プログラミング演習 最終課題レポート

08-252019 小倉直己

1　選択した課題

今回は、潜水艦ゲームの課題を選択した。

2　AI のアルゴリズムの説明

今回作成した AI は、以下のアルゴリズムに則って動作するようにした。

2.1　船の配置

味方の艦の配置は、全ての艦が以下のルールを満たすように配置される。

- 艦は 5×5 のマス目の中に配置される。
- ある艦から半径 2 マスの正方形の範囲内に他の艦が存在しない。

この配置を実装すると、以下のプログラムのようになる。

```
def place_ship(self):
        distance = 2  # 2マス以上離す
        placed_positions = set()
        ship_types = ['w', 'c', 's']
        max_attempts = 500
        for ship_type in ship_types:
            placed = False
            attempts = 0
            while not placed and attempts < max_attempts:
                attempts += 1

                # choice a random position which is not occupied, and scattered.
                row = self.rng.randint(0, self.field.height - 1)
                col = self.rng.randint(0, self.field.width - 1)
                position = [col, row]

                if tuple(position) not in placed_positions:
```

```
                is_isolated = True
                for placed_pos in placed_positions:
                    if abs(position[1] - placed_pos[1]) <= distance and abs(position[0] - placed
                        # If the new position is too close to any placed ship, break
                        is_isolated = False
                        break

                if is_isolated:
                    placed_positions.add(tuple(position))
                    placed = True
                    logging.info(f"Placed {ship_type} at {position}")
        if not placed:
            logging.warning(f"Failed to place {ship_type} after {max_attempts} attempts. Trying
            for r in range(self.field.height):
                for c in range(self.field.width):
                    if (c, r) not in placed_positions:
                        placed_positions.add((c, r))
                        logging.info(f"Placed {ship_type} at {[c, r]}")
                        break
                if placed:
                    break

    return {
        'w': list(list(placed_positions)[0]),
        'c': list(list(placed_positions)[1]),
        's': list(list(placed_positions)[2])
    }
```

このコードは、今回作成した original_player.py の一部であり、OriginalPlayer クラスの place_ship メソッドとして実装されている。このプログラムでは、ランダムに艦を配置する試行を 100 回繰り返し、先ほど述べたルールに従って配置できなかった場合は、ルールと関係なくランダムに配置するようにしている。

2.2 プレイの方針について

プレイの方針は、以下のように定めた。

- 基本的に、常に敵の艦を攻撃する。
- マップ上で攻撃の優先度が 0 以上、かつ自分の艦から攻撃可能である位置が存在しない場合のみ、艦を移動する。

これは、今回の潜水艦ゲームのルールにおいて、自分の艦を移動することが戦略的なアドバンテージを生ま

ないばかりか、自分の艦の位置を相手に知られるヒントになりうるためである。例えば、自分の戦艦 (w) が右に 2 マス移動すると、相手は自分の戦艦が右に 2 マス移動したことを知ることができる。すると、自分の戦艦は現在 5×5 マスの盤面のうち、右側 3 列にしか存在しないことがわかる。このように、艦の移動は相手に自分の艦の位置を知られるヒントになりうるため、艦の移動は攻撃可能な位置が存在しない場合に限り行うこととした。

2.3 攻撃時のアルゴリズム

攻撃のアルゴリズムは、以下のように定めた。

- 攻撃可能な位置のうち、優先度が最も高い位置を攻撃する。
- 優先度が同じマスが複数存在する場合は、ランダムに 1 つ選ぶ。

優先度は、以下のように定義した。

- 敵の艦が存在するかしないか判明していない場合は、優先度を 0 とする。
- 敵の艦が存在する可能性がある (水飛沫の情報とヒット情報で判断) 場合は、優先度を 1 とする。
- 敵の艦が存在する可能性がない場合は、優先度を-0.25 とする。

この優先度は、敵の艦ごとに個別にマップとして保持される。マップは初期状態では全て 0 であり、味方が攻撃を行った時、ヒットが発生した場合は、攻撃した位置の優先度を 1 し、それ以外のマスの優先度を-0.25 とする。また、near の判定 (水飛沫) が出た場合は、攻撃した位置には艦は存在し得ないので、攻撃した位置の優先度を-0.25 とし、周囲 8 マスの優先度を 1 とする。その他のマスの優先度は-0.25 とする。hit も near もなかった場合は、攻撃した位置とその周囲 8 マスの優先度を-0.25 とする。

2.4 敵の移動の監視

今回のルールでは、敵の艦の移動の履歴が与えられるため、それをもとに優先度マップを更新する。敵の艦が移動した場合、以下のように優先度マップを更新する。

- 移動した方向とマス目を取得する。
- 例えば右に 2 マス移動した場合、マップの左側 2 列には敵の艦が存在しないことがわかるので、左側 2 列の優先度を-0.25 とする。
- 残りの右側 3 列の優先度は、移動前のマップの左側 3 列の優先度を右に 2 マスずらしたものとする。
- 移動した艦の位置が既に判明している場合は、その位置を移動に合わせて更新する。

3 random_player との比較

対 random_player の勝率は、以下のようになった。