1 Monte-Carlo strategy

1.1 The approach

Given a game state s_0, an optimal action is one that maximizes the expected winning probability:

$$\underset{a \in \mathcal{A}(s_0)}{\operatorname{argmax}} E\left(v\left(Sf\right) \mid S0 = s_0, A_0 = a\right).$$

Here the expectation is with respect to the distribution over terminal states S_f, and $v\left(s\right) \in \{0, 1\}$ indicates whether our robot is the winner in S_f. Solving this exactly requires knowledge of the action-conditional distribution over terminal states $\rho\left(S_f \mid S_0 = s_0, A_0\right)$, which is a function of the unknown strategies of the opponents.

Instead, the Monte-Carlo strategy estimates the expectation corresponding to each action a_0 by sampling terminal states s_f from an empirical distribution $\hat{\rho}\left(S_f \mid s_0, a_0\right)$ and averaging the values $v\left(s_f\right)$ so found. The sampling procedure is described in Algorithm 1. In short, a simulated game is played out in which our robot's first action is a_0 and all further actions by all players are made according to some playout strategy $\pi\left(s\right) \in \mathcal{A}\left(s\right)$. Note that the resulting empirical distribution $\hat{\rho}$ is unlikely to be a good approximation to the actual distribution. However, since the simulated game has all players playing according to the same strategy π, the expectations with respect to the two distributions are positively correlated.

While the strategy as described above works in theory, in practice not enough samples are collected to provide reliable estimates of the expectations. This is because it takes on average over a thousand turns to reach a terminal state. Longer simulations take longer to compute and result in fewer samples per unit time. At the same time, due to the uncertainty introduced with every turn, the resulting terminal state is much less dependent on the initial action.

In order to improve the number of samples and their information content, the simulated games are played only up until a finite depth d. Furthermore, the win indicator function v is replaced by a heuristic evaluation function $\hat{v}\left(s\right) \in \mathbb{R}$ that is also defined for non-terminal states:

$$\underset{a \in \mathcal{A}(s_0)}{\operatorname{argmax}} E\left(\hat{v}\left(S_d\right) \mid A_0 = a\right).$$

This works well if $\hat{v}\left(s\right)$ positively correlates with the actual winning probability $E\left(v\left(S_f\right) \mid S_0 = s_0\right)$. Section 1.3 discusses the evaluation funcions that were implemented.

1.2 The action space

At the beginning of each turn, each player is dealt a hand $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ with which to populate a sequence of registers $r_1, r_2, \ldots, r_k$. This gives rise to a set $\tilde{\mathcal{A}}$ of $m!/\left(m-k\right)!$ actions, each of which corresponds to an ordered

k-sequence of distinct cards. Typically there exist sets of actions for which it can be deduced that they produce identical terminal state distributions.

As an example of this, consider the set of cards $\mathcal{H} = \{11, 83, 57, 49, 35, 21, 3, 50, 4\}$. As the cards 49 and 50 have the same movement effect and their priorities are adjacent, their order with respect to each other makes no difference to the game. That is, some action $a = \{49, r_2, 50, r_4, \ldots, r_k\}$ necessarily has the same effect as the action $a' = \{50, r_2, 49, \ldots, r_k\}$. A similar relation holds for rotation cards, except that the priorities do not need to be adjacent, since rotating robots do not interfere with each other.

This leads to the following equivalence relation:

$$\forall h, h' \in \mathcal{H} h \equiv h' \iff \left\{ \begin{array}{ll} \omega(h) = \omega(h') & \text{if } h, h' <= 42 \\ \tau(h) = \tau(h') \wedge \left(|h - h'| = 1 \vee \exists \tilde{h} \in \mathcal{H} \colon |h - \tilde{h}| = 1 \wedge \tilde{h} \equiv h' \right) & \text{otherwise} \end{array} \right\}.$$

where $\omega(h)$ denotes the rotational effect of card h, and $\tau(h)$ the translational effect. The equivalence partition so induced allows the set of cards $\mathcal{H}$ to be treated as a multiset containing each equivalence class of cards $[h_i]$ as an item with multiplicity $|\,[h_i]\,|$. This results in a severely reduced action space $\mathcal{A}$ where each action assigns an equivalence class rather than a card to each register.

This analysis serves two purposes. Firstly, sampling uniformly from $\mathcal{A}$ is more useful than sampling uniformly from $\tilde{\mathcal{A}}$, since it is closer to sampling uniformly from the set of possible outcomes. Secondly, there are now many fewer actions for which expectations need to be estimated, which means more samples are available to estimate these expectations.

Note that there are circumstances under which it is possible to reduce the action space further. For example, if $49, 51 \in \mathcal{H}$ and 50 is locked in some player's last register, then the relative order of 49 and 51 has no effect if both are played in an earlier register. However, these cases are rare.

Finally, it is useful in practice to have a bijective mapping between elements of $\mathcal{A}$ and the natural numbers. This allows efficient lookup tables to be used. Algorithm 2 computes the number corresponding to an action, and Algorithm **??** computes the action from the number. The encoding is a generalization of the factoradic system to k-sequences taken from multisets.

1.3 Evaluation Functions

- **Minimum Checkpoint Advantage** compares our robot to the opponent that is furthest ahead of the other opponents. It computes the difference $a - b$ between the number of checkpoints a reached by our robot and the number of checkpoints b reached by the opponent. Clearly, this correlates with winning probability. However, it is sparse and delayed: it does not reward approaching a checkpoint until it is actually reached.

- **Manhattan Distance to Checkpoint** computes the L_1 distance to the

Algorithm 1 Sampling terminal states

Require: Playout strategy π, initial state $s_0 \in \mathcal{S}$, initial action $a_0 \in \mathcal{A}(s_0)$

procedure SAMPLE-TERMINAL-STATE(s_0, a_0, π)
    $t \leftarrow 0$
    Fill registers of our robot according to a_t
    Deal cards for each opponent and fill their registers according to $\pi(s_t)$
    while $\neg terminal(s_t)$ **do**
        Perform the turn, obtaining $s_{t+1} \leftarrow successor(s_t)$
        $t \leftarrow t + 1$
        Deal cards for each player and fill their registers according to $\pi(s_t)$
    end while
    return s
end procedure

Algorithm 2 Compute action number

Require: Hand $\mathcal{H}$ with equivalence classes $c_1, c_2, ...c_m$, action a

procedure ACTION-NUMBER($\mathcal{H}$, a)
    $multiplicities \leftarrow [|c_i||i = 1, \ldots, m]$
    $digits \leftarrow [r_i|i = 1, \ldots, k]$
                                ▷ Determine the digits and radices
    $radix \leftarrow m$
    $radices \leftarrow []$
    for $i \leftarrow 0$ **to** $k - 1$ **do**
        $radices[i] \leftarrow radix$
        Decrement $multiplicities[digits[i]]$
        if $multiplicities[digits[i]] = 0$ **then**
            Decrement $radix$
        end if
        for $j \leftarrow 0$ **to** $digits[i] - 1$ **do**
            if $multiplicities[j] = 0$ **then**
                Decrement $digits[i]$
            end if
        end for
    end for
                ▷ Compute in such a way that the least-significant radix is m
    $n_a \leftarrow 0$
    for $i \leftarrow 0$ **to** $k - 1$ **do**
        $n_a \leftarrow n_a * radices[i]$
        $n_a \leftarrow n_a + digits[i]$
    end for
    return n_a
end procedure

Algorithm 3 Reconstruct action from number

Require: Hand $\mathcal{H}$ with equivalence classes $c_1, c_2, ...c_m$, action number n_a

procedure NUMBER-ACTION($\mathcal{H}$, n_a)
    $multiplicities \leftarrow [|c_i||i = 1, \ldots, m]$
    $digits \leftarrow [0|i = 1, \ldots, k]$
        ▷ Recover the digits, knowing that the least-significant radix is m
    $radix \leftarrow m$
    for $i \leftarrow 0$ **to** $k - 1$ **do**
        $digits[i] \leftarrow n_a \bmod radix$
        $n_a \leftarrow \lfloor n_a / radix \rfloor$
        for $j \leftarrow 0$ **to** $digits[i]$ **do**
            if $multiplicities[j] = 0$ **then**
                Increment $digits[i]$
            end if
        end for
        Decrement $multiplicities[digits[i]]$
        if $multiplicities[digits[i]] = 0$ **then**
            Decrement $radix$
        end if
    end for

Ensure: $n_a = 0$

    return $digits$
end procedure

next checkpoint.

- **Informed Distance to Checkpoint** finds the shortest path to the next checkpoint when traveling around walls and pits. As a reward signal, it is less likely to get the robot stuck in a local optimum where it has to first distance itself from the checkpoint and then approach it again. However, it is too expensive to compute, and since the Monte-Carlo strategy searches d turns ahead, it is already able to escape local optima.

- The **Hodgepodge** is a linear combination of several features:

Feature	Weight	Domain
Whether our robot has won or lost or neither	100	$\{-1, 0, 1\}$
Whether our robot is active[1]	3	$\{-1, 1\}$
Minimum Checkpoint Advantage	5	$[-3, 3]$
Absolute checkpoint progress	1	$[1, 4]$
Damage	-0.3	$[0, 9]$
Manhattan Distance to Checkpoint	-0.1	$[0, 144]$

  The weights were tuned by hand based on observed behavior in test games. There was not enough time to apply a learning algorithm.

section: playout strategies