Trigame: Formal Prelude to the Single-Tile Kit Problem

The purpose of this document is to define, with precision, the notation, local structure, rules, and scoring conventions needed to state a reduced one-tile optimization problem in Trigame.

The ultimate goal is to determine the best possible single-tile play for the character **Kit**, a player whose preferences differ from those of a general strategist: Kit dislikes placing tiles, prefers placing tokens, may skip once per game, and is assumed to possess strong memory but weak large-scale planning.

1. Players, Tokens, and Tiles

Players

A game may contain up to 12 players. Players are indexed by positive integers and denoted in game records by forms such as `1.`, `2.`, `3.`, and so on.

In two-player examples, players may also be referred to by color, such as **red** and **green**, particularly when discussing graphical realizations of the game.

Tokens

A **token** is a player's placable game piece. All tokens are identical in form and are distinguished only by ownership. Ownership is represented visually by color.

Tiles

The board is built from triangular **tiles**. Each tile is oriented either point-up or point-down. The first tile placed is the center tile and is always point-up. It is denoted by `[C]`.

2. Sockets and Paths on a Tile

Sockets

Each tile contains exactly seven distinguished token positions, called **sockets**:

- one center socket, numbered `0`,
- six outer sockets, numbered clockwise around the tile.

The outer numbering convention is orientation-dependent but standardized:

- for a point-up tile, socket `1` is the top corner,
- for a point-down tile, socket `1` is the bottom corner.

The remaining outer sockets are then numbered in clockwise order. Thus the numbering rule is: **socket 1 is always the point of the tile, and the rest proceed clockwise**.

Paths

Each tile contains nine path segments:

- six boundary paths around the perimeter,
- three radial paths from corner sockets to the center socket.

These paths partition the tile into three internal regions, hereafter called **subs** (sub-tiles or sub-areas).

3. Coordinate Language for Tiles and Sockets

Tile Coordinates

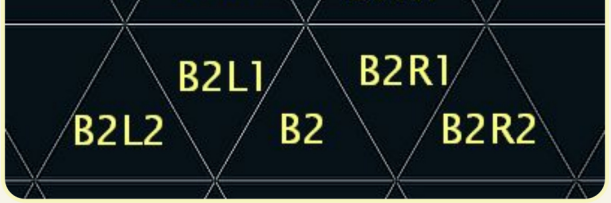
Tiles are named relative to the initial center tile `[C]`. Rows are indexed vertically:

- `C` = center row,
- `A1, A2, A3, ...` = successive rows above center,
- `B1, B2, B3, ...` = successive rows below center.

Horizontal displacement within a row is then indicated by `L1, L2, L3, ...` for left, and `R1, R2, R3, ...` for right.

Examples:

- `[C]` = center tile,
- `[B1]` = the tile directly below `[C]`,
- `[B1R1]` = the tile immediately right of `[B1]`,
- `[CL1]` = the tile immediately left of `[C]`.



Socket Coordinates

Each tile has exactly seven sockets: one central socket and six outer sockets. The central socket is always numbered `0`. The outer sockets are numbered clockwise, beginning at the point of the triangle.

- For a **point-up** tile, socket `1` is the **top corner**.
- For a **point-down** tile, socket `1` is the **bottom corner**.

Thus the numbering is orientation-dependent, but the rule is always the same: **start at the point and count clockwise**.

A token placed on a socket is written by appending the socket number to the tile designation. Thus:

- `[C]` denotes the tile only,
- `[C]1` denotes a token on socket `1` of the center tile,
- `[B1R1]0` denotes a token on the center socket of tile `[B1R1]`.

Examples:

- `[C]0` = token in the center of the center tile,
- `[C]1` = token on the pointed corner of the center tile,
- `[C]4` = token on socket `4` of the center tile, counted according to the tile's orientation,
- `[CL1]3` = token on socket `3` of the tile immediately left of the center tile.

Point-down tile. Socket `0` is the center. Socket `1` is the point of the triangle. The remaining sockets are numbered clockwise.

Point-up tile. Socket `0` is the center. Socket `1` is the point of the triangle. The remaining sockets are numbered clockwise.

4. Events, Turns, and the Game Record Language

Events

An **event** is a single atomic action of one player. The possible events are:

- place a tile,
- place a token,
- remove a tile,
- remove a token,
- skip the remainder of the turn.

Turn Structure

A standard turn consists of two events.

- On the first event, a player may place or remove either a tile or a token.
- On the second event, a player may only place a tile or place a token.

If a player captures a full tile during the turn, one extra event is awarded **immediately** and added to that same turn.

Game Record Syntax

Each line of a game record corresponds to one player's turn. The player number appears first, followed by the events of that turn, separated by colons.

Example:

```
1. [C] : [C]0
```

This means Player 1:

- placed the center tile `[C]`, then
- placed a token at socket `0` of that tile.

Removal Notation

To denote removal, prefix the event with a minus sign.

Example:

```
2. -[C]0 : [C]
```

This means Player 2 removed the token at `[C]0`, then placed tile `[C]`.

5. Skip Rule

A player may skip the remainder of a turn, including the whole turn, but:

- a player may skip **at most once per game**,
- a skip immediately ends that player's turn.

If a player skips before any tile has been placed, then no game-record line is produced for that skipped turn. Thus if Player 1 skips the opening and Player 2 begins play, the record begins with Player 2.

6. Token Propagation Along Paths

When a player places a token on a socket, that token may induce additional tokens. The rule is:

> A newly placed token creates tokens in every straight-line direction toward the next token already owned by the same player, provided that every path segment between them is already of that player's color.

Equivalently: along any straight direction, if a newly placed token and an already owned token of the same player bound a path whose constituent segments are already in that player's color, then every intermediate socket on that line is filled with that player's tokens.

Automatic token placement applies only along paths already of that player's color. It does not occur along neutral or opponent-colored paths.

7. Shared Edges and Tile Interaction

When one tile is placed adjacent to another, the newly placed tile overlaps the common boundary edge. Consequently, that edge is thereafter treated as belonging to the new tile rather than the old one. For example, if a red tile originally has all nine red paths, then placing a green tile beside it causes the common edge to become green rather than red, thereby reducing the number of paths belonging to the red tile.

A tile may not be placed adjacent to another tile if any token occupies a socket on the common edge.

8. Capturing Subs and Full Tiles

Subs

A sub is captured when a player occupies all four sockets that bound it. Because those four sockets determine all four of its boundary paths, capturing all four boundary sockets is equivalent to owning the entire sub.

Full Tile

A full tile is captured when a player has captured all three of its subs. Upon capturing a full tile, the player immediately receives one extra event on that same turn.

9. Scoring

Scoring is as follows:

- **0 points** for placing a tile,
- **1 point** for each token currently owned on the board,
- **2 points** for each captured sub,
- **4 additional points** for capturing a full tile.

Thus the game record language itself does not encode scoring, but the problem under study does depend on it.

10. Locked Games and Removal Restrictions

- A token may not be removed once it is connected in a path.
- Once no legal tile placements remain, the game is said to be **locked**.
- In a locked game, tokens may not be removed.

11. Restriction to the Single-Tile Problem

The full Trigame language supports arbitrarily many tiles and players, but the present problem is much smaller:

> Determine the best possible play for Kit in every possible **single-tile game** against exactly one opponent.

In this reduced setting:

- only one tile is ever relevant, namely `[C]`,
- the first player may or may not choose to skip,
- all subsequent play occurs on that one tile,
- and the state space is finite and heavily reducible by symmetry.

12. Kit as a Specialized Player

Kit is not modeled as a universal strategist. Instead:

- Kit dislikes going first,
- Kit dislikes placing tiles,

- Kit prefers placing tokens whenever possible,
- Kit may use the once-per-game skip to avoid opening the board,
- Kit has a good memory and has played many games.

Thus Kit is best modeled not as a strong general optimizer, but as a **specialist in a small solved subgame**.

13. Symmetry Reduction of the Opening

If Kit is forced to move first and to place the center tile, then his first token has seven apparent destinations, but these reduce, up to symmetry, to only three classes:

Class	Representative	Description
Center	`[C]0`	Token at the center socket
Corner	`[C]1`	Any corner socket, reduced by rotation
Edge-midpoint	`[C]2`	Any edge-middle socket, reduced by rotation

Therefore the opening problem reduces from seven first-token placements to three symmetry classes. The same reduction applies to the opponent's opening if the opponent moves first.

14. Formal Statement of the Problem

Problem.

Let the game be restricted to a single tile `[C]`, with the rules and scoring above. Determine, for Kit, an optimal policy for every possible single-tile opening state against one opponent.

In particular:

1. Should Kit skip if required to be the player who places `[C]`?
2. If Kit does not skip, which of the three symmetry-reduced opening token classes is optimal?
3. If the opponent moves first, how should Kit respond in each symmetry-reduced case?
4. How do path propagation, sub capture, and the possibility of immediate extra events alter the optimal one-tile policy?

15. Suggested Mathematical Viewpoint

This problem belongs primarily to:

- **combinatorics**,
- **combinatorial game theory**,
- **graph theory**, and
- **finite symmetry-reduced state-space analysis**.

A suggested route is:

1. Formalize the one-tile game state as a finite labeled graph state.
2. Reduce equivalent states under the rotational and reflection symmetries of the tile.
3. Enumerate all legal event sequences under the skip, removal, and propagation rules.
4. Score terminal states exactly.
5. Back-propagate values by minimax or equivalent finite search.

The intended spirit is not merely "solve a toy problem," but:

> Construct a complete and optimal single-tile memory table for Kit, so that Kit may be globally child-like and locally devastating.

16. Example Record

```
Match
1: Player 1
2: Player 2

1. [C] : [C]2
2. [C]1 : [C]3
1. [C]4 : [C]5
2. [C]6 : [C]0
```

This represents a fully legal single-tile game in the Game Record language. The mathematical problem is not to read such a record, but to determine which such records can arise under optimal play for Kit.