb (int tsb[][10], int sb[][10])
{
  int i, j;

  printf ("   1 2 3 4 5 6 7 8 9 10\n");
  printf ("  ---------------------\n");

 for (i = 0; i < 10; i++)
    {
      printf ("%2d", i + 1);
      for (j = 0; j < 10; j++)
{
  putchar ('|');
  switch (sb[i][j])
    {                   /*攻撃情報表示*/
    case BM:
      switch (tsb[i][j])
{
case SK:
  if (!skp)
    {
      putchar('5');
    }
  else
    {
      putchar('1');
    }
```

```
  break;
case JK:
  if (!jkp)
    {
      putchar('4');
    }
  else
    {
      putchar('1');
    }
  break;
case SM:
  if (!smp)
    {
      putchar('3');
    }
  else
    {
      putchar('1');
    }
  break;
case KK:
  if (!kkp)
    {
      putchar('3');
    }
  else
    {
      putchar('1');
    }
  break;
case YS:
  if (!ysp)
    {
      putchar('2');
    }
  else
    {
      putchar('1');
    }
}
      break;
    case BH:
```

```
      putchar ('X');
      break;
    default:
      putchar (' ');
      break;
    }
}
      printf ("|\n");
    }
  printf ("  ---------------------\n");
}

int
zanteki (int skp, int jkp, int kkp, int smp, int ysp)
{
  if (skp <= 0 && jkp <= 0 && kkp <= 0 && smp <= 0 && ysp <= 0)
    {
      printf ("全艦撃沈！\n");
      return 1;
    }

  printf ("残敵＝");
  if (skp > 0)
    {
      printf ("戦艦 ");
    }
  if (jkp > 0)
    {
      printf ("巡洋艦 ");
    }
  if (kkp > 0)
    {
      printf ("駆逐艦 ");
    }
  if (smp > 0)
    {
      printf ("潜水艦 ");
    }
  if (ysp > 0)
    {
      printf ("輸送船");
    }
  putchar ('\n');
```

```
  return 0;
}


int
meichup (int tsb[][10], int sb[][10], int x, int y)
{
  int a, b;

  a = tsb[x][y];
  b = sb[x][y];

  if (b != N)
    {
      return BH;
    }
  if (a == SK)
    {
      sb[x][y] = BM;
      skp--;
      return SK;
    }
  if (a == JK)
    {
      sb[x][y] = BM;
      jkp--;
      return JK;
    }
  if (a == KK)
    {
      sb[x][y] = BM;
      kkp--;
      return KK;
    }
  if (a == SM)
    {
      sb[x][y] = BM;
      smp--;
      return SM;
    }
  if (a == YS)
    {
      sb[x][y] = BM;
```

```
      ysp--;
      return YS;
    }
  sb[x][y] = BH;
  return BH;
}

int
meichud (int k)
{
  if (k == BH)
    {
      printf ("ハズレ！\n");
      return BH;
    }
  else if (k == SK)
    {
      printf ("命中！");
      if (skp <= 0)
{
  printf ("戦艦沈没！\n");
  return SK;
}
      putchar ('\n');
    }
  else if (k == JK)
    {
      printf ("命中！");
      if (jkp <= 0)
{
  printf ("巡洋艦沈没！\n");
  return JK;
}
      putchar ('\n');
    }
  else if (k == KK)
    {
      printf ("命中！");
      if (kkp <= 0)
{
  printf ("駆逐艦沈没！\n");
  return KK;
}
```

```
      putchar ('\n');
    }
  else if (k == SM)
    {
      printf ("命中！");
      if (smp <= 0)
{
  printf ("潜水艦沈没！\n");
  return SM;
}
      putchar ('\n');
    }
  else if (k == YS)
    {
      printf ("命中！");
      if (ysp <= 0)
{
  printf ("輸送船沈没！\n");
  return YS;
}
      putchar ('\n');
    }
  return BM;
}
```

A.3 Score

Score とは攻撃関数のアルゴリズムを平均攻撃回数によって評価するプログラムである。

A.3.1 senkan100.c

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

char line[100];

#define N 0
#define BM 1
#define BH 2
#define SK 11
#define JK 12
#define KK 13
#define SM 14
#define YS 15

void haichi (int sb[][10]);
void dispsb (int sb[][10]);
int zanteki (int skp, int jkp, int kkp, int smp, int ysp);
void shoot (int sb[][10], int i, int k, int *x, int *y);
int meichup (int tsb[][10], int sb[][10], int x, int y);
int meichud (int k);

int skp = 5, jkp = 4, kkp = 3, smp = 3, ysp = 2;

int
main (void)
{

  int tsb[10][10], sb[10][10];
  static int flg[10][10]={0};
  int i, j, a, b, x, y, k, kekka, total = 0;

  for (j = 1; j <= 100; j++)
    {
      skp = 5, jkp = 4, kkp = 3, smp = 3, ysp = 2;

      haichi (tsb);
```

```
      for (a = 0; a < 10; a++)
{
  for (b = 0; b < 10; b++)
    {
      sb[a][b] = N;
    }
}

      zanteki (skp, jkp, kkp, smp, ysp);

      do
{
  shoot (sb, 1, BH, &x, &y);
}
      while((x < 0 || x > 9 || y < 0 || y > 9)
    || flg[x][y]);

      for (i = 2; i <= 100; i++)
{
  k = meichup (tsb, sb, x, y);
  kekka = meichud (k);
  if (zanteki (skp, jkp, kkp, smp, ysp))
    {
      break;
    }

  do
    {
      shoot (sb, i, kekka, &x, &y);
    }
  while((x < 0 || x > 9 || y < 0 || y > 9)
|| flg[x][y]);

  flg[x][y] = 1;
}

      for (a = 0; a < 10; a++)
{
  for (b = 0; b < 10; b++)
    {
      flg[a][b] = 0;
    }
}
```

```
      i--;
      total = total + i;
      printf ("%3d 回目のプレイ：攻撃回数 %3d 回\n", j, i);
    }
  printf ("平均攻撃回数＝ %6.2f\n", total / 100.0);
}

void
haichi (int sb[][10])
{
  int i, j, x, y;
  static unsigned int seed = 1;

  srand ((unsigned int) time (NULL) * seed);

  for (i = 0; i < 10; i++)
    {
      for (j = 0; j < 10; j++)
{
  sb[i][j] = N;
}
    }

  if (rand () % 2 == 0)
    { /* 戦艦縦方向 */
      y = rand () % 10;
      x = rand () % 6;
      sb[x][y] = SK;
      sb[x + 1][y] = SK;
      sb[x + 2][y] = SK;
      sb[x + 3][y] = SK;
      sb[x + 4][y] = SK;
    }
  else
    { /* 戦艦横方向 */
      x = rand () % 10;
      y = rand () % 6;
      sb[x][y] = SK;
      sb[x][y + 1] = SK;
      sb[x][y + 2] = SK;
      sb[x][y + 3] = SK;
      sb[x][y + 4] = SK;
```

```
    }

  if (rand () % 2 == 0)
    { /* 巡洋艦縦方向 */
      y = rand () % 10;
      x = rand () % 7;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N
     || sb[x + 3][y] != N)
{
  y = rand () % 10;
  x = rand () % 7;
}
      sb[x][y] = JK;
      sb[x + 1][y] = JK;
      sb[x + 2][y] = JK;
      sb[x + 3][y] = JK;
    }
  else
    { /* 巡洋艦横方向 */
      x = rand () % 10;
      y = rand () % 7;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N
     || sb[x][y + 3] != N)
{
  x = rand () % 10;
  y = rand () % 7;
}
      sb[x][y] = JK;
      sb[x][y + 1] = JK;
      sb[x][y + 2] = JK;
      sb[x][y + 3] = JK;
    }

  if (rand () % 2 == 0)
    { /* 駆逐艦縦方向 */
      y = rand () % 10;
      x = rand () % 8;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
}
      sb[x][y] = KK;
```

```
      sb[x + 1][y] = KK;
      sb[x + 2][y] = KK;
    }
  else
    { /* 駆逐艦横方向 */
      x = rand () % 10;
      y = rand () % 8;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
}
      sb[x][y] = KK;
      sb[x][y + 1] = KK;
      sb[x][y + 2] = KK;
    }

  if (rand () % 2 == 0)
    { /* 潜水艦縦方向 */
      y = rand () % 10;
      x = rand () % 8;
      while (sb[x][y] != N || sb[x + 1][y] != N || sb[x + 2][y] != N)
{
  y = rand () % 10;
  x = rand () % 8;
}
      sb[x][y] = SM;
      sb[x + 1][y] = SM;
      sb[x + 2][y] = SM;
    }
  else
    { /* 潜水艦横方向 */
      x = rand () % 10;
      y = rand () % 8;
      while (sb[x][y] != N || sb[x][y + 1] != N || sb[x][y + 2] != N)
{
  x = rand () % 10;
  y = rand () % 8;
}
      sb[x][y] = SM;
      sb[x][y + 1] = SM;
      sb[x][y + 2] = SM;
    }
```

```
  if (rand () % 2 == 0)
    { /* 輸送船縦方向 */
      y = rand () % 10;
      x = rand () % 9;
      while (sb[x][y] != N || sb[x + 1][y] != N)
{
  y = rand () % 10;
  x = rand () % 9;
}
      sb[x][y] = YS;
      sb[x + 1][y] = YS;
    }
  else
    { /* 輸送船横方向 */
      x = rand () % 10;
      y = rand () % 9;
      while (sb[x][y] != N || sb[x][y + 1] != N)
{
  x = rand () % 10;
  y = rand () % 9;
}
      sb[x][y] = YS;
      sb[x][y + 1] = YS;
    }
  seed = rand ();
}

void
dispsb (int sb[][10])
{
  int i, j;

  printf ("   1 2 3 4 5 6 7 8 9 10\n");
  printf ("  ---------------------\n");

  for (i = 0; i < 10; i++)
    {
      printf ("%2d", i + 1);
      for (j = 0; j < 10; j++)
{
  putchar ('|');
  switch (sb[i][j])
```

```
    {
    case BM:
      putchar ('O');
      break;
    case BH:
      putchar ('X');
      break;
    default:
      putchar (' ');
      break;
    }
}
      printf ("|\n");
    }

  printf ("  ---------------------\n");
}

int
zanteki (int skp, int jkp, int kkp, int smp, int ysp)
{
  if (skp <= 0 && jkp <= 0 && kkp <= 0 && smp <= 0 && ysp <= 0)
    {
      /* printf("全艦撃沈！\n"); */
      return 1;
    }
/*
printf("残敵＝");
if(skp > 0) {
printf("戦艦 ");
}
if(jkp > 0) {
printf("巡洋艦 ");
}
if(kkp > 0) {
printf("駆逐艦 ");
}
if(smp > 0) {
printf("潜水艦 ");
}
if(ysp > 0) {
printf("輸送船");
}
```

```
putchar('\n');
*/
  return 0;
}

int
meichup (int tsb[][10], int sb[][10], int x, int y)
{
  int a, b;

  a = tsb[x][y];
  b = sb[x][y];

  if (b != N)
    {
      return BH;
    }
  if (a == SK)
    {
      sb[x][y] = BM;
      skp--;
      return SK;
    }
  if (a == JK)
    {
      sb[x][y] = BM;
      jkp--;
      return JK;
    }
  if (a == KK)
    {
      sb[x][y] = BM;
      kkp--;
      return KK;
    }
  if (a == SM)
    {
      sb[x][y] = BM;
      smp--;
      return SM;
    }
  if (a == YS)
    {
```

```
      sb[x][y] = BM;
      ysp--;
      return YS;
    }
  sb[x][y] = BH;
  return BH;
}

int
meichud (int k)
{
  if (k == BH)
    {
      return BH;
    }
  else if (k == SK && skp <= 0)
    {
      return SK;
    }
  else if (k == JK && jkp <= 0)
    {
      return JK;
    }
  else if (k == KK && kkp <= 0)
    {
      return KK;
    }
  else if (k == SM && smp <= 0)
    {
      return SM;
    }
  else if (k == YS && ysp <= 0)
    {
      return YS;
    }
  return BM;
}
```